伙伴匹配系统面试题

后端

请介绍一下你在项目中使用的 Redis，它有哪些优势，为什么选择使用 Redis 实现分布式 Session?

> 前半句背诵类题目，后半句主观回答

Redis 是一个开源的、基于内存的 K / V 存储中间件。由于基于内存，其读写性能非常高，很适用于缓存。此外，Redis 支持多种数据结构、各类编程语言的客户端、支持持久数据，其生态也非常广泛。

本项目中，我使用 Redis 分布式 Session 来代替 Tomcat 本地的 Session 存储，能够在分布式多机场景下保证获取登录用户信息的一致性。用 Redis 实现分布式 Session 的优点是非常简单方便，只需要引入 Redis 和 spring-session-data-redis 依赖，然后在配置文件中指定 Redis 的地址和 session 的 store-type 为 redis，即可自动生效，不用自己额外编码。

你在用户登录功能中提到使用 Hash 代替 String 存储用户信息，这样的做法有什么好处? 在实际应用中，Hash 与 String 存储方式有哪些区别?

> 主观回答

Redis 的 Hash 结构采用 key / value 键值对的形式存储数据，使用 Redis 的 Hash 来存储用户信息后，能够很方便地对用户每个属性进行独立的更新和查询操作，而不是更新和返回整个 JSON 字符串，性能会更高。

举个例子，你想要获取用户的昵称（就 4 个字符串），但是用户的简介有 100 KB 的大小。如果用 Hash 结构，可以只获取昵称，网络传输的内容大小就很小；而如果用 String 结构整体存储，网络传输数据时会把所有的用户信息都返回出来，增加传输开销。

此外，相比于直接在 Spring Boot 中使用 String 类型存储用户信息，使用 Hash 结构不用额外存储序列化对象信息，可以一定程度上节省内存。

请解释一下 Java 8 Stream API 和 Lambda 表达式的作用，以及在项目中如何应用它们来简化集合处理?

> 前半句背诵，后半句主观回答

Stream API 和 Lambda 表达式是 Java 8 提供的语法糖，它们的作用是使集合处理更加简洁、易读和高效。

Lambda 表达式是一种匿名函数，允许你以更紧凑的方式传递代码块，简化代码的编写，比如：

```
// 使用 Lambda 表达式过滤以 "A" 开头的名字
List<String> filteredNames = names.stream()
    .filter(name -> name.startsWith("A"))
    .collect(Collectors.toList());
```

Stream API 是一种流式操作集合的方法，提供了丰富的集合处理操作，比如过滤、映射、排序等，还支持延迟加载和并行处理，简化代码、并提高编码效率。

上述代码示例同样也是 Stream API 的应用。

在本项目中，匹配相似用户时，将存储用户和匹配度的 List 转化为 Stream 流，用 sorted 方法进行编辑距离由小到大排序，用 limit 方法取流的前 N 项，用 collect 终结操作将处理好的流转化为集合 List。还有使用 Stream API 的 map 方法对用户列表中的每个用户信息进行脱敏、使用 ParallelStream 实现并发流等。

你提到使用 Easy Excel 进行批量导入数据库，能否介绍一下 Easy Excel 的使用方法和优势?

> 背诵类题目，但是最好实践过

项目中，我使用 Easy Excel 从 Excel 表格中批量导入用户信息到 MySQL，之所以选择 Easy Excel，不仅是因为它简单易用、并且性能优越，更多的是它能够节约内存、解决大文件内存溢出问题。比如一个 3M 的 excel 用 POI sax 解析依然需要 100M 左右内存，改用 easyexcel 可以降低到几 M，并且再大的 excel 也不会出现内存溢出。

在项目中，对于数据量小的文件，我采用同步模式一次性获取所有表格数据并存储到 List 中；对于数据量大的文件，我采用自定义 Listener 的方式异步逐行读取 Excel 并分批插入到数据库中。

你是如何自定义线程池的？如何合理设置线程池的参数?

> 背诵类题目，但是最好实践过

项目中，我使用 ThreadPoolExecutor 实现灵活的自定义线程池，并通过 ArrayBlockingQueue 存放任务。针对每类不同的业务，我分别定义不同的线程池，让它们互不影响。

示例代码：

```
ExecutorService executorService = new ThreadPoolExecutor(40, 1000, 10000, Time
```

自定义线程池参数如下：

1. 核心线程数（corePoolSize）：线程池中一直保持活动的线程数。可以使用corePoolSize方法来设置。一般情况下，可以根据系统的资源情况和任务的特性来设置合适的值。

2．最大线程数（maximumPoolSize）：线程池中允许存在的最大线程数。可以使用maximumPoolSize方法来设置。如果所有线程都处于活动状态，而此时又有新的任务提交，线程池会创建新的线程，直到达到最大线程数。

3．空闲线程存活时间（keepAliveTime）：当线程池中的线程数量超过核心线程数时，如果这些线程在一定时间内没有执行任务，则这些线程会被销毁。可以使用keepAliveTime和TimeUnit方法来设置。

4．阻塞队列（workQueue）：用于存放等待执行的任务的阻塞队列。可以根据任务的特性选择不同类型的队列，如LinkedBlockingQueue、ArrayBlockingQueue等。默认情况下，使用无界阻塞队列，即LinkedBlockingQueue，但也可以根据需要设置有界队列。

5．线程工厂（threadFactory）：用于创建线程的工厂。可以通过实现ThreadFactory接口自定义线程的创建逻辑。

6．拒绝策略（rejectedExecutionHandler）：当线程池无法接受新的任务时，会根据设置的拒绝策略进行处理。常见的拒绝策略有AbortPolicy、DiscardPolicy、DiscardOldestPolicy和CallerRunsPolicy。

我是根据任务的类型以及消耗资源的情况来调整线程池的参数。比如针对更消耗 CPU 资源的计算密集型任务，我会将核心线程数设置为和 CPU 核心数相同，充分利用系统资源；针对更消耗网络等 IO 的 IO 密集型任务，我会将核心线程数设置得更大，比如 CPU 核心数的 2 - 4 倍，能够增加并发度、并且提高 CPU 的利用率。

有一个经验值公式，其中 N 为 CPU 核心数：CPU 密集型任务，核心线程数设置为 N（或 N + 1）；IO 密集型任务，核心线程数设置为 2N。

你是如何测试批量导入数据库的性能的？用了哪些工具或方法？

> 主观回答

首先我编写了一个单元测试类，使得测试代码不影响主业务代码。然后通过 Spring 的 StopWatch 实现批量导入数据库的耗时统计。

在测试过程中，我不断地调整数据批量插入的组数、每组数据量、以及并发线程数，尝试找到一个性能最佳的配置。

注意，这里不是对接口的压力测试，没有必要使用 JMeter 压测工具。

你在使用 Redis 缓存高频访问用户信息时提到了自定义序列化器，为什么需要自定义序列化器，以及自定义序列化器的实现方式？

> 主观回答

由于 Spring Boot Data Redis 默认使用 JDK 序列化器，会将存储到 Redis 的键值对转化为字节数组，不利于在 Redis 可视化工具中阅读、并且不利于跨语言兼容，所以需要指定序列化器。

所以我通过新建 RedisTemplateConfig 配置类来创建自定义的 RedisTemplate Bean，并且通过 redisTemplate.setKeySerializer(RedisSerializer.string()) 指定了 Redis Key 的序列化方式。

示例代码如下：

```
@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;
    }
}
```

你在项目中是如何实现 Redis 缓存的？选用了哪种 Redis 数据结构？

> 主观回答

我的项目中，使用 Redis 缓存实现了登录用户信息的存储、主页推荐用户列表的存储。

具体的实现方式：

- 对于登录用户信息的存储，直接使用 spring-session-data-redis 依赖开启对 Redis 分布式 Session 的支持。
- 对于主页用户推荐列表的存储，我使用 Spring Data Redis 整合 Redis，并通过 RedisTemplate 来操作 Redis，根据业务类型设计了缓存 key 的规则，选用 string 数据结构来存储推荐用户列表。

使用 Redis 缓存时，有哪些可能出现的常见问题？你又是如何解决的？

> 背诵类题目

建议先列举使用缓存可能出现的常见问题，然后再挑其中一点举例。

使用 Redis 缓存可能的常见问题：

1）缓存击穿： 缓存击穿指的是某个热门的缓存键在过期后，同时有大量并发请求到达，导致所有请求都穿透缓存直接访问数据库，造成数据库压力激增。解决方法包括：

- 使用互斥锁来保护缓存访问，只允许一个线程重新生成缓存。
- 针对缓存失效时的并发请求使用分布式锁，确保只有一个线程重新生成缓存。

2）缓存雪崩： 缓存雪崩指的是大量缓存键在相同时间失效，导致大量请求落到数据库上，造成数据库压力激增。解决方法包括：

- 为缓存键设置不同的失效时间，使失效时间分散。
- 使用热点数据预热，提前加载热门数据到缓存。

3）缓存过期问题： 缓存中的数据过期后可能会导致数据不一致或数据不可用。解决方法包括：

- 设置合理的缓存失效时间，避免缓存数据长时间不更新。
- 使用缓存的时候检查数据是否过期，如果过期则重新生成缓存。

4）缓存内存问题： 如果缓存数据量很大，可能会导致内存占用过多。解决方法包括：

- 设置合理的内存限制，避免缓存数据过多。
- 使用LRU（Least Recently Used）策略或淘汰算法来淘汰不常用的缓存数据。

5）缓存数据一致性问题： 缓存数据和数据库数据不一致。解决方法包括：

- 使用缓存更新策略，当数据库数据发生变化时，及时更新缓存。
- 使用双写策略，即同时更新数据库和缓存，确保数据一致性。

6）缓存安全问题： 某些敏感数据可能不应该被缓存，如果被缓存可能引发安全问题。解决方法包括：

- 避免缓存敏感数据。
- 使用加密或其他安全措施来保护缓存数据。

7）缓存监控和调优问题： 缓存需要监控和调优，以确保性能和稳定性。解决方法包括：

- 使用监控工具来监测缓存的命中率、内存占用等性能指标。
- 定期调整缓存配置，优化性能。

在本项目中，我通过给不同的缓存设置不同的随机过期时间（N + n）来解决缓存雪崩问题。

在解决首页加载过慢的问题中，你使用了 Spring Scheduler 定时任务和分布式锁，请解释一下定时任务的执行原理和此处分布式锁的作用。

> 主观回答

项目使用 Spring Scheduler 实现定时任务，我将每个任务定义为独立的 Job 类，并且给实际需要定时执行的方法增加 @Scheduled 注解来开启定时任务。

在 @Scheduled 注解中，我使用 crontab 表达式来定义执行定时任务的时间周期，Spring Scheduler 会根据这些定义，在时机到达时开启独立的线程来执行任务。

在分布式场景下，可能有多个服务器实例同时执行同一个定时任务，导致并发问题或重复执行，所以用分布式锁来保证定时任务执行的唯一性。当定时任务要执行时，先去抢锁，只有抢到锁的服务器实例才会执行定时任务。

你在项目中使用 Redisson 分布式锁解决了接口幂等性的问题，请简要介绍一下 Redisson 分布式锁的使用场景和实现原理。

> 背诵类题目

Redisson 是一个基于 Redis 的数据网格，它提供了开箱即用的分布式锁功能，用于解决分布式环境下的并发控制问题。

比如在项目中，使用 Redisson 分布式锁保证接口幂等性，防止多个用户同时操作或重复提交带来的数据不一致。

Redisson 分布式锁的实现是基于 Redis 的 SETNX 命令和 Lua 脚本，具体的实现原理如下：

1. 获取锁：当客户端请求获取锁时，Redisson 会向 Redis 发送一个 SETNX 命令，尝试将一个特定的键（锁的标识）设置为一个特定的值（客户端标识），并设置锁的超时时间。
2. 争用锁：如果多个客户端同时尝试获取同一个锁，只有一个客户端能够成功设置键的值，其他客户端的 SETNX 命令将失败，它们会继续尝试获取锁。
3. 锁超时：为了防止某个客户端获取锁后发生异常导致锁永远不会被释放，Redisson 设置了锁的超时时间。当锁的超时时间到达后，Redisson 会自动释放锁，允许其他客户端获取锁。
4. 释放锁：当客户端执行完锁保护的操作后，可以主动释放锁，这将删除锁的标识键，或者锁的自动超时也会导致锁的释放。
5. 锁的可重入性：Redisson 支持可重入锁，允许同一客户端多次获取同一个锁，然后多次释放锁。只有所有获取锁的次数都释放后，锁才会被完全释放。
6. 锁的续期：如果一个客户端在持有锁时，锁的超时时间即将到期，Redisson会自动为锁续期，防止锁在操作过程中被自动释放。

编辑距离算法是什么，它在你实现的用户匹配功能中起到了什么作用？请解释一下编辑距离算法的实现原理。

> 背诵类题目

编辑距离算法是一种用于度量两个字符串之间的相似度或差异性的算法，常用于字符串相似度比较、拼写检查等场景。

在用户匹配功能中，我使用编辑距离算法来计算用户输入的搜索关键词与已有用户信息的匹配程度，并按照相似度进行排序，从而实现最相似用户的推荐。

编辑距离算法的实现原理：https://blog.csdn.net/DBC_121/article/details/104198838 <https://blog.csdn.net/DBC_121/article/details/104198838> ，仅做了解即可，不用背诵。

你提到使用优先队列来减少 TOP N 运算过程中的内存占用，能否解释一下优先队列的特点和在项目中的具体应用?

> 背诵类题目

优先队列中的元素按照优先级顺序进行排列，一般优先级越高的元素越早出队。优先队列的插入和删除操作非常高效，并且能够快速访问到优先级最高 / 低的元素，很适用于查找 Top N 元素、任务调度等场景。

在项目中，使用优先队列来存储从数据库中查询出来的 TopN 最相似用户。把相似度作为优先级，淘汰相似度小于当前队列 TopN 的用户，存入相似度大于当前队列 TopN 的用户，将队列的元素个数始终维持在 N 个，从而减少了内存占用。

在项目中，你自主编写了 Dockerfile 来实现自动化镜像构建及容器部署，请介绍一下用 Docker 的优势?

> 背诵类题目

可以把 Docker 镜像想象成应用的安装包，我通过编写 Dockerfile 制作了项目的安装包，开发者可以使用该 Docker 镜像一键快速启动项目，无需手动安装 Java 等依赖项、并且手动输入启动 jar 包的命令，便于分发应用程序、并且提高应用部署效率。

此外，通过给 Docker 镜像打 tag，可以控制应用程序的版本，便于项目的持续发布和回滚。

你在项目中使用 Knife4j 和 Swagger 自动生成后端接口文档，请解释一下 Swagger 的作用，以及在项目中使用 Swagger 的好处。

> 背诵类题目，也可以有主观回答

使用 Swagger 接口文档生成工具后，我不需要在开发完项目后手动编写一套接口文档，而是直接交由系统自动根据 Controller 接口层的代码自动生成文档，大幅节省时间。

使用 Swagger 生成的接口文档不仅能够分组查看请求参数和响应，还支持灵活的在线调试，可以直接通过界面发送请求来测试接口，提高开发调试效率。

此外，引入 Swagger 后，可以得到基于 OpenAPI 规范的接口定义 JSON，可以配合第三方工具来根据 JSON 自动生成前端请求代码、自动生成客户端调用 SDK 等。

前端

项目前端使用了 Vant UI 组件库，请列举几个你用到的 Vant UI 组件并介绍它们的用途?

> 主观回答

我参照官方文档使用 Vant UI 组件库，项目中用到了：

- Toast 轻提示组件：用于给用户的操作反馈，比如弹出操作成功、失败的提示
- Cell 单元格组件：逐行展示用户的个人信息
- List 列表组件：展示多条用户信息卡片
- Tag 标签组件：美化展示用户的标签信息

如何基于 Vue Router 实现动态切换导航栏标题？请详细描述一下实现的过程。

> 主观回答

首先在 route.ts 路由配置文件中，给每个路由增加标题属性。

然后在项目全局通用布局组件（BasicLayout）中使用路由守卫 beforeEach 函数来监听页面路由的变化。当页面路由改变时，会从已定义的路由配置列表中根据路由地址找到对应的路由标题，并更改导航栏的 state 切换标题。

什么是前端异步编程？请介绍一下 Promise、async/await 在 JavaScript 中的作用及用法。

> 背诵类题目

前端异步编程是一种编程模式，用于处理那些可能会花费较长时间来执行的操作，如网络请求、文件读写、定时任务等。在异步编程中，代码不会阻塞（即不会等待操作完成），而是通过回调、Promise、async/await 等机制来处理异步操作的结果。

Promise 是一种用于处理异步操作的对象，它可以表示一个操作的最终完成（或失败），并提供了一种更可控的方式来处理异步操作。

创建Promise：使用 `new Promise()` 构造函数创建一个Promise对象，它接受一个执行函数，该函数包含异步操作的代码，并具有 `resolve` 和 `reject` 两个参数，分别表示操作成功和失败时的处理。

```
const myPromise = new Promise((resolve, reject) => {
  // 异步操作
  if (/* 操作成功 */) {
    resolve(result); // 成功时调用resolve
  } else {
    reject(error); // 失败时调用reject
  }
});
```

8230字 处理Promise：使用 `.then()` 方法来注册处理成功和失败情况的回调函数。

```
myPromise
  .then(result => {
    // 处理成功
  })
  .catch(error => {
    // 处理失败
  });
```

async/await 是一种语法糖，用于更直观地处理 Promise 的异步操作。

使用 `async` 关键字定义一个函数，这个函数会返回一个 Promise：

```
async function fetchData() {
  // 异步操作
  return result; // 返回结果会被包装成Promise
}
```

在 async 函数内部，可以使用 `await` 关键字等待一个 Promise 完成，并获取其结果。

```
async function fetchData() {
  const result = await somePromise; // 等待Promise完成并获取结果
  // 继续执行后续操作
}
```

async/await 的好处在于它可以让异步代码看起来更像同步代码，使代码可读性更强，并且可以更容易地处理异步操作中的错误。

请介绍一下 Vue 3 的新特性和与 Vue 2 相比有哪些变化?

> 背诵类题目，也可以有主观回答

详细参考官方文档：Vue 3 迁移指南 | Vue 3 迁移指南 <https://v3-migration.vuejs.org/zh/#%E5%80%BC%E5%BE%97%E6%B3%A8%E6%84%8F%E7%9A%84%E6%96%B0%E7%89%B9%E6%80%A7>

1）更快的渲染性能：Vue 3 引入了新的响应式系统（Proxy-based），相比Vue 2的Object.defineProperty，提供了更高效的数据监听和更新机制，从而提高了渲染性能。

2）Composition API：Composition API 是 Vue 3的核心特性之一，它允许开发者更灵活地组织和重用组件逻辑。它将组件的逻辑拆分为可复用的函数式组合，并提供了setup()函数来配置组件。

3）Teleport：Vue 3引入了Teleport组件，可以轻松将内容渲染到DOM中的不同位置，这在处理模态框、对话框等场景时非常有用。

4）Fragments：Vue 3支持Fragments，允许组件返回多个根节点，而无需包裹额外的HTML元素。

需要注意的是，虽然 Vue 3 引入了许多新特性，但它仍然保持了 Vue 2 的核心理念和语法，因此 Vue 2 的开发者可以相对容易地迁移到 Vue 3，并逐步采用新的特性和优化。

移动端网站和响应式网站有什么区别？你在项目中是如何处理移动端适配的？

> 前半句背诵类题目，后半句主观回答

两者最主要的区别是在于设计目标和目标用户的不同。

移动端网站是专门为移动设备（如智能手机和平板电脑）设计和优化的网站。它的设计目标是提供在小屏幕上良好的用户体验，通常包括更简化的布局和功能，以适应移动设备的特点。

响应式网站是一种网站设计方法，旨在使网站能够在不同大小和类型的设备上自动适应，包括桌面电脑、平板电脑和移动设备。响应式网站的设计目标是提供一致的用户体验，无论用户使用哪种设备访问网站。

本项目属于移动端网站，暂时不考虑 PC 端用户，所以直接使用 Vant UI 移动端组件库实现，该组件库提供了非常好的移动端适配和交互体验。

Vue.js 中的组件通信方式有哪些？

> 背诵类题目，也可以有主观回答

组件通信的目的是为了在不同组件之间共享数据、触发事件、交互调用，主要是以下几种常用的方式：

1. Props（父传子）：通过 `props` 属性，父组件可以向子组件传递数据。子组件通过 `props` 接收父组件传递的数据。
2. 自定义事件（子传父）：子组件可以通过 `$emit` 方法触发自定义事件，然后父组件通过在子组件上使用 `@` 或 `v-on` 指令监听这些事件，以接收子组件的消息。
3. 事件总线（兄弟组件）：通过创建一个事件总线实例，不同组件可以通过事件总线来进行通信。这是一种适用于兄弟组件之间的通信方式。
4. Vuex（状态管理）：Vuex是Vue.js的官方状态管理库，用于管理全局状态。不同组件可以通过Vuex来共享和管理应用程序的状态。
5. $refs（父子组件）：父组件可以通过 `ref` 属性引用子组件，并直接访问子组件的属性和方法。这种方式主要用于父组件控制子组件的行为。
6. 插槽（分发内容）：插槽允许父组件将内容分发到子组件的特定位置，以实现更灵活的组件复用和布局控制。

在你的项目中，为什么选择了全局通用的 Layout 组件而不是局部组件？

> 主观回答

因为对于我的项目来说，几乎所有页面都具有顶部导航栏、底部 Tab 栏等公共组件，所以我选择使用全局通用的 Layout 组件来统一页面的结构和风格，便于维护并且减少重复代码。

你是如何初始化项目的？是否有使用脚手架？

> 主观回答

这个项目使用了 Vite 作为打包工具，并且使用 Vite 官方提供的脚手架来生成初始化项目，非常方便。

你在前端项目中有做过性能优化吗？请分享一些常见的性能优化措施和实践经验。

> 背诵类题目，但尽量多包含主观回答

优化网页的加载性能和渲染性能是前端开发中的关键任务之一，它可以提高用户体验并降低网站的跳出率。以下是一些常见的优化策略和技巧：

1）优化图片：

- 使用适当的图片格式：选择合适的图片格式（如WebP、JPEG、PNG）以减小文件大小。
- 压缩图片：使用压缩工具或在线服务来减小图片文件的大小。
- 使用响应式图片：为不同屏幕大小提供不同尺寸的图片，以避免加载过大的图片。

2）使用CDN（内容分发网络）：

- 将静态资源（如CSS、JavaScript、图片等）托管到CDN上，加速资源的加载。
- CDN可以将内容缓存到全球各地的服务器上，减少距离造成的延迟。

3）懒加载和预加载：

- 使用懒加载技术延迟加载图片和其他资源，只有当用户滚动到可见区域时才加载。
- 使用 `<link rel="preload">` 标签预加载关键资源，以提前下载可能需要的资源。

4）压缩和合并文件：

- 压缩JavaScript和CSS文件，减小文件大小。
- 合并多个CSS或JavaScript文件为一个，减少HTTP请求次数。

本项目使用 Vite 打包工具，会自动对代码文件进行打包合并。

5）使用浏览器缓存：

- 使用 HTTP缓存头（如 `Cache-Control` 和 `ETag` ）来允许浏览器缓存资源。
- 使用 Service Worker 来创建离线应用程序，使用户在离线状态下也能访问网站。

6）减少重排和重绘：

- 避免频繁修改DOM，可以使用文档片段（DocumentFragment）进行批量操作。
- 使用CSS动画而不是JavaScript来实现动画效果，以减少重排和重绘。

7）代码分割（Code Splitting）：

- 将应用程序拆分为多个模块或块，按需加载模块，减小初始加载时间。
- 使用工具如 Webpack 的 `import()` 语法来实现代码分割。

8）使用异步加载：

- 将非关键的JavaScript代码标记为异步加载，以防止它们阻塞页面加载。
- 使用 `<script async>` 标签来异步加载脚本。

9）前端框架和库的优化：使用轻量级的前端框架或库，以减少不必要的代码。

本项目没有使用大而全的第三方框架，而是使用 Vite 脚手架 + Vant UI 组件库，支持按需加载组件，还是比较轻量的。

10）监控和性能分析：

- 使用工具如 Google PageSpeed Insights、Lighthouse 等来检查和分析性能问题。
- 使用性能监控工具来实时追踪网站的性能指标，以便及时优化。

注意，性能优化是一个持续的过程，需要不断地监测、测试和调整，以确保网站保持高性能。

你在项目中使用了 Vue Router 全局路由守卫，请解释一下路由守卫的概念和用法，并介绍一下它在你项目中的实际应用场景?

> 主观题目

建议参考官方文档：https://router.vuejs.org/zh/guide/advanced/navigation-guards.html <https://router.vuejs.org/zh/guide/advanced/navigation-guards.html>

路由守卫是 Vue Router 提供的一种机制，用于在导航过程中对路由进行拦截和控制，以执行一些特定的操作或验证逻辑。路由守卫允许开发者在路由切换前、切换后或在路由被确认（导航完成）时执行自定义代码，从而实现一些常见的任务，如身份验证、权限控制、页面切换动画等。

在项目中，我通过 router.beforeEach 定义全局前置守卫，根据当前 url 地址和路由的配置动态改变全局导航栏的标题。

通用

请介绍一下本项目的完整业务流程?

首先用户需要登录注册，然后更新个人信息。

接下来系统主页会自动推荐相似用户，用户也可以根据标签自由搜索用户。

用户还可以自主创建房间发起组队，或者加入（退出）其他队伍，从而找到志同道合的伙伴共同完成目标。

在开发过程中，你遇到过比较复杂的技术问题或挑战吗？如果有，请谈谈你是如何解决这些问题的？

可以从以上任意一道主观的面试题出发去讲，比如你在线上多机部署项目时发现定时任务重复执行，然后通过 Redisson 分布式锁解决。