内省 Introspector：

用于对JavaBean操作，JavaBean是一种特殊的Java类，主要用来传递数据信息，

这种Java类的方法主要用来访问私有的字段，且方法名符合某种命名规则(get\set方法)。

如果要在两个模块之间传递多个信息，可以将这些信息封装到一个JavaBean中，这种JavaBean的实例对象

通常称之为值对象(Value Object: VO)。

JavaBean的属性是根据其中的 setter和getter方法来确定的，而不是根据其中的成员变量。如果方法名为

setId,去掉set前缀，就得到了JavaBean的属性名(get方法同理)。

public class Person{

private int x;

public int getAge(){return x}; public void setAge(int age){this.x=age}

}

getAge ---> 如果第二个字母是小写，则把第一个字母编程小写的 --->age;

gettime ---> time ; setTime ---> time ;

getCPU ---> CPU (如果第二个字母是大写，则保持原样)。

总之，一个Java类被被当作JavaBean使用时，JavaBean的属性是根据方法名推断出来的，

它根本看不到Java类内部的成员变量。

使用BeanUtils工具包操作JavaBean,需要用到

(commons-beanutils-1.9.1.jar、commons-logging-1.1.3.jar)

/\*\*

\* 内省IntroSpector的简单应用

\* \*/

**public** **class** IntroSpectorTest {

**public** **static** **void** main(String[] args) **throws** Exception {

ReflectPoint pt1 = **new** ReflectPoint(3, 5);

String propertyName = "x";

// "x" ---> "X" ---> "getX" ---> MethodGetX

Object retVal = *getProperty*(pt1, propertyName);

System.*out*.println(retVal);

Object value = 7;

*setProperties*(pt1, propertyName, value);

System.*out*.println(pt1.getX());

//采用BeanUtils操作JavaBean,BeanUtils以字符串形式操作JavaBean

System.*out*.println(BeanUtils.*getProperty*(pt1, "x").getClass().getName());

BeanUtils.*setProperty*(pt1, "x", "9"); //BeanUtils自动完成数据类型的转换

System.*out*.println(pt1.getX());

/\*

//java7的新特性

Map map = {name:"zxx", age:18};

BeanUtils.setProperty(map, "name", "lhm");

\*/

//BeanUtils支持属性的级联操作，birthday属于Date类型，而Date下面有time属性

BeanUtils.*setProperty*(pt1, "birthday.time", "111"); // 111代表ms

System.*out*.println(BeanUtils.*getProperty*(pt1, "birthday.time")); //必须将pt1中的Date属性进行初始化，否则会报空指针

//PropertyUtils以属性本身的类型进行操作

PropertyUtils.*setProperty*(pt1, "x", 9);

System.*out*.println(PropertyUtils.*getProperty*(pt1, "x").getClass().getName());

}

**private** **static** **void** setProperties(ReflectPoint pt1, String propertyName,

Object value) **throws** IntrospectionException,

IllegalAccessException, InvocationTargetException {

PropertyDescriptor pd2 = **new** PropertyDescriptor(propertyName,

pt1.getClass());

Method methodSetX = pd2.getWriteMethod();

methodSetX.invoke(pt1, value);

}

**private** **static** Object getProperty(Object pt1, String propertyName)

**throws** IntrospectionException, IllegalAccessException,

InvocationTargetException {

/\*

\* PropertyDescriptor pd = new PropertyDescriptor(propertyName,pt1.getClass());

\* Method methodGetX = pd.getReadMethod();

\* Object retVal = methodGetX.invoke(pt1);

\*/

//另一种实现方式，相对复杂不方便

BeanInfo beanInfo = Introspector.*getBeanInfo*(pt1.getClass());

PropertyDescriptor[] pds = beanInfo.getPropertyDescriptors();

Object retVal = **null**;

**for** (PropertyDescriptor pd : pds) {

**if** (pd.getName().equals(propertyName)) {

Method methodGetX = pd.getReadMethod();

retVal = methodGetX.invoke(pt1);

**break**;

}

}

**return** retVal;

}

}

**public** **class** ReflectPoint {

**private** Date birthday = **new** Date();

**public** Date getBirthday() {

**return** birthday;

}

**public** **void** setBirthday(Date birthday) {

**this**.birthday = birthday;

}

**private** **int** x;

**public** **int** y;

//alt+shift+s快捷键生成构造方法

**public** ReflectPoint(**int** x, **int** y) {

**super**();

**this**.x = x;

**this**.y = y;

}

//alt+shift+s生成getter\setter方法

**public** **int** getX() {

**return** x;

}

**public** **void** setX(**int** x) {

**this**.x = x;

}

**public** **int** getY() {

**return** y;

}

**public** **void** setY(**int** y) {

**this**.y = y;

}

}