视听媒体微服务技术架构规范

1 范围

本文件规定了视听媒体微服务的技术架构及相关组成部分的功能要求。

本文件适用于视听媒体微服务架构系统的设计、研发、建设、运行和维护。

2 规范性引用文件

本文件没有规范性引用文件。

3 术语和定义

下列术语和定义适用于本文件。

3.1

微服务 microservice

可独立部署，并提供可实现某应用中特定功能的服务的制品。

[来源：GB/T 42568—2023，3.1.3]

3.2

技术架构 technology architecture

整个或部分技术系统的可重复设计，表现为一组抽象构件及构件实例间交互的方法。

[来源：GB/T 43335—2023，3.1.4]

3.3

微服务架构 microservice architecture

一种软件开发的架构和组织方法，其中软件由若干小型独立的服务组成，这些服务通过定义的应用程序编程接口进行通信。

注：该架构以治理微小粒度服务群组的形式来实现应用程序，通过服务发现、调用和协同的方式来完成业务构建和能力服务。该架构能支持业务快速构建和组装编排，使应用程序更易于扩展和迭代升级，并提供了多样化的技术选型，极大提升了开发人员的效率。

3.4

视听媒体微服务架构 audiovisual media microservice architecture；MMA

面向视听媒体，以微服务为技术手段的系统构建方式，针对视听媒体相关的复杂应用场景，在系统研发、部署及维护中，采用微小化、分布式的手段向用户提供服务，从而避免随着业务场景不断增多而造成系统难以维护和升级的问题。

[来源：ITU-T J.1306:2023，3.1.9]

3.5

组件 component

可以复用的软件组成部分（如一组相互关联的微服务、软件应用的某些复用部分等），也称作构件。

3.6

服务网格 service mesh

一组处理服务间大量进程以及相互网络通信的代理组件和任务管理组件。

注：代理组件处理入站和出站数据包，任务管理组件控制流量，保障服务之间复杂调用的可靠性和易用性。

[来源：ITU-T J.1306:2023，3.2.12]

3.7

容器 container

一套软件，用于提供隔离、资源控制和可移植性应用程序的虚拟化处理。

[来源：ITU-T Y.3535:2022，3.2.1]

3.8

容器编排 container orchestration

对容器的部署和组织，提供调度和管理容器集群的能力，包括容器自动化部署、管理、弹性伸缩和容器网络管理等。

[来源：ITU-T J.1306:2023，3.2.3]

3.9

开发运维一体化 development and operations；DevOps

一组过程、方法与系统的统称，用于促进开发（应用程序/软件工程）、技术运维和质量保障部门之间的沟通、协作与整合。

[来源：ITU-T J.1306:2023，3.2.6]

3.10

持续集成与发布 continuous integration and continuous deployment；CI/CD

一种帮助团队成员频繁集成和发布其工作成果的软件开发实践方法。

注：持续集成的每次集成都会经过自动检验，以尽快发现集成错误，缩短系统开发生命周期。持续发布能够自动将已经验证的代码发布到存储库，从而建立可以随时部署到生产环境的代码库。持续集成与发布在很大程度上需要依赖精细设计的自动化测试。

[来源：ITU-T J.1306:2023，3.2.4]

3.11

API网关 API gateway

在服务端实现的、对服务调用者提供统一接入管理的系统，对外部调用提供了统一的出入口，屏蔽了内部服务的实现机制。

3.12

第三方微服务模块 third party microservice component

通过容器镜像运行的方式，直接在MMA中进行管理和调用，由异构系统实现的微服务或微服务模块在纳入MMA治理时的称谓。

3.13

适应度函数 fitness function

用于计算潜在解决方案与既定目标差距的一种目标函数。

注：在演化计算中，可决定一个算法是否在持续提升。

[来源：ITU-T J.1306:2023，3.2.7]

3.14

数据仓库 data warehouse

在数据准备之后用于永久性存储数据的数据库。

[来源：GB/T 35295—2017，2.1.35]

3.15

数据湖 data lake

存储来自多个数据源、多种数据类型的原始数据，数据无需经过结构化处理，就可以进行存取、处理、分析和传输，集中存储各类结构化和非结构化数据的一个大型数据仓库。

[来源：ITU-T J.1306:2023，3.2.5]

3.16

灰度发布 grayscale release

一种软件产品在生产环境安全上线、平滑过渡的迭代发布方式。

注：灰度发布保证整体系统的稳定，能够在初始阶段发现问题，减少影响的范围。

3.17

框架 framework

被应用开发者定制的应用骨架，遵照架构实施实现，包括一系列供开发者选用、完成系统实施的组件。

3.18

开放应用模型 open application model

一种用于描述和规范应用程序的范式。

注：开放应用模型在应用的生命周期内，通过提供规范的沟通方式，将应用开发者、应用运维人员和基础设施运维人员以一种标准化的方式连接起来，从而让云原生应用的开发、交付和运维变得更加简洁、高效并且可控。

[来源：ITU-T J.1306:2023，3.2.11]

3.19

低代码开发平台 low-code development platform

无需编程或通过少量代码就能快速生成应用程序的开发平台。

注：低代码开发平台通过可视化的方式进行应用程序开发，使开发人员可以通过图形化的用户界面，使用拖拽组件和模型驱动的逻辑来创建各类应用程序。

[来源：ITU-T J.1306:2023，3.2.10]

4 缩略语

下列缩略语适用于本文件。

AI 人工智能（Artificial Intelligence）

API 应用程序编程接口（Application Programming Interface）

BFF 服务于前端的后端（Backend For Frontend）

BI 商务智能（Business Intelligence）

CDN 内容分发网络（Content Distribution Network）

HTTP 超文本传输协议（Hyper Text Transfer Protocol）

HTTPS 超文本传输安全协议（Hyper Text Transfer Protocol Secure）

H5 超文本标记语言第5版（Hyper Text Markup Language,the 5th version）

IaaS 基础设施即服务（Infrastructure as a Service）

IDL 接口定义语言（Interface Definition Language）

JSON JavaScript对象表示法（JavaScript Object Notation）

MAC 媒体访问控制（Media Access Control）

OCR 光学字符识别（Optical Character Recognition）

PaaS 平台即服务（Platform as a Service）

REST 表述性状态转移（Representational State Transfer）
RPC 远程过程调用（Remote Procedure Call）
SaaS 软件即服务（Software as a Service）
SPI 服务商提供接口（Service Provider Interface）
TPS 每秒事务数（Transaction Per Second）
URL 统一资源定位符（Uniform Resource Locator）
XML 可扩展标记语言（eXtensible Markup Language）

5 总体架构

MMA总体上应与图1相符合，包括基础设施适配层、微服务治理能力层、媒体业务服务层、平台服务层和应用集成层共五层。其中，基础设施适配层和微服务治理能力层主要负责相关的基础资源调用和技术维护以及微服务的治理；媒体业务服务层、平台服务层和应用集成层主要负责提供视听媒体业务功能域的各类微服务，以及通过这些基础的业务功能性微服务形成平台服务和集成应用的能力。该架构实施的简要过程见附录A；关于系统接口描述，见附录B；MMA宜结合DevOps实施，见附录C；常见的微服务类别和微服务架构风格的描述，见附录D；MMA的典型应用案例见附录E。

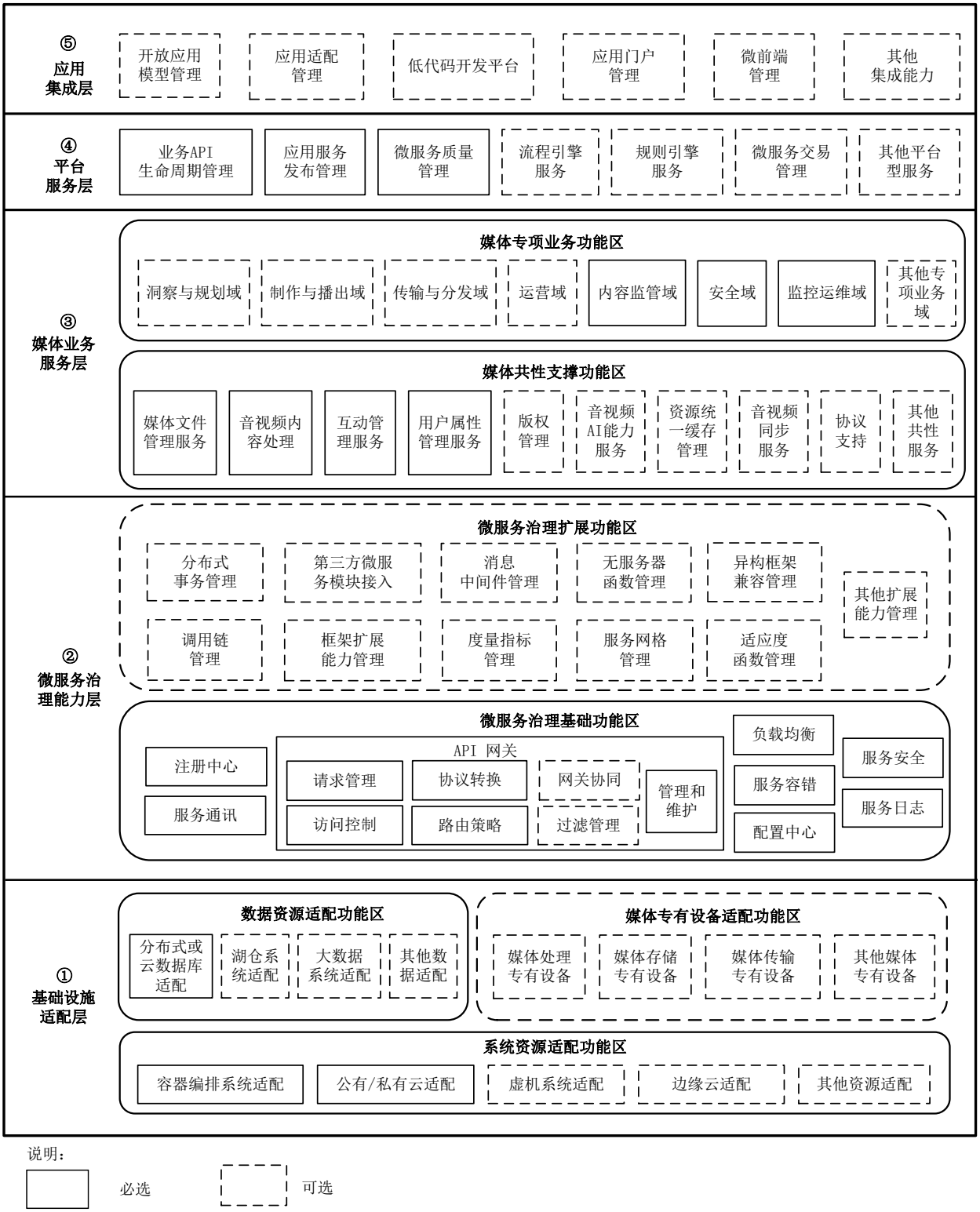

图1　MMA总体架构

6　基础设施适配层

6.1　系统资源适配功能区

系统资源适配功能区对底层的基础设施（计算、存储、网络等）进行调用，提供对以下基础资源的适配。

a) 应支持容器编排系统适配：提供与容器编排系统相适配的能力，包括与CPU/GPU资源相适配的能力，根据需求，容器编排可与各类具体资源耦合，连接更丰富的基础资源。
b) 应支持公有/私有云适配：提供公有云、私有云以及混合云的资源对接，支持以插件的形式提供不同云的接口，实现单一平台管理不同的云。
c) 宜支持虚机系统适配：提供与虚机系统相适配的能力。
d) 宜支持边缘云适配：提供与边缘云相适配的能力。
e) 宜支持其他资源适配：提供与其他资源相适配的能力。

6.2 数据资源适配功能区

数据资源适配功能区对数据资源进行调用，支持与以下数据资源的适配。

a) 应支持与分布式或云数据库的适配：提供与分布式数据库或云数据库相适配和调用的能力，应具备生成全局唯一标识符的能力。
b) 宜支持与湖仓系统的适配：提供与数据仓库/数据湖、数据湖仓一体、实时离线数仓一体等系统相适配能力。
c) 宜支持与大数据系统的适配：提供与其他大数据系统相适配的能力。
d) 提供与其他数据适配：由人工智能大模型带来的数据需求等，可考虑在此进行适配调用。

6.3 媒体专有设备适配功能区

媒体行业的媒体处理专有设备、媒体存储专有设备、媒体传输专有设备、其他媒体专有设备等根据需求可适配融入MMA。

7 微服务治理能力层

7.1 微服务治理基础功能区

7.1.1 注册中心

微服务注册中心具备以下能力。

a) 应高可用。
b) 应提供服务注册、服务订阅、服务续约和服务下线管理能力。
c) 应提供服务发现的能力。
d) 应支持维护服务注册表或库的能力。
e) 应保证业务服务调用的安全，在服务注册、服务订阅、服务续约和服务下线等环节都进行严格的服务鉴权。
f) 当注册中心的部分节点出现故障时，应不影响当前业务系统之间进行服务调用。
g) 宜支持无损发布，如在确定微服务实例已经能提供正常服务之后，才能被注册发布。
h) 宜支持容器编排型的服务注册发现能力。
i) 宜支持单元化：服务按照不同的逻辑单元进行单元化部署，单元之间会进行逻辑隔离，允许跨单元调度。

7.1.2 服务通讯

微服务间的通讯具备以下能力。

a) 应支持协议扩展能力（如通过服务商提供接口插件形式），对新协议进行管理。
b) 应支持同步通讯能力，对外的API调用以REST（HTTP/HTTPS+JSON）方式实现，对内的微服务通讯推荐以RPC方式实现。
c) 宜支持异步通讯能力（消息系统或任务管理，如Java消息服务、高级消息队列协议等异步消息协议，可扩展消息和表示协议、消息队列遥测传输等实时消息服务协议）。

7.1.3 API 网关

7.1.3.1 基础功能

API网关应具备如下基础功能。
a) 请求管理：具备统一接入能力，需承担服务调用、负载均衡和协议转换的功能；需对接外部微服务、APP、小程序等各种前端渠道的服务调用。
b) 访问控制：具备基础的访问权限控制能力，如黑白名单控制等。
c) 协议转换：将外部请求的协议转换为内部服务支持的协议；支持异步与同步协议的能力；宜提供屏蔽异步过程的能力，即从前端看与正常同步调用无区别。
d) 路由策略：提供路由前端请求到服务的不同实现的路由策略能力。

7.1.3.2 扩展功能

API网关宜具备如下扩展功能。
a) 网关协同：支持多注册中心，提供分布式网关集群下对多注册中心集群的切换管理功能。
b) 过滤管理：支持以过滤器的方式对网关功能进行无侵入的热插拔扩展，支持第三方插件扩展特性及动态加载机制。

7.1.3.3 管理和维护功能

API网关具备以下对网关的管理和维护功能。
a) 应具备网关集群管理、日志查询、预警管理等服务治理功能。
b) 宜支持可视化管理，提供实时路由拓扑、网关集群拓扑展示功能。
c) 可进行格式检查：可以使用模板引擎等技术手段配置和验证检查规则。
d) 可支持数据转换：支持将返回数据转换成JSON或XML等不同的格式，可以使用数据模板引擎。
e) 可支持BFF模式：将BFF作为服务网关的插件，在服务网关中插拔实现。
f) 可支持可视化消息查看：提供任意两个服务间的可视化消息查看。

7.1.4 负载均衡

7.1.4.1 微服务的负载均衡

为保证微服务的高可用性，支持以下对微服务的负载均衡能力。
a) 应支持后端微服务负载均衡能力。
b) 应支持常见的负载平衡策略，如随机、轮询、加权轮询、IP哈希、最小连接数等。
c) 宜支持客户端微服务负载均衡能力。
d) 宜支持粘性会话类型的负载均衡策略。
e) 宜支持对于同端口下、不同上下文根请求的请求分发。

7.1.4.2 网络的负载均衡

支持以下对网络的负载均衡能力，保证网络和硬件间负载均衡的协作。

a) 应支持L7层负载均衡：在该层的负载均衡能力除了根据IP加端口进行负载外，可根据URL、浏览器类别、请求头中关键字段信息等信息来负载均衡。
b) 应支持L4层负载均衡：负载均衡服务器在收到第一个来自客户端的同步请求后，通过修改数据包的地址信息（IP+端口号）将流量转发到应用服务器。
c) 宜支持L3层负载均衡：支持负载均衡服务器获取网络数据包。
d) 可支持L2层负载均衡：支持链路层通过修改帧数据包中的MAC地址来达到转发的目的，所有的真实服务器和负载均衡服务器都有相同的IP（或虚拟IP）地址。

7.1.5 服务容错

通过以下服务容错手段保障系统的可用性。
a) 应支持关闭、打开和半打开三种模式的断路器容错模式，在断路器出现关闭状态或半打开状态时，提供必要的告警或通知手段。
b) 应支持系统对处理的服务数量进行静态或动态的阈值设置，超过该阈值的请求会被直接拒绝或按配置降级返回。
c) 应支持对重要微服务的自动及手动熔断降级、限流处理，允许对单个微服务进行手动限流、降级和熔断处理。
d) 宜对一些不重要的服务进行降级设置，通过限流自动管理特定微服务的最大访问量限制或转流。
e) 宜支持微服务自动容错处理方式，包含提供服务健康自动检查、服务上下线过程中稳定性的保障手段、应用实例异常自动排除等。
f) 宜支持微服务的全链路灰度发布能力。

7.1.6 配置中心

配置中心提供以下功能。
a) 应提供静态与动态配置：支持在系统初始化时加载配置信息并生效、不在运行期更改信息的静态配置方式；支持配置信息的动态刷新，在应用程序不重启的情况下修改配置信息并动态生效的动态配置方式。
b) 应具备针对不同环境（开发或发布等阶段）、不同集群配置能力，宜具备配置权限、流程管理功能。
c) 宜提供控制管理功能：将分布式系统的配置集中管理，具备管理控制界面，支持权限管理，可以对配置信息进行编辑、发布、回滚，可以查看配置信息的历史版本，监控各节点的配置应用情况。
d) 宜支持多种配置模式：支持针对属性文件、XML文件等文件类型的配置管理，配置文件的内容可以加载到配置管理端，在管理端进行编辑，配置信息可以通过管理端实时刷新到业务系统内存中。支持通过控制台将配置下发到服务器的指定目录，应用程序无需重启即可通过读取该目录下的配置文件实现特殊的业务逻辑。

7.1.7 服务安全

通过以下手段保障微服务的安全性。
a) 应通过API网关对所有客户端请求进行安全的接收和管理。
b) 应具备基于HTTP/HTTPS的认证、基于会话的认证和基于令牌的开放安全协议认证。
c) 应将身份认证存储到专门的授权服务器，为微服务提供访问权限。

d) 如消息包含敏感信息，应对消息内容进行加密，对更重要的消息还宜提供验证以确保消息没有被篡改。
e) 应支持常见鉴权方式，宜支持自定义鉴权插件。
f) 宜支持网关上的令牌转换，API网关提取访问令牌，并将其发送到授权服务器以鉴权。
g) 宜支持多因子认证，含基于黑白名单等服务鉴权能力。
h) 当与外部系统通信时，宜对消息进行加密。

7.1.8 服务日志

服务日志满足以下要求。

a) 应记录系统基础日志信息：包括微服务系统运行的服务器、网络、存储、中间件、数据库等基础监控信息。
b) 应保证日志的完整性：在记录日志时，对产生日志事件的等级、产生事件的类别和方法名称、相关堆栈跟踪信息和异常消息等，要进行完整存储记录。
c) 应保证记录的ID的唯一性：系统的应用性能和业务日志记录里每个请求都应具有唯一的ID（可由系统自动生成字段和多个具备业务意义的字段组成）。
d) 应对应用日志和业务日志的属性进行记录：除了记录实际的日志消息之外，还应该包含附加属性。
e) 应对严重错误立即生成告警：当处理过程中出现严重错误，而且会一直影响业务并需要人工干预时，应给出日志告警提示。
f) 宜具备自定义系统业务日志内容的功能：根据不同业务系统的业务规则自定义日志文件的格式和输出内容。
g) 宜支持灵活的检索能力：支持查看单条日志的上下文日志，支持关键词、正则表达式搜索日志，支持按调用链ID搜索日志查询。

7.2 微服务治理扩展功能区

7.2.1 分布式事务管理

分布式事务管理具备以下能力。

a) 应提供基础的基于分段式提交机制的事务管理能力。
b) 应提供基于补偿机制的事务管理能力，同时覆盖短的和长的事务管理。
c) 可提供基于可靠消息队列机制的事务管理能力。
d) 可支持其他高级分布式事务管理能力。

7.2.2 调用链管理

调用链管理支持以下功能。

a) 应提供关于任意一个微服务调用的可追溯性，主要是提供全面的对微服务调用链的追踪和管理能力。
b) 对每个调用链，应提供微服务实例相关信息，如微服务名称、微服务的开始时间和结束时间等基本信息，应提供对其他调用链的引用情况，宜提供微服务的客户化标识。
c) 对每个请求调用，应利用跟踪ID和跨度ID，将整个调用链以及其中的每个调用环节串联起来，如请求发起时间、每个环节调用方和被调用方微服务名称、调用的开始时间和结束时间等基本信息。
d) 应具备关于追踪数据的数据采集、数据持久化和数据展示等管理能力。

e） 应支持调用链和日志的联动，在服务调用中可自动关联调用日志，提供快速排障的能力。
f） 宜支持服务和中间件的调用联动，支持中间件（如消息队列、远程字典服务、远程缓存等）、数据库的调用链层级展现。

7.2.3 第三方微服务模块接入

第三方微服务模块接入应支持以下能力。
a） 提供容器云平台和镜像仓库用以支持第三方微服务模块镜像的上载。
b） 提供支持主流微服务模块开发语言的运行环境。

7.2.4 框架扩展能力管理

框架扩展能力管理支持以下扩展方式。
a） 应具备过滤器方式：适用于简单的扩展需求，支持动态添加和删除。
b） 宜具备服务提供商插件（SPI Plug-in）方式：适用于对性能要求高或扩展系统复杂的情况。

7.2.5 消息中间件管理

7.2.5.1 微服务间异步通信管理的消息中间件

当消息中间件主要作为微服务间异步通信管理时，本模块具备如下消息管理的能力。
a） 应提供保留消息数据的持久化机制。
b） 应提供发布订阅、轮询分发、消息拉回等消息分发机制。
c） 应具备分布式消息处理能力。
d） 不应使用HTTP协议作为消息传递协议。
e） 宜支持高级消息队列协议。
f） 宜支持基于流处理、批流融合处理等消息协议。
g） 可支持消息队列遥测传输等适合物联网的消息协议。

7.2.5.2 流计算引擎的消息中间件

流计算引擎的消息中间件具备如下能力。
a） 应同时支持两类流系统：顺序处理流系统，该类系统对流元素进行缓冲后重排序；无序处理系统，该类系统通过信号来追踪数据流的处理进度，不会缓冲数据以强制排序。
b） 应满足数据一致性要求：跨越所有流计算上下游系统的数据，反映相同的信息。
c） 宜支持流数据的完整性推理要求：即支持边界标记（Punctuation）、低水位标记（Low Watermark）、宽限时间（Slack Time）、心跳检测（Heartbeat）等数据完整性推理方案。

7.2.6 度量指标管理

度量指标管理支持监控指标获取、日志采集等方式提供以下系统的运行状态指标。
a） 应提供关键性能参数：微服务API的调用次数、响应时间、响应码状况和TPS值；网络传输中的关键指标延迟、丢包、抖动、带宽等指标；多媒体应用中的编码、码率、帧率、分辨率和加密状态。
b） 应提供关键事件参数：业务流程中的事件名称、发生时间，被触发的微服务发送或接收消息的状态。
c） 宜提供关键路径参数：业务流程中经过的服务节点、参与流程的服务、节点的地址、微服务的端口号、节点之间的网络拓扑状况、连接数和跳数。

d) 宜提供业务级别的标识信息、业务流水号、用户的标识信息等。
e) 宜支持对度量指标的可视化图形展现（如在Java环境下，支持对Java虚拟机的可视化监控，包括线程、方法栈的调用等）。

7.2.7 无服务器函数管理

无服务器函数支持以下管理功能。
a) 应支持函数的生命周期管理和函数的触发执行：包括函数的上线、运行、删除、更新等操作。
b) 应支持函数冷启动：该启动流程需要考虑到资源调度、容器启动、函数代码下载、函数启动、函数初始化和函数处理调用请求。
c) 应支持函数弹性伸缩：依据应用的资源负载进行弹性决策，或依据请求流量进行弹性决策。
d) 对有状态函数宜提供程序状态管理功能：状态管理的操作包括状态的生成、状态的访问、状态的操作、状态的删除及状态的回收。

7.2.8 服务网格管理

服务网格管理提供以下功能。
a) 应提供高性能通信代理（sidecar），负责各个服务之间通信；应具备对sidecar进行管理的能力，支持查看及管理当前工作空间中的sidecar实例，提供sidecar状态、注入时间等信息。
b) 具备以下控制面功能。
   1) 应提供遥测记录服务，处理遥测记录，将代理sidecar产生的请求指标送到指定的后端。
   2) 应提供策略执行服务，主要负责集群管理策略的执行。
   3) 应提供认证服务：在服务和服务、用户和服务之间进行认证，实现访问控制；宜包含公开密钥体系，在服务网格中为每个微服务提供客户端传输层安全性协议证书的生成、轮换、撤销以及端到端的验证服务。
   4) 宜支持多种开发语言应用接入。
   5) 宜将业务与服务治理能力相隔离，对代码无侵入。
   6) 宜提供自定义资源控制器，处理自定义资源的变更，并向服务网络中相关的其他服务进行内容分发；宜提供引导服务，对服务网格中的自定义资源，应提供规范配置和编排管理，引导其注入到代理的sidecar容器中。
c) 应具备数据面功能，能处理服务网格中微服务之间的网络连接，从控制面接收微服务的路由和策略规则，并向控制面报告连接被处理的结果。
d) 可与资源解耦，提供灵活的部署方式，支持部署在虚拟机与容器上。

7.2.9 异构框架兼容管理

异构框架兼容管理通过以下功能对异构类别的微服务和跨微服务框架的微服务进行兼容管理。
a) 应支持统一的服务管理应对后端服务的访问请求，以提高链接收敛、服务治理、高速访问等能力。
b) 应支持异构应用互相通信、统一服务治理，提供异构应用的统一管理能力。
c) 宜使用一些中立的IDL定义服务的API，使得在不同平台上运行的对象和用不同语言编写的程序可以相互通信交流，从而帮助实现最大限度的跨平台微服务API调用。
d) 可使用运行时来实现微服务的核心业务功能，而运行时根据执行环境的变化可以进行自动适应。

注：运行时是指云计算语义下过程级别的虚拟机。

7.2.10 适应度函数管理

适应度函数支持以下管理工具。

a) 应支持度量工具：对复杂网络中节点和团组进行度量的工具。
b) 宜支持模拟工具：对度量参数进行模拟计算的工具。

7.2.11 其他扩展能力管理

其他扩展能力管理包括以下内容。

a) 针对视听媒体领域微服务计算、存储和网络资源开销大的情况，可提供微服务在多云、多地、多中心的高效率、低成本的综合弹性伸缩能力。
b) 针对视听媒体领域存在的受法律法规、版权、运营商规则等限制比较严格的情况，可提供可配置的API合规接入能力，例如涉及隐私请求转发到外部的微服务实例、限制非版权许可地区用户访问某影片、尽量少进行效果不可预测的跨运营商音视频内容传输等。
c) 针对处理时间较长、重要程度较高的任务环节（如内容生产流程中的渲染、编码等）的情况，可提供对长任务处理过程的监控能力。

8 媒体业务服务层

8.1 媒体共性支撑功能区

媒体共性支撑功能区包含以下视听媒体共性服务能力。

a) 媒体文件管理服务：应包含对媒体文件的新增、加载、查询、传输和加速等服务，以及防盗链服务等。
b) 音视频内容处理：应包含关于多媒体流的编码服务、压缩服务、推送/接收服务、协议/格式转换、音频控制、抽帧服务、流媒体切片服务、流媒体直播加速，以及AI超清分辨率服务等。
c) 互动管理服务：应提供短信/彩信/消息网关、评论服务、弹幕服务等。
d) 用户属性管理服务：应提供IP归属地服务、儿童成人用户信息服务、无障碍访问服务等。
e) 版权管理：宜包括版权凭证存档、版权确权登记、版权资产仓库管理、版权数据洞察等基础能力，可包括侵权判定、证据固定等维权保护能力，可增加针对版权缺乏查询、广告审核机制等风控预警等能力。
f) 音视频AI能力服务：宜具备针对音视频内容的生成和处理能力，主要包括“人工智能模型即服务”（含视频内容分析、视频特效生成、人脸识别、语音识别、OCR识别、AI文本生成和AI图片和视频生成等）的能力以及模型生成的辅助功能等。
g) 资源统一缓存管理：宜提供对媒体大文件的本地缓存、分布式缓存、反向代理缓存、浏览器缓存和CDN缓存能力，宜提供热点缓存和多级缓存等缓存策略。
h) 音视频同步服务：宜提供关于多媒体流的音频和视频同步服务。
i) 协议支持：宜支持与媒体业务相关的协议，如终端和媒体网关协议（如基于IP的语音传输、会话邀请协议等），远程资源共享及控制协议（远程桌面协议、远程帧缓冲等），流媒体传输协议（实时传输协议、安全实时传输协议、实时流协议等）。
j) 其他共性服务：宜提供区块链支持的其他媒体服务等。

8.2 媒体专项业务功能区

8.2.1 洞察与规划域

洞察与规划域的微服务组件宜具备以下能力。

a） 线索汇聚：开展面向多渠道的线索汇聚。

b） 舆情热点分析：可通过对时间、事件、人物、地域等进行个性化设置，分析舆情热点。

c） 选题策划：开展统一选题策划，实现选题创建和编辑、选题筛选、选题审核派发、组合报道等。

8.2.2 制作与播出域

制作与播出域的微服务组件宜具备以下能力。

a） 内容汇聚：通过多渠道对文字、图片、音频、视频等内容进行汇聚。

b） 内容制作：面向多种发布渠道，实现视听媒体内容制作。

c） 内容播出：面向多种发布渠道，实现视听媒体节目的播出。

8.2.3 传输与分发域

传输与分发域的微服务组件宜具备以下能力。

a） 广播电视传输分发：面向广播电视频率频道进行节目的传输覆盖。

b） 客户端发布：面向媒体接收客户端发布视听媒体内容。

c） 网站发布：通过媒体网站发布视听媒体内容。

8.2.4 运营域

运营域的微服务组件宜具备以下能力。

a） 支持对平台用户和各类软件应用的运营管理。

b） 具备统一门户，支持用户统一认证登录。

c） 提供多租户服务。

d） 为用户提供个性化、可编排的服务界面。

e） 具备向各应用系统提供必要用户信息的能力。

8.2.5 内容监管域

内容监管域的微服务组件具备以下能力。

a） 应具备对视听媒体内容审核能力。

b） 宜具备对视听媒体技术质量进行检测的能力。

8.2.6 安全域

安全域的微服务组件应具备以下能力。

a） 登录认证：支持对登录的用户进行身份标识和鉴别。

b） 访问控制：通过授权、认证、访问控制列表、角色、单点登录、双因素认证对用户或其他系统的访问进行限制和管理。

c） 安全审计：支持对重要的用户行为和重要安全事件进行审计。

d） 数据安全：支持通过密码技术保证重要数据在传输、存储过程中的保密性。

e） 签名验签：支持通过校验或密码保证重要数据在传输、存储过程中的签名验签完整性。

f） 威胁监测：支持对恶意活动和未经授权的行为的持续发现，识别各类潜在威胁并输出分析结果。

8.2.7 监控运维域

监控运维域的微服务组件具备以下能力。

a） 业务运行状态监测：应支持对业务系统、应用程序、网络设备等的运行状态进行实时监测。

b) 业务运行性能监测:宜支持对业务系统的业务处理、数据传输等各项性能指标进行监测和分析。
c) 业务资源使用率监测：宜支持对业务系统所使用的资源（如CPU、内存、磁盘空间、带宽等）进行监测和分析。
d) 用户访问量：宜支持对一定时间范围内访问页面、资源或功能的用户数量进行统计。
e) 自动化业务巡检：宜支持对业务系统的安全策略（如告警、监控、访问控制等）进行巡检、自动发现和诊断业务问题。

8.2.8 其他专项业务域

可支持其他媒体专项业务域的微服务组件。

9 平台服务层

9.1 业务 API 生命周期管理

业务API生命周期管理应支持以下服务能力。
a) 对业务型API的设计、开发、测试、部署、发布以及运行和维护等全生命周期进行管理。
b) 对API的安全性和可用性进行合规检测。

9.2 应用服务发布管理

应用服务发布管理应支持以下服务能力。
a) 支持微服务应用的滚动发布、应用启停、回滚。
b) 支持服务实例按每秒查询率、响应时间、CPU、内存等条件自动弹性伸缩，对业务流量发挥云的弹性能力。
c) 支持操作行为、执行状态信息记录，便于操作回溯及审计。
d) 支持优雅下线，版本发布时，通过反注册等手段实现实例的无损下线，避免业务流量异常。

注：优雅下线是能够完成已经接受的请求并不再接受新的请求，避免造成正在运行的微服务中断或错误发生的一种稳健的系统服务下线机制。

9.3 微服务质量管理

微服务的质量管理提供以下能力。
a) 应提供微服务契约测试的能力。
b) 应提供微服务性能测试能力。
c) 宜提供微服务虚拟化的能力。

9.4 流程引擎服务

流程引擎支持以下服务能力。
a) 应提供可定义自助服务流程，扩展服务能力。
b) 应通过系统管理和监控接口对流程实例的状态进行监控与管理。
c) 宜提供图形化、可视化过程定义工具。
d) 宜提供工作流安全机制，包括认证、授权、访问控制、审计、数据保密性、数据完整性、防否认、安全管理等。
e) 可提供工作流执行服务，完成工作流流程实例的创建、执行与管理。

9.5 规则引擎服务

支持以下规则引擎服务能力。

a） 应支持脚本式规则集，宜支持向导式规则集。

b） 应支持普通决策表和交叉决策表（又称决策矩阵）。

c） 应支持普通评分卡，宜支持复杂评分卡功能。

d） 宜支持决策树功能。

9.6 微服务交易管理

微服务交易管理支持以下能力。

a） 应具备创建新服务实例、修改微服务实例、续订服务实例和退订服务实例的基本能力。

b） 应提供订购管理能力。

c） 应提供记账管理能力。

d） 宜具备定价能力。

e） 宜具备支付管理能力。

9.7 其他平台型服务

可支持如公共对象请求代理中间件、公共语言运行库、数据库连接中间件等，对本层能力做进一步的扩展。

10 应用集成层

10.1 总则

视听媒体软件应用的构建宜以媒体业务服务层中的微服务组件做业务池为基础，依托平台服务能力层做协同，再通过本层的应用集成能力来进行组装实现。

10.2 开放应用模型管理

开放应用模型管理具备以下功能。

a） 应用程序应进行组件化划分，这些组件可以调用相关微服务构建自身的功能。

b） 应具备应用部署配置文件来描述组件与其调用微服务之间的关系。

c） 宜提供应用运维特征信息，描述应用在具体部署环境中的运维特征。

10.3 应用适配管理

应用适配管理具备以下功能。

a） 应具备应用接口适配中心，提供API托管和适配服务。

b） 宜具备应用连接器插件框架模型，支持在基础连接器基础上进行插件化扩展。

c） 可具备中间转化语言能力，通过定义路由以及中间规则，让不同领域的应用有交互可能。

10.4 低代码开发平台

低代码开发平台具备以下基本功能。

a） 应提供可视化设计器，能够对表单、门户、流程、规则等常用的应用搭建基本构件进行可视化编程处理。

b） 应具备可视化搭建能力，能通过拖拽等可视化操作对业务建模、流程设计、用户页面设计等应用关键构成进行搭建和重构。

c） 应具备对软件应用基本构成部件进行可视化展示的能力。
d） 宜具备对代码库和微服务组件等软件应用基本构成部件进行配置管理的能力。
e） 宜提供一键发布、可视化应用状态监督等辅助能力。
f） 宜具备通过网页UI进行操作的能力。

10.5 应用门户管理

应用门户管理具备以下功能。
a） 应具备将不同应用的选取内容展示到特定UI窗口的能力。
b） 应具备对各类UI窗口进行统一布局和管理的能力。
c） 宜具备对单个应用进行门户个性化订制的功能。
d） 宜具备对管理的应用进行统一权限管理的能力。

10.6 微前端管理

微前端管理具备以下功能。
a） 应具备对UI组件的编排和管理能力。
b） 应将前端组件的控制和展示进行分离。
c） 宜为前端应用提供沙箱运行环境。

10.7 其他集成能力

可支持生产系统、媒资管理等视听媒体应用的其他集成能力。

附 录 A
（资料性）
MMA应用过程参考

A.1 MMA应用开发流程

MMA系统构建的实施步骤见图A.1。

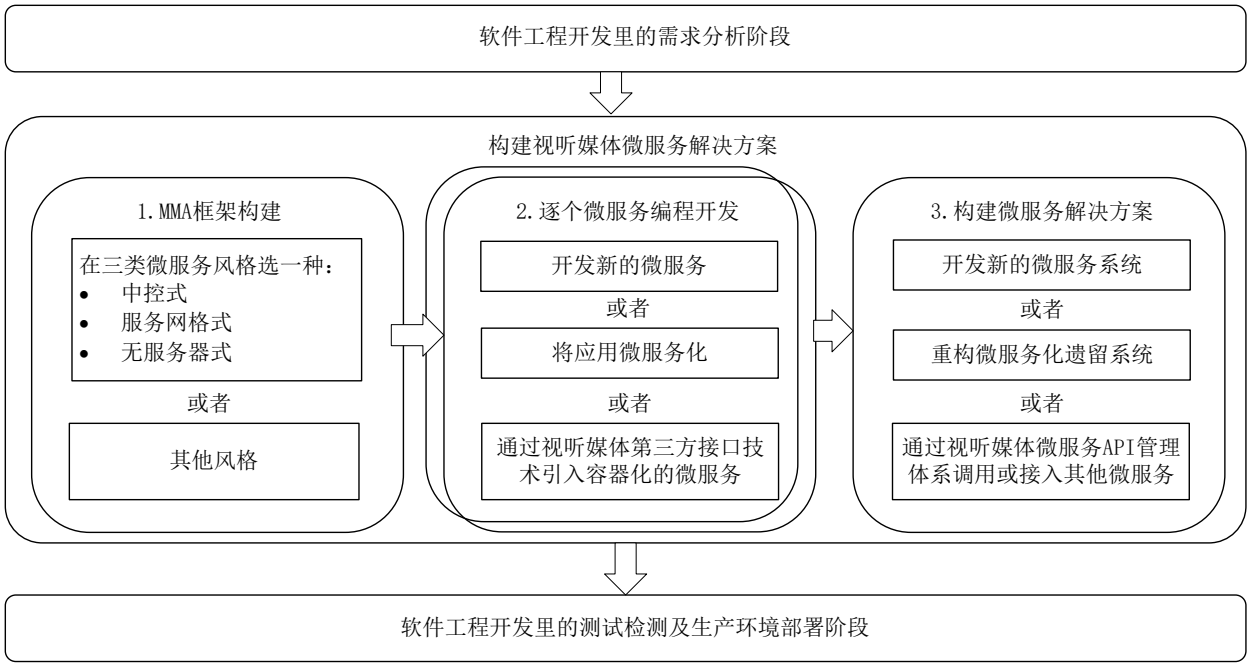

图 A.1 MMA 系统构建的实施步骤

A.2 确定缺省微服务框架

使用视听媒体微服务参考架构的首要和关键环节，是企业和机构依据自身开发人员对微服务架构的熟悉和认可，以及根据具体业务的复杂度，选取 MMA 的缺省微服务架构。

视听媒体微服务参考架构主要关注主流微服务架构之间的兼容性和扩展性，让企业选择的缺省微服务架构融入 MMA 之中，帮助实现更多微服务间的无缝调用、实现更多适合视听媒体业务的标准化微服务模块。

A.3 选择微服务框架兼容方式

MMA 中的异构框架兼容能力由 API 网关（主要是其中的路由策略和网关协同能力）、无服务器函数管理、服务网格管理、第三方微服务模块接入、框架多模态兼容管理、多运行时扩展管理等模块相互配合形成。依据框架兼容的难易程度（从浅层 API 接口调用向底层架构机制的交互逐步深入），MMA 可实现以下数种方式进行不同风格微服务框架之间的兼容。

a） 微服务间直接调用：在一种微服务框架风格下，去调用另一种微服务框架风格下的服务，如果不考虑安全等问题，可以通过直接调用相关 RESTful 的 API 完成。

1） 适用性：临时性功能的实现。

2) 优点：简单易用。
3) 缺点：直接调用的弊端是被调用的微服务不在本微服务框架的治理控制范围内，因此，无法与其他微服务一起获得微服务框架治理带来的弹性计算和容错处理等优异性能，这种API 直接调用无法实现异构微服务框架治理之间的兼容。

b) 通过 API 网关进行桥接和级联：由于所有微服务几乎都是由微服务框架中的 API 网关作为第一“经手人”进行治理协调的，将不同微服务框架中的 API 网关进行桥接和级联就成为异构框架兼容能力形成的主要手段。如果在某类微服务框架中能够找到适合另一类框架的网关插件，则调用该网关插件进行 A 和 B 网关间的同步协调；这种同步是由服务端的注册和配置中心自行调用网关插件，自动进行跨网关配置来实现兼容。如果没有合适的微服务框架实现方提供的网关插件，将在 MMA 的 API 网关中的“网关协同”模块中将客户端手动网关配置行为进行自动化处理（在 MMA 的网关协同中以网关桥接的方式实现对两种风格微服务调用的内部协调；对更复杂的兼容还需要实现网关不同层级间的级联能力）。
1) 适用性：一般发生在一种旧微服务框架向新微服务框架迁移的过程中，此时旧系统没有完全转换完成，因此使用“并立运行”的兼容方式。
2) 优点：是一种通用型的不同风格微服务治理间的协调方式，由于 API 网关的特殊性，不同系统间的协调问题适合在此统一解决。
3) 缺点：需要被兼容异构的微服务框架同步运行。

c) 通过无服务器函数管理和服务网格管理模块进行兼容：多数异构微服务框架风格的兼容，都会有一种微服务框架风格占主流，这种异构的兼容方式将通过其他微服务框架风格的最核心部分打包成为主流框架的插件、在主流框架管理下运行。由于中控式微服务架构是微服务的最早实现和管理方式，这种架构下的插件也较为丰富，MMA 选择了中控式微服务架构作为缺省框架。当前对无服务器风格和服务网格风格的微服务治理还没有形成业内共识，因此这种方式的异构兼容取决于在 MMA 的扩展治理中对无服务器函数管理模块和服务网格管理模块功能的覆盖程度和代码质量。鼓励使用一些 IDL 去描述这些管理模块的接口功能，并通过代码自动生成工具去实现适合所需特殊微服务框架的运行代码，从而帮助快速且稳定地实现异构微服务框架风格的兼容调用。
1) 适用性：需要长期利用不同微服务治理风格优点的复杂系统。
2) 优点：是一种通用型的协调方式，只需运行和维护一套微服务治理框架。
3) 缺点：实现难度较大，需要有对被兼容系统风格的深入把控能力。

d) 通过容器镜像文件管理进行兼容：通过将异构框架风格下的微服务打包成容器镜像文件（形成“黑盒”应用），通过系统资源适配功能区中的“容器编排系统适配”模块将其融入选用的容器编排系统。依赖容器系统的服务发现和管理机制，对这些“黑盒”微服务进行统一管理。这种类型的异构微服务框架兼容能力涵盖在“第三方微服务模块接入”模块中（隶属于 MMA 中的微服务治理扩展功能区）。
1) 适用性：对外部功能的丰富性需求强烈，但对性能要求不高的系统。
2) 优点：实现难度较小，是一种通用型的协调方式，有统一安全处理。
3) 缺点：处理性能有损外，只能对“黑盒”进行粗粒度治理，不易实现精细高级的微服务治理能力。

A.4 微服务技术应用成熟度划分参考

微服务技术的成熟度级别分成如下五个级别。

a) 第一级成熟度（微服务启动）：初始化成熟度，这是入门级别。在总体上还是采用传统的单体应用模式。采用了微服务思想、了解了微服务的大致架构和思路，可采用一些简单的微服务技术，如分布式技术、容错技术和容器技术等。基本按照单体结构来部署，总体上还是单体应用的实现方式。

b) 第二级成熟度（MMA 就绪）：基于对微服务概念和知识的理解，在一些边缘业务中采用了微服务架构；基于一些成熟开源的微服务框架来解决业务上和技术上的一些迫切问题，采用了部分自动化工具链等。在一些业务方面，开发团队和运维团队已经进行了合并。对偏计算类和占用 IT 资源较重的新系统或者视听媒体小型企业或构建新系统时，建议考虑该级别成熟度。

c) 第三级成熟度（MMA 友好）：基本实现微服务架构中的所有环节，从开发到运维，团队按照价值链方式进行组织。建议企业选用中控式成熟的微服务框架。

d) 第四级成熟度（MMA 成熟）：全面采用微服务体系内容，包括架构、分析、组织和流程，只是各个级别应用程度有差异。引入了量化测量指标，采用度量手段来实现对微服务应用质量的评价。建议企业除了沿用成熟的微服务框架外，也尝试使用网格化微服务框架。

e) 第五级成熟度（MMA 卓越）：全面采用微服务体系内容，包括架构、分析、组织和流程，完全达到了微服务的终极愿景。

当企业或机构对微服务的实践实现了本文件中所有对“应”类别的规范要求，其微服务实践属于 MMA 成熟级别的实践（覆盖得越全面，MMA 成熟度越高）。在此基础上，如还实现了“宜”“可”类条目或者“其他”中的扩展，都属于 MMA 卓越级别的实践。

附 录 B
（资料性）
MMA应用API接口描述

B.1 面向资源的API设计

MMA推荐采用REST风格的应用API接口设计。

REST架构的核心原则是定义一组命名资源，用少量相对统一的方法来控制这些资源。

资源对应业务域中的具体对象。在设计REST API时，需要对业务域进行分析，整理出需要通过API来访问的业务对象。如视听媒体相关的业务域中，素材、节目、不同类型的任务，都是典型的REST API资源。

方法是对资源的操作和控制。资源的创建、查询、修改、删除是REST API中的方法。根据具体业务场景，不同类型的资源可以支持全部方法，或只支持部分方法。

基于HTTP协议的REST API，资源映射到HTTP的URL地址，方法映射到HTTP协议的POST、GET、PUT/PATCH、DELETE等方法，分别对应资源的创建、查询、修改（PUT对应全量更新，PATCH对应部分更新）、删除这几个方法。

当方法不能很好地与业务场景匹配时，可以增加自定义方法。

进行REST API设计时，建议按下面的步骤进行。

a) 确定API提供的资源类型。
b) 确定资源之间的关系。
c) 确定资源的命名原则。
d) 确定每种资源对应的数据结构。
e) 为资源附加满足业务需求的最小化操作方法（包括自定义方法）。

B.2 URL格式参考

基于HTTP协议的REST API中，URL是API调用的目标地址，其中包含了协议类型、域名、服务名、资源等信息。自定义方法名通常会包含在URL中，另外也可以包含其他需要的信息。

典型的REST API URL格式为：

http(s)://{域名}[:{端口号}][/{服务名}]/{版本号}/[/{可选项}]/{资源}[/{方法}]

其中，[]里面是可选项。

各项参数解释如下。

a) http或https是协议类型，对外的服务普遍使用https协议。
b) 域名通常体现了API服务所属的系统信息。域名中也可能会包含服务信息，这时后面的服务名不再需要。
c) 端口号缺省时对应默认值，HTTP默认80端口，HTTPS默认443端口。
d) 服务名标识了服务信息。如果服务信息包含在域名中，此处可以忽略。
e) 版本号标识了REST API服务的版本信息，通常使用v1、v2这样的版本标识。当API有变更导致不能向后兼容时，采用新增服务版本号的方式。老版本的服务通常继续保留，以兼容老的客户端。
f) 根据业务需求，URL中可以有其他可选项信息。

g) 资源信息是URL中的重要部分，URL中的资源名通常采用复数形式。在获取某个资源的具体信息时，资源中通常包含资源ID。资源之间的父子关系，在URL中通常也会体现出来。如：projects/proj1/clips/clip1，这个资源信息表示在项目ID为“proj1”的项目中，素材ID为“clip1”的这条素材。

h) 为了简练，通用方法名称通常不放在URL中。但自定义方法的名称通常要体现在URL中，以便与通用方法进行区分。

B.3 API文档

API文档通常是使用工具根据API代码定义和代码注释自动生成的，这样可以很好地保持文档与代码的一致性。

API文档中包括对API的总体说明、对每种资源的具体说明以及对资源支持的每个方法的说明。对于每个方法，文档要描述方法的功能，对应的URL、HTTP方法、输入/输出参数详细信息，输入/输出样例等。

API的文档描述要清晰具体，能有效指导API调用方和实现方的开发。

B.4 API鉴权

API 需要提供鉴权机制，保证只有被授权的客户端才能对服务正常调用。

MMA 中，对 API 调用的鉴权处理统一在 API 网关中通过配置方式实现，API 内部通常不需要执行鉴权相关的判断逻辑。这种设计方式简化了 API 服务的开发，使开发人员能重点关注业务逻辑的实现。

代表调用方身份和授权的信息通常以 Token 方式通过 HTTP 消息头进行传递。在需要时，API 服务可以通过 Token 获得调用方的身份信息。

附 录 C
（资料性）
DevOps简述

C.1 概述

本附录对DevOps的重要组成部分过程管理、应用设计和安全管理做了描述，并对关键的CI/CD能力做了介绍，也对大型微服务系统持续维护中的防止系统逐渐退化腐败提出了防腐管理的建议。

DevOps 是一组过程、方法与系统的统称，用于促进开发（应用程序/软件工程）、技术运维和质量保障部门之间的沟通、协作与整合。它覆盖端到端软件交付生命周期全流程，是一套体系化的方法论，包括过程管理、应用设计、安全风险管理等方面。

C.2 过程管理

DevOps 过程管理主要包括敏捷开发管理、持续交付和技术运营三部分内容。

敏捷开发管理：包括从价值交付管理、敏捷过程管理、敏捷组织模式这三个维度，从开发过程中的有序迭代、灵活响应，以及价值的快速交付提出能力要求。

a) 价值交付管理主要是从需求工件、需求活动两部分内容，体现需求管理过程中的分析、测试、验收三个阶段。需求工件从需求内容和形式、需求测试用例编写、需求测试用例验证以及需求测试用例管理四个维度进行描述。需求活动从需求分析协作、需求管理方式、需求验收频率、需求验收范围以及需求验收反馈效率五个维度进行描述。
b) 敏捷过程管理是围绕业务价值交付进行的软件研发过程，分为价值流和仪式活动两部分。价值流分为交付与需求、交付质量和交付反馈与度量、价值流动四个维度进行描述。仪式活动分为交付计划、交付活动和人员组织三个维度进行描述。
c) 敏捷组织模式是团队在研发过程中的角色定义、角色能力以及之间的协作，团队结构的工作方式、团队间的协作模式等方面的要求，从而提升交付过程的流畅度，分为敏捷角色和团队结构两个维度的能力进行描述。

持续交付：是应用软件集成交付环节，通过配置管理、构建与持续集成、测试管理、部署与发布管理、环境管理、数据管理和度量与反馈的能力建设和工程实践保证软件持续、顺畅、高质量的对用户完成发布。

a) 配置管理是对软件产品及其开发过程和生命周期进行控制、规范的一系列软件工程过程。配置管理分为版本控制和变更管理两个维度的能力进行评估。
b) 构建与持续集成主要分为构建实践和持续集成两个维度的能力进行描述。构建将软件源代码通过构建工具转换为可执行程序的过程；持续集成是通过频繁的代码提交，自动化构建和测试，尽快验证和发现集成错误。
c) 测试管理是在软件开发过程中，对测试相关的过程、方法等进行定义和管理，分为测试分层策略、代码质量管理和自动化测试三个维度的能力进行描述。
d) 部署与发布管理是软件生命周期中，将软件应用系统向最终用户交付的过程，分为部署与发布模式和部署流水线两个维度的能力进行描述。
e) 环境管理是对生命周期管理、一致性管理和环境版本管理的过程，主要从环境类型、环境构建和环境依赖与配置管理三方面的能力进行描述。
f) 数据管理是系统开发过程中为满足不同测试需求，保证生产数据安全而人为准备的测试数据，主要从测试数据管理、数据变更管理两个维度的能力进行描述。

g） 度量与反馈是通过设立度量指标且及时有效的反馈机制，主要从度量指标和度量驱动改进两个维度的能力进行描述。

技术运营：该环节主要是应用系统服务发布后的环节，涉及运维成本服务、高可用架构服务、用户体验服务、客户服务、监控服务、产品运行服务和运营数据服务，保障良好的用户体验，打造持续的业务价值反馈流。技术运营从监控管理、事件管理、变更管理、容量管理、成本管理、连续性管理、用户体验管理、运营一体化平台八个维度的能力进行描述。

C.3 应用设计

应用设计需考虑可扩展性、可伸缩性、可观测性、可用性及安全性。应用软件技术架构为支撑敏捷开发管理、持续交付、技术运营、安全及风险管理等目标的关键能力。

a） 可扩展性是软件系统适应变化的能力，是对软件适应功能需求变化能力的度量。可扩展性越好，表示软件适应功能需求变化的能力越强。可扩展性主要从模块切分、模块依赖、模块兼容等维度评估。

b） 可伸缩性是不需要改变软件系统软硬件架构设计，仅通过改变部署服务器资源数量，就可扩大或缩小软件系统服务处理能力。可伸缩性是对软件适应性能和容量需求变化能力的度量。

c） 可观测性是通过应用软件外部输出的信息分析系统内部状态的能力，是对软件运行中掌握软件运行状态、发现问题、定位问题的能力的度量。可观测性越好，表示越容易发现和定位系统内部问题。

d） 可用性是在某个考察的时间区间内，软件系统处于可执行规定功能状态的能力。可用性越高，表示软件无中断执行其功能的能力越强。一般通过系统服务不中断运行时间占实际运行时间的比例来度量。

e） 安全性是为关键资产提供机密性、完整性和可用性保护的设计，可覆盖应用安全、数据安全、网络安全、信息安全等。

C.4 安全及风险管理

DevOps 包括了研发安全运营一体化（DevSecOps）。软件或相关服务在研发安全运营一体化的开发模式下，相比于传统开发模型，安全融入每个阶段过程，开发、安全、运营各部门紧密合作。DevSecOps 强调在安全风险可控的前提下，帮助企业提升效能，更好地实现 DevOps。安全及风险管理技术包括控制通用风险、控制开发过程风险、控制交付过程风险、控制运营过程风险。

a） 控制通用风险在DevOps安全内建于开发、交付、运营过程中，通用风险覆盖三个过程中的共性安全要求，包括：组织建设和人员管理、安全工具链、基础设施管理、第三方管理、数据管理、度量与反馈改进。

b） 控制开发过程风险是从应用的开发过程开始实施安全风险管理工作，可以保障进入交付过程的代码是安全的，降低后续交付、运营中的安全风险，保障研发运营一体化的整体安全，包括：需求管理、设计管理和开发过程管理。

c） 控制交付过程风险是从代码提交到应用发布给用户使用，安全交付是将安全内建到交付过程中，包括配置管理、构建管理、测试管理、部署与发布管理等环节。

d） 控制运营过程风险是应用发布给用户后的过程，将安全内建于运营过程中，通过监控、运营、响应、反馈等实现技术运营的安全风险闭环管理，包括：安全监控、运营安全、应急响应、运营反馈。

C.5 CI/CD管理

CI/CD管理能力包括。

a） 建立管道：通过一系列工具将代码集成和打包发布等工作串接起来，形成自动化的管道。
b） 提交代码：对新提交代码进行自动编辑和版本更新，有时还包含对代码质量的自动检测。
c） 提交测试：涵盖契约测试、单元测试、集成测试等能力。
d） 发布部署：将代码变动编辑成不同的软件包，自动发布到测试、集成、生产等不同环境。

C.6 防腐管理

防腐管理能力一般包括。

a） 提供绞杀者模式：通过逐步替换而非一次性替换的方式，来保证新旧系统的平滑过渡。
b） 提供挎斗模式：将应用程序的辅助组件部署为单独的容器或进程以提供隔离和封装。
c） 提供舱壁模式：隔离了每个工作负载或服务的关键资源，如连接池、内存和CPU。使用舱壁避免了单个工作负载（或服务）消耗掉所有资源，从而导致其他服务出现故障的场景。这种模式主要是通过防止由一个服务引起的级联故障来增加系统的弹性。

其他高级防腐模式能力还可包括。

a） 提供统一的存储内部文件迁移服务，通过文件硬链接的方式将文件进行共享操作，不需要物理文件真正意义上的拷贝和迁移。
b） 将所有文件信息注册到平台服务层，由平台服务层统一管理，包括生命周期的管理、逻辑存储空间的管理。
c） 实现平台内部及与外部系统间的数据迁移、传输。
d） 支持将指定文件迁移至另一个业务网，包含新建迁移任务、查询迁移任务、暂停继续任务、取消任务、修改任务优先级等。