Nacos致力于解决微服务中的统一配置、服务注册与发现等问题。它提供了一组简单易用的特性集，帮助开发者快速实现动态服务发现、服务配置、服务元数据及流量管理。

服务发现和服务健康监测

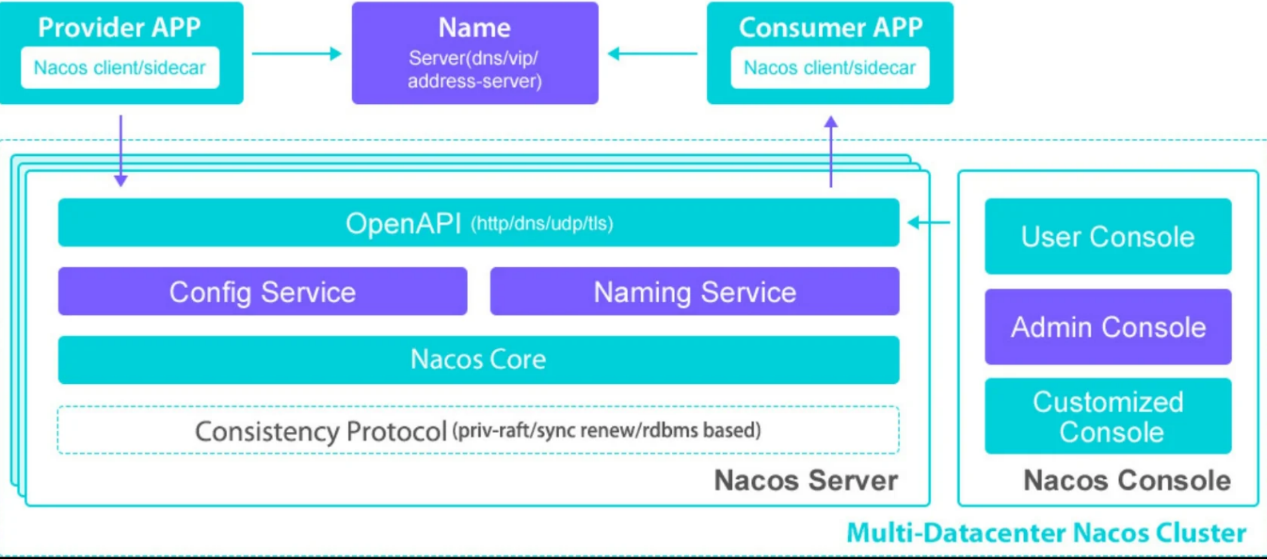
Nacos支持基于DNS和基于RPC的服务发现。服务提供者使用原生SDK、OpenAPI或一个独立的Agent TODO注册service后，服务消费者可以使用DNS或HTTP&API查找和发现服务。

Nacos提供对服务的实时的健康检查，阻止向不健康的主机或服务实例发送请求。Nacos支持传输层（PING或TCP）和应用层（HTTP、MYSQL、用户自定义的健康检查）。对于复杂的云环境和网络拓扑环境中（如VPC、边缘网络等）服务的健康检查，Nacos提供了agent上报和服务端主动检测两种健康检查机制。

动态配置服务

业务服务一般都会维护一个本地配置文件，然后把一些常量配置在这个文件中。这种方式在某些场景中会存在问题，比如配置需要变更时要重新部署应用。而动态配置服务可以中心化、外部化和动态化的方式管理所有环境的应用配置和服务配置，可以是配置管理变得更加高效和敏捷。配置中心化管理让实现无状态服务变得更简单，让服务按需弹性扩展变得更容易。

Nacos架构图：



Provider app：服务提供者

Consumer app：服务消费者

Name server：通过VIP（vritual IP）或者DNS的方式实现nacos高可用集群的服务路由

Nacos server：nacos服务提供者，里面包含的open api是功能访问的入口，config service、naming service 是nacos提供的配置服务，名字服务模块。Consistency protocol是一致性协议，用来实现nacos集群节点的数据同步，这里使用的是raft算法（使用类似算法的中间件还有Etcd、Redis烧饼选举）

Nacos console：nacos控制台

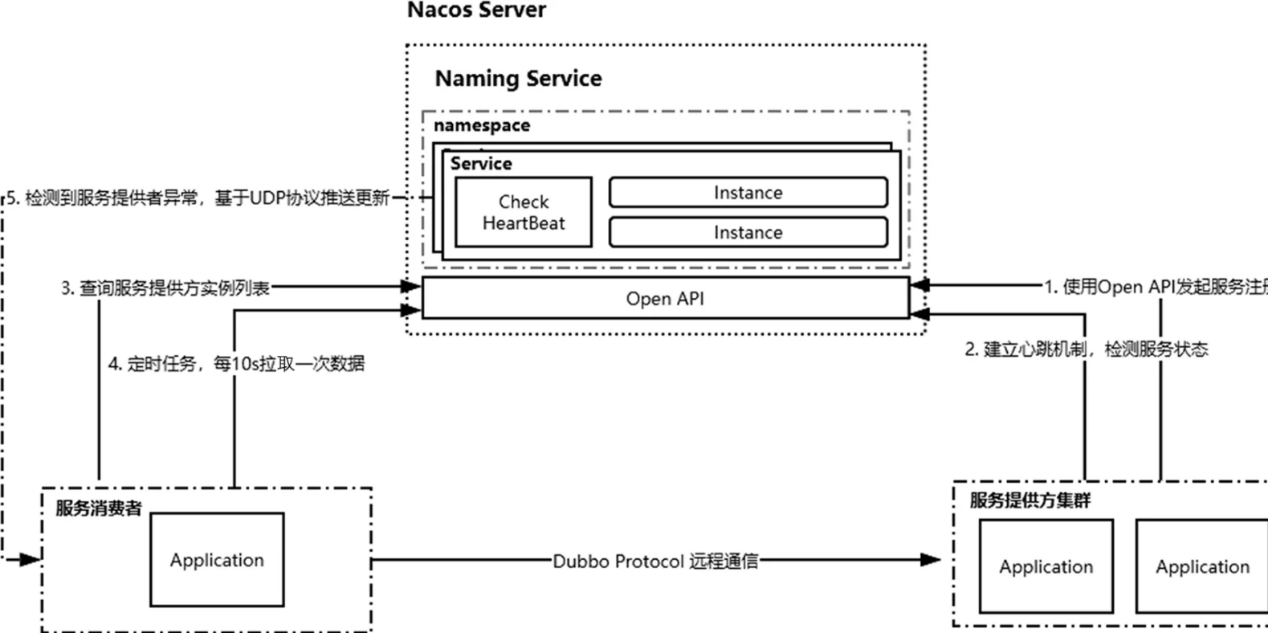
整体来说，服务提供者通过VIP访问nacos server高可用集群，基于Open API完成服务的注册和服务的查询。Nacos server本身可以支持主备模式，所以底层会采用数据一致性协议算法来完成从节点的数据同步。服务消费者也是如此，基于Open API从Nacos Server中查询服务列表。

注册中心原理

服务实例在启动时注册到服务注册表，并在关闭时注销。

服务消费者查询服务注册表，获得可用实例

服务注册中心需要调用服务实例的健康检查api来验证它是否能够处理请求。



服务注册完整过程

Nacos客户端通过Open API 的形式发送服务注册请求。

Naocs服务端收到请求之后，做以下三件事：

1. 构建一个Service对象保存到ConcurrentHashMao集合中
2. 使用定时任务对当前服务下的所有实例建立心跳检测机制
3. 基于数据一致性协议将服务数据进行同步

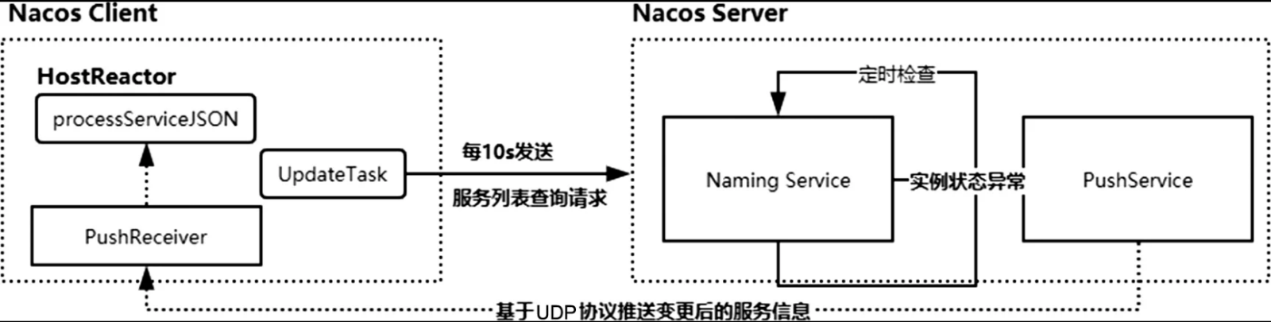
服务动态感知：

Nacos客户端有一个HostReactor类，他的功能是实现服务的动态更新，基本原理是：

客户端发起事件订阅后，在HostReactor中有一个UpdateTask线程，每10s发送一次Pull请求，获得服务端最新的地址列表。

对于服务端，它和服务提供者的实例之间维持了心跳检测，一旦服务提供者出现异常。则会发送一个push消息给nacos客户端，也就是服务消费者。

服务消费者收到请求之后，使用hostReactor中提供的processServiceJSON解析消息，并更新本地服务地址列表





Namespace 用于解决多环境及多租户数据的隔离问题，比如在多套环境下，可以根据不同的namespace，实现多环境的隔离，或者在多用户场景中，每个用户维护自己的namespace，实现每个用户的配置数据和注册数据的隔离。需要注意的是，在不同namespace下，可以存在相同的group和dataid。

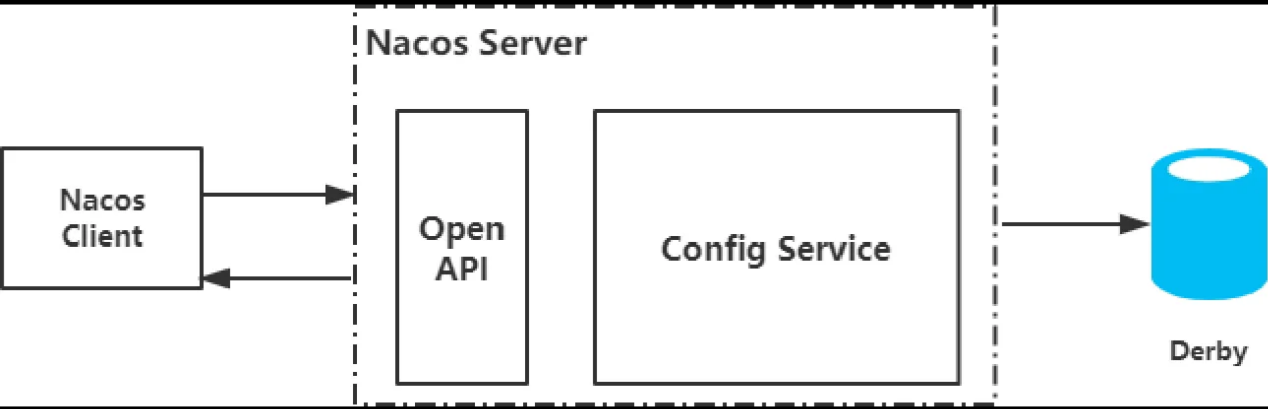
Group是nacos中用来是实现Data ID分组管理的机制

官方建议，通过namespace来区分不同的环境，而group可以专注在业务层面的数据分组。

配置的crud

nacos config，就是提供配置的集中式管理。然后对外提供CRUD的访问接口。

对于服务端来说，就是配置如何存储，以及是否需要持久化，对于客户端来说，就是通过接口从服务端查询到相应的数据，然后返回。



动态监听：

客户端和服务端之间进行数据交互：pull、push

Pull 表示客户端从服务端主动拉数据

Push表示服务端主动把数据推送到客户端

Push模式，服务端需要维持与客户端的长连接，如果客户端数量较多，那么服务端需要耗费大量的内存资源来保存每个连接，并且为了检测链接有效性，还需要心跳机制来维持每个连接的状态

Pull模式，客户端需要定时从的服务端拉数据，定时任务存在间隔，所以不能保证数据的实时性。并且在服务端配置长时间不更新的情况下，客户端的定时任务会做一些无效的pull

Nacos采用的是pull模式，但并不是简单的pull，而是一种长轮询机制，他结合push和pull两者的优势。客户端采用长轮询的方式定时发起pull请求，去检查服务端配置信息是否发生了变更，如果发生了变更，则客户端会根据变更的数据获取最新的配置。所谓长轮询，是客户局端发起轮询请求之后，服务端如果有配置发生变更，就直接返回



如果客户端发起pull请求后，发现服务端的配置和客户端的配置保持一致，那么服务端会先hold住这个请求，也就是服务端拿到这个连接之后再指定的时间段内一直不返回结果，直到这段时间内配置发生改变，服务端会把原先hold住的请求进行返回。

Nacos服务端收到请求之后，先检查配置是否发生变更，如果没有，则设置一个定时任务，延期29.5s执行，并且把当前客户端长轮询加入allSubs队列，这时候有两种方式触发该连接结果的返回：

1. 在等待29.5s后触发自动检查机制，这个时候不管配置有没有发生变化，都会把结果返回客户端。而29.5s就是这个长连接保持的时间。
2. 在29.5s内任意一个时刻，通过nacosDashboard或者api的方式对配置进行了修改，这会触发一个事件机制，监听到该事件的任务会便利allSubs队列，找到发生变更的配置项对应的clientLongPolling任务，将变更的数据通过该任务中的链接进行返回，就完成了一次“推送”操作

这样既能保证客户端实时感知配置的变化，也降低服务端的压力。其中，这个长连接的会话超时时间默认是30s

