一、设计注册中心

1、高可用

99.99% 集群

2、高并发

系统能够同时处理请求多少

垂直扩展： 增加服务器的性能

水平扩展：增加实例

3、高性能

程序处理的速度

数据存储结构、访问机制、 集群同步方式 cp ap

三 nacos

二、源码分析

1、准备代码

2、看源码

2.1 找入口

怎样找入口，找自动装配类 spring.factories

观察者模式

事件驱动

注册

2.2 注册

判断变量 1、 debug 2、 全文搜索 定位赋值位置

客户端和服务端进行通信的时候，一般是通过请求参数关联客户端和服务端 对应类 如果客户端是 InstanceRequest 那服务端处理就应该是 InstanceRequestXXX

msb-order 注册到注册中心 注册的是服务 还是实例？

msb-order 服务 msb-order -1 msb-order -2

key：命名空间 value：set 对应服务

```
 namespaceSingletonMaps.computeIfAbsent(result.getNamespace(), (namespace) -> new
ConcurrentHashSet<>())
```

思考： Client ConcurrentHashMap<Service, InstancePublishInfo>

Nacos源码中大量利用观察者模式（事件驱动）

2.3 注册表

性能高： grpc 、注册表结构相对于1.4 更加简单

ConcurrentMap<Service, Set> publisherIndexes

2.4 服务发现

通过debug 我们发现ribbon 调用.NacosNamingService#selectInstances 获取实例

第一步从缓存拿 ？ 那这个缓存一定地方更新？

启动一个定时任务每6秒更新一次 最长时间60秒

更新过程：首先更新缓存、然后更新磁盘 ，这就是nacos崩溃了，我们服务还能用的原因

服务启动的时候读取磁盘内容。