**package** introspectorTest;

**import** java.io.BufferedReader;

**import** java.io.FileReader;

**import** java.lang.reflect.Constructor;

**import** java.lang.reflect.Field;

/\*\*

\* 需求：编写一个工厂方法，根据配置文件的内容，工厂方法返回对应的对象，并且把对象要有对应的属性值

\*

\* 1.读取配置文件信息

\* 2.根据信息构建对象（利用反射动态构造对象）

\* 3.如果一个类中没有set和get方法，那么内省就没用了！！！，因为内省是根据这两个方法来操纵属性数据的

\*

\* 以后我们开发框架的时候，经常需要把一些数据封装到对象中的

\*

\*

\* ！！！以下是使用普通反射进行的操作，在Demo2.java中将使用内省类来操作

\* 内省类是通过反射获得对象或者类的属性描述器从而来操纵属性的工具（Method类的invoke方法）

\*

\*

\* Apache开发了一套比内省更好用的工具：BeanUtils

\*

\*/

**public** **class** Demo1 {

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

Person p = (Person)*getInstance*();

System.***out***.println(p);

}

//根据配置文件的内容产生对象的对象，并且要把对象的属性值封装到对象中

**public** **static** Object getInstance() **throws** Exception{

BufferedReader bufferedReader = **new** BufferedReader(**new** FileReader("obj.txt"));

String className = bufferedReader.readLine();//读取配置文件，获取到完整的类名

Class<?> clazz = Class.*forName*(className);

//通过class对象获取到无参构造方法

Constructor<?> constructor = clazz.getConstructor();

//通过构造器对象创建对象

Object o = constructor.newInstance();

//读取属性值

String line = **null**;

**while**((line = bufferedReader.readLine()) != **null**){

/\*

split字符串，根据给定正则表达式的匹配拆分此字符串，返回String[]数组

左边的为datas[0]，右边的为datas[1]

\*/

String[] datas = line.split("=");

//通过属性名获取到对应的Field对象

Field field = clazz.getDeclaredField(datas[0]);

**if**(field.getType() == **int**.**class**){

field.set(o, Integer.*parseInt*(datas[1]));

}**else**{

field.set(o, datas[1]);

}

}

bufferedReader.close();

**return** **null**;

}

}

**package** introspectorTest;

**import** java.beans.BeanInfo;

**import** java.beans.IntrospectionException;

**import** java.beans.Introspector;

**import** java.beans.PropertyDescriptor;

**import** java.lang.reflect.Method;

/\*\*

\* 内省类，一个变态的反射，是

\*

\* 内省类主要解决的问题：把对象的属性数据封装到对象中

\*

\* 如果一个类中没有set和get方法，那么内省就没用了！！！，因为内省是根据这两个方法来操纵属性数据的

\*

\*/

**public** **class** Demo2 {

/\*

普通的通过属性描述器，获得已知对象的属性的入口方法，从而来填入属性

\*/

**public** **void** testProperty() **throws** Exception {

Person p = **new** Person();

//属性描述器 （property即是属性）

PropertyDescriptor descriptor = **new** PropertyDescriptor("id", Person.**class**);

//获取属性对应的get或者set方法来设置或者获取属性

Method m = descriptor.getWriteMethod();//获取属性的set方法

//执行该方法设置属性值

m.invoke(p, 100);

Method readMethod = descriptor.getReadMethod();//获取属性的get方法

System.***out***.println(readMethod.invoke(p));

}

/\*

通过BeanInfo获得一个类中的所有属性描述器

\*/

**public** **void** getAllProperty() **throws** IntrospectionException{

//IntroSpector 内省类

BeanInfo beanInfo = Introspector.*getBeanInfo*(Person.**class**);

//通过BeanInfo获取所有的属性描述器

PropertyDescriptor[] descriptors = beanInfo.getPropertyDescriptors();//获取一个类中的所有属性描述器

**for**(PropertyDescriptor p : descriptors){

System.***out***.println(p.getReadMethod());//获取一个类中所有的get方法

}

}

}

**package** introspectorTest;

**import** java.text.SimpleDateFormat;

**import** java.util.Date;

**import** org.apache.commons.beanutils.BeanUtils;

**import** org.apache.commons.beanutils.ConvertUtils;

**import** org.apache.commons.beanutils.Converter;

/\*

BeanUtils：

BeanUtils主要解决 的问题： 把对象的属性数据封装 到对象中。

BeanUtils的好处：

1. BeanUtils设置属性值的时候，如果属性是基本数据 类型，BeanUtils会自动帮我转换数据类型。

2. BeanUtils设置属性值的时候底层也是依赖于get或者Set方法设置以及获取属性值的。

3. BeanUtils设置属性值,如果设置的属性是其他的引用 类型数据，那么这时候必须要注册一个类型转换器。

BeanUtilss使用的步骤：

1. 导包commons-logging.jar 、 commons-beanutils-1.8.0.jar

\*/

**public** **class** Demo3 {

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

//从文件中读取到的数据都是字符串的数据，或者是表单提交的数据获取到的时候也是字符串的数据。

String id ="110";

String name="ixenos";

String salary = "1000.0";

String birthday = "2013-12-10";//引用类型使用BeanUtils要注册类型转换器

//注册一个类型转换器

ConvertUtils.*register*(**new** Converter() {

@Override

**public** Object convert(@SuppressWarnings("rawtypes") Class type, Object value) { // type : 目前所遇到的数据类型。 value :目前参数的值。

Date date = **null**;

**try**{

SimpleDateFormat dateFormat = **new** SimpleDateFormat("yyyy-MM-dd");

date = dateFormat.parse((String)value); //手动格式化转回Date类型

}**catch**(Exception e){

e.printStackTrace();

}

**return** date;

}

}, Date.**class**);

//创建一个默认bean

Emp e = **new** Emp();

//填入bean状态

BeanUtils.*setProperty*(e, "id", id);

BeanUtils.*setProperty*(e, "name",name);

BeanUtils.*setProperty*(e, "salary",salary);

BeanUtils.*setProperty*(e, "birthday",birthday);

System.***out***.println(e);

}

}

**package** introspectorTest;

**import** java.util.Date;

**public** **class** Emp {

**private** **int** id;

**private** String name;

**private** **double** salary;

**private** Date birthday;

**public** Date getBirthday() {

**return** birthday;

}

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

**this**.birthday = birthday;

}

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **double** getSalary() {

**return** salary;

}

**public** **void** setSalary(**double** salary) {

**this**.salary = salary;

}

**public** Emp(**int** id, String name, **double** salary) {

**super**();

**this**.id = id;

**this**.name = name;

**this**.salary = salary;

}

**public** Emp(){}

@Override

**public** String toString() {

**return** "��ţ�"+**this**.id+" ������"+ **this**.name+ " нˮ��"+ **this**.salary+" ���գ�"+ birthday;

}

}

**package** introspectorTest;

//实体类--javabean

**public** **class** Person {

**private** **int** id;

**private** String name;

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** Person(**int** id, String name) {

**super**();

**this**.id = id;

**this**.name = name;

}

**public** Person(){

}

@Override

**public** String toString() {

**return** **this**.id + **this**.name;

}

}

以下是obj.txt

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

introspectorTest.Person

id=100

name=doggy