## MyBatis与SpringMVC结合

1. 在web.xml中创建spring listener

|  |
| --- |
| <!--Spring配置： needed for ContextLoaderListener -->  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:spring/spring-\*.xml</param-value>  </context-param>  <!-- Bootstraps the root web application context before servlet initialization -->  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener> |

1. 在src/main/resources下创建spring-mybatis.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:p=*"http://www.springframework.org/schema/p"*  xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:tx=*"http://www.springframework.org/schema/tx"*  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-4.0.xsd*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd*  *http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd*  *http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.0.xsd"*>  <!-- 数据库连接池 -->  <!-- 加载配置文件 -->  <context:property-placeholder location=*"classpath:resource/\*.properties"* />  <!-- 数据库连接池 -->  <bean id=*"dataSource"* class=*"com.alibaba.druid.pool.DruidDataSource"*  destroy-method=*"close"*>  <property name=*"url"* value=*"${jdbc.url}"* />  <property name=*"username"* value=*"${jdbc.username}"* />  <property name=*"password"* value=*"${jdbc.password}"* />  <property name=*"driverClassName"* value=*"${jdbc.driver}"* />  <property name=*"maxActive"* value=*"10"* />  <property name=*"minIdle"* value=*"5"* />  </bean>  <!-- 配置sqlsessionFactory -->  <bean id=*"sqlSessionFactory"* class=*"org.mybatis.spring.SqlSessionFactoryBean"*>  <property name=*"configLocation"* value=*"classpath:mybatis/SqlMapConfig.xml"*></property>  <property name=*"dataSource"* ref=*"dataSource"*></property>  </bean>  <!-- 配置扫描包，加载mapper代理对象 -->  <bean class=*"org.mybatis.spring.mapper.MapperScannerConfigurer"*>  <property name=*"basePackage"* value=*"mapper"*></property>  </bean>  </beans> |

1. 在src/main/resources 下面创建resource/db.properties

|  |
| --- |
| jdbc.driver=com.mysql.jdbc.Driver  jdbc.url=jdbc:mysql://127.0.0.1:3306/mybatis?characterEncoding=utf-8  jdbc.username=root  jdbc.password=123456 |

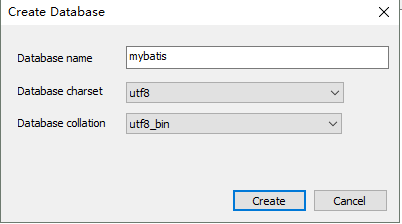
1. 在src/main/resources 下面创建\mybatis\SqlMapConfig.xml

|  |
| --- |
| <?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> |

1. 在pom.xml中添加依赖

|  |
| --- |
| <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis</artifactId>  <version>3.2.8</version>  </dependency>    <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>5.1.24</version>  </dependency>    <!-- Mybatis和SpringMVC整合的依赖 -->  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis-spring</artifactId>  <version>1.2.2</version>  </dependency>  <!-- 导入数据源 -->  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid</artifactId>  <version>1.0.9</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-tx</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-jdbc</artifactId>  <version>${spring.version}</version>  </dependency> |

1. 创建数据库



1. 创建表

|  |
| --- |
| USE mybatis;  CREATE TABLE computer(  id INT(11) NOT NULL AUTO\_INCREMENT,  tradeMark VARCHAR(20),  price FLOAT(10),  pic VARCHAR(255),  PRIMARY KEY (id)  ) |

1. 创建与表对应的pojo类

|  |
| --- |
| **package** pojo;  **public** **class** ComputerPojo {  **private** **int** cid;  **private** String tradeMark;  **private** **float** price;  **private** String pic;  **public** ComputerPojo() {  **super**();  }  **public** ComputerPojo(**int** cid, String tradeMark, **float** price, String pic) {  **super**();  **this**.cid = cid;  **this**.tradeMark = tradeMark;  **this**.price = price;  **this**.pic = pic;  }  **public** **int** getCid() {  **return** cid;  }  **public** **void** setCid(**int** cid) {  **this**.cid = cid;  }  **public** String getTradeMark() {  **return** tradeMark;  }  **public** **void** setTradeMark(String tradeMark) {  **this**.tradeMark = tradeMark;  }  **public** **float** getPrice() {  **return** price;  }  **public** **void** setPrice(**float** price) {  **this**.price = price;  }  **public** String getPic() {  **return** pic;  }  **public** **void** setPic(String pic) {  **this**.pic = pic;  }  @Override  **public** String toString() {  **return** "ComputerPojo [cid=" + cid + ", tradeMark=" + tradeMark + ", price=" + price + ", pic=" + pic + "]";  }    } |

1. 创建一个mapper的包,在里面创建ComputerPojo的接口

|  |
| --- |
| **package** mapper;  **import** java.util.List;  **import** pojo.ComputerPojo;  **public** **interface** ComputerMapper {  **public** List<ComputerPojo> getAllComputers();  } |

1. 在ComputerMapper.java同级目录下创建一个同名的ComputerMapper.xml

ComputerMapper.xml内容如下

|  |
| --- |
| <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <!--  namespace: mapper接口的全限定名  select id: mapper接口中的方法  resultType: mapper接口中的方法的返回类型/注意如果是List<ComputerPojo>,  则应该写成ComputerPojo而不是java.util.List  -->  <mapper namespace=*"mapper.ComputerMapper"*>  <select id=*"getAllComputers"* resultType=*"pojo.ComputerPojo"*>  SELECT id AS cid, tradeMark, price, pic FROM computer  </select>  </mapper> |

1. Service类如下

|  |
| --- |
| **package** service;  **import** java.util.List;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Service;  **import** mapper.ComputerMapper;  **import** model.Computer;  **import** pojo.ComputerPojo;  @Service  **public** **class** ComputerService {  @Autowired  **private** ComputerMapper computerMapper;    **public** List<ComputerPojo> getAllComputerPojo(){    List<ComputerPojo> comList = computerMapper.getAllComputers();    **return** comList;  }  } |

1. Controller类如下

|  |
| --- |
| package controller;  import java.util.List;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.validation.BindingResult;  import org.springframework.validation.annotation.Validated;  import org.springframework.web.bind.annotation.RequestBody;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestMethod;  import org.springframework.web.bind.annotation.ResponseBody;  import org.springframework.web.bind.annotation.RestController;  import MyException.PCFormException;  import model.AjaxModel;  import model.Computer;  import pojo.ComputerPojo;  import service.ComputerService;  @RestController  @RequestMapping(path="/RESTComputer")  public class RestFullComputerController {    @Autowired  private ComputerService computerService;  @RequestMapping(path="/getComPojos",method=RequestMethod.GET)  @ResponseBody  public List<ComputerPojo> getComputerPojos(){    return computerService.getAllComputerPojo();  }  } |

可能遇到问题 Invalid Bound Statement,这是因为资源文件没有在运行时被加载，解决方案：在pom.xml中添加

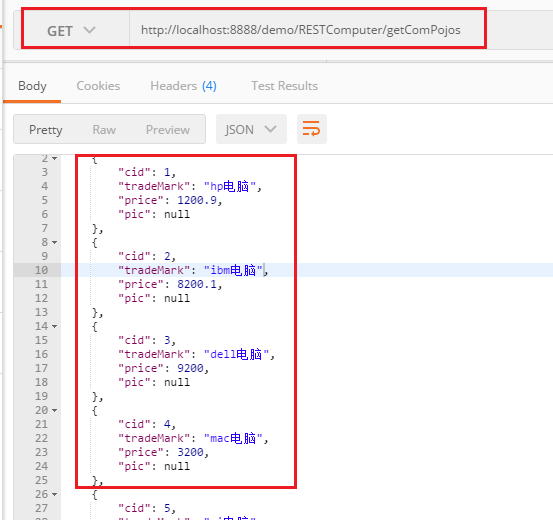
|  |
| --- |
| <resources>  <resource>  <directory>src/main/resources</directory>  <includes>  <include>\*\*/\*.properties</include>  <include>\*\*/\*.xml</include>  </includes>  <filtering>false</filtering>  </resource>  <resource>  <directory>src/main/java</directory>  <includes>  <include>\*\*/\*.properties</include>  <include>\*\*/\*.xml</include>  </includes>  <filtering>false</filtering>  </resource>  </resources> |

测试:

Mysql 中数据



POSTMAN中进行，看是否返回json数据



## 查询操作:

### 查询结果是List封装的java对象

|  |
| --- |
| <!--  namespace: mapper接口的全限定名  select id: mapper接口中的方法  resultType: mapper接口中的方法的返回类型/注意如果是List<ComputerPojo>,  则应该写成ComputerPojo而不是java.util.List  -->  <mapper namespace=*"mapper.ComputerMapper"*>  <select id=*"getAllComputers"* resultType=*"pojo.ComputerPojo"*>  SELECT id as cid, tradeMark, price, pic FROM computer  </select>  </mapper> |

附带查询条件情况: 接口文件如下

|  |
| --- |
| package mapper;  import java.util.List;  import pojo.ComputerPojo;  public interface ComputerMapper {  public List<ComputerPojo> getAllComputers();    //按id进行查询  public ComputerPojo getById(Integer cid);  } |

Mapper.xml

|  |
| --- |
| <!-- public ComputerPojo getById(); -->  <select id=*"getById"* resultType=*"pojo.ComputerPojo"*>  SELECT id AS cid, tradeMark, price, pic FROM computer WHERE id = #{cid}  </select> |

在多参的情况,

A)可以考虑使用@Param

|  |
| --- |
| **public** ComputerPojo getByIdAndTradeMark(@Param(value="cid")Integer cid  , @Param(value="tradeMark")String tradeMark); |

1. 可以使用MaBatis默认的参数Map

|  |
| --- |
| <!-- public ComputerPojo getByIdAndTradeMarkUsingParamMap(Integer cid, String tradeMark); -->  <!-- MyBatis传参时,默认封装到Map结合里  key param1,param2,param3  value #{param1},#{param2},#{param3}  -->  <select id=*"getByIdAndTradeMarkUsingParamMap"* resultType=*"pojo.ComputerPojo"*>  SELECT id AS cid, tradeMark, price, pic FROM computer WHERE id = #{param1}  AND tradeMark LIKE #{param2}  </select> |

1. 可以将参数封装到Pojo对象里

Mapper.java

|  |
| --- |
| **public** ComputerPojo getByPojo(ComputerPojo computerPojo); |

Mapper.xml

|  |
| --- |
| <!-- public ComputerPojo getByPojo(ComputerPojo computerPojo); -->  <select id=*"getByPojo"* resultType=*"pojo.ComputerPojo"*>  SELECT id AS cid, tradeMark, price, pic FROM computer WHERE id = #{cid}  AND tradeMark LIKE #{tradeMark}  </select> |

测试类:

|  |
| --- |
| @Test  **public** **void** testGetByPojo(){  ComputerPojo computerPojo = **new** ComputerPojo();  computerPojo.setCid(1);  computerPojo.setTradeMark("hp%");  System.***out***.println(computerMapper.getByPojo(computerPojo));  } |

## $与#的区别

#：在数据库中会进行sql的预编译

$: 只是简单的字符串替换，容易造成SQL注入攻击，从而泄露数据

什么情况下使用$, 在数据进行了切片，例如分别存到不同的表中

table\_${year},然后在mapper.java中可以传${year}的变量值

本例如下

Mapper.xml

|  |
| --- |
| <!-- public List<ComputerPojo> getAllComputersByTabName(String tableName); -->  <select id=*"getAllComputersByTabName"* resultType=*"pojo.ComputerPojo"*>  SELECT id as cid, tradeMark, price, pic FROM ${tableName}  </select> |

Mapper.java

|  |
| --- |
| **public** List<ComputerPojo> getAllComputersByTabName(@Param(value="tableName")String tableName); |

测试类

|  |
| --- |
| @Test  **public** **void** testGetAllComputersByTabName(){  List<ComputerPojo> comList = computerMapper.getAllComputersByTabName("computer");  System.***out***.println(comList);  } |

## 插入操作：

|  |
| --- |
| <!-- public void insertComputer(ComputerPojo computerPojo); -->  <insert id=*"insertComputer"* >  insert into computer(tradeMark, price, pic) values(#{tradeMark},#{price},#{pic})  </insert> |

## 删除操作:

|  |
| --- |
| <!-- public void deleteComputer(@Param(value="cid") Integer cid); -->  <delete id=*"deleteComputer"*>  delete from computer where id = #{cid}  </delete> |

## 更新操作：

|  |
| --- |
| <!-- public void updateComputerPrice(ComputerPojo computerPojo); -->  <update id=*"updateComputerPrice"*>  UPDATE computer SET price = #{price} WHERE id = #{cid}  </update> |