视频教程笔记

往期项目

鱼皮系列项目以 **实战** 为主，用全程直播的方式，从 0 到 1 带大家学习技术知识，并立即实践运用到项目中。

此外，还提供了详细的直播笔记、项目源码、项目扩展思路、现成的简历写法等。

每个项目的侧重点不同：

- 用户中心：适合学完框架的新手入门，系统学习完整的项目开发流程
- 伙伴匹配系统：巩固开发流程，学习 Redis、事务、并发编程、大数据推荐思想等后端知识
- API 开放平台：系统学习并实践架构设计 + sdk 开发 + API 签名认证 + RPC + 微服务网关
- 聚合搜索平台：系统学习并实践爬虫 + Elastic Stack + 设计模式
- 智能 BI 项目：系统学习并实践异步化、线程池、RabbitMQ、AIGC

第 1 期 - 实现方案和项目初始化

项目介绍

OJ = Online Judge 在线判题评测系统

用户可以选择题目，在线做题，编写代码并且提交代码；系统会对用户提交的代码，根据我们出题人设置的答案，来判断用户的提交结果是否正确。

ACM（程序设计竞赛），也是需要依赖判题系统来检测参赛者的答案是否合理

OJ 系统最大的难点在于 **判题系统**

用于在线评测编程题目代码的系统，能够根据用户提交的代码、出题人预先设置的题目输入和输出用例，进行编译代码、运行代码、判断代码运行结果是否正确。

判题系统作为一个开放 API 提供给大家，便于开发者开发自己的 OJ 系统。

OJ 系统的常用概念

ac 表示你的题目通过，结果正确

题目限制：时间限制、内存限制

题目介绍

题目输入

题目输出

题目输入用例

题目输出用例

普通测评：管理员设置题目的输入和输出用例，比如我输入 1，你要输出 2 才是正确的；交给判题机去执行用户的代码，给用户的代码喂输入用例，比如 1，看用户程序的执行结果是否和标准答案的输出一致。

（比对用例文件）

特殊测评（SPJ）：管理员设置题目的输入和输出，比如我输入 1，用户的答案只要是 > 0 或 < 2 都是正确的；特判程序，不是通过对比用例文件是否一致这种死板的程序来检验，而是要

专门根据这道题目写一个特殊的判断程序，程序接收题目的输入（1）、标准输出用例（2）、用户的结果（1.5），特判程序根据这些值来比较是否正确。

交互测评：让用户输入一个例子，就给一个输出结果，交互比较灵活，没办法通过简单的、死板的输入输出文件来搞定

不能让用户随便引入包、随便遍历、暴力破解，需要使用正确的算法。 => 安全性

判题过程是异步的 => 异步化

提交之后，会生成一个提交记录，有运行的结果以及运行信息（时间限制、内存限制）

为什么带大家做这个项目?

1. 这个项目网上教程很少，基本上找不到教程
2. 比较新颖，写在简历上会有区分度、有亮点（人家写外卖，你写 OJ）
3. 能学到东西，相比于传统的 CRUD 来讲，这个项目的 CRUD 成分很少，更多的在于一些编程思想、计算机基础、架构设计方面的知识
4. 复杂度 “高”，很多同学觉得 OJ 很难做，一起来攻克它
5. 可扩展性非常强

复习做项目的流程

1. 项目介绍、项目调研、需求分析
2. 核心业务流程
3. 项目要做的功能（功能模块）
4. 技术选型（技术预研）
5. 项目初始化
6. 项目开发
7. 测试
8. 优化
9. 代码提交、代码审核
10. 产品验收

11．上线

写文档、持续调研、持续记录总结

现有系统调研

https://github.com/HimitZH/HOJ <https://github.com/HimitZH/HOJ>（适合学习）

https://github.com/QingdaoU/OnlineJudge <https://github.com/QingdaoU/OnlineJudge>（python，不好学，很成熟）

https://github.com/hzxie/voj <https://github.com/hzxie/voj>（星星没那么多，没那么成熟，但相对好学）

https://github.com/vfleaking/uoj <https://github.com/vfleaking/uoj>（php 实现的）

https://github.com/zhblue/hustoj <https://github.com/zhblue/hustoj>（成熟，但是 php）

https://github.com/hydro-dev/Hydro <https://github.com/hydro-dev/Hydro>（功能强大，Node.js 实现）

实现核心

1）权限校验

谁能提代码，谁不能提代码

2）代码沙箱（安全沙箱）

用户代码藏毒：写个木马文件、修改系统权限

沙箱：隔离的、安全的环境，用户的代码不会影响到沙箱之外的系统的运行

资源分配：系统的内存就 2 个 G，用户疯狂占用资源占满你的内存，其他人就用不了了。所以要限制用户程序的占用资源。

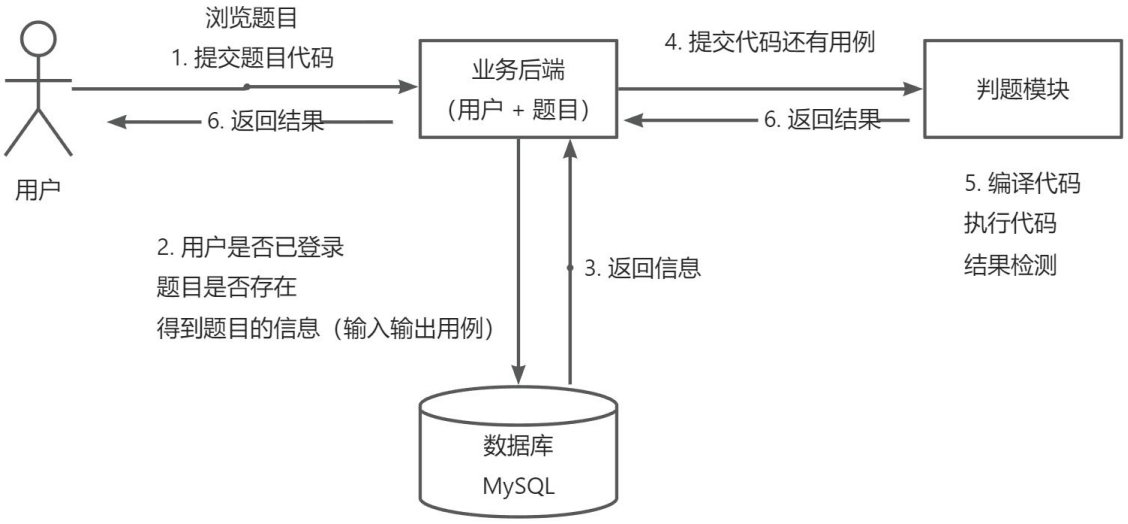
3）判题规则

题目用例的比对，结果的验证

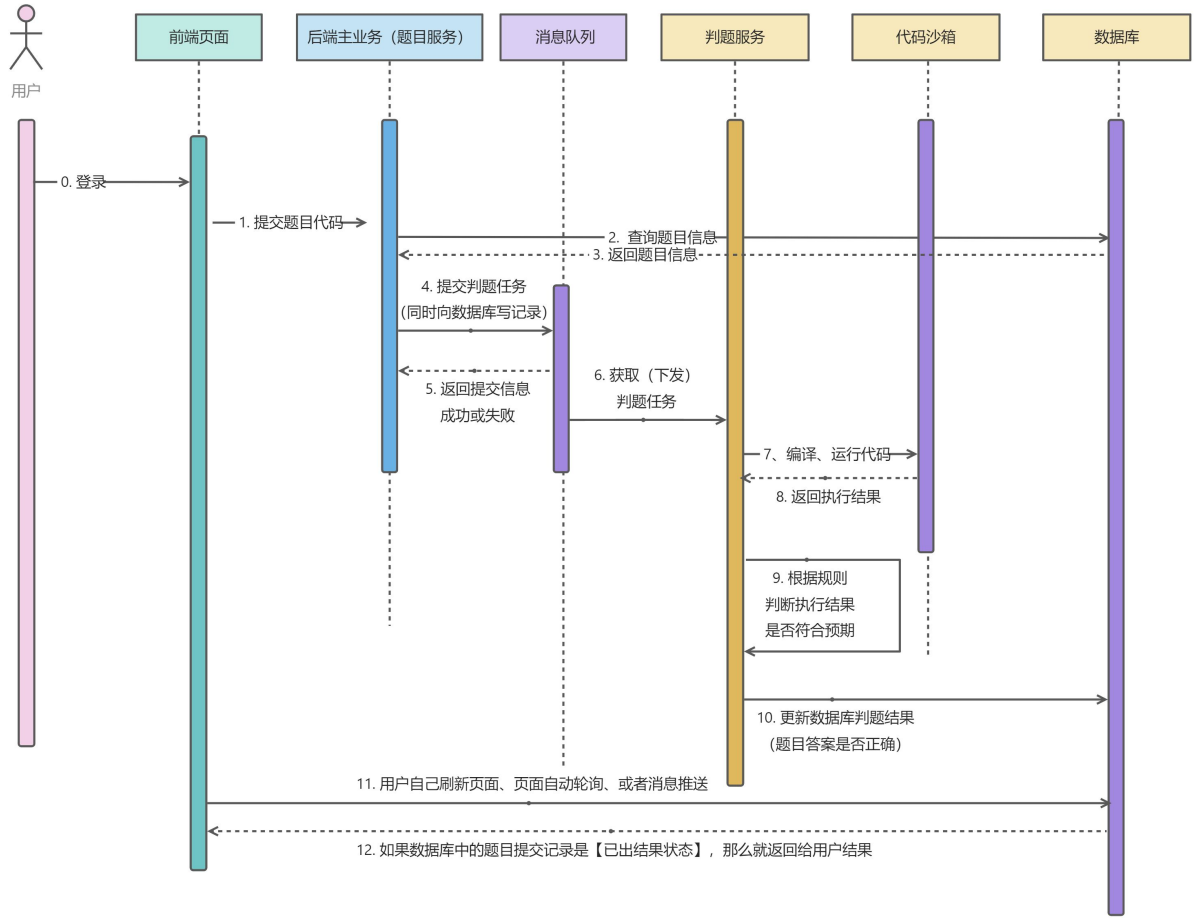
4）任务调度

服务器资源有限，用户要排队，按照顺序去依次执行判题，而不是直接拒绝

核心业务流程



> 为啥要编译?
> 因为有些语言不编译不能运行



判题服务：获取题目信息、预计的输入输出结果，返回给主业务后端：用户的答案是否正确

代码沙箱：只负责运行代码，给出结果，不管什么结果是正确的。

实现了解耦

功能

1. 题目模块
    a. 创建题目（管理员）
    b. 删除题目（管理员）
    c. 修改题目（管理员）
    d. 搜索题目（用户）
    e. 在线做题
    f. 提交题目代码
2. 用户模块
    a. 注册
    b. 登录
3. 判题模块
    a. 提交判题（结果是否正确与错误）
    b. 错误处理（内存溢出、安全性、超时）
    c. **自主实现** 代码沙箱（安全沙箱）
    d. 开放接口（提供一个独立的新服务）

项目扩展思路

1. 支持多种语言
2. Remote Judge
3. 完善的评测功能：普通测评、特殊测评、交互测评、在线自测、子任务分组评测、文件IO
4. 统计分析用户判题记录
5. 权限校验

技术选型

前后端全栈，所有都有，每行代码都是直播

前端：Vue3、Arco Design 组件库、手撸项目模板、在线代码编辑器、在线文档浏览

Java 进程控制、Java 安全管理器、部分 JVM 知识点

虚拟机（云服务器）、Docker（代码沙箱实现）

Spring Cloud 微服务 、消息队列、多种设计模式

架构设计

直播计划

1. 介绍 OJ 系统概念、介绍 OJ 原理、介绍做项目流程、介绍需求分析、技术选型、架构设计、主流实现方案、前后端项目初始化、前端通用项目模板的搭建
2. 主业务流程的前后端开发（争取把代码沙箱之外的全部搞定）
3. 专攻代码沙箱（自主实现，不止一种实现方案，层层递进，通过实战用例来进行安全优化）
4. 系统优化（微服务改造、系统扩展思路）

9 月前一定搞定！

主流的 OJ 系统实现方案

开发原则：能用别人现成的，就不要自己写

1、用现成的 OJ 系统

网上有很多开源的 OJ 项目，比如青岛 OJ、HustOJ 等，可以直接下载开源代码自己部署。

比较推荐的是 judge0，这是一个非常成熟的商业 OJ 项目，支持 60 多种编程语言！

> 代码：https://github.com/judge0/judge0 <https://github.com/judge0/judge0>

2、用现成的服务

如果你不希望完整部署一套大而全的代码，只是想复用他人已经实现的、最复杂的判题逻辑，那么可以直接使用现成的 **判题 API** 、或者现成的 **代码沙箱** 等服务。

比如 judge0 提供的判题 API，非常方便易用。只需要通过 HTTP 调用 submissions 判题接口，把用户的代码、输入值、预期的执行结果作为请求参数发送给 judge0 的服务器，它就能自动帮你编译执行程序，并且返回程序的运行结果。

如下图，发送了一段打印 "hello world" 的程序，得到了程序执行的时间、状态等：

> API 的作用：接受代码、返回执行结果

Judge0 API 地址：https://rapidapi.com/judge0-official/api/judge0-ce <https://rapidapi.com/judge0-official/api/judge0-ce>

官方文档：https://ce.judge0.com/#submissions-submission-post <https://ce.judge0.com/#submissions-submission-post>

流程

1. 先注册
2. 再开通订阅
3. 然后测试 language 接口
4. 测试执行代码接口 submissions

示例接口参数：

预期返回：

3、自主开发

这种方式就不多说了，判题服务和代码沙箱都要自己实现，适合学习，但不适用于商业项目。我这次带大家做的 OJ 系统，就选择了自主开发，主打一个学习。

4、把 AI 来当做代码沙箱

现在 AI 的能力已经十分强大了，我们可以把各种本来很复杂的功能直接交给 AI 来实现。

比如把 AI 当做代码沙箱，我们直接扔给他一段代码、输入参数，问他能否得到预期的结果，就实现了在线判题逻辑！

如下图：

之前带大家做的 智能 BI 项目 <https://www.code-nav.cn/course/1790980531403927553> ，就是把 AI 当做了智能数据分析师，来生成图表和分析结论。

只要你脑洞够大，AI + 编程 = 无限的可能~

5、移花接木

这种方式最有意思、也最 “缺德” 。很多同学估计想不到。

那就是可以通过让程序来操作模拟浏览器的方式，用别人已经开发好的 OJ 系统来帮咱们判题。

比如使用 Puppeteer + 无头浏览器，把咱们系统用户提交的代码，像人一样输入到别人的 OJ 网页中，让程序点击提交按钮，并且等别人的 OJ 系统返回判题结果后，再把这个结果返回给我们自己的用户。

这种方式的缺点就是把核心流程交给了别人，如果别人服务挂了，你的服务也就挂了；而且别人 OJ 系统不支持的题目，可能你也支持不了。

前端项目初始化 - 通用模板实现

> 自主打造一套前端开发项目模板！

确认环境！！！

nodeJS 版本：v18.16.0 或 16

检测命令：

```
```

切换和管理 node 版本的工具：https://github.com/nvm-sh/nvm <https://github.com/nvm-sh/nvm>

npm 版本：9.5.1

```
```

初始化

使用 vue-cli 脚手架：https://cli.vuejs.org/zh/ <https://cli.vuejs.org/zh/>

安装脚手架工具：

```
```

检测是否安装成功：

```
```

如果找不到命令，那么建议去重新到安装 npm，重新帮你配置环境变量。

创建项目：

```
```

运行项目，能运行就成功了

前端工程化配置

脚手架已经帮我们配置了代码美化、自动校验、格式化插件等，无需再自行配置

但是需要在 webstorm 里开启代码美化插件：

在 vue 文件中执行格式化快捷键，不报错，表示配置工程化成功

脚手架自动整合了 vue-router

自己整合

代码规范：https://eslint.org/docs/latest/use/getting-started <https://eslint.org/docs/latest/use/getting-started>

代码美化：https://prettier.io/docs/en/install.html <https://prettier.io/docs/en/install.html>

直接整合：https://github.com/prettier/eslint-plugin-prettier#recommended-configuration <https://github.com/prettier/eslint-plugin-prettier#recommended-configuration> （包括了 https://github.com/prettier/eslint-config-prettier#installation <https://github.com/prettier/eslint-config-prettier#installation> ）

引入组件

> Vue Router 路由组件已自动引入，无需再引入：https://router.vuejs.org/zh/introduction.html <https://router.vuejs.org/zh/introduction.html>

组件库：https://arco.design/vue <https://arco.design/vue>

快速上手：https://arco.design/vue/docs/start <https://arco.design/vue/docs/start>

执行安装：

改变 main.ts：

引入一个组件，如果显示出来，就表示引入成功

项目通用布局

新建一个布局， 在 app.vue 中引入

app.vue 代码如下：

选用 arco design 的 layout 组件（https://arco.design/vue/component/layout <https://arco.design/vue/component/layout> ）

先把上中下布局编排好，然后再填充内容：

实现通用路由菜单

菜单组件：https://arco.design/vue/component/menu <https://arco.design/vue/component/menu>

目标：根据路由配置信息，自动生成菜单内容。实现更通用、更自动的菜单配置。

步骤：

1）提取通用路由文件

2）菜单组件读取路由，动态渲染菜单项

3）绑定跳转事件

4）同步路由的更新到菜单项高亮

同步高亮原理：首先点击菜单项 => 触发点击事件，跳转更新路由 => 更新路由后，同步去更新菜单栏的高亮状态。

使用 Vue Router 的 afterEach 路由钩子实现：

全局状态管理

vuex：https://vuex.vuejs.org/zh/guide/ <https://vuex.vuejs.org/zh/guide/>（vue-cli 脚手架已自动引入）

什么是全局状态管理?

所有页面全局共享的变量，而不是局限在某一个页面中。

适合作为全局状态的数据：已登录用户信息（每个页面几乎都要用）

Vuex 的本质：给你提供了一套增删改查全局变量的 API，只不过可能多了一些功能（比如时间旅行）

可以直接参考购物车示例：https://github.com/vuejs/vuex/tree/main/examples/classic/

shopping-cart <https://github.com/vuejs/vuex/tree/main/examples/classic/shopping-cart>

state：存储的状态信息，比如用户信息

mutation（尽量同步）：定义了对变量进行增删改（更新）的方法

actions（支持异步）：执行异步操作，并且触发 mutation 的更改（actions 调用 mutation）

modules（模块）：把一个大的 state（全局变量）划分为多个小模块，比如 user 专门存用户的状态信息

实现

先在 store 目录下定义 user 模块，存储用户信息：

然后在 store 目录下定义 index.ts 文件，导入 user 模块：

在 Vue 页面中可以获取已存储的状态变量：

在 Vue 页面中可以修改状态变量：

> 使用 dispatch 来调用之前定义好的 actions

全局权限管理

目标：能够直接以一套通用的机制，去定义哪个页面需要那些权限。而不用每个页面独立去判断权限，提高效率。

思路：

1. 在路由配置文件， 定义某个路由的访问权限
2. 在全局页面组件 app.vue 中，绑定一个全局路由监听。每次访问页面时，根据用户要访问页面的路由信息，先判断用户是否有对应的访问权限。
3. 如果有，跳转到原页面；如果没有，拦截或跳转到 401 鉴权或登录页

示例代码如下：

优化页面布局

1、底部 footer 布局优化

2、优化 content、globalHeader 的样式

3、优化导航栏用户名称的换行

通用导航栏组件 - 根据配置控制菜单的显隐

1）routes.ts 给路由新增一个标志位，用于判断路由是否显隐

2）不要用 v-for + v-if 去条件渲染元素，这样会先循环所有的元素，导致性能的浪费

推荐：先过滤只需要展示的元素数组

根据权限隐藏菜单

需求：只有具有权限的菜单，才对用户可见

原理：类似上面的控制路由显示隐藏，只要判断用户没有这个权限，就直接过滤掉

1）新建 access 目录，专门用一个文件来定义权限

2）定义一个公用的权限校验方法

为什么？因为菜单组件中要判断权限、权限拦截也要用到权限判断功能，所以抽离成公共方法

创建 checkAccess.ts 文件，专门定义检测权限的函数：

3）修改 GlobalHeader 动态菜单组件，根据权限来过滤菜单

注意，这里使用计算属性，是为了当登录用户信息发生变更时，触发菜单栏的重新渲染，展示新增权限的菜单项

全局项目入口

app.vue 中预留一个可以编写全局初始化逻辑的代码：

后端项目初始化

先把通用的后端框架跑起来。

1）从星球代码库下载 springboot-init 万用模板（已经在本地的话直接复制）

2） `ctrl+shift+R` 全局替换 springboot-init 为项目名（yuoj-backend）

3）全局替换springbootinit 包名为新的包名（yuoj）

4）修改 springbootinit 文件夹的名称为新的包名对应的名称（yuoj）

5）本地新建数据库，直接执行 sql/create_table.sql 脚本，修改库名为 yuoj，执行即可

6）改 application.yml 配置，修改 MySQL 数据库的连接库名、账号密码，端口号（8121）

初始化模板讲解

1）先阅读 README.md

2）sql/create_table.sql 定义了数据库的初始化建库建表语句

3）sql/post_es_mapping.json 帖子表在 ES 中的建表语句

4）aop：用于全局权限校验、全局日志记录

5）common：万用的类，比如通用响应类

6）config：用于接收 application.yml 中的参数，初始化一些客户端的配置类（比如对象存储客户端）

7）constant：定义常量

8）controller：接受请求

9）esdao：类似 mybatis 的 mapper，用于操作 ES

10）exception：异常处理相关

11）job：任务相关（定时任务、单次任务）

12）manager：服务层（一般是定义一些公用的服务、对接第三方 API 等）

13）mapper：mybatis 的数据访问层，用于操作数据库

14）model：数据模型、实体类、包装类、枚举值

15）service：服务层，用于编写业务逻辑

16）utils：工具类，各种各样公用的方法

17）wxmp：公众号相关的包

18）test：单元测试

19）MainApplication：项目启动入口

20）Dockerfile：用于构建 Docker 镜像

前后端联调

问：前端和后端怎么连接起来的？ 接口 / 请求

答：前端发送请求调用后端接口

1）安装请求工具类 Axios

官方文档：https://axios-http.com/docs/intro <https://axios-http.com/docs/intro>

代码：

2）编写调用后端的代码

传统情况下，每个请求都要单独编写代码。至少得写一个请求路径

完全不用！！！

直接自动生成即可：https://github.com/ferdikoomen/openapi-typescript-codegen <https://github.com/ferdikoomen/openapi-typescript-codegen>

首先安装：

然后执行命令生成代码：

3）直接使用生成的 Service 代码，直接调用函数发送请求即可，比如获取登录信息

如果想要自定义请求参数，怎么办？

1）使用代码生成器提供的全局参数修改对象：

文档：https://github.com/ferdikoomen/openapi-typescript-codegen/blob/master/docs/openapi-object.md <https://github.com/ferdikoomen/openapi-typescript-codegen/blob/master/docs/openapi-object.md>

2）直接定义 axios 请求库的全局参数，比如全局请求响应拦截器

文档：https://axios-http.com/docs/interceptors <https://axios-http.com/docs/interceptors>

示例代码：

用户登录功能

自动登录

1）在 store\user.ts 编写获取远程登陆用户信息的代码：

2）在哪里去触发 getLoginUser 函数的执行？应当在一个全局的位置

有很多选择：

1. 路由拦截 ✔
2. 全局页面入口 app.vue
3. 全局通用布局（所有页面都共享的组件）

此处选择第一种方案，可以直接在全局权限管理的路由拦截中判断用户是否已经登录了。

全局权限管理优化

1）新建 access\index.ts 文件，把原有的路由拦截、权限校验逻辑放在独立的文件中

优势：只要不引入、就不会开启、不会对项目有影响

2）编写权限管理和自动登录逻辑

如果没登陆过，自动登录：

如果用户访问的页面不需要登录，是否需要强制跳转到登录页？

答：不需要

access\index.ts 示例代码：

支持多套布局

1）在 routes 路由文件中新建一套用户路由，使用 vue-router 自带的子路由机制，实现布局和嵌套路由

2）新建 UserLayout、UserLoginView、UserRegisterView 页面，并且在 routes 中引入

3）在 app.vue 根页面文件，根据路由去区分多套布局

注意，当前这种 app.vue 中通过 if else 区分布局的方式，不是最优雅的，理想情况下是直接读取 routes.ts，在这个文件中定义多套布局，然后自动使用页面布局。

小扩展：你可以尝试实现上面的思路，并且根据嵌套路由生成嵌套的子菜单。

如下图：

登陆页面开发

登陆页面的核心是表单，只需要从组件库中找到表单组件，修改表单字段名称和后端完全匹配就足够了。

界面示例代码：

本期成果

前端通用页面布局：

登陆页面：

后端接口文档：

本期作业

1. 理解 OJ 系统的概念以及核心流程
2. 开发完成前端初始化项目模板
3. 完成后端项目初始化
4. 实现用户登录功能
5. 自行实现用户注册页面

第 2 期 - 后端接口开发

友情提示

如果 windows 系统拉取前端代码后报错，请按照此文修改：https://blog.csdn.net/CSDN_YYD_997/article/details/129673984 <https://blog.csdn.net/CSDN_YYD_997/article/details/129673984>

计划

> 本期主打写代码

1. 系统功能的梳理
2. 库表设计（后端）
3. 后端接口开发

系统功能的梳理

1. 用户模块
    a. 注册（后端已实现）
    b. 登录（后端已实现，前端已实现）
2. 题目模块
    a. 创建题目（管理员）
    b. 删除题目（管理员）
    c. 修改题目（管理员）
    d. 搜索题目（用户）
    e. 在线做题（题目详情页）
3. 判题模块
    a. 提交判题（结果是否正确与错误）
    b. 错误处理（内存溢出、安全性、超时）
    c. **自主实现** 代码沙箱（安全沙箱）
    d. 开放接口（提供一个独立的新服务）

库表设计

用户表

只有管理员才能发布和管理题目，普通用户只能看题

题目表

题目标题

题目内容：存放题目的介绍、输入输出提示、描述、具体的详情

题目标签（json 数组字符串）：栈、队列、链表、简单、中等、困难

题目答案：管理员 / 用户设置的标准答案

提交数、通过题目的人数等：便于分析统计（可以考虑根据通过率自动给题目打难易度标签）

判题相关字段：

> 如果说题目不是很复杂，用例文件大小不大的话，可以直接存在数据库表里
> 但是如果用例文件比较大，> 512 KB 建议单独存放在一个文件中，数据库中只保存文件 url（类似存储用户头像）

- 输入用例：1、2
- 输出用例：3、4
- 时间限制
- 内存限制

judgeConfig 判题配置（json 对象）：

- 时间限制 timeLimit
- 内存限制 memoryLimit

judgeCase 判题用例（json 数组）

- 每一个元素是：一个输入用例对应一个输出用例

存 json 的好处：便于扩展，只需要改变对象内部的字段，而不用修改数据库表（可能会影响数据库）

存 json 的前提：

1. 你不需要根据某个字段去倒查这条数据
2. 你的字段含义相关，属于同一类的值
3. 你的字段存储空间占用不能太大

其他扩展字段：

- 通过率
- 判题类型

代码：

题目提交表

哪个用户提交了哪道题目，存放判题结果等

提交用户 id：userId

题目 id：questionId

语言：language

用户的代码：code

判题状态：status（0 - 待判题、1 - 判题中、2 - 成功、3 - 失败）

判题信息（判题过程中得到的一些信息，比如程序的失败原因、程序执行消耗的时间、空间）：

judgeInfo（json 对象）

判题信息枚举值：

- Accepted 成功
- Wrong Answer 答案错误
- Compile Error <http://poj.org/showcompileinfo?solution_id=24259830> 编译错误
- Memory Limit Exceeded 内存溢出
- Time Limit Exceeded 超时
- Presentation Error 展示错误
- Output Limit Exceeded 输出溢出
- Waiting 等待中
- Dangerous Operation 危险操作
- Runtime Error 运行错误（用户程序的问题）
- System Error 系统错误（做系统人的问题）

小知识 - 数据库索引

什么情况下适合加索引？如何选择给哪个字段加索引？

答：首先从业务出发，无论是单个索引、还是联合索引，都要从你实际的查询语句、字段枚举值的区分度、字段的类型考虑（where 条件指定的字段）

比如：where userId = 1 and questionId = 2

可以选择根据 userId 和 questionId 分别建立索引（需要分别根据这两个字段单独查询）；也可以选择给这两个字段建立联合索引（所查询的字段是绑定在一起的）。

原则上：能不用索引就不用索引；能用单个索引就别用联合 / 多个索引；不要给没区分度的字段加索引（比如性别，就男 / 女）。因为索引也是要占用空间的。

后端接口开发

后端开发流程

1）根据功能设计库表

2）自动生成对数据库基本的增删改查（mapper 和 service 层的基本功能）

3）编写 Controller 层，实现基本的增删改查和权限校验（复制粘贴）

4）去根据业务定制开发新的功能 / 编写新的代码

更好地方法，编写自己的代码生成器（https://github.com/liyupi/sql-father-frontend-public <https://github.com/liyupi/sql-father-frontend-public> ）

代码生成方法

1）安装 MyBatisX 插件

2）根据项目去调整生成配置，建议生成代码到独立的包，不要影响老的项目

3）把代码从生成包中移到实际项目对应目录中

4）找相似的代码去复制 Controller

- 单表去复制单表 Controller（比如 question => post）
- 关联表去复制关联表（比如 question_submit => post_thumb ）

5）复制实体类相关的 DTO、VO、枚举值字段（用于接受前端请求、或者业务间传递信息）

复制之后，调整需要的字段

updateRequest 和 editRequest 的区别：前者是给管理员更新用的，可以指定更多字段；后者是给普通用户试用的，只能指定部分字段。

6）为了更方便地处理 json 字段中的某个字段，需要给对应的 json 字段编写独立的类，比如 judgeConfig、judgeInfo、judgeCase。

示例代码：

小知识：什么情况下要加业务前缀？什么情况下不加？

加业务前缀的好处，防止多个表都有类似的类，产生冲突；不加的前提，因为可能这个类是多个业务之间共享的，能够复用的。

定义 VO 类：作用是专门给前端返回对象，可以节约网络传输大小、或者过滤字段（脱敏）、保证安全性。

比如 judgeCase、answer 字段，一定要删，不能直接给用户答案。

7）校验 Controller 层的代码，看看除了要调用的方法缺失外，还有无报错

8）实现 Service 层的代码，从对应的已经编写好的实现类复制粘贴，全局替换（比如 question => post）

9）编写 QuestionVO 的 json / 对象转换工具类

10）用同样的方法，编写 questionSubmit 提交类，这次参考 postThumb 相关文件

10）编写枚举类

参考代码：

编写好基本代码后，记得通过 Swagger 或者编写单元测试去验证。

小知识

为了防止用户按照 id 顺序爬取题目，建议把 id 的生成规则改为 ASSIGN_ID 而不是从 1 开始自增，示例代码如下：

查询提交信息接口

功能：能够根据用户 id、或者题目 id、编程语言、题目状态，去查询提交记录

注意事项：

仅本人和管理员能看见自己（提交 userId 和登录用户 id 不同）提交的代码

实现方案：先查询，再根据权限去脱敏

核心代码：

本期成果

后端题目、题目提交相关接口

本期作业

1. 熟悉快速增删改查的开发方法，可以试着自己完成题目表、题目提交表的编写
2. 可以根据自己之前写过的一些代码，完善通用的 Controller、Service 代码

第 3 期 - 前端页面开发

计划

以开发前端页面为主：

1）用户注册页面

2）创建题目页面（管理员）

3）题目管理页面（管理员）

- 查看（搜索）
- 删除
- 修改
- 快捷创建

4）题目列表页（用户）

5）题目详情页（在线做题页）

- 判题状态的查看

6）题目提交列表页

> 扩展：提交统计页、用户个人页

接入要用到的组件

先接入可能用到的组件，再去写页面，避免因为后续依赖冲突、整合组件失败带来的返工。

Markdown 编辑器

为什么用 Markdown?

一套通用的文本编辑语法，可以在各大网站上统一标准、渲染出统一的样式，比较简单易学。

推荐的 Md 编辑器：https://github.com/bytedance/bytemd <https://github.com/bytedance/bytemd>

阅读官方文档，下载编辑器主体、以及 gfm（表格支持）插件、highlight 代码高亮插件

新建 MdEditor 组件，编写代码：

隐藏编辑器中不需要的操作图标（比如 GitHub 图标）：

要把 MdEditor 当前输入的值暴露给父组件，便于父组件去使用，同时也是提高组件的通用性，需要定义属性，把 value 和 handleChange 事件交给父组件去管理：

MdEditor 示例代码：

代码编辑器

微软官方编辑器：https://github.com/microsoft/monaco-editor <https://github.com/microsoft/monaco-editor>

官方提供的整合教程：https://github.com/microsoft/monaco-editor/blob/main/docs/integrate-esm.md <https://github.com/microsoft/monaco-editor/blob/main/docs/integrate-esm.md>

1）安装编辑器

2）vue-cli 项目（webpack 项目）整合 monaco-editor。

先安装 monaco-editor-webpack-plugin（https://github.com/microsoft/monaco-editor/blob/main/webpack-plugin/README.md <https://github.com/microsoft/monaco-editor/blob/main/webpack-plugin/README.md> ）：

在 vue.config.js 中配置 webpack 插件：

全量加载：

按需加载：

如何使用 Monaco Editor？查看示例教程：

https://microsoft.github.io/monaco-editor/playground.html?source=v0.40.0#example-creating-the-editor-hello-world <https://microsoft.github.io/monaco-editor/playground.html?source=v0.40.0#example-creating-the-editor-hello-world>

整合教程参考：http://chart.zhenglinglu.cn/pages/2244bd/#%E5%9C%A8-vue-%E4%B8%AD%E4%BD%BF%E7%94%A8 <http://chart.zhenglinglu.cn/pages/2244bd/#%E5%9C%A8-vue-%E4%B8%AD%E4%BD%BF%E7%94%A8>

注意，monaco editor 在读写值的时候，要使用 toRaw(编辑器实例）的语法来执行操作，否则会卡死。

示例整合代码如下：

通 Md 编辑器一样，也要接受父组件的传值，把显示的输入交给父组件去控制，从而能够让父组件实时得到用户输入的代码：

> 项目扩展：用 diff editor 对比用户代码和标准答案的区别

页面开发

创建题目页面

重新根据后端生成前端请求代码：

注意 - 用户登录后仍显示未登录：

是因为代码生成后，OpenAPI 文件的 CREDENTIALS 参数重置了，应该改为 true。

示例代码：

需要用户输入的值：

小知识 - 自定义代码模板

在 JetBrains 系列编辑器中打开设置，搜索 live Templates，先创建一个自定义模板组，在组下创建代码模板。

效果：输入缩写，即可生成模板代码。

示例模板：

注意，其中的 ID 是根据表达式自动生成的：camelCase(fileNameWithoutExtension())

使用表单组件，先复制示例代码，再修改：https://arco.design/vue/component/form <https://arco.design/vue/component/form>

此处我们用到了

- 嵌套表单：https://arco.design/vue/component/form#nest <https://arco.design/vue/component/form#nest>
- 动态增减表单：https://arco.design/vue/component/form#dynamic <https://arco.design/vue/component/form#dynamic>

注意，我们自定义的代码编辑器组件不会被组件库识别，需要手动指定 value 和 handleChange 函数。

题目管理页面开发

1）使用表格组件：https://arco.design/vue/component/table#custom <https://arco.design/vue/component/table#custom> （需要找到自定义操作的示例）

2）查询数据

3）定义表格列

4）加载数据

5）调整格式

比如 json 格式不好看，有 2 种方法调整：

1. 使用组件库自带的语法，自动格式化（更方便）
2. 完全自定义渲染，想展示什么就展示什么（更灵活）

6）添加删除、更新操作

删除后要执行 loadData 刷新数据

> 如果想学习 React And Design 框架的管理页面开发，去看用户中心项目

更新页面开发

策略：由于更新和创建都是相同的表单，所以完全没必要开发 / 复制 2 遍，可以直接复用创建页面。

关键实现：如何区分两个页面？

1. 路由（/add/question 和 /update/question）
2. 请求参数（id = 1）

更新页面相比于创建页面，多了 2 个改动：

1）在加载页面时，更新页面需要加载出之前的数据

2）在提交时，请求的地址不同

本期成果

引入了文本编辑器、代码编辑器，完成题目创建页面、题目更新页面、题目管理页面。

本期作业

1. 完成前端页面的开发
2. 引入文档编辑器、代码编辑器，并且通过阅读官方文档来自定义编辑器的样式或功能

第 4 期 - 判题机架构

前端页面开发

当前代码优化

1）先处理菜单项的权限控制和显示隐藏

通过 meta.hideInMenu 和 meta.access 属性控制

2）管理页面分页问题的修复

可以参考聚合搜索项目的搜索条件改变和 url 状态同步

核心原理：在分页页号改变时，触发 @page-change 事件，通过改变 searchParams 的值，并且通过 watchEffect 监听 searchParams 的改变（然后执行 loadData 重新加载速度），实现了页号变化时触发数据的重新加载。

3）修复刷新页面未登录问题

修改 access\index.ts 中的获取登录用户信息，把登录后的信息更新到 loginUser 变量上

题目列表搜索页

核心实现：表格组件

1）复制管理题目页的表格

2）只保留需要的 columns 字段

3）自定义表格列的渲染

标签：使用 tag 组件

通过率：自行计算

创建时间：使用 moment 库进行格式化（https://momentjs.com/docs/#/displaying/format/ <https://momentjs.com/docs/#/displaying/format/> ）

操作按钮：补充跳转到做题页的按钮

4）编写搜索表单，使用 form 的 layout = inline 布局，让用户的输入和 searchParams 同步，并且给提交按钮绑定修改 searchParams，从而被 watchEffect 监听到，触发查询

题目浏览页

1）先定义动态参数路由，开启 props 为 true，可以在页面的 props 中直接获取到动态参数（题目 id）

2）定义布局：左侧是题目信息，右侧是代码编辑器

3）左侧题目信息：

- tabs 切换展示的内容
- 定义 MdViewer 组件展示题目内容
- 使用descriptions 组件展示判题配置 https://arco.design/vue/component/descriptions <https://arco.design/vue/component/descriptions>

4）使用 select 组件让用户选择编程语言

在代码编辑器中监听属性的变化，注意监听 props 要使用箭头函数

https://blog.csdn.net/wuyxinu/article/details/124477647 <https://blog.csdn.net/wuyxinu/article/details/124477647>

> todo：代码编辑器没有更改语言

判题机模块架构

> 目的：先跑通完整的业务流程，再进行代码沙箱复杂的实现

判题模块和代码沙箱的关系

判题模块：调用代码沙箱，把代码和输入交给代码沙箱去执行

代码沙箱：只负责接受代码和输入，返回编译运行的结果，不负责判题（可以作为独立的项目 / 服务，提供给其他的需要执行代码的项目去使用）

这两个模块完全解耦：

思考：为什么代码沙箱要接受和输出一组运行用例？

前提：我们的每道题目有多组测试用例

如果是每个用例单独调用一次代码沙箱，会调用多次接口、需要多次网络传输、程序要多次编译、记录程序的执行状态（重复的代码不重复编译）

> 这是一种很常见的性能优化方法！（批处理）

代码沙箱架构开发

1）定义代码沙箱的接口，提高通用性

之后我们的项目代码只调用接口，不调用具体的实现类，这样在你使用其他的代码沙箱实现类时，就不用去修改名称了， 便于扩展。

> 代码沙箱的请求接口中，timeLimit 可加可不加，可自行扩展，即时中断程序

扩展思路：增加一个查看代码沙箱状态的接口

2）定义多种不同的代码沙箱实现

示例代码沙箱：仅为了跑通业务流程

远程代码沙箱：实际调用接口的沙箱

第三方代码沙箱：调用网上现成的代码沙箱，https://github.com/criyle/go-judge <https://github.com/criyle/go-judge>

3）编写单元测试，验证单个代码沙箱的执行

但是现在的问题是，我们把 new 某个沙箱的代码写死了，如果后面项目要改用其他沙箱，可能要改很多地方的代码。

4）使用工厂模式，根据用户传入的字符串参数（沙箱类别），来生成对应的代码沙箱实现类

此处使用静态工厂模式，实现比较简单，符合我们的需求。

> 扩展思路：如果确定代码沙箱示例不会出现线程安全问题、可复用，那么可以使用单例工厂模式
> 聚合搜索项目中有讲过

由此，我们可以根据字符串动态生成实例，提高了通用性：

5）参数配置化，把项目中的一些可以交给用户去自定义的选项或字符串，写到配置文件中。这样开发者只需要改配置文件，而不需要去看你的项目代码，就能够自定义使用你项目的更多功能。

application.yml 配置文件中指定变量：

在 Spring 的 Bean 中通过 @Value 注解读取：

6）代码沙箱能力增强

比如：我们需要在调用代码沙箱前，输出请求参数日志；在代码沙箱调用后，输出响应结果日志，便于管理员去分析。

每个代码沙箱类都写一遍 log.info？难道每次调用代码沙箱前后都执行 log？

使用代理模式，提供一个 Proxy，来增强代码沙箱的能力（代理模式的作用就是增强能力）

原本：需要用户自己去调用多次

使用代理后：不仅不用改变原本的代码沙箱实现类，而且对调用者来说，调用方式几乎没有改变，也不需要在每个调用沙箱的地方去写统计代码。

代理模式的实现原理：

1. 实现被代理的接口

2．通过构造函数接受一个被代理的接口实现类

3．调用被代理的接口实现类，在调用前后增加对应的操作

CodeSandboxProxy 示例代码：

使用方式：

7）实现示例的代码沙箱

小知识 - Lombok Builder 注解

以前我们是使用 new 对象后，再逐行执行 set 方法的方式来给对象赋值的。

还有另外一种可能更方便的方式 builder。

1）实体类加上 @Builder 等注解：

2）可以使用链式的方式更方便地给对象赋值：

判题服务开发

定义单独的 judgeService 类，而不是把所有判题相关的代码写到 questionSubmitService 里，有利于后续的模块抽离、微服务改造。

判题服务业务流程

1）传入题目的提交 id，获取到对应的题目、提交信息（包含代码、编程语言等）

2）如果题目提交状态不为等待中，就不用重复执行了

3）更改判题（题目提交）的状态为 “判题中”，防止重复执行，也能让用户即时看到状态

4）调用沙箱，获取到执行结果

5）根据沙箱的执行结果，设置题目的判题状态和信息

判断逻辑

1. 先判断沙箱执行的结果输出数量是否和预期输出数量相等
2. 依次判断每一项输出和预期输出是否相等
3. 判题题目的限制是否符合要求
4. 可能还有其他的异常情况

策略模式优化

我们的判题策略可能会有很多种，比如：我们的代码沙箱本身执行程序需要消耗时间，这个时间可能不同的编程语言是不同的，比如沙箱执行 Java 要额外花 10 秒。

我们可以采用策略模式，针对不同的情况，定义独立的策略，便于分别修改策略和维护。而不是把所有的判题逻辑、if ... else ... 代码全部混在一起写。

实现步骤如下：

1）定义判题策略接口，让代码更加通用化：

2）定义判题上下文对象，用于定义在策略中传递的参数（可以理解为一种 DTO）：

3）实现默认判题策略，先把 judgeService 中的代码搬运过来

4）再新增一种判题策略，通过 if ... else ... 的方式选择使用哪种策略：

但是，如果选择某种判题策略的过程比较复杂，如果都写在调用判题服务的代码中，代码会越来越复杂，会有大量 if ... else ...，所以建议单独编写一个判断策略的类。

5）定义 JudgeManager，目的是尽量简化对判题功能的调用，让调用方写最少的代码、调用最简单。对于判题策略的选取，也是在 JudgeManager 里处理的。

示例代码如下：

本期成果

完成前端题目列表页、在线做题页面开发：

完成后端代码沙箱的架构和判题服务的开发。

本期作业

1. 完成前端题目列表页、在线做题页面开发
2. 理解代码沙箱架构的实现，掌握工厂模式、策略模式、代理模式
3. 实现判题服务的完整流程

第 5 期 - 代码沙箱原生实现

历史问题修复

代码编辑器切换语言失败问题

解决方案：监听 language 属性，动态更改编辑器的语言

代码如下：

代码沙箱项目初始化

代码沙箱的定位：只负责接受代码和输入，返回编译运行的结果，不负责判题（可以作为独立的项目 / 服务，提供给其他的需要执行代码的项目去使用）

以 Java 编程语言为主，带大家实现代码沙箱，重要的是 **学思想、学关键流程** 。

> 扩展：可以自行实现 C++ 语言的代码沙箱

由于代码沙箱是能够通过 API 调用的独立服务，所以新建一个 Spring Boot Web 项目。最终这个项目要提供一个能够执行代码、操作代码沙箱的接口。

使用 IDEA 的 Spring Boot 项目初始化工具，一定要选择 Java 8、Spring Boot 2.7 版

本！！！

编写启动配置：

编写测试接口，验证能否访问：

将 yuoj-backend 项目的 JudgeInfo 类移到 model 目录下，然后复制 model 包和 CodeSandbox 接口到该沙箱项目，便于字段的统一。

Java 原生实现代码沙箱

原生：尽可能不借助第三方库和依赖，用最干净、最原始的方式实现代码沙箱

通过命令行执行

Java 程序执行流程

接收代码 => 编译代码（javac） => 执行代码（java）

先编写示例代码，注意要去掉包名，放到 resources 目录下：

用 javac 命令编译代码：

用 java 命令执行代码：

程序中文乱码问题

为什么编译后的 class 文件出现中文乱码呢?

原因：命令行终端的编码是 GBK，和 java 代码文件本身的编码 UTF-8 不一致，导致乱码。

通过 `chcp` 命令查看命令行编码，GBK 是 936，UTF-8 是 65001。

但是 **不建议** 大家改变终端编码来解决编译乱码，因为其他运行你代码的人也要改变环境，兼

容性很差。

推荐的 javac 编译命令，用 `-encoding utf-8` 参数解决：

java 执行：

统一类名

实际的 OJ 系统中，对用户输入的代码会有一定的要求。便于系统进行统一处理和判题。

此处我们把用户输入代码的类名限制为 Main（参考 Poj），可以减少编译时类名不一致的风险；而且不用从用户代码中提取类名，更方便。

文件名 Main.java，示例代码如下：

实际执行命令时，可以统一使用 Main 类名：

核心流程实现

核心实现思路：用程序代替人工，用程序来操作命令行，去编译执行代码

核心依赖：Java 进程类 Process

1. 把用户的代码保存为文件
2. 编译代码，得到 class 文件
3. 执行代码，得到输出结果
4. 收集整理输出结果
5. 文件清理，释放空间
6. 错误处理，提升程序健壮性

1、保存代码文件

引入 Hutool 工具类，提高操作文件效率：

新建目录，将每个用户的代码都存放在独立目录下，通过 UUID 随机生成目录名，便于隔离和

维护：

2、编译代码

使用 Process 类在终端执行命令：

执行 process.waitFor 等待程序执行完成，并通过返回的 exitValue 判断程序是否正常返回，然后从 Process 的输入流 inputStream 和错误流 errorStream 获取控制台输出。

示例代码如下：

可以把上述代码提取为工具类 ProcessUtils，执行进程并获取输出，并且使用 StringBuilder 拼接控制台输出信息：

3、执行程序

同样是使用 Process 类运行 java 命令，命令中记得增加 `-Dfile.encoding=UTF-8` 参数，解决中文乱码：

上述命令适用于执行从输入参数（args）中获取值的代码。

很多 OJ 都是 ACM 模式，需要和用户交互，让用户不断输入内容并获取输出，比如：

对于此类程序，我们需要使用 OutputStream 向程序终端发送参数，并及时获取结果，注意最后要关闭流释放资源。

示例代码如下：

4、整理输出

1）通过 for 循环遍历执行结果，从中获取输出列表

2）获取程序执行时间

可以使用 Spring 的 StopWatch 获取一段程序的执行时间：

此处我们使用最大值来统计时间，便于后续判题服务计算程序是否超时：

> 扩展：可以每个测试用例都有一个独立的内存、时间占用的统计

3）获取内存信息

实现比较复杂，因为无法从 Process 对象中获取到子进程号，也不推荐在 Java 原生实现代码沙箱的过程中获取。

5、文件清理

防止服务器空间不足，删除代码目录：

6、错误处理

封装一个错误处理方法，当程序抛出异常时，直接返回错误响应。

示例代码如下：

Java 程序异常情况

到目前为止，核心流程已经实现，但是想要上线的话，安全么？

用户提交恶意代码，怎么办？

1、执行超时

占用时间资源，导致程序卡死，不释放资源：

要把写好的代码复制到 resources 中，并且一定要把类名改为 Main！包名一定要去掉！

2、占用内存

占用内存资源，导致空间浪费：

实际运行上述程序时，我们会发现，内存占用到达一定空间后，程序就自动报错：`java.lang.OutOfMemoryError: Java heap space`，而不是无限增加内存占用，直到系统死机。

这是 JVM 的一个保护机制。

可以使用 JVisualVM 或 JConsole 工具，连接到 JVM 虚拟机上来可视化查看运行状态。

如图：

3、读文件，信息泄露

比如直接通过相对路径获取项目配置文件，获取到密码：

4、写文件，植入木马

可以直接向服务器上写入文件，比如一个木马程序：`java -version 2>&1`（示例命令）

> 1. java -version 用于显示 Java 版本信息。这会将版本信息输出到标准错误流（stderr）而不是标准输出流（stdout）。
> 2. 2>&1 将标准错误流重定向到标准输出流。这样，Java 版本信息就会被发送到标准输出流。

代码如下：

5、运行其他程序

直接通过 Process 执行危险程序，或者电脑上的其他程序：

6、执行高危操作

甚至都不用写木马文件，直接执行系统自带的危险命令！

- 比如删除服务器的所有文件（太残暴、不演示）
- 比如执行 dir（windows）、ls（linux） 获取你系统上的所有文件信息

Java 程序安全控制

针对上面的异常情况，分别有如下方案，可以提高程序安全性。

1. 超时控制
2. 限制给用户程序分配的资源
3. 限制代码 - 黑白名单
4. 限制用户的操作权限
5. 运行环境隔离

1、超时控制

通过创建一个守护线程，超时后自动中断 Process 实现。

代码如下：

2、限制资源分配

我们不能让每个 java 进程的执行占用的 JVM 最大堆内存空间都和系统默认的一致（鱼皮的 JVM 默认最大占用 8G 内存），实际上应该更小（执行用户的题目代码也不需要这么多），比如说 256MB。

在启动 Java 程序时，可以指定 JVM 的参数：-Xmx256m（最大堆空间大小）

示例命令如下：

注意！-Xmx 参数、JVM 的堆内存限制，不等同于系统实际占用的最大资源，可能会超出。

如果需要更严格的内存限制，要在系统层面去限制，而不是 JVM 层面的限制。

如果是 Linux 系统，可以使用 cgroup 来实现对某个进程的 CPU、内存等资源的分配。

小知识 - 什么是 cgroup?

`cgroup` 是 Linux 内核提供的一种机制，可以用来限制进程组（包括子进程）的资源使用，例如内存、CPU、磁盘 I/O 等。通过将 Java 进程放置在特定的 `cgroup` 中，你可以实现限制其使用的内存和 CPU 数。

小知识 - 常用 JVM 启动参数

1．内存相关参数：

○ -Xms: 设置 JVM 的初始堆内存大小。

○ -Xmx: 设置 JVM 的最大堆内存大小。

○ -Xss: 设置线程的栈大小。

○ -XX:MaxMetaspaceSize: 设置 Metaspace（元空间）的最大大小。

○ -XX:MaxDirectMemorySize: 设置直接内存（Direct Memory）的最大大小。

2. 垃圾回收相关参数：

○ -XX:+UseSerialGC: 使用串行垃圾回收器。

○ -XX:+UseParallelGC: 使用并行垃圾回收器。

○ -XX:+UseConcMarkSweepGC: 使用 CMS 垃圾回收器。

○ -XX:+UseG1GC: 使用 G1 垃圾回收器。

3. 线程相关参数：

○ -XX:ParallelGCThreads: 设置并行垃圾回收的线程数。

○ -XX:ConcGCThreads: 设置并发垃圾回收的线程数。

○ -XX:ThreadStackSize: 设置线程的栈大小。

4. JIT 编译器相关参数：

○ -XX:TieredStopAtLevel: 设置 JIT 编译器停止编译的层次。

5. 其他资源限制参数：

○ -XX:MaxRAM: 设置 JVM 使用的最大内存。

3、限制代码 - 黑白名单

实现

先定义一个黑白名单，比如哪些操作是禁止的，可以就是一个列表：

还可以使用字典树代替列表存储单词，用 **更少的空间** 存储更多的敏感词汇，并且实现 **更高效** 的敏感词查找。

字典树的原理：

> 字典树相关的应用可以写在简历上

此处使用 HuTool 工具库的字典树工具类：WordTree，不用自己写字典树！

1）先初始化字典树，插入禁用词：

2）校验用户代码是否包含禁用词：

本方案缺点

1）你无法遍历所有的黑名单

2）不同的编程语言，你对应的领域、关键词都不一样，限制人工成本很大

4、限制权限 - Java 安全管理器

目标：限制用户对文件、内存、CPU、网络等资源的操作和访问。

Java 安全管理器使用

Java 安全管理器（Security Manager）是 Java 提供的保护 JVM、Java 安全的机制，可以实现更严格的资源和操作限制。

编写安全管理器，只需要继承 Security Manager。

1）所有权限放开：

2）所有权限拒绝：

3）限制读权限：

4）限制写文件权限：

5）限制执行文件权限：

6）限制网络连接权限：

结合项目运用

实际情况下，不应该在主类（开发者自己写的程序）中做限制，只需要限制子程序的权限即可。

启动子进程执行命令时，设置安全管理器，而不是在外层设置（会限制住测试用例的读写和子命令的执行）。

具体操作如下：

1）根据需要开发自定义的安全管理器（比如 MySecurityManager）

2）复制 MySecurityManager 类到 `resources/security` 目录下， **移除类的包名**

3）手动输入命令编译 MySecurityManager 类，得到 class 文件

4）在运行 java 程序时，指定安全管理器 class 文件的路径、安全管理器的名称。

命令如下：

> 注意，windows 下要用分号间隔多个类路径！

依次执行之前的所有测试用例，发现资源成功被限制。

安全管理器优点

1. 权限控制很灵活
2. 实现简单

安全管理器缺点

1. 如果要做比较严格的权限限制，需要自己去判断哪些文件、包名需要允许读写。粒度太细了，难以精细化控制。
2. 安全管理器本身也是 Java 代码，也有可能存在漏洞。本质上还是程序层面的限制，没深入系统的层面。

5、运行环境隔离

原理：操作系统层面上，把用户程序封装到沙箱里，和宿主机（我们的电脑 / 服务器）隔离开，使得用户的程序无法影响宿主机。

实现方式：Docker 容器技术（底层是用 cgroup、namespace 等方式实现的），也可以直接使用 cgroup 实现。

本期成果

实现了 Java 原生代码沙箱：http://gitlab.code-nav.cn/root/yuoj-code-sandbox <http://gitlab.code-nav.cn/root/yuoj-code-sandbox>

本期作业

1. 理解 Java 的编译执行过程，能通过程序实现
2. 理解 Java 程序常见的安全漏洞和解决方案
3. 实践 Java 安全管理器
4. 提前安装虚拟机和远程开发环境：https://www.bilibili.com/video/BV1h94y1k7Jf <https://www.bilibili.com/video/BV1h94y1k7Jf>

第 6 期 - 代码沙箱 Docker 实现

大纲

1. 讲解 Docker 的概念
2. Docker 的基本用法
    a. 命令行的用法
    b. Java 操作 Docker
3. 使用 Docker 实现代码沙箱
4. 怎么提升 Docker 沙箱的安全性

Docker 容器技术

为什么要用 Docker 容器技术?

为了进一步提升系统的安全性，把不同的程序和宿主机进行隔离，使得某个程序（应用）的执行不会影响到系统本身。

Docker 技术可以实现程序和宿主机的隔离。

什么是容器?

理解为对一系列应用程序、服务和环境的封装，从而把程序运行在一个隔离的、密闭的、隐私的空间内，对外整体提供服务。

可以把一个容器理解为一个新的电脑（定制化的操作系统）。

Docker 基本概念

镜像：用来创建容器的安装包，可以理解为给电脑安装操作系统的系统镜像

容器：通过镜像来创建的一套运行环境，一个容器里可以运行多个程序，可以理解为一个电脑实例

Dockerfile：制作镜像的文件，可以理解为制作镜像的一个清单

镜像仓库：存放镜像的仓库，用户可以从仓库下载现成的镜像，也可以把做好的镜像放到仓库里

推荐使用 docker 官方的镜像仓库：https://hub.docker.com/search?q=nginx <https://hub.docker.com/search?q=nginx>

Docker 实现核心

> 对应题目：Docker 能实现哪些资源的隔离?

看图理解：

1）Docker 运行在 Linux 内核上

2）CGroups：实现了容器的资源隔离，底层是 Linux Cgroup 命令，能够控制进程使用的资源

3）Network 网络：实现容器的网络隔离，docker 容器内部的网络互不影响

4）Namespaces 命名空间：可以把进程隔离在不同的命名空间下，每个容器他都可以有自己的命名空间，不同的命名空间下的进程互不影响。

5）Storage 存储空间：容器内的文件是相互隔离的，也可以去使用宿主机的文件

docker compose：是一种同时启动多个容器的集群操作工具（容器管理工具），一般情况下，开发者仅做了解即可，实际使用 docker compose 时去百度配置文件

安装 Docker

一般情况下，不建议在 Windows 系统上安装。

Windows 本身就自带了一个虚拟机叫 WSL，但是我不推荐，肯定不如一个专业的、隔离的虚拟机软件要方便。

此处使用免费的 VMWare Workstation Player 软件：https://www.vmware.com/cn/products/workstation-player.html <https://www.vmware.com/cn/products/workstation-player.html>

请先完整观看教程，安装 Linux 虚拟机、Docker 环境和远程开发环境：https://www.bilibili.com/video/BV1h94y1k7Jf/ <https://www.bilibili.com/video/BV1h94y1k7Jf/> （Ubuntu 系统）

命令行操作 Docker

1）查看命令用法

查看具体子命令的用法：

2）从远程仓库拉取现成的镜像

用法：

示例：

3）根据镜像创建容器实例：

启动实例，得到容器实例 containerId：

4）查看容器状态：

5）启动容器：

启动示例：

6）查看日志：

启动示例：

7）删除容器实例：

删除示例：

8）删除镜像：

示例，强制删除：

9）其他：构建镜像（build）、推送镜像（push）、运行容器（run）、执行容器命令（exec）等

Java 操作 Docker

前置准备

使用 Docker-Java：https://github.com/docker-java/docker-java <https://github.com/docker-java/docker-java>

官方入门：https://github.com/docker-java/docker-java/blob/main/docs/getting_started.md <https://github.com/docker-java/docker-java/blob/main/docs/getting_started.md>

先引入依赖：

DockerClientConfig：用于定义初始化 DockerClient 的配置（类比 MySQL 的连接、线程数配置）

DockerHttpClient：用于向 Docker 守护进程（操作 Docker 的接口）发送请求的客户端，低层封装（不推荐使用），你要自己构建请求参数（简单地理解成 JDBC）

DockerClient（推荐）：才是真正和 Docker 守护进程交互的、最方便的 SDK，高层封装，对 DockerHttpClient 再进行了一层封装（理解成 MyBatis），提供了现成的增删改查

远程开发

使用 IDEA Development 先上传代码到 Linux，然后使用 JetBrains 远程开发完全连接 Linux 实时开发。

如果无法启动程序，修改 settings 的compiler 配置：`-Djdk.lang.Process.launchMechanism=vfork`

如果启动失败：

1）增加权限

2）重启虚拟机！重启远程开发环境！重启程序！

> 鱼皮直播时就是虚拟机卡了，重启后就好了

常用操作

1）拉取镜像：

2）创建容器：

3）查看容器状态：

4）启动容器：

5）查看日志：

6）删除容器：

7）删除镜像：

Docker 实现代码沙箱

实现思路：docker 负责运行 java 程序，并且得到结果。

流程几乎和 Java 原生实现流程相同：

1. 把用户的代码保存为文件
2. 编译代码，得到 class 文件
3. 把编译好的文件上传到容器环境内
4. 在容器中执行代码，得到输出结果
5. 收集整理输出结果
6. 文件清理，释放空间
7. 错误处理，提升程序健壮性

> 扩展：模板方法设计模式，定义同一套实现流程，让不同的子类去负责不同流程中的具体实现。执行步骤一样，每个步骤的实现方式不一样。

创建容器，上传编译文件

自定义容器的两种方式：

1）在已有镜像的基础上再扩充：比如拉取现成的 Java 环境（包含 jdk），再把编译后的文件复制到容器里。适合新项目、跑通流程

2）完全自定义容器：适合比较成熟的项目，比如封装多个语言的环境和实现

思考：我们每个测试用例都单独创建一个容器，每个容器只执行一次 java 命令？

浪费性能，所以要创建一个 **可交互** 的容器，能接受多次输入并且输出。

创建容器时，可以指定文件路径（Volumn） 映射，作用是把本地的文件同步到容器中，可以让容器访问。

> 也可以叫容器挂载目录

启动容器，执行代码

执行代码

Docker 执行容器命令（操作已启动容器）：

示例执行：

注意，要把命令按照空格拆分，作为一个数组传递，否则可能会被识别为一个字符串，而不是多个参数。

创建命令：

执行命令，通过回调接口来获取程序的输出结果，并且通过 StreamType 来区分标准输出和错误输出。

示例代码如下：

尽量复用之前的 `ExecuteMessage` 对象，在异步接口中填充正常和异常信息，这样之后流程的代码都可以复用。

获取程序执行时间

和 Java 原生一样，使用 StopWatch 在执行前后统计时间。

获取程序占用内存

程序占用的内存每个时刻都在变化，所以你不可能获取到所有时间点的内存。

我们要做的是，定义一个周期，定期地获取程序的内存。

Docker-Java 提供了内存定期统计的操作，示例代码如下：

注意，程序执行完后要关闭统计命令：

Docker 容器安全性

超时控制

执行容器时，可以增加超时参数控制值：

但是，这种方式无论超时与否，都会往下执行，无法判断是否超时。

解决方案：可以定义一个标志，如果程序执行完成，把超时标志设置为 false。

示例代码如下：

内存资源

通过 HostConfig 的 withMemory 等方法，设置容器的最大内存和资源限制：

网络资源

创建容器时，设置网络配置为关闭：

权限管理

Docker 容器已经做了系统层面的隔离，比较安全，但不能保证绝对安全。

1）结合 Java 安全管理器和其他策略去使用

2）限制用户不能向 root 根目录写文件：

3）Linux 自带的一些安全管理措施，比如 seccomp（Secure Computing Mode）是一个用于 Linux 内核的安全功能，它允许你限制进程可以执行的系统调用，从而减少潜在的攻击面和提高容器的安全性。通过配置 seccomp，你可以控制容器内进程可以使用的系统调用类型和参数。

示例 seccomp 配置文件 profile.json：

在 hostConfig 中开启安全机制：

本期成果

1. 学习 Java 操作 Docker 的方法
2. 用 Docker 实现了安全的代码沙箱

本期作业

1. 搭建 Linux + Docker 远程开发环境，掌握远程开发
2. 实践使用命令行和 Java 操作 Docker
3. 用 Docker 实现代码沙箱

第 7 期

计划

1、模板方法优化代码沙箱

2、代码沙箱提供 API

3、跑通完整的单机项目流程

- 补充前端提交列表页面

4、单机项目改造成微服务

5、把项目的模块调用改为消息队列的解耦

模板方法优化代码沙箱

模板方法：定义一套通用的执行流程，让子类负责每个执行步骤的具体实现

模板方法的适用场景：适用于有规范的流程，且执行流程可以复用

作用：大幅节省重复代码量，便于项目扩展、更好维护

1、抽象出具体的流程

定义一个模板方法抽象类。

先复制具体的实现类，把代码从完整的方法抽离成一个一个子写法

2、定义子类的具体实现

Java 原生代码沙箱实现，直接复用模板方法定义好的方法实现：

Docker 代码沙箱实现，需要自行重写 RunFile：

给代码沙箱提供开放 API

直接在 controller 暴露 CodeSandbox 定义的接口：

调用安全性

如果将服务不做任何的权限校验，直接发到公网，是不安全的。

1）调用方与服务提供方之间约定一个字符串 **（最好加密）**

优点：实现最简单，比较适合内部系统之间相互调用（相对可信的环境内部调用）

缺点：不够灵活，如果 key 泄露或变更，需要重启代码

代码沙箱服务，先定义约定的字符串：

改造请求，从请求头中获取认证信息，并校验：

调用方，在调用时补充请求头：

2）API 签名认证

给允许调用的人员分配 accessKey、secretKey，然后校验这两组 key 是否匹配

详细请见 API 开放平台项目。

跑通整个项目流程

1）移动 questionSubmitController 代码到 questionController 中

2）由于后端改了接口地址，前端需要重新生成接口调用代码

还需要更改前端调用的 Controller

3）后端调试

4）开发提交列表页面

> 扩展：每隔一段时间刷新一下提交状态，因为后端是异步判题的

单体项目改造为微服务

新建一个项目

什么是微服务?

服务：提供某类功能的代码

微服务：专注于提供某类特定功能的代码，而不是把所有的代码全部放到同一个项目里。会把整个大的项目按照一定的功能、逻辑进行拆分，拆分为多个子模块，每个子模块可以独立运行、独立负责一类功能，子模块之间相互调用、互不影响。

一个公司：一个人干活，这个人 icu 了，公司直接倒闭

一个公司有多个不同类的岗位，多个人干活，一个组跨了还有其他组可以正常工作，不会说公司直接倒闭。各组之间可能需要交互，来完成大的目标。

微服务的几个重要的实现因素：服务管理、服务调用、服务拆分

微服务实现技术?

Spring Cloud

Spring Cloud Alibaba（本项目采用）

Dubbo（DubboX）

RPC（GRPC、TRPC）

本质上是通过 HTTP、或者其他的网络协议进行通讯来实现的。

Spring Cloud Alibaba

https://github.com/alibaba/spring-cloud-alibaba <https://github.com/alibaba/spring-cloud-alibaba>

推荐参考中文文档来学习：https://sca.aliyun.com/zh-cn/ <https://sca.aliyun.com/zh-cn/>

本质：是在 Spring Cloud 的基础上，进行了增强，补充了一些额外的能力，根据阿里多年的业务沉淀做了一些定制化的开发

1. Spring Cloud Gateway：网关
2. Nacos：服务注册和配置中心
3. Sentinel：熔断限流
4. Seata：分布式事务
5. RocketMQ：消息队列，削峰填谷
6. Docker：使用Docker进行容器化部署
7. Kubernetes：使用k8s进行容器化部署

注意，一定要选择对应的版本：https://sca.aliyun.com/zh-cn/docs/2021.0.5.0/overview/version-explain <https://sca.aliyun.com/zh-cn/docs/2021.0.5.0/overview/version-explain>

此处选择 2021.0.5.0：

Nacos：集中存管项目中所有服务的信息，便于服务之间找到彼此；同时，还支持集中存储整个项目中的配置。

整个微服务请求流程：

> 扩展：感兴趣可以了解另一个分布式微服务框架 https://github.com/Nepxion/Discovery <https://github.com/Nepxion/Discovery>

改造前思考

从业务需求出发，思考单机和分布式的区别。

用户登录功能：需要改造为分布式登录

其他内容：

- 有没有用到单机的锁？改造为分布式锁（伙伴匹配系统讲过）
- 有么有用到本地缓存？改造为分布式缓存（Redis）
- 需不需要用到分布式事务？比如操作多个库

改造分布式登录

1）application.yml 增加 redis 配置

2）补充依赖：

```
```

3）主类取消 Redis 自动配置的移除

4）修改 session 存储方式：

```
```

5）使用 redis-cli 或者 redis 管理工具，查看是否有登录后的信息

微服务的划分

从业务出发，想一下哪些功能 / 职责是一起的？

> 公司老板给员工分工

依赖服务：

- 注册中心：Nacos
- 微服务网关（yuoj-backend-gateway）：Gateway 聚合所有的接口，统一接受处理前端的请求

公共模块：

- common 公共模块（yuoj-backend-common）：全局异常处理器、请求响应封装类、公用的工具类等
- model 模型模块（yuoj-backend-model）：很多服务公用的实体类
- 公用接口模块（yuoj-backend-service-client）：只存放接口，不存放实现（多个服务之间要共享）

业务功能：

1. 用户服务（yuoj-backend-user-service：8102 端口）：
    a. 注册（后端已实现）
    b. 登录（后端已实现，前端已实现）
    c. 用户管理
2. 题目服务（yuoj-backend-question-service：8103）
    a. 创建题目（管理员）
    b. 删除题目（管理员）
    c. 修改题目（管理员）
    d. 搜索题目（用户）
    e. 在线做题（题目详情页）
    f. 题目提交
3. 判题服务（yuoj-backend-judge-service，8104 端口，较重的操作）
    a. 执行判题逻辑
    b. 错误处理（内存溢出、安全性、超时）
    c. **自主实现** 代码沙箱（安全沙箱）
    d. 开放接口（提供一个独立的新服务）

> 代码沙箱服务本身就是独立的，不用纳入 Spring Cloud 的管理

路由划分

用 springboot 的 context-path 统一修改各项目的接口前缀，比如：

用户服务：

- /api/user
- /api/user/inner（内部调用，网关层面要做限制）

题目服务：

- /api/question（也包括题目提交信息）
- /api/question/inner（内部调用，网关层面要做限制）

判题服务：

- /api/judge
- /api/judge/inner（内部调用，网关层面要做限制）

Nacos 注册中心启动

一定要选择 2.2.0 版本！！！

教程：https://sca.aliyun.com/zh-cn/docs/2021.0.5.0/user-guide/nacos/overview <https://sca.aliyun.com/zh-cn/docs/2021.0.5.0/user-guide/nacos/overview>

Nacos 官网教程：https://nacos.io/zh-cn/docs/quick-start.html <https://nacos.io/zh-cn/docs/quick-start.html>

到官网下载 Nacos：https://github.com/alibaba/nacos/releases/tag/2.2.0 <https://github.com/alibaba/nacos/releases/tag/2.2.0>

安装好后，进入 bin 目录启动：

新建工程

Spring Cloud 有相当多的依赖，参差不齐，不建议大家随意找一套配置、或者自己写。

建议用脚手架创建项目：https://start.aliyun.com/ <https://start.aliyun.com/>

给项目增加全局依赖配置文件。

创建完初始项目后，补充 Spring Cloud 依赖：

依次使用 new modules 和 spring boot Initializr 创建各模块：

需要给各模块之间绑定子父依赖关系，效果如下：

父模块定义 modules，子模块引入 parent 语法，可以通过继承父模块配置，统一项目的定义和版本号。

同步代码和依赖

1）common 公共模块（yuoj-backend-common）：全局异常处理器、请求响应封装类、公用的工具类等

在外层的 pom.xml 中引入公共类：

2）model 模型模块（yuoj-backend-model）：很多服务公用的实体类

直接复制 model 包，注意代码沙箱 model 的引入

3）公用接口模块（yuoj-backend-service-client）：只存放接口，不存放实现（多个服务之间要共享）

先无脑搬运所有的 service，judgeService 也需要搬运

需要指定 openfeign（客户端调用工具）的版本：

4）具体业务服务实现

给所有业务服务引入公共依赖：

主类引入注解

引入 application.yml 配置

服务内部调用

现在的问题是，题目服务依赖用户服务，但是代码已经分到不同的包，找不到对应的 Bean。

可以使用 Open Feign 组件实现跨服务的远程调用。

Open Feign：Http 调用客户端，提供了更方便的方式来让你远程调用其他服务，不用关心服务的调用地址

Nacos 注册中心获取服务调用地址

1）梳理服务的调用关系，确定哪些服务（接口）需要给内部调用

用户服务：没有其他的依赖

题目服务：

userService.getById(userId)

userService.getUserVO(user)

userService.listByIds(userIdSet)

userService.isAdmin(loginUser)

userService.getLoginUser(request)

judgeService.doJudge(questionSubmitId)

判题服务：

questionService.getById(questionId)

questionSubmitService.getById(questionSubmitId)

questionSubmitService.updateById(questionSubmitUpdate)

2）确认要提供哪些服务

用户服务：没有其他的依赖

userService.getById(userId)

userService.getUserVO(user)

userService.listByIds(userIdSet)

userService.isAdmin(loginUser)

userService.getLoginUser(request)

题目服务：

questionService.getById(questionId)

questionSubmitService.getById(questionSubmitId)

questionSubmitService.updateById(questionSubmitUpdate)

判题服务：

judgeService.doJudge(questionSubmitId)

3）实现 client 接口

对于用户服务，有一些不利于远程调用参数传递、或者实现起来非常简单（工具类），可以直接用默认方法，无需远程调用，节约性能

开启 openfeign 的支持，把我们的接口暴露出去（服务注册到注册中心上），作为 API 给其他服务调用（其他服务从注册中心寻找）

需要修改每个服务提供者的 context-path 全局请求路径

服务提供者：理解为接口的实现类，实际提供服务的模块（服务注册到注册中心上）

服务消费者：理解为接口的调用方，需要去找到服务提供者，然后调用。（其他服务从注册中心寻找）

注意事项：

1. 要给接口的每个方法打上请求注解，注意区分 Get、Post
2. 要给请求参数打上注解，比如 RequestParam、RequestBody
3. FeignClient 定义的请求路径一定要和服务提供方实际的请求路径保持一致

示例代码：

4）修改各业务服务的调用代码为 feignClient

5）编写 feignClient 服务的实现类，注意要和之前定义的客户端保持一致

6）开启 Nacos 的配置，让服务之间能够互相发现

所有模块引入 Nacos 依赖，然后给业务服务（包括网关）增加配置：

给业务服务项目启动类打上注解，开启服务发现、找到对应的客户端 Bean 的位置：

全局引入负载均衡器依赖：

7）启动项目，测试依赖能否注入，能否完成相互调用

微服务网关

微服务网关（yuoj-backend-gateway）：Gateway 聚合所有的接口，统一接受处理前端的请

求

为什么要用?

- 所有的服务端口不同，增大了前端调用成本
- 所有服务是分散的，你可需要集中进行管理、操作，比如集中解决跨域、鉴权、接口文档、服务的路由、接口安全性、流量染色、限流

> Gateway：想自定义一些功能，需要对这个技术有比较深的理解

Gateway 是应用层网关：会有一定的业务逻辑（比如根据用户信息判断权限）

Nginx 是接入层网关：比如每个请求的日志，通常没有业务逻辑

接口路由

统一地接受前端的请求，转发请求到对应的服务

如何找到路由？可以编写一套路由配置，通过 api 地址前缀来找到对应的服务

聚合文档

以一个全局的视角集中查看管理接口文档

使用 Knife4j 接口文档生成器，非常方便：https://doc.xiaominfo.com/docs/middleware-sources/spring-cloud-gateway/spring-gateway-introduction <https://doc.xiaominfo.com/docs/middleware-sources/spring-cloud-gateway/spring-gateway-introduction>

1）先给所有业务服务引入依赖，同时开启接口文档的配置

https://doc.xiaominfo.com/docs/middleware-sources/spring-cloud-gateway/spring-gateway-introduction#%E6%89%8B%E5%8A%A8%E9%85%8D%E7%BD%AE%E8%81%9A%E5%90%88manual <https://doc.xiaominfo.com/docs/middleware-sources/spring-cloud-gateway/spring-gateway-introduction#%E6%89%8B%E5%8A%A8%E9%85%8D%E7%BD%AE%E8%81%9A%E5%90%88manual>

2）给网关配置集中管理接口文档

网关项目引入依赖：

引入配置：

3）访问地址即可查看聚合接口文档：http://localhost:8101/doc.html#/home <http://localhost:8101/doc.html#/home>

分布式 Session 登录

必须引入 spring data redis 依赖：

解决 cookie 跨路径问题：

跨域解决

全局解决跨域配置：

权限校验

可以使用 Spring Cloud Gateway 的 Filter 请求拦截器，接受到请求后根据请求的路径判断能否访问。

> 扩展：可以在网关实现 Sentinel 接口限流降级，参考教程https://sca.aliyun.com/zh-cn/docs/2021.0.5.0/user-guide/sentinel/overview <https://sca.aliyun.com/zh-cn/docs/2021.0.5.0/user-guide/sentinel/overview>
> 扩展：可以使用 JWT Token 实现用户登录，在网关层面通过 token 获取登录信息，实现鉴权

Redisson RateLimiter 也可以实现限流，智能 BI 项目讲过。

思考

真的有必要用微服务么？

真的有必要用 Spring Cloud 实现微服务么？

企业内部一般使用 API（RPC、HTTP）实现跨部门、跨服务的调用，数据格式和调用代码全部自动生成，保持统一，同时解耦。

消息队列解耦

此处选用 RabbitMQ 消息队列改造项目，解耦判题服务和题目服务，题目服务只需要向消息队列发消息，判题服务从消息队列中取消息去执行判题，然后异步更新数据库即可

> 智能 BI 项目有对这部分知识点的详细讲解

基本代码引入

1）引入依赖

注意，使用的版本一定要和你的 springboot 版本一致！！！！！！！

2）在 yml 中引入配置：

3）创建交换机和队列

4）生产者代码

5）消费者代码

6）单元测试执行

项目异步化改造

要传递的消息是什么？题目提交 id

题目服务中，把原本的本地异步执行改为向消息队列发送消息：

判题服务中，监听消息，执行判题：

> 扩展：处理消息重试，避免消息积压
> 扩展：压力测试，验证

本期成果

1. 完成微服务项目改造
2. 浏览题目提交列表页面

本期作业

1. 完成单体项目的微服务改造：http://gitlab.code-nav.cn/root/yuoj-backend-microservice <http://gitlab.code-nav.cn/root/yuoj-backend-microservice>
2. 整理属于自己的微服务模板配置

项目上线

如何快速部署微服务项目？保姆级教程 <https://www.code-nav.cn/course/1788871761736998913/section/1789186997333499906?type=>