

# 一 单体项目的问题

## 1 什么时单体项目

传统的项目系统，都是单体项目。

互联网领域里单体项目表示一个所有功能集中的项目。

例如:order-user 系统，集中实现了订单功能、用户功能，如果做功能扩展，像购物车，商品，秒杀等等功能都要集中在一个运行程序中。

简言之：功能集中到一起的项--->>单体项目

## 2 单体项目的问题

### 2.A 功能强耦合

开发过程中，企业、架构师，一般会把不同的功能交给不同的人员开发，每个开发者只需要了解自己领域里的业务知识。而单体项目难以满足这样需求。

例如:订单支付逻辑，完成积分新增逻辑，造成一个开发者，团队接触过多的业务，导致系统，企业的管理复杂。

### 2.B 并发集中

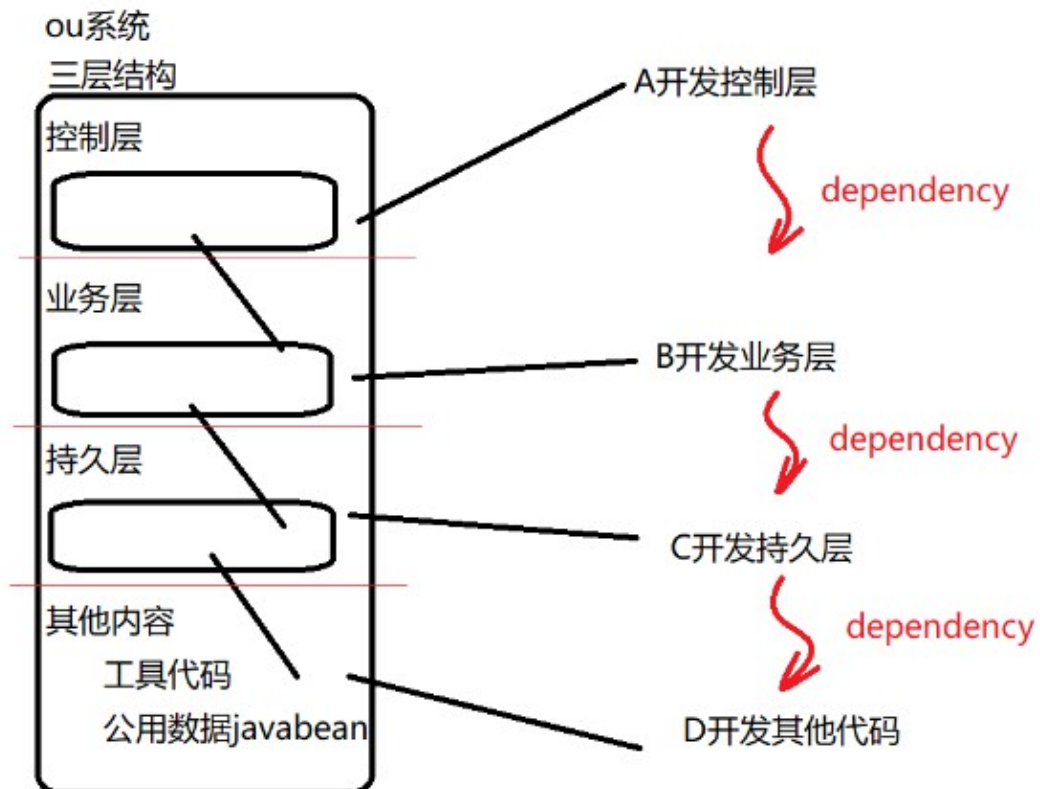
所有功能最终都运行在同一个系统中，其中某个功能在高并发情况下导致系统的资源占满，其他功能都不能使用。

需要从框架的角度去解决这个问题---微服务框架

# 二 项目拆分(单体项目的初步解决)

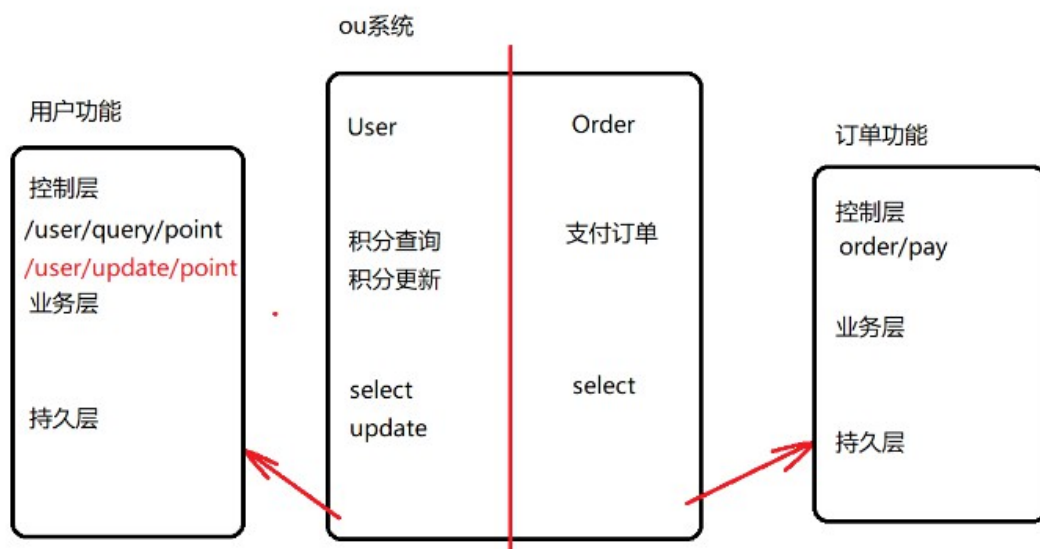
## 1 横向拆分

比如利用 maven 这种管理工具，将一个大型项目分模块交给不同人员开发，最终还是运行一个应用的过程.所以横向拆分不能解决单体项目问题。



## 2 纵向拆分

是按照系统功能进行拆分，每个拆分出来的内容，都可以独立运行，接收请求。



### 2.A 解决了功能强耦合，如何解决系统之间的调用问题

需要增加接口，功能之间相互调用

例如：订单支付完成，调用积分接口增加积分

```
public void payOrder(String orderId) {
    //利用 orderId 将数据库中数据查询 order 对象
    Order order=orderMapper.selectOrderById(orderId);
    //模拟支付：订单 a 成功支付 5000 元
```

```
if (order == null) { // 没有利用 orderId 从数据库查询出来数据
    System.out.println("您支付的订单为空, id 是否正确?");
    return;
}
// order 不为 null 执行正常的支付逻辑
System.out.println("订单" + orderId + ", 成功支付:" + order.getOrderMoney() + "元");
// 积分更新功能, 从订单发送一个请求访问 /user/update/point?money=500
String url = "http://localhost:9001/user/update/point?";
money = order.getOrderMoney();
RestTemplate template = new RestTemplate();
Integer success = template.getForObject(url, Integer.class);
if (success != 1) {
    // 更新积分失败
    // 写日志
    throw new RuntimeException("支付成功了, 但是更新积分失败了");
}
```

## 2.B 负载均衡调用

微服务架构

# 三 分布式事务

# 四 微服务思想

## 1 纵向拆分结构的潜在问题

### 1.A nginx 静态文件配置过于复杂

随着业务功能变得越来越复杂, 拆分的系统越来越多, 对应 nginx upstream 静态配置越来越多. 不仅多, 维护麻烦.

## 1.B 故障调用没有处理机制

在负载均衡过程中，在拆分的系统中，存在系统之间的内部调用，随着业务功能越来越复杂，系统之间调用关系越多，一旦出现某一个环节由于节点故障网络波动导致无法访问，超时，长此以往，一定会导致整个集群的瘫痪。

目前拆分结构的问题不止这二个问题，还有其他的各种各样的潜在的问题，所以最终解决单体项目的问题不仅要通过项目的纵向拆分，还要引入架构技术管理这些拆分之后的系统→微服务架构。

## 2 微服务概念

### 2.A 微

微不是描述的功能简单，系统单一，而是经过纵向拆分之后，系统相对于原有项目变的微小了。

### 2.B 服务

拆分出来的独立运行的系统，每一个具体的功能都需要被别人调用，叫做服务的调用，所以每一个具体的功能都是一个服务。

## 3 微服务解决方案

- springboot + dubbo  
springboot 构建项目  
dubbo 形成微服务管理框架
- ★ springboot+springcloud  
springboot 构建项目  
springcloud 形成微服务管理框架
- springboot +spring cloud alibaba  
springcloud 整合 alibaba 的微服务框架+springboot 构建

## 4 SpringCloud 微服务框架

定义:由 spring 团队整合市面成熟技术行程一套解决微服务问题的框架技术集合。  
包括:

eureka  
ribbon  
zuul  
springcloud config  
feign  
hystrix

## 五 SpringCloud 框架集

### 六 1.1eureka 服务治理

### 七 1.2ribbon 负载均衡

### 八 1.3zuul 网关

### 九 1.4spring cloud config

### 十 1.5feign 负载均衡

### 十一 1.6hystrix 熔断器组件

## 十二 Eureka 服务治理组件

### 1 简介

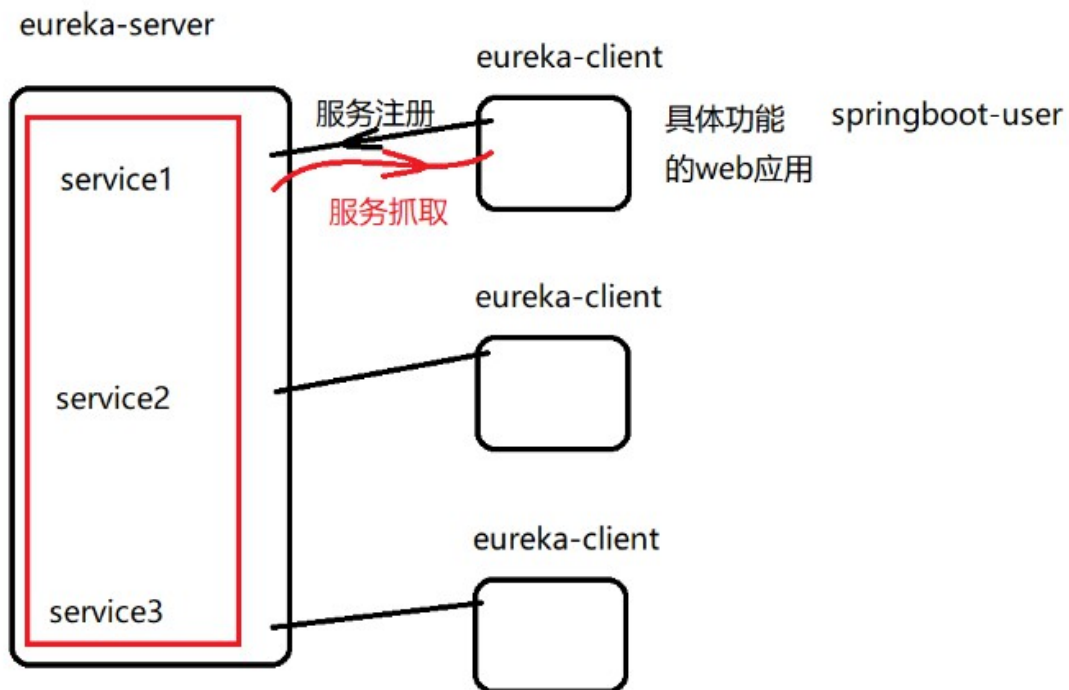
netflix 公司(视频网站)的服务注册发现机制的技术，被 springcloud 拿来实现服务治理，springcloud 中**唯一**可以使用的一个**服务治理组件**.springcloud **核心组件**，没有这个组件，无法实现其他的微服务功能.

## 2 功能

### 2.A eureka 组件两个角色

- 搭建的功能可以使用 eureka 不同角色实现服务治理的不同功能
  - **客户端**:eureka-client, 就是配合拆分的项目, 实现注册与发现.
  - **服务端**:eureka-server, 管理所有客户端的注册信息.

## 3 eureka 治理组件的结构



## 十三 搭建注册中心 eureka-server

### 1 创建一个父级工程

#### 1.A 继承 springboot-parent

```
<!--继承 spring boot-->
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
```

```
<version>1.5.9.RELEASE</version>
</parent>
```

## 1.B 定义为父工程类型

```
<!--定义为父工程-->
<packaging>pom</packaging>
```

## 1.C 定义子工程 spring cloud 组件版本

```
<!--定义所有子工程的 spring cloud 组件版本-->
<!--子工程可不写 version 默认使用该版本-->
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-dependencies</artifactId>
      <version>Edgware.RELEASE</version>
      <!--导入父级工程-->
      <scope>import</scope>
      <!--父级工程类型-->
      <type>pom</type>
    </dependency>
  </dependencies>
</dependencyManagement>
```

## 1.D pom.xml 完整配置

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>cn.shu</groupId>
  <artifactId>eureka-demo-parent</artifactId>
  <version>1.0-SNAPSHOT</version>
  <!--定义为父工程-->
  <packaging>pom</packaging>

  <!--继承 spring boot-->
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.5.9.RELEASE</version>
  </parent>
  <!--定义所有子工程的 spring cloud 组件版本-->
  <!--子工程可不写 version 默认使用该版本-->
  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>org.springframework.cloud</groupId>
```

```
        <artifactId>spring-cloud-dependencies</artifactId>
        <version>Edgware.RELEASE</version>
        <!--导入父级工程-->
        <scope>import</scope>
        <!--父级工程类型-->
        <type>pom</type>
    </dependency>
</dependencies>
</dependencyManagement>
</project>
```

## 2 创建注册中心子工程

### 2.A 继承父工程

```
<parent>
    <artifactId>eureka-demo-parent</artifactId>
    <groupId>cn.shu</groupId>
    <version>1.0-SNAPSHOT</version>
</parent>
```

### 2.B 添加 eureka-server 依赖

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

### 2.C pom.xml 完整配置

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

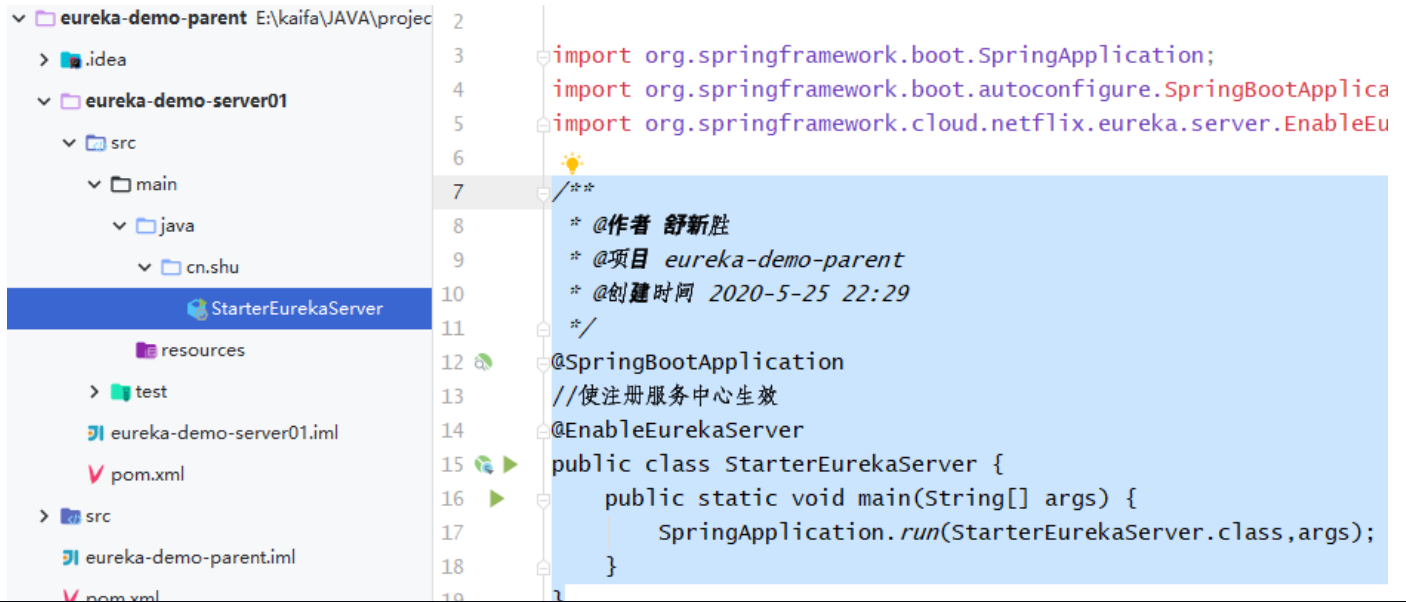
    <!--继承父工程-->
    <parent>
        <artifactId>eureka-demo-parent</artifactId>
        <groupId>cn.shu</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>eureka-demo-server01</artifactId>
    <!--导入 eureka server 依赖-->
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-eureka-server</artifactId>
        </dependency>
```



```
</dependencies>
</project>
```

## 2.D 创建启动类



```
/**
 * @作者 舒新胜
 * @项目 eureka-demo-parent
 * @创建时间 2020-5-25 22:29
 */
@SpringBootApplication
//使注册服务中心生效
@EnableEurekaServer
public class StarterEurekaServer {
    public static void main(String[] args) {
        SpringApplication.run(StarterEurekaServer.class, args);
    }
}
```

## 2.E 创建注册中心配置文件(application.yml)

### 搭建 springboot 的 eureka-server 的过程

一旦依赖了 eureka-server springboot 自动实现整个配置过程，启动时与运行 eureka-server 进程

搭建的 eureka-server 名字叫做注册中心。默认情况下把当前工程运行的进程既看成服务端，也堪称是客户端(默认自己在自己的注册中心注册自己的服务信息)

**服务名称：**高可用时，多个 eureka-server 公用的一个服务名称

**ip 优先：**如果不设置 ip 优先，所有治理通信使用域名，本机 localhost 无法在服务器集群中使用，true 使用网卡 ip 地址

**关闭抓取**：由于注册中心自己把自己看成 eurekaclient，如果不关闭注册抓取，会在启动时报错，找不到服务端。

**注册中心地址**：高可用结构中，多个注册中心相互注册时配置的值。

#定义端口

```
server:
  port: 8001
spring:
  application:
```

#定义

名称

```
name: eureka-demo-server01
eureka:
  instance:
```

#开启 ip 优先, 本地运行开

开都一样, 但是一旦发布到服务器, 必须开启外界访问的 ip

#否则可能会使用默认的域名, 比如 localhost

```
prefer-ip-address: true
client:
```

#eureka-server 也会向自己发起注册申请 先关闭自我注册, 并关闭抓取

#因为当 server 启动时, 自己本身未启动成功, 再自己注册到自己会抛异常

```
register-with-eureka: false
fetch-registry: false
```

#当前注册中心作为客户端 可以访问的 其它注册中心地址

#这里写的自己 自己注册到自己?

```
service-url:
  defaultZone: http://127.0.0.1:8001/eureka
```

## 十四 搭建客户端 eureka-client

客户端功能需要携带信息到注册中心注册数据, 等待被别人(外界客户端, 外界浏览器, 内部其他服务)调用发现.所以 eureka-client 组件所在的工程都是微服务--服务提供者.

### 1 创建子工程

### 2 配置 pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```

        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <!--继承父工程-->
    <parent>
        <artifactId>eureka-demo-parent</artifactId>
        <groupId>cn.shu</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>eureka-demo-client01</artifactId>
    <!--导入 eureka client 依赖-->
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-eureka</artifactId>
        </dependency>
    </dependencies>

</project>

```

### 3 创建配置文件

```

#定义端口
server:
    port: 8101
spring:
    application:
        #定义客户端名称
        name: eureka-demo-client01
eureka:
    instance:
        #开启 ip 优先, 本地运行开
开都一样, 但是一旦发布到服务器, 必须开启外界访问的 ip
        #否则可能会使用默认的域名, 比如 localhost
        prefer-ip-address: true

        #注册中心地址 客户端回去注册
    service-url:
        defaultZone: http://127.0.0.1:8001/eureka

```


### 4 创建启动类

```

/**
 * @作者 舒新胜
 * @项目 eureka-demo-parent
 * @创建时间 2020-5-25 22:29
 */
@SpringBootApplication
// 客户端生效 1

```

```
@EnableEurekaClient
public class StarterEurekaClient {
    public static void main(String[] args) {
        SpringApplication.run(StarterEurekaClient.class, args);
    }
}
```


HOME
LAST 1000 SINCE STARTUP

### System Status

|             |         |                          |                           |
|-------------|---------|--------------------------|---------------------------|
| Environment | test    | Current time             | 2020-05-25T23:20:22 +0800 |
| Data center | default | Uptime                   | 00:15                     |
|             |         | Lease expiration enabled | true                      |
|             |         | Renews threshold         | 5                         |
|             |         | Renews (last min)        | 8                         |

### DS Replicas

127.0.0.1

### Instances currently registered with Eureka

| Application          | AMIs    | Availability Zones | Status                                       |
|----------------------|---------|--------------------|--|
| EUREKA-DEMO-CLIENT01 | n/a (1) | (1)                | UP (1) - localhost:eureka-demo-client01:8101 |
| EUREKA-DEMO-SERVER01 | n/a (1) | (1)                | UP (1) - localhost:eureka-demo-server01:8001 |

### General Info

| Name                 | Value                         |
|----------------------|-------------------------------|
| total-avail-memory   | 523mb                         |
| environment          | test                          |
| num-of-cpus          | 8                             |
| current-memory-usage | 185mb (35%)                   |
| server-uptime        | 00:15                         |
| registered-replicas  | http://127.0.0.1:8001/eureka/ |

## 十五 eureka 治理组件的原理

### 1 eureka 客户端:

通过配置注册中心地址 <http://127.0.0.1:8001/eureka>，对客户端提供了一个可以访问注册中心的接口地址，客户端总会通过 http 请求访问这个地址实现各种功能的调用。

**client:**

#注册中心地址 客户端回去注册

**service-url:**

**defaultZone:** <http://127.0.0.1:8001/eureka>

## 1.A 注册

客户端会在启动时，携带自身信息(服务名称，ip 地址，端口号，等等)，在注册中心维护一份注册信息，注册的开启与关闭就是通过配置 `eureka.client.register-with-eureka` 实现的

## 1.B 抓取

客户端会访问注册中心服务端，实现每隔 30 秒中抓取一次注册中心的注册信息  
抓取注册中心的双层 map 信息

## 1.C 心跳

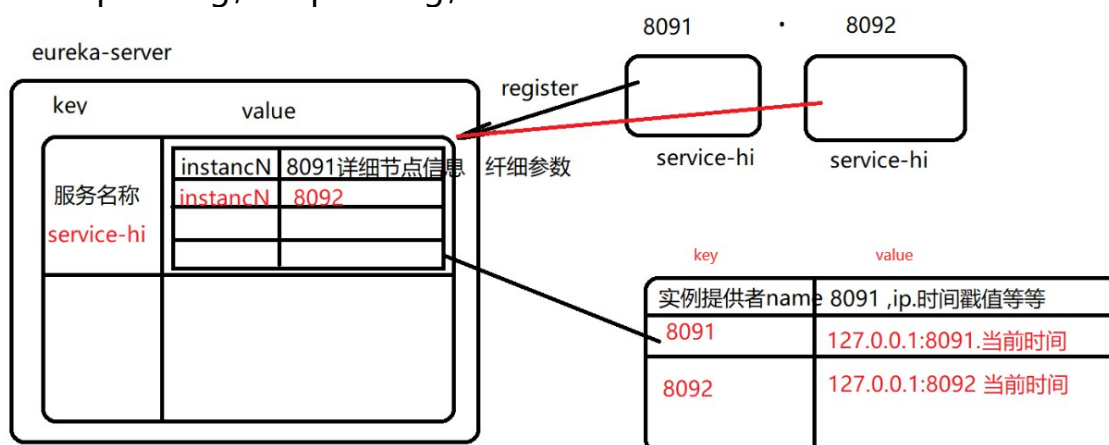
客户端为了保证让服务端总是记录自己的信息而不做超时剔除，会每隔 30 秒钟发送一次心跳

# 2 eureka 服务端

## 2.A 管理注册:

- 根据请求过来的客户端携带的信息，在内存中生成一份当前微服务集群里所有微服务服务信息数据(双层 map)，而客户端抓取的注册信息就是这个双层 map 对象.双层 map 代码结构

`map<String, map<String, Instance>>`



**第一层 Map key (服务名称) -> value(第二层 map)**

整个集群中，同一个服务名称只能在一个注册中心中存在一个 key

**第二层 Map key (实例名称.默认结构 域名:服务名称:端口号) -> value (该实例的详细信息，包含 ip 端口 时间戳等)**

## 2.B 判断超时监听

每隔 60 秒，判断一次注册信息中所有实例的最后时间戳（心跳一次刷新）是否超时（距离

现在时间超过 90 秒)，如果客户端没有发送心跳检测，则会超过时间点，注册中心将会把实例从 map 对象中 key-value 剔除

## 2.C 保护机制

为了防止由于网络波动导致的大面积的服务提供者未能有效的发送心跳续约，从而按照超时逻辑被剔除，而被提出的服务并没有宕机故障。会造成微服务可用，甚至瘫痪。所以出现了保护机制。当 eureka 服务端，一次性判断大量服务宕机故障（15%以上），会认为是网络波动导致没有接收到客户端的心跳信息，而不是真正宕机故障。这时开启保护机制，保存所有微服务不剔除。阈值（15%）测试环境中非常容易达到。达到阈值后会有如下提示：

```
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.
```

观察超时剔除可以关闭自我保护机制：

在注册中心中，关闭配置属性

```
eureka.server.enable-self-preservation=false
```

# 十六 eureka 治理中专业名词

## 1 注册中心：

不要把他单独看成服务端，也具备客户端角色

## 2 服务提供者（服务实例）：

实际就是客户端所在的工程

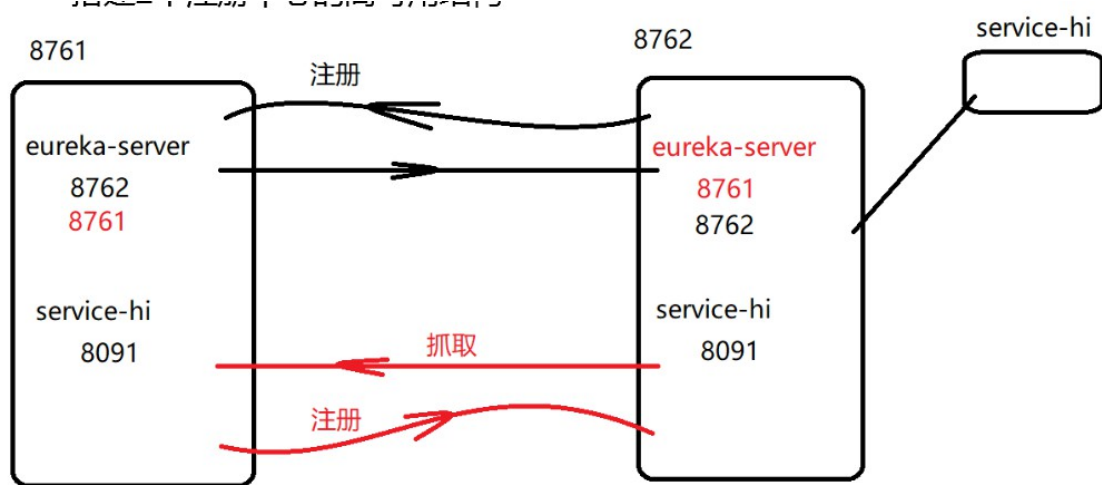
## 十七 eureka 的高可用结构

**客户端集群：**一旦将多个客户端都安服务提供者启动后注册在注册中心，为同一个服务提供调用功能，这个集群就是高可用。

**注册中心高可用：**目前结构只有一个注册中心。没法形成高可用。一定要形成注册中心的集群。

**原理：**将多个（2个）注册中心，**相互作为客户端进行注册**，相互抓取注册信息同步数据。

搭建 2 个注册中心的高可用结构



Server2 启动时会到指定的地址(Server1)去注册

```
client:
  service-url:
    defaultZone: http://127.0.0.1:8001/eureka
```

Server1 启动时会到指定的地址(Server2)去注册

```
client:
  service-url:
    defaultZone: http://127.0.0.1:8002/eureka
```

这样搭建的高可用注册中心，一定会在启动时报个错误

因为总有一个 server 先启动，先启动会注册失败，另一个 Server 未启动

cannot excute request on any know server