目录

[1. 注册中心 1](#_Toc96529821)

[1.1. 概念 1](#_Toc96529822)

[1.2. 三大角色：服务提供者、服务消费者、注册中心 2](#_Toc96529823)

[1.3. 注册中心功能 2](#_Toc96529824)

[1.4. 技术栈 2](#_Toc96529825)

[2. Eureka 3](#_Toc96529826)

[2.1. 服务注册与发现包括两部分：服务器端Server、客户端Client 3](#_Toc96529828)

[2.2. 原理 3](#_Toc96529829)

[2.3. Eureka Client部分源码 4](#_Toc96529830)

[2.4. Eureka Server部分源码 10](#_Toc96529831)

[2.5. 注意 23](#_Toc96529832)

[3. Zookeeper 23](#_Toc96529833)

[3.1. 说明 23](#_Toc96529835)

[3.2. 模型和节点 23](#_Toc96529836)

[3.3. 用作注册中心 23](#_Toc96529837)

[3.4. 感知服务的上线&下线 24](#_Toc96529838)

[3.5. 统一配置管理 25](#_Toc96529839)

[3.6. 统一服务命名 25](#_Toc96529840)

[3.7. 分布式锁 26](#_Toc96529841)

[4. Nacos 27](#_Toc96529842)

[4.1. 服务注册 27](#_Toc96529847)

[4.2. 服务发现 27](#_Toc96529848)

[4.3. 注册原理 29](#_Toc96529849)

[4.4. 发现原理 30](#_Toc96529850)

[4.5. 负载均衡 31](#_Toc96529851)

[4.6. 完整流程 32](#_Toc96529852)

[5. Consul 33](#_Toc96529853)

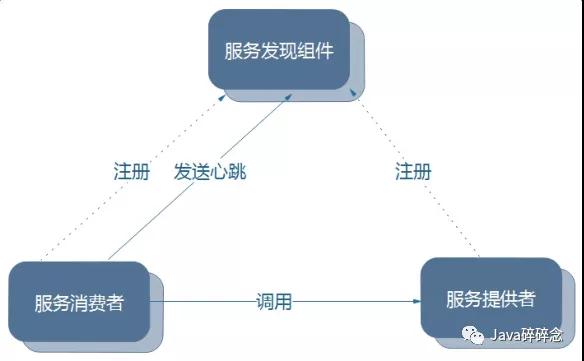
# 注册中心

## 概念

在传统应用组件间调用，通过接口规范约束来实现的，从而实现不同模块间良好协作；但是被拆分成微服务后，每个微服务实例的数量和网络地址都可能动态变化，使用原来硬编码的地址极不方便，故需要一个中心化的组件来进行服务器的登记和管理。

## 三大角色：服务提供者、服务消费者、注册中心

1. 各个微服务在启动时，将自己的网络地址等信息注册到注册中心，注册中心存储这些数据。
2. 服务消费者从注册中心查询服务提供者的地址，并通过该地址调用服务提供者的接口。
3. 各个微服务与注册中心使用一定机制（例如心跳）通信。如果注册中心与某微服务长实际无法通信，就会注销该实例。
4. 微服务网络地址发生变化（例如实例增加或IP变动等）时，会重新注册到注册中心，服务消费者就无需人工修改提供者的网络地址了。



## 注册中心功能

1. 服务注册表：服务注册表是注册中心的核心，它用来记录各个微服务的信息，例如微服务的名称、IP、端口等。服务注册表提供查询API和管理API，查询API用于查询可用的微服务实例，管理API用于服务的注册与注销。
2. 服务注册与发现：服务注册是指微服务在启动时，将自己的信息注册到注册中心的过程。服务发现是指查询可以用的微服务列表及网络地址的机制。
3. 服务检查：注册中心使用一定的机制定时检测已注册的服务，如果发现某实例长时间无法访问，就会从服务注册表移除该实例。

## 技术栈

Eureka、Nacos、Consul、Zookeeper

# Eureka

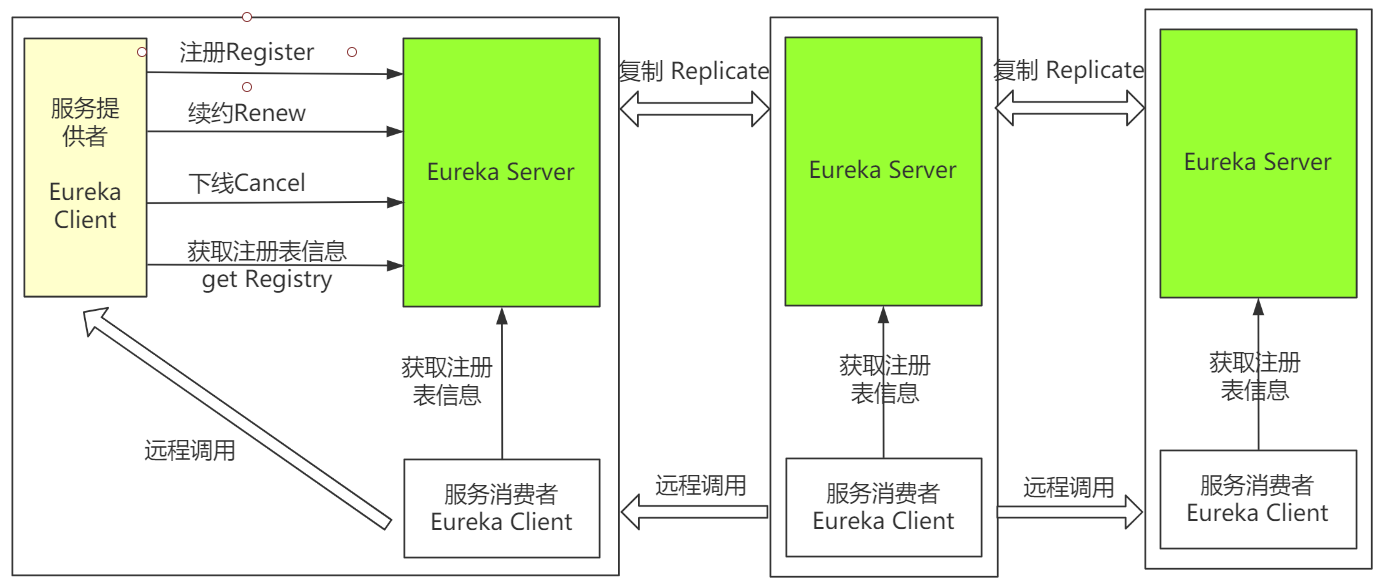


## 服务注册与发现包括两部分：服务器端Server、客户端Client

1. Server是一个公共服务，为Client提供服务注册和发现的功能，维护注册到自身的Client的相关，同时提供接口给Client获取注册表中其他服务的信息，使得动态变化得Client能够进行服务间得相互调用。
2. Client将自己得服务信息通过一定得方式登记到Server上，并在正常范围内维护自己信息一致性，方便其他服务发现自己，同时可以通过Server获取到自己依赖得其他服务信息，完成服务调用，还内置了负载均衡器，用来进行基本的负载均衡。
3. Spring Cloud以Eureka作为服务注册中心，是一个RESTful风格服务，是服务注册和发现的基础组件，它屏蔽了
4. Servcer和Client的交互细节，使得开发者将精力放在业务上。
5. Eureka也支持高可用，集群环境搭建与单节点环境类似，需要注意客户端配置地址defaultZone时候，尽量多写个地址（写一个也行，EurekaServer注册表会自动同步，避免极端情况数据同步（Eureka是AP模型）不及时）。

## 原理

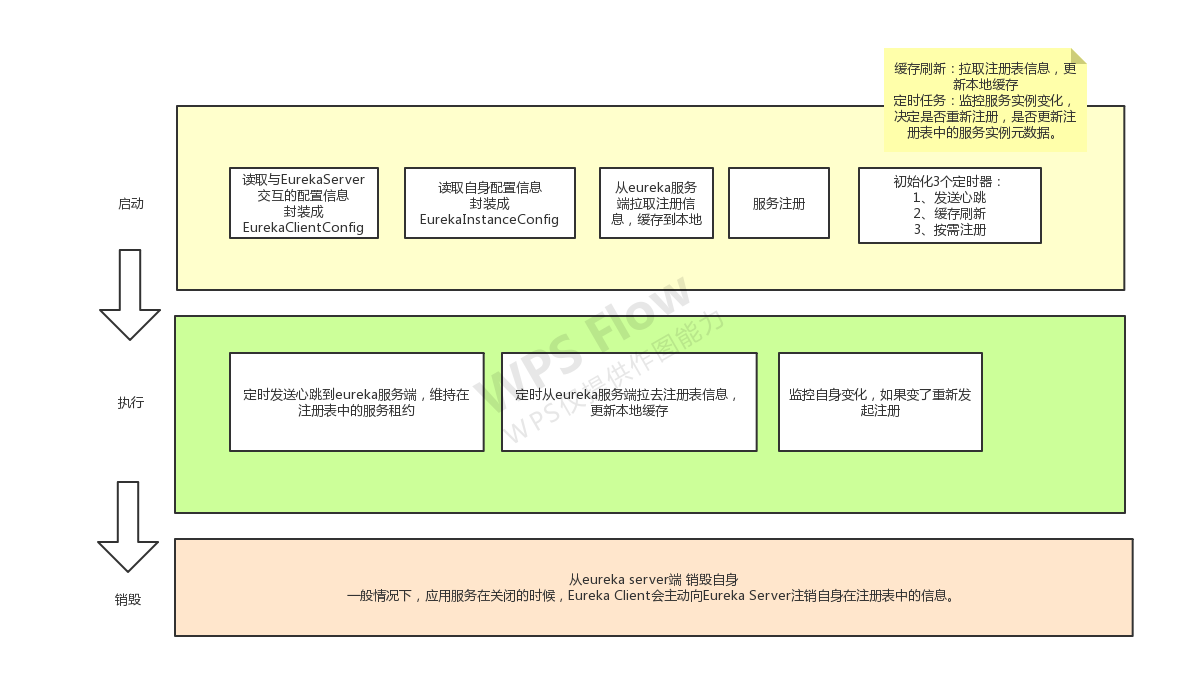
1. 本质：EurekaClient从EurekaServer同步获取服务注册列表，通过一定的规则选择一个服务进行调用。Eureka架构图如下：



1. 服务提供者：是一个Eureka client，向Eureka Server注册和更新自己的信息，同时能从Eureka Server注册表中获取到其他服务的信息。
2. 服务注册中心：提供服务注册和发现的功能，每个Client向Server注册自己的信息，也可以通过Server获取到其他服务的信息，达到发现和调用其他服务的目的。
3. 服务消费者：是一个Client，通过Server获取注册的其他服务信息，从而找到所需要的服务发起远程调用。
4. 注册：服务提供者向Server端注册自身的元数据以供服务发现。
5. 续约：通过发送心跳到Server以维持和更新注册表中服务实例元数据的有效性。在一定时长内，Server没收到Client的心跳信息，将默认下线，会把服务实例信息从注册表中删除。
6. 下线：服务提供方在关闭时候主动向Server注销服务实例元数据，这时服务提供方实例数据将从Server的注册表中删除。
7. 获取注册表：服务消费者向Server请求注册表信息，用于服务发现，从而发起远程调用。

## Eureka Client部分源码

1. 工作流程：



1. Eureka Client通过Spring boot自动装配，加载相关类，META-INF/spring.factories如下配置：

EurekaClientAutoConfiguration：Eureka client自动配置类，负责client中关键beans的配置和初始化。

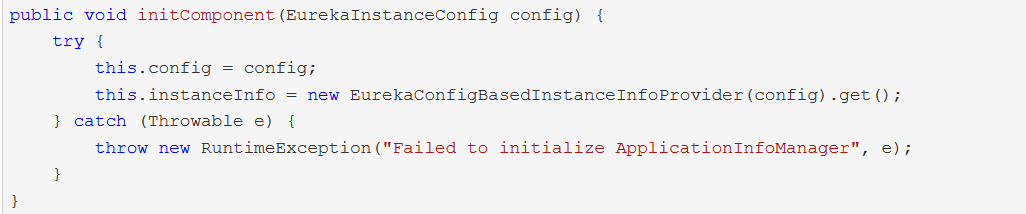
RibbonEurekaAutoConfiguration：Ribbon负载均衡相关配置。

EurekaDiscoveryClientConfiguration：配置自动注册和应用的健康检查器。

1. Eureka相关配置

EurekaClientConfigBean：封装了Eureka Client和Eureka Server交互所需要的配置信息。

EurekaInstanceConfigBean：封装了Eureka Client自身服务实例的配置信息，主要用于构建InstanceInfo。



1. Eureka客户端DiscoveryClient是核心类，实现了EurekaClient接口（EurekaClient继承LookupService接口），关键代码如下：

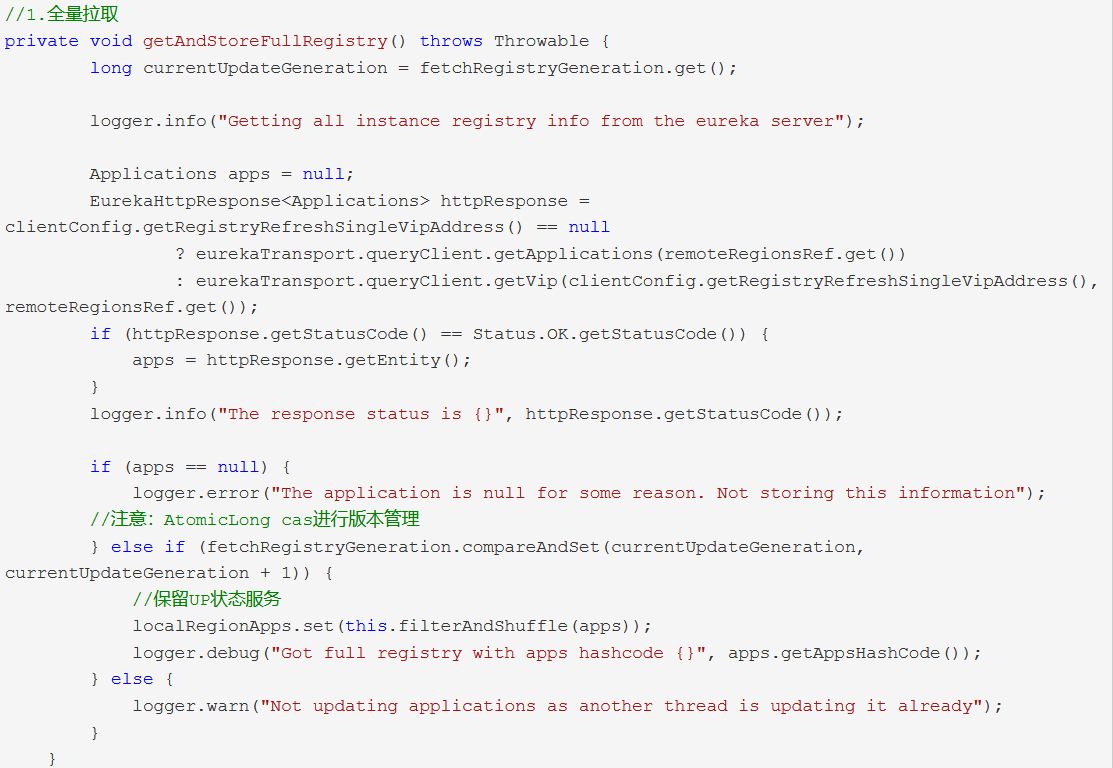
构造方法：



拉取注册表



全量拉取



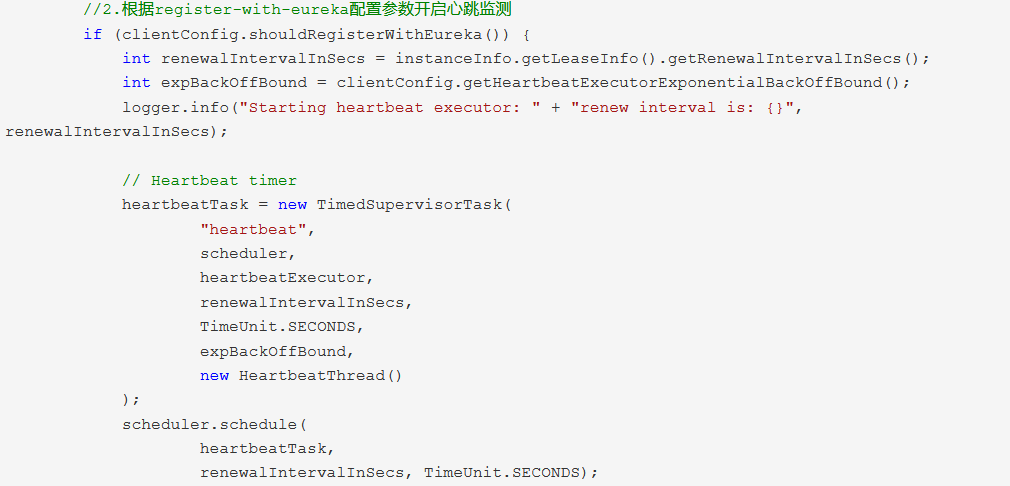
增量拉取



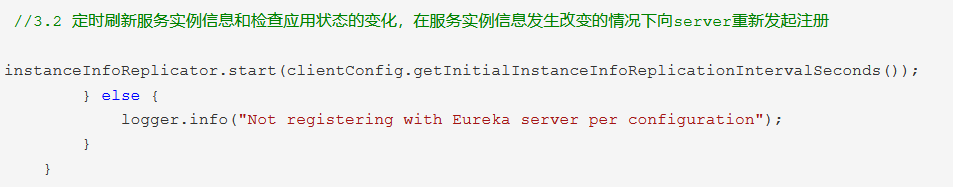
服务注册

初始化三个定时任务：initScheduledTasks









续租



服务下线



## Eureka Server部分源码

1. Server主要有以下几个功能：接收服务注册、接收服务心跳、服务剔除、服务下线、集群同步、获取注册表中服务实例信息。
2. Server同时也是一个Client，在不禁止Server的客户端行为时，他会向配置文件中的其他Server进行拉取注册表、服务注册和发送心跳等操作。
3. Server通过spring boot自动装配，加载相关类，META-INF/spring.factories如下配置：

EurekaServerAutoConfiguration：向spring的bean工厂添加eureka-server相关功能的bean。

EurekaServerAutoConfiguration生效条件是通过 EurekaServerMarkerConfiguration.Marker实现的，所以服务端需要加上@EnableEurekaServer注解。

1. EurekaServerInitializerConfiguration是Server初始化类



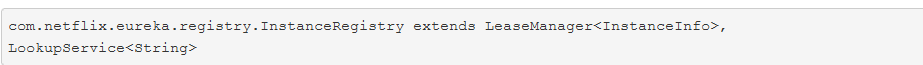
EurekaServerBootstrap启动类



1. Server是围绕注册表管理的，有两个InstanceRegistry。

com.netflix.eureka.registry.InstanceRegistry是Server中管理注册的核心接口，职责是在内存中管理注册到server中的服务实例信息，实现类有PeerAwareInstanceRegistryImpl。

org.springframework.cloud.netflix.eureka.server.InstanceRegistry对PeerAwareInstanceRegistryImpl进行了继承和扩展，使其适配Spring Cloud的使用环境，主要实现还是由PeerAwareInstanceRegistryImpl提供。

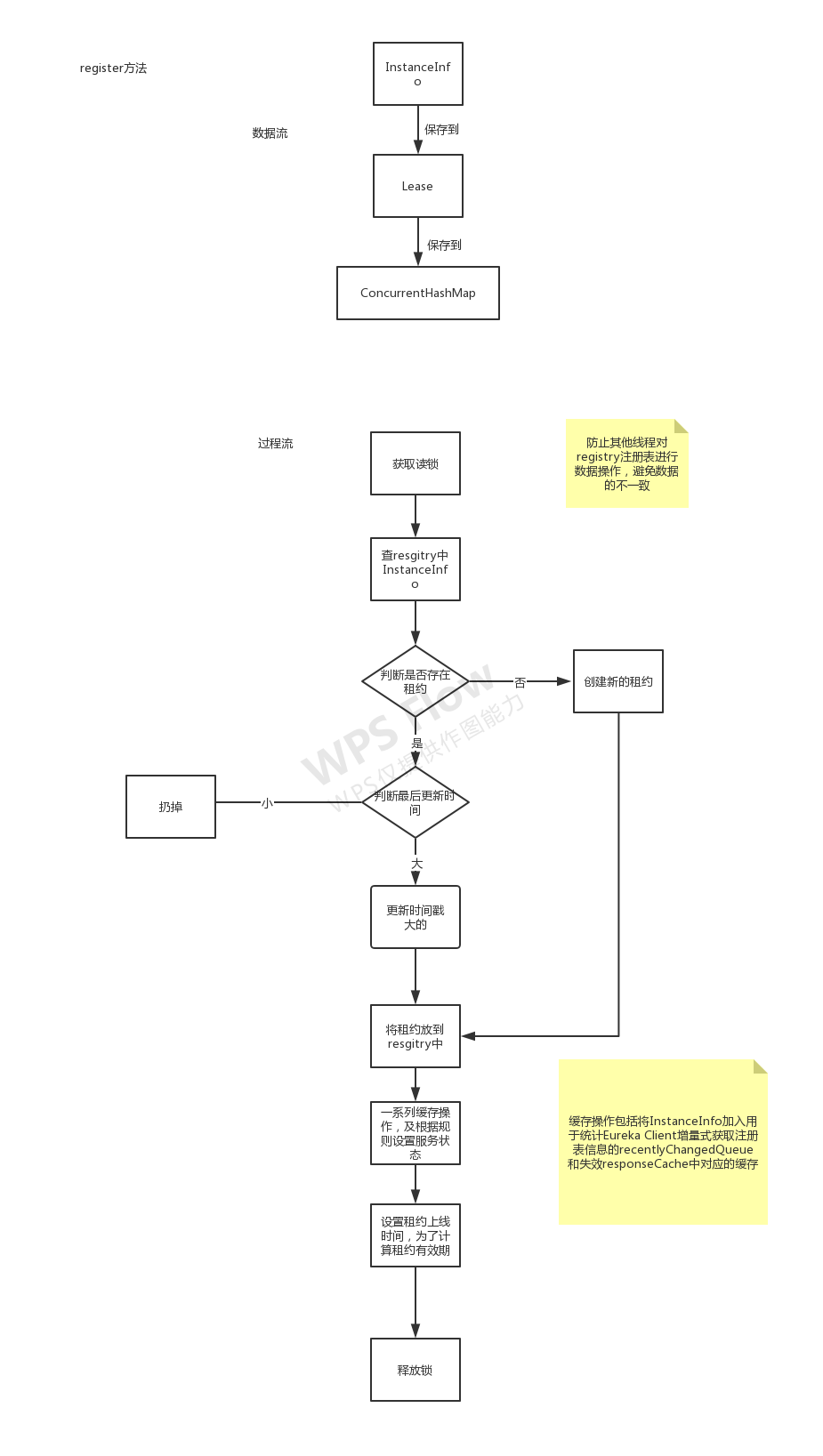


LookupService是提供服务实例的检索查询功能。

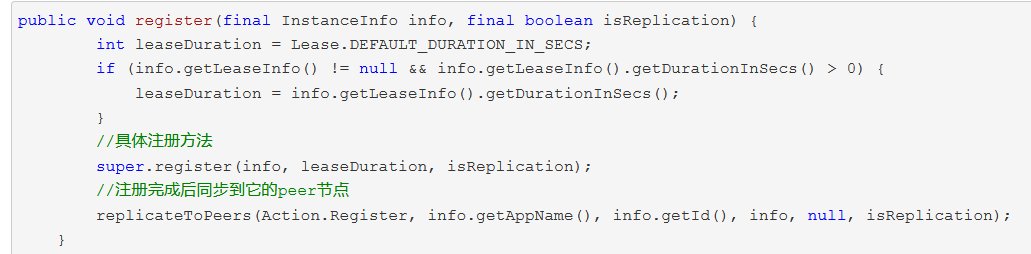
LeaseManager是对注册到Server中的服务实例租约进行管理，方法有：注册、下线、续约、剔除等。

1. PeerAwareInstanceRegistryImpl关键代码如下：

服务注册流程图



服务注册核心代码



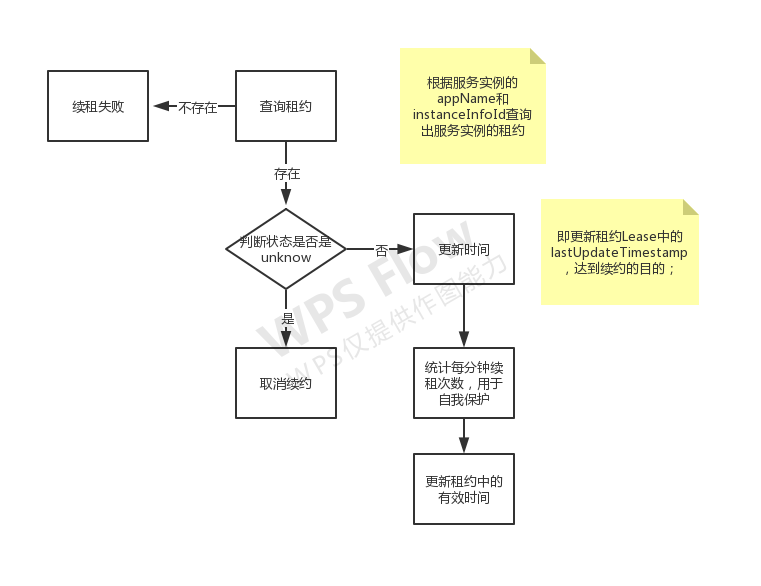




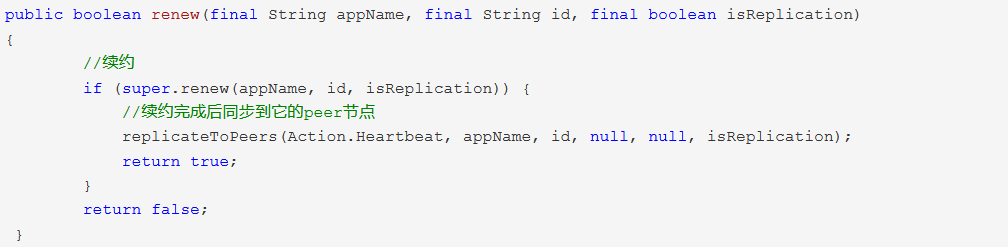




心跳续约流程图

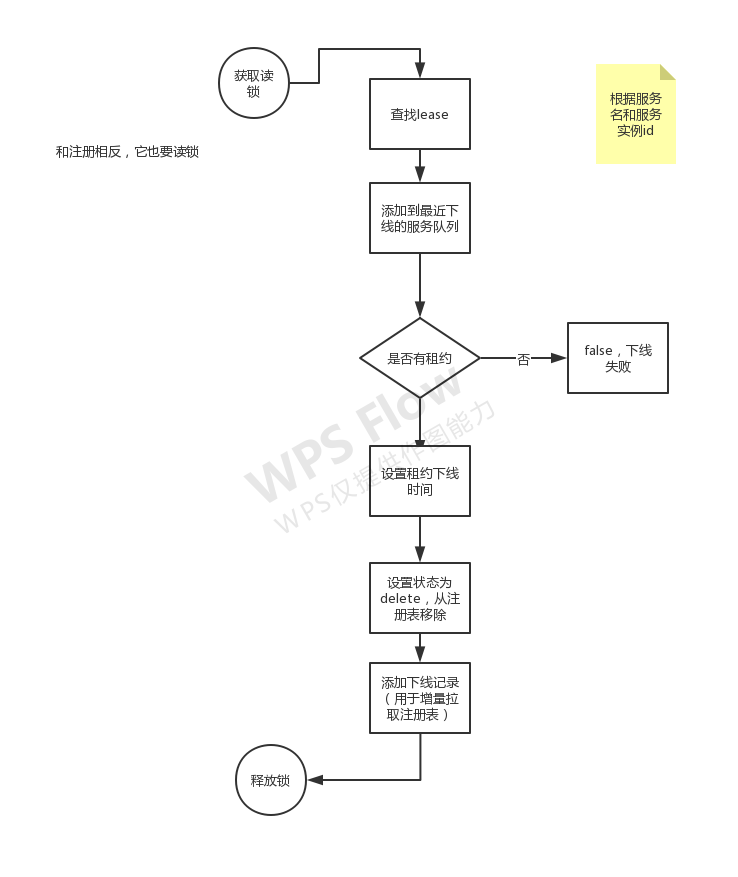


续约核心代码

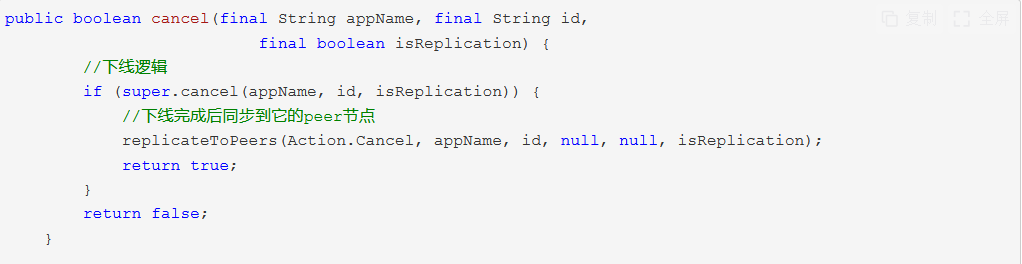


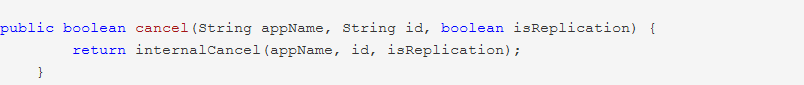


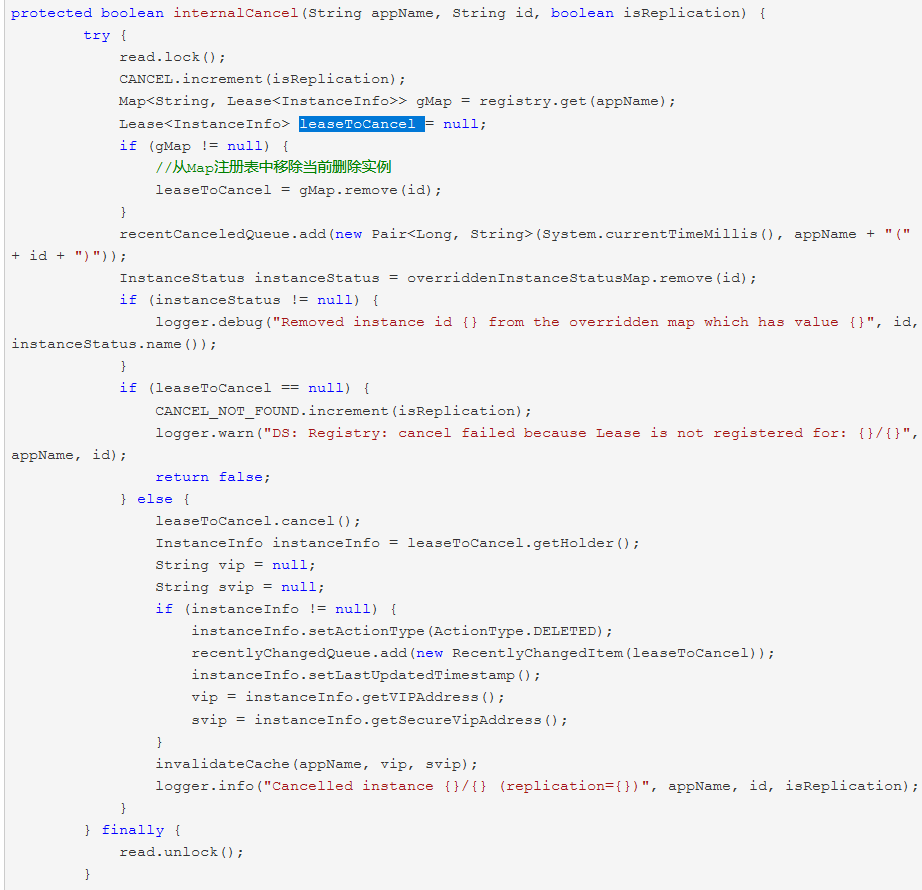
服务下线流程图

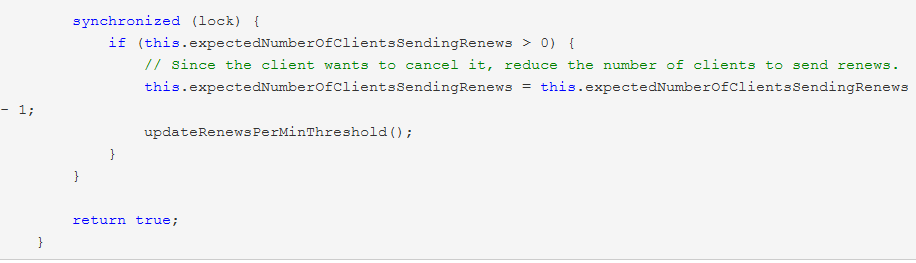


服务下线核心代码

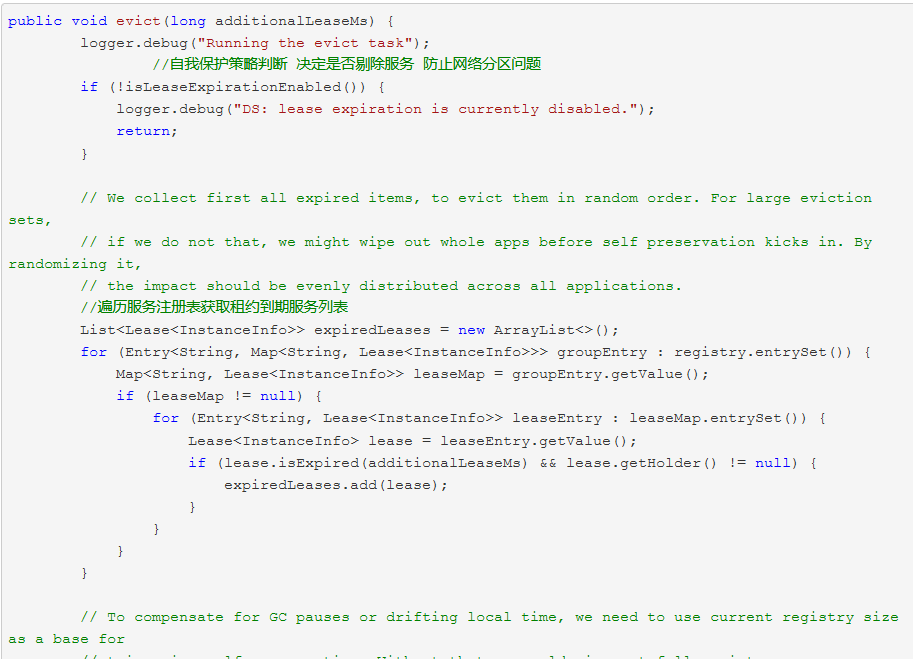






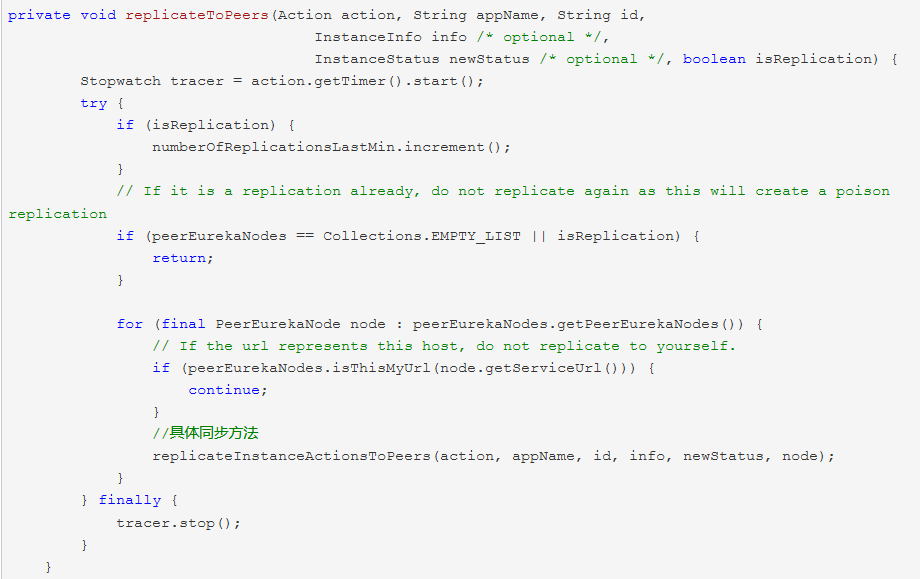


服务剔除核心代码





服务同步核心代码



服务同步核心代码



1. 自我保护

eureka.server.lease-renewal-interval-in-seconds（心跳发送频率）:默认30秒一次

expectedNumberOfRenewsPerMin（期望每分钟续约数）：当前应用注册实例数 \* 心跳发送频率

eureka.server.renewalPercentThreshold（续租百分比）：默认0.85

numberOfRenewsPerMinThreshold（期望阈值）：期望每分钟续约数 \* 续租百分比

renewsLastMin（每分钟心跳次数）

eureka.server.enable-self-preservation = true：开启自动保护模式

当每分钟心跳次数（renewsLastMin）小于期望阈值（numberOfRenewsPerMinThreshold）时，触发自我保护机制，不在自动过期租约。

1. Eureka事件

EurekaInstanceCanceledEvent：服务下线事件

EurekaInstanceRegisteredEvent：服务注册事件

EurekaInstanceRenewedEvent：服务续约事件

EurekaRegistryAvailableEvent：注册中心可用事件

EurekaServerStartedEvent：注册中心启动

1. 话花

## 注意

由于集群间的同步复制是通过HTTP的方式进行，基于网络的不可靠性，集群中的Eureka Server间的注册表信息难免存在不同步的时间节点，不满足CAP中的C（数据一致性）

# Zookeeper



## 说明

1. Zookeeper主要服务于分布式系统，可以用来做：统一配置管理、统一命名服务、分布式锁、集群管理。
2. 使用分布式系统就无法避免对节点管理的问题，而由于这些问题处理起来可能相对麻烦和提高了系统的复杂性，zookeeper作为一个能够通用解决这些问题的中间件就应运而生了。

## 模型和节点

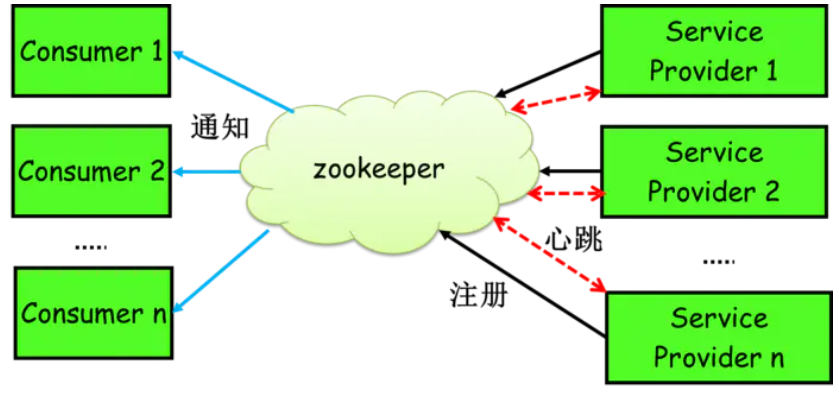
Zookeeper的数据结构类似文件系统的树形结构。

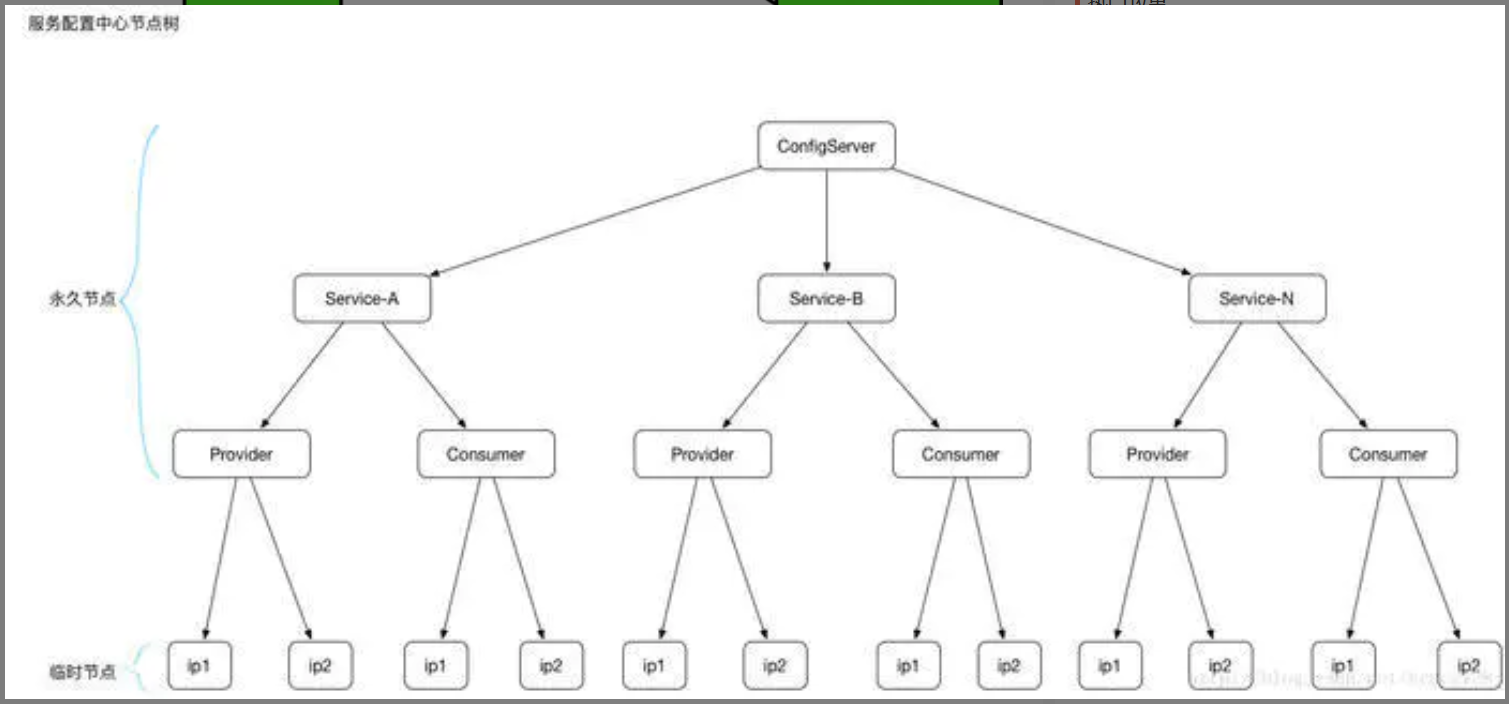
Zookeeper的节点类型有持久节点和临时节点俩种。

## 用作注册中心

具体来说，zookeeper就算一个分布式文件系统，每当一个服务提供者部署后都将自己的服务注册到zookeepr的某一路径上/{service}/{version}/{ip:port}。

在zookeeper中，进行服务注册，实际上就是在zookeeper中创建了一个znode节点，该节点存储了该服务的ip、端口、调用方式（协议、序列化方式）等。该节点承担着重要的职责，它由服务提供者（发布服务时）创建，以供服务消费者获取节点中的信息，从而定位到服务提供者真正网络拓扑位置以及得知如何调用。





1. 服务提供者启动时，会将其服务名称、ip地址注册到配置中心。
2. 服务消费者在第一次调用用服务时，会通过注册中心找到相应服务的ip地址列表，并缓存到本地，以供后续使用。当消费者调用服务时，不会再去请求注册中心，而是直接通过负载均衡算法从ip列表中取一个服务提供者的服务器调用服务。
3. 当服务提供者的某台服务器宕机或下线时，相应的ip会从服务提供者ip列表中移除。同时，注册中心会将新的服务ip地址列表发送给服务消费者机器，缓存在消费者本机。
4. 当某个服务的所有服务器都下线了，那么这个服务也就下线了。
5. 同样，当服务提供者的某台服务器上线时，注册中心会将新的服务ip地址列表发送给服务消费者服务器，缓存在消费者本机。
6. 服务提供方可以根据服务消费者的数量来作为服务下线的依据。

## 感知服务的上线&下线

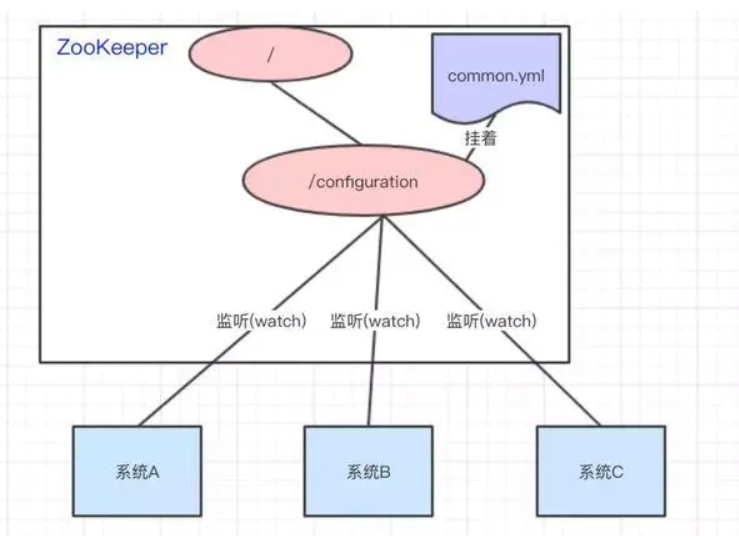
Zookeeper提供了心跳检测功能，它会定时向各个服务提供者发送一个请求（实际上建立的是一个socket长连接），如果长期没有响应，服务中心就认为该服务提供者已经宕机，并将其剔除。

服务消费者会去监听相应路径，一旦路径上的数据有变化（增加或减少），zookeeper都会通知服务消费方，服务提供者的地址列已经发生改变，从而进行更新。

Zookeeper与生俱来的容错容灾能力（例如leader选举），可以确保服务注册表的高可用。

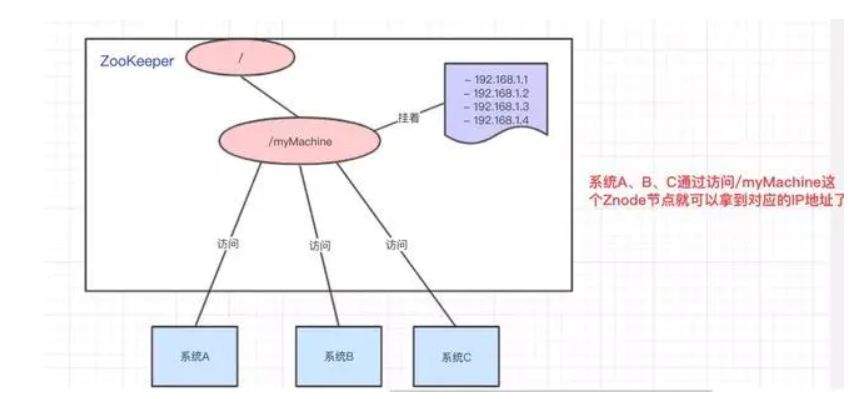
## 统一配置管理

将common.yml配置文件放在zookeeper的Znode节点中，系统A、B、C监听这个Znode节点有无变更，如果变更了，及时响应。

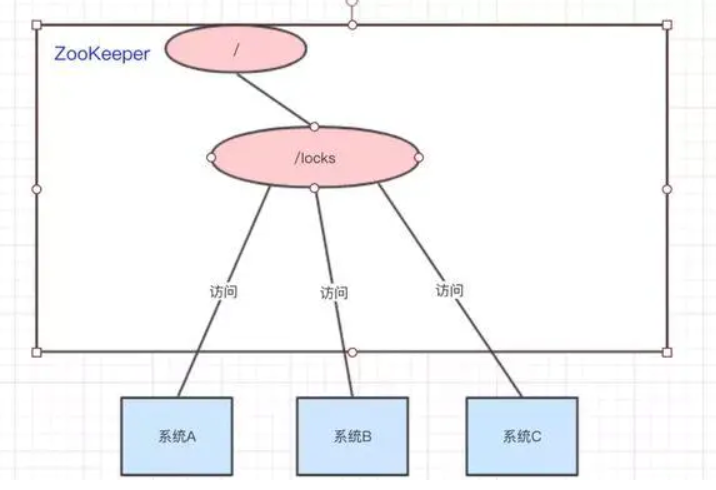


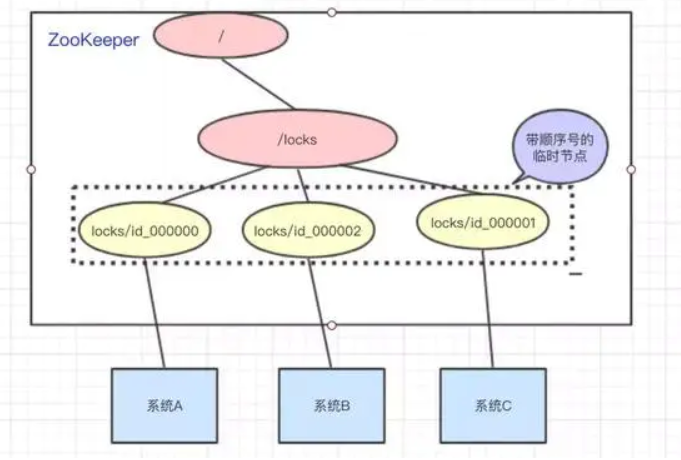
## 统一服务命名

统一命名服务的理解其实跟域名一样，是为这某一部分的资源给它取一个名字，别人通过这个名字就可以拿到对应的资源。



## 分布式锁





系统A、B、C都去访问/locks节点，访问的时候会创建带顺序号的临时节点（EPHEMERAL\_SEQUENTIAL）。比如，系统A创建了id\_000000节点，系统B创建了id\_000002节点，系统C创建了id\_000001节点。接着拿到/locks节点下的所有子节点，判断自己创建的是不是最小的那个节点：如果是，则拿到锁，执行完操作后，把创建的节点给删除掉，释放锁；如果不是，则监听比自己要小1的节点变化。

1. 系统A拿到/locks节点下的所有子节点，经过比较，发现自己是最小的，所以得到锁。
2. 系统B拿到/locks节点下的所有子节点，经过比较，发现自己不是最小的，所以监听比自己小1的节点id\_000001的状态。
3. 系统C拿到/locks节点下的所有子节点，经过比较，发现自己不是最小的，所以监听比自己小1的节点id\_000000的状态。
4. 等到系统A执行完操作以后，将自己创建的节点删除。通过监听，系统C发现id\_000000节点已经删除了，发现自己已经是最小的节点了，于是顺利拿到锁。
5. 系统B如上。

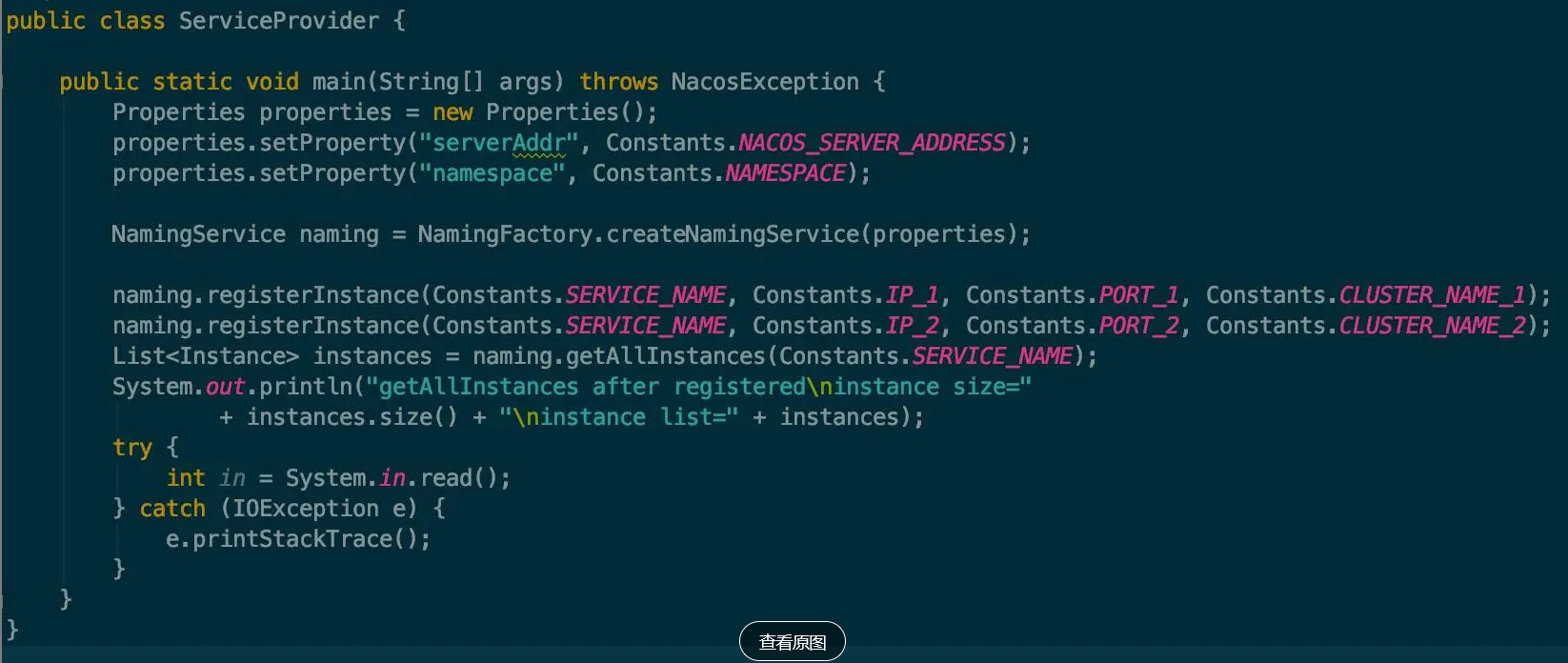
# Nacos

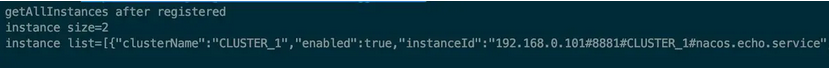


## 服务注册

通过NamingService接口的registerInstance方法就可以将服务进行注册了，该方法有很多重载的方法。

注册完后，通过调用getAllInstances方法，立即获取所有可用的实例。



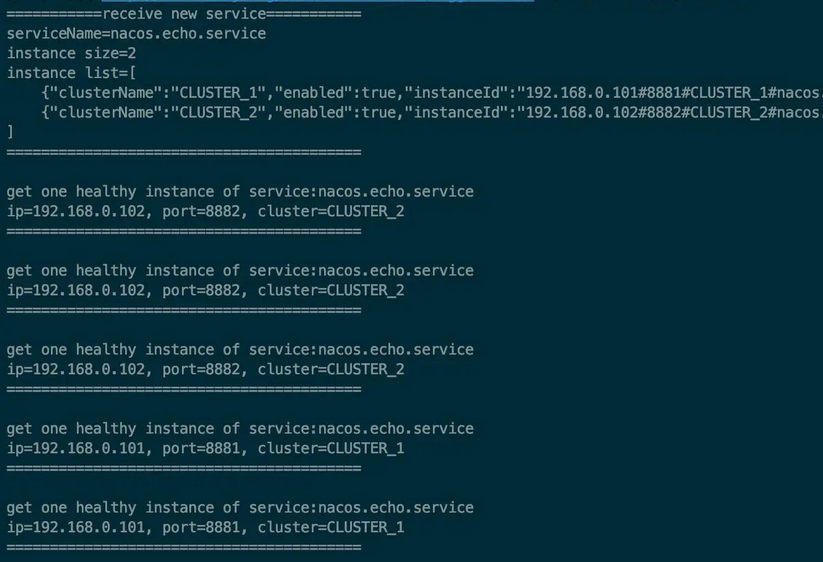


## 服务发现

服务注册之后，服务的消费者就可以向注册中心订阅自己所需要的服务了，注册中心会将所有的实例“推送”给消费者，这里将推送打上引号，原因是实际上获取服务是客户端主动轮询的，跟客户端获取配置中心的配置项原理一样。

创建一个消费者，然后向注册中心订阅一个服务，当收到注册中心返回的服务列表之后，执行5次select服务实例的操作。

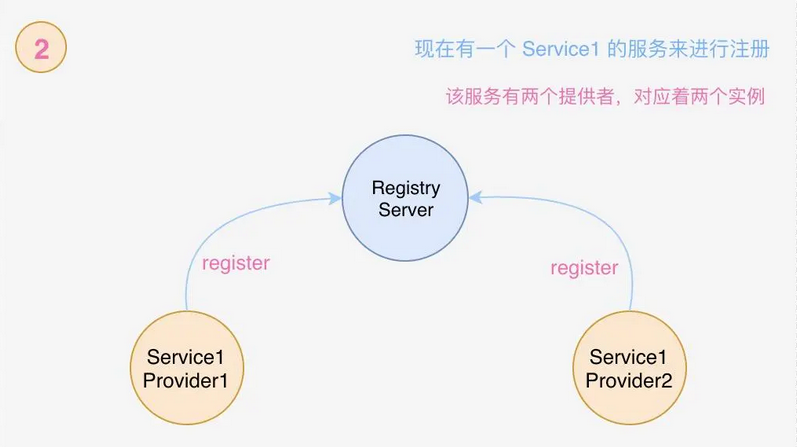


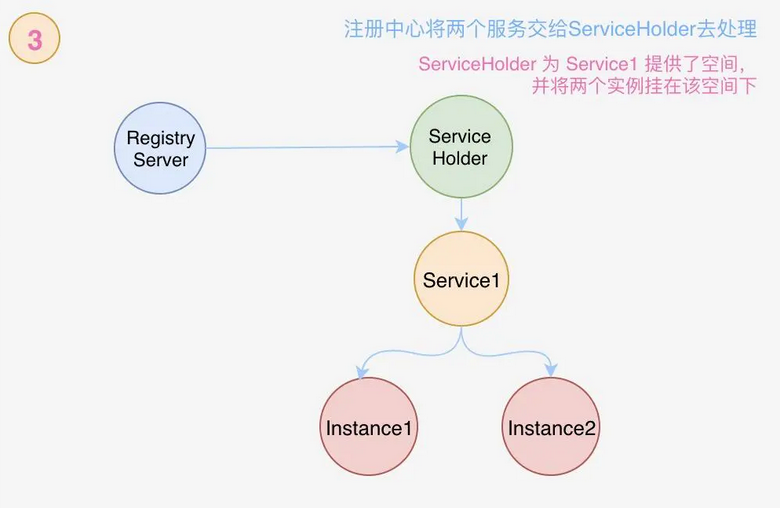


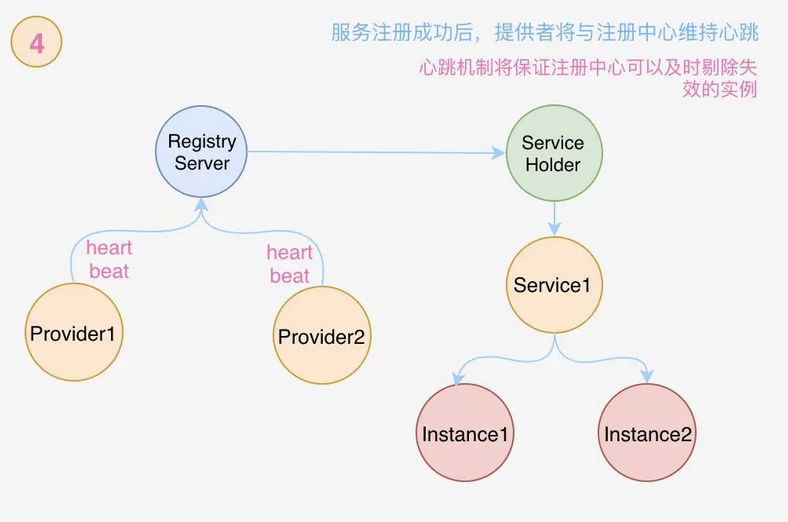
## 注册原理

1. 注册中心有一个类似于服务持有容器的管理器（Service Holder）。
2. 现有一个Service1的服务来进行注册，该服务有2个提供者者。
3. 注册中心将两个服务提供者交给Service Holder去处理，Service Holder为Service1提供命名空间，并将两个实例挂在该空间下。
4. 注册成功后，注册中心通过netty（长连接）与服务提供者维持心跳，服务提供者向注册中心定时发送心跳。



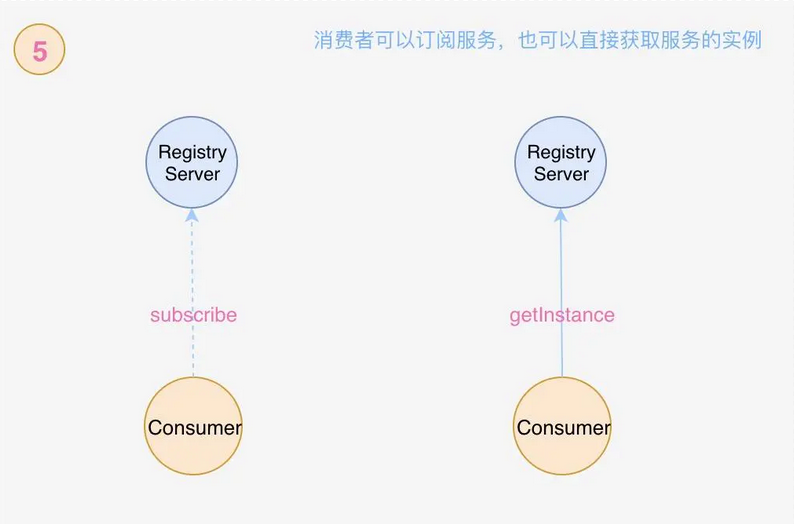


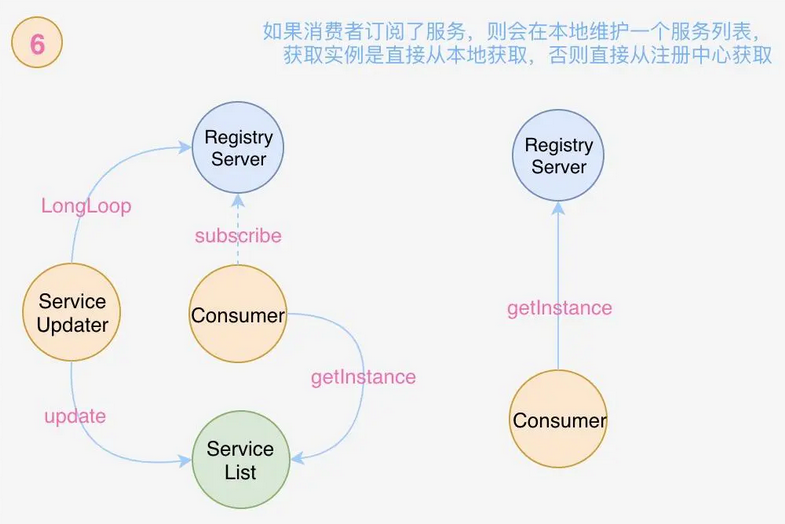




## 发现原理

1. 消费者直接向注册中心发送获取某个服务实例的请求，注册中心将返回所有可用的服务实例给消费者，但是一般部推荐这种情况。
2. 消费者向注册中心订阅某个服务，并提交一个监视器，当注册中心服务发生变化时，监听器会收到通知，这时消费者更新本地的服务实例列表，以保证所有的服务均可用。





## 负载均衡

负载均衡有很多实现方式，包括轮询法，随机方法法，对请求ip做hash后取模等等。从负载的维度考虑又分为：服务端负载均衡和客户端负载均衡。

Nacos的客户端在获取到服务的完整实例列表后，会在客户端进行负载均衡算法来获取一个可用的实例，模式使用的时随机获取的方式。



## 完整流程

Nacos客户端进行服务注册有两个部分组成，一个是将服务信息注册到服务端，另一个是向服务端发送心跳包，这俩个操作都是通过NamingProxy和服务端进行数据交互的。

Nacos客户端进行服务订阅时也有两部分组成，一个是不断从服务端查询可用服务实例的定时任务，宁一个是不断从已变服务队列中取出服务并通知EventListener持有者的定时任务。



# Consul