東南大學

软件设计与体系结构课实验报告

题　目 Rouyi-cloud 系统分析

软 件 学 院 院（系） 软 件 工 程 专业

学　　号 71119103

学生姓名 许 润

指导教师 廖 力

起止日期 2021.10.20-2021.11.20

目录

第一章 课题概述

1.1、选题背景及意义

一直想做一款后台管理系统，看了很多优秀的开源项目但是发现没有合适的。于是利用空闲休息时间开始自己写了一套后台系统。若依是一套全部开源的快速开发平台，毫无保留给个人及企业免费使用。如此有了若依。她可以用于所有的 Web 应用程序，如网站管理后台，网站会员中心，CMS，CRM，OA。所有前端后台代码封装过后十分精简易上手，出错概率低。同时支持移动客户端访问。系统会陆续更新一些实用功能。

她是基于 SpringBoot 的权限管理系统 易读易懂、界面简洁美观。 核心技术采用 Spring、MyBatis、Shiro 没有任何其它重度依赖。直接运行即可用。

1.2、实验报告的组织结构

本文主要分为以下几个部分：

第一章是绪论，主要主要介绍 Ruoyi-cloud 后台管理系统研究意义，研究背景，主要工作，章节安排等。

第二章是 Ruoyi-cloud 系统的需求逆向分析，按照总分总结构介绍系统的需求。

第三章是 Ruoyi-cloud 系统的架构逆向恢复，从上一章的需求分析入手，结合文件结构来恢复系统的架构，并对各组件、连接件依次进行分析。

第四章是 Ruoyi-cloud 系统的架构分析评价，分析了微服务架构的优缺点，对微服务架构的一些思考。

第五章是 Ruoyi-cloud 系统的软件测试结果，主要是系统整个流程步骤的展示和执行结果。

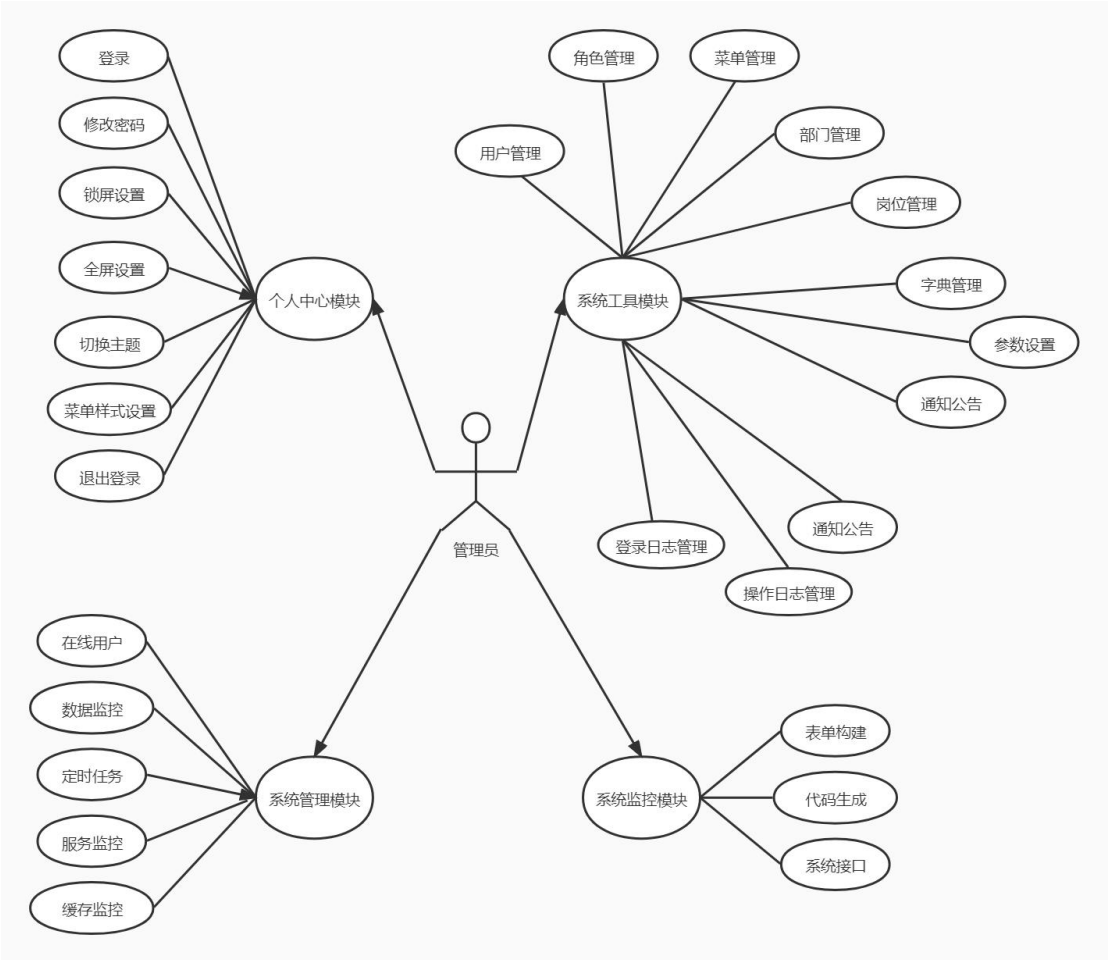
第六章是实验总结，总结反思实验中的收货与心得，以及自己对架构的一些新的认识。

第二章 软件需求逆向分析

2.1、系统总体需求分析

RuoYi-Cloud 是一款基于 Spring Boot、Spring Cloud & Alibaba、Vue、Element 的前后端分离微服务极速后台开发框架。内置模块如：部门管理、角色用户、菜单及按钮授权、数据权限、系统参数、日志管理、代码生成等。在线定时任务配置；支持集群，支持多数据源。

RuoYi-Cloud 的总体 use case 图如下：



如图所示，整个系统分为四个模块，分别是个人中心模块、系统管理模块、系统监控模块、系统工具模块，而该系统的四个模块均面向系统管理员。下一节中我将对各模块的需求进行单独分析。

2.2、各模块需求分析

2.2.1、个人中心模块需求分析

• 登录--管理员通过注册的账号密码登录系统，账号和密码验证成功后即可登录系统。

• 基本资料修改--修改头像、用户名称、手机号码、邮箱、性别等基本信息。

• 密码修改--用户输入两次新密码，确认新密码无误后对密码进行重置。

• 锁屏设置--将系统锁屏，需要重新输入密码并验证成功后才能进入系统。

• 全屏设置--设置面板状态为全屏显示。

• 切换主题--切换系统的主题背景颜色。

• 菜单样式设置--更改菜单的显示方式—左侧菜单或者横向菜单。

• 退出登录--用户退出系统。

2.2.2、系统管理模块需求分析

• 用户管理--该功能主要用于各公司各部门系统用户信息的管理与配置，包括编辑信息、删除、重置密码、分配角色等等。此外，为了方便检索特定用户，管理员还可以通过登录名称、手机号码、用户状态、创建时间等信息进行搜索。

• 角色管理--管理角色菜单权限分配、设置角色按机构进行数据范围权限划分，以及对角色信息的增删改查、搜索等。

• 菜单管理--管理系统菜单的属性值，包括是否可见，操作权限，权限标识等，并提供编辑、删除、新增等操作；对特定关键词的检索功能。

• 部门管理--管理系统部门（公司、部门），包括编辑、删除、新增、检索等，树结构展现各部门所属公司以及状态等等。

• 岗位管理--管理系统用户所属岗位，包括编辑、删除、新增、导出等功能；同时可以对特定关键词进行检索。

• 字典管理--对系统中经常使用的一些较为固定的数据进行管理，包括编辑、显示列表、删除、新增、修改、导出、搜索功能。

• 参数设置--配置系统动态常用参数，如参数名称、参数键名、参数键值、系统

内置；并可以按参数名称、参数键名等特定关键词进行检索。

• 通知公告--系统通知公告信息发布维护，包括编辑、修改、新增、编辑、删除，对特定关键词的检索功能。

• 操作日志管理--系统操作日志记录和查询；系统异常信息日志记录和查询。表单内容包括日志编号、系统模块、操作类型、请求方式、操作人员、操作地址、操作地点等信息。提供基础的删除、清空、导出以及搜索功能。

• 登录日志管理--系统登录日志记录查询包含登录异常，对登录的管理。表单内容包括访问编号、用户名称、登录地址、登录地点、浏览器、操作系统、登录状态、操作内容、登录日期等信息。并提供基础的删除、清空、导出、搜索、重置功能。

2.2.3、系统监控模块需求分析

• 在线用户管理--监控当前系统中活跃用户状态，展示会话编号、登录名称、部门名称、主机等用户信息，还可以进行强退操作以及搜索功能。

• 定时任务--在线（添加、修改、删除、是否执行)任务调度，还可以显示调度执行结果日志。

• 服务监控--监视当前系统 CPU、内存、服务器信息、Java 虚拟机信息、磁盘状态、堆栈等相关信息。

• 缓存监控--监控系统的缓存列表、键名列表、缓存内容，对系统的缓存查询，删除、清空等操作。

2.2.4、系统工具模块需求分析

• 表单构建--拖拽式构建表单示例。

• 代码生成--在线配置表信息生成对应的代码，一键生成模块，包含增删改查/排序/导出/权限控制等操作，编译即可使用。

• 系统接口--根据业务代码自动生成相关的 api 接口文档。

第三章 软件架构逆向恢复

3.1、文件结构

3.1.1、前端结构

略。这里我们主要分析后端的结构，因此不再展示前端的文件结构。

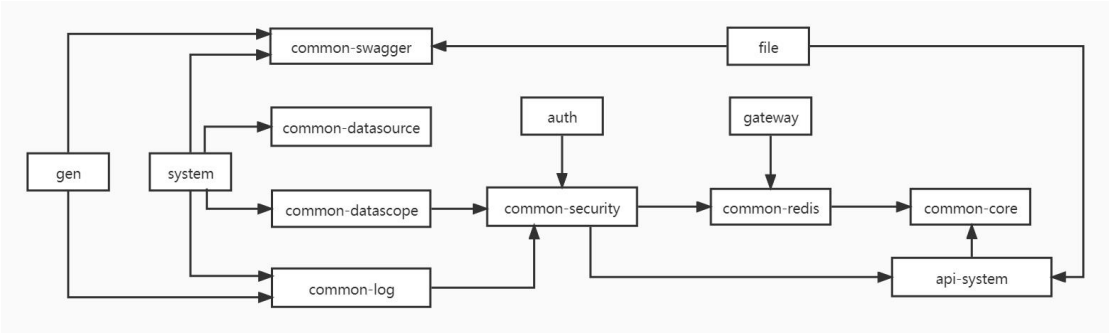
3.1.2、后端结构

```
├── bin                // bin
├── docker             // docker
├── logs               // 日志
├── ruoyi-api          // 接口模块
│       └── ruoyi-api-system                    // 系统接口
├── ruoyi-auth         // 认证中心 [9200]
├── ruoyi-common       // 通用模块
│       └── ruoyi-common-core                   // 核心模块
│       └── ruoyi-common-datascope              // 权限范围
│       └── ruoyi-common-datasource             // 多数据源
│       └── ruoyi-common-log                    // 日志记录
│       └── ruoyi-common-redis                  // 缓存服务
│       └── ruoyi-common-security               // 安全模块
│       └── ruoyi-common-swagger                // 系统接口
├── ruoyi-gateway      // 网关模块 [8080]
├── ruoyi-modules      // 业务模块
│       └── ruoyi-system                        // 系统模块 [9201]
│       └── ruoyi-gen                           // 代码生成 [9202]
│       └── ruoyi-job                           // 定时任务 [9203]
│       └── ruoyi-file                          // 文件服务 [9300]
├── ruoyi-visual       // 图形化管理模块
│       └── ruoyi-visual-monitor                // 监控中心 [9100]
├── sql                // 数据库
├──pom.xml             // 公共依赖
```

分析后端文件结构的整体框架，我们可以看出 RuoYi-Cloud 是基于 docker 的分布式微服务架构，有接口模块、认证中心、通用模块、网关模块、业务模块以及图形化管理模块。并且是前后端分离的，前端使用 Vue+Element UI；后端采用了 Spring Boot、Spring Cloud 技术。注册中心、配置中心选型 Nacos，权限认证使用 Redis。流量控制框架选型 Sentinel，分布式事务选型 Seata。

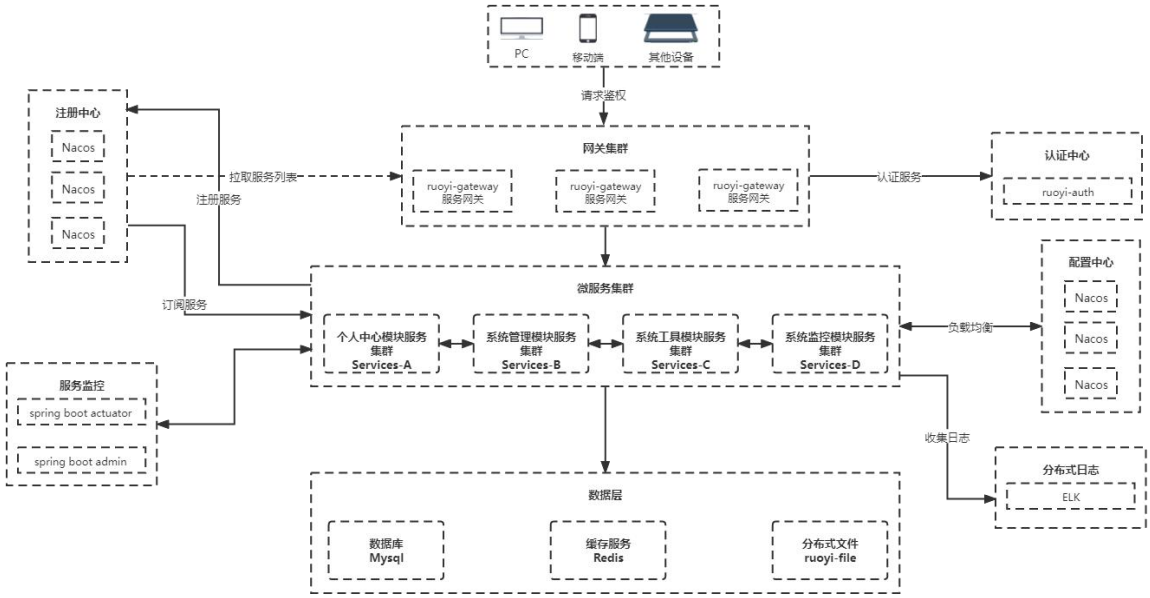
3.2、微服务架构图

3.3.1、maven 依赖图



其中 common-datasource 负责数据源相关配置以及相关依赖引入；common-datascope 负责数据权限切面以及相关注解；common-log 负责日志切面以及相关注解；common-security 为全局异常处理器；gateway 负责网关配置，路由配置，过滤器，验证码处理；common-core 内含常用工具类、常用自定义异常、通用枚举、通用实体、通用常量、通用注释等。

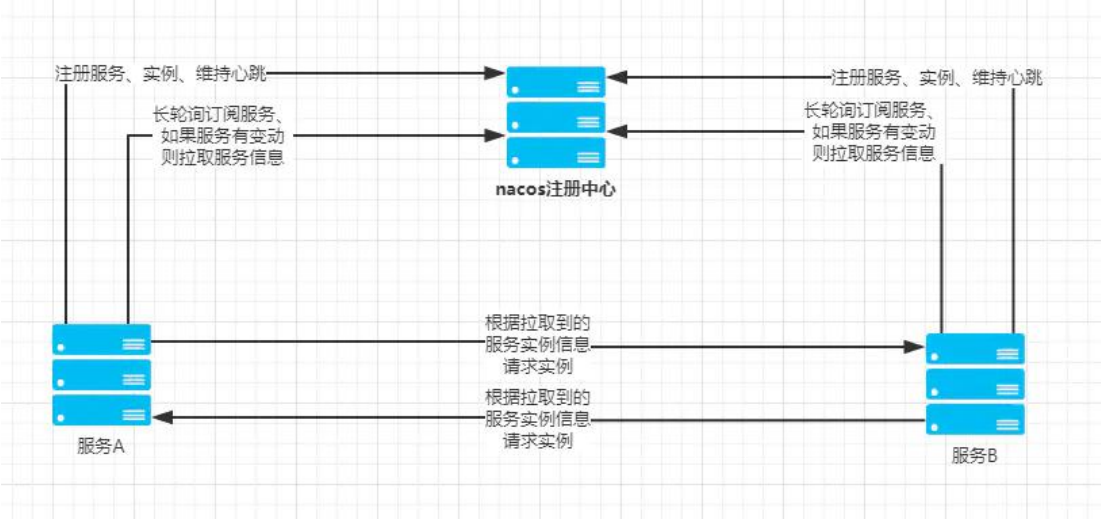
3.3.2、系统整体架构图



•**网关集群：**ruoyi-gateway 基于 Filter 链的方式提供了网关基本的功能，如安全，监控/埋点和限流等，为若依系统提供了一种简单而有效的统一的 API 路由管理方式。

• **微服务集群：**RuoYi-Cloud 系统的微服务集群可分为四个模块，分别为个人中心、系统管理、系统工具、系统监控，每个模块都包括了若干微服务并且能够独立运行，保障了系统的高可用性。在微服务架构中很多功能都需要调用多个服务才能完成某一项功能，服务之间存在 Feign 接口互相调用，

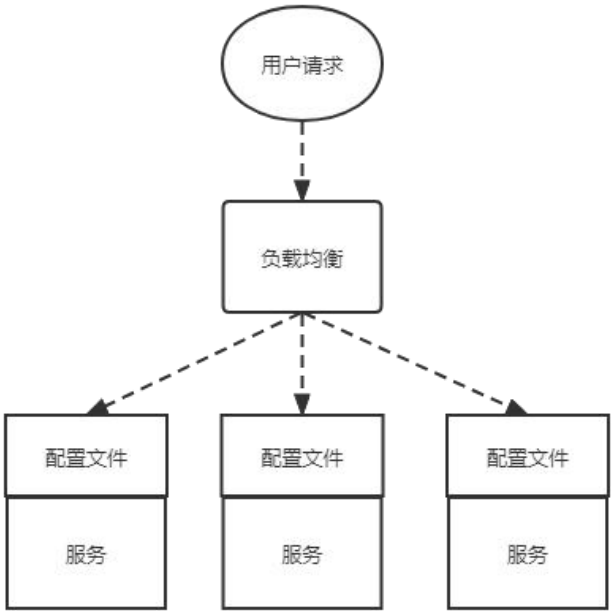
• **注册中心：**注册中心在微服务项目中扮演着非常重要的角色，是微服务架构中的纽带，类似于通讯录，它记录了服务和服务地址的映射关系。在分布式架构中，服务会注册到这里，当服务需要调用其它服务时，就到这里找到服务的地址，进行调用。RuoYi-Cloud 采用 Nacos 注册中心，下图是其工作原理：



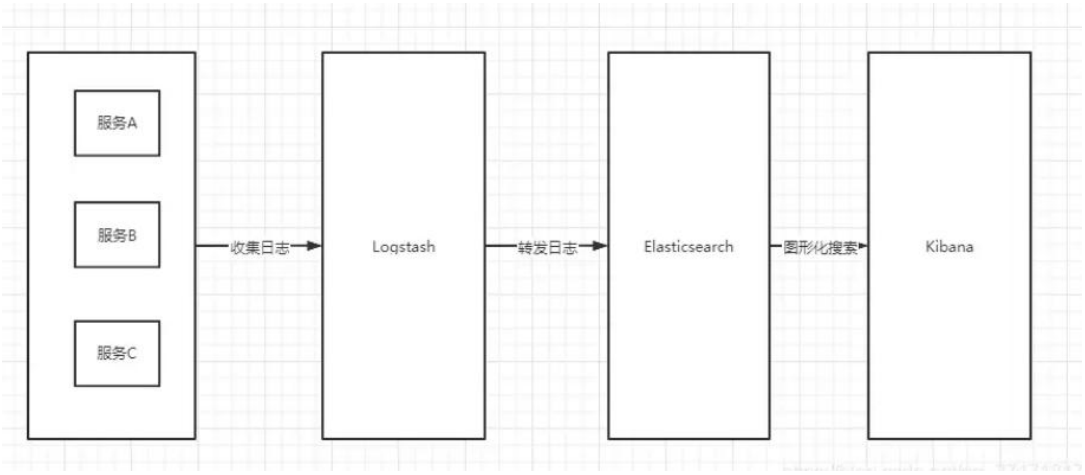
• **认证中心：**系统通过核对用户输入的用户名和口令，看其是否与系统中存储的该用户的用户名和口令一致，来判断用户身份是否正确。

• **服务监控：**监视当前系统应用状态、内存、线程、堆栈、日志等等相关信息，主要目的在服务出现问题或者快要出现问题时能够准确快速地发现以减小影响范围。

• **配置中心：**配置中心就是一种统一管理各种应用配置的基础服务组件，配置中心将配置从各应用中剥离出来，对配置进行统一管理，应用自身不需要自己去管理配置。如下图：



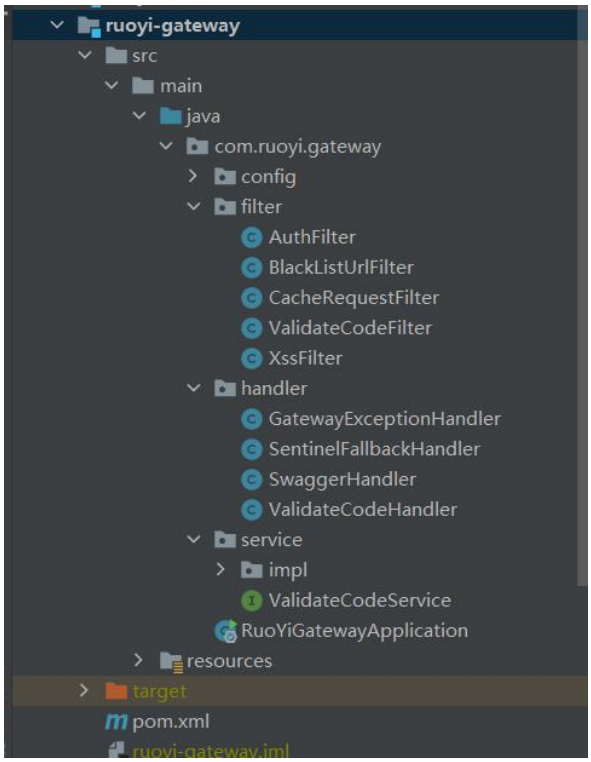
•**分布式日志：**将所有服务器上的日志收集汇总，建立集中式日志收集系统 ELK，将所有节点上的日志统一收集，管理，访问。实际上 ELK 是三款软件的简称，分别是 Elasticsearch、 Logstash、Kibana 组成。工作原理如下：



• **缓存服务 Redis：** Redis 是一个开源的使用 ANSI C 语言编写、支持网络、可基于内存亦可持久化的日志型、Key-Value 数据库，并提供多种语言的 API。

3.3、各组件分析

3.3.1、网关中心

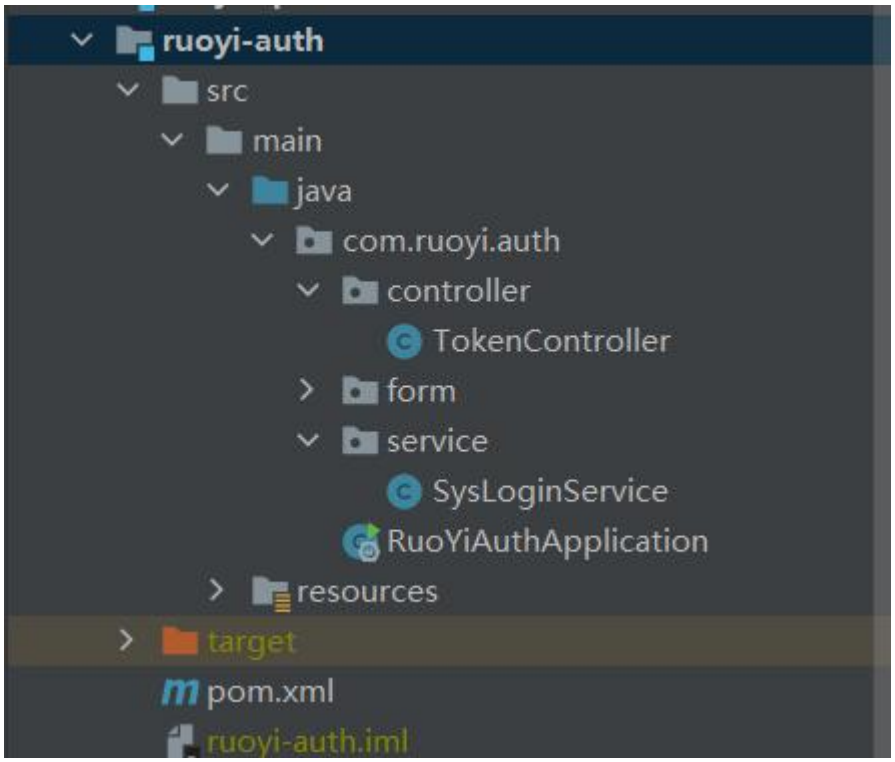


网关封装了系统内部架构，为每个客户端提供定制的 API。客户端相关的请求直接发送到网关，由网关根据请求标识解析判断出具体的微服务地址，再把请求转发到微服务实例。这其中的记忆功能就全部交由网关来操作了。

网关模块微服务由 ruoyi-gataway 实现，用于路由转发、异常处理、限流、降级、接口、鉴权等等。5 个 filter 是 AuthFilter、BlackListUrlFilter、CacheRequestFilter、ValidateCodeFilter 和 XssFilter,分别提供网关鉴权、黑名单过滤器、获取 body 请求数据、验证码过滤器、跨站脚本过滤器功能；4 个 handler 分别是 GatewayExceptionHandler—网关统一异常处理，SentinelFallbackHandler—自定义限流异常处理，SwaggerHandler，ValidateCodeHandler—验证码获取。ValidateCodeService 提供了生成验证码【createCapcha()】接口以及校验验证码【checkCapcha】接口，并在 ValidateCodeServiceImpl 中实现。

```
public interface ValidateCodeService
{
    /**
     * 生成验证码
     */
    public AjaxResult createCapcha() throws IOException, CaptchaException;

    /**
     * 校验验证码
     */
    public void checkCapcha(String key, String value) throws CaptchaException;
}
```

3.3.2、认证中心



认证中心模块由 ruoyi-auth 实现，用于登录认证，系统退出，刷新令牌。1 个 controller—TokenController 用于 token 控制，核对用户输入的用户名和口令，看其是否与系统中存储的该用户的用户名和口令一致，来判断用户身份是否正确。一个 service—SysLoginService 实现登录校验方法，包括登录、登出、注册、记住登录信息功能。

3.3.2.1、登录校验服务

SysLoginService 对登录校验服务进行了实现，TokenController 控制器 login 方法会进行用户验证，如果验证通过会保存登录日志并返回 token，同时缓存中会存入 login_tokens:xxxxxx（包含用户、权限信息）。其中【login】方法实现登录功能、【logout】方法实现登出功能、【register】方法实现注册功能、【recordLogininfor】方法实现记录登记信息的功能。

```
/**
 * 登录
 */
public LoginUser login(String username, String password)
{
```

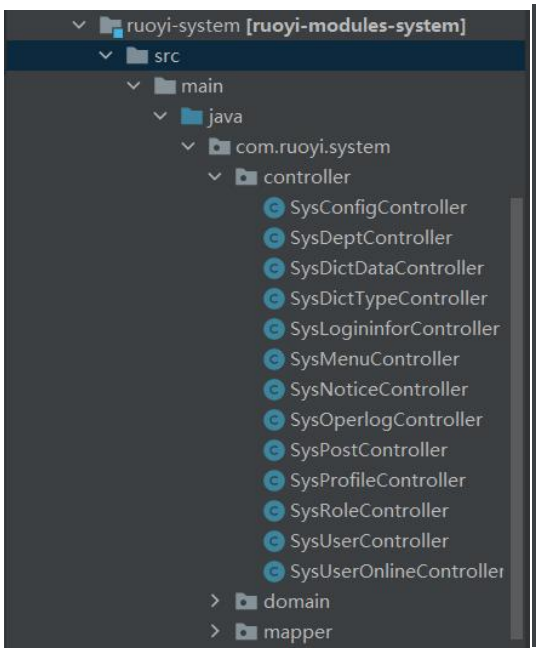
```
public void logout(String loginName) { recordLogininfor(loginName, Constants.LOGOUT, message: "退出成功"); }
```

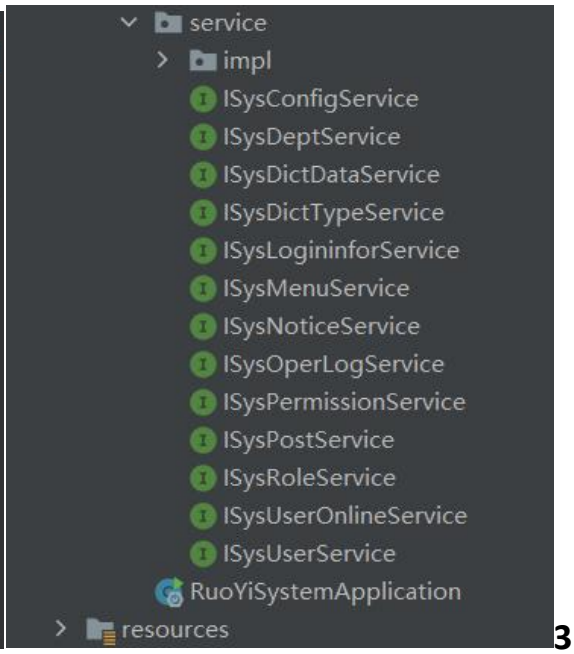
```java
/**
 * 注册
 */
public void register(String username, String password)
{
```

```java
/**
 * 记录登录信息
 *
 * @param username 用户名
 * @param status 状态
 * @param message 消息内容
 * @return
 */
public void recordLogininfor(String username, String status, String message)
{
```

3.3.3、微服务集群

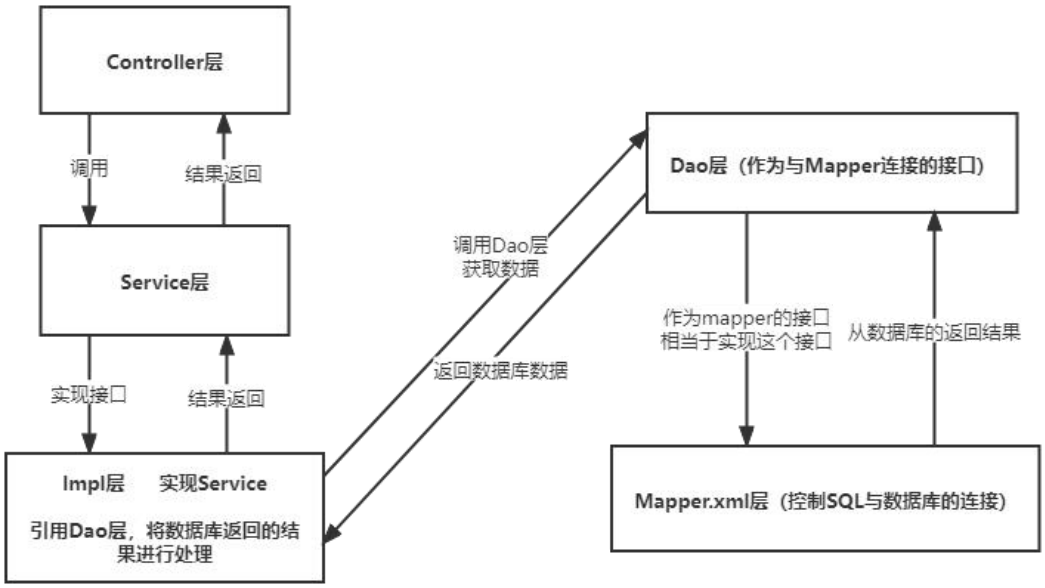
微服务集群由 ruoyi-system 实现，包含了微服务的四个模块。包中有 13 个 controller、13 个 service 接口，mapper 文件以及对应的 impl，分别用于实现参数配置操作处理、部门管理、字典业务、系统访问日志情况信息服务、菜单业务、公告服务、操作日志服务、权限管理、岗位信息服务、角色信息、在线用户信息以及用户业务





3.

微服务集群的 SSM 结构流程图如下：



由于包含的类太多，此处就不再在图中一一列出，仅展示整体结构，下文将对每个类分别分析。

3.3.1、系统管理

3.3.3.1.1、用户管理服务

用户管理功能由 SysUserController 类实现。基本实现了获取用户列表、获取当前用户信息、注册用户信息、返回用户信息、根据用户编号获取学校信息、新增用户、修改用户、删除用户、重置密码、状态修改、根据用户编号获取授权角色等功能。

部分接口如下：

```
/**
 * 获取当前用户信息
 */
@InnerAuth
@GetMapping("/info/{username}")
public R<LoginUser> info(@PathVariable("username") String username)
{...}

/**
 * 注册用户信息
 */
@InnerAuth
@PostMapping("/register")
public R<Boolean> register(@RequestBody SysUser sysUser)
{...}

/**
 * 获取用户信息
 *
 * @return 用户信息
 */
@GetMapping("getInfo")
public AjaxResult getInfo()
{...}
```

3.3.3.1.2、角色管理服务

角色管理功能由 SysRoleController 类实现。基本实现了根据角色编号获取详细信息、新增角色、修改保存角色、修改保存数据权限、状态修改、删除角色、查询已分配用户角色列表、查询未分配用户角色列表、取消授权用户等功能。

部分接口如下：

```
    * 根据角色编号获取详细信息
    */
   @RequiresPermissions("system:role:query")
   @GetMapping(value = "/{roleId}")
   public AjaxResult getInfo(@PathVariable Long roleId)
   {...}

   /**
    * 新增角色
    */
   @RequiresPermissions("system:role:add")
   @Log(title = "角色管理", businessType = BusinessType.INSERT)
   @PostMapping
   public AjaxResult add(@Validated @RequestBody SysRole role)
   {...}

   /**
    * 修改保存角色
    */
   @RequiresPermissions("system:role:edit")
   @Log(title = "角色管理", businessType = BusinessType.UPDATE)
   @PutMapping
   public AjaxResult edit(@Validated @RequestBody SysRole role)
   {...}
```

3.3.3.1.3、菜单管理服务

菜单管理功能由 SysMenuController 类实现。基本实现了获取菜单列表、根据菜单号获取详细信息、获取菜单下拉树列表、加载对应角色菜单列表树、新增菜单、修改菜单、删除菜单等功能。

部分接口如下：

```
   /**
    * 获取菜单列表
    */
   @RequiresPermissions("system:menu:list")
   @GetMapping("/list")
   public AjaxResult list(SysMenu menu)
   {...}

   /**
    * 根据菜单编号获取详细信息
    */
   @RequiresPermissions("system:menu:query")
   @GetMapping(value = "/{menuId}")
   public AjaxResult getInfo(@PathVariable Long menuId)
   {...}

   /**
    * 获取菜单下拉树列表
    */
   @GetMapping("/treeselect")
   public AjaxResult treeselect(SysMenu menu)
   {...}
```

3.3.3.1.4、部门管理服务

部门管理功能由 SysDeptController 类实现。基本实现了获取部门列表、查询部门列表、根据部门列表获取详细信息、获取部门下拉树列表、加载对应角色部门列表树、新增部门、修改部门、删除部门等功能。

部分接口如下：

```
/**
 * 获取部门列表
 */
@RequiresPermissions("system:dept:list")
@GetMapping("/list")
public AjaxResult list(SysDept dept)
{...}

/**
 * 查询部门列表（排除节点）
 */
@RequiresPermissions("system:dept:list")
@GetMapping("/list/exclude/{deptId}")
public AjaxResult excludeChild(@PathVariable(value = "deptId", required = false) Long deptId)
{...}

/**
 * 根据部门编号获取详细信息
 */
@RequiresPermissions("system:dept:query")
@GetMapping(value = "/{deptId}")
public AjaxResult getInfo(@PathVariable Long deptId)
{...}
```

3.3.3.1.5、岗位管理服务

菜单岗位管理功能由 SysPostController 类实现。基本实现了获取岗位列表、根据岗位编号获取详细信息、获取岗位选择框列表、新增岗位、修改岗位、删除岗位等功能。

部分接口如下：

```
/**
 * 获取岗位列表
 */
@RequiresPermissions("system:post:list")
@GetMapping("/list")
public TableDataInfo list(SysPost post)
{...}

@Log(title = "岗位管理", businessType = BusinessType.EXPORT)
@RequiresPermissions("system:post:export")
@PostMapping("/export")
public void export(HttpServletResponse response, SysPost post) throws IOException
{...}

/**
 * 根据岗位编号获取详细信息
 */
@RequiresPermissions("system:post:query")
@GetMapping(value = "/{postId}")
public AjaxResult getInfo(@PathVariable Long postId)
{...}
```

3.3.3.1.6、字典管理服务

字典管理功能由 SysDictTypeController 类实现。

部分接口如下：

```
/**
 * 查询字典数据详细
 */
@RequiresPermissions("system:dict:query")
@GetMapping(value = "/{dictCode}")
public AjaxResult getInfo(@PathVariable Long dictCode)
{...}

/**
 * 根据字典类型查询字典数据信息
 */
@GetMapping(value = "/type/{dictType}")
public AjaxResult dictType(@PathVariable String dictType)
{...}

/**
 * 新增字典类型
 */
@RequiresPermissions("system:dict:add")
@Log(title = "字典数据", businessType = BusinessType.INSERT)
@PostMapping
public AjaxResult add(@Validated @RequestBody SysDictData dict)
{...}
```

3.3.3.1.7、参数设置服务

参数设置功能由 SysConfigController 类实现。基本实现了获取参数配置列表、根据参数编号获取详细信息、根据参数键名查询参数值、新增参数配置、修改参数配置、删除参数配置、刷新参数缓存等功能。

部分接口如下：

```
/**
 * 根据参数编号获取详细信息
 */
@GetMapping(value = "/{configId}")
public AjaxResult getInfo(@PathVariable Long configId)
{...}

/**
 * 根据参数键名查询参数值
 */
@GetMapping(value = "/configKey/{configKey}")
public AjaxResult getConfigKey(@PathVariable String configKey)
{
    return AjaxResult.success(configService.selectConfigByKey(configKey));
}

/**
 * 新增参数配置
 */
@RequiresPermissions("system:config:add")
@Log(title = "参数管理", businessType = BusinessType.INSERT)
@PostMapping
public AjaxResult add(@Validated @RequestBody SysConfig config)
{...}
```

3.3.3.1.8、通知公告服务

通知公告服务功能由 SysNoticeController 类实现。基本实现了获取通知公告列表、根据通知编号获取详细信息、新增通知公告、修改通知公告、删除通知公告等功能。

部分接口如下：

```
/**
 * 获取通知公告列表
 */
@RequiresPermissions("system:notice:list")
@GetMapping("/list")
public TableDataInfo list(SysNotice notice)
{...}

/**
 * 根据通知公告编号获取详细信息
 */
@RequiresPermissions("system:notice:query")
@GetMapping(value = "/{noticeId}")
public AjaxResult getInfo(@PathVariable Long noticeId)
{...}

/**
 * 新增通知公告
 */
@RequiresPermissions("system:notice:add")
@Log(title = "通知公告", businessType = BusinessType.INSERT)
@PostMapping
public AjaxResult add(@Validated @RequestBody SysNotice notice)
{...}
```

3.3.3.1.9、操作日志服务

操作日志功能由 SysOperlogController 类实现。基本实现了新增操作日志、查询系统操作日志集合、批量删除系统操作日志、查询操作日志详情、清空操作日志的功能。

部分接口如下：

```
@Log(title = "操作日志", businessType = BusinessType.EXPORT)
@RequiresPermissions("system:operlog:export")
@PostMapping("/export")
public void export(HttpServletResponse response, SysOperLog operLog) throws IOException
{...}

@Log(title = "操作日志", businessType = BusinessType.DELETE)
@RequiresPermissions("system:operlog:remove")
@DeleteMapping("/{operIds}")
public AjaxResult remove(@PathVariable Long[] operIds)
{...}

@RequiresPermissions("system:operlog:remove")
@Log(title = "操作日志", businessType = BusinessType.CLEAN)
@DeleteMapping("/clean")
public AjaxResult clean()
{...}

@InnerAuth
@PostMapping
public AjaxResult add(@RequestBody SysOperLog operLog) { return toAjax(operLogService.insertOperlog(operLog)); }
}
```

3.3.3.1.10、登录日志服务

登录日志功能由 SysLogininforController 类实现。基本实现了新增登录日志、查询系统登录日志集合、批量删除系统登录日志、查询登录日志详情、清空登录日志的功能。

部分接口如下：

```
    @Log(title = "登录日志", businessType = BusinessType.EXPORT)
    @RequiresPermissions("system:logininfor:export")
    @PostMapping("/export")
    public void export(HttpServletResponse response, SysLogininfor logininfor) throws IOException
    {...}

    @RequiresPermissions("system:logininfor:remove")
    @Log(title = "登录日志", businessType = BusinessType.DELETE)
    @DeleteMapping("/{infoIds}")
    public AjaxResult remove(@PathVariable Long[] infoIds)
    {...}

    @RequiresPermissions("system:logininfor:remove")
    @Log(title = "登录日志", businessType = BusinessType.DELETE)
    @DeleteMapping("/clean")
    public AjaxResult clean()
    {...}

    @InnerAuth
    @PostMapping
    public AjaxResult add(@RequestBody SysLogininfor logininfor)
    {...}
}
```

3.3.3.2、系统监控模块

3.3.3.2.1、在线用户管理服务

在线用户管理功能由 ruoyi-system 包中的 SysUserOnlineController 类实现。实现了通过登录地址、用户名查询信息，设置在线用户信息、强退用户的功能。

接口如下：

```
    @RequiresPermissions("monitor:online:list")
    @GetMapping("/list")
    public TableDataInfo list(String ipaddr, String userName)
    {...}

    /**
     * 强退用户
     */
    @RequiresPermissions("monitor:online:forceLogout")
    @Log(title = "在线用户", businessType = BusinessType.FORCE)
    @DeleteMapping("/{tokenId}")
    public AjaxResult forceLogout(@PathVariable String tokenId)
    {...}
}
```

3.3.3.2.2、定时任务服务

在线用户管理功能由 ruoyi-job 包中的 SysUserJobController 类和 SysUserJobLogController 类实现。SysUserJobController 实现了调度任务信息操作处理，包括查询定时任务列表、导出定时任务列表、获取定时任务详细信息、新增定时任务、修改定时任务、删除定时任务等功能；SysUserJobLogControlle 实现了调度日志操作处理，包括查询定时任务调度日志列表、导出定时任务调度日志列表、根据调度编号获取详细信息、删除定时任务调度日志、清空定时任务调度日志。

SysUserJobController 部分接口如下：

```
/**
 * 查询定时任务列表
 */
@RequiresPermissions("monitor:job:list")
@GetMapping("/list")
public TableDataInfo list(SysJob sysJob)
{...}

/**
 * 导出定时任务列表
 */
@RequiresPermissions("monitor:job:export")
@Log(title = "定时任务", businessType = BusinessType.EXPORT)
@PostMapping("/export")
public void export(HttpServletResponse response, SysJob sysJob) throws IOException
{...}

/**
 * 获取定时任务详细信息
 */
@RequiresPermissions("monitor:job:query")
@GetMapping(value = "/{jobId}")
public AjaxResult getInfo(@PathVariable("jobId") Long jobId)
{...}
```

SysUserJobLogController 部分接口如下：

```
/**
 * 查询定时任务调度日志列表
 */
@RequiresPermissions("monitor:job:list")
@GetMapping("/list")
public TableDataInfo list(SysJobLog sysJobLog)
{...}

/**
 * 导出定时任务调度日志列表
 */
@RequiresPermissions("monitor:job:export")
@Log(title = "任务调度日志", businessType = BusinessType.EXPORT)
@PostMapping("/export")
public void export(HttpServletResponse response, SysJobLog sysJobLog) throws IOException
{...}

/**
 * 根据调度编号获取详细信息
 */
@RequiresPermissions("monitor:job:query")
@GetMapping(value = "/{configId}")
public AjaxResult getInfo(@PathVariable Long jobLogId)
{...}
```

3.3.3.2.3、服务监控服务

服务监控功能由 ruoyi-monitor 包中的 WebSecurityConfigurer 类实现，通过继承 WebSecurityConfigurerAdapter，从而实现监控权限配置。

3.3.3.2..4、缓存监控服务

略。

3.3.3.3、系统工具

3.3.3.3.1、表单构建服务

略。

3.3.3.3.2、代码生成服务

代码生成功能由 ruoyi-gen 包中的 GenController 类实现，GenController 实现了代码生成操作处理，包括查询代码生成列表、修改代码生成业务、查询数据库列表、查询数据表字段列表、导入表结构、修改保存代码生成业务、删除代码生成、预览代码、同步数据库、批量生成代码、生成 zip 文件等功能。

部分接口如下：

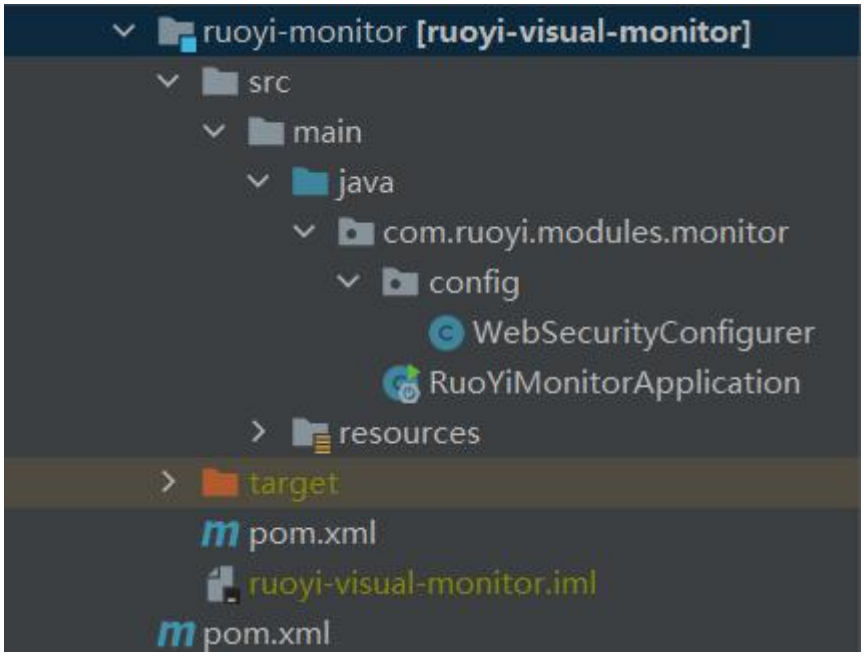
```
/**
 * 删除代码生成
 */
@RequiresPermissions("tool:gen:remove")
@Log(title = "代码生成", businessType = BusinessType.DELETE)
@DeleteMapping("/{tableIds}")
public AjaxResult remove(@PathVariable Long[] tableIds)
{...}

/**
 * 预览代码
 */
@RequiresPermissions("tool:gen:preview")
@GetMapping("/preview/{tableId}")
public AjaxResult preview(@PathVariable("tableId") Long tableId) throws IOException
{...}

/**
 * 生成代码（下载方式）
 */
@RequiresPermissions("tool:gen:code")
@Log(title = "代码生成", businessType = BusinessType.GENCODE)
@GetMapping("/download/{tableName}")
public void download(HttpServletResponse response, @PathVariable("tableName") String tableName) throws IOException
{...}
```

3.3.3.3.3、系统接口服务

系统接口功能由 ruoyi-api-system 包来完成，三个 service 为 RemoteFileService、RemoteLogService、RemoteUserService，分别提供文件服务接口、日志服务接口（保存系统日志、保存访问记录）、用户服务接口（通过用户名查询用户信息、注册用户信息）

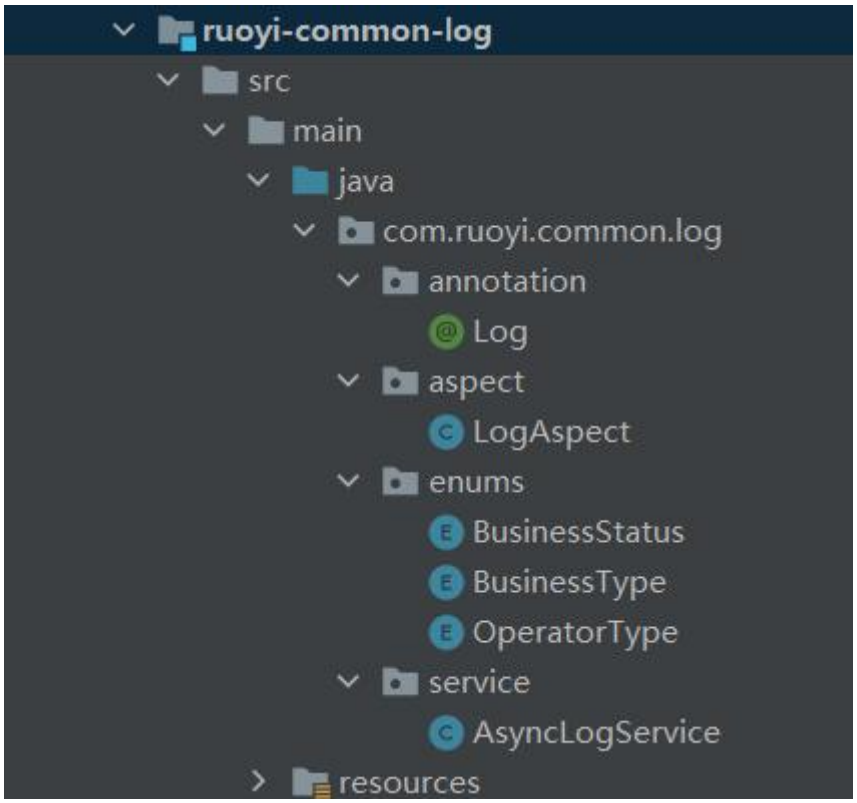
3.3.5、服务监控

服务监控组件由 ruoyi-monitor 包中的 WebSecurityConfigurer 类实现，通过继承 WebSecurityConfigurerAdapter，从而实现监控权限配置。监视当前系统应用状态、内存、线程、堆栈、日志等等相关信息，主要目的在服务出现问题或者快要出现问题时能够准确快速地发现以减小影响范围。



3.3.6、分布式日志

分布式日志组件由 ruoyi-common-log 包实现，三个 enums 类为 BusinessStatus、BusinesType、Operator,分别记录操作状态、业务操作类型、操作人类别。而 AsyncLogService 提供了异步调用日志服务的功能。



3.3.7、缓存服务 Redis

缓存服务组件由 ruoyi-common-redis 包下的 RedisService【spring redis 工具类】实现，功能包括设置有效时间、获取有效时间、判读 key 是否存在、获取缓存的基本对象、删除单个对象、删除集合对象、缓存 List 数据、缓存 Set、缓存 Map、缓存 Hash 等等。大体实现了缓存服务的基本功能。

第四章 软件架构分析评价

对于 RuoYi-cloud 系统，我们不难发现其功能众多，工作量浩大（这点我在分析若依系统需求的时候已经深有体会，QAQ）。而若依系统 RuoYi-cloud 采用的基 SpringBoot 的微服务架构很好的解决了这一问题，它将复杂的应用程序分解成多模块、多服务。虽然功能总量不变，但应用程序已分解为可管理的块或服务。并且在分解的同时，规避了原本复杂度无止境的积累。每一个微服务专注于单一功能，并通过定义良好的接口清晰表述服务边界。由于每个微服务相对简单，故需要对技术栈进行升级时所面临的风险就较低，甚至完全重构一个微服务也是可行的。每个服务也可以根据后续的需求独立扩展。

RuoYi-cloud 系统架构的关键问题和最重要经验就是不同于一般的软件，RuoYi-cloud 是一个集成了各种模块的开发框架，正是微服务架构低耦合、高度自治和轻量级的特点保障了开发的灵活性，以及各个功能模块之间互不干扰独立进行，最符合 RuoYi-cloud 集成开发平台的特点。因此对于 RuoYi-cloud 来说，架构演化的方向是向低耦合，高独立性的方向发展，但是要保证统一的配置中心，在加强独立性的同时，仍要保持统一的掌控。

然而，微服务架构也存在不少缺点。在分析架构的过程中，我发现一个主要的缺点就是系统设计的复杂性—即更高数量级的活动组件（服务、数据库、进程、容器、框架等等），这就导致文件包众多，每个包下子文件的设计也是五花八门，我分析起来也比较的棘手，所以 RuoYi-cloud 的部分文件我暂时还没有完全分析透彻。其次就是分布式系统带来的复杂性，需要选择和实现基于消息传递或 RPC 的进程间通信机制。此外，他们还必须编写代码来处理部分故障。毫无疑问，这些增都加了系统的规模。

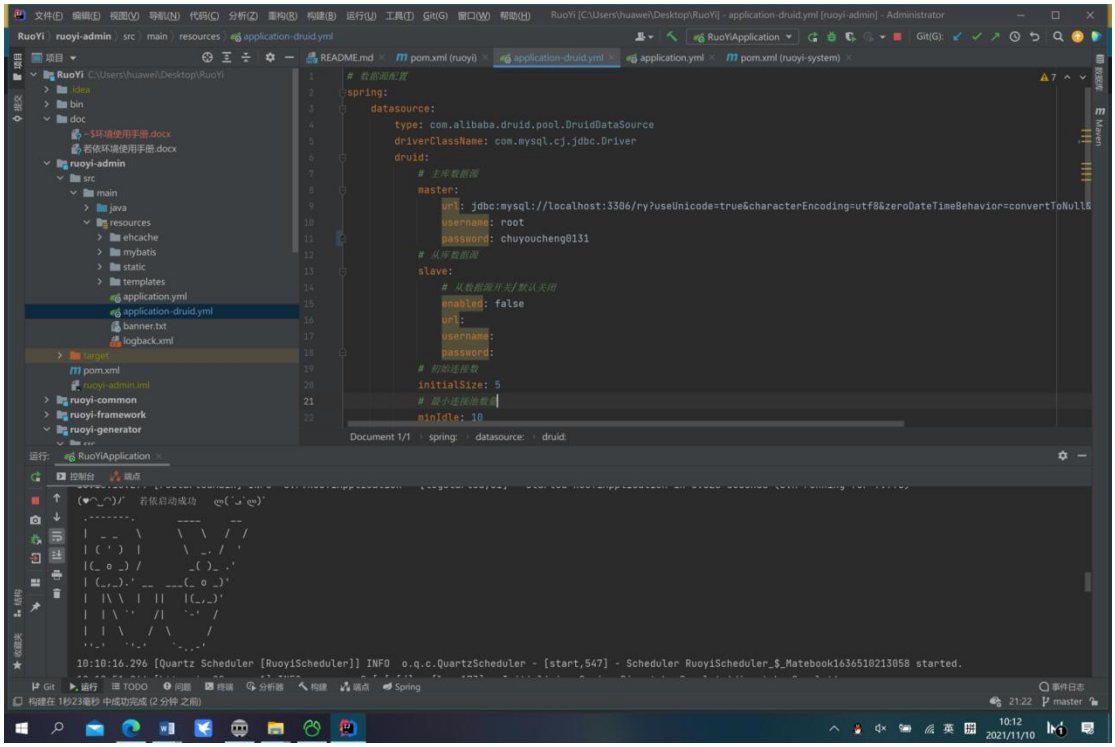
在服务的划分方面，我也发现了一些问题，系统的有些服务划分并不合理，甚至有为了划分而划分的嫌疑。这是因为缺乏统一的标准，只能根据架构师的个人工程经验。但我认为，微服务是达到目的的手段，而不是目标。微服务的目标是充分分解应用程序，以促进敏捷开发和持续集成部署。

最后是我对微服务的一些思考。对于微服务架构，我认为更多的应该是思维上的转变：技术上不是问题，意识比工具重要。以下是关于微服务的几点设计出发点：

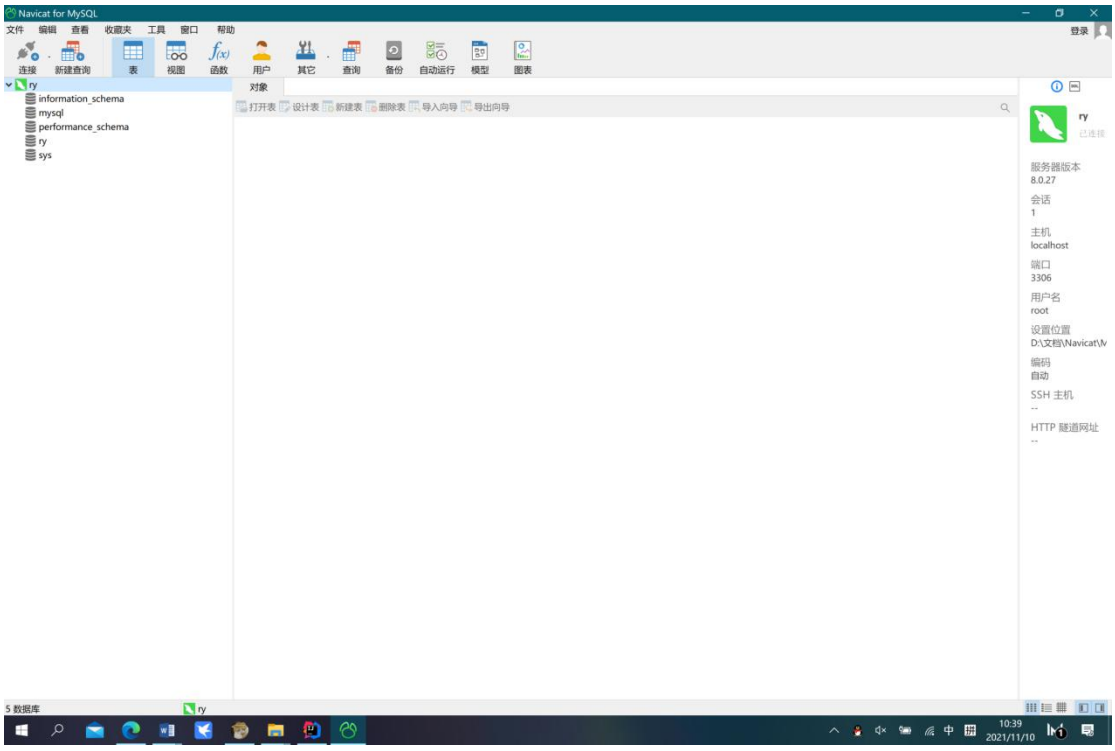
- 应用程序的核心是业务逻辑，按照业务或客户需求组织资源（这是最难的）
- 做有生命的产品，而不是项目
- 后台服务贯彻 Single Responsibility Principle（单一职责原则）
- VM->Docker （to PE）
- DevOps (to PE)

第五章 软件测试结果

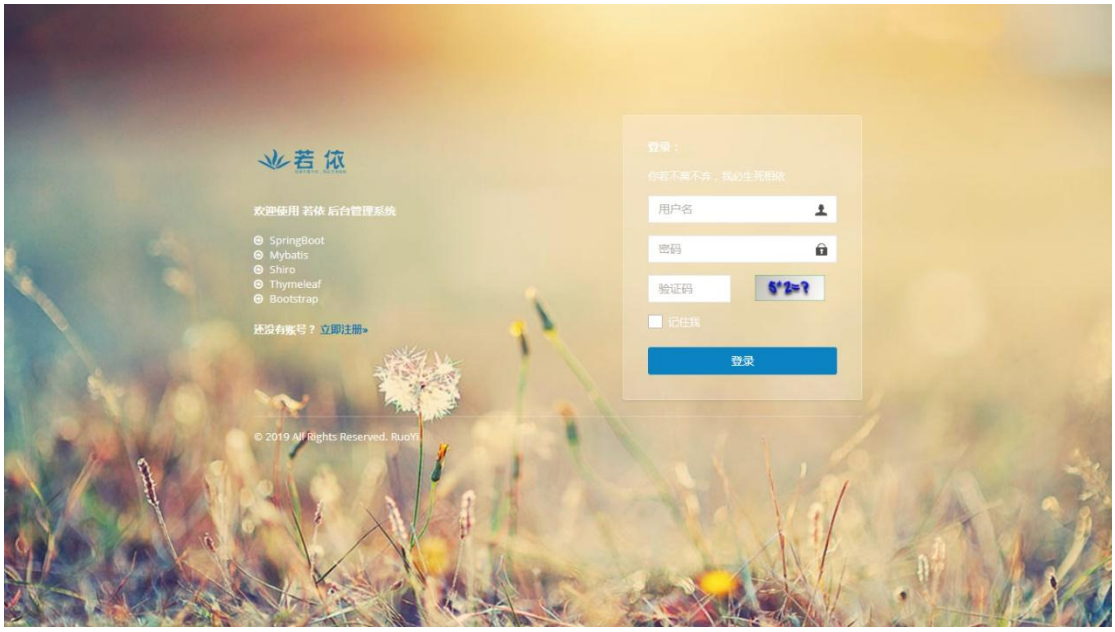
5.1、后端：



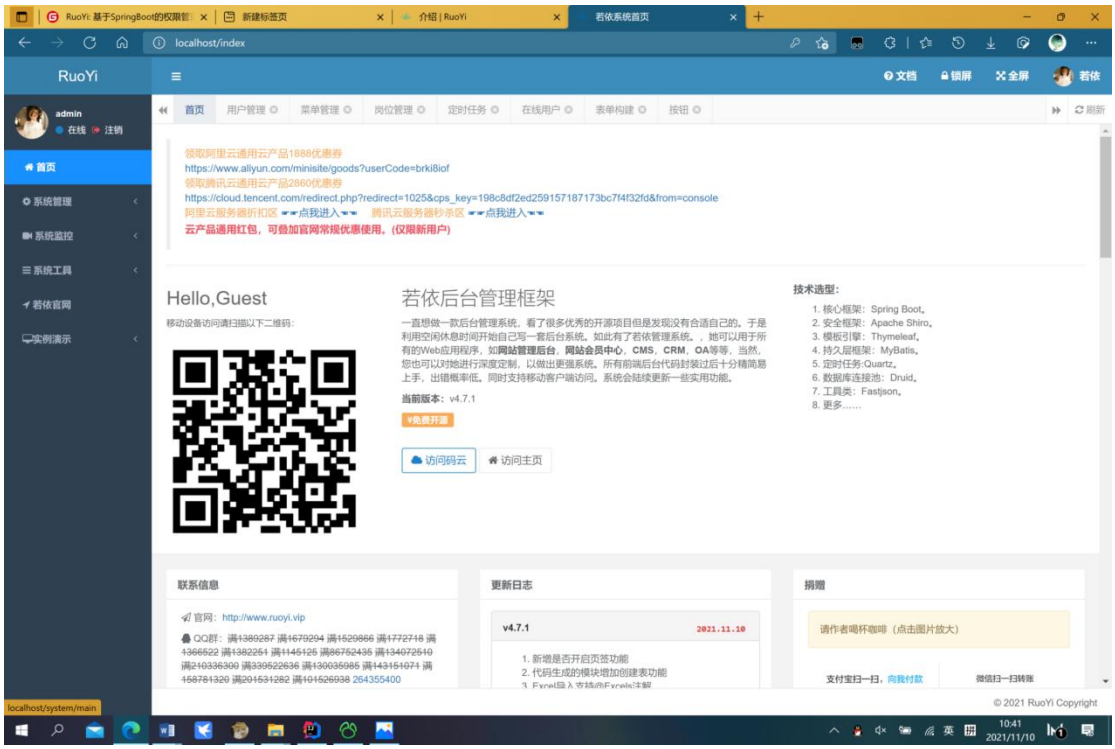
5.2、数据库：



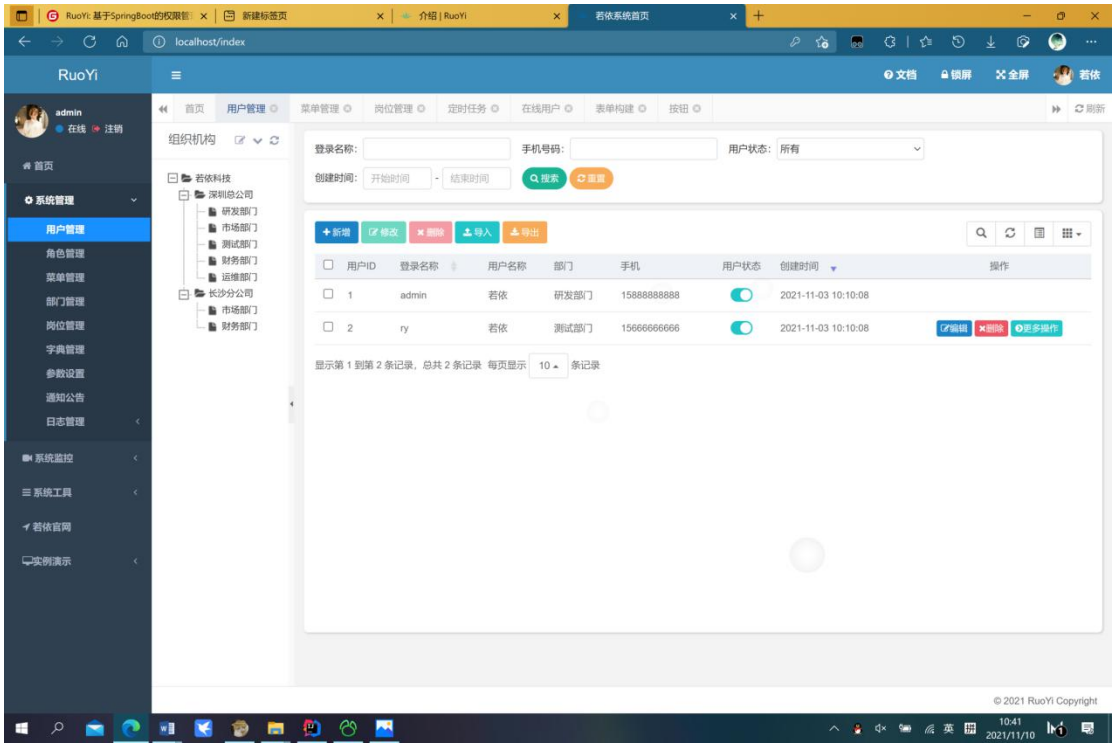
5.3、登陆界面与验证界面：



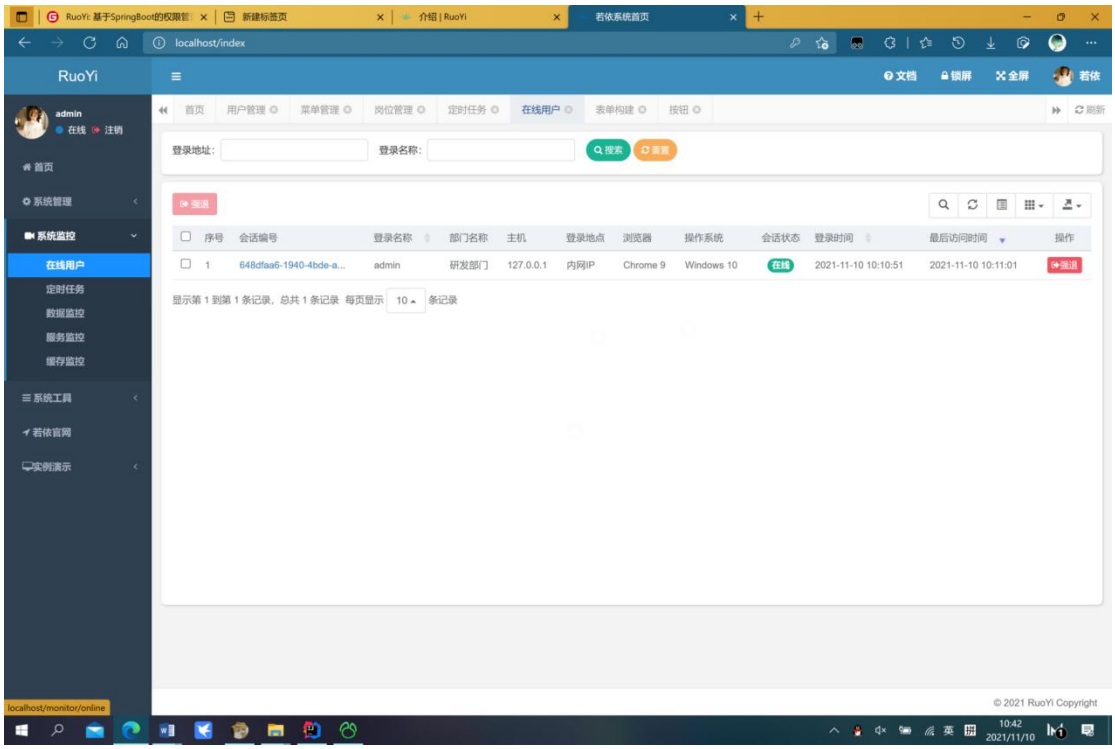
5.4、首页：



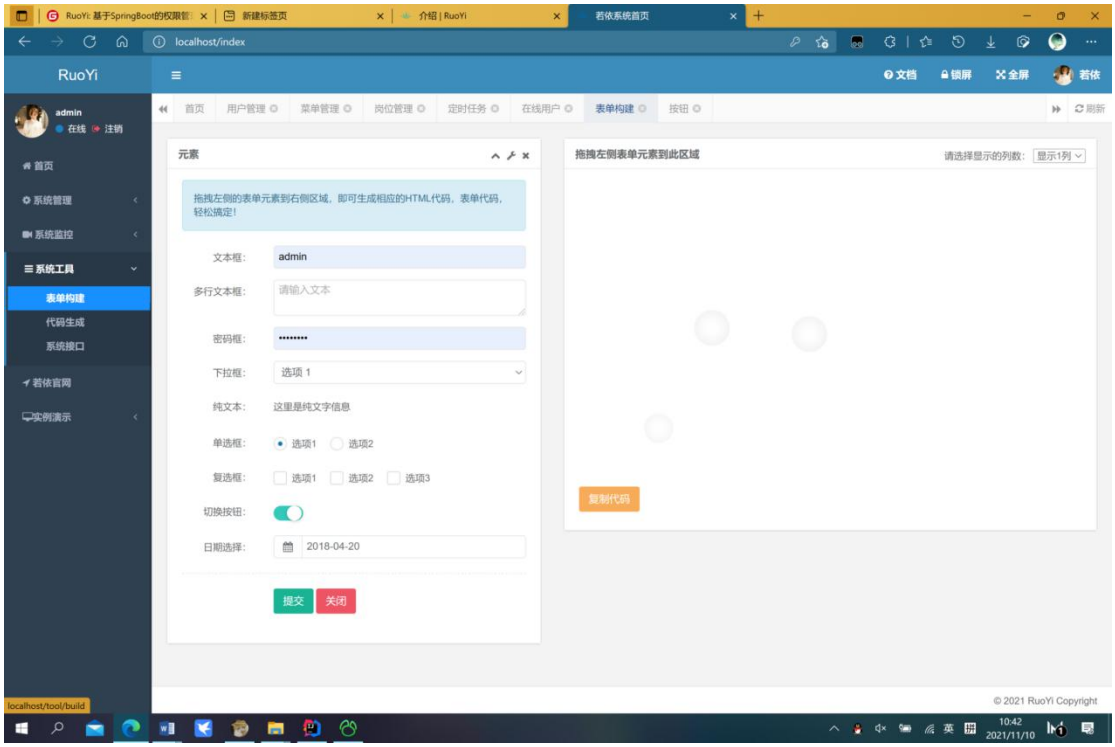
5.5、系统管理:



5.6、系统监控:



5.7、系统工具


RuoYi
admin
在线 注销
首页
系统管理
系统监控
系统工具
表单构建
代码生成
系统接口
若依官网
实例演示
localhost/index
文档
锁屏
全屏
若依
首页
用户管理
菜单管理
岗位管理
定时任务
在线用户
表单构建
按钮
刷新
元素
拖拽左侧的表单元素到右侧区域，即可生成相应的HTML代码，表单代码，轻松搞定！
文本框:
admin
多行文本框:
请输入文本
密码框:
下拉框:
选项 1
纯文本:
这里是纯文字信息
单选框:
选项1
选项2
复选框:
选项1
选项2
选项3
切换按钮:
日期选择:
2018-04-20
提交
关闭
拖拽左侧表单元素到此区域
请选择显示的列数:
显示1列
复制代码
© 2021 RuoYi Copyright
localhost/tool/build
10:42
2021/11/10

第六章 总结

通过本次实验，我了解了业界比较先进的架构模型—分布式微服务架构，了解了架构领域最新的发展趋势：微服务化、分布式，深入了解了微服务架构的模块组成与结构，并深入分析了微服务模块的优势与劣势以及一些思考。同时我也了解了 springboot、springcloud、nacos 等技术，对应用业有了更新的了解。

首先是自我批判部分，第一次接触一个这么大的一个项目（虽然只是进行简单的架构分析），尤其是第一次接触微服务架构，内心其实是害怕的。在跑通前后端并完成需求分析后，面对繁杂冗余的文件包以及各种类，我对架构的分析迟迟下不去手。虽然已经被分为多个模块，但各模块的文件量依旧吓人，类图无从下手，实在是无能为力。因此，我换了个思路，将各组件单独分析，找出实现该组件的包、类，勉强完成了任务。

架构方面，我得出的架构图和官网文档中的架构图相比，还是有很多小细节没有考虑到，比如说消息队列中间件、数据同步工具等等。由于个人能力问题以及 RuoYi-cloud 系统的复杂度、时间问题等多方因素，没有将所有模块的类图落实到位，部分文件也没有研究透彻，是本报告中存在的最大的问题。

最后是关于我对架构的一些新的认知，什么样的架构才是好架构呢？

在通过本次实验的实践后，我认为并不是越复杂越好，而是相反，简单就是硬道理也体现在这里。这也是微服务能够流行的原因，看看市场上曾经出现的服务架构：EJB、SCA、Dubbo 等等，都比微服务先进，都比微服务功能完善，但它们都没有微服务这么深入民心，就是因为他们过于复杂。简单就是高科技，大公司都是由小公司发展起来的，如果小公司在开始技术选型时感觉某个框架费时费力就不会选择，而小公司发展到大公司的过程，一般也伴随着系统不断优化的过程，而不断优化往往不会重新选择开发技术和框架，而是在原来基础改进，这也许就是简单框架流行的本质。