# mybatis框架

## mybatis是什么？

mybatis是一个持久层的框架，是apache下的顶级项目。

mybatis让程序将主要精力放在sql上，通过mybatis提供的映射方式，自由灵活生成（半自动化，大部分需要程序员编写sql）满足需要sql语句。

mybatis可以将向 preparedStatement中的输入参数自动进行输入映射，将查询结果集灵活映射成java对象。

## mybatis框架原理

SqlMapConfig.xml（是mybatis的全局配置文件，名称不固定的）

配置了数据源、事务等mybatis运行环境

SqlSessionFactory（会话工厂），根据配置文件创建工厂

作用：创建SqlSession

msyql

SqlSession（会话），是一个接口，面向用户（程序员）的接口

作用：操作数据库（发出sql增、删、改、查）

Executor（执行器），是一个接口（基本执行器、缓存执行器）

作用：SqlSession内部通过执行器操作数据库

mapped statement（底层封装对象）

作用：对操作数据库存储封装，包括 sql语句，输入参数、输出结果类型

输入参数类型

java简单类型

hashmap

pojo自定义

输出结果 类型

java简单类型

hashmap

pojo自定义

# 入门程序

## 需求

根据学生id（主键）查询用户信息

根据学生名称模糊查询用户信息

添加学生

删除学生

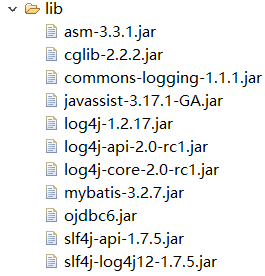
修改学生

## 准备工作

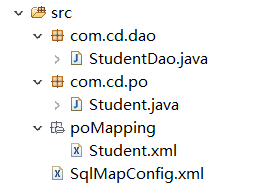
使用Oracle数据库，创建学生表

## 程序开始

创建web项目，导入所需的jar包：



即将创建的包结构及文件：



接下来依次创建，首先在src目录下创建SqlMapConfig.xml文件，内容如下：

<?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>

<!-- 和spring整合后 environments配置将废除-->

<environments default=*"development"*>

<environment id=*"development"*>

<!-- 使用jdbc事务管理-->

<transactionManager type=*"JDBC"* />

<!-- 数据库连接池-->

<dataSource type=*"POOLED"*>

<property name=*"driver"* value=*"oracle.jdbc.driver.OracleDriver"* />

<property name=*"url"* value=*"jdbc:oracle:thin:@localhost:1521:orcl"* />

<property name=*"username"* value=*"scott"* />

<property name=*"password"* value=*"tiger"* />

</dataSource>

</environment>

</environments>

<mappers>

<mapper resource=*"poMapping/Student.xml"*/>

</mappers>

</configuration>

接下来创建Student实体类：

代码略

接下来在poMapping包下创建Student.xml映射文件：

<?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=*"po.student"*>

<select id=*"selectStuById"* parameterType=*"int"* resultType=*"com.cd.po.Student"*>

select \* from student where stuid=#{aaa}

</select>

</mapper>

这里注意要将Student.xml文件添加到SqlMapConfig.xml文件中（上面的SqlMapConfig.xml文件中已经添加好了）

下面就可以在StudentDao中使用了：

**public class** StudentDao {

**public** Student queryById(**int** id) **throws** IOException {

Student stu = **null**;

// 1.读取SqlMapConfig.xml配置文件

String path = "SqlMapConfig.xml";

InputStream in = Resources.*getResourceAsStream*(path);

// 创建会话工厂

SqlSessionFactory factory = **new** SqlSessionFactoryBuilder().build(in);

// 获得会话

SqlSession session = factory.openSession();

// 第一个参数：映射文件中select标签的id（statement的id）

// 第二个参数：和映射文件中指定的parameterType类型相同的参数

stu = session.selectOne("selectStuById", id);

**return** stu;

}

}

根据姓氏查询：

Student.xml:

<select id=*"selectByFirstName"* parameterType=*"java.lang.String"* resultType=*"com.cd.po.Student"*>

select \* from student where stuname like '${value}%'

</select>

StudentDao:

**public** List<Student> selectByFirstName(String firstName) **throws** IOException {

List<Student> list = **null**;

// 1.读取SqlMapConfig.xml配置文件

String path = "SqlMapConfig.xml";

InputStream in = Resources.*getResourceAsStream*(path);

// 创建会话工厂

SqlSessionFactory factory = **new** SqlSessionFactoryBuilder().build(in);

// 获得会话

SqlSession session = factory.openSession();

list = session.selectList("selectByFirstName", firstName);

**return** list;

}

增加：

Student.xml:

<insert id=*"addStudent"* parameterType=*"com.cd.po.Student"*>

insert into student values(#{stuid},#{stuno},#{stuname},#{gender},#{age})

</insert>

StudentDao:

**public** **void** addStudent(Student stu) **throws** IOException {

// 1.读取SqlMapConfig.xml配置文件

String path = "SqlMapConfig.xml";

InputStream in = Resources.*getResourceAsStream*(path);

// 创建会话工厂

SqlSessionFactory factory = **new** SqlSessionFactoryBuilder().build(in);

// 获得会话

SqlSession session = factory.openSession();

session.insert("addStudent", stu);

// 提交事务

session.commit();

session.close();

}

删除：

Student.xml:

<delete id=*"deleteStudentById"* parameterType=*"java.lang.Integer"*>

delete from student where stuid=#{stuid}

</delete>

StudentDao：

**public** **void** deleteStudentById(**int** id) **throws** IOException {

// 1.读取SqlMapConfig.xml配置文件

String path = "SqlMapConfig.xml";

InputStream in = Resources.*getResourceAsStream*(path);

// 创建会话工厂

SqlSessionFactory factory = **new** SqlSessionFactoryBuilder().build(in);

// 获得会话

SqlSession session = factory.openSession();

session.delete("deleteStudentById", id);

// 提交事务

session.commit();

session.close();

}

修改：

Student.xml

<update id=*"updateStudentById"* parameterType=*"com.cd.po.Student"*>

update student set stuno=#{stuno},stuname=#{stuname},gender=#{gender},age=#{age} where stuid=#{stuid}

</update>

studentDao：

**public** **void** updateStudentById(Student stu) **throws** IOException {

// 1.读取SqlMapConfig.xml配置文件

String path = "SqlMapConfig.xml";

InputStream in = Resources.*getResourceAsStream*(path);

// 创建会话工厂

SqlSessionFactory factory = **new** SqlSessionFactoryBuilder().build(in);

// 获得会话

SqlSession session = factory.openSession();

session.update("updateStudentById", stu);

// 提交事务

session.commit();

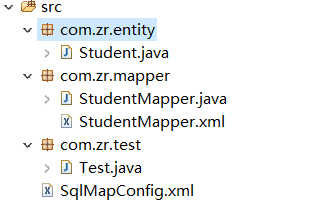
session.close();

}

## mapper接口模式

以上是传统的dao方式的实现，mybatis还可以使用mapper接口模式，程序员只需要编写mapper接口即可。

以下是采用mapper接口模式的包结构：



StudentMapper.xml文件：

<?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">

<!-- 这里的namespace是mapper接口的路径 -->

<mapper namespace=*"com.zr.mapper.StudentMapper"*>

<select id=*"selectStuById"* parameterType=*"int"* resultType=*"com.zr.entity.Student"*>

select \* from student where stuid=#{aaa}

</select>

<select id=*"selectByFirstName"* parameterType=*"java.lang.String"* resultType=*"com.zr.entity.Student"*>

select \* from student where stuname like '${value}%'

</select>

<insert id=*"addStudent"* parameterType=*"com.zr.entity.Student"*>

insert into student values(#{stuid},#{stuno},#{stuname},#{gender},#{age})

</insert>

<delete id=*"deleteStudentById"* parameterType=*"java.lang.Integer"*>

delete from student where stuid=#{stuid}

</delete>

<update id=*"updateStudentById"* parameterType=*"com.zr.entity.Student"*>

update student set stuno=#{stuno},stuname=#{stuname},gender=#{gender},age=#{age} where stuid=#{stuid}

</update>

</mapper>

接下来编写StudentMapper.java接口

这个接口要符合一定的规范：

1、在mapper.xml中namespace等于mapper接口地址

2、mapper.java接口中的方法名和mapper.xml中statement的id一致

3、mapper.java接口中的方法输入参数类型和mapper.xml中statement的parameterType指定的类型一致。

1. mapper.java接口中的方法返回值类型和mapper.xml中statement的resultType指定的类型一致。

根据以上规范编写的StudentMapper：

**public** **interface** StudentMapper {

**public** Student selectStuById(**int** stuid);

**public** List<Student> selectByFirstName(String firstName);

**public** **void** addStudent(Student stu);

**public** **void** deleteStudentById(**int** id);

**public** **void** updateStudentById(Student stu);

}

在SqlMapConfig.xml中配置StudentMapper.xml文件，这里采用批量的方式：

<mappers>

<!--

批量加载mapper，遵循的规范：

1.使用mapper接口方式

2.mapper接口及其对应的mapper.xml文件在同一目录且名称一致

-->

<package name=*"com.zr.mapper"*/>

</mappers>

最后测试一下：

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

// 1.读取SqlMapConfig.xml配置文件

String path = "SqlMapConfig.xml";

InputStream in = Resources.*getResourceAsStream*(path);

// 创建会话工厂

SqlSessionFactory factory = **new** SqlSessionFactoryBuilder().build(in);

// 获得会话

SqlSession session = factory.openSession();

//mybatis自动生成mapper代理对象

StudentMapper sm = session.getMapper(StudentMapper.**class**);

System.*out*.println(sm.selectStuById(4));

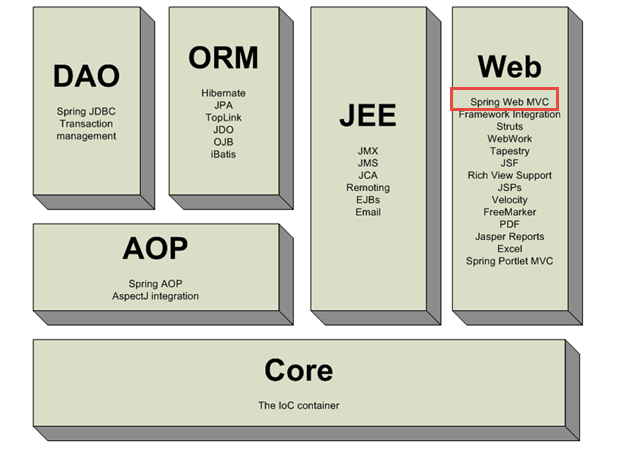
}

# springmvc框架

## 什么是springmvc

springmvc是spring框架的一个模块，springmvc和spring无需通过中间整合层进行整合。

springmvc是一个基于mvc的web框架。



## mvc在b/s系统 下的应用

mvc是一个设计模式，mvc在b/s系统 下的应用：

C 控制器

request请求

M 模型(model)

pojo

action

service

dao

请求模型进行处理

处理结果返回

V 视图

view

视图渲染

将模型数据填充到request域

response响应

## springmvc框架

1request请求 url

前端控制器

DispatcherServlet

接口用户请求，响应

Handler处理器

平常叫做controller

处理器映射器

HandlerMapping

2请求查找Handler

3返回一个执行链

Handler

处理器适配器

HandlerAdapter

去执行Handler

4请求适配器执行Handler

5执行

6返回ModelAndView

7返回ModelAndView

view视图

jsp

freemarker

excel

pdf

视图解析器

View resolver

8请求进行视图解析

9返回view

10视图渲染

将模型数据填充到request域

11response响应

组件：

1、前端控制器DispatcherServlet（不需要程序员开发）

作用接收请求，响应结果，相当于转发器，中央处理器。

有了DispatcherServlet减少了其它组件之间的耦合度。

2、处理器映射器HandlerMapping(不需要程序员开发)

作用：根据请求的url查找Handler

3、处理器适配器HandlerAdapter

作用：按照特定规则（HandlerAdapter要求的规则）去执行Handler

4、处理器Handler(需要程序员开发)

注意：编写Handler时按照HandlerAdapter的要求去做，这样适配器才可以去正确执行Handler

5、视图解析器View resolver(不需要程序员开发)

作用：进行视图解析，根据逻辑视图名解析成真正的视图（view）

6、视图View(需要程序员开发jsp)

View是一个接口，实现类支持不同的View类型（jsp、freemarker、pdf...）

# 入门程序

Web.xml中配置中央控制器：

<!-- springmvc前端控制器(中央控制器) -->

<servlet>

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

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

<!-- contextConfigLocation配置springmvc加载的配置文件（配置处理器映射器、适配器等等）

如果不配置contextConfigLocation，默认加载的是/WEB-INF/servlet名称-serlvet.xml（springmvc-servlet.xml）

-->

<init-param>

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

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

</init-param>

</servlet>

<servlet-mapping>

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

<!--

\*.action，访问以.action结尾 由DispatcherServlet进行解析

-->

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

</servlet-mapping>

在src目录下创建springmvc.xml:

<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"*

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

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

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

*http://www.springframework.org/schema/mvc*

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

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

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

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

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

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

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

<!-- 处理器映射器 -->

<bean class=*"org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"*></bean>

<!-- 另一个处理器映射器 -->

<bean class=*"org.springframework.web.servlet.handler.SimpleUrlHandlerMapping"*>

<property name=*"mappings"*>

<props >

<prop key=*"/abc.action"*>StudentController</prop>

<prop key=*"/aaa.action"*>StudentController</prop>

</props>

</property>

</bean>

<!-- 处理器适配器:要求handler必须实现Controller接口 -->

<bean class=*"org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter"*></bean>

<!-- 另一个处理器适配器：要求handler必须实现HttpRequestHandler接口 -->

<bean class=*"org.springframework.web.servlet.mvc.HttpRequestHandlerAdapter"*></bean>

<!-- 注解的方式配置处理器适配器和处理器映射器 -->

<!-- <bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping"></bean>

<bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter"></bean> -->

<!-- 注解二合一 -->

<mvc:annotation-driven></mvc:annotation-driven>

<!-- 视图解析器 -->

<bean class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*></bean>

<!-- 配置handller -->

<bean id=*"StudentController"* name=*"/stulist.action"* class=*"com.cd.controller.StudentController"*></bean>

<bean name=*"/reqlist.action"* class=*"com.cd.controller.StudentRequestController"*></bean>

<!-- <bean class="com.cd.controller.StudentAnnotionController"></bean> -->

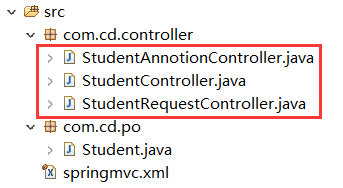
<!-- 对于注解的Handler建议使用组件扫描-->

<context:component-scan base-package=*"com.cd.controller"*></context:component-scan>

</beans>

创建Student实体类

这里讲解三种实现Handler的方式：



第一种：实现Controller接口，重写handleRequest方法：

**public** **class** StudentController **implements** Controller {

@Override

**public** ModelAndView handleRequest(HttpServletRequest arg0,

HttpServletResponse arg1) **throws** Exception {

ArrayList<Student> list = **new** ArrayList<>();

list.add(**new** Student(1, "张三", 20, 100, **new** Date()));

list.add(**new** Student(2, "张三2", 22, 200, **new** Date()));

//创建ModelAndView

ModelAndView mv = **new** ModelAndView();

//相当于rquest.setAttribute()

mv.addObject("list", list);

//指定要跳转的jsp

mv.setViewName("/index.jsp");

**return** mv;

}

}

第二种：实现HttpRequestHandler接口，实现handleRequest方法：

**public** **class** StudentRequestController **implements** HttpRequestHandler {

@Override

**public** **void** handleRequest(HttpServletRequest arg0, HttpServletResponse arg1)

**throws** ServletException, IOException {

ArrayList<Student> list = **new** ArrayList<>();

list.add(**new** Student(1, "张三", 20, 100, **new** Date()));

list.add(**new** Student(2, "张三2", 22, 200, **new** Date()));

arg0.setAttribute("list", list);

//指定要跳转的jsp

arg0.getRequestDispatcher("/index.jsp").forward(arg0, arg1);

}

}

第三种：注解方式

@Controller

**public** **class** StudentAnnotionController {

@RequestMapping("/stus")

**public** ModelAndView queryStuList(){

ArrayList<Student> list = **new** ArrayList<>();

list.add(**new** Student(1, "张三", 20, 100, **new** Date()));

list.add(**new** Student(2, "张三2", 22, 200, **new** Date()));

//创建ModelAndView

ModelAndView mv = **new** ModelAndView();

//相当于rquest.setAttribute()

mv.addObject("list", list);

//指定要跳转的jsp

mv.setViewName("/index.jsp");

**return** mv;

}

}

Index.jsp 页面

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

<body>

<table border=*"1"* width=*"100%"*>

<tr>

<th>sid</th>

<th>name</th>

<th>age</th>

<th>sal</th>

<th>birthday</th>

</tr>

<c:forEach items=*"*${list }*"* var=*"a"*>

<tr>

<td>${a.sid }</td>

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

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

<td>${a.sal }</td>

<td>${a.birthday }</td>

</tr>

</c:forEach>

</table>

</body>