standalone 的话仅会使用 Derby，即使在 application.properties 里边配置 MySQL 也照样无视；
cluster 模式会自动使用 MySQL，这时候如果没有 MySQL 的配置，是会报错的。
防止 Nacos 宕机或重启后数据丢失,Nacos 支持将数据统一持久化到数据库 Mysql(在不配置Nacos持久化到Mysql时,
默认 Nacos 内置了一个嵌入式数据库derby,将一些数据保存到了内置的数据库上,多台 Nacos 就会出现多个内置数据库)。

1、整体架构
2.界面功能介绍，分组等功能介绍
一致性协议raft
2、cloud使用方法，拉取配置自动刷新
    按照profile 拉取配置
各种数据拉取的配置，顺序，
一次拉取多个配置文件，顺序等问题

命名空间（namespace ）、组（group）、dataid
命名空间：相当于我们的gitee上的仓库名字（默认的命名空间：public）
组：相当于我们的分支（master、dev等）（默认的组为：DEFAULT_GROUP）
dataid：相当于唯一表示（命名规范：服务名-扩展名.后缀（如:customer-dev.yml））

默认使用的是NacosPropertySourceLocator linkedhashset 顺序

可以发现，在拉取配置时会分为三步：

1. 拉取dataid为user的配置
2. 拉取dataid为user.properties的配置
3. 拉取dataid为user-${spring.profiles.active}.properties的配置

并且优先级依次增高。

拉取多个配置

一个应用可能不止需要一个配置，有时可能需要拉取多个配置，此时可以利用

1. spring.cloud.nacos.config.extension-configs[0].data-id=datasource.properties
2. spring.cloud.nacos.config.shared-configs[0].data-id=common.properties

extension-configs表示拉取额外的配置文件，shared-configs也表示拉取额外的配置文件，只不过：

1. extension-configs表示本应用特有的
2. shared-configs表示多个应用共享的

注意优先级：

- extension-configs[2] > extension-configs[1] > extension-configs[0
- shared-configs[2] > shared-configs[1] > shared-configs[0]
- 主配置 > extension-configs > shared-configs

临时实例与持久实例

默认情况下，注册给nacos的实例都是**临时实例**，临时实例表示会通过客户端与服务端之间的心跳来保活，默认情况下，客户端会每**隔5s发送一次心跳**。

```java
public static final long DEFAULT_HEART_BEAT_INTERVAL = TimeUnit.SECONDS.toMillis(5);
```

在服务端测，如果**超过15s没有收到客户端的心跳，那么就会把实例标记为不健康状态**

```java
public static final long DEFAULT_HEART_BEAT_TIMEOUT = TimeUnit.SECONDS.toMillis(15);
```

在服务端测，如果**超过30s没有收到客户端的心跳，那么就会删除实例**

```java
public static final long DEFAULT_IP_DELETE_TIMEOUT = TimeUnit.SECONDS.toMillis(30);
```

而对于持久实例，就算服务实例下线了，那么也不会被删除，我们可以通过：

```java
spring.cloud.nacos.discovery.ephemeral=false
```

来配置为持久实例，表示实例信息会持久化到磁盘中去。

那什么时候用持久实例呢？我们可以发现持久实例与临时实例的区别在于，持久实例会永远在线，而临时实例不会，所以如果消费端在某种情况下想拿到已经下线的实例的实例信息，那么就可以把实例注册为持久实例。

保护阈值

在使用过程中，我们可以设置一个0-1的一个比例，表示如果服务的所有实例中，健康实例的比重低于这个比重就会触发保护，一旦触发保护，在服务消费端侧就会把所有实例拉取下来，不管是否健康，这样就起到了保护的作用，因为正常来说消费端只会拿到健康实例，但是如果健康实例占总实例比例比较小了，那么就会导致所有流量都会压到健康实例上，这样仅剩的几个健康实例也会被压垮，所以只要触发了保护，消费端就会拉取到所有实例，这样部分消费端仍然会访问到不健康的实例从而请求失败，但是也有一部分请求能访问到健康实例，达到保护的作用。

在SpringCloud Tencent中，这个功能叫“全死全活”。

给8070这个实例设置了权重为2，这样它的权重就是8071的两倍，那么就应该要承受2被的流量。

不过我们在消费一个服务时，通常是通过ribbon来进行负载均衡的，所以默认情况下nacos配置的权重是起不到作用的，因为ribbon使用的是自己的负载均衡策略，而如果想要用到nacos的权重，可以：

Java | 复制代码

```java
@Bean
public IRule ribbonRule() {
    return new NacosRule();
}
```

这样就会利用到nacos中所配置的权重了。

```java
@SpringBootApplication
public class MyApplication {

    @Bean
    @LoadBalanced
    public RestTemplate restTemplate(){
        return new RestTemplate();
    }

    @Bean
    public IRule ribbonRule() {
        return new NacosRule();
    }
```