



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：完成优品购用户注册功能（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）的轻量级替代品。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，让更多的人更快对 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>1.4.0.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



- 📦 Maven Dependencies
 - spring-boot-starter-web-1.4.0.RELEASE.jar -
 - spring-boot-starter-1.4.0.RELEASE.jar - D:\re
 - spring-boot-1.4.0.RELEASE.jar - D:\repositor
 - spring-boot-autoconfigure-1.4.0.RELEASE.ja
 - spring-boot-starter-logging-1.4.0.RELEASE.j
 - logback-classic-1.1.7.jar - D:\repository_ssh
 - logback-core-1.1.7.jar - D:\repository_ssh\c
 - slf4j-api-1.7.21.jar - D:\repository_ssh\org\s
 - jcl-over-slf4j-1.7.21.jar - D:\repository_ssh\o
 - jul-to-slf4j-1.7.21.jar - D:\repository_ssh\org
 - log4j-over-slf4j-1.7.21.jar - D:\repository_ssl
 - spring-core-4.3.2.RELEASE.jar - D:\repositor
 - snakeyaml-1.17.jar - D:\repository_ssh\org\
 - spring-boot-starter-tomcat-1.4.0.RELEASE.ja
 - tomcat-embed-core-8.5.4.jar - D:\repository
 - tomcat-embed-el-8.5.4.jar - D:\repository_s
 - tomcat-embed-websocket-8.5.4.jar - D:\rep
 - hibernate-validator-5.2.4.Final.jar - D:\repos
 - validation-api-1.1.0.Final.jar - D:\repository_
 - jboss-logging-3.3.0.Final.jar - D:\repository_

.....

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

我们发现默认情况下工程的 JDK 版本是 1.6,而我们通常用使用 1.7 的版本，所以我们需要在 pom.xml 中添加以下配置

```
<properties>

    <java.version>1.7</java.version>

</properties>
```

添加后更新工程，会发现版本已经变更为 1.7

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

public class Application {

    public static void main(String[] args) {

        SpringApplication.run(Application.class, args);

    }

}
```

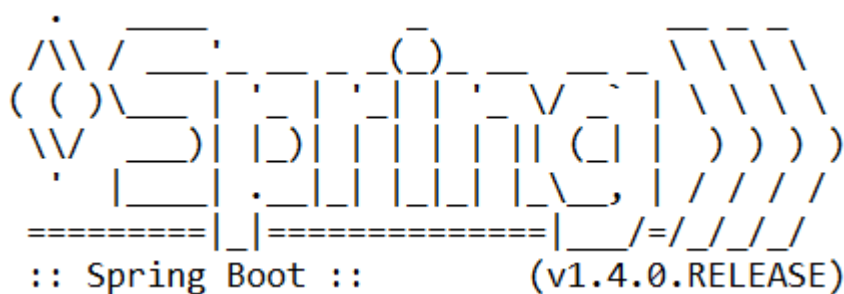
简单解释一下，@SpringBootApplication 其实就是以下三个注解的总和

@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



你能看出来上边这个图是什么东西？🤔

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

public class 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>
```

赶快试试看吧，是不是很爽。 😁

1.3 Spring Boot 与 ActiveMQ 整合

1.3.1 使用内嵌服务

(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

public class 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")

    public void 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.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 使用外部服务

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

Queues

Name ↑	Number Of Pending Messages	Number Of Consumers	Messages Enqueued	Messages Dequeued	Views	Operations
itcast	0	1	1	1	Browse Active Consumers Active Producers	Send To Purge Delete

1.3.3 发送 Map 信息

(1) 修改 QueueController.java

```
@RequestMapping("/sendmap")

public void sendMap(){

    Map map=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")

public void readMap(Map map){

    System.out.println(map);

}
```

2.短信发送平台-阿里大于

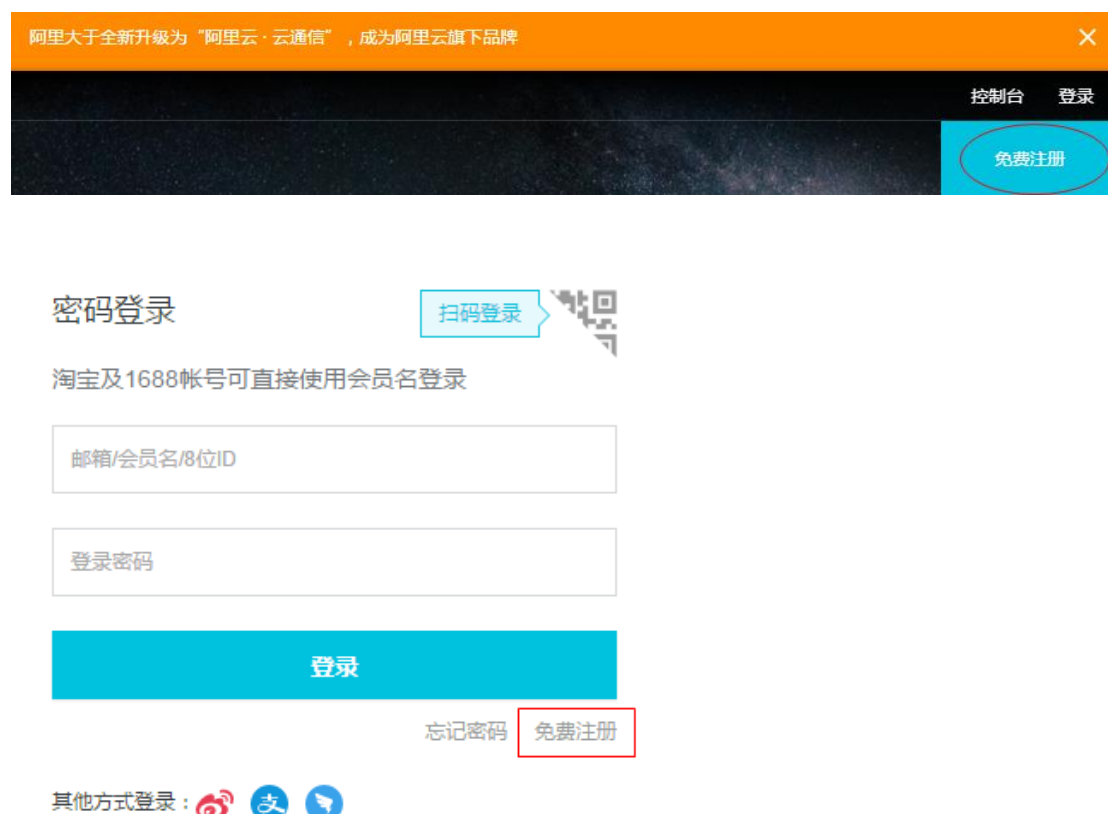
2.1 阿里大于简介

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 (<https://dayu.aliyun.com/>)



阿里大于全新升级为“阿里云·云通信”，成为阿里云旗下品牌

控制台 登录

免费注册

密码登录 扫码登录

淘宝及1688帐号可直接使用会员名登录

邮箱/会员名/8位ID

登录密码

登录

忘记密码 免费注册

其他方式登录： 微博 支付宝 微信



欢迎注册阿里云

+86

>>

请按住滑块，拖动到最右边

同意条款并注册

☒ 《阿里云网站服务条款》 | 《法律声明和隐私权政策》

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。

密码登录

扫码登录

淘宝及1688帐号可直接使用会员名登录

登录

[忘记密码](#) [免费注册](#)

其他方式登录：  



点击进入控制台

阿里大于全新升级及服务搬迁公告

阿里大于升级为“阿里云·云通信”，是阿里云旗下品牌。

大于原有服务可在阿里云控制台继续使用。

您可在下方直接选择要使用的服务，我们将协助您跳转至阿里云并完成开通。



点击使用短信服务





3.2.3 申请签名

短信签名

<input type="checkbox"/>	签名名称	工单号	签名类型	创建时间	审核状态(全部)	操作
没有数据						

* 签名类型: ☒ 验证码或短信通知(0.045元/条)

☐ 推广短信或群发助手(0.055元/条)

• 企业用户可启用推广短信和群发助手, 立即[申请升级为企业用户](#)

* 签名: 2/12

自定义签名数量已达到上限, 请使用已有签名或删除原签名后重新申请, 也可[升级为企业用户](#)享更高权限

- 必须含中文, 可以包含数字、英文
- 若签名 / 模版内容侵犯到第三方权益必须获得第三方真实授权
- 无须添加【】、()、[]符号, 签名发送会自带【】符号, 避免重复
- 签名/模版申请规范详见https://help.aliyun.com/document_detail/55324.html

* 签名用途: ☒ 个人使用, 签名为自己产品名 / 网站名等

☐ 个人使用, 签名为他人产品名 / 网站名等

☐ 企业使用, 签名为公司名, 或旗下产品名 / 网站名等

☐ 企业使用, 且该公司是媒体、报社、学校、医院、机关事业单位, 签名为公司名, 或旗下产品名 / 网站名等

申请说明:

6/100

<input type="checkbox"/>	签名名称	工单号	签名类型	创建时间	审核状态(全部)	操作
<input type="checkbox"/>	黑马	84545070	验证码或短信通知	2017-08-15 21:57:38	● 审核中	

3.2.4 申请模板



* 模版类型: ☒ 验证码(0.045元/条)

☐ 短信通知(0.045元/条)

☐ 推广短信(0.055元/条)

☐ 群发助手(0.055元/条)

• 企业用户可启用推广短信和群发助手，立即[申请升级为企业用户](#)

* 模版名称: 品优购注册模板 7/30

* 模版内容: 您好，欢迎注册

29/500

- 验证码模板
- 不能发送营销类短信(验证码除外)
- 签名/模板内容

* 申请说明: 品优购用户注册模板

9/100

模版预览 确定

提交成功

1. 预计2小时完成审核。

2. 审核工作时间: 周一至周日9:00-23:00(法定节假日顺延)。

我知道了

请输入模版名称搜索

查询

添加短信模版

刷新

<input type="checkbox"/>	模版名称	工单号	模版CODE	模版类型(全部) ▾	创建时间	审核状态(全部) ▾	操作
<input type="checkbox"/>	品优购注册模板	84495087	SMS_85735065	验证码	2017-08-15 22:03:49	● 审核中	详情
<input type="checkbox"/>	删除						

< 上一页 1 下一页 >



3.2.5 创建 accessKey

The screenshot shows the Alibaba Cloud console interface. The top navigation bar includes '工单', '备案', '支持', 'heimahuangshu', and '简体中文'. The 'heimahuangshu' dropdown menu is open, displaying various settings options: '基本资料', '实名认证', '安全设置', '安全管控', '访问控制', 'accesskeys' (highlighted with a red box), '会员权益', '会员积分', and '云大使管理'. Below the menu is a '安全风险警告' (Security Risk Warning) dialog box. It contains a warning icon and text: '云账号AccessKey具有所有云产品API的访问权限，一旦泄露将导致极大的安全风险！强烈建议遵循[阿里云安全最佳实践](#)，根据最小权限原则创建并使用子用户来进行API访问和控制台操作。'. At the bottom of the dialog, there are two buttons: '继续使用AccessKey' (circled in red) and '开始使用子用户AccessKey'. Below the dialog, there are two buttons: '刷新' and '创建Access Key' (circled in red). The '创建Access Key' button is highlighted with a red circle.

安全风险警告

云账号AccessKey具有所有云产品API的访问权限，一旦泄露将导致极大的安全风险！强烈建议遵循[阿里云安全最佳实践](#)，根据最小权限原则创建并使用子用户来进行API访问和控制台操作。

继续使用AccessKey 开始使用子用户AccessKey

刷新 创建Access Key

创建Access Key

请阅读并同意《[API使用规范](#)》

同意并创建 取消

手机验证

您绑定的手机：

(更换手机)

* 校验码：

点击获取

确定


取消

① Access Key ID和Access Key Secret是您访问阿里云API的密钥，具有该账户完全的权限，请您妥善保管。

Access Key ID	Access Key Secret	状态	创建时间
	显示	启用	2017-08-18 17:09:11

3.3 SDK 安装

从阿里云通信官网上下载 Demo 工程

 dysmsapi_demo_sdk_java.zip

解压后导入 Eclipse

- alicom-dysms-api
- alicom-mns-receive-samples
- aliyun-java-sdk-core
- aliyun-java-sdk-dysmsapi

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

(删除 aliyun-java-sdk-core 的单元测试类)

本地 jar 包安装后 alicom-dysms-api 工程引入依赖

```
<dependencies>

    <dependency>

        <groupId>com.aliyun</groupId>
```



```
<artifactId>aliyun-java-sdk-dysmsapi</artifactId>

<version>1.0.0-SNAPSHOT</version>

</dependency>

<dependency>

    <groupId>com.aliyun</groupId>

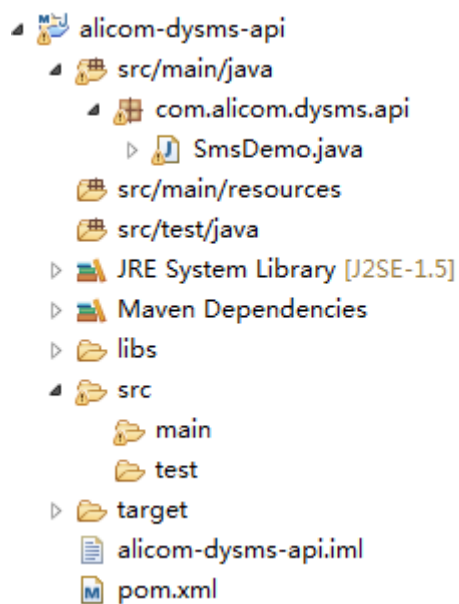
    <artifactId>aliyun-java-sdk-core</artifactId>

    <version>3.2.5</version>

</dependency>

</dependencies>
```

红叉消失了 :-)



3.4 发送短信测试

(1) 打开 SmsDemo

替换下列几处代码



```
// 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 = "yourAccessKeyId";
static final String accessKeySecret = "yourAccessKeySecret";
```

这个 accessKeyId 和 accessSecret 到刚才申请过的

手机号，短信签名和模板号

```
//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("17711112222");
//必填:短信签名-可在短信控制台中找到
request.setSignName("黑马");
//必填:短信模板-可在短信控制台中找到
request.setTemplateCode("SMS_85735065");
```

模板参数

```
request.setTemplateParam("{\"number\":\"292811\"}");
```

number 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





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 itcast_sms （JAR 工程），POM 文件引入依赖

```
<properties>

    <java.version>1.7</java.version>

</properties>

<parent>

    <groupId>org.springframework.boot</groupId>

    <artifactId>spring-boot-starter-parent</artifactId>

    <version>1.4.0.RELEASE</version>

</parent>

<dependencies>

    <dependency>

        <groupId>org.springframework.boot</groupId>

        <artifactId>spring-boot-starter-web</artifactId>

    </dependency>

    <dependency>
```



```
<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-activemq</artifactId>

</dependency>

<dependency>

    <groupId>com.aliyun</groupId>

    <artifactId>aliyun-java-sdk-dysmsapi</artifactId>

    <version>1.0.0-SNAPSHOT</version>

</dependency>

<dependency>

    <groupId>com.aliyun</groupId>

    <artifactId>aliyun-java-sdk-core</artifactId>

    <version>3.2.5</version>

</dependency>

</dependencies>
```

(2) 创建引导类

```
package cn.itcast.sms;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class Application {

    public static void 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://192.168.25.135:61616

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 cn.itcast.sms;

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 java.text.SimpleDateFormat;

import java.util.Date;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.core.env.Environment;
```



```
import org.springframework.stereotype.Component;

/**
 * 短信工具类
 *
 * @author Administrator
 *
 */
@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";

    @Autowired

    private Environment env;

    // 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)

    /**
     * 发送短信
     *
     * @param mobile 手机号
     *
     * @param template_code 模板号
     */
}
```



```
* @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 {

    String accessKeyId =env.getProperty("accessKeyId");

    String accessKeySecret = env.getProperty("accessKeySecret");

    //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

    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);

//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.getAcResponse(request);

return sendSmsResponse;
}

public QuerySendDetailsResponse querySendDetails(String mobile,String bizId)
throws ClientException {

String accessKeyId =env.getProperty("accessKeyId");
```



```
String accessKeySecret = env.getProperty("accessKeySecret");

//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

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;

    }

}
```

3.2.3 消息监听类

创建 SmsListener.java

```
package cn.itcast.sms;

import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.jms.annotation.JmsListener;

import org.springframework.stereotype.Component;

import com.aliyuncs.dysmsapi.model.v20170525.SendSmsResponse;

import com.aliyuncs.exceptions.ClientException;

/**
 * 消息监听类
 *
 * @author Administrator
 */
@Component
public class SmsListener {

    @Autowired

    private SmsUtil smsUtil;
```



```
@JmsListener(destination="sms")

public void sendSms(Map<String,String> map){

    try {

        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());

    } catch (ClientException e) {

        e.printStackTrace();

    }

}
```

3.3 代码测试

修改 springboot-demo 的 QueueController.java

```
@RequestMapping("/sendsms")

public void sendSms(){

    Map map=new HashMap<>();
```



```
map.put("mobile", "13900001111");

map.put("template_code", "SMS 85735065");

map.put("sign_name", "黑马");

map.put("param", "{\n\"number\":\n\"102931\n\"}");

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

Code=OK

Message=OK

RequestId=6B4EBD28-5305-40FD-B735-FC9823AC697E

BizId=515723303063309745^0

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



用户名：

登录密码：

确认密码：

手机号：

短信验证码： [获取短信验证码](#)

☒ 同意协议并注册《品优购用户协议》

[完成注册](#)

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-interface pinyougou-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 applicationContext-tx.xml

```
<dubbo:protocol name="dubbo" port="20886" />

<dubbo:annotation package="com.pinyougou.user.service.impl" />

<dubbo:application name="pinyougou-user-service"/>
```



```
<dubbo:registry address="zookeeper://192.168.25.135:2181"/>
```

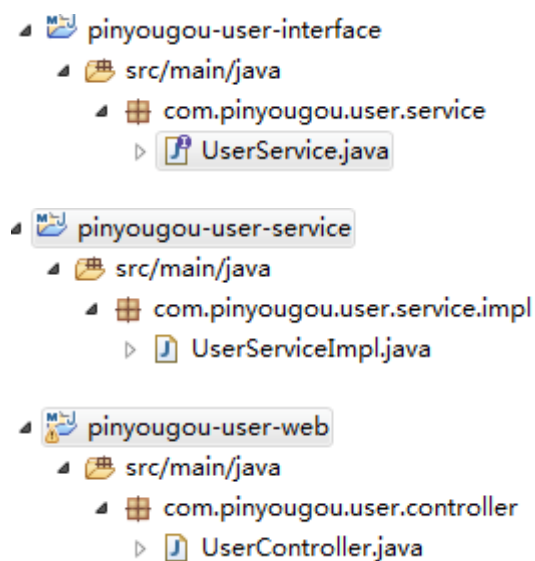
4.2.3 用户中心 WEB 层

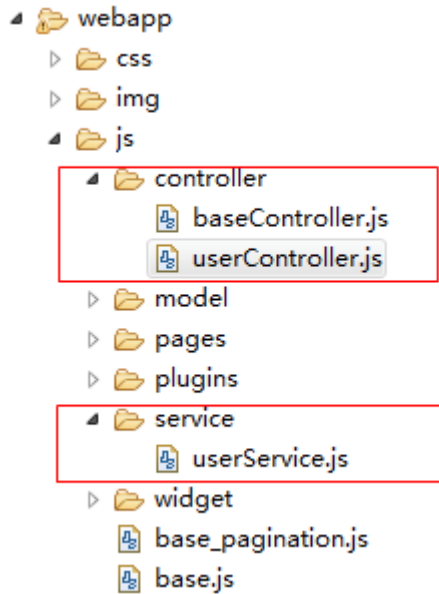
创建 war 工程 pinyougou-user-web 我们将注册功能放入此工程

- (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





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

```
<script type="text/javascript" src="plugins/angularjs/angular.min.js"></script>

<script type="text/javascript" src="js/base.js"></script>

<script type="text/javascript" src="js/service/userService.js"></script>
```



```
<script type="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>

</form>
```



```
</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;&nbsp;&nbsp;&nbsp;&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

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。

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");

    }

}
```

(5) pinyougou-user-web 的 userService.js

```
//发送验证码

this.sendCode=function(phone){

    return $http.get("../user/sendCode.do?phone="+phone);

}
```

(6) 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);  
  
    }  
  
);  
  
}
```

(7) 修改页面 register.html

```
<a ng-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  
  
 */  
  
public boolean checkSmsCode(String phone,String code);
```

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

```
/**  
  
 * 判断验证码是否正确  
  
 */  
  
public boolean 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);

    }
```




```
        return new Result(true, "增加成功");

    } catch (Exception e) {

        e.printStackTrace();

        return new Result(false, "增加失败");

    }

}
```

(4) 修改 pinyougou-user-web 的 userService.js

```
//增加

this.add=function(entity,smscode){

    return $http.post('../user/add.do?smscode='+smscode ,entity );

}
```

(5) 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 -->

<bean id="targetConnectionFactory"
class="org.apache.activemq.ActiveMQConnectionFactory">

    <property name="brokerURL" value="tcp://192.168.25.135:61616"/>

</bean>

<!-- Spring 用于管理真正的 ConnectionFactory 的 ConnectionFactory -->

<bean id="connectionFactory"
class="org.springframework.jms.connection.SingleConnectionFactory">

    <!-- 目标 ConnectionFactory 对应真实的可以产生 JMS Connection 的 ConnectionFactory -->

    <property name="targetConnectionFactory" ref="targetConnectionFactory"/>

</bean>

<!-- Spring 提供的 JMS 工具类，它可以进行消息发送、接收等 -->

<bean id="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 -->

    <property name="connectionFactory" ref="connectionFactory"/>

</bean>

<!-- 这个是点对点消息 -->

<bean id="smsDestination" class="org.apache.activemq.command.ActiveMQQueue">

    <constructor-arg value="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
 */

public void 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 JMSEException {

        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("number", 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 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 Message Service API) 。

6) spring-boot-starter-batch

支持 Spring Batch，包括 HSQLDB 数据库。

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 Persistence API)，包括 spring-data-jpa、spring-orm、hibernate。

12) spring-boot-starter-data-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

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 spring-social-twitter

32) spring-boot-starter-test

支持常规的测试依赖，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

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.springframework'

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
```