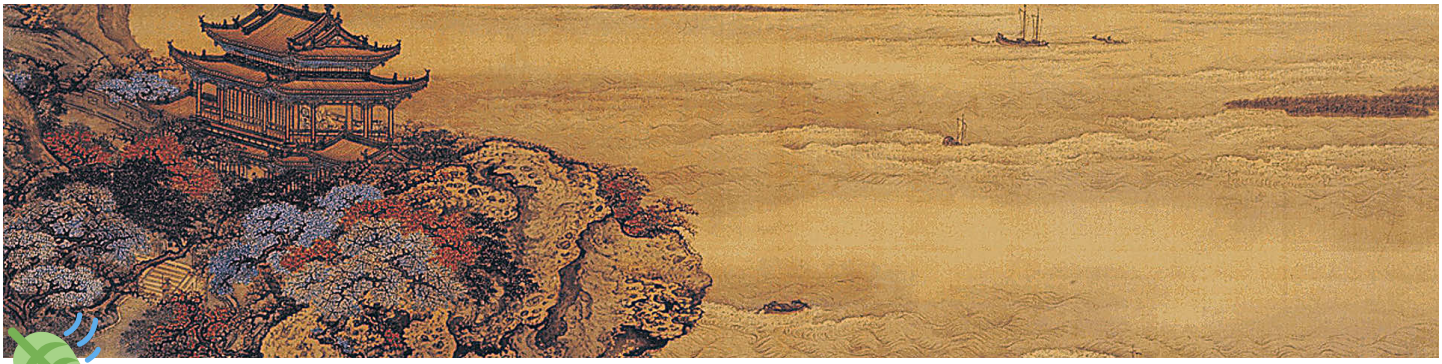
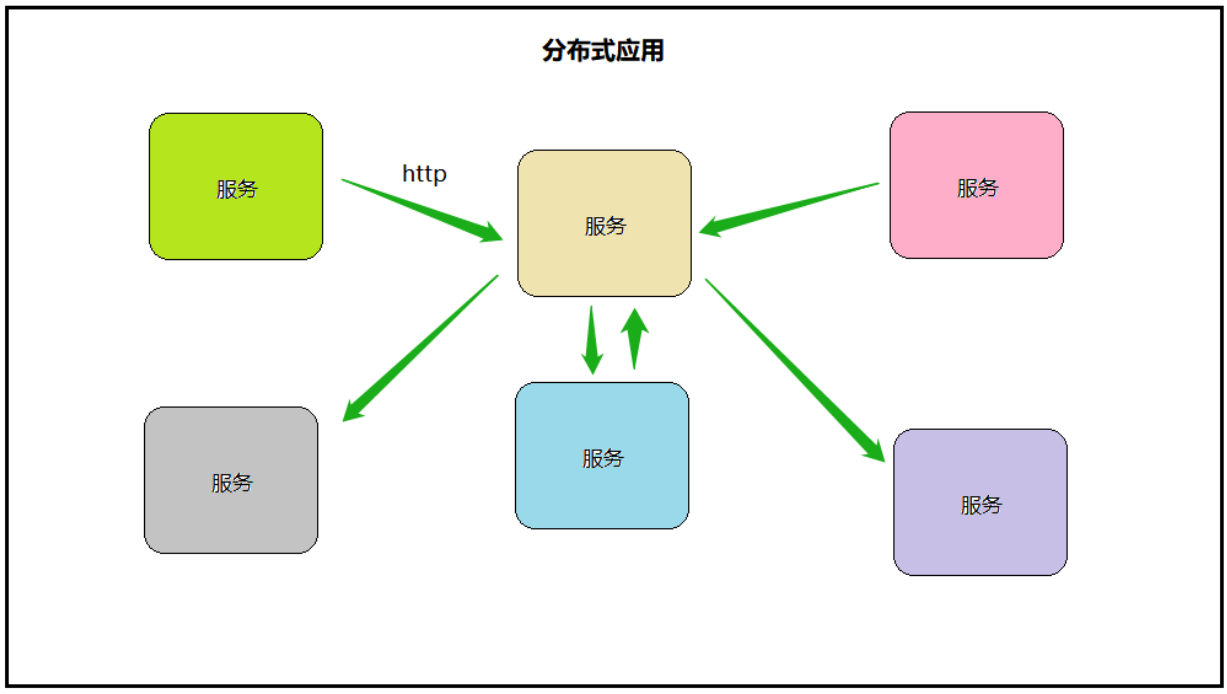


注册中心| Eureka、Consul

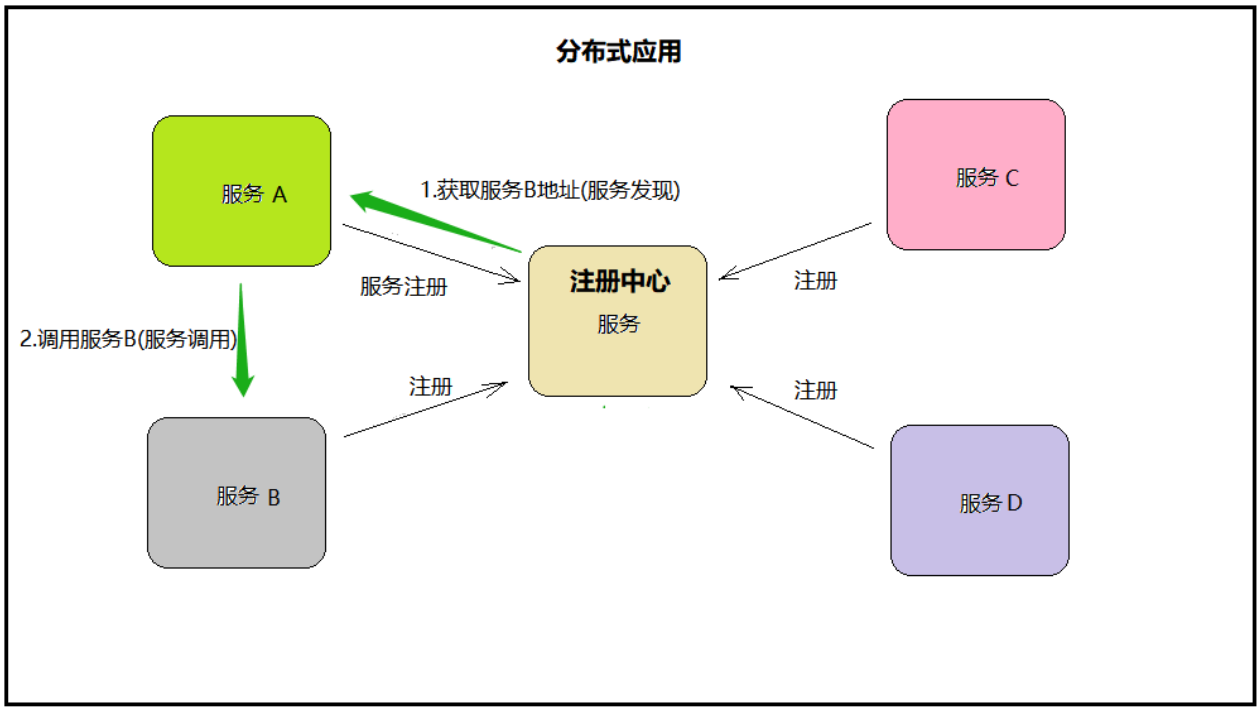
在SpringCloud服务集群中，存在多个功能独立的微型服务，这些服务之间通过相互调用的方式进行通信。

在服务调用过程中，调用服务的一方称为消费者，被调用的一方称为生产者，它们也都属于服务的一员。

传统的项目中我们可以使用Http工具来进行服务之间的调用，然而这种方式需要将生产者的服务地址写在代码中，当服务数量增多时，这些地址将变得难以维护。



使用服务注册中心后，生产者将自身信息提供给注册中心，消费者从注册中心拿到生产者信息后对其进行调用，因此消费者只需知道生产者的标识即可进行服务调用。



CAP 原理: C、A、P无法共存，两两组合。

- C： 一致性，集群之中多个节点数据一致。违反时有时候会拿到旧数据。
- A： 可用性，所有节点都高可用，且无延迟。违反时有时服务无法访问。
- P： 分区容忍，可以分区存储数据，节点之间可以通信。

BASE 理论：此理论是CAP原理的折中，即最终一致、基本可用、软状态。满足AP的注册中心也可满足此理论。

需要注意的是，服务无论是注册到注册中心还是从注册中心调取服务，都需要配置为对应注册中心的客户端。

Eureka (AP)

Eureka 是 Netflix 出品的用于实现服务注册和发现的工具，Spring Cloud 封装了 Netflix 公司开发的 Eureka 模块来实现服务注册和发现。

搭建 Eureka 服务

创建eureka服务，并添加依赖：eureka server

XML

```xml
<!--引入EurekaService-->
<dependencies>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
    </dependency>
</dependencies>
```

yml配置

YAML

```yaml
server: #服务端口
 port: 8081
#------------------------eureka配置
eureka:
 instance:
   hostname: localhost
 client:
   register-with-eureka: false # 是否将自己注册到注册中心
   fetch-registry: false       # 是否从注册中心获取注册信息。
   service-url:                # 暴露自身地址，即给Eureka Client的请求地址
     defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/

```

注册中心本身也是一个服务，最后一行配置中，Eureka 将自身地址暴露出去，服务调用者、服务提供者通过此地址来进行注册、调用。

开启 Eureka 服务：@EnableEurekaServer 注解

```Java
@SpringBootApplication
@EnableEurekaServer//激活eurekaService
public class EurekaApplication {

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

到此，eureka 服务就搭建好了，当然，调用者、提供者也需要一些配置才行。

服务注册与发现

服务注册，即服务将自身注册到注册中心，以供其它服务进行调用。

依赖：eureka client

```XML
<!--引入EurekaClient-->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
```

配置文件中添加 EurekaService 的信息：

```YAML
#eureka配置
eureka:
 client:
   service-url:
     defaultZone: http://localhost:8081/eureka/ #配置注册中心service地址,可访问Eureka面板
 instance:
   prefer-ip-address: true #使用ip地址注册(可选)

```

可以使用 @EnableEurekaClient 注解（可以省略）来开启服务客户端。暴露的地址 IP+端口直接在浏览器访问 Eureka 控制台，可以看到服务信息。

如果想在控制台查看服务 IP,则可以如下配置：

```YAML
eureka:
    instance:
      prefer-ip-address: true # 使用ip地址注册(可选)
      # 向服务注册中心注册服务id
      instance-id: ${spring.cloud.client.ip-address}:${server.port}

```

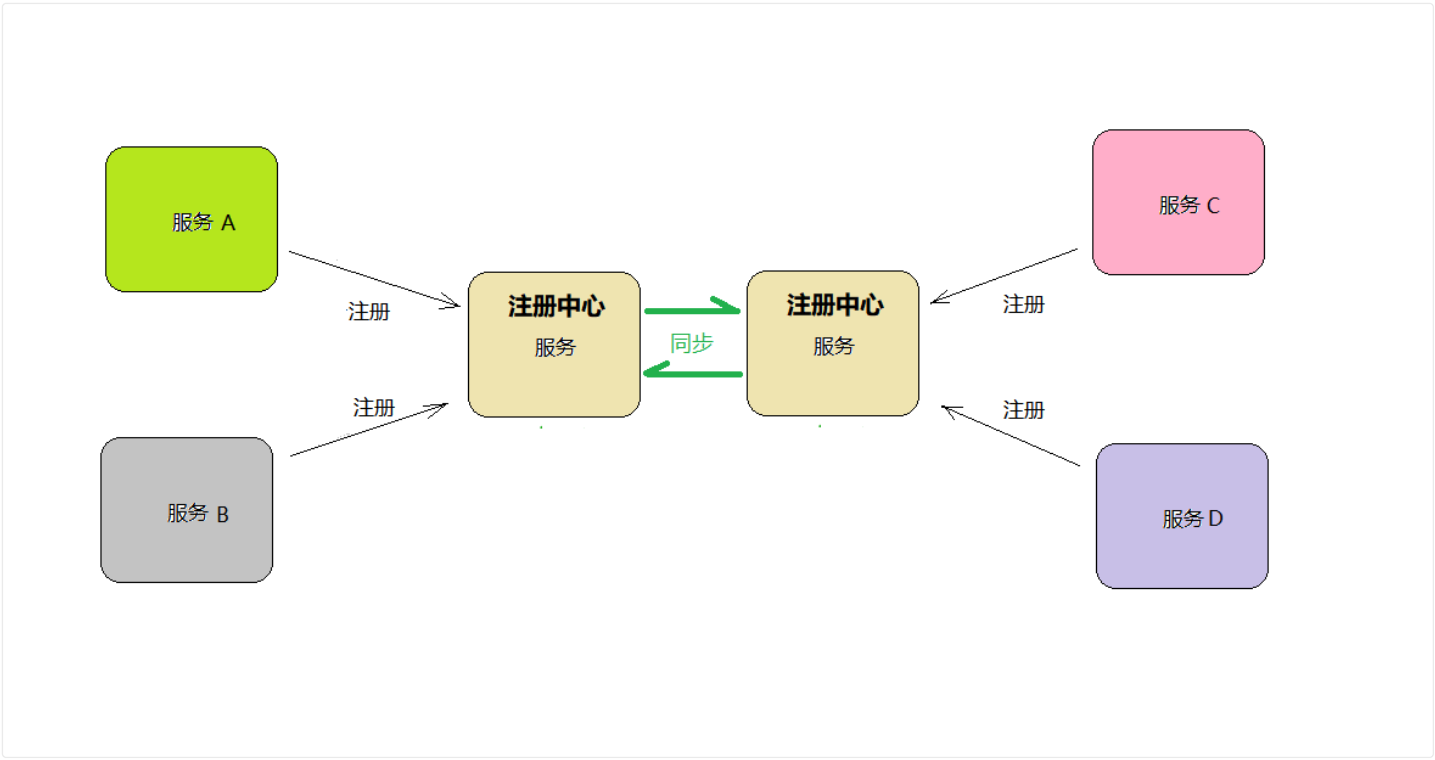
服务高可用、Eureka 集群

服务高可用是指服务能够持续地对外提供服务。Eureka 作为 SpringCloud 服务中的一员，必然存在宕机的风险，如果注册中心挂掉，那么整个服务将不可用。

搭建多个 Eureka 服务，组成一个集群，其中一个宕机了，其它的可以进行替代，从而保证注册中心的高可用。

搭建 Eureka 集群

创建多个 Eureka 服务，所有生产者都注册到所有 Eureka 服务、所有消费者都查询所有 Eureka 服务。多个 Eureka 服务之间相互注册，通过同步来保证数据的一致性。



实际的代码步骤如下：

YAML

```yaml
server: #服务端口
 port: 8081
 application:
   name: eureka-service #服务名
#eureka配置
eureka:
 instance:
   hostname: localhost
#-------------------------------------------------暴露
#  client:
#    register-with-eureka: false #是否将自己注册到注册中心
#    fetch-registry: false #是否从注册中心获取注册信息。
#    service-url: #配置暴露给Eureka Client的请求地址
#      defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/

#-------------------------------------------------注册
 client:
   service-url: # 配置注册中心地址
     defaultZone: http://localhost:8082/eureka/ # 注册到8082（2号），并暴露自身

```

通常 Eureka 服务集群有 3–5 个，生产者可以通过如下配置注册到多个注册中心：

YAML

```yaml
defaultZone: http://localhost:8082/eureka/ http://localhost:8083/eureka/
```

服务剔除机制、自我保护机制

服务剔除机制

注册到 Eureka 上的服务，默认每 30s 向注册中心发送”心跳“，如果注册中心超过 90s 未收到某个服务的心跳，则会自动认为该服务宕机并剔除该服务。

要注册的服务配置中配置自己的心跳：

YAML

```yaml
eureka:
   instance:
     prefer-ip-address: true #使用ip地址注册(可选)
     #向服务注册中心注册服务id
     instance-id: ${spring.cloud.client.ip-address}:${server.port}
     lease-renewal-interval-in-seconds: 33 #心跳间隔
     lease-expiration-duration-in-seconds: 99 #到期时间

```

自我保护机制

Eureka 可以根据服务心跳频率来保护服务，防止由于网络慢等原因导致未能及时发送心跳的可用服务被误剔除。

注册中心剔除服务的配置：

YAML

```yaml
eureka:
   server:
     enable-self-preservation: false  # 关闭自我保护
     eviction-interval-timer-in-ms: 4000 # 剔除服务间隔，单位ms
```

Consul (CP)

Consul 是HashiCorp公司推出的开源工具，由Go语言开发，容易部署，支持http、dns协议接口。

Consul 集群中的主节点宕机时，将暂时停止对外服务，直到选举出新的主节点后恢复可用。

搭建 Consul 服务

1. 下载consul安装包： 官网: https://www.consul.io/
2. 启动consul，管理后台：http://localhost:8500

Bash

```bash
# Dos命令：以开发者模式快速启动
consul agent -dev -client=0.0.0.0

# 参数：
# agent：启动一个consul的守护进程。
# dev：开发者模式
# client ：consul代理，和consulService交互。一个微服务对应一个client，它们需要部署在一起。
# service：真正使用的consul服务，(3-5个)
```

服务注册与发现

服务注册到注册consul，则需要进行如下配置：

引入依赖：

XML

```xml
<!--引入springCloud基于consul的服务发现-->
<dependency>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-starter-consul-discovery</artifactId>
</dependency>
<!--引入actuator的健康检查-->
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

添加consul相关配置：

YAML

```yaml
#开始配置consul的服务注册
spring:
  cloud:
    consul:
      host: localhost  #注册的服务器地址
      port: 8500  #ip地址
      discovery:
        register: true  #是否需要注册，默认为true
        instance-id: ${spring.application.name}-1 #注册的实例ID,-1为随便加的，用于区分
        service-name: spring.application.name #服务名称
        port: ${server.port}  #服务端口
        prefer-ip-address: true #是否开启ip地址注册
        ip-address: ${spring.cloud.client.ip-address}  #当前服务的ip地址

```

服务高可用、Consul 集群

consul 强调数据一致性，使用 Gossip 协议(流言协议)来保持微服务与所有 service 之间数据的一致性。

以下协议主要是为了保证注册中心集群中各节点数据保持一致：

Gossip 协议：所有 consul 都参与 gossip 协议中(多节点中数据赋值)。详情

Raft 协议：此过程有三个角色，详情

- Leader ： 领导者，server 集群中唯一处理客户端请求的。
- Follower：选民，被动接收数据。
- Candidate：候选者，可以被选为 leader，即其它 Consul。

consul 集群的范围是 3–5 台，Linux 搭建步骤如下：

```bash
#从官网下载最新版本的Consul服务
wget https:releases.hashicorp.com/consul/1.5.3/consul_1.5.3_linux_amd64.zip

#使用unzip解压
unzip consul_1.5.3_linux_amd64.zip

#将解压好的consul可执行命令拷贝到/usr/local/bin目录下
cp consul /usr/local/bin

#测试结果
consul
```

将三台主机分别安装并启动 consul 服务：

```bash
# Linux启动consul服务
consul agent -server -bootstarp-expect 3 -data-dir /etc/consul.d -node=server-1 -bind=S1的ip -ui ->client 0.0.0.0 &

# 参数：
# server：以server身份启动
# bootstarp-expect：集群里要求最少的server数量，数量比它低时，集群失效。
# data-dir  ：data存放的目录
# node：节点id，同一集群内不可重复。
# bind：监听的ip地址。
# client：客户端的ip地址，(0.0.0.0表示不限制)
# &：在后台运行，Linux脚本语法
```

选择一台作为主节点，并在其它两个节点执行如下操作即可：

```bash
# 添加到主节点
consul join 主节点ip
```

开发时，需要本地的 consul 也作为 client 注册到主节点，因为服务注册在本地的 consul：

```bash
# 启动本地client
consul agent -client=0.0.0.0 -data-dir /etc/consul.d -node=client-1

# 注册到主节点
consul join 主节点ip
```

查看所有节点信息：

```bash
# 查看集群节点信息
consul members
```

Consul 的注册、发现、删除测试

实际注册过程就是发送 put 请求到 consul 注册中心: http://localhost:8500/v1/catalog/register

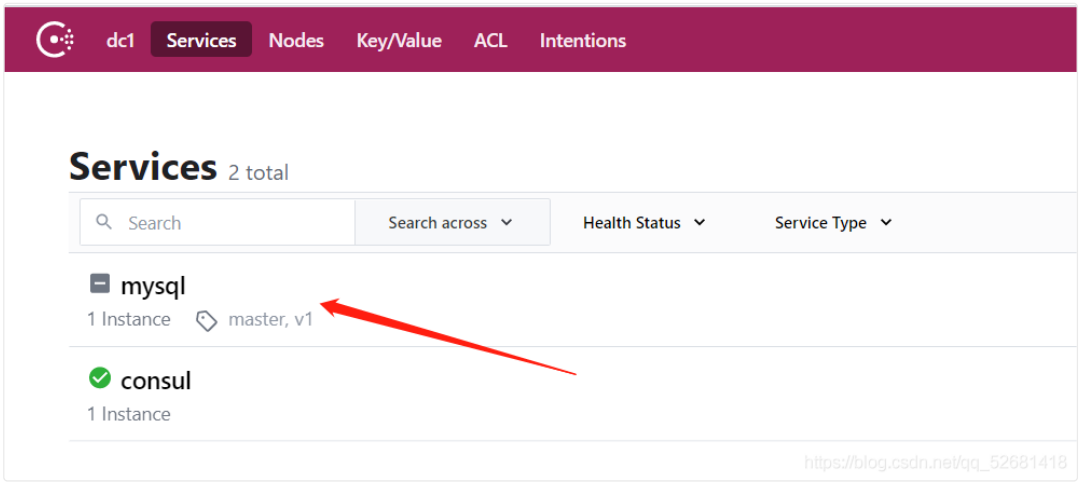
实际删除过程就是发送 put 请求到 consul 注册中心: http://localhost:8500/v1/catalog/deregister

发送 get 请求到 consul 注册中心查看服务健康信息：http://127.0.0.1:8500/v1/health/service/服务名

通过postman发送如下内容测试服务注册：

```JSON
{
    "Datacenter":"dc1",
    "Node":"node01",
    "Address":"127.0.0.1",
    "Service":{
        "ID":"mysql-01",
        "Service":"mysql",
        "tags":["master","v1"],
        "Address":"127.0.0.1",
        "port":3306
    }
}
```

响应结果为true，表示成功注册，此时访问：http://localhost:8500 就能看到注册的服务。



在这里插入图片描述

通过postman发送如下内容测试服务删除：

```JSON
{
    "Datacenter":"dc1",
    "Node":"node01",
    "ServiceID":"mysql-01"
}
```

返回为true表示删除成功，刷新consul控制台就会发现服务已经被删除了。

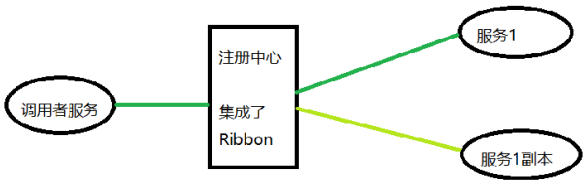
注册中心拓展

负载均衡

负载均衡是指服务请求过大时，将请求均摊给"自己的分身"，防止压力过大而死掉。

- 功能服务负载均衡：在服务调用时，可以通过Ribbon来实现调用时的负载均衡配置。
- 注册中心负载均衡：使用代理服务来将请求均匀分配到注册中心集群，从而实现注册中心的负载均衡。

服务负载均衡



注册中心负负载均衡