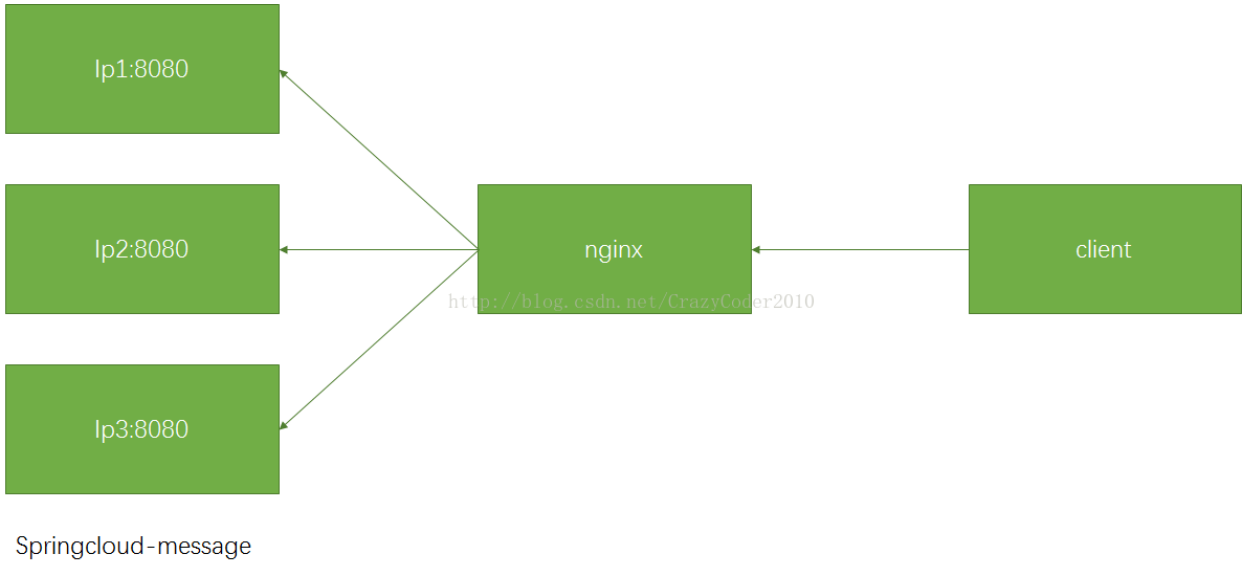
1.先前架构的问题及改进

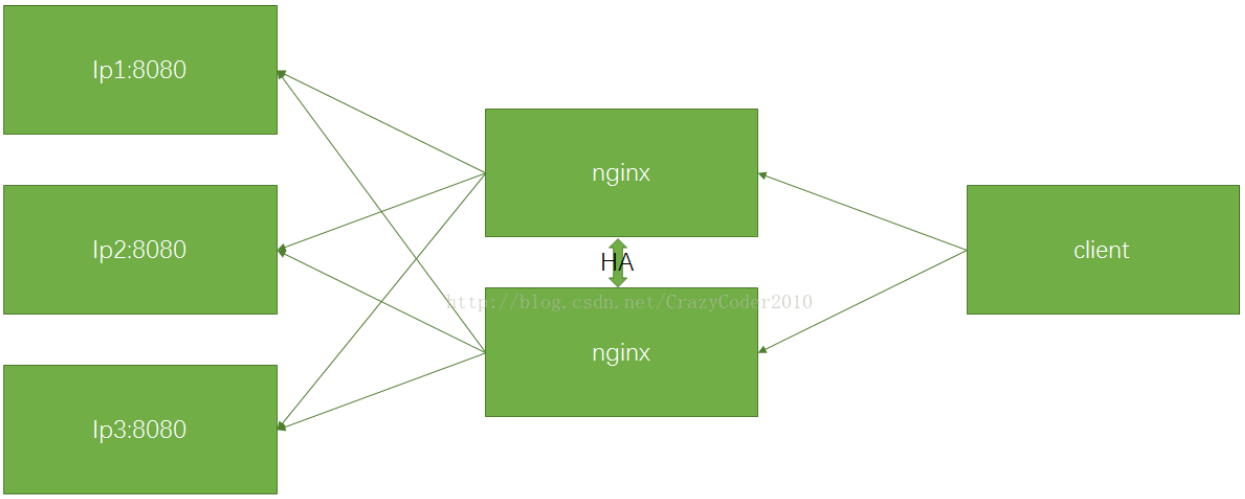
通过前两章的学习，我们已经掌握了通过SpringCloud/SpringBoot来提供一个Rest服务接口，并且可以通过RestTemplate来调用服务，整个世界看起来非常清洁:

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

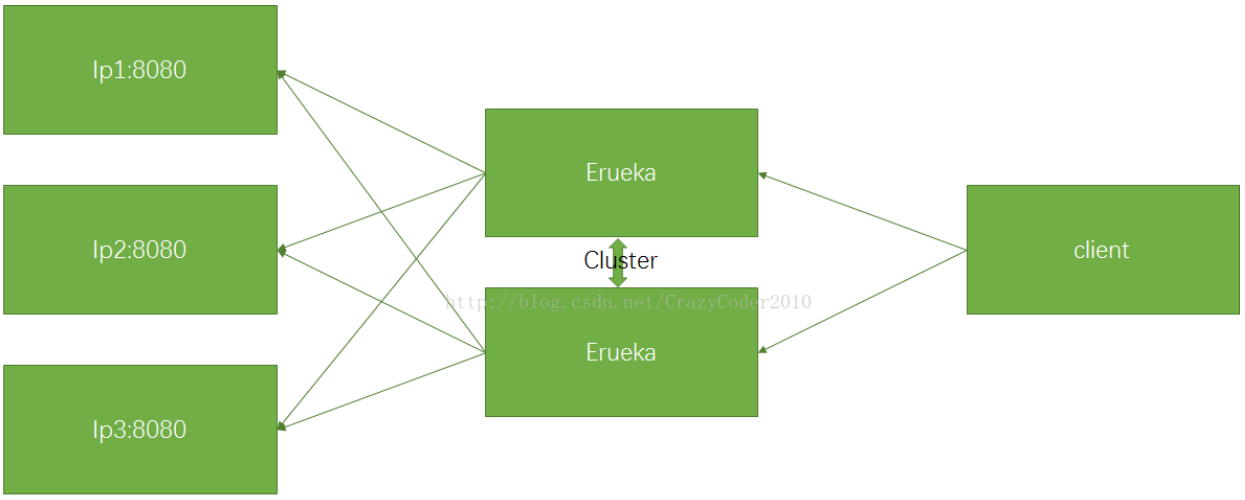


通过如上的改造，我们的服务节点单点故障解决了，访问压力也能通过nginx分发到各个节点上执行了，是否这样就完美了呢？如果你仔细看的话，会发现，其实nginx还是存在单点故障，也就是说，一旦nginx当即故障，client端的调用就全完了---即便这个时候所有的message服务都可用，于是，我们的架构不得不再次升级，来解决nginx的单点故障问题，于是架构可能升级到了如下版本：



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，整个集群就处于可用状态。

首先建立父工程，其子工程以module形式创建，主工程配置公共依赖，子工程配置各自所需要的依赖，以保证配置文件清晰明了。pom.xml如下：

```
<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>2-springcloud-study</name>

        <groupId>lizzy.springcloud</groupId>
        <artifactId>2-springcloud-study</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <packaging>pom</packaging>

        <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>
                <spring-cloud.version>Dalston.SR4</spring-cloud.version>
        </properties>

        <modules>
                <module>springcloud-message</module>
                <module>springcloud-eureka</module>
                <module>springcloud-user</module>
        </modules>

        <dependencies>
                <!--Lombok-->
                <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>

	<dependencyManagement>
		<dependencies>
			<dependency>
				<groupId>org.springframework.cloud</groupId>
				<artifactId>spring-cloud-
dependencies</artifactId>
				<version>${spring-cloud.version}</version>
				<type>pom</type>
				<scope>import</scope>
			</dependency>
		</dependencies>
	</dependencyManagement>
</project>
```

2.启动Eureka服务器

a)新建一个工程，并添加eureka相关的依赖

pom.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>2-springcloud-eureka</name>
	<description>Eureka注册与发现</description>

	<parent>
		<groupId>lizzy.springcloud</groupId>
		<artifactId>2-springcloud-study</artifactId>
		<version>0.0.1-SNAPSHOT</version>
	</parent>

	<artifactId>springcloud-eureka</artifactId>
	<packaging>jar</packaging>

	<dependencies>
```

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

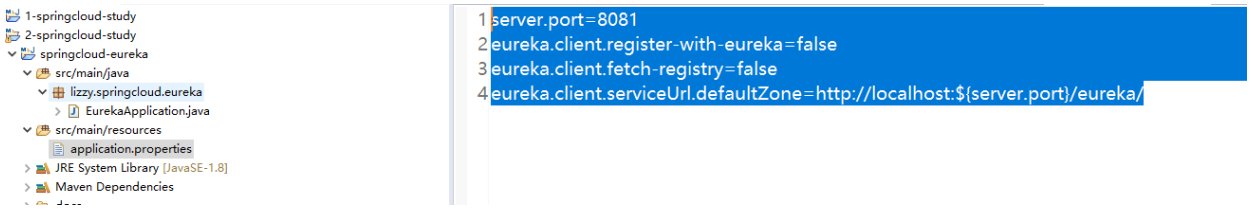
b)main函数中启用Erueka的注解

```
package lizzy.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);
        }
}
```

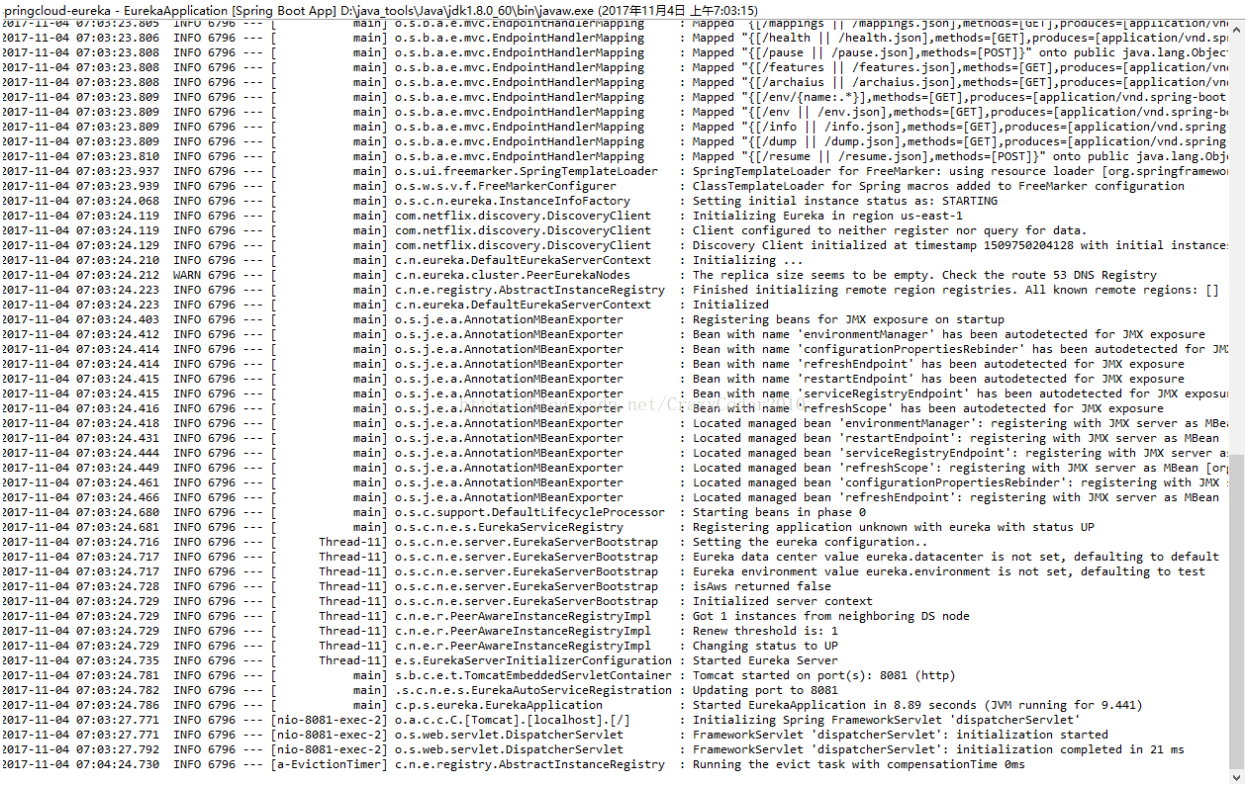
c)修改端口以及eureka自身的信息

application.properties

```
server.port=8081
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
eureka.client.serviceUrl.defaultZone=http://localhost:${server.port}/eureka/
```

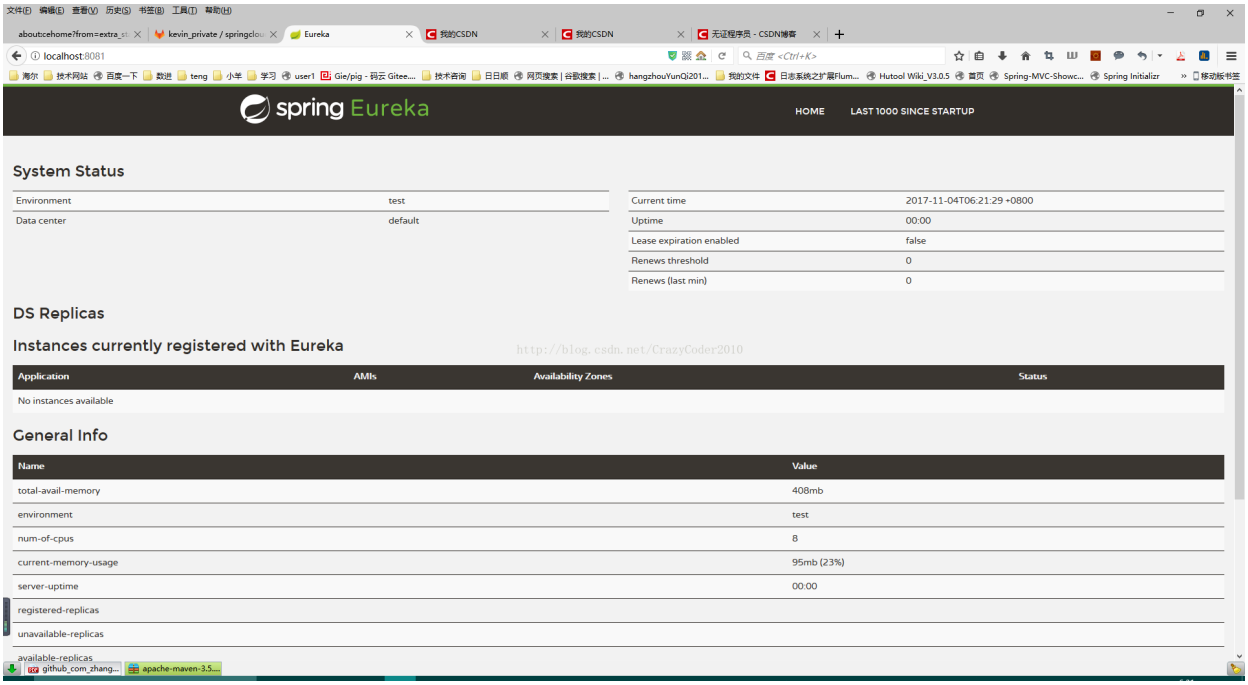


d)启动main函数



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

将上一节中的message服务作为module加入到工程中，进行稍微的改造，就可以将其注册到注册中心里了。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

pom.xml中增加eureka的客户端依赖

引入eureka客户端

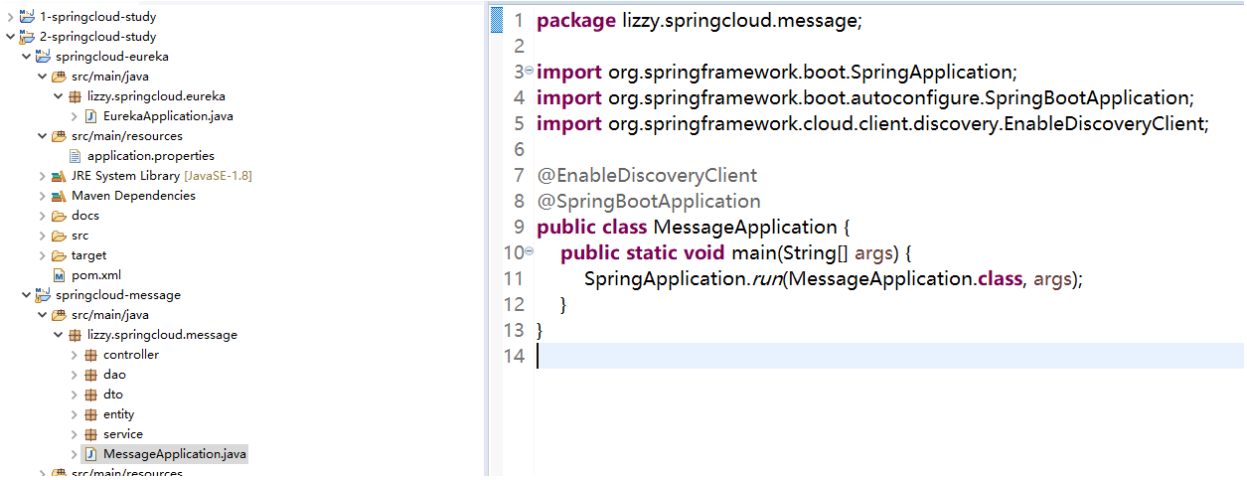
```
    <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-starter-eureka</artifactId>
        </dependency>
```

如图所示：

```
7   <parent>
8     <groupId>lizzy.springcloud</groupId>
9     <artifactId>2-springcloud-study</artifactId>
0     <version>0.0.1-SNAPSHOT</version>
1   </parent>
2
3   <name>springcloud-message</name>
4   <description>消息处理微服务</description>
5
6   <dependencies>
7     <dependency>
8       <groupId>org.springframework.cloud</groupId>
9       <artifactId>spring-cloud-starter-eureka</artifactId>
0     </dependency>
1     <dependency>
2       <groupId>org.springframework.boot</groupId>
3       <artifactId>spring-boot-starter-web</artifactId>
4     </dependency>
5     <dependency>
6       <groupId>org.springframework.boot</groupId>
7       <artifactId>spring-boot-starter-freemarker</artifactId>
8     </dependency>
9   </dependencies>
0 </project>
1
```

c)添加Eureka注解

在MessageApplication上添加@EnableDiscoveryClient注解

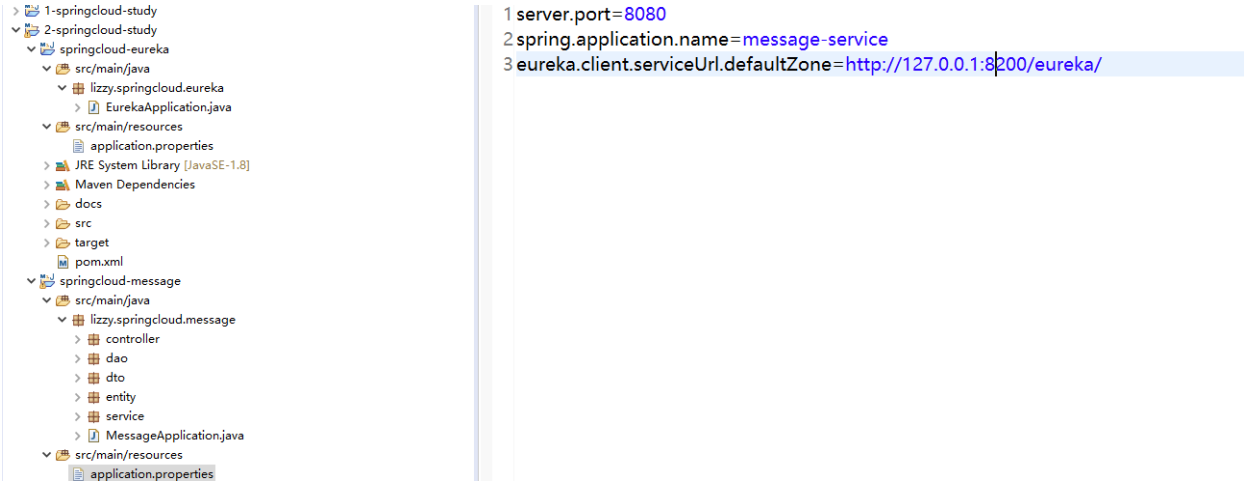


d)message服务中设置注册中心地址

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

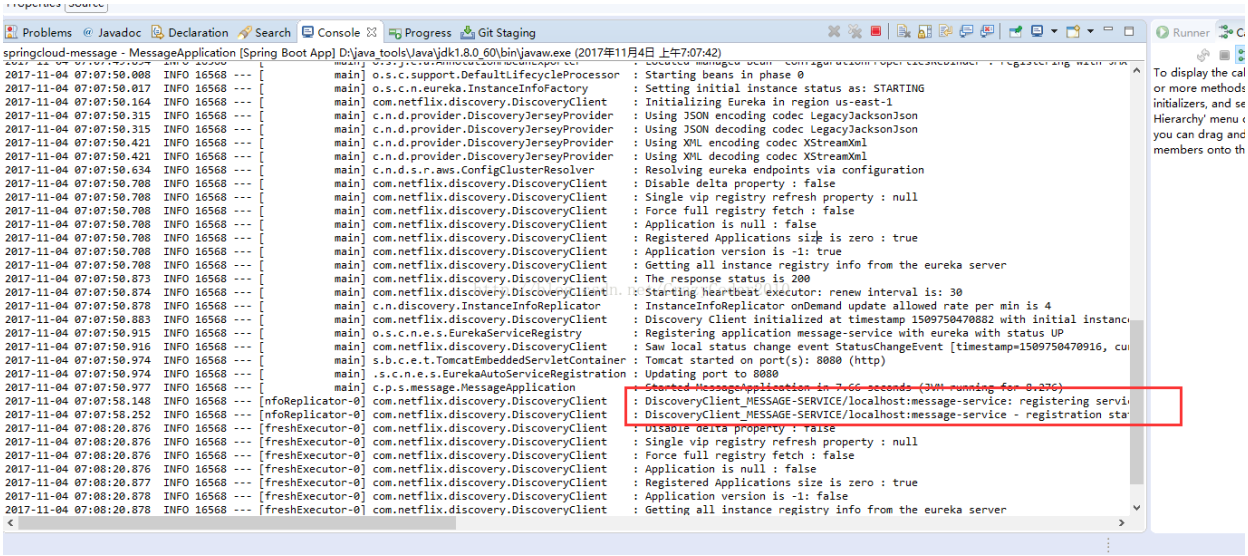
spring.application.name=message-service

eureka.client.serviceUrl.defaultZone=http://127.0.0.1:8200/eureka/



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

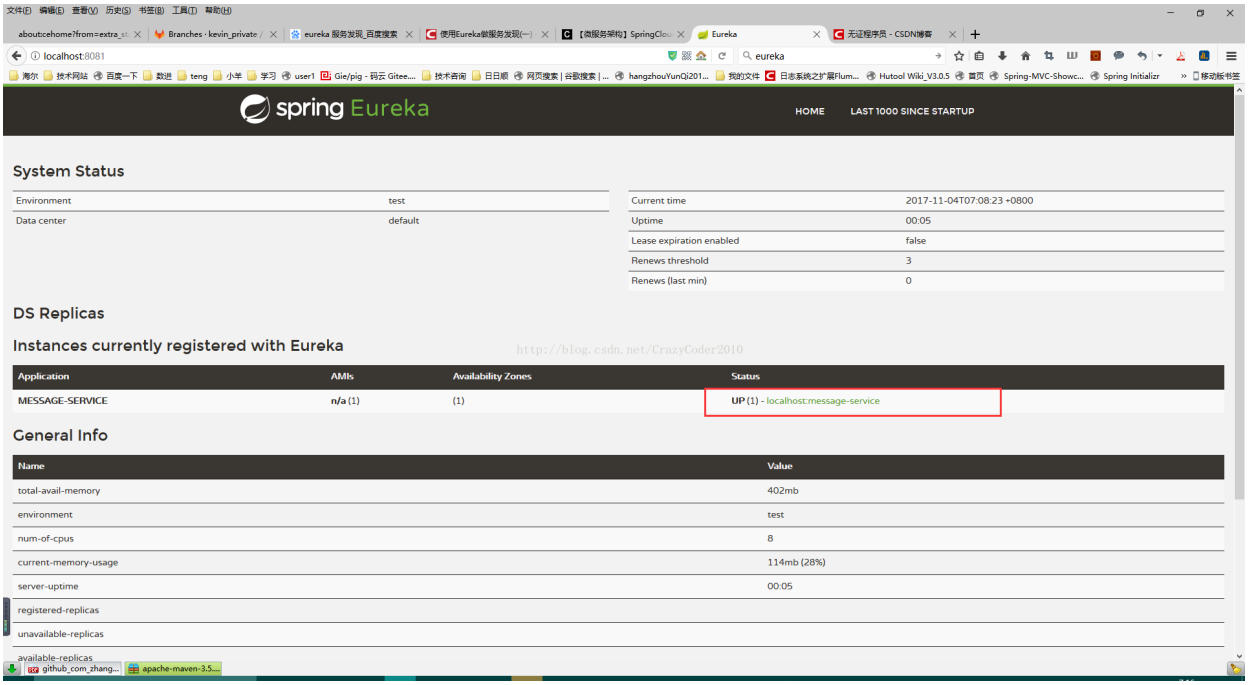
从日志可以看到，message服务已经注册到eureka中了



f)通过eureka控制台查看已经注册的服务

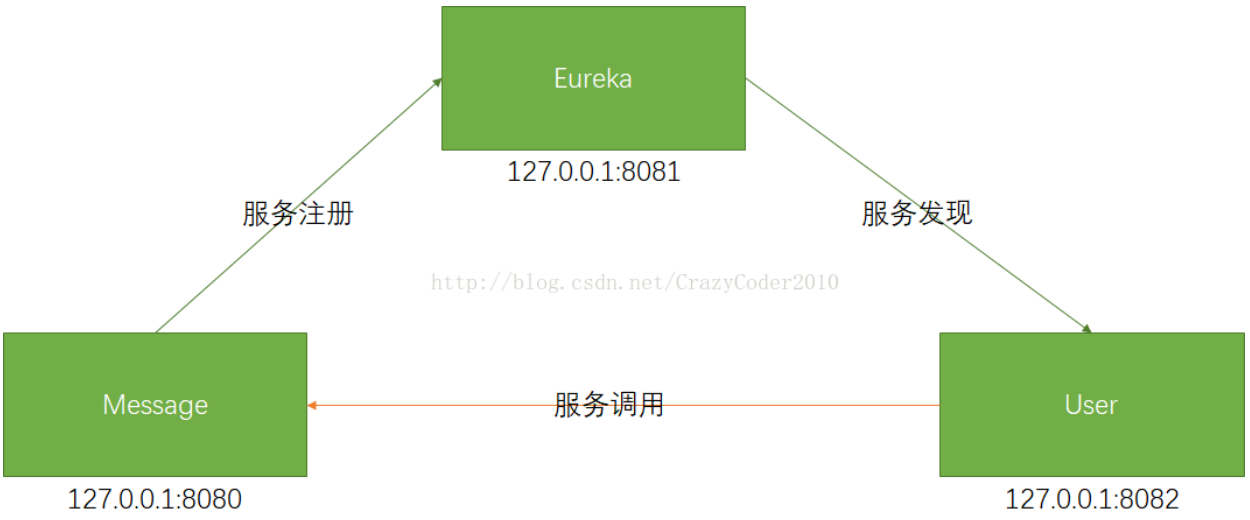
http://127.0.0.1:8081

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



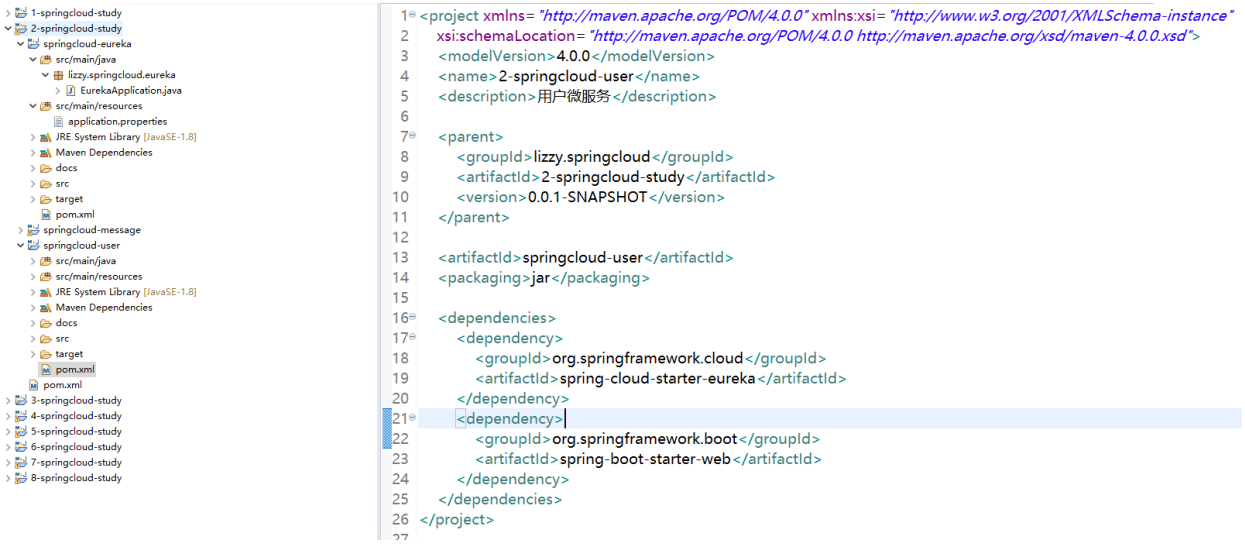
4.基于注册中心调用服务

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

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



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

```
server.port=8082
spring.application.name=user-service
eureka.client.serviceUrl.defaultZone=http://127.0.0.1:8081/eureka/
```

c)编写项目启动函数UserApplication

```
package lizzy.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

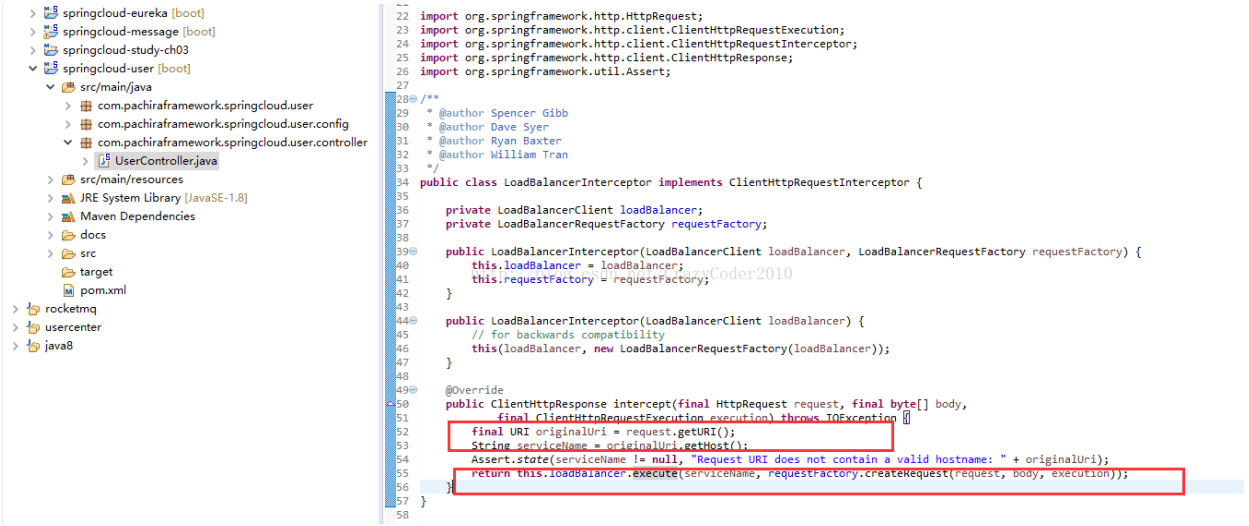
SpringCloud给提供了一个@LoadBalanced注解，和RestTemplate一起使用就可以从注册中心中以负载均衡的方式调用服务。WebConfig.java

```
package lizzy.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 {
	//从注册中心中以负载均衡的方式调用服务
	@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 lizzy.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;
```

```
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