服务注册发现

对于微服务来讲，服务之间也是需要做服务发现的，常见的框架是 Dubbo 和 Spring Cloud，服务的注册中心可以是 ZooKeeper、Consul、etcd、Eureka 、nacos等。

为啥需要服务注册与发现?

前面介绍了，一个服务的吞吐量为1000qps

> 亿级用户，需要10W qps， 简单计算， 需要100个服务实例

100实例，如何发现，如何管理?

答案：服务注册与发现

> 服务注册发现，是服务治理的重要内容

什么是服务注册与发现?

在服务启动时，服务提供者会向注册中心注册服务，暴露自己的地址和端口等，注册中心会更新服务列表。

服务消费者启动时会向注册中心请求可用的服务地址，并且在本地缓存一份提供者列表，这样即便注册中心宕机了，仍然可以正常调用服务。

如果提供者集群发生变更，注册中心会将变更推送给服务消费者，更新可用的服务地址列表。

服务注册组件

三种典型的服务注册组件，分别是 ZooKeeper、Eureka 和 Nacos。

ZooKeeper

ZooKeeper 主要应用在 Dubbo 的注册中心实现，Dubbo + ZooKeeper 的典型服务化方案

服务提供者在启动的时候，会在 ZooKeeper 上注册服务。

注册服务:

以 com.dubbo.DemoService 为例，其实就是在 ZooKeeper 的 /dubbo/com.dubbo.DemoService/providers 节点下创建一个**子节点**，并写入自己的 URL 地址，这就代表了 com.dubbo.DemoService 这个服务的一个提供者。

服务发现:

服务消费者在启动的时候，会向 ZooKeeper 注册中心订阅服务列表，就是读取并订阅 ZooKeeper 上 /dubbo/com.dubbo.DemoService/providers 节点下的所有子节点，并解析出所有提供者的 URL 地址来作为该服务地址列表。

zookeeper的高可用

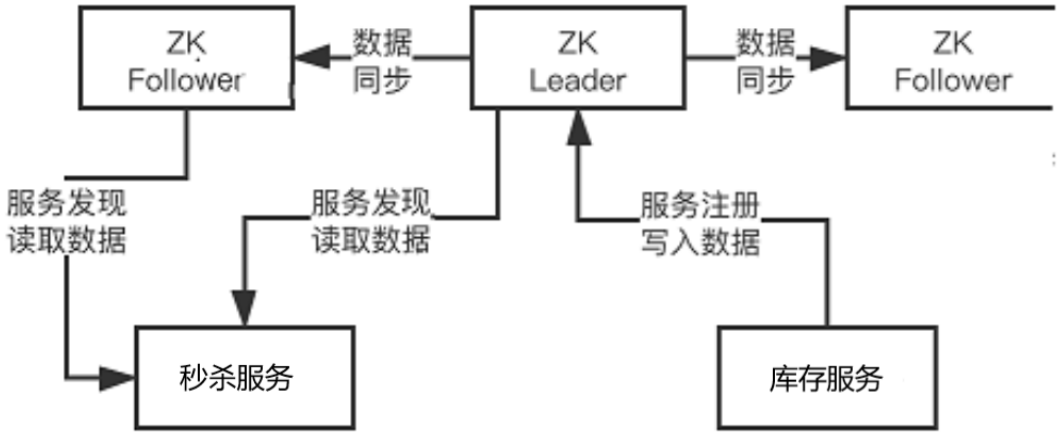
zab协议

https://www.cnblogs.com/crazymakercircle/p/14339702.html

zookeeper集群的原理，leader+follower，leader写，同步到follower，follower可以读，保证顺序一致性，就是基本尽量保证到数据一致的，主动推送，

典型的CP，leader崩溃的时候，为了保证数据一致性，尽量不要读到不一致的数据，此时要重新选举leader以及做数据同步，此时集群会短暂的不可用

比如这里的库存服务是provider service，秒杀服务是consumer service，大致流程是这样的：



Eureka

测试环境的地址：

> http://cdh1:7777/

在 Spring Cloud 中，提供了 Eureka 来实现服务发现功能。Eureka 采用的是 Server 和 Client 的模式进行设计，Eureka Server 扮演了服务注册中心的角色，为 Client 提供服务注册和发现的功能。

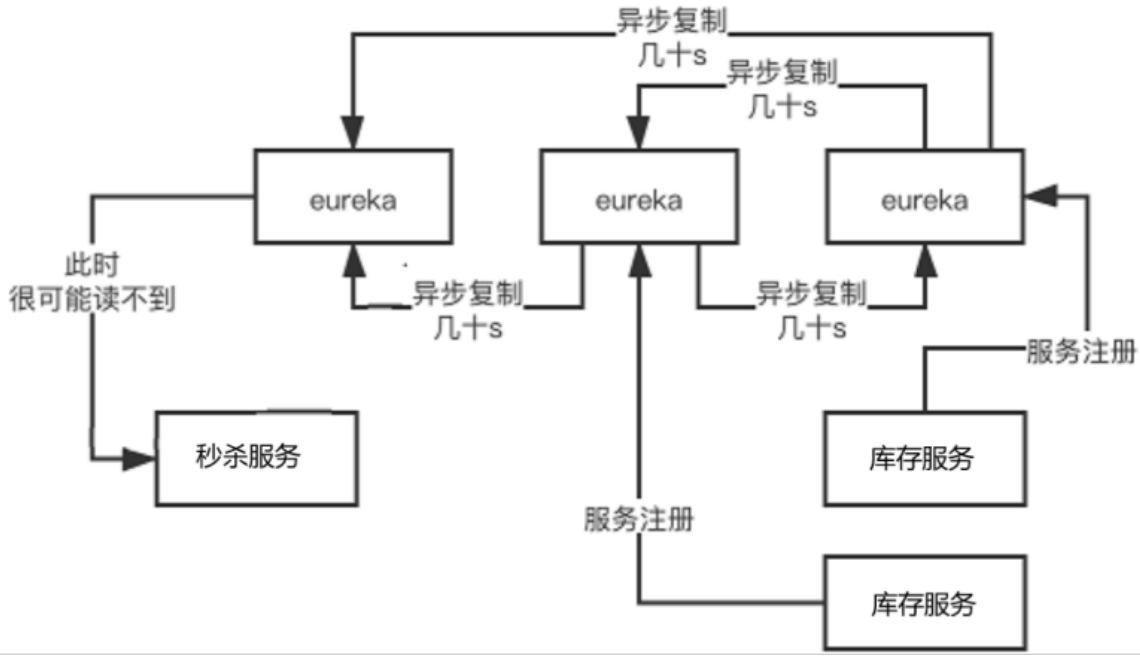
Eureka Client 通过客户端注册的方式暴露服务，通过注解等方式嵌入到服务提供者的代码中，当服务启动时，服务发现组件会向注册中心注册自身提供的服务，并周期性地发送心跳来更新服务。

如果连续多次心跳不能够发现服务，那么 Eureka Server 就会将这个服务节点从服务注册表中移除，各个服务之间会通过注册中心的注册信息来实现调用。

Eureka高可用

集群式 eureka的原理，peer-to-peer，大家都能写也都能读，每个节点都要同步给其他节点，但是是异步复制的，所以随时读任何一个节点，可能读到的数据都不一样，任何一个节点宕机，其他节点正常工作，可用性超高，但是数据一致性不行，AP

比如这里的库存服务是provider service，秒杀服务是consumer service，大致流程是这样的：



nacos

测试环境的地址：

> http://cdh1:8848/nacos/#/login

Nacos 是阿里开源的，可以方便地集成 Spring Cloud 框架。nacos现在用的越来越多，以后也会是一个大的趋势，但是现在可能还没那么的普及

Nacos是基于raft算法的CP模型，同时也支持配置成类似eureka的AP。

其实CP或者AP也都行，CP就是偶尔可能短时间不可用，AP就是可能数据不一致，两个都有问题，但是在生产环境下，无论CP还是AP其实都用的很多

除了服务注册和发现之外，Nacos 还提供了配置管理、元数据管理和流量管理等功能，并且提供了一个可视化的控制台管理界面。

具体，请参考：

https://www.cnblogs.com/crazymakercircle/p/14231815.html

Eureka的使用

Eureka本身是Netflix开源的一款注册中心产品，并且Spring Cloud提供了相应的集成封装，选择其作为注册中心的讲解实例是出于以下的原因：

（1）Eureka在业界的应用十分广泛（尤其是国外），整个框架也经受住了Netflix严酷生产环境的考验。

（2）除了Eureka注册中心，Netflix的其他服务治理功能也十分强大，包括Ribbon、Hystrix、Feign、Zuul等组件结合到一起组成了一套完整的服务治理框架，使得服务的调用、路由也变得异常容易。

那么，Netflix和Spring Cloud是什么关系呢?

Netflix是一家互联网流媒体播放商，是美国视频巨头，访问量非常大。也正是如此，Netflix把整体的系统迁移到了微服务架构，并且Netflix就把它的几乎整个微服务治理生态中的组件都开源贡献给了Java社区，叫做Netflix OSS。

Spring Cloud是Spring背后的Pivotal公司（由EMC和VMware联合成立的公司）在2015年推出的开源产品，主要对Netflix开源组件的进一步封装，方便Spring开发人员构建微服务架构的应用。

Spring Cloud Eureka是Spring Cloud Netflix微服务套件的一部分，基于Netflix Eureka做了二次封装，主要负责完成微服务实例的自动注册与发现，这也是微服务架构中最为核心和基础的功能。

Eureka所治理的每一个微服务实例被称之为Provider Instance(提供者实例)。每一个Provider Instance微服务实例包含一个Eureka Client客户端组件（相当于注册中心客户端组件），其主要的工作为：

（1）向Eureka Server完成Provider Instance的注册、续约和下线等操作，主要的注册信息包括服务名、机器IP、端口号、域名等。

（2）向Eureka Server获取Provider Instance清单，并且缓存在本地。

一般来说，Eureka Server作为服务治理应用会独立地部署和运行。一个Eureka Server注册中心应用在新建时，首先需要在pom.xml文件中添加上eureka-server依赖库。

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

然后，需要在启动类中添加注解@EnableEurekaServer，声明这个应用是一个Eureka Server，启动类的代码如下：

```
package com.crazymaker.springcloud.cloud.center.eureka;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;
//在启动类中添加注解 @EnableEurekaServer
@EnableEurekaServer
@SpringBootApplication
public class EurekaServerApplication {
    public static void main(String[] args) {
        SpringApplication.run(EurekaServerApplication.class, args);
    }
}
```

Eureka Server的配置

接下来，在应用配置文件application.yml中对Eureka Server的一些参数进行配置。一份基础的配置文件大致如下：

```
server:
    port: 7777
spring:
    application:
        name: eureka-server
eureka:
    client:
        register-with-eureka: false
        fetch-registry: false
        service-url:
            #服务注册中心的配置内容，指定服务注册中心的位置
            defaultZone:
${SCAFFOLD_EUREKA_ZONE_HOSTS:http://localhost:7777/eureka/}
    instance:
hostname: ${EUREKA_ZONE_HOST:localhost}
    server:
         enable-self-preservation: true # 开启自我保护
eviction-interval-timer-in-ms: 60000 # 扫描失效服务的间隔时间（单位毫秒，默认是
60*1000）即60秒
```

以上的配置文件中，包含了三类配置项：

作为服务注册中心的配置项（eureka.server.）、*作为Provider提供者的配置项（eureka.instance.*）、

作为注册中心客户端组件的配置项（eureka.client.*），至于具体的原因稍后介绍。

Eureka Server的打包

fatjar（肥包）

生产环境，主要使用docker镜像的模式，推送到git仓库

Eureka Server的部署

配置完成后，通过运行启动类EurekaServerApplication就可以启动Eureka Server，然后通过浏览器访问Eureka Server的控制台界面（其端口为server.port配置项的值），大致如图2-2所示。

http://cdh1:7777/

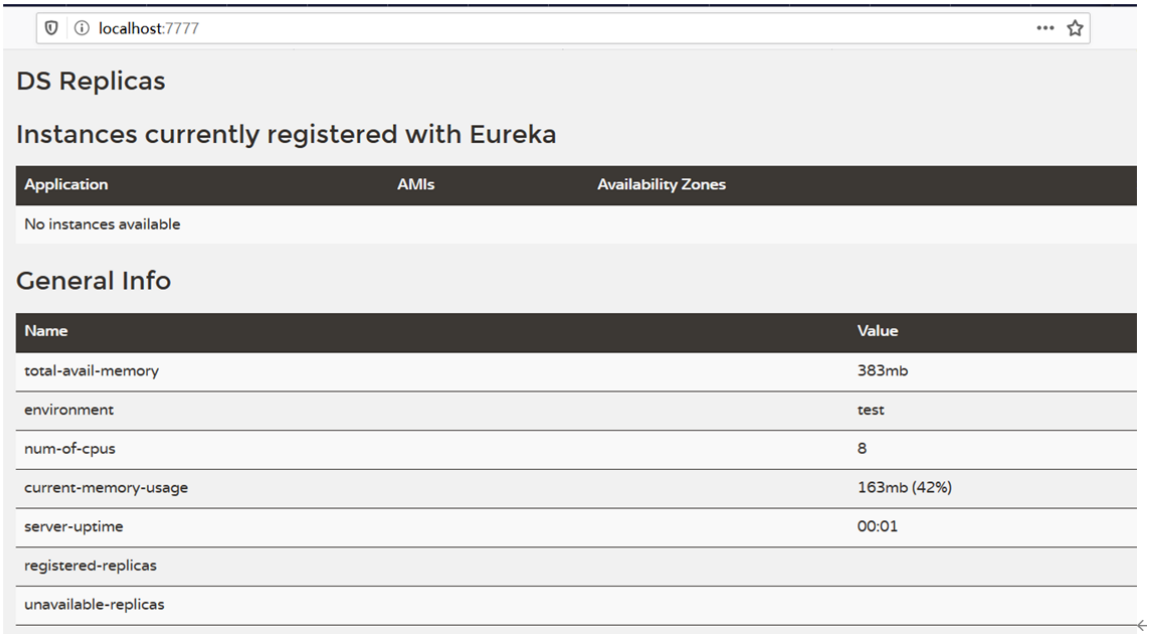


图2-2 Eureka Server的控制台界面