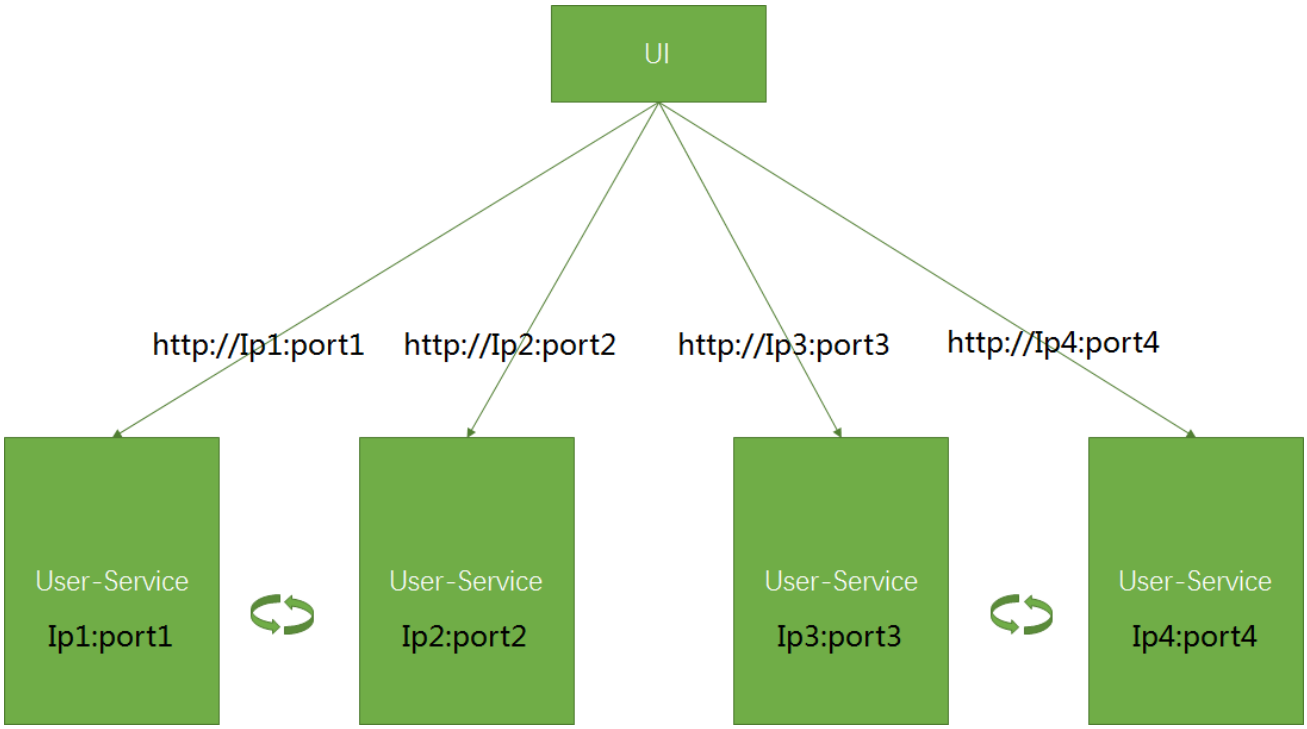
第六章 SpringCloud Zuul网关

本章完整源码地址：https://github.com/kwang2003/springcloud-study-ch06.git

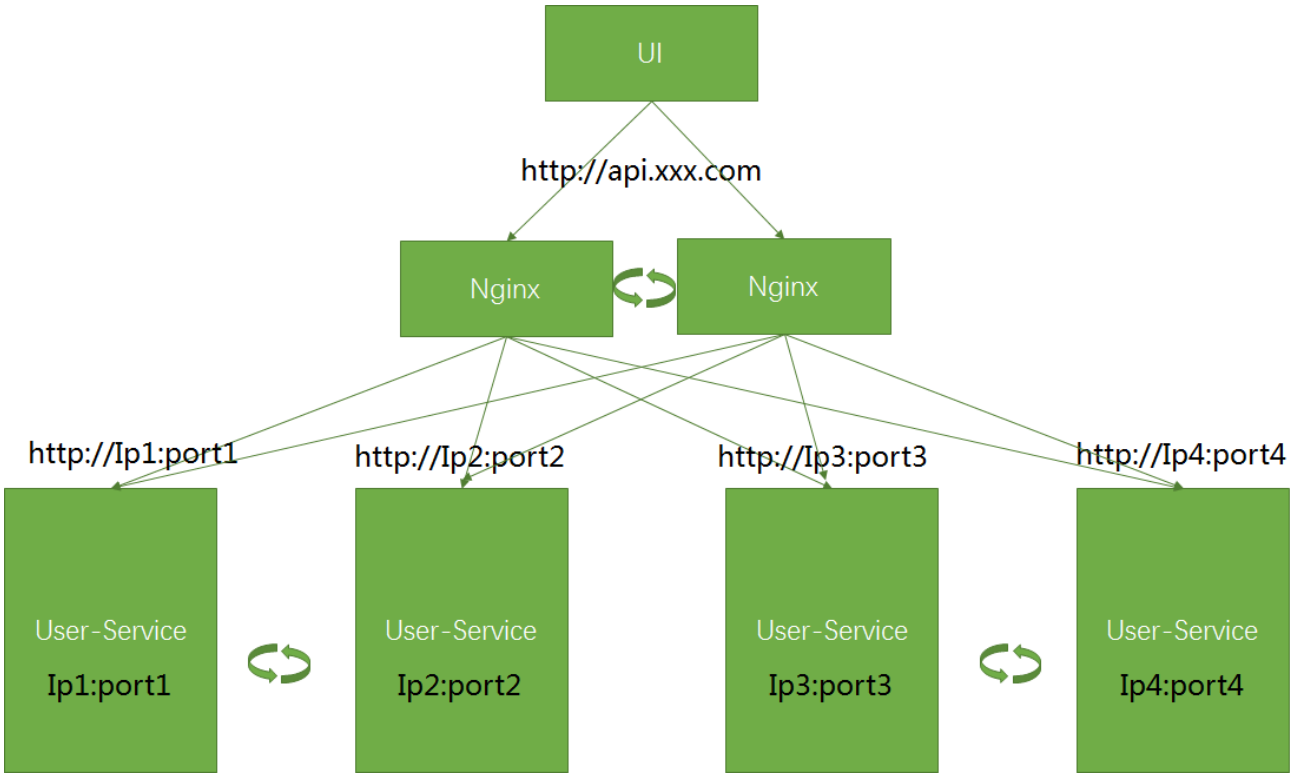
1.项目概要

这一章节的 内容以第五章的代码为基础改造而成https://github.com/kwang2003/springcloud-study-ch05.git。

随着微服务数量的增多，系统的复杂性也在增加，通过前几章的学习，我们已经掌握了微服务之间通过Eureka注册中心+Feign/RestTemplate相互调用，如果我们要在页面中调用一系列的微服务怎么办？而且特别是我们的各个微服务都部署了多个节点的情况下，情况可能会更糟糕，如下：

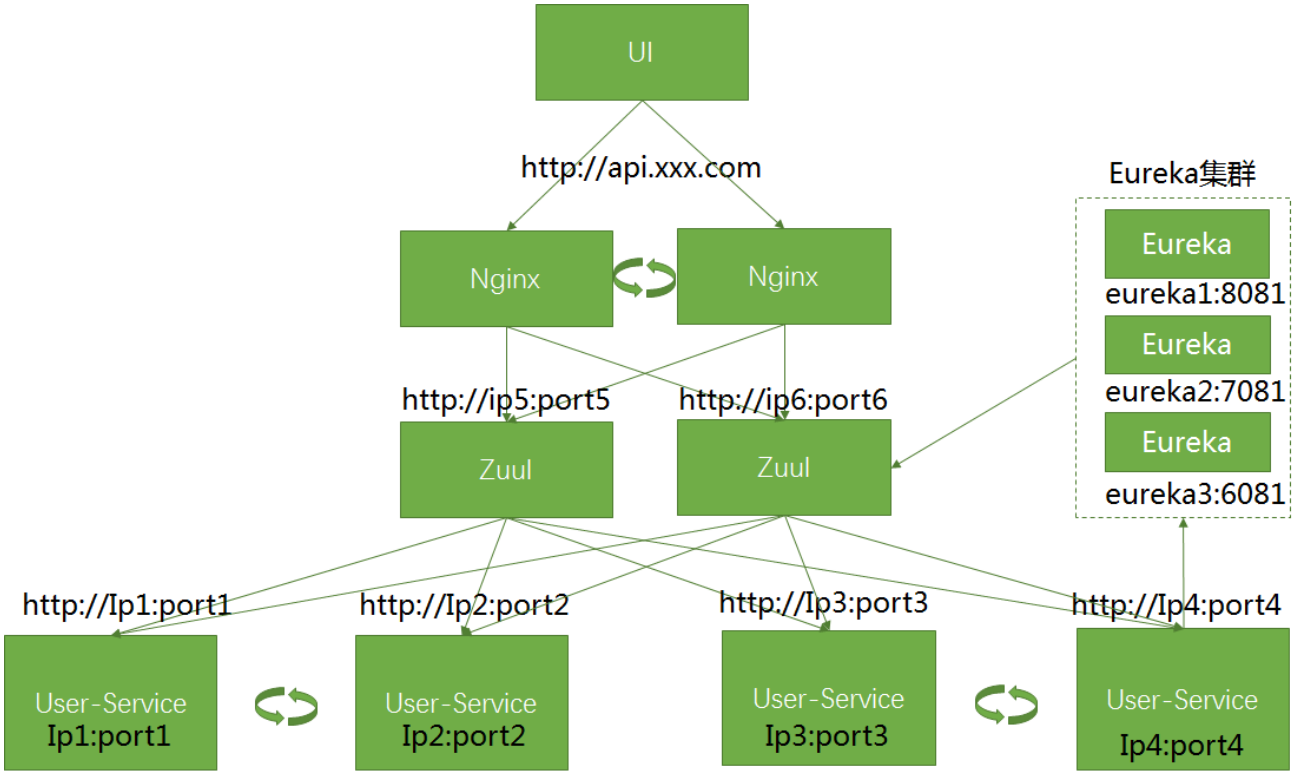


API的调用方需要了解各个微服务部署的ip和端口，并且自行实现客户端的负载均衡策略，当增减服务的时候，对UI的改造也非常繁琐，当然了，你可能也考虑到了，在UI与微服务之间增加一个nginx转发一下不就可以了吗？像这样



对于小型的系统架构，这个已经完全足够了，但是软件最经常变化的就是变化，当我们面临的是一个电商系统时，在一些节日如双11，系统的访问流量和压力会瞬间非常大，但是平时的量级却不是那么大，一个好的架构可以通过横向扩展服务节点的部署数量来分摊系统的流量--这个通过上面的架构也能做到，也就是说我们在节日的时候多部署一些节点，在nginx上把新增的节点配置上去，当节日过后，再去把

这些节点下线掉---你这么做考虑过运维工程师的感受么，当服务的节点成百上千个时，这些配置依靠手动去完成，出错的几率就会增大，还是那句话，能通过计算机自动完成的就自动完成，不要依赖人，于是，本章节的主角Zuul网关就闪亮登场：

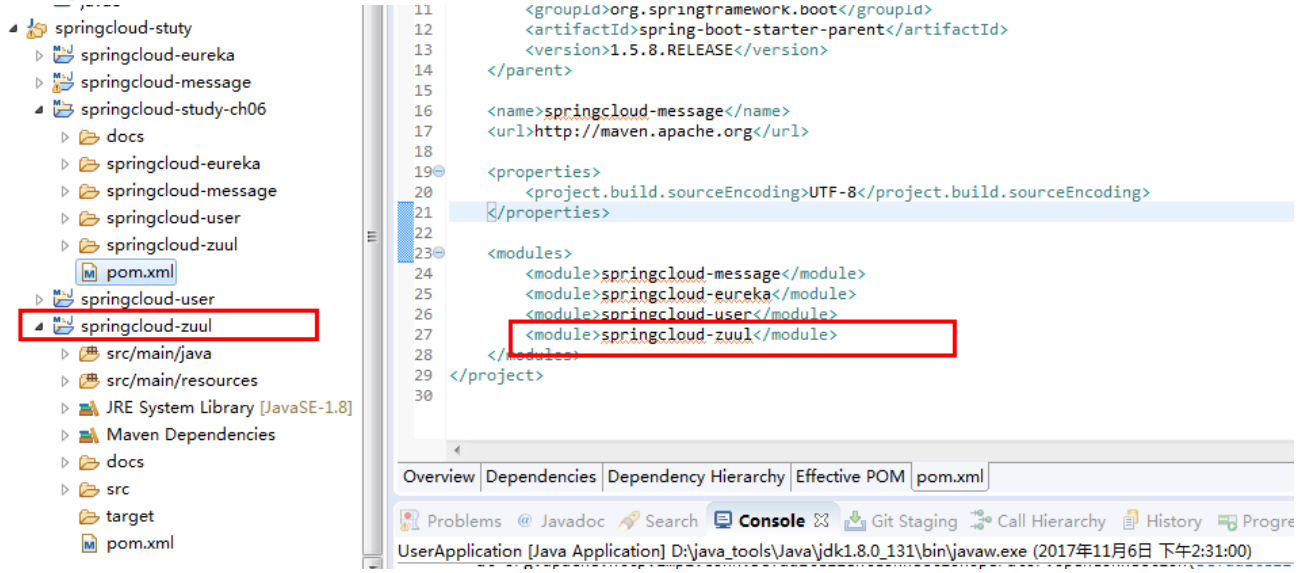


在整个微服务架构中，Zuul充当了服务网关的作用，它为系统中所有的微服务提供了一个统一的出口，结合一定的路由规则，将请求url转发到下层的微服务中。Zuul只负责服务的路由转发，本身并不执行具体的业务逻辑，因此它是轻量级的，Zuul节点的部署数量在整个架构中是相对稳定的，不会有大的变化，因此，nginx+zuul这一层次是相对固定不变的，变化的是下面的微服务，这里zuul和具体的微服务之间通过eureka注册中心间接依赖，当增加/下线微服务时，通过eureka注册中心相互感知，而不需要在Zuul上直接配置具体的微服务，因此可以说，zuul和微服务之间是松耦合的。

除了做微服务的路由以外，如果我们希望对微服务做安全控制或接口调用的日志记录等操作，可以在zuul上统一处理。

2.项目改造

a)新增springcloud-zuul模块

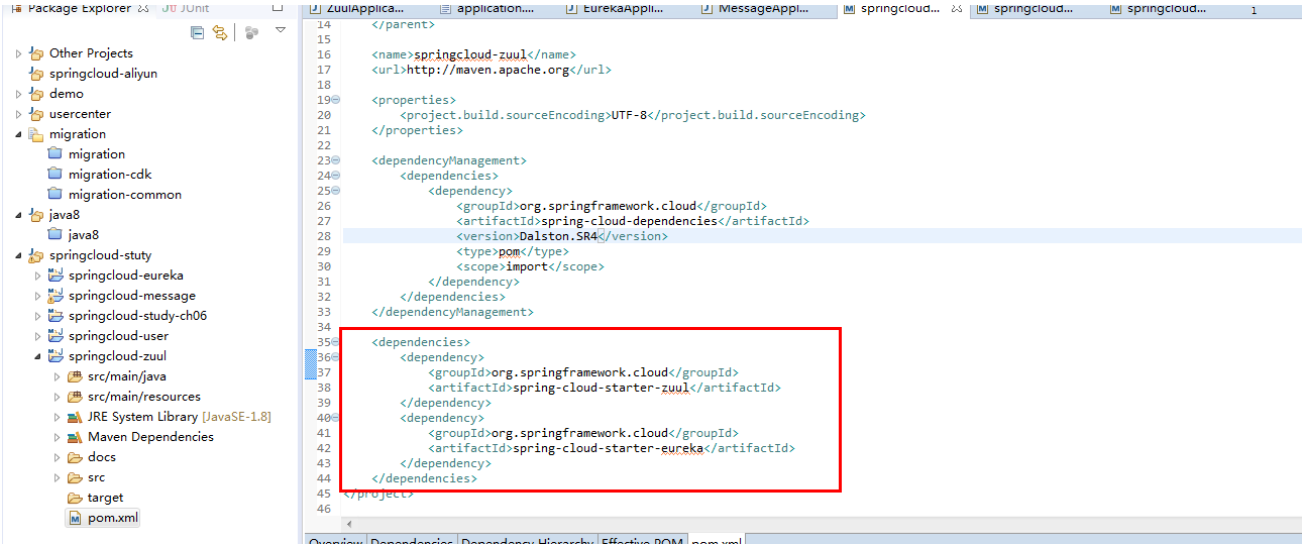


b)增加zuul的maven依赖

在springcloud-zuul模块中添加

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

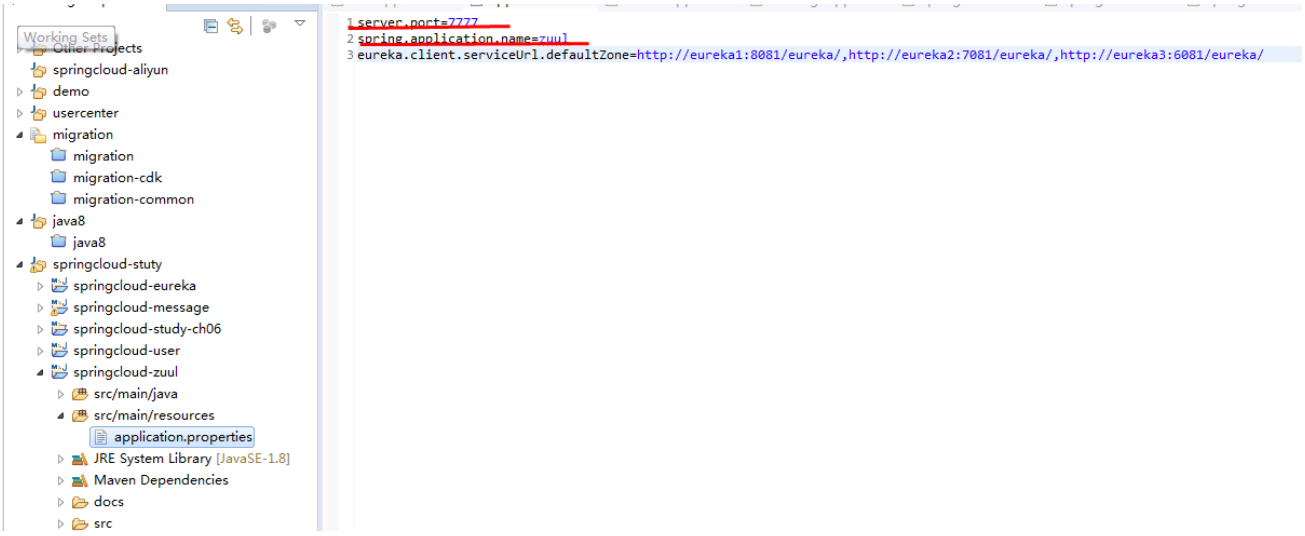
</dependency>



我们需要用到zuul组件，因此需要将spring-cloud-starter-zuul引入进来，因为需要zuul从注册中心eureka中获取微服务的列表，因此需要将spring-cloud-starter-eureka引入进来。

c)修改配置项

主要是修改启动端口为7777，并修改服务的名称以及指定注册中心eureka的地址



d)添加启动函数ZuulApplication

有两个重要的注解，@EnableZuulProxy是要启用zuul的代理服务器功能，@EnableDiscoveryClient表示要启动服务注册发现功能

```
package com.pachiraframework.springcloud.zuul;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.cloud.netflix.zuul.EnableZuulProxy;

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

e)启动ZuulApplication

启动后就能在注册中心发现zuul服务了

spring Eureka

HOME LAST 1000 SINCE STARTUP

System Status

Environment	test	Current time	2017-11-06T14:36:23 +0800
Data center	default	Uptime	00:06
		Lease expiration enabled	false
		Renews threshold	6
		Renews (last min)	6

EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.

DS Replicas

eureka3

eureka2

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
MESSAGE-SERVICE	n/a (1)	(1)	UP (1) - 10.130.52.80:message-service:8080
UNKNOWN	n/a (1)	(1)	UP (1) - 10.130.52.80:8081
USER-SERVICE	n/a (1)	(1)	UP (1) - 10.130.52.80:user-service:8082
ZUUL	n/a (1)	(1)	UP (1) - 10.130.52.80:zuul:7777

General Info

Name	Value
total-avail-memory	350mb
environment	test
num-of-cpus	4
current-memory-usage	83mb (23%)
server-uptime	00:06
registered-replicas	http://eureka3:6081/eureka/, http://eureka2:7081/eureka/
unavailable-replicas	http://eureka3:6081/eureka/,http://eureka2:7081/eureka/,
available-replicas	

Instance Info

Name	Value
ipAddr	10.130.52.80
status	UP

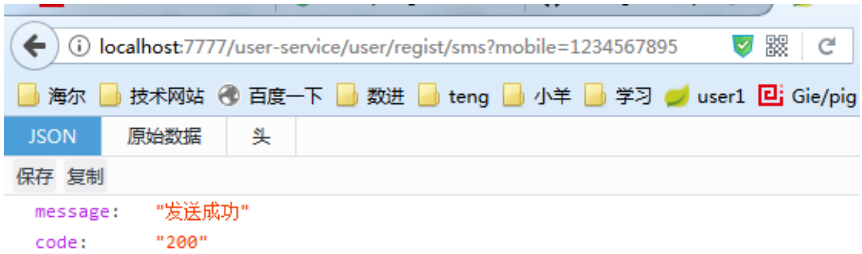
f)访问测试

在之前的案例中，我们是通过具体的微服务ip+端口进行访问的，如下

http://localhost:8082/user/regist/sms?mobile=1234567895

经过了zuul代理以后，我们就可以通过zuul的ip+端口来访问了

http://localhost:7777/user-service/user/regist/sms?mobile=1234567895



说明：

这时候大家注意到了我们zuul后面增加了一个user-service的上下文，这也是zuul默认的代理路由功能，即我们通过zuul来访问具体的微服务时，需要在zuul的地址后面加上具体的服务id作为上下文，然后后面跟的是具体微服务的访问url，这样请求经过zuul的时候，zuul会根据user-service这个id去注册中心中查找对应的微服务实例，然后通过一定的负载均衡算法选举一个可用节点，然后把后面的请求连接和参数转发过去执行。