# MyBatis框架

**内参保密不得外传**

**讲师:王涛**

**2018**

# 第一天：mybatis基础



## 传统框架的缺点

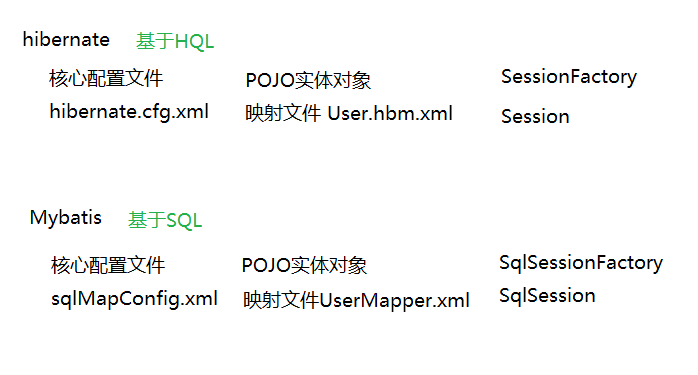
### Hibernate的缺点

* 封装过重，概念繁多，如自己管理对象，对象的游离态等
* 创造了HQL面向对象从查询语言，但学习成本高，晦涩难懂，不易维护，底层最终还是翻译成SQL语句执行
* 过渡封装后造成性能急剧降低
* 适合小型项目

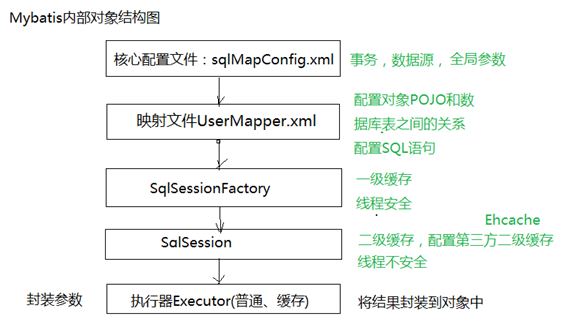
## mybatis介绍

mybatis它是轻量级持久层框架，由ibatis演化而来。它完成将数据库的结果集封装到对象中POJO。业务层控制层和使用Hibernate框架一样。Hibernate基于hql是完全面向对象。全自动ORM。Mybatis基于sql是半面向对象。半自动的ORM。

### 和hibernate比较图



### 内部组件结构图



## mybatis第一个例子

### 开发步骤

#### 创建JavaProject工程导入jar包

asm-3.3.1.jar

cglib-2.2.2.jar

commons-logging-1.1.1.jar

javassist-3.17.1-GA.jar

log4j-1.2.17.jar

slf4j-api-1.7.5.jar

slf4j-log4j12-1.7.5.jar

mybatis-3.2.2.jar 核心包

mysql-connector-java-5.1.28-bin.jar 数据库驱动

#### 配置核心配置文件sqlMapConfig.xml

注意：配置dtd时，个别的电脑配置完后不起作用。

<?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>

<!-- 1. 组织数据源 -->

<environments default="test">

<environment id="test">

<!-- 事务：JDBC/MANAGE -->

<transactionManager type="JDBC"/>

<!-- 数据库配置：JNDI / POOLED池化 / UNPOOLED非池化 -->

<dataSource type="POOLED">

<property name="driver" value="com.mysql.jdbc.Driver" />

<property name="url" value="jdbc:mysql://localhost:3306/mybatisdb?characterEncoding=utf-8" />

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

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

</dataSource>

</environment>

</environments>

<!-- 2. xml配置映射文件 -->

<mappers>

<mapper resource="mybatis/pojo/UserMapper.xml"/>

</mappers>

</configuration>

#### 创建数据库表和测试数据

创建mybatis数据库，创建user表

CREATE TABLE `user\_c` (

`id` varchar(40) NOT NULL,

`name` varchar(30) default NULL,

`age` int(11) default NULL,

`address` varchar(200) default NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

/\*Data for the table `user\_c` \*/

insert into `user\_c`(`id`,`name`,`age`,`address`) values ('1','夏言',73,'桂州村'),('2','严嵩',87,'分宜县城介桥村'),('3','徐阶',80,'明松江府华亭县'),('4','高拱',66,'河南省新郑市高老庄村'),('5','张居正',58,'江陵');

#### 配置一个映射文件UserMapper.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">

<!-- 找到某个业务的方法时，就要靠这个命名空间参数来区分 -->

<mapper namespace="cn.mybatis.po">

<!-- 查询所有 -->

<!-- namespance+id要全局唯一 -->

<!-- 原则：在mybatis中配置集合对象时，只需声明集合对象的类型 -->

<select id="findAll" resultType="cn.mybatis.po.User">

SELECT \* FROM user\_c

</select>

</mapper>

#### 测试类

package test;

import java.io.IOException;

import java.io.InputStream;

import java.util.List;

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 org.junit.Test;

import cn.mybatis.po.User;

public class TestMybatis {

@Test

public void SqlSessionFactory() throws IOException{

//创建factory

String resource = "sqlMapConfig.xml";

InputStream is = Resources.getResourceAsStream(resource);

SqlSessionFactory factory = new SqlSessionFactoryBuilder().build(is);

SqlSession session = factory.openSession();

//查询方法 namespace.id=cn.mybatis.po.findAll

List<User> userList = session.selectList("cn.mybatis.po.findAll");

for(User u : userList){

System.out.println(u);

}

}

}

## 基础的CRUD操作

### 新增

1）文件中增加insert SQL

<!-- 新增 -->

<insert id="insert">

INSERT INTO user\_c (id,NAME,age,address)

VALUES('6','tony',80,'beijing')

</insert>

5） 利用sqlsession调用

@Test //带参数，新增

public void testInsert(){

SqlSession session = factory.openSession();

User user = new User();

user.setId("7");

user.setName("tom");

user.setAge(8);

user.setAddress("xi'an");

session.insert("mybatis.pojo.UserMapper.insert", user);

session.commit(); //默认不自动提交

}

### 修改

<!-- 修改 -->

<update id="update" parameterType="mybatis.pojo.User">

UPDATE user\_c SET age=#{age} WHERE id=#{id}

</update>

@Test

public void testUpdate(){

SqlSession session = factory.openSession();

User user = new User();

user.setAge(3);

user.setId("7");

session.update("mybatis.pojo.UserMapper.update", user);

session.commit(); //默认不自动提交

}

### 删除

<!-- 删除一条 -->

<delete id="deleteOne" parameterType="string">

delete from user\_c where id=#{pid}

</delete>

<!-- 批量删除(数组) in ('x','y','z') -->

<delete id="delete" parameterType="string">

delete from user\_c where id in

<foreach collection="array" item="id" open="(" close=")" separator=",">

#{id}

</foreach>

@Test

public void testDeleteOne(){

SqlSession session = factory.openSession();

session.delete("mybatis.pojo.UserMapper.deleteOne", "7");

session.commit();

}

@Test

public void testDelete(){

SqlSession session = factory.openSession();

String[] ids = new String[]{"7"};

session.delete("mybatis.pojo.UserMapper.delete", ids);

session.commit();

}

### 查询一个

selectOne返回的结果集不能有多条记录。

<!-- 查询一个 -->

<select id="get" parameterType="string" resultType="mybatis.pojo.User">

select \* from user\_c where id=#{pid}

</select>

@Test //查询一个

public void testGet(){

SqlSession session = factory.openSession();

User user = session.selectOne("mybatis.pojo.UserMapper.get", "5");

System.out.println(user);

}

### 带条件的查询，使用map来封装参数

<!-- 带条件查询 -->

<select id="find" parameterType="map" resultType="mybatis.pojo.User">

select \* from user\_c

<![CDATA[

where address like #{address} and age>#{beginAge} and age<#{endAge}

]]>

</select>

@Test //带条件查询

public void testFind(){

SqlSession session = factory.openSession();

//通过map结构封装查询参数

Map<String,Object> map = new HashMap<String,Object>();

map.put("address", "%村%");

map.put("beginAge", 70);

map.put("endAge", 80);

List<User> userList = session.selectList("mybatis.pojo.UserMapper.find", map);

for(User user : userList){

System.out.println(user);

}

}

注意，

当sql中含有特殊字符时，通过<![CDATA[]]>包起来，xml解析时就不解析

Like时需要特殊字符，可以直接放在调用时赋值。

日期，整数，长整形，浮点数，可能查询条件是一个范围。只要map的key随意定义，开始值，结束值，在映射文件中参数必须和map的key一致。不然值默认为null。

### 排序

<!-- 动态排序，参数age -->

<select id="findOrderBy" parameterType="map" resultType="mybatis.pojo.User">

SELECT \* FROM user\_c ORDER BY ${orderValue}

</select>

@Test //动态排序条件

public void testOrderBy(){

SqlSession session = factory.openSession();

Map<String,Object> map = new HashMap<String,Object>();

map.put("orderValue", "name,age");

List<User> userList = session.selectList("mybatis.pojo.UserMapper.findOrderBy", map);

for(User user : userList){

System.out.println(user);

}

}

## 基础使用方法

### SQL标签

<!-- 简化，定义列 -->

<sql id="cols">

id,user\_name,age,address

</sql>

一处声明多处调用

### alias别名

在sqlMapConfig.xml配置，在映射文件中直接写对象名称即可

<typeAliases>

<typeAlias type="cn.mybatis.domain.User" alias="User"/>

</typeAliases>

### 自增主键

字段类型必须为int/long，数据还需支持mysql,sqlserver，oracle不支持

<?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="cn.mybatis.po.PersonMapper">

<!-- mybatis自增主键

useGeneratedKeys 启用mybatis自增

keyProperty 主键

-->

<insert id="insert" parameterType="cn.mybatis.po.Person" keyProperty="id" keyColumn="id" useGeneratedKeys="true">

insert into person\_c

(name)

values

(#{name})

</insert>

</mapper>

### XML配置文件中的特殊字符

<![CDATA[

where age < #{age}

]]>

### 参数和返回值

#### 参数值类型

只有paramterType，paramterMap已经过时，后期版本可能删掉。

#### 返回值类型

ResultType 简单类型如：string/int/long

ResultMap 用于对复杂对象结构时，对应的ResultMap结构名称

### 参数占位符

mybatis本质就是拼接SQL语句。

#### #{name}

使用的jdbc的ParpareStatment。#{} 很赞，防止SQL注入；如果参数是一个字符串类型。chen 拼接SQL语句时会根据类型，自动加相关符号。例如字符串类型'chen'。

#### ${orderby}

${} 原样输出，很危险，有SQL注入风险。

## 动态SQL语句

是SQL语句活跃起来，变化多样。

<if> 进行if判断，根据属性值，来判断是否拼接if标签中的内容。

一般按null判断<if test=”属性值!=null”>xxx</if>

<where> 去掉多个条件拼接时最前面的and或者or

<set> 去掉评价多个值时，可能最后一个有多余的逗号，它会自动去掉。没有多余的就不管。

<foreach> 就用在in子查询中。拼接多个id。

### 集合参数

#### array

<!-- 批量删除array id in ('x','y')-->

<delete id="deleteArray" parameterType="string">

delete from user\_c

where id in

<foreach collection="array" open="(" close=")" item="id" separator=",">

#{id}

</foreach>

</delete>

#### list

<delete id="deleteList" parameterType="string">

delete from user\_c

where id in

<foreach collection="list" item="id" open="(" close=")" separator=",">

#{id}

</foreach>

</delete>

#### map

map最为灵活，ids为map的key

<delete id="deleteMap" parameterType="map">

delete from user\_c

where id in

<foreach collection="ids" item="id" open="(" close=")" separator=",">

#{id}

</foreach>

</delete>

## 接口开发方式

### UserMapper接口

package cn.mybatis.domain;

import java.util.List;

import java.util.Map;

public interface UserMapper {

public List<User> find(Map<String,Object> map);

}

### 测试类

package test;

import java.io.IOException;

import java.io.InputStream;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

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 org.junit.Before;

import org.junit.Test;

import cn.mybatis.domain.User;

import cn.mybatis.domain.UserMapper;

public class TestMapperInterface {

private SqlSessionFactory factory;

@Before //在调用其他方法测试前被执行

public void init() throws IOException{

//创建SqlSessionFactory

String resource = "sqlMapConfig.xml";

InputStream is = Resources.getResourceAsStream(resource); //加载核心配置文件

factory = new SqlSessionFactoryBuilder().build(is);

}

@Test //测试接口调用

public void testMapper(){

SqlSession session = factory.openSession();

//获取mapper接口

UserMapper mapper = session.getMapper(UserMapper.class);

Map map = new HashMap();

map.put("name", "张%");

List<User> userList = mapper.find(map);

System.out.println(userList.size());

}

}

## 自动代码生成工具

### generator的插件

大量减少开发的工作。

1）将插件的目录拷贝到eclispe插件目录中dropins

2）要进行配置sqlMapGeneractor.xml

配置产生映射文件，配置相关的表

可以生成映射文件，PO文件，调用代码，测试代码

指定生成的目录，为哪个项目生成代码。

### generatorConfig.properties

suppressAllComments=false

driverClass=com.mysql.jdbc.Driver

url=jdbc:mysql://localhost:3306/mybatisdb?characterEncoding=utf-8

username=root

password=root

modeltargetPackage=cn.mybatis.model

targetProject=t\_MybatisStart

sqltargetPackage=cn.mybatis.mapper

clienttargetPackage=client

### sqlMapGeneractor.xml

<?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>

<!-- 配置属性文件，这样有变更只需改配置文件 -->

<properties resource="generatorConfig.properties"/>

<!-- 制定mysql的驱动包的路径 千万别放中文路径下 -->

<classPathEntry location="D:\javaenv\all\_jar\database\mysql-connector-java-5.1.26.jar" />

<!-- 配置数据源和生成的代码所存放的位置 -->

<context id="tarena">

<!-- 是否生成注释 true不生成 false生成 -->

<commentGenerator>

<property name="suppressAllComments" value="${suppressAllComments}"/>

</commentGenerator>

<jdbcConnection driverClass="${driverClass}" connectionURL="${url}" userId="${username}" password="${password}" />

<!-- 所生成的实体类的位置默认资源包src -->

<javaModelGenerator targetPackage="${modeltargetPackage}" targetProject="${targetProject}" />

<!-- 所生成的sqlMap的影射文件的位置,默认资源包src -->

<sqlMapGenerator targetPackage="${sqltargetPackage}" targetProject="${targetProject}" />

<javaClientGenerator targetPackage="${clienttargetPackage}" targetProject="${targetProject}" type="XMLMAPPER" />

<!-- 为哪些表生成代码 tableName:表名 schema:不用填写 -->

<table schema="" tableName="order\_c" />

</context>

</generatorConfiguration>

# 第二天：mybatis加强

## 关联关系

一对多

一对一

多对一

多对多

Mybatis关联只有两种情况，对几？

规则：对多，后面就是配置一个集合；对一，就是配置一个对象。

规则：

结果集中有同名字段，导致mybatis无法映射，关系混乱。

Mybatis中结果集不能有同名字段！！！！

### 一对多

1. POJO对象单项关联，部门对象配置list集合用户对象

public class Dept implements Serializable{

//一个部门下有多个用户,对多

private List<User> users;

1. 编写一个resultMap，嵌套关联关系

<!-- 配置多个resultMap -->

<resultMap type="cn.mybatis.po.Dept" id="deptRM">

<id property="id" column="DEPT\_ID"/>

<result property="name" column="DEPT\_NAME"/>

<!-- 关联关系,对多。property对应POJO对象中配置的属性，ofType集合中元素类型 -->

<collection property="users" ofType="cn.mybatis.po.User">

<id property="id" column="ID" javaType="string"/>

<result property="name" column="USER\_NAME" javaType="string"/>

<result property="age" column="AGE" javaType="integer"/>

<result property="address" column="ADDRESS" javaType="string"/>

</collection>

</resultMap>

关联关系

立即加载，通过sql语句左链接查询获得结果集。这个结果集中包括两个对象所需要的所有的属性。

扩展。购销合同，包括多个货物，一个货物下包括多个附件。两层的一对多。

注意：结果集中不能有同名字段。不然mybatis找不到映射规则。映射的结果面目全非。根本不是需要要的结果。

### 一对一

和一对多差异，对一assocaition。

改变PO对象，在主对象中加关联的对象

在映射文件中加resultMap，在map中加对一映射association。

注意在结果集中不能有同名字段，同名时，通过在映射时换名称。

<!-- 用户和扩展信息，一对一

type都是填主对象

resultMap结构可以继承，extends关键字

-->

<resultMap type="cn.mybatis.po.User" id="userUserInfoRM" extends="userRM">

<!-- 配置关系，对一

property填写PO对应关联对象

javaType 对应的元素类型

-->

<association property="userInfo" javaType="cn.mybatis.po.UserInfo">

<id property="id" column="USER\_INFO\_ID"/>

<result property="school" column="SCHOOL"/>

</association>

</resultMap>

<select id="findUserById" parameterType="string" resultMap="userUserInfoRM">

SELECT

u.id,u.user\_name,u.age,u.address,

i.id AS user\_info\_id,i.school

FROM

(SELECT \* FROM user\_c) u

LEFT JOIN

(SELECT \* FROM user\_info\_c) i

ON u.id=i.id

WHERE u.id=#{id}

</select>

### 多对多

SELECT

u.id,u.user\_name,u.age,u.address,

o.id AS order\_id,o.order\_no

FROM

(

SELECT

u.id,u.user\_name,u.age,u.address,

uo.order\_id

FROM

(SELECT id,user\_name,age,address FROM user\_c WHERE id='5') u

LEFT JOIN

(SELECT user\_id,order\_id FROM user\_order\_c) uo

ON u.id=uo.user\_id

) u

LEFT JOIN

(SELECT id,order\_no FROM order\_c) o

ON u.order\_id=o.id

<?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">

<!-- 找到某个业务的方法时，就要靠这个命名空间参数来区分 -->

<mapper namespace="cn.mybatis.po.OrderMapper">

<!-- 结果集映射对象 -->

<resultMap type="cn.mybatis.po.User" id="userOrderRM">

<id property="id" column="ID" javaType="string"/>

<result property="name" column="USER\_NAME" javaType="string"/>

<result property="age" column="AGE" javaType="integer"/>

<result property="address" column="ADDRESS" javaType="string"/>

<!-- 用户和订单多对多 -->

<collection property="orders" ofType="cn.mybatis.po.Order">

<id property="orderId" column="ORDER\_ID" javaType="string"/>

<result property="orderNo" column="ORDER\_NO" javaType="string"/>

</collection>

</resultMap>

<select id="findOrderByUserId" parameterType="string" resultMap="userOrderRM">

SELECT

u.id,u.user\_name,u.age,u.address,

o.id AS order\_id,o.order\_no

FROM

(

SELECT

u.id,u.user\_name,u.age,u.address,

uo.order\_id

FROM

(SELECT id,user\_name,age,address FROM user\_c WHERE id=#{userId}) u

LEFT JOIN

(SELECT user\_id,order\_id FROM user\_order\_c) uo

ON u.id=uo.user\_id

) u

LEFT JOIN

(SELECT id,order\_no FROM order\_c) o

ON u.order\_id=o.id

</select>

</mapper>

Mybatis使用时，严格按业务需求来获取数据。效率高的根源。

## 缓存

一级缓存Session级别，自己管理

二级缓存SessionFactory级别

### 一级缓存

@Test //一级缓存，Session级别

public void testLevelOne(){

SqlSession s1 = factory.openSession();

UserMapper mapper = s1.getMapper(UserMapper.class);

User user = mapper.get("5");

System.out.println(user);

UserMapper mapper2 = s1.getMapper(UserMapper.class);

User user2 = mapper2.get("5");

System.out.println(user2);

}

### 二级缓存默认开启

在全局配置文件中配置开启二级缓存。

在全局配置文件中sqlMapConfig.xml中配置启用

<!-- 全局配置 -->

<settings>

<!-- 开启二级缓存，默认开启，也就是true -->

<setting name="cacheEnabled" value="true"/>

</settings>

注意它所放置的位置有先后要求。它会有错误的提示。

还需要在映射文件中开启

<mapper namespace="cn.mybatis.po.UserMapper">

<!-- 对本Mapper映射是否开启二级缓存 -->

<cache/>

测试方法

public void testLevelTwo() throws IOException{

SqlSession session = factory.openSession();

User user = session.selectOne("mybatis.pojo.UserMapper.get", "5");

session.close(); // SqlSession不关闭它使用的是local session缓存，所以必须关闭

SqlSession session2 = factory.openSession();

User user2 = session2.selectOne("mybatis.pojo.UserMapper.get", "5");

}

注意：pojo对象必须序列化。

### 第三方二级缓存

Mybatis中的二级缓存的性能一般，它提供扩展，用户可以配置自己的缓存框架。习惯配置EhCache。

主流的缓存框架：EchCache（小型项目，单个项目），MemCache（小中型项目，集群），Redis（最主流，支持大中小，集群，分布式）

步骤：

1）导入jar支持

ehcache-core-2.6.5.jar 核心包

mybatis-ehcache-1.0.2.jar 和mybatis整合包

1. 配置ehcache配置文件。这个文件在core jar中ehcache-failsafe.xml，将它拷贝到项目的根目录，将它的名字改成ehcache.xml
2. 在映射文件中配置

<cache type="org.mybatis.caches.ehcache.EhcacheCache"/>

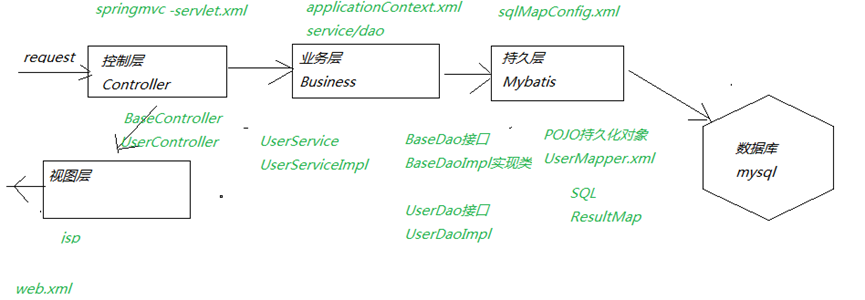
注意：

ehcache是基于本地电脑的内存使用缓存，所以使用缓存取数据时直接在本地电脑上取。转换成java对象就会是同一个内存地址，而redis它是在装有redis服务的电脑上（一般是另一台电脑），所以取数据时经过传输到本地，会对应到不同的内存地址

## 三大框架整合SSM，SpringMVC+Spring+Mybatis

步骤：

### 框架图



### 开发步骤

#### 创建web工程，导入jar

aopalliance-1.0.jar

asm-3.3.1.jar

c3p0-0.9.1.2.jar

cglib-2.2.2.jar

com.springsource.org.aspectj.weaver-1.6.8.RELEASE.jar

commons-fileupload-1.2.2.jar

commons-io-2.0.1.jar

commons-lang3-3.1.jar

commons-logging-1.1.1.jar

ehcache-core-2.6.5.jar

freemarker-2.3.19.jar

javassist-3.11.0.GA.jar

javassist-3.17.1-GA.jar

jstl-api-1.2.jar

jstl-impl-1.2.jar

junit-4.9.jar

log4j-1.2.16.jar

log4j-1.2.17.jar

mybatis-3.2.2.jar

mybatis-ehcache-1.0.2.jar

mybatis-spring-1.2.0.jar

mysql-connector-java-5.1.10-bin.jar

mysql-connector-java-5.1.28-bin.jar

ognl-3.0.6.jar

slf4j-api-1.6.2.jar

slf4j-api-1.7.5.jar

slf4j-log4j12-1.6.2.jar

slf4j-log4j12-1.7.5.jar

spring-aop-3.2.2.jar

spring-beans-3.2.2.jar

spring-context-3.2.2.jar

spring-core-3.2.2.jar

spring-expression-3.2.2.jar

spring-jdbc-3.2.2.jar

spring-orm-3.2.2.jar

spring-tx-3.2.2.jar

spring-web-3.2.2.jar

spring-webmvc-3.2.2.jar

standard-1.1.2.jar

#### PO对象

package cn.ssm.domain;

import java.io.Serializable;

public class User implements Serializable{

private String id;

private String name;

private Integer age;

private String address;

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public Integer getAge() {

return age;

}

public void setAge(Integer age) {

this.age = age;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

}

#### UserMapper.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="mapper.UserMapper">

<!-- 查询所有 -->

<select id="findAll" resultType="cn.ssm.domain.User">

select \* from user\_c

</select>

</mapper>

#### BaseDao.java

package cn.ssm.dao;

import java.util.List;

//泛型接口

public abstract interface BaseDao<T> {

public List<T> findAll();

}

#### BaseDaoImpl.java

package cn.ssm.dao.impl;

import java.util.List;

import javax.annotation.Resource;

import org.apache.ibatis.session.SqlSessionFactory;

import org.mybatis.spring.support.SqlSessionDaoSupport;

import cn.ssm.dao.BaseDao;

import cn.ssm.po.UserMapper;

public class BaseDaoImpl<T> extends SqlSessionDaoSupport implements BaseDao<T>{

private String namespace;

public String getNamespace() {

return namespace;

}

public void setNamespace(String namespace) {

this.namespace = namespace;

}

//注入sqlSessionFactory 版本 mybatis3.2.2 插件mybatis-spring1.2.0 自己注入

@Resource

public void setSqlSessionFactory(SqlSessionFactory sqlSessionFactory) {

super.setSqlSessionFactory(sqlSessionFactory);

}

@Override

public List<T> findAll() {

return this.getSqlSession().selectList(namespace+".findAll");

}

}

#### UserDao.java

package cn.ssm.dao;

import cn.ssm.po.User;

public interface UserDao extends BaseDao<User>{

}

#### UserDaoImpl.java

package cn.ssm.dao.impl;

import java.util.List;

import org.springframework.stereotype.Component;

import cn.ssm.dao.UserDao;

import cn.ssm.po.User;

@Component

public class UserDaoImpl extends BaseDaoImpl<User> implements UserDao{

//只初始化一次

public UserDaoImpl() {

super.setNamespace("cn.ssm.po.UserMapper");

}

}

#### UserService.java

package cn.ssm.service;

import java.util.List;

import cn.ssm.po.User;

public interface UserService {

public List<User> findAll();

}

#### UserServiceImpl.java

package cn.ssm.service.impl;

import java.util.List;

import javax.annotation.Resource;

import org.springframework.stereotype.Service;

import cn.ssm.dao.UserDao;

import cn.ssm.po.User;

import cn.ssm.service.UserService;

@Service

public class UserServiceImpl implements UserService{

@Resource

private UserDao userDao;

public List<User> findAll() {

return userDao.findAll();

}

}

#### BaseController.java

package cn.ssm.web.controller;

import java.text.SimpleDateFormat;

import org.springframework.beans.propertyeditors.CustomDateEditor;

import org.springframework.web.bind.ServletRequestDataBinder;

import org.springframework.web.bind.annotation.InitBinder;

public class BaseController {

//日期转换，通过注解 @InitBinder

@InitBinder

public void InitBinder (ServletRequestDataBinder binder){

binder.registerCustomEditor(

java.util.Date.class,

new CustomDateEditor(new SimpleDateFormat("yyyy-MM-dd"), true));

}

}

#### UserController.java

package cn.ssm.web.controller;

import java.util.List;

import javax.annotation.Resource;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.RequestMapping;

import cn.ssm.po.User;

import cn.ssm.service.UserService;

@Controller

@RequestMapping("/user")

public class UserController extends BaseController{

@Resource

private UserService userService;

//列表

@RequestMapping("/list") // /user/list.action

public String list(Model model){

List<User> dataList = userService.findAll();

model.addAttribute("dataList", dataList);

return "user/jUserList.jsp";

}

}

#### db-mysql.properties

c3p0.driverClass=com.mysql.jdbc.Driver

c3p0.jdbcUrl=jdbc:mysql://localhost:3306/mybatisdb?characterEncoding=utf-8

c3p0.user=root

c3p0.password=root

#### log4j.properties

log4j.rootLogger=INFO, Console

#Console

log4j.appender.Console=org.apache.log4j.ConsoleAppender

log4j.appender.Console.layout=org.apache.log4j.PatternLayout

log4j.appender.Console.layout.ConversionPattern=%d [%t] %-5p [%c] - %m%n

log4j.logger.java.sql.ResultSet=INFO

log4j.logger.org.apache=INFO

log4j.logger.java.sql.Connection=DEBUG

log4j.logger.java.sql.Statement=DEBUG

log4j.logger.java.sql.PreparedStatement=DEBUG

#### sqlMapConfig.xml

<?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>

</configuration>

#### applicationContext.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:p="http://www.springframework.org/schema/p"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:tx="http://www.springframework.org/schema/tx"

xmlns:aop="http://www.springframework.org/schema/aop"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.1.xsd

http://www.springframework.org/schema/aop

http://www.springframework.org/schema/aop/spring-aop-3.1.xsd

http://www.springframework.org/schema/tx

http://www.springframework.org/schema/tx/spring-tx-3.1.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.1.xsd">

<!-- 包扫描 -->

<context:component-scan base-package="cn/ssm/service,cn/ssm/dao"/>

<!-- 数据源，sessionfactory，tx事务 -->

<context:property-placeholder location="classpath:db-mysql.properties"/>

<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">

<!-- 数据库连接参数 -->

<property name="driverClass" value="${c3p0.driverClass}"/>

<property name="jdbcUrl" value="${c3p0.jdbcUrl}"/>

<property name="user" value="${c3p0.user}"/>

<property name="password" value="${c3p0.password}"/>

</bean>

<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">

<property name="dataSource" ref="dataSource"/>

<!-- 和mybatis进行整合 -->

<property name="configLocation" value="classpath:sqlMapConfig.xml"/>

<property name="mapperLocations" value="classpath:mapper/\*Mapper.xml"></property>

</bean>

<bean id="txManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

<property name="dataSource" ref="dataSource"/>

</bean>

<tx:advice id="txAdvice" transaction-manager="txManager">

<tx:attributes>

<tx:method name="insert\*" propagation="REQUIRED"/>

<tx:method name="update\*" propagation="REQUIRED"/>

<tx:method name="save\*" propagation="REQUIRED"/>

<tx:method name="delete\*" propagation="REQUIRED"/>

<tx:method name="find\*" read-only="true"/>

<tx:method name="get\*" read-only="true"/>

<tx:method name="\*" read-only="true"/>

</tx:attributes>

</tx:advice>

<aop:aspectj-autoproxy proxy-target-class="true"/>

<aop:config>

<aop:pointcut expression="execution(\* cn.ssm.service.\*.\*(..))" id="pointCut"/>

<aop:advisor advice-ref="txAdvice" pointcut-ref="pointCut"/>

</aop:config>

</beans>

#### smvc-servlet.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:mvc="http://www.springframework.org/schema/mvc"

xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/mvc

http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.2.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.2.xsd">

<!-- 注解支持，AnnotactionUrlHandlerMapping+jackson -->

<mvc:annotation-driven/>

<!-- 包扫描 -->

<context:component-scan base-package="cn/ssm/web/controller"/>

<!-- 内部资源视图解析器 前缀/WEB-INF/+逻辑名hello+后缀.jsp -->

<bean id="jspViewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="prefix" value="/WEB-INF/pages/"/>

<property name="suffix" value=""/>

</bean>

<!-- 文件上传解析器，规定，要求id属性必须是multipartResolver，限定上传10M -->

<bean id="multipartResolver" class="org.springframework.web.multipart.commons.CommonsMultipartResolver">

<property name="maxUploadSize" value="10485760"/>

</bean>

</beans>

#### web.xml

<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns="http://java.sun.com/xml/ns/javaee"

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"

id="WebApp\_ID" version="3.0">

<display-name>ssm</display-name>

<welcome-file-list>

<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

<!-- spring监听 -->

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:applicationContext.xml</param-value>

</context-param>

<!-- 核心分发器DispacherServlet -->

<servlet>

<servlet-name>springmvc</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:smvc-servlet.xml</param-value>

</init-param>

</servlet>

<!-- 解决中文乱码，通过spring提供的过滤器 -->

<filter>

<filter-name>CharacterEncodingFilter</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>utf-8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>CharacterEncodingFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<servlet-mapping>

<servlet-name>springmvc</servlet-name>

<url-pattern>\*.action</url-pattern>

</servlet-mapping>

</web-app>

#### Jsp页面

/ssm/WebContent/WEB-INF/pages/sysadmin/user/jUserList.jsp

<%@ page language="java" contentType="text/html; charset=UTF-8"

pageEncoding="UTF-8"%>

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<title>Insert title here</title>

</head>

<body>

<table>

<tr>

<td>序号</td>

<td>姓名</td>

<td>年龄</td>

<td>地址</td>

</tr>

<c:forEach items="${dataList}" var="u" varStatus="status">

<tr>

<td>${status.index+1}</td>

<td>${u.name}</td>

<td>${u.age}</td>

<td>${u.address}</td>

</tr>

</c:forEach>

</table>

</body>

</html>