1. Client 与 Server 有两个重要的交互，服务注册与心跳发送

2. C 向 S 注册一个服务，继而以执行任务的模式向 S 发送一次心跳，心跳包需要带上 C 的全部信息，站 在 C 的角度 S 集群所有的节点对等，所以请求 s 集群中的节点是随机的

3. C 如果请求失败则换一个 S 的节点重新发送请求

4. S 集群的任意一个节点都存储所有数据 ,但每个节点只负责其中一部分服务 ,在接收到 C 的“写”(注册、心跳、下线等) 请求后，s 节点判断请求的服务是否为自己负责，如果是，则处理，否则交由负责的节点处理

5. 每个 S 节点主动发送健康检查到其他节点，响应的节点被该节点视为健康节点

6. S 在接收到 C 的心跳后，如果该服务不存在，则将该心跳请求当做注册请求来处理

7. S 如果 15s 没有收到 C 心跳，则把该 C 标记为不健康；如果 30S 没有收到心跳，则下线该 C

8. 负责的 S 节点在接收到 C 的服务注册、服务心跳等写请求后将数据写入后即返回，后台异步地将数据同步给其他节点

9. s 节点在收到读请求后直接从本机获取后返回，不管数据是否为最新

com.alibaba.nacos.naming.consistency.ephemeral.distro.DistroConsistencyServiceImpl#put
taskDispatcher.addTask(key);

10. 任意一个 s 节点在启动的时候会去判断当前 s 集群当中的节点是否大于 1；如果大于 1 则会去执行远程同步；否则会让线程睡眠 1s 继续判断集群节点是否大于 1

11. C 在启动的时候 (spring 容器启动的时候会实例化 NacosAutoServiceRegistration，会实例化 NacosServiceRegistry 构造方法里面会调用 namingServiceInstance--->createNamingService---->createNamingService-->new NacosNamingService---构造方法--- init----xxxx----udp 等待 s 发送数据) 会建立一个 UDP 的接受方；C 在向 S 获取服务列表的时候 S 会收集到 C 的 UDP 信息；以后当 S 更新的时候会把更新信息推送给 C；但是 s 默认不发这个 c 的？因为 s 不知道这个 c 需要数据---

12. C 在第一次想 S 获取服务列表的时候 ,会开启一个任务每隔 10s 钟会去 S 那一次最新的服务列表，同时还会把当前 c 的所有信息（ udp ）s 接受到信息之后把 c 添加到发送 upd 的列表当中

13. distro 判断一个请求属于 S 集群中的那个节点负责用的是一个非常简单的 hash 算法(int target = distroHash(serviceName) % healthyList.size();) ，假设 s 某个节点挂了，则会重新计算，非常蛋疼，其实可以采用一致性的 hash 算法；这样只会重新计算以前分发到这个挂掉的机器上的请求；不会全部计算