1. 项目概要介绍

本节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。

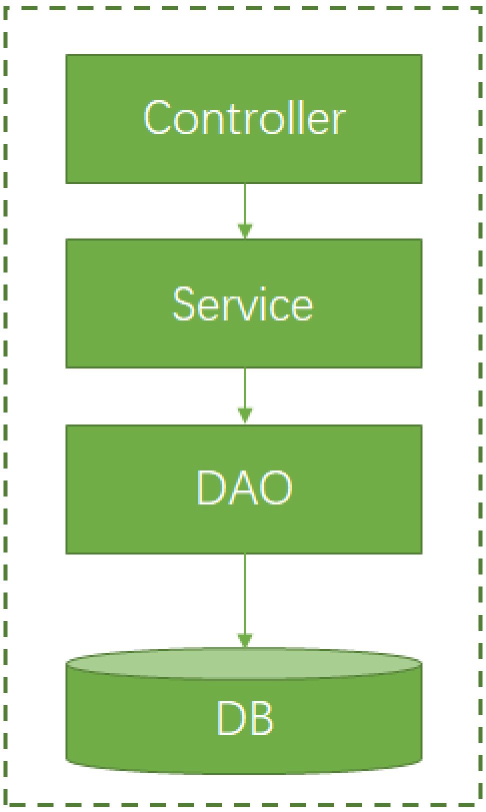
2. 项目所涉及到的技术或工具

核心技术选型：

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

3. 项目架构

系统分层架构：



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

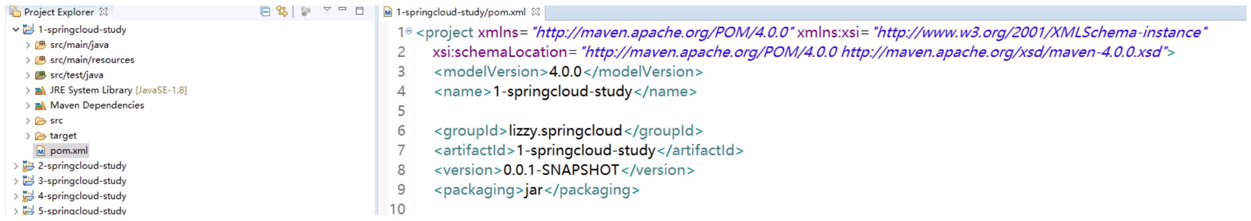
使用Eclipse的Maven工程创建向导创建一个maven单模块项目，这里我们命名为springcloud-message,你也可以根据自己的喜好，选择自己熟练的IDE来创建工程，创建好项目的基本信息如下

group:lizzy.springcloud

artifact-id:springcloud-study

version:0.0.1-SNAPSHOT

packing:jar



```xml
<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>
  <name>1-springcloud-study</name>

  <groupId>lizzy.springcloud</groupId>
  <artifactId>1-springcloud-study</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>
```

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，JDK版本和Encoding设置，这在以后的工程中均统一设定。**

```
<parent>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-parent</artifactId>
			<version>1.5.8.RELEASE</version>
		</parent>
		<properties>
			<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
			<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
			<java.version>1.8</java.version>
		</properties>
```

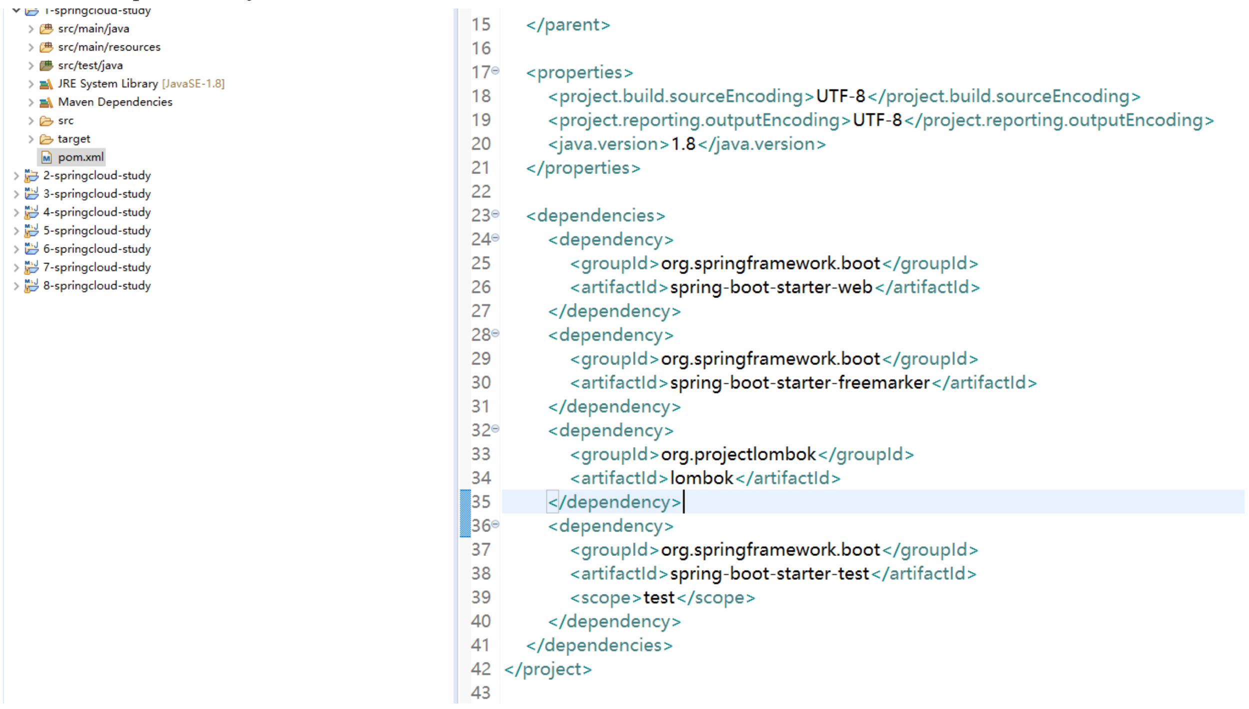
```xml
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>1.5.8.RELEASE</version>
</parent>

<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
  <java.version>1.8</java.version>
</properties>
```

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>
```

引入freemarker

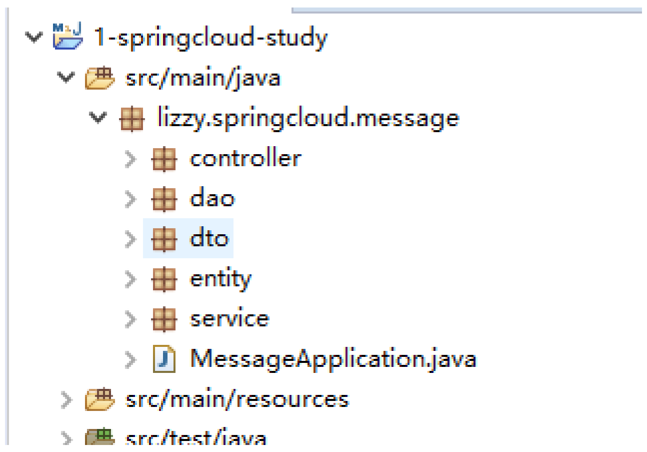
```
<dependency>
    <groupId>org. springframework. boot</groupId>
    <artifactId>spring-boot-starter-freemarker</artifactId>
</dependency>
```
在工程中我们将采用freemarker模版技术来为短信模版处理变量替换。
然后引入lombok和SpringBoot配套的JUnit包。
```
<dependency>
        <groupId>org. projectlombok</groupId>
    <artifactId>lombok</artifactId>
</dependency>
<dependency>
        <groupId>org. springframework. boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
</dependency>
```


```
    </parent>

    <properties>
        <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-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-freemarker</artifactId>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>
```

lombok引入后，导入工程会报错，需正确加载配置到Eclipse中。

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


1-springcloud-study
src/main/java
lizzy.springcloud.message
controller
dao
dto
entity
service
MessageApplication.java
src/main/resources
src/test/java

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：

- 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.java

```
package lizzy.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.java

```
package lizzy.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;
        /**
         * 要发送的手机号
         */
        private String mobile;
        /**
         * 模版中携带的参数信息
         */
        private Map<String, Object> params;
}
```

SmsSendResponse.java

```
package lizzy.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;
}
```

MessageTemplateDao.java
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 lizzy.springcloud.message.dao;
import org.springframework.stereotype.Repository;
import lizzy.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.java
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 lizzy.springcloud.message.service.impl;

import java.io.StringReader;
import java.io.StringWriter;

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

import freemarker.template.Configuration;
import freemarker.template.Template;
import lizzy.springcloud.message.dao.MessageTemplateDao;
import lizzy.springcloud.message.dto.SmsSendRequest;
import lizzy.springcloud.message.dto.SmsSendResponse;
import lizzy.springcloud.message.entity.MessageTemplate;
import lizzy.springcloud.message.service.SmsService;
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());
                String templateContent = messageTemplate.getContent();
                Template template = new Template(request.getTemplateId(), new
StringReader(templateContent), configuration);
                StringWriter out = new StringWriter();
                template.process(request.getParams(), out);
                String content = out.toString();
                return doSend(request.getMobile(), content);
        }

        //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;
        }
}
```

SmsService.java

```
package lizzy.springcloud.message.service;

import lizzy.springcloud.message.dto.SmsSendRequest;
import lizzy.springcloud.message.dto.SmsSendResponse;
```

```
public interface SmsService {
        public SmsSendResponse send(SmsSendRequest request);
}
```

SmsController.java
Rest服务中我们使用了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 lizzy.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.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

import lizzy.springcloud.message.dto.SmsSendRequest;
import lizzy.springcloud.message.dto.SmsSendResponse;
import lizzy.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);
        }
}
```

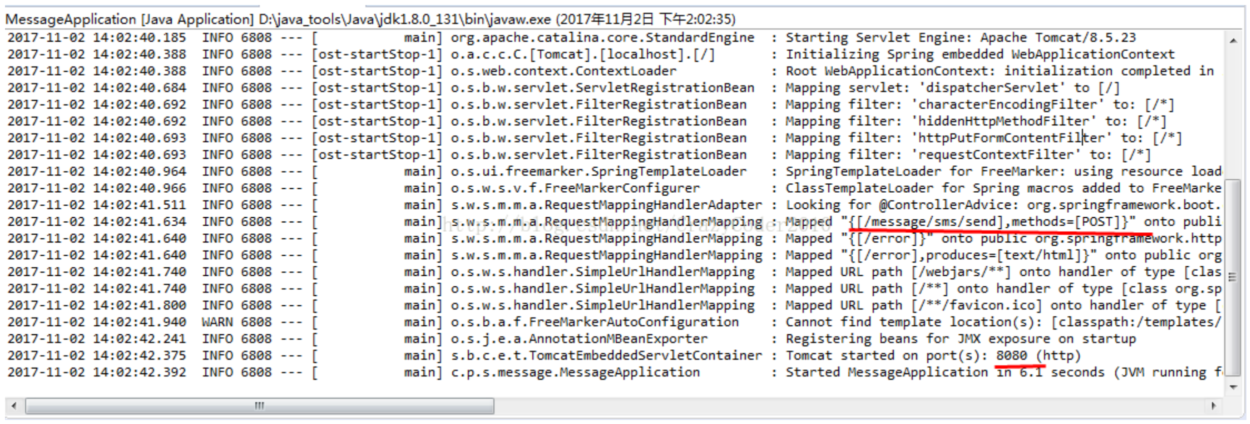
MessageApplication.java

通过一个main函数将服务启动起来

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

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

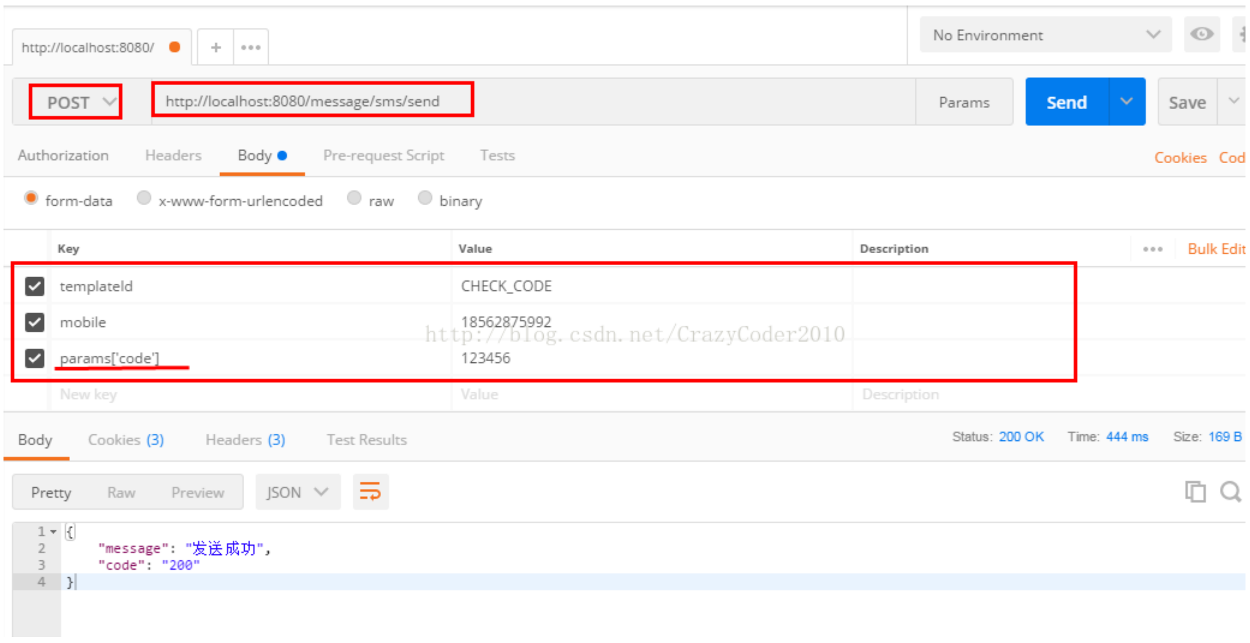
请求URL:http://localhost:8080/message/sms/send

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



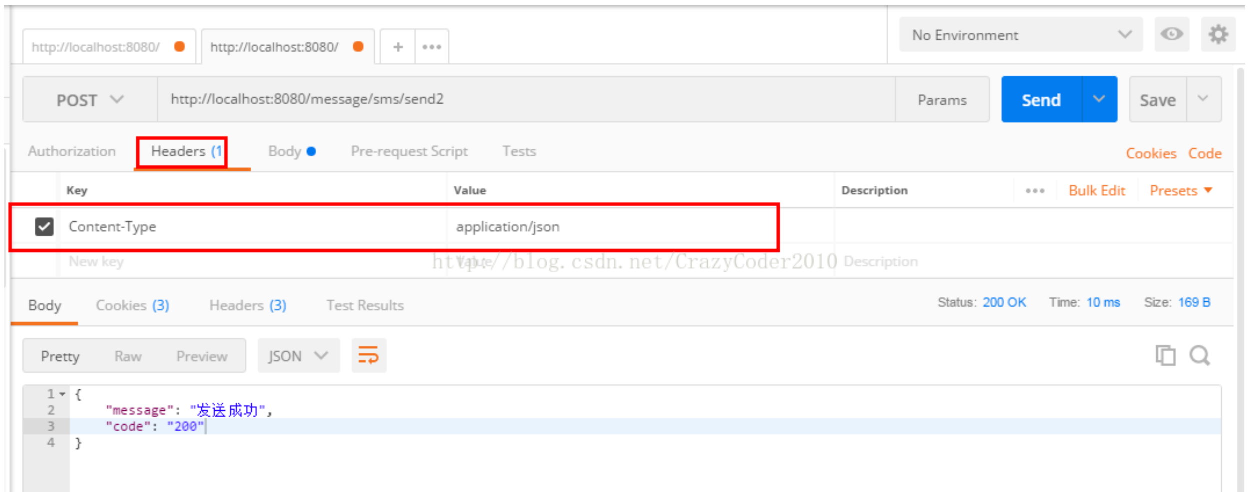
POST http://localhost:8080/message/sms/send

Key	Value	Description
templateId	CHECK_CODE	
mobile	18562875992	
params['code']	123456	

Body Cookies (3) Headers (3) Test Results — Status: 200 OK Time: 444 ms Size: 169 B

```
{
    "message": "发送成功",
    "code": "200"
}
```

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



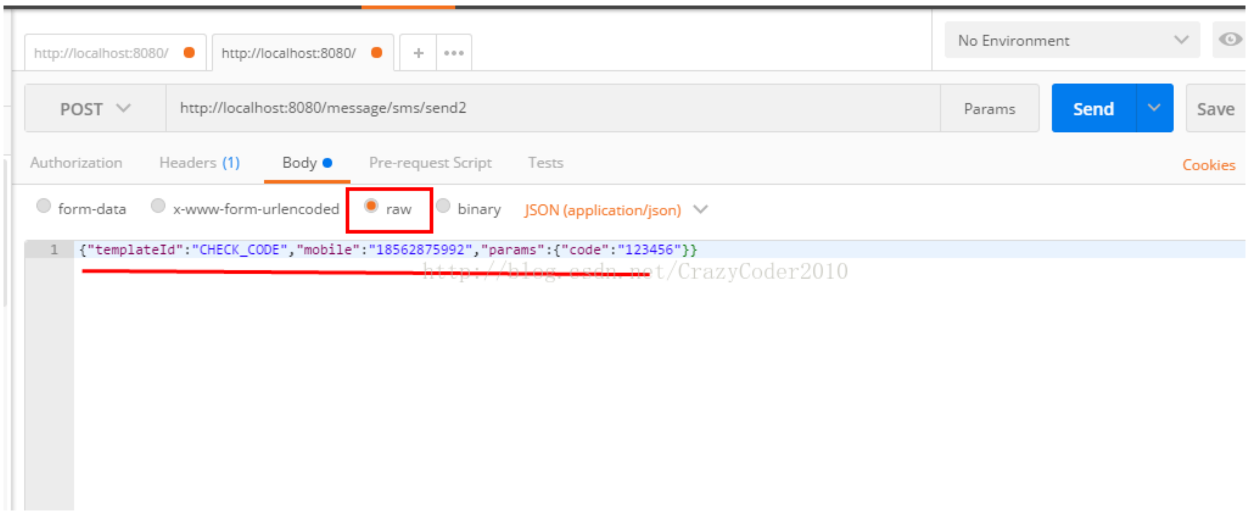
POST http://localhost:8080/message/sms/send2

Key	Value	Description
Content-Type	application/json	

Body Cookies (3) Headers (3) Test Results — Status: 200 OK Time: 10 ms Size: 169 B

```
{
    "message": "发送成功",
    "code": "200"
}
```

然后设置post内容，注意设置类型是raw


POST http://localhost:8080/message/sms/send2

Body: raw, JSON (application/json)

```
{"templateId":"CHECK_CODE","mobile":"18562875992","params":{"code":"123456"}}
```

点击发送，得到的结果和上个一样的

9.通过RestTemplate调用Rest服务

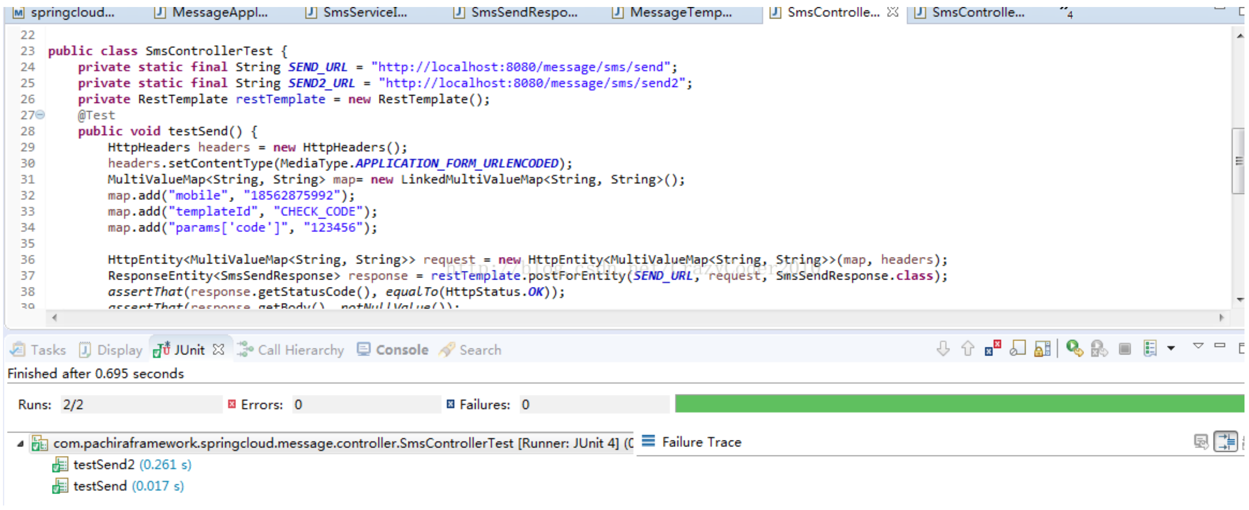
针对2中不同方式的rest方法，使用RestTemplate调用的方式也是不一样的，如下代码所示：

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 lizzy.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.springframework.http.ResponseEntity;
import org.springframework.util.LinkedMultiValueMap;
import org.springframework.util.MultiValueMap;
import org.springframework.web.client.RestTemplate;

import lizzy.springcloud.message.dto.SmsSendRequest;
import lizzy.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 sendResponse = response.getBody();
```

```
                assertThat(sendResponse.getCode(), equalTo("200"));
                assertThat(sendResponse.getMessage(), equalTo("发送成功"));
        }


}
```

执行测试用例



```
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());
```

```
Finished after 0.695 seconds
Runs:  2/2        Errors:  0        Failures:  0

com.pachiraframework.springcloud.message.controller.SmsControllerTest [Runner: JUnit 4] (C
    testSend2 (0.261 s)
    testSend (0.017 s)
```