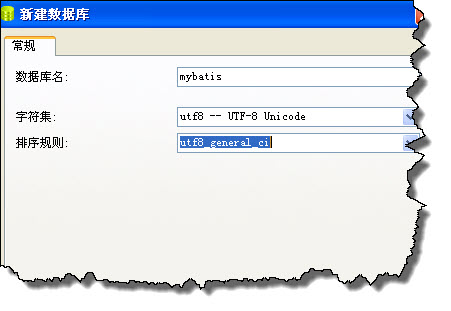
写在这个系列前面的话:

以前曾经用过ibatis,这是mybatis的前身，当时在做项目时，感觉很不错，比hibernate灵活。性能也比hibernate好。而且也比较轻量级，因为当时在项目中，没来的及做很很多笔记。后来项目结束了，我也没写总结文档。已经过去好久了。但最近突然又对这个ORM 工具感兴趣。因为接下来自己的项目中很有可能采用这个ORM工具。所以在此重新温习了一下 mybatis, 因此就有了这个系列的 mybatis 教程.  
  
什么是mybatis  
MyBatis是支持普通SQL查询，存储过程和高级映射的优秀持久层框架。MyBatis消除了几乎所有的JDBC代码和参数的手工设置以及结果集的检索。MyBatis使用简单的XML或注解用于配置和原始映射，将接口和Java的POJOs（Plan Old Java Objects，普通的Java对象）映射成数据库中的记录.  
  
orm工具的基本思想  
无论是用过的hibernate,mybatis,你都可以法相他们有一个共同点：  
1. 从配置文件(通常是XML配置文件中)得到 sessionfactory.  
2. 由sessionfactory  产生 session  
3. 在session 中完成对数据的增删改查和事务提交等.  
4. 在用完之后关闭session 。

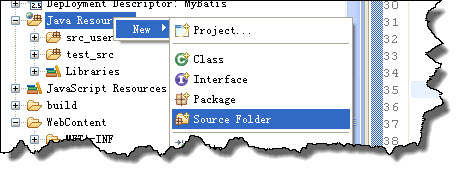
5. 在java 对象和 数据库之间有做mapping 的配置文件，也通常是xml 文件。

**mybatis实战教程(mybatis in action)之一：开发环境搭建**

mybatis 的开发环境搭建，选择: eclipse j2ee 版本，mysql 5.1 ,jdk 1.7,mybatis3.2.0.jar包。这些软件工具均可以到各自的官方网站上下载。  
首先建立一个名字为 MyBaits 的 dynamic web project   
1. 现阶段，你可以直接建立java 工程，但一般都是开发web项目，这个系列教程最后也是web的，所以一开始就建立web工程。  
2. 将 mybatis-3.2.0-SNAPSHOT.jar，mysql-connector-java-5.1.22-bin.jar 拷贝到 web工程的lib目录.  
3. 创建mysql 测试数据库和用户表,注意，这里采用的是 utf-8 编码  
  
创建用户表,并插入一条测试数据

程序代码 程序代码

Create TABLE `user` (  
  `id` int(11) NOT NULL AUTO\_INCREMENT,  
  `userName` varchar(50) DEFAULT NULL,  
  `userAge` int(11) DEFAULT NULL,  
  `userAddress` varchar(200) DEFAULT NULL,  
  PRIMARY KEY (`id`)  
) ENGINE=InnoDB AUTO\_INCREMENT=2 DEFAULT CHARSET=utf8;  
  
  
Insert INTO `user` VALUES ('1', 'summer', '100', 'shanghai,pudong');

到此为止，前期准备工作就完成了。下面开始真正配置mybatis项目了。  
1. 在MyBatis 里面创建两个源码目录，分别为 src\_user,test\_src, 用如下方式建立,鼠标右键点击 JavaResource.  
  
  
2. 设置mybatis 配置文件:Configuration.xml, 在src\_user目录下建立此文件，内容如下:

程序代码 程序代码

< ?xml version="1.0" encoding="UTF-8" ?>  
< !DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  
"<http://mybatis.org/dtd/mybatis-3-config.dtd>">  
< configuration>  
    <typeAliases>   
        <typeAlias alias="User" type="com.yihaomen.mybatis.model.User"/>   
    </typeAliases>   
  
    <environments default="development">  
        <environment id="development">  
        <transactionManager type="JDBC"/>  
            <dataSource type="POOLED">  
            <property name="driver" value="com.mysql.jdbc.Driver"/>  
            <property name="url" value="jdbc:mysql://127.0.0.1:3306/mybatis" />  
            <property name="username" value="root"/>  
            <property name="password" value="password"/>  
            </dataSource>  
        </environment>  
    </environments>  
      
    <mappers>  
        <mapper resource="com/yihaomen/mybatis/model/User.xml"/>  
    </mappers>  
< /configuration>

3. 建立与数据库对应的 java class,以及映射文件.  
在src\_user下建立package:com.yihaomen.mybatis.model ,并在这个 package 下建立 User 类:

程序代码 程序代码

package com.yihaomen.mybatis.model;  
  
public class User {  
      
    private int id;  
    private String userName;  
    private String userAge;  
    private String userAddress;  
      
    public int getId() {  
        return id;  
    }  
    public void setId(int id) {  
        this.id = id;  
    }  
    public String getUserName() {  
        return userName;  
    }  
    public void setUserName(String userName) {  
        this.userName = userName;  
    }  
    public String getUserAge() {  
        return userAge;  
    }  
    public void setUserAge(String userAge) {  
        this.userAge = userAge;  
    }  
    public String getUserAddress() {  
        return userAddress;  
    }  
    public void setUserAddress(String userAddress) {  
        this.userAddress = userAddress;  
    }  
  
}

同时建立这个User 的映射文件 User.xml:

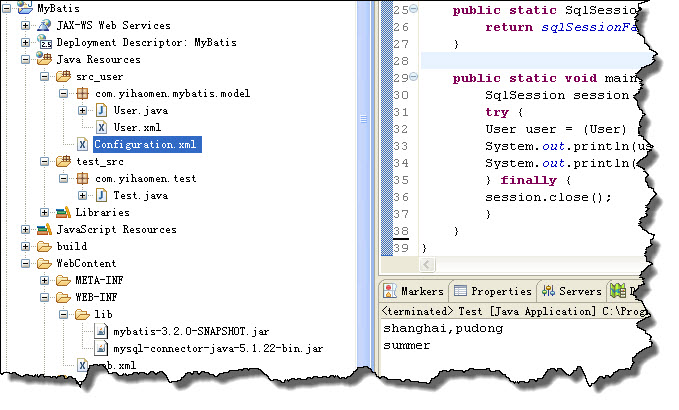
程序代码 程序代码

< ?xml version="1.0" encoding="UTF-8" ?>  
< !DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"   
"<http://mybatis.org/dtd/mybatis-3-mapper.dtd>">  
  
< mapper namespace="com.yihaomen.mybatis.models.UserMapper">  
    <select id="selectUserByID" parameterType="int" resultType="User">  
        select \* from `user` where id = #{id}  
    </select>  
< /mapper>

下面对这几个配置文件解释下：  
1.Configuration.xml 是 mybatis 用来建立 sessionFactory 用的，里面主要包含了数据库连接相关东西，还有 java 类所对应的别名，比如 <typeAlias alias="User" type="com.yihaomen.mybatis.model.User"/> 这个别名非常重要，你在 具体的类的映射中，比如User.xml 中 resultType 就是对应这里的。要保持一致，当然这里的 resultType 还有另外单独的定义方式，后面再说。  
2.  Configuration.xml 里面 的<mapper resource="com/yihaomen/mybatis/model/User.xml"/>是包含要映射的类的xml配置文件。  
3. 在User.xml 文件里面 主要是定义各种SQL 语句，以及这些语句的参数，以及要返回的类型等.  
  
开始测试  
在test\_src 源码目录下建立com.yihaomen.test这个package,并建立测试类Test:

程序代码 程序代码

package com.yihaomen.test;  
  
import java.io.Reader;  
  
import org.apache.ibatis.io.Resources;  
import org.apache.ibatis.session.SqlSession;  
import org.apache.ibatis.session.SqlSessionFactory;  
import org.apache.ibatis.session.SqlSessionFactoryBuilder;  
  
import com.yihaomen.mybatis.model.User;  
  
public class Test {  
    private static SqlSessionFactory sqlSessionFactory;  
    private static Reader reader;   
  
    static{  
        try{  
            reader    = Resources.getResourceAsReader("Configuration.xml");  
            sqlSessionFactory = new SqlSessionFactoryBuilder().build(reader);  
        }catch(Exception e){  
            e.printStackTrace();  
        }  
    }  
  
    public static SqlSessionFactory getSession(){  
        return sqlSessionFactory;  
    }  
      
    public static void main(String[] args) {  
        SqlSession session = sqlSessionFactory.openSession();  
        try {  
        User user = (User) session.selectOne("com.yihaomen.mybatis.models.UserMapper.selectUserByID", 1);  
        System.out.println(user.getUserAddress());  
        System.out.println(user.getUserName());  
        } finally {  
        session.close();  
        }  
    }  
}

现在运行这个程序，是不是得到查询结果了。恭喜你，环境搭建配置成功，接下来第二章，将讲述基于接口的操作方式，增删改查。  
整个工程目录结构如下:  


除非申明,文章均为[一号门](http://www.yihaomen.com/)原创,转载请注明本文地址,谢谢!

**mybatis实战教程(mybatis in action)之二：以接口的方式编程**

前面一章，已经搭建好了eclipse,mybatis,mysql的环境，并且实现了一个简单的查询。请注意，这种方式是用SqlSession实例来直接执行已映射的SQL语句：  
session.selectOne("com.yihaomen.mybatis.models.UserMapper.selectUserByID", 1)  
其实还有更简单的方法，而且是更好的方法，使用合理描述参数和SQL语句返回值的接口（比如IUserOperation.class），这样现在就可以至此那个更简单，更安全的代码，没有容易发生的字符串文字和转换的错误.下面是详细过程:  
  
在src\_user源码目录下建立 com.yihaomen.mybatis.inter 这个包，并建立接口类 IUserOperation , 内容如下：

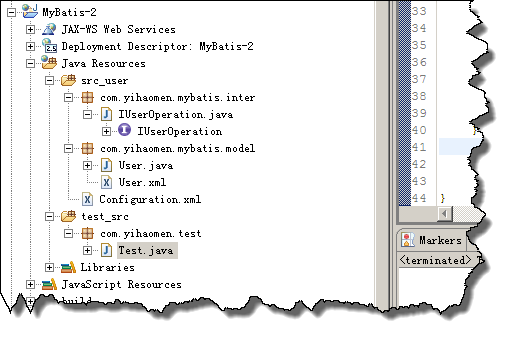
程序代码 程序代码

package com.yihaomen.mybatis.inter;  
import com.yihaomen.mybatis.model.User;  
  
public interface IUserOperation {      
    public User selectUserByID(int id);  
      
}

请注意，这里面有一个方法名 selectUserByID 必须与 User.xml 里面配置的 select 的id 对应（<select id="selectUserByID"）  
  
重写测试代码

程序代码 程序代码

public static void main(String[] args) {  
        SqlSession session = sqlSessionFactory.openSession();  
        try {  
            IUserOperation userOperation=session.getMapper(IUserOperation.class);  
            User user = userOperation.selectUserByID(1);  
            System.out.println(user.getUserAddress());  
            System.out.println(user.getUserName());  
        } finally {  
            session.close();  
        }  
    }

整个工程结构图现在如下：  
  
  
运行这个测试程序，就可以看到结果了。

除非申明,文章均为[一号门](http://www.yihaomen.com/)原创,转载请注明本文地址,谢谢!

**mybatis实战教程(mybatis in action)之三:实现数据的增删改查**

前面已经讲到用接口的方式编程。这种方式，要注意的一个地方就是。在User.xml  的配置文件中，mapper namespace="com.yihaomen.mybatis.inter.IUserOperation" ，命名空间非常重要，不能有错，必须与我们定义的package 和 接口一致。如果不一致就会出错,这一章主要在上一讲基于接口编程的基础上完成如下事情:  
1. 用 mybatis 查询数据，包括列表  
2. 用 mybatis 增加数据  
3. 用 mybatis 更新数据.  
4. 用 mybatis 删除数据.  
  
查询数据，前面已经讲过简单的，主要看查询出列表的  
查询出列表，也就是返回list, 在我们这个例子中也就是 List<User> , 这种方式返回数据，需要在User.xml 里面配置返回的类型 resultMap, 注意不是 resultType, 而这个resultMap 所对应的应该是我们自己配置的

程序代码 程序代码

< !-- 为了返回list 类型而定义的returnMap -->  
    <resultMap type="User" id="resultListUser">  
        <id column="id" property="id" />  
        <result column="userName" property="userName" />  
        <result column="userAge" property="userAge" />  
        <result column="userAddress" property="userAddress" />  
    </resultMap>

查询列表的语句在 User.xml 中

程序代码 程序代码

< !-- 返回list 的select 语句，注意 resultMap 的值是指向前面定义好的 -->  
    <select id="selectUsers" parameterType="string" resultMap="resultListUser">  
        select \* from user where userName like #{userName}  
    </select>

在 IUserOperation 接口中增加方法：public List<User> selectUsers(String userName);      
现在在 Test 类中做测试

程序代码 程序代码

public void getUserList(String userName){  
        SqlSession session = sqlSessionFactory.openSession();  
        try {  
            IUserOperation userOperation=session.getMapper(IUserOperation.class);            
            List<User> users = userOperation.selectUsers(userName);  
            for(User user:users){  
                System.out.println(user.getId()+":"+user.getUserName()+":"+user.getUserAddress());  
            }  
              
        } finally {  
            session.close();  
        }  
    }

现在在main  方法中可以测试：

程序代码 程序代码

public static void main(String[] args) {  
        Test testUser=new Test();  
        testUser.getUserList("%");  
    }

可以看到，结果成功查询出来。如果是查询单个数据的话，用第二讲用过的方法就可以了。  
  
用mybatis 增加数据   
在 IUserOperation 接口中增加方法：public void addUser(User user);  
在 User.xml 中配置

程序代码 程序代码

< !--执行增加操作的SQL语句。id和parameterType    
       分别与IUserOperation接口中的addUser方法的名字和    
       参数类型一致。以#{name}的形式引用Student参数    
       的name属性，MyBatis将使用反射读取Student参数    
       的此属性。#{name}中name大小写敏感。引用其他    
       的gender等属性与此一致。seGeneratedKeys设置    
       为"true"表明要MyBatis获取由数据库自动生成的主    
       键；keyProperty="id"指定把获取到的主键值注入    
       到Student的id属性-->   
    <insert id="addUser" parameterType="User"   
        useGeneratedKeys="true" keyProperty="id">   
        insert into user(userName,userAge,userAddress)    
             values(#{userName},#{userAge},#{userAddress})    
    </insert>

然后在 Test 中写测试方法:

程序代码 程序代码

/\*\*  
     \* 测试增加,增加后，必须提交事务，否则不会写入到数据库.  
     \*/  
    public void addUser(){  
        User user=new User();  
        user.setUserAddress("人民广场");  
        user.setUserName("飞鸟");  
        user.setUserAge(80);  
        SqlSession session = sqlSessionFactory.openSession();  
        try {  
            IUserOperation userOperation=session.getMapper(IUserOperation.class);  
            userOperation.addUser(user);  
            session.commit();  
            System.out.println("当前增加的用户 id为:"+user.getId());  
        } finally {  
            session.close();  
        }  
    }

用mybatis 更新数据  
方法类似，先在 IUserOperation 中增加方法：public void addUser(User user);  
然后配置 User.xml

程序代码 程序代码

<update id="updateUser" parameterType="User" >  
        update user set userName=#{userName},userAge=#{userAge},userAddress=#{userAddress} where id=#{id}  
    </update>

Test 类总的测试方法如下：

程序代码 程序代码

public void updateUser(){  
        //先得到用户,然后修改，提交。  
        SqlSession session = sqlSessionFactory.openSession();  
        try {  
            IUserOperation userOperation=session.getMapper(IUserOperation.class);  
            User user = userOperation.selectUserByID(4);              
            user.setUserAddress("原来是魔都的浦东创新园区");  
            userOperation.updateUser(user);  
            session.commit();  
              
        } finally {  
            session.close();  
        }  
    }

用mybatis 删除数据   
同理，IUserOperation 增加方法：public void deleteUser(int id);  
配置User.xml

程序代码 程序代码

<delete id="deleteUser" parameterType="int">  
        delete from user where id=#{id}  
    </delete>

然后在Test类中写测试方法:

程序代码 程序代码

    /\*\*  
     \* 删除数据，删除一定要 commit.  
     \* @param id  
     \*/  
    public void deleteUser(int id){  
        SqlSession session = sqlSessionFactory.openSession();  
        try {  
            IUserOperation userOperation=session.getMapper(IUserOperation.class);            
            userOperation.deleteUser(id);  
            session.commit();              
        } finally {  
            session.close();  
        }  
    }

这样，所有增删改查都完成了，注意在增加，更改，删除的时候要调用session.commit()，这样才会真正对数据库进行操作，否则是没有提交的。  
  
到此为止，简单的单表操作，应该都会了，接下来的时间了，我会讲多表联合查询，以及结果集的选取。 

除非申明,文章均为[一号门](http://www.yihaomen.com/)原创,转载请注明本文地址,谢谢!

**mybatis实战教程(mybatis in action)之四:实现关联数据的查询**

有了前面几章的基础，对一些简单的应用是可以处理的，但在实际项目中，经常是关联表的查询，比如最常见到的多对一，一对多等。这些查询是如何处理的呢，这一讲就讲这个问题。我们首先创建一个Article 这个表，并初始化数据.

程序代码 程序代码

Drop TABLE IF EXISTS `article`;  
Create TABLE `article` (  
  `id` int(11) NOT NULL auto\_increment,  
  `userid` int(11) NOT NULL,  
  `title` varchar(100) NOT NULL,  
  `content` text NOT NULL,  
  PRIMARY KEY  (`id`)  
) ENGINE=InnoDB AUTO\_INCREMENT=5 DEFAULT CHARSET=utf8;  
  
-- ----------------------------  
-- 添加几条测试数据  
-- ----------------------------  
Insert INTO `article` VALUES ('1', '1', 'test\_title', 'test\_content');  
Insert INTO `article` VALUES ('2', '1', 'test\_title\_2', 'test\_content\_2');  
Insert INTO `article` VALUES ('3', '1', 'test\_title\_3', 'test\_content\_3');  
Insert INTO `article` VALUES ('4', '1', 'test\_title\_4', 'test\_content\_4');

你应该发现了，这几个文章对应的userid都是1，所以需要用户表user里面有id=1的数据。可以修改成满足自己条件的数据.按照orm的规则，表已经创建了，那么肯定需要一个对象与之对应，所以我们增加一个 Article 的class

程序代码 程序代码

package com.yihaomen.mybatis.model;  
  
public class Article {  
      
    private int id;  
    private User user;  
    private String title;  
    private String content;  
      
    public int getId() {  
        return id;  
    }  
    public void setId(int id) {  
        this.id = id;  
    }  
      
    public User getUser() {  
        return user;  
    }  
    public void setUser(User user) {  
        this.user = user;  
    }  
    public String getTitle() {  
        return title;  
    }  
    public void setTitle(String title) {  
        this.title = title;  
    }  
    public String getContent() {  
        return content;  
    }  
    public void setContent(String content) {  
        this.content = content;  
    }  
  
}

注意一下，文章的用户是怎么定义的，是直接定义的一个User对象。而不是int类型。  
  
多对一的实现  
场景:在读取某个用户发表的所有文章。当然还是需要在User.xml 里面配置 select 语句, 但重点是这个 select 的resultMap 对应什么样的数据呢。这是重点，这里要引入 association 看定义如下:

程序代码 程序代码

< !-- User 联合文章进行查询 方法之一的配置 (多对一的方式)  -->      
    <resultMap id="resultUserArticleList" type="Article">  
        <id property="id" column="aid" />  
        <result property="title" column="title" />  
        <result property="content" column="content" />  
          
        <association property="user" javaType="User">  
            <id property="id" column="id" />  
            <result property="userName" column="userName" />  
            <result property="userAddress" column="userAddress" />              
        </association>          
    </resultMap>  
  
< select id="getUserArticles" parameterType="int" resultMap="resultUserArticleList">  
       select user.id,user.userName,user.userAddress,article.id aid,article.title,article.content from user,article  
              where user.id=article.userid and user.id=#{id}  
    </select>

这样配置之后，就可以了，将select 语句与resultMap 对应的映射结合起来看，就明白了。用association 来得到关联的用户，这是多对一的情况，因为所有的文章都是同一个用户的。  
  
还有另外一种处理方式，可以复用我们前面已经定义好的 resultMap ,前面我们定义过一个 resultListUser ,看这第二种方法如何实现:

程序代码 程序代码

<resultMap type="User" id="resultListUser">  
        <id column="id" property="id" />  
        <result column="userName" property="userName" />  
        <result column="userAge" property="userAge" />  
        <result column="userAddress" property="userAddress" />  
    </resultMap>  
  
    <!-- User 联合文章进行查询 方法之二的配置 (多对一的方式) -->      
    <resultMap id="resultUserArticleList-2" type="Article">  
        <id property="id" column="aid" />  
        <result property="title" column="title" />  
        <result property="content" column="content" />          
        <association property="user" javaType="User" resultMap="resultListUser" />              
    </resultMap>  
      
    <select id="getUserArticles" parameterType="int" resultMap="resultUserArticleList">  
       select user.id,user.userName,user.userAddress,article.id aid,article.title,article.content from user,article  
              where user.id=article.userid and user.id=#{id}  
    </select>

将 association  中对应的映射独立抽取出来，可以达到复用的目的。  
  
好了，现在在Test 类中写测试代码:

程序代码 程序代码

public void getUserArticles(int userid){  
        SqlSession session = sqlSessionFactory.openSession();  
        try {  
            IUserOperation userOperation=session.getMapper(IUserOperation.class);            
            List<Article> articles = userOperation.getUserArticles(userid);  
            for(Article article:articles){  
                System.out.println(article.getTitle()+":"+article.getContent()+  
                        ":作者是:"+article.getUser().getUserName()+":地址:"+  
                         article.getUser().getUserAddress());  
            }  
        } finally {  
            session.close();  
        }  
    }

漏掉了一点，我们一定要在 IUserOperation 接口中，加入 select 对应的id 名称相同的方法：  
public List<Article> getUserArticles(int id);  
  
然后运行就可以测试。  
  
整个程序下载：  
下载文件[点击下载此文件](http://www.yihaomen.com/attachments/month_1211/i20121115164851.rar)

除非申明,文章均为[一号门](http://www.yihaomen.com/)原创,转载请注明本文地址,谢谢!

**mybatis实战教程(mybatis in action)之五:与spring3集成(附源码)**

在这一系列文章中，前面讲到纯粹用mybatis 连接数据库，然后 进行增删改查，以及多表联合查询的的例子，但实际项目中，通常会用 spring 这个沾合剂来管理 datasource 等。充分利用spring 基于接口的编程，以及aop ,ioc 带来的方便。用spring 来管理 mybatis 与管理hibernate 有很多类似的地方。今天的重点就是数据源管理以及 bean的配置。  
  
你可以下载源码后，对比着看，源代码没有带jar包，太大了，空间有限. 有截图，你可以看到用到哪些jar包，源码在本文最后.  
  
1. 首先对前面的工程结构做一点改变，在src\_user源代码目录下建立文件夹config ,并将原来的 mybatis 配置文件 Configuration.xml 移动到这个文件夹中, 并在config 文家夹中建立 spring 配置文件：applicationContext.xml ，这个配置文件里最主要的配置：

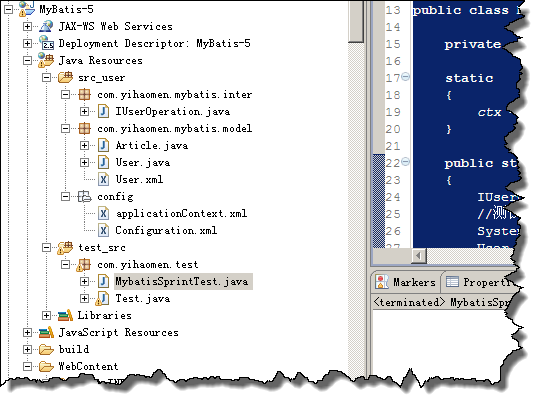
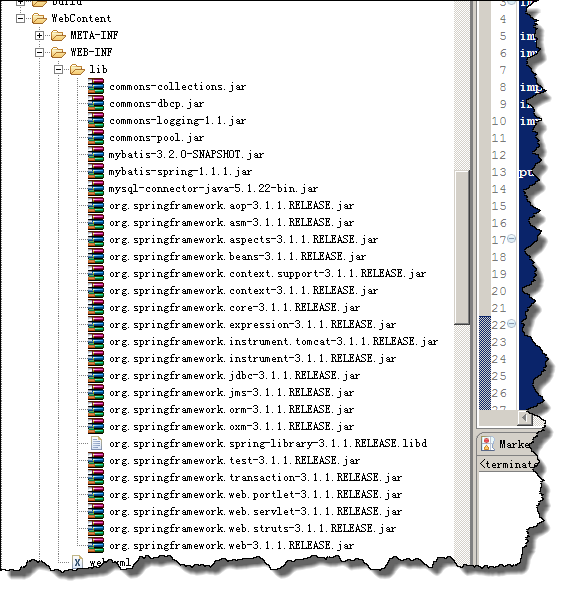
程序代码 程序代码

< !--本示例采用DBCP连接池，应预先把DBCP的jar包复制到工程的lib目录下。 -->   
  
  <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">  
     <property name="driverClassName" value="com.mysql.jdbc.Driver"/>   
     <property name="url" value="jdbc:mysql://127.0.0.1:3306/mybatis?characterEncoding=utf8"/>  
     <property name="username" value="root"/>   
     <property name="password" value="password"/>   
  </bean>   
  
  <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">   
     <!--dataSource属性指定要用到的连接池-->   
     <property name="dataSource" ref="dataSource"/>   
     <!--configLocation属性指定mybatis的核心配置文件-->   
     <property name="configLocation" value="config/Configuration.xml"/>   
  </bean>   
  
  <bean id="userMapper" class="org.mybatis.spring.mapper.MapperFactoryBean">   
     <!--sqlSessionFactory属性指定要用到的SqlSessionFactory实例-->   
     <property name="sqlSessionFactory" ref="sqlSessionFactory" />   
     <!--mapperInterface属性指定映射器接口，用于实现此接口并生成映射器对象-->   
     <property name="mapperInterface" value="com.yihaomen.mybatis.inter.IUserOperation" />  
  </bean>

[b]这里面的重点就是 org.mybatis.spring.SqlSessionFactoryBean 与 org.mybatis.spring.mapper.MapperFactoryBean[b] 实现了 spring  的接口，并产生对象。详细可以查看 mybatis-spring 代码。（<http://code.google.com/p/mybatis/>）,如果仅仅使用，固定模式，这样配置就好。  
  
然后写测试程序

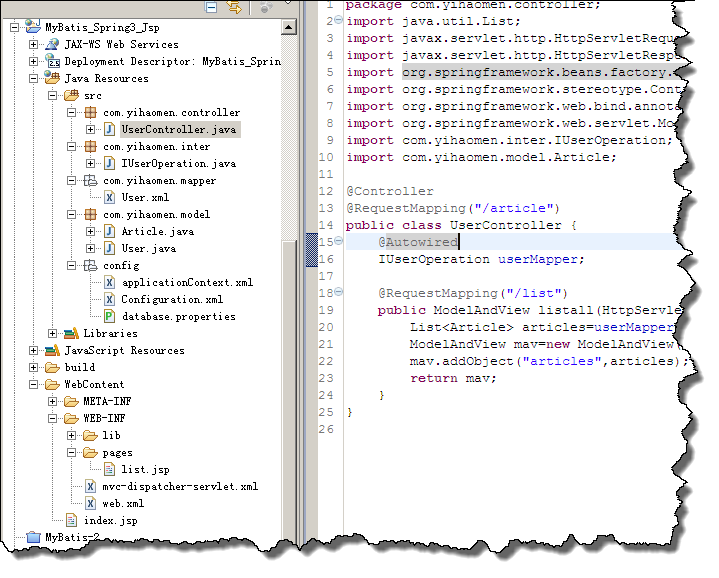
程序代码 程序代码

package com.yihaomen.test;  
  
import java.util.List;  
  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
import com.yihaomen.mybatis.inter.IUserOperation;  
import com.yihaomen.mybatis.model.Article;  
import com.yihaomen.mybatis.model.User;  
  
  
public class MybatisSprintTest {  
      
    private static ApplicationContext ctx;    
      
    static   
    {    
        ctx = new ClassPathXmlApplicationContext("config/applicationContext.xml");   
    }          
        
    public static void main(String[] args)    
    {    
        IUserOperation mapper = (IUserOperation)ctx.getBean("userMapper");   
        //测试id=1的用户查询，根据数据库中的情况，可以改成你自己的.  
        System.out.println("得到用户id=1的用户信息");  
        User user = mapper.selectUserByID(1);  
        System.out.println(user.getUserAddress());   
          
        //得到文章列表测试  
        System.out.println("得到用户id为1的所有文章列表");  
        List<Article> articles = mapper.getUserArticles(1);  
          
        for(Article article:articles){  
            System.out.println(article.getContent()+"--"+article.getTitle());  
        }  
          
    }    
  
      
}

运行即可得到相应的结果.  
工程图：  
  
用到的jar包，如下图：  
  
  
源代码下载，不带 jar  包，  
下载文件[点击下载此文件](http://www.yihaomen.com/attachments/month_1211/a20121119173723.rar)

除非申明,文章均为[一号门](http://www.yihaomen.com/)原创,转载请注明本文地址,谢谢!

**mybatis实战教程(mybatis in action)之六:与Spring MVC 的集成**

前面几篇文章已经讲到了mybatis与spring 的集成。但这个时候，所有的工程还不是web工程，虽然我一直是创建的web 工程。今天将直接用mybatis与Spring mvc 的方式集成起来，源码在本文结尾处下载.主要有以下几个方面的配置  
1. web.xml 配置 spring dispatchservlet ,比如为:mvc-dispatcher  
2. mvc-dispatcher-servlet.xml 文件配置  
3. spring applicationContext.XML文件配置(与数据库相关，与mybatis sqlSessionFaction 整合，扫描所有mybatis mapper 文件等.)  
4. 编写controller 类  
5. 编写页面代码.  
先有个大概映像，整个工程图如下:  
[/code]  
  
1. web.xml 配置 spring dispatchservlet ,比如为:mvc-dispatcher

程序代码 程序代码

<context-param>  
    <param-name>contextConfigLocation</param-name>  
    <param-value>classpath\*:config/applicationContext.xml</param-value>  
  </context-param>  
  <listener>  
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  
  </listener>  
  <listener>  
    <listener-class>  
            org.springframework.web.context.ContextCleanupListener</listener-class>  
  </listener>  
  <servlet>  
    <servlet-name>mvc-dispatcher</servlet-name>  
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  
    <load-on-startup>1</load-on-startup>  
  </servlet>  
  <servlet-mapping>  
    <servlet-name>mvc-dispatcher</servlet-name>  
    <url-pattern>/</url-pattern>  
  </servlet-mapping>

2. 在web.xml 同目录下配置 mvc-dispatcher-servlet.xml 文件,这个文件名前面部分必须与你在web.xml里面配置的DispatcherServlet 的servlet名字对应.其内容为:

程序代码 程序代码

<beans xmlns="<http://www.springframework.org/schema/beans>"  
    xmlns:context="<http://www.springframework.org/schema/context>"  
    xmlns:mvc="<http://www.springframework.org/schema/mvc>" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"  
    xsi:schemaLocation="  
        <http://www.springframework.org/schema/beans>       
        <http://www.springframework.org/schema/beans>/spring-beans-3.0.xsd  
        <http://www.springframework.org/schema/context>   
        <http://www.springframework.org/schema/context>/spring-context-3.0.xsd  
        <http://www.springframework.org/schema/mvc>  
        <http://www.springframework.org/schema/mvc>/spring-mvc-3.0.xsd">  
  
    <context:component-scan base-package="com.yihaomen.controller" />  
    <mvc:annotation-driven />  
      
    <mvc:resources mapping="/static/\*\*" location="/WEB-INF/static/"/>    
    <mvc:default-servlet-handler/>    
       
    <bean  
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">  
        <property name="prefix">  
            <value>/WEB-INF/pages/</value>  
        </property>  
        <property name="suffix">  
            <value>.jsp</value>  
        </property>  
    </bean>  
  
< /beans>

3. 在源码目录 config 目录下配置 spring 配置文件 applicationContext.xml

程序代码 程序代码

< !--本示例采用DBCP连接池，应预先把DBCP的jar包复制到工程的lib目录下。 -->     
    <context:property-placeholder    location="classpath:/config/database.properties" />  
          
    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"  
        destroy-method="close" p:driverClassName="com.mysql.jdbc.Driver"  
        p:url="jdbc:mysql://127.0.0.1:3306/mybatis?characterEncoding=utf8"   
        p:username="root" p:password="password"  
        p:maxActive="10" p:maxIdle="10">  
    </bean>  
      
    <bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">  
      <property name="dataSource" ref="dataSource" />  
    </bean>  
      
       
  <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">   
     <!--dataSource属性指定要用到的连接池-->   
     <property name="dataSource" ref="dataSource"/>   
     <!--configLocation属性指定mybatis的核心配置文件-->   
     <property name="configLocation" value="classpath:config/Configuration.xml" />  
     <!-- 所有配置的mapper文件 -->  
     <property name="mapperLocations" value="classpath\*:com/yihaomen/mapper/\*.xml" />  
  </bean>   
    
  <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">  
     <property name="basePackage" value="com.yihaomen.inter" />       
  </bean>

不知道为什么，一旦我用了 MapperScannerConfigurer 去扫描所有的mapper 接口时，数据库配置datasource 就不能用读取database.properties文件了。报错： Cannot load JDBC driver class '${jdbc.driverClassName}'，网上有人说在spring 3.1.1 下用 sqlSessionFactionBean 注入可以解决，但我用 spring 3.1.3 还是有问题，所以只好把数据库连接信息直接配置在了XML 文件里面。  
  
4. 编写 controller 层

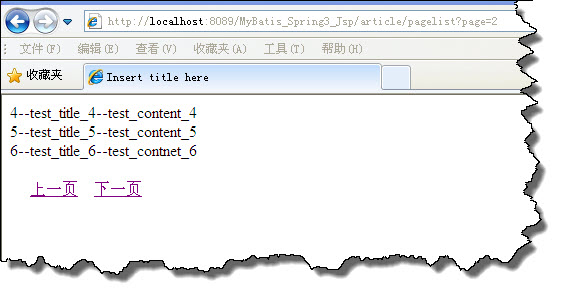
程序代码 程序代码

package com.yihaomen.controller;  
import java.util.List;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.servlet.ModelAndView;  
import com.yihaomen.inter.IUserOperation;  
import com.yihaomen.model.Article;  
  
@Controller  
@RequestMapping("/article")  
public class UserController {  
    @Autowired  
    IUserOperation userMapper;  
  
    @RequestMapping("/list")  
    public ModelAndView listall(HttpServletRequest request,HttpServletResponse response){  
        List<Article> articles=userMapper.getUserArticles(1);   
        ModelAndView mav=new ModelAndView("list");  
        mav.addObject("articles",articles);  
        return mav;  
    }  
}  
  
5. 页面文件:  
[code]  
< c:forEach items="${articles}" var="item">    
        ${item.id }--${item.title }--${item.content }<br />  
    </c:forEach>

运行结果：  
  
  
当然还有 mybatis 的Configure.xml  配置文件，与上一讲的差不多，唯一不同的就是不用再配置类似如下的：   <mapper resource="com/yihaomen/mapper/User.xml"/> ，所有这些都交给 在配置 sqlSessionFactory 的时候，由  <property name="mapperLocations" value="classpath\*:com/yihaomen/mapper/\*.xml" /> 去导入了。  
      
源码下载：  
  
下载文件[mybatis spring3 MVC 程序下载](http://www.yihaomen.com/attachments/month_1211/x2012112717434.rar)  
  
数据库下载：  
  
下载文件[spring mvc 数据库测试文件](http://www.yihaomen.com/attachments/month_1211/62012112717854.rar)

除非申明,文章均为[一号门](http://www.yihaomen.com/)原创,转载请注明本文地址,谢谢!

**mybatis实战教程(mybatis in action)之七:实现mybatis分页(源码下载)**

上一篇文章里已经讲到了mybatis与spring MVC的集成，并且做了一个列表展示，显示出所有article 列表，但没有用到分页，在实际的项目中，分页是肯定需要的。而且是物理分页，不是内存分页。对于物理分页方案，不同的数据库，有不同的实现方法,对于mysql 来说 就是利用 limit offset,pagesize 方式来实现的。oracle 是通过rownum 来实现的，如果你熟悉相关数据库的操作，是一样的很好扩展，本文以mysql 为例子来讲述.先看一下效果图(源代码在文章最后提供下载):  
  
实现mybatis 物理分页，一个最简单的方式是，是在你的mapper的SQL语句中直接写类似如下方式 :

程序代码 程序代码

<select id="getUserArticles" parameterType="Your\_params" resultMap="resultUserArticleList">  
       select user.id,user.userName,user.userAddress,article.id aid,article.title,article.content from user,article  
              where user.id=article.userid and user.id=#{id} limit #{offset},#{pagesize}  
    </select>

请注意这里的 parameterType 是你传入的参数类，或者map ，里面包含了offset,pagesize ,和其他你需要的参数，用这种方式，肯定可以实现分页。这是简单的一种方式。但更通用的一种方式是用 mybatis 插件的方式. 参考了网上的很多资料 ，mybatis plugin 方面的资料。写自己的插件.

程序代码 程序代码

package com.yihaomen.util;  
  
import java.lang.reflect.Field;  
import java.sql.Connection;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.util.List;  
import java.util.Map;  
import java.util.Properties;  
  
import javax.xml.bind.PropertyException;  
  
import org.apache.ibatis.builder.xml.dynamic.ForEachSqlNode;  
import org.apache.ibatis.executor.ErrorContext;  
import org.apache.ibatis.executor.Executor;  
import org.apache.ibatis.executor.ExecutorException;  
import org.apache.ibatis.executor.statement.BaseStatementHandler;  
import org.apache.ibatis.executor.statement.RoutingStatementHandler;  
import org.apache.ibatis.executor.statement.StatementHandler;  
import org.apache.ibatis.mapping.BoundSql;  
import org.apache.ibatis.mapping.MappedStatement;  
import org.apache.ibatis.mapping.ParameterMapping;  
import org.apache.ibatis.mapping.ParameterMode;  
import org.apache.ibatis.plugin.Interceptor;  
import org.apache.ibatis.plugin.Intercepts;  
import org.apache.ibatis.plugin.Invocation;  
import org.apache.ibatis.plugin.Plugin;  
import org.apache.ibatis.plugin.Signature;  
import org.apache.ibatis.reflection.MetaObject;  
import org.apache.ibatis.reflection.property.PropertyTokenizer;  
import org.apache.ibatis.session.Configuration;  
import org.apache.ibatis.session.ResultHandler;  
import org.apache.ibatis.session.RowBounds;  
import org.apache.ibatis.type.TypeHandler;  
import org.apache.ibatis.type.TypeHandlerRegistry;  
  
@Intercepts({ @Signature(type = StatementHandler.class, method = "prepare", args = { Connection.class }) })  
public class PagePlugin implements Interceptor {  
  
    private static String dialect = "";  
    private static String pageSqlId = "";  
  
    @SuppressWarnings("unchecked")  
    public Object intercept(Invocation ivk) throws Throwable {  
  
        if (ivk.getTarget() instanceof RoutingStatementHandler) {  
            RoutingStatementHandler statementHandler = (RoutingStatementHandler) ivk  
                    .getTarget();  
            BaseStatementHandler delegate = (BaseStatementHandler) ReflectHelper  
                    .getValueByFieldName(statementHandler, "delegate");  
            MappedStatement mappedStatement = (MappedStatement) ReflectHelper  
                    .getValueByFieldName(delegate, "mappedStatement");  
  
            if (mappedStatement.getId().matches(pageSqlId)) {  
                BoundSql boundSql = delegate.getBoundSql();  
                Object parameterObject = boundSql.getParameterObject();  
                if (parameterObject == null) {  
                    throw new NullPointerException("parameterObject error");  
                } else {  
                    Connection connection = (Connection) ivk.getArgs()[0];  
                    String sql = boundSql.getSql();  
                    String countSql = "select count(0) from (" + sql + ") myCount";  
                    System.out.println("总数sql 语句:"+countSql);  
                    PreparedStatement countStmt = connection  
                            .prepareStatement(countSql);  
                    BoundSql countBS = new BoundSql(  
                            mappedStatement.getConfiguration(), countSql,  
                            boundSql.getParameterMappings(), parameterObject);  
                    setParameters(countStmt, mappedStatement, countBS,  
                            parameterObject);  
                    ResultSet rs = countStmt.executeQuery();  
                    int count = 0;  
                    if (rs.next()) {  
                        count = rs.getInt(1);  
                    }  
                    rs.close();  
                    countStmt.close();  
  
                    PageInfo page = null;  
                    if (parameterObject instanceof PageInfo) {  
                        page = (PageInfo) parameterObject;  
                        page.setTotalResult(count);  
                    } else if(parameterObject instanceof Map){  
                        Map<String, Object> map = (Map<String, Object>)parameterObject;  
                        page = (PageInfo)map.get("page");  
                        if(page == null)  
                            page = new PageInfo();  
                        page.setTotalResult(count);  
                    }else {  
                        Field pageField = ReflectHelper.getFieldByFieldName(  
                                parameterObject, "page");  
                        if (pageField != null) {  
                            page = (PageInfo) ReflectHelper.getValueByFieldName(  
                                    parameterObject, "page");  
                            if (page == null)  
                                page = new PageInfo();  
                            page.setTotalResult(count);  
                            ReflectHelper.setValueByFieldName(parameterObject,  
                                    "page", page);  
                        } else {  
                            throw new NoSuchFieldException(parameterObject  
                                    .getClass().getName());  
                        }  
                    }  
                    String pageSql = generatePageSql(sql, page);  
                    System.out.println("page sql:"+pageSql);  
                    ReflectHelper.setValueByFieldName(boundSql, "sql", pageSql);  
                }  
            }  
        }  
        return ivk.proceed();  
    }  
  
    private void setParameters(PreparedStatement ps,  
            MappedStatement mappedStatement, BoundSql boundSql,  
            Object parameterObject) throws SQLException {  
        ErrorContext.instance().activity("setting parameters")  
                .object(mappedStatement.getParameterMap().getId());  
        List<ParameterMapping> parameterMappings = boundSql  
                .getParameterMappings();  
        if (parameterMappings != null) {  
            Configuration configuration = mappedStatement.getConfiguration();  
            TypeHandlerRegistry typeHandlerRegistry = configuration  
                    .getTypeHandlerRegistry();  
            MetaObject metaObject = parameterObject == null ? null  
                    : configuration.newMetaObject(parameterObject);  
            for (int i = 0; i < parameterMappings.size(); i++) {  
                ParameterMapping parameterMapping = parameterMappings.get(i);  
                if (parameterMapping.getMode() != ParameterMode.OUT) {  
                    Object value;  
                    String propertyName = parameterMapping.getProperty();  
                    PropertyTokenizer prop = new PropertyTokenizer(propertyName);  
                    if (parameterObject == null) {  
                        value = null;  
                    } else if (typeHandlerRegistry  
                            .hasTypeHandler(parameterObject.getClass())) {  
                        value = parameterObject;  
                    } else if (boundSql.hasAdditionalParameter(propertyName)) {  
                        value = boundSql.getAdditionalParameter(propertyName);  
                    } else if (propertyName  
                            .startsWith(ForEachSqlNode.ITEM\_PREFIX)  
                            && boundSql.hasAdditionalParameter(prop.getName())) {  
                        value = boundSql.getAdditionalParameter(prop.getName());  
                        if (value != null) {  
                            value = configuration.newMetaObject(value)  
                                    .getValue(  
                                            propertyName.substring(prop  
                                                    .getName().length()));  
                        }  
                    } else {  
                        value = metaObject == null ? null : metaObject  
                                .getValue(propertyName);  
                    }  
                    TypeHandler typeHandler = parameterMapping.getTypeHandler();  
                    if (typeHandler == null) {  
                        throw new ExecutorException(  
                                "There was no TypeHandler found for parameter "  
                                        + propertyName + " of statement "  
                                        + mappedStatement.getId());  
                    }  
                    typeHandler.setParameter(ps, i + 1, value,  
                            parameterMapping.getJdbcType());  
                }  
            }  
        }  
    }  
  
  
    private String generatePageSql(String sql, PageInfo page) {  
        if (page != null && (dialect !=null || !dialect.equals(""))) {  
            StringBuffer pageSql = new StringBuffer();  
            if ("mysql".equals(dialect)) {  
                pageSql.append(sql);  
                pageSql.append(" limit " + page.getCurrentResult() + ","  
                        + page.getShowCount());  
            } else if ("oracle".equals(dialect)) {  
                pageSql.append("select \* from (select tmp\_tb.\*,ROWNUM row\_id from (");  
                pageSql.append(sql);  
                pageSql.append(")  tmp\_tb where ROWNUM<=");  
                pageSql.append(page.getCurrentResult() + page.getShowCount());  
                pageSql.append(") where row\_id>");  
                pageSql.append(page.getCurrentResult());  
            }  
            return pageSql.toString();  
        } else {  
            return sql;  
        }  
    }  
  
    public Object plugin(Object arg0) {  
        // TODO Auto-generated method stub  
        return Plugin.wrap(arg0, this);  
    }  
  
    public void setProperties(Properties p) {  
        dialect = p.getProperty("dialect");  
        if (dialect ==null || dialect.equals("")) {  
            try {  
                throw new PropertyException("dialect property is not found!");  
            } catch (PropertyException e) {  
                // TODO Auto-generated catch block  
                e.printStackTrace();  
            }  
        }  
        pageSqlId = p.getProperty("pageSqlId");  
        if (dialect ==null || dialect.equals("")) {  
            try {  
                throw new PropertyException("pageSqlId property is not found!");  
            } catch (PropertyException e) {  
                // TODO Auto-generated catch block  
                e.printStackTrace();  
            }  
        }  
    }  
  
}

此插件有两个辅助类:PageInfo,ReflectHelper,你可以下载源代码参考。  
写了插件之后，当然需要在 mybatis 的配置文件Configuration.xml 里配置这个插件

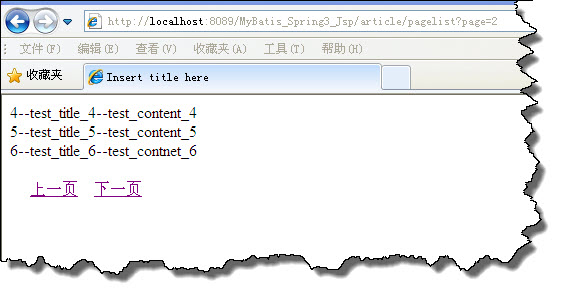
程序代码 程序代码

    <plugins>  
        <plugin interceptor="com.yihaomen.util.PagePlugin">  
            <property name="dialect" value="mysql" />  
            <property name="pageSqlId" value=".\*ListPage.\*" />  
        </plugin>  
    </plugins>

请注意，这个插件定义了一个规则，也就是在mapper中sql语句的id 必须包含ListPage才能被拦截。否则将不会分页处理.  
  
插件写好了，现在就可以在 spring mvc 中的controller 层中写一个方法来测试这个分页:

程序代码 程序代码

@RequestMapping("/pagelist")  
    public ModelAndView pageList(HttpServletRequest request,HttpServletResponse response){  
        int currentPage = request.getParameter("page")==null?1:Integer.parseInt(request.getParameter("page"));  
        int pageSize = 3;  
        if (currentPage<=0){  
            currentPage =1;  
        }  
        int currentResult = (currentPage-1) \* pageSize;  
          
        System.out.println(request.getRequestURI());  
        System.out.println(request.getQueryString());  
          
        PageInfo page = new PageInfo();  
        page.setShowCount(pageSize);  
        page.setCurrentResult(currentResult);  
        List<Article> articles=iUserOperation.selectArticleListPage(page,1);  
          
        System.out.println(page);  
          
        int totalCount = page.getTotalResult();  
          
        int lastPage=0;  
        if (totalCount % pageSize==0){  
            lastPage = totalCount % pageSize;  
        }  
        else{  
            lastPage =1+ totalCount / pageSize;  
        }  
          
        if (currentPage>=lastPage){  
            currentPage =lastPage;  
        }  
          
        String pageStr = "";  
  
        pageStr=String.format("<a href=\"%s\">上一页</a>    <a href=\"%s\">下一页</a>",  
                        request.getRequestURI()+"?page="+(currentPage-1),request.getRequestURI()+"?page="+(currentPage+1) );  
  
  
        //制定视图，也就是list.jsp  
        ModelAndView mav=new ModelAndView("list");  
        mav.addObject("articles",articles);  
        mav.addObject("pageStr",pageStr);  
        return mav;  
    }

然后运行程序，进入分页页面，你就可以看到结果了:  
  
  
源代码下载:   
下载文件[点击下载此文件](http://www.yihaomen.com/attachments/month_1212/w201212920237.rar)  
相关jar 包下载，请到下载这里例子中的jar  
<http://www.yihaomen.com/article/java/318.htm> (文章最后有源代码下载，里面有jar 包,拷贝到上面源代码里面所需要的lib 目录下.)  
  
另外，你还得在前面提到的数据库artilce表里面，多插入一些记录，分页效果就更好。 

除非申明,文章均为[一号门](http://www.yihaomen.com/)原创,转载请注明本文地址,谢谢!

**mybatis实战教程(mybatis in action)之八:mybatis 动态sql语句**

mybatis 的动态sql语句是基于OGNL表达式的。可以方便的在 sql 语句中实现某些逻辑. 总体说来mybatis 动态SQL 语句主要有以下几类:  
1. if 语句 (简单的条件判断)  
2. choose (when,otherwize) ,相当于java 语言中的 switch ,与 jstl 中的choose 很类似.  
3. trim (对包含的内容加上 prefix,或者 suffix 等，前缀，后缀)  
4. where (主要是用来简化sql语句中where条件判断的，能智能的处理 and or ,不必担心多余导致语法错误)  
5. set (主要用于更新时)  
6. foreach (在实现 mybatis in 语句查询时特别有用)  
下面分别介绍这几种处理方式  
  
1. mybaits if 语句处理

程序代码 程序代码

    <select id="dynamicIfTest" parameterType="Blog" resultType="Blog">  
        select \* from t\_blog where 1 = 1  
        <if test="title != null">  
            and title = #{title}  
        </if>  
        <if test="content != null">  
            and content = #{content}  
        </if>  
        <if test="owner != null">  
            and owner = #{owner}  
        </if>  
    </select>

这条语句的意思非常简单，如果你提供了title参数，那么就要满足title=#{title}，同样如果你提供了Content和Owner的时候，它们也需要满足相应的条件，之后就是返回满足这些条件的所有Blog，这是非常有用的一个功能，以往我们使用其他类型框架或者直接使用JDBC的时候， 如果我们要达到同样的选择效果的时候，我们就需要拼SQL语句，这是极其麻烦的，比起来，上述的动态SQL就要简单多了  
  
2.2. choose (when,otherwize) ,相当于java 语言中的 switch ,与 jstl 中的choose 很类似

程序代码 程序代码

    <select id="dynamicChooseTest" parameterType="Blog" resultType="Blog">  
        select \* from t\_blog where 1 = 1   
        <choose>  
            <when test="title != null">  
                and title = #{title}  
            </when>  
            <when test="content != null">  
                and content = #{content}  
            </when>  
            <otherwise>  
                and owner = "owner1"  
            </otherwise>  
        </choose>  
    </select>

when元素表示当when中的条件满足的时候就输出其中的内容，跟JAVA中的switch效果差不多的是按照条件的顺序，当when中有条件满足的时候，就会跳出choose，即所有的when和otherwise条件中，只有一个会输出，当所有的我很条件都不满足的时候就输出otherwise中的内容。所以上述语句的意思非常简单， 当title!=null的时候就输出and titlte = #{title}，不再往下判断条件，当title为空且content!=null的时候就输出and content = #{content}，当所有条件都不满足的时候就输出otherwise中的内容。  
  
3.trim (对包含的内容加上 prefix,或者 suffix 等，前缀，后缀)

程序代码 程序代码

    <select id="dynamicTrimTest" parameterType="Blog" resultType="Blog">  
        select \* from t\_blog   
        <trim prefix="where" prefixOverrides="and |or">  
            <if test="title != null">  
                title = #{title}  
            </if>  
            <if test="content != null">  
                and content = #{content}  
            </if>  
            <if test="owner != null">  
                or owner = #{owner}  
            </if>  
        </trim>  
    </select>

trim元素的主要功能是可以在自己包含的内容前加上某些前缀，也可以在其后加上某些后缀，与之对应的属性是prefix和suffix；可以把包含内容的首部某些内容覆盖，即忽略，也可以把尾部的某些内容覆盖，对应的属性是prefixOverrides和suffixOverrides；正因为trim有这样的功能，所以我们也可以非常简单的利用trim来代替where元素的功能  
  
4. where (主要是用来简化sql语句中where条件判断的，能智能的处理 and or 条件

程序代码 程序代码

    <select id="dynamicWhereTest" parameterType="Blog" resultType="Blog">  
        select \* from t\_blog   
        <where>  
            <if test="title != null">  
                title = #{title}  
            </if>  
            <if test="content != null">  
                and content = #{content}  
            </if>  
            <if test="owner != null">  
                and owner = #{owner}  
            </if>  
        </where>  
    </select>

where元素的作用是会在写入where元素的地方输出一个where，另外一个好处是你不需要考虑where元素里面的条件输出是什么样子的，MyBatis会智能的帮你处理，如果所有的条件都不满足那么MyBatis就会查出所有的记录，如果输出后是and 开头的，MyBatis会把第一个and忽略，当然如果是or开头的，MyBatis也会把它忽略；此外，在where元素中你不需要考虑空格的问题，MyBatis会智能的帮你加上。像上述例子中，如果title=null， 而content != null，那么输出的整个语句会是select \* from t\_blog where content = #{content}，而不是select \* from t\_blog where and content = #{content}，因为MyBatis会智能的把首个and 或 or 给忽略。  
  
5.set (主要用于更新时)

程序代码 程序代码

    <update id="dynamicSetTest" parameterType="Blog">  
        update t\_blog  
        <set>  
            <if test="title != null">  
                title = #{title},  
            </if>  
            <if test="content != null">  
                content = #{content},  
            </if>  
            <if test="owner != null">  
                owner = #{owner}  
            </if>  
        </set>  
        where id = #{id}  
    </update>

set元素主要是用在更新操作的时候，它的主要功能和where元素其实是差不多的，主要是在包含的语句前输出一个set，然后如果包含的语句是以逗号结束的话将会把该逗号忽略，如果set包含的内容为空的话则会出错。有了set元素我们就可以动态的更新那些修改了的字段  
  
6. foreach (在实现 mybatis in 语句查询时特别有用)  
foreach的主要用在构建in条件中，它可以在SQL语句中进行迭代一个集合。foreach元素的属性主要有item，index，collection，open，separator，close。item表示集合中每一个元素进行迭代时的别名，index指定一个名字，用于表示在迭代过程中，每次迭代到的位置，open表示该语句以什么开始，separator表示在每次进行迭代之间以什么符号作为分隔符，close表示以什么结束，在使用foreach的时候最关键的也是最容易出错的就是collection属性，该属性是必须指定的，但是在不同情况下，该属性的值是不一样的，主要有一下3种情况：  
如果传入的是单参数且参数类型是一个List的时候，collection属性值为list  
如果传入的是单参数且参数类型是一个array数组的时候，collection的属性值为array  
如果传入的参数是多个的时候，我们就需要把它们封装成一个Map了，当然单参数也可以封装成map，实际上如果你在传入参数的时候，在MyBatis里面也是会把它封装成一个Map的，map的key就是参数名，所以这个时候collection属性值就是传入的List或array对象在自己封装的map里面的key  
  
1.1.单参数List的类型

程序代码 程序代码

    <select id="dynamicForeachTest" resultType="Blog">  
        select \* from t\_blog where id in  
        <foreach collection="list" index="index" item="item" open="(" separator="," close=")">  
            #{item}  
        </foreach>  
    </select>

上述collection的值为list，对应的Mapper是这样的

程序代码 程序代码

public List<Blog> dynamicForeachTest(List<Integer> ids);

测试代码

程序代码 程序代码

    @Test  
    public void dynamicForeachTest() {  
        SqlSession session = Util.getSqlSessionFactory().openSession();  
        BlogMapper blogMapper = session.getMapper(BlogMapper.class);  
        List<Integer> ids = new ArrayList<Integer>();  
        ids.add(1);  
        ids.add(3);  
        ids.add(6);  
        List<Blog> blogs = blogMapper.dynamicForeachTest(ids);  
        for (Blog blog : blogs)  
            System.out.println(blog);  
        session.close();  
    }

2.数组类型的参数

程序代码 程序代码

    <select id="dynamicForeach2Test" resultType="Blog">  
        select \* from t\_blog where id in  
        <foreach collection="array" index="index" item="item" open="(" separator="," close=")">  
            #{item}  
        </foreach>  
    </select>

对应mapper

程序代码 程序代码

public List<Blog> dynamicForeach2Test(int[] ids);

3. Map 类型的参数

程序代码 程序代码

    <select id="dynamicForeach3Test" resultType="Blog">  
        select \* from t\_blog where title like "%"#{title}"%" and id in  
        <foreach collection="ids" index="index" item="item" open="(" separator="," close=")">  
            #{item}  
        </foreach>  
    </select>

mapper 应该是这样的接口:

程序代码 程序代码

public List<Blog> dynamicForeach3Test(Map<String, Object> params);

通过以上方法，就能完成一般的mybatis 的 动态SQL 语句.最常用的就是  if where foreach这几个，一定要重点掌握.

除非申明,文章均为[一号门](http://www.yihaomen.com/)原创,转载请注明本文地址,谢谢!

**mybatis实战教程(mybatis in action)之九:mybatis 代码生成工具的使用**

mybatis 应用程序，需要大量的配置文件，对于一个成百上千的数据库表来说，完全手工配置，这是一个很恐怖的工作量. 所以mybatis 官方也推出了一个mybatis代码生成工具的jar包. 今天花了一点时间，按照 mybatis generator 的doc 文档参考，初步配置出了一个可以使用的版本，我把源代码也提供下载,mybatis 代码生成工具，主要有一下功能:  
1.生成pojo 与 数据库结构对应  
2.如果有主键，能匹配主键  
3.如果没有主键，可以用其他字段去匹配  
4.动态select,update,delete 方法  
5.自动生成接口(也就是以前的dao层)  
6.自动生成sql mapper，增删改查各种语句配置，包括动态where语句配置  
7.生成Example 例子供参考  
  
下面介绍下详细过程  
  
1. 创建测试工程,并配置mybatis代码生成jar包  
下载地址:<http://code.google.com/p/mybatis/downloads/list?can=3>&q=Product%3DGenerator  
mysql 驱动下载:<http://dev.mysql.com/downloads/connector/j/>  
这些jar包，我也会包含在源代码里面，可以在文章末尾处，下载源代码，参考。  
  
用 eclipse 建立一个dynamic web project。  
解压下载后的 mybatis-generator-core-1.3.2-bundle.zip 文件，其中有两个目录：一个目录是文档目录docs，主要介绍这个代码生成工具如何使用，另一个是lib目录，里面的内容主要是jar 包，这里我们需要 mybatis-generator-core-1.3.2.jar，这个 jar 包. 将它拷贝到我们刚刚创建的 web工程的 WebContent/WEB-INF/lib  目录下.在这个目录下也放入 mysql 驱动jar包.因为用 mysql  做测试的.  
  
2.在数据库中创建测试表   
在mybatis数据库中创建 用来测试的category表(如果没有mybatis这个数据库,要创建，这是基于前面这个系列文章而写的，已经有了mybatis 这个数据库)

程序代码 程序代码

Drop TABLE IF EXISTS `category`;  
Create TABLE `category` (  
  `id` int(11) NOT NULL AUTO\_INCREMENT,  
  `catname` varchar(50) NOT NULL,  
  `catdescription` varchar(200) DEFAULT NULL,  
  PRIMARY KEY (`id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

3. 配置mybatis 代码生成工具的配置文件   
在创建的web工程中，创建相应的package 比如 :  
com.yihaomen.inter 用来存放mybatis 接口对象.  
com.yihaomen.mapper用来存放sql mapper对应的映射，sql语句等.  
com.yihaomen.model 用来存放与数据库对应的model 。  
在用mybatis 代码生成工具之前，这些目录必须先创建好，作为一个好的应用程序，这些目录的创建也是有规律的。  
  
根据mybatis代码生成工具文档，需要一个配置文件，这里命名为:mbgConfiguration.xml 放在src 目录下. 配置文件内容如下:

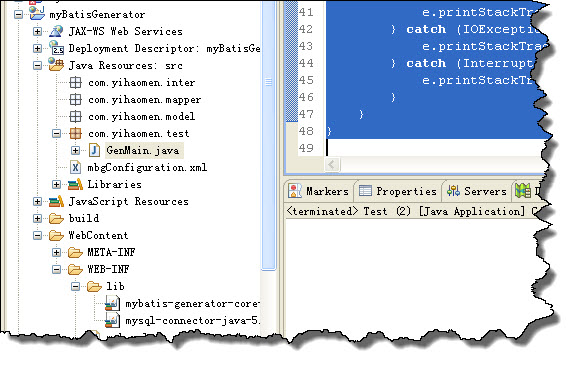
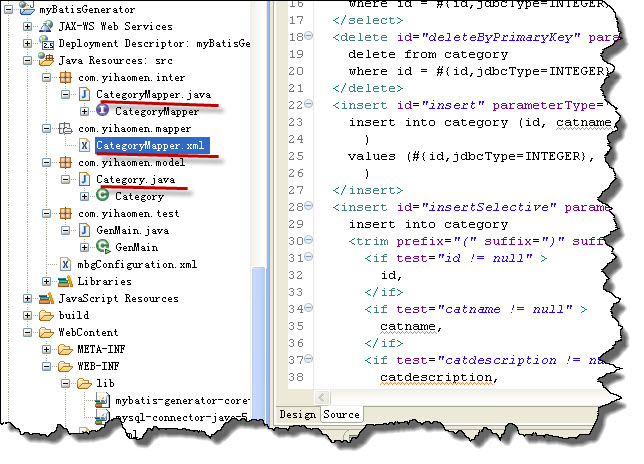
程序代码 程序代码

< ?xml version="1.0" encoding="UTF-8"?>  
< !DOCTYPE generatorConfiguration  
  PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"  
  "<http://mybatis.org/dtd/mybatis-generator-config_1_0.dtd>">  
  
< generatorConfiguration>  
    
  <!-- 配置mysql 驱动jar包路径.用了绝对路径 -->  
  <classPathEntry location="D:\Work\Java\eclipse\workspace\myBatisGenerator\WebContent\WEB-INF\lib\mysql-connector-java-5.1.22-bin.jar" />  
  
  <context id="yihaomen\_mysql\_tables" targetRuntime="MyBatis3">  
    
    <!-- 为了防止生成的代码中有很多注释，比较难看，加入下面的配置控制 -->  
    <commentGenerator>  
      <property name="suppressAllComments" value="true" />  
      <property name="suppressDate" value="true" />  
    </commentGenerator>  
    <!-- 注释控制完毕 -->  
    
    <!-- 数据库连接 -->  
    <jdbcConnection driverClass="com.mysql.jdbc.Driver"  
        connectionURL="jdbc:mysql://127.0.0.1:3306/mybatis?characterEncoding=utf8"  
        userId="root"  
        password="password">  
    </jdbcConnection>  
  
    <javaTypeResolver >  
      <property name="forceBigDecimals" value="false" />  
    </javaTypeResolver>  
      
    <!-- 数据表对应的model 层  -->  
    <javaModelGenerator targetPackage="com.yihaomen.model" targetProject="src">  
      <property name="enableSubPackages" value="true" />  
      <property name="trimStrings" value="true" />  
    </javaModelGenerator>  
      
    <!-- sql mapper 隐射配置文件 -->  
    <sqlMapGenerator targetPackage="com.yihaomen.mapper"  targetProject="src">  
      <property name="enableSubPackages" value="true" />  
    </sqlMapGenerator>  
      
    <!-- 在ibatis2 中是dao层，但在mybatis3中，其实就是mapper接口 -->  
    <javaClientGenerator type="XMLMAPPER" targetPackage="com.yihaomen.inter"  targetProject="src">  
      <property name="enableSubPackages" value="true" />  
    </javaClientGenerator>  
      
    <!-- 要对那些数据表进行生成操作，必须要有一个. -->  
    <table schema="mybatis" tableName="category" domainObjectName="Category"   
        enableCountByExample="false" enableUpdateByExample="false"  
        enableDeleteByExample="false" enableSelectByExample="false"  
        selectByExampleQueryId="false">       
    </table>  
  
  </context>  
< /generatorConfiguration>

用一个main 方法来测试能否用mybatis 成生成刚刚创建的`category`表对应的model,sql mapper等内容.  
创建一个com.yihaomen.test 的package ,并在此package 下面建立一个测试的类GenMain：

程序代码 程序代码

package com.yihaomen.test;  
  
import java.io.File;  
import java.io.IOException;  
import java.sql.SQLException;  
import java.util.ArrayList;  
import java.util.List;  
  
import org.mybatis.generator.api.MyBatisGenerator;  
import org.mybatis.generator.config.Configuration;  
import org.mybatis.generator.config.xml.ConfigurationParser;  
import org.mybatis.generator.exception.InvalidConfigurationException;  
import org.mybatis.generator.exception.XMLParserException;  
import org.mybatis.generator.internal.DefaultShellCallback;  
  
public class GenMain {  
    public static void main(String[] args) {  
        List<String> warnings = new ArrayList<String>();  
        boolean overwrite = true;  
        String genCfg = "/mbgConfiguration.xml";  
        File configFile = new File(GenMain.class.getResource(genCfg).getFile());  
        ConfigurationParser cp = new ConfigurationParser(warnings);  
        Configuration config = null;  
        try {  
            config = cp.parseConfiguration(configFile);  
        } catch (IOException e) {  
            e.printStackTrace();  
        } catch (XMLParserException e) {  
            e.printStackTrace();  
        }  
        DefaultShellCallback callback = new DefaultShellCallback(overwrite);  
        MyBatisGenerator myBatisGenerator = null;  
        try {  
            myBatisGenerator = new MyBatisGenerator(config, callback, warnings);  
        } catch (InvalidConfigurationException e) {  
            e.printStackTrace();  
        }  
        try {  
            myBatisGenerator.generate(null);  
        } catch (SQLException e) {  
            e.printStackTrace();  
        } catch (IOException e) {  
            e.printStackTrace();  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
        }  
    }  
}

到此为止,eclipse项目工程图应该如下:  
  
  
4.运行测试的main 方法,生成mybatis 相关代码  
运行GenMain类里的main方法，并刷新 工程,你会发现 各自package 目录下已经响应生成了对应的文件，完全符合mybatis 规则，效果图如下:  
  
  
5.注意事项   
如果你想生成example 之类的东西，需要在<table></table>里面去掉

程序代码 程序代码

enableCountByExample="false" enableUpdateByExample="false"  
enableDeleteByExample="false" enableSelectByExample="false"  
selectByExampleQueryId="false"

这部分配置，这是生成Example而用的，一般来说对项目没有用.  
  
另外生成的sql mapper 等，只是对单表的增删改查，如果你有多表join操作，你就可以手动配置，如果调用存储过程，你也需要手工配置. 这时工作量已经少很多了。  
  
如果你想用命令行方式处理，也是可以的.

程序代码 程序代码

比如:  
java -jar mybatis-generator-core-1.3.2.jar -mbgConfiguration.xm -overwrite

这时，要用绝对路径才行. 另外mbgConfiguration.xml  配置文件中targetProject 的配置也必须是绝对路径了。  
  
代码下载：下载文件[mybatis 代码生成工具](http://www.yihaomen.com/attachments/month_1212/b20121216131924.rar)

除非申明,文章均为[一号门](http://www.yihaomen.com/)原创,转载请注明本文地址,谢谢!

**mybatis SqlSessionDaoSupport的使用(附代码下载)**

前面的系列mybatis 文章，已经基本讲到了mybatis的操作，但都是基于mapper隐射操作的，在mybatis 3中这个mapper 接口貌似充当了以前在ibatis 2中的 DAO 层的作用。但事实上，如果有这个mapper接口不能完成的工作，或者需要更复杂的扩展的时候，你就需要自己的DAO 层. 事实上 mybatis 3 也是支持DAO 层设计的，类似于ibatis 2 .下面介绍下.  
  
在此之前，请下载 上一篇文章提供的代码:<http://www.yihaomen.com/article/java/326.htm>  
  
首先创建一个com.yihaomen.dao的package.然后在里面分别创建接口UserDAO,以及实现该接口的UserDAOImpl

程序代码 程序代码

package com.yihaomen.dao;  
import java.util.List;  
import com.yihaomen.model.Article;  
public interface UserDAO {  
    public List<Article> getUserArticles(int userid);  
}

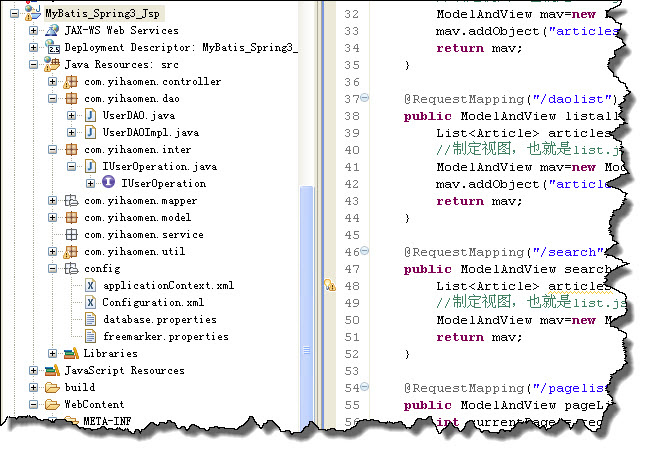
程序代码 程序代码

package com.yihaomen.dao;  
import java.util.List;  
import org.mybatis.spring.support.SqlSessionDaoSupport;  
import org.springframework.stereotype.Repository;  
import com.yihaomen.model.Article;  
  
@Repository  
public class UserDAOImpl extends SqlSessionDaoSupport implements UserDAO {  
    @Override  
    public List<Article> getUserArticles(int userid) {          
        return this.getSqlSession().selectList("com.yihaomen.inter.IUserOperation.getUserArticles",userid);  
    }  
}

执行的SQL 语句采用了命名空间+sql 语句id的方式，后面是参数.  
  
注意继承了 "SqlSessionDaoSupport" ，利用方法 getSqlSession() 可以得到 SqlSessionTemplate ,从而可以执行各种sql语句，类似于hibernatetemplate一样，至少思路一样.  
  
如果与spring 3 mvc 集成要用  autowire的话，在daoimpl 类上 加上注解 “@Repository” ，另外还需要在spring 配置文件中加入<context:component-scan base-package="com.yihaomen.dao" /> 这样在需要调用的地方，就可以使用autowire自动注入了。  
  
当然，你也可以按一般程序的思路，创建一个service 的package, 用service 去调用 dao层，我这里就没有做了，因为比较简单，用类似的方法，也机注意自动注入时，也要配置 <context:component-scan base-package="com.yihaomen.service" /> 等这样的。  
  
在controller层中测试,直接调用dao层方法  
在controller中加入方法:

程序代码 程序代码

    @Autowired  
    UserDAO userDAO;  
  
        .......  
  
    @RequestMapping("/daolist")  
    public ModelAndView listalldao(HttpServletRequest request,HttpServletResponse response){  
        List<Article> articles=userDAO.getUserArticles(1);  
        //制定视图，也就是list.jsp  
        ModelAndView mav=new ModelAndView("list");  
        mav.addObject("articles",articles);  
        return mav;  
    }

这样可以得到同样的结果，而且满足了一般程序的设计方法.代码结构如下:  
  
  
完成后，程序代码下载.