项目笔记

介绍：帮助大家寻找志同道合小伙伴的移动端 H5 网页。

项目资料

前端源码：http://gitlab.code-nav.cn/root/yupao-frontend <http://gitlab.code-nav.cn/root/yupao-frontend>

后端源码：http://gitlab.code-nav.cn/root/yupao-backend <http://gitlab.code-nav.cn/root/yupao-backend>

需求分析

1. 用户去添加标签，标签的分类（要有哪些标签、怎么把标签进行分类）学习方向 java / c++，工作 / 大学
2. 主动搜索：允许用户根据标签去搜索其他用户
    a. Redis 缓存
3. 组队
    a. 创建队伍
    b. 加入队伍
    c. 根据标签查询队伍
    d. 邀请其他人
4. 允许用户去修改标签
5. 推荐
    a. 相似度计算算法 + 本地实时计算

技术栈

前端

1. Vue 3 开发框架（提高页面开发的效率）
2. Vant UI（基于 Vue 的移动端组件库）（React 版 Zent）
3. Vite 2（打包工具，快！）
4. Nginx 来单机部署

后端

1. Java 编程语言 + SpringBoot 框架
2. SpringMVC + MyBatis + MyBatis Plus（提高开发效率）
3. MySQL 数据库
4. Redis 缓存
5. Swagger + Knife4j 接口文档

第 1 集

主要内容：

1. 找伙伴系统介绍和计划
2. 技术选型及介绍
3. 前端项目初始化
4. 页面设计及基础布局实现
5. 后端数据库表设计
6. 按标签搜索用户功能开发及性能分析

前端项目初始化

1）用脚手架初始化项目：

- Vue CLI：https://cli.vuejs.org/zh/ <https://cli.vuejs.org/zh/>
- Vite 脚手架：https://vitejs.cn/guide/#scaffolding-your-first-vite-project（推荐）<https://vitejs.cn/guide/#scaffolding-your-first-vite-project（推荐）>

2）整合组件库 Vant：

- 安装 Vant

- 配置按需引入：npm i vite-plugin-style-import@1.4.1 -D

前端开发页面经验

1. 多参考
2. 从整体到局部
3. 先想清楚页面要做成什么样子，再写代码

前端主页

设计

从上到下依次为：

1）导航条：展示当前页面名称

2）主页搜索框：点击跳转到搜索页 => 搜索结果页（标签筛选页）

3）内容展示

4）tab 栏：

- 主页（推荐页 + 广告）
  - 搜索框
  - banner
  - 推荐信息流
- 队伍页
- 用户页

开发

很多页面要复用组件 / 样式，重复写很麻烦、不利于维护，所以抽象一个通用的布局（Layout）。

数据库表设计

该项目的库表设计建立在星球用户中心项目的基础上。可以直接把用户中心的源码拿来二次开发。

SQL 语句分类

DDL：define，建表、操作表

DML：manage，更新删除数据，影响实际表里的内容

DCL：control，控制，权限

DQL：query 查询，select

参考：https://www.cnblogs.com/fan-yuan/p/7879353.html
<https://www.cnblogs.com/fan-yuan/p/7879353.html>

标签表（分类表）

建议用标签，而不是用分类，更灵活。

标签分类

性别：男、女

方向：Java、C++、Go、前端

正在学：Spring

目标：考研、春招、秋招、社招、考公、竞赛（蓝桥杯）、转行、跳槽

段位：初级、中级、高级、王者

身份：小学、初中、高中、大一、大二、大三、大四、学生、待业、已就业、研一、研二、研三

状态：乐观、有点丧、一般、单身、已婚、有对象

还可以支持用户自己定义标签。

表设计

id int 主键

标签名 varchar 非空（必须唯一，唯一索引）

上传标签的用户 userId int（如果要根据 userId 查已上传标签的话，最好加上，普通索引）

父标签 id ， parentId，int（分类）

是否为父标签 isParent, tinyint（0 不是父标签、1 - 父标签）

创建时间 createTime，datetime

更新时间 updateTime，datetime

是否删除 isDelete， tinyint（0、1）

验证设计

主要是思考该设计能不能满足业务操作诉求：

1）怎么查询所有标签，并且把标签分好组？

答：按父标签 id 分组，能实现

2）根据父标签查询子标签？

答：根据 id 查询，能实现

用户表

思考：怎么给用户表补充标签？

一定要根据自己的实际需求来！！！

常见方案有 2 种：

1）直接在用户表补充 tags 字段，格式为 json 字符串，比如：['Java', '男']

优点：查询方便、不用新建关联表，标签是用户的固有属性（除了该系统、其他系统可能要用到，标签是用户的固有属性）节省开发成本

缺点：根据标签查用户时只能用模糊查询，或者遍历用户列表，性能不高。

2）加一个关联表，记录用户和标签的关系

关联表的优点（适用场景）：查询灵活，可以正查、反查。

缺点：要多建一个表、多维护一个表

提示：企业大项目开发中尽量减少关联查询，很影响扩展性，而且会影响查询性能

此处选择第一种，可以用缓存来提高查询效率。

开发后端接口

按标签搜索用户

1．允许用户传入多个标签，多个标签都存在才搜索出来 and。like '%Java%' and like '%C++%'。
2．允许用户传入多个标签，有任何一个标签存在就能搜索出来 or。like '%Java%' or like '%C++%'

2 种实现方式：

1．SQL 查询：实现简单，可以通过拆分查询进一步优化
2．内存查询：灵活，可以通过并发进一步优化

方案选择：

- 如果参数可以分析，根据用户的参数去选择查询方式，比如标签数
- 如果参数不可分析，并且数据库连接足够、内存空间足够，可以并发同时查询，谁先返回用谁。
- 还可以 SQL 查询与内存计算相结合，比如先用 SQL 过滤掉部分 tag

建议通过实际测试来分析哪种查询比较快，数据量大的时候验证效果更明显！

解析 JSON 字符串

序列化：java 对象转成 json

反序列化：把 json 转为 java 对象

java json 序列化库有很多：

1．gson（google 的，推荐）
2．fastjson alibaba（阿里出品，快，但是漏洞太多）
3．jackson
4．kryo（性能极高的序列化库）

第 2 集

主要内容：

1．标签搜索接口调试
2．前端整合路由，简单介绍原理
3．前端页面开发，搜索页、用户页、用户修改页。（这里做的慢了些，大家可以倍速观看）

Java 8 新特性

1. stream / parallelStream 流式处理
2. Optional 可选类

前端整合路由

Vue-Router：https://router.vuejs.org/zh/guide/#html，直接看官方文档引入。<https://router.vuejs.org/zh/guide/#html，直接看官方文档引入。>

Vue-Router 其实就是帮助你根据不同的 url 来展示不同的页面（组件），不用自己写 if / else。

路由配置影响整个项目，所以建议单独用 config 目录、单独的配置文件去集中定义和管理。

有些组件库可能自带了和 Vue-Router 的整合，所以尽量先看组件文档、省去自己写的时间。

第 3 集

主要内容：

1. Swagger + Knife4j 接口文档整合
2. 分析星球接口以及后端 Excel 处理（大家可以倍速观看）

后端整合 Swagger + Knife4j 接口文档

什么是接口文档？写接口信息的文档。

每个接口的信息包括：

- 请求参数
- 响应参数
    - 错误码
- 接口地址
- 接口名称
- 请求类型
- 请求格式
- 备注

谁用接口文档?

答：一般是后端或者负责人来提供，后端和前端都要使用。

为什么需要接口文档?

- 有个书面内容（背书或者归档），便于大家参考和查阅，便于 **沉淀和维护** ，拒绝口口相传
- 接口文档便于前端和后端开发对接，前后端联调的 **介质** 。后端 => 接口文档 <= 前端
- 好的接口文档支持在线调试、在线测试，可以作为工具提高我们的开发测试效率

怎么做接口文档?

- 手写：比如腾讯文档、Markdown 笔记
- 自动化接口文档生成：自动根据项目代码生成完整的文档或在线调试的网页。Swagger、Postman（侧重接口管理）（国外）；apifox、apipost、eolink（国产）

使用 Swagger

1. 引入依赖（Swagger 或 Knife4j：https://doc.xiaominfo.com/knife4j/documentation/get_start.html）<https://doc.xiaominfo.com/knife4j/documentation/get_start.html）>
2. 自定义 Swagger 配置类
3. 定义需要生成接口文档的代码位置（Controller）
4. 千万注意：线上环境不要把接口暴露出去！！！可以通过在 SwaggerConfig 配置文件开头加上 `@Profile({"dev", "test"})` 限定配置仅在部分环境开启
5. 启动即可
6. 可以通过在 controller 方法上添加 @Api、@ApiImplicitParam(name = "name",value = "姓名",required = true) @ApiOperation(value = "向客人问好") 等注解来自定义生成的接口描述信息

如果 springboot version >= 2.6，需要添加如下配置：

```
spring:
  mvc:
      pathmatch:
      matching-strategy: ANT_PATH_MATCHER
```

用户信息导入及同步

计划：

1．把所有星球用户的信息导入

2．把写了自我介绍的同学的标签信息导入

怎么抓取网页内容?

推荐安装 FeHelper 前端辅助插件。

1）分析原网站是怎么获取这些数据的？哪个接口?

按 F12 打开控制台，查看网络请求，复制 curl 代码便于查看和执行，比如：

```
curl "https://api.zsxq.com/v2/hashtags/48844541281228/topics?count=20" ^
  -H "authority: api.zsxq.com" ^
  -H "accept: application/json, text/plain, */*" ^
  -H "accept-language: zh-CN,zh;q=0.9" ^
  -H "cache-control: no-cache" ^
  -H "origin: https://wx.zsxq.com" ^
  -H "pragma: no-cache" ^
  -H "referer: https://wx.zsxq.com/" ^
  --compressed
```

2） **用程序去调用接口** （java okhttp httpclient / python request 库等都可以）

3）处理（清洗）一下数据，之后就可以写到数据库里

导入流程

1．从 excel 中导入全量用户数据并去重（使用 Easy Excel 库来实现）

2．抓取写了自我介绍的同学信息，提取出用户昵称、用户唯一 id、自我介绍信息

3．从自我介绍中提取信息，然后写入到数据库中

EasyExcel

地址：https://alibaba-easyexcel.github.io/index.html <https://alibaba-easyexcel.github.io/index.html>

两种读对象的方式：

1．确定表头：建立对象，和表头形成映射关系

2．不确定表头：每一行数据映射为 Map<String, Object>

两种读取模式：

1. 监听器：先创建监听器、在读取文件时绑定监听器。单独抽离处理逻辑，代码清晰易于维护；一条一条处理，适用于数据量大的场景。
2. 同步读：无需创建监听器，一次性获取完整数据。方便简单，但是数据量大时会有等待时常，也可能内存溢出。

第 4 集

主要内容：

1. 用户搜索页、列表页前后端开发及联调完成
2. 后端分布式登录改造

前端页面跳转传值的方式

1. query => url searchParams，url 后附加参数，传递的值长度有限
2. vuex（全局状态管理），搜索页将关键词塞到状态中，搜索结果页从状态取值

Session 共享

Cookie 范围

种 session 的时候注意范围，Spring Boot 项目中可以通过 cookie.domain 来配置。

比如两个域名：

- aaa.yupi.com
- bbb.yupi.com

如果要共享 cookie，可以种一个更高层的公共域名，比如 yupi.com

为什么需要共享?

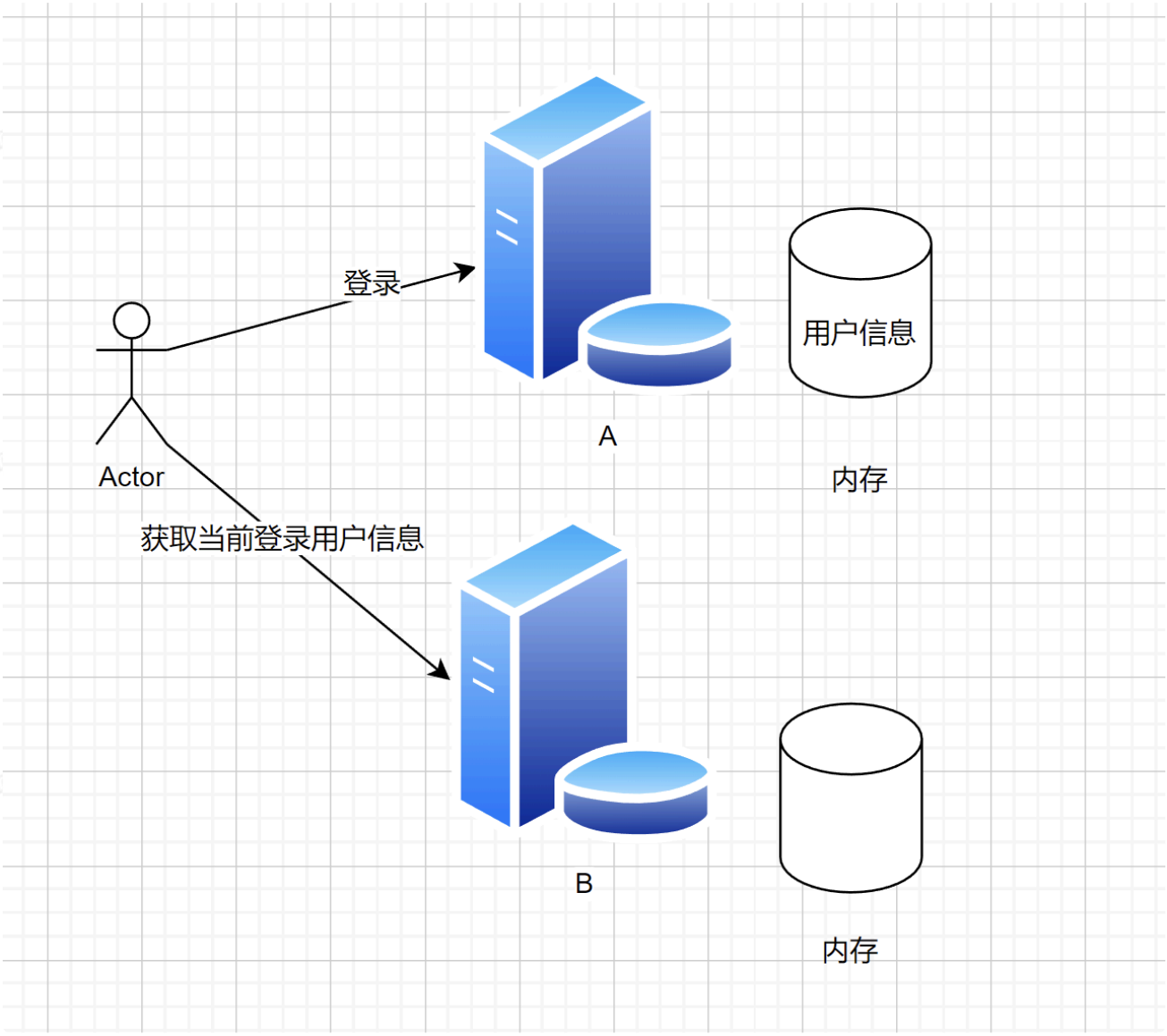
思考：为什么服务器 A 登录后，请求发到服务器 B，不认识该用户?

原因如下：

1. 用户在 A 登录，所以 session（用户登录信息）存在了 A 上

2．结果请求 B 时，B 没有用户信息，所以不认识。

如图：



解决方案： **共享存储** ，而不是把数据放到单台服务器的内存中

如何共享存储?

核心思想：把数据放到同一个地方去集中管理。

1．Redis（基于内存的 K / V 数据库）此处选择 Redis，因为用户信息读取 / 是否登录的判断极其 **频繁** ，Redis 基于内存，读写性能很高，简单的数据单机 qps 5w - 10w。

2．MySQL

3．文件服务器 ceph

Session 共享实现

1）安装 Redis

官网：https://redis.io/ <https://redis.io/>

windows 下载 Redis 5.0.14：

链接：https://pan.baidu.com/s/1XcsAIrdeesQAyQU2lE3cOg <https://pan.baidu.com/s/1XcsAIrdeesQAyQU2lE3cOg>

提取码：vkoi

Redis 管理工具 quick redis：https://quick123.net/ <https://quick123.net/>

2）引入 redis，能够操作 redis：

```
<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-
<dependency>
    <groupId>org.springframework.boot</groupId>

    <artifactId>spring-boot-starter-data-redis</artifactId>

    <version>2.6.4</version>

</dependency>

```

3）引入 spring-session 和 redis 的整合，使得自动将 session 存储到 redis 中：

```
<!-- https://mvnrepository.com/artifact/org.springframework.session/spring-se
<dependency>
    <groupId>org.springframework.session</groupId>

    <artifactId>spring-session-data-redis</artifactId>

    <version>2.6.3</version>

</dependency>

```

4）修改 spring-session 存储配置 `spring.session.store-type`

默认是 none，表示存储在单台服务器

store-type: redis，表示从 redis 读写 session

其他单点登录方案

常用的就是 JWT。

Redis Session 对比 JWT 的优缺点：https://zhuanlan.zhihu.com/p/108999941 <https://zhuanlan.zhihu.com/p/108999941>

第 5 集

> 这一集讲的较慢，请大家根据自己的进度选择性观看

主要内容：

1. 开发用户修改页面
2. 开发用户登录功能

第 6 集

在线回放：https://meeting.tencent.com/v2/cloud-record/share?id=65c1d5f7-8465-4c70-99bb-eb3e148ceb0c&from=3（访问密码：wr5M）
<https://meeting.tencent.com/v2/cloud-record/share?id=65c1d5f7-8465-4c70-99bb-eb3e148ceb0c&from=3（访问密码：wr5M）>

主要内容：

1. 主页前端开发（列表组件抽象）
2. 批量插入数据功能开发 + 经验分享
3. 定时任务注解
4. 测试及优化批量导入功能（涉及性能优化 + 并发编程知识）

开发主页

最简单：直接 list 列表组件实现

模拟 1000 万个用户，再去查询

导入数据

几种导入数据的方式

1. 用可视化界面：适合一次性导入、数据量可控

2．写程序：for 循环，建议分批，不要一把梭哈（可以用接口来控制）。 **要保证可控、幂等，注意线上环境和测试环境是有区别的！**
3．执行 SQL 语句：适用于小数据量

编写一次性任务

for 循环插入数据的特点：

1．频繁建立和释放数据库链接（用批量查询解决）
2．for 循环是绝对线性的（可以并发提速）

使用并发时要注意数据的插入先后顺序是否无所谓。

并发时不要用到非并发类的集合。

建立执行器（线程池）：

```
1    private ExecutorService executorService = new ThreadPoolExecutor(16, 1000, 100
```

连接池参数设置：

```
1    // CPU 密集型：分配的核心线程数 = CPU - 1
2    // IO 密集型：分配的核心线程数可以大于 CPU 核数
```

第 7 集

在线回放：https://meeting.tencent.com/v2/cloud-record/share?id=cbf47b3d-7215-4d4c-9808-8b832a0b1521&from=3（访问密码：4x4z） <https://meeting.tencent.com/v2/cloud-record/share?id=cbf47b3d-7215-4d4c-9808-8b832a0b1521&from=3（访问密码：4x4z）>

主要内容（后端）：

1．缓存和分布式缓存讲解
2．Redis 介绍（5 种数据结构等）
3．Java 操作 Redis 的方法（4 种方法 + 对比分析）
4．Java Redis Template 序列化（包含源码追踪）
5．首页缓存开发与注意事项
6．缓存预热设计与实现
7．定时任务介绍和实现

数据查询慢怎么办？

用缓存：提前把数据取出来保存好（通常保存到读写更快的介质，比如内存），就不用再查数据库了，可以更快地读写。

用定时任务：预加载缓存，定时更新缓存。

思考：多个机器都要执行同一个任务么？（可以用分布式锁解决：控制同一时间只有一台机器去执行定时任务，其他机器不用重复执行了）

缓存分类

分布式缓存：

- Redis（分布式缓存）
- memcached（分布式）
- Etcd（云原生架构的一个分布式存储， **存储配置** ，扩容能力）

单机缓存：

- ehcache
- Java 内存集合，如 HashMap
- Caffeine（Java 内存缓存性能之王，高性能）
- Google Guava

Redis 缓存实现

> NoSQL 数据库

key - value 存储系统（区别于 MySQL，他存储的是键值对）

Redis 数据结构

基本：

- String 字符串类型： name: "yupi"
- List 列表：names: ["yupi", "dogyupi", "yupi"]
- Set 集合：names: ["yupi", "dogyupi"]（值不能重复）
- Hash 哈希：nameAge: { "yupi": 1, "dogyupi": 2 }

- Zset 集合：names: { yupi - 9， dogyupi - 12 }（适合做排行榜）

高级：

- bloomfilter（布隆过滤器，主要从大量的数据中快速过滤值，比如邮件黑名单拦截）
- geo（计算地理位置）
- hyperloglog（pv / uv）
- pub / sub（发布订阅，类似消息队列）
- BitMap （1001010101010101010101010101）

自定义序列化

为了防止写入 Redis 的数据乱码、浪费空间等，可以自定义序列化器。示例代码如下：

```
package com.yupi.yupao.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.data.redis.serializer.RedisSerializer;

@Configuration
public class RedisTemplateConfig {

    @Bean
    public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory
        RedisTemplate<String, Object> redisTemplate = new RedisTemplate<>();
        redisTemplate.setConnectionFactory(connectionFactory);
        redisTemplate.setKeySerializer(RedisSerializer.string());
        return redisTemplate;
    }
}
```

> 引入一个新库时，记得先写测试类

Java 操作 Redis

Spring Data Redis（推荐）

地址：spring-data-redis
<https://mvnrepository.com/artifact/org.springframework.data/spring-data-redis>

Spring Data：通用的数据访问框架，定义了一组 **增删改查** 的接口

还可以操作：mysql、redis、jpa

使用方式如下：

1）引入

```
<dependency>
    <groupId>org.springframework.boot</groupId>

    <artifactId>spring-boot-starter-data-redis</artifactId>

    <version>2.6.4</version>

</dependency>

```

2）配置 Redis 地址

```
spring:
  # redis 配置
  redis:
    port: 6379
    host: localhost
    database: 0
```

Jedis

独立于 Spring 操作 Redis 的 Java 客户端

要配合 Jedis Pool 使用

Lettuce

高阶 的操作 Redis 的 Java 客户端

异步、连接池

Redisson

分布式操作 Redis 的 Java 客户端，让你像在使用本地的集合一样操作 Redis（分布式 Redis 数据网格）

JetCache

不再赘述

几种库对比

1. 如果你用的是 Spring，并且没有过多的定制化要求，可以用 Spring Data Redis，最方便
2. 如果你用的不是 Spring，并且追求简单，并且没有过高的性能要求，可以用 Jedis + Jedis Pool
3. 如果你的项目不是 Spring，并且追求高性能、高定制化，可以用 Lettuce，支持异步、连接池
4. 如果你的项目是分布式的，需要用到一些分布式的特性（比如分布式锁、分布式集合），推荐用 redisson

设计缓存 key

关键点：不同用户看到的数据不同

建议格式：

systemId:moduleId:func:options（不要和别人冲突）

比如：yupao:user:recommed:userId

注意！redis 内存不能无限增加，一定要设置过期时间！！！

缓存预热

问题：即使用了缓存，第一个用户访问还是很慢（假如第一个访客是老板，哦豁！）。

缓存预热的优点：

1. 解决上面的问题，可以让用户始终访问很快
2. 也能一定程度上保护数据库

缺点：

1. 增加开发成本（你要额外的开发、设计）
2. 预热的时机和时间如果错了，有可能你缓存的数据不对或者太老
3. 需要占用额外空间

怎么缓存预热?

1. 定时任务
2. 手动触发

实现

用定时任务，每天刷新所有用户的推荐列表。

注意点：

1. 缓存预热的意义（新增少、总用户多）
2. 缓存的空间不能太大，要预留给其他缓存空间
3. 缓存数据的周期（此处每天一次）

> 分析优缺点的时候，要打开思路，从整个项目从 0 到 1 的链路上去分析

第 8 集

在线回放：https://meeting.tencent.com/v2/cloud-record/share?id=8a8667dc-f061-4972-86e9-1b9d7e27aac5&from=3（访问密码：QzER）
<https://meeting.tencent.com/v2/cloud-record/share?id=8a8667dc-f061-4972-86e9-1b9d7e27aac5&from=3（访问密码：QzER）>

主要内容（后端）：

1. 分布式定时任务执行控制
2. 锁的概念
3. 分布式锁概念和常见问题
4. 分布式锁实战

定时任务实现

1. Spring Scheduler（spring boot 默认整合了，推荐这种方式）
2. Quartz（独立于 Spring 存在的定时任务框架）
3. XXL-Job 之类的分布式任务调度平台（界面 + sdk）

采用第一种方式：

1. 主类开启 @EnableScheduling
2. 给要定时执行的方法添加 @Scheduling 注解，指定 cron 表达式或者执行频率

记住，不要去背 cron 表达式！！！用现成的工具即可：

- https://cron.qqe2.com/ <https://cron.qqe2.com/>
- https://www.matools.com/crontab/ <https://www.matools.com/crontab/>

控制定时任务的执行

要控制定时任务在同一时间只有 1 个服务器能执行。

为啥?

1. 浪费资源，想象 10000 台服务器同时 “打鸣”
2. 脏数据，比如重复插入

怎么做？几种方案：

1. 分离定时任务程序和主程序，只在 1 个服务器运行定时任务。成本太大
2. 写死配置，每个服务器都执行定时任务，但是只有 ip 符合配置的服务器才真实执行业务逻辑，其他的直接返回。成本最低；但是我们的 IP 可能是不固定的，把 IP 写的太死了
3. 动态配置，配置是可以轻松的、很方便地更新的（代码无需重启），但是只有 ip 符合配置的服务器才真实执行业务逻辑。
    - 数据库
    - Redis
    - 配置中心（Nacos、Apollo、Spring Cloud Config）

问题：服务器多了、IP 不可控还是很麻烦，还是要人工修改

4. 分布式锁，只有抢到锁的服务器才能执行业务逻辑。坏处：增加成本；好处：不用手动配置，多少个服务器都一样。

注意，只要是单机，就会存在单点故障。

锁

有限资源的情况下，控制同一时间（段）只有某些线程（用户 / 服务器）能访问到资源。

Java 实现锁：synchronized 关键字、并发包的类

但存在问题：只对单个 JVM 有效

分布式锁

为啥需要分布式锁?

1．有限资源的情况下，控制同一时间（段）只有某些线程（用户 / 服务器）能访问到资源。

2．单个锁只对单个 JVM 有效

分布式锁实现的关键

抢锁机制

怎么保证同一时间只有 1 个服务器能抢到锁?

核心思想 就是：先来的人先把数据改成自己的标识（服务器 ip），后来的人发现标识已存在，就抢锁失败，继续等待。

等先来的人执行方法结束，把标识清空，其他的人继续抢锁。

MySQL 数据库：select for update 行级锁（最简单），或者用乐观锁。

Redis 实现：内存数据库，读写速度快。支持 setnx、lua 脚本，比较方便我们实现分布式锁。

setnx：set if not exists 如果不存在，则设置；只有设置成功才会返回 true，否则返回 false。

注意事项

1）用完锁要释放（腾地方）

2）锁一定要加过期时间

3）如果方法执行时间过长，锁提前过期了?

会导致问题：

1．连锁效应：释放掉别人的锁

2．这样还是会存在多个方法同时执行的情况

解决方案：续期

比如：

```
boolean end = false;

new Thread(() -> {
    if (!end)}{
    续期
})

end = true;
```

4）释放锁的时候，有可能先判断出是自己的锁，但这时锁过期了，最后还是释放了别人的锁

解决方案：Redis + lua 脚本保证操作原子性

```
// 原子操作
if(get lock == A) {
    // set lock B
    del lock
}
```

5）Redis 如果是集群（而不是只有一个 Redis），如果分布式锁的数据不同步怎么办?

解决方案：https://blog.csdn.net/feiying0canglang/article/details/113258494 <https://blog.csdn.net/feiying0canglang/article/details/113258494>

拒绝自己实现！！！直接用现成的。

Redisson 实现分布式锁

Redisson 是一个 java 操作 Redis 的客户端， **提供了大量的分布式数据集来简化对 Redis 的操作和使用，可以让开发者像使用本地集合一样使用 Redis，完全感知不到 Redis 的存在。**

关键词：Java Redis 客户端，分布式数据网格，实现了很多 Java 里支持的集合。

2 种引入方式

1. spring boot starter 引入（不推荐，版本迭代太快，容易冲突）：https://github.com/redisson/redisson/tree/master/redisson-spring-boot-starter <https://github.com/redisson/redisson/tree/master/redisson-spring-boot-starter>
2. 直接引入：https://github.com/redisson/redisson#quick-start <https://github.com/redisson/redisson#quick-start>

使用 Redisson

示例代码如下，创建分布式列表：

```
// list，数据存在本地 JVM 内存中
List<String> list = new ArrayList<>();
list.add("yupi");
System.out.println("list:" + list.get(0));

list.remove(0);

// 数据存在 redis 的内存中
RList<String> rList = redissonClient.getList("test-list");
rList.add("yupi");
System.out.println("rlist:" + rList.get(0));
rList.remove(0);
```

分布式锁保证定时任务不重复执行

实现代码如下：

```
void testWatchDog() {
    RLock lock = redissonClient.getLock("yupao:precachejob:docache:lock");
    try {
        // 只有一个线程能获取到锁
        if (lock.tryLock(0, -1, TimeUnit.MILLISECONDS)) {
            // todo 实际要执行的方法
            doSomeThings();
            System.out.println("getLock: " + Thread.currentThread().getId());
        }
    } catch (InterruptedException e) {
        System.out.println(e.getMessage());
    } finally {
        // 只能释放自己的锁
        if (lock.isHeldByCurrentThread()) {
            System.out.println("unLock: " + Thread.currentThread().getId());
            lock.unlock();
        }
    }
}
```

注意：

1. waitTime 设置为 0，只抢一次，抢不到就放弃
2. 注意释放锁要写在 finally 中

Redisson 看门狗机制

> redisson 中提供的续期机制

开一个监听线程，如果方法还没执行完，就帮你重置 redis 锁的过期时间。

原理：

1. 监听当前线程，默认过期时间是 30 秒，每 10 秒续期一次（补到 30 秒）
2. 如果线程挂掉（注意 debug 模式也会被它当成服务器宕机），则不会续期

建议阅读：https://blog.csdn.net/qq_26222859/article/details/79645203 <https://blog.csdn.net/qq_26222859/article/details/79645203>

其他分布式锁实现方式

Zookeeper 实现（不推荐）

第 9 - 11 集

9 在线回放：https://meeting.tencent.com/v2/cloud-record/share?id=ce65608f-752f-448f-864a-3fd7d6e34cd2&from=3 <https://meeting.tencent.com/v2/cloud-record/share?from=3&id=ce65608f-752f-448f-864a-3fd7d6e34cd2> （访问密码：beMk）

10 在线回放：https://meeting.tencent.com/v2/cloud-record/share?id=83a15e95-1284-42a6-aa06-f7a710fc38bd&from=3 <https://meeting.tencent.com/v2/cloud-record/share?from=3&id=83a15e95-1284-42a6-aa06-f7a710fc38bd>

11 在线回放：https://meeting.tencent.com/v2/cloud-record/share?id=2f096a64-d837-4bbd-9d67-9a37019e4eec&from=3 <https://meeting.tencent.com/v2/cloud-record/share?from=3&id=2f096a64-d837-4bbd-9d67-9a37019e4eec>

主要内容（前后端新功能开发）：

1. 组队功能需求分析
2. 组队功能系统设计
3. 创建队伍功能开发及测试
4. 搜索队伍
5. 更新队伍
6. 加入队伍
7. 退出队伍后端接口
8. 解散队伍后端接口

9. 前端创建队伍、队伍列表页开发

组队功能开发

理解为王者荣耀匹配。

理想的应用场景

我要跟别人一起参加竞赛或者做项目，可以发起队伍或者加入别人的队伍

需求分析和排期

用户可以 **创建** 一个队伍，设置队伍的人数、队伍名称（标题）、描述、超时时间 P0。

> 队长、剩余的人数
> 聊天?
> 公开 或 private 或加密
> **用户创建队伍最多 5 个**

展示队伍列表，根据名称搜索队伍 P0，信息流中不展示已过期的队伍

修改队伍信息 P0 ~ P1

用户可以加入队伍（其他人、未满、未过期），允许加入多个队伍，但是要有个上限 P0

> 是否需要队长同意？筛选审批?

用户可以退出队伍（如果队长退出，权限转移给第二早加入的用户 —— 先来后到） P1

队长可以解散队伍 P0

分享队伍 => 邀请其他用户加入队伍 P1

业务流程：

1. 生成分享链接（分享二维码）
2. 用户访问链接，可以点击加入

队伍人满后发送消息通知 P1

系统（接口）设计

1、创建队伍

用户可以 **创建** 一个队伍，设置队伍的人数、队伍名称（标题）、描述、超时时间 P0

> 队长、剩余的人数
> 聊天?
> 公开 或 private 或加密
> 信息流中不展示已过期的队伍

1. 请求参数是否为空?
2. 是否登录，未登录不允许创建
3. 校验信息
    a. 队伍人数 > 1 且 <= 20
    b. 队伍标题 <= 20
    c. 描述 <= 512
    d. status 是否公开（int）不传默认为 0（公开）
    e. 如果 status 是加密状态，一定要有密码，且密码 <= 32
    f. 超时时间 > 当前时间
    g. 校验用户最多创建 5 个队伍
4. 插入队伍信息到队伍表
5. 插入用户 => 队伍关系到关系表

2、查询队伍列表

分页展示队伍列表，根据名称、最大人数等搜索队伍 P0，信息流中不展示已过期的队伍。

1. 从请求参数中取出队伍名称等查询条件，如果存在则作为查询条件
2. 不展示已过期的队伍（根据过期时间筛选）
3. 可以通过某个 **关键词** 同时对名称和描述查询
4. **只有管理员才能查看加密还有非公开的房间**
5. 关联查询已加入队伍的用户信息
6. **关联查询已加入队伍的用户信息（可能会很耗费性能，建议大家用自己写 SQL 的方式实现）**

实现方式：

1）自己写 SQL

```
// 1. 自己写 SQL
// 查询队伍和创建人的信息
// select * from team t left join user u on t.userId = u.id
// 查询队伍和已加入队伍成员的信息
// select *
// from team t
//         left join user_team ut on t.id = ut.teamId
//         left join user u on ut.userId = u.id;
```

2）用 MyBatis Plus 构造查询

3、修改队伍信息

1. 判断请求参数是否为空
2. 查询队伍是否存在
3. 只有管理员或者队伍的创建者可以修改
4. 如果用户传入的新值和老值一致，就不用 update 了（可自行实现，降低数据库使用次数）
5. **如果队伍状态改为加密，必须要有密码**
6. 更新成功

4、用户可以加入队伍

其他人、未满、未过期，允许加入多个队伍，但是要有个上限 P0

1. 用户最多加入 5 个队伍
2. 队伍必须存在，只能加入未满、未过期的队伍
3. 不能加入自己的队伍，不能重复加入已加入的队伍（幂等性）
4. 禁止加入私有的队伍
5. 如果加入的队伍是加密的，必须密码匹配才可以
6. 新增队伍 - 用户关联信息

注意，一定要加上事务注解！！！！

5、用户可以退出队伍

请求参数：队伍 id

1. 校验请求参数
2. 校验队伍是否存在

3．校验我是否已加入队伍

4．如果队伍

a．只剩一人，队伍解散

b．还有其他人

ⅰ．如果是队长退出队伍，权限转移给第二早加入的用户 —— 先来后到（只用取 id 最小的 2 条数据）

ⅱ．非队长，自己退出队伍

6、队长可以解散队伍

请求参数：队伍 id

业务流程：

1．校验请求参数

2．校验队伍是否存在

3．校验你是不是队伍的队长

4．移除所有加入队伍的关联信息

5．删除队伍

7、获取当前用户已加入的队伍

8、获取当前用户创建的队伍

复用 listTeam 方法，只新增查询条件，不做修改（开闭原则）

使用事务注解

@Transactional(rollbackFor = Exception.class)

要么数据操作都成功，要么都失败

数据库表设计

队伍表 team

字段：

- id 主键 bigint（最简单、连续，放 url 上比较简短，但缺点是爬虫）
- name 队伍名称
- description 描述
- maxNum 最大人数
- expireTime 过期时间
- userId 创建人 id
- status 0 - 公开，1 - 私有，2 - 加密
- password 密码
- createTime 创建时间
- updateTime 更新时间
- isDelete 是否删除

示例 SQL：

```
create table team
(
    id           bigint auto_increment comment 'id'
        primary key,
    name   varchar(256)                  not null comment '队伍名称',
    description varchar(1024)                    null comment '描述',
    maxNum    int      default 1                 not null comment '最大人数',
    expireTime    datetime  null comment '过期时间',
    userId            bigint comment '用户id',
    status    int      default 0                 not null comment '0 - 公开，1
    password varchar(512)                       null comment '密码',

        createTime   datetime default CURRENT_TIMESTAMP null comment '创建时间
    updateTime   datetime default CURRENT_TIMESTAMP null on update CURRENT_TI
    isDelete     tinyint  default 0                 not null comment '是否删除
)
    comment '队伍';
```

用户 - 队伍表 user_team

两个关系：

1. 用户加了哪些队伍？
2. 队伍有哪些用户？

两种实现方式：

1. 建立用户 - 队伍关系表 teamId userId（便于修改，查询性能高一点，可以选择这个，不用全表遍历）
2. 用户表补充已加入的队伍字段，队伍表补充已加入的用户字段（便于查询，不用写多对多的代码，可以直接根据队伍查用户、根据用户查队伍）

字段：

- id 主键
- userId 用户 id
- teamId 队伍 id
- joinTime 加入时间
- createTime 创建时间
- updateTime 更新时间
- isDelete 是否删除

示例 SQL：

```
create table user_team
(
    id           bigint auto_increment comment 'id'
        primary key,
    userId            bigint comment '用户id',
    teamId            bigint comment '队伍id',
    joinTime datetime  null comment '加入时间',
    createTime   datetime default CURRENT_TIMESTAMP null comment '创建时间',
    updateTime   datetime default CURRENT_TIMESTAMP null on update CURRENT_TI
    isDelete     tinyint  default 0                 not null comment '是否删除
)
    comment '用户队伍关系';
```

为什么需要请求参数包装类？

1. 请求参数名称 / 类型和实体类不一样
2. 有一些参数用不到，如果要自动生成接口文档，会增加理解成本
3. 对个实体类映射到同一个对象

为什么需要包装类？

可能有些字段需要隐藏，不能返回给前端

或者有些字段某些方法是不关心的

第 12 - 13 集

12 在线回放：https://meeting.tencent.com/v2/cloud-record/share?id=91e7942d-ab8e-489f-ad39-c0ac22072099&from=3 <https://meeting.tencent.com/v2/cloud-record/share?from=3&id=91e7942d-ab8e-489f-ad39-c0ac22072099>

访问密码：HfrN

13 在线回放：https://meeting.tencent.com/v2/cloud-record/share?id=bd47843d-fb08-43ca-8549-a3f32c7da65b&from=3 <https://meeting.tencent.com/v2/cloud-record/share?from=3&id=bd47843d-fb08-43ca-8549-a3f32c7da65b>

访问密码：K4a7

主要内容：

1. 开发完成个人队伍页面、队伍检索等功能（前端 + 后端）
2. 开发用户匹配功能（编辑距离算法）
3. 优化用户匹配功能后端，开发前端用户匹配模式
4. 优化前端加载效果，使用骨架屏
5. 优化前端队伍操作权限
6. 实现前端导航标题动态切换

前端不同页面怎么传递数据?

1. url querystring（xxx?id=1） 比较适用于页面跳转
2. url（/team/:id，xxx/1）
3. hash (/team#1)
4. localStorage
5. context（全局变量，同页面或整个项目要访问公共变量）

随机匹配

需求背景：为了帮大家更快地发现和自己兴趣相同的朋友

思考：匹配 1 个还是匹配多个?

答：匹配多个，并且按照匹配的相似度从高到低排序

思考：怎么匹配？（根据什么匹配）

答：标签 tags

> 还可以根据 user_team 匹配加入相同队伍的用户

问题本质：找到有相似标签的用户

举例：

- 用户 A：[Java, 大一, 男]
- 用户 B：[Java, 大二, 男]
- 用户 C：[Python, 大二, 女]
- 用户 D：[Java, 大一, 女]

1、怎么匹配?

1. 找到有共同标签最多的用户（TopN）
2. 共同标签越多，分数越高，越排在前面
3. 如果没有匹配的用户，随机推荐几个（降级方案）

两种算法

编辑距离算法：https://blog.csdn.net/DBC_121/article/details/104198838 <https://blog.csdn.net/DBC_121/article/details/104198838>

> 最小编辑距离：字符串 1 通过最少多少次增删改字符的操作可以变成字符串 2

余弦相似度算法：https://blog.csdn.net/m0_55613022/article/details/125683937（如果需要带权重计算，比如学什么方向最重要，性别相对次要）<https://blog.csdn.net/m0_55613022/article/details/125683937（如果需要带权重计算，比如学什么方向最重要，性别相对次要）>

2、怎么对所有用户匹配，取 TOP?

直接取出所有用户，依次和当前用户计算分数，取 TOP N（54 秒）

优化方法

1．切忌不要在数据量大的时候循环输出日志（取消掉日志后 20 秒）

2．Map 存了所有的分数信息，占用内存

解决：维护一个固定长度的有序集合（sortedSet），只保留分数最高的几个用户（时间换空间）

e.g.【3, 4, 5, 6, 7】取 TOP 5，id 为 1 的用户就不用放进去了

3．细节：剔除自己 √

4．尽量只查需要的数据：

a．过滤掉标签为空的用户 √

b．根据部分标签取用户（前提是能区分出来哪个标签比较重要）

c．只查需要的数据（比如 id 和 tags） √（7.0s）

5．提前查？（定时任务）

a．提前把所有用户给缓存（不适用于经常更新的数据）

b．提前运算出来结果，缓存（针对一些重点用户，提前缓存）

类比大数据推荐机制

大数据推荐场景：比如说有几亿个商品，难道要查出来所有的商品？难道要对所有的数据计算一遍相似度？

大数据推荐流程：

- 检索 => 召回 => 粗排 => 精排 => 重排序等等
- 检索：尽可能多地查符合要求的数据（比如按记录查）
- 召回：查询可能要用到的数据（不做运算）
- 粗排：粗略排序，简单地运算（运算相对轻量）
- 精排：精细排序，确定固定排位

分表学习建议

1）mycat、sharding sphere 框架

2）一致性 hash 算法

队伍操作权限控制

权限整理

加入队伍： 仅非队伍创建人、且未加入队伍的人可见

更新队伍：仅创建人可见

解散队伍：仅创建人可见

退出队伍：创建人不可见，仅已加入队伍的人可见

权限控制

仅加入队伍和创建队伍的人能看到队伍操作按钮（listTeam 接口要能获取我加入的队伍状态）

方案 1：前端查询我加入了哪些队伍列表，然后判断每个队伍 id 是否在列表中（前端要多发一次请求）

方案 2：在后端去做上述事情（推荐）

解决：使用 router.beforeEach，根据要跳转页面的 url 路径 匹配 config/routes 配置的 title 字段。

第 14 集

在线回放：https://meeting.tencent.com/v2/cloud-record/share?id=2ddda24d-72d6-4817-8c91-359fa3ddd0f6&from=3 <https://meeting.tencent.com/v2/cloud-record/share?from=3&id=2ddda24d-72d6-4817-8c91-359fa3ddd0f6>

访问密码：jz6r

主要内容：

1. 优化前端、完善部分功能
2. 免备案方式上线前后端

项目优化

1）前端导航栏死【标题】问题

2）强制登录，自动跳转到登录页

解决：axios 全局配置响应拦截、并且添加重定向

3）区分公开和加密房间；加入有密码的房间，要指定密码

4）展示已加入队伍人数

5）并发时，重复加入队伍的问题

解决：加锁、分布式锁

上线

先区分多环境：前端区分开发和线上接口，后端 prod 改为用线上公网可访问的数据库

免备案上线方案

前端：Vercel（免费）https://vercel.com/ <https://vercel.com/>

后端：微信云托管（部署容器的平台，付费）https://cloud.weixin.qq.com/cloudrun/service <https://cloud.weixin.qq.com/cloudrun/service>

如何改造项目为小程序?

cordova、跨端开发框架 taro、uniapp 等