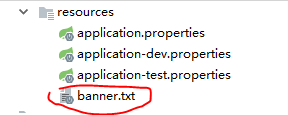
Spring Boot

* **开启自动配置，关闭某些配置 :**

@EnableAutoConfiguration(exclude = {MongoAutoConfiguration.**class**})

该注解会使SpringBoot根据项目依赖的jar包自动配置项目的配置项。比如：添加web的依赖，项目会自动配置tomcat和SpringMVC

* **自定义bannner**



http://patorjk.com/software/taag/

* **全局配置文件 application.properties application.yml :**

*#server.port=8088*

*端口号  
#server.servlet.context-path=/ww*

*上下文路径*

* **属性注入：**

**application.properties**

**book.author**=**Tom  
book.name**=**SpringBoot**

*@Value("${book.author}")***private** String **author**;

@Value注解不需要set/get方法

* **类型安全配置**

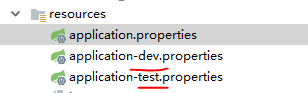
@ConfigurationProperties(prefix = **"book"**)

需要set/get方法

* **Profile配置：**

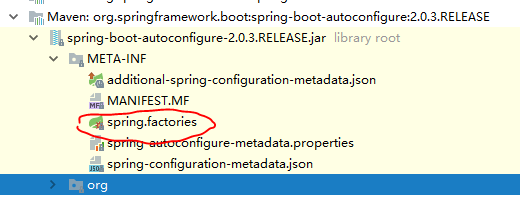
application.properties

**spring.profiles.active**=**test**

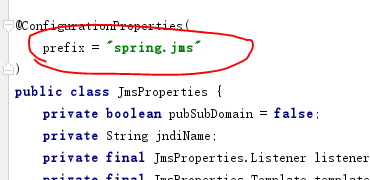


* **自动配置原理：**

1. 读取文件



1. 根据这个文件中的类名，实例化对象，并且初始化属性默认值
2. 如果在全局配置文件中定义属性，则覆盖初始化默认属性。如果没有定义，则仍用初始化默认属性值



* **创建父工程：**

*<?***xml version="1.0" encoding="UTF-8"***?>*<**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**>  
  
 <**groupId**>com.sky</**groupId**>  
 <**artifactId**>springboot\_parent</**artifactId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
 <**packaging**>pom</**packaging**>  
  
 <**dependencyManagement**>  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-dependencies</**artifactId**>  
 <**version**>2.0.2.RELEASE</**version**>  
 <**type**>pom</**type**>  
 <**scope**>import</**scope**>  
 </**dependency**>  
 </**dependencies**>  
 </**dependencyManagement**>  
  
 <**build**>  
 <**plugins**>  
 <**plugin**>  
 <**groupId**>org.apache.maven.plugins</**groupId**>  
 <**artifactId**>maven-compiler-plugin</**artifactId**>  
 <**configuration**>  
 <**source**>1.8</**source**>  
 <**target**>1.8</**target**>  
 </**configuration**>  
 </**plugin**>  
 </**plugins**>  
 </**build**>  
  
  
</**project**>

子工程不需要定义版本

*<?***xml version="1.0" encoding="UTF-8"***?>*<**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"**>  
 <**parent**>  
 <**artifactId**>springboot\_parent</**artifactId**>  
 <**groupId**>com.sky</**groupId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
 </**parent**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
  
 <**artifactId**>springboot\_child1</**artifactId**>  
  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-web</**artifactId**>  
 </**dependency**>  
 </**dependencies**>  
  
</**project**>

* **整合测试 :**

Test包只用于测试阶段所以scope是test

<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-test</**artifactId**>  
 <**scope**>test</**scope**>  
</**dependency**>  
  
<**dependency**>  
 <**groupId**>junit</**groupId**>  
 <**artifactId**>junit</**artifactId**>  
 <**scope**>test</**scope**>  
</**dependency**>

指明要测试的类

@SpringBootTest(classes = SpringController.**class**)

指明实现测试的类

@RunWith(SpringJUnit4ClassRunner.**class**)

定义和Web的整合

@WebAppConfiguration

Demo:

@SpringBootTest(classes = SpringController.**class**)  
@RunWith(SpringJUnit4ClassRunner.**class**)  
@WebAppConfiguration  
**public class** TestSpringController {  
  
 @Autowired  
 **private** SpringController **springController**;  
  
 @Test  
 **public void** test1() {  
 TestCase.*assertEquals*(**this**.**springController**.yes(), **"hello"**);  
 }  
}

* **配置包扫描，导入Controller :**

@ComponentScan(value = **"com.sky.controller"**)

* **组合注解@SpringBootApplication :**

可以代替@ComponentScan和@EnableAutoConfiguration

**注意点：**

默认情况下扫描的是@SpringBootApplication所在的包及其子包，如果你要注入的类不在这个范围内，需要指定扫描的包名@SpringBootApplication(scanBasePackages{**"com.sky.controller"**})

* **@ResponseBody :**

返回Restful内容，不适用该注解，会进行url跳转

* **@RestController :**

如果每个功能都是restful内容，那么我们可以使用该注解，**@ResponseBody**就不需要了

@RestController  
**public class** SpringController {  
  
 @RequestMapping(**"/hello"**)**public** String yes() {  
 **return "hello"**;  
 }

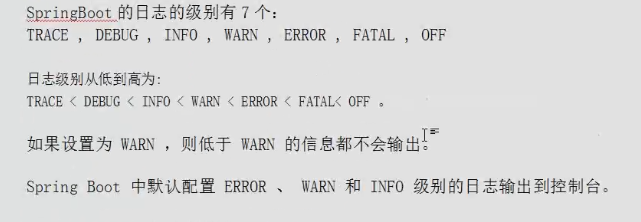
}

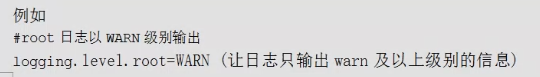
* **支持Rest风格 :**

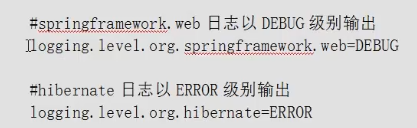
@RequestMapping(**"/info/{msg}"**)  
**public** String show(@PathVariable(value = **"msg"**) String m) {  
 **return "show "** + m;  
}

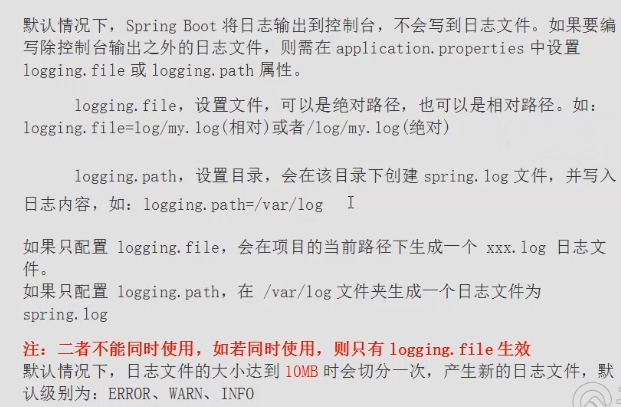
**Mapping: Msg->m**

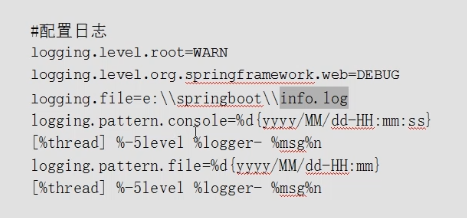
* **日志 :**



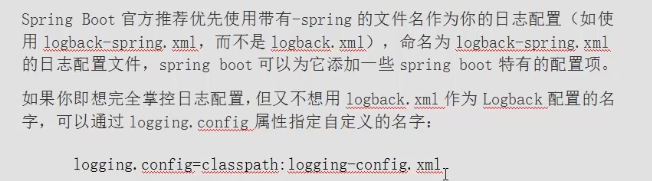


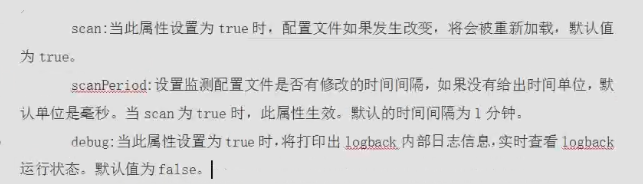






* **自定义日志配置文件 :**





**application.properties**

**logging.config**=**classpath:logback-spring.xml**

****

* **通过profile定义日志输出 :**

**application.properties**

**spring.profiles.active**=**prod**

**logback-spring.xml**

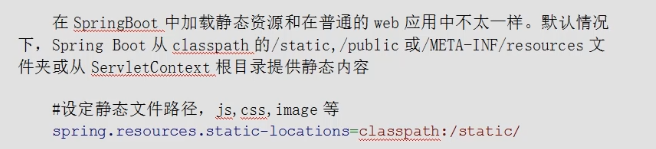
<**springProfile name="prod"**>  
 <**logger name="com.sky.controller.TestController" level="ERROR"**/>  
</**springProfile**>

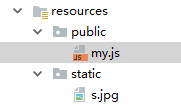
<**layout class="ch.qos.logback.classic.PatternLayout"**>  
 <**springProfile name="test,dev"**>  
 <**pattern**>%d{yyyy-MM-dd HH:mm:ss.SSS} ---->%contextName [%thread] ---> %-5level %logger{50} - %msg%n</**pattern**>  
 </**springProfile**>  
 *<!--<springProfile name="!dev">-->* <**springProfile name="prod"**>  
 <**pattern**>%d{yyyy-MM-dd HH:mm:ss.SSS} ====>%contextName [%thread] ==== %-5level %logger{50} - %msg%n</**pattern**>  
 </**springProfile**>  
</**layout**>

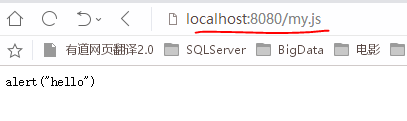
* **配置工程为开发模式（热部署）:**

<**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>springloaded</**artifactId**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-devtools</**artifactId**>  
</**dependency**>

* **访问静态资源 ：**







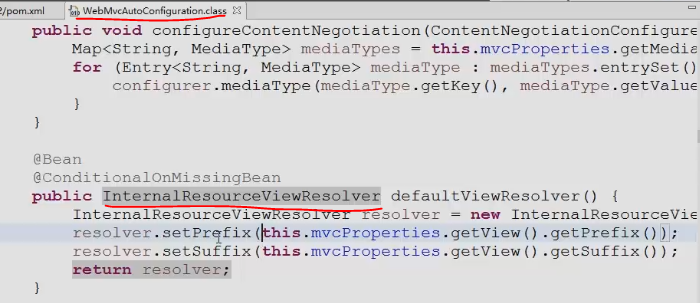
自己指定application.properties，会覆盖原有配置，现在只有static有效

**spring.resources.static-locations**=**classpath:/static**

* **视图解析器 :**

**核心接口：WebMvcConfigurer**

SpingBoot的视图解析器做了自动配置。

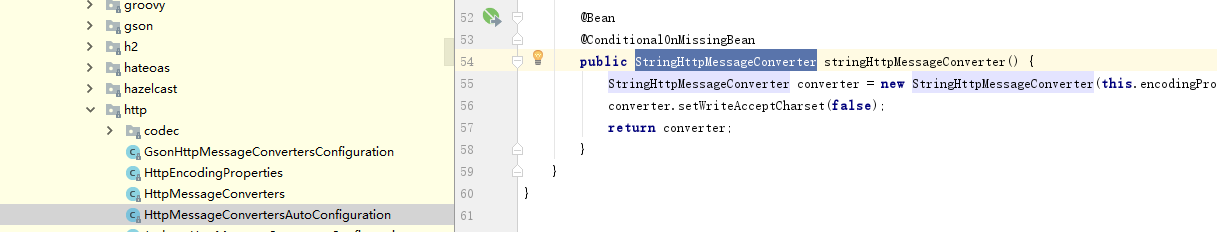


* **消息转换器 :**

SpringBoot默认配置了消息转换器。

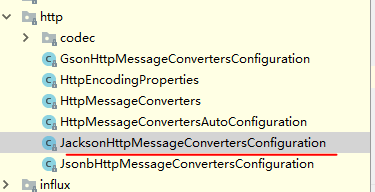
*// 定义消息转换器* @Bean  
 **public** StringHttpMessageConverter stringHttpMessageConverter() {  
 StringHttpMessageConverter converter = **new** StringHttpMessageConverter(Charset.*forName*(**"UTF-8"**));  
 **return** converter;  
 }

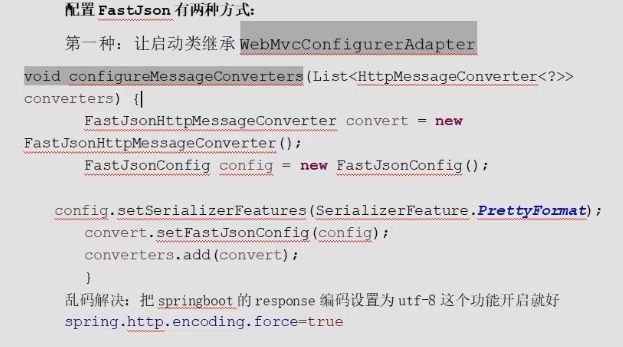
spring-boot-autoconfigure-2.0.2.RELEASE.jar



* **使用FastJson解析Json数据 :**

SpringBoot默认是用了Jackson





第二种

@Bean  
**public** HttpMessageConverters fastJsonMessageConverter(){  
 FastJsonHttpMessageConverter convert = **new** FastJsonHttpMessageConverter();  
 FastJsonConfig config = **new** FastJsonConfig();  
 config.setSerializerFeatures(SerializerFeature.***PrettyFormat***);  
 convert.setFastJsonConfig(config);  
 **return new** HttpMessageConverters(convert);  
}

* **自定义拦截器 :**

@Configuration  
**public class** MyInterceptor **implements** WebMvcConfigurer{  
 @Override  
 **public void** addInterceptors(InterceptorRegistry registry) {  
 HandlerInterceptor inter = **new** HandlerInterceptor() {  
 @Override  
 **public boolean** preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) **throws** Exception {  
 System.***out***.println(**"preHandle MyInterceptor"**);  
 **return true**;  
 }  
  
 @Override  
 **public void** postHandle(HttpServletRequest request, HttpServletResponse response, Object handler, @Nullable ModelAndView modelAndView) **throws** Exception {  
 System.***out***.println(**"postHandle MyInterceptor"**);  
 }  
  
 @Override  
 **public void** afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, @Nullable Exception ex) **throws** Exception {  
 System.***out***.println(**"afterCompletion MyInterceptor"**);  
 }  
 };  
 registry.addInterceptor(inter).addPathPatterns(**"/\*\*"**);  
  
 }  
}

* **全局异常处理器 :**

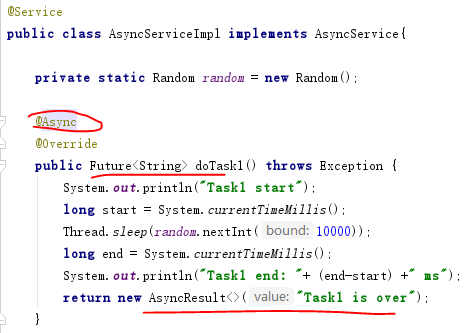
@ControllerAdvice  
**public class** GlobalExceptionHandler {  
  
 @ExceptionHandler(Exception.**class**)  
 @ResponseBody  
 **public** Map<String, Object> handleException(Exception e) {  
 Map<String, Object> map = **new** HashMap<>();  
 map.put(**"errorCode"**, 500);  
 map.put(**"errorMsg"**, e.toString());  
 **return** map;  
 }  
  
}

* **异步调用 :**

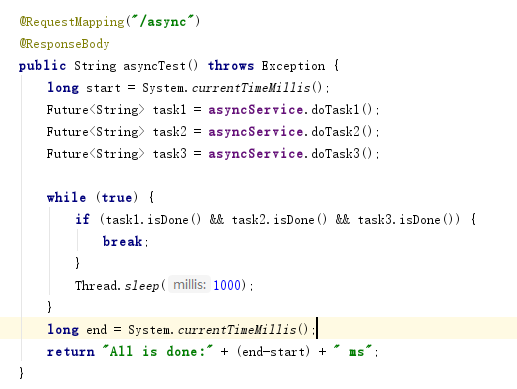
启动类开启异步调用



设计异步调用方法



异步调用



* **整合Jsp :**

添加依赖

<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-web</**artifactId**>  
</**dependency**>

<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-tomcat</**artifactId**>  
</**dependency**>  
  
<**dependency**>  
 <**groupId**>org.apache.tomcat.embed</**groupId**>  
 <**artifactId**>tomcat-embed-jasper</**artifactId**>  
</**dependency**>

packing=war时添加插件

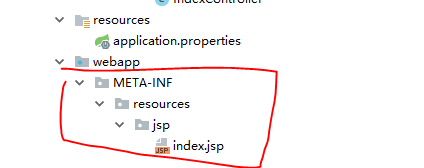
<**build**>  
 <**plugins**>  
 <**plugin**>  
 <**groupId**>org.apache.maven.plugins</**groupId**>  
 <**artifactId**>maven-war-plugin</**artifactId**>  
 <**version**>2.3</**version**>  
 <**configuration**>  
 <**failOnMissingWebXml**>false</**failOnMissingWebXml**>  
 </**configuration**>  
 </**plugin**>  
 </**plugins**>  
</**build**>

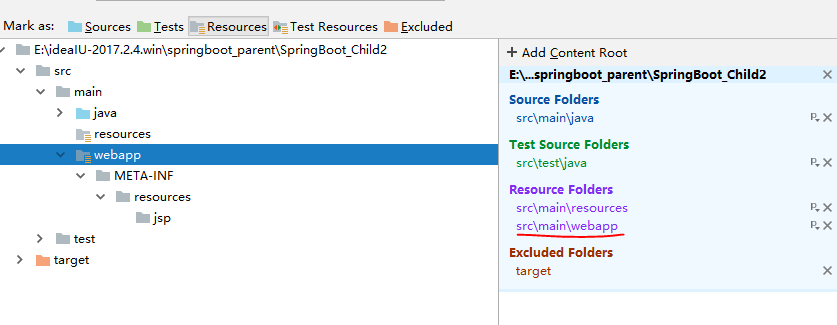
application.properties：

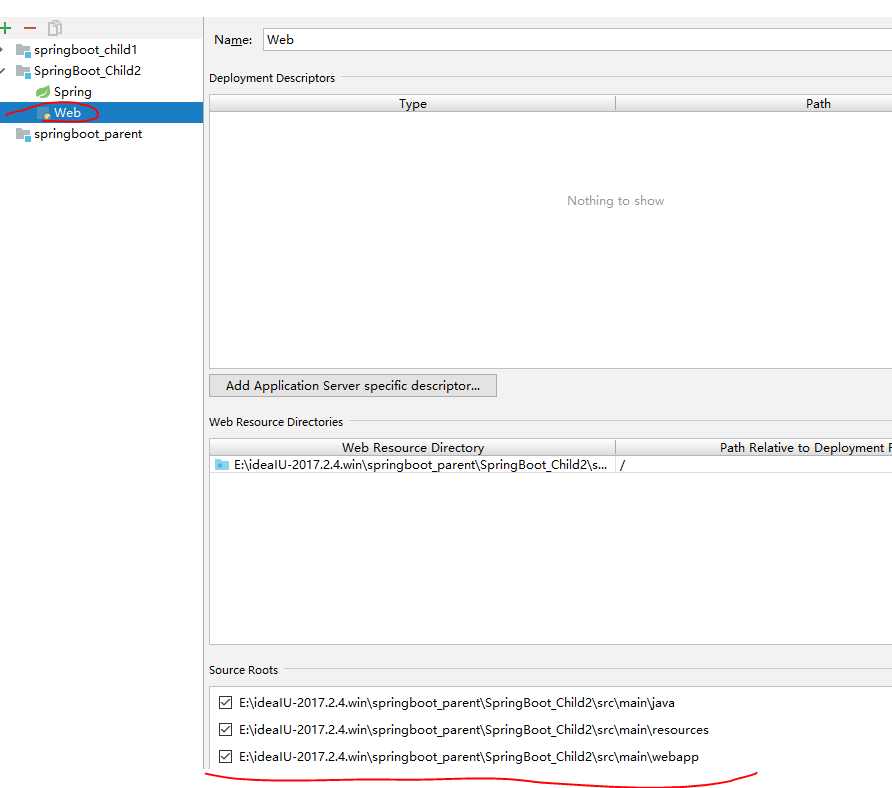
**spring.mvc.view.prefix**=**/jsp/  
spring.mvc.view.suffix**=**.jsp**

定义controller：配合之前定义的前缀和后缀

@RequestMapping(**"/index"**)  
**public** String show(Model model) {  
 model.addAttribute(**"name"**, **"zhaosi"**);  
 **return "index"**; *//返回页面名称*}





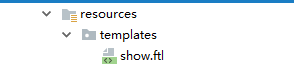


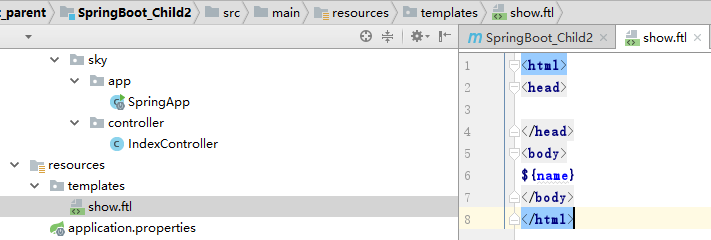
* **整合Freemarker :**

添加依赖

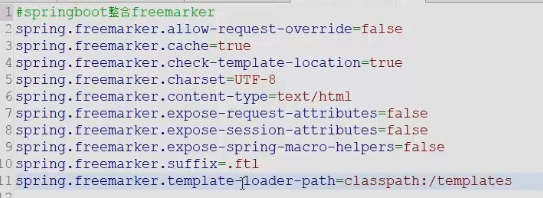
<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-freemarker</**artifactId**>  
</**dependency**>

@RequestMapping(**"/show"**)  
**public** String show(Model model) {  
 model.addAttribute(**"name"**, **"qianfeng"**);  
 **return "show"**; *//返回页面名称*}





自定义配置：

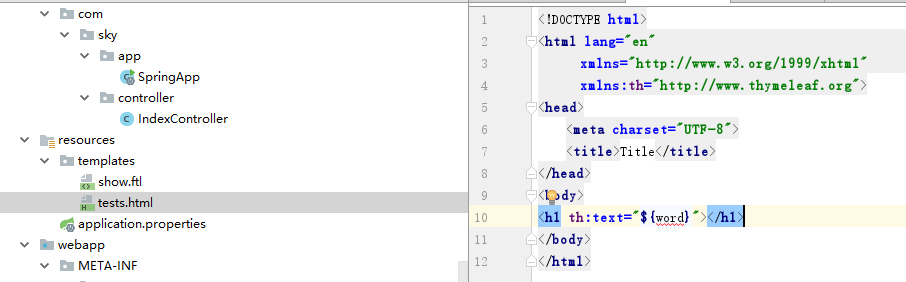


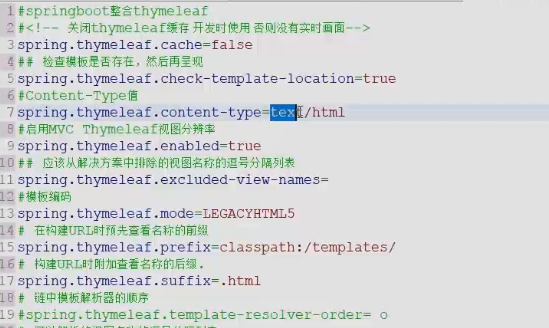
* **整合Thymeleaf :**

添加依赖

<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-thymeleaf</**artifactId**>  
</**dependency**>

@RequestMapping(**"/thymeleafpage"**)  
**public** String thymeleafpage(Model model) {  
 model.addAttribute(**"word"**, **"qianfeng"**);  
 **return "tests"**; *//返回页面名称*}



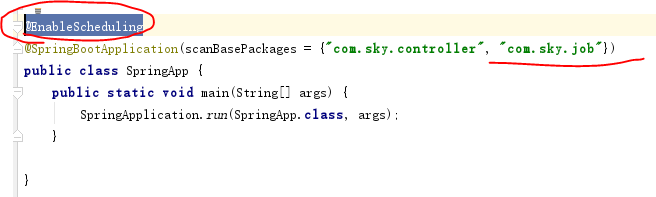


* **配置QuartZ任务调度框架 :**

添加依赖

<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-web</**artifactId**>  
</**dependency**>

@Component  
**public class** MyJob {  
  
 @Scheduled(fixedRate = 1000)  
 **public void** run() {  
 System.***out***.println(**new** SimpleDateFormat(**"yyyy-MM-dd HH:mm:ss"**).format(**new** Date()));  
 }  
  
}



* **整合JdbcTemplate :**

添加依赖

<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-jdbc</**artifactId**>  
</**dependency**>  
  
<**dependency**>  
 <**groupId**>mysql</**groupId**>  
 <**artifactId**>mysql-connector-java</**artifactId**>  
</**dependency**>

@Repository  
**public class** DeptDao {  
  
 @Autowired  
 **private** JdbcTemplate **jdbcTemplate**;  
  
 **public void** addDept(Dept dept) {  
 **jdbcTemplate**.update(**"INSERT INTO dept(dname, db\_source) VALUES (?,?)"** , **new** Object[]{dept.getDname(), dept.getDb\_source()});  
  
  
 }  
  
}

application.properties

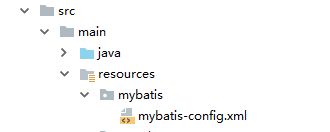
**spring.datasource.driver-class-name**=**com.mysql.jdbc.Driver  
spring.datasource.username**=**root  
spring.datasource.password**= **spring.datasource.url**=**jdbc:mysql://192.168.1.151:3306/cloudDB01**

* **整合MyBatis（xml配置方式）:**

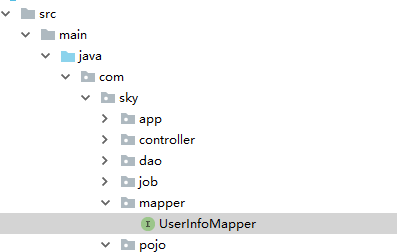
配置依赖

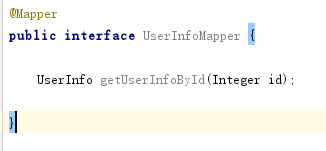
<**dependency**>  
 <**groupId**>org.mybatis.spring.boot</**groupId**>  
 <**artifactId**>mybatis-spring-boot-starter</**artifactId**>  
 <**version**>1.3.2</**version**>  
</**dependency**>  
  
<**dependency**>  
 <**groupId**>com.alibaba</**groupId**>  
 <**artifactId**>druid-spring-boot-starter</**artifactId**>  
 <**version**>1.1.0</**version**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>com.alibaba</**groupId**>  
 <**artifactId**>druid</**artifactId**>  
 <**version**>1.0.11</**version**>  
</**dependency**>  
*<!--分页-->*<**dependency**>  
 <**groupId**>com.github.pagehelper</**groupId**>  
 <**artifactId**>pagehelper-spring-boot-starter</**artifactId**>  
 <**version**>1.1.2</**version**>  
</**dependency**>

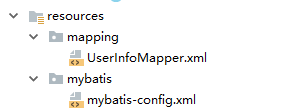
配置Mybatis config



*<?***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**>  
  
 <**settings**>  
 <**setting name="cacheEnabled" value="true"** />*<!-- 二级缓存开启 -->* </**settings**>  
  
</**configuration**>



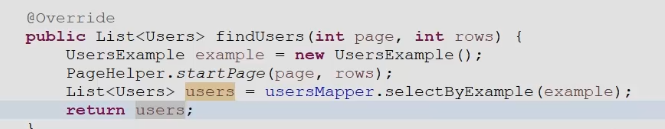


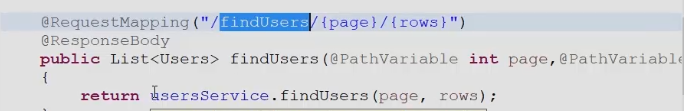


配置application.yml

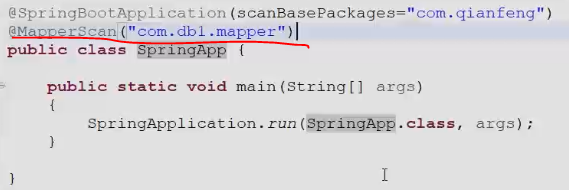
**spring:  
 datasource:  
 name:** test  
 **url:** jdbc:mysql://192.168.1.151/cloudDB01  
 **username:** root  
 **password:** Citi2013  
 **type:** com.alibaba.druid.pool.DruidDataSource  
 **driver-class-name:** com.mysql.jdbc.Driver  
 **filters:** stat  
 **maxActive:** 20  
 **initialSize:** 1  
 **maxWait:** 60000  
 **minIdle:** 1  
 **timeBetweenEvictionRunsMillis:** 60000  
 **minEvictableIdleTimeMillis:** 300000  
 **validationQuery:** select 1  
 **testWhileIdele:** true  
 **testOnBorrow:** false  
 **testOnReturn:** false  
 **poolPreparedStatements:** true  
 **maxOpenPreparedStatements:** 20  
  
**mybatis:  
 mapper-locations:** classpath:mapping/\*\*/\*.xml  
 **config-location:** classpth:mybatis/-config.xml  
 *# type-aliases-package: com.sky.pojo***pagehelper:  
 helper-dialect:** mysql  
 **reasonable:** true  
 **support-methods-arguments:** true  
 **params:** count=countSql

PageHelper 使用

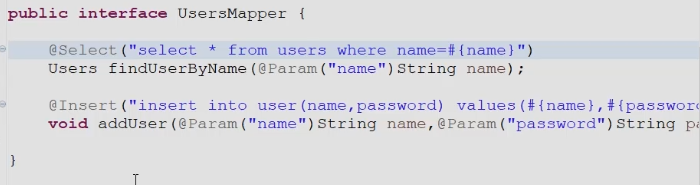


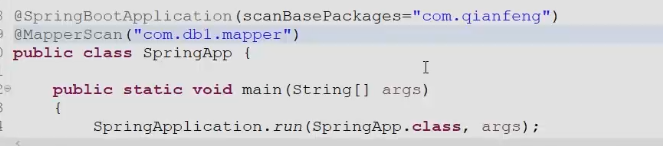


如果没有添加@Mapper，则要扫描

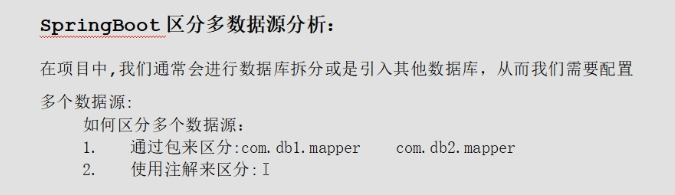


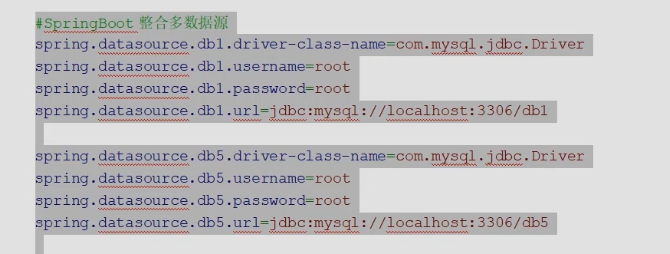
* **整合MyBatis（注解方式）:**



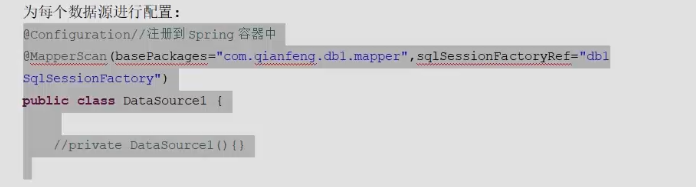


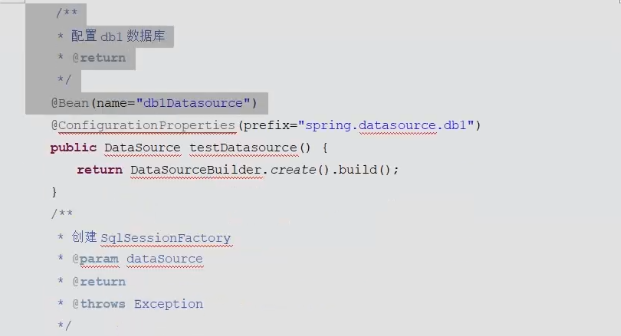
* **区分多数据源 :**

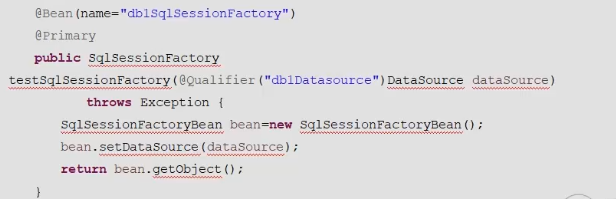




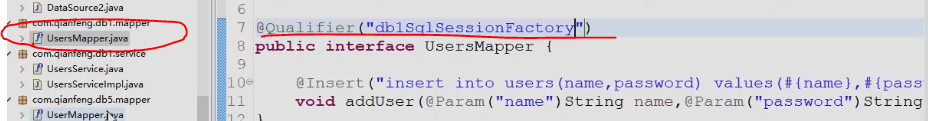




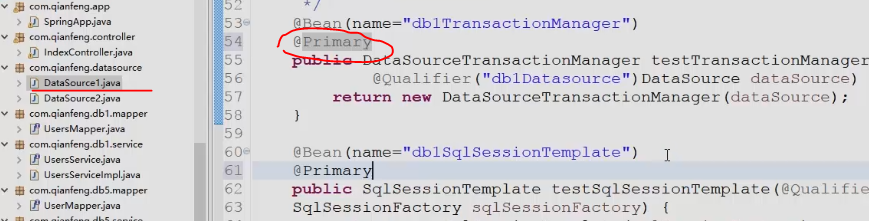




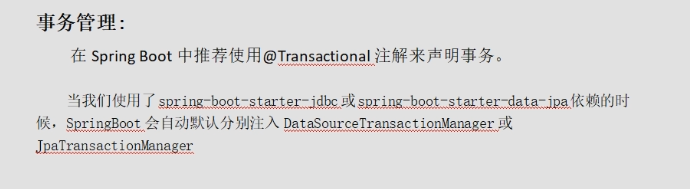




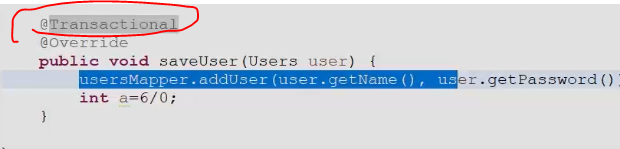
需要指定默认数据源，每个方法上都增加



* **事务管理 :**

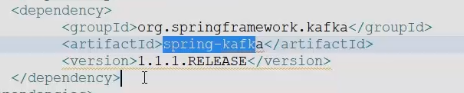


在方法上添加@Transactional, 增加事务管理

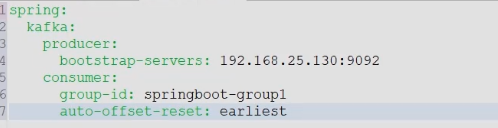


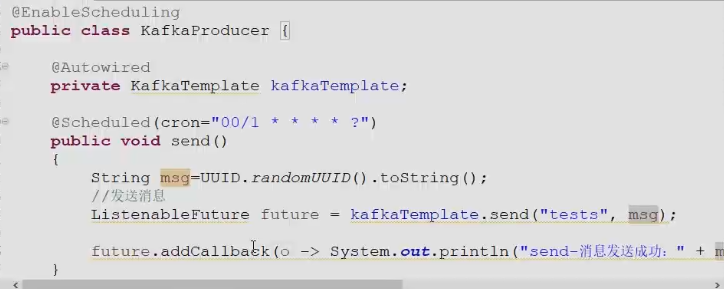
* **整合Kafka :**

添加依赖



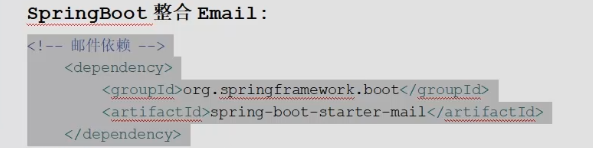
application.yml





* **整合Email :**

添加依赖



application.properties



