1. **服务提供方与调用方接口依赖方式太强：我们为每个微服务定义了各自的service抽象接口，并通过持续集成发布到私有仓库中，调用方应用对微服务提供的抽象接口存在强依赖关系**，因此不论开发、测试、集成环境都需要严格的管理版本依赖，才不会出现服务方与调用方的不一致导致应用无法编译成功等一系列问题，以及这也会直接影响本地开发的环境要求，往往一个依赖很多服务的上层应用，每天都要更新很多代码并install之后才能进行后续的开发。若没有严格的版本管理制度或开发一些自动化工具，这样的依赖关系会成为开发团队的一大噩梦。而REST接口相比RPC更为轻量化，服务提供方和调用方的依赖只是依靠一纸契约，不存在代码级别的强依赖，当然REST接口也有痛点，因为接口定义过轻，很容易导致定义文档与实际实现不一致导致服务集成时的问题，但是该问题很好解决，只需要通过每个服务整合swagger，让每个服务的代码与文档一体化，就能解决。所以在分布式环境下，REST方式的服务依赖要比RPC方式的依赖更为灵活。

2. 服务对平台敏感，难以简单复用：通常我们在提供对外服务时，都会以REST的方式提供出去，这样可以实现跨平台的特点，任何一个语言的调用方都可以根据接口定义来实现。那么在Dubbo中我们要提供REST接口时，不得不实现一层代理，用来将RPC接口转换成REST接口进行对外发布。若我们每个服务本身就以REST接口方式存在，当要对外提供服务时，主要在API网关中配置映射关系和权限控制就可实现服务的复用了。

Dubbo实现了服务治理的基础，但是要完成一个完备的微服务架构，还需要在各环节去扩展和完善以保证集群的健康，以减轻开发、测试以及运维各个环节上增加出来的压力，这样才能让各环节人员真正的专注于业务逻辑