**摘要:** 【问题】 近年来，随着互联网行业的迅猛发展，公司或组织业务的不断扩张，需求的快速变化以及用户量的不断增加，传统的单块（Monolithic）软件架构面临着越来越多的挑战，已逐渐无法适应互联网时代对软件的要求。在这一背景下，微服务架构模式（Microservice Architecture Pattern）逐渐流行，它强调将单一业务功能开发成微服务的形式，每个微服务运行在一个进程中；采用HTTP等通用协议和轻量级API实现微服务之间的协作与通信。这些微服务可以使用不同的开发语言以及不同数据存储技术，能够通过自动化部署工具独立发布，并保持最低限制的集中式管理。 请围绕“论微服务架构及其应用”论题，依次从以下三个方面进行论述。 1．概要叙述你参与管理和开发的、采用微服务架构的软件开发项目及在其中所担任的主要工作。 2．与单块架构相比较，微服务架构有哪些特点？请列举至少4个特点并进行说明。 3．结合你参与管理和开发的软件开发项目，描述该软件的架构，说明该架构是如何采用微服务架构模式的，并说明在采用微服务架构后，在软件开发过程中遇到的实际问题和解决方案。 【摘要】 2017年1月，我司承接某视频会议云直播项目。该项目主要为了解决视频会议系统对用户终端硬件要求过高、入会用户数量受限的问题，通过增加云直播的方式，将云直播与视频会议对接，以支持更多用户在电脑网页和手机端观看会议实况。我在项目中担任技术负责人，负责项目服务划分、架构设计和技术指导工作。本文结合我的亲身实践，论述了微服务架构在项目中的应用。在服务分层设计阶段，以客户需求为依据，对系统功能进行总体服务分层架构设计，各层各施其职。在服务开发阶段，对较粗粒度的服务进一步细化成粒度更小微服务，并采用开源组件框架进行有效组装，完成项目微服务化开发。在部署阶段，采用容器化技术，完成微服务的容器化部署和集群管理。通过上述过程的实施，项目最终顺利交付。

互联网直播业务在2012年至2016年呈现快速增长，这种业务形态对传统的视频会议行业带来了新的机会。传统视频会议常常需要用户购买价格不菲的设备，且支持的同时在线的用户量级别只能在百级～千级，而直播方式支持同时观看用户量都在十万级以上。  
我所在公司的产品线深耕视频会议行业多年，并于2017年1月，中标某视频会议云直播项目。需求的出发点是对接视频会议系统，构建一个云直播平台，提供会议直播、直播监控、录制存储、点播等能力，对用户提供在网页、手机上观看会议直播或回放等功能。视频会议是我司已有的产品形态，此时我们需要基于现有的技术积累，开发出支持直播的流媒体服务、录制存储服务，还需要全新开发上层业务、Web端和手机客户端。不同的服务相互独立，且采用的开发语言或数据库有所不同，这意味着既需要进行服务化划分，还需要通过某种机制实现服务间通讯，打通全业务流程。同时，该项目还需要以云服务SaaS的方式运营，服务需要容易部署和扩展。项目启动后，我在项目中担任技术负责人的角色，负责服务划分、架构设计、技术指导等工作。  
在项目架构选型上，我根据自己多年的工作经验，选用了微服务架构。相对传统的单体架构，微服务架构在支持技术异构、各服务组件独立开发、动态扩容、独立部署等方面都有很多优点。2010年，我们曾实施过一个项目，当时采用的是单体架构，所有的服务都集中在一个工程进行开发，虽然功能上都能正常开发实现，但后来发现存在较多问题，一是服务之间耦合度非常大，项目臃肿难以维护；二是技术异构难，整个项目只能采用单一的开发语言、单一数据库；三是服务横向扩容难，当用户访问量增加时，单体架构无法很好扩容；四是项目不能实现轻量化部署，即使更新一张图片，也要整体发布。根据这些经验考虑，当前视频会议云直播项目，我采用微服务化架构。一是此项目涉及功能组件较多，如有视频会议服务、直播流媒体服务、录制服务、文件存储服务、直播监控服务、在线点播服务以及其他上层服务等，不同组件技术异构大，采用微服务架构可以解决技术异构问题。二是各业务团队专注于某业务领域的开发，通过约定轻量级通讯接口，实现服务通讯。三是各微服务可以单独部署、互不影响。四是支持动态扩容。  
下面从服务分层设计、微服务开发、微服务部署三个阶段分别论述系统的微服务架构落地过程。  
一、在服务分层设计阶段，以客户需求为依据，对系统功能进行总体服务分层架构设计，各层各施其职。  
为了避免后期不同服务间重复处理某些业务逻辑，我们进行了业务流分析和分层设计。从用户基本需求分析业务流程，视频会议开始前，系统会提前下发通知给用户；视频会议系统调用直播媒体服务器接口，开启直播并推流；直播服务器开启直播，再调用录制服务接口开启录制，并把码流保存到分布式存储系统；用户在预定的时间点击链接或打开客户端登录后观看直播，鉴权成功后进入某个直播间；在直播过程中，客户端收集直播性能指标并上报服务；直播结束后支持点播。为了有效贯穿所以业务流，我把整个系统自上而下划分为五层：前端层、对外网关层、业务层、核心计算层、存储层。前端层涉及WEB前端、手机客户端，提供人机交互。对外网关层实现负载均衡、路由、鉴权、日志记录。业务层涉及用户管理、会议管理、直播间管理、监控管理等基础服务。核心计算层包括直播、录制存储，媒体点播等服务。存储层包括数据库、分布式缓存以及分布式文件系统。服务分层设计，可以减少重复开发，降低服务开发复杂度。以鉴权功能为例，如果后端各服务每次都要调用鉴权，效率就低了，我把它提到对外网关层。  
二、在服务开发阶段，对较粗粒度的服务进一步细化成粒度更小微服务，并采用开源组件框架进行有效组装，完成项目微服务化开发。  
在上述分层服务化架构确定后，我对某些粗粒度的服务进一步划化。比如业务层细分成用户管理、会议管理、直播间管理、监控管理，提问管理等，不同的微服务由不同的开发小组独立完成。有些微服务适合采用NoSQL分布式数据库的选用NoSQL数据库，有些微服务采用MySQL关系型数据库，互不影响。微服务化之后，还需要把各个微服务有效组装起来，使得它们可以相互协作，彼此通过轻量通讯框架调用，这时就需要引入微服务治理的框架，当时选用的是Spring Cloud，此时增加服务的注册中心、配置中心、微服务网关。在使用Spring Cloud过程中发现了一些性能问题，因为Spring Cloud实现微服务间通讯是采用RESTful，即走的是HTTP协议，某些接口调用频繁，性能较差。后来引入了Dubbo开源框架配合使用解决这个问题。Dubbo采用RPC通讯，比RESTful性能更高，用在微服务内部；Spring Cloud可以提供RESTful服务，则用到与其他异构服务通讯和对前端提供服务。  
三、在部署阶段，采用容器化技术，完成微服务的容器化部署和集群管理。  
微服务化后的系统明显解决了单体架构的重度耦合，难以扩展的问题。但拆分出来的微服务过多，且每个都是独立部署，这就增加系统维护复杂性。如果手工部署明显工作量较大。为了解决这个问题，我采用了Docker容器化技术。公司内部提供了容器化的测试环境、各开发团队检入的代码会触发自动化构建脚本，自动构建生成Docker镜像，并更新到仓库和部署到测试环境中运行。这样，测试人员就可以针对该改动进行测试。版本经测试人员严格测试通过后，再一键部署到生产环境。在服务容器化实施过程中，我们遇到的主要问题是IP映射和服务间通讯地址的问题。比如，直播媒体服务的端口需要直接暴露给客户端访问，如果某个媒体服务发生宕机，需要及时启动其他容器，同时还需要通过某些机制，保证客户端能发现故障并平滑切换到新的IP和端口上。而服务间通讯也需要解决跨宿主机通讯的问题。后来，我们采用K8S实现把容器组成集群的环境进行管理，动态地启动服务和绑定对外端口；也解决了跨宿主机微服务间通讯的问题。  
整个项目划分成多个团队协助、迭代开发。通过10个多月有效实施，项目于2017年11月整体上线运行，顺利交付。运行在阿里云上生产环境的虚拟主机达10台，微服务个数50个。且支持根据业务变化动态启动微服务实例。整个系统稳定，项目获得客户的认可和领导的赞赏。  
但在系统实施过程中，也遇到一些问题。例如直播监控功能是通过分析日志及时发现客户端播放存在的异常，包括延迟过大、播放卡顿等。在客户端我们埋点记录异常信息，并上报后台服务保存；在服务端所有服务的运行期间的异常日志都会保存到文件。当时我们采用定时任务分析日志，但后来发现当访问量陡增时，定时任务出现严重滞后的现象，即使微服务扩容，获得的效果也不明显。最终我们在大数据部门的支持下，引入数据采集引擎Flume和Spark Streaming实时计算引擎进行了完美解决，实现直播高效的故障告警和定位，效果很好。