**1.SpringBoot的作用**

--创建独立的spring应用程序

--嵌入tomcat，无须部署war文件

--简化maven配置

--自动配置Spring

--没有代码生成和对XML没有要求配置

**2. Spring的java配置方式**

Java配置是Spring4.x推荐的配置方式，可以完全替代xml配置

2.1 @Configuration和@Bean

--@Configuration作用于类上，相当于一个xml配置文件

--@Bean作用于方法上，相当于xml配置中的<bean>

2.2实例

该实例演示了通过java配置的方式进行配置spring，并实现了ioc的功能

2.2.1创建工程以及导入依赖

|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.5.2.RELEASE</version>  </parent>  <groupId>cn.itcast.springboot</groupId>  <artifactId>itcast-springboot</artifactId>  <version>1.0.0-SNAPSHOT</version>  <packaging>war</packaging>  <dependencies>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</artifactId>  <version>4.3.7.RELEASE</version>  </dependency>  <!-- 连接池 -->  <dependency>  <groupId>com.jolbox</groupId>  <artifactId>bonecp-spring</artifactId>  <version>0.8.0.RELEASE</version>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  </dependencies>  <build>  <finalName>${project.artifactId}</finalName>  <plugins>  <!-- 资源文件拷贝插件 -->  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-resources-plugin</artifactId>  <configuration>  <encoding>UTF-8</encoding>  </configuration>  </plugin>  <!-- java编译插件 -->  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <configuration>  <source>1.7</source>  <target>1.7</target>  <encoding>UTF-8</encoding>  </configuration>  </plugin>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  <pluginManagement>  <plugins>  <!-- 配置Tomcat插件 -->  <plugin>  <groupId>org.apache.tomcat.maven</groupId>  <artifactId>tomcat7-maven-plugin</artifactId>  <version>2.2</version>  </plugin>  </plugins>  </pluginManagement>  </build>  </project> |

2.2.2 编写user对象

|  |
| --- |
| **package** cn.itcast.springboot.javaconfig;  **public** **class** User {  **private** String username;  **private** String password;  **private** Integer age;  **public** String getUsername() {  **return** username;  }  **public** **void** setUsername(String username) {  **this**.username = username;  }  **public** String getPassword() {  **return** password;  }  **public** **void** setPassword(String password) {  **this**.password = password;  }  **public** Integer getAge() {  **return** age;  }  **public** **void** setAge(Integer age) {  **this**.age = age;  }  } |

2.2.3编写userdao对象

|  |
| --- |
| package cn.itcast.springboot.javaconfig;  import java.util.ArrayList;  import java.util.List;  public class UserDAO {  public List<User> queryUserList() {  List<User> result = new ArrayList<User>();  // 模拟数据库的查询  for (int i = 0; i < 10; i++) {  User user = new User();  user.setUsername("username\_" + i);  user.setPassword("password\_" + i);  user.setAge(i + 1);  result.add(user);  }  return result;  }  } |

2.2.4 编写userService

|  |
| --- |
| package cn.itcast.springboot.javaconfig;  import java.util.List;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Service;  @Service  public class UserService {  @Autowired  // 注入Spring容器中的bean对象  private UserDAO userDAO;  public List<User> queryUserList() {  // 调用userDAO中的方法进行查询  return this.userDAO.queryUserList();  }  } |

2.2.5编写springconfiguration

|  |
| --- |
| package cn.itcast.springboot.javaconfig;  import javax.sql.DataSource;  import org.springframework.beans.factory.annotation.Value;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.ComponentScan;  import org.springframework.context.annotation.Configuration;  import org.springframework.context.annotation.PropertySource;  import com.jolbox.bonecp.BoneCPDataSource;  @Configuration  // 通过该注解来表明该类是一个Spring的配置，相当于一个xml文件  @ComponentScan(basePackages = "cn.itcast.springboot.javaconfig")  // 配置扫描包  @PropertySource(value = { "classpath:jdbc.properties" }, ignoreResourceNotFound = true)  public class SpringConfig {  @Bean  // 通过该注解来表明是一个Bean对象，相当于xml中的<bean>  public UserDAO getUserDAO() {  return new UserDAO(); // 直接new对象做演示  }  @Value("${jdbc.url}")  private String jdbcUrl;  @Value("${jdbc.driverClassName}")  private String jdbcDriverClassName;  @Value("${jdbc.username}")  private String jdbcUsername;  @Value("${jdbc.password}")  private String jdbcPassword;  @Bean(destroyMethod = "close")  public DataSource dataSource() {  BoneCPDataSource boneCPDataSource = new BoneCPDataSource();  // 数据库驱动  boneCPDataSource.setDriverClass(jdbcDriverClassName);  // 相应驱动的jdbcUrl  boneCPDataSource.setJdbcUrl(jdbcUrl);  // 数据库的用户名  boneCPDataSource.setUsername(jdbcUsername);  // 数据库的密码  boneCPDataSource.setPassword(jdbcUsername);  // 检查数据库连接池中空闲连接的间隔时间，单位是分，默认值：240，如果要取消则设置为0  boneCPDataSource.setIdleConnectionTestPeriodInMinutes(60);  // 连接池中未使用的链接最大存活时间，单位是分，默认值：60，如果要永远存活设置为0  boneCPDataSource.setIdleMaxAgeInMinutes(30);  // 每个分区最大的连接数  boneCPDataSource.setMaxConnectionsPerPartition(100);  // 每个分区最小的连接数  boneCPDataSource.setMinConnectionsPerPartition(5);  return boneCPDataSource;  }  } |

**3.Spring boot**

**3.1什么是spring boot**

使用“习惯优于配置”（项目中存在大量的配置，此外还内置一个习惯性的配置，让你无须手动进行配置）的理念让你的项目快速运行起来。使用spring Boot很容易创建一个独立运行、准生产级别的基于Spring框架的项目，使用Spring boot你可以不用或者只需要很少的spring 配置。

**3.2Spring boot优缺点**

优点：

--快速构建项目

--对主流开发框架的无配置集成

--项目可独立运行，无须外部依赖servlet容器

--提供运行时的应用监控

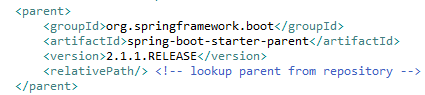
--极大地提高了开发、部署效率

--与云计算的天然集成

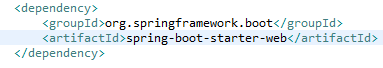
**3.3快速入门**

3.3.1 设置spring boot的parent

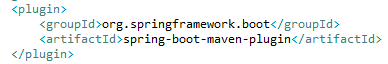
Springboot的项目必须要将parent设置为spring boot的parent，该parent包含了大量默认的配置，大大简化了我们的开发



3.3.2 导入spring boot 的web支持



3.3.3添加spring boot插件

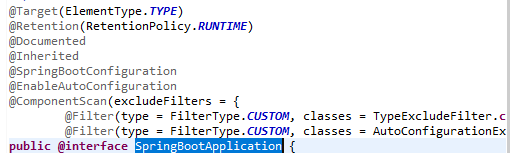


**3.4Spring boot的核心**

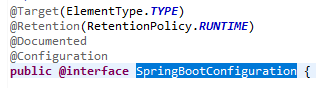
**3.4.1 入口类和@SpringBootApplication**

Spring boot的项目一般都会有\*Application的入口类，入口类中会有main方法，这是一个标准的java应用程序的入口方法

**--@SpringBootApplication**注解是Spring boot的核心注解，它其实是一个组合注解



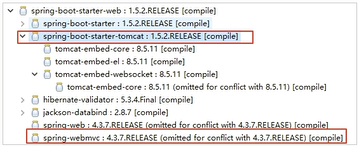
@SpringBootConfiguration注解也是一个组合注解



推荐使用@SpringBootConfiguration代替@Configuration

**-- @EnableAutoConfiguration：**启用自动配置，该注解会使Spring boot根据项目中依赖的jar自动配置项目的配置项；

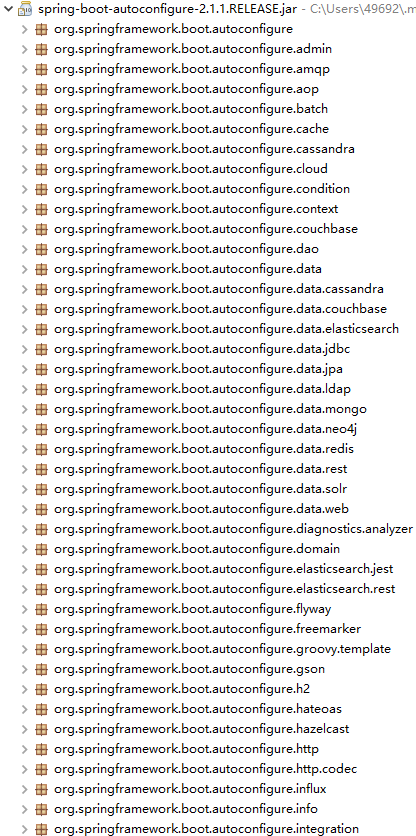
--如：我们添加了Spring-boot-starter-web的依赖，项目中也就会引入SpringMvc的依赖，Spring boot就会自动配置tomcat和springmvc



--@ComponentScan：默认扫描@SpringBootApplication所在类的**同级目录以及他的子目录**

**3.4.2 关闭自动配置**

通过上述，我们得知，Spring boot会根据项目中的jar包依赖，自动做出配置，spring boot支持的自动配置



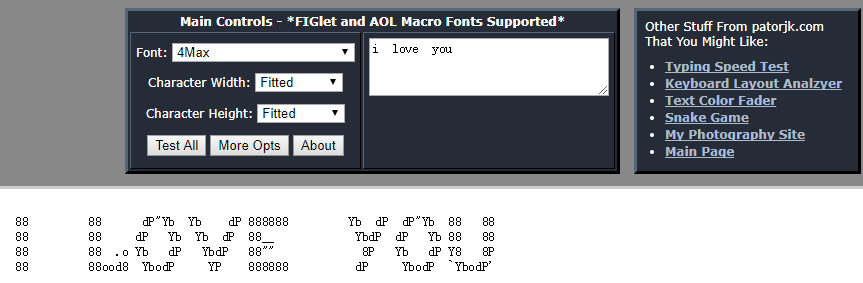
如果我们不需要springboot自动配置，想关闭某一项的自动配置，该如何设置呢？

比如：我们不想自动配置redis，想手动配置



**3.4.3自动义banner**

<http://patorjk.com/software/taag/#p=display&h=1&v=1&f=4Max&t=i%20%20love%20%20you>



将文字复制保存到一个文本文档中，banner.txt

将banne.txt拷贝到项目的resource目录中

然后重启程序，运行即可

如果不想看banner，可以将其关闭



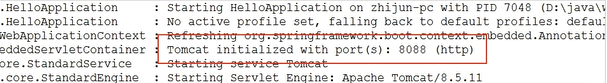
**3.4.4 全局配置文件**

Spring boot项目使用一个全局的配置文件application.properties或者是application.yml，在resources目录下或者类路径下的/config下，一般我们放到resource下

--修改tomcat的端口为8088

server.port=8088

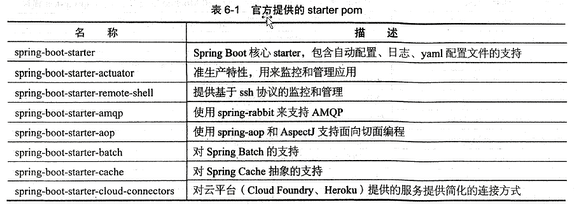
重新启动应用，查看效果：



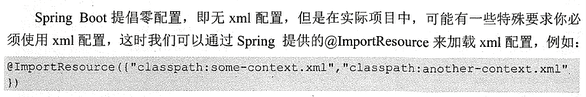
--修改进入DispatcherServlet的规则为：\*.html



**3.4.5 Starter pom**



**3.4.6 xml配置文件**



**3.4.7 日志**

Spring boot对各种日志框架都做了支持，我们可以通过配置来修改默认的日志的配置

#设置日志的级别

logging.level.org.springframework=DEBUG

格式:

Logging.level.\*= # log levels severity mapping. For instance `logging.level.org.springframework=DEBUG`

**3.5 Spring boot的自动配置的原理**

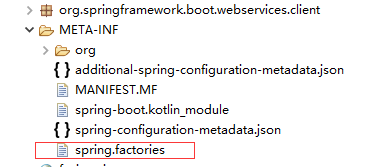
Spring boot在进行springApplication对象实例化时，会加载META-INF/spring.factories文件，将该配置文件中的配置载入到spring容器中

**3.5.1 maven下载源码**

通过dependency:sources该命令可以下载该项目所有的依赖的包的源码

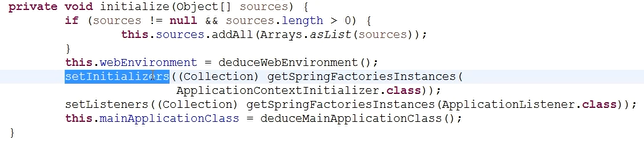
**3.5.2源码分析**

Spring-boot-release-》



打开SpringApplication这个类对象

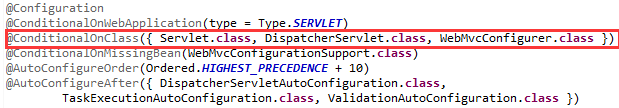




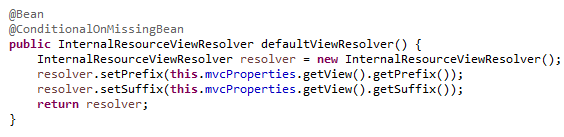
**4. Spring boot的web开发**

Web开发的自动配置类，org. springframework. boot.autoconfigure. web. WebMvcAutoConfiguration

4.1自动配置的ViewResolver



条件注解，表示在有servlet，dispatcherServlet，webmvc等情况下才回进行自动配置、



对照springmvc中的



**4.2自动配置静态资源**

将静态资源的路径设置为

spring.resource.static-location=classpath:/META-INF/resources/,classpath:/resources, classpath:/static/,classpath:/public/

**4.2.1 进入规则为\*.xxx或者不指定静态文件路径时**

将静态资源放置到webapp下面的static目录中即可通过地址访问

**4.3 自定义消息转化器**

自定义消息转化器，只要在@Configuration的类中添加消息转化器的@bean加入到spring 容器中，就会被spring boot自动加入到容器中



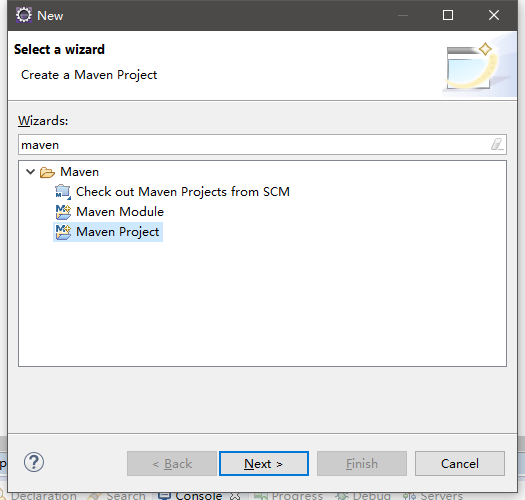
**4.4 自定义springmvc的配置**

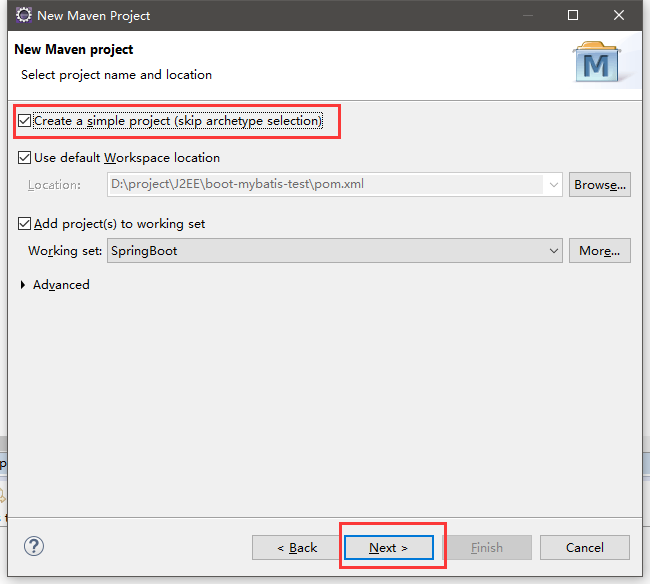
有些时候我们需要自己配置springmvc而不是采用默认，比如说增加一个拦截器，这个时候就得通过继承webmvcConfigurerAdapter，然后重写父类中的方法进行扩展。

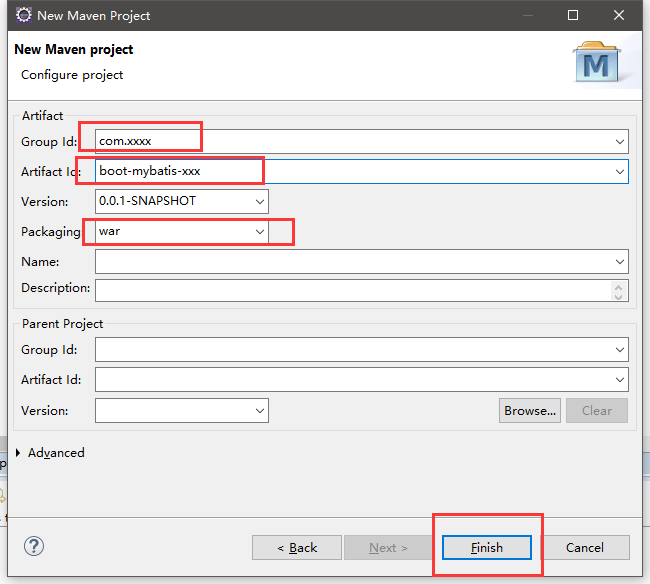
|  |
| --- |
| package cn.itcast.springboot.demo;  import java.nio.charset.Charset;  import java.util.List;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  import org.springframework.context.annotation.Configuration;  import org.springframework.http.converter.HttpMessageConverter;  import org.springframework.http.converter.StringHttpMessageConverter;  import org.springframework.web.servlet.HandlerInterceptor;  import org.springframework.web.servlet.ModelAndView;  import org.springframework.web.servlet.config.annotation.InterceptorRegistry;  import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;  @Configuration //申明这是一个配置  public class MySrpingMVCConfig extends WebMvcConfigurerAdapter{  // 自定义拦截器  @Override  public void addInterceptors(InterceptorRegistry registry) {  HandlerInterceptor handlerInterceptor = new HandlerInterceptor() {  @Override  public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler)  throws Exception {  System.out.println("自定义拦截器............");  return true;  }  @Override  public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler,  ModelAndView modelAndView) throws Exception {  }  @Override  public void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler,  Exception ex) throws Exception {  }  };  registry.addInterceptor(handlerInterceptor).addPathPatterns("/\*\*");  }  // 自定义消息转化器的第二种方法  @Override  public void configureMessageConverters(List<HttpMessageConverter<?>> converters) {  StringHttpMessageConverter converter = new StringHttpMessageConverter(Charset.forName("UTF-8"));  converters.add(converter);  }  } |

**5.** **springboot+mybatis整合（最详细）从maven项目开始**

**5.1 新建maven项目**

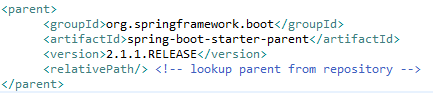




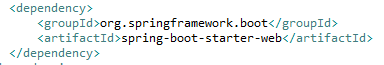


**5.2 在pom.xml中加入spring boot 所需要的配置，创建spring boot的application.java执行文件**

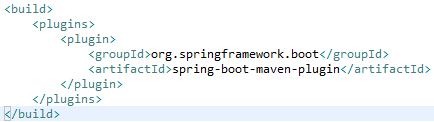
5.2.1 在pom.xml中引入spring-boot-start-parent，这是必须的



5.2.2需要在pom.xml中引入spring-boot-starter-web依赖，依赖放在<dependencies>中，新建<dependency>,spring官方解释spring-boot-start-web包含了spring webmvc和tomcat等web开发的特性。这样就可以不用配置web.xml文件了

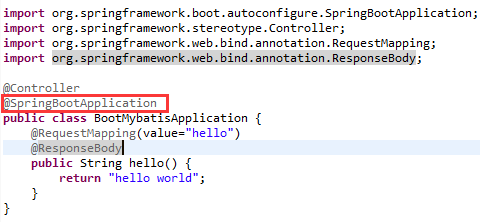


5.2.3如果我们要直接Main启动spring，那么以下plugin必须要添加，否则是无法启动的。如果使用maven的spring-boot:run的话就不需要此配置。



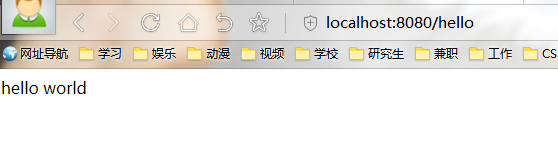
到这里spring boot的pom.xml文件，基本配置完成

**5.3为了测试环境，设计一个启动类进行测试**



如果发现SpringBootApplication这个注解无法识别或者导入，可能是版本不匹配的问题，看看导入的maven包里面的spring-boot-autoConfigure的版本与自定义的版本一不一致，如果不一致，就到本地依赖包里面将repository\org\springframework\boot\spring-boot-autoconfigure文件夹删除，然后update项目即可，重启eclipse





至此，一个spring boot项目已经搭建完毕，接下来整合Mybatis项目；

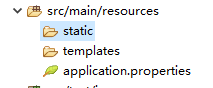
**5.4整合mybatis项目**

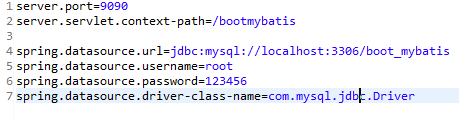
5.4.1在pom.xml文件中加入mybatis需要的依赖



如果没有写mysql的版本，可能会默认下载最新版，这样的话mysql的驱动写法可能发生改变，需要注意一下。

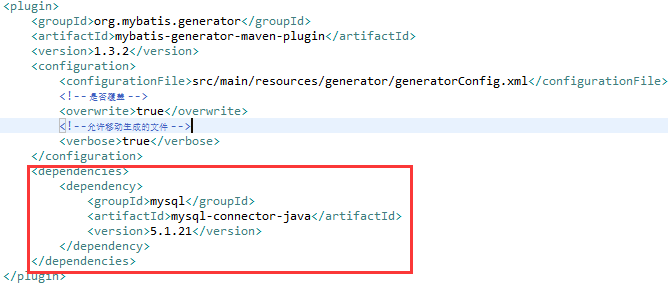
5.4.2配置application.properties文件





5.4.3 mybatis generator的相关配置文件

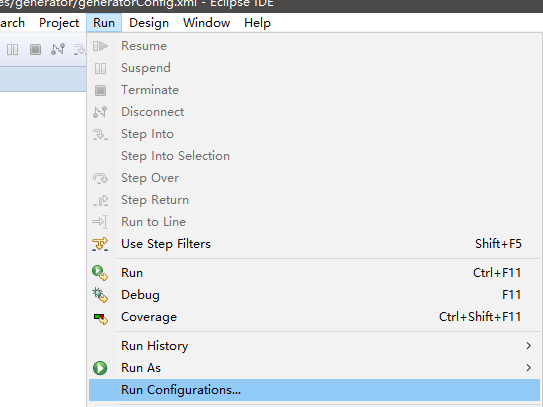
Pom.xml的添加：

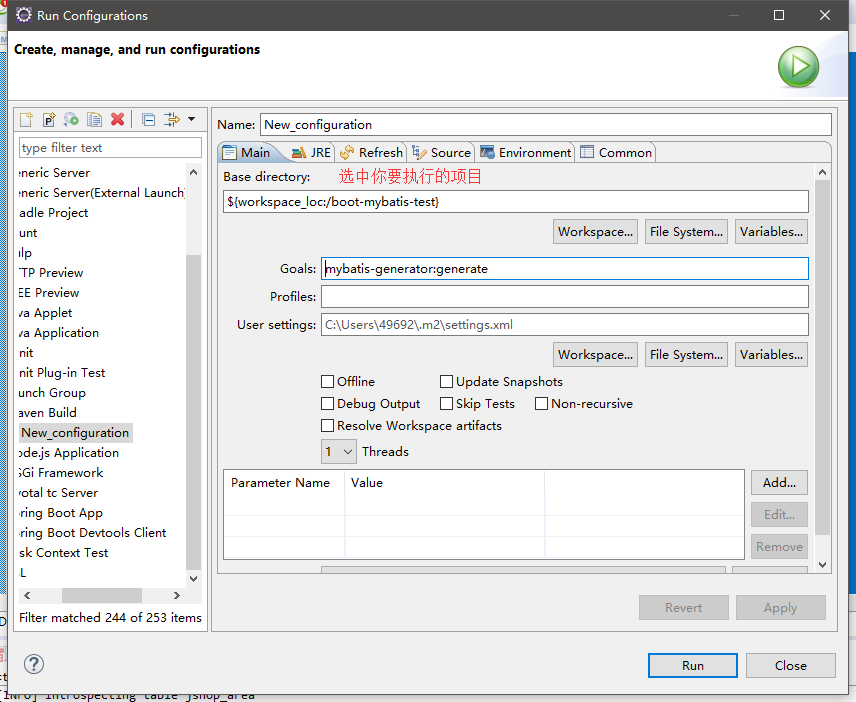


在resources/generator/创建一个generatorConfig.xml文件

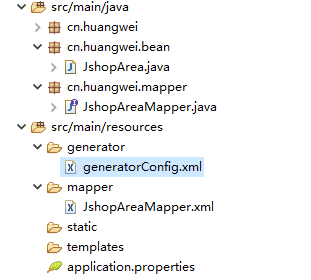
|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!DOCTYPE generatorConfiguration  PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"  "http://mybatis.org/dtd/mybatis-generator-config\_1\_0.dtd">  <generatorConfiguration>  <!-- 引入配置文件 -->  <properties resource=*"application.properties"*/>  <!-- 一个数据库一个context -->  <!--  defaultModelType:指定生成对象的样式  flat：所有内容（主键，blob）等全部生成在一个对象中  targetRuntime:  MyBatis3Simple：类似MyBatis3，只是不生成XXXBySample  -->  <context id=*"Mysql"* targetRuntime=*"MyBatis3Simple"* defaultModelType=*"flat"*>  <!--  delimitIdentifiers：参考tableName的解释，  注意，默认的delimitIdentifiers是双引号，  如果类似MYSQL这样的数据库，使用的是`（反引号，  那么还需要设置context的beginningDelimiter和endingDelimiter属性）  -->  <property name=*"beginningDelimiter"* value=*"`"*/>  <property name=*"endingDelimiter"* value=*"`"*/>    <!-- 注释 -->  <commentGenerator >  <property name=*"suppressAllComments"* value=*"true"*/><!-- 是否取消注释 -->  <property name=*"suppressDate"* value=*"true"* /> <!-- 是否生成注释代时间戳-->  </commentGenerator>    <!-- jdbc连接 -->  <jdbcConnection driverClass=*"${spring.datasource.driver-class-name}"*  connectionURL=*"${spring.datasource.url}"*  userId=*"${spring.datasource.username}"*  password=*"${spring.datasource.password}"*>  </jdbcConnection>    <!-- 类型转换 -->  <javaTypeResolver>    <!-- 是否使用bigDecimal， false可自动转化以下类型（Long, Integer, Short, etc.） -->  <property name=*"forceBigDecimals"* value=*"false"*/>  </javaTypeResolver>    <!-- 生成实体类地址 -->  <javaModelGenerator targetPackage=*"cn.huangwei.bean"* targetProject=*"src/main/java"*>    <!-- 是否在当前路径下新加一层schema,eg：fase路径com.oop.eksp.user.model， true:com.oop.eksp.user.model.[schemaName] -->  <property name=*"enableSubPackages"* value=*"false"*/>    <!-- 是否针对string类型的字段在set的时候进行trim调用 -->  <property name=*"trimStrings"* value=*"true"*/>  </javaModelGenerator>    <!-- 生成mapper.xml文件 -->  <sqlMapGenerator targetPackage=*"mapper"* targetProject=*"src/main/resources"* >  <property name=*"enableSubPackages"* value=*"false"* />  </sqlMapGenerator>    <!-- 生成mapper.xml对应mapper.java，也就是接口dao -->  <javaClientGenerator targetPackage=*"cn.huangwei.mapper"* targetProject=*"src/main/java"*  type=*"XMLMAPPER"*>  <property name=*"enableSubPackages"* value=*"true"*/>  </javaClientGenerator>  <table tableName=*"jshop\_area"* domainObjectName=*"JshopArea"*  enableInsert=*"true"* enableSelectByPrimaryKey=*"true"*  enableSelectByExample=*"false"* enableUpdateByPrimaryKey=*"true"*  enableDeleteByPrimaryKey=*"true"* enableDeleteByExample=*"false"*  enableCountByExample=*"false"* enableUpdateByExample=*"false"*  selectByPrimaryKeyQueryId=*"false"* selectByExampleQueryId=*"true"*>  <!--  schema即为数据库名 tableName为对应的数据库表 domainObjectName  是要生成的实体类 enable\*ByExample  是否生成 example类 -->  </table>  </context>  </generatorConfiguration> |

执行generator代码





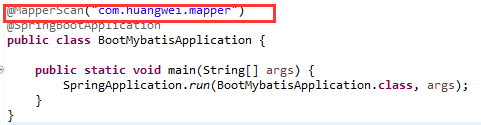
就能看到生成的代码了



5.4.4 在application添加mybatis别名与扫描包路径



在springboot的启动文件BootMybatisApplication.java添加注解

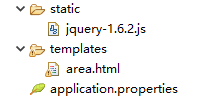


另外还得在mapper.java上添加@Mapper注解



添加controller（JshopAreaController.java）

|  |
| --- |
| **package** cn.huangwei.controller;  **import** java.util.ArrayList;  **import** java.util.List;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.RequestBody;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.ResponseBody;  **import** com.huangwei.service.AreaService;  **import** cn.huangwei.bean.JshopArea;  @Controller  @RequestMapping(value="/wap/integral/")  **public** **class** JshopAreaController {  @Autowired  **private** AreaService areaService;    @ResponseBody  @RequestMapping(value="")  **public** String getAreaHtml() {  **return** "area";  }    @ResponseBody  @RequestMapping(value="queryAreaList")  **public** List<JshopArea> integralAreaList(Long id) {  List<JshopArea> resultList = areaService.findByParentId(id);  **return** resultList;  }  @ResponseBody  @RequestMapping(value = "selectedArea")  **public** List<JshopArea> getSelectedArea(@RequestBody List<String> stringIds){  List<Long> ids = **new** ArrayList<>();  **for**(String id : stringIds){  ids.add(Long.*parseLong*(id));  }  List<JshopArea> resultList = areaService.findBySelectedId(ids);  **return** resultList;  }  } |



添加area.html

|  |
| --- |
| <!DOCTYPE html>  <html lang=*"en"*>  <head>  <meta charset=*"UTF-8"*>  <meta name=*"viewport"*  content=*"width=device-width, initial-scale=1, minimum-scale=1, maximum-scale=1, user-scalable=no"*>  <title>收货地址管理</title>    </head>  <body>  <div>  <div id=*"result"*>  </div>  <table id=*"area"*>  <thead>  <td>选择</td>  <td>区域</td>  </thead>  </table>  <br/>  <input type=*"button"* value=*"提交"* onclick="getArea()"/>  </div>  <script type=*"text/javascript"* src=*"/jquery-1.6.2.js"*></script>  <script>  $(**function** () {  initpage();  });  **function** getArea() {  **var** divResult = $("#result");  divResult.empty();  **var** stringIds = **new** Array();  **var** checkedId = $('input[name="header"]:checked');  checkedId.each(**function** (i) {  stringIds[i] = $(**this**).val();  });  $.ajax({  type:'post',  url:'/wap/integral/selectedArea',  data:JSON.stringify(stringIds),  contentType:"application/json",  success: **function**(data){  //data = JSON.parse(data);  /\* if (data.code == "000000") {  for (var i = 0; i < data.dataArray.length; i++) {  divResult.append("<span>" + data.dataArray[i].areaname + "</span><br/>")  }  } \*/  **for** (**var** i = 0; i < data.length; i++) {  divResult.append("<span>" + data[i].areaname + "</span><br/>")  }  }  });  }  **function** initpage() {  $.ajax({  url: '/wap/integral/queryAreaList',  success: **function** (data) {  //data = JSON.parse(data);  /\*if (data.code == "000000") {  for (var i = 0; i < data.dataArray.length; i++) {  var trNode = $("<tr></tr>");  trNode.append($("<input type='checkbox' value='"+data.dataArray[i].id+"' name='header'/>"));  trNode.append($("<td>" + data.dataArray[i].areaname + "</td>"));  $("#area").append(trNode);  }  }\*/  **for** (**var** i = 0; i < data.length; i++) {  **var** trNode = $("<tr></tr>");  trNode.append($("<input type='checkbox' value='"+data[i].id+"' name='header'/>"));  trNode.append($("<td>" + data[i].areaname + "</td>"));  $("#area").append(trNode);  }  }  });  }  </script>  </body>  </html> |

5.5 测试结果

正常放在webapp是可以访问htm文件的

然后如果将html文件放在static文件夹中，得配置静态资源的访问

配置application.properties文件

