**工厂设计模式**

**一、概念**

简单工厂模式是用一个实际的工厂来创建所有你想创建的对象，如果创建的对象太多，那么反而不利于维护，所以还必须要分配责任，这就有了工厂方法模式，相比于简单工厂，它把那个实际工厂划分出来许多小工厂，每个小工厂负责创建一个产品（对象），为了可以扩展，你可以抽象出来一个抽象的工厂类，然后每个小工厂实现这个抽象工厂接口。其他和简单工厂没什么区别。

**二、使用时机**

如果在创建对象时，创建的对象多且负责的情况下就可以使用工厂方法模式

**三、优点和缺点**

增加一个产品很简单，只要添加一个创建这个产品的实际工厂和一个此产品就可以了，简单清晰。缺点是增加了代码量，每个产品都必须增加一个工厂，虽然有两个产品有某种联系，但必须要使用两个工厂来创建这两个产品。

四、工厂模式使用：

Properties prop = **new** Properties();//1.创建读取properties文件的对象

InputStream is = getProperties.**class**.getClassLoader().getResourceAsStream(fileName);

prop.load(is);

//prop.load(类名.class.getClassloader().getResourceAsStream(“文件问.properties”));2加载文件的位置

str = prop.getProperty(key);//3由于propertiest都是key value对应的，getProperty就是拿到其中值

**注意的一点是：**

**Static 中的方法和static属性，都不可能出现this,this就是对象自身引用，而static是静态区，对象是在堆区**

**类名：**

**不加包名的类名**

**完整类名： 包名+类名**

**Java发射机制：通过拿到一个类的字节码，然后可以做与这个类相关的所有操作**

1. **一个java类只会被加载一次，一个类只有一个字节码**
2. **Class代表在内存中的字节码**
3. **如何获取字节码:**
4. **Class cl = T.Class;**
5. **Class c2 = new T().getClass();**
6. **Class c3 = Class.forName(“完整的类名”);**

1. 在DBUtil中使用工厂模式是由于其中用户使用的数据库不同而导致其中MVC的Dao中使用的sql语言和注册sql和连接数据库不同

**package** com.ecjtu.util;

**import** java.io.IOException;

**import** java.io.InputStream;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.PreparedStatement;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**import** java.util.Properties;

**import** com.ecjtu.util.prop.getProperties;

**public** **class** DBUtil {

**private** **static** String *classNames* ;

**private** **static** String *url* ;

**private** **static** String *dbName* ;

**private** **static** String *username*;

**private** **static** String *password*;

**private** DBUtil(){}

**static**{

*classNames* = getProperties.*getValueKey*("JDBC.properties","classNames");

*url* = getProperties.*getValueKey*("JDBC.properties","url");

*dbName* = getProperties.*getValueKey*("JDBC.properties","dbName");

*username* = getProperties.*getValueKey*("JDBC.properties","username");

*password* = getProperties.*getValueKey*("JDBC.properties","password");

**try** {

Class.*forName*(*classNames*);

} **catch** (ClassNotFoundException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

**public** **static** Connection getConn(){

Connection conn = **null**;

**try** {

conn = DriverManager.*getConnection*(*url*+*dbName*,*username*,*password*);

} **catch** (SQLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**return** conn;

}

**public** **static** PreparedStatement getPstem(Connection conn,String sql){

PreparedStatement pstem = **null**;

**try** {

pstem = conn.prepareStatement(sql);

} **catch** (SQLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**return** pstem;

}

**public** **static** ResultSet getRs(PreparedStatement pstem){

ResultSet rs = **null**;

**try** {

rs = pstem.executeQuery();

} **catch** (SQLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**return** rs;

}

**public** **static** **void** getClose(Connection conn, PreparedStatement pstem,ResultSet rs){

**if**(rs !=**null**){

**try** {

rs.close();

} **catch** (SQLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

rs = **null**;

}

**if**(pstem !=**null**){

**try** {

pstem.close();

} **catch** (SQLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

pstem = **null**;

}

**if**(conn !=**null**){

**try** {

conn.close();

} **catch** (SQLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

conn = **null**;

}

}

}

2、由于有多个需要工厂模式，所以需要封装

package com.ecjtu.util.prop;

import java.io.IOException;

import java.io.InputStream;

import java.util.Properties;

public class getProperties {

public static String getValueKey(String filName,String key){

String str = null;

Properties prop = new Properties();

InputStream is = getProperties.class.getClassLoader().getResourceAsStream(filName);

try {

prop.load(is);

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}finally{

try {

is.close();

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

str = prop.getProperty(key);

return str;

}

}

3、properties文件

ProductDao4MySqlImpl=com.ecjtu.product.dao.impl.ProductDao4MySqlImpl

4、在MVC格式中service需要与dao中的对象

**private** **static** ProductDao *productDao* ;

**public** **static** ProductDao getProductDao(){

String p = getProperties.*getValueKey*("Product.properties", "ProductDao4MySqlImpl");

// System.out.println(p);

**try** {

*productDao* = (ProductDao) Class.*forName*(p).newInstance();

} **catch** (InstantiationException e1) {

// **TODO** Auto-generated catch block

e1.printStackTrace();

} **catch** (IllegalAccessException e1) {

// **TODO** Auto-generated catch block

e1.printStackTrace();

} **catch** (ClassNotFoundException e1) {

// **TODO** Auto-generated catch block

e1.printStackTrace();

}

**return** *productDao*;

}