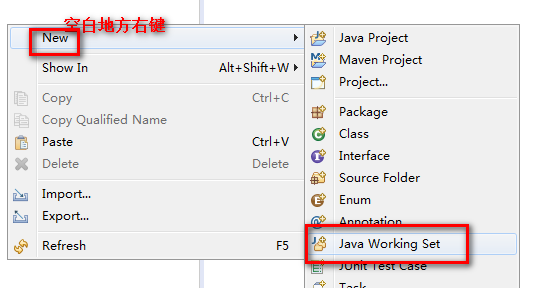
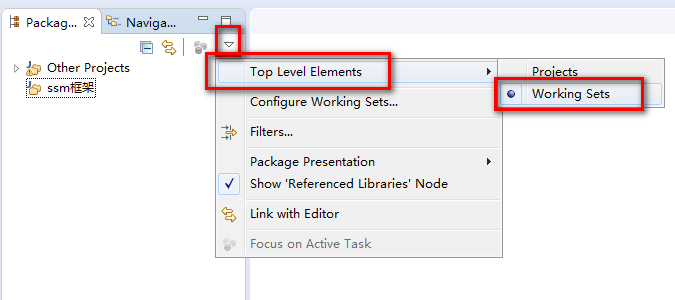
# 3.eclipse搭建ssm框架

## 3.1.创建一个工程

如果左侧项目过多，可以创建working set来管理





1.选择骨架（webapps）

2.groupId：cn.tedu artifactid：ORDER-USER-SSM

3.添加文件夹结构：src/main/java

4.在pom文件中添加插件资源

添加source生成插件（打包时连同源码一起打包）（课前资料，maven中的插件）

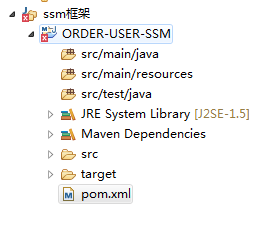
|  |
| --- |
| <build>  <finalName>ORDER-USER-SSM</finalName>  <plugins>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-source-plugin</artifactId>  <configuration>  <attach>true</attach>  </configuration>  <executions>  <execution>  <phase>compile</phase>  <goals>  <goal>jar</goal>  </goals>  </execution>  </executions>  </plugin>  </plugins>  </build> |

添加编译插件

|  |
| --- |
| <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <configuration>  <source>1.8</source>  <target>1.8</target>  <encoding>UTF-8</encoding>  </configuration>  </plugin> |

编译插件在maven工程进行编译时加载，按照java1.8使用

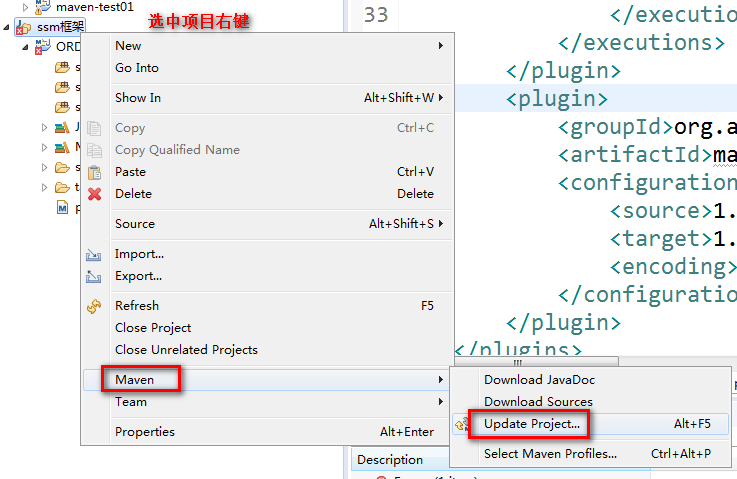
问题：加入插件后项目报错



原因是项目没有更新



利用eclipse插件执行项目更新



## 3.2.spring框架项目测试代码编写

### 3.2.1.引入spring-context依赖资源

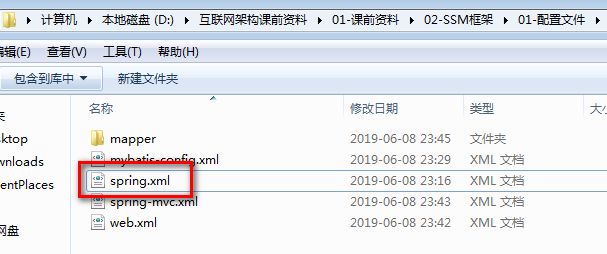
由于依赖的传递性，其他的jar包也会被传递进来，例如spring-beans、spring-aop等

在<dependencies>标签中添加：

|  |
| --- |
| <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  <version>4.3.7.RELEASE</version>  </dependency> |

### 3.2.2.spring配置文件spring.xml（application.xml）

将课前资料中的spring.xml拷贝到src/main/resources中，并清空之前的内容



添加扫描注解路径的配置

|  |
| --- |
| <!-- 扫描当前工程注解的路径cn.tedu -->  <context:component-scan base-package=*"cn.tedu"*></context:component-scan> |

### 3.2.3.编写代码实现类

HelloController(@Controller)

|  |
| --- |
| **package** cn.tedu.controller;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Controller;  @Controller  **public** **class** HelloController {  @Autowired  **private** HelloService helloService;  **public** String sayHi(String name){  **return** helloService.sayHi(name);  }  } |

HelloService(@Service)

|  |
| --- |
| **package** cn.tedu.service;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Service;  @Service  **public** **class** HelloService {  @Autowired  **private** HelloRepository helloRepository;  **public** String sayHi(String name) {  **return** helloRepository.sayHi(name);  }  } |

HelloRepository(@Repository)

|  |
| --- |
| **package** cn.tedu.repository;  **import** org.springframework.stereotype.Repository;  @Repository  **public** **class** HelloRepository {  **public** String sayHi(String name) {  String msg = "hello "+name+" good bye!";  **return** msg;  }  } |

实现了一个简单的业务逻辑，模拟调用业务层，调用持久层访问字符串，返回”hello...”字符串

### 3.2.4.加载配置文件，启动spring容器测试

在src/test/java中创建测试类

|  |
| --- |
| **package** cn.tedu.test;  **import** org.junit.Test;  **import** org.springframework.context.support.ClassPathXmlApplicationContext;  **import** cn.tedu.controller.HelloController;  **public** **class** SpringTest {  @Test  **public** **void** startSpring(){  //读取配置文件  ClassPathXmlApplicationContext context = **new**  ClassPathXmlApplicationContext("classpath:spring.xml");  //利用context读取容器中的controller对象  HelloController controller = context.  getBean(HelloController.**class**);  //调用controller的sayHi方法  String hi = controller.sayHi("孙悟空");  System.***out***.println(hi);  }  } |

## 3.3.mybatis持久层整合

### 3.3.1.配置数据源

Bean标签指向一个类，如：c3p0\druid

首先需要连接池的依赖、数据库依赖、jdbc、mybatis依赖

|  |
| --- |
| <!-- jdbc依赖 -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-jdbc</artifactId>  <version>4.3.7.RELEASE</version>  </dependency>  <!-- mysql依赖 -->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>5.0.8</version>  </dependency>  <!-- 连接池依赖c3p0\druid -->  <dependency>  <groupId>com.mchange</groupId>  <artifactId>c3p0</artifactId>  <version>0.9.5.2</version>  </dependency>  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid</artifactId>  <version>1.0.14</version>  </dependency>  <!-- mybatis依赖 -->  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis</artifactId>  <version>3.4.5</version>  </dependency>  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis-spring</artifactId>  <version>1.3.1</version>  </dependency> |

利用bean标签连接数据源，在spring.xml中配置

|  |
| --- |
| <!-- dataSource -->  <!-- <bean id="dataSource"  class="com.mchange.v2.c3p0.ComboPooledDataSource">  4个属性配置  <property name="driverClass" value="com.mysql.jdbc.Driver"></property>  <property name="jdbcUrl" value="jdbc:mysql:///test"></property>  <property name="user" value="root"></property>  <property name="password" value="root></property>  </bean> -->  <bean id=*"dataSource"*  class=*"com.alibaba.druid.pool.DruidDataSource"*>  <property name=*"driverClassName"* value=*"com.mysql.jdbc.Driver"*></property>  <property name=*"url"* value=*"jdbc:mysql:///test"*></property>  <property name=*"username"* value=*"root"*></property>  <property name=*"password"* value=*"root"*></property>  </bean> |

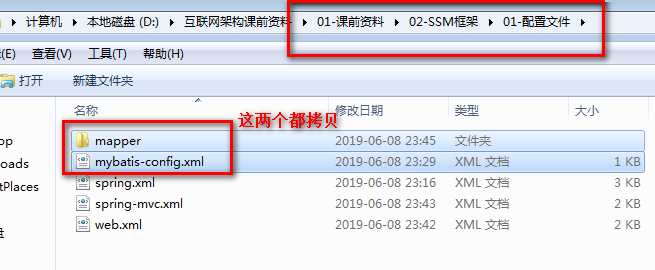
### 3.3.2.将数据源绑定sqlSession

在spring.xml中添加一个bean：

|  |
| --- |
| <!-- 配置sqlSession，动态代理实现持久层接口的抽象方法 -->  <bean id=*"sqlSession"*  class=*"org.mybatis.spring.SqlSessionFactoryBean"*>  <!-- 绑定数据源 -->  <property name=*"dataSource"* ref=*"dataSource"*></property>  <!-- 如果有mybatis独立配置，可以加载 -->  <property name=*"configLocation"* value=*"classpath:mybatis-config.xml"*></property>  <!-- 扫描映射文件，统一放在resources/mapper -->  <property name=*"mapperLocations"* value=*"classpath:mapper/\*.xml"*></property>  </bean> |

配置文件mybatis和映射文件mapper.xml都没有，所有需要加入这些配置文件

可以到课前资料中拷贝，将以下两个文件\文件夹拷贝到src/main/resources中



注意，拷贝过来的mapper.xml中的增删改查标签删除，清空

### 3.3.3.mapper接口类所在的包的扫描配置

在spring.xml中配置一个bean标签

|  |
| --- |
| <!-- mapper接口所在包的扫描 -->  <bean  class=*"org.mybatis.spring.mapper.MapperScannerConfigurer"*>  <property name=*"basePackage"* value=*"cn.tedu.mapper"*></property>  </bean> |

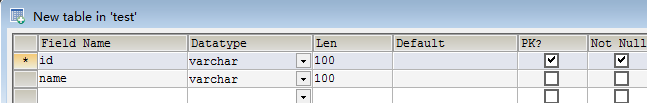
### 3.3.4.查询案例，查询一个student表

可以按照sqlyog操作数据库（傻瓜式安装）

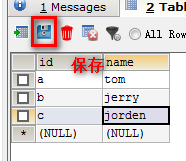
首先在test库中创建一个student表字段：

id varchar

name varchar



添加数据



**编写持久层的顺序**

1.编写一个封装数据库表的实体类entity类（一个类对应一个表）

Student（String id，String name）

|  |
| --- |
| **package** cn.tedu.entity;  **public** **class** Student {  **private** String id;  **private** String name;  Get&set&toString  } |

如果数据库字段类型中有int，long等，我们在封装类中最好用包装类：Integer、Long，这样可以更好的体现业务意义。

2.mapper接口类，按照扫描包的结构

|  |
| --- |
| **package** cn.tedu.mapper;  **public** **interface** StudentMapper {  } |

3.映射的StudentMapper.xml文件

注意xml文件名和namespace要对应接口类的名称

|  |
| --- |
| <?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.tedu.mapper.StudentMapper"*>    </mapper> |

**案例代码的实现：根据id查询student的数据**

**StudentController**

|  |
| --- |
| **package** cn.tedu.controller;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Controller;  **import** cn.tedu.entity.Student;  @Controller  **public** **class** StudentController {  @Autowired  **private** StudentService studentService;  **public** Student queryStudentById(String id){  **return** studentService.queryStudentById(id);  }  } |

**StudentService**

|  |
| --- |
| **package** cn.tedu.service;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Service;  **import** cn.tedu.entity.Student;  **import** cn.tedu.mapper.StudentMapper;  @Service  **public** **class** StudentService {  @Autowired  **private** StudentMapper studentMapper;  **public** Student queryStudentById(String id) {  **return** studentMapper.queryStudentById(id);  }  } |

**StudentMapper**

|  |
| --- |
| **package** cn.tedu.mapper;  **import** cn.tedu.entity.Student;  **public** **interface** StudentMapper {  Student queryStudentById(String id);  } |

**StrudentMapper.xml**

|  |
| --- |
| <select id=*"queryStudentById"* parameterType=*"String"*  resultType=*"cn.tedu.entity.Student"*>  select \* from student where id=#{id};  </select> |

如果实体类中属性名和数据库字段名一致，则可以自动将查询结果封装到实体类中，如果字段名和属性名不一致，解决方法：

1.可以利用sql的别名解决：select name studentName from student;

2.利用resultMap定义返回结果和属性的1对一、1对多的封装

**测试代码，启动spring容器**

|  |
| --- |
| @Test  **public** **void** queryStudent(){  //读取配置文件  ClassPathXmlApplicationContext context = **new**  ClassPathXmlApplicationContext("classpath:spring.xml");  //利用context读取容器中的controller对象  StudentController controller = context.getBean(StudentController.**class**);  //调用controller的查询方法  Student student = controller.queryStudentById("a");  System.***out***.println(student);  } |

注意：在写bean标签中，property标签中的value内容必须写到value属性中，不能写到标签体中

三层结构调用是ssm的基础调用结构

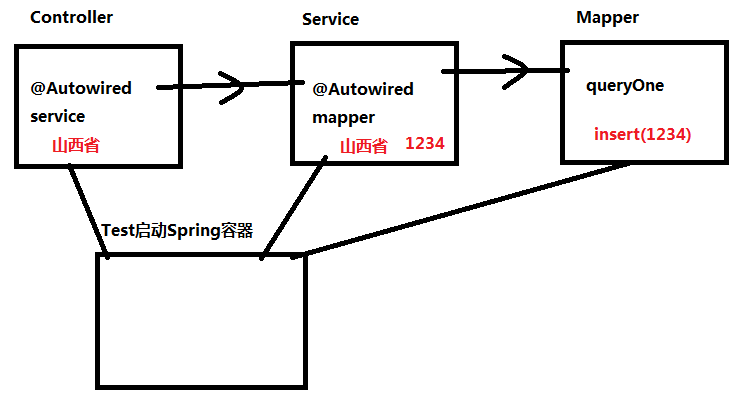
都是通过controller调用service，再调用mapper最终访问数据库，返回结果

问题：页面中填写的数据需要增加到数据库，页面结构和数据库表结构一定一样吗？

不一定：1.有可能内容会存入多张表

2.有可能内容会经过整理后存储，和整理之前完全不同

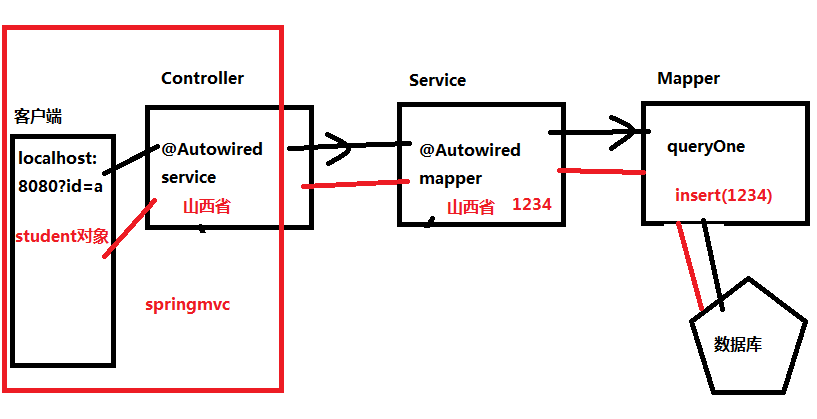
假设存储地址时，山西省太原市杏花村，但是在数据库存时，可以通过service中的逻辑，存入-省：1023（存的是编号）



## 3.4.springmvc的整合（最终由tomcat容器启动所有内容）

引入maven的tomcat插件（也可以将项目打成war包，扔到tomcat中启动）

实现需求：通过浏览器传递一个get请求，请求参数：id=a，项目返回从数据库student包中查询的student对象，显示到浏览器



### 3.4.1.pom.xml中引入springmvc相关依赖

Spring-web\spring-webmvc\对象转成json的辅助依赖jackson

辅助依赖jackson作用：springmvc框架可以根据视图解析器进行拼接，例如在controller中：

//返回的是一个字符串，通过视图解析器，将字符串拼接为：/WEB-INF/student.jsp

**return** "student";

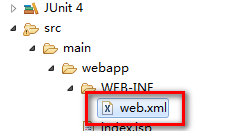
可以通过@ResponseBody：springmvc注解，实现解析数据时，将controller返回的所有内容放到响应体中，如果返回的是Student对象，配合jackson依赖，可以将Student对象转换为json格式放入响应体中响应到页面

Pom.xml中依赖：

|  |
| --- |
| <!-- springmvc相关依赖 -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-web</artifactId>  <version>4.3.7.RELEASE</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</artifactId>  <version>4.3.7.RELEASE</version>  </dependency>  <!-- 对象转json的辅助依赖jackson -->  <dependency>  <groupId>com.fasterxml.jackson.core</groupId>  <artifactId>jackson-core</artifactId>  <version>2.8.8</version>  </dependency>  <dependency>  <groupId>com.fasterxml.jackson.core</groupId>  <artifactId>jackson-databind</artifactId>  <version>2.8.8</version>  </dependency> |

### 3.4.2.tomcat启动的web.xml

在web.xml中配置dispacherservlet，扫描spring\*.xml文件，tomcat容器启动加载spring所有的配置内容，从而使得所有注解生效

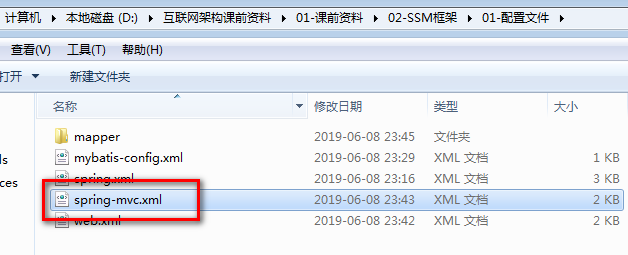


在web.xml中配置：

|  |
| --- |
| <servlet>  <servlet-name>springmvc</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <!-- tomcat的配置，监听需要加载的容器配置文件spring\*.xml -->  <init-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:spring\*.xml</param-value>  </init-param>  </servlet>  <!-- servlet映射 -->  <servlet-mapping>  <servlet-name>springmvc</servlet-name>  <url-pattern>/\*</url-pattern>  </servlet-mapping> |

### 3.4.3.springmvc的xml配置文件

从课前资料中拷贝到resources



其他的可以清空

加入开启注解的配置

|  |
| --- |
| <!-- 开启注解生效的配置 -->  <mvc:annotation-driven/> |

### 3.4.4.业务实现代码

修改Controller，在Controller中只需添加两个注解即可

|  |
| --- |
| **package** cn.tedu.controller;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.ResponseBody;  **import** cn.tedu.entity.Student;  **import** cn.tedu.service.StudentService;  @Controller  **public** **class** StudentController {  @Autowired  **private** StudentService studentService;  @RequestMapping("queryStudent")  @ResponseBody//此注解将return的内容放入响应体而非视图解析器  //由于添加了jackson，所以返回的student对象会转换为json格式  **public** Student queryStudentById(String id){  **return** studentService.queryStudentById(id);  //返回的是一个字符串，通过视图解析器，将字符串拼接为：WEB-INF/student.jsp  //return "student";  }  } |

### 3.4.5.maven的tomcat插件

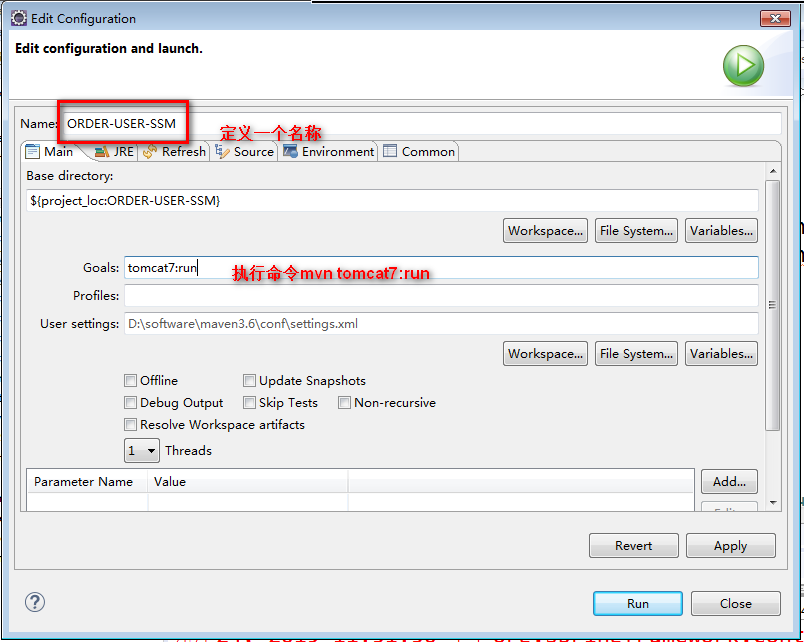
我们可以利用外部的tomcat启动整个项目，不方便，需要打war包，可以引入maven的tomcat插件，一个项目专门有一个tomcat的进程进行维护，如果多个项目中使用多个tomcat，注意端口号不能冲突

在pom.xml中添加tomcat插件

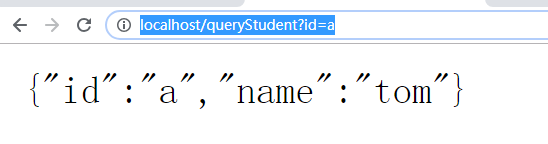
|  |
| --- |
| <plugin>  <groupId>org.apache.tomcat.maven</groupId>  <artifactId>tomcat7-maven-plugin</artifactId>  <version>2.2</version>  <!-- tomcat启动的配置 -->  <configuration>  <!-- 端口号 -->  <port>80</port>  <!-- 应用程序访问的路径 -->  <path>/</path>  <!-- 编码 -->  <uriEncoding>utf-8</uriEncoding>  <useBodyEncodingForURI>utf-8</useBodyEncodingForURI>  </configuration>  </plugin> |

启动当前工程，需要运行tomcat插件命令：mvn tomcat7:run

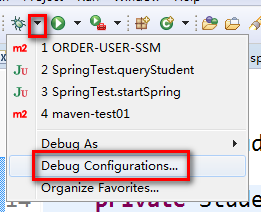
在runas->maven build中创建一个运行的命令goals:tomcat7:run，然后点击run运行

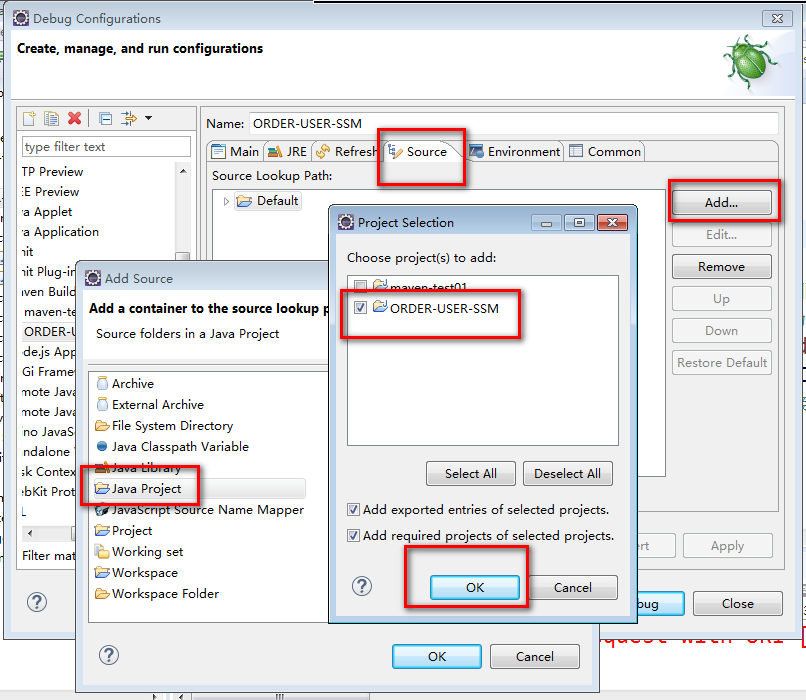


运行后可以在浏览器访问http://localhost/queryStudent?id=a



如果需要debug，可以





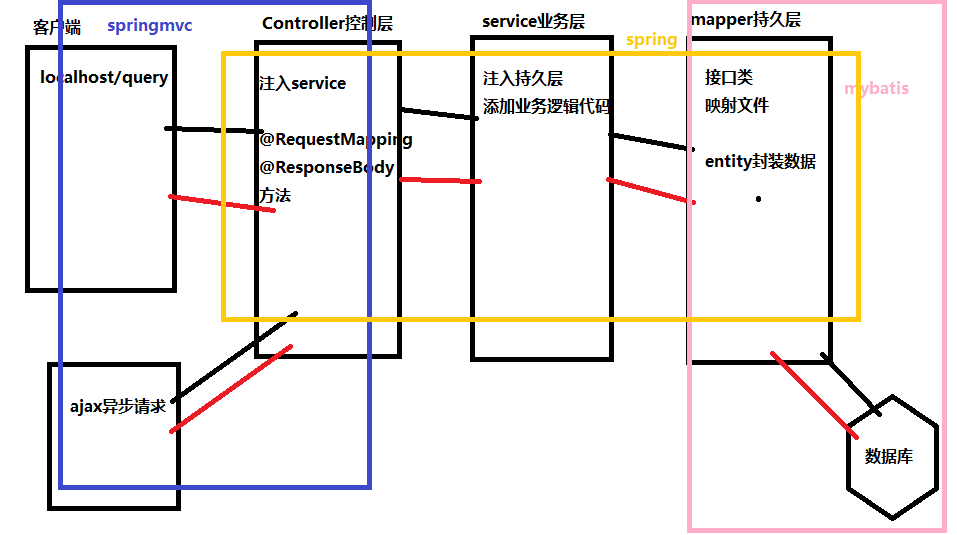
json（javascript object notation）js对象简谱：一种轻量级的文本格式，进行数据的传输，web应用中json进行后台和前端的数据交互，web前端html+ajax的js代码实现请求,实现前后端分离，使用json字符串，可以在ajax回调函数data中直接使用，相当于就是js对象

依赖的jackson资源，在springmvc数据解析是，一旦扫描到@ResponseBody注解，不再进行页面的寻找，而是直接调用jackson的代码将返回的数据解析成json字符串放入响应体中响应

一个Demo测试对象转换json过程

|  |
| --- |
| **package** cn.tedu.test;  **import** org.junit.Test;  **import** com.fasterxml.jackson.core.JsonProcessingException;  **import** com.fasterxml.jackson.databind.ObjectMapper;  **import** cn.tedu.entity.Student;  **public** **class** JacksonTest {  @Test  **public** **void** jsonTrans() **throws** JsonProcessingException{  Student stu = **new** Student();  stu.setId("e");  stu.setName("王麻子");  ObjectMapper mapper = **new** ObjectMapper();  String json = mapper.writeValueAsString(stu);  System.***out***.println(stu);  System.***out***.println(json);  }  } |

**ssm框架结构调用图**



Springmvc：前端交互

Mybatis：和数据库交互

Spring贯穿整体

浏览器发起请求：一般需要返回页面的响应数据，html格式的文本

Ajax发起请求：异步请求，一般需要返回一个json字符串

## 3.5.其他配置的添加

### 3.5.1.springmvc.xml中配置的视图解析器

|  |
| --- |
| <bean id=*"viewResolver"*  class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>  <property name=*"prefix"* value=*"/WEB-INF/views/"*></property>  <property name=*"suffix"* value=*".jsp"*></property>  </bean> |

### 3.5.2.解决springmvc返回数据的编码问题

|  |
| --- |
| <!-- springmvc返回数据编码问题 -->  <bean class=*"org.springframework.http.converter.StringHttpMessageConverter"*>  <property name=*"supportedMediaTypes"* value=*"text/html;charset=utf-8"*></property>  </bean> |

### 3.5.3.mybatis-config.xml

驼峰命名

|  |
| --- |
| <!-- 开启驼峰命名 -->  <settings>  <setting name=*"mapUnderscoreToCamelCase"* value=*"true"*/>  </settings> |

解决对象字段与数据库字段的封装格式

例如数据库中字段为：order\_pay\_product\_price

封装类中一般变量名的写法：orderPayProductPrice

### 3.5.4.别名包设置spring.xml

|  |
| --- |
| <bean id=*"sqlSession"*  class=*"org.mybatis.spring.SqlSessionFactoryBean"*>  <!-- 绑定数据源 -->  <property name=*"dataSource"* ref=*"dataSource"*></property>  <!-- 如果有mybatis独立配置，可以加载 -->  <property name=*"configLocation"* value=*"classpath:mybatis-config.xml"*></property>  <!-- 扫描映射文件，统一放在resources/mapper -->  <property name=*"mapperLocations"* value=*"classpath:mapper/\*.xml"*></property>  <!-- 别名包 -->  <property name=*"typeAliasesPackage"* value=*"cn.tedu.entity"*></property>  </bean> |

别名包设置后，在mapper.xml的映射文件中，resultType="cn.tedu.entity.Student"可以简写为resultType="Student"

### 3.5.5.应用中静态资源springmvc访问

不是所有的请求需要经过springmvc的dispatcherservlet，比如访问html

在springmvc.xml中：

|  |
| --- |
| <mvc:resources location=*"/"* mapping=*"/\*\*"*></mvc:resources> |

## 3.6.用户支付积分案例

### 3.6.1.需求

完成一个ssm框架搭建的具有页面效果的系统，没有代码框架实现的页面跳转

实现两个功能：

1.查询用户的积分，这里用户的id写死了

2.点击支付按钮，完成订单的支付，并且根据支付金额和积分逻辑，对用户积分进行增加

### 3.6.2.准备数据库数据

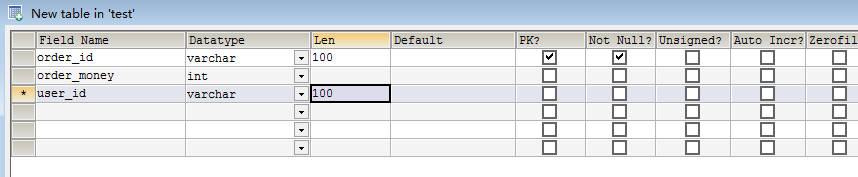
在test数据库中准备两个表格：t\_order\t\_user

**t\_order表格**

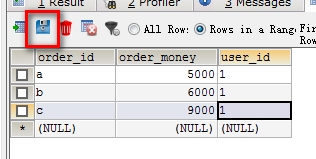
三个字段：order\_id varchar 主键

order\_money int 订单金额

user\_id varchar 用户id



添加数据：



**t\_user表**

字段：

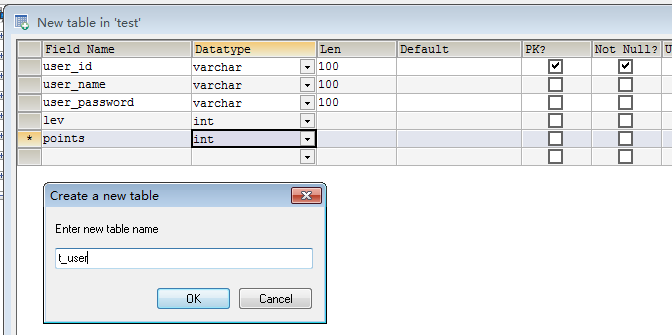
user\_id varchar 主键

user\_name varchar 用户名

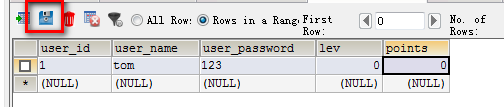
user\_password varchar 密码

lev int 用户等级，0表示普通用户，1表示银牌用户，2表示金牌用户

points int 用户的积分

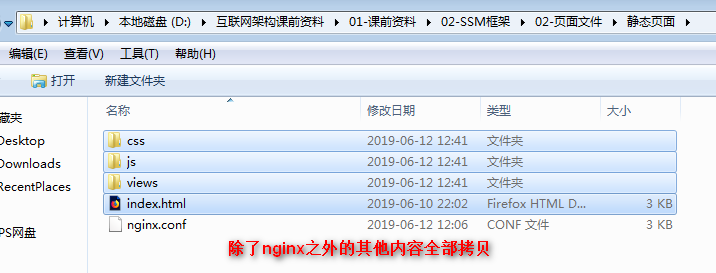


准备数据

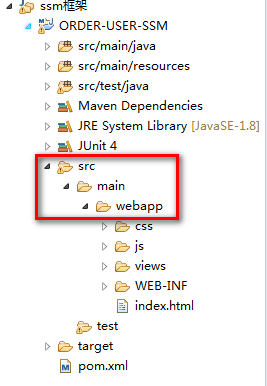


### 3.6.3.准备页面静态文件

课前资料中直接拷贝



拷贝到ORDER-USER-SSM项目下的webapp目录中



为了防止dispatcherservlet拦截静态资源访问，在spring-mvc.xml中添加配置文件：

### 3.6.4.开发功能+分析前端页面

分析前端页面，一般后端开发工程师不做的

**用户积分查询页面分析**

分析views下的point.html页面ajax代码

$(**function**() //相当于window.onload()，当页面加载完成时调用次函数

{

$.ajax({

type: "GET",//请求方式为get请求

url: '/user/query/point?userId=1',//请求地址

dataType:"json",//返回的数据类型

success: **function**(data) {//表示请求成功的时候，调用function回调函数，参数data就表示响应体的数据，例如响应体为data={“points”:”5000”}

// 将请求的数据放输出到div里面

$('#div02').html(data.points+"分");

}

});

});

ajax回忆：页面前端异步请求技术，做到页面的局部刷新、局部数据获取等

原生ajax是js代码，分为4步

1.获取ajax请求对象

2.打开请求通道open

3.传递参数，发起请求

4.接收返回值，判断状态码

jQuery封装的ajax：

$.get()发起get请求

$.post()发起post请求

$.ajax()可以配置请求方式

页面分析结果：

请求地址：/user/query/point

请求参数：String userId

请求方式：get

返回数据：对象的json字符串，至少包含一个points属性

在后台开发时，可以根据这个分析结果开发后台代码，无需了解前端页面逻辑

**用户积分查询代码实现**

UserController\UserService\UserMapper接口\entity user\UserMapper.xml映射文件

**User**

|  |
| --- |
| **package** cn.tedu.entity;  **public** **class** User {  **private** String userId;  **private** String userName;  **private** String userPassword;  **private** Integer lev;  **private** Integer points;  get&set  } |

**UserController**

|  |
| --- |
| **package** cn.tedu.controller;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.ResponseBody;  @Controller  **public** **class** UserController {  /\*请求地址：/user/query/point  请求参数：String userId  请求方式：get  返回数据：对象的json字符串，至少包含一个points属性\*/  @Autowired  **private** UserService userService;  @RequestMapping("/user/query/point")  @ResponseBody  **public** User queryUserPoint(String userId){  **return** userService.queryUserPoint(userId);  }  } |

UserService

|  |
| --- |
| **package** cn.tedu.service;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Service;  **import** cn.tedu.entity.User;  @Service  **public** **class** UserService {  @Autowired  **private** UserMapper userMapper;  **public** User queryUserPoint(String userId) {  **return** userMapper.queryUser(userId);  }  } |

UserMapper接口

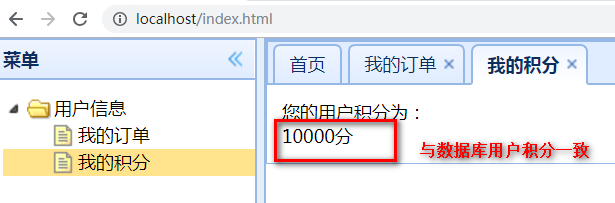
|  |
| --- |
| **package** cn.tedu.mapper;  **import** cn.tedu.entity.User;  **public** **interface** UserMapper {  User queryUser(String userId);  } |

UserMapper.xml（拷贝其他的映射文件，修改文件名、namespace、清空内容）

|  |
| --- |
| <mapper namespace=*"cn.tedu.mapper.UserMapper"*>  <select id=*"queryUser"* resultType=*"User"* parameterType=*"String"*>  select \* from t\_user where user\_id=#{userId}  </select>  </mapper> |

最后启动tomcat插件，可以访问localhost/user/query/point?userId=1,查看是否有数据返回（未整合页面情况下）

整合页面后，直接访问localhost/index.html



### 3.6.5.接口文件

**什么是接口文件**

大型团队中，系统与系统，人与人，团队与团队，部门与部门，甚至是不同公司间配合工作，沟通交流需要极大的成本维护，为了协调工作之间的信息沟通，需要一种规范文档，这个文档就是接口文件，信息交互规范都在这个文件中定义

**意义：**

1.高效的沟通

2.责任划分

**前后端交互的接口文件**

功能：用户积分查询

|  |  |
| --- | --- |
| 请求地址 | localhost/user/query/point?userId=1 |
| 后台接收地址 | /user/query/point |
| 请求方式 | get |
| 请求参数 | String userId |
| 返回数据 | 对象的json字符串，至少包含一个points属性 |
| 备注 | select \* from t\_user where user\_id=#{userId} |

功能：支付并积分

|  |  |
| --- | --- |
| 请求地址 | localhost/order/pay?orderId=a |
| 后台接收地址 | /order/pay |
| 请求方式 | get |
| 请求参数 | String orderId 表示订单id |
| 返回数据 | “1”表示成功，其他表示失败 |

**利用接口文件开发订单支付与积分业务**

OrderController、OrderService、OrderMapper接口、OrderMapper.xml映射文件、Order实体类

Order

|  |
| --- |
| **package** cn.tedu.entity;  **public** **class** Order {  **private** String orderId;  **private** Integer orderMoney;  **private** String userId;  get&set  } |

OrderController

|  |
| --- |
| **package** cn.tedu.controller;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.ResponseBody;  @Controller  **public** **class** OrderController {  @Autowired  **private** OrderService orderService;  @RequestMapping("/order/pay")  @ResponseBody  **public** Integer orderPay(String orderId){  **try** {  orderService.orderPay(orderId);  **return** 1;  } **catch** (Exception e) {//有异常表示支付失败  e.printStackTrace();  **return** 0;  }  }  } |

OrderService

|  |
| --- |
| **package** cn.tedu.service;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Service;  **import** cn.tedu.entity.Order;  **import** cn.tedu.mapper.UserMapper;  @Service  **public** **class** OrderService {  @Autowired  **private** OrderMapper orderMapper;  @Autowired  **private** UserMapper userMapper;  **public** **void** orderPay(String orderId) {  System.***out***.println("支付成功");  //通过orderId获取order对象，并获取orderMoney  Order order = orderMapper.queryOrder(orderId);  Integer orderMoney = order.getOrderMoney();  //根据orderMoney修改user表的积分,money:point=1:1  userMapper.updateUserPoint(orderMoney);  }  } |

OrderMapper

|  |
| --- |
| **package** cn.tedu.mapper;  **import** cn.tedu.entity.Order;  **public** **interface** OrderMapper {  Order queryOrder(String orderId);  } |

UserMapper加入一行

|  |
| --- |
| **void** updateUserPoint(Integer orderMoney); |

OrderMapper.xml（复制其他修改）

|  |
| --- |
| <mapper namespace=*"cn.tedu.mapper.OrderMapper"*>  <select id=*"queryOrder"* resultType=*"Order"* parameterType=*"String"*>  select \* from t\_order where order\_id=#{orderId}  </select>  </mapper> |

UserMapper.xml

|  |
| --- |
| <update id=*"updateUserPoint"* parameterType=*"int"*>  update t\_user set points=points+#{orderMoney} where user\_id=1;  </update> |

测试：localhost/index.html

## 3.7.单体项目总结

### 3.7.1.什么是单体系统

**系统：**就是用户可以访问使用的应用系统，例如：淘宝、京东等

**单体系统：**一个系统的所有功能（用户相关功能，订单相关功能，购物车相关功能等）都只启动加载到一个web应用中，这种系统就是单体系统，例如ORDER-USER-SSM就是单体系统

### 3.7.2.单体系统的问题

**高并发问题**

并发的瓶颈在tomcat的技术（tomcat并发在200-500之间，cpu性能很高，可以调教到2000左右），想办法搭建tomcat集群，多个tomcat可能ip、端口不同，如何访问

**静态页面资源没有分离**

目前静态资源也是在tomcat中维护，浪费tomcat处理动态数据的性能，想办法将静态资源分离出来