【海优禾•e农项目V1.0】

系统设计说明书模板

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **版本号** | **修订内容** | **描述** | **修订人** | **修订时间** |
| v1.0 | 初稿 | 完成文件初稿拟定 | 李海峰 | 2020.7.1 |
|  |  |  |  |  |

目录

[1. 引言 4](#_Toc46321858)

[1.1 目标 4](#_Toc46321859)

[1.2 范围 4](#_Toc46321860)

[1.3 术语和缩略语 4](#_Toc46321861)

[1.4 参考资料 4](#_Toc46321862)

[2. 整体说明 5](#_Toc46321863)

[2.1 简介 5](#_Toc46321864)

[2.2 系统目标和约束 5](#_Toc46321865)

[3. 整体设计 7](#_Toc46321866)

[3.1 系统架构 7](#_Toc46321867)

[3.1.1 架构概述 7](#_Toc46321868)

[3.1.2 架构图 8](#_Toc46321869)

[3.1.3 客户端架构 9](#_Toc46321870)

[3.1.4 服务端架构 9](#_Toc46321871)

[3.1.5 业务服务拆分 12](#_Toc46321872)

[3.2 用例视图 13](#_Toc46321873)

[3.2.1用户登录、注册视图 13](#_Toc46321874)

[3.2.2 用户企业视图 14](#_Toc46321875)

[3.2.3 用户与农场视图 14](#_Toc46321876)

[3.2.4 用户与地块视图 15](#_Toc46321877)

[3.2.5 用户与作物视图 16](#_Toc46321878)

[3.2.6 用户与农事视图 16](#_Toc46321879)

[3.3 逻辑视图 17](#_Toc46321880)

[3.3.1 逻辑视图 17](#_Toc46321881)

[3.3.2 分层 18](#_Toc46321882)

[3.3.3 设计机制 20](#_Toc46321883)

[3.4 部署视图 22](#_Toc46321884)

[3.5 数据视图 23](#_Toc46321885)

[3.6 公共平台使用 23](#_Toc46321886)

[4. 接口设计 23](#_Toc46321887)

[5. 数据库设计 23](#_Toc46321888)

[5.1 数据库物理结构设计 24](#_Toc46321889)

[5.2 数据转换 24](#_Toc46321890)

[6. 与其他系统之间的依赖关系 24](#_Toc46321891)

[7. 性能及其它方面 25](#_Toc46321892)

[7.1 可维护性 25](#_Toc46321893)

[7.2 安全性 25](#_Toc46321894)

[7.3 健壮性 25](#_Toc46321895)

[7.4 稳定性/可靠性 26](#_Toc46321896)

[7.5 性能优化 26](#_Toc46321897)

[8. 附件 27](#_Toc46321898)

# 1. 引言

## 1.1 目标

本文档主要用来指导海优禾•e农平台的详细设计工作，为详细设计提供统一的参照标准，其中包括系统的总体设计、内外部接口、系统架构、编程模型以及其他各种主要问题的解决方案。

## 1.2 范围

本文档主要介绍“海优禾•e农”项目的软件系统详细设计。以技术的角度诠释“海优禾•e农”系统的实现，技术人员或者想要了解技术相关的客户或者工作人员可以通过文档清晰看到系统实现的逻辑，便于开发人员进行交流。

本说明书的预期读者包括：

* 项目开发人员，特别是编码人员
* 软件维护人员
* 技术管理人员
* 执行软件质量保证计划的专门人员

## 1.3 术语和缩略语

列出文中全部术语并给出定义，列出文中全部缩略语，并给出全程和定义。

|  |  |  |
| --- | --- | --- |
| **序号** | **术语/缩略语** | **全称和说明** |
| 1 | http | 超文本传输协议 |
| 2 | AOP | 面向切面变成 |
| 3 | 微服务 | 一种新型软件架构，是SOA架构下的最终产物 |
| 4 | ER模型 | 实体关系模型 |

## 1.4 参考资料

无

# 2. 整体说明

## 2.1 简介

海优禾·e农平台是面向农户和农场集成物联设备、管理系统、数据处理的综合农场管理平台。平台通过互联农场、物联监测、智能管控等功能化手段实现农场远程化、自动化管理，让农场管理更高效、准确、便捷。海优禾·e农不仅是管理工具，更可以通过溯源数据追踪提升消费者认可度、信任度提升品牌核心竞争力、卖的多、卖的赚的营销利器。

## 2.2 系统目标和约束

**1.系统目标**

系统的建设围绕系统性能开展，系统的目标可从以下几点进行考虑：

1）标准化和开放性

系统依照国家的有关标准和规范，采用开放构件技术进行建设，系统能够提供开放的客户接口和完全开放式的网络计算环境，可方便地进行自身拓展和实现与其他相关系统的无缝连接。系统的网络体系结构和通信协议遵循统一的国际工业标准。实现业务流程统一、应用系统接口统一、软硬件平台统一，具有良好的开放性和可扩展性，以适应系统规模和应用功能的不断完善和扩展。

2）先进性和成熟性

软件开发保证技术的先进性，以避免在软件交付使用后过早面临被淘汰的危险，同时软件开发所采用的技术必须是成熟的技术，以免技术不成熟给系统运行造成困难。

3）参数化设计和灵活性

系统将系统属性、软硬件配置等设计为参数配置，这样可以按不同的情况灵活设置，以保证系统应用程序的统一性。

4）可扩展性

包括用户数量上的可扩展性（性能可扩展性）及业务上的可扩展性（功能可扩展性）。同时系统平台应具有良好的可伸缩性，良好的可管理性，便于对系统的统一管理，统一监控，降低管理成本。

5）可管理性

便于系统管理员的管理，并在系统发生任何问题的时候都能够很容易地进行诊断，并立即采取有效的措施，使得系统时刻处在良好运行的状态。

**2.设计约束**

1）服务器软硬环境

* 服务器CPU：4核+
* 服务器内:16GB+
* 服务器硬盘：500GB+
* 操作系统：CentOS 7.5+
* Java运行环境：jdk1.8
* Mysql数据库：mysql 5.6+

2）客户端设备环境

* PC操作系统：window7以上
* PC浏览器：Chrome/Firefox等现代主流浏览器
* 移动设备：Android 8+，IOS10+

3）开发环境配置

* 操作系统：window7,window10
* 开发工具: intellij idea/webstorm或Visual Studio Code
* 浏览器：谷歌浏览器（Chrome）

4）接口约束

* 所有接口统一接入模式。
* 和个人信息相关页面的接口必须用户登录鉴权后才能调用，未登录调用，需要给出对应的错误提示。

5）用户界面约束

系统采用B/S模式，用户在PC端使用浏览器访问web网站，在移动设备端使用app访问系统。系统界面设计应遵循：

* 界面交互尽量保持一致性，界面设计简洁明了
* 设计完整的交互过程：系统的每一次对话操作，都应该有明确的次数：开始、中检处理过程、结束提示等
* 提供简单明了的错误处理机制
* 提供信息反馈：用多种信息提示用户当前软件的运行状态，软件界面元件的功能。

# 3. 整体设计

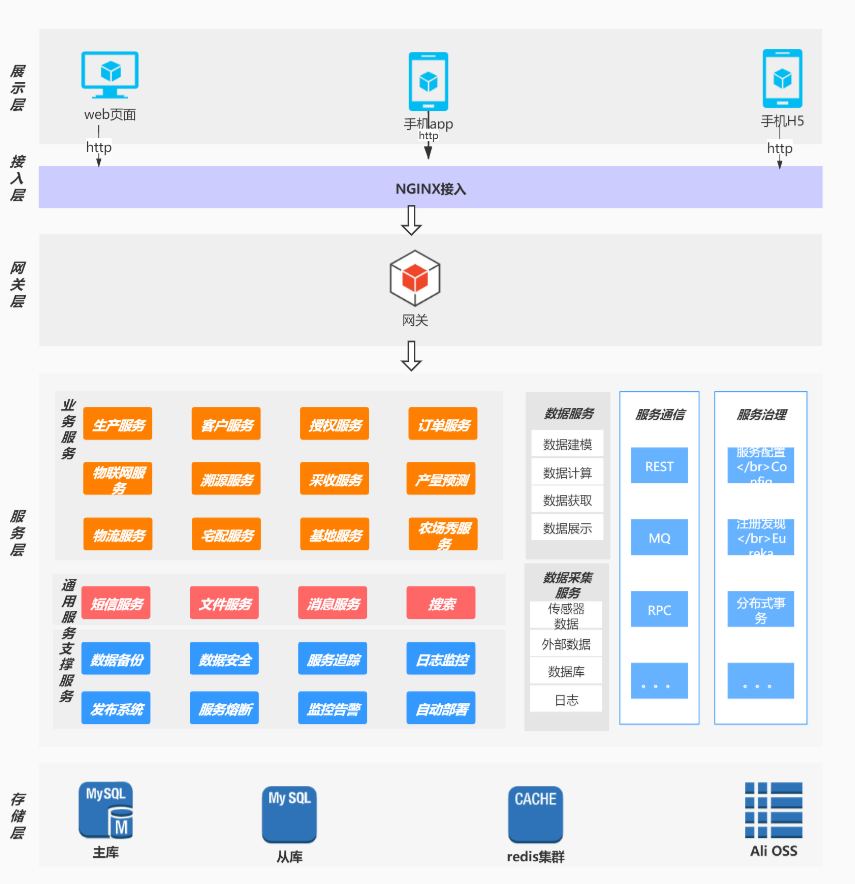
## 3.1 系统架构

### 3.1.1 架构概述

系统的架构归结起来有如下几点：

1. 系统采用B/S模式，并采用前后端分离架构。
2. 前端独立开发，独立部署，可以根据需要平台不同选择合适的展现方式，比如app，web页面，小程序等等。
3. 服务端只提供接口，不涉及页面。服务端接口进行加密，token校验，保证接口安全。
4. 无特殊说明，前后端统一采用http接口进行交互。
5. 后文中的架构设计未特殊注明的情况多都指服务端系统架构设计。
6. 服务端采用微服务结构进行业务拆分，微服务之间采用REST和MQ进行数据通信和交互。

### 3.1.2 架构图



1. **展示层**

前端展现层，根据不同账号体系（即端）独立拆开。

系统采用前后端分离后，该层由前端人员根据各端特点选择合适的展现技术（WEB，移动H5，混合app，微信小程序等）。

在前后端分离模式下，该层等价于客户端。客户端可游离于服务端单独进行设计。摆脱了传统jsp页面前后端依赖的弊端，加快了客户端开发速度，也更加有利于客户端人员根据自己的技术特点选择合适的展示技术。

1. **接入层**

页面通过http接口向服务端拉取数据，接入层提供接入的入口，实现域名管理和负载均衡**。**

可以使用nginx作为接入层组件，如果公司有自己的负载均衡方案，也可以使用，

1. **网关层**

对客户端提供的接口，使用网关层统一封装，网关层在系统具有重要作用：鉴权，限流，防刷，黑白名单，频率控制都在该层实现。

1. **服务层**

服务层为系统的重要业务逻辑层。该层含有内容较多。该层主要功能会在后文细说。可分为几部分：业务服务、基础服务、支撑服务、服务间通讯、服务治理。

该层是系统业务开发的核心，需要根据业务特点进行横向和纵向的拆分。

1. **存储层**

该层是整个系统的数据存储重要一层。当前系统中，根据数据特点选择了如下的存储系统

* 通用的业务数据，选择mysql数据（数据库需要进行主从备份和读写分离）
* 热点数据选择用redis作为缓存
* 文件采用阿里云oss服务

### 3.1.3 客户端架构

**1.神农口袋Web端**

1）主体构架：采用前后端分离开发模式，主架构采用jquery及渐进式框架vue,路由方面使用jquery.hash监听路由变化，通过$.load切换渲染页面实现spa效果，监听路由变化配合ajax监听拦截，可拦截白名单外页面，无token情况，无权限等情况

2）打包构建方面：项目发布使用打包构建工具gulp对最终的产品进行了less解析,es6编译，和图片压缩校验合并等操作。

3）缓存方面：通过动态修改引用文件的后缀添加hash防止缓存，每个hash值都是根据文件内容进行MD5加密计算得到，如此每个修改后的文件发布都不会被缓存，而未修改的文件可不影响缓存

4）UI界面方面：主体参考layui-ui设计风格，采用自身模块规范编写的前端 UI 框架，遵循原生 HTML/CSS/JS 的书写与组织形式，门槛低，拿来即用。其外在极简，却又不失饱满的内在，体积轻盈，组件丰盈，从核心代码到 API 的每一处细节都经过精心雕琢，非常适合界面的快速开发。她区别于那些基于 MVVM 底层的 UI 框架，却并非逆道而行，而是信奉返璞归真之道。并layui 兼容主流全部浏览器（IE6/7除外）。

**2.神农口袋app端**

主体构架：采用混合开发的框架APICloud为主，平台提供了线上打包,发布,版本迭代,版本更新等功能，页面使用渐进式框架vue构建用户界面，apicloud界面由Window+Frame/FrameGroup组成，打开每个页面都会被推入页面栈，可缓存页面，通过自定义事件广播，调用跨窗口调用函数等实现页面间通信，

**3.溯源H5**

采用vue配合vue-router,vuex,axios等实现的单页面应用，根据溯源id向服务端拉取数据，首先判断溯源有效性，拦截无效溯源，再根据皮肤数据动态加载对应皮肤，可见模块组件，采用懒加载组件，配合动态组件实现页面加载内容最小化，通过vuex完成状态管理，存储组件间通信数据。

**4.代码规范**

1）与海尔开发代码规范大致相同

2）方法名采用驼峰命名，变量采用小驼峰命名，常量，localstorage以大写为主,类的首字母大写

3）函数方法标注函数作用，关键字说明，及返回值

4）html按场景使用语义化标签

5）css尽量避免id选择器用于样式

6）Class要符合场景能反映元素目的和用途的名称

### 3.1.4 服务端架构

服务端采用Java作为业务代码的开发语言，采用的jdk版本1.8。主要的技术选择如下：

* Springcloud作为系统微服务主体架构（包含eureka,Robbin,Zuul,Hytrix等等组件）
* Mybatis作为系统的ORM框架
* 系统间的同步数据交互采用Feign
* 系统将的异步数据交互采用RabbitMQ
* 热点缓存和用户token信息存储采用redis
* 系统分布式锁采用基于redis的reddision锁

1. **微服务架构**

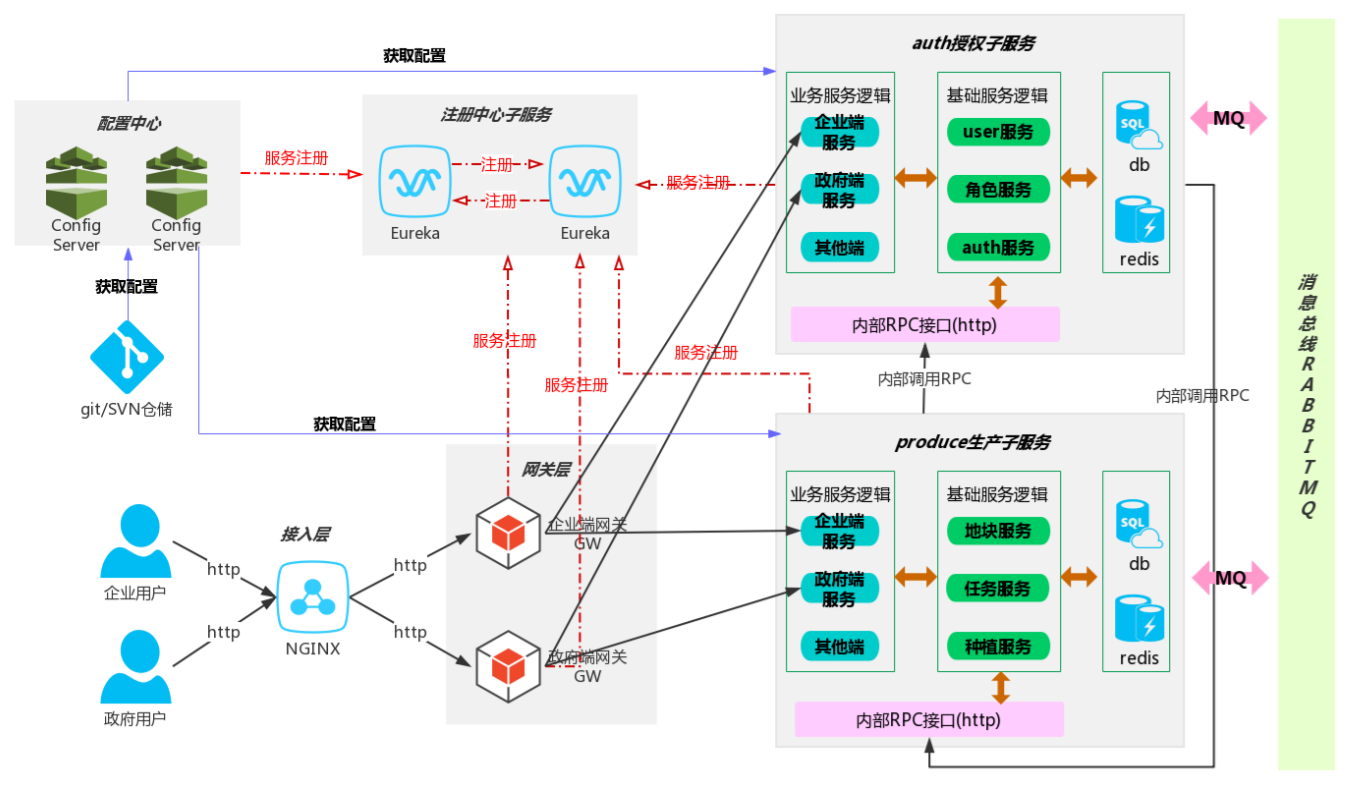
系统采用微服务架构，避免以前单一系统架构的水平垂直拓展限制。在技术选型上，我们选择使用SpringCloud全套体系作为微服务架构。

微服务化以后，重点是对业务逻辑的服务拆分，拆分后，技术上要解决：服务注册与发现，服务治理，服务间通信，分布式事务方面问题。

下图为一个微服务的详细架构，其中以“企业端”和“政府端”两端为例，详细描述了一个各个服务之间的关系以及请求的流向。

图中以Springcloud Eureka组件作为服务治理核心组件，构建注册中心，维护服务的注册于发现。通过SpringCloud Config组件构建配置中心，让所有服务的配置文件统一管理。无论是网关还是子服务都会向注册中心注册自身服务，并从配置中心获取自身服务的配置信息。服务之间的通信基本采用两种模式：异步（MQ）和同步（RPC：HTTP）。

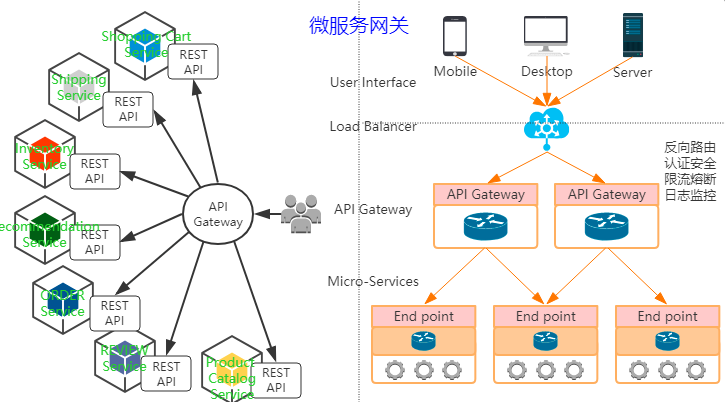
而图中更是对一个子服务的内部结构做了一个简略说明，从图中可以看出，一个子服务对应一个数据库，实现数据库的业务拆分。



1. **网关**

系统的API网关是一个服务器，是系统的唯一入口。API网关封装了系统内部架构，为每个客户端提供一个定制的API。它还具有其它职责，如身份验证、监控、负载均衡、缓存、流控。API网关方式的核心要点是，所有的客户端和消费端都通过统一的网关接入微服务，在网关层处理所有的非业务功能。

下图为网关组件的说明示意图。



1. **服务层**

该层是系统最为庞大，也是最复杂的一层。系统的所有业务逻辑都在此层实现。从之前的架构图已经能看出，该层按照功能可以分成以下几部分：业务服务、通用服务、支撑服务、服务间通信、服务治理。

我们对服务按照一定的抽象维度进行的划分，分成三种：业务服务、通用服务、支撑服务。

无论是那种服务，都作为一个子服务，独立部署。

• 业务服务

和产品业务逻辑相关的操作具有一定的相似性组成一个集体，合成服务。例如架构图中的“生产服务”，“客户服务”。

• 通用服务

该服务是对业务服务的封装或沉底。其实他本是业务服务，我们对一些比较通用的业务服务进行统一封装，作为一个共有组件。例如“短信服务”，“文件服务”，“推送服务”。

• 支撑服务

业务逻辑无关，但是对于整个系统的运行提供支撑保障的服务。例如“日志监控”，“性能报警”，“自动部署”等，该部分从逻辑来说，没有该部分，系统也能运行，但是安全性大大折扣。

### 3.1.5 业务服务拆分

从当前系统的业务特性，按照服务内聚原则，将当前的系统拆分成如下的子系统。

service-foundation：基础模块子服务，通用服务，包括文件上传、短信通道、图片验证码。该服务器为基础服务模块，及一些基础功能的沉底服务。也可以再后续根据实际需要将该服务按照单一功能进行拆分成多个子服务。该服务属于通用服务。

service-idgen：唯一id生成子服务，通用服务，统一生成唯一id，避免使用数据库id自增在分布式系统的不适用。当前该服务器的id生成策略使用自定义的逻辑进行生成，如果后续想替换成其他生成算法，只需要在该服务中修改核心的生成算法逻辑即可。所欲id的适用方没有任何的影响。

service-user：企业端用户子服务，业务服务。该服务与企业端用户的相关操作有关，提供企业端用户的注册、登录、token生成、登录验证等等用户相关服务，后续可以作为系统的统一用户中心使用。

service-produce：生产子服务，业务服务。该服务器是海优禾•e农系统的重要业务子系统，该服务器提供农场、地块、种养计划、采收记录、溯源、农事、一物一码等等相关的业务服务。该服务是重要的系统ERP管理子服务。

service-customer ：客户子服务，业务服务，该服务器提供客户、宅配、订单等想干的业务服务。

service-operate：运营数据子服务，业务服务。该服务计划作为系统的后续统计子服务。比如该子服务中统计系统的活跃企业、活跃农场、活跃用户等相关数据。该服务会从其他子服务读取元数据，夜晚进行定时器统计，并将统计数据归档，对外提供这些统计数据的展示等服务。

service-appconfig：配置子服务，业务服务。系统的一些基础参数，都可以在该服务进行配置。

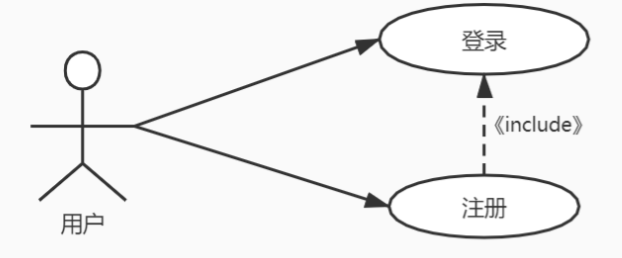
service-admiuser：管理后台用户账号子服务，业务服务。管理后台的用户中心，提供管理后台的用户、角色、资源权限等相关服务。

service-operuser：运营商端用户账号服务，业务服务。系统支持多运营商结构，每个运营商可以使用自己的域名访问我们系统。该服务就是提供运营商相关的配置、用户、登录等服务。

## 3.2 用例视图

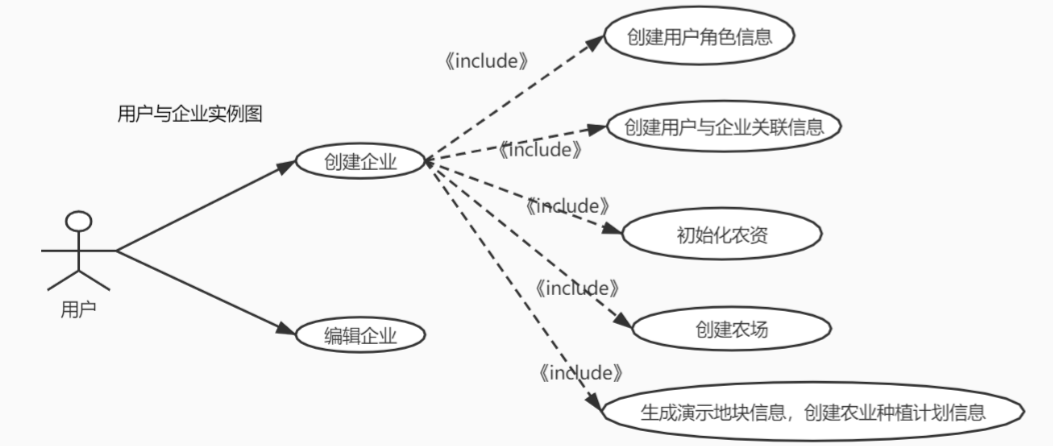
### 3.2.1用户登录、注册视图

用户输入手机号、验证码系统会判断这个手机号是否存在，如果存在，就直接登录；反之进行注册。



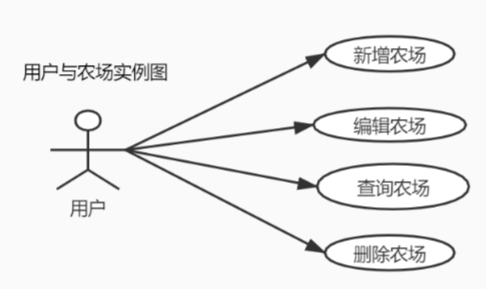
### 3.2.2 用户企业视图

用户登录进去后，可以填写企业、农场信息。



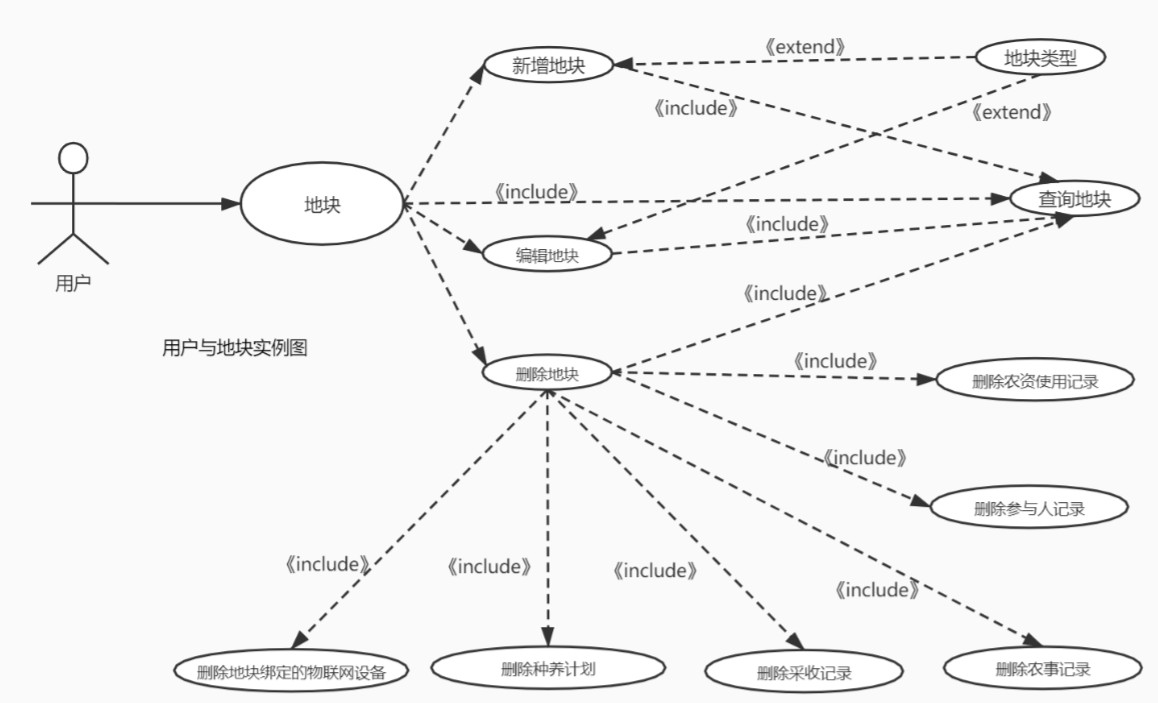
### 3.2.3 用户与农场视图

系统支持一个企业可以拥有多个农场，用户可以针对某一个具体的农场进行新增、编辑、查询、删除的操作。



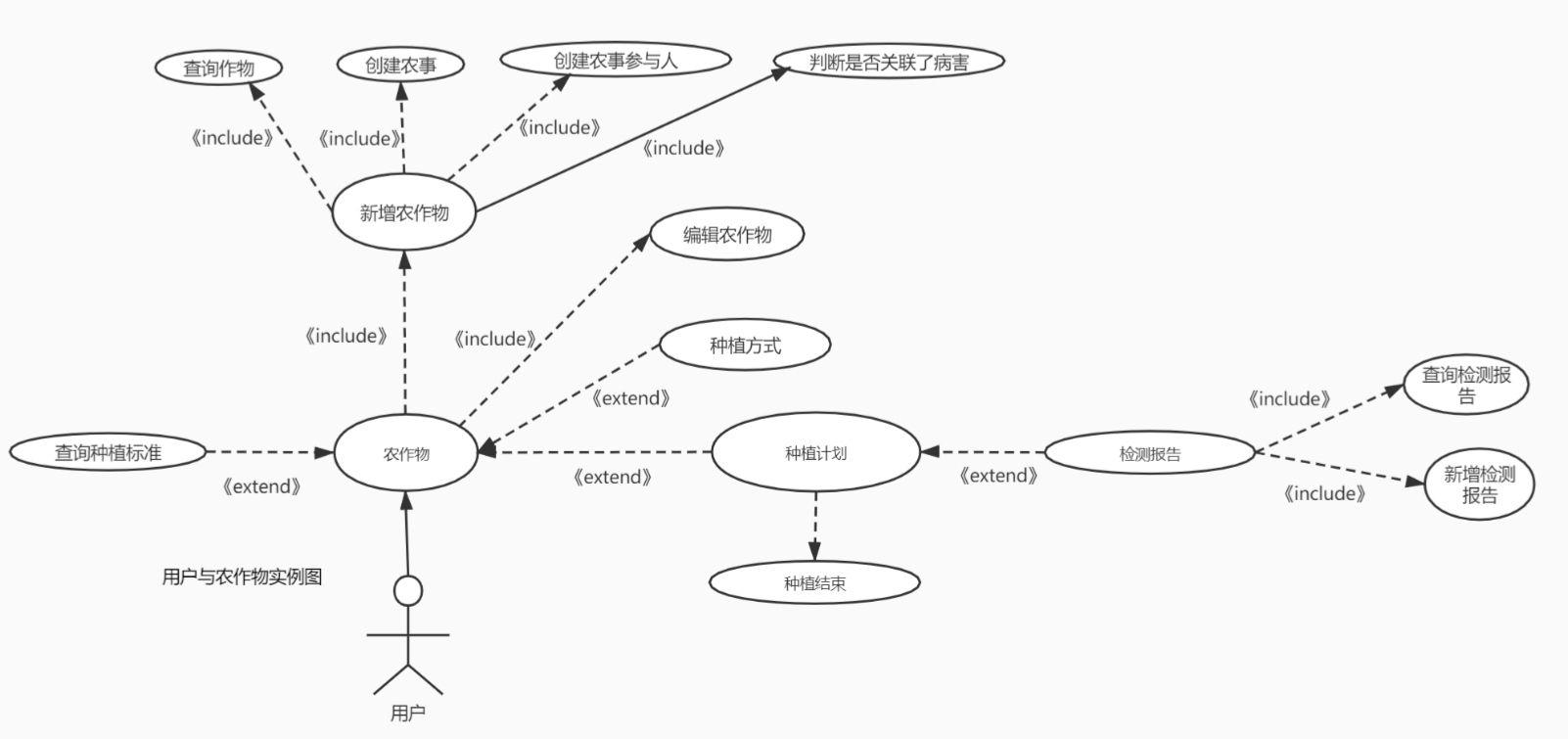
### 3.2.4 用户与地块视图

系统支持一个农场下面可以有多个地块，用户可以针对某个农场下面的某个具体的地块进行新增、编辑、查询、删除操作。



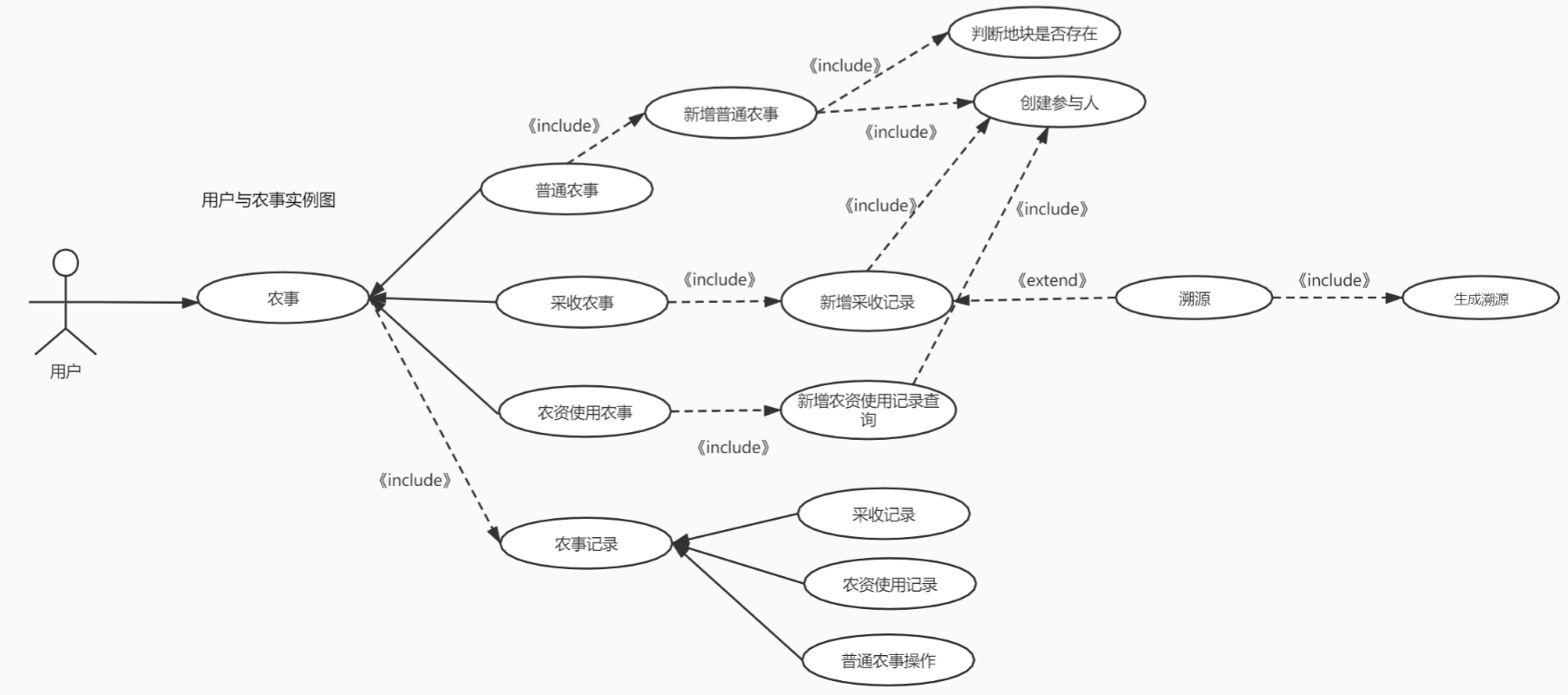
### 3.2.5 用户与作物视图

当用户创建完地块后，可以进行添加种养品种(农作物)操作、编辑种养品种(农作物)操作，种养农作物需要选择种植标准、种植方式，其他比如移栽时间、预计采收时间、预计产量、农作物株数、间距等可以选择性填写。种植完成后就会生成相应的种植计划，后续针对此次种植计划可以进行一系列的检测，然后生成相应的检测报告，生成的检测报告可以提供给客户查看。



### 3.2.6 用户与农事视图

用户种植完农作物后，接下来可以进行农事操作，农事主要包含普通农事、农资农事、采收农事三类，比如移栽某个农作物，这就是一件普通农事，需要记录下来；后期针对农作物进行的施肥、除虫、除草操作，由于涉及到了农资的使用，所以这种操作称之为农资农事；经过一定周期，农作物就可以进行采收，每一次的采收也是一条农事，我们称之为采收农事。操作者可以查询到整个农事记录、农资农事记录以及采收农事记录。同时我们可以把这个作物的整个生命周期形成一条记录链(系统中称之为溯源)，这样顾客可以通过溯源码查看到这个农作物的整个生命周期。



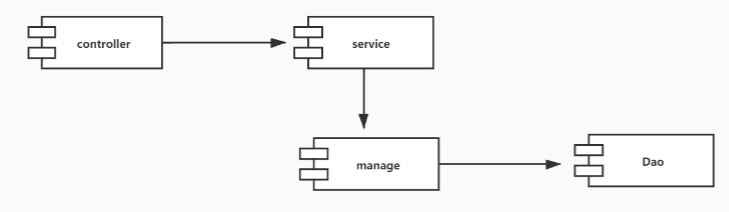
## 3.3 逻辑视图

### 3.3.1 逻辑视图

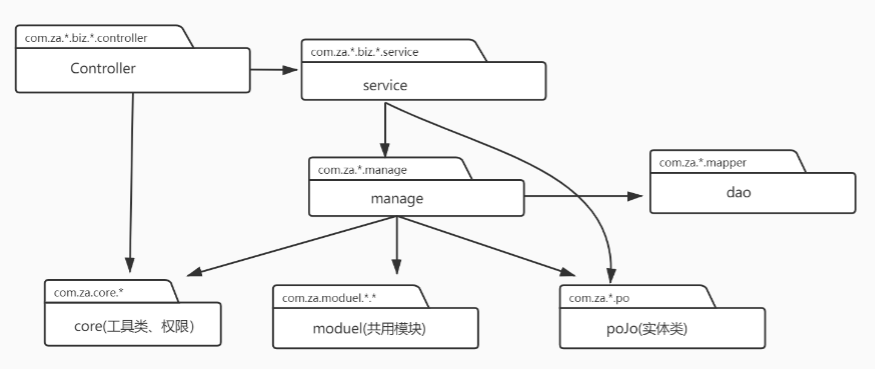
系统引入了微服务后，需要对业务进行横向拆分，按照业务逻辑内聚特性拆分成不同的业务子系统。

当前的包含有的子服务有：用户子服务（user-server）、生产子服务（produce-server）、管理后台用户子服务（adminuser-server）、基础子服务（foundation-server）等，详细参见前文的业务服务拆分。每一个业务子系统都独立开发和部署。

每一个业务子系统都遵循MVC的设计模式，可以分为四个层级：控制层controller、逻辑聚合层service、核心业务层manage、以及持久层Dao。下图为架构分层示意图。

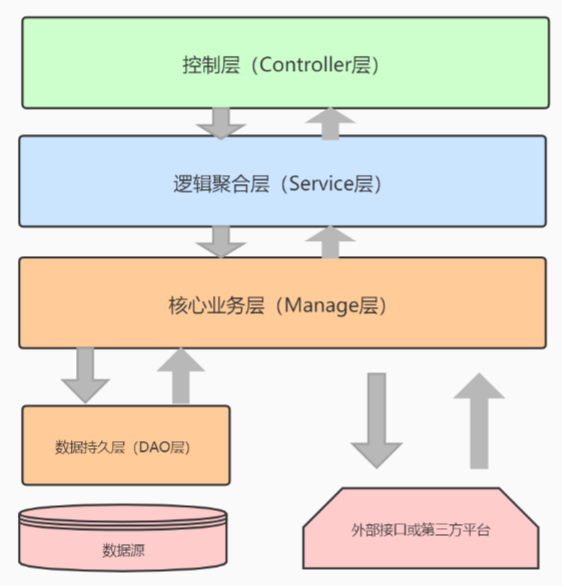


基于上述的MVC分层结构，子服务的代码中包的设计就已经比较清晰了，除了核心的四个分层外，代码中还涉及到一些其他的辅助功能，比如core中的工具类，module中的一些共用功能的封装等，具体如下图所示。



### 3.3.2 分层

四层：控制层（Controller）、逻辑聚合层（Service）、核心业务层（Manage）、以及数据持久层（DAO）。下面分别介绍每层功能和使用。



1. **控制层（Controller）**

在前后端分离模式下，控制层的主要作用就是数据接口的封装层，在该层指定和接口相关的属性：如接口地址、请求参数、返回参数、接口说明、http方法等等。在我们的代码规范中，控制层只能作为薄薄的一层出现，即该层只做接口的定义功能，而没有任何的聚合逻辑代码。

由于在我们的系统中，包含企业端和管理后台两个不同登录用户使用的网站系统，为了区别这个端的接口，我们对控制层Controller进行如下的分包约定：com.za.\*.biz.\*.controller。

第一个\*代表的是不同的子服务，比如用户子服务（user-server）的包路径为：com.za.user.biz.\*.controller。生产子服务（produce-server）的包路径为：com.za.produce.biz.\*.controller。

第二个\*代表的是不同的用户端。比如管理后台端（admin）的包路径为：com.za.\*.biz.admin.controller。企业端（co）的包路径为：com.za.\*.biz.co.controller。

举例：综合上面的定义。如果用户服务的管理后台端的控制器包路径为：com.za.user.biz.admin.controller。生产服务的企业端的控制器包路径为：com.za.produce.biz.co.controller。

1. **逻辑聚合层（Service）**

由于控制层（Controller）非常的薄，功能单一，接口中相关的业务聚合逻辑代码行数较多，只能放到逻辑聚合层(Service)中来实现。他的包路径规则和Controller层规定一致。

在service层中，我们可以很方便通过封装的方法拿到当前操作的用户，通过底层的查询接口得到需要组合聚合的业务数据。

1. **核心业务层（Manage）**

具体的业务逻辑层，比如对某一个实例进行增删改操作的时候，都需要在该层进行逻辑封装。

该层包命名方式为com.za.\*.manage，其中\*是用来区分不同服务的。比如用户服务的中间层完整包名为com.za.user.manage。

该层的主要作用可以归结如下：

* DAO层交互，对DAO层的业务通用能力的封装
* 复杂事务逻辑的封装
* 分布式锁的存在于约束
* 缓存方案、中间件通用处理封装
* 与第三方平台封装，预处理返回结果及转化异常信息

1. **数据持久层（DAO）**

数据访问层，与底层的mysql等进行数据交互。包的命名方式和Manage层相同。

### 设计机制

1. **分布式事务**

由于采用了微服务架构，不可避免的出现了分布式事务问题。研究本系统的数据特点会发现，系统的内聚性比较高，在进行横向子服务拆分的时候，可以让每一个系统都高度的内聚，这样减少了子服务之间的交互，也层根本上避免分布式事务的问题。

系统高度内聚是进行子服务拆分的重要维度。

进行了横向拆分后，剩余的一些需要进行分布式操作的事务，都是对数据实时一致性要求较低的操作，系统统一采用“最终一致性”的原则处理分布式事务问题。引入消息队列MQ进行数据一致性的维护。即需要进行分布式事务的多个服务操作的时候，服务之间通过MQ发送异步消息，进行数据更新，只要再最终维持住数据的一致性即可满足要求。

1. **并发与分布式锁**

微服务架构可以方便的进行扩容，从而来应对并发的压力。在进行扩容的时候，重点需要考虑的就是分布式锁的问题。

原来单体工程，最大的优势是简单快速。单体工程应对并发时，只需要使用Java自带的同步锁即可实现，但是微服务化扩容后，需要将单体的同步锁，扩展到系统的分布式锁。

我们当前选择的分布式锁时基于redis的reddision锁来实现。

Redisson是架设在Redis基础上的一个Java驻内存数据网格（In-Memory Data Grid）。充分的利用了Redis键值数据库提供的一系列优势，基于Java实用工具包中常用接口，为使用者提供了一系列具有分布式特性的常用工具类。使得原本作为协调单机多线程并发程序的工具包获得了协调分布式多机多线程并发系统的能力，大大降低了设计和研发大规模分布式系统的难度。同时结合各富特色的分布式服务，更进一步简化了分布式环境中程序相互之间的协作。

1. **缓存设计**

系统使用redis作为缓存存放系统的部分热点数据、用户token数据等。在进行缓存设计的时候遵循以下几个原则：

* 所有缓存数据都应该设置失效时间
* 缓存的过期时间应当分散以避免集中过期
* 缓存的key应该具备可读性
* 避免不同业务出现同名缓存key
* 对于key应该适当的缩写以节省空间
* 避免缓存穿透：数据库中未查询到的数据，可在Redis中设置特殊标识，以避免因缓存中无数据而导致每次请求均达到数据库
* 可以适当的缓存预热：对于上线后可能会有大量读请求的应用，在上线之前可预先将数据写入缓存中
* 读的顺序是先缓存后数据库；写的顺序是先数据库后缓存
* 数据一致性问题：数据源发生变更时可能导致缓存中数据与数据源中数据不一致，应根据实际业务需求来选择适当的缓存更新策略：
  + 主动更新：在数据源发生变更时同步更新缓存数据或将缓存数据过期。一致性高，维护成本较高。
  + 被动删除：根据缓存设置的过期时间有Redis负责数据的过期删除。一致性较低，维护成本较低

1. **唯一ID生成方案**

数据库业务表中的主键id值，不采用系统自增id，也不选择uuid。分布式系统中，自增id天然具备一些不易操作的特点，特别是考虑以后数据量大，需要进行分库分表时，限制较多。而uuid的可读性较差，值是乱序，不具备升序特性。

本系统当前采用自己设计的id生成机制。微服务中有idgen-server服务专门用于生成各表的id值。后续如果需要修改id生成的机制，只需要统一修改idgen-server服务中生成溯源即可。

1. **数据库设计**

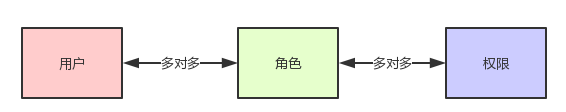
系统采用mysql数据库。详细的数据库设计参见本文档第五部分数据库设计部分。

1. **权限管理**

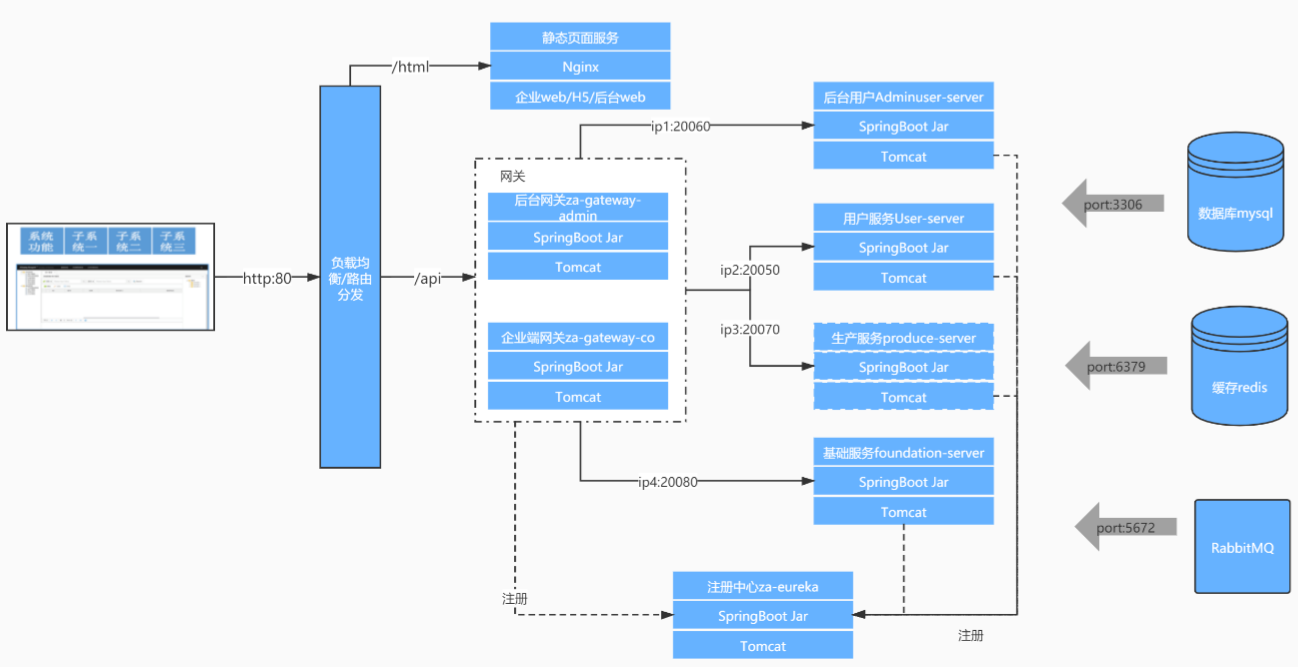
无论是企业端还是管理后台，都存在权限判断机制。系统中的权限判断，统一采用AOP思想进行实现。

自定义权限注解@HasPerm。在需要进行权限校验的接口上直接使用该注解，并指定需要的权限标识，即可在系统使用AOP自动判断权限，并没有权限的访问，统一返回错误编码和提示。

系统采用的权限设计模型是RBAC0模型，即基于角色的访问控制（Role-Based Access Control)。权限模型如下图所示。



## 3.4 部署视图



上图为按照数据流从前到后的顺序将需要部署的服务都标注在视图中。可以将图中的服务分成以下几个组成：

* 负载均衡/路由分发服务：该服务主要承担的职责，是域名解析、路由分发等功能，可以有软件nginx承担，如果公司有其他的复杂均衡方案，也可以直接使用。
* 客户端服务：客户端的静态页面资源(如web页面，溯源H5页面等)都统一在这个服务进行启动部署，该服务可以使用nginx或者node等组件快速启动web服务。
* 注册中心组：系统的服务注册中心使用eureka搭建。
* 网关组：网关是服务端接口的统一封装层，前文已经详细描述该层功能。在当前的系统部署中，需要部署两个网关：管理后台网关和企业端网关。
* 服务端子服务组：该部分是重要的业务服务组成部分，根据业务的内聚特性拆分为不同的子服务，每个子服务都需要部署成一个独立的服务运行。
* 中间件服务：主要是系统用到的一些中间件服务，如数据库mysql，缓存redis，消息队列RabbitMQ等

其中注册中心组、网关、服务端子服务组都可以根据实际的并发流量来决定部署一套还是多套。比如生产服务produce-server并发流量较大，可以单独将该服务从原来的一个升级到多个。

## 3.5 数据视图

参见数据库设计部分

## 3.6 公共平台使用

无

# 4. 接口设计

附件中上传接口设计文档：海尔数字科技业务小微软件项目接口规范及文档模板。

# 5. 数据库设计

再附件中需上传：数字科技业务小微软件项目技术对接流程——数据库设计模板

## 5.1 数据库物理结构设计

系统中业务数据属于关系型数据，采用mysql来存储业务。业务服务横向拆分成多个子服务，每个子服务的数据库也可以单独分开，比如user-server服务对应的数据库frp\_user库，produce-serve对应的数据库为frp\_produce服务。

Mysql数据库的存储引擎有多种，我们重点选择如下两种。

* MyISAM存储引擎：非事务的，锁粒度是表级的，主要应用于select，insert，不适合应用于读写操作频繁的场景，因为对于读写操作会进行锁表操作。
* Innodb存储引擎：支持事务，支持MVCC行级锁，适合任何场景

另外在进行数据库设计时遵循如下的一些原则：

* 数据库、表及字段的命名要遵循可读性原则
* 数据库、表及字段的命名要遵循表意行原则
* 当一个列可以选择多种数据类型时，优先萱蕚数值类型，其次日期类型或二进制类型，最后是字符类型。根据相同级别的数据类型，应该优先选择占用空间小的数据类型
* 表与表之间的关系不在数据库中直接建立外键，而是交给代码逻辑维护数据的一致性。

## 5.2 数据转换

无

# 6. 与其他系统之间的依赖关系

无

# 7. 性能及其它方面

## 7.1 可维护性

系统的微服务拆分符合内聚特点，减少服务之间的耦合，使得软件系统之间的架构清晰明了，因此良好的系统架构和设计是后续开发维护的先决条件，有效的降低了系统的开发维护难度、节省时间、减少系统的出错率。

本系统采用语言和语言版本都是当前使用度较广的，比如后端的Java，前端vue等，主流、先进（技术先进不等同于难度高）的技术有利于后续的开发维护。

软件开发过程进行统一的规范和过程管理，如命名、注释等各种约束都遵循规范进行，这样能有效的降低后续人员接手后对系统的上手难度。

系统基于springcloud搭建，每一个子服务都是基于springboot的jar包进行部署，操作方便。特别是当前系统运行在海尔的cosmoplat平台的paas上，已经完成dokcer镜像的打包，需要部署或扩容时，只需要找到对应的系统服务，在后台点击启动或扩容即可。

每一个子服务都会按照指定的日志格式，输出服务运行日志，在cosmoplat平台，统一输出到指定的存储位置中，方便进行日志的回溯和问题定位。

## 7.2 安全性

用户只能用帐号登陆到应用软件，通过应用软件访问数据库，而没有其它途径操作数据库。

对用户帐号的密码进行加密处理，确保在任何地方都不会出现密码的明文。

每个角色对数据库表的操作权限，如创建、检索、更新、删除等。每个角色拥有刚好能够完成任务的权限，不多也不少。在应用时再为用户分配角色，则每个用户的权限等于他所兼角色的权限之和。

## 7.3 健壮性

当前系统运行于cosmoplat平台paas上，cosmoplat平台作物平台提供者，本身就有对断电、断网等突然应急灾害的应急处理预案，从而在一定程度上保证了系统的稳定运行。

当前系统，如果真的遇到断网、断电等情况导致的服务终止，也只需要在状态恢复后，重启系统服务即可继续稳定运行。

在数据库设计时，没有使用数据库的外键进行数据一致性的约束，而是交给代码来维护，如果遇到因为断电导致服务器中断，引起的数据不一致，在重服务后，系统程序会在用户获取该数据时记性数据一致性检查，从而纠正已经发生数据错误的问题。

针对空间满的情况，当前的系统只需要在cosmoplat平台上进行新的硬盘资源的挂在，通过路径配置后，即可实现系统硬盘资源的扩容。

## 7.4 稳定性/可靠性

当前系统为微服务架构，支持动态扩容，当因为流量增加等原因导致当前服务器资源不够时，可以动态的进行容量扩容，在扩容过程不需要停止当前服务。

做好容量规划，计算出当前容量以及接下来可能的容量增长，特别是部分日期时可能出现的容量突增情况，可以根据容量规划进行合理的资源扩容。

当前的系统运行在海尔的cosmoplat平台paas层上，使用docker进行系统部署，cosmoplat本来就是提供稳定的paas服务的平台，运到因为系统升级导致服务中断的，只需要cosmoplat启动，即可在管理后台快速启动我们的系统服务。

## 7.5 性能优化

此处描述处理性能优化使用的方案和技术实现手段。

微服务架构的出现，本身就是为了解决系统的解耦和并发容量的动态扩容问题。我们系统在服务拆分上，遵循系统的内聚原则，使得服务间的耦合降低，从而能大大提高系统的性能，特别是可以针对系统当前的用户容量，快速方便的进行容量扩容。

缓存机制的引入，针对系统的热点数据，直接存放redis缓存中，增加系统的读取速度。

预留的数据库分表机制，当前的系统在数据库设计时就考虑到数据库后续的分表问题，比如唯一id的动态生成方案等都是为了数据库后续分表方便而引入，可以针对部分业务数据较大量时进行分表操作，从而提供系统数据的读取速度。另外针对部分数据可以做到数据库的读写分离，从而大大增加数据库的读写速度。

浏览器访问主要提升在用户终端浏览器打开网页的速度，通常采用的优化技术包括减少与服务端的HTTP请求、使用浏览器本地缓存（如浏览器自身缓存、AppCache、本地数据库等）、启用压缩等技术手段对浏览器访问进行优化。

使用CDN加速静态资源的访问速度。

反向代理，代理服务器来接受客户端的网络访问连接请求，然后服务器将请求有策略的转发给网络中实际工作的业务服务器，并将从业务服务器处理的结果，返回给网络上发起连接请求的客户端。

# 8. 附件

附件要求必须上传以下文档：

1.数字科技业务小微软件项目技术对接流程——数据库设计模板

2. 海尔数字科技业务小微软件项目接口规范及文档模板