

SpringCloud

Spring Cloud 是一套完整的微服务解决方案，基于 Spring Boot 框架，准确的说，它不是一个框架，而是一个大的容器，它将市面上较好的微服务框架集成进来，从而简化了开发者的代码量。

Spring Cloud 现状 目前，国内之前使用 Spring Cloud 技术的公司并不多见，不是因为 Spring Cloud 不好，主要原因有以下几点：

1、Spring Cloud 中文文档较少，出现问题网上没有太多的解决方案。 2、国内创业型公司技术老大大多是阿里系员工，而阿里系多采用 Dubbo 来构建微服务架构。 3、大型公司基本都有自己的分布式解决方案，而中小型公司的架构很多用不上微服务，所以没有采用 Spring Cloud 的必要性。但是，微服务架构是一个趋势，而 Spring Cloud 是微服务解决方案的佼佼者。

Spring Cloud 优缺点 其主要优点有：

1、集大成者，Spring Cloud 包含了微服务架构的方方面面。 2、约定优于配置，基于注解，没有配置文件。 3、轻量级组件，Spring Cloud 整合的组件大多比较轻量级，且都是各自领域的佼佼者。 4、开发简便，Spring Cloud 对各个组件进行了大量的封装，从而简化了开发。 5、开发灵活，Spring Cloud 的组件都是解耦的，开发人员可以灵活按需选择组件。 缺点：


1、项目结构复杂，每一个组件或者每一个服务都需要创建一个项目。 2、部署门槛高，学习成本高。

一、Eureka服务注册与发现

Spring Cloud Eureka 是 Spring Cloud Netflix 微服务套件中的一部分，它基于 Netflix 的 Eureka 做了二次封装，主要负责完成微服务架构中的服务治理功能。Spring Cloud 通过为 Eureka 增加了 Spring Boot 风格的自动化配置，我们只需通过简单引入依赖和注解配置就能让 Spring Boot 构建的微服务应用轻松地与 Eureka 服务治理体系进行整合。

服务治理可以说是微服务架构中最为核心和基础的模块，它主要用来实现各个微服务实例的自动化注册与发现。

搭建springbootcloud工程

 New Module ×

Project Metadata

Group:	<input type="text" value="com.soft863"/>
Artifact:	<input type="text" value="springcloud-consumer_zuul"/>
Type:	<input type="text" value="Maven Project (Generate a Maven based project archive.)"/>
Language:	<input type="text" value="Java"/>
Packaging:	<input type="text" value="Jar"/>
Java Version:	<input type="text" value="8"/>
Version:	<input type="text" value="0.0.1-SNAPSHOT"/>
Name:	<input type="text" value="springcloud-consumer_zuul"/>
Description:	<input type="text" value="Demo project for Spring Boot"/>
Package:	<input type="text" value="com.soft863.springcloudconsumer_zuul"/>

New Project

GroupId

springbootcloud

☒ Inherit

ArtifactId

springbootcloud

Version

1.0-SNAPSHOT

☒ Inherit

Previous

Next

Cancel

Help

New Project

Project name:

springbootcloud

Project location:

E:\prjcode\springbootcloud

...

▼ More Settings

Module name:

springbootcloud

Content root:

E:\prjcode\springbootcloud

Module file location:

E:\prjcode\springbootcloud

Project format:

.idea (directory based)

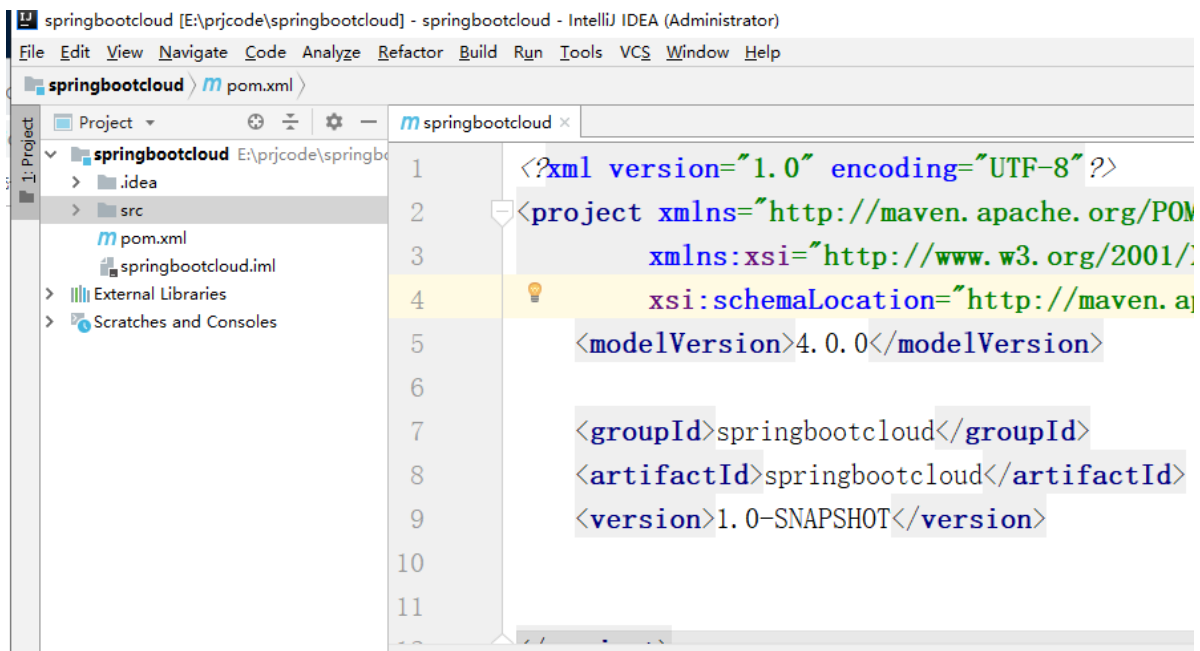
▼

Previous

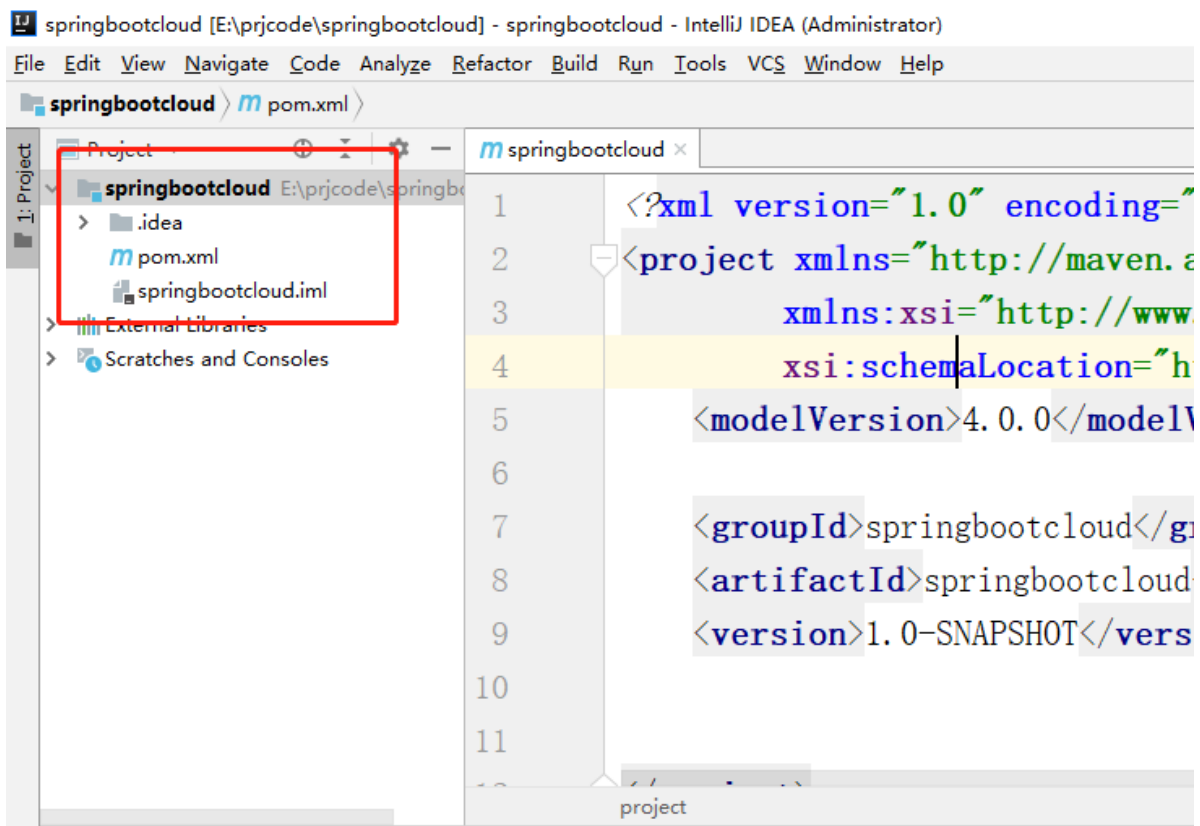
Finish

Cancel

Help

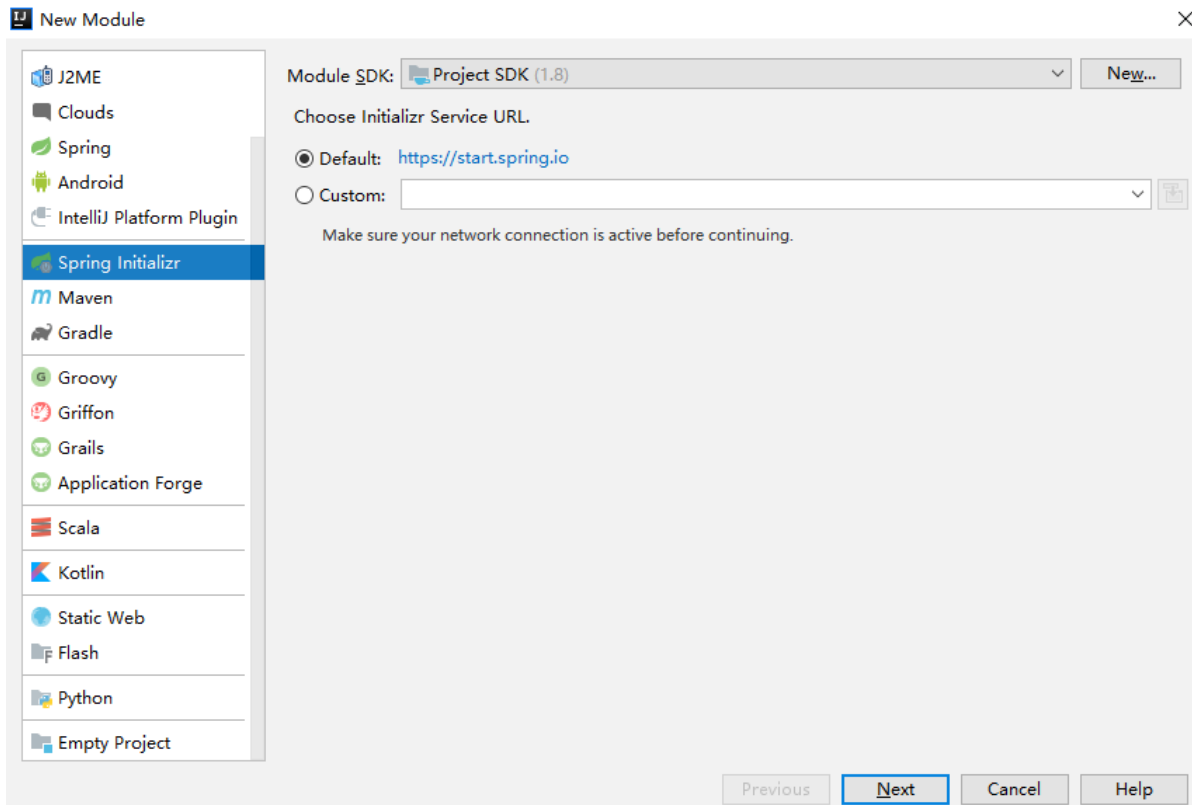
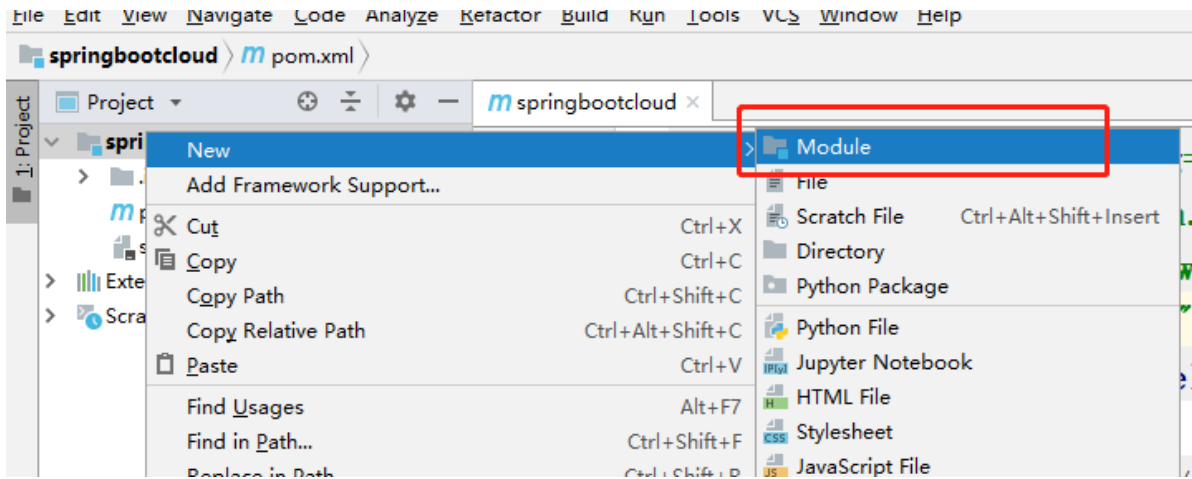


删除src文件夹



1.1、搭建SpringCloud项目（服务端）

1、创建项目springcloudserver（注册中心）



Project Metadata

Group:	com.soft863
Artifact:	springcloudserver
Type:	Maven Project (Generate a Maven based project archive.)
Language:	Java
Packaging:	Jar
Java Version:	8
Version:	0.0.1-SNAPSHOT
Name:	springcloudserver
Description:	Demo project for Spring Boot
Package:	com.soft863.springcloudserver

Previous

Next

Cancel

Help

Dependencies

Developer Tools

Web

Template Engines

Security

SQL

NoSQL

Messaging

I/O

Ops

Testing

Spring Cloud

Spring Cloud Security

Spring Cloud Tools

Spring Cloud Config

Spring Cloud Discovery

Spring Cloud Routing

Spring Cloud Circuit Breaker

Spring Cloud Tracing

Spring Cloud Messaging

Pivotal Cloud Foundry

Amazon Web Services

Spring Boot 2.2.2

☐ Eureka Discovery Client☒ Eureka Server☐ Apache Zookeeper Discovery☐ Cloud Foundry Discovery☐ Consul Discovery

Selected Dependencies

Developer Tools

Spring Boot DevToo

Spring Cloud Discovery

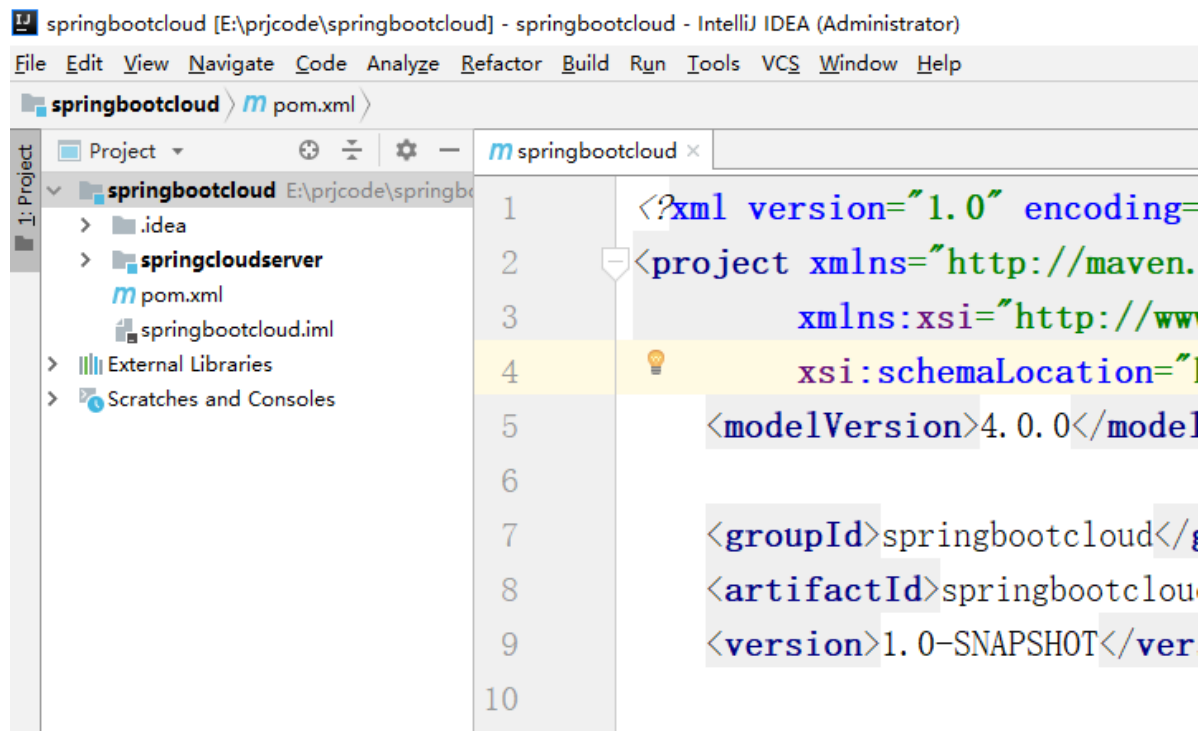
Eureka Server

Previous

Next

Cancel

Help



2、添加pom引用

生成文件后自动生成pom文件，无需修改，pom文件如下

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   https://maven.apache.org/xsd/maven-4.0.0.xsd">
4      <modelVersion>4.0.0</modelVersion>
5      <parent>
6          <groupId>org.springframework.boot</groupId>
7          <artifactId>spring-boot-starter-parent</artifactId>
8          <version>1.5.7.RELEASE</version>
9          <relativePath/> <!-- lookup parent from repository -->
10     </parent>
11     <groupId>com.soft863</groupId>
12     <artifactId>s<?xml version="1.0" encoding="UTF-8"?>
13 <project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
14     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   https://maven.apache.org/xsd/maven-4.0.0.xsd">
15     <modelVersion>4.0.0</modelVersion>
16     <parent>
17         <groupId>org.springframework.boot</groupId>
18         <artifactId>spring-boot-starter-parent</artifactId>
19         <version>2.2.2.RELEASE</version>
20         <relativePath/> <!-- lookup parent from repository -->
21     </parent>
22     <groupId>com.soft863</groupId>
23     <artifactId>springcloudserver</artifactId>
24     <version>0.0.1-SNAPSHOT</version>
25     <name>springcloudserver</name>

```

```
26     <description>Demo project for Spring Boot</description>
27
28     <properties>
29         <java.version>1.8</java.version>
30         <spring-cloud.version>Hoxton.RELEASE</spring-cloud.version>
31     </properties>
32
33     <dependencies>
34         <dependency>
35             <groupId>org.springframework.cloud</groupId>
36             <artifactId>spring-cloud-starter-netflix-eureka-
server</artifactId>
37         </dependency>
38
39         <dependency>
40             <groupId>org.springframework.boot</groupId>
41             <artifactId>spring-boot-devtools</artifactId>
42             <scope>runtime</scope>
43             <optional>true</optional>
44         </dependency>
45         <dependency>
46             <groupId>org.springframework.boot</groupId>
47             <artifactId>spring-boot-starter-test</artifactId>
48             <scope>test</scope>
49             <exclusions>
50                 <exclusion>
51                     <groupId>org.junit.vintage</groupId>
52                     <artifactId>junit-vintage-engine</artifactId>
53                 </exclusion>
54             </exclusions>
55         </dependency>
56     </dependencies>
57
58     <dependencyManagement>
59         <dependencies>
60             <dependency>
61                 <groupId>org.springframework.cloud</groupId>
62                 <artifactId>spring-cloud-dependencies</artifactId>
63                 <version>${spring-cloud.version}</version>
64                 <type>pom</type>
65                 <scope>import</scope>
66             </dependency>
67         </dependencies>
68     </dependencyManagement>
69
70     <build>
71         <plugins>
72             <plugin>
73                 <groupId>org.springframework.boot</groupId>
74                 <artifactId>spring-boot-maven-plugin</artifactId>
75             </plugin>
76         </plugins>
77     </build>
78
79 </project>
```

注意：版本号要对应，否则出错

3、添加properites配置

```

1  spring.application.name=eureka
2  server.port=8761
3
4  # 主机名字
5  eureka.instance.hostname=localhost
6  # 禁止自己作为服务注册
7  eureka.client.register-with-eureka=false
8  # 获取注册中心，false 表示不获取，自己就是注册中心
9  eureka.client.fetch-registry=false
10 # 关闭自我保护模式
11 eureka.server.enable-self-preservation=false
12 # 修改默认的请求地址，默认地址为：http://localhost:8761/eureka/
13 eureka.client.server-
    url.defaultZone=http://${eureka.instance.hostname}:${server.port}/eureka/

```

4、启动类添加注解

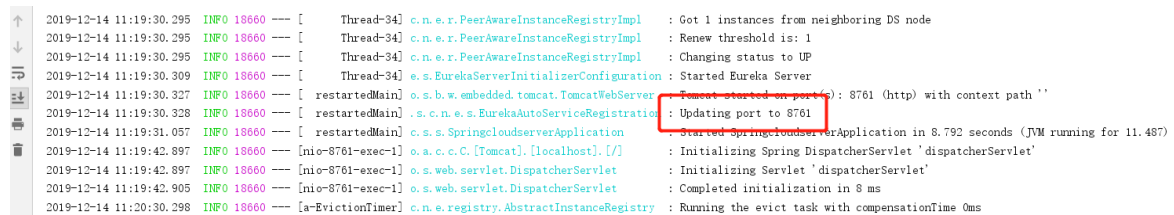
添加注解：@EnableEurekaServer

```

1  @SpringBootApplication
2  @EnableEurekaServer
3  public class SpringclouddemoApplication {
4      public static void main(String[] args) {
5          SpringApplication.run (SpringclouddemoApplication.class, args);
6      }
7  }

```

5、启动服务：



```

2019-12-14 11:19:30.295 INFO 18660 --- [ Thread-34] c.n.e.r.PeerAwareInstanceRegistryImpl : Got 1 instances from neighboring DS node
2019-12-14 11:19:30.295 INFO 18660 --- [ Thread-34] c.n.e.r.PeerAwareInstanceRegistryImpl : Renew threshold is: 1
2019-12-14 11:19:30.295 INFO 18660 --- [ Thread-34] c.n.e.r.PeerAwareInstanceRegistryImpl : Changing status to UP
2019-12-14 11:19:30.309 INFO 18660 --- [ Thread-34] e.s.EurekaServerInitializerConfiguration : Started Eureka Server
2019-12-14 11:19:30.327 INFO 18660 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8761 (http) with context path ''
2019-12-14 11:19:30.328 INFO 18660 --- [ restartedMain] .s.c.n.e.s.EurekaAutoServiceRegistration : Updating port to 8761
2019-12-14 11:19:31.057 INFO 18660 --- [ restartedMain] c.s.s.SpringcloudserverApplication : Started SpringcloudserverApplication in 8.792 seconds (JVM running for 11.487)
2019-12-14 11:19:42.897 INFO 18660 --- [nio-8761-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2019-12-14 11:19:42.897 INFO 18660 --- [nio-8761-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2019-12-14 11:19:42.905 INFO 18660 --- [nio-8761-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 8 ms
2019-12-14 11:20:30.298 INFO 18660 --- [a-EvictionTimer] c.n.e.registry.AbstractInstanceRegistry : Running the evict task with compensationTime 0ms

```

浏览器输入网址：

<http://localhost:8761/>

springEureka

HOME LAST 1000 SINCE STARTUP

System Status

Environment	test	Current time	2019-12-14T11:19:42 +0800
Data center	default	Uptime	00:00
		Lease expiration enabled	false
		Renews threshold	1
		Renews (last min)	0

DS Replicas

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
No instances available			

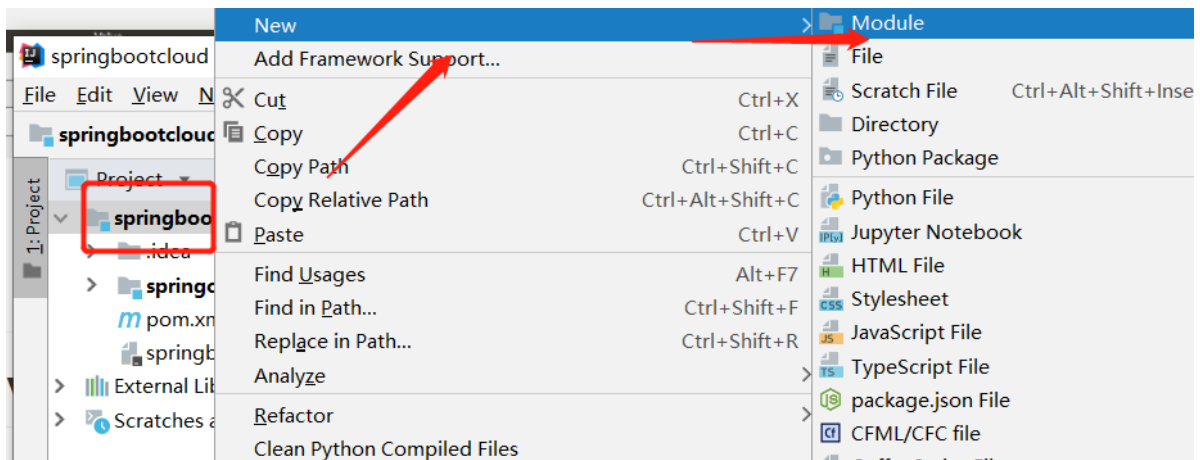
General Info

Name	Value
total-avail-memory	381mb
environment	test
num-of-cpus	8
current-memory-usage	61mb (16%)

1.2、创建服务提供者

1、创建项目springcloud-provider

工程右键



New Module

Module SDK: Project SDK (1.8) New...

Choose Initializr Service URL.

☒ Default: <https://start.spring.io>

☐ Custom:

Make sure your network connection is active before continuing.

Previous **Next** Cancel Help

Left Sidebar:

- J2ME
- Clouds
- Spring
- Android
- IntelliJ Platform Plugin
- Spring Initializr**
- Maven
- Gradle
- Groovy
- Griffon
- Grails
- Application Forge
- Scala
- Kotlin
- Static Web
- Flash
- Python
- Empty Project

New Module

Project Metadata

Group: com.soft863

Artifact: springcloud-provider

Type: Maven Project (Generate a Maven based project archive.)

Language: Java

Packaging: Jar

Java Version: 8

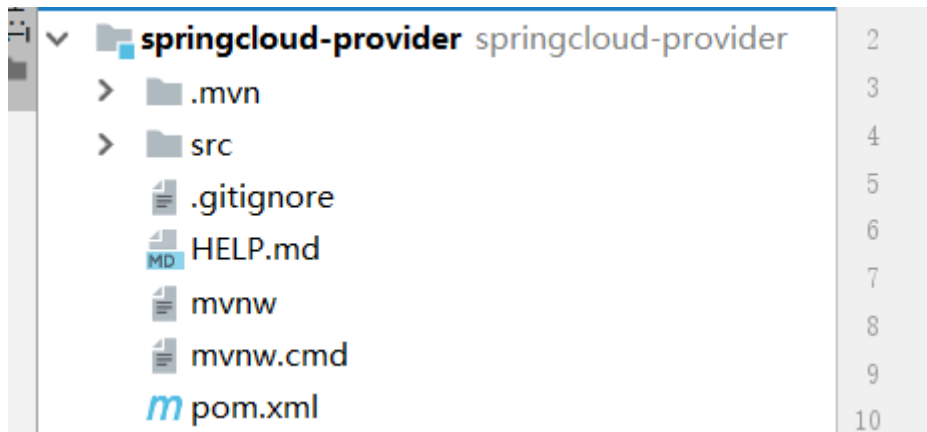
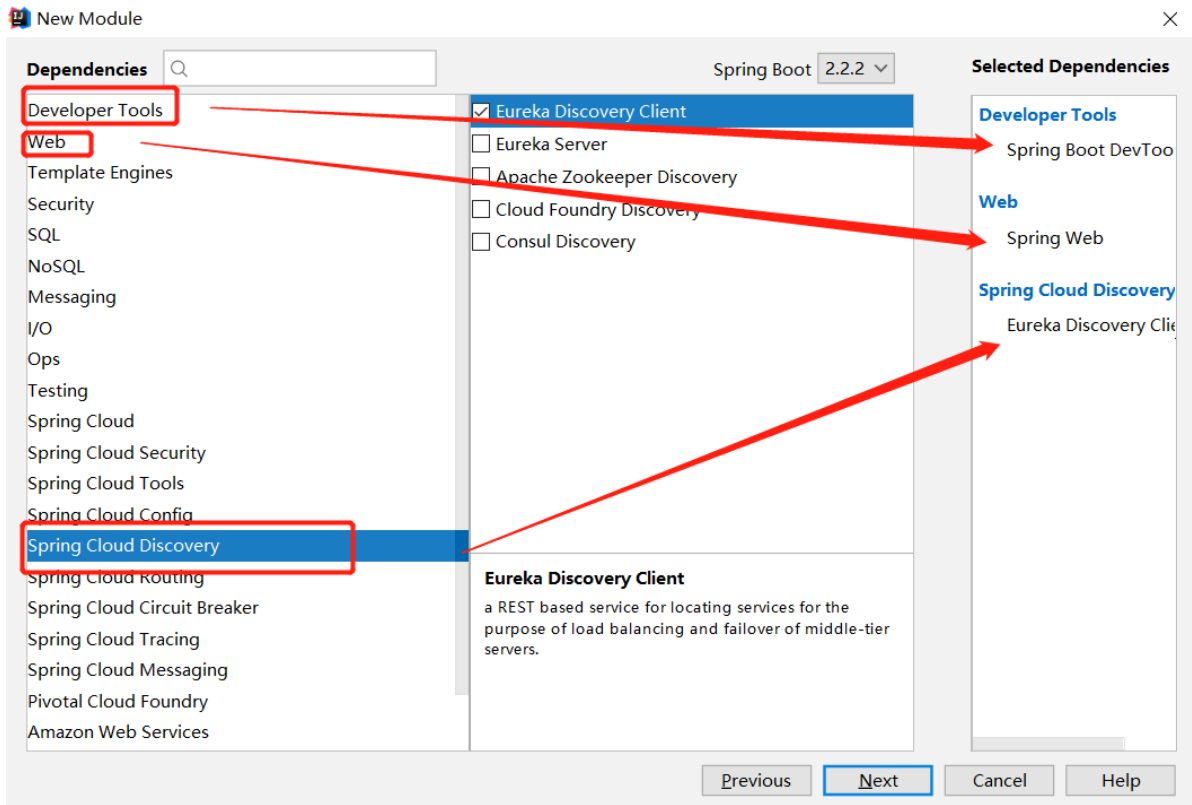
Version: 0.0.1-SNAPSHOT

Name: springcloud-provider

Description: Demo project for Spring Boot

Package: com.soft863.springcloudprovider

Previous **Next** Cancel Help



2、添加引用

自动生成pom无需修改

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
4     <modelVersion>4.0.0</modelVersion>
5     <parent>
6         <groupId>org.springframework.boot</groupId>
7         <artifactId>spring-boot-starter-parent</artifactId>
8         <version>2.2.2.RELEASE</version>
9         <relativePath/> <!-- lookup parent from repository -->
10    </parent>
11    <groupId>com.soft863</groupId>
12    <artifactId>springcloud-provider</artifactId>
13    <version>0.0.1-SNAPSHOT</version>
```

```
14 <name>springcloud-provider</name>
15 <description>Demo project for Spring Boot</description>
16
17 <properties>
18     <java.version>1.8</java.version>
19     <spring-cloud.version>Hoxton.RELEASE</spring-cloud.version>
20 </properties>
21
22 <dependencies>
23     <dependency>
24         <groupId>org.springframework.boot</groupId>
25         <artifactId>spring-boot-starter-web</artifactId>
26     </dependency>
27     <dependency>
28         <groupId>org.springframework.cloud</groupId>
29         <artifactId>spring-cloud-starter-netflix-eureka-
client</artifactId>
30     </dependency>
31
32     <dependency>
33         <groupId>org.springframework.boot</groupId>
34         <artifactId>spring-boot-devtools</artifactId>
35         <scope>runtime</scope>
36         <optional>true</optional>
37     </dependency>
38     <dependency>
39         <groupId>org.springframework.boot</groupId>
40         <artifactId>spring-boot-starter-test</artifactId>
41         <scope>test</scope>
42         <exclusions>
43             <exclusion>
44                 <groupId>org.junit.vintage</groupId>
45                 <artifactId>junit-vintage-engine</artifactId>
46             </exclusion>
47         </exclusions>
48     </dependency>
49 </dependencies>
50
51 <dependencyManagement>
52     <dependencies>
53         <dependency>
54             <groupId>org.springframework.cloud</groupId>
55             <artifactId>spring-cloud-dependencies</artifactId>
56             <version>${spring-cloud.version}</version>
57             <type>pom</type>
58             <scope>import</scope>
59         </dependency>
60     </dependencies>
61 </dependencyManagement>
62
63 <build>
64     <plugins>
65         <plugin>
66             <groupId>org.springframework.boot</groupId>
67             <artifactId>spring-boot-maven-plugin</artifactId>
```

```

68         </plugin>
69     </plugins>
70 </build>
71
72 </project>

```

3、添加properties配置

```

1  server.port=2003
2  eureka.instance.hostname=localhost
3  # 应用名称
4  spring.application.name=user-service
5  # EurekaServer地址
6  eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka/
7  # 当调用getHostname获取实例的hostname时，返回ip而不是host名称
8  eureka.instance.prefer-ip-address=true
9  # 指定自己的ip信息，不指定的话会自己寻找
10 eureka.instance.ip-address=127.0.0.1

```

4、添加Controller类

com.soft863下面添加controller包，创建UserController类，内容如下

```

1  @RestController
2  public class UserController {
3
4      @Value("${server.port}")
5      String port;
6
7      @GetMapping("/test")
8      public String log() {
9          return "hello user";
10     }
11
12     @RequestMapping(value = "/getUser", method = RequestMethod.GET)
13     public String product(@RequestParam String name) {
14
15         return name + "恭喜您，登录成功， " + "我来自于端口:" + port;
16     }
17 }

```

5、启动类添加注解

```

1  @ComponentScan("com.soft863")
2  @EnableDiscoveryClient

```

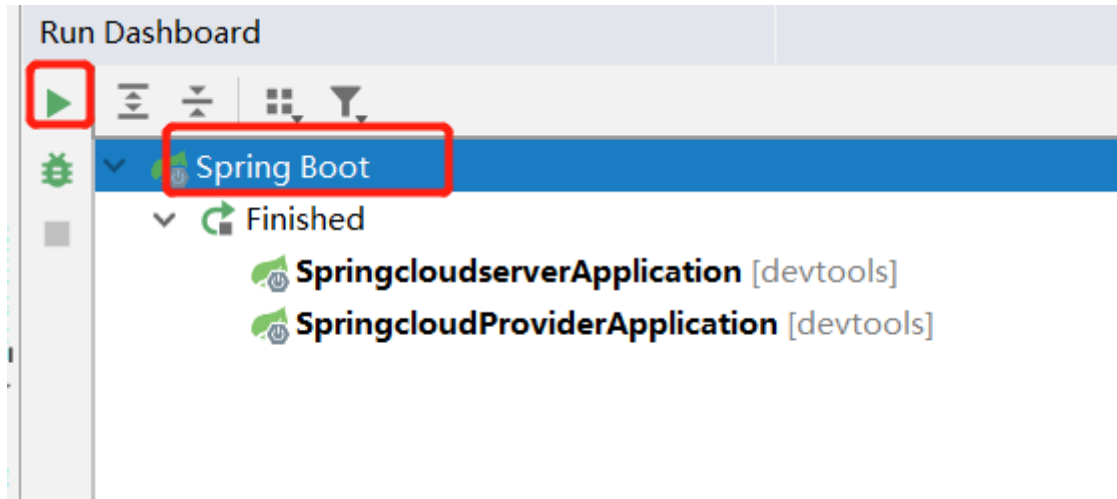
```
@SpringBootApplication
@ComponentScan("com.soft863")
@EnableDiscoveryClient
public class SpringcloudProviderApplication {

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

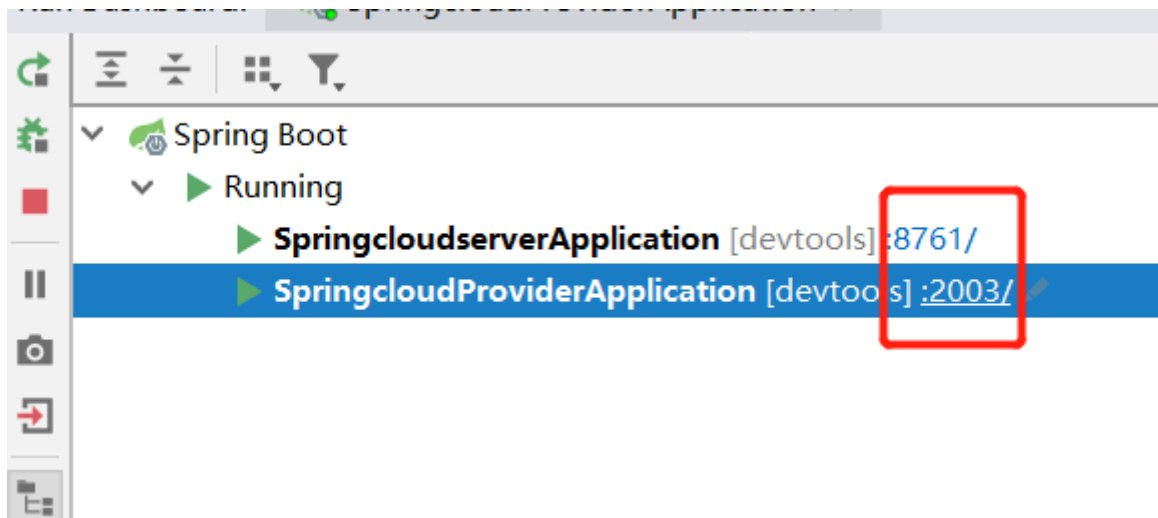
6、启动

启动顺序:server—provider

一键启动:



启动后效果:



查看网址: <http://localhost:8761/>

spring Eureka

HOME LAST 1000 SINCE STARTUP

System Status

Environment	test	Current time	2019-12-14T11:46:35 +0800
Data center	default	Uptime	00:00
		Lease expiration enabled	false
		Renews threshold	3
		Renews (last min)	0

DS Replicas

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
USER-SERVICE	n/a (1)	(1)	UP (1) - wcj-ecuser-service.2003

General Info

Name	Value
total-avail-memory	463mb
environment	test
num-of-cpus	8

浏览器输入: <http://localhost:2003/getUser?name=wangchaojie>

wangchaojie恭喜您, 登录成功, 我来自于端口:2003

1.3、创建服务消费端

1、创建项目springcloud-consumer

Project Structure

springclouddemo

Name: springclouddemo

Sources Paths Dependencies

New Module

Project Metadata

Group: com.soft863

Artifact: springcloud-consumer

Type: Maven Project (Generate a Maven based project archive.)

Language: Java

Packaging: Jar

Java Version: 8

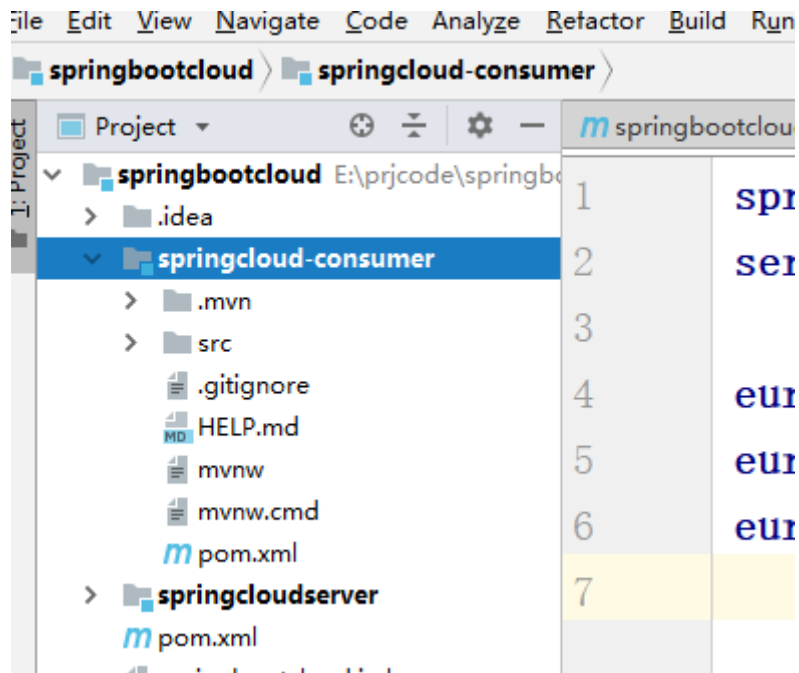
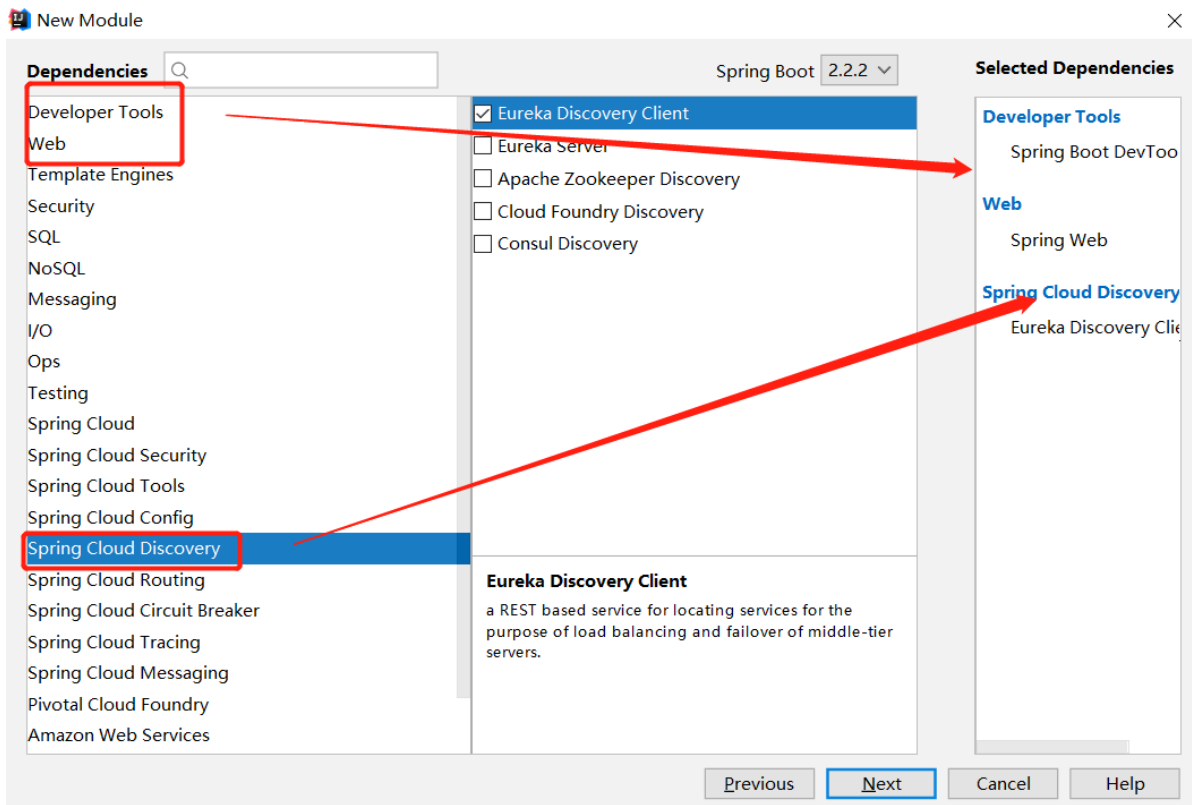
Version: 0.0.1-SNAPSHOT

Name: springcloud-consumer

Description: Demo project for Spring Boot

Package: com.soft863.springcloudconsumer

Previous Next Cancel Help



2、添加引用

pom自动生成，无需修改

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  https://maven.apache.org/xsd/maven-4.0.0.xsd">
4     <modelVersion>4.0.0</modelVersion>
5     <parent>
6         <groupId>org.springframework.boot</groupId>
7         <artifactId>spring-boot-starter-parent</artifactId>

```



```
8         <version>2.2.2.RELEASE</version>
9         <relativePath/> <!-- lookup parent from repository -->
10    </parent>
11    <groupId>com.soft863</groupId>
12    <artifactId>springcloud-consumer</artifactId>
13    <version>0.0.1-SNAPSHOT</version>
14    <name>springcloud-consumer</name>
15    <description>Demo project for Spring Boot</description>
16
17    <properties>
18        <java.version>1.8</java.version>
19        <spring-cloud.version>Hoxton.RELEASE</spring-cloud.version>
20    </properties>
21
22    <dependencies>
23        <dependency>
24            <groupId>org.springframework.boot</groupId>
25            <artifactId>spring-boot-starter-web</artifactId>
26        </dependency>
27        <dependency>
28            <groupId>org.springframework.cloud</groupId>
29            <artifactId>spring-cloud-starter-netflix-eureka-
client</artifactId>
30        </dependency>
31
32        <dependency>
33            <groupId>org.springframework.boot</groupId>
34            <artifactId>spring-boot-devtools</artifactId>
35            <scope>runtime</scope>
36            <optional>true</optional>
37        </dependency>
38        <dependency>
39            <groupId>org.springframework.boot</groupId>
40            <artifactId>spring-boot-starter-test</artifactId>
41            <scope>test</scope>
42            <exclusions>
43                <exclusion>
44                    <groupId>org.junit.vintage</groupId>
45                    <artifactId>junit-vintage-engine</artifactId>
46                </exclusion>
47            </exclusions>
48        </dependency>
49    </dependencies>
50
51    <dependencyManagement>
52        <dependencies>
53            <dependency>
54                <groupId>org.springframework.cloud</groupId>
55                <artifactId>spring-cloud-dependencies</artifactId>
56                <version>${spring-cloud.version}</version>
57                <type>pom</type>
58                <scope>import</scope>
59            </dependency>
60        </dependencies>
61    </dependencyManagement>
```

```

62
63     <build>
64         <plugins>
65             <plugin>
66                 <groupId>org.springframework.boot</groupId>
67                 <artifactId>spring-boot-maven-plugin</artifactId>
68             </plugin>
69         </plugins>
70     </build>
71
72 </project>
73

```

3、添加配置

```

1  server.port=2101
2  #应用名称
3  spring.application.name=user-consumer
4  # EurekaServer地址
5  eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka/
6  # 当调用getHostname获取实例的hostname时，返回ip而不是host名称
7  eureka.instance.prefer-ip-address=true
8  # 指定自己的ip信息，不指定的话会自己寻找
9  eureka.instance.ip-address=127.0.0.1
10
11 user-service.url:http://127.0.0.1:2003/

```

4、启动Controller类

com.soft863下面添加controller包，创建LoginController类，内容如下

```

1  package com.soft863.controller;
2
3  import org.springframework.beans.factory.annotation.Autowired;
4  import org.springframework.beans.factory.annotation.Value;
5  import org.springframework.web.bind.annotation.GetMapping;
6  import org.springframework.web.bind.annotation.RestController;
7  import org.springframework.web.client.RestOperations;
8
9  @RestController
10 public class LoginController {
11     @Autowired
12     private RestOperations restTemplate;
13     @Value("${user-service.url}")
14     private String userServerURL;
15
16     @GetMapping("/consumer")
17     public String consumer() {
18         return "这是一个测试...";
19     }
20
21     @GetMapping("/login")
22     public String login() {

```

```

23         return restTemplate.getForObject (userServerURL+"getUser?
    name=soft863", String.class);
24     }
25 }

```

5、启动类添加注解

```

1  @ComponentScan("com.soft863")
2  @EnableDiscoveryClient

```

添加RestTemplate自动装配

```

1  @Autowired
2  private RestTemplateBuilder builder;
3
4  @Bean
5  public RestTemplate restTemplate(){
6      return builder.build();
7  }

```

完整代码如下：

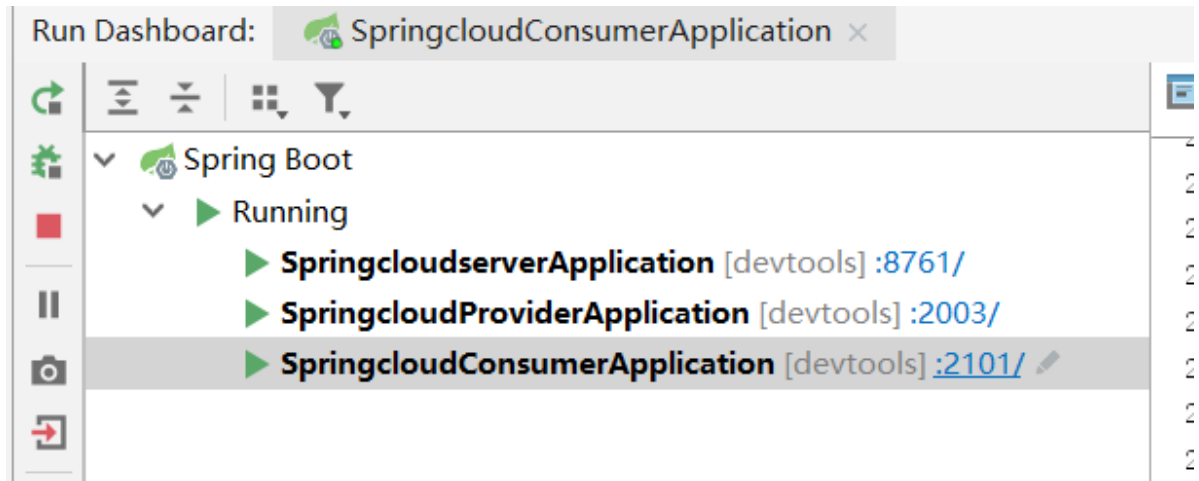
```

1  package com.soft863.springcloudconsumer;
2
3  import org.springframework.beans.factory.annotation.Autowired;
4  import org.springframework.boot.SpringApplication;
5  import org.springframework.boot.autoconfigure.SpringBootApplication;
6  import org.springframework.boot.web.client.RestTemplateBuilder;
7  import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
8  import org.springframework.context.annotation.Bean;
9  import org.springframework.context.annotation.ComponentScan;
10 import org.springframework.web.client.RestTemplate;
11
12 @SpringBootApplication
13 @ComponentScan("com.soft863")
14 @EnableDiscoveryClient
15 public class SpringcloudConsumerApplication {
16
17     @Autowired
18     private RestTemplateBuilder builder;
19
20     @Bean
21     public RestTemplate restTemplate(){
22         return builder.build();
23     }
24
25     public static void main(String[] args) {
26         SpringApplication.run(SpringcloudConsumerApplication.class, args);
27     }
28
29 }

```

6、启动

启动顺序:server—provider—consumer



服务查看<http://localhost:8761/>

浏览器中输入: <http://localhost:2101/login>

soft863恭喜您, 登录成功, 我来自于端口:2003

1.4、获取对象

1、添加User

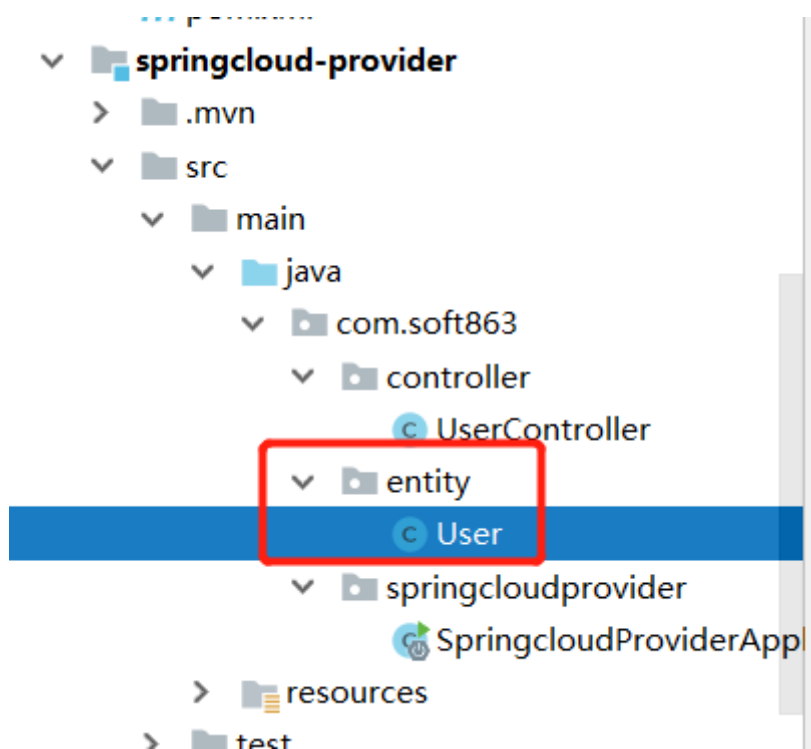
上述服务提供者和服务消费者两个工程下分别创建包entity, 然后创建User用户, 代码如下

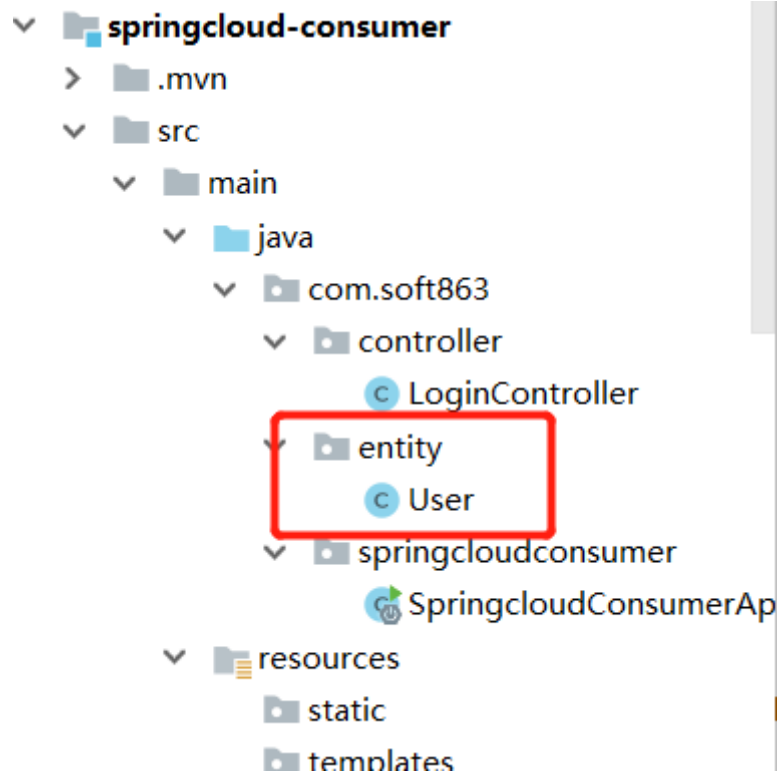
```
1 package com.soft863.entity;
2
3 public class User {
4     private Integer id;
5     private String username;
6     private String password;
7
8     public Integer getId() {
9         return id;
10    }
11
12    public void setId(Integer id) {
13        this.id = id;
14    }
15
16    public String getUsername() {
17        return username;
18    }
19
20    public void setUsername(String username) {
21        this.username = username;
22    }
23
24    public String getPassword() {
```

```

25     return password;
26 }
27
28 public void setPassword(String password) {
29     this.password = password;
30 }
31
32 @Override
33 public String toString() {
34     return "User{" +
35         "id=" + id +
36         ", username='" + username + '\'' +
37         ", password='" + password + '\'' +
38         '}';
39 }
40 }
41

```





2、springcloud-provider完善

UserController下面添加代码如下

```
1 @RequestMapping(value = "/getUserByID", method = RequestMethod.GET)
2 public User login(@RequestParam Integer id) {
3     User user = new User();
4     user.setId(1);
5     user.setUsername("admin");
6     user.setPassword("123456");
7     return user;
8 }
```

3、springcloud-consumer完善

LoginController下面添加

```
1 @GetMapping("/loginObj")
2 public String loginUser() {
3     User user = restTemplate.getForObject(userServerURL + "getUserByID?id=1",
4     user.class);
5     return "登录成功: " + user;
6 }
```

4、启动

重启上述两个服务

浏览器输入: <http://localhost:2101/loginObj>

结果:

登录成功: User{id=1, username='admin', password='123456'}

1.5、通过服务名称获取服务

LoginController类中添加如下代码

```
1 @GetMapping("/loginObj1")
2 public String loginUser1() {
3     String serverName = "user-service";
4     List<ServiceInstance> serviceInstances =
5     discoveryClient.getInstances(serverName);
6     if(serviceInstances==null||serviceInstances.size()==0){
7         return "未找到服务";
8     }
9     ServiceInstance serviceInstance = serviceInstances.get(0);
10    String url ="http://" +
11    serviceInstance.getHost()+":"+serviceInstance.getPort();
12    User user = restTemplate.getForObject(url + "/getUserById?id=1",
13    User.class);
14    return "登录成功: " + user;
15 }
```

<http://localhost:2101/loginObj1>

登录成功: User{id=1, username='admin', password='123456'}

也可以通过ribbon来使用服务名字获取服务

二、Ribbon

Spring Cloud Ribbon是一个基于HTTP和TCP的客户端负载均衡工具，它基于Netflix Ribbon实现。通过Spring Cloud的封装，可以让我们轻松地将面对服务的REST模块请求自动转换成客户端负载均衡的服务调用。Spring Cloud Ribbon几乎存在于每一个Spring Cloud构建的微服务和基础设施中。基于Spring Cloud Ribbon实现客户端负载均衡非常简单，主要由以下步骤：

1. 服务提供者需要启动多个服务实例并注册到一个或多个相关联的服务注册中心上；
2. 服务消费者直接通过带有@LoadBalanced注解的RestTemplate向服务提供者发送请求以实现客户端的负载均衡。

1、创建springcloud-provider1工程

根据上述二步骤创建一模一样的工程内容

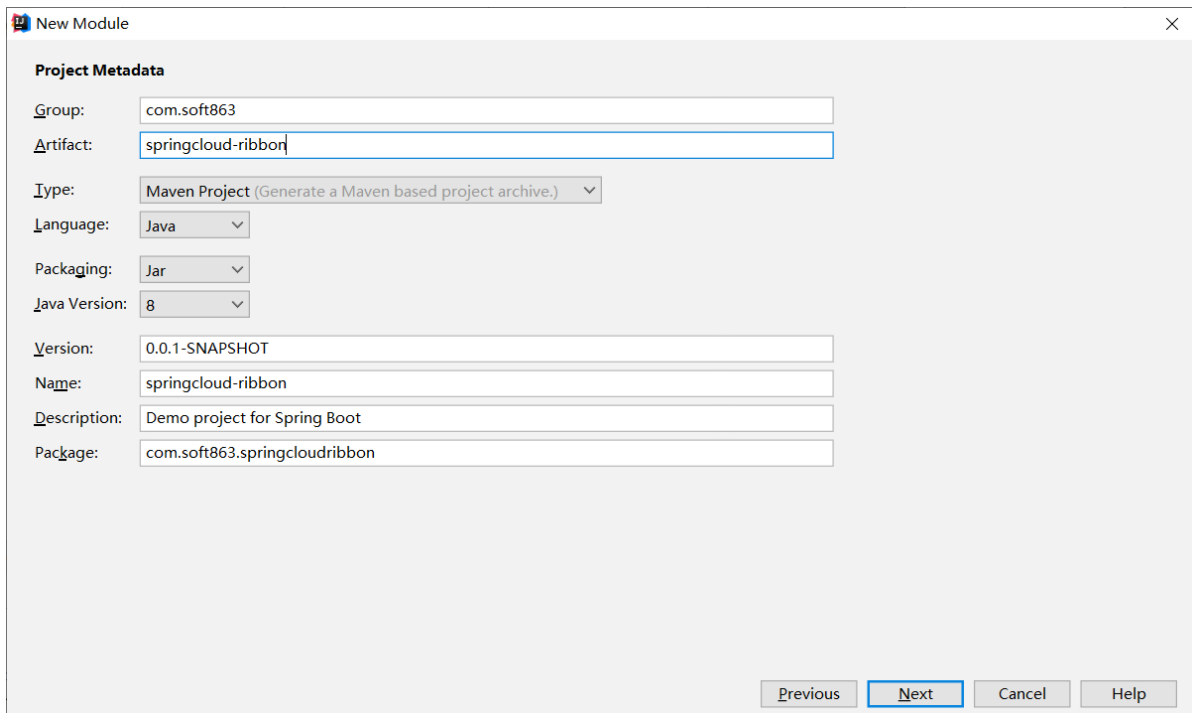
2、修改配置

将新工厂中properites配置文件中接口改成2004

将Controller代码修改如下

```
1 @RequestMapping(value = "/getUserByID", method = RequestMethod.GET)
2 public User login(@RequestParam Integer id) {
3     User user = new User();
4     user.setId(2);
5     user.setUsername("root");
6     user.setPassword("hadoop");
7     return user;
8 }
```

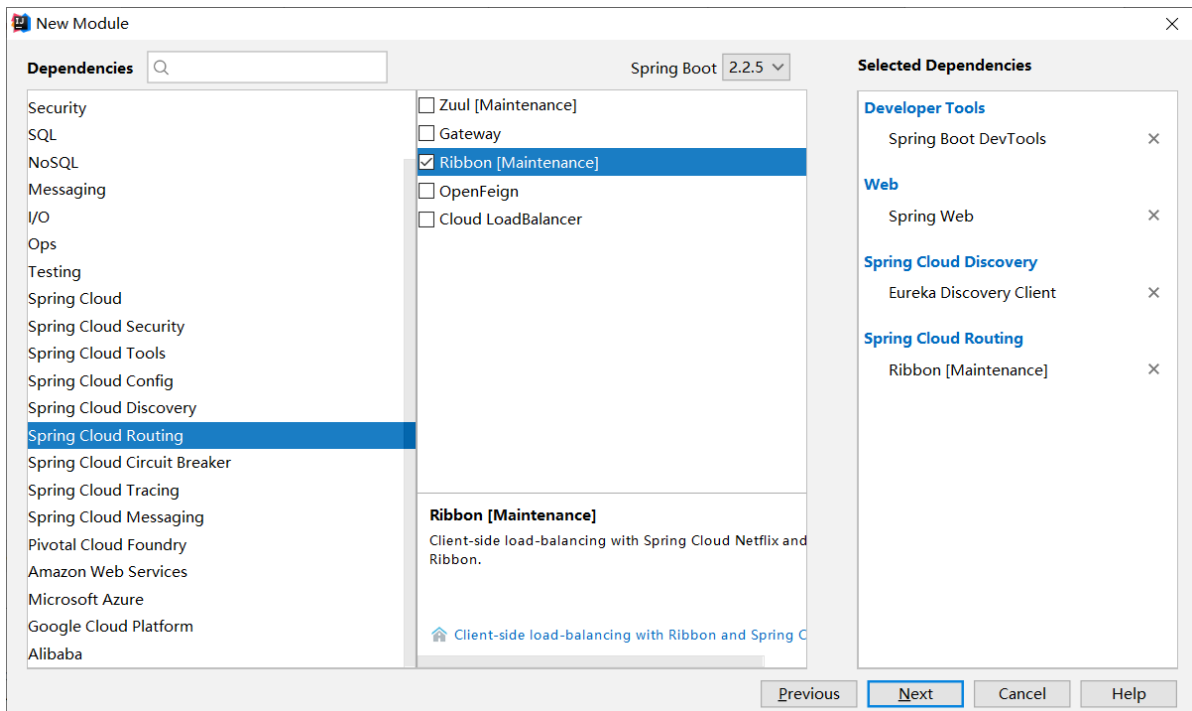
3、创建springcloud-ribbon工程



The 'New Module' dialog box is shown with the following fields:

- Group:** com.soft863
- Artifact:** springcloud-ribbon
- Type:** Maven Project (Generate a Maven based project archive.)
- Language:** Java
- Packaging:** Jar
- Java Version:** 8
- Version:** 0.0.1-SNAPSHOT
- Name:** springcloud-ribbon
- Description:** Demo project for Spring Boot
- Package:** com.soft863.springcloudribbon

Buttons at the bottom: Previous, Next, Cancel, Help.



The 'New Module' dialog box is shown with the following sections:

- Dependencies:** A list of categories on the left (Security, SQL, NoSQL, Messaging, I/O, Ops, Testing, Spring Cloud, Spring Cloud Security, Spring Cloud Tools, Spring Cloud Config, Spring Cloud Discovery, Spring Cloud Routing, Spring Cloud Circuit Breaker, Spring Cloud Tracing, Spring Cloud Messaging, Pivotal Cloud Foundry, Amazon Web Services, Microsoft Azure, Google Cloud Platform, Alibaba) and a list of dependencies on the right (Zuul [Maintenance], Gateway, Ribbon [Maintenance], OpenFeign, Cloud LoadBalancer). The 'Spring Boot' version is set to 2.2.5.
- Selected Dependencies:** A list of selected dependencies on the right, including Developer Tools (Spring Boot DevTools), Web (Spring Web), Spring Cloud Discovery (Eureka Discovery Client), and Spring Cloud Routing (Ribbon [Maintenance]).

Buttons at the bottom: Previous, Next, Cancel, Help.

创建实体类：User

```
1 package com.soft863.entity;
2
3 public class User {
4     private Integer id;
5     private String username;
6     private String password;
7
8     public Integer getId() {
9         return id;
10    }
11
12    public void setId(Integer id) {
13        this.id = id;
14    }
15
16    public String getUsername() {
17        return username;
18    }
19
20    public void setUsername(String username) {
21        this.username = username;
22    }
23
24    public String getPassword() {
25        return password;
26    }
27
28    public void setPassword(String password) {
29        this.password = password;
30    }
31
32    @Override
33    public String toString() {
34        return "User{" +
35            "id=" + id +
36            ", username='" + username + '\'' +
37            ", password='" + password + '\'' +
38            '}';
39    }
40 }
41
```

创建控制层类

```
1 package com.soft863.controller;
2
3 import com.soft863.entity.User;
4 import org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.cloud.client.ServiceInstance;
6 import org.springframework.cloud.client.discovery.DiscoveryClient;
7 import org.springframework.web.bind.annotation.GetMapping;
8 import org.springframework.web.bind.annotation.RestController;
9 import org.springframework.web.client.RestOperations;
```

```

10
11 import java.util.List;
12
13 @RestController
14 public class LoginController {
15     @Autowired
16     private RestOperations restTemplate;
17
18     @Autowired
19     private DiscoveryClient discoveryClient;
20     private String serverName = "user-service";
21
22     @GetMapping("/loginObj1")
23     public String loginUser1() {
24         List<ServiceInstance> serviceInstances =
25         discoveryClient.getInstances(serverName);
26         if (serviceInstances == null || serviceInstances.size() == 0) {
27             return "未找到服务";
28         }
29         ServiceInstance serviceInstance = serviceInstances.get(0);
30         String url = "http://" + serviceInstance.getHost() + ":" +
31         serviceInstance.getPort();
32         User user = restTemplate.getForObject(url + "/getUserByID?id=1",
33         User.class);
34         return "登录成功: " + user;
35     }
36
37     @GetMapping("/loginObj2")
38     public String loginUser2() {
39         String url = "http://" + serverName;
40         User user = restTemplate.getForObject(url + "/getUserByID?id=1",
41         User.class);
42         return "登录成功: " + user;
43     }
44 }

```

启动类添加内容

```

1 package com.soft863.springcloudribbon;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.boot.SpringApplication;
5 import org.springframework.boot.autoconfigure.SpringBootApplication;
6 import org.springframework.boot.web.client.RestTemplateBuilder;
7 import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
8 import org.springframework.cloud.client.loadbalancer.LoadBalanced;
9 import org.springframework.context.annotation.Bean;
10 import org.springframework.context.annotation.ComponentScan;
11 import org.springframework.web.client.RestTemplate;
12
13 @SpringBootApplication
14 @ComponentScan("com.soft863")
15 @EnableDiscoveryClient
16 public class SpringcloudRibbonApplication {
17     @Autowired

```

```

18     private RestTemplateBuilder builder;
19
20     @Bean
21     @LoadBalanced
22     public RestTemplate restTemplate() {
23         return builder.build();
24     }
25
26     public static void main(String[] args) {
27         SpringApplication.run(SpringcloudRibbonApplication.class, args);
28     }
29 }
30

```

添加配置文件

```

1  server.port=2100
2  #应用名称
3  spring.application.name=user-consumer
4  # EurekaServer地址
5  eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka/
6  # 当调用getHostname获取实例的hostname时，返回ip而不是host名称
7  eureka.instance.prefer-ip-address=true
8  # 指定自己的ip信息，不指定的话会自己寻找
9  eureka.instance.ip-address=127.0.0.1
10
11 user-service.url=http://127.0.0.1:2003/
12 user-
    service.ribbon.NFLoadBalancerRuleClassName=com.netflix.loadbalancer.RandomRule

```

4、运行验证

启动工程

浏览器中查看服务: <http://localhost:8761/>

Instances currently registered with Eureka			
Application	AMIs	Availability Zones	Status
USER-CONSUMER	n/a (1)	(1)	UP (1) - wcj-pc:user-consumer:2101
USER-SERVICE	n/a (2)	(2)	UP (2) - wcj-pc:user-service:2003, wcj-pc:user-service:2004

浏览器中输入: <http://localhost:2100/loginObj2>

登录成功: User{id=1, username='admin', password='123456'}

登录成功: User{id=2, username='root', password='hadoop'}

两个会随机出现，验证负载均衡效果

负载均衡策略配置，默认RoundRobinRule轮询策略，可以进行设置

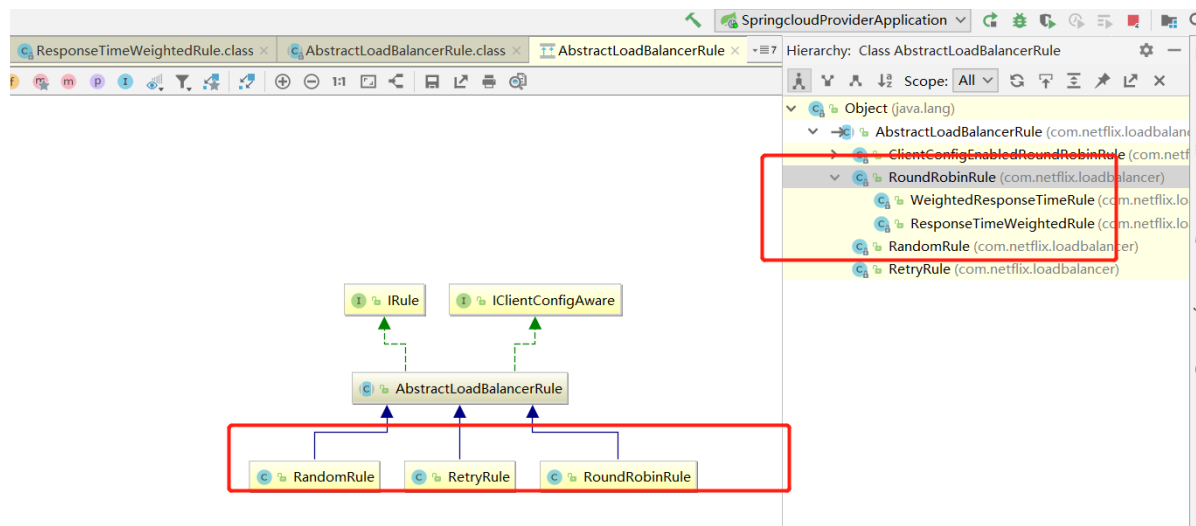
consumer工程配置文件中设置:

{服务名称}.ribbon.NFLoadBalancerRuleClassName

```
1 user-  
  service.ribbon.NFLoadBalancerRuleClassName=com.netflix.loadbalancer.RandomRule
```

```
server.port=2101  
#应用名称  
spring.application.name=user-consumer  
# EurekaServer地址  
eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka/  
# 当调用getHostname获取实例的hostname时，返回ip而不是host名称  
eureka.instance.prefer-ip-address=true  
# 指定自己的ip信息，不指定的话会自己寻找  
eureka.instance.ip-address=127.0.0.1  
  
user-service.url=http://127.0.0.1:2003/  
user-service.ribbon.NFLoadBalancerRuleClassName=com.netflix.loadbalancer.RandomRule
```

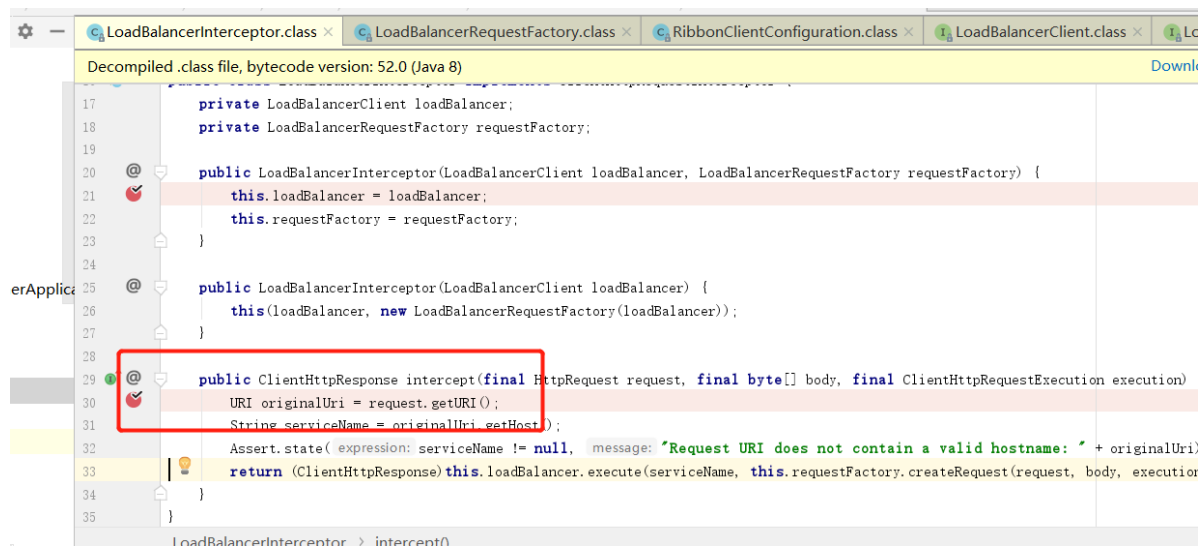
其中负载均衡可以选的策略有



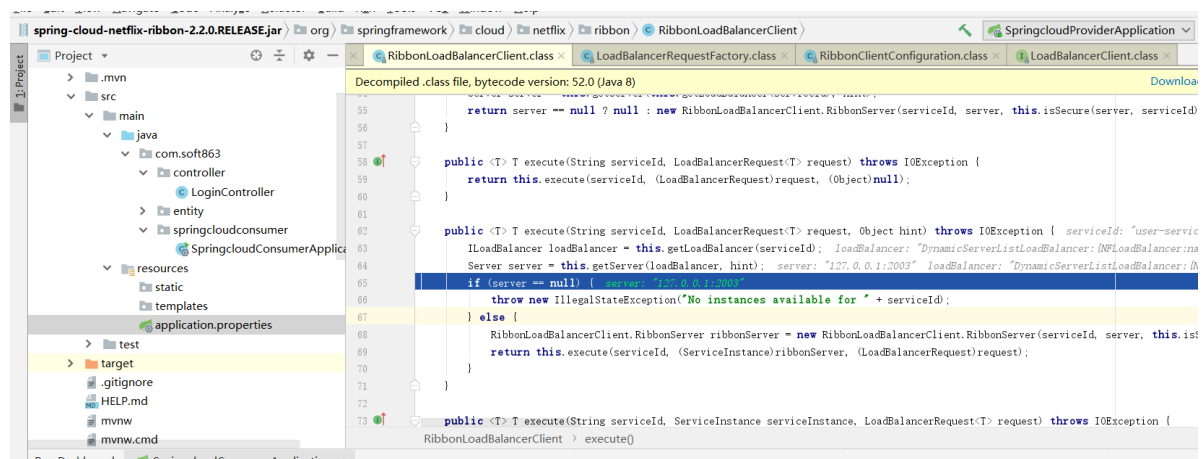
SpringcloudConsumerApplication工程中返回RestTemplate方法上添加LoadBalanced注解

负载均衡器： `LoadBalancerInterceptor`

可以在此类中设置断点，进行分析



```
17 private LoadBalancerClient loadBalancer;
18 private LoadBalancerRequestFactory requestFactory;
19
20 @
21 public LoadBalancerInterceptor(LoadBalancerClient loadBalancer, LoadBalancerRequestFactory requestFactory) {
22     this.loadBalancer = loadBalancer;
23     this.requestFactory = requestFactory;
24 }
25
26 @
27 public LoadBalancerInterceptor(LoadBalancerClient loadBalancer) {
28     this(loadBalancer, new LoadBalancerRequestFactory(loadBalancer));
29 }
30
31 @
32 public ClientHttpResponse intercept(final HttpRequest request, final byte[] body, final ClientHttpRequestExecution execution) {
33     URI originalUri = request.getURI();
34     String serviceName = originalUri.getHost();
35     Assert.state(expression: serviceName != null, message: "Request URI does not contain a valid hostname: " + originalUri);
36     return (ClientHttpResponse) this.loadBalancer.execute(serviceName, this.requestFactory.createRequest(request, body, execution));
37 }
```



```
55 return server == null ? null : new RibbonLoadBalancerClient.RibbonServer(serviceId, server, this.isSecure(server, serviceId));
56 }
57
58 public <T> T execute(String serviceId, LoadBalancerRequest<T> request) throws IOException {
59     return this.execute(serviceId, (LoadBalancerRequest)request, (Object)null);
60 }
61
62 public <T> T execute(String serviceId, LoadBalancerRequest<T> request, Object hint) throws IOException {
63     LoadBalancer loadBalancer = this.getLoadBalancer(serviceId);
64     Server server = this.getServer(loadBalancer, hint);
65     if (server == null) {
66         throw new IllegalStateException("No instances available for " + serviceId);
67     }
68     RibbonLoadBalancerClient.RibbonServer ribbonServer = new RibbonLoadBalancerClient.RibbonServer(serviceId, server, this.isSecure(server, serviceId));
69     return this.execute(serviceId, (ServiceInstance)ribbonServer, (LoadBalancerRequest)request);
70 }
71
72 public <T> T execute(String serviceId, ServiceInstance serviceInstance, LoadBalancerRequest<T> request) throws IOException {
73     return this.execute(serviceId, (LoadBalancerRequest)request, serviceInstance);
74 }
```

注意使用负载均衡后不能用ip进行访问，需要使用服务名访问

三、Hystrix

在分布式环境中，许多服务依赖关系中的一些必然会失败。Hystrix是一个库，它通过添加延迟容忍和容错逻辑来帮助控制这些分布式服务之间的交互。Hystrix通过隔离服务之间的访问点、停止跨服务的级联故障并提供回退选项来实现这一点，所有这些选项都提高了系统的总体弹性

Hystrix的设计目的如下：

- 为通过第三方客户端库访问的依赖项(通常通过网络)提供保护和控制延迟和故障。
- 停止复杂分布式系统中的级联故障。
- 故障快速恢复。
- 在可能的情况下，后退并优雅地降级。
- 启用近实时监视、警报和操作控制。
- 为了解决什么问题？

复杂分布式体系结构中的应用程序有几十个依赖项，每个依赖项在某个时候都不可避免地会失败。如果主机应用程序没有从这些外部故障中隔离出来，那么它就有可能与这些外部故障一起宕机。

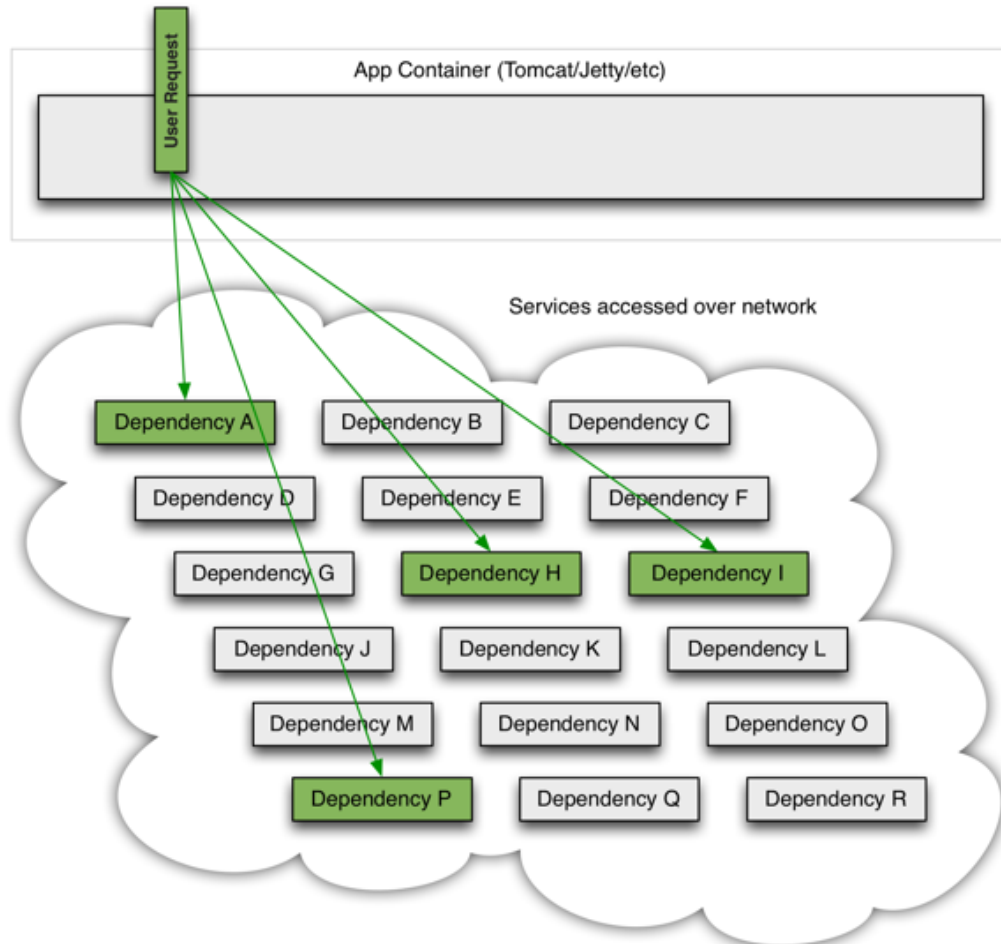
例如，对于一个依赖于30个服务的应用程序，其中每个服务都有99.99%的正常运行时间，您可以这样期望：

99.9930 = 99.7% uptime 0.3% of 1 billion requests = 3,000,000 failures 2+ hours
downtime/month even if all dependencies have excellent uptime.

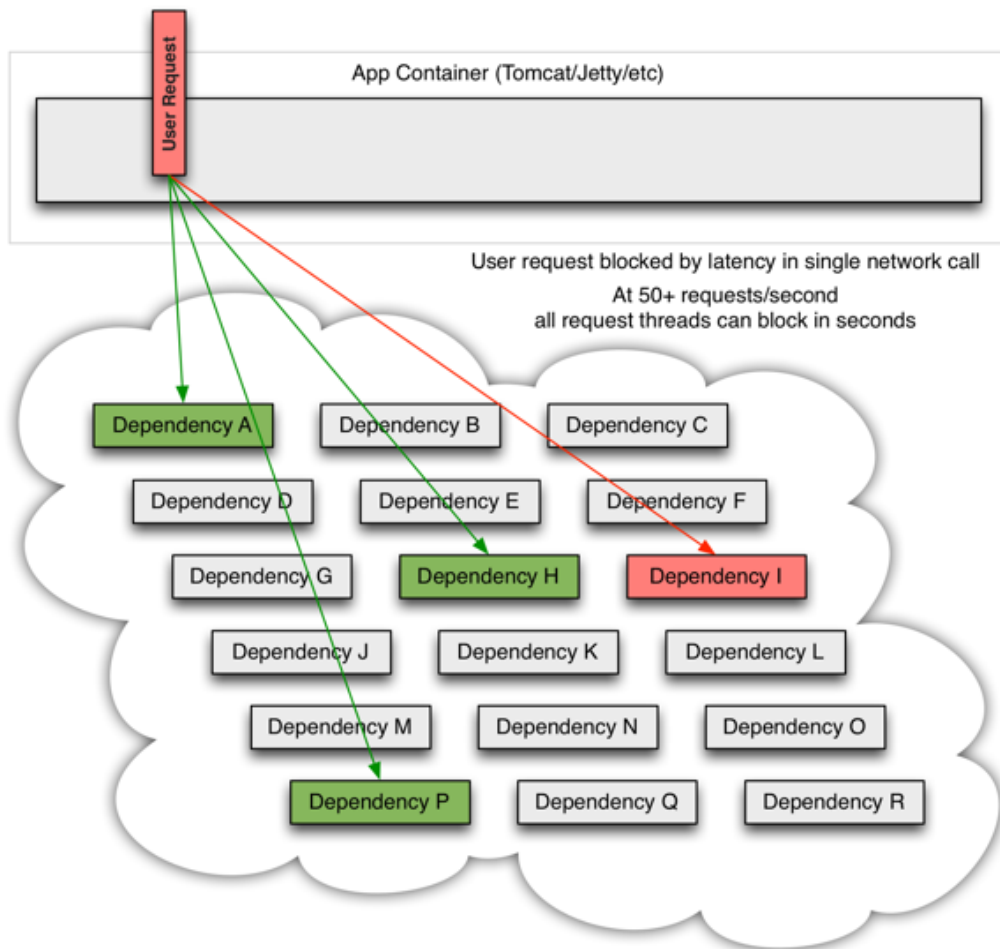
现实通常更糟。即使当所有依赖项都运行良好时，即使0.01%的停机时间对几十个服务中的每个服务的总体影响也相当于一个月潜在的停机时间(如果您不为恢复而设计整个系统)。

如下面的图演变：

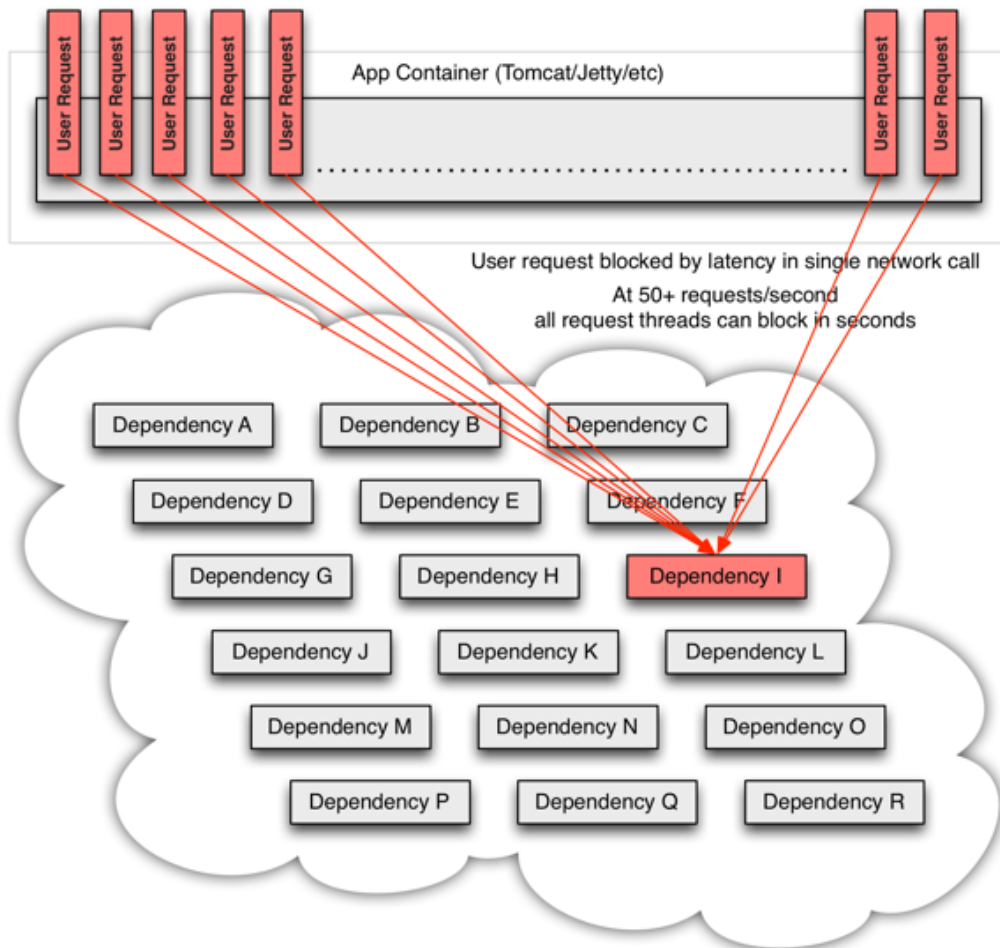
当一切正常时，请求流可以是这样的：



当许多后端系统之一成为潜在，它可以阻止整个用户请求：



对于高流量，一个后端依赖项成为潜在，可能会导致所有服务器上的所有资源在几秒钟内饱和。应用程序中通过网络或客户机库到达可能导致网络请求的每个点都是潜在故障的来源。比故障更糟的是，这些应用程序还可能导致服务之间的延迟增加，从而备份队列、线程和其他系统资源，从而导致系统中出现更多级联故障。



1、创建工程springcloud-consumer_hystrix

New Module

Project Metadata

Group: com.soft863

Artifact: springcloud-consumer_hystrix

Type: Maven Project (Generate a Maven based project archive.)

Language: Java

Packaging: Jar

Java Version: 8

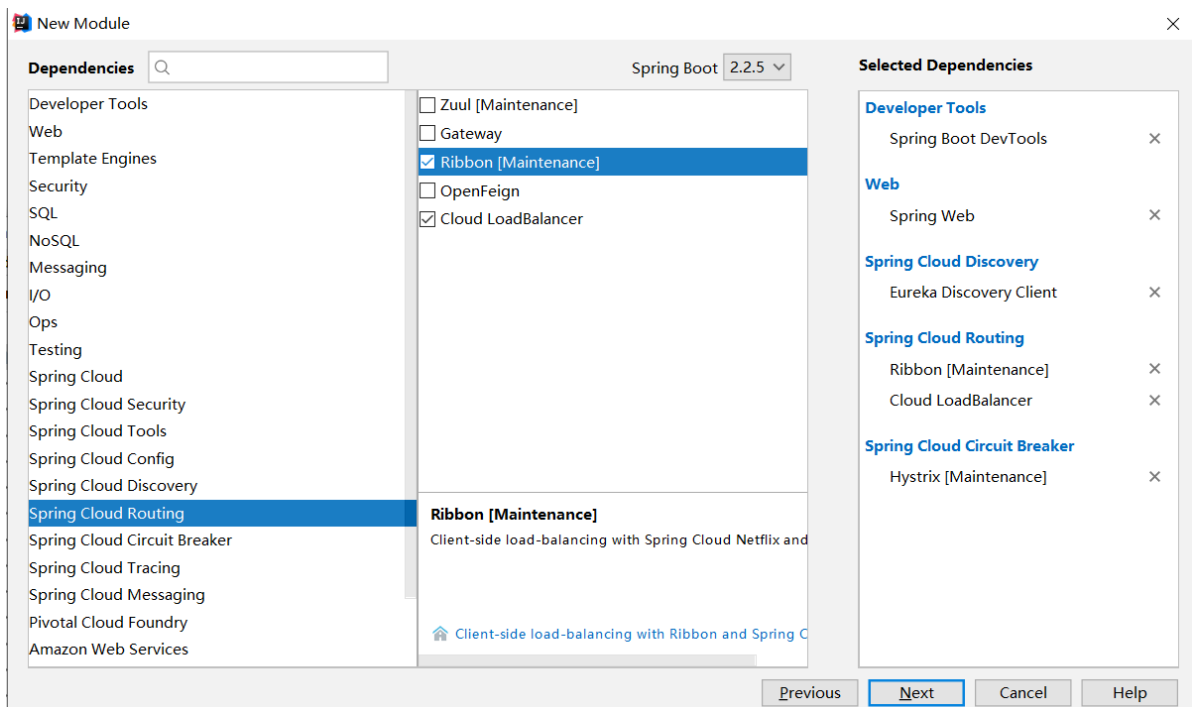
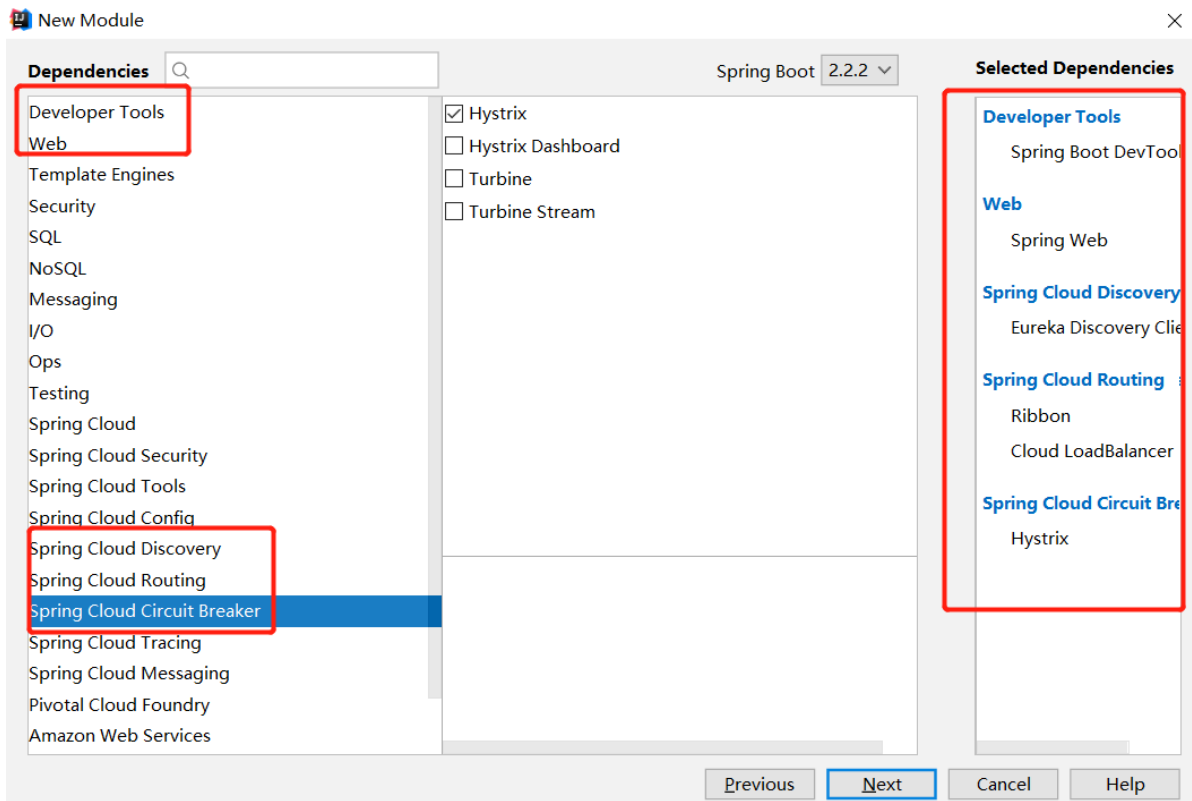
Version: 0.0.1-SNAPSHOT

Name: springcloud-consumer_hystrix

Description: Demo project for Spring Boot

Package: com.soft863.springcloudconsumer_hystrix

Previous Next Cancel Help



2、添加pom

自动生成如下

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  https://maven.apache.org/xsd/maven-4.0.0.xsd">
4     <modelVersion>4.0.0</modelVersion>
5     <parent>
6         <groupId>org.springframework.boot</groupId>
7         <artifactId>spring-boot-starter-parent</artifactId>

```

```
8         <version>2.2.2.RELEASE</version>
9         <relativePath/> <!-- lookup parent from repository -->
10    </parent>
11    <groupId>com.soft863</groupId>
12    <artifactId>springcloud-consumer1</artifactId>
13    <version>0.0.1-SNAPSHOT</version>
14    <name>springcloud-consumer1</name>
15    <description>Demo project for Spring Boot</description>
16
17    <properties>
18        <java.version>1.8</java.version>
19        <spring-cloud.version>Hoxton.RELEASE</spring-cloud.version>
20    </properties>
21
22    <dependencies>
23        <dependency>
24            <groupId>org.springframework.boot</groupId>
25            <artifactId>spring-boot-starter-web</artifactId>
26        </dependency>
27        <dependency>
28            <groupId>org.springframework.cloud</groupId>
29            <artifactId>spring-cloud-starter-loadbalancer</artifactId>
30        </dependency>
31        <dependency>
32            <groupId>org.springframework.cloud</groupId>
33            <artifactId>spring-cloud-starter-netflix-eureka-
client</artifactId>
34        </dependency>
35        <dependency>
36            <groupId>org.springframework.cloud</groupId>
37            <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
38        </dependency>
39        <dependency>
40            <groupId>org.springframework.cloud</groupId>
41            <artifactId>spring-cloud-starter-netflix-ribbon</artifactId>
42        </dependency>
43
44        <dependency>
45            <groupId>org.springframework.boot</groupId>
46            <artifactId>spring-boot-devtools</artifactId>
47            <scope>runtime</scope>
48            <optional>true</optional>
49        </dependency>
50        <dependency>
51            <groupId>org.springframework.boot</groupId>
52            <artifactId>spring-boot-starter-test</artifactId>
53            <scope>test</scope>
54            <exclusions>
55                <exclusion>
56                    <groupId>org.junit.vintage</groupId>
57                    <artifactId>junit-vintage-engine</artifactId>
58                </exclusion>
59            </exclusions>
60        </dependency>
61    </dependencies>
```

```

62
63     <dependencyManagement>
64         <dependencies>
65             <dependency>
66                 <groupId>org.springframework.cloud</groupId>
67                 <artifactId>spring-cloud-dependencies</artifactId>
68                 <version>${spring-cloud.version}</version>
69                 <type>pom</type>
70                 <scope>import</scope>
71             </dependency>
72         </dependencies>
73     </dependencyManagement>
74
75     <build>
76         <plugins>
77             <plugin>
78                 <groupId>org.springframework.boot</groupId>
79                 <artifactId>spring-boot-maven-plugin</artifactId>
80             </plugin>
81         </plugins>
82     </build>
83
84 </project>

```

3、添加配置文件

```

1  server.port=2102
2  #应用名称
3  spring.application.name=user-consumer_hystrix
4  # EurekaServer地址
5  eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka/
6  # 当调用getHostname获取实例的hostname时，返回ip而不是host名称
7  eureka.instance.prefer-ip-address=true
8  # 指定自己的ip信息，不指定的话会自己寻找
9  eureka.instance.ip-address=127.0.0.1
10 #服务请求超时时间，超时则使用熔断器，不配置时此项默认一秒
11 hystrix.command.default.execution.isolation.thread.timeoutInMilliseconds=100
   0

```

4、启动类上添加注解

```

1  @SpringBootApplication
2  @EnableDiscoveryClient
3  @EnableCircuitBreaker
4  @ComponentScan("com.soft863")

```

其中@SpringCloudApplication注解是

@SpringBootApplication+@EnableDiscoveryClient+@EnableCircuitBreaker的结合，所以也可以配置成

```

1  @SpringCloudApplication
2  @ComponentScan("com.soft863")

```

综合如下:

```
1 package com.soft863.springcloudconsumer_hystrix;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.boot.SpringApplication;
5 import org.springframework.boot.autoconfigure.SpringBootApplication;
6 import org.springframework.boot.web.client.RestTemplateBuilder;
7 import org.springframework.cloud.client.SpringCloudApplication;
8 import org.springframework.cloud.client.loadbalancer.LoadBalanced;
9 import org.springframework.context.annotation.Bean;
10 import org.springframework.context.annotation.ComponentScan;
11 import org.springframework.web.client.RestTemplate;
12
13 @SpringCloudApplication
14 @ComponentScan("com.soft863")
15 public class SpringcloudConsumerHystrixApplication {
16     @Autowired
17     private RestTemplateBuilder builder;
18
19     @Bean
20     @LoadBalanced
21     public RestTemplate restTemplate(){
22         return builder.build();
23     }
24     public static void main(String[] args) {
25         SpringApplication.run(SpringcloudConsumerHystrixApplication.class,
26         args);
27     }
28 }
```

添加实体类

```
1 package com.soft863.entity;
2
3 public class User {
4     private Integer id;
5     private String username;
6     private String password;
7
8     public Integer getId() {
9         return id;
10    }
11
12    public void setId(Integer id) {
13        this.id = id;
14    }
15
16    public String getUsername() {
17        return username;
18    }
19
20    public void setUsername(String username) {
21        this.username = username;
22    }
23 }
```

```

22     }
23
24     public String getPassword() {
25         return password;
26     }
27
28     public void setPassword(String password) {
29         this.password = password;
30     }
31
32     @Override
33     public String toString() {
34         return "User{" +
35             "id=" + id +
36             ", username='" + username + '\'' +
37             ", password='" + password + '\'' +
38             '}';
39     }
40 }
41

```

5、添加Controller类

配置局部熔断器

```

1  @GetMapping("/loginObj1/{id}")
2  @HystrixCommand(fallbackMethod = "feedbackError")
3  public String loginUser1(@PathVariable int id) {
4      logger.debug("开始执行...");
5      return "登录成功: " + restTemplate.getForObject("http://user-
service/getUserByID?id=1", User.class);
6  }

```

```

1  private String feedbackError(int id){
2      logger.error("查询信息失败"+id);
3      return "网络堵塞，请稍后重试";
4  }

```

配置全局熔断器

添加方法（注意方法中不要有参数，且该类中所有方法返回类型要与处理失败的方法的返回类型一致）

```

1  private String feedbackError(){
2      logger.error("查询信息失败");
3      return "已经超时了，请联系管理员";
4  }

```

类上面添加注解

```

1  @DefaultProperties(defaultFallback = "feedbackError")

```

方法上使用

```

1  @GetMapping("/loginObj2/{id}")
2  @HystrixCommand
3  public String loginUser2(@PathVariable int id) {
4      logger.info("开始执行..." + id);
5      try {
6          Thread.sleep(1500);
7      } catch (InterruptedException e) {
8          e.printStackTrace();
9      }
10     return "登录成功: " + restTemplate.getForObject("http://user-
service/getUserByID?id=1", User.class);
11 }

```

完整代码如下

```

1  package com.soft863.controller;
2
3  import com.netflix.hystrix.contrib.javanica.annotation.DefaultProperties;
4  import com.netflix.hystrix.contrib.javanica.annotation.HystrixCommand;
5  import com.soft863.entity.User;
6  import org.slf4j.Logger;
7  import org.slf4j.LoggerFactory;
8  import org.springframework.beans.factory.annotation.Autowired;
9  import org.springframework.web.bind.annotation.GetMapping;
10 import org.springframework.web.bind.annotation.PathVariable;
11 import org.springframework.web.bind.annotation.RequestMapping;
12 import org.springframework.web.bind.annotation.RestController;
13 import org.springframework.web.client.RestOperations;
14
15 @RestController
16 @DefaultProperties(defaultFallback = "feedbackError")
17 public class LoginController {
18     private final Logger logger =
19     LoggerFactory.getLogger(LoginController.class);
20     @Autowired
21     private RestOperations restTemplate;
22
23     @GetMapping("/loginObj1/{id}")
24     @HystrixCommand(fallbackMethod = "feedbackError")
25     public String loginUser1(@PathVariable int id) {
26         logger.debug("开始执行...");
27         return "登录成功: " + restTemplate.getForObject("http://user-
service/getUserByID?id=1", User.class);
28     }
29
30     @GetMapping("/loginObj2/{id}")
31     @HystrixCommand
32     public String loginUser2(@PathVariable int id) {
33         logger.info("开始执行..." + id);
34         try {
35             Thread.sleep(1500);
36         } catch (InterruptedException e) {
37             e.printStackTrace();
38         }
39     }
40 }

```

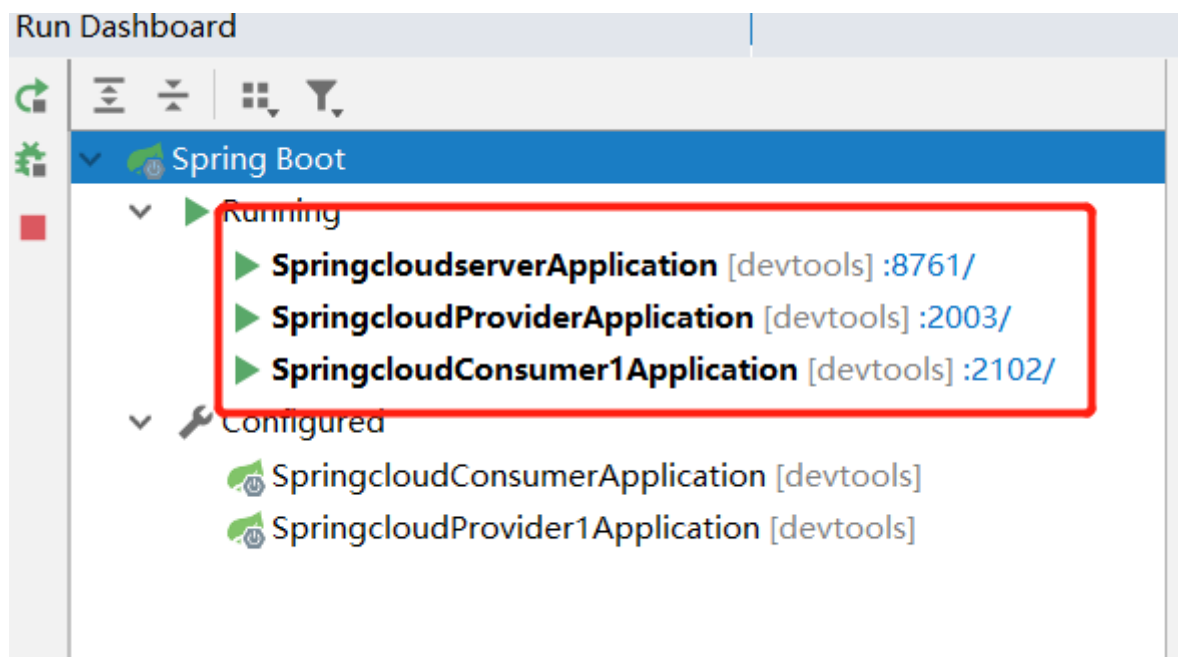
```

38         return "登录成功: " + restTemplate.getForObject("http://user-
service/getUserByID?id=1", User.class);
39     }
40
41     private String feedbackError(int id){
42         logger.error("查询信息失败"+id);
43         return "网络堵塞, 请稍后重试";
44     }
45     private String feedbackError(){
46         logger.error("查询信息失败");
47         return "已经超时了, 请联系管理员";
48     }
49 }

```

6、启动与测试

启动工程测试



浏览器输入: <http://localhost:2102/loginObj1/100>

结果如下:

登录成功: User{id=1, username='admin', password='123456'}

浏览器输入: <http://localhost:2102/loginObj2/100>

结果如下:

已经超时了, 请联系管理员

因为代码中休眠, 超过1秒, 可以看出使用了全局熔断器

此时将代码修改如下

```

1  @GetMapping("/loginObj1/{id}")
2  @HystrixCommand(fallbackMethod = "feedbackError")
3  public String loginUser1(@PathVariable int id) {
4      logger.debug("开始执行...");
5      try {
6          Thread.sleep(3000);
7      } catch (InterruptedException e) {
8          e.printStackTrace();
9      }
10     return "登录成功: " + restTemplate.getForObject("http://user-
        service/userByid?id=1", User.class);
11 }

```

重启服务，浏览其中输入<http://localhost:2102/loginObj1/100>

结果如下：

网络堵塞，请稍后重试

因为代码中休眠3秒，超过1秒，可以看出使用了局部熔断器，因为参数类型一样，都是 (int id)

调整配置如下

```

1  hystrix.command.default.execution.isolation.thread.timeoutInMilliseconds=2000

```

重启服务，浏览其中输入<http://localhost:2102/loginObj2/100>

登录成功: User{id=1, username='admin', password='123456'}

可以看出此服务休眠 Thread.sleep(1500)，在配置的2秒超时时间范围内，故服务正常运行

浏览器中输入<http://localhost:2102/loginObj1/100>

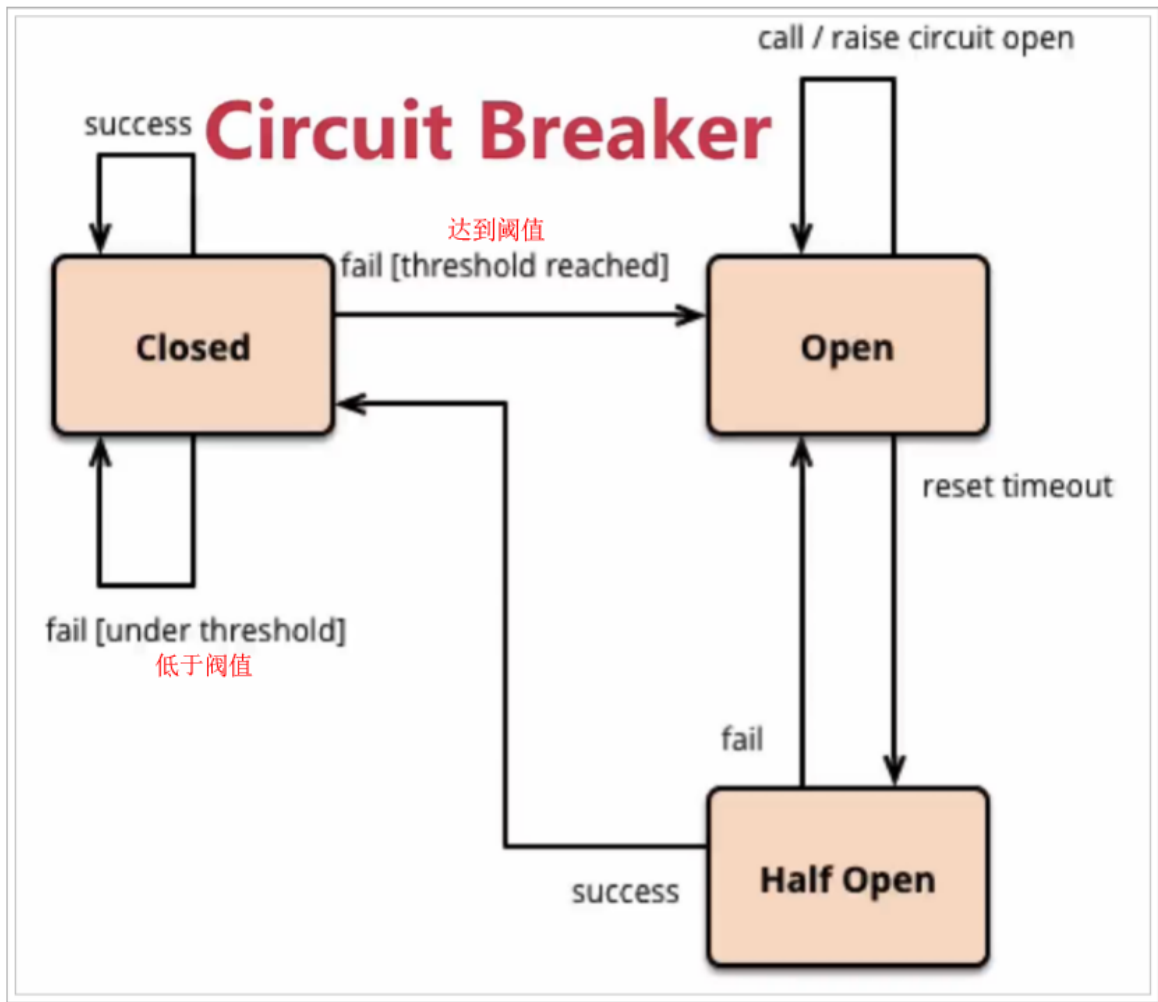
因为此时 Thread.sleep(3000)，超时配置的超时时间范围，故依然使用熔断器，结果如下

网络堵塞，请稍后重试

7、服务熔断

在服务熔断中，使用的熔断器，也叫断路器，其英文单词为：Circuit Breaker 熔断机制与家里使用的电路熔断原理类似；当如果电路发生短路的时候能立刻熔断电路，避免发生灾难。在分布式系统中应用服务熔断后；服务调用方可以自己进行判断哪些服务反应慢或存在大量超时，可以针对这些服务进行主动熔断，防止整个系统被拖垮。Hystrix的服务熔断机制，可以实现弹性容错；当服务请求情况好转之后，可以自动重连。通过断路的方式，将后续请求直接拒绝，一段时间（默认5秒）之后允许部分请求通过，如果调用成功则回到断路器关闭状态，否则继续打开，拒绝请求的服务。

状态机有3个状态：Closed：关闭状态（断路器关闭），所有请求都正常访问。Open：打开状态（断路器打开），所有请求都会被降级。Hystrix会对请求情况计数，当一定时间内失败请求百分比达到阈值，则触发熔断，断路器会完全打开。默认失败比例的阈值是50%，请求次数少不低于20次。Half Open：半开状态，不是永久的，断路器打开后会进入休眠时间（默认是5S）。随后断路器会自动进入半开状态。此时会释放部分请求通过，若这些请求都是健康的，则会关闭断路器，否则继续保持打开，再次进行休眠计时



修改配置

```
1 #配置熔断策略
2 # 熔断触发小请求次数，默认值是20
3 hystrix.command.default.circuitBreaker.requestVolumeThreshold=5
4 # 熔断后休眠时长，默认值5秒
5 hystrix.command.default.circuitBreaker.sleepWindowInMilliseconds=20000
6 # 触发熔断错误比例阈值，默认值50%
7 hystrix.command.default.circuitBreaker.errorThresholdPercentage=10
8 # 熔断超时设置，默认为1秒
9 hystrix.command.default.execution.isolation.thread.timeoutInMilliseconds=2000
```

Controller中添加

```
1 private String feedbackError(int id){
2     logger.error("查询信息失败"+id);
3     return "网络堵塞，请稍后重试";
4 }
```

```
1 @GetMapping("/loginObj3/{id}")
2 @HystrixCommand(fallbackMethod = "feedbackException")
3 public String loginUser3(@PathVariable int id) {
4     logger.info("开始执行..." + id);
5     if(id==0){
6         throw new RuntimeException("异常测试");
7     }
8     return "登录成功: " + restTemplate.getForObject("http://user-
    service/getUserByID?id=1", User.class);
9 }
```

完整代码如下：

测试：

浏览器里面输入：<http://localhost:2102/loginObj3/1>

登录成功: User{id=1, username='admin', password='123456'}

浏览器里面输入：<http://localhost:2102/loginObj3/0>

被测试异常了

连续执行6次

然后再执行：<http://localhost:2102/loginObj3/1>

被测试异常了

然后等待20秒，再执行<http://localhost:2102/loginObj3/1>

登录成功: User{id=1, username='admin', password='123456'}

四、Feign

Feign也叫伪装，Feign可以把Rest的请求进行隐藏，伪装成类似SpringMVC的Controller一样。你不用再自己拼接url，拼接参数等等操作，一切都交给Feign去做。

使用Ribbon+RestTemplate时,利用RestTemplate对http请求的封装处理,形成了一套模板化的调用方法.但是在实际开发中,由于对服务依赖的调用可能不止一处,往往一个接口会被多处调用,所以通常都会针对每个微服务自行封装一些客户端类来包装这些依赖服务的调用.所以,Feign在此基础上做了进一步封装,由他来帮助我们定义和实现依赖服务接口的定义.在Feign的实现下,我们只需要创建一个接口并使用注解的方式来配置它(以前是Dao接口上标注Mapper注解,现在是一个微服务接口上面标注一个Feign注解即可),即可完成对服务提供方的接口绑定,简化了使用Spring cloud Ribbon时,自动封装服务调用客户端的开发量。

1、创建工程springcloud-consumer_feign

New Module

Project Metadata

Group:

Artifact:

Type:

Language:

Packaging:

Java Version:

Version:

Name:

Description:

Package:

New Module

Spring Boot 2.2.2

Dependencies

- Developer Tools
- Web
- Template Engines
- Security
- SQL
- NoSQL
- Messaging
- I/O
- Ops
- Testing
- Spring Cloud
- Spring Cloud Security
- Spring Cloud Tools
- Spring Cloud Config
- Spring Cloud Discovery
- Spring Cloud Routing
- Spring Cloud Circuit Breaker
- Spring Cloud Tracing
- Spring Cloud Messaging
- Pivotal Cloud Foundry
- Amazon Web Services

☐ Zuul

☐ Gateway

☒ Ribbon

☒ OpenFeign

☐ Cloud LoadBalancer

Selected Dependencies

- Developer Tools
 - Spring Boot DevTools
- Web
 - Spring Web
- Spring Cloud Routing
 - Ribbon
 - OpenFeign
- Spring Cloud Circuit Breaker
 - Hystrix

2、pom中添加依赖

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  https://maven.apache.org/xsd/maven-4.0.0.xsd">
4     <modelVersion>4.0.0</modelVersion>
5     <parent>
6         <groupId>org.springframework.boot</groupId>
7         <artifactId>spring-boot-starter-parent</artifactId>
8         <version>2.2.2.RELEASE</version>
9         <relativePath/> <!-- lookup parent from repository -->
```

```
10     </parent>
11     <groupId>com.soft863</groupId>
12     <artifactId>springcloud-consumer_zuul</artifactId>
13     <version>0.0.1-SNAPSHOT</version>
14     <name>springcloud-consumer_zuul</name>
15     <description>Demo project for Spring Boot</description>
16
17     <properties>
18         <java.version>1.8</java.version>
19         <spring-cloud.version>Hoxton.RELEASE</spring-cloud.version>
20     </properties>
21
22     <dependencies>
23         <dependency>
24             <groupId>org.springframework.boot</groupId>
25             <artifactId>spring-boot-starter-web</artifactId>
26         </dependency>
27         <dependency>
28             <groupId>org.springframework.cloud</groupId>
29             <artifactId>spring-cloud-starter-netflix-eureka-
server</artifactId>
30         </dependency>
31         <dependency>
32             <groupId>org.springframework.cloud</groupId>
33             <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
34         </dependency>
35         <dependency>
36             <groupId>org.springframework.cloud</groupId>
37             <artifactId>spring-cloud-starter-netflix-ribbon</artifactId>
38         </dependency>
39         <dependency>
40             <groupId>org.springframework.cloud</groupId>
41             <artifactId>spring-cloud-starter-netflix-zuul</artifactId>
42         </dependency>
43         <dependency>
44             <groupId>org.springframework.cloud</groupId>
45             <artifactId>spring-cloud-starter-openfeign</artifactId>
46         </dependency>
47
48         <dependency>
49             <groupId>org.springframework.boot</groupId>
50             <artifactId>spring-boot-devtools</artifactId>
51             <scope>runtime</scope>
52             <optional>true</optional>
53         </dependency>
54         <dependency>
55             <groupId>org.springframework.boot</groupId>
56             <artifactId>spring-boot-starter-test</artifactId>
57             <scope>test</scope>
58             <exclusions>
59                 <exclusion>
60                     <groupId>org.junit.vintage</groupId>
61                     <artifactId>junit-vintage-engine</artifactId>
62                 </exclusion>
63             </exclusions>
```

```

64         </dependency>
65     </dependencies>
66
67     <dependencyManagement>
68         <dependencies>
69             <dependency>
70                 <groupId>org.springframework.cloud</groupId>
71                 <artifactId>spring-cloud-dependencies</artifactId>
72                 <version>${spring-cloud.version}</version>
73                 <type>pom</type>
74                 <scope>import</scope>
75             </dependency>
76         </dependencies>
77     </dependencyManagement>
78
79     <build>
80         <plugins>
81             <plugin>
82                 <groupId>org.springframework.boot</groupId>
83                 <artifactId>spring-boot-maven-plugin</artifactId>
84             </plugin>
85         </plugins>
86     </build>
87
88 </project>
89
90

```

3、添加配置文件

```

1  server.port=2103
2  #应用名称
3  spring.application.name=user-consumer_feign
4  # EurekaServer地址
5  eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka/
6  # 当调用getHostname获取实例的hostname时，返回ip而不是host名称
7  eureka.instance.prefer-ip-address=true
8  # 指定自己的ip信息，不指定的话会自己寻找
9  eureka.instance.ip-address=127.0.0.1

```

4、添加启动类注解

```

1  package com.soft863.springcloudconsumer_feign;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5  import org.springframework.cloud.client.circuitbreaker.EnableCircuitBreaker;
6  import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
7  import org.springframework.cloud.openfeign.EnableFeignClients;
8  import org.springframework.context.annotation.ComponentScan;
9
10
11  @SpringBootApplication

```

```

12 @EnableDiscoveryClient
13 @EnableCircuitBreaker
14 @ComponentScan("com.soft863")
15 @EnableFeignClients(basePackages = "com.soft863")
16 public class SpringcloudConsumerFeignApplication {
17
18     public static void main(String[] args) {
19         SpringApplication.run(SpringcloudConsumerFeignApplication.class,
20             args);
21     }
22 }

```

5、创建相关类

User

```

1 package com.soft863.entity;
2
3 public class User {
4     private Integer id;
5     private String username;
6     private String password;
7
8     public Integer getId() {
9         return id;
10    }
11
12    public void setId(Integer id) {
13        this.id = id;
14    }
15
16    public String getUsername() {
17        return username;
18    }
19
20    public void setUsername(String username) {
21        this.username = username;
22    }
23
24    public String getPassword() {
25        return password;
26    }
27
28    public void setPassword(String password) {
29        this.password = password;
30    }
31
32    @Override
33    public String toString() {
34        return "User{" +
35            "id=" + id +
36            ", username='" + username + '\'' +
37            ", password='" + password + '\'' +
38            '}';
39    }

```

UserClient

```
1 package com.soft863.client;
2
3 import com.soft863.entity.User;
4 import feign.Param;
5 import org.springframework.cloud.openfeign.FeignClient;
6 import org.springframework.stereotype.Component;
7 import org.springframework.web.bind.annotation.GetMapping;
8 import org.springframework.web.bind.annotation.PathVariable;
9 import org.springframework.web.bind.annotation.RequestParam;
10
11 @Component
12 @FeignClient("user-service")
13 public interface UserClient {
14     @GetMapping("/getUserByID1/{id}")
15     User getUserByID1(@PathVariable("id") int id);
16
17     @GetMapping("/getUserByID")
18     User getUserByID(@RequestParam("id") int id);
19
20 }
```

LoginController

```
1 @RestController
2 public class LoginController1 {
3
4     @Autowired
5     private UserClient userClient;
6
7     @GetMapping("/login1/{id}")
8     User getUserByID1(@PathVariable int id) {
9         return userClient.getUserByID1(id);
10    }
11
12    @GetMapping("/login")
13    User getUserByID(@RequestParam int id) {
14        return userClient.getUserByID(id);
15    }
16
17 }
```

6、测试

<http://localhost:2103/login?id=1>

<http://localhost:2103/login1/1>

GET

http://localhost:2103/login1/1

Params

Authorization

Headers (7)

Body

Pre-request Script

Tests

Settings

Query Params

	KEY	VALUE
	Key	Value

Body

Cookies

Headers (5)

Test Results

Pretty

Raw

Preview

Visualize

JSON

1

{

2

"id": 1,

3

"username": "admin",

4

"password": "123456"

5

}

GET

http://localhost:2103/login?id=1

Params

Authorization

Headers (7)

Body

Pre-request Script

Tests

Settings

Query Params

	KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/>	id	1	
	Key	Value	Description

Body

Cookies

Headers (5)

Test Results

Status: 200 OKTime: 1022msSize:

Pretty

Raw

Preview

Visualize

JSON

1

{

2

"id": 1,

3

"username": "admin",

4

"password": "123456"

5

}

五、Zuul

为什么需要网关呢？

我们知道我们要进入一个服务本身，很明显我们没有特别好的办法，直接输入IP地址+端口号，我们知道这样的做法很糟糕的，这样的做法大有问题，首先暴露了我们实体机器的IP地址，别人一看你的IP地址就知道服务部署在哪里，让别人很方便的进行攻击操作。

第二，我们这么多服务，我们是不是要挨个调用它呀，我们这里假设做了个权限认证，我们每一个客户访问的都是跑在不同机器上的不同的JVM上的服务程序，我们每一个服务都需要一个服务认证，这样做烦不烦呀，明显是很烦的。

那么我们这时候面临着这两个极其重要的问题，这时我们就需要一个办法解决它们。首先，我们看IP地址的暴露和IP地址写死后带来的单点问题，我是不是对这么服务本身我也要动态的维护它服务的列表呀，我需要调用这服务本身，是不是也要一个负载均衡一样的玩意，

还有关于IP地址暴露的玩意，我是不是需要做一个代理呀，像Nginx的反向代理一样的东西，还有这玩意上部署公共的模块，比如所有入口的权限校验的东西。因此我们现在需要Zuul API网关。它就解决了上面的问题，你想调用某个服务，它会给你映射，把你服务的IP地址映射成

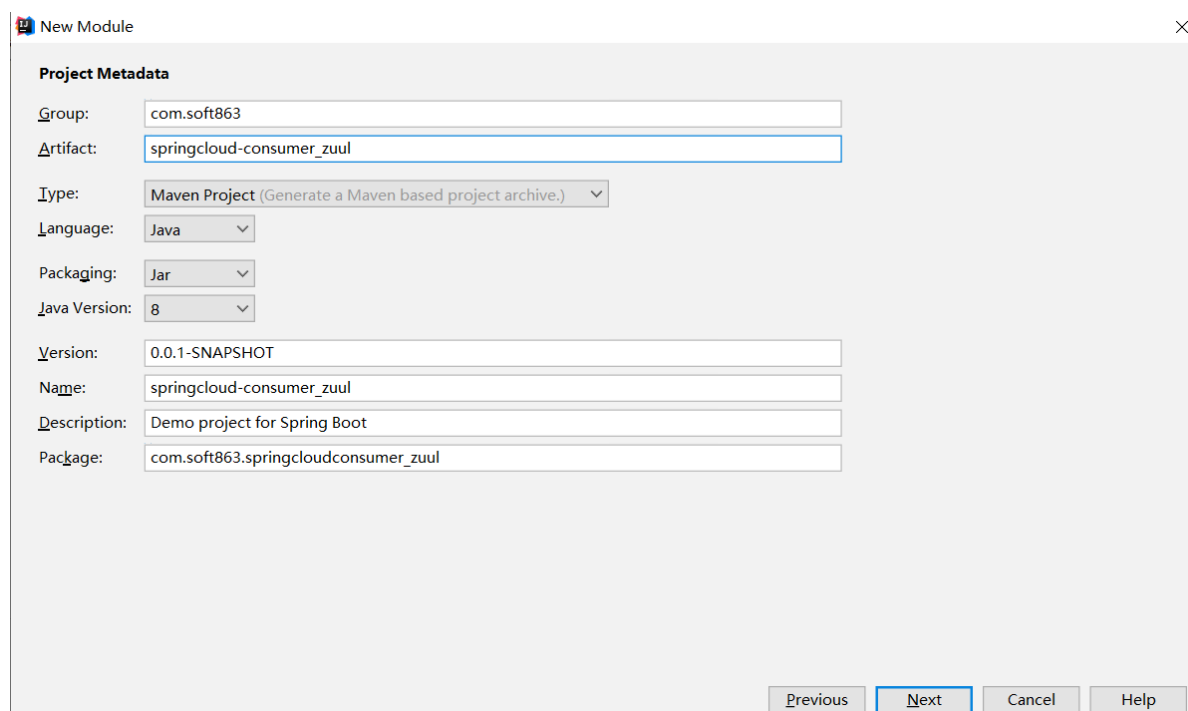
某个路径，你输入该路径，它匹配到了，它就去替你访问这个服务，它会有个请求转发的过程，像Nginx一样，服务机器的具体实例，它不会直接去访问IP，它会去Eureka注册中心拿到服务的实例ID，即服务的名字。我再次使用客户端的负载均衡ribbon访问其中服务实例中的一台。

API网关主要为了服务本身对外的调用该怎么调用来解决的，还有解决权限校验的问题，你可以在这里整合调用一系列过滤器的，例如整合shiro,springsecurity之类的东西。

Zuul可以通过加载动态过滤机制，从而实现以下各项功能：

- 1.验证与安全保障: 识别面向各类资源的验证要求并拒绝那些与要求不符的请求。
- 2.审查与监控: 在边缘位置追踪有意义数据及统计结果，从而为我们带来准确的生产状态结论。
- 3.动态路由: 以动态方式根据需要将请求路由至不同后端集群处。
- 4.压力测试: 逐渐增加指向集群的负载流量，从而计算性能水平。
- 5.负载分配: 为每一种负载类型分配对应容量，并弃用超出限定值的请求。
- 6.静态响应处理: 在边缘位置直接建立部分响应，从而避免其流入内部集群。
- 7.多区域弹性: 跨越AWS区域进行请求路由，旨在实现ELB使用多样化并保证边缘位置与使用者尽可能接近。

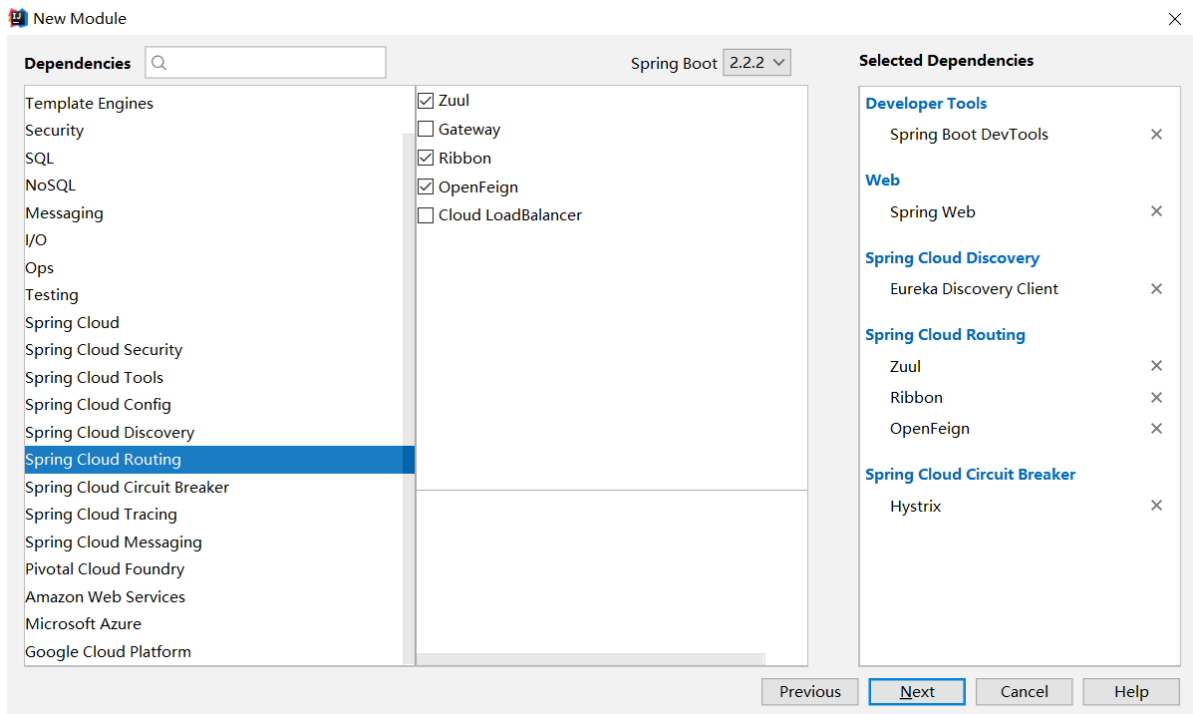
1、创建工程springcloud-consumer_zuul



The screenshot shows the 'New Module' dialog box in an IDE. The 'Project Metadata' section contains the following fields:

- Group: com.soft863
- Artifact: springcloud-consumer_zuul
- Type: Maven Project (Generate a Maven based project archive.)
- Language: Java
- Packaging: Jar
- Java Version: 8
- Version: 0.0.1-SNAPSHOT
- Name: springcloud-consumer_zuul
- Description: Demo project for Spring Boot
- Package: com.soft863.springcloudconsumer_zuul

At the bottom right, there are four buttons: 'Previous', 'Next' (highlighted), 'Cancel', and 'Help'.



2、pom中添加依赖

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   https://maven.apache.org/xsd/maven-4.0.0.xsd">
4      <modelVersion>4.0.0</modelVersion>
5      <parent>
6          <groupId>org.springframework.boot</groupId>
7          <artifactId>spring-boot-starter-parent</artifactId>
8          <version>2.2.2.RELEASE</version>
9          <relativePath/> <!-- lookup parent from repository -->
10     </parent>
11     <groupId>com.soft863</groupId>
12     <artifactId>springcloud-consumer_zuul</artifactId>
13     <version>0.0.1-SNAPSHOT</version>
14     <name>springcloud-consumer_zuul</name>
15     <description>Demo project for Spring Boot</description>
16
17     <properties>
18         <java.version>1.8</java.version>
19         <spring-cloud.version>Hoxton.RELEASE</spring-cloud.version>
20     </properties>
21
22     <dependencies>
23         <dependency>
24             <groupId>org.springframework.boot</groupId>
25             <artifactId>spring-boot-starter-web</artifactId>
26         </dependency>
27         <dependency>
28             <groupId>org.springframework.cloud</groupId>
29             <artifactId>spring-cloud-starter-netflix-eureka-
   client</artifactId>
30         </dependency>

```

```
31     <dependency>
32         <groupId>org.springframework.cloud</groupId>
33         <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
34     </dependency>
35     <dependency>
36         <groupId>org.springframework.cloud</groupId>
37         <artifactId>spring-cloud-starter-netflix-ribbon</artifactId>
38     </dependency>
39     <dependency>
40         <groupId>org.springframework.cloud</groupId>
41         <artifactId>spring-cloud-starter-netflix-zuul</artifactId>
42     </dependency>
43     <dependency>
44         <groupId>org.springframework.cloud</groupId>
45         <artifactId>spring-cloud-starter-openfeign</artifactId>
46     </dependency>
47
48     <dependency>
49         <groupId>org.springframework.boot</groupId>
50         <artifactId>spring-boot-devtools</artifactId>
51         <scope>runtime</scope>
52         <optional>true</optional>
53     </dependency>
54     <dependency>
55         <groupId>org.springframework.boot</groupId>
56         <artifactId>spring-boot-starter-test</artifactId>
57         <scope>test</scope>
58         <exclusions>
59             <exclusion>
60                 <groupId>org.junit.vintage</groupId>
61                 <artifactId>junit-vintage-engine</artifactId>
62             </exclusion>
63         </exclusions>
64     </dependency>
65 </dependencies>
66
67 <dependencyManagement>
68     <dependencies>
69         <dependency>
70             <groupId>org.springframework.cloud</groupId>
71             <artifactId>spring-cloud-dependencies</artifactId>
72             <version>${spring-cloud.version}</version>
73             <type>pom</type>
74             <scope>import</scope>
75         </dependency>
76     </dependencies>
77 </dependencyManagement>
78
79 <build>
80     <plugins>
81         <plugin>
82             <groupId>org.springframework.boot</groupId>
83             <artifactId>spring-boot-maven-plugin</artifactId>
84         </plugin>
85     </plugins>
```

```
86     </build>
87
88 </project>
```

3、添加配置文件

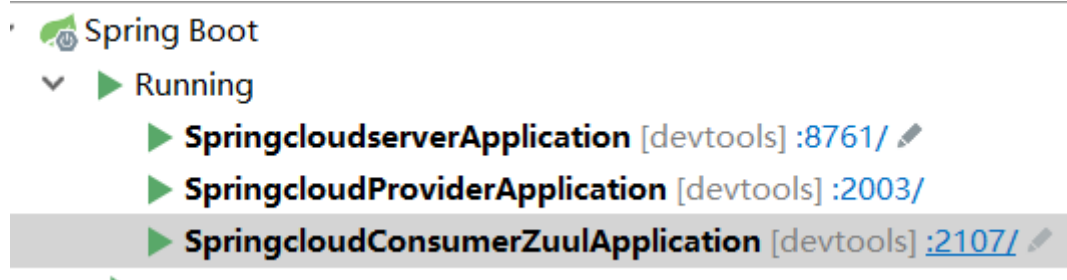
```
1  server.port=2107
2  #应用名称
3  spring.application.name=user-consumer_zuul
4
5  # EurekaServer地址
6  eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka/
7  # 当调用getHostname获取实例的hostname时，返回ip而不是host名称
8  eureka.instance.prefer-ip-address=true
9  # 指定自己的ip信息，不指定的话会自己寻找
10 eureka.instance.ip-address=127.0.0.1
```

4、启动类中添加@EnableZuulProxy注解

```
1  @SpringCloudApplication
2  @ComponentScan("com.soft863")
3  @EnableZuulProxy
4  public class SpringcloudConsumerZuulApplication {
5
6      public static void main(String[] args) {
7          SpringApplication.run(SpringcloudConsumerZuulApplication.class,
8              args);
9      }
10 }
```

5、验证

启动



验证: <http://localhost:8761/>

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
USER-CONSUMER_ZUUL	n/a (1)	(1)	UP (1) - wcj-pc:user-consumer_zuul:2104
USER-SERVICE	n/a (1)	(1)	UP (1) - wcj-pc:user-service:2003

浏览器输入: <http://localhost:2107/user-service/test>

可以直接访问到user-server服务的test方法

hello user

以上采用服务注册自动注册路由。如果不想使用这种方式，可以自定义添加

6、自定义路由

配置如下

```
1 #网关前缀
2 zuul.prefix=/super
3 #忽略服务发现自动映射，可以指定，这里全部忽略
4 zuul.ignored-services='*'
```

验证: <http://localhost:2107/super/user-service/test>

hello user

附录：SpringBoot参考资料

<https://www.springcloud.cc/>

-
-
-