#### 第二章 使用SpringCloud框架实现一个微服务

#### 1.项目概要介绍

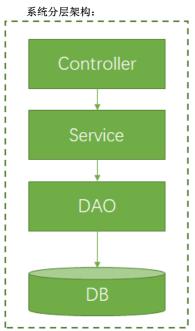
在这一章节中,我们尝试通过一个可以运行的简单的示例来学习SpringCloud的功能,在案例的选择上,我们也是从项目实际出发,选取了一个消息服务(Message-Service),因为实际的项目中都可能会用到通过短信网关或者Email发送一些通知消息的功能,我们编写的示例代码也是尽可能的接近于真实的生产代码,在后续的章节中,我们会随着学习的深入,对此示例进行不同程度的改写和重构,以满足大型分布式企业使用需求。

### 2.项目所涉及到的技术或工具

核心技术选型:

- JDK1.8
- Maven 3.5.2
- spring-boot-starter-parent 1.5.8
- spring-boot-starter-freemarker 1.5.8
- spring-cloud-dependencies Dalston.SR4
- lombok 1.16.18

### 3.项目架构



该演示系统遵循典型的三层架构模式,Controller层提供Rest服务,业务逻辑层放在Service层实现,Controller层与Service层通过接口进 行调用依赖,DAO层负责数据库的读写操作,DB是该微服务所涉及到的数据库表信息。

## 4.项目创建

使用Eclipse的Maven工程创建向导创建一个maven单模块项目,这里我们命名为springcloud-message,你也可以根据自己的喜好,选择自己熟练的IDE来创建工程,创建好项目的基本信息如下

group:com.pachiraframework artifact-id:springcloud-message version:0.0.1-SNAPSHOT packing:jar 如图所示:

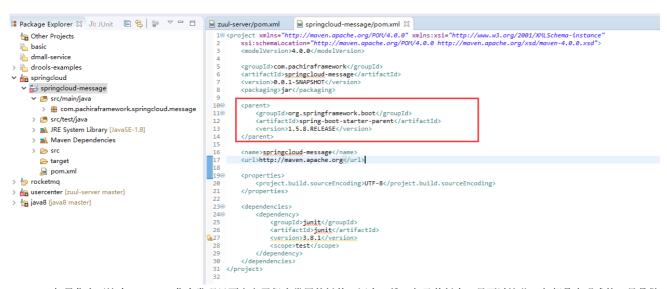
```
🖺 Package Explorer 🛭 Jv JUnit 🕒 💲 🔝 🔻 🗖 📗 zuul-server/pom.xml 🔝 springcloud-message/pom.xml 🗵
                                                                                                                                                                                                   10 Topict 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">
documentation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
documentation="http://maven.apache.org/xsd/maven-4.0.0.xsd">
documentation="http://maven.apache.org/xsd/maven-4.0.0.xsd
          👍 Other Projects
         abasic
          a dmall-service
  > 🏊 drools-examples
                                                                                                                                                                                                                   <groupId>com.pachiraframework</groupId>
                                                                                                                                                                                                                   🗸 🌆 springcloud

▼ " springcloud-message
                  <name>springcloud-message</name>
<url>http://maven.apache.org</url>
                            > # com.pachiraframework.springcloud.message
                   > 乃 src/test/java
                   > M JRE System Library [JavaSE-1.8]
                                                                                                                                                                                                                 > Mayen Dependencies
                                                                                                                                                                                              15
16
17<sup>©</sup>
18<sup>©</sup>
                   > 冯 src
                          target
                                                                                                                                                                                                                         <dependency)
                             m pom.xml
                                                                                                                                                                                                                          <qroupId>junit</groupId>
<artifactId>junit</artifactId>
<artifactId>junit</artifactId>
<version>3.8.1</version>
<scope>test</scope>
</dependency>
                                                                                                                                                                                               19
 > b rocketmq
 > in usercenter [zuul-server master]
 > 🔓 java8 [java8 master]
                                                                                                                                                                                                                    </denendencies
```

## 5.添加SpringBoot相关依赖

使用springboot的一个优势就是它为我们提供了非常多的starter组件,用来简化我们的开发,一旦使用它以后,你会发现以后的开发越来越离不开它--因为它实在是太方便了。

• 给该工程pom设置一个parent依赖spring-boot-starter-parent



如果你点开这个parentpom你会发现里面定义了很多常用的插件,还有一堆jar包及其版本,只不过这些jar包都是声明式的,只是做了声明,并没有真实的引入到工程中,这样你在自己的pom文件中就不需要写依赖jar包的版本了,就像在这个截图中展示的,我们把工程的parent指向spring-boot-starter-parent后,junit的依赖上出现了一个黄色警告信息,告诉我们这个版本号已经在父类中声明过了,在项目中就没必要声明版本号了。

• 引入spring-boot-starter-web

```
加入如下依赖
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

```
□ 🗎 🔝 springcloud-message/pom.xml 🛭
<sup>3</sup> Package Explorer ⊠ Jπ JUnit
                                                                                                            E 🕏 🕞 ▽
 4 👸 springcloud-stuty [springcloud-message mast 🛧
       🗸 📸 springcloud-message [springcloud-messag
                                                                                                                            <groupId>com.pachiraframework</groupId>
             ▷ 려 src/main/java
                                                                                                                           <artifactId>springcloud-message</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>jar</packaging>
                # src/test/java

■ JRE System Library [JavaSE-1.6]

              4 Naven Dependencies
                                                                                                                           ⊳ 🚠 spring-boot-1.5.8.RELEASE.jar - C:\U

→ 
→ spring-boot-autoconfigure-1.5.8.REL

                  ▶ 🚮 spring-boot-starter-logging-1.5.8.RE
                                                                                                                            <name>springcloud-message</name
                  ▶ 👼 logback-classic-1.1.11.jar - C:\Users\
                  D logback-core-1.1.11.jar - C:\Users\A₁≡

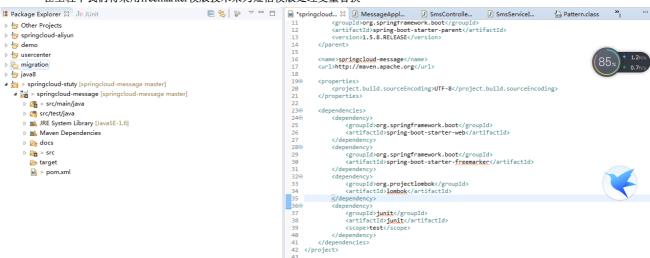
→ 

Milling

Discription

                                                                                                                            </properties>
                  <dependency?
                                                                                                                                             <groupId>org.springframework.boot</groupId>
                  <artifactId>spring-boot-starter-web</artifactId>
                   ▶ 5 snakeyaml-1.17.jar - C:\Users\Admir
                       a spring-boot-starter-tomcat-1.5.8.REL
                                                                                                                                            pendency>
<groupId><mark>junit</mark></groupId>
<artifactId><mark>junit</mark></artifactId>
                            tomcat-embed-core-8.3.23.jar
                  ▶ 👼 tomcat-annotations-api-8.5.23.jar - 🤇
                                                                                                                                             <scope>test</scope>
                  ▶ 👼 tomcat-embed-el-8.5.23.jar - C:\User
                  hibernate-validator-5.3.5.Final.jar - C
```

- 引入freemarker
- <dependency>
- <groupId>org.springframework.boot</groupId>
- <artifactId>spring-boot-starter-freemarker</artifactId>
- </dependency>
- 在工程中我们将采用freemarker模版技术来为短信模版处理变量替换



### 6.核心代码编写

按照系统架构分层创建对应的工程包结构出来

- ⇒ springcloud-stuty [springcloud-message master]
   ⇒ springcloud-message [springcloud-message master]
   ⇒ src/main/java
   ⇒ com.pachiraframework.springcloud.message
   ⇒ com.pachiraframework.springcloud.message.controller
   ⇒ com.pachiraframework.springcloud.message.dao
   ⇒ com.pachiraframework.springcloud.message.dto
   ⇒ com.pachiraframework.springcloud.message.entity
   ⇒ com.pachiraframework.springcloud.message.entity
   ⇒ com.pachiraframework.springcloud.message.service
   ⇒ src/main/framework.springcloud.message.service.impl
   ⇒ Src/test/java
   ⇒ JRE System Library [JavaSE-1.6]
   ⇒ Maven Dependencies
   ⇒ docs
   ⇒ src
- 其中
- controller 包中包含是的对应的rest服务接口
- dao中包含数据库操作

- dto 包含参数传递所需的变量,如入参对象或返回值对象
- entity包含数据库表模型对象
- service 定义了服务层的接口
- service.impl 是service接口层的实现,会调用dao组件完成业务操作

在这例子中,我们将短信内容以模版的方式存储在数据库表中,在发送短信时,请求对象需要携带要使用模版的ID和参数,然后替换模版中的变量,最终得到要发送的短信的内容。

核心对象:

\* 要发送的手机号

private String mobile;

\*/

- SmsSendRequest 客户端发送短信所需参数的封装, 如短信模版ID,模版参数列表,要发送给哪个手机号
- SmsSendResponse 发送短信的结果,如发送后短信网关的返回值code,消息提示等
- MessageTemplate是一个entity实体,表示一个短信模版,由一个唯一短信模版ID,模版内容组成,当然可以根据实际需要添加诸如短信签名,模版分类(通知类、广告营销)等

MessageTemplate的代码如下: package com.pachiraframework.springcloud.message.entity; import lombok.Data; @Data public class MessageTemplate { **/\*\*** \* 模版ID \*/ private String id; /\*\* \* 模版名称 private String name; \* 模版内容 private String content; SmsSendRequest对象的代码如下: package com.pachiraframework.springcloud.message.dto; import java.util.Map; import lombok.Data; /\*\* \* 发送sms消息对象 \* @author wangxuzheng \*/ @Data public class SmsSendRequest { /\*\* \* 短信模版ID \*/ private String templateId;

```
* 模版中携带的参数信息
private Map<String, Object> params;
SmsSendResponse代码如下:
package com.pachiraframework.springcloud.message.dto;
import lombok.Data;
@Data
public class SmsSendResponse {
/**
 * 返回消息
 */
private String message;
 * 返回状态码
private String code;
DAO中的方法很简单,为了演示,模拟了一个从数据库中根据模版ID获取模版信息的例子,这个例子中我们的短信模版中有个${code}变
量,表示要从客户端程序中传递过来的真实的数据。
package com.pachiraframework.springcloud.message.dao;
import org.springframework.stereotype.Repository;
import com.pachiraframework.springcloud.message.entity.MessageTemplate;
@Repository
public class MessageTemplateDao {
public MessageTemplate get(String id) {
//改成从数据库中读取模版信息
 MessageTemplate template = new MessageTemplate();
 template.setId(id);
 template.setName("注册验证码通知短信");
 template.setContent("验证码${code},请在页面输入此验证码并完成手机验证。XXX公司");
 return template;
}
SmsServiceImpl中实现发送短信的逻辑,这里我们并没有调用实际的短信网关,需要根据项目的真实情况,调用对应的短信服务,将
doSend()方法中改成调用短信服务即可。
package com.pachiraframework.springcloud.message.service.impl;
import java.io. StringReader;
import java.io. String Writer;
import org.springframework.beans.factory.annotation.Autowired;
import\ org. spring framework. stereotype. Service;
import com.pachiraframework.springcloud.message.dao.MessageTemplateDao;
import\ com. pachira framework. spring cloud. message. dto. Sms Send Request;
import com.pachiraframework.springcloud.message.dto.SmsSendResponse;
import com.pachiraframework.springcloud.message.entity.MessageTemplate;
import com.pachiraframework.springcloud.message.service.SmsService;
```

```
import freemarker.template.Configuration;
import freemarker.template.Template;
import lombok. SneakyThrows;
import lombok.extern.slf4j.Slf4j;
@Slf4j
@Service
public class SmsServiceImpl implements SmsService {
@Autowired
private MessageTemplateDao messageTemplateDao;
@Autowired
private Configuration configuration;
@Override
@SneakyThrows
public SmsSendResponse send(SmsSendRequest request) {
 MessageTemplate messageTemplate = messageTemplateDao.get(request.getTemplateId());//根据模版ID从数据库中加载模版明细
 String templateContent = messageTemplate.getContent();
 Template template = new Template(request.getTemplateId(), new StringReader(templateContent), configuration);
 StringWriter out = new StringWriter();
 template.process(request.getParams(), out);//模版内容+传递进来的参数=最终要发送的短信内容
 String content = out.toString();
 return doSend(request.getMobile(), content);//调用实际的短信网关服务,发送短信
//改成调用实际的短息网关发送消息
private SmsSendResponse doSend(String mobile,String content) {
 SmsSendResponse response = new SmsSendResponse();
 response.setCode("200");
 response.setMessage("发送成功");
 log.info("发送完毕,手机号:{},发送内容:{},状态码:{}",mobile,content,response.getCode());
 return response;
}
Rest服务,这里我们使用了2中方式来提供服务,实际生产环境中,根据需要只选择一种即可,只因为这两种服务提供方式不同,对应的客
户端调用也是不一样的
package com.pachiraframework.springcloud.message.controller;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import\ org. spring framework. web. bind. annotation. Request Method;
import org.springframework.web.bind.annotation.RestController;
import\ com. pachira framework. spring cloud. message. dto. Sms Send Request;
import\ com. pachira framework. spring cloud. message. dto. Sms Send Response;
import com.pachiraframework.springcloud.message.service.SmsService;
@RestController
@RequestMapping("/message/sms/")
public class SmsController {
@Autowired
private SmsService smsService;
@RequestMapping(method=RequestMethod.POST,value="send")
public ResponseEntity<SmsSendResponse> send(SmsSendRequest request){
```

```
SmsSendResponse response = smsService.send(request);
return ResponseEntity.ok(response);
}

@RequestMapping(method=RequestMethod.POST,value="send2")
public ResponseEntity<SmsSendResponse> send2(@RequestBody SmsSendRequest request){
SmsSendResponse response = smsService.send(request);
return ResponseEntity.ok(response);
}
}

通过一个main函数将服务启动起来
package com.pachiraframework.springcloud.message;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MessageApplication {
public static void main(String[] args) {
    SpringApplication.run(MessageApplication.class, args);
}
}
```

#### 7.启动服务

运行步骤6中的MessageApplication中的main函数,将服务启动起来,通过后台的log我们可以看出,实际上SpringBoot框架给我们启动了一个内嵌的tomcat容器,并监听8080端口(可以通过配置文件修改),启动成功,则会出现如下图所示的一些信息

```
MessageApplication [Java Application] D:\java_tools\Java\jdk1.8.0_131\bin\javaw.exe (2017年11月2日 下午2:02:35)
                                                                                                                                                                                                                              Starting Servlet Engine: Apache Tomcat/8.5.23
Initializing Spring embedded WebApplicationContext
Root WebApplicationContext: initialization completed in
2017-11-02 14:02:40.185
2017-11-02 14:02:40.388
                                                      INFO 6808 ---
INFO 6808 ---
                                                                                       [ main] org.apache.catalina.core.StandardE
[ost-startStop-1] o.a.c.c.C.[Tomcat].[localhost].[/]
 2017-11-02 14:02:40.388
                                                       INFO 6808 ---
                                                                                       [ost-startStop-1] o.s.web.context.ContextLoader
                                                                                                                                                                                                                             Root WebApplicationContext: initialization comp
Mapping servlet: 'dispatcherServlet' to [/]
Mapping filter: 'characterEncodingFilter' to: [
Mapping filter: 'hiddenHttpMethodFilter' to: [/*]
Mapping filter: 'httpPutFormContentFilker' to:
Mapping filter: 'requestContextFilter' to: [/*]
2017-11-02 14:02:40.684
2017-11-02 14:02:40.692
                                                      INFO 6808 ---
INFO 6808 ---
                                                                                       [ost-startStop-1] o.s.b.w.servlet.ServletRegistrationBean [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean
                                                                                      [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean
[ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean
[ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean
 2017-11-02 14:02:40.692
                                                        INFO 6808 ---
2017-11-02 14:02:40.693
2017-11-02 14:02:40.693
                                                       INFO 6808 ---
INFO 6808 ---
                                                                                                                                                                                                                             Mapping filter: 'requestContextFilter' to: [/*] SpringTemplateLoader for FreeMarker: using resource load ClassTemplateLoader for Spring macros added to FreeMarke Looking for @ControllerAdvice: org.springframework.boot. Mapped "{[/message/sms/send],methods[POST]}" onto publi Mapped "{[/error]}" onto public org.springframework.http Mapped "{[/error]}, produces=[text/html]}" onto public org Mapped URL path [/webjars/**] onto handler of type [class Mapped URL path [/**] onto handler of type [class org.sp Mapped URL path [/**/favicon.ico] onto handler of type [Cannot find template location(s): [classpath:/templates/ Registering beans for JMX exposure on startup
                                                                                                                main] o.s.ui.freemarker.SpringTemplateLoader
main] o.s.w.s.v.f.FreeMarkerConfigurer
main] s.w.s.m.m.a.RequestMappingHandlerAdapt
 2017-11-02 14:02:40.964
                                                       INFO 6808 ---
 2017-11-02 14:02:40.966
2017-11-02 14:02:41.511
                                                       INFO 6808 ---
                                                                                                                main] s.w.s.m.m.a.RequestMappingHandlerMapping
main] s.w.s.m.m.a.RequestMappingHandlerMapping
main] s.w.s.m.m.a.RequestMappingHandlerMapping
 2017-11-02 14:02:41.634
                                                        INFO 6808 ---
 2017-11-02 14:02:41.640
2017-11-02 14:02:41.640
                                                       INFO 6808 ---
                                                                                                                 main] o.s.w.s.handler.SimpleUrlHandlerMapping
 2017-11-02 14:02:41.740
                                                       INFO 6808 ---
 2017-11-02 14:02:41.740
2017-11-02 14:02:41.800
                                                       INFO 6808 ---
INFO 6808 ---
                                                                                                                main] o.s.w.s.handler.SimpleUrlHandlerMapping
main] o.s.w.s.handler.SimpleUrlHandlerMapping
 2017-11-02 14:02:41.940
                                                        WARN 6808 ---
                                                                                                                 mainl o.s.b.a.f.FreeMarkerAutoConfiguration
2017-11-02 14:02:42.241
2017-11-02 14:02:42.375
                                                                                                                                                                                                                             Registering beans for JMX exposure on startup
Tomcat started on port(s): 8080 (http)
Started MessageApplication in 6.1 seconds (JVM running f
                                                       INFO 6808 ---
                                                                                                                 main o.s.j.e.a.AnnotationMBeanExpo
                                                                                                                               s.b.c.e.t.TomcatEmbeddedServletContainer
2017-11-02 14:02:42.392
                                                      INFO 6808 --- [
                                                                                                                main | c.p.s.message.MessageApplication
```

# 8.通过Postman工具测试rest服务

1)测试send方法

根据我们Rest接口的url和参数,在Postman中按照如下内容进行设置

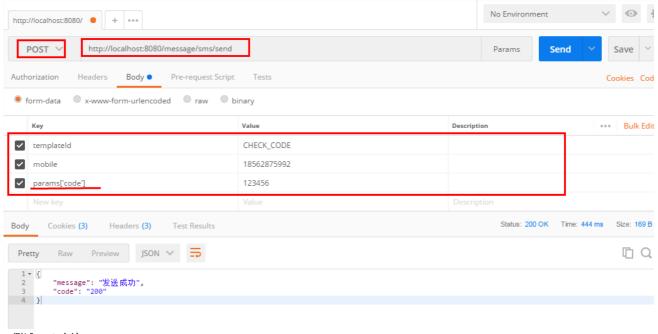
请求方法: POST

请求URL:http://localhost:8080/message/sms/send

参数列表: --由于我们在send方法中接收的是SmsSendRequest对象,根据SpringMVC参数绑定的规则,我们需要根据SmsSendRequest中的属性进行参数传递即可

templateId:CHECK\_CODE --当然这个案例中你传递任意的字符都是可以的,因为我们在MessageTemplateDao中并没有真实的去查询数据库:) mobile:18562875992 -- 要发送的手机号地址

params[code]:123456 这个参数比较有趣,因为在SmsSendRequest对象中我们把模版要传递的参数定义成了一个Map<String,Object>类型,因此这种传递表示要往params属性的key='code'传递value=123456,即params.set("code","123456"),如果模板中含有多个参数,可以通过这种方式传递多个key-value的组合



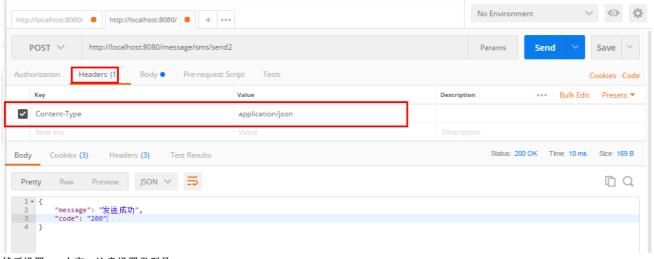
#### 2)测试send2方法

请求方法: POST

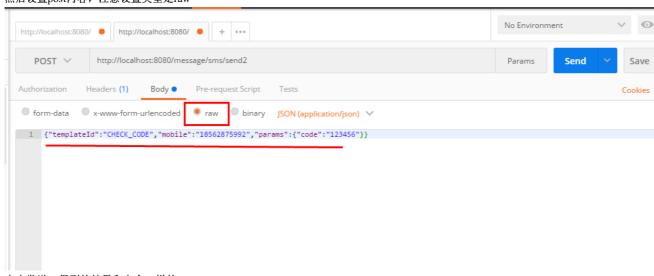
请求URL:http://localhost:8080/message/sms/send2

参数列表: send2方法在接收参数时使用的是@RequestBody注解,它接收的是一个json串,然后把接收到的json串绑定到参数对象上,因此这种方法和上面的不同

首先要设置请求内容类型是application/json



然后设置post内容,注意设置类型是raw



点击发送,得到的结果和上个一样的

# 9.通过RestTemplate调用Rest服务

```
针对2中不同方式的rest方法,使用RestTemplate调用的方式也是不一样的,如下代码所示
第一种方式需要逐个填写参数名称和对应的参数值,第二种方法更间接,直接把SmsSendRequest对象封装好,然后发送请求即可
package com.pachiraframework.springcloud.message.controller;
import static org.hamcrest.CoreMatchers.equalTo;
import static org.hamcrest.CoreMatchers.notNullValue;
import static org.junit.Assert.assertThat;
import java.util.HashMap;
import java.util.Map;
import org.junit.Test;
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import\ org. spring framework. http. Response Entity;
import org.springframework.util.LinkedMultiValueMap;
import org.springframework.util.MultiValueMap;
import\ org. spring framework. we b. c \ lient. Rest Template;
import com.pachiraframework.springcloud.message.dto.SmsSendRequest;
import com.pachiraframework.springcloud.message.dto.SmsSendResponse;
public class SmsControllerTest {
private static final String SEND_URL = "http://localhost:8080/message/sms/send";
private static final String SEND2_URL = "http://localhost:8080/message/sms/send2";
private RestTemplate restTemplate = new RestTemplate();
@Test
public void testSend() {
 HttpHeaders headers = new HttpHeaders();
 headers.setContentType(MediaType.APPLICATION\_FORM\_URLENCODED);
 MultiValueMap<String, String> map= new LinkedMultiValueMap<String, String>();
 map.add("mobile", "18562875992");
 map.add("templateId", "CHECK_CODE");
 map.add("params['code']", "123456");
 HttpEntity<MultiValueMap<String, String>> request = new HttpEntity<MultiValueMap<String, String>>(map, headers);
 ResponseEntity<SmsSendResponse> response = restTemplate.postForEntity(SEND_URL, request, SmsSendResponse.class);
 assertThat(response.getStatusCode(), equalTo(HttpStatus.OK));
 assertThat(response.getBody(), notNullValue());
 SmsSendResponse sendResponse = response.getBody();
 assertThat(sendResponse.getCode(), equalTo("200"));
 assertThat(sendResponse.getMessage(), equalTo("发送成功"));
}
//这种方式客户端直接传递SmsSendRequest参数,RestTemplate内部会将其转换成json传传输
@Test
public void testSend2() {
 SmsSendRequest request = new SmsSendRequest();
 request.setMobile("18562875992");
 request.setTemplateId("CHECK CODE");
 Map<String, Object> params = new HashMap<String, Object>();
 params.put("code", "123456");
 request.setParams(params);
 ResponseEntity<SmsSendResponse> response = restTemplate.postForEntity(SEND2 URL, request, SmsSendResponse.class);
```

```
assertThat(response.getStatusCode(), equalTo(HttpStatus.OK));
  assertThat(response.getBody(), notNullValue());
  SmsSendResponse = response.getBody();
  assertThat(sendResponse.getCode(),\ equalTo("200"));\\
  assertThat(sendResponse.getMessage(), equalTo("发送成功"));
 }
执行测试用例
                          型 MessageAppl...
型 SmsServiceI...
型 SmsSendRespo...
型 MessageTemp...
型 SmsControlle...
図 J SmsControlle...
public class SmsControllerTest {
            private static final String SEND_URL = "http://localhost:8080/message/sms/send";
private static final String SEND_URL = "http://localhost:8080/message/sms/send2";
private RestTemplate restTemplate = new RestTemplate();
  276
28
29
30
31
32
33
34
35
36
37
             public void testSend() {
   HttpHeaders headers = new HttpHeaders();
   headers.setContentType(MediaType.APPLICATION_FORM_URLENCODED);
                  MultiValueMap<String, String> map= new LinkedMultiValueMap<String, String>(); map.add("mobile", "18562875992"); map.add("templateId", "CHECK_CODE"); map.add("params['code']", "123456");
                  HttpEntity<MultiValueMap<String, String>> request = new HttpEntity<MultiValueMap<String, String>>(map, headers);
ResponseEntity<SmsSendResponse> response = restTemplate.postForEntity(SEND_URL, request, SmsSendResponse.class);
assertThat(response.getStatusCode(), equalTo(HttpStatus.OK));
assertThat(response_getBody(), entital());
                                                                                                                                                                          ↓ ↑ № □ □ □ ▼
 🔊 Tasks 🗓 Display 🚮 JUnit 🛭 🍰 Call Hierarchy 📮 Console 🔗 Search
 Finished after 0.695 seconds

    □ Errors: 0     □ Failures: 0
                                                                                                                                                                                                                         5
  🗸 🔃 com.pachiraframework.springcloud.message.controller.SmsControllerTest [Runner: JUnit 4] (C 🗏 Failure Trace
         # testSend2 (0.261 s)
        testSend (0.017 s)
```

#### 源码地址:

https://github.com/kwang2003/springcloud-message.git

#### 附:工程搭建过程中的常见问题及解决办法

• 使用eclipse向导创建工程后,工程的默认jdk是1.6

如下图所示,系统中已经没有jdk1.6的版本了,但是自动创建的工程仍然标识为1.6



由于使用了最新的springcloud组件,必须在jdk1.8或以上版本上运行,因此需要做出改变。 修改settings.xml文件,添加一个jdk-1.8的profile,并启用它

```
profile>
         <id>jdk-1.8</id>
          <activation>
             <activeByDefault>true</activeByDefault>
             <jdk>1.8</jdk>
F
         </activation>
          cproperties>
              <maven.compiler.source>1.8</maven.compiler.source>
              <maven.compiler.target>1.8</maven.compiler.target>
              <maven.compiler.compilerVersion>1.8</maven.compiler.compilerVersion>
上早早
     </profile>
     cprofile>
       <1d>jdk-1.6</1d>
          <activation>
            <jdk>1.6</jdk>
F
         </activation>
          cproperties>
              <maven.compiler.source>1.6</maven.compiler.source>
             <maven.compiler.target>1.6</maven.compiler.target>
             <maven.compiler.compilerVersion>1.6</maven.compiler.compilerVersion>
          </properties>
      </profile>
   </profiles>
<activeProfiles>
     <activeProfile>artifactory</activeProfile>
<activeProfile>jdk-1.8</activeProfile>
    </activeProfiles>
      做完这些修改后,重新创建工程,可以看到新建的工程已经使用idk1.8了,如图
springcloud-message
   > 乃 src/main/java
   > 乃 src/test/java
   JRE System Library [JavaSE-1.8]

Mayen Dependent
   > 🗁 src
     target
```