mybatis实战

http://wiki.jikexueyuan.com/project/mybatis-in-action/environment-to-build.html

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。

如何搭建

首先是要建立一个configuration.xml来管理mybatis
在这里我们使用的是user的表的例子 在极客网站上有具体的表结构

```
<?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.mybatis.eneity.User"/>
    </typeAliases>


    <environments default="development">
        <environment id="development">
            <transactionManager type="JDBC">
                <property name="" value=""/>
            </transactionManager>
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.jdbc.Driver"/>
                <property name="url" value="jdbc:mysql://localhost:3306/spitter" />
                <property name="username" value="root"/>
                <property name="password" value="liuziye"/>
            </dataSource>
        </environment>
    </environments>



    <mappers>
        <mapper resource="mybatis/User.xml"/>
    </mappers>

</configuration>
```

需要注意 mybatis配置文件中的各个元素的顺序是不能乱的
然后下面是建立的是数据库对应的javabean 以及连接数据库与javabean xml文件

```
package com.mybatis.eneity;

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 连接数据库与javaBean的具体配置xml（同时也是与相应接口方法对应的xml） User.xml:

```
<?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.mybatis.inter.IUserOperation">
    <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。

下面这样就能初步运行了

```
import java.io.Reader;

import com.mybatis.eneity.User;
import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;


public class Test {
private static SqlSessionFactory sqlSessionFactory;
private static Reader reader;

static{
try{
reader    = Resources.getResourceAsReader("mybatis/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.mybatis.inter.IUserOperation.selectUserByID", 1);
System.out.println(user.getUserAddress());
System.out.println(user.getUserName());
        } finally {
session.close();
        }
    }
}
```

这种方式是用 **SqlSession** 实例来直接执行已映射的**SQL**语句

```
User user = (User) session.selectOne("com.mybatis.inter.IUserOperation.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:

建立具体的接口类 内容如下：（同时注意 在User.xml的mapper的NameSpace需要改成对应我们需要下面实现的接口的路径）

```
public interface IUserOperation {
public User selectUserByID(int id);
}
```

这样子就能更简单的也更合理的达到需求了

```
public  void TestselectUserById() throws IOException {
/*获取Session*/
SqlSession sqlSession = mybatisdb.getSqlSession();
try {
/*获取对应的映射xml*/
IUserOperation userOperation=sqlSession.getMapper(IUserOperation.class);
/*通过xml和接口同时定义的selectUserById来进行查询*/
User user = userOperation.selectUserByID(1);
/*输出查询的值*/
System.out.println(user.getUserAddress());
System.out.println(user.getUserName());

    } finally {
sqlSession.close();
    }
}
```

所以说有俩种获取映射xml的途径 一种是通过session的selectone直接指定具体的mapper和参数匹配 然后使用
另外一种是通过直接匹配mapper 通过与mapper对应的接口的方法来进行使用
第一种是方法查找
第二种是mapper查找

增删改查

这样写好之后 就很简单了 只需要在关键的接口与映射xml中进行修改就可以进行达到目的

查

比如 加入一个新的方法

```
public List<User> selectUsers(String userName);
```

这个方法是用来查询Users的集合的 返回的是一个List集合
我们需要在映射xml中配置返回值类型 然后再定义方法

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

```
<select id="selectUsers" parameterType="string" resultMap="resultListUser">
SELECT * FROM  USER  WHERE  username LIKE  #{username}
</select>
```

下面是测试方法

```
public static void main(String[] args) {
MybatisTest mybatisTest = new MybatisTest();
mybatisTest.TestselectUsers("%");
  }


public  void TestselectUsers(String username){
/*获取Session*/
SqlSession sqlSession = mybatisdb.getSqlSession();
try{
IUserOperation userOperation= sqlSession.getMapper(IUserOperation.class);
/*查询Users列表*/
List<User> users=userOperation.selectUsers(username);
for (User user:users){
System.out.println(user.getId()+":"+user.getUserName()+":"+user.getUserAddress());
          }

      }finally {
sqlSession.close();
      }

  }
```

既然知道了上面这张 下面的几种就很容易看懂了

增

在映射xml中需要定义如下

```
<!--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User参数的name属性，MyBatis将使用反射读取User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。
    useGeneratedKeys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User的id属性-->
<insert id="addUser" parameterType="User"
useGeneratedKeys="true" keyProperty="id">
INSERT  INTO USER(userName,userAge,userAddress)
     VALUES (#{userName},#{userAge},#{userAddress})
</insert>
```

在接口中同样需要定义一下

```
public  void addUser(User user);
```

然后就是具体执行了 需要注意 必须要提交事务 不然是不会提交的

```
/**
 * 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 TestaddUser( ) {
User user = new User();
user.setUserAddress("广场");
user.setUserName("竹杠");
user.setUserAge("80");
SqlSession sqlSession = mybatisdb.getSqlSession();
try {
IUserOperation userOperation = sqlSession.getMapper(IUserOperation.class);
userOperation.addUser(user);
sqlSession.commit();
System.out.println("当前增加的用户 id为"+user.getId());
    }finally {
sqlSession.close();
    }
}
```

改

改的话也是非常相似的

映射xml中

```
<update id="updateUser" parameterType="User">
UPDATE  USER  SET  userName=#{userName},userAge=#{userAge},userAddress=#{userAddress}
    where id=#{id}
</update>
```

对应的接口中

```
public void updateUser(User user);
```

在具体测试的时候

```
/*测试更新*/
public  void TestUpdateUser() {
SqlSession sqlSession = mybatisdb.getSqlSession();

try {
IUserOperation userOperation = sqlSession.getMapper(IUserOperation.class);
User user=userOperation.selectUserByID(3);
user.setUserAddress("china jiangsu");
userOperation.updateUser(user);
sqlSession.commit();
       }finally {
sqlSession.close();
       }
   }
```

删除

删除同样非常相似

对应的映射xml

```
<delete id="deleteUser" parameterType="int">
DELETE  FROM  USER  WHERE  id=#{id}
</delete>
```

对应的接口

```
public  void deleteUser(int id);
```

具体的执行方法

```
/*测试删除*/
public  void TestDeleteUser() {
SqlSession sqlSession = mybatisdb.getSqlSession();
try {
IUserOperation userOperation=  sqlSession.getMapper(IUserOperation.class);
userOperation.deleteUser(3);
sqlSession.commit();
       }finally {
sqlSession.close();
       }
   }
```

需要一定要注意的是 在对数据库进行修改的时候 必须要进行事务的提交 不然是不会在数据库显示出结果的

进行关联数据的修改

，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。

```
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。

```
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;
    }

}
```

需要在对应的映射文件中进行定义相关查询语句的实现 然后就是实现了

```
<!--为返回多对一自定义返回值-->
<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>


<!--查询具体的多对一的关联数据查询-->
<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>
```

还有另外一种处理方式

```
</resultMap>
<!--为返回多对一自定义返回值-->
```

```
<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" resultMap="resultListUser" >
<!--  <id property="id" column="id"/>
        <result property="userName" column="userName"/>
        <result property="userAddress" column="userAddress"/>-->
</association>

</resultMap>
```

这种就是事实上的外键 通过查询语句实现的功能
具体测试如下

```
public  void TestgetArticles(String title){

SqlSession sqlSession = mybatisdb.getSqlSession();
try {

IUserOperation userOperation= sqlSession.getMapper(IUserOperation.class);
List<Article> articles= userOperation.getArticles(title);

for (Article article: articles) {
System.out.println(article);
        }
    }finally {
sqlSession.close();
    }
}
```

关于Spring与mybatis的集成

首先需要导包
这个导包很麻烦
暂且粘贴在这了

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-test</artifactId>
        <version>4.1.1.RELEASE</version>
        <scope>test</scope>
    </dependency>


    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>4.2.2.Final</version>
    </dependency>

    <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.16</version>
    </dependency>

    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-api</artifactId>
        <version>1.6.1</version>
    </dependency>

    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-nop</artifactId>
        <version>1.6.4</version>
    </dependency>

    <dependency>
        <groupId>javassist</groupId>
```

```
        <artifactId>javassist</artifactId>
        <version>3.11.0.GA</version>
    </dependency>

<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.38</version>
    </dependency>


<!-- https://mvnrepository.com/artifact/org.mybatis/mybatis -->
<dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis</artifactId>
        <version>3.3.1</version>
    </dependency>

<!--Spring-->
<dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>3.1.1.RELEASE</version>
    </dependency>



    <dependency>
        <groupId>commons-collections</groupId>
        <artifactId>commons-collections</artifactId>
        <version>3.2.1</version>
    </dependency>

    <dependency>
        <groupId>commons-dbcp</groupId>
        <artifactId>commons-dbcp</artifactId>
        <version>1.2.2</version>
    </dependency>

    <dependency>
        <groupId>commons-logging</groupId>
        <artifactId>commons-logging</artifactId>
        <version>1.1</version>
    </dependency>


    <dependency>
        <groupId>commons-pool</groupId>
        <artifactId>commons-pool</artifactId>
        <version>1.6</version>
    </dependency>

    <dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis-spring</artifactId>
        <version>1.2.2</version>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-aop</artifactId>
        <version>3.1.1.RELEASE</version>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-asm</artifactId>
        <version>3.1.1.RELEASE</version>
    </dependency>


    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-aspects</artifactId>
```

```
    <version>3.1.1.RELEASE</version>
</dependency>


<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-beans</artifactId>
    <version>3.1.1.RELEASE</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context-support</artifactId>
    <version>3.1.1.RELEASE</version>
</dependency>



<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>3.1.1.RELEASE</version>
</dependency>


<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-expression</artifactId>
    <version>3.1.1.RELEASE</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-instrument-tomcat</artifactId>
    <version>3.1.1.RELEASE</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-instrument</artifactId>
    <version>3.1.1.RELEASE</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>3.1.1.RELEASE</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jms</artifactId>
    <version>3.1.1.RELEASE</version>
</dependency>


<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
    <version>3.1.1.RELEASE</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-oxm</artifactId>
    <version>3.1.1.RELEASE</version>
</dependency>


<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
    <version>3.1.1.RELEASE</version>
</dependency>
```

```
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-tx</artifactId>
        <version>3.1.1.RELEASE</version>
    </dependency>


    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc-portlet</artifactId>
        <version>3.1.1.RELEASE</version>
    </dependency>


    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>3.1.1.RELEASE</version>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-web</artifactId>
        <version>3.1.1.RELEASE</version>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-struts</artifactId>
        <version>3.1.1.RELEASE</version>
    </dependency>

</dependencies>
```

然后就写一个关于mybatis的spring xml配置

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/bea
ns/spring-beans.xsd">


       <bean id="datasource" class="org.apache.commons.dbcp.BasicDataSource">
              <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
              <property name="url" value="jdbc:mysql://localhost:3306/spitter?characterEncoding=utf8"/>

              <property name="username" value="root"/>
              <property name="password" value="liuziye"/>

       </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"/>

<!--Configuration属性mybatis的核心配置文件-->
<property name="configLocation" value="Spring/mybatis/Configuration.xml"/>
        </bean>


       <bean id="userMapper" class="org.mybatis.spring.mapper.MapperFactoryBean">
<!--sqlSeesionFactory属性指定要用到的Sqlsession实例-->
<property name="sqlSessionFactory" ref="sqlSessionFactory"/>

<!-- mapperInterface属性指定映射器接口-->
<property name="mapperInterface" value="com.mybatis.inter.IUserOperation"/>
       </bean>



</beans>
```

然后就进行具体测试

```
public class MybatisSpringTest {

private static ApplicationContext ctx;

static {
ctx=new ClassPathXmlApplicationContext("Spring/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> articleList=mapper.getUserArticles(1);

for (Article article:articleList){
System.out.println(article.getContent()+"--"+article.getTitle());
        }
    }
}
```

关于Spring MVC与mybatis的集成 其实相差不大

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

首先是web.xml
要注意加载几个监听器 还有几个xml也需要加载到配置中

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
  <display-name>Archetype Created Web Application</display-name>

       <display-name>Archetype Created Web Application</display-name>
<!-- Spring的log4j监听器 -->
<listener>
          <listener-class>org.springframework.web.util.Log4jConfigListener</listener-class>
       </listener>

     <context-param>
         <param-name>contextConfigLocation</param-name>
         <param-value>classpath*:Spring/applicationContext.xml</param-value>
         <param-value>classpath*:Spring/mybatis/Configuration.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>
        <init-param>
```

```
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath*:Spring/MVC/mvc-dispatcher-servlet.xml</param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>mvc-dispatcher</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

</web-app>
```

然后就是相关配置的mvc-dispatcher-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<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-instanc
e"
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.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>
```

相关的applicationcontxt.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/bea
ns/spring-beans.xsd http://www.springframework.org/schema/context http://www.springframework.org/schema/c
ontext/spring-context.xsd">

    <context:component-scan base-package="com.mybatis.Spring"/>
    <context:annotation-config/>
    <context:component-scan base-package="com.controller"/>



    <bean id="datasource" class="org.apache.commons.dbcp.BasicDataSource">
              <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
              <property name="url" value="jdbc:mysql://localhost:3306/spitter?characterEncoding=utf8"/>

              <property name="username" value="root"/>
              <property name="password" value="liuziye"/>

       </bean>


    <bean id="transationManager" 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"/>

<!--Configuration属性mybatis的核心配置文件-->
<property name="configLocation" value="Spring/mybatis/Configuration.xml"/>

            <property name="mapperLocations" value="classpath*:mybatis/User.xml"/>
        </bean>


    <bean   class="org.mybatis.spring.mapper.MapperScannerConfigurer">
      <property name="basePackage" value="com.mybatis.inter"></property>
    </bean>

<!--<bean id="MybatisSpringTest" class="com.mybatis.Spring.MybatisSpringTest"/>-->

     <!--  <bean id="userMapper" class="org.mybatis.spring.mapper.MapperFactoryBean">
         &lt;!&ndash;sqlSeesionFactory属性指定要用到的Sqlsession实例&ndash;&gt;
              <property name="sqlSessionFactory" ref="sqlSessionFactory"/>

              &lt;!&ndash; mapperInterface属性指定映射器接口&ndash;&gt;
              <property name="mapperInterface" value="com.mybatis.inter.IUserOperation"/>
       </bean>-->

</beans>
```

编写controller层

```
@Controller
@RequestMapping("/article")
@Repository
public class UserController {




@RequestMapping("/list")
public ModelAndView listall(HttpServletRequest request,HttpServletResponse response){
ApplicationContext context=new ClassPathXmlApplicationContext("Spring/applicationContext.xml");
IUserOperation userMapper= (IUserOperation) context.getBean("IUserOperation");
List<Article> articles=userMapper.getUserArticles(1);
ModelAndView mav=new ModelAndView("list");
mav.addObject("articles",articles);
return mav;
    }

}
```

最后用JSP页面接一下

```
<%@page isELIgnored="false"%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@taglib uri="http://java.sun.com/jsp/jstl/core"  prefix="c" %>
<html>
<head>
    <title></title>

</head>
<body>
<c:forEach items="${articles}" var="item">
${item.id }--${item.title }--${item.content }<br />
</c:forEach>
Test
</body>
</html>
```