第三章 SpringCloud 使用Erueak服务注册中心

1.先前架构的问题及改进

通过前两章的学习，我们已经掌握了通过SpringCloud/SpringBoot来提供一个Rest服务接口，并且可以通过RestTemplate来调用服务，整个世界看起来非常清洁:


Springcloud-message

但是这个架构是非常脆弱的，在生产环境下，我们的message服务不可能只部署一份，因为这样意味着该节点一旦出现问题或宕机，client端就没法使用了，同时服务节点多了就意味着能应对更大的客户端访问流量，于是我们的架构可能升级到下面的样子:

我们会考虑中间建一层nginx作为负载均衡代理，客户端只对nginx接口进行访问，由nginx通过路由规则将请求分发到下游的各个节点上的服务
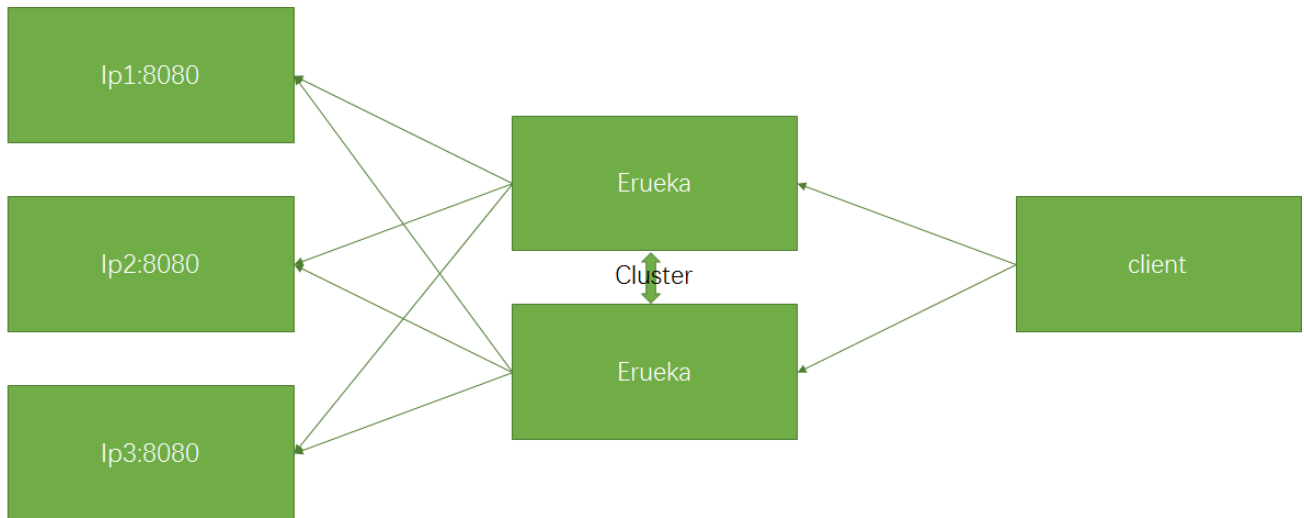

Springcloud-message
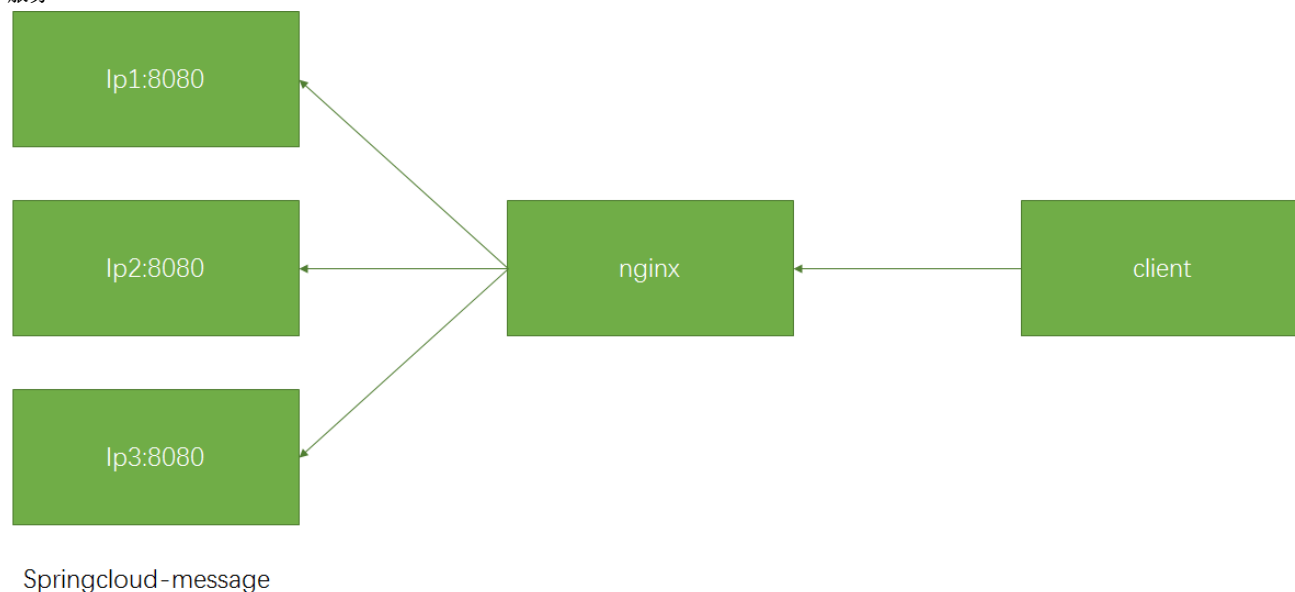
通过如上的改造，我们的服务节点单点故障解决了，访问压力也能通过nginx分发到各个节点上执行了，是否这样就完美了呢？如果你仔细看的话，会发现，其实nginx还是存在单点故障，也就是说，一旦nginx当即故障，client端的调用就全完了---即便这个时候所有的message服务都可用，于是，我们的架构不得不再次升级，来解决nginx的单点故障问题，于是架构可能升级到了如下版本:

Springcloud-message
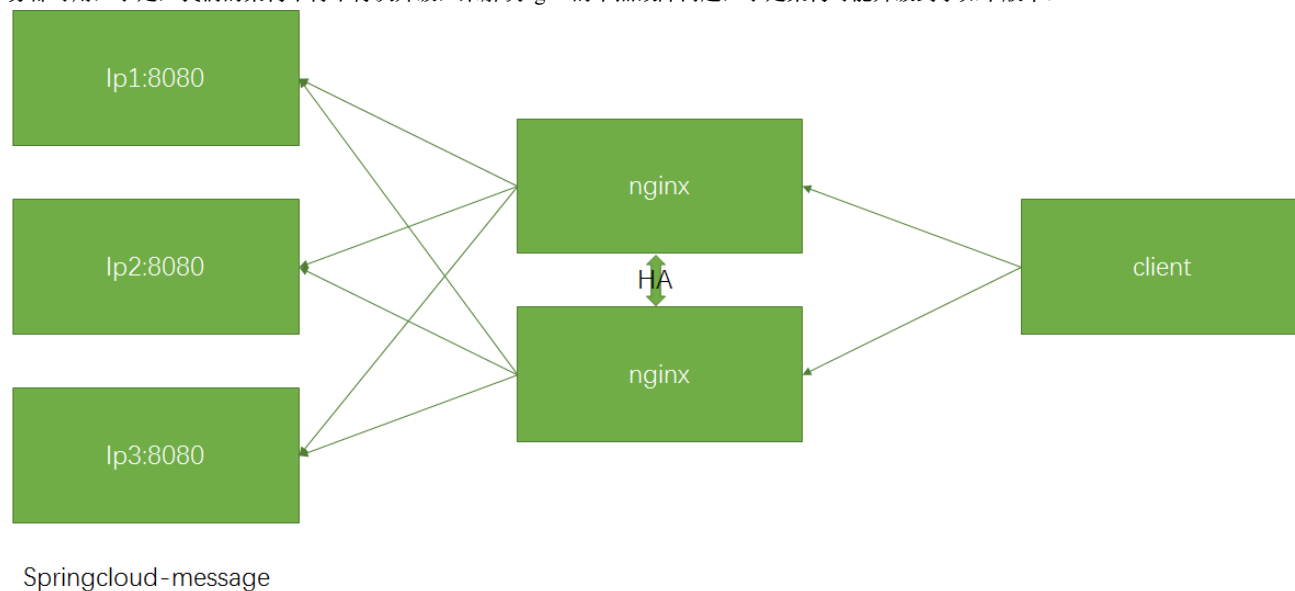
这个架构看起来比较完美了吧，nginx和message服务的单点故障都解决了，但是忽然有一天，我们的系统访问压力增加了，需要快速的部

署一些新的message节点上来，当一段时间以后，当访问压力下来的时候我们又得对这些节点资源进行回收---如电商的双11节点，这个时候运维工程师不得不去nginx服务器上去逐一设置新增节点的ip和端口增加上去，然后重新加载nginx，当节日过后，他们又不得不逐一对已经增加过的节点进行下架，重启nginx---说了半天什么意思呢，这架构对于运维节点并不是非常友好，当节点发生变化的时候，必须有人为介入进行处理，一旦有手工处理的地方就意味着有出错的可能性，服务节点数量少的时候没问题，当服务节点成千上万，经常变化的时候呢？计算机程序能自动处理的事情就不要让人去处理，因为计算机比人要严谨的多，于是引入了这一章节的主角-Eureka

Springcloud-message

采用了Eureak的架构和nginx的架构非常类似，但是eureak的先进性在于当有服务节点的增删时，并不需要人为手动干预，而是服务启动时自动将服务节点信息注册到Eureka注册中心来，而当服务节点宕机故障时，Eureka自动将故障节点排除，客户端通过服务ID来调用注册中心中的服务，让后通过一定的负载均衡算法（轮询、权重、绑定。。）找到一个活动状态的服务节点进行通信。同时Erueka多个节点之间可以组成一个集群，集群中的Eureak之间保持数据的同步，因此只要Eureka集群中有一个节点存活，真个集群就处于可用状态

2.启动Eureka服务器

a)新建一个工程，并添加eureka相关的依赖

设置parent

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

加入springcloud的声明

```
<dependencyManagement>
  <dependencies>
   <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-dependencies</artifactId>
    <version>Dalston.SR4</version>
    <type>pom</type>
    <scope>import</scope>
   </dependency>
  </dependencies>
 </dependencyManagement>
```

引入eureka server依赖

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-eureka-server</artifactId>
</dependency>
```

完整的项目依赖如下

b)main函数中启用Erueka的注解

```java
package com.pachiraframework.springcloud.eureka;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@EnableEurekaServer
@SpringBootApplication
public class EurekaApplication {
 public static void main(String[] args) {
  SpringApplication.run(EurekaApplication.class, args);
 }
}
```
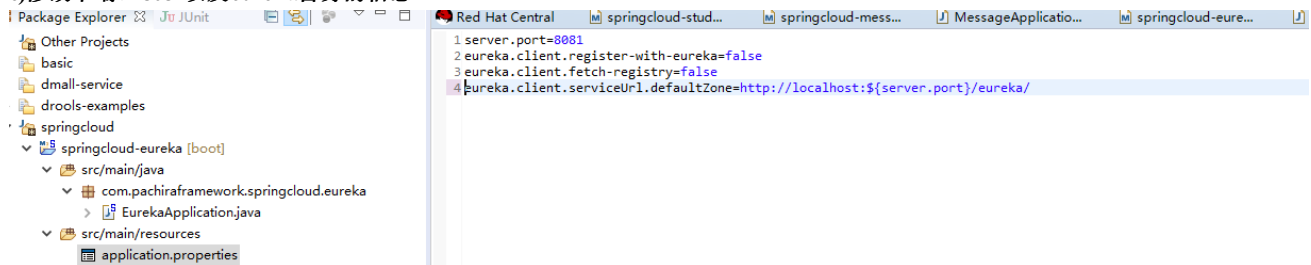
c)修改下端口8081以及eureka自身的信息

d)启动main函数

```
2017-11-04 07:03:23.805  INFO 6796 --- [           main] o.s.b.a.e.mvc.EndpointHandlerMapping     : Mapped "{[/mappings || /mappings.json],methods=[GET],produces=[application/vnd
2017-11-04 07:03:23.806  INFO 6796 --- [           main] o.s.b.a.e.mvc.EndpointHandlerMapping     : Mapped "{[/health || /health.json],methods=[GET],produces=[application/vnd.spr
2017-11-04 07:03:23.808  INFO 6796 --- [           main] o.s.b.a.e.mvc.EndpointHandlerMapping     : Mapped "{[/pause || /pause.json],methods=[POST]}" onto public java.lang.Object
2017-11-04 07:03:23.808  INFO 6796 --- [           main] o.s.b.a.e.mvc.EndpointHandlerMapping     : Mapped "{[/features || /features.json],methods=[GET],produces=[application/vn
2017-11-04 07:03:23.808  INFO 6796 --- [           main] o.s.b.a.e.mvc.EndpointHandlerMapping     : Mapped "{[/archaius || /archaius.json],methods=[GET],produces=[application/vn
2017-11-04 07:03:23.809  INFO 6796 --- [           main] o.s.b.a.e.mvc.EndpointHandlerMapping     : Mapped "{[/env/{name:.*}],methods=[GET],produces=[application/vnd.spring-boot
2017-11-04 07:03:23.809  INFO 6796 --- [           main] o.s.b.a.e.mvc.EndpointHandlerMapping     : Mapped "{[/env || /env.json],methods=[GET],produces=[application/vnd.spring-bi
2017-11-04 07:03:23.809  INFO 6796 --- [           main] o.s.b.a.e.mvc.EndpointHandlerMapping     : Mapped "{[/info || /info.json],methods=[GET],produces=[application/vnd.spring
2017-11-04 07:03:23.809  INFO 6796 --- [           main] o.s.b.a.e.mvc.EndpointHandlerMapping     : Mapped "{[/dump || /dump.json],methods=[GET],produces=[application/vnd.spring
2017-11-04 07:03:23.810  INFO 6796 --- [           main] o.s.b.a.e.mvc.EndpointHandlerMapping     : Mapped "{[/resume || /resume.json],methods=[POST]}" onto public java.lang.Obj
2017-11-04 07:03:23.937  INFO 6796 --- [           main] o.s.ui.freemarker.SpringTemplateLoader   : SpringTemplateLoader for FreeMarker: using resource loader [org.springframewo
2017-11-04 07:03:23.939  INFO 6796 --- [           main] o.s.w.s.v.f.FreeMarkerConfigurer         : ClassTemplateLoader for Spring macros added to FreeMarker configuration
2017-11-04 07:03:24.068  INFO 6796 --- [           main] o.s.c.n.eureka.InstanceInfoFactory       : Setting initial instance status as: STARTING
2017-11-04 07:03:24.119  INFO 6796 --- [           main] com.netflix.discovery.DiscoveryClient    : Initializing Eureka in region us-east-1
2017-11-04 07:03:24.119  INFO 6796 --- [           main] com.netflix.discovery.DiscoveryClient    : Client configured to neither register nor query for data.
2017-11-04 07:03:24.129  INFO 6796 --- [           main] com.netflix.discovery.DiscoveryClient    : Discovery Client initialized at timestamp 1509750204128 with initial instance
2017-11-04 07:03:24.210  INFO 6796 --- [           main] c.n.eureka.DefaultEurekaServerContext    : Initializing ...
2017-11-04 07:03:24.212  WARN 6796 --- [           main] c.n.eureka.cluster.PeerEurekaNodes       : The replica size seems to be empty. Check the route 53 DNS Registry
2017-11-04 07:03:24.223  INFO 6796 --- [           main] c.n.e.registry.AbstractInstanceRegistry  : Finished initializing remote region registries. All known remote regions: []
2017-11-04 07:03:24.223  INFO 6796 --- [           main] c.n.eureka.DefaultEurekaServerContext    : Initialized
2017-11-04 07:03:24.403  INFO 6796 --- [           main] o.s.j.e.a.AnnotationMBeanExporter         : Registering beans for JMX exposure on startup
2017-11-04 07:03:24.412  INFO 6796 --- [           main] o.s.j.e.a.AnnotationMBeanExporter         : Bean with name 'environmentManager' has been autodetected for JMX exposure
2017-11-04 07:03:24.414  INFO 6796 --- [           main] o.s.j.e.a.AnnotationMBeanExporter         : Bean with name 'configurationPropertiesRebinder' has been autodetected for JMI
2017-11-04 07:03:24.414  INFO 6796 --- [           main] o.s.j.e.a.AnnotationMBeanExporter         : Bean with name 'refreshEndpoint' has been autodetected for JMX exposure
2017-11-04 07:03:24.415  INFO 6796 --- [           main] o.s.j.e.a.AnnotationMBeanExporter         : Bean with name 'restartEndpoint' has been autodetected for JMX exposure
2017-11-04 07:03:24.415  INFO 6796 --- [           main] o.s.j.e.a.AnnotationMBeanExporter         : Bean with name 'serviceRegistryEndpoint' has been autodetected for JMX exposu
2017-11-04 07:03:24.416  INFO 6796 --- [           main] o.s.j.e.a.AnnotationMBeanExporter         : Bean with name 'refreshScope' has been autodetected for JMX exposure
2017-11-04 07:03:24.418  INFO 6796 --- [           main] o.s.j.e.a.AnnotationMBean                 : Located managed bean 'environmentManager': registering with JMX server as MBea
2017-11-04 07:03:24.431  INFO 6796 --- [           main] o.s.j.e.a.AnnotationMBean                 : Located managed bean 'restartEndpoint': registering with JMX server as MBean
2017-11-04 07:03:24.444  INFO 6796 --- [           main] o.s.j.e.a.AnnotationMBean                 : Located managed bean 'serviceRegistryEndpoint': registering with JMX server a:
2017-11-04 07:03:24.449  INFO 6796 --- [           main] o.s.j.e.a.AnnotationMBean                 : Located managed bean 'refreshScope': registering with JMX server as MBean [or
2017-11-04 07:03:24.461  INFO 6796 --- [           main] o.s.j.e.a.AnnotationMBean                 : Located managed bean 'configurationPropertiesRebinder': registering with JMX :
2017-11-04 07:03:24.466  INFO 6796 --- [           main] o.s.j.e.a.AnnotationMBean                 : Located managed bean 'refreshEndpoint': registering with JMX server as MBean
2017-11-04 07:03:24.680  INFO 6796 --- [           main] o.s.c.support.DefaultLifecycleProcessor  : Starting beans in phase 0
2017-11-04 07:03:24.681  INFO 6796 --- [           main] o.s.c.n.e.s.EurekaServiceRegistry        : Registering application unknown with eureka with status UP
2017-11-04 07:03:24.716  INFO 6796 --- [     Thread-11] o.s.c.n.e.server.EurekaServerBootstrap   : Setting the eureka configuration..
2017-11-04 07:03:24.717  INFO 6796 --- [     Thread-11] o.s.c.n.e.server.EurekaServerBootstrap   : Eureka data center value eureka.datacenter is not set, defaulting to default
2017-11-04 07:03:24.717  INFO 6796 --- [     Thread-11] o.s.c.n.e.server.EurekaServerBootstrap   : Eureka environment value eureka.environment is not set, defaulting to test
2017-11-04 07:03:24.728  INFO 6796 --- [     Thread-11] o.s.c.n.e.server.EurekaServerBootstrap   : isAws returned false
2017-11-04 07:03:24.729  INFO 6796 --- [     Thread-11] o.s.c.n.e.server.EurekaServerBootstrap   : Initialized server context
2017-11-04 07:03:24.729  INFO 6796 --- [     Thread-11] c.n.e.r.PeerAwareInstanceRegistryImpl    : Got 1 instances from neighboring DS node
2017-11-04 07:03:24.729  INFO 6796 --- [     Thread-11] c.n.e.r.PeerAwareInstanceRegistryImpl    : Renew threshold is: 1
2017-11-04 07:03:24.729  INFO 6796 --- [     Thread-11] c.n.e.r.PeerAwareInstanceRegistryImpl    : Changing status to UP
2017-11-04 07:03:24.735  INFO 6796 --- [     Thread-11] e.s.EurekaServerInitializerConfiguration : Started Eureka Server
2017-11-04 07:03:24.781  INFO 6796 --- [           main] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat started on port(s): 8081 (http)
2017-11-04 07:03:24.782  INFO 6796 --- [           main] .s.c.n.e.s.EurekaAutoServiceRegistration : Updating port to 8081
2017-11-04 07:03:24.786  INFO 6796 --- [           main] c.p.s.eureka.EurekaApplication           : Started EurekaApplication in 8.89 seconds (JVM running for 9.441)
2017-11-04 07:03:27.771  INFO 6796 --- [nio-8081-exec-2] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring FrameworkServlet 'dispatcherServlet'
2017-11-04 07:03:27.771  INFO 6796 --- [nio-8081-exec-2] o.s.web.servlet.DispatcherServlet        : FrameworkServlet 'dispatcherServlet': initialization started
2017-11-04 07:03:27.792  INFO 6796 --- [nio-8081-exec-2] o.s.web.servlet.DispatcherServlet        : FrameworkServlet 'dispatcherServlet': initialization completed in 21 ms
2017-11-04 07:04:24.730  INFO 6796 --- [a-EvictionTimer] c.n.e.registry.AbstractInstanceRegistry  : Running the evict task with compensationTime 0ms
```
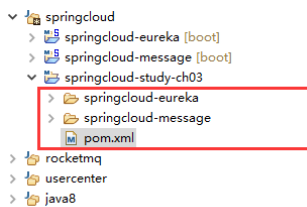
e)访问

http://localhost:8081访问注册中心
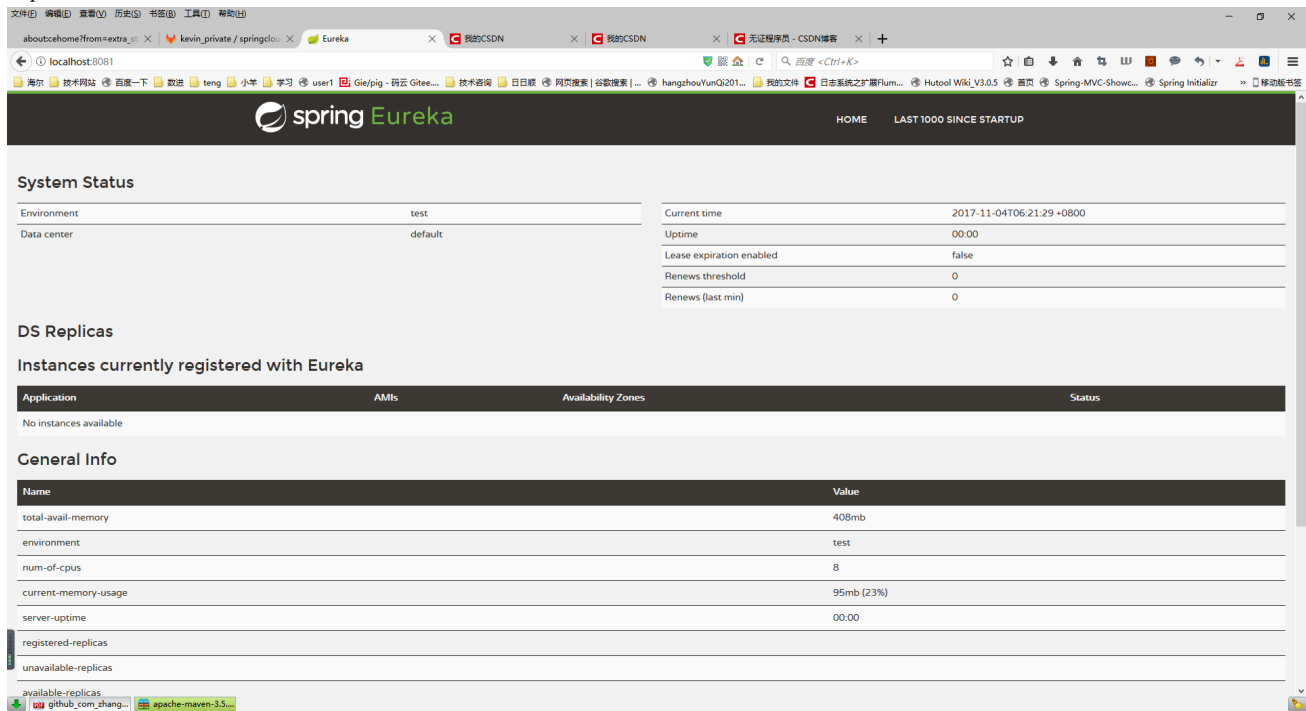
3.将message服务注册到注册中心

a)工程结构重构

通过对上一节中的message服务进行稍微的改造，就可以将其注册到注册中心里来了

为了对每个章节中的代码能够保持独立，每个章节都设置了一个端独的git工程，因为这个章节用到了上一章的代码进行改造，如果不加以独立，我们回头来翻阅代码的时候就很难找到先前的代码了。如图所示，我们通过maven的多模块机制将第一章的message作为本章的一个子模块引入进来，代码完全一样，另一个模块是刚新增的eureka模块，通过这种联系把2个组件绑定在一个工程结构中。

b)引入eureka依赖

增加eureka的客户端依赖

引入springcloud声明

```xml
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-dependencies</artifactId>
      <version>Dalston.SR4</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

引入eureka客户端

```xml
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-eureka</artifactId>
</dependency>
```

如图所示：
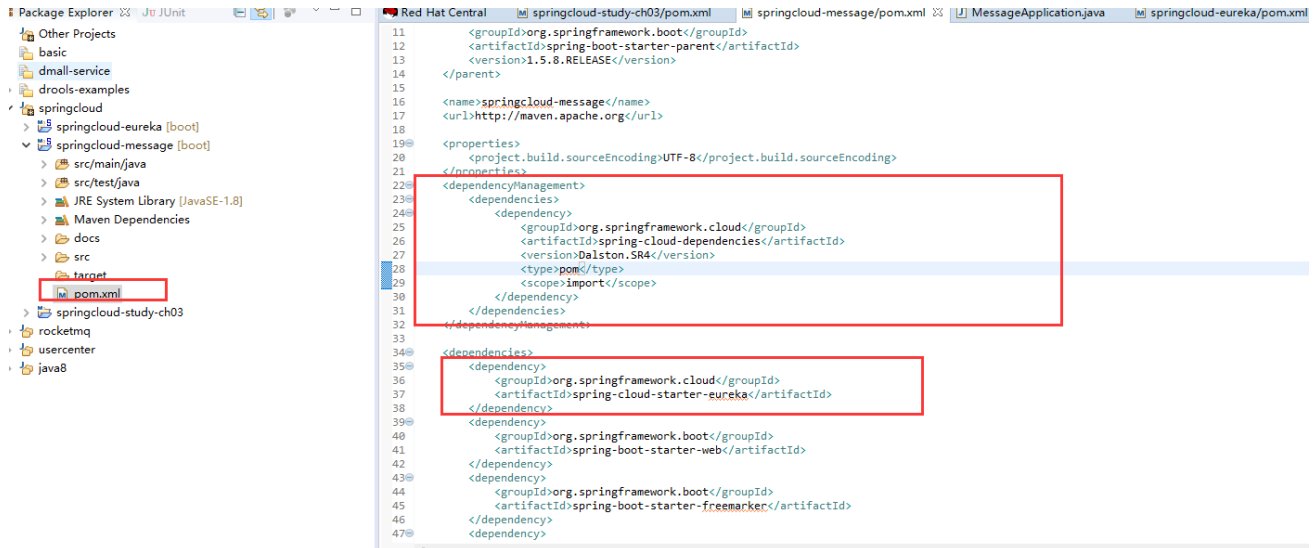
c)添加Eureka注解

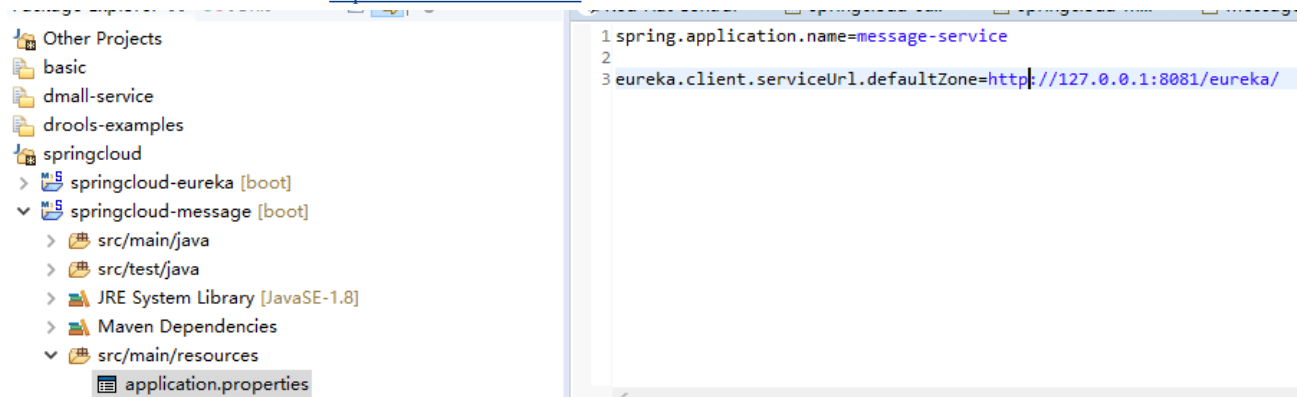在MessageApplication上添加@EnableDiscoveryClient注解
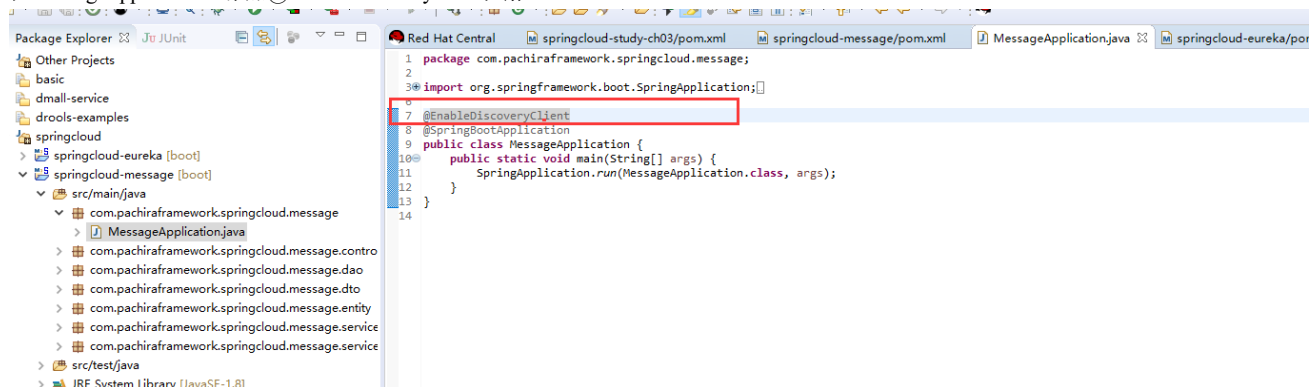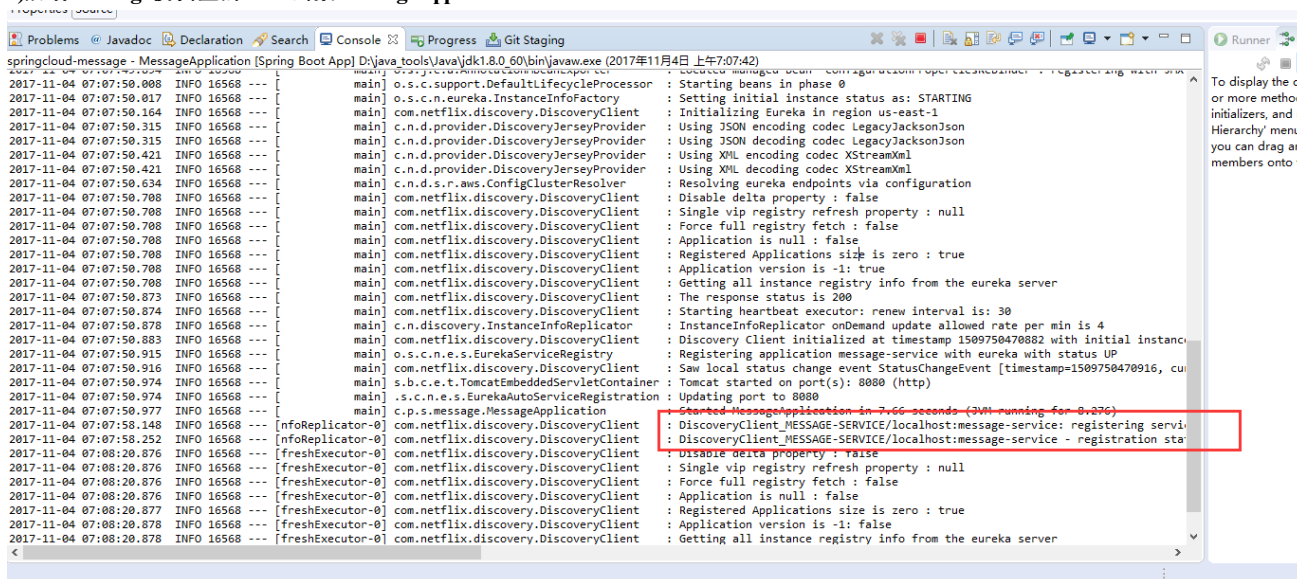
d)message服务中设置注册中心地址

打开message服务项目中的application.properties文件，设置一下服务的名称和注册中心地址，注意为每一个微服务设置一个有意义的，唯一的名称是非常必要的

spring.application.name=message-service

eureka.client.serviceUrl.defaultZone=http://127.0.0.1:8081/eureka/

e)启动message模块里的main函数MessageApplication.main

从日志可以看到，message服务已经注册到eureka中了

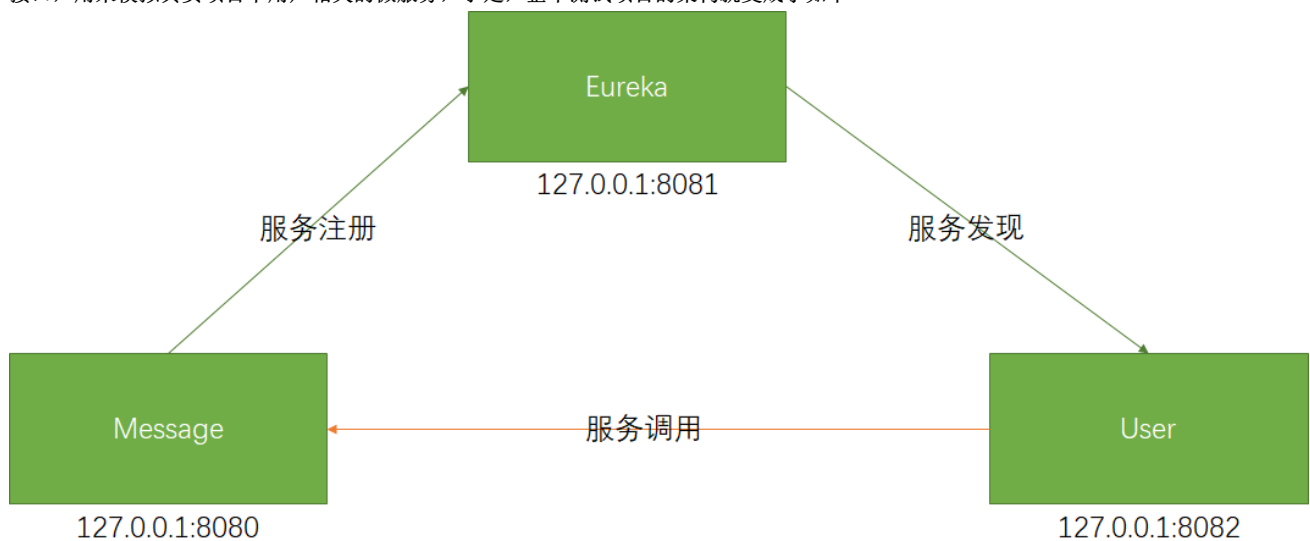**f)通过eureka控制台查看已经注册的服务**

http://127.0.0.1:8081
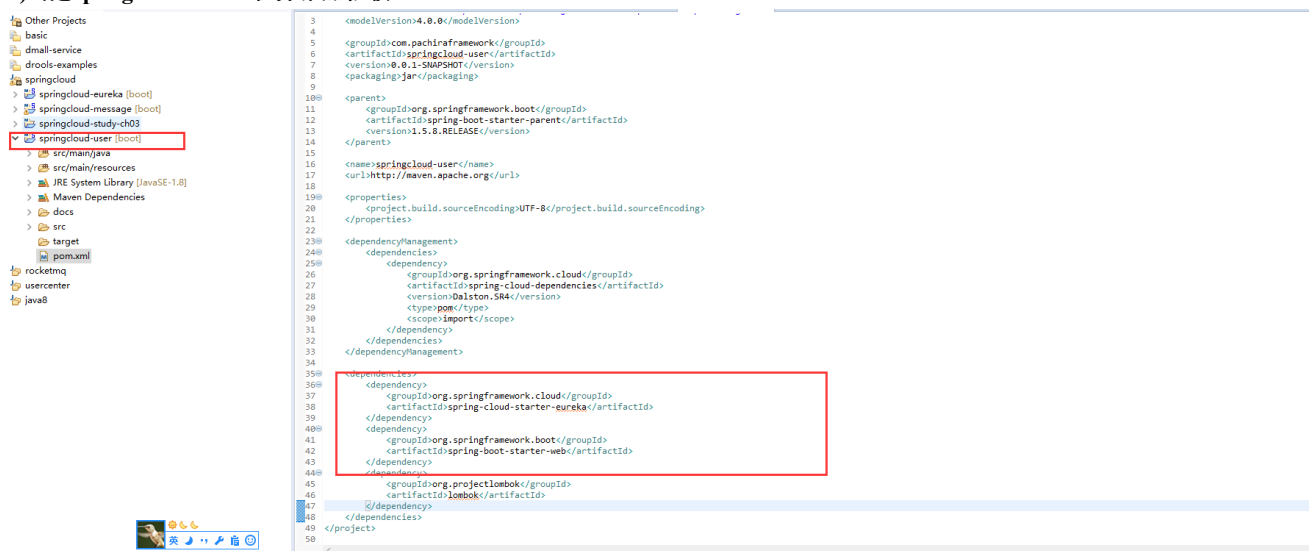
可以看到我们在步骤d)中设置的微服务名称已经出现在了服务列表中了
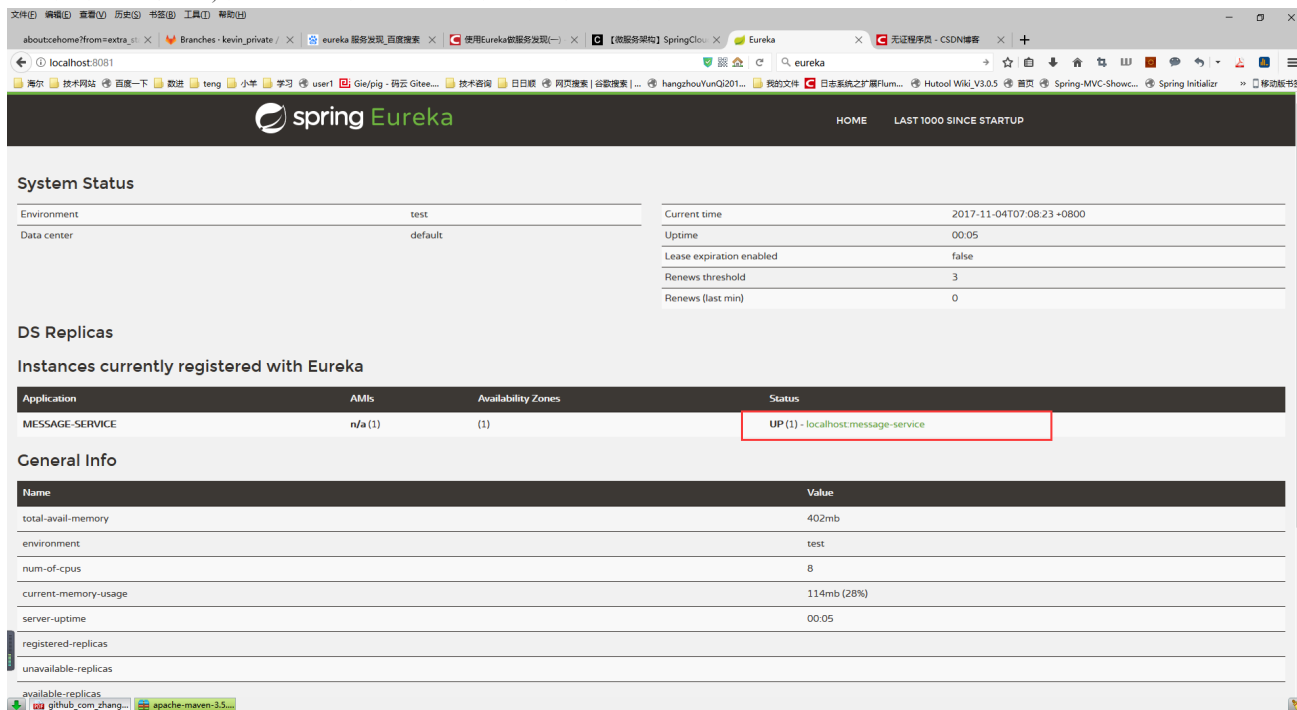
4.基于注册中心调用服务

在上个章节中，我们提供了2中方式进行Rest接口的调用，为了更加真实的模拟真实项目，我们在这个章节中对服务调用者进行了分离，单独成一个user-service，user-service中有一个用户注册的功能，当用户注册时，需要给用户发送短信验证码，从而调用message服务中的短信接口，用来模拟真实项目中用户相关的微服务，于是，整个测试项目的架构就变成了如下
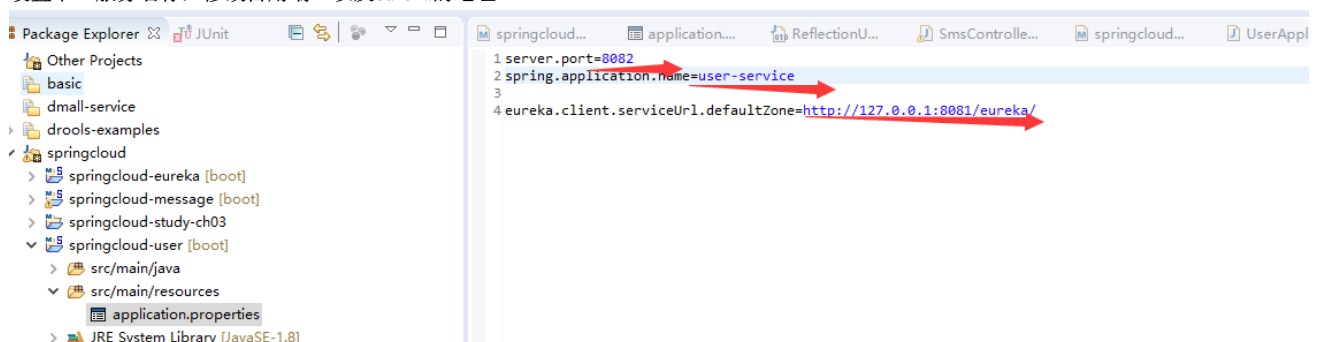

a)创建springcloud-user工程并添加依赖

b)增加配置项application.properties

设置唯一服务名称，修改占用端口以及eureka的地址

c)编写项目启动函数UserApplication

package com.pachiraframework.springcloud.user;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@EnableDiscoveryClient

@SpringBootApplication

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

d)配置 RestTemplate

SpringCloud给提供了一个@LoadBalanced注解，和RestTemplate一起使用就可以从注册中心中以负载均衡的方式调用服务

package com.pachiraframework.springcloud.user.config;

import org.springframework.cloud.client.loadbalancer.LoadBalanced;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.web.client.RestTemplate;

@Configuration

public class WebConfig {

 @LoadBalanced

 @Bean

 public RestTemplate restTemplate() {

  return new RestTemplate();

 }

}

e)编写 UserController

注意这个类中高亮的部分，通过@Autowired标签注入上个步骤设置的@RestTemplate，另外调用的时候写的不是具体的服务ip和端口，而是我们在application.properties文件中设置的服务的名称！，通过@Loadbalanced和RestTemplate搭配，SpringCloud会在生成RestTemplate这个Bean的时候，加入一些拦截器，而在拦截器中会对服务ID进行替换，具体参考源代码中的LoadBalancerInterceptor类

package com.pachiraframework.springcloud.user.controller;

import java.util.Random;

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

import org.springframework.http.HttpEntity;

import org.springframework.http.HttpHeaders;

import org.springframework.http.MediaType;

import org.springframework.http.ResponseEntity;

import org.springframework.util.LinkedMultiValueMap;

import org.springframework.util.MultiValueMap;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

import org.springframework.web.client.RestTemplate;

import lombok.Data;

```java
import lombok.extern.slf4j.Slf4j;

@Slf4j
@RestController
@RequestMapping("/user/")
public class UserController {
 @Autowired
 private RestTemplate restTemplate;
 @RequestMapping(value= {"regist/sms"})
 public ResponseEntity<SmsSendResponse> sms(String mobile){
  Random random = new Random();
  int next = random.nextInt(10000000);
  String code = ""+(10000000-next);
  ResponseEntity<SmsSendResponse> response = doSend(mobile, code);
  return response;

 }

 public ResponseEntity<SmsSendResponse> doSend(String  mobile,String code) {
  final String sendUrl = "http://message-service/message/sms/send";
  HttpHeaders headers = new HttpHeaders();
  headers.setContentType(MediaType.APPLICATION_FORM_URLENCODED);
  MultiValueMap<String, String> map= new LinkedMultiValueMap<String, String>();
  map.add("mobile", mobile);
  map.add("templateId", "CHECK_CODE");
  map.add("params['code']", code);
  log.info("发送参数：{}",map);

  HttpEntity<MultiValueMap<String, String>> request = new HttpEntity<MultiValueMap<String, String>>(map, headers);
  ResponseEntity<SmsSendResponse> response = restTemplate.postForEntity(sendUrl, request, SmsSendResponse.class);
  return response;
 }

 @Data
 public static class SmsSendResponse {
  /**
   * 返回消息
   */
  private String message;
  /**
   * 返回状态码
   */
  private String code;
 }
}
```
f)启动UserApplication.main

g)访问

http://localhost:8082/user/regist/sms?mobile=18562875992