（3）Config Service 和 Admin Service 都是多实例、无状态部署，所以需要将自己注册到 Eureka 中并保持心跳。

（4）在 Eureka 之上构建了一层 Meta Server，用于封装 Eureka 的服务发现接口。

（5）Client 通过域名访问 Meta Server 获取 Config Service 服务列表（IP + Port），而后直接通过 IP + Port 访问服务，同时在 Client 侧会负载均衡和错误重试。

（6）Portal 通过域名访问 Meta Server 获取 Admin Service 服务列表（IP + Port），而后直接通过 IP + Port 访问服务，同时在 Portal 侧会做负载均衡和错误重试。

（7）为了简化部署，实际使用中会把 Config Service、Eureka 和 Meta Server 这 3 个逻辑角色部署在同一个 JVM 进程中。

学习完 Apollo 的各个层次，轮廓就清晰多了，我们了解 Apollo 拥有 7 个模块，其中 4 个模块是和配置功能相关的核心模块：Config Service、Admin Service、Client 和 Portal，另外 3 个模块是辅助服务发现的模块：Eureka、Meta Server 和 NginxLB。欲了解这 7 个模块的细节，读者可以参照该项目的源码。

在应用的配置方面，Apollo 支持 4 个维度管理 Key-Value 格式的配置：

（1）application（应用）：这个很好理解，就是实际使用配置的应用，Apollo 客户端在运行时需要知道当前应用是谁，从而可以去获取对应的配置；每个应用都需要有唯一的身份标识 appID，应用默认的身份是跟着代码走的，所以需要在代码中配置。

（2）environment（环境）：配置对应的环境，Apollo 客户端在运行时需要知道当前应用处于哪个环境，从而可以去获取应用的配置；环境和代码无关，同一份代码部署在不同的环境就应该能够获取到不同环境的配置；所以环境默认是通过读取机器上的配置（server.properties 中的 env 属性）指定的，不过为了开发方便，同时也支持运行时通过 System Property 来指定。

（3）cluster（集群）：一个应用下不同实例的分组，比如典型的可以按照数据中心来分，把上海机房的应用实例分为一个集群，把北京机房的应用实例分为另一个集群。对不同的 cluster，同一个配置可以有不一样的值，如 zookeeper 地址。集群默认是通过读取机器上的配置（server.properties 中的 idc 属性）指定的，不过也支持运行时通过 System Property 来指定。

（4）namespace（命名空间）：一个应用下不同配置的分组，可以简单地把 namespace 类比为文件，不同类型的配置存放在不同的文件中，如数据库配置文件、RPC 配置文件、应用自身的配置文件等；应用可以直接读取到公共组件的配置 namespace，如 DAL，RPC 等；应用也可以通过继承公共组件的配置 namespace 来对公共组件的配置做调整，如 DAL 的初始数据库连接数。

我们可以根据具体的业务场景，创建对应的维度来管理应用的配置信息。