

第七章 Spring Cloud服务注册 与发现

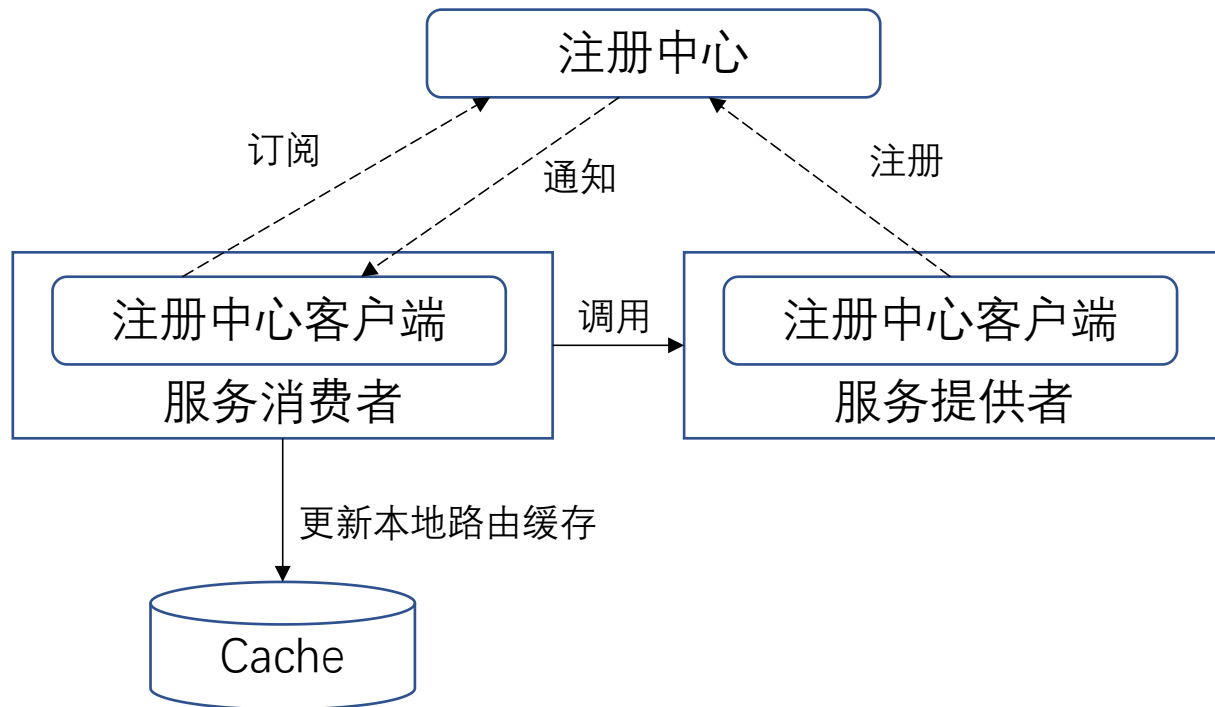
服务注册中心

是路由信息的存储仓库，也是服务提供者和服务消费者进行交互的媒介，充当着服务注册和发现服务器的作用

服务注册中心交互过程

三种角色：

- 服务提供者：注册、发送心跳
- 服务消费者：订阅、发起调用
- 注册中心：发生变化

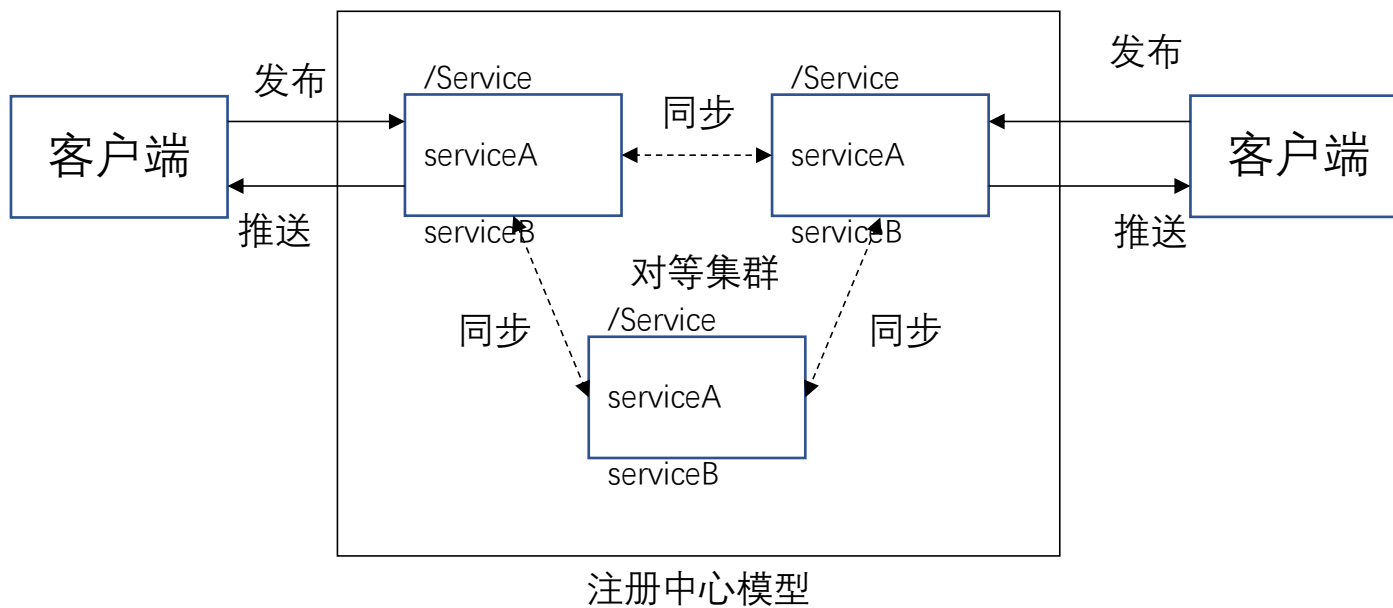


注册中心核心功能

- 发布-订阅功能：
 - 服务提供者发布服务，服务消费者订阅
 - 注册中心服务定义变化时，主动推送变更
 - 需要保持数据一致性
- 保持高可用性：
 - 需要构建对等集群
- 注册中心的客户端程序一般嵌入在服务提供者和消费者的应用程序中

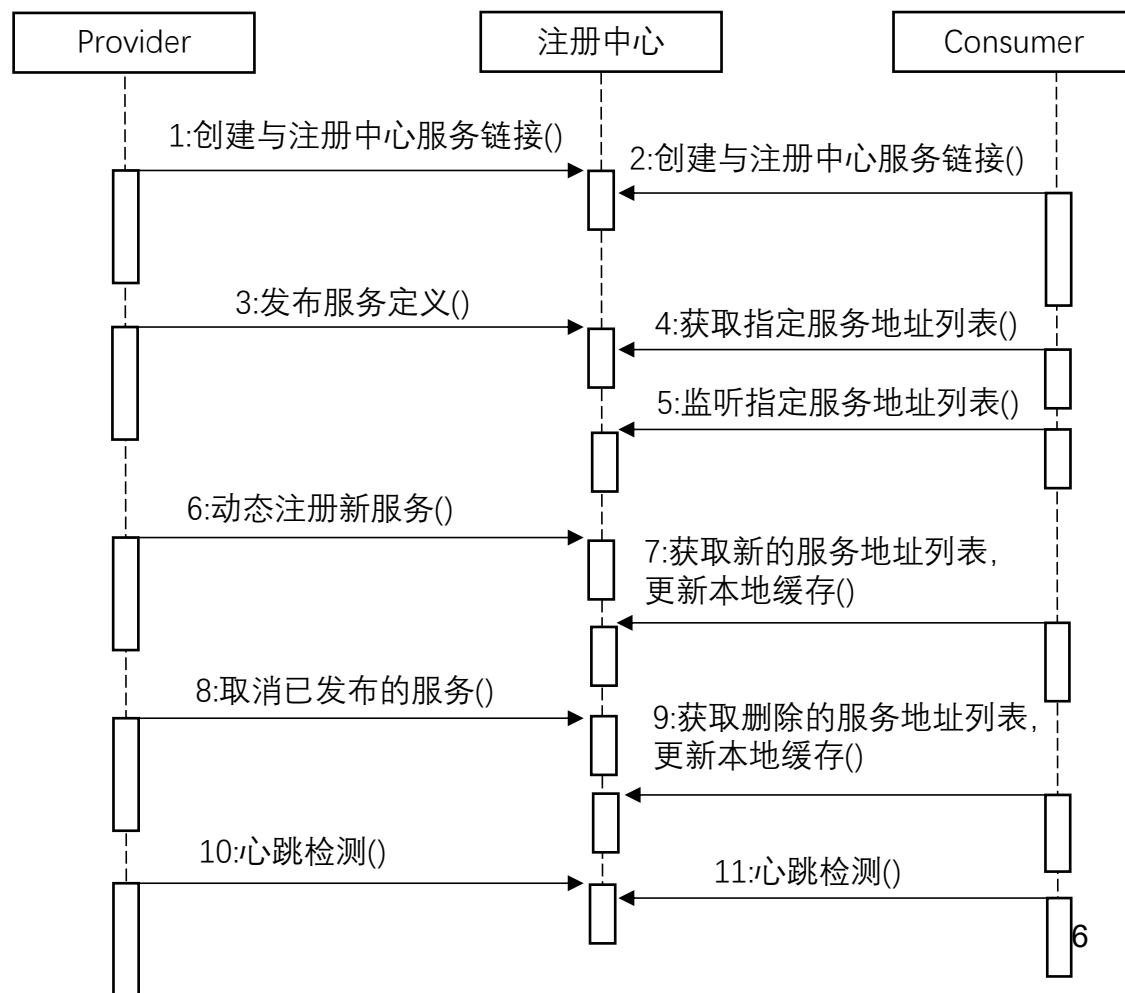
注册中心模型

客户端既可以是服务提供者，也可以是服务消费者



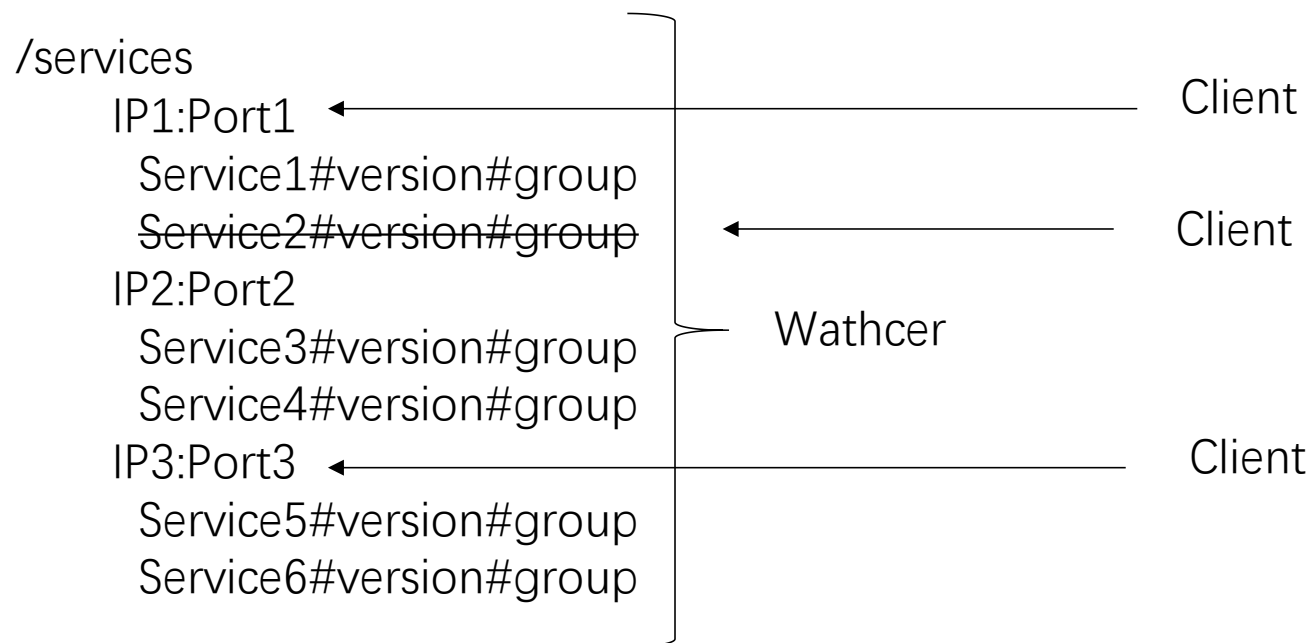
服务发布订阅流程

- 服务的提供者和消费者使用通信机制与注册中心建立连接，通过注册中心的操作接口完成发布和更新



注册中心变更通知机制

- 注册中心的服务监听机制确保消费者能够实时监控服务更新状态



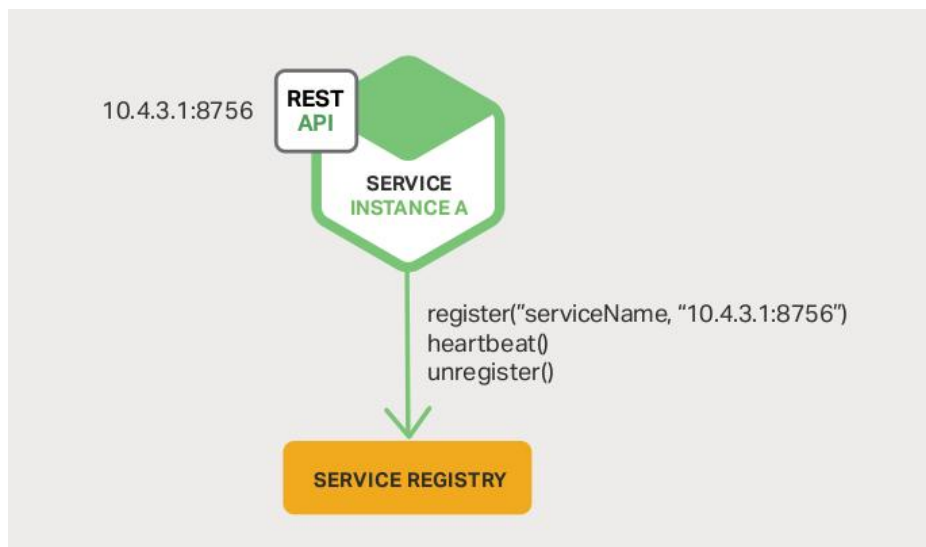
注册中心变更通知机制示意图

服务注册

- 维护一个登记簿，管理系统内的所有服务地址
- 当新服务启动后，它会向登记簿登记自己的地址信息，服务的依赖方可以直接向登记簿要服务地址
- 确保高可用，负责服务状态的维护
- 服务注册的两种形式：
 - 客户端注册
 - 第三方注册

服务注册

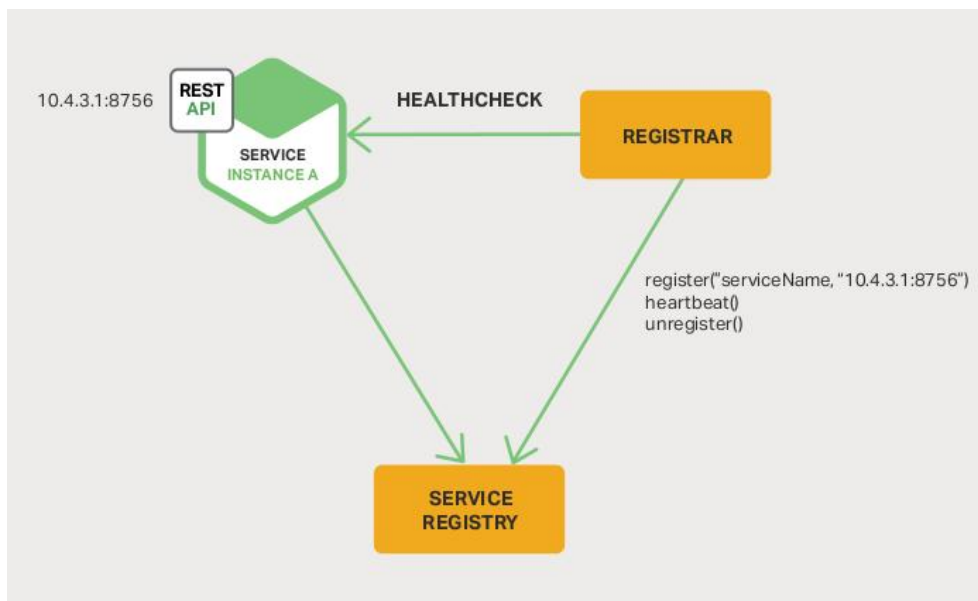
- 客户端注册
 - 服务自身负责注册和注销工作
 - 与注册中心保持心跳
 - 缺点是注册工作和服务耦合在一起，不同语言实现一套注册逻辑



服务注册

- 第三方注册

- 由一个独立的服务Registrar负责注册与注销
- 缺点是Registrar必须是一个高可用的系统，否则注册工作没法进展。



注册中心品牌

- Zookeeper: 一段时间内流行的注册中心, 紧遵CP原则
- Eureka: Netflix开源开发, 紧遵AP原则
- Nacos: 阿里开源开发, 基于 DNS 和基于 RPC 的服务发现
- Consul: HashiCorp 公司推出的开源工具, 用于实现分布式系统的服务发现与配置, 紧遵CP原则

注册中心对比

- 目前各个软件产品特性

	Nacos	Eureka	Consul	zookeeper
数据一致性	AP/CP	AP	CP	CP
健康检查	TCP/HTTP/MySQL/ClientBeat	ClientBeat	TCP/HTTP/grpc/Cmd	Keep Alive
负载均衡策略	权重/metadata/Selector	Ribbon	Fabio	—
雪崩保护	有	有	无	无
容量	100w	5000	百万级	百万级
自动注销实例	√	√	×	√
访问协议	HTTP/DNS	HTTP	HTTP/DNS	TCP
监听支持	√	√	√	√
多数据中心	√	√	√	×
跨注册中心同步	√	×	√	×
Spring Cloud 集成	√	√	√	×
Dubbo集成	√	×	×	√
K8s集成	√	×	√	×

认识Eureka

- Eureka 是 Netflix 出品的用于实现服务注册和发现的工具。Spring Cloud 集成了 Eureka，并提供了开箱即用的支持。其中，Eureka 又可细分为 Eureka Server 和 Eureka Client。
- 在之前的案例中，用户微服务对外提供服务，需要对外暴露自己的地址。而购票微服务（调用者）需要记录服务提供者的地址。将来地址出现变更，还需要及时更新。这在服务较少的时候并不觉得有什么，但是在现在日益复杂的互联网环境，一个项目肯定会拆分出十几，甚至数十个微服务。此时如果还人为管理地址，不仅开发困难，将来测试、发布上线都会非常麻烦。

用户微服务的controller中对外提供服务

```
/** 根据id查询用户 */
@RequestMapping(value = "{id}", method = RequestMethod.GET)
public User findById(@PathVariable Integer id) { return ... }

/** 添加用户 */
@RequestMapping(method = RequestMethod.POST)
public String add(@RequestBody User user){
    userService.add(user);
    return "添加成功";
}
```

购票微服务（调用者）的controller中记录服务提供者的地址

```
*/
User user = restTemplate.getForObject(url: "http://localhost:9001/user/"+id, User.class);
System.out.println(user+"==正在购票...");
return "购票成功";
```



认识Eureka

- Eureka负责管理、记录服务提供者的信息。服务调用者无需自己寻找服务，而是把自己的需求告诉Eureka，然后Eureka会把符合你需求的服务告诉你。
- 同时，服务提供方与Eureka之间通过“心跳”机制进行监控，当某个服务提供方出现问题，Eureka自然会把它从服务列表中剔除。

Eureka实现了**服务的自动注册、发现、状态监控**。

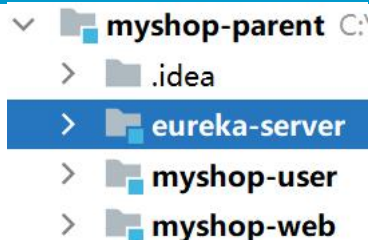
搭建Eureka Server

- 1) 创建一个新模块
- 2) 导入依赖
 - 在父工程导入Spring Cloud的依赖
 - 在子工程导入Eureka Server的依赖
- 3) 编写application.yml配置Eureka
- 4) 编写启动类（添加@EnableEurekaServer注解）

1) 创建新模块eureka-server

2) 导入依赖

- 在父工程导入Spring Cloud的依赖



pom.xml (myshop-parent) x

```
<dependencies>
<!-- 定义SpringCloud版本 -->
<repositories>
  <repository>
    <id>spring-snapshots</id>
    <name>Spring Snapshots</name>
    <url>https://repo.spring.io/snapshot</url>
    <snapshots>
      <enabled>true</enabled>
    </snapshots>
  </repository>
  <repository>
    <id>spring-milestones</id>
    <name>Spring Milestones</name>
    <url>https://repo.spring.io/milestone</url>
    <snapshots>
      <enabled>false</enabled>
    </snapshots>
  </repository>
</repositories>
```

```
<!-- 锁定SpringCloud版本 -->
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-dependencies</artifactId>
      <version>Finchley.M9</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

2) 导入依赖

- 在子工程导入Eureka Server的依赖

```
<!-- Eureka服务器 -->
```

```
<dependencies>
```

```
  <dependency>
```


```
    <groupId>org.springframework.cloud</groupId>
```

```
    <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
```

```
  </dependency>
```


```
</dependencies>
```

▼  eureka-server

>  生命周期

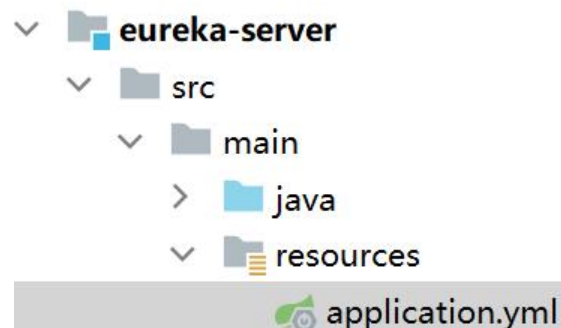
>  插件

▼  依赖项

>  org.springframework.cloud:spring-cloud-starter-netflix-eureka-server:2.0.0.M8

3) 编写application.yml配置Eureka

```
server:
  port: 8888
spring:
  application:
    name: eureka-server
    # 单机版配置
eureka:
  client:
    fetch-registry: false # 是否需要从Eureka获取注册信息
    register-with-eureka: false # 是否需要把该服务注册到Eureka
    service-url: # 暴露Eureka注册地址
      defaultZone: http://localhost:${server.port}/eureka
  server:
    # 修改扫描失效服务间隔时间（默认是60s，单位是毫秒）
    eviction-interval-timer-in-ms: 5000
    # 取消自我保护机制（默认true）
    enable-self-preservation: false
```



4) 编写启动类（添加@EnableEurekaServer注解）

```
package com.example.eureka;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

/**
 * Eureka 微服务
 */
@SpringBootApplication
@EnableEurekaServer // 开启Eureka服务端自动配置
public class EurekaApplication {

    public static void main(String[] args) {

        SpringApplication.run(EurekaApplication.class, args);
    }
}
```

启动Eureka

The screenshot displays an IDE environment with a project named 'myshop-parent' located at 'C:\Users\86134\project\'. The project structure includes a 'eureka-server' module with a 'src/main/java/com.example.eureka' package. The 'EurekaApplication.java' file is open, showing the following code:

```
7  /**
8   * Eureka 微服务
9   */
10 @SpringBootApplication
11 @EnableEurekaServer // 开启Eureka服务端自动配置
12 public class EurekaApplication {
13
14     public static void main(String[] args) {
15
16         SpringApplication.run(EurekaApplication.class, args);
17     }
18 }
```

Below the code editor, the '服务' (Services) tab is active, showing the 'Spring Boot' service. The '正在运行' (Running) state is indicated, and the 'EurekaApplication' is listed as '未启动' (Not Started). The '控制台' (Console) tab shows the following output:

```
r.EurekaServerBootstrap : Eureka data center value eureka.datacenter is not set, defaulting
r.EurekaServerBootstrap : Eureka environment value eureka.environment is not set, defaulting
r.EurekaServerBootstrap : isAws returned false
r.EurekaServerBootstrap : Initialized server context
reInstanceRegistryImpl : Got 1 instances from neighboring DS node
reInstanceRegistryImpl : Renew threshold is: 1
reInstanceRegistryImpl : Changing status to UP
r.InitializerConfiguration : Started Eureka Server
d.tomcat.TomcatWebServer : Tomcat started on port(s): 8888 (http) with context path ''
kaAutoServiceRegistration : Updating port to 8888
eka.EurekaApplication : Started EurekaApplication in 6.576 seconds (JVM running for 7.35s)
```

浏览器访问Eureka



System Status

Environment	test
Data center	default

Current time	2023-10-08T23:29:08 +0800
Uptime	00:03
Lease expiration enabled	true
Renews threshold	1
Renews (last min)	0

THE SELF PRESERVATION MODE IS TURNED OFF.THIS MAY NOT PROTECT INSTANCE EXPIRY IN CASE OF NETWORK/OTHER PROBLEMS.

DS Replicas

localhost

Instances currently registered with Eureka

DS Replicas

localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
No instances available			

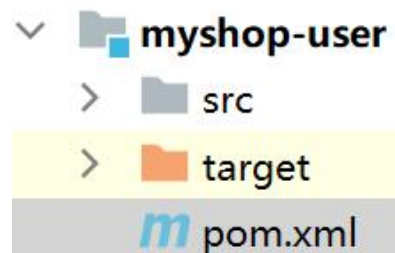
General Info

Name	Value
total-avail-memory	67mb
environment	test
num-of-cpus	8
current-memory-usage	42mb (62%)
server-uptime	00:03
registered-replicas	http://localhost:8888/eureka/
unavailable-replicas	http://localhost:8888/eureka/,

搭建Eureka Client

- 在购票微服务（服务调用者）和用户微服务（服务提供者）都作为Eureka Client注册到Eureka Server
1. 导入Eureka Client的依赖
 2. 在application.yml配置连接Eureka Server
 3. 在启动类添加@EnableEurekaClient注解

1. 导入Eureka Client的依赖



`<!-- 导入Eureka 客户端依赖-->`

`<dependency>`

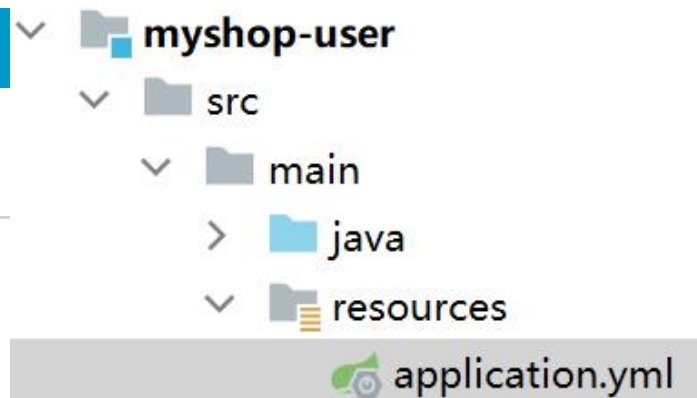
`<groupId>org.springframework.cloud</groupId>`

`<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>`

`</dependency>`

2. 在application.yml配置连接Eureka Server

```
server:
  port: 9001
spring: # 服务名称
  application:
    name: myshop-user
  datasource:
    url: jdbc:mysql://192.168.221.133:3306/springcloud?characterEncoding=UTF8&useSSL=false
    driver-class-name: com.mysql.jdbc.Driver
    username: root
    password: 123456
  jpa:
    show-sql: true # 是否打印sql语句
    generate-ddl: true # 是否自动建表
    database: mysql # jpa连接的数据库类型
eureka:
  client:
    register-with-eureka: true # 作为客户端需要注册到Eureka
    fetch-registry: true # 作为客户端需要从Eureka获取注册信息
    service-url: # 客户端注册地址
      defaultZone: http://localhost:8888/eureka
  instance:
    # 优先使用该服务的IP地址注册到Eureka, 在生产环境建议改为true
    prefer-ip-address: true
```



3. 在启动类添加@EnableEurekaClient注解

```
/**
 * 用户微服务
 */
@SpringBootApplication
@EnableEurekaClient // 开启Eureka客户端自动配置
public class UserApplication {

    public static void main(String[] args) {

        SpringApplication.run(UserApplication.class, args);
    }
}
```

启动用户微服务，并用浏览器访问Eureka

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
MYSHOP-USER	n/a (1)	(1)	UP (1) - TF-laptop:myshop-user:9001
MYSHOP-WEB	n/a (1)	(1)	UP (1) - TF-laptop:myshop-web:9002

```
spring: # 服务名称，暂时没有几
  application:
    name: myshop-user
```

使用Eureka实现服务调用

@Autowired

```
private RestTemplate restTemplate;
```

@Autowired

```
private DiscoveryClient discoveryClient;
```

```
/**
```

```
 * 购票方法(使用Eureka版本)
```

```
 * @return
```

```
 */
```

```
@RequestMapping(value = "/order", method = RequestMethod.GET)
```

```
public String order(){
```

```
    // 模拟当前用户
```

```
    Integer id = 2;
```

```
    // 到Eureka发现用户微服务
```

```
    // 返回值为什么是List集合呢? 获取到所有同名的微服务
```

```
    List<ServiceInstance> instances = discoveryClient.getInstances( serviceId: "myshop-user"); // 参
```

```
    // 没有使用负载均衡, 只能获取第一个服务
```

```
    ServiceInstance serviceInstance = instances.get(0);
```

```
    User user = restTemplate.getForObject( url: "http://" + serviceInstance.getHost() + ":"
```

```
        + serviceInstance.getPort() + "/user/" + id, User.class);
```

```
    System.out.println(user + "==正在购票...");
```

```
    return "购票成功";
```

2 3 1 ^

启动应用，查看结果

← ↻ 🏠 ⓘ localhost:9002/web/order

购票成功

购票微服务后台结果

User{id=2, username='txc', password='123456', sex='男', money=6000.0}==正在购票...

2023-10-12 11:32:07.824 INFO 10512 --- [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver

用户微服务后台结果

Hibernate: insert into tf_user (money, password, sex, username) values (?, ?, ?, ?)

Hibernate: select user0_.id as id1_0_0_, user0_.money as money2_0_0_, user0_.password as password3_0_0_

2023-10-12 11:32:20.139 INFO 18620 --- [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver : R

搭建高可用的Eureka Server

```
server:
  port: 8888
spring:
  application:
    name: eureka-server
    # 集群版配置
eureka:
  client:
    fetch-registry: true # 是否需要从Eureka获取注册信息
    register-with-eureka: true # 是否需要把该服务注册到Eureka
    service-url: # 暴露Eureka注册地址
    defaultZone: http://localhost:9999/eureka
```

```
server:
  port: 9999
spring:
  application:
    name: eureka-server
    # 集群版配置
eureka:
  client:
    fetch-registry: true # 是否需要从Eureka获取注册信息
    register-with-eureka: true # 是否需要把该服务注册到Eureka
    service-url: # 暴露Eureka注册地址
    defaultZone: http://localhost:8888/eureka
```


搭建高可用的Eureka Server

← ↻ 🏠 ⓘ localhost:8888 A 🌟

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
EUREKA-SERVER	n/a (2)	(2)	UP (2) - TF-laptop:eureka-server:8888 , TF-laptop:eureka-server:9999

← ↻ 🏠 ⓘ localhost:9999 A 🌟

Application	AMIs	Availability Zones	Status
EUREKA-SERVER	n/a (2)	(2)	UP (2) - TF-laptop:eureka-server:8888 , TF-laptop:eureka-server:9999

服务端注册到一台Eureka服务器

```
server:
  port: 9002
spring:
  # 服务名称
  application:
    name: myshop-web
eureka:
  client:
    register-with-eureka: true # 作为客户端需要注册到Eureka
    fetch-registry: true # 作为客户端需要从Eureka获取注册信息
    service-url: # 客户端注册地址
      defaultZone: http://localhost:8888/eureka
  instance:
    # 优先使用该服务的IP地址注册到Eureka，在生产环境建议改为true
    prefer-ip-address: true
```

两台Eureka服务器上均能查看到用户微服务和购票微服务

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
EUREKA-SERVER	n/a (2)	(2)	UP (2) - TF-laptop:eureka-server:8888 , TF-laptop:eureka-server:9999
MYSHOP-USER	n/a (1)	(1)	UP (1) - TF-laptop:myshop-user:9001
MYSHOP-WEB	n/a (1)	(1)	UP (1) - TF-laptop:myshop-web:9002

测试停止8888的Eureka服务器，购票服务是否能正常调用用户微服务

Instances currently registered with Eureka			
Application	AMIs	Availability Zones	Status
EUREKA-SERVER	n/a (1)	(1)	UP (1) - TF-laptop:eureka-server:9999
MYSHOP-USER	n/a (1)	(1)	UP (1) - TF-laptop:myshop-user:9001
MYSHOP-WEB	n/a (1)	(1)	UP (1) - TF-laptop:myshop-web:9002

购票微服务成功调用用户微服务



购票成功

服务提供者

- 服务提供者要向EurekaServer注册服务，并且完成服务续约等工作。
1. 服务注册: `eureka.client.register-with-eureka=true`
 2. 服务续约: 在注册服务完成以后，服务提供者会定时向EurekaServer发起Rest请求，告诉EurekaServer：“我还活着”。这个我们称为服务的续约(renew)心跳，续约参数：

`eureka:`

`instance:`

`lease-expiration-duration-in-seconds: 90`

`lease-renewal-interval-in-seconds: 30`

`lease-renewal-interval-in-seconds`: 服务续约(renew)的间隔，默认为30秒

`lease-expiration-duration-in-seconds`: 服务失效时间，默认值90秒

服务调用者

- 获取服务注册信息: `eureka.client.fetch-registry=true`
- 默认每隔30秒重新获取并更新注册信息, 修改参数

```
eureka:  
  client:  
    registry-fetch-interval-seconds: 5
```

Eureka Server失效剔除与自我保护

- 失效剔除

默认情况下，Eureka Server每隔60秒对失效的服务（超过90秒未续约的服务）进行剔除。以下参数可以修改剔除时间

`eureka.server.eviction-interval-timer-in-ms`

- 自我保护

- 有时关停服务时会看到Eureka Server出现一行红色字

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

这是Eureka的自我保护机制。

Eureka会统计最近15分钟心跳失败的服务实例的比例是否超过了85%。在生产环境下，因为网络延迟等原因，心跳失败实例的比例很有可能超标，但是此时就把服务剔除并不妥当，因为服务可能没有宕机。Eureka就会把当前实例的注册信息保护起来，不予剔除。生产环境下这很有效，保证了大多数服务依然可用。

server:

port: 8888

spring:

application:

name: eureka-server

集群版配置

eureka:

client:

fetch-registry: true *# 是否需要从Eureka获取注册信息*

register-with-eureka: true *# 是否需要把该服务注册到Eureka*

service-url: *# 暴露Eureka注册地址*

defaultZone: http://localhost:9999/eureka

server:

修改扫描失效服务间隔时间（默认是60s，单位是毫秒）

eviction-interval-timer-in-ms: 5000

取消自我保护机制（默认true）

enable-self-preservation: false