## 需求场景

基于Spring MVC实现Restful接口的Web层，工作需要的场景CodeServer直接面向用户登录场景。

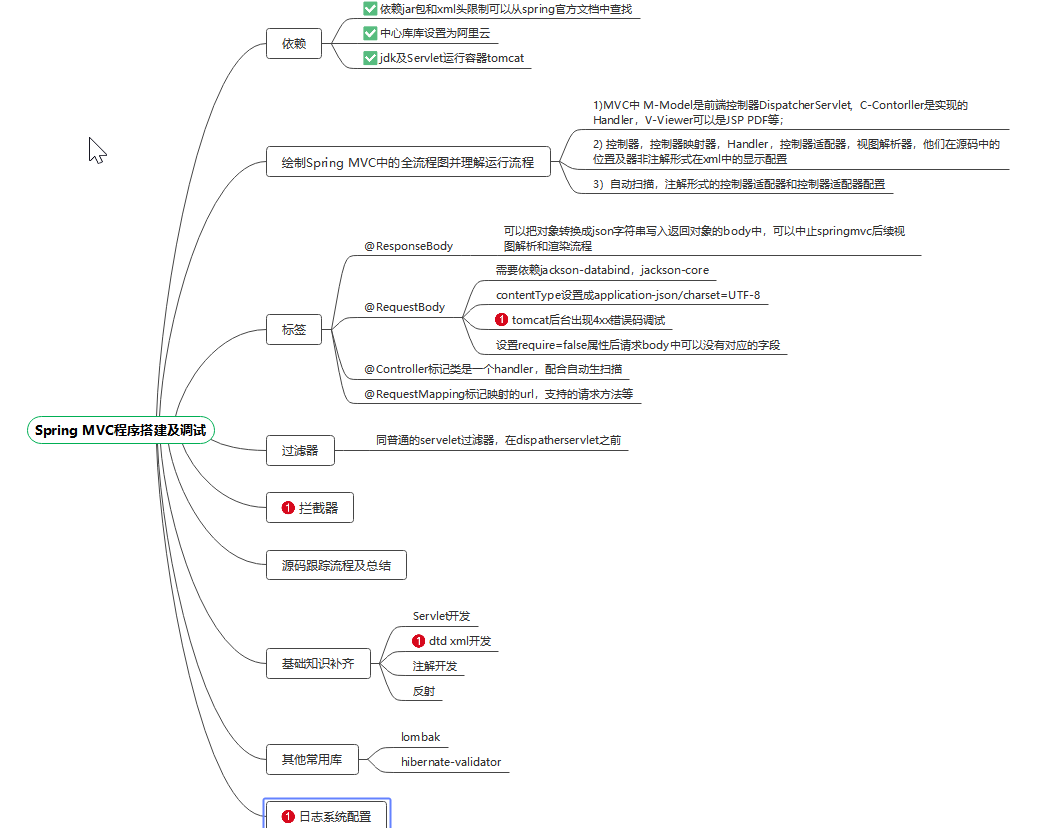
搭建一个微服务可以通过接口调用实现jsoup网页扫描功能。

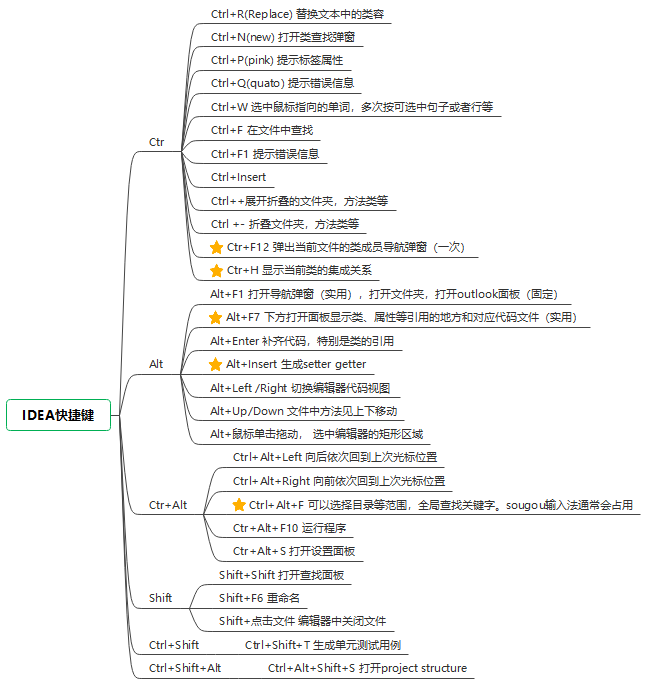
http://localhost:8080/jsoup/getherf

http://47.98.193.71:8080/ServletForTomcat/jsoup/getherf

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 说明 | 参数名 | 类型 | 是否必须 | 样例 |
| 输入参数 | 页面url | sourcePageID | Int | 是 | {"sourcePageID":xx} |
| 输出参数 | 页面hash码 | sourcePageID | Int | 是 | Herf信息列表，每页做多1000条  [{   "sourcePageID": 1,   "pictureHerf": "xx",  "relativePageHerf": "xx"  },...] |
| 图片url | pictureHerf | String | 是 |
| 页面url | relativePageHerf | String | 是 |

## 任务拆解





## 依赖包管理

|  |  |  |
| --- | --- | --- |
| 框架 | 依赖 | 说明 |
| Springmvc | <dependency> <groupId>org.springframework</groupId> <artifactId>spring-webmvc</artifactId> <version>5.1.12.RELEASE</version> </dependency> </dependencies> | 引用全量的包么？ |
| jackson | <dependency> <groupId>com.fasterxml.jackson.core</groupId>  <artifactId>jackson-core</artifactId>  <version>2.11.4</version> </dependency> <dependency>  <groupId>com.fasterxml.jackson.core</groupId>  <artifactId>jackson-databind</artifactId>  <version>2.11.4</version> </dependency> | 序列化使用 |
| J2EE | <dependency>  <groupId>javax.servlet</groupId>  <artifactId>javax.servlet-api</artifactId>  <version>3.1.0</version> </dependency> |  |
| JUnit | <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.11</version>  <scope>test</scope> </dependency> |  |
| Mybatis支持包 | <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis</artifactId>  <version>3.4.5</version>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>8.0.12</version>  </dependency>  <dependency>  <groupId>ch.qos.logback</groupId>  <artifactId>logback-classic</artifactId>  <version>1.2.3</version>  </dependency>  <!-- mybatis与spring整合的jar包-->  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis-spring</artifactId>  <version>2.0.5</version>  </dependency>  <!--spring管理的 jdbc ,以及事务相关的-->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-jdbc</artifactId>  <version>5.2.12.RELEASE</version>  </dependency>  <!-- 数据库连接池 alibaba 的 druid -->  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid</artifactId>  <version>1.1.7</version>  </dependency>  <!-- 日志桥接 -->  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>jcl-over-slf4j</artifactId>  <version>1.7.25</version>  </dependency> |  |

项目工程目录

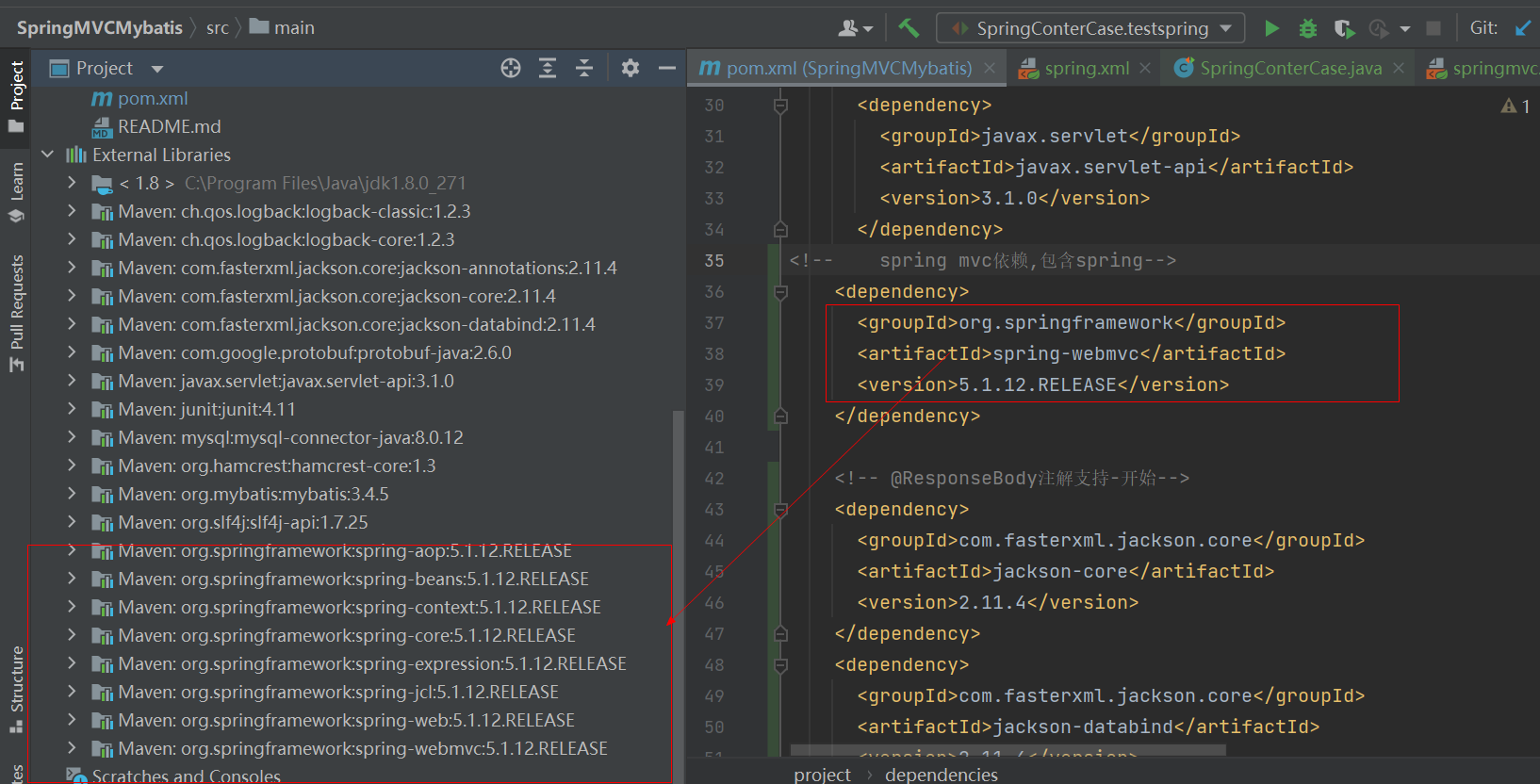
D:\workshop\javaservice\SpringMVCMybatis

【问题】Maven会加载间接依赖？

很多时候，我们引用的第三方jar需要一些其他的第三方jar，这个时候默认情况下，间接需要依赖的第三方jar是不会自动被引入的，如果希望这些额外的三方jar被自动引入，则在Maven仓库中除了提交jar文件外，还需要把maven install创建的jar名.pom文件一并提交到maven仓库，这样第三方的jar就会被主工程自动加载，而不需要开发人员自己去寻找依赖的三方间接jar。

PS:

在POM 4中，<dependency>中还引入了<scope>，它主要管理依赖的部署。目前<scope>可以使用5个值：   
    \* compile，缺省值，适用于所有阶段，会随着项目一起发布。   
    \* provided，类似compile，期望JDK、容器或使用者会提供这个依赖。如servlet.jar。   
    \* runtime，只在运行时使用，如JDBC驱动，适用运行和测试阶段。   
    \* test，只在测试时使用，用于编译和运行测试代码。不会随项目发布。   
    \* system，类似provided，需要显式提供包含依赖的jar，Maven不会在Repository中查找它。



## Tomcat日志

|  |  |
| --- | --- |
| 类型 | 说明 |
| Localhost Access log | **访问日志**localhost\_access\_log  该日志也非常重要，在分析很多应用的流量，带宽需求，及负载时非常有用。该日志是在server.xml中配置的。  该日志格式可以自己定义： <Valve className=“org.apache.catalina.valves.AccessLogValve” directory=“logs”prefix=“localhost\_access\_log.” suffix=“.txt” pattern=“%h %l %u %t “%r” %s %b” resolveHosts=“false”/> |
| catalina.2021-08-15.log | **Cataline引擎的日志**文件，文件名catalina.日期.log  在排查服务器启动、应用的部署错时比较有用。 配置日志是在catalina.sh中配置的。 |

配置文件

D:\workshop\apache-tomcat-8.5.69\conf\logging.properties

【问题】tomcat中各种日志文件在哪里配置？

## Spring MVC集成（web层）

### 配置

Web.xml

|  |
| --- |
| <!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>Jsoup crawler</display-name>  <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:springmvc.xml,classpath:applicationContext.xml</param-value>  </init-param>  </servlet>  <servlet-mapping>  <servlet-name>springmvc</servlet-name>  <!--  方式一：带后缀，比如\*.action \*.do \*.aaa 该种方式比较精确、方便，在以前和现在企业中都有很大的使用比例  方式二：/ 不会拦截 .jsp，但是会拦截.html等静态资源（静态资源：除了servlet和jsp之外的js、css、png等）  为什么配置为/ 会拦截静态资源？？？因为tomcat容器中有一个web.xml（父），你的项目中也有一个web.xml（子），  是一个继承关系父web.xml中有一个DefaultServlet, url-pattern 是一个 /此时我们自己的web.xml中  也配置了一个 / ,覆写了父web.xml的配置 为什么不拦截.jsp呢？  因为父web.xml中有一个JspServlet，这个servlet拦截.jsp文件，而我们并没有覆写这个配置，  所以springmvc此时不拦截jsp，jsp的处理交给了tomcat  如何解决/拦截静态资源这件事？  方式三：/\* 拦截所有，包括.jsp  -->  <!--拦截匹配规则的url请求，进入springmvc框架处理-->  <url-pattern>/</url-pattern>  </servlet-mapping>  </web-app> |

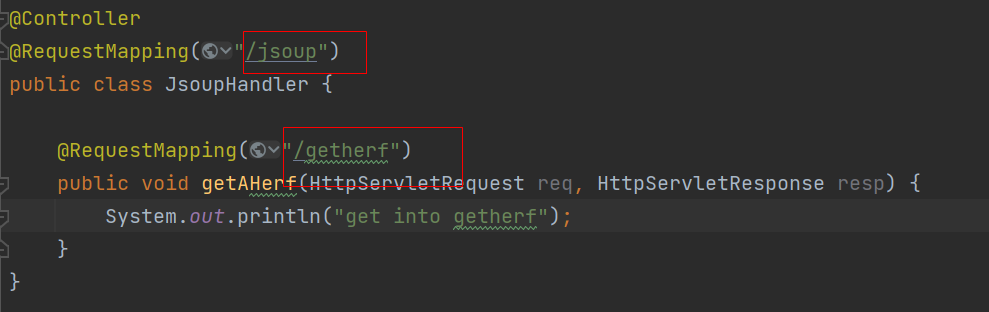
Springmvc.xml

|  |
| --- |
| <?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"  xmlns:mvc="http://www.springframework.org/schema/mvc"  xsi:schemaLocation="  http://www.springframework.org/schema/beans  https://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/context  https://www.springframework.org/schema/context/spring-context.xsd  http://www.springframework.org/schema/mvc  https://www.springframework.org/schema/mvc/spring-mvc.xsd">  <!--开启自动扫描-->  <context:component-scan base-package="tech.pariote.controller"/>  <!--  自动注册最合适的处理器映射器，处理器适配器(调用handler方法)  -->  <mvc:annotation-driven/>  <!-- @ResponseBody注解支持-开始 -->  <bean id="mappingJacksonHttpMessageConverter"  class="org.springframework.http.converter.json.MappingJackson2HttpMessageConverter">  <property name="supportedMediaTypes">  <list>  <value>application/json;charset=UTF-8</value>  </list>  </property>  </bean>  <!-- @ResponseBody注解支持-结束 -->  </beans> |

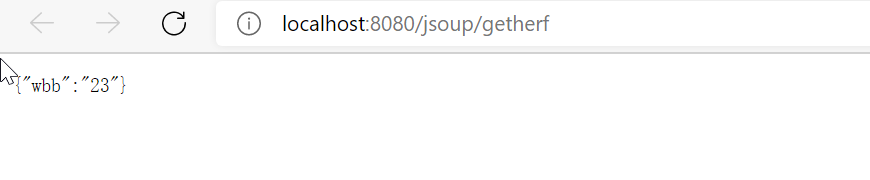
### 测试

测试是[localhost:8080/jsoup/getherf](http://localhost:8080/jsoup/getherf)，因为配置如下，

* web.xml中配置/
* 本地tomcat Application contex配置空
* controller中配置



浏览器测试get请求 【经验】调试的使用使用get+自启动浏览器效率更高



### 问题

【问题】解决@ResponseBody使用问题，经验：使用工程项目验证可以运行的版本



总结：

* @ResponseBody是讲对象序列化才成json字符串后返回到http响应的body中。
* 依赖jackson相关包，注意springmvc和jackson对应的版本号。
* @ResponseBody修饰带有@RequestMapping标签的类，类中方法返回pojo要有setter和getter
* 返回json传中key是pojo的成员名，value是其对应的值。

参考文献：

Article:Spring MVC源码(三) ----- @RequestBody和@ResponseBody原理解析

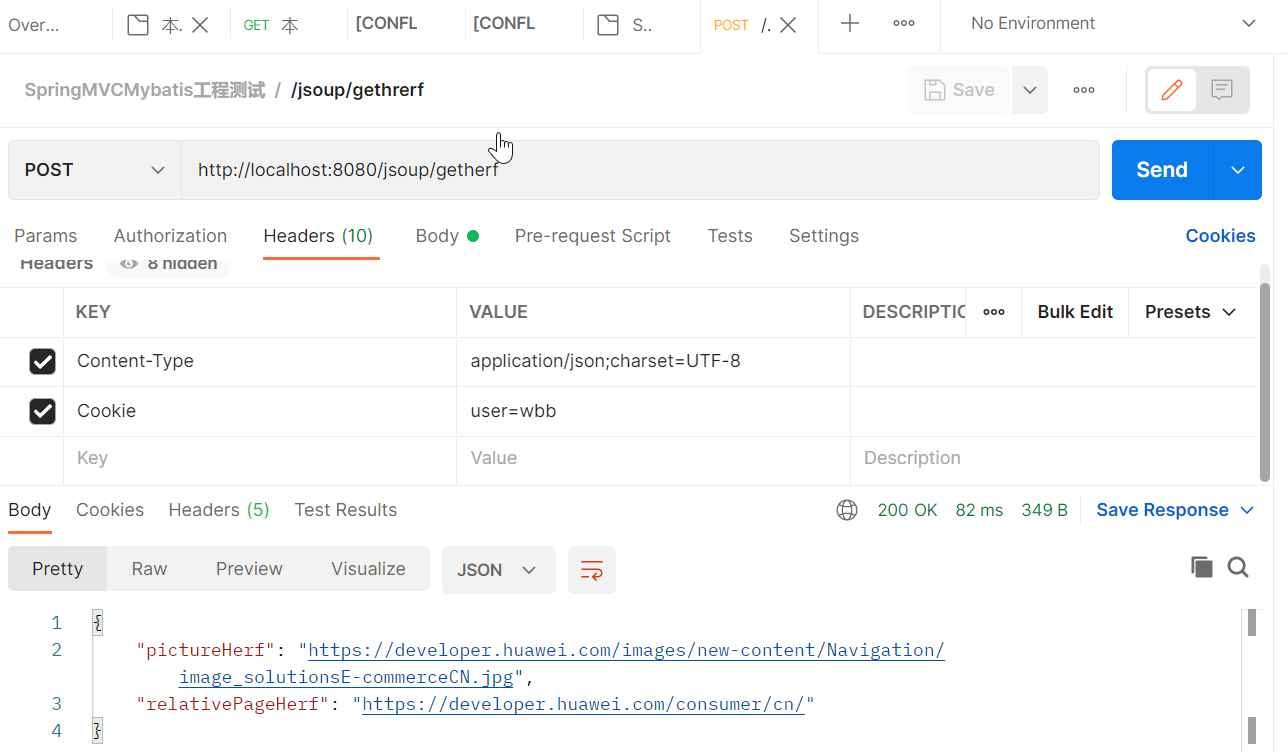
<https://www.cnblogs.com/java-chen-hao/p/11187914.html>

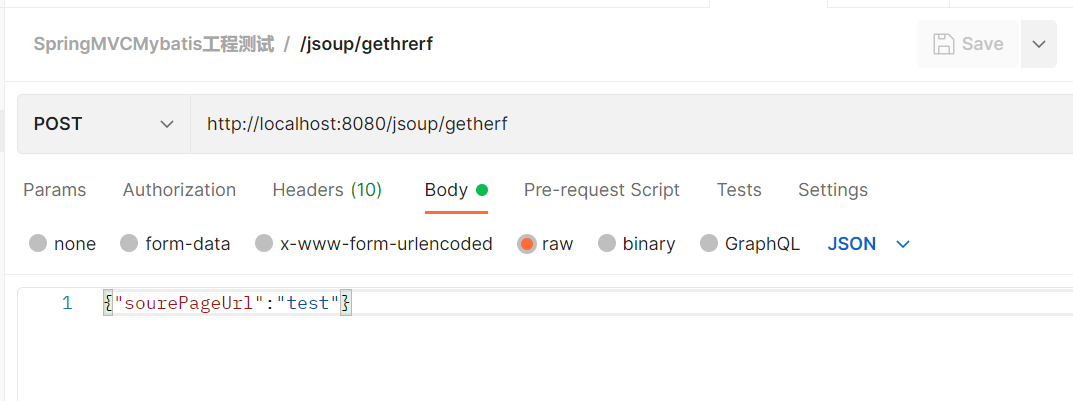
<https://www.cnblogs.com/lvbinbin2yujie/p/10513530.html>

https://www.cnblogs.com/liuyj-top/p/13137628.html

【经验】测试@RequestBody使用，url搞错了调试方法先打断点看程序是否进入

【总结】Postman中Headers设置Content-Type，Body中选择json并填入json数据。注意json中key需要时映射pojo类型的成员名。





## Spring集成（业务层）

Spring的基本配置和Spring与Mybatis的整合

↑ <https://blog.csdn.net/qq_42371269/article/details/83505269>

Pom依赖，引入如下依赖后会自动引入其他包

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.1.12.RELEASE</version>

</dependency>

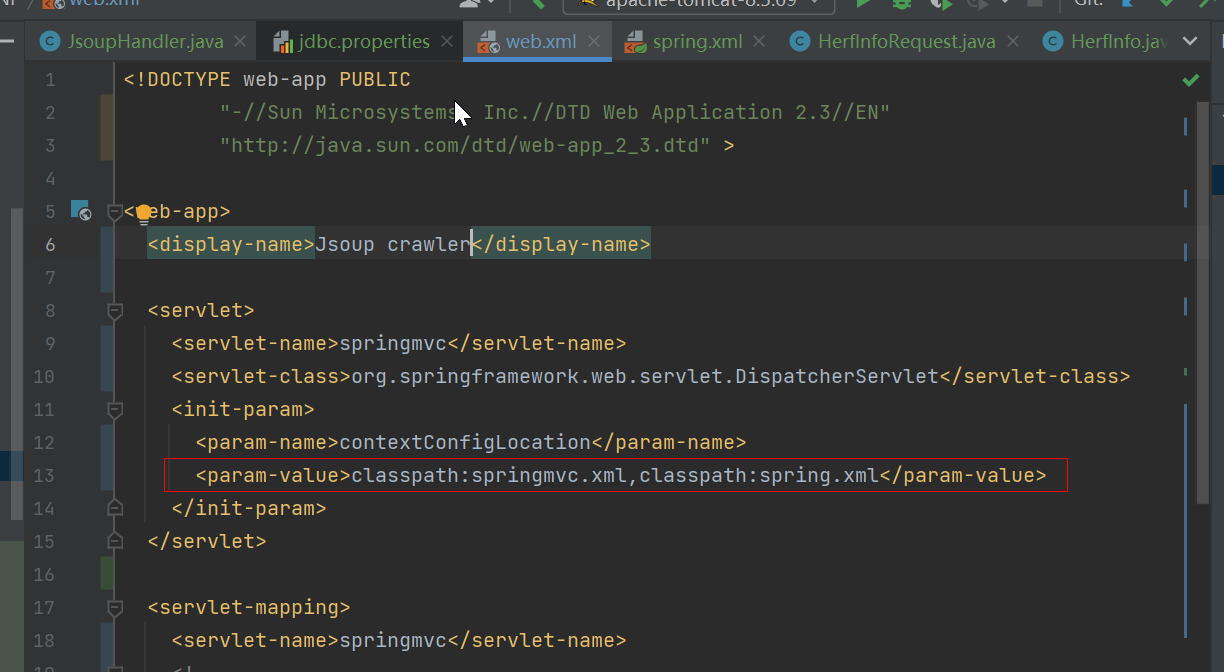
applicationContext.xml配置

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns:context="http://www.springframework.org/schema/context"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context.xsd">  <bean id="getHerfRequest" class="tech.pariote.bean.GetHerfRequest">  <constructor-arg index="0" value="sourcepage"/>  </bean>  </beans> |

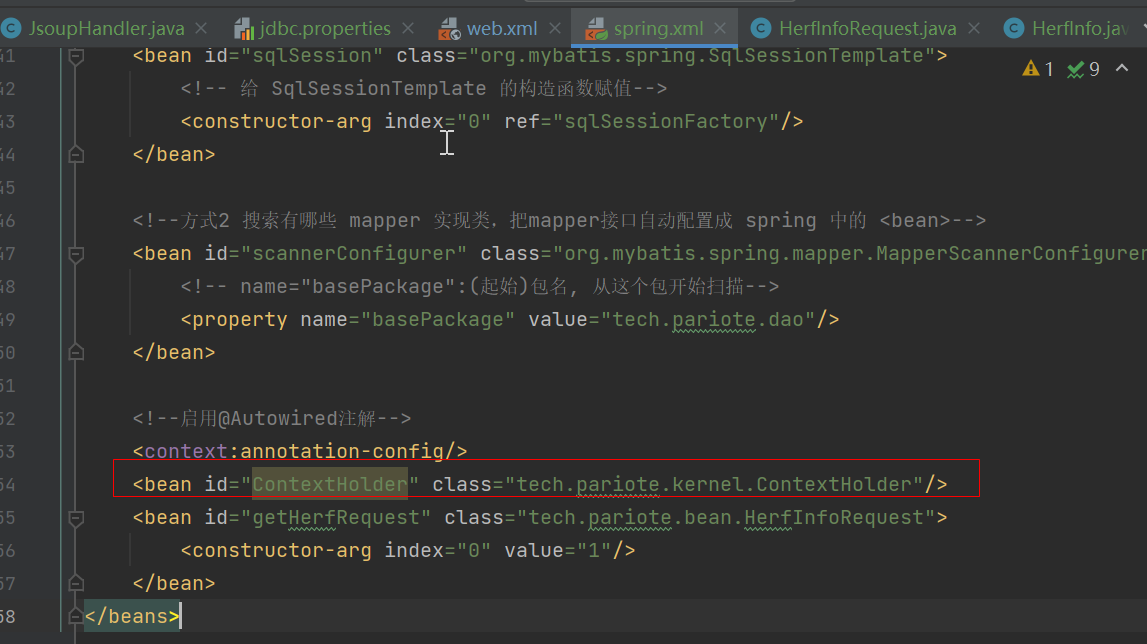
测试是否配置成功

|  |
| --- |
| package tech.pariote;  import org.junit.Test;  import org.springframework.context.support.ClassPathXmlApplicationContext;  import tech.pariote.bean.GetHerfRequest;  public class SpringConterCase {  static ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("spring.xml");  @Test  public void testspring() {  GetHerfRequest herfRequest = (GetHerfRequest) context.getBean("getHerfRequest");  System.out.println(herfRequest.getSourePageUrl());  }  } |

【功能1】实现一个上下文管理器 ContextHolder



实例化bean



实现了ApplicationContextAware接口的方法，spring容器加载过程中会调用该方法。



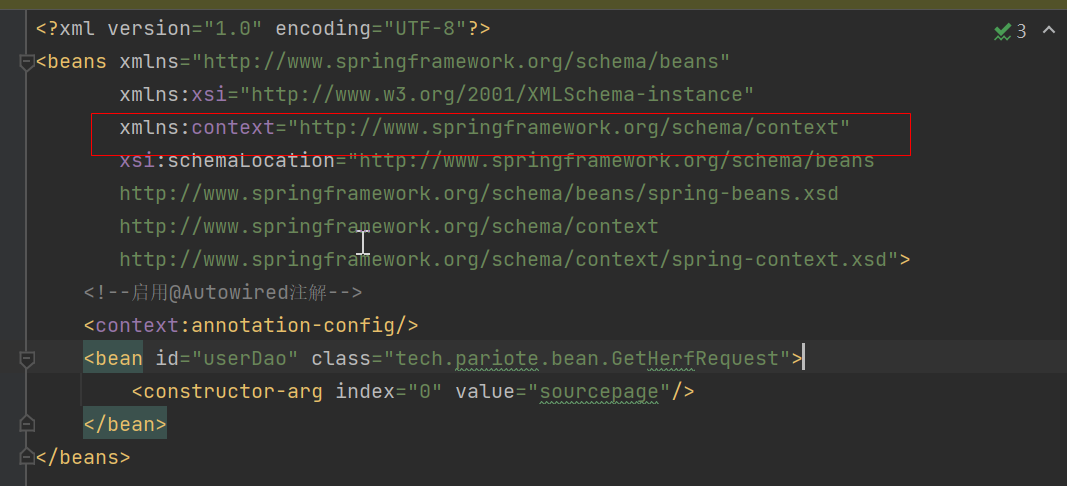
参考链接

web.xml中contextConfigLocation的作用  
https://blog.csdn.net/elice\_/article/details/87865610

【问题】spring.xml中配置自动组装提升如下错误

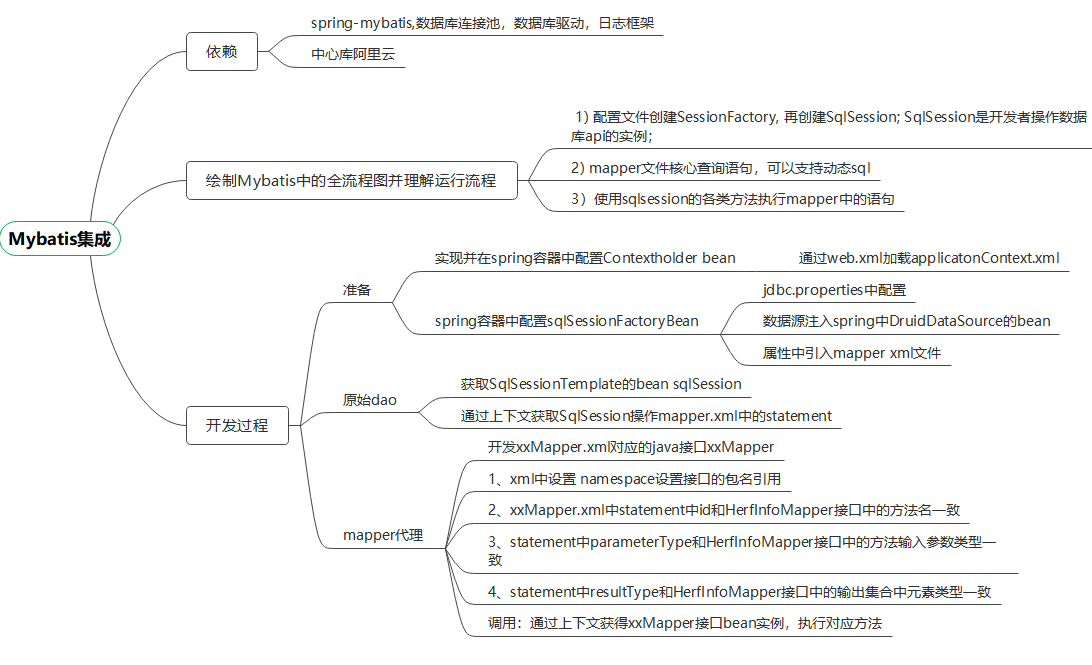
The matching wildcard is strict, but no declaration can be found for element 'context:annotation-config'

<https://www.jianshu.com/p/3ab5a3111df4>



## Mybatis集成（dao层）

### 整合思路



参考文档

Spring的基本配置和Spring与Mybatis的整合

↑ <https://blog.csdn.net/qq_42371269/article/details/83505269>

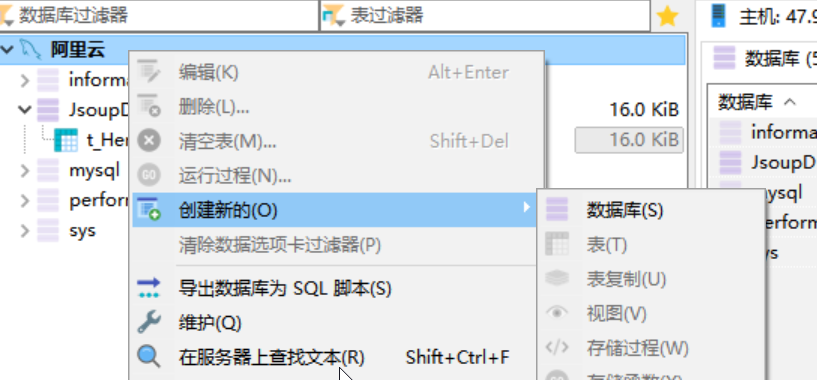
### 关键配置

applicationContext.xml

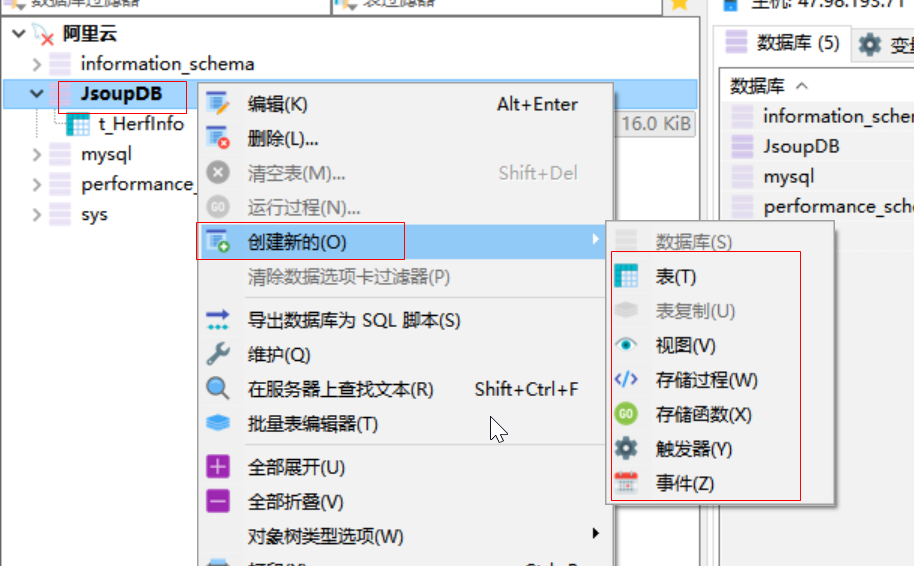
|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns:context="http://www.springframework.org/schema/context"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context.xsd">  <!--读取 jdbc.properties 中的内容 property-placeholder: 占位符 location： 属性文件的位置-->  <context:property-placeholder location="classpath:jdbc.properties"/>  <!-- 1) 获得数据库连接池对象，并交由 spring 同一管理 -->  <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource">  <!-- 连接数据库的驱动，连接字符串，用户名和登录密码-->  <property name="driverClassName" value="${drivername}"/>  <property name="url" value="${url}"/>  <property name="username" value="${jdbc.username}"/>  <property name="password" value="${password}"/>  <!-- 数据池中最大连接数和最小连接数-->  <property name="maxActive" value="${max}"/>  <property name="minIdle" value="${min}"/>  </bean>  <!-- 2) 获取 SqlSessionFactory 对象，并交由 spring 管理-->  <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">  <!-- 注入连接池  给 sqlsessionFactory 的属性 dataSource 赋值  ref="引用该 spring容器 中的另一个 bean的id"  -->  <property name="dataSource" ref="dataSource"/>  <!-- 注入 映射文件 mapper  给 sqlsessionFactory 的属性 mapperLocation 赋值  value="映射文件 XXXmapper.xml 的相对路径"  -->  <property name="mapperLocations" value="classpath:mapper/\*.xml"/>  </bean>  <!-- 方式1 获取 SqlSession 对象，并交由 spring 管理 用SqlSessionTemplate得到的SqlSession可以不用我们自己操心事务的管理，以及关闭操作-->  <bean id="sqlSession" class="org.mybatis.spring.SqlSessionTemplate">  <!-- 给 SqlSessionTemplate 的构造函数赋值-->  <constructor-arg index="0" ref="sqlSessionFactory"/>  </bean>  <!--方式2 搜索有哪些 mapper 实现类，把mapper接口自动配置成 spring 中的 <bean>-->  <bean id="scannerConfigurer" class="org.mybatis.spring.mapper.MapperScannerConfigurer">  <!-- name="basePackage":(起始)包名, 从这个包开始扫描-->  <property name="basePackage" value="tech.pariote.dao"/>  </bean>  <!--启用@Autowired注解-->  <context:annotation-config/>  <bean id="ContextHolder" class="tech.pariote.kernel.ContextHolder"/>  <bean id="getHerfRequest" class="tech.pariote.bean.HerfInfoRequest">  <constructor-arg index="0" value="1"/>  </bean>  </beans> |

### 数据库创建

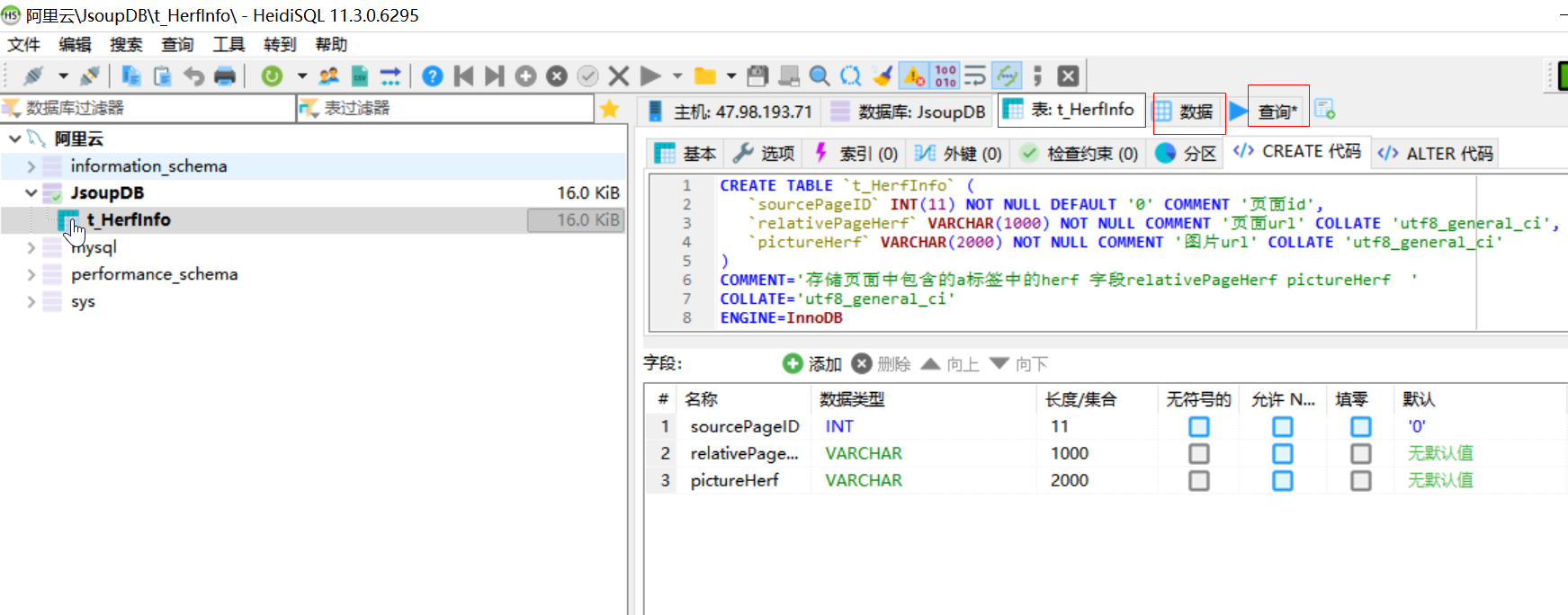
通过ide创建数据库



通过ide创建表等

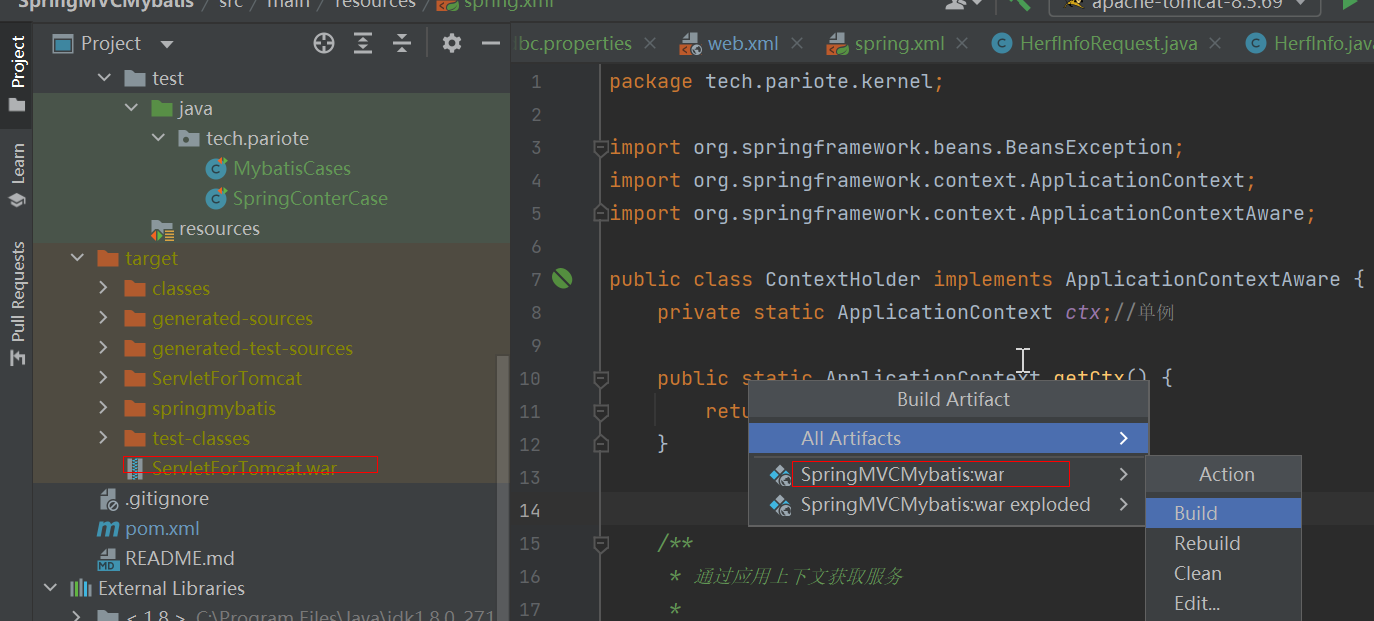


通过ide获取表create语句用于升级现网，插入测试数据，运行sql语句等。



## 部署阿里云调用成功

编译一个war包，部署到tomcat webapps目录下，如果tomcat运行中会自动解压到同名目录下部署。



测试部署后成功

