****

课程设计报告书

**题目：网络应用开发**

**学 院 计算机科学与工程学院**

**专 业 网络工程**

**学生姓名 黄梓恩**

**学生学号 202130440705**

**指导教师 布 社辉**

**课程编号**

**课程学分 1.0**

**提交日期 2024年5月15日**

**代码托管地址**

**https://github.com/SonderHZE/MyGpt2.0/tree/master**

**部署在线网站**

**http://47.99.133.237:8080/**

**测试账号口令**

**用户名：scut 密码：scut**

华南理工大学

**需求分析报告**

|  |
| --- |
| **概述** |
| 【基本要求】  根据选定的题目，任务清单，以及要达到的技术指标，确定满足开发需求所必须实现的功能，并选定整体技术路线，评估完成所需功能所需要的时间，指定整体开发规划。  【基本内容】  Step 1 : 根据所选择的应用领域，请认真分析业务的场景以及相关的业务流程，找出当前业务之中所存在的痛点需要解决的课题；  Step 2 : 根据所找出的痛点，提出可能的解决方案；  Step 3 :对所列出的解决方案进行可性能分析，提出解决方案；  Step 4 : 根据最优的解决方案，列出具体的实施方案以及所要达到的技术指标，并确定计划实施进度计划。 |
| **报告内容**   * **业务场景分析与痛点识别**   在AI聊天平台，如通义千问、文心一言的应用领域中，业务主要围绕着提供智能化、自然语言驱动的交互体验展开。用户通过文本输入与AI进行交流，获取信息、解答问题或进行娱乐互动。然而，当前业务流程中存在两个显著痛点：   * + **个性化与定制化需求未得到满足：**尽管这些平台提供了高质量的对话体验，但用户界面和交互模式相对固定，不能根据个人偏好深度定制聊天机器人的回复风格、领域知识或专业水平。这限制了用户的个性化体验和某些专业场景的应用潜力。   + **底层参数调优受限：**对于有高级需求的用户或开发者来说，平台隐藏了如top\_k、temperature等底层参数设置，限制了他们根据特定场景优化模型响应的能力，影响了模型的适应性和创造性输出。   + **聊天记录管理不便：**用户难以高效回顾和管理与不同模型的聊天记录，影响了连贯对话体验和信息查找效率。 * **可行解决方案的提出**   + **多模型的实时切换**   **模型集成：**整合多个主流大语言模型，如通义千问、文心一言等，让用户在聊天界面轻松选择或即时切换模型。  **上下文传递**：实现模型间上下文的无缝传递，确保用户在切换模型时，对话连贯性不受影响。   * + **深化个性化定制功能**   **增强设置选项：**扩展个性化设置菜单，包括多维度的语气风格调控及模型回复偏好的学习。   * **可行性分析与技术策略**   + **技术可行性与架构设计** * **多模型实时切换与适配器模式：**采用适配器设计模式，为不同模型设计接口适配层，统一对外提供标准化的API。这种方式可以有效封装模型差异，降低前端与后端的耦合度，实现模型的灵活切换。技术上，利用Spring Boot等框架快速构建服务架构，便于模型服务的管理和扩展。 * **深化个性化定制与动态界面：**Vue3配合Element-UI框架，不仅能够快速构建出美观且响应式的用户界面，还能够充分利用Vue3的Composition API特性，动态调整UI组件和样式，实现用户设置的即时反馈。此外，通过Vuex管理全局状态，使得用户个性化设置的同步与应用变得简单高效。 * **聊天记录与设置缓存策略：**   + **Redis缓存：**利用Redis的高性能特点，缓存聊天记录和用户个性化设置，支持快速读取和写入。特别是对于频繁查询的最近聊天记录，可以极大提高响应速度。   + **MySQL持久化存储：**作为数据持久化存储，MySQL负责存储所有的聊天记录和用户设置信息，确保数据的可靠性和完整性。通过定时任务或事务机制，保持Redis与MySQL数据的一致性。 * **单点登录**   + **单点登录：**借助Redis的会话共享机制，实现跨域的单点登录。用户认证信息在首次登录后存储于Redis中，后续访问其他子系统时直接验证Redis中的凭证，无需重复登录。 * **流量削峰策略**：   + **Redis旁路缓存**：利用Redis高速缓存特性，作为数据库访问的前置缓冲区，可以迅速响应大部分读请求，尤其是在高并发场景下，有效减轻数据库的压力，避免数据库成为瓶颈。   + **分布式缓存集群**：部署Redis集群，通过分片技术分散缓存数据，不仅可以提升缓存容量，还能增加系统的可用性和伸缩性，确保在高负载情况下仍能稳定服务。 * **缓存一致性策略**：   + **主动过期通知与备份键机制**：当Redis中的键接近过期时，可以触发一个事件通知机制，后台任务监听这些事件，及时将数据从Redis同步回MySQL数据库。同时，采用备份键策略，在主键过期前，预先复制一份数据到备份键，确保数据不因主键过期而立即丢失，实现平滑的数据迁移。   + **数据库异步写入**：对于写操作，可以先写入Redis，然后异步批量写入MySQL，减少对数据库的直接写压力，提高写入效率，同时通过消息队列保证最终一致性。 * **容灾与高可用设计**：   + **Redis集群与主备切换**：构建Redis哨兵或Cluster集群，实现主从复制和自动故障转移。当主节点发生故障时，系统能迅速切换到备用节点，确保服务不间断，防止数据丢失。   + **数据备份与恢复策略**：定期对Redis数据进行备份，并制定数据恢复计划，确保在极端情况下能够快速恢复服务，减少数据损失。  1. **具体实施方案** 2. **项目启动与需求分析阶段（1周）**    * 深入分析业务需求，细化功能模块，制定详细的功能需求文档。 3. **系统设计与架构搭建阶段（2周）**    * 设计系统架构图，确定技术栈，包括前端Vue3+Element-UI、后端Spring Boot、数据库MySQL+Redis。    * 开发模型适配层，确保各AI模型能够统一接入系统。    * 设计数据库表结构，制定缓存与持久化策略。 4. **开发与集成阶段（3-7周）**    * **前端开发**：构建用户界面，实现动态设置调整、聊天记录展示等功能。使用Vuex管理状态，确保UI与数据同步。    * **后端开发**：实现用户认证、多模型切换逻辑、上下文传递机制。开发缓存与数据库交互逻辑，确保数据一致性。    * **模型集成与测试**：逐一集成AI模型，进行单元测试、集成测试，确保各模型与系统稳定交互。 5. **性能优化与安全加固阶段（7-9周）**    * 优化Redis与MySQL的交互逻辑，实施流量削峰策略，提升系统承载能力。 |
| **小结** |
|  |
| **指导教师评语及成绩** |
| 评语：  成绩：    指导教师签名：                                                  批阅日期： |

**系统设计报告**

|  |
| --- |
| **概述** |
| 【基本要求】  根据所必须实现的功能，设计系统整体，以及所需的模块，确定详细的开发计划。  【基本内容】  根据 《需求分析报告》所列出的具体实施方案以及计划进度， 提出整体框架设计方案；  根据整体框架设计方案，列出各个模块所需要实现的功能以及所需要达到的技术指标；  各个模块的具体实施方案；  重点考察对应大数据的方案；  必须要严格遵守软件工程的规范。  假设你作为PM(Project Manager)，在项目实施过程之中及时处理各种各样的问题，因此在项目开发的规划设计阶段中需要评估可能出现的各种风险。针对该项目，结合自己在完成实验时的经验请分析并评估可能出现的问题，以及针对这些问题所设定的可行的预案，并解释说明其预案的可行性。 |
| **报告内容**   1. **系统架构**    1. **系统架构概述**   本系统采纳前后端分离的架构设计，其中前端专注于用户交互与视觉呈现，后端则承担业务逻辑处理、数据管理及AI模型的集成重任。在技术选型上，我们利用Redis作为缓存和消息中间件，以提升响应速度和处理能力，同时MySQL作为数据持久化存储，确保数据的可靠性和完整性。后端架构遵循经典的MVC模式，并融合微服务思想，以确保系统的高度扩展性与维护便利性，为未来服务的多样化发展奠定坚实基础。   * 1. **前端设计**   前端采用Vue3与Element-UI框架，结合Vue3的Composition API，构建出既美观又响应式的用户界面，同时提升代码的模块化与复用性。Vue Router实现页面跳转逻辑管理，Vuex确保状态跨组件的统一管理与共享。Axios作为HTTP请求工具，封装网络交互，统一处理错误与异常，简化前后端通讯流程。   * 1. **后端架构**   后端架构深化为以下模块**：**  **• 模型层(Model)：**负责数据模型的定义与数据库交互，包括实体对象映射与DAO层，确保与MySQL的无缝对接。  **• 视图层(View)：**前端展示，后端提供JSON格式数据接口，由前端解析展示。  **• 控制器层(Controller)：**接收HTTP请求，执行业务逻辑，调度服务层处理，响应结果。  **• 服务层(Service)：**封装复杂业务逻辑，调用模型层处理数据，集成AI模型服务，通过适配器模式实现模型调用的灵活切换。  **• 辅助类：**引入ChatUtils、CookieUtil、HashUtil、JWTUtil等工具类，为系统提供实用功能支持，如处理聊天数据、管理Cookies、实现安全认证等。   * 1. **中间件**   **• Redis：**  **缓存策略：**利用Redis缓存热点数据，如聊天记录，采用LRU/LFU策略自动淘汰机制优化存储。  **会话管理：**通过Redis存储用户会话信息，实现单点登录，跨域共享。  **消息队列：采**用Redis Pub/Sub 或 Stream 实现异步处理，如异步写库，提升性能。  **•MySQL：**  **持久化存储：**负责持久存储聊天记录、用户配置等。  **数据一致性：**利用事务确保操作原子性，通过定时任务或触发机制维持Redis与MySQL数据一致。    图1. 系统架构图   1. **各个模块所需实现的功能及技术指标**    1. **用户注册和登录模块**   **功能：**实现手机号注册、密码设置与验证。  **技术指标：**密码加密存储（如使用bcrypt）；JWT令牌实现无状态认证。   * 1. **AI聊天功能模块**   **功能：**集成至少两种大语言模型，提供API接口，实现用户输入解析、模型调用、回复生成与返回。  **技术指标：**支持多模型切换；延迟低于500ms；准确率≥90%；支持上下文记忆，维持连贯对话。   * 1. **个性化设置模块**   **功能：**允许用户自定义system、tempreture等参数，以及模型的自由切换。  **技术指标：**实时预览更改效果；存储个性化设置至数据库；   * 1. **聊天历史记录模块**   **功能：**保存用户的聊天记录，支持按对话、时间筛选查看。  **技术指标：**高效查询，加载历史记录不超过2秒；数据加密存储；提供清除聊天记录功能。   1. **模块的具体实施方案**    1. **用户认证模块实施方案**   **前端实现：**   * + 设计简洁直观的注册和登录界面，引导用户输入手机号进行验证。   + 对密码进行MD5加密处理，防止传输过程中的密码泄露。   + 登录时，通过AJAX异步请求后端API，提交加密后的用户凭证，获取JWT令牌。   **后端实现：**   * + 对用户密码进行盐值加密处理，并存储到数据库。   + 开发API接口，接收前端传来的手机号和/密码，验证后发放JWT令牌。   + 集成JWTUTils，对所有需要权限的路由进行令牌验证。   **数据库设计：**   * + 创建用户表，包含手机号、密码哈希值等字段。   1. **AI聊天功能模块实施方案**   **模型集成：**   * + 筛选并集成至少两种大型语言模型，如通过API接入阿里云、百度或Hugging Face等平台提供的模型服务。   + 开发模型切换逻辑，允许根据用户选择动态调用不同的模型API。   **API设计与实现：**   * + 设计RESTful API，接收用户消息，根据预设逻辑选择模型，调用模型接口并处理返回的原始数据。   + 实现上下文管理机制，每次交互时携带前几次对话信息，以保持对话连贯性。   **性能优化：**   * + 利用缓存技术减少重复调用，提高响应速度。   + 异步处理模型调用，减少用户等待时间。   1. **个性化设置模块实施方案**   **UI设计：**   * + 设计易于操作的设置面板，展示可调整的参数选项，如对话风格、模型温度等。   + 实现即时预览窗口，展示设置变化后的模拟对话效果。   **前端交互：**   * + 采用Vue和elementUI框架，实现动态表单，实时更新预览。   **后端存储：**   * + 开发API接口，接收前端发送的个性化设置数据，存储到用户对应的数据库记录中。   + 使用Redis等内存数据库缓存频繁访问的个性化设置，加快读取速度。   1. **数据库设计**      1. **用户表 (users)** * userID: 自动递增的整数，作为用户的唯一标识符，也是表的主键。 * userName: 用户名，长度限制为255个字符，不允许为空，且必须唯一。 * mobilePhone: 手机号码，长度限制为255个字符，不允许为空，且必须唯一，确保每个手机号只能对应一个用户账户。 * password: 密码字段，存储用户密码，长度限制为255个字符，不能为空。 * defaultModel: 默认使用的AI模型，类型为文本，允许为空，用户可以设置默认使用的AI交互模型。 * defaultPrompt: 默认的提示词，长度限制为255个字符，可为空，用于定制每次对话的初始引导信息。   此表通过userName和mobilePhone的唯一约束保证用户名和手机号的唯一性，避免重复注册。   * + 1. **聊天记录表 (chats)** * chatID: 自动递增的整数，作为每条聊天记录的唯一标识符，也是表的主键。 * userID: 外键，关联到users表的userID，表示这条聊天记录属于哪个用户。 * chatTitle: 聊天标题，长度限制为255个字符，不能为空，用于区分或标记不同的聊天主题。 * time: 聊天记录的时间戳，默认为创建时的当前时间，可以用来排序或查询聊天的历史记录。 * chatModel: 使用的AI模型名称，长度限制为255个字符，可为空，记录这次对话采用的AI模型。   通过外键约束chats\_ibfk\_1，确保chats表中的userID与users表的userID相对应，保持数据的一致性。   * + 1. **聊天信息详情表 (chatinfo)** * chatID: 作为主键，与chats表的chatID关联，表示这是某次聊天的详细信息。 * messageList: 存储聊天消息的列表，类型为文本，可以是序列化后的数据或文本格式，用于保存整个聊天的对话内容。   通过外键约束chatinfo\_ibfk\_1，保证chatinfo表中的chatID与chats表的chatID匹配，关联聊天记录与具体聊天内容。  此外，在chats表上创建了一个名为userID的索引，这有助于加速基于用户ID的查询操作，如快速查找某个用户的全部聊天记录。 |
| **小结** |
| **系统架构设计亮点**   1. **前后端分离架构**：采用Vue3与Element-UI构建的前端界面，与后端基于MVC模式及微服务架构的设计，有效提升了开发效率与系统的可维护性。前端的现代化框架与组件库保证了用户界面的美观和交互体验，而后端的分层设计则确保了业务逻辑的清晰与模块间的低耦合。 2. **高性能中间件集成**：通过集成Redis作为缓存与消息中间件，系统实现了热点数据的快速访问与异步处理机制，显著提高了响应速度和并发处理能力。同时，利用Redis实现单点登录和会话管理，增强了用户体验的一致性。 3. **数据存储与一致性策略**：MySQL作为持久化存储，确保了数据的安全与完整性，而通过事务处理和同步机制维护Redis与MySQL间的数据一致性，保障了系统的可靠性。   **关键模块实施成效**   1. **用户认证模块**的成功实施，展示了MD5加盐加密与JWT令牌机制的有效结合，确保了用户凭证的安全传输与验证，同时也提供了便捷的无状态认证体验。 2. **AI聊天功能模块**的集成与优化，不仅实现了多模型的灵活切换，还通过上下文管理保持了对话的连贯性，达到了高准确率与低延迟的技术指标，显著提升了聊天体验。 3. **个性化设置模块**的实现，让用户可以根据个人偏好调整聊天设置，即时预览功能增加了互动性和满意度，而Redis缓存的运用进一步加速了设置读取，体现了用户为中心的设计理念。   **技术挑战与解决方案**   * **安全性挑战**：通过密码加密存储、JWT令牌验证及数据加密传输，有效应对了用户数据保护的挑战。 * **性能优化**：面对模型调用延迟和大量数据处理的压力，采取了缓存策略、异步处理及模型调用适配器模式，显著提升了系统性能。 * **个性化与用户体验**：个性化设置的即时反馈机制与聊天历史的高效管理，解决了提升用户参与度和满意度的难题。   **结论**  本次实验不仅成功构建了一个功能全面、性能优越的AI聊天系统，还深入实践了现代Web开发的最佳实践和技术趋势。通过前后端分离、微服务架构、中间件的有效利用以及对安全性和性能的深度考量，系统展现了高度的灵活性、可扩展性和用户友好性，为未来的功能拓展和技术创新奠定了坚实的基础。 |
| **指导教师评语及成绩** |
| 评语：  成绩：    指导教师签名：                                                  批阅日期： |

**系统实现报告**

|  |
| --- |
| **概述** |
| 【基本要求】  依据开发计划，进行代码实现，并在报告之中列出全部代码，解释说明部署过程步骤。  【基本内容】  对选用的开发环境以及选用的开发工具的说明；  必须要列出所有代码以及配置文件；  必须对在线部署过程进行说明。  假设你作为PM(Project Manager)，针对该项目在开发过程中所遇到的问题以及对应的解决方案，请评价在设计阶段所做的预估和预案的准确性和有效性，并总结在项目实施过程之中所获得的心得体会。 |
| **报告内容**   1. **开发环境与工具说明**    1. **开发环境**   **操作系统：**项目选择Windows系统作为主要的开发和部署平台，以适应团队的技术栈和提升开发效率。  **容器化技术：**通过Docker部署应用镜像，确保开发、测试和生产环境的一致性，同时增强应用的可移植性和安全性。   * 1. **编程语言**   **后端：**采用Java语言构建后端服务，利用其稳健的生态系统和丰富的框架，实现高效的服务端逻辑处理。  **前端：**使用JavaScript结合HTML和CSS，打造动态且用户友好的前端交互界面。   * 1. **开发工具**   **集成开发环境(IDE)：**后端开发选用IntelliJ IDEA，充分发挥其强大的代码管理和调试功能，提升开发效率。  **版本控制：**采用Git作为版本控制系统，确保代码的版本管理和多人协作开发的顺畅进行。  **包管理与构建自动化：**结合Maven进行项目依赖管理和自动化构建，简化项目构建流程。  **框架选择：**后端采用Spring Boot框架，简化配置并加速应用开发和部署过程。  **前端框架：**前端开发使用Vue 3，结合Element UI框架，快速构建响应式的用户界面，实现高效的前端开发。   1. **核心代码与配置文件**    1. **前端配置文件**     图2.main.js  这段代码是 Vue.js 项目的入口文件，它负责初始化和配置应用。  首先，它导入了一些必要的资源和库。'./assets/main.css' 是项目的主要样式文件。'vue' 是 Vue.js 框架本身。'element-plus' 是一个基于 Vue.js 的 UI 组件库，用于构建用户界面。'element-plus/dist/index.css' 是 Element Plus 的样式文件。'./App.vue' 是应用的主组件。'axios' 是一个用于发送 HTTP 请求的库，虽然在这段代码中没有直接使用，但可能在其他地方使用。'./router' 是 Vue Router，用于处理应用的路由。  然后，它创建了一个 Vue 应用实例，参数是主组件 App。  接下来，它遍历 ElementPlusIconsVue 对象的所有条目，将每个条目注册为全局组件。这样，就可以在应用的任何地方使用 Element Plus 的图标组件，而无需在每个组件中单独导入它们。  最后，它使用 ElementPlus 和 router 插件，并将应用挂载到 id 为 'app' 的 DOM 元素上。    图3.route.js  这段代码是 Vue.js 项目中的路由配置文件。它定义了应用的路由规则，并创建了一个路由实例。  首先，它从 vue-router 包中导入了 createRouter 和 createWebHistory 函数。createRouter 用于创建路由实例，createWebHistory 用于创建一个 HTML5 history 模式的路由历史记录。  然后，它导入了四个 Vue 组件：App、Chat、User 和 Tools。这些组件将被用作路由的目标视图。  接下来，它定义了一个名为 routes 的数组，数组中的每个对象都代表一个路由规则。每个路由规则都有一个 path 属性，表示 URL 的路径；一个 name 属性，表示路由的名字；以及一个 component 属性，表示当 URL 匹配到该路由规则时，应该显示哪个组件。  例如，第一个路由规则的 path 是 '/'，name 是 'home'，component 是 Chat。这意味着当 URL 的路径为 '/' 时，应用将显示 Chat 组件，并将此路由命名为 'home'。  然后，它使用 createRouter 函数创建了一个路由实例。createRouter 的参数是一个对象，该对象有两个属性：history 和 routes。history 属性的值是 createWebHistory() 的返回值，表示使用 HTML5 history 模式。routes 属性的值是之前定义的 routes 数组，表示应用的路由规则。  最后，它导出了创建的路由实例，以便在其他文件中使用。   * 1. **后端配置文件**     图4.application.properties  这是一个Spring Boot应用程序的配置文件。在这个文件中，可以设置各种应用程序参数，包括数据源、Redis和异步请求超时等。以下是每个配置项的详细解释：  spring.application.name=demo：这是应用程序的名称，它可以用于日志或其他需要显示应用程序名称的地方。  spring.datasource.url=jdbc:mysql://db:3306/mygpt：这是应用程序连接到的数据库的URL。在这个例子中，应用程序连接到名为mygpt的MySQL数据库。  spring.datasource.username=root 和 spring.datasource.password=18675206891：这是连接到数据库时使用的用户名和密码。  spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver：这是应用程序用于连接到数据库的JDBC驱动类的名称。  spring.data.redis.host=redis 和 spring.data.redis.port=6379：这是应用程序连接到的Redis服务器的主机名和端口号。  spring.data.redis.jedis.pool.max-active=15：这是Jedis连接池的最大活动连接数。  spring.mvc.async.request-timeout=120000：这是异步请求的超时时间，单位是毫秒。在这个例子中，超时时间设置为120秒。   * 1. **前端核心代码**      1. **多轮聊天**   function sendMessage() {      if (props.ifLogin === false) {          ElMessage.error('请先登录')          return      }      if (event.shiftKey && event.keyCode === 13) {          //插入换行符          inputValue.value += '\n'          return      }      if (!inputValue.value) {          return      }      emit('add-message', 'human', inputValue.value)      aiChat()      clearInput()  }  function aiChat() {      // 禁止输入框输入      forbidInput.value = true      let chatID = props.currentChatID      // 携带cookie发送请求      const source = new EventSource("http://47.99.133.237:8080/aiChat?inputValue=" + inputValue.value +          "&chatID=" + chatID +          "&chatModel=" + chatModel.value +          "&temperature=" + temperature.value +          "&top\_p=" + top\_p.value +          "&system=" + system.value          , {              withCredentials: true          })      let ifClose = false      let count = 0;      let result = ref('')      source.onmessage = function (event) {          if (ifClose) {              chatID = event.data              emit('change-chat-id', chatID)              forbidInput.value = false              source.close()              return          }          if (event.data === 'CHAT COMPLETED!') {              ifClose = true          }          // 如果当前id为-1或者还没有向父组件发出信号，说明是新数据，则插入，否则拼接在最后一条AI数据后面          if (event.data !== 'CHAT COMPLETED!') {              if (count === 0) {                  emit('add-message', 'ai', event.data)                  count++;              } else {                  emit("append-message", event.data)              }          }      }      source.onerror = function (event) {          console.log(event)          ElMessage.error('请求失败')          //终止请求          source.close()          forbidInput.value = false      }  }  这两个函数主要用于处理用户与 AI 的聊天交互。  sendMessage 函数的逻辑如下：  首先，检查 props.ifLogin 是否为 false。如果用户未登录，将显示错误消息并退出函数。  然后，检查是否按下了 Shift 键和 Enter 键。如果是，将在 inputValue.value（可能是一个文本输入框的值）后添加一个换行符，并退出函数。  如果 inputValue.value 为空，函数将直接返回，不执行任何操作。  如果满足以上所有条件，将触发一个名为 'add-message' 的事件，并传递 'human' 和 inputValue.value 作为参数。这是将用户的消息添加到聊天记录中。  调用 aiChat 函数，开始 AI 的回复过程。  调用 clearInput 函数，是为了清空输入框。  aiChat 函数的逻辑如下：  首先，禁止输入框输入。  获取当前的聊天 ID。  创建一个新的 EventSource 对象，用于发送一个携带多个参数和 cookie 的请求到指定的 URL。  定义一个 ifClose 变量，用于判断是否需要关闭 EventSource。  定义一个 count 变量，用于计数收到的消息。  当 EventSource 收到消息时，如果 ifClose 为 true，则将聊天 ID 更新为收到的数据，触发一个 'change-chat-id' 事件，允许输入框输入，并关闭 EventSource。  如果收到的数据是 'CHAT COMPLETED!'，则将 ifClose 设置为 true。  如果收到的数据不是 'CHAT COMPLETED!'，则根据 count 的值决定是触发 'add-message' 事件还是 'append-message' 事件。'add-message' 事件用于添加新的 AI 消息，'append-message' 事件用于在最后一条 AI 消息后面添加内容。  如果 EventSource 发生错误，将在控制台打印错误，显示一个错误消息，关闭 EventSource，并允许输入框输入。    图5.前端发送信息时序图   * 1. **后端关键代码**      1. **Control层的ai对话处理**    // 调用Ai接口      @GetMapping("/aiChat")      public SseEmitter handleGetRequest(@RequestParam String chatModel, @RequestParam Integer chatID, @RequestParam String inputValue,                                         @RequestParam Float temperature, @RequestParam Double top\_p, @RequestParam String system,                                         HttpServletRequest request, HttpServletResponse response) throws IOException, InterruptedException {          SseEmitter sseEmitter = new SseEmitter();          //1. 从cookie中获取token          String token = CookieUtil.getCookieValue(request, "token");          if (token == null) {              sseEmitter.send("请先登录");              return sseEmitter;          }          Integer userID = Integer.parseInt(JWTUtil.parseToken(token, "userID"));            //2. 实例化Chat对象          // 使用AtomicReference包装Chat对象，以便在多线程环境下修改Chat对象          AtomicReference<Chat> chat = new AtomicReference<>(new Chat());          chat.get().setUserID(userID);          chat.get().setChatID(chatID);          chat.get().setInputValue(inputValue);          chat.get().setChatModel(chatModel);          chat.get().setTemperature(temperature);          chat.get().setTop\_p(top\_p);          chat.get().setSystem(system);          //3. 从redis中获取对话记录          if(stringRedisTemplate.hasKey("messageList"+chatID)){              String value = stringRedisTemplate.opsForValue().get("messageList"+chatID);              chat.get().setMessageList(value);              String systemValue = stringRedisTemplate.opsForValue().get("chatinfo:chatID"+chatID+":system");              // 更新备份键的过期时间              stringRedisTemplate.expire("backupKey"+"messageList"+chatID, 60\*30+20, TimeUnit.SECONDS);          }          Thread thread = new Thread(() -> {              try {                  if(chatModel.equals("通义千问")){                      chat.set(aliServiceImpl.aliChat(chat.get(), sseEmitter, userID));                  }else{                      chat.set(baiduServiceImpl.baiduChat(chat.get(), sseEmitter, userID));                  }              } catch (Exception e) {                  e.printStackTrace();              }            try{                // 将对话记录存入redis以及过期时间                stringRedisTemplate.opsForValue().set("messageList"+chat.get().getChatID(), chat.get().getMessageList());                stringRedisTemplate.expire("messageList"+chat.get().getChatID(), 60\*30, TimeUnit.SECONDS);                stringRedisTemplate.opsForValue().set("backupKey"+"messageList"+chat.get().getChatID(), chat.get().getMessageList());                stringRedisTemplate.expire("backupKey"+"messageList"+chat.get().getChatID(), 60\*30+20, TimeUnit.SECONDS);            }catch (Exception e){                e.printStackTrace();              }          });          thread.start();          return sseEmitter;      }  这段代码是一个Java Spring Boot控制器中的一个方法，该方法用于处理AI聊天请求。它使用了@GetMapping注解，这意味着它响应HTTP GET请求。这个方法的路径是/aiChat。  方法接受多个请求参数，包括聊天模型(chatModel)、聊天ID(chatID)、输入值(inputValue) temperature、top\_p和system。它还接受一个HttpServletRequest和一个HttpServletResponse作为参数。  方法首先创建一个SseEmitter对象。SseEmitter是Spring框架提供的一个类，用于发送服务器端事件(Server-Sent Events)。  然后，方法从cookie中获取token，并检查token是否为null。如果token为null，它会发送一个消息"请先登录"，然后返回SseEmitter。  如果token不为null，方法会解析token以获取用户ID。  接下来，方法创建一个Chat对象，并使用AtomicReference包装它。AtomicReference是一个可以在多线程环境下安全使用的引用类型。然后，方法设置Chat对象的各种属性。  然后，方法检查Redis中是否存在对话记录。如果存在，它会从Redis中获取对话记录，并设置Chat对象的messageList属性。同时，它还会更新备份键的过期时间。  接下来，方法创建一个新的线程来处理AI聊天。如果聊天模型是"通义千问"，它会调用aliServiceImpl.aliChat方法；否则，它会调用baiduServiceImpl.baiduChat方法。这两个方法都会修改Chat对象，并将修改后的Chat对象发送给SseEmitter。  最后，方法将对话记录存入Redis，并设置过期时间。同时，它还会将对话记录存入备份键，并设置备份键的过期时间。  最后，方法返回SseEmitter对象。    图6.aichat时序图   * + 1. **Service层的ai对话处理**   @Transactional      public Chat aliChat(Chat chat, SseEmitter sseEmitter, Integer userID) throws NoApiKeyException, InputRequiredException, IOException {          Timestamp time = new Timestamp(System.currentTimeMillis());          //1. 获取对话ID          Integer chatID = chat.getChatID();          if(chatID.equals(-1)){              // 是一个新对话，需要插入数据库              chat.setTime(time);              chat.setChatTitle(chat.getInputValue());              chatMapper.insertChat(chat);          }else{              // 是一个已有对话              chat.setTime(time);              if(chat.getMessageList()==null){                  System.out.println("redis没命中，重新从数据库中获取");                  chat.setMessageList(chatinfoMapper.getMessageList(chatID));              }              // 更新对话时间              chatMapper.updateChatTime(chatID, time);          }          //2. 调用通义千问接口          ChatUtils.aliStreamCall(chat, sseEmitter);          return chat;      }    图7. Aichat时序图2  这段代码是一个名为aliChat的方法，它在AliServiceImpl.java文件中。这个方法接收三个参数：一个Chat对象，一个SseEmitter对象，和一个userID的整数。  首先，它创建一个Timestamp对象，表示当前时间。  然后，它从Chat对象中获取对话ID。如果对话ID等于-1，那么这是一个新的对话，需要将其插入到数据库中。在插入之前，它会设置对话的时间和标题。如果对话ID不等于-1，那么这是一个已经存在的对话。它会更新对话的时间，并检查消息列表是否为空。如果消息列表为空，那么它会从数据库中重新获取消息列表。然后，它会更新数据库中对话的时间。  接下来，它调用ChatUtils.aliStreamCall方法，这个方法可能是用来与某个接口进行交互。  最后，它返回更新后的Chat对象。  这个方法使用了@Transactional注解，这意味着这个方法是一个事务，也就是说，如果在执行这个方法的过程中发生了错误，那么所有的数据库操作都会被回滚，数据库的状态会恢复到调用这个方法之前的状态。   * + 1. **ChatUTils**   package com.example.demo.Utils;  public class ChatUtils {      public static List<Message> createAliMessageList(String messageList) {          //1. 将messageList转换为List<Message>对象          List<Message> messages = new ArrayList<>();          // 将messageList按"user: "分割          String[] userMessages = messageList.split("user: ");          //遍历userMessages，将每个元素按"assistant: "分割          for (String userMessage : userMessages) {              //如果开头是"system:"，则设置为系统信息              if (userMessage.startsWith("system:")) {                  Message system = Message.builder()                          .role(Role.SYSTEM.getValue())                          .content(userMessage.substring(8))                          .build();                  messages.add(system);                  continue;              }              // 如果为空字符串，则跳过              if (userMessage.equals("")) {                  continue;              }              String[] assistantMessages = userMessage.split("assistant: ");              //此时第一个信息一定是用户信息，将其添加到messages中              Message user = Message.builder()                      .role(Role.USER.getValue())                      .content(assistantMessages[0])                      .build();              messages.add(user);              //如果有第二个信息，则一定是助手信息，将其添加到messages中              if (assistantMessages.length > 1) {                  Message assistant = Message.builder()                          .role(Role.ASSISTANT.getValue())                          .content(assistantMessages[1])                          .build();                  messages.add(assistant);              }          }          return messages;      }      public static StringBuilder messageListToJson(List<Message> messages, String system, Float temperature, Double top\_p) {          StringBuilder json = new StringBuilder("{\"messages\": [");          for (Message message : messages) {              String role = message.getRole();              if(role.equals("system")){                  continue;              }              String content = message.getContent();              content = content.replace("\"", "“");              content = content.replace("\n", "");              json.append("{\"role\":\"").append(role).append("\",\"content\":\"").append(content).append("\"},");          }          json.deleteCharAt(json.length() - 1);          json.append("],\"stream\":true,\"disable\_search\":false,\"enable\_citation\":false");          json.append(",\"system\":\"").append(system).append("\"");          json.append(",\"temperature\":").append(temperature);          json.append(",\"top\_p\":").append(top\_p);          json.append("}");          return json;      }      public static void aliStreamCall(Chat chat, SseEmitter sseEmitter) throws NoApiKeyException, InputRequiredException, IOException {          //1. 获取用户输入的问题，并创建一个List<Message>对象          String inputValue = chat.getInputValue();          String messageList = chat.getMessageList();          // 用于存储对话记录          List<Message> messages = new ArrayList<>();          //如果messageList不为空，则用其构建一个Message对象          if (messageList != null) {              messages = ChatUtils.createAliMessageList(messageList);          }else{              Message systemMessage = Message.builder()                      .role(Role.SYSTEM.getValue())                      .content(chat.getSystem())                      .build();              messages.add(systemMessage);          }          //2. 创建新会话          Generation generation = new Generation();          //3. 用户新输入          Message userMessage =                  Message.builder().                          role(Role.USER.getValue()).                          content(inputValue).                          build();          messages.add(userMessage);          //4. 创建GenerationParam对象          GenerationParam generationParam = GenerationParam.builder()                  .model("qwen-max-0428")                  .messages(messages)                  .resultFormat(GenerationParam.ResultFormat.MESSAGE)                  .temperature(chat.getTemperature())                  .topP(chat.getTop\_p()).enableSearch(true)                  .incrementalOutput(true)                  .enableSearch(true)                  .build();          //5. 流式调用通义千问接口          Flowable<GenerationResult> result = generation.streamCall(generationParam);          StringBuilder fullContent = new StringBuilder();          result.blockingForEach(message -> {              StringBuilder content = new StringBuilder(message.getOutput().getChoices().get(0).getMessage().getContent());              fullContent.append(content);              sseEmitter.send(content.toString());          });          //6. 将对话记录存储到chat对象中          StringBuilder userContent = new StringBuilder(inputValue);          StringBuilder assistantContent = new StringBuilder(fullContent);          if(chat.getMessageList() == null){              chat.setMessageList("system:" + chat.getSystem() + "\nuser: " + userContent + "\nassistant: " + assistantContent + "\n");          }else{              chat.setMessageList(chat.getMessageList() + "user: " + userContent + "\nassistant: " + assistantContent + "\n");          }          try {              sseEmitter.send("CHAT COMPLETED!");              sseEmitter.send(chat.getChatID());          } catch (IOException e) {              e.printStackTrace();          } finally {              sseEmitter.complete();          }      }        public static void baiduStreamCall(Chat chat, SseEmitter sseEmitter) throws IOException {          // 获得用户输入的问题          String inputValue = chat.getInputValue();          String messageList = chat.getMessageList();          // 如果messageList不为空，则将inputValue添加到messageList中          if (messageList != null) {              messageList += "user: " + inputValue + "\n";          } else {              messageList = "user: " + inputValue + "\n";          }          // 创建Json对象          StringBuilder json = ChatUtils.messageListToJson(ChatUtils.createAliMessageList(messageList), chat.getSystem(), chat.getTemperature(), chat.getTop\_p());          // 向百度接口发送请求          String accessToken = getAccessToken();          final Boolean[] isEnd = {false};          MediaType mediaType = MediaType.parse("application/json");          Request request = new Request.Builder()                  .url("https://aip.baidubce.com/rpc/2.0/ai\_custom/v1/wenxinworkshop/chat/completions?access\_token=" + accessToken)                  .post(RequestBody.create(mediaType, json.toString()))                  .addHeader("Content-Type", "application/json")                  .build();          String finalMessageList = messageList;          HTTP\_CLIENT.newCall(request).enqueue(new Callback() {              @Override              public void onFailure(Call call, IOException e) {                  // 请求失败的处理                  e.printStackTrace();              }              @Override              public void onResponse(Call call, Response response) throws IOException {                  if (!response.isSuccessful()) {                      throw new IOException("Unexpected code " + response);                  } else {                      try (ResponseBody responseBody = response.body()) {                          if (responseBody != null) {                              // 流式处理响应体                              responseBody.source().timeout().timeout(60, TimeUnit.SECONDS);                              Buffer buffer = new Buffer();                              StringBuilder content = new StringBuilder();                              while (true) {                                  long read = responseBody.source().read(buffer, 8192);                                  if (read == -1) {                                      break;                                  }                                  String all = buffer.readString(Charset.defaultCharset());                                  int start = all.indexOf("result") + 9;                                  int end = all.indexOf("need\_clear\_history") - 3;                                  String result = all.substring(start, end);                                  content.append(result);                                  sseEmitter.send(result);                              }                              // 发送完成消息                              sseEmitter.send("CHAT COMPLETED!");                              sseEmitter.send(chat.getChatID());                              // 将对话记录存储到chat对象中，拼接上user输入和assistant回答                              if(chat.getMessageList()==null){                                  chat.setMessageList("system:" + chat.getSystem() + "\n" + finalMessageList + "assistant: " + content + "\n");                              }else{                                  chat.setMessageList(finalMessageList + "assistant: " + content + "\n");                              }                              sseEmitter.complete();                              isEnd[0] = true;                          }                      }                  }              }          });          while(!isEnd[0]){              try {                  Thread.sleep(1000);              } catch (InterruptedException e) {                  e.printStackTrace();              }          }      }  }    图8.chatUtils时序图  这些代码位于ChatUtils.java文件中，定义了一个名为ChatUtils的工具类，该类包含了一些静态方法，用于处理与聊天相关的操作。  createAliMessageList(String messageList)：这个方法接收一个字符串messageList，然后将其转换为Message对象的列表。它首先按照"user: "将messageList分割，然后遍历分割后的字符串，如果字符串以"system:"开头，那么创建一个系统消息，否则，将字符串按照"assistant: "分割，创建用户消息和助手消息。  messageListToJson(List<Message> messages, String system, Float temperature, Double top\_p)：这个方法接收一个Message对象的列表，一个系统字符串，一个温度值和一个top\_p值，然后将这些信息转换为JSON格式的字符串。  aliStreamCall(Chat chat, SseEmitter sseEmitter)：这个方法接收一个Chat对象和一个SseEmitter对象，然后进行一系列的操作，包括获取用户输入的问题，创建新会话，用户新输入，创建GenerationParam对象，流式调用通义千问接口，将对话记录存储到chat对象中，最后发送"CHAT COMPLETED!"和聊天ID。  baiduStreamCall(Chat chat, SseEmitter sseEmitter)：这个方法接收一个Chat对象和一个SseEmitter对象，然后进行一系列的操作，包括获取用户输入的问题，创建Json对象，向百度接口发送请求，流式处理响应体，发送完成消息，将对话记录存储到chat对象中，最后等待请求结束。  这些方法主要用于处理与聊天相关的操作，包括创建消息，将消息转换为JSON格式，以及与阿里和百度的接口进行交互。 |
| **小结** |
|  |
| **指导教师评语及成绩** |
| 评语：  成绩：    指导教师签名：                                                  批阅日期： |

**测试报告**

|  |
| --- |
| **概述** |
| 【基本要求】  需要进行黑盒测试，功能测试，将测试通过的代码共享至公共的代码托管平台，并将该网站部署上线并提供验证测试。  【基本内容】  测试方法（主要包括：黑盒测试，功能测试）；  测试的指标；  测试结果（需要提供截图）。 |
| **报告内容**  **1. 功能测试**  **1.1 用户注册**  目标：验证用户注册功能的正确性、容错处理能力以及安全性。  **1.1.1 正常注册用例**   * **测试目的**：验证用户能够成功注册账号。 * **测试步骤**：   + 输入有效的手机号（国内格式正确，未被注册）   + 合法的用户名（未被使用） * **预期结果**：用户成功注册，收到注册确认信息 * **指标**：注册成功率100%，响应时间<1s。       **1.1.2 错误注册用例：手机号格式错误**   * **测试目的**：验证系统能否识别并拒绝不合规的手机号格式。 * **测试步骤**：输入错误格式的手机号（如包含字母、特殊字符、过短于标准长度等） * **预期结果**：系统提示手机号格式错误，注册失败。 * **指标**：错误提示准确率100%，响应时间<500ms。     **1.1.3 错误注册用例：用户名已注册**   * **测试目的**：检测系统是否防止用户名重复。 * **测试步骤**：尝试使用已存在的用户名注册 * **预期结果**：系统提示用户名已存在，注册失败。 * **指标**：重复检测准确率100%，响应时间<500ms。       **1.1.4 错误注册用例：手机号已注册**   * **测试目的**：测试手机号唯一性验证机制。 * **测试步骤**：使用已注册过的手机号尝试再次注册 * **预期结果**：系统提示手机号已注册，注册失败。 * **指标**：手机号验证准确率100%，响应时间<500ms。       **1.2 用户登录**  目标：验证用户登录功能的正确性、异常处理以及安全性。  **1.2.1 正常登录用例**   * **测试目的**：验证合法用户能够成功登录。 * **测试步骤**：   + 输入正确的手机号   + 正确的密码 * **预期结果**：登录成功，跳转至首页，显示欢迎信息。 * **指标**：登录成功率100%，响应时间<500ms。       **1.2.3 错误登录用例：密码错误**   * **测试目的**：验证密码验证机制。 * **测试步骤**：正确手机号，错误密码 * **预期结果**：密码错误提示，账户锁定或重试错次数限制。 * **指标**：错误提示准确率100%，响应时间<500ms。       **1.3 AI聊天功能**  **1.3.1 正常聊天用例**   * **测试目的**：验证AI聊天功能的正常对话能力。 * **测试步骤**：用户发送常见问题，如天气查询、新闻热点讨论。 * **预期结果**：AI提供相关、准确、及时回复。 * **指标**：回复相关度≥90%，响应时间<2s。     **1.3.2 个性化设置测试**   * **测试目的**：验证个性化设置对AI回复的影响。 * **测试步骤**：调整AI的语气（正式、幽默）、回复风格等。 * **预期结果**：AI回复风格随设置变化。 * **指标**：个性化设置反应度≥90%。       **1.3.3 上下文理解测试**   * **测试目的**：检查AI对上下文的跟踪能力。 * **测试步骤**：连续提问，前一问句与后一问相关。 * **预期结果**：AI能理解并基于上下文回答。     **1.3.4 文生图测试**   * **测试目的：**评估AI根据文字描述生成图像的能力。 * **测试步骤：**用户提供具体文本描述，如“一个穿着古装的女子站在樱花树下”。 * **预期结果**：AI生成的图像与描述高度匹配，细节丰富，情感色彩符合描述场景。 * **指标：**图像与描述相符度≥85%，创意表现和细节完整性高。       **1.5 聊天历史记录**  **1.5.1 保存历史记录**   * **测试目的：**验证聊天记录保存机制。 * **测试步骤：**   + 进行多次聊天交互，包含文字、图片、表情、链接、特殊字符混合内容。 * **预期结果：**所有聊天记录均被完整保存至数据库，无遗漏。 * **指标：**记录完整性100%，响应时间<5s。     **1.5.2 删除历史记录**   * **测试目的：**验证历史记录删除功能。 * **测试步骤：**   + 选择一段历史聊天记录，发起删除操作。 * **预期结果：**指定聊天记录被删除，不影响其他记录。 * **指标：**删除准确率100%，响应时间<2s。       **1.5.3 查看历史记录**   * **测试目的**：历史记录的查看功能。 * **测试步骤：**   + 按照时间线性筛选某一天内的聊天记录，通过关键词搜索特定话题。 * **预期结果：**快速定位到相关聊天记录，按时间或关键词准确匹配。 * **指标：**搜索精度≥80%，响应时间<2s。         **1.6 界面优化**  **1.6.1 易用性**   * **测试目的**：界面的用户友好性。 * **测试步骤**：新手用户操作，无指导下尝试功能。 * **预期结果**：容易上手。 * **指标**：易用性评分≥4分（满分5分）。 |
| **小结** |
|  |
| **指导教师评语及成绩** |
| 评语：  成绩：    指导教师签名：                                                  批阅日期： |