面向对象课程设计报告·4

一、项目需求的进一步理解和描述

客服功能应当降低人工成本，可以考虑接入大语言模型，实现 AI 客服的功能。

二、相关技术

C/S结构和B/S结构

C/S结构和B/S结构是两种常见的网络应用架构，它们各有特点，适用于不同的应用场景。

(1) C/S结构（Client/Server，客户端/服务器）

1. **概述**：
   - C/S结构主要应用于局域网内，是一种两层结构的网络体系结构。
   - 客户端负责完成与用户的交互任务，而服务器负责数据的管理。
2. **应用场景**：
   - 适用于需要高性能、低延迟、安全性要求较高的应用场景，如企业内部管理系统、数据库管理系统等。

(2) B/S结构（Browser/Server，浏览器/服务器）

1. **概述**：
   - B/S结构是WEB兴起后的一种网络结构模式，统一了客户端，将系统功能实现的核心部分集中到服务器上。
   - 客户端只需安装一个浏览器，即可通过Web Server与数据库进行数据交互。
2. **应用场景**：
   - 适用于需要跨平台、易维护、用户群广泛的应用场景，如Web应用、OA系统、城市消防联网等。

（3）C/S结构与B/S结构的比较

特性	C/S结构	B/S结构
硬件环境	专用网络，如局域网	广域网，用户较多
安全性	高，面向固定用户群	较低，面向广泛用户群
便捷性	低，需安装客户端	高，无需安装客户端
性能	充分发挥客户端处理能力	主要事务逻辑在服务器端实现
扩展性	可水平或垂直扩展	易于扩展和维护
应用场景	企业内部管理系统、 数据库管理系统等	Web应用、OA系统、 城市消防联网等

综上所述，C/S结构和B/S结构各有优缺点，应根据项目的具体需求、用户群、安全性要求、便捷性需求以及性能要求等因素综合考虑，选择最合适的架构。本项目选择 C/S 架构以满足电子商务系统的需求。

MVC框架

MVC框架是一个设计模式，它强制性地使应用程序的输入、处理和输出分开，从而使同一个程序可以使用不同的表现形式。

（1）MVC框架的基本概念

MVC（Model-View-Controller）框架将软件系统分为三个基本部分：模型（Model）、视图（View）和控制器（Controller）。每个部分都有明确的职责：

1. **模型（Model）**：用于封装与应用程序的业务逻辑相关的数据以及对数据的处理方法。它代表应用程序的核心功能和业务逻辑，处理数据的存储、检索和更新。
2. **视图（View）**：负责用户界面的展示，向用户显示数据。它呈现来自模型的数据，并允许用户与之交互。视图层通常使用HTML、CSS和JavaScript等技术来构建。
3. **控制器（Controller）**：处理用户的输入，调用相应的模型和视图以完成用户请求。它作为用户接口和应用程序之间的桥梁，接收用户的请求，并根据请求调用模型层来处理业务逻辑，然后选择合适的视图层来显示结果。

（2）MVC框架的工作原理

MVC框架的工作原理可以概括为以下步骤：

1. 用户通过视图层与应用程序进行交互，如点击按钮或输入数据。
2. 视图层将用户的请求传递给控制器层。
3. 控制器层根据请求调用模型层来处理业务逻辑，如查询数据库或执行计算。
4. 模型层处理完业务逻辑后，将结果返回给控制器层。
5. 控制器层根据处理结果选择合适的视图层来显示数据，并将数据传递给视图层进行渲染。
6. 渲染完成后，视图层将结果呈现给用户。

（3）MVC框架的优点

1. **关注点分离**：MVC框架实现了关注点分离，将应用程序的输入、处理和输出分开，使得代码更加清晰、易于维护。
2. **高重用性**：由于模型、视图和控制器是独立的组件，因此可以很容易地在不同的项目之间重用这些组件。
3. **灵活性**：MVC框架允许使用不同的视图来展示相同的数据，从而提供了更大的灵活性。
4. **可测试性**：由于模型、视图和控制器是独立的，因此可以单独对它们进行测试，从而提高测试的效率和质量。

（4）常见的MVC框架实现

在Java领域，Spring MVC是一个常见的MVC框架实现。它提供了丰富的功能和灵活的配置选项，使得开发者可以轻松地构建复杂的Web应用程序。此外，在PHP领域，也有许多流行的MVC框架，如$Laravel$ 、$Symfony$ 等。

综上所述，MVC框架是一种强大的设计模式，它通过将应用程序的输入、处理和输出分开，提高了代码的可维护性、重用性和灵活性。在Web应用开发领域，MVC框架具有广泛的应用前景。本项目采用 $Spring$ 框架实现。

持久层

持久层是系统逻辑层面上一个相对独立的领域，专注于实现数据持久化。

（1）定义与目的

持久层（Persistence Layer）负责将内存中的数据持久化到可掉电式存储设备中，如磁盘，以供以后使用。这一层的主要目的是确保数据的持久性和可靠性，即使在系统断电或崩溃后，数据仍能被恢复和访问。

（2）技术实现

持久层的技术实现通常依赖于关系数据库，但也可以通过其他存储技术如 $NoSQL$ 数据库来实现。在实现过程中，持久层会使用一系列类和组件来与数据存储器进行交互。这些类和组件通常包括：

1. **数据访问对象（DAO）**：负责直接访问数据库，执行CRUD（创建、读取、更新、删除）操作。
2. **实体类**：代表数据库中的表或记录，通常与数据库表有直接的映射关系。
3. **ORM框架**：如 $Hibernate$ 、$MyBatis$ 等，它们通过对象关系映射将实体类与数据库表关联起来，简化了数据访问层的开发。

（3）持久层的作用

1. **数据持久化**：将内存中的数据保存到磁盘上，确保数据的持久性和可靠性。
2. **简化开发**：通过ORM框架等技术，简化了数据访问层的开发，降低了开发成本。
3. **提高性能**：持久层框架通常提供连接复用、SQL优化等功能，提高了数据库访问的性能。
4. **事务管理**：持久层框架支持事务管理，确保数据的一致性和完整性。

综上所述，持久层在系统架构中扮演着至关重要的角色，它负责数据的持久化和访问，是确保数据可靠性和一致性的关键所在。在本项目中，采用 $MyBatis$ 框架。

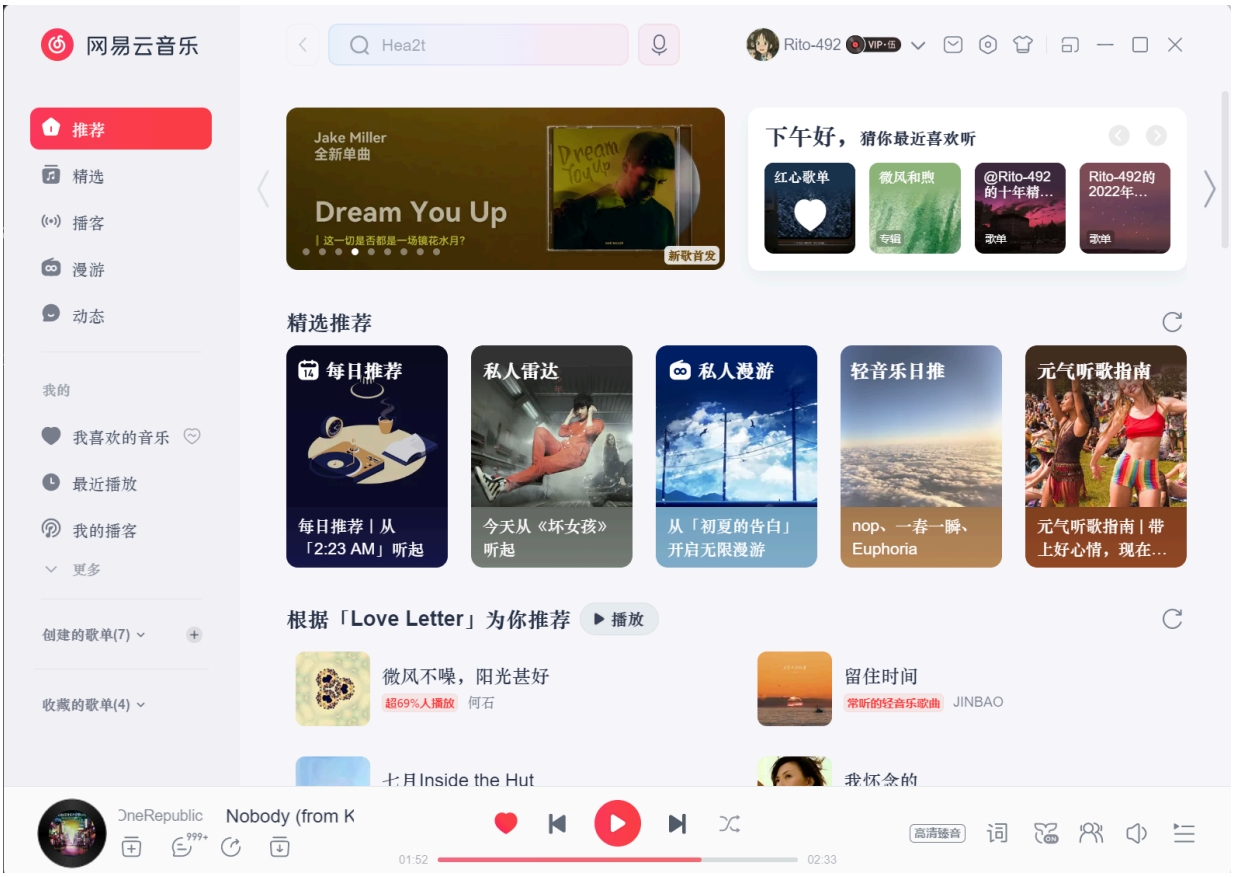
三、前端和后端概要设计

模块划分

主要分为个模块，包括：客户端和服务端的UI设计模块、数据处理模块、数据库模块。

UI设计

- 主界面的布局设计类似于网易云音乐客户端，如下图所示：



- 各个按钮的设计可以参考网易云音乐客户端，个人认为，它的UI设计清新简约
- 尝试使用 QSS 对界面进行优化。

数据库设计

数据库主要包括三个表：$customers$、$commodities$、$orders$，分别存储客户信息、商品信息、订单信息等数据。

在 $customers$ 表中，设置 $customer\_id$ 作为主键，其中 $customer\_id$ 保持数据库内部自增，在删除特定 id 的对象后，不会有重复的 id 继续生成。且该表中存在必须填写的项目：如姓名和密码是必须填写的，其他选填，购买的数量是自动统计的。

在 $commodities$ 表中，设置 $commodity\_id$ 作为主键，不可重复，且保持自增，其他属性中：名称，价格，为不可缺少的属性，其他属性可以选填。

在 $orders$ 表中，其属性较多，且将 $customers$ 、$commodities$ 的属性作为外键，其中所有的属性均为非空，由于 $orders$ 是由订单产生，得到了最后的结果一般为完整信息。

通讯协议

采用 TCP/IP 协议。

协议层次结构：

- 应用层：处理具体的业务逻辑，如用户登录、商品查询、订单处理等。
- 传输层：使用现有的传输协议（如TCP/IP）来保证数据的可靠传输。
- 网络层：处理数据的路由和转发。
- 数据链路层：负责数据的帧封装和错误检测。

注意：

- 为每种业务逻辑设计唯一的消息类型
- 定义错误码，用于表示各种可能的错误情况
- 在消息头中包含状态码，以便接收方知道操作是否成功