**品优购电商系统开发**

**第14章**

**SpringBoot框架与短信解决方案**

传智播客.黑马程序员

# 课程目标

目标1：掌握Spring Boot框架的搭建方法

目标2：能够使用阿里大于发送短信

目标3：运用SpringBoot、阿里大于和ActiveMQ 开发短信微服务

目标4：完成品优购用户注册功能（短信验证码认证）

# 1.Spring Boot入门

## 1.1什么是Spring Boot

Spring 诞生时是 Java 企业版（Java Enterprise Edition，JEE，也称 J2EE）的轻量级代替品。无需开发重量级的 Enterprise JavaBean（EJB），Spring 为企业级Java 开发提供了一种相对简单的方法，通过依赖注入和面向切面编程，用简单的Java 对象（Plain Old Java Object，POJO）实现了 EJB 的功能。

虽然 Spring 的组件代码是轻量级的，但它的配置却是重量级的。一开始，Spring 用 XML 配置，而且是很多 XML 配置。Spring 2.5 引入了基于注解的组件扫描，这消除了大量针对应用程序自身组件的显式 XML 配置。Spring 3.0 引入了基于 Java 的配置，这是一种类型安全的可重构配置方式，可以代替 XML。所有这些配置都代表了开发时的损耗。因为在思考 Spring 特性配置和解决业务问题之间需要进行思维切换，所以写配置挤占了写应用程序逻辑的时间。和所有框架一样，Spring 实用，但与此同时它要求的回报也不少。

除此之外，项目的依赖管理也是件吃力不讨好的事情。决定项目里要用哪些库就已经够让人头痛的了，你还要知道这些库的哪个版本和其他库不会有冲突，这难题实在太棘手。并且，依赖管理也是一种损耗，添加依赖不是写应用程序代码。一旦选错了依赖的版本，随之而来的不兼容问题毫无疑问会是生产力杀手。

Spring Boot 让这一切成为了过去。

Spring Boot 是 Spring 社区较新的一个项目。该项目的目的是帮助开发者更容易的创建基于 Spring 的应用程序和服务，让更多人的人更快的对 Spring 进行入门体验，为 Spring 生态系统提供了一种固定的、约定优于配置风格的框架。

Spring Boot 具有如下特性：

（1）为基于 Spring 的开发提供更快的入门体验

（2）开箱即用，没有代码生成，也无需 XML 配置。同时也可以修改默认值来满足特定的需求。

（3）提供了一些大型项目中常见的非功能性特性，如嵌入式服务器、安全、指标，健康检测、外部配置等。

（4）Spring Boot 并不是不对 Spring 功能上的增强，而是提供了一种快速使用 Spring 的方式。

## 1.2 Spring Boot入门小Demo

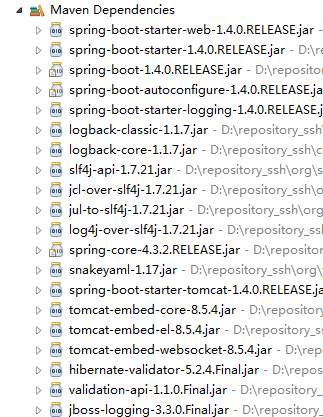
### 1.2.1起步依赖

创建Maven工程 springboot\_demo（打包方式jar）

在pom.xml中添加如下依赖

|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.5.8.RELEASE</version>  </parent>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  </dependencies> |

我们会惊奇地发现，我们的工程自动添加了好多好多jar包



........

而这些jar包正式我们做开发时需要导入的jar包。因为这些jar包被我们刚才引入的spring-boot-starter-web所引用了，所以我们引用spring-boot-starter-web后会自动把依赖传递过来。

### 1.2.2变更JDK版本

|  |
| --- |
| <properties>  <java.version>1.8</java.version>  </properties> |

添加后更新工程，会发现版本已经变更为1.8

### 1.2.3引导类

只需要创建一个引导类 .

|  |
| --- |
| **package** cn.itcast.demo;  **import** org.springframework.boot.SpringApplication;  **import** org.springframework.boot.autoconfigure.SpringBootApplication;  @SpringBootApplication  **publicclass** Application {  **publicstaticvoid** main(String[] args) {  SpringApplication.*run*(Application.**class**, args);  }  } |

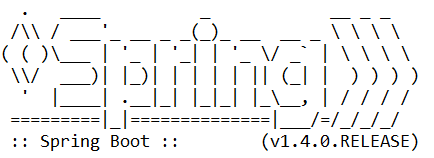
简单解释一下，@SpringBootApplication其实就是以下三个注解的总和

@Configuration： 用于定义一个配置类

@EnableAutoConfiguration ：Spring Boot会自动根据你jar包的依赖来自动配置项目。

@ComponentScan：告诉Spring 哪个packages 的用注解标识的类 会被spring自动扫描并且装入bean容器。

我们直接执行这个引导类，会发现控制台出现的这个标识



### 1.2.4 Spring MVC实现Hello World输出

我们现在开始使用spring MVC框架，实现json数据的输出。如果按照我们原来的做法，需要在web.xml中添加一个DispatcherServlet的配置，再添加一个spring的配置文件，配置文件中需要添加如下配置

|  |
| --- |
| <!-- 使用组件扫描，不用将controller在spring中配置 -->  <context:component-scan base-package="cn.itcast.demo.controller" />  <!-- 使用注解驱动不用在下边定义映射器和适配置器 -->  <mvc:annotation-driven>  <mvc:message-converters register-defaults=*"true"*>  <bean class=*"com.alibaba.fastjson.support.spring.FastJsonHttpMessageConverter"*>  <property name=*"supportedMediaTypes"* value=*"application/json"*/>  <property name=*"features"*>  <array>  <value>WriteMapNullValue</value>  <value>WriteDateUseDateFormat</value>  </array>  </property>  </bean>  </mvc:message-converters>  </mvc:annotation-driven> |

但是我们用SpringBoot，这一切都省了。我们直接写Controller类

|  |
| --- |
| **package** cn.itcast.demo.controller;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RestController;  @RestController  **publicclass** HelloWorldController {  @RequestMapping("/info")  **public** String info(){  **return**"HelloWorld";  }  } |

我们运行启动类来运行程序

在浏览器地址栏输入 http://localhost:8080/info 即可看到运行结果

### 1.2.5修改tomcat启动端口

在src/main/resources下创建application.properties

|  |
| --- |
| server.port=8088 |

重新运行引导类。地址栏输入

<http://localhost:8088/info>

### 1.2.6读取配置文件信息

在src/main/resources下的application.properties 增加配置

|  |
| --- |
| url=http://www.itcast.cn |

我要在类中读取这个配置信息，修改HelloWorldController

|  |
| --- |
| @Autowired  **private** Environment env;  @RequestMapping("/info")  **public** String info(){  **return**"HelloWorld~~"+env.getProperty("url");  } |

### 1.2.7热部署

我们在开发中反复修改类、页面等资源，每次修改后都是需要重新启动才生效，这样每次启动都很麻烦，浪费了大量的时间，能不能在我修改代码后不重启就能生效呢？可以，在pom.xml中添加如下配置就可以实现这样的功能，我们称之为热部署。

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

## 1.3 Spring Boot与ActiveMQ整合

### 1.3.1使用内嵌MQ服务

（1）在pom.xml中引入ActiveMQ起步依赖

|  |
| --- |
| <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-activemq</artifactId>  </dependency> |

（2）创建消息生产者

|  |
| --- |
| /\*\*  \* 消息生产者  \* **@author** Administrator  \*/  @RestController  **publicclass** QueueController {  @Autowired  **private** JmsMessagingTemplate jmsMessagingTemplate;  @RequestMapping("/send")  **public void** send(String text){  jmsMessagingTemplate.convertAndSend("itcast", text);  }  } |

（3）创建消息消费者

|  |
| --- |
| @Component  **Public class** Consumer {  @JmsListener(destination="itcast")  **publicvoid** readMessage(String text){  System.***out***.println("接收到消息："+text);  }  } |

测试：启动服务后，在浏览器执行

[http://localhost:8088/send](http://localhost:8080/send).do?text=aaaaa

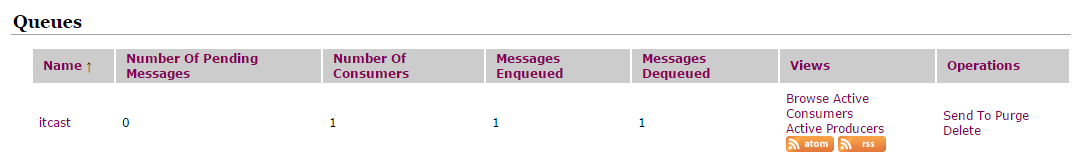
即可看到控制台输出消息提示。Spring Boot内置了ActiveMQ的服务，所以我们不用单独启动也可以执行应用程序。

### 1.3.2使用自定义MQ服务

在src/main/resources下的application.properties增加配置, 指定ActiveMQ的地址

|  |
| --- |
| spring.activemq.broker-url=tcp://192.168.25.135:61616 |

运行后，会在activeMQ中看到发送的queue



### 1.3.3发送Map信息

（1）修改QueueController.java

|  |
| --- |
| @RequestMapping("/sendmap")  **publicvoid** sendMap(){  Mapmap=**new** HashMap<>();  map.put("mobile", "13900001111");  map.put("content", "恭喜获得10元代金券");  jmsMessagingTemplate.convertAndSend("itcast\_map",map);  } |

（2）修改Consumer.java

|  |
| --- |
| @JmsListener(destination="itcast\_map")  **publicvoid** readMap(Map map){  System.***out***.println(map);  } |

# 2.短信发送平台-阿里大于

## 2.1阿里大于简介

阿里大于是**阿里云**旗下产品，融合了三大运营商的通信能力，通过将传统通信业务和能力与互联网相结合，创新融合阿里巴巴生态内容，全力为中小企业和开发者提供优质服务阿里大于提供包括短信、语音、流量直充、私密专线、店铺手机号等个性化服务。通过阿里大于打通三大运营商通信能力，全面融合阿里巴巴生态，以开放API及SDK的方式向开发者提供通信和数据服务，更好地支撑企业业务发展和创新服务。

## 2.2准备工作

### 2.2.1注册账户

首先我们先进入“阿里大于”[www.alidayu.com](http://www.baidu.com/baidu.php?url=ZzmK000GQJC8hW1POcJEP4KEa9MqOc64Kv9VaSYkPBGRtKKgrM9zjpMjkg0QSyMnQxKRQWKjPMwVPRtuZHdquplwvyTojNksmJ4fXyH6PR2oD2XMXttLlESx3RspZ_3aMmqjEv4UOpBl4_suPNxrqoJgxNO3dD62O8C6tR-EGupmJLpXA6.7D_iuTZ-GYkYApacrSapBsAprFMw6wKL9JuBBHwmEukmr_X5is1gsdeEQx3txQ_erMktJOAEWkl_LT8FB8zzEWgEvNSQQPLuy--rexduYt8FWojG4THZgZG4THZy7MHWuxJB4THAe_tpqOudEmtxZ_L2N9h9mLU2Srqf.U1Yk0ZDq_2Q0eeH1YSn0IjvsC_jEV55H3sKGUHYznWR0u1dBuHms0ZNG5yF9pywd0ZKGujYk0APGujY1rjR0UgfqnH0YP7tknjDLg1DsnH-xn1msnfKopHYs0ZFY5Hm4P0K-pyfqnHfdndtznHDzPdtkP10snNtznW01PNtznWDLPdtznWDdPsKBpHYznjf0UynqnHR4PWRzPWbdP-t1PWn4rHmknj9xn1mYPjbsnHDYg100TgKGujYs0Z7Wpyfqn0KzuLw9u1Ys0AqvUjYzPWm4Qywlg1cvPW-mQywlg1msn1nVn7t3PHbVnsK9mWYsg100ugFM5H00TZ0qn0K8IM0qna3snj0snj0sn0KVIZ0qn0KbuAqs5H00ThCqn0KbugmqTAn0uMfqn0KspjYs0Aq15H00mMTqnH00UMfqn0K1XWY0IZN15HDLPjnzPjnzP1b4rjnzP1RvPWD10ZF-TgfqnHf3rjcYPjDLPjmvP6K1pyfqmyPbryNBmH6snjK9ujRdnsKWTvYqrjuAwjNarDRdnH03rH7aPsK9m1Yk0ZK85H00TydY5H00Tyd15H00XMfqn0KVmdqhThqV5HDsPjuxn7tsg100uA78IyF-gLK_my4GuZnqn7tsg1Kxn0KbIA-b5H00ugwGujYVnfK9TLKWm1Ys0ZNspy4Wm1Ys0Z7VuWYs0AuWIgfqn0KhXh6qn0KlTAkdT1Ys0A7buhk9u1Yk0APzm1YznWnz&us=0.0.0.0.0.0.10&us=0.0.0.0.0.0.30&ck=0.0.0.0.0.0.0.0&shh=www.baidu.com) （https://dayu.aliyun.com/）







注册账号后，再在手机下载“阿里云”应用，登陆，然后进行在线实名认证。

### 2.2.2登陆系统

使用刚才注册的账号进行登陆。





点击进入控制台



点击使用**短信服务**



### 3.2.3申请签名







### 3.2.4申请模板





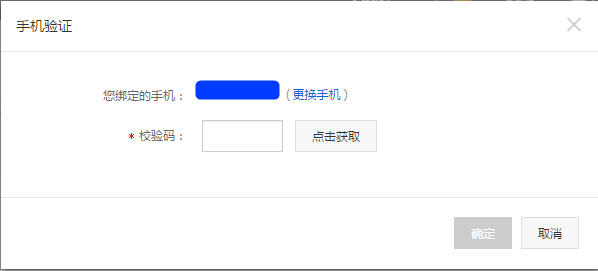
### 3.2.5创建accessKey(秘钥)













## 3.3 SDK安装

从阿里云通信官网上下载Demo工程



红线框起来的两个工程就是阿里云通信的依赖jar源码，我们将其安装到本地仓库，并删除对应报错的测试类。

备注：当然也可以直接从maven中央仓库下载即可：

在alicom-dysms-api所在的工程中，添加如下依赖：

|  |
| --- |
| <dependency>  <groupId>com.aliyun</groupId>  <artifactId>aliyun-java-sdk-core</artifactId>  <version>3.2.8</version>  </dependency>  <dependency>  <groupId>com.aliyun</groupId>  <artifactId>aliyun-java-sdk-dysmsapi</artifactId>  <version>1.1.0</version>  </dependency> |

## 3.4发送短信测试

1.加入依赖

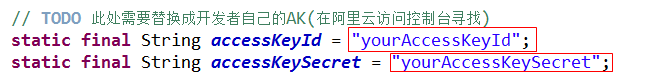
2.下载demo

3.类copy过来

4.发送（需要改所对应的属性）

（1）打开SmsDemo

替换下列几处代码



这个accessKeyId和accessSecret到刚才申请过的

手机号，短信签名和模板号



模板参数



code是我们申请模板时写的参数

执行main方法我们就可以在手机收到短信啦



https://help.aliyun.com/document\_detail/55284.html?spm=5176.10629532.106.1.73b91cbe5GUILn

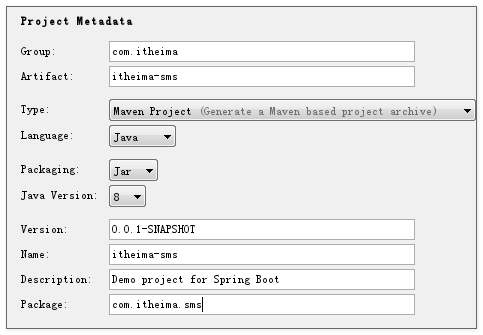
# 3.短信微服务

## 3.1需求分析

构建一个通用的短信发送服务（独立于品优购的单独工程），接收activeMQ的消息（MAP类型） 消息包括手机号（mobile）、短信模板号（template\_code）、签名（sign\_name）、参数字符串（param ）

## 3.2代码实现

### 3.2.1工程搭建



（1）创建工程item-sms （JAR工程），POM文件引入依赖

|  |
| --- |
| *<?***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.itheima</**groupId**>  <**artifactId**>itheima-sms</**artifactId**>  <**version**>0.0.1-SNAPSHOT</**version**>  <**packaging**>jar</**packaging**>   <**name**>itheima-sms</**name**>  <**description**>Demo project for Spring Boot</**description**>   <**parent**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-parent</**artifactId**>  <**version**>1.5.8.RELEASE</**version**>  <**relativePath**/> *<!-- lookup parent from repository -->* </**parent**>   <**properties**>  <**maven.compiler.source**>1.8</**maven.compiler.source**>  <**maven.compiler.target**>1.8</**maven.compiler.target**>  <**project.build.sourceEncoding**>UTF-8</**project.build.sourceEncoding**>  <**project.reporting.outputEncoding**>UTF-8</**project.reporting.outputEncoding**>  <**java.version**>1.8</**java.version**>  </**properties**>   <**dependencies**>  <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-activemq</**artifactId**>  </**dependency**>   <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-test</**artifactId**>  <**scope**>test</**scope**>  </**dependency**>  <**dependency**>  <**groupId**>com.aliyun</**groupId**>  <**artifactId**>aliyun-java-sdk-dysmsapi</**artifactId**>  <**version**>1.0.0</**version**>  </**dependency**>  <**dependency**>  <**groupId**>com.aliyun</**groupId**>  <**artifactId**>aliyun-java-sdk-core</**artifactId**>  <**version**>3.2.8</**version**>  </**dependency**>  </**dependencies**>   <**build**>  <**plugins**>  <**plugin**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-maven-plugin</**artifactId**>  </**plugin**>  </**plugins**>  </**build**>   </**project**> |

（2）创建引导类

|  |
| --- |
| **package** cn.itcast.sms;  **import** org.springframework.boot.SpringApplication;  **import** org.springframework.boot.autoconfigure.SpringBootApplication;  @SpringBootApplication  **publicclass** Application {  **publicstaticvoid** main(String[] args) {  SpringApplication.*run*(Application.**class**, args);  }  } |

（3）创建配置文件application.properties

|  |
| --- |
| server.port=9003  spring.activemq.broker-url=tcp://192.168.25.133:61616  accessKeyId=不告诉你  accessKeySecret=不告诉你 |

### 3.2.2短信工具类

参照之前的短信demo创建短信工具类

|  |
| --- |
| **package** com.itheima.sms.util;  **import** com.aliyuncs.DefaultAcsClient; **import** com.aliyuncs.IAcsClient; **import** com.aliyuncs.dysmsapi.model.v20170525.QuerySendDetailsRequest; **import** com.aliyuncs.dysmsapi.model.v20170525.QuerySendDetailsResponse; **import** com.aliyuncs.dysmsapi.model.v20170525.SendSmsRequest; **import** com.aliyuncs.dysmsapi.model.v20170525.SendSmsResponse; **import** com.aliyuncs.exceptions.ClientException; **import** com.aliyuncs.profile.DefaultProfile; **import** com.aliyuncs.profile.IClientProfile; **import** org.springframework.beans.factory.annotation.Autowired; **import** org.springframework.core.env.Environment; **import** org.springframework.stereotype.Component;  **import** java.text.SimpleDateFormat; **import** java.util.Date;  /\*\*  \* **@author** 三国的包子  \* **@version** 1.0  \* **@description** 描述  \* **@title** 标题  \* **@package** com.itheima.sms.util  \* **@company** www.itheima.com  \*/  @Component **public class** SmsUtil {   *//产品名称:云通信短信API产品,开发者无需替换* **static final** String ***product*** = **"Dysmsapi"**;  *//产品域名,开发者无需替换* **static final** String ***domain*** = **"dysmsapi.aliyuncs.com"**;   *//* ***TODO 此处需要替换成开发者自己的AK(在阿里云访问控制台寻找)*** *//static final String accessKeyId = "LTAIoVgq10jSIf6s";  //static final String accessKeySecret = "u1d9fvAnhEOtBuFpiyfdoboVgS3GV6";* @Autowired  **private** Environment **env**;     /\*\*  \* 发送短信  \* **@param *mobile*** 电话号码  \* **@param *template\_code*** 模板编码  \* **@param *sign\_name*** 签名  \* **@param *param*** 参数  \* **@return** \* **@throws** ClientException  \*/  **public** SendSmsResponse sendSms(String mobile,String template\_code,String sign\_name,String param) **throws** ClientException {   *//可自助调整超时时间* System.*setProperty*(**"sun.net.client.defaultConnectTimeout"**, **"10000"**);  System.*setProperty*(**"sun.net.client.defaultReadTimeout"**, **"10000"**);   *//初始化acsClient,暂不支持region化* String accessKeyId = **env**.getProperty(**"accessKeyId"**);  String accessKeySecret = **env**.getProperty(**"accessKeySecret"**);  IClientProfile profile = DefaultProfile.*getProfile*(**"cn-hangzhou"**, accessKeyId, accessKeySecret);  DefaultProfile.*addEndpoint*(**"cn-hangzhou"**, **"cn-hangzhou"**, ***product***, ***domain***);  IAcsClient acsClient = **new** DefaultAcsClient(profile);   *//组装请求对象-具体描述见控制台-文档部分内容* SendSmsRequest request = **new** SendSmsRequest();  *//必填:待发送手机号* request.setPhoneNumbers(mobile);  *//必填:短信签名-可在短信控制台中找到* request.setSignName(sign\_name);  *//必填:短信模板-可在短信控制台中找到* request.setTemplateCode(template\_code);  *//可选:模板中的变量替换JSON串,如模板内容为"亲爱的${name},您的验证码为${code}"时,此处的值为* request.setTemplateParam(param);*//"{\"code\":\"123456\"}"   //选填-上行短信扩展码(无特殊需求用户请忽略此字段)  //request.setSmsUpExtendCode("90997");   //可选:outId为提供给业务方扩展字段,最终在短信回执消息中将此值带回给调用者* request.setOutId(**"yourOutId"**);   *//hint 此处可能会抛出异常，注意catch* SendSmsResponse sendSmsResponse = acsClient.getAcsResponse(request);   **return** sendSmsResponse;  }    *//查询相关的发送的短信信息* **public** QuerySendDetailsResponse querySendDetails(String mobile,String bizId) **throws** ClientException {   *//可自助调整超时时间* System.*setProperty*(**"sun.net.client.defaultConnectTimeout"**, **"10000"**);  System.*setProperty*(**"sun.net.client.defaultReadTimeout"**, **"10000"**);  String accessKeyId = **env**.getProperty(**"accessKeyId"**);  String accessKeySecret = **env**.getProperty(**"accessKeySecret"**);   *//初始化acsClient,暂不支持region化* IClientProfile profile = DefaultProfile.*getProfile*(**"cn-hangzhou"**, accessKeyId, accessKeySecret);  DefaultProfile.*addEndpoint*(**"cn-hangzhou"**, **"cn-hangzhou"**, ***product***, ***domain***);  IAcsClient acsClient = **new** DefaultAcsClient(profile);   *//组装请求对象* QuerySendDetailsRequest request = **new** QuerySendDetailsRequest();  *//必填-号码* request.setPhoneNumber(mobile);  *//可选-流水号* request.setBizId(bizId);  *//必填-发送日期 支持30天内记录查询，格式yyyyMMdd* SimpleDateFormat ft = **new** SimpleDateFormat(**"yyyyMMdd"**);  request.setSendDate(ft.format(**new** Date()));  *//必填-页大小* request.setPageSize(10L);  *//必填-当前页码从1开始计数* request.setCurrentPage(1L);   *//hint 此处可能会抛出异常，注意catch* QuerySendDetailsResponse querySendDetailsResponse = acsClient.getAcsResponse(request);   **return** querySendDetailsResponse;  }  *// public static void main(String[] args) throws ClientException, InterruptedException { // // //发短信 // SendSmsResponse response = sendSms(); // System.out.println("短信接口返回的数据----------------"); // System.out.println("Code=" + response.getCode()); // System.out.println("Message=" + response.getMessage()); // System.out.println("RequestId=" + response.getRequestId()); // System.out.println("BizId=" + response.getBizId()); // // Thread.sleep(3000L); // // //查明细 // if (response.getCode() != null && response.getCode().equals("OK")) { // QuerySendDetailsResponse querySendDetailsResponse = querySendDetails(response.getBizId()); // System.out.println("短信明细查询接口返回数据----------------"); // System.out.println("Code=" + querySendDetailsResponse.getCode()); // System.out.println("Message=" + querySendDetailsResponse.getMessage()); // int i = 0; // for (QuerySendDetailsResponse.SmsSendDetailDTO smsSendDetailDTO : querySendDetailsResponse.getSmsSendDetailDTOs()) { // System.out.println("SmsSendDetailDTO[" + i + "]:"); // System.out.println("Content=" + smsSendDetailDTO.getContent()); // System.out.println("ErrCode=" + smsSendDetailDTO.getErrCode()); // System.out.println("OutId=" + smsSendDetailDTO.getOutId()); // System.out.println("PhoneNum=" + smsSendDetailDTO.getPhoneNum()); // System.out.println("ReceiveDate=" + smsSendDetailDTO.getReceiveDate()); // System.out.println("SendDate=" + smsSendDetailDTO.getSendDate()); // System.out.println("SendStatus=" + smsSendDetailDTO.getSendStatus()); // System.out.println("Template=" + smsSendDetailDTO.getTemplateCode()); // } // System.out.println("TotalCount=" + querySendDetailsResponse.getTotalCount()); // System.out.println("RequestId=" + querySendDetailsResponse.getRequestId()); // } // // }* } |

### 3.2.3消息监听类

创建SmsListener.java

|  |
| --- |
| **package** com.itheima.sms.listener;  **import** com.aliyuncs.dysmsapi.model.v20170525.SendSmsResponse; **import** com.aliyuncs.exceptions.ClientException; **import** com.itheima.sms.util.SmsUtil; **import** org.springframework.beans.factory.annotation.Autowired; **import** org.springframework.jms.annotation.JmsListener; **import** org.springframework.stereotype.Component;  **import** java.util.Map;  /\*\*  \* **@author** 三国的包子  \* **@version** 1.0  \* **@description** 描述  \* **@title** 标题  \* **@package** com.itheima.listener  \* **@company** www.itheima.com  \*/ @Component **public class** SmsListener {   @Autowired  **private** SmsUtil **smsUtil**;   *//用于监听消息* @JmsListener(destination = **"sms"**)  **public void** reieve(Map<String,String> map) **throws** ClientException {  SendSmsResponse response = **smsUtil**.sendSms(map.get(**"mobile"**), map.get(**"template\_code"**), map.get(**"sign\_name"**), map.get(**"param"**));  System.***out***.println(**"Code="** + response.getCode());  System.***out***.println(**"Message="** + response.getMessage());  System.***out***.println(**"RequestId="** + response.getRequestId());  System.***out***.println(**"BizId="** + response.getBizId());  }  } |

## 3.3代码测试

修改springboot-demo 的QueueController.java

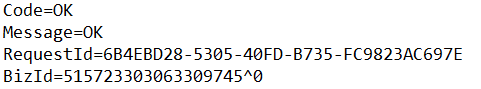
|  |
| --- |
| @RequestMapping("/sendsms")  **publicvoid** sendSms(){  Mapmap=**new** HashMap<>();  map.put("mobile", "13900001111");  map.put("template\_code", "SMS\_85735065");  map.put("sign\_name", "黑马");  map.put("param", "{\"code\":\"102931\"}");  jmsMessagingTemplate.convertAndSend("sms",map);  } |

启动itcast\_sms

启动springboot-demo

地址栏输入：<http://localhost:8088/sendsms.do>

观察控制台输出



随后短信也成功发送到你的手机上

# 4.用户注册

## 4.1需求分析

完成用户注册功能



## 4.2工程搭建

### 4.2.1用户服务接口层

（1）创建pinyougou-user-interface（jar）

（2）引入pojo依赖

### 4.2.2用户服务实现层

（1）创建pinyougou-user-service（war）

（2）引入spring dubbox activeMQ相关依赖，引入依赖（ pinyougou-user-interfacepinyougou-dao pinyougou-common），运行端口为9006

（3）添加web.xml

（4）创建Spring 配置文件applicationContext-service.xml 和applicationContent-tx.xml

|  |
| --- |
| <dubbo:protocolname=*"dubbo"*port=*"20886"*/>  <dubbo:annotationpackage=*"com.pinyougou.user.service.impl"*/>  <dubbo:applicationname=*"pinyougou-user-service"*/>  <dubbo:registryaddress=*"zookeeper://192.168.25.135:2181"*/> |

### 4.2.3用户中心WEB层

创建war工程 pinyougou-user-web 我们将注册功能放入此工程

（1）添加web.xml

（2）引入依赖pinyougou-user-interface 、spring相关依赖（参照其它web工程）,tomcat运行端口9106

（3）添加spring配置文件

（4）拷贝静态原型页面register.html 及相关资源

## 4.3基本注册功能实现

### 4.3.1生成和拷贝代码

使用代码生成器生成JS生成后台代码。

### 4.3.2后端服务实现层

修改pinyougou-user-service的UserServiceImpl.java

|  |
| --- |
| /\*\*  \* 增加  \*/  @Override  **public void** add(TbUser user) {  user.setCreated(**new** Date());//创建日期  user.setUpdated(**new** Date());//修改日期  String password = DigestUtils.*md5Hex*(user.getPassword());//对密码加密  user.setPassword(password);  userMapper.insert(user);  } |

### 4.3.3前端控制层

修改userController.js

|  |
| --- |
| //控制层  app.controller('userController' ,**function**($scope,$controller ,userService){  //注册  $scope.reg=**function**(){  **if**($scope.entity.password!=$scope.password) {  alert("两次输入的密码不一致，请重新输入");  **return** ;  }  userService.add( $scope.entity ).success(  **function**(response){  alert(response.message);  }  );  }  }); |

### 4.3.4修改页面

修改页面register.html ，引入js

|  |
| --- |
| <scripttype=*"text/javascript"*src=*"plugins/angularjs/angular.min.js"*></script>  <scripttype=*"text/javascript"*src=*"js/base.js"*></script>  <scripttype=*"text/javascript"*src=*"js/service/userService.js"*></script>  <scripttype=*"text/javascript"*src=*"js/controller/userController.js"*></script> |

指令

|  |
| --- |
| <body ng-app=*"pinyougou"* ng-controller=*"userController"*> |

绑定表单

|  |
| --- |
| <form class=*"sui-form form-horizontal"*>  <div class=*"control-group"*>  <label class=*"control-label"*>用户名：</label>  <div class=*"controls"*>  <input type=*"text"* placeholder=*"请输入你的用户名"*ng-model=*"entity.username"* class=*"input-xfat input-xlarge"*>  </div>  </div>  <div class=*"control-group"*>  <label for=*"inputPassword"* class=*"control-label"*>登录密码：</label>  <div class=*"controls"*>  <input type=*"password"* placeholder=*"设置登录密码"*ng-model=*"entity.password"* class=*"input-xfat input-xlarge"*>  </div>  </div>  <div class=*"control-group"*>  <label for=*"inputPassword"* class=*"control-label"*>确认密码：</label>  <div class=*"controls"*>  <input type=*"password"* placeholder=*"再次确认密码"*ng-model=*"password"*class=*"input-xfat input-xlarge"*>  </div>  </div>  <div class=*"control-group"*>  <label class=*"control-label"*>手机号：</label>  <div class=*"controls"*>  <input type=*"text"* placeholder=*"请输入你的手机号"*ng-model=*"entity.phone"* class=*"input-xfat input-xlarge"*>  </div>  </div>  <div class=*"control-group"*>  <label for=*"inputPassword"* class=*"control-label"*>短信验证码：</label>  <div class=*"controls"*>  <input type=*"text"* placeholder=*"短信验证码"*class=*"input-xfat input-xlarge"*><a href=*"#"*>获取短信验证码</a>  </div>  </div>  <div class=*"control-group"*>  <label for=*"inputPassword"* class=*"control-label"*>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;</label>  <div class=*"controls"*>  <input name=*"m1"* type=*"checkbox"* value=*"2"* checked=*""*><span>同意协议并注册《品优购用户协议》</span>  </div>  </div>  <div class=*"control-group"*>  <label class=*"control-label"*></label>  <div class=*"controls btn-reg"*>  <a class=*"sui-btn btn-block btn-xlarge btn-danger"*ng-click=*"reg()"* target=*"\_blank"*>完成注册</a>  </div>  </div>  </form> |

## 4.4 注册判断短信验证码

### 4.4.1实现思路

生成验证码：点击页面上的”获取短信验证码”连接，向后端传递手机号。后端随机生成6位数字作为短信验证码，将其保存在redis中（手机号作为KEY），并发送到短信网关。

校验验证码：用户注册时，后端根据手机号查询redis中的验证码与用户填写的验证码是否相同，如果不同则提示用户不能注册。

### 4.4.2生成验证码

（1）修改pinyougou-user-interface工程UserService.java，增加方法

|  |
| --- |
| /\*\*  \* 生成短信验证码  \* **@return**  \*/  **public void** createSmsCode(String phone); |

（2）修改pinyougou-user-service工程的UserServiceImpl.java

|  |
| --- |
| @Autowired  **private** RedisTemplate<String , Object> redisTemplate;  /\*\*  \* 生成短信验证码  \*/  **public void** createSmsCode(String phone){  //生成6位随机数  String code = (**long**) (Math.*random*()\*1000000)+"";  System.***out***.println("验证码："+code);  //存入缓存  redisTemplate.boundHashOps("smscode").put(phone, code);  //发送到activeMQ ....  } |

（3）在 pinyougou-common 添加工具类PhoneFormatCheckUtils.java，用于验证手机号

（4）修改pinyougou-user-web的UserController.java

|  |
| --- |
| /\*\*  \* 发送短信验证码  \* **@param** phone  \* **@return**  \*/  @RequestMapping("/sendCode")  **public** Result sendCode(String phone){  //判断手机号格式  **if**(!PhoneFormatCheckUtils.*isPhoneLegal*(phone)){  **return new** Result(**false**, "手机号格式不正确");  }  **try** {  userService.createSmsCode(phone);//生成验证码  **return new** Result(**true**, "验证码发送成功");  } **catch** (Exception e) {  e.printStackTrace();  **return new** Result(**true**, "验证码发送失败");  }  } |

1. pinyougou-user-web的userService.js

|  |
| --- |
| //发送验证码  **this**.sendCode=**function**(phone){  **return** $http.get("../user/sendCode.do?phone="+phone);  } |

1. pinyougou-user-web的userController.js

|  |
| --- |
| //发送验证码  $scope.sendCode=**function**(){  **if**($scope.entity.phone==**null**){  alert("请输入手机号！");  **return** ;  }  userService.sendCode($scope.entity.phone).success(  **function**(response){  alert(response.message);  }  );  } |

1. 修改页面register.html

|  |
| --- |
| <ang-click=*"sendCode()"*>获取短信验证码</a> |

### 4.4.3用户注册判断验证码

（1）修改pinyougou-user-interface的UserService.java

|  |
| --- |
| /\*\*  \* 判断短信验证码是否存在  \* **@param** phone  \* **@return**  \*/  **publicboolean** checkSmsCode(String phone,String code); |

（2）修改pinyougou-user-service的 UserServiceImpl.java

|  |
| --- |
| /\*\*  \* 判断验证码是否正确  \*/  **publicboolean** checkSmsCode(String phone,String code){  //得到缓存中存储的验证码  String sysCode = (String) redisTemplate.boundHashOps("smscode").get(phone);  **if**(sysCode==**null**){  **return false**;  }  **if**(!sysCode.equals(code)){  **return false**;  }  **return true**;  } |

（3）修改pinyougou-user-web的UserController.java

|  |
| --- |
| /\*\*  \* 增加  \* **@param** user  \* **@return**  \*/  @RequestMapping("/add")  **public** Result add(@RequestBody TbUser user,String smscode){  **boolean** checkSmsCode = userService.checkSmsCode(user.getPhone(), smscode);  **if**(checkSmsCode==**false**){  **return new** Result(**false**, "验证码输入错误！");  }  **try** {  userService.add(user);  **returnnew** Result(**true**, "增加成功");  } **catch** (Exception e) {  e.printStackTrace();  **returnnew** Result(**false**, "增加失败");  }  } |

1. 修改pinyougou-user-web的userService.js

|  |
| --- |
| //增加  **this**.add=**function**(entity,smscode){  **return** $http.post('../user/add.do?smscode='+smscode ,entity );  } |

1. 修改pinyougou-portal-web 的UserController.java

|  |
| --- |
| //保存  $scope.reg=**function**(){  userService.add( $scope.entity, $scope.smscode ).success(  **function**(response){  alert(response.message);  }  );  } |

（6）修改页面，绑定变量

|  |
| --- |
| <input type=*"text"* placeholder=*"短信验证码"*ng-model=*"smscode"*class=*"input-xfat input-xlarge"*><a href=*"#"*ng-click=*"sendCode()"*>获取短信验证码</a> |

### 4.4.4短信验证码发送到手机

（1）在pinyougou-user-service添加配置文件applicationContext-activemq.xml

|  |
| --- |
| <!-- 真正可以产生Connection的ConnectionFactory，由对应的 JMS服务厂商提供-->  <beanid=*"targetConnectionFactory"*class=*"org.apache.activemq.ActiveMQConnectionFactory"*>  <propertyname=*"brokerURL"*value=*"tcp://192.168.25.135:61616"*/>  </bean>  <!-- Spring用于管理真正的ConnectionFactory的ConnectionFactory -->  <beanid=*"connectionFactory"*class=*"org.springframework.jms.connection.SingleConnectionFactory"*>  <!-- 目标ConnectionFactory对应真实的可以产生JMS Connection的ConnectionFactory -->  <propertyname=*"targetConnectionFactory"*ref=*"targetConnectionFactory"*/>  </bean>  <!-- Spring提供的JMS工具类，它可以进行消息发送、接收等 -->  <beanid=*"jmsTemplate"*class=*"org.springframework.jms.core.JmsTemplate"*>  <!-- 这个connectionFactory对应的是我们定义的Spring提供的那个ConnectionFactory对象 -->  <propertyname=*"connectionFactory"*ref=*"connectionFactory"*/>  </bean>  <!--这个是点对点消息 -->  <beanid=*"smsDestination"*class=*"org.apache.activemq.command.ActiveMQQueue"*>  <constructor-argvalue=*"sms"*/>  </bean> |

（2）修改pinyougou-user-service的UserServiceImpl.java

|  |
| --- |
| @Autowired  **private** JmsTemplate jmsTemplate;  @Autowired  **private** Destination smsDestination;  @Value("${template\_code}")  **private** String template\_code;    @Value("${sign\_name}")  **private** String sign\_name;  /\*\*  \* 生成短信验证码  \*/  **publicvoid** createSmsCode(**final**String phone){  //生成6位随机数  **final**String code = (**long**) (Math.*random*()\*1000000)+"";  System.***out***.println("验证码："+code);  //存入缓存  redisTemplate.boundHashOps("smscode").put(phone, code);  //发送到activeMQ  jmsTemplate.send(smsDestination, **new** MessageCreator() {  @Override  **public** Message createMessage(Session session) **throws** JMSException {  MapMessage mapMessage = session.createMapMessage();  mapMessage.setString("mobile", phone);//手机号  mapMessage.setString("template\_code", "SMS\_85735065");//模板编号  mapMessage.setString("sign\_name", "黑马");//签名  Map m=**new** HashMap<>();  m.put("code", code);  mapMessage.setString("param", JSON.*toJSONString*(m));//参数  **return** mapMessage;  }  });  } |

（3）在pinyougou-common的properties目录下创建配置文件sms.properties

|  |
| --- |
| template\_code=SMS\_85735065  sign\_name=\u9ED1\u9A6C |

## 附录A. Spring Boot应用启动器

[spring](http://lib.csdn.net/base/javaee) Boot应用启动器基本的一共有44种，具体如下：

1）spring-boot-starter   
这是Spring Boot的核心启动器，包含了自动配置、日志和YAML。

2）spring-boot-starter-actuator   
帮助监控和管理应用。

3）spring-boot-starter-amqp   
通过spring-rabbit来支持AMQP协议（Advanced Message Queuing Protocol）。

4）spring-boot-starter-aop   
支持面向方面的编程即AOP，包括spring-aop和AspectJ。

5）spring-boot-starter-artemis   
通过Apache Artemis支持JMS的API（[Java](http://lib.csdn.net/base/java)Message Service API）。

6）spring-boot-starter-batch   
支持Spring Batch，包括HSQLDB[数据库](http://lib.csdn.net/base/mysql)。

7）spring-boot-starter-cache   
支持Spring的Cache抽象。

8）spring-boot-starter-cloud-connectors   
支持Spring Cloud Connectors，简化了在像Cloud Foundry或Heroku这样的云平台上连接服务。

9）spring-boot-starter-data-elasticsearch   
支持ElasticSearch搜索和分析引擎，包括spring-data-elasticsearch。

10）spring-boot-starter-data-gemfire   
支持GemFire分布式数据存储，包括spring-data-gemfire。

11）spring-boot-starter-data-jpa   
支持JPA（[Java](http://lib.csdn.net/base/javase) Persistence API），包括spring-data-jpa、spring-orm、[hibernate](http://lib.csdn.net/base/javaee)。

12）spring-boot-starter-data-[MongoDB](http://lib.csdn.net/base/mongodb)   
支持MongoDB数据，包括spring-data-mongodb。

13）spring-boot-starter-data-rest   
通过spring-data-rest-webmvc，支持通过REST暴露Spring Data数据仓库。

14）spring-boot-starter-data-solr   
支持Apache Solr搜索平台，包括spring-data-solr。

15）spring-boot-starter-freemarker   
支持FreeMarker模板引擎。

16）spring-boot-starter-groovy-templates   
支持Groovy模板引擎。

17）spring-boot-starter-hateoas   
通过spring-hateoas支持基于HATEOAS的RESTful Web服务。

18）spring-boot-starter-hornetq   
通过HornetQ支持JMS。

19）spring-boot-starter-integration   
支持通用的spring-integration模块。

20）spring-boot-starter-jdbc   
支持JDBC数据库。

21）spring-boot-starter-jersey   
支持Jersey RESTful Web服务框架。

22）spring-boot-starter-jta-atomikos   
通过Atomikos支持JTA分布式事务处理。

23）spring-boot-starter-jta-bitronix   
通过Bitronix支持JTA分布式事务处理。

24）spring-boot-starter-mail   
支持javax.mail模块。

25）spring-boot-starter-mobile   
支持spring-mobile。

26）spring-boot-starter-mustache   
支持Mustache模板引擎。

27）spring-boot-starter-[Redis](http://lib.csdn.net/base/redis)   
支持Redis键值存储数据库，包括spring-redis。

28）spring-boot-starter-security   
支持spring-security。

29）spring-boot-starter-social-facebook   
支持spring-social-facebook

30）spring-boot-starter-social-linkedin   
支持pring-social-linkedin

31）spring-boot-starter-social-twitter   
支持pring-social-twitter

32）spring-boot-starter-test   
支持常规的[测试](http://lib.csdn.net/base/softwaretest)依赖，包括JUnit、Hamcrest、Mockito以及spring-test模块。

33）spring-boot-starter-thymeleaf   
支持Thymeleaf模板引擎，包括与Spring的集成。

34）spring-boot-starter-velocity   
支持Velocity模板引擎。

35）spring-boot-starter-web   
S支持全栈式Web开发，包括Tomcat和spring-webmvc。

36）spring-boot-starter-websocket   
支持WebSocket开发。

37）spring-boot-starter-ws   
支持Spring Web Services。

Spring Boot应用启动器面向生产环境的还有2种，具体如下：

1）spring-boot-starter-actuator   
增加了面向产品上线相关的功能，比如测量和监控。

2）spring-boot-starter-remote-shell   
增加了远程ssh shell的支持。

最后，Spring Boot应用启动器还有一些替换技术的启动器，具体如下：

1）spring-boot-starter-jetty   
引入了Jetty HTTP引擎（用于替换Tomcat）。

2）spring-boot-starter-log4j   
支持Log4J日志框架。

3）spring-boot-starter-logging   
引入了Spring Boot默认的日志框架Logback。

4）spring-boot-starter-tomcat   
引入了Spring Boot默认的HTTP引擎Tomcat。

5）spring-boot-starter-undertow   
引入了Undertow HTTP引擎（用于替换Tomcat）。

## 附录B. Spring Boot 配置文件application.properties

|  |
| --- |
| #########COMMON SPRING BOOT PROPERTIES  ######========CORE PROPERTIES===========  #SPRING CONFIG (ConfigFileApplicationListener)  spring.config.name= # config file name (default to 'application')  spring.config.location= # location of config file  #PROFILES  spring.profiles= # comma list of active profiles  #APPLICATION SETTINGS (SpringApplication)  spring.main.sources=  spring.main.web-environment= # detect by default  spring.main.show-banner=true  spring.main....= # see class for all properties  #LOGGING  logging.path=/var/logs  logging.file=myapp.log  logging.config=  #IDENTITY (ContextIdApplicationContextInitializer)  spring.application.name=  spring.application.index=  #EMBEDDED SERVER CONFIGURATION (ServerProperties)  server.port=8080  server.address= # bind to a specific NIC  server.session-timeout= # session timeout in seconds  server.context-path= # the context path, defaults to '/'  server.servlet-path= # the servlet path, defaults to '/'  server.tomcat.access-log-pattern= # log pattern of the access log  server.tomcat.access-log-enabled=false # is access logging enabled  server.tomcat.protocol-header=x-forwarded-proto # ssl forward headers  server.tomcat.remote-ip-header=x-forwarded-for  server.tomcat.basedir=/tmp # base dir (usually not needed, defaults to tmp)  server.tomcat.background-processor-delay=30; # in seconds  server.tomcat.max-threads = 0 # number of threads in protocol handler  server.tomcat.uri-encoding = UTF-8 # character encoding to use for URL decoding  #SPRING MVC (HttpMapperProperties)  http.mappers.json-pretty-print=false # pretty print JSON  http.mappers.json-sort-keys=false # sort keys  spring.mvc.locale= # set fixed locale, e.g. enUK  spring.mvc.date-format= # set fixed date format, e.g. dd/MM/yyyy  spring.mvc.message-codes-resolver-format= # PREFIXERRORCODE / POSTFIXERROR\_CODE  spring.view.prefix= # MVC view prefix  spring.view.suffix= # ... and suffix  spring.resources.cache-period= # cache timeouts in headers sent to browser  spring.resources.add-mappings=true # if default mappings should be added  #THYMELEAF (ThymeleafAutoConfiguration)  spring.thymeleaf.prefix=classpath:/templates/  spring.thymeleaf.suffix=.html  spring.thymeleaf.mode=HTML5  spring.thymeleaf.encoding=UTF-8  spring.thymeleaf.content-type=text/html # ;charset=<encoding> is added  spring.thymeleaf.cache=true # set to false for hot refresh  #FREEMARKER (FreeMarkerAutoConfiguration)  spring.freemarker.allowRequestOverride=false  spring.freemarker.allowSessionOverride=false  spring.freemarker.cache=true  spring.freemarker.checkTemplateLocation=true  spring.freemarker.contentType=text/html  spring.freemarker.exposeRequestAttributes=false  spring.freemarker.exposeSessionAttributes=false  spring.freemarker.exposeSpringMacroHelpers=false  spring.freemarker.prefix=  spring.freemarker.requestContextAttribute=  spring.freemarker.settings.\*=  spring.freemarker.suffix=.ftl  spring.freemarker.templateEncoding=UTF-8  spring.freemarker.templateLoaderPath=classpath:/templates/  spring.freemarker.viewNames= # whitelist of view names that can be resolved  #GROOVY TEMPLATES (GroovyTemplateAutoConfiguration)  spring.groovy.template.allowRequestOverride=false  spring.groovy.template.allowSessionOverride=false  spring.groovy.template.cache=true  spring.groovy.template.configuration.\*= # See Groovy's TemplateConfiguration  spring.groovy.template.contentType=text/html  spring.groovy.template.prefix=classpath:/templates/  spring.groovy.template.suffix=.tpl  spring.groovy.template.templateEncoding=UTF-8  spring.groovy.template.viewNames= # whitelist of view names that can be resolved  #VELOCITY TEMPLATES (VelocityAutoConfiguration)  spring.velocity.allowRequestOverride=false  spring.velocity.allowSessionOverride=false  spring.velocity.cache=true  spring.velocity.checkTemplateLocation=true  spring.velocity.contentType=text/html  spring.velocity.dateToolAttribute=  spring.velocity.exposeRequestAttributes=false  spring.velocity.exposeSessionAttributes=false  spring.velocity.exposeSpringMacroHelpers=false  spring.velocity.numberToolAttribute=  spring.velocity.prefix=  spring.velocity.properties.\*=  spring.velocity.requestContextAttribute=  spring.velocity.resourceLoaderPath=classpath:/templates/  spring.velocity.suffix=.vm  spring.velocity.templateEncoding=UTF-8  spring.velocity.viewNames= # whitelist of view names that can be resolved  #INTERNATIONALIZATION (MessageSourceAutoConfiguration)  spring.messages.basename=messages  spring.messages.cacheSeconds=-1  spring.messages.encoding=UTF-8  #SECURITY (SecurityProperties)  security.user.name=user # login username  security.user.password= # login password  security.user.role=USER # role assigned to the user  security.require-ssl=false # advanced settings ...  security.enable-csrf=false  security.basic.enabled=true  security.basic.realm=Spring  security.basic.path= # /\*\*  security.headers.xss=false  security.headers.cache=false  security.headers.frame=false  security.headers.contentType=false  security.headers.hsts=all # none / domain / all  security.sessions=stateless # always / never / if\_required / stateless  security.ignored=false  #DATASOURCE (DataSourceAutoConfiguration & DataSourceProperties)  spring.datasource.name= # name of the data source  spring.datasource.initialize=true # populate using data.sql  spring.datasource.schema= # a schema (DDL) script resource reference  spring.datasource.data= # a data (DML) script resource reference  spring.datasource.platform= # the platform to use in the schema resource (schema-${platform}.sql)  spring.datasource.continueOnError=false # continue even if can't be initialized  spring.datasource.separator=; # statement separator in SQL initialization scripts  spring.datasource.driverClassName= # JDBC Settings...  spring.datasource.url=  spring.datasource.username=  spring.datasource.password=  spring.datasource.max-active=100 # Advanced configuration...  spring.datasource.max-idle=8  spring.datasource.min-idle=8  spring.datasource.initial-size=10  spring.datasource.validation-query=  spring.datasource.test-on-borrow=false  spring.datasource.test-on-return=false  spring.datasource.test-while-idle=  spring.datasource.time-between-eviction-runs-millis=  spring.datasource.min-evictable-idle-time-millis=  spring.datasource.max-wait-millis=  #MONGODB (MongoProperties)  spring.data.mongodb.host= # the db host  spring.data.mongodb.port=27017 # the connection port (defaults to 27107)  spring.data.mongodb.uri=mongodb://localhost/test # connection URL  spring.data.mongo.repositories.enabled=true # if spring data repository support is enabled  #JPA (JpaBaseConfiguration, HibernateJpaAutoConfiguration)  spring.jpa.properties.\*= # properties to set on the JPA connection  spring.jpa.openInView=true  spring.jpa.show-sql=true  spring.jpa.database-platform=  spring.jpa.database=  spring.jpa.generate-ddl=false # ignored by Hibernate, might be useful for other vendors  spring.jpa.hibernate.naming-strategy= # naming classname  spring.jpa.hibernate.ddl-auto= # defaults to create-drop for embedded dbs  spring.data.jpa.repositories.enabled=true # if spring data repository support is enabled  #SOLR (SolrProperties})  spring.data.solr.host=http://127.0.0.1:8983/solr  spring.data.solr.zkHost=  spring.data.solr.repositories.enabled=true # if spring data repository support is enabled  #ELASTICSEARCH (ElasticsearchProperties})  spring.data.elasticsearch.cluster-name= # The cluster name (defaults to elasticsearch)  spring.data.elasticsearch.cluster-nodes= # The address(es) of the server node (comma-separated; if not specified starts a client node)  spring.data.elasticsearch.local=true # if local mode should be used with client nodes  spring.data.elasticsearch.repositories.enabled=true # if spring data repository support is enabled  #FLYWAY (FlywayProperties)  flyway.locations=classpath:db/migrations # locations of migrations scripts  flyway.schemas= # schemas to update  flyway.initVersion= 1 # version to start migration  flyway.prefix=V  flyway.suffix=.sql  flyway.enabled=true  flyway.url= # JDBC url if you want Flyway to create its own DataSource  flyway.user= # JDBC username if you want Flyway to create its own DataSource  flyway.password= # JDBC password if you want Flyway to create its own DataSource  #LIQUIBASE (LiquibaseProperties)  liquibase.change-log=classpath:/db/changelog/db.changelog-master.yaml  liquibase.contexts= # runtime contexts to use  liquibase.default-schema= # default database schema to use  liquibase.drop-first=false  liquibase.enabled=true  #JMX  spring.jmx.enabled=true # Expose MBeans from Spring  #ABBIT (RabbitProperties)  spring.rabbitmq.host= # connection host  spring.rabbitmq.port= # connection port  spring.rabbitmq.addresses= # connection addresses (e.g. myhost:9999,otherhost:1111)  spring.rabbitmq.username= # login user  spring.rabbitmq.password= # login password  spring.rabbitmq.virtualhost=  spring.rabbitmq.dynamic=  #REDIS (RedisProperties)  spring.redis.host=localhost # server host  spring.redis.password= # server password  spring.redis.port=6379 # connection port  spring.redis.pool.max-idle=8 # pool settings ...  spring.redis.pool.min-idle=0  spring.redis.pool.max-active=8  spring.redis.pool.max-wait=-1  #ACTIVEMQ (ActiveMQProperties)  spring.activemq.broker-url=tcp://localhost:61616 # connection URL  spring.activemq.user=  spring.activemq.password=  spring.activemq.in-memory=true # broker kind to create if no broker-url is specified  spring.activemq.pooled=false  #HornetQ (HornetQProperties)  spring.hornetq.mode= # connection mode (native, embedded)  spring.hornetq.host=localhost # hornetQ host (native mode)  spring.hornetq.port=5445 # hornetQ port (native mode)  spring.hornetq.embedded.enabled=true # if the embedded server is enabled (needs hornetq-jms-server.jar)  spring.hornetq.embedded.serverId= # auto-generated id of the embedded server (integer)  spring.hornetq.embedded.persistent=false # message persistence  spring.hornetq.embedded.data-directory= # location of data content (when persistence is enabled)  spring.hornetq.embedded.queues= # comma separate queues to create on startup  spring.hornetq.embedded.topics= # comma separate topics to create on startup  spring.hornetq.embedded.cluster-password= # customer password (randomly generated by default)  #JMS (JmsProperties)  spring.jms.pub-sub-domain= # false for queue (default), true for topic  #SPRING BATCH (BatchDatabaseInitializer)  spring.batch.job.names=job1,job2  spring.batch.job.enabled=true  spring.batch.initializer.enabled=true  spring.batch.schema= # batch schema to load  #AOP  spring.aop.auto=  spring.aop.proxy-target-class=  #FILE ENCODING (FileEncodingApplicationListener)  spring.mandatory-file-encoding=false  #SPRING SOCIAL (SocialWebAutoConfiguration)  spring.social.auto-connection-views=true # Set to true for default connection views or false if you provide your own  #SPRING SOCIAL FACEBOOK (FacebookAutoConfiguration)  spring.social.facebook.app-id= # your application's Facebook App ID  spring.social.facebook.app-secret= # your application's Facebook App Secret  #SPRING SOCIAL LINKEDIN (LinkedInAutoConfiguration)  spring.social.linkedin.app-id= # your application's LinkedIn App ID  spring.social.linkedin.app-secret= # your application's LinkedIn App Secret  #SPRING SOCIAL TWITTER (TwitterAutoConfiguration)  spring.social.twitter.app-id= # your application's Twitter App ID  spring.social.twitter.app-secret= # your application's Twitter App Secret  #SPRING MOBILE SITE PREFERENCE (SitePreferenceAutoConfiguration)  spring.mobile.sitepreference.enabled=true # enabled by default  #SPRING MOBILE DEVICE VIEWS (DeviceDelegatingViewResolverAutoConfiguration)  spring.mobile.devicedelegatingviewresolver.enabled=true # disabled by default  spring.mobile.devicedelegatingviewresolver.normalPrefix=  spring.mobile.devicedelegatingviewresolver.normalSuffix=  spring.mobile.devicedelegatingviewresolver.mobilePrefix=mobile/  spring.mobile.devicedelegatingviewresolver.mobileSuffix=  spring.mobile.devicedelegatingviewresolver.tabletPrefix=tablet/  spring.mobile.devicedelegatingviewresolver.tabletSuffix=  ######========ACTUATOR PROPERTIES===========  #MANAGEMENT HTTP SERVER (ManagementServerProperties)  management.port= # defaults to 'server.port'  management.address= # bind to a specific NIC  management.contextPath= # default to '/'  #ENDPOINTS (AbstractEndpoint subclasses)  endpoints.autoconfig.id=autoconfig  endpoints.autoconfig.sensitive=true  endpoints.autoconfig.enabled=true  endpoints.beans.id=beans  endpoints.beans.sensitive=true  endpoints.beans.enabled=true  endpoints.configprops.id=configprops  endpoints.configprops.sensitive=true  endpoints.configprops.enabled=true  endpoints.configprops.keys-to-sanitize=password,secret  endpoints.dump.id=dump  endpoints.dump.sensitive=true  endpoints.dump.enabled=true  endpoints.env.id=env  endpoints.env.sensitive=true  endpoints.env.enabled=true  endpoints.health.id=health  endpoints.health.sensitive=false  endpoints.health.enabled=true  endpoints.info.id=info  endpoints.info.sensitive=false  endpoints.info.enabled=true  endpoints.metrics.id=metrics  endpoints.metrics.sensitive=true  endpoints.metrics.enabled=true  endpoints.shutdown.id=shutdown  endpoints.shutdown.sensitive=true  endpoints.shutdown.enabled=false  endpoints.trace.id=trace  endpoints.trace.sensitive=true  endpoints.trace.enabled=true  #MVC ONLY ENDPOINTS  endpoints.jolokia.path=jolokia  endpoints.jolokia.sensitive=true  endpoints.jolokia.enabled=true # when using Jolokia  endpoints.error.path=/error  #JMX ENDPOINT (EndpointMBeanExportProperties)  endpoints.jmx.enabled=true  endpoints.jmx.domain= # the JMX domain, defaults to 'org.springboot'  endpoints.jmx.unique-names=false  endpoints.jmx.enabled=true  endpoints.jmx.staticNames=  #JOLOKIA (JolokiaProperties)  jolokia.config.\*= # See Jolokia manual  #REMOTE SHELL  shell.auth=simple # jaas, key, simple, spring  shell.command-refresh-interval=-1  shell.command-path-pattern= # classpath:/commands/, classpath:/crash/commands/  shell.config-path-patterns= # classpath:/crash/  shell.disabled-plugins=false # don't expose plugins  shell.ssh.enabled= # ssh settings ...  shell.ssh.keyPath=  shell.ssh.port=  shell.telnet.enabled= # telnet settings ...  shell.telnet.port=  shell.auth.jaas.domain= # authentication settings ...  shell.auth.key.path=  shell.auth.simple.user.name=  shell.auth.simple.user.password=  shell.auth.spring.roles=  #GIT INFO  spring.git.properties= # resource ref to generated git info properties file |