我们在开发过程中，经常会遇到这样的场景：

```
class Person {
    private String name;
    private String gender;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getGender() {
        return gender;
    }
    public void setGender(String gender) {
        this.gender = gender;
    }

}
public class Test {
    public static void main(String[] args) throws Exception {
        Person person = new Person();
        // 利用set/get方法设置属性值
        person.setName("张三");
        person.setGender("男");
    }
}
```

当类中属性只有一两个的时候，还可以耐心的调用set/get方法，如果类中有几十个，成百上千个属性需要设置呢，怎么办?

如果这种情况下还一个一个设置，那岂不是要忙死了，而且主方法代码不久爆炸多了吗。

如果我们可以通过传入一个字符串，就把所有的属性设置了，这不就简单了很多了吗。如上面的程序，我们可以传入：

```
// 字符串的格式随便都可以定义，我们这里选择如下格式
// 类名.属性名1:属性值1|类名.属性名2:属性值2|...
String value = "person.name:张三|person.gender:男";
```

接下来，我们实现一下这种方式。

# 单级VO操作

上面所讨论的方法其实是Java中的 VO操作。

vo:value object,值对象，使用一个值作为数据的存储与传递。

vo一般是来做值的存储与传递。

我们这里的值就是字符串。

# 设计思路

我们看一个图，以此为基准进行程序设计。

## Person类

我们要操作的类，这里以Person类为例子。

```
package class.vo;

public class Person {
    private String name;
    private String gender;

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getGender() {
        return gender;
    }
    public void setGender(String gender) {
        this.gender = gender;
    }

    @
    Override
    public String toString() {
        return "Person[" +
            "name: " + this.name + ", gender: " + this.gender +"]";
    }
}
```

## PersonAction类

其实就是实际要操作的类上层的一个封装，根据我们工程中不同的操作类进行相对的修改。

```
package class.vo;

import class13.util.BeanOpreation;

public class PersonAction {
    private Person person = new Person();
    public void setValue(String value) throws Exception {
        BeanOpreation.setBeanValue(this, value);
    }

    public Person getPerson() {
        return this.person;
```

```
    }
}
```

# BeanOpreation工具类

单级VO的核心操作，利用到大量的反射操作

```
// 工具类一般都存放在util包下
package class13.util;

import java.lang.reflect.Field;
import java.lang.reflect.Method;

public class BeanOpreation {
    private BeanOpreation() {}

    // 传入修改命令，格式为 "类名.属性名1:属性值1|类名.属性名2:属性值2|..."
    public static void setBeanValue(Object actionObject, String msg) throws Exception{
        // 根据 '|' 把命令分为两部分
        String[] result = msg.split("\\|");
        for(int i = 0; i < result.length; i++) {
            // 根据 ':' 再分，得到 类.属性 和 属性值
            String[] temp = result[i].split("\\:");
            // 拿到 类名
            String className = temp[0].split("\\.")[0];
            // 拿到 属性名
            String fieldName = temp[0].split("\\.")[1];
            // 拿到属性内容
            String fieldValue = temp[1];

            // 获取要操作的类(通过PersonAction类拿到Person类)
            Object curObject = getObject(actionObject, className);
            // 通过当前操作类的set方法，进行属性值修改
            setObject(curObject, fieldName, fieldValue);
        }
    }

    // 因为在PersonAction类中获取Person类的方法叫做getPerson,而传入的命令中，类名首字母是小写的，所以这里要对类名做一下修改
    public static String initCap(String str) {
        return str.substring(0, 1).toUpperCase()+str.substring(1);
    }


    // 获取要操作的类(通过PersonAction类拿到Person类)
    public static Object getObject(Object obj, String className) throws Exception {
        // 通过真实操作类的类名字符串，转换为Action类中获取真实操作类得的方法名
        String getClassMethodName = "get"+initCap(className);

        // 在正式获取get方法前做一个判断，判断Action类中是否按照约定定义了private person 对象，如果没有这个对象，何谈之后的修改
        Field field = obj.getClass().getDeclaredField(className);
        if(field == null) {
            // 再给一次机会，在父类中查找
            field = obj.getClass().getField(className);
        }
        if(field == null) {
```

```
            return null;
        }
        // 通过上面经过处理的来的 get方法 方法名在Action类中查找该方法
        Method getClassMethod = obj.getClass().getMethod(getClassMethodName);
        return getClassMethod.invoke(obj);
    }

    // 传入一个类对象，属性名，属性值，调用它的set将该属性修改为给定的属性值
    public static void setObject(Object curObj, String fieldName, String fieldValue) throws
Exception {
        Field field = curObj.getClass().getDeclaredField(fieldName);
        // 判断该属性是否存在
        if(field == null) {
            field = curObj.getClass().getField(fieldName);
        }
        if(field == null) {
            return;
        }
        String setMethodName = "set"+initCap(fieldName);
        Method setMethod = curObj.getClass().getMethod(setMethodName, field.getType());
        setMethod.invoke(curObj, fieldValue);
    }
}
```

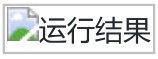
# Test测试类

```
package class;

import class.vo.PersonAction;

public class Test {
    public static void main(String[] args) throws Exception {
        String value = "person.name:张三|person.gender:男";
        PersonAction personAction = new PersonAction();
        personAction.setValue(value);
        System.out.println(personAction.getPerson());
    }
}
```

运行结果:




完全符合我们的预期。

上面的例子只是以一个简单的Person类为例子，核心是BeanOpreation工具类与Action类的配合使用。

BeanOpreation工具类可以适应不同的类，只要有相应的Action类即可，所以说反射VO的作用还是很强大的，希望大家都能够掌握。