ICS 35.240.80
C 07

YB

医保网络安全和信息化标准

XJ-B01-2019

医疗保障信息平台
应用系统技术架构规范

2019-09-06 发布　　2019-09-06 实施

国家医疗保障局网络安全和信息化领导小组办公室　发布

目次

前言

本标准按照 GB/T 1.1-2009 给出的规则起草。

请注意本文件的某些内容可能涉及专利。本文件的发布机构不承担识别这些专利的责任。

本标准由国家医疗保障局规划财务和法规司提出并归口。

本标准起草单位：国家医疗保障局规划财务和法规司。

医疗保障信息平台 应用系统技术架构规范

1 范围

本规范规定了医疗保障信息平台建设总体技术架构,给出了业务子系统应用架构分层设计、核心服务框架和云平台适配框架设计说明，提出了框架相关技术选型、框架总体应用要求，明确了框架版本更新机制等方面内容。

本规范适用于医疗保障信息平台相关业务应用子系统和业务中台的建设。

2 术语和定义

2.1

架构 architecture

有关软件整体结构与组件的抽象描述，用于指导大型软件系统各个方面的设计。

2.2

医疗保障应用框架 Healthcare Security Application Framework（HSAF）

为了实现医保信息化统一技术架构标准目标,为业务子系统提供技术架构标准所要求的基础功能软件产品和服务。采用分布式云架构，封装核心云支撑服务适配接口，用于实现云产品解耦设计。

3 缩略语

下列英文缩略语适用于本文件。

HSAF 医疗保障应用框架（Healthcare Security Application Framework）
IaaS 基础设施即服务（Infrastructure-as-a-Service）
PaaS 平台即服务（Platform-as-a-Service）
Web 全球广域网（World Wide Web）
API 应用程序编程接口(Application Programming Interface)
SDK 软件开发工具包（Software Development Kit）
SQL 结构化查询语言(Structured Query Language)
TCP 传输控制协议（Transmission Control Protocol）
HTTP 超文本传输协议(HyperText Transfer Protocol)
HTTPS 超文本传输安全协议(HyperText Transfer Protocol Secure)
XML 可扩展标记语言（Extensible Markup Language）
JSON Java 脚本对象简谱(JavaScript Object Notation)
ORM 对象关系映射（Object Relational Mapping）
JWT JSON Web 令牌（JSON Web Token）
IoC 控制反转（Inversion of Control）
DI 依赖注入（Dependency Injection）

AOP 面向切面编程（Aspect Oriented Programming）

OLTP 联机事务处理过程（On-Line Transaction Processing）

HA 高可用（High Available）

ECS 阿里云服务器（Elastic Compute Service）

HSF 阿里云淘宝服务框架 （High-speed Service Framework）

EDAS 阿里云企业级分布式应用服务（Enterprise Distributed Application Service）

DRDS 阿里云分布式关系型数据库服务（Distributed Relational Database Service）

OSS 阿里云对象存储服务（Object Storage Service）

TSF 腾讯微服务平台（Tencent Service Framework）

CMQ 腾讯云消息队列（Cloud Message Queue）

TDSQL 腾讯云分布式数据库服务（TencentDB for TDSQL）

CLS 腾讯云的日志服务（Cloud Log Service）

ELK Elasticsearch、Logstash 和 Kibana 简称

RPC 远程过程调用（Remote Procedure Call）

4 总体技术架构

4.1 总体技术架构

总体技术架构参见图 1 及图 2。系统总体技术架构采用分布式云架构，在基础设施层上，结合云平台，提供分布式服务支撑。通过业务中台构建业务中心，支持实时交易型应用；通过数据中台实现数据汇聚、数据治理等，开展大数据应用。基于统一的应用分层架构建设经办管理类、公共服务类、智能监管类、宏观决策类业务子系统应用。

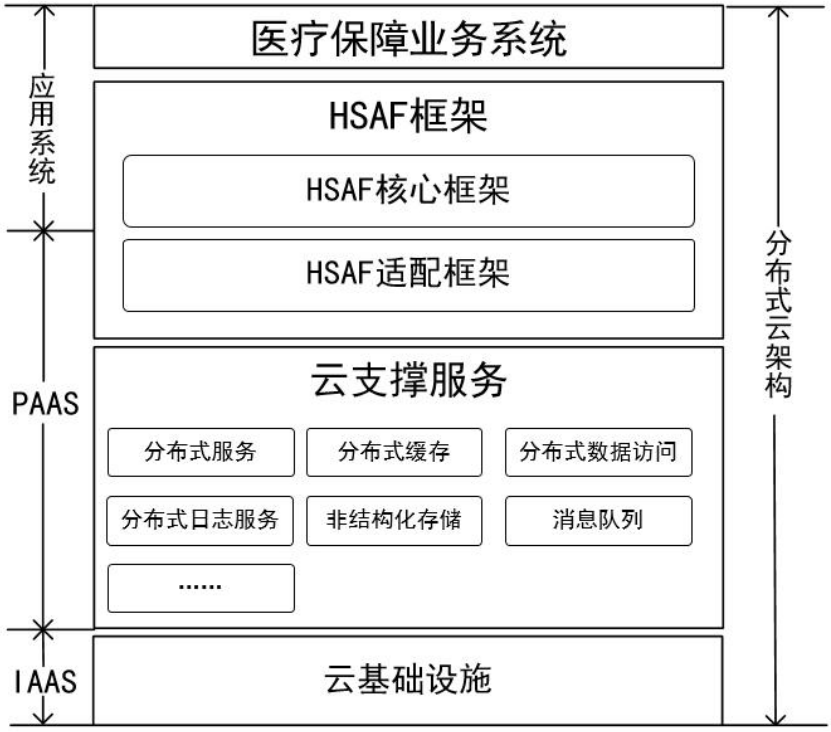

图 1 总体技术架构——概念图

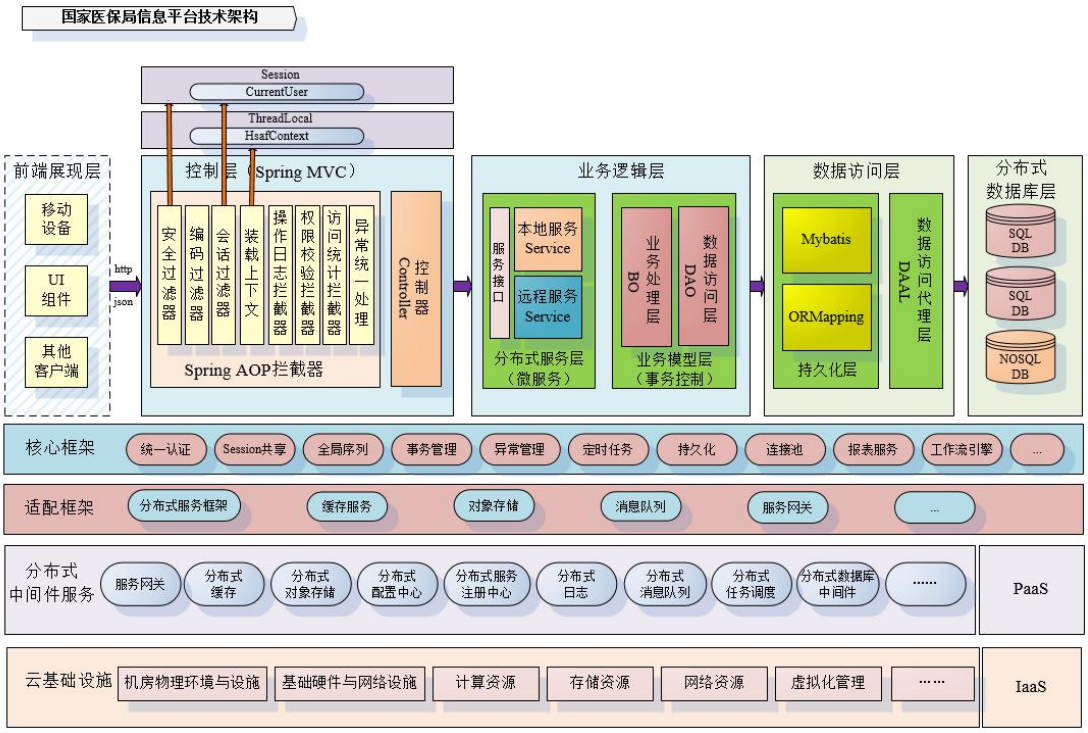

图 2 总体技术架构——逻辑图

总体技术架构描述如下：

a) 业务系统：基于医疗保障应用框架（Healthcare Security Application Framework，简称：HSAF）开发的支撑医疗保障业务运行的应用子系统；

b) HSAF 框架：采用分布式云架构，封装核心云支撑服务适配接口，用于实现云产品解耦设计，详见 4.3；

c) 适配层：基于 HSAF 的适配技术，将应用层依赖的分布式技术与具体厂商的分布式技术进行适配，实现应用层可以适配多家厂商的分布式技术；

d) 云支撑服务层：基于云基础设施，为应用层提供通用的技术支撑服务，包括分布式服务、分布式缓存、分布式数据访问、日志服务、非结构化存储和消息队列等；

e) 云基础设施层：采用云架构，在物理设备基础上，实现计算资源、存储资源、网络资源的动态管理和资源调配。

4.2 架构设计思路

为满足全国医疗保障信息系统部署模式的可灵活选择要求，相对传统系统技术架构，医疗保障信息平台在架构中增加了“适配层”，将应用层依赖的分布式技术与具体厂商的分布式技术进行适配，实现应用层可以适配多家厂商的分布式技术。地方可根据实际情况向各个云资源提供商（包括政务云和专有云等）租用或申请资源使用，也可自建数据中心。云基础设施建设参见图 3。

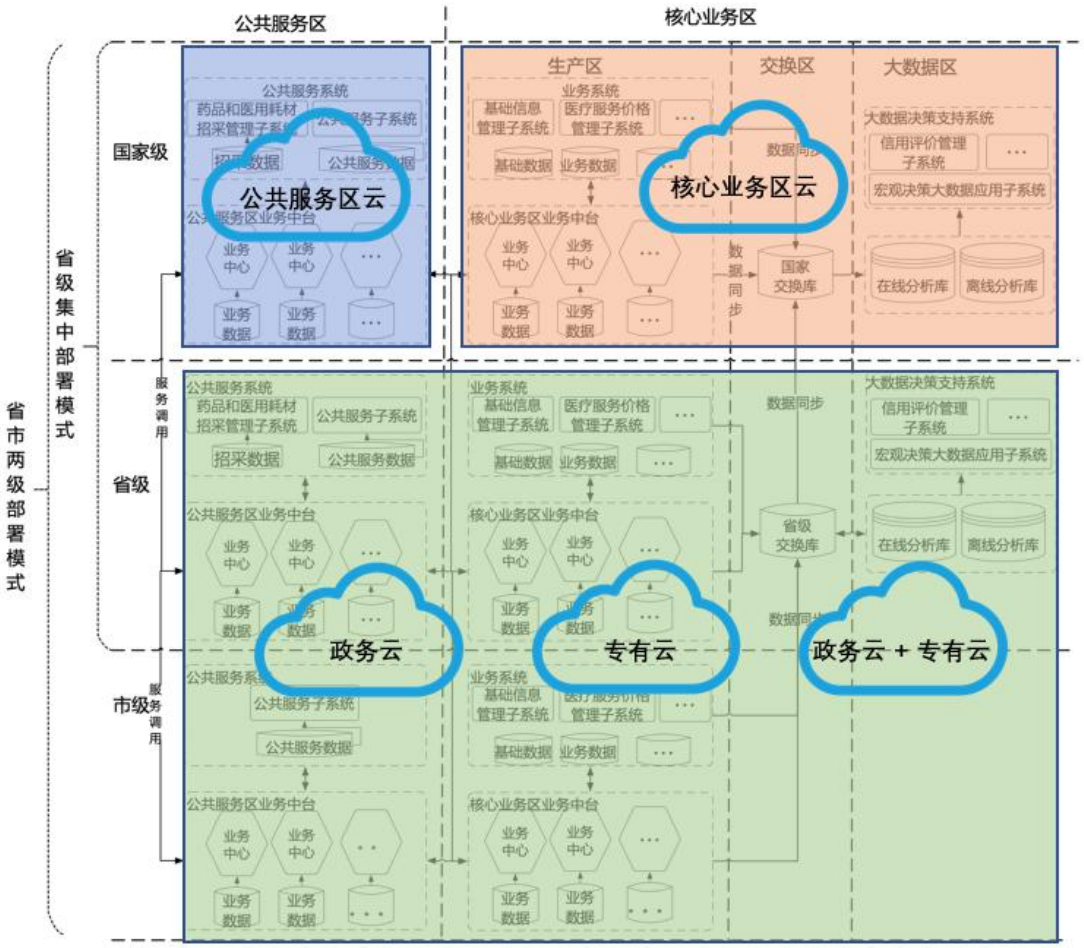

图 3 云基础设施建设

在总体技术架构设计和代码开发时，应遵循一个重要原则：**框架需满足技术扩展性的要求，当前框架能适配至少三种云平台分布式技术，新的分布式技术的加入可以通过框架的扩展实现。**

4.3 应用技术分层架构

为了保证业务子系统应用具有良好的横向扩展能力，以应对未来业务的快速发展，整个应用架构采用分布式云架构设计，业务能力通过微服务框架基于高内聚低耦合的思路实现。所有服务均为无状态服务，实现在线应用的扩缩容能力。应用分层架构参见图 4。

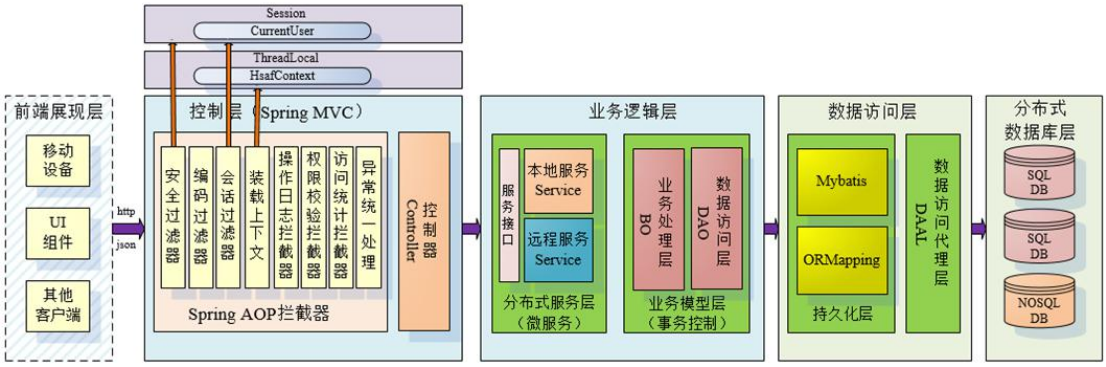

图 4 应用技术分层架构

业务应用子系统分为客户端和服务器端两大部分：

a） 客户端：前端展现层；

b） 服务器端：

——控制层；

——业务逻辑层；

——数据访问层；

——分布式数据库层。

业务应用子系统需严格按照该分层架构和调用层次进行应用程序开发和服务调用。

4.4 HSAF 框架

4.4.1 概述

HSAF 框架设计采用 1+N 模式，即 1 套核心框架，多套云支撑服务技术平台适配。HSAF 框架中包含控制层、业务层和持久化层接口抽象，定义了分布式服务框架、分布式缓存、分布式消息队列、非结构化存储、日志服务等分布式中间件服务的接口。

基于 4.2 架构设计思路，HSAF 框架分为两部分：

a） HSAF 核心框架，提供业务子系统运行相关的基础框架和服务支撑；

b） HSAF 适配框架，提供对不同云支撑服务技术平台的适配和可移植支撑。

HSAF 框架总体结构参见图 5。

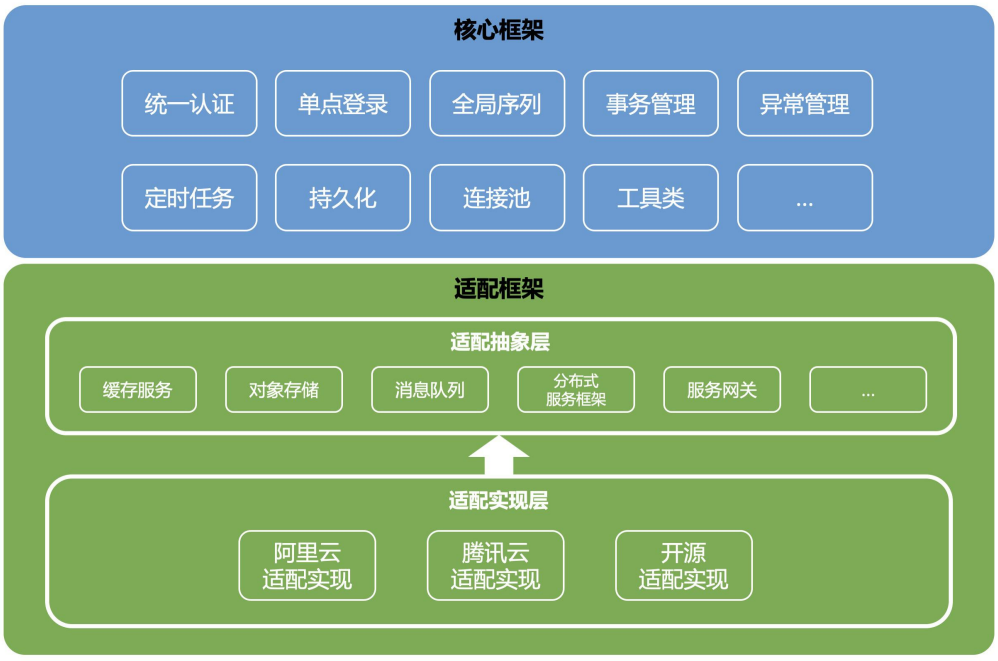

图 5 HSAF 框架结构

4.4.2 HSAF 核心框架

HSAF 核心框架，统一封装了 Spring 框架相关组件、ORM 持久化框架、数据库连接池组件、会话上下文、异常统一拦截器、操作日志拦截器、权限认证拦截器、安全过滤拦截器等开发基础服务，能使业务系统开发人员尽可能只关注业务逻辑，而无需过多关注技术细节。它主要提供了以下功能和服务：

a） 统一认证服务（详见 6.1）；

b) 单点登录服务（详见 6.2）；
c) 全局 ID 序列服务（详见 6.3）；
d) 事务管理（详见 6.4）；
e) 异常管理（详见 6.5）；
f) 定时任务（详见 6.6）；
g) 持久化服务（详见 6.7）；
h) 数据库连接池服务（详见 6.8）；
i) 报表服务（详见 6.9）。

4.4.3 HSAF 适配框架

4.4.3.1 适配抽象层

适配抽象层是对适配层中的分布式服务的具体技术实现进行抽象，提供了分布式缓存、远程服务调用、非结构化存储、消息队列等 PaaS 层服务的抽象接口。在代码中，应基于抽象接口层进行开发，而不能直接使用具体的扩展派生类，否则代码将绑定某种具体技术方案。适配抽象层设计详见 7.1。

4.4.3.2 适配实现层

适配实现层基于适配抽象层，扩展了分布式服务的具体技术实现。医疗保障信息平台适配的技术实现，已预设提供以下三套方案：阿里云专有云产品（详见 7.2）、腾讯云专有云产品（详见 7.3）、开源产品中的一种选型（详见 7.4）。各地医保系统做技术合规选型时，如采用 HSAF 框架预设适配实现技术之外的技术，应做相应适配、开发工作。

4.5 远程服务调用

远程服务调用是分布式服务框架的基础和核心功能。目前主流的分布式服务框架使用两种远程服务调用协议：

a) RPC 协议，以 Dubbo、Thrift、gRPC、rpcx、Motan 为代表的框架使用的协议；
b) HTTP 协议，以 Spring Cloud 为代表的框架使用的协议。

两种协议在不同的分层上提供服务，RPC 协议是在 Service 层提供服务，HTTP 协议是在 Controller 层提供服务。

为了兼容两种不同的服务提供方式，实现一套业务代码适配不同的分布式服务框架所使用的远程服务调用协议，如图 6 所示，架构要求如下：

a) 对于 RPC 协议，通过提供不同的服务注册发现配置文件来实现协议的切换；
b) 对于 HTTP 协议，Service 服务需要额外包装一层 Controller 层来实现服务的协议转换。

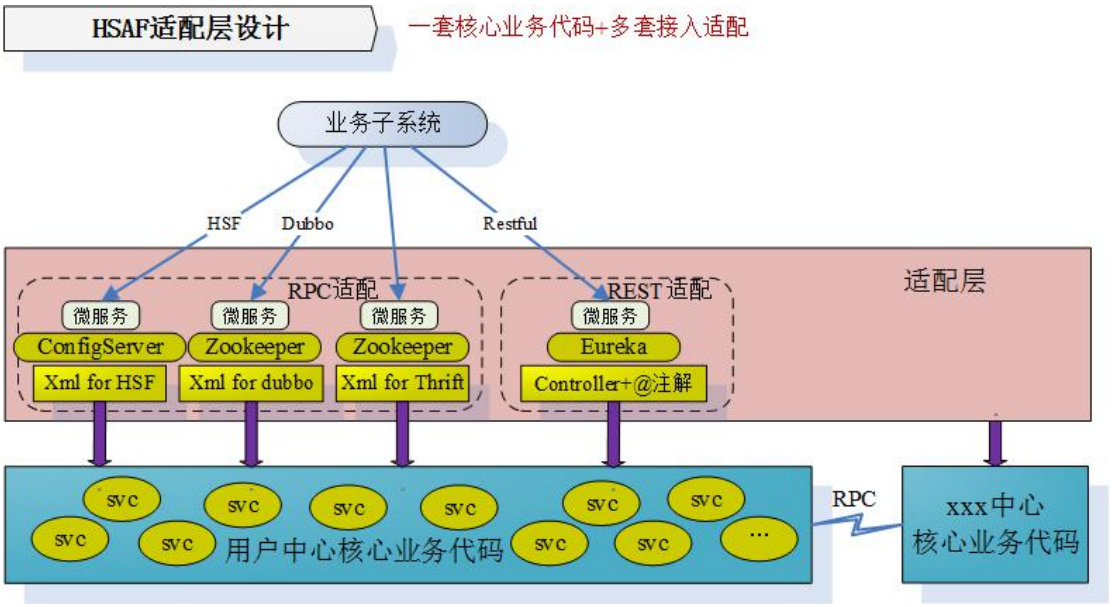

图 6 远程服务调用不同协议适配方案

5 应用技术分层架构

5.1 客户端

5.1.1 概述

客户端包含 PC 端、移动端、大屏幕端等类型。客户端与服务器端之间的通讯协议为 HTTP 协议，交互的数据格式为 JSON 格式。

5.1.2 前端展现层

前端展现层泛指一切在客户端直接与用户交互的客户界面（UI），本层是 MVC 架构中的视图层（V）。各类型客户端使用各自的组件作为前端展现层的 UI 实现方式。前端展现层将用户对 UI 的操作转化成基于 HTTP 协议的 JSON 格式的字符串去访问服务器，并将服务器返回的数据通过各自的组件展现给用户。

服务器端能响应符合 HTTP/JSON 的业务请求，客户端应发出符合业务要求的 HTTP/JSON 请求，由服务器端处理和响应。

HSAF 框架在浏览器上的前端展现层实现是参考实现。HSAF 框架提供了基于 PC 端的前端展现层实现，其他类型客户端部分需要开发商自行实现。

5.2 服务器端

5.2.1 概述

服务器端包含控制层、业务处理逻辑层、数据访问层和分布式数据库层四部分。

5.2.2 控制层

控制层包含过滤器拦截器层、控制器层（Controller）两部分。

过滤器拦截器层主要处理全局性问题，一切访问都会经过过滤器拦截器层处理，不会绕过过滤器拦截器直接访问控制层。本层次提供的能力包括：

a) 分布式会话管理：用户会话信息统一写入分布式缓存中；
b) 装载上下文信息；
c) 记录操作日志；
d) 安全管理：通过一个过滤器链拦截进入的请求，并将这些请求转给认证和访问决策管理器处理，从而增强安全性。

控制器层负责请求的全生命周期处理，包含接收——分发——调用业务——视图展现的全过程。HSAF 的核心控制框架是围绕前端控制器（DispatcherServlet）展开的，前端控制器负责将请求派发到特定的控制器。当控制器类接收到一个请求时，它会在自己内部寻找一个合适的处理方法来处理请求。控制器在选择好适合处理请求的方法时，传入收到的请求(根据方法参数类型，可能以不同的类型传入)，并且调用该方法中的逻辑来进行处理。方法逻辑会调用业务逻辑。业务逻辑运行完毕之后，会委派给一个视图，由该视图来处理方法的返回值。返回的视图名称会返回给前端控制器，它会根据一个视图解析器将视图名称解析为一个具体的视图实现。

5.2.3 业务逻辑层

业务逻辑层包含 Service、BO 两个子层次。在 View、Controller、Service、BO 这三个层次之间通过 DTO 进行业务对象的传递，在 BO 与 DAO 之间通过 DO 进行数据对象的传递。Service 是服务的发布层，提供对外服务的接口，并调用 BO 完成接口任务。BO 层实现具体的业务逻辑。DAO 层书写数据库访问逻辑，调用持久化层实现数据库的访问工作。

5.2.4 数据访问层

数据访问层分为持久化层和数据访问代理层两部分。

持久化层实现 O/R Mapping 的工作并调用分布式数据库。

数据访问代理层统一接收数据访问层的请求并对请求进行解析、优化、路由、分发给分布式数据库，提供对分库分表、读写分离的透明支持，并且提供对跨库信息进行合并等操作，将数据库结果返回给数据访问层。

5.2.5 分布式数据库层

分布式数据库层通过分布式数据访问代理服务，访问分布式关系型数据库，能使多个关系型数据库工作如在同一个关系型数据库一样。

5.3 框架层次调用

5.3.1 对象模型定义

分层中涉及的对象模型定义如下：

a) DTO(Data Transfer Object):数据传输对象，Service 或 Manager 向外传输的对象；
b) BO(Business Object):业务对象，由 Service 层输出的封装业务逻辑的对象；
c) DO(Data Object):数据对象，此对象与数据库表结构一一对应，可扩展虚拟字段（如一些查询条件、计算字段、汇总信息等），通过 DAO 层向上传输数据源对象。

5.3.2 调用顺序

框架层次调用参见图 4，框架层次调用顺序如下：

a) 客户端浏览器发起 HTTP/JSON 请求；
b) 服务器端过滤器过滤请求，做安全、编码、session 管理等处理，拦截器拦截请求做记录日志、访问统计等操作，并装载上下文信息；

c） 控制器接收前端传过来的 DTO 对象，并调用本地服务或者远程服务；
d） Service 分为接口与实现两部分。Service 服务接口接受控制层的远程或本地调用，接收 DTO 对象，调用本地 BO 中的业务逻辑，并将 BO 返回的业务结果（DTO 对象）返回给控制层（远程或本地客户端）；
e） BO 分为接口与实现两部分。BO 服务接口接受 Service 的本地调用，接收 DTO 对象，实现业务逻辑，使用 DO 对象调用 DAO 来访问数据库，将 DAO 返回 DO 对象转换为 DTO 对象，返回给 Service；
f） DAO 层接受 BO 层的调用，接收 DO 对象，实现具体的 SQL 逻辑，访问数据库，并将返回的数据模型返回给 BO；
g） 数据访问层实现数据持久化工作，将数据库的库表结构与 java 对象做映射；
h） 数据访问代理层接受持久化层的数据访问，对数据访问进行数据操作的解析和执行，包括会话管理、分库分表策略、语义解析、请求路由、数据合并、切换控制等，数据访问代理层将数据操作处理后分派到具体的分布式数据库；
i） 分布式数据库层接受数据访问代理层的访问，进行数据库处理，将数据返回给数据访问代理层。

5.3.3 调用要求

各层次的调用顺序及对象模型的传输，应遵循 5.3.2 所述。
对象模型传输示意参见图 7。

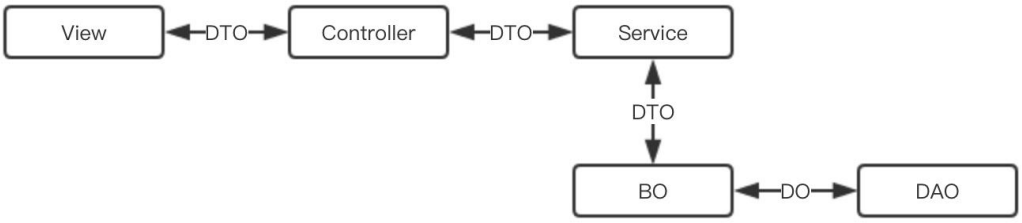

图 7 对象模型传输示意图

6 核心框架设计

6.1 统一认证服务

6.1.1 概述

用户认证指验证某个用户是否为系统中的合法主体，即判断用户能否访问该系统。用户认证一般要求用户提供用户名和密码。系统通过校验用户名和密码来完成认证过程。

6.1.2 认证技术

HSAF 框架中采用 OAuth 2.0 标准、JWT（JSON Web Token）进行登录认证，使用 Spring Security 框架完成相关的认证和授权验证。

6.1.2.1 OAuth

OAuth 是一种开放的协议，为桌面、手机或 Web 应用提供了一种简单的、标准的方式去访问需要用户授权的 API 服务。

OAuth2.0 具有以下特点：

a) 简单：不管是 OAuth 服务提供者还是应用开发者，都很容易于理解与使用；

b) 安全：没有涉及到用户密钥等信息，更安全更灵活；

c) 开放：任何服务提供商都可以实现 OAuth，任何软件开发商都可以使用 OAuth。

OAuth 2.0 定义了四种授权方式：

a) 授权码模式（Authorization Code）；

b) 简化模式（Implicit）；

c) 密码模式（Password）；

d) 客户端模式（Client Credentials）。

OAuth 2.0 定义了以下几种角色：

a) Resource Owner：资源拥有者；

b) Resource Server：资源服务器；

c) Client：第三方应用客户端；

d) Authorization Server：授权服务器。

6.1.2.2 JWT

JWT 是一个开放标准（RFC 7519），它定义了一种紧凑且独立的方式，用于在各方之间作为 JSON 对象安全地传输信息。此信息可以通过数字签名进行验证和信任。JWT 可以使用秘密（使用 HMAC 算法）或使用 RSA 或 ECDSA 的公钥/私钥对进行签名。

6.1.2.3 Spring Security

Spring Security 是一个能够为基于 Spring 的企业应用系统，提供声明式的安全访问控制解决方案的安全框架。它提供了一组可以在 Spring 应用上下文中配置的 Bean，充分利用了 Spring IoC，DI 和 AOP 功能，为应用系统提供声明式的安全访问控制功能，减少了为企业系统安全控制编写大量重复代码的工作。

6.1.3 认证服务

6.1.3.1 认证服务场景

认证服务提供以下三种场景的认证能力：

a) 内部统一门户认证；

b) 客户端应用认证；

c) 开放授权认证。

6.1.3.2 内部统一门户认证

内部统一门户子系统为医保局业务经办工作人员提供统一登录入口，认证服务能提供统一的身份认证能力和单点登录服务（详见 6.2）。

6.1.3.3 客户端应用认证

公共服务子系统涉及移动 APP、网厅、短信、微信/小程序、自助终端等客户端，认证服务提供客户端统一身份认证能力。如图 8 所示，认证服务提供的客户端统一身份认证流程如下：

a) 用户登录客户端；

b) 客户端向认证服务请求认证；

c) 认证服务使用密码模式进行认证；

d) 认证服务认证通过，生成 JWT 格式的 access_token，返回给客户端；
e) 客户端存储 JWT，返回登录成功；
f) 用户在客户端进行后续操作；
g) 客户端携带 JWT 向 API 服务请求 API 接口；
h) API 服务验证 JWT 的有效性，验证通过则执行后续逻辑，返回结果；
i) 客户端向用户展示结果。

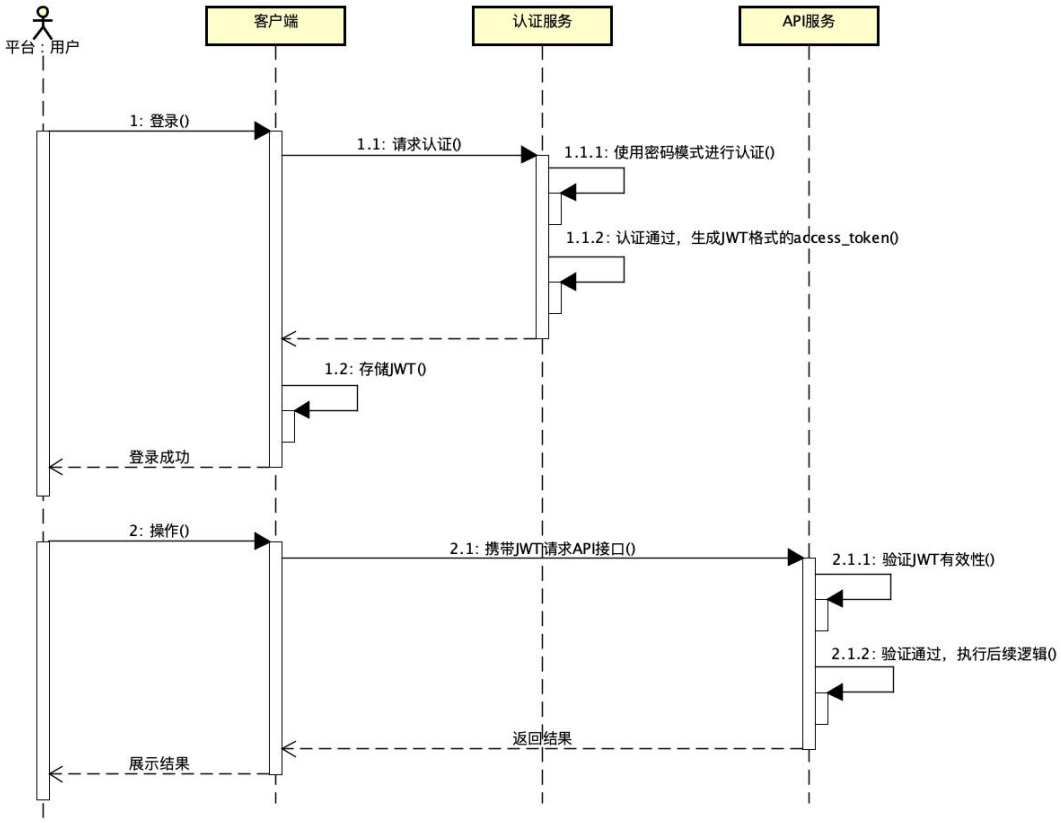

图 8 客户端应用认证流程

6.1.3.4 开放授权认证

基于 OAuth 标准，认证服务能提供开放开发授权认证的能力。如图 9 所示，开放授权认证流程如下：

a) 用户（Resource Owner 角色）向第三方系统发起请求；
b) 第三方系统（Client 角色）向用户返回用户授权认证操作；
c) 用户输入平台认证信息，向认证服务（Authorization Server 角色）请求授权认证；
d) 认证服务认证通过，生成 code，并跳转到授权确认页面；
e) 用户同意授权；
f) 认证服务生成 code 并跳转到重定向页面；
g) 第三方系统使用 code 请求认证服务的 token 接口；
h) 认证服务生成 JWT 格式的 access_token 并返回；
i) 第三方系统本地存储 access_token；
j) 第三方系统后续使用 access_token 向开放资源服务（Resource Server 角色）请求开放资源，并经过处理后展现给用户。

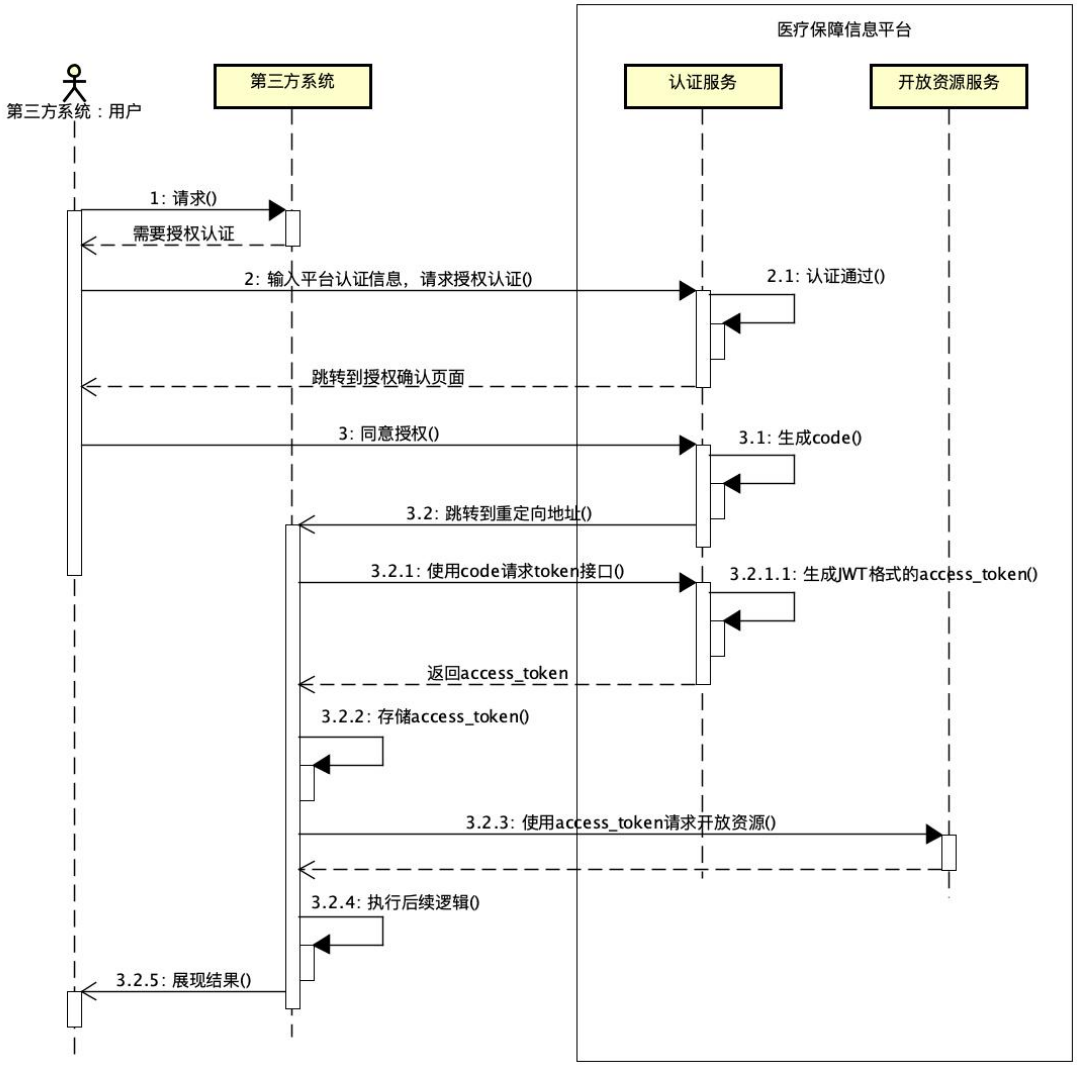

图 9 开放授权认证场景

6.2 单点登录服务

6.2.1 实现方式

单点登录（Single Sign On）服务是指在多个相互信任的应用系统中，用户只需要登录一次就可以访问所有相互信任的应用系统的服务。

系统采用Session共享方案来实现单点登录。共享Session可谓是实现单点登录最直接、最简单的方式。将用户认证信息保存于 Session 中，即以 Session 内存储的值为用户凭证，这在单个站点内使用是很正常也很容易实现的，而在用户验证、用户信息管理与业务应用分离的场景下即会遇到单点登录的问题，在应用体系简单，子系统很少的情况下，可以考虑采用 Session 共享的方法来处理这个问题。

Session 服务用于存储与用户相关的数据，默认由 Web 容器进行管理。单机情况下，所有用户的 Session 数据由一台服务器持有，不存在 Session 数据不一致的情况，但在分布式情况下，用户的前后两次请求可能会转发到不同的后端服务器上，如果不进行 Session 共享会出现 Session 数据不一致的情况。因此打造一个高可用的分布式系统，应将 Session 管理从容器中独立出来成为统一的脱离容器的 Session 存储机制。HSAF 框架的 Session 管理通

过基于 Redis 服务器的方案来实现 Session 的共享存储。

6.2.2 单点登录流程

如图 10 所示，单点登录流程如下：

a) 接收用户认证信息；
b) 将用户认证信息提交认证中心进行认证；
c) 认证成功生成有效凭证并通过用户中心获取对应用户详细信息；
d) 将用户信息写入 Session 对象，保存至 Session 共享服务中；
e) 通过 Session 共享服务获取认证服务共享的 Session 信息，完成单点登录；
f) 通过用户过滤器获取当前登录用户，并将用户信息及所属机构信息写入线程对象 HsafContextHolder 中。

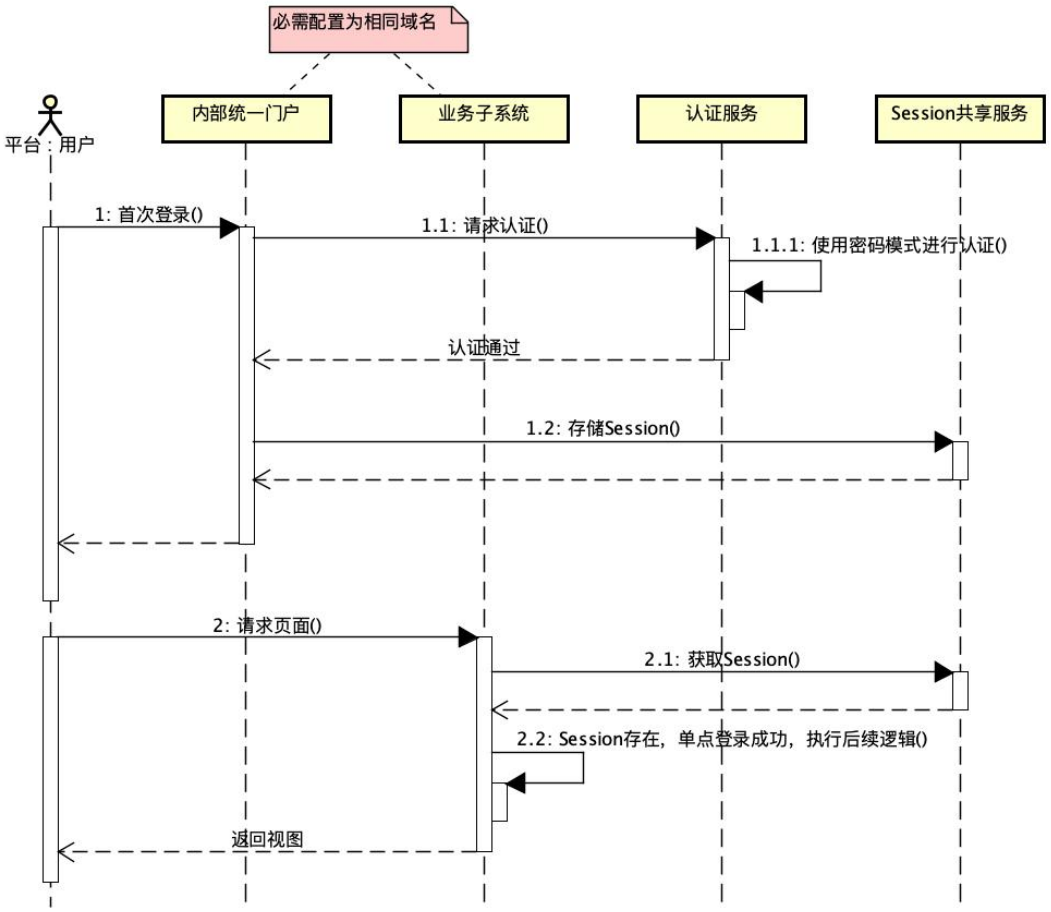

图 10 单点登录流程

6.2.3 应用要求

为保证医疗保障信息平台各个业务子系统 Web 应用的 Session 共享，避免出现因跨域导致的 Session 不能共享问题，各业务子系统 Web 应用应配置相同的域名。

Session 共享服务应使用独立的缓存服务器进行存储，与业务使用的缓存服务器完全分离，互不影响。

6.3 全局 ID 序列服务

全局唯一序列号生成是在设计一个分布式系统时常常会遇见的需求。在分布式系统架构

下，尤其是数据库使用分库分表的时候，数据库自增主键已无法保证全局唯一。

HSAF 框架提供了统一的组件，实现全局 ID 序列生成服务，可被直接调用生成全局序列 ID。

6.4 事务管理

6.4.1 实现方式

HSAF 框架使用了 Spring 框架的事务管理机制。Spring 支持编程式事务管理和声明式事务管理两种方式。声明式事务管理是非代码侵入式的开发方式，这是 Spring 所倡导的开发方式。声明式事务管理也有两种常用的方式，一种是基于 tx 和 AOP 命名空间的 XML 配置文件，另一种就是基于@Transactional 注解。

6.4.2 应用要求

为了方便系统灵活地进行事务隔离等级、事务传播行为、事务超时、事务回滚规则等参数的控制，应使用基于 XML 配置文件的声明式事务管理方式，不应使用基于@Transactional 注解的方式。

另外，根据分层设计，Service 是服务的发布层，BO 层是具体的业务逻辑实现层，Service 层提供对外服务的接口，并调用 BO 层完成接口任务。架构要求事务只能声明在 BO 层。

6.5 异常管理

6.5.1 异常类设计

HSAF 框架统一异常处理，实现对控制层、业务处理层及数据访问层的异常捕获和异常信息封装。业务开发人员只需要在各个层次中，抛出对应的业务异常和数据访问异常，框架会进行统一拦截，不需要开发人员手动进行异常处理，可减少代码量，提高开发效率，同时也实现了整个系统的异常统一处理和管理。

异常类设计如图 11 所示。框架所有的异常均继承自“cn.hsa.hsaf.framework.exception.AppException”这个基类，每个业务子系统（或者业务中台）定义自己的异常类（统一继承自 BusinessException）。系统运行过程中，在控制层（Controller）、业务服务层（Service）、业务逻辑层(BO)、数据访问层(DAO)如果发生业务异常（BusinessException）、数据访问异常(DataAccessException)或者其它运行时异常，SpringMVC 调用框架提供的 HSAFExceptionHandler 对异常信息进行统一拦截处理，记录异常日志，并将异常信息经业务封装后返回 UI 层进行提示。

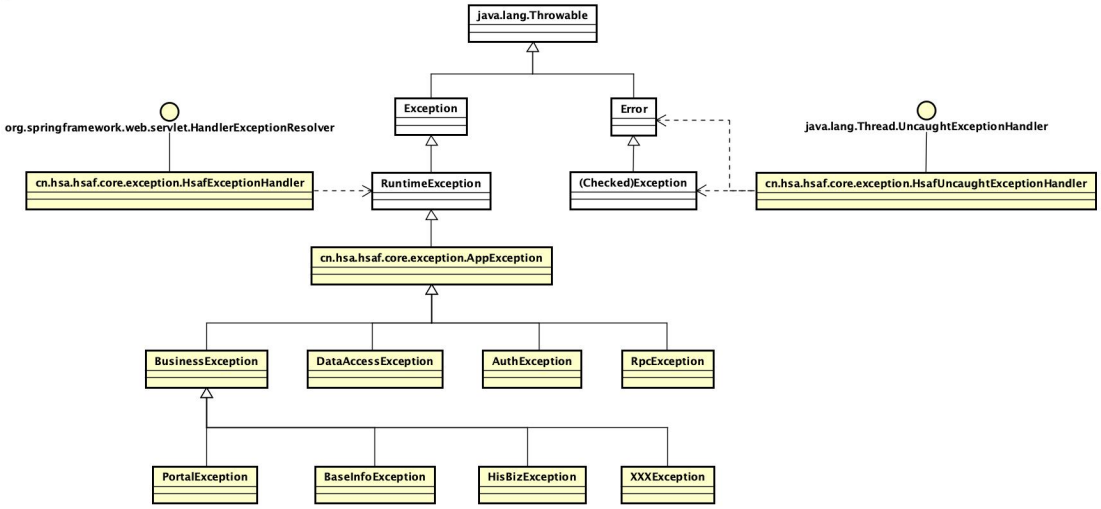

图 11 异常类设计

6.5.2 异常错误码

框架采用前后端分离技术，前端通过 HTTP 请求后台服务，数据统一使用 JSON 格式，服务端响应数据使用统一的规范格式和状态码：

示例：

```
{
    "type":"success",
    "code": 0,
    "message": "这里显示错误简短信息",
    "data":{
    }
}
```

如图 12 所示，为了方便问题的定位排查，以及服务质量的统计分析，在 HSAF 框架的异常基类中定义错误码 code 属性，应用代码抛出异常时，需要定义该属性。

```
public class AppException extends RuntimeException {
    private static final long serialVersionUID = -7611643172712984323L;

    //异常错误码，详见开发指南中的错误码定义
    int code;

    public AppException(final int code) {
        super();
        this.code=code;
    }

    public AppException(final int code,final String message) {
        super(message);
        this.code=code;
    }
```

图 12 错误码定义

业务异常错误码采用六位字符串，格式为“两位系统编号+两位模块编号+两位异常编号”，具体的异常错误码见表 1。

表 1 错误码使用范围说明

错误码	使用范围	说明
0	系统级	成功
-1	系统级	未知异常
-2	系统级	请求参数异常
-3	系统级	服务端请求超时
-4	系统级	权限校验异常
-5	系统级	无效的请求地址
-6	系统级	触发限流
10xxxx	框架级	
11xxxx	内部统一门户	
12xxxx	基础信息管理	
13xxxx	医保业务基础	
14xxxx	支付方式管理	
15xxxx	医疗服务价格管理	
16xxxx	信用评价管理	
17xxxx	跨省异地就医	
18xxxx	药品和医用耗材招采	
19xxxx	公共服务	
20xxxx	基金运行及审计监管	
21xxxx	医疗保障智能监管	
22xxxx	运行监测	
23xxxx	宏观决策大数据应用	
24xxxx	内部控制	
25xxxx	业务中台	

6.5.3 不可捕获异常的处理

HSAFExceptionHandler 可以拦截处理所有应用层抛出的 AppException 异常的子类，但是系统运行过程中，还有一些“特殊”的异常是无法捕获的，原因是：

a) AppException 异常基类是继承自 RuntimeException，而 java 的异常根类为 Throwable，Throwable 的两个子类 Error 和 Exception，Exception 的两个子类

CheckedException 和 RuntimeException；因此能被捕获的实际上只是 RuntimeException 相关的异常（当然这能够处理大部分的异常情况），对于 Error 以及 CheckedException 无法捕获；

b) 正常的代码中一般使用 try...catch(AppException ex){}来捕获异常，执行过程中可能因为线程中断或阻塞了，导致 catch 块中的代码并没有正常的执行到。

为了统一对不可捕获异常的处理，HSAF 框架统一封装实现了 Thread.UncaughtExceptionHandler 接口，该接口声明了某一个线程执行过程中，对于未捕获的异常处理，如图 13 所示。

```
public class HsafUncaughtExceptionHandler implements Thread.UncaughtExceptionHandler{

    @Override
    public void uncaughtException(Thread t, Throwable e) {
        // 打印出现异常的线程和异常名称
        System.out.println("捕获到异常 : 线程名[" + t.getName() + "], 异常名[" + e + "]");
        // TODO 记录异常日志

        // 异常栈的信息
        e.printStackTrace();
        // TODO ... 如果对异常还需要做特殊处理,可以在此处继续实现处理方法
    }

}
```

图 13 未捕获的异常处理

如图 14 所示，应用请求的入口一般是 Controller，因此 HSAF 框架通过 AOP 切面的方式在请求的入口方法处设置当前线程的 UncaughtExceptionHandler，以处理当前线程中无法正常捕获的各种异常。

```
package cn.hsa.hsaf.core.framework.web.exception;

import org.aspectj.lang.JoinPoint;
import org.springframework.beans.factory.annotation.Autowired;

/**
 *
 * @ClassName: HsafUncaughtExceptionHandlerAspect
 * @Description: 不可捕获异常的统一处理，在controller入口代码处添加aop注入
 * @author: siniu
 * @date: 2019年8月12日
 *
 *  <aop:aspectj-autoproxy proxy-target-class="true"></aop:aspectj-autoproxy>

    <!-- 不可捕获异常的统一拦截处理 -->
    <bean id="hsafUncaughtExceptionHandlerAspect"
          class="cn.hsa.hsaf.core.framework.web.exception.HsafUncaughtExceptionHandlerAspect" />

    <aop:config>
        <aop:aspect ref="hsafUncaughtExceptionHandlerAspect" >
            <!-- 切面为Controller下的所有方法 -->
            <aop:pointcut id="performance" expression="execution(* cn.hsa.hsaf..controller.*Impl.*(..))" />
            <aop:before method="doBefore" pointcut-ref="performance"/>
        </aop:aspect>
    </aop:config>
 */
public class HsafUncaughtExceptionHandlerAspect {

    @Autowired
    private HsafUncaughtExceptionHandler hsafUncaughtExceptionHandler;

    public void doBefore(JoinPoint jp){
        Thread.setDefaultUncaughtExceptionHandler(hsafUncaughtExceptionHandler);
    }
}
```

图 14 AOP 切面设置当前线程的 UncaughtExceptionHandler

6.5.4 应用要求

国家或地方在开展医疗保障信息平台建设时，异常派生类应继承框架统一的异常类结构设计（如图 11 所示）。

代码开发中，除非有明确约定，否则对捕获的异常直接抛到上一层，由 HSAF 框架在控制层进行统一拦截。

6.6 定时任务

HSAF 框架基于 XXL-JOB 扩展定时任务组件，提供分布式定时任务支持。

XXL-JOB 是一个轻量级分布式任务调度平台，其核心设计目标是开发迅速、学习简单、轻量级、易扩展。现已开放源代码并接入多家公司线上产品线，接入场景如电商业务，O2O 业务和大数据作业等，易上手，开箱即用。

其设计思想为将调度行为抽象形成“调度中心”公共平台，而平台自身并不承担业务逻辑，“调度中心”负责发起调度请求。将任务抽象成分散的 JobHandler，交由“执行器”统一管理，“执行器”负责接收调度请求并执行对应的 JobHandler 中业务逻辑。因此，“调度”和“任务”两部分可以相互解耦，提高系统整体稳定性和扩展性。

6.7 持久化服务

HSAF 框架使用了 MyBatis 提供持久化服务。

MyBatis 是支持定制化 SQL、存储过程以及高级映射的优秀的持久层框架。MyBatis 避免了几乎所有的 JDBC 代码和手动设置参数以及获取结果集。MyBatis 可以对配置和原生 Map 使用简单的 XML 或注解，将接口和 Java 的 POJOs(Plain Old Java Objects，普通的 Java 对象)映射成数据库中的记录。

6.8 数据库连接池服务

数据库连接是一种关键的、有限的、昂贵的资源，对数据库连接的管理能显著影响到整个应用系统的伸缩性和健壮性，影响到应用系统的性能指标。为了减少数据库连接频繁创建、释放所产生的开销，应用系统开发中应采用数据库连接池技术。

数据库连接池负责分配、管理和释放数据库连接，它允许应用系统重复使用一个现有的数据库连接，而不是再重新建立一个；释放空闲时间超过最大空闲时间的数据库连接来避免因为没有释放数据库连接而引起的数据库连接遗漏。这项技术能明显提高对数据库操作的性能。数据库连接池的基本思想就是为数据库连接建立一个“池”。预先在池中放入一定数量的连接，当需要建立数据库连接时，只需从“池”中取出一个，使用完毕之后再放回去。可通过设定连接池最大连接数来防止系统过多地与数据库连接。

6.9 报表服务

报表就是用表格、图表等形式来动态显示数据，可以用公式表示为：“报表 = 多样的格式 + 动态的数据”。报表可以帮助使用者访问、格式化数据，并把数据信息以可靠和安全的方式呈现给使用者。报表是管理的基本措施和途径，是应用系统的基本业务要求，也是实施 BI 战略的基础。

框架中提供了统一的报表服务，主要功能是提供报表资源的管理，用户身份与权限管理，数字签章、服务转发及报表展现。报表服务管理的报表资源主要包含数据源、报表模板、报表输出结果等。

报表服务是独立运行的单独服务，可进行分布式部署，模板的存储采用分布式对象存储（OSS），全网任意节点均可访问。报表服务对外采用 WeB 和 RESTful 方式进行服务，WEB 用

于管理人员对报表进行资源及权限配置，RESTful 用于业务系统对接报表服务。报表服务支持多种数据导出格式，如 PDF、Excel、图片等进行存储，亦可通过 RESTful 方式调用报表将结果页面，其嵌入到业务系统中。

报表服务还提供了严谨的服务转发设置，能为每一张报表进行报表工具配置，当业务系统发起服务请求时，报表服务会根据配置进行转发，未配置的请求将会被作为非法请求丢弃并记录监控日志。

报表工具具有独立模板设计器，可视化地开发报表更方便、快捷，它具有如下特点：

a) 提供拖拽式、所见即所得的报表编辑器；
b) 提供多样的向导来简化复杂的报表设计任务；
c) 提供多样化的排版和格式化工具；
d) 报表可转换为 PDF、HTML、EXCEL 等格式；
e) 支持数据源调试；
f) 支持无限次数的撤消和重做；
g) 内置超过 20 种的图表支持；
h) 集成超过 15 种语言；
i) 提供报表模板与报表库样式管理；
j) 提供源文件的备份；
k) 提供文档结构浏览器。

7 适配框架设计

7.1 适配抽象层设计

7.1.1 分布式服务框架

分布式服务框架是分布式系统架构的基础。分布式服务关键不仅仅是分布式服务本身，而是一套治理框架，这种框架使得分布式服务可以独立的部署、运行、升级，还能实现分布式服务之间在结构上的“松耦合”，而在功能上表现为一个统一的整体，体现为统一风格的界面，统一的权限管理，统一的安全策略，统一的上线过程，统一的日志和审计方法，统一的调度方式，统一的访问入口等等。分布式服务框架具备如下能力：

a) RPC 远程过程调用；
b) 服务注册与发现；
c) 服务治理；
d) 负载均衡；
e) 消息总线，轻量级的 MQ 或 HTTP；
f) 链路跟踪；
g) 配置中心。

HSAF 采用配置文件的方法来切换不同的分布式服务框架。

7.1.2 分布式缓存

缓存的主要作用是降低应用和数据库的负载，提高系统性能、客户端访问速度。在架构和业务设计上，可以考虑将访问量较大、不经常修改的，比如字典表和系统参数，或对数据库性能影响较大的查询的结果进行缓存，提高系统整体性能。

HSAF 框架使用了 Spring Cache 框架作为缓存层的抽象框架。Spring Cache 不是一个具体的缓存实现方案，而是一个对缓存使用的抽象。HSAF 框架提供了基于 Redis 的接口实现，

能够兼容不同云平台基于 Redis 或提供类 Redis 接口服务的缓存方案。

分布式缓存服务抽象类接口如图 15 所示，定义 cn.hsa.hsaf.core.cache.HsafCacheManager 缓存统一管理器。

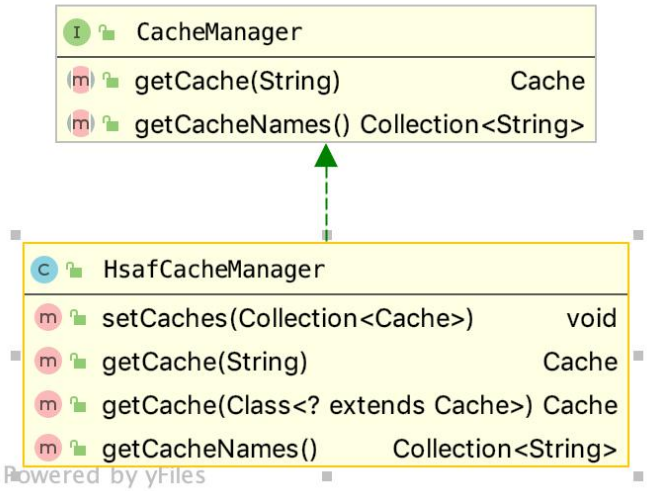

图 15 分布式缓存服务抽象类接口

7.1.3 分布式消息队列

分布式消息队列，是分布式系统中重要、常用的中间件，它使用事件驱动架构，通过在低耦合的模块之间传输事件消息，以保持模块的松散耦合，并借助事件消息的通信完成模块间合作。HSAF 引入并使用消息队列组件作为分布式服务架构下各个应用之间消息通讯的中间件。

分布式消息队列服务抽象类接口如图 16 所示，类设计：

a) 定义 cn.hsa.hsaf.core.mq.MQBusinessHandler 消息监听业务处理接口类；

b) 定义 cn.hsa.hsaf.core.mq.MQProducer 消息队列生产者接口类；

c) 定义 cn.hsa.hsaf.core.mq.MQConsumer 消息队列消费者接口类；

d) 定义 cn.hsa.hsaf.core.mq.MQMessage 消息实体类。

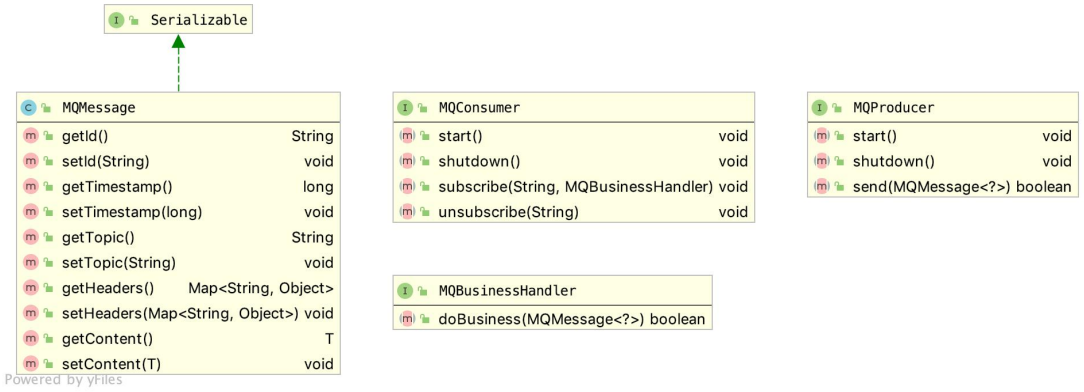

图 16 分布式消息队列服务抽象类接口

7.1.4 分布式数据库服务

在数据量逐渐增大的情况下。传统数据库存在着先天性的弊端，分布式数据访问代理服

务主要能使得传统数据库易于扩展，可切分，提升性能，规避单体结构缺陷。

分布式数据库屏蔽了后端的分布式数据库层的复杂性，能让用户从逻辑上得到与访问单机数据库近乎相同的体验。分布式数据访问代理服务本身是一个集群，支持 HA，避免了单点故障。分布式数据访问代理服务支持分库分表存储，支持横向和纵向切分，负责数据操作的解析和执行，提供会话管理、分库分表策略、语义解析、请求路由、数据合并、切换控制等功能。

7.1.5 非结构化存储

HSAF 定制一套相对通用的非结构化文档存储接口标准规范，一套接口多种实现，即一套管理操作 API，支持 AWS S3、MongoDB、OSS、TStack、FastDFS 等多种主流非结构化存储管理操作实现；存储环境切换，应用程序代码零调整，即只需简单参数变更即可完成应用程序的切换。选择使用面向 Java 方向的 AWS S3 网络存储服务协议。

非结构化存储服务抽象类接口如图 17 所示，类设计：

a) 定义了存储实体类 cn.hsa.hsaf.core.fs.FSEntity；

b) 定义了管理类 cn.hsa.hsaf.core.fs.FSManager，包含存储对象（putObject）、获取对象（getObject）、删除（deleteObject）三个接口。

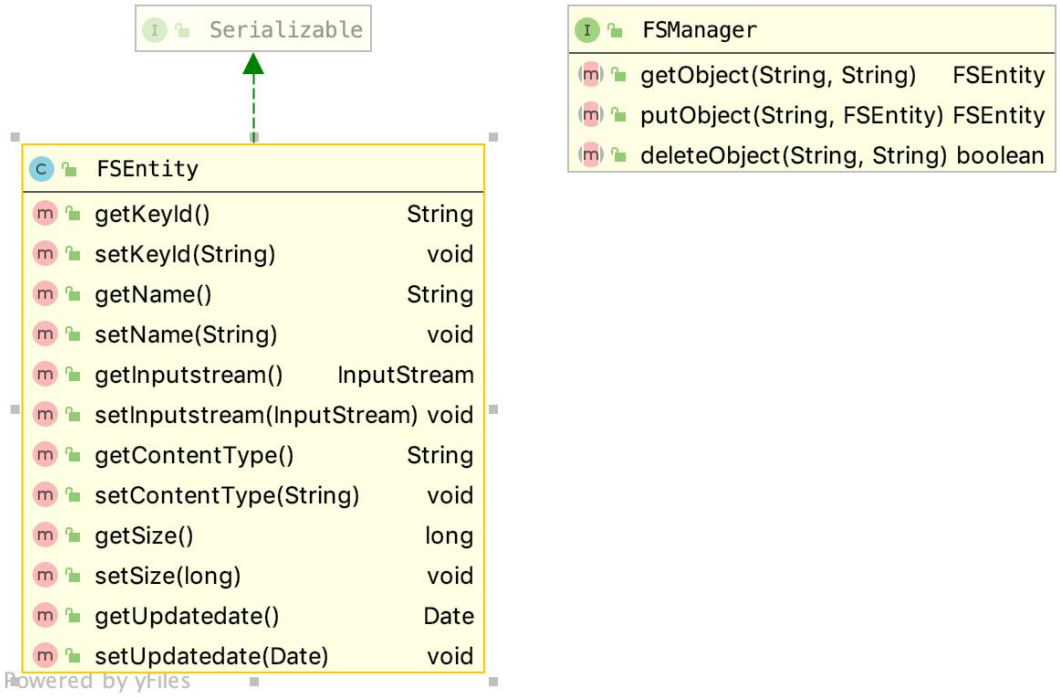

图 17 非结构化存储服务抽象类接口

7.1.6 日志服务

软件系统中存在着各式各样的日志，包括应用日志、业务日志、系统日志、访问日志、行为日志等。

日志服务能将分布式环境的日志统一收集聚合，存储在分布式数据库中，同时经过一定的处理可以使非结构化的文本内容结构化，以便于做结构化的查询和分析。日志服务提供以下主要能力：

a) 聚合：从多个数据源收集和发送日志的能力；

b) 处理：将日志消息转换为有意义的数据以便于分析的能力；

c) 存储：能够长时间存储数据，以便用于监控、趋势分析和安全性方面的使用场景；

d) 分析：通过查询数据并在其上创建可视化和仪表板来分析数据的能力。

7.2 阿里云适配实现设计

7.2.1 分布式服务框架

阿里云的分布式服务框架适配使用企业级分布式应用服务 EDAS（Enterprise Distributed Application Service）实现。

EDAS 是一个应用托管和微服务管理的 PaaS 平台，提供应用开发、部署、监控、运维等全栈式解决方案，同时支持 Dubbo、Spring Cloud 等微服务框架。

EDAS 拥有如下特点和功能：

a) 多样的应用托管平台：可以根据应用系统和资源需求，选择独享实例的 ECS 集群、基于 Kubernetes 的容器服务、Kubernetes 集群或 EDAS Serverless 部署并托管应用；

b) 丰富的微服务框架：可以基于原生 Dubbo、原生 Spring Cloud 和 HSF 开发应用及服务，并托管到 EDAS 中；

c) 完整的应用管理：可以使用 EDAS 控制台对应用系统进行全生命周期管理、服务治理及微服务管理：

   1) 应用生命周期管理：EDAS 提供应用全生命周期管理服务，包括应用的部署、扩容、缩容、停止和删除等；

   2) 服务治理：EDAS 集成了弹性伸缩、限流降级、流量管理、健康检查等服务治理组件，高效应对突发的流量洪峰和服务依赖所引发的雪崩问题，提高了平台的稳定性；

   3) 微服务管理：EDAS 提供服务拓扑、服务报表和调用链查询等功能，帮助管理分布式系统的中的每一个组件和服务。

d) 全面的监控和诊断：可以使用 EDAS 控制台实时监控应用中资源和服务的状态，以便及时发现问题，并通过日志和诊断组件快速定位：

   1) 应用监控：EDAS 实时监控应用的 IaaS 层资源、服务的健康状态，帮助您快速发现、定位问题；

   2) 应用诊断：EDAS 提供了基于容器的应用诊断功能，提供相应数据来判断 GC、类加载、连接器、对象内存分布、线程热点、Druid 连接池和 Commons Pool 等发生的运行问题。

7.2.2 分布式缓存

阿里云使用开源的 Redis4.0 作为分布式缓存实现，阿里云适配实现无需特别定制代码，统一使用开源版分布式缓存设计（详见 7.4.2），直接访问阿里云版 Redis 服务。

7.2.3 分布式消息队列

阿里云的分布式消息队列适配使用 RocketMQ。

RocketMQ 是阿里巴巴集团自主研发的专业消息中间件，基于高可用分布式集群技术，提供消息订阅和发布、消息轨迹查询以及定时（延时）消息、资源统计、监控报警等一系列消息云服务。

阿里云分布式消息队列类设计如图 18 所示，类设计：

a) 定义了消息队列消费者实现类 MQConsumerByOns；

b) 定义了消息队列生产者实现类 MQProducerByOns。

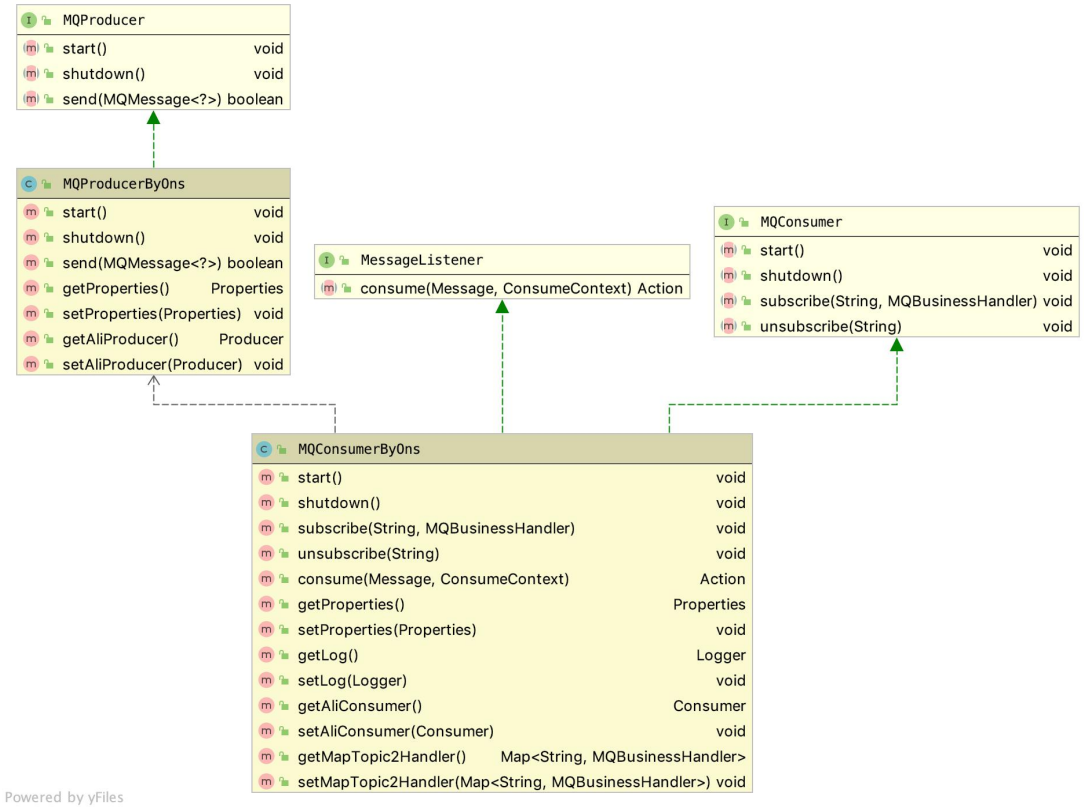

图 18 阿里云分布式消息队列类设计

7.2.4 分布式数据库服务

阿里云的分布式数据库服务适配使用 DRDS。

DRDS 是一款基于 MySQL 存储、采用分库分表技术进行水平扩展的分布式 OLTP 数据库服务产品，支持 RDS for MySQL 以及 POLARDB for MySQL。

7.2.5 非结构化存储

阿里云的非结构化存储服务适配使用 OSS。

阿里云对象存储服务（Object Storage Service，简称 OSS），是阿里云提供的海量、安全、低成本、高可靠的云存储服务。OSS 提供与平台无关的 RESTful API 接口。

阿里云非结构化存储类设计如图 19 所示，定义了对象存储 OSS 实现类 FSStoreAliManagerImpl。

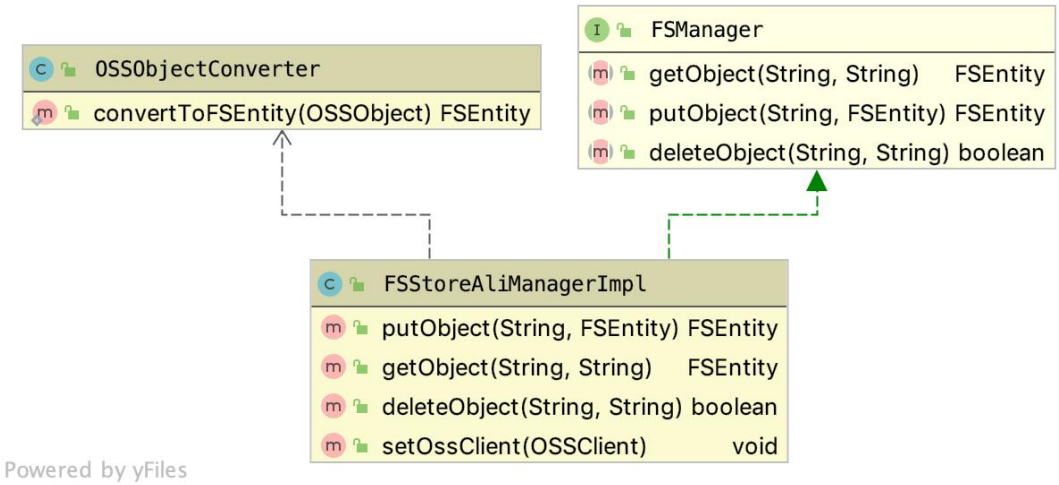

图 19 阿里云非结构化存储类设计

7.2.6 日志服务

阿里云的日志服务适配使用阿里云日志服务。

阿里云日志服务（Log Service，简称 LOG）是针对日志类数据的一站式服务，无需开发就能快捷完成日志数据采集、消费、投递以及查询分析等功能。Log Service 拥有如下特点和功能：

a) 实时采集与消费（LogHub）：用于数据清洗（ETL）、流计算（Stream Compute）、监控与报警、机器学习与迭代计算，具体如下：
   1) 通过 ECS、容器、移动端，开源软件，JS 等接入实时日志数据（例如 Metric、Event、BinLog、TextLog、Click 等）；
   2) 提供实时消费接口，与实时计算及服务对接；
b) 查询与实时分析（Search/Analytics）：提供实时索引、查询分析数据相关功能，可用于 DevOps/线上运维，日志实时数据分析，安全诊断与分析等；
c) 投递数仓（LogShipper）：提供稳定可靠的日志投递服务，能将日志中枢数据投递至存储类服务进行存储，支持压缩、自定义 Partition、以及行列等各种存储方式，能用于数据仓库与数据分析、审计、推荐系统与用户画像。

7.3 腾讯云适配实现设计

7.3.1 分布式服务框架

腾讯云的分布式服务框架适配使用腾讯微服务平台 TSF（Tencent Service Framework）。

TSF 是一个围绕着应用和微服务的 PaaS 平台，提供应用全生命周期管理、数据化运营、立体化监控和服务治理等功能。TSF 支持 Spring Cloud、Service Mesh 微服务框架。

TSF 以腾讯云中间件团队多款成熟的分布式产品为核心基础组件，提供秒级推送的分布式配置服务、链路追踪等高可用稳定性组件。此外，TSF 与腾讯云 API 网关和消息队列打通，方便松构建大型分布式系统。

TSF 拥有如下特点和功能：

a) 服务注册发现：TSF 服务注册发现包括三个角色：服务提供者、服务调用者和服务注册中心。服务提供者和服务调用者将地址信息注册到服务注册中心，并从服务注册中心获取所有注册服务的实例列表，服务调用者使用服务提供者的实例地址进行

调用；

b) 应用生命周期管理：TSF 提供从创建应用到运行应用的全程管理，功能包括创建、删除、部署、回滚、扩容、下线、启动和停止应用。TSF 提供部署组来实现应用的版本控制功能。TSF 将每次操作记录下来，用户可以在应用的变更记录页面中查看和搜索变更记录；

c) 分布式配置管理：配置管理包括应用配置、全局配置和文件配置。用户可以通过控制台进行分布式配置版本管理、发布配置到部署组或者命名空间范围内的实例；

d) 细粒度服务治理：TSF 提供服务级和 API 级别的服务治理能力，支持通过控制台配置服务路由、服务限流、服务鉴权规则。在 TSF 控制台中，用户可以通过配置、权重标签的形式进行细粒度的流量控制，实现灰度发布、就近路由、部分账号内测、流量限制、访问权限控制等功能；

e) 数据化运营：TSF 提供全面的监控和分布式调用链分析工具，帮助用户把握应用上线后的运行状况，并提供日志分析能力，具体说明如下：

   1) 监控包括应用监控，应用监控的指标包括应用的 QPS、请求时间和请求出错率等；

   2) 分布式调用链分析包括调用链查询和调用链详情，用户可以根据时间范围和服务名等条件查询一组调用链，调用链详情显示了请求经过每个服务的层次关系和耗时情况等信息；

   3) 通过日志分析能力，自动获取用户的业务日志并支持在 TSF 控制台上进行日志查看、日志检索，支持日志关键词告警功能，并提供日志与调用链联动排查线上问题。

f) 分布式事务：TSF 集成了分布式事务能力，支持 TCC 模式分布式事务管理功能。对于跨数据库、跨服务的分布式场景，用户可以通过控制台查看事务运行情况并进行超时事务处理，保证事务的一致性。

7.3.2 分布式缓存

腾讯云使用开源的 Redis 作为分布式缓存实现，腾讯云适配实现无需特别定制代码，统一使用开源版分布式缓存设计（详见 7.4.2），直接访问腾讯云版 Redis 服务。

7.3.3 分布式消息队列

腾讯云的分布式消息队列适配使用腾讯云消息队列 CMQ（Cloud Message Queue）。

CMQ 是一种分布式消息队列服务，它能够提供可靠的基于消息的异步通信机制，能够将分布式部署的不同应用（或同一应用的不同组件）之间的收发消息，存储在可靠有效的 CMQ 队列中，防止消息丢失。CMQ 支持多进程同时读写，收发互不干扰，无需各应用或组件始终处于运行状态。

腾讯云分布式消息队列类设计如图 20 所示，类设计：

a) 定义了消息生产者实现类 MQConsumerByCMQ；

b) 定义了消息消费者实现类 MQProducerByCMQ。

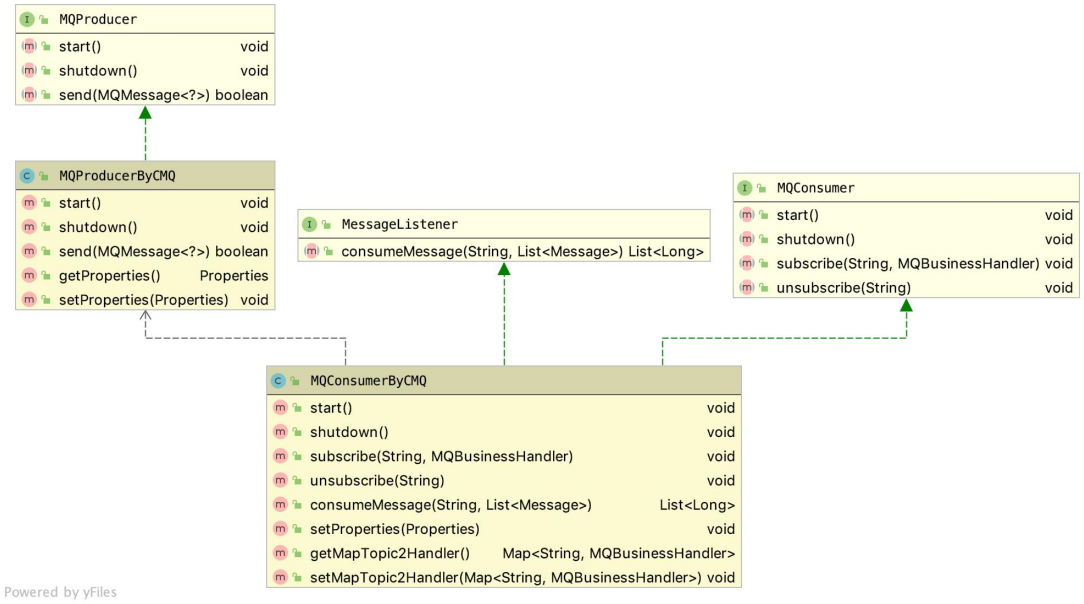

图 20 腾讯云分布式消息队列类设计

7.3.4 分布式数据库服务

腾讯云的分布式数据库服务适配使用 TDSQL（TencentDB for TDSQL）。

TDSQL 是部署在腾讯云上的一种支持自动水平拆分的 Shared Nothing 架构的分布式数据库服务。目前 TDSQL 默认部署主备架构，且提供了容灾、备份、恢复、监控、迁移等方面的全套解决方案。

7.3.5 非结构化存储

腾讯云的非结构化存储服务适配使用 TStack-Ceph。

TSTACK-CEPH 是由腾讯云推出的无目录层次结构、无数据格式限制，可容纳海量数据且支持 HTTP/HTTPS 协议访问的分布式存储服务。腾讯云 TSTACK-CEPH 的存储桶空间无容量上限，无需分区管理，适用于 CDN 数据分发、数据万象处理或大数据计算与分析等多种场景。TSTACK-CEPH 提供网页端管理界面、多种主流开发语言的 SDK、API 以及命令行和图形化工具，并且兼容 S3 的 API 接口，方便用户直接使用社区工具和插件。

腾讯云非结构化存储类设计如图 21 所示，定义了对象存储 TSTACK-CEPH 实现类 FSStoreTencentManagerImpl。

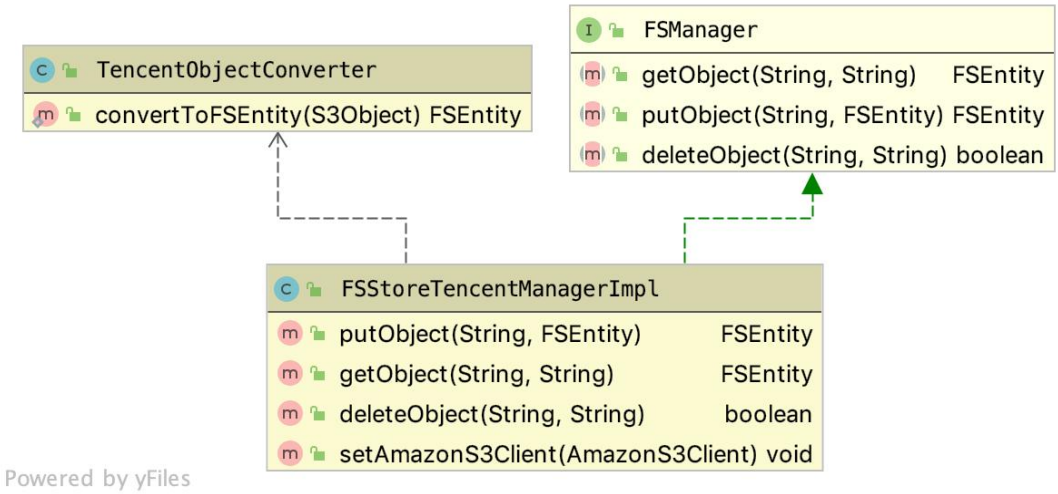

图 21 腾讯云非结构化存储类设计

7.3.6 日志服务

腾讯云的日志服务适配使用 CLS（Cloud Log Service）。

CLS 提供一站式的日志数据解决方案，提供日志采集、日志存储、日志内容搜索、统计分析等日志服务，有助于解决业务问题定位，指标监控、安全审计等日志问题。

日志服务主要提供以下功能：

a) 日志采集：通过 LogListener、API 等方式从不同日志采集端采集日志至日志服务；
b) 日志存储：使用日志服务存储日志数据；
c) 日志索引：开启日志索引对日志进行查询，可帮助用户快速定位日志问题；
d) 日志投递：用户可以将指定日志投递至其他云产品中，如 TSTACK-CEPH，满足存储或其他计算需求。

7.4 开源适配实现设计

7.4.1 分布式服务框架

开源版的分布式服务框架适配使用 Dubbo。

Dubbo 是一款高性能、轻量级的开源 Java RPC 框架，它提供了三大核心能力：面向接口的远程方法调用，智能容错和负载均衡，以及服务自动注册和发现。

Dubbo 主要核心部件包括：

a) Remoting:网络通信框架,实现了 sync-over-async 和 request-response 消息机制；
b) RPC:一个远程过程调用的抽象，支持负载均衡、容灾和集群功能；
c) Registry:服务目录框架用于服务的注册和服务事件发布和订阅。

7.4.2 分布式缓存

开源适配方案使用开源的 Redis 作为分布式缓存实现。

开源分布式缓存服务类设计如图 22 所示，类设计：

a) 定义了 Redis 缓存管理器实现类 HsafRedisCacheManager；
b) 定义了 Redis 缓存实现类 HsafRedisCache；
c) 定义了 Redis 缓存实现类的值类型适配抽象类 HsafRedisAbstractValueAdaptingCache；

d) 定义了 Redis 缓存写入器 HsafRedisCacheWriter；

e) 定义了 Redis 客户端 Lettuce 的配置类 HsafLettuceCache。

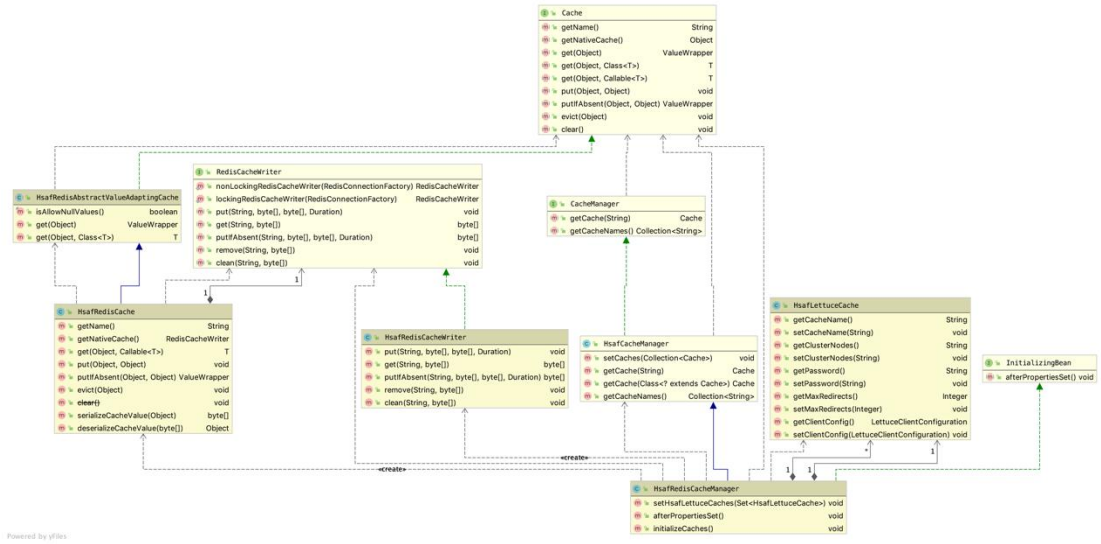

图 22 分布式缓存服务类设计

7.4.3 分布式消息队列

开源版的消息队列适配使用 Kafka。Kafka 是一款由 cala 和 Java 编写的开源流处理平台。

开源版分布式消息队列类设计如图 23 所示，类设计：

a) 定义了消息生产者实现类 MQConsumerByKafka；

b) 定义了消息消费者实现类 MQProducerByKafka。

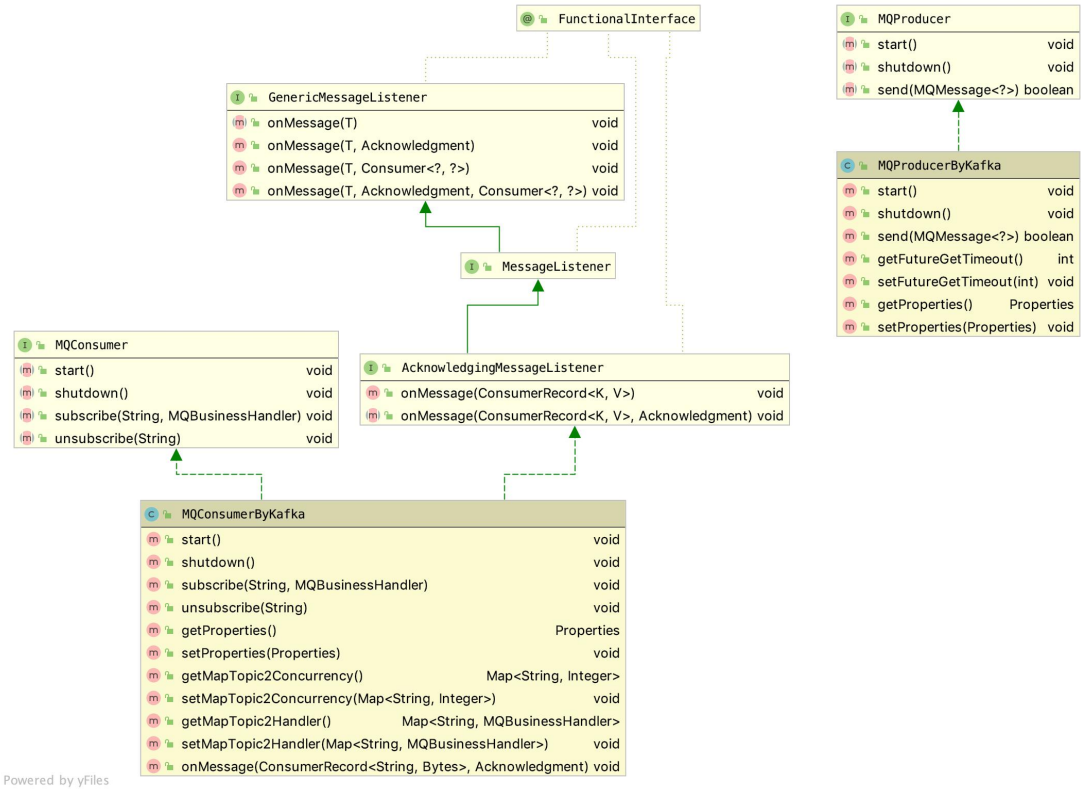

图 23 开源版分布式消息队列类设计

7.4.4 分布式数据库

开源版的分布式数据库服务适配使用 MyCat。

MyCat 是一个开源的实现了 MySQL 协议的分布式数据库服务，前端用户可以把它看作是一个数据库代理，用 MySQL 客户端工具和命令行访问，而后端可以用 JDBC 协议与大多数主流数据库服务器通信，其核心能力是支持分表分库。

7.4.5 非结构化存储

开源版的非结构化存储服务适配使用 FastDFS。FastDFS 是一个开源的轻量级分布式文件系统，它对文件进行管理功能包括文件存储、文件同步、文件访问（文件上传、文件下载）等，解决了大容量存储和负载均衡的问题。FastDFS 充分考虑了冗余备份、负载均衡、线性扩容等机制，并注重高可用、高性能等指标，适合以文件为载体的在线服务。

开源版非结构化存储类设如图 24 所示，定义了开源对象存储实现类：FSStoreGenericManagerImpl。

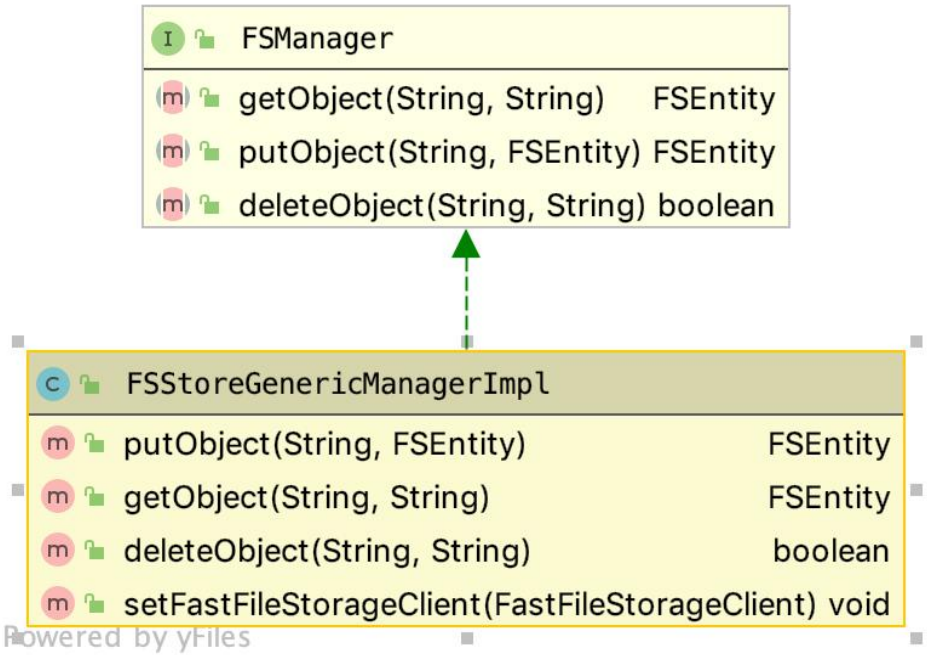

图 24 开源版非结构化存储类设计

7.4.6 日志服务

开源版的日志服务适配使用 ELK。ELK 即 Elasticsearch、Logstash 和 Kibana 简称。Elasticsearch 本质上是一个 NoSQL 数据库，以 Lucene 搜索引擎实现的。Logstash 是一个日志管道系统，可以接收数据，转换数据，并将其加载 Elasticsearch 中。Kibana 是 Elasticsearch 之上的可视化层。

8 框架技术选型

8.1 核心框架版本选型

表 2 规定了 HSAF 核心框架关键技术点的技术选型和对应版本。

表 2 HSAF 核心框架技术和版本选型

关键技术点		技术选型	版本
Java 运行环境		JDK	1.8
项目依赖管理		Maven	3.6.1
基础技术框架		SpringFrameWork	5.0.13
基础技术框架		SpringBoot	2.0.9
Web 基础框架		SpringMVC	5.0.13
安全认证框架		SpringSecurity	5.0.12
日志框架	基础	Slf4j	1.7.26
	框架	Logback	1.2.3
缓存基础框架		SpringCache	2.0.9
分布式缓存框架		SpringSession	2.0.10
持久化框架		MyBatis	3.4.6
连接池框架		Druid	1.1.19
单元测试框架		JUnit	4.12
任务调度框架		XXL-JOB	2.0.1
报表服务		JasperReport	6.9.0

注：最终版本号以国家下发的应用系统框架为准。

8.2 适配实现技术选型

表 3 规定了 HSAF 适配层实现的产品技术和版本选型。

表 3 HSAF 适配层产品技术和版本选型

产品类型	阿里云产品	腾讯云产品	开源产品
微服务架构（rpc）	HSF（ConfigServer）V3.8	TSF（基于 SpringCloud，consul）V1.12.4	Dubbo（zookeeper）V2.7
分布式缓存	ApsaraDB for Redis V3.8	Redis 集群版（自研）V4.0	Redis V5.0
分布式消息队列	RocketMQ V3.8	CMQ（自研）V1.0.0	Kafka V2.3
分布式数据库	DRDS V3.8	TDSQL（基于 MySQL 内核）V152_v1	Mycat V1.6
分布式非结构化存储	OSS V3.8	TStack 提供 V6.3（基于 ceph，支持 S3 和 swift 接口）	fastDFS V5.0
分布式定时任务调度	xxl-job V2.1		

表 3（续）HSAF 适配层产品技术和版本选型

产品类型	阿里云产品	腾讯云产品	开源产品
Web 运行容器	EDAS V3.8	TSF 提供（tomcat）V1.12.4	Tomcat V9.0
服务网关 (api gateway)	CSB V3.8	里约网关 V1.0.0	Kong V1.0.1 konga 0.14.3

注：最终版本号以国家下发的应用系统框架为准。

8.3 关系型数据库选型

关系型数据库选型应符合以下原则：

a) 优先选择符合国产化要求的数据库，优先选择分布式关系型数据库；
b) 国家局下发的数据库物理模型针对分布式关系型数据库进行了分库分表优化设计，对于集中式关系型数据库，可以结合数据库的单库单表容量经验值适当调整分库分表策略；
c) 数据库使用过程中，SQL 语句应符合 SQL2003 标准，不应使用具体数据库的专有特性。

9 架构总体应用要求

国家或地方在开展医疗保障信息平台建设时，应依托“基于统一的技术框架”的原则，遵循以下要求：

a) 使用统一的 HSAF 框架和版本进行开发；
b) 可以从 HSAF 框架提供的阿里云产品、腾讯云产品、开源产品三套技术产品方案中，选择具体的适配层实现（经过框架应用开发测试），或者自行扩展新的技术实现；
c) 所使用的开发框架组件需与 HSAF 框架中依赖的组件版本一致，不能自行引用其它版本，以免出现版本不兼容的问题。
d) 各省须逐步建立省级双数据中心，并行运行、互为容灾，进行生产维护、日常操作等工作。两个数据中心（数据中心 A、数据中心 B）网络系统的总体设计保持一致。

10 框架版本更新机制

HSAF 框架会不定期进行统一的版本更新，并且每次版本更新都会进行严格的测试以保证版本向前兼容，版本更新不会对已有的代码造成影响。