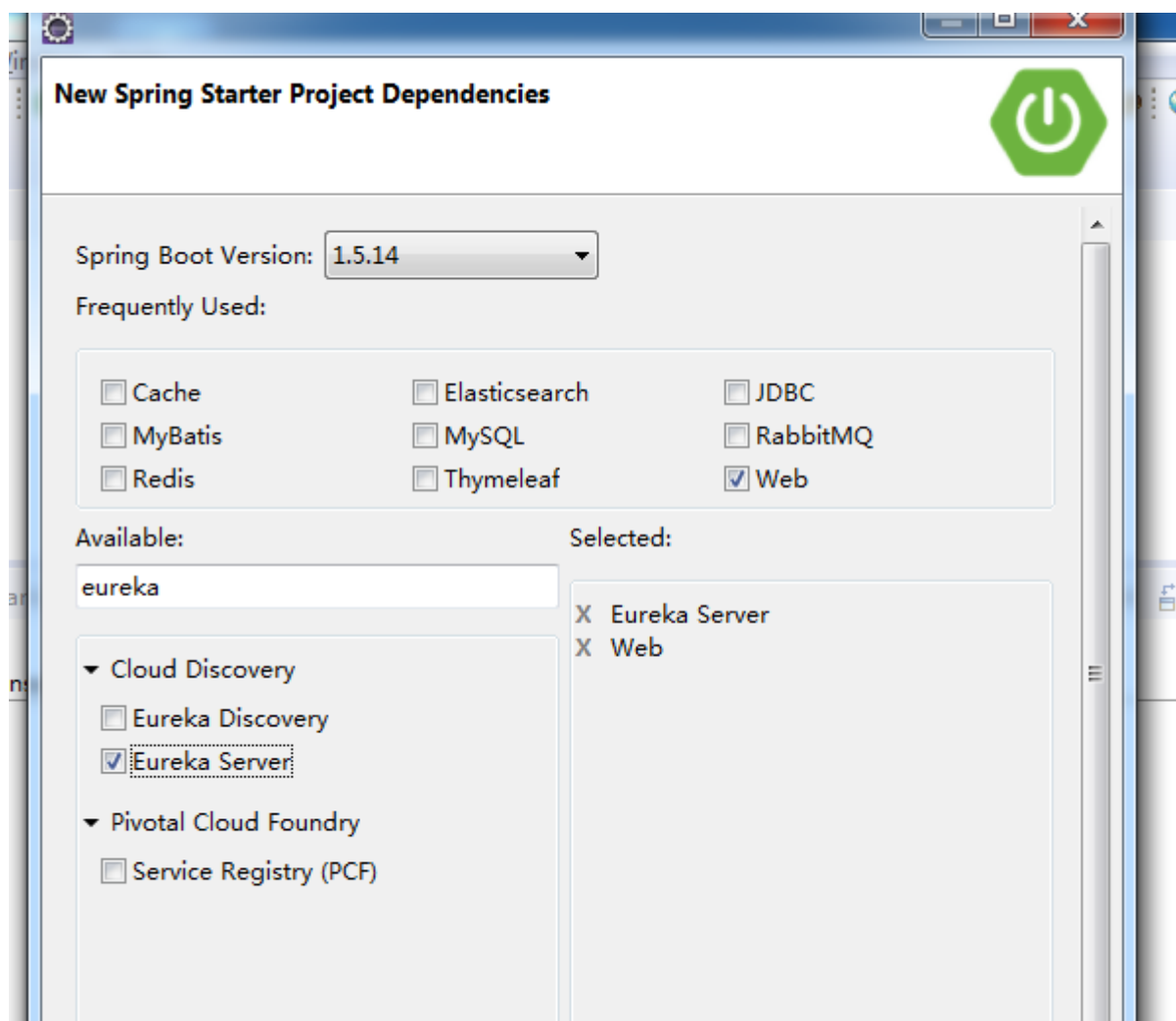


一、项目创建

分别创建eureka-server,eureka-provider,eureka-consumer三个项目。

eureka-server项目选中web和Eureka Server模块，eureka-provider,eureka-consumer都选中web和Eureka Discovery模块



二、eureka-server代码示例

1.配置文件

```
1 server:
2   port: 8761
3 eureka:
4   instance:
5     hostname: eureka-server    #eureka实例的主机名
6   client:
7     register-with-eureka: false  #不把自己注册到
    eureka上
8     fetch-registry: false        #不从eureka上来
    获取服务的注册地址
9     service-url:
10      defaultZone: http://localhost:8761/eureka
```

2.@EnableEurekaServer注解

在主启动类上使用这个注解：开启基于注解的EurekaServer

```

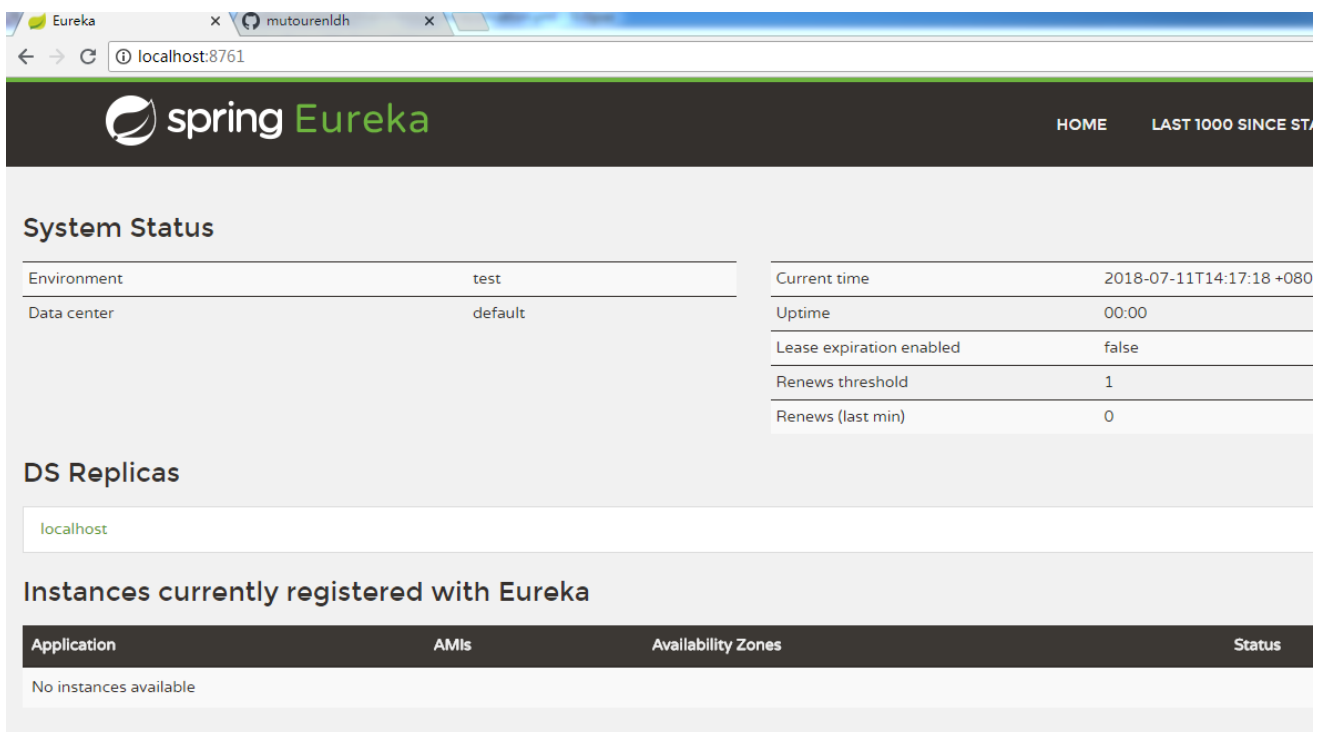
1 @EnableEurekaServer
2 @SpringBootApplication
3 public class SpringbootEurekaServerApplication {
4
5     public static void main(String[] args) {
6
7         SpringApplication.run(SpringbootEurekaServerAppli
8             cation.class, args);
9     }
10 }

```

3.测试

地址：<http://localhost:8761/>

效果如下：



The screenshot shows the Spring Eureka web interface in a browser window. The address bar shows 'localhost:8761'. The page has a dark header with the 'spring Eureka' logo and navigation links 'HOME' and 'LAST 1000 SINCE ST'. Below the header, there is a 'System Status' section with two tables. The first table shows 'Environment: test' and 'Data center: default'. The second table shows 'Current time: 2018-07-11T14:17:18 +0800', 'Uptime: 00:00', 'Lease expiration enabled: false', 'Renews threshold: 1', and 'Renews (last min): 0'. Below this is a 'DS Replicas' section with a single entry 'localhost'. At the bottom, there is a section 'Instances currently registered with Eureka' with a table that has columns 'Application', 'AMIs', 'Availability Zones', and 'Status'. The table content shows 'No instances available'.

System Status	
Environment	test
Data center	default

Current time	2018-07-11T14:17:18 +0800
Uptime	00:00
Lease expiration enabled	false
Renews threshold	1
Renews (last min)	0

DS Replicas

localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
No instances available			

三、eureka-provider代码

1.配置文件

```
1 server:
2   port: 8002
3 spring:
4   application:
5     name: provider-ticket
6
7 eureka:
8   instance:
9     prefer-ip-address: true    #注册服务的时候使用
    IP进行注册
10  client:
11    service-url:
12      defaultZone: http://localhost:8761/eureka
```

2. service

```
1 @Service
2 public class TicketService {
3
4     public String getTicket() {
5         System.out.println(8002);
6         return "第一张电影票";
7     }
8
9 }
```

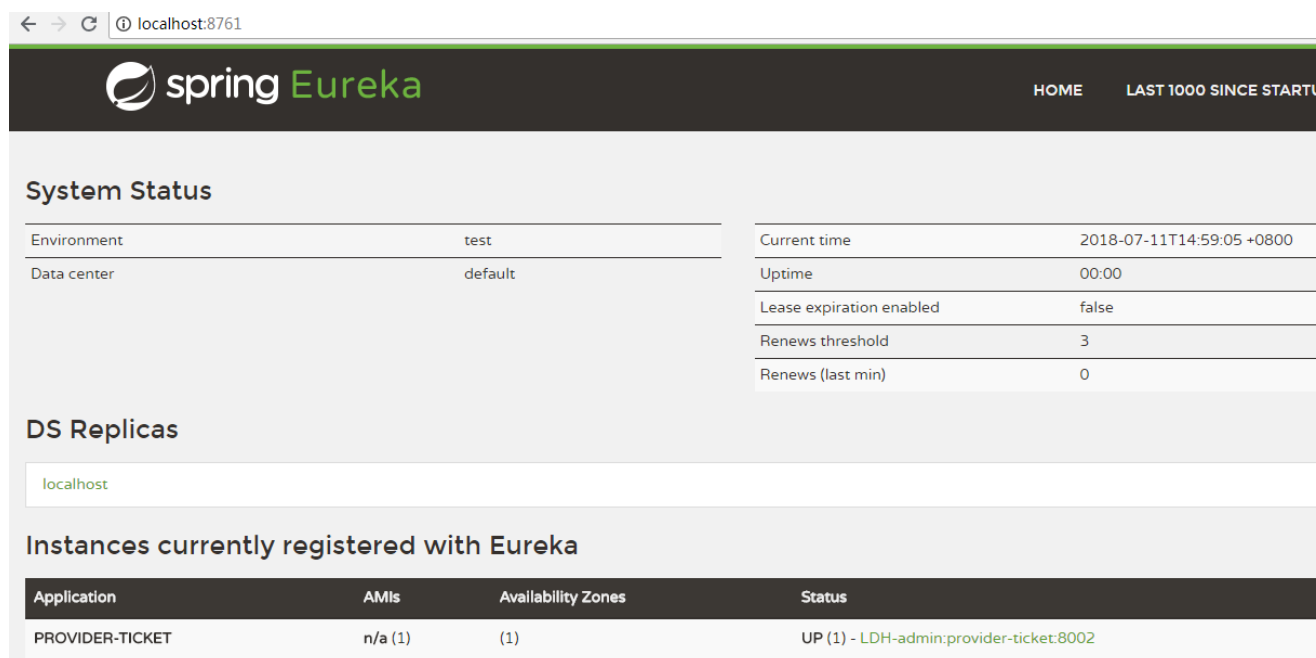
3. controller

```
1 @RestController
2 public class TicketController {
3
4     @Autowired
5     TicketService ticketService;
6
7     @GetMapping("/ticket")
8     public String getTicket() {
9         String ticket =
10 ticketService.getTicket();
11         return ticket;
12     }
13 }
```

4.测试

地址：<http://localhost:8002/ticket>

访问注册中心：<http://localhost:8761/>，我们看到已经有一个服务提供者注册在eureka上



The screenshot shows the Spring Eureka web interface. The browser address bar indicates the URL is `localhost:8761`. The page header features the "spring Eureka" logo and navigation links for "HOME" and "LAST 1000 SINCE STARTU".

System Status

Environment	test	Current time	2018-07-11T14:59:05 +0800
Data center	default	Uptime	00:00
		Lease expiration enabled	false
		Renews threshold	3
		Renews (last min)	0

DS Replicas

localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
PROVIDER-TICKET	n/a (1)	(1)	UP (1) - LDH-admin:provider-ticket:8002

四、eureka-consumer代码

1.配置文件

```
1 spring:
2   application:
3     name: consumer-user
4 server:
5   port: 8200
6
7
8 eureka:
9   instance:
10     prefer-ip-address: true    #注册服务的时候使用
    IP进行注册
11   client:
12     service-url:
13       defaultZone: http://localhost:8761/eureka
```

2. 主启动程序

```
1 @EnableDiscoveryClient    //开启服务发现的功能
2 @SpringBootApplication
3 public class SpringbootEurekaConsumerApplication
4 {
5     public static void main(String[] args) {
6
7     SpringApplication.run(SpringbootEurekaConsumerAp
    plication.class, args);
8     }
```

```

9      //使用RestTemplate进行远程服务调用
10     @Bean
11     @LoadBalanced//允许使用负载均衡机制,默认是轮询
    机制
12     public RestTemplate restTemplate() {
13         return new RestTemplate();
14     }
15 }

```

3. controller

```

1  @RestController
2  public class UserController {
3      //测试地址 http://localhost:8200/buy?
    name=zhangsan
4      @Autowired
5      RestTemplate restTemplate;
6      @GetMapping("buy")
7      public String buy(String name) {
8
9          String s =
    restTemplate.getForObject("http://provider-
    ticket/ticket", String.class);
10         return name+"购买了"+s;
11     }
12 }



```

4.测试

<http://localhost:8200/buy?name=zhangsan>

5.负载均衡

我们修改提供者端口号，生成端口号为8001,8002的提供者jar包。启动之后，我们可以看到在注册中心有两个提供者，访问<http://localhost:8200/buy?name=zhangsan>的时候，使用轮询的负载均衡机制，依次访问每个服务提供者。

名称	修改日期	类型	大小
 provider-8001.jar	2018/7/11 14:32	Executable Jar File	35,038 KB
 provider-8002.jar	2018/7/11 14:37	Executable Jar File	35,038 KB

启动命令

```
1 java -jar provider-8001.jar
```

五、springboot热部署

```
1 <dependency>
2
3     <groupId>org.springframework.boot</groupId>
4         <artifactId>spring-boot-
devtools</artifactId>
5         <optional>true</optional>
6 </dependency>
```

六、springboot监控

1.pom依赖

```
1 <dependency>
2
3 <groupId>org.springframework.boot</groupId>
4 <artifactId>spring-boot-starter-actuator</artifactId>
5 </dependency>
```

2.配置文件

```
1 #开启监控功能
2 management:
3     security:
4         enabled: false
```

3.监控和管理端点

端点名	描述
autoconfig	所有自动配置信息
auditevents	审计事件
beans	所有Bean的信息
configprops	所有配置属性
dump	线程状态信息
env	当前环境信息
health	应用健康状况
info	当前应用信息
metrics	应用的各项指标
mappings	应用@RequestMapping映射路径
shutdown	关闭当前应用（默认关闭）
trace	追踪信息（最新的http请求）

例如：<http://localhost:8080/health>

4.定制端点信息

4.1规则

通过endpoints+端点名+属性名 来设置

修改端点IP :endpoints.beans.id=mybeans

关闭端点：endpoints.beans.enabled=false

开启某个端点

endpoints.enabled=false //关闭所有端点

endpoints.beans.enabled=true //开启beans端点

定制端点访问根路径 management.context-path=/manage

例子：

```
1 #开启监控功能
2 management:
3   security:
4     enabled: false    #开启监控功能
5     context-path: /manager #定制管理端点根访问路径
6     port: 8181    #定制管理端点端口号
7 endpoints:
8   beans:
9     id: mybean    #定义beans端点的id
10    path: /bean    #定义beans端点的访问路径
```

如上配置之后我们访问beans管理端点路径为：

<http://localhost:8181/manager/bean>

5. 自定义别的组件的health信息

5.1 代码示例

```
1 @Component
2 public class MyAppHealthIndicator implements
   HealthIndicator{
3
4     @Override
5     public Health health() {
6         // TODO Auto-generated method stub
7
8         //获取信息进行判断组件的状态
9         // return Health.up().build();//代表组件健康
10        return Health.down().withDetail("msg",
11        "服务down掉了。。。").build();
12    }
13 }
```

5.2 原则

- 1) 名字必须为 xxxHealthIndicator
- 2) 必须实现接口HealthIndicator
- 3) 使用@Component将组件添加在容器中

5.3 测试

测试地址：<http://localhost:8181/manager/health>

测试结果：

```
{"status":"DOWN","myApp":{"status":"DOWN","msg":"服务down掉了。。。"},"diskSpace":  
{"status":"UP","total":392726310912,"free":268464934912,"threshold":10485760},"db":  
{"status":"UP","database":"MySQL","hello":1}}
```