而是由Nacos主动发送请求，并确认对应状态。（有点像是变相的心跳响应）若其宕机，Nacos也不会剔除，仅仅标记其为**不健康服务**。

在Eureka中，服务消费者会对Eureka定时拉取 **(pull)** ，拉取更新服务列表的信息。而由于Nacos与服务提供者的不同特性，相比之下服务更新的状态时效性较差。于是Nacos中增加了新的特性。在提供者和Nacos的状态更新中，若**有实例宕机或健康状态改变，Nacos会主动推送(push)实例的变更信息。** 从而保证服务列表中状态的一致性。

但是主动推送这一特性，对服务器带来的压力会较大。所以需结合使用场景，斟酌好使用临时实例或非临时实例。

如何将服务配置为临时或 非临时实例？

修改配置文件如：

```
spring.cloud.nacos.discovery.ephemeral: flase
# 非临时实例
spring.cloud.nacos.discovery.ephemeral: true
# 临时实例
```

CAP

Eureka和Nacos在CAP模式规则设计上的差别

> CAP 定理，也称为 Brewer 定理，是分布式系统中的一个基本原则，它阐明了三个关键属性之间的固有权衡：一致性、可用性和分区容错性。CAP 定理由计算机科学家 Eric Brewer 于 2000 年提出，现已成为架构师和工程师设计分布式系统的基石概念。**这些模式描述了在分布式系统中不同的设计目标和优先级。**

* C -> 一致性（Consistency）

一致性强调在所有分布式节点具有同时的数据视图，当需要修改时，确保修改后的最新数据能同步到所有节点当中 能会导致不必要的数据加载和内存消耗。**

一次性把所有数据和服务都加载到内存当中，启动时缓慢，但是减少使用对应服务时的加载时间。

2. **惰性加载（Lazy Loading）** ：

* 惰性加载是指在需要使用对象或数据时才进行加载，延迟加载到最后可能需要使用的时候。
* 在关系数据库中，惰性加载可以通过延迟执行查询来实现，例如在 ORM 中使用延迟加载的关联对象，在需要访问关联对象数据时才真正执行查询。
* 在前端开发中，惰性加载也常用于延迟加载页面内容或组件，例如使用懒加载技术来减少初始页面加载时的资源消耗。
* **节省了不必要的资源消耗和加载时间。**
* **增加访问时的延迟。**

用到什么再加载什么，大量使用不同的未加载的服务可能导致效率非常低

当Eureka Server启动时，它会立即尝试从所有注册的服务中心获取服务实例信息，并将这些信息加载到内存中。这一方式确保在系统启动后尽快获取和管理服务实例的信息，以便其他服务可以快速地发现和调用这些服务。

在Nacos中，服务注册和发现是惰性加载的。当Nacos Server启动时，并不会立即加载所有服务实例信息到内存中，而是在有具体的服务实例注册或者查询时才会进行加载和处理。节省了不必要的资源和内存消耗。

一般来说，Eureka的饥饿加载适合于对**服务实例信息实时性要求较高**的场景，如支付服务等，与**内存资源充足**的场景，如云原生应用。

而Nacos的惰性加载则适合于对**系统资源消耗和加载时间有一定要求**的场景，如电子商城中所有的商品具体信息。