# 设计模式与用例实现

## **设计模式**

### **单例模式**

单例模式确保一个类只有一个实例，并提供一个全局访问点。  
在我们的音乐推荐系统中，我们会使用单例模式来管理全局的配置信息，如数据库连接配置，这样我们可以在整个应用中共享同一份配置信息。此外，音乐播放器也是一个单例，因为在整个应用中我们只需要一个音乐播放器实例。  
在我们的音乐推荐系统中，单例模式被应用在登录控制器上，以确保在整个系统中只有一个登录控制器实例。这样可以避免因创建多个登录控制器实例而导致的资源浪费和潜在的错误。以下是单例模式在登录控制器中的应用：

public class LoginManager {  
 private static LoginManager instance = null;  
  
 private LoginManager() {  
 // 私有化构造方法，防止外部创建新的实例  
 }  
  
 public static synchronized LoginManager getInstance() {  
 if (instance == null) {  
 synchronized (LoginManager.class) {  
 if (instance == null) {  
 instance = new LoginManager();  
 }  
 }  
 }  
 return instance;  
 }  
}

在这个例子中，LoginManager 是我们的登录控制器。我们在 LoginManager 类中定义了一个静态成员 instance，这个成员是 LoginManager 类型的，它用来保存 LoginManager 的唯一实例。我们将 LoginManager 的构造方法私有化，这样外部就不能直接创建 LoginManager 的新实例。然后我们提供了一个公共的静态方法 getInstance，这个方法用来获取 LoginManager 的唯一实例。在 getInstance 方法中，我们首先检查 instance 是否为 null，如果是，那么我们就创建一个新的 LoginManager 实例并赋值给 instance；如果不是，那么我们就直接返回 instance。注意，我们在创建 LoginManager 实例的过程中使用了 synchronized 关键字，这是为了防止在多线程环境下创建多个 LoginManager 实例。我们首先在方法级别上使用了 synchronized，这样在同一时间只能有一个线程进入 getInstance 方法。然后我们在创建 LoginManager 实例的代码块上再次使用了 synchronized，这样在同一时间只能有一个线程执行这个代码块。这种双重检查锁定的方式可以确保在多线程环境下 LoginManager 的唯一性。通过这种方式，我们可以确保 LoginManager.getInstance() 总是返回全局唯一的登录控制器实例。

### **职责链模式**

职责链模式是一种行为型设计模式，它创建了一个对象链。这个链上的每个对象都有机会处理请求，直到某个对象处理它为止。这种模式在处理流程中非常有用，因为它可以使多个处理逻辑不同的对象能够处理请求，并将这些对象连成一条链，沿着这条链式结构传递请求，进行请求的处理。

在我们的音乐推荐系统中，我们可以使用职责链模式来处理各种不同的用户请求。例如，我们可能有一个请求处理链来处理用户的情绪分析记录请求、个性化情绪音乐推荐请求、音乐室社交系统的请求等。每个处理器在链中都有机会处理请求，如果它不能处理，就将请求传递给链中的下一个处理器。这样，我们可以将处理逻辑分散到各个处理器中，使得系统更加灵活和可扩展。

此外，职责链模式也可以帮助我们避免请求的发送者与处理者之间的耦合。在我们的系统中，请求的发送者不需要知道请求是由哪个处理器处理的，它只需要将请求发送到处理链上，然后由处理链上的处理器来决定谁来处理这个请求。这样，我们可以在不改变请求发送者的代码的情况下，动态地改变请求的处理逻辑，增加或删除处理器，这使得我们的系统更加灵活和易于维护。

例如，我们可能有一个处理用户情绪分析记录请求的处理器，这个处理器在链中的位置可能会根据系统的需求和设计而变化。如果我们发现用户情绪分析记录请求的处理逻辑需要改变，我们可以很容易地在处理链中添加一个新的处理器来处理这个请求，而不需要修改请求的发送者的代码。

### **观察者模式**

观察者模式定义了对象之间的一对多依赖关系，当一个对象的状态改变时，所有依赖于它的对象都会得到通知并自动更新。  
在我们的系统中，我们会使用观察者模式来实现事件驱动的功能。例如，当用户的情绪得分改变时，我们需要通知其他系统组件进行相应的更新，如个性化情绪音乐推荐系统需要根据用户的新的情绪得分来更新推荐的音乐。

### **代理模式**

代理模式为其他对象提供一个代理或占位符，以控制对这个对象的访问。  
在我们的系统中，我们可能会使用代理模式来控制对某些资源的访问。例如，我们会使用代理模式来实现网络请求的缓存，以提高系统的性能。我们可以创建一个音乐请求的代理类，这个类在接收到音乐请求时，首先检查缓存中是否有请求的结果，如果有，就直接返回缓存的结果，如果没有，就执行网络请求，然后将结果存入缓存。

### **策略模式**

策略模式定义了一系列的算法，并将每一个算法封装起来，使它们可以互相替换。策略模式让算法的变化独立于使用算法的客户。  
在我们的系统中，我们可能会使用策略模式来实现不同的业务逻辑。例如，我们可能会有不同的策略来处理不同类型的用户请求。每个策略都封装了一种处理请求的算法，我们可以在运行时根据请求的类型来选择使用哪种策略。例如，我们可能有一个策略用于处理用户的情绪分析记录，另一个策略用于处理个性化情绪音乐推荐，另一个策略用于处理音乐室社交系统的请求，等等。这样，我们可以将不同的处理逻辑分散到各个策略中，使得系统更加灵活和可扩展。

### **解释器模式**

解释器模式为给定的语言定义了一个表示，并定义了一个解释器，这个解释器使用该表示来解释语言中的句子。

在我们的音乐推荐系统中，解释器模式可能会被用于处理用户的搜索请求或者其他复杂的用户输入。例如，用户可能会输入一些复杂的查询，如"我想听一些快乐的音乐"或"我需要一些帮助我放松的音乐"。这些查询可能包含了一些复杂的语义，我们需要一个解释器来解析这些查询，然后将它们转换成系统可以理解的形式。

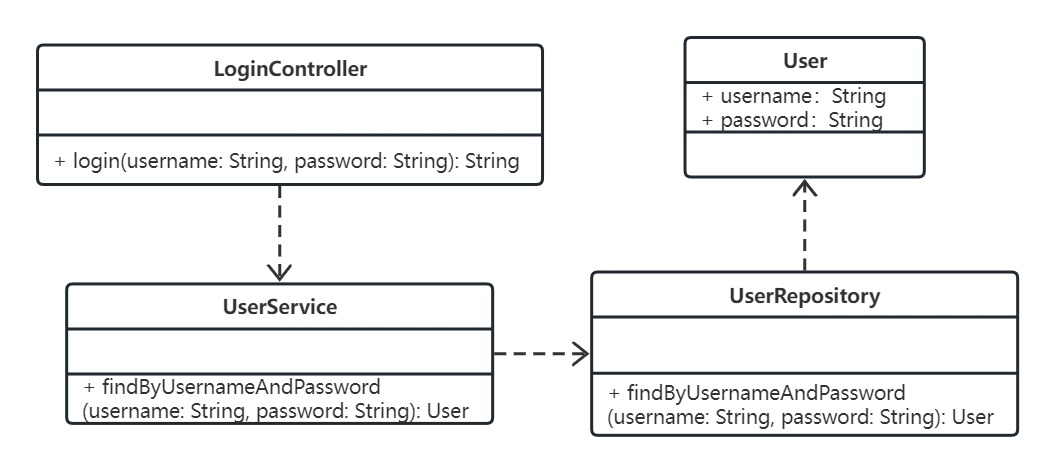
具体来说，我们可以定义一个MusicQuery接口，这个接口定义了解析音乐查询的方法。然后我们可以创建一个MusicQueryInterpreter类，这个类实现了MusicQuery接口，它可以解析用户的音乐查询，并将它们转换成系统可以理解的形式。当用户输入一个音乐查询时，我们的系统就可以创建一个MusicQueryInterpreter实例，然后使用这个实例来解析用户的查询。这样，我们就可以根据用户的查询来推荐适合的音乐，而不需要修改系统的内部表示。

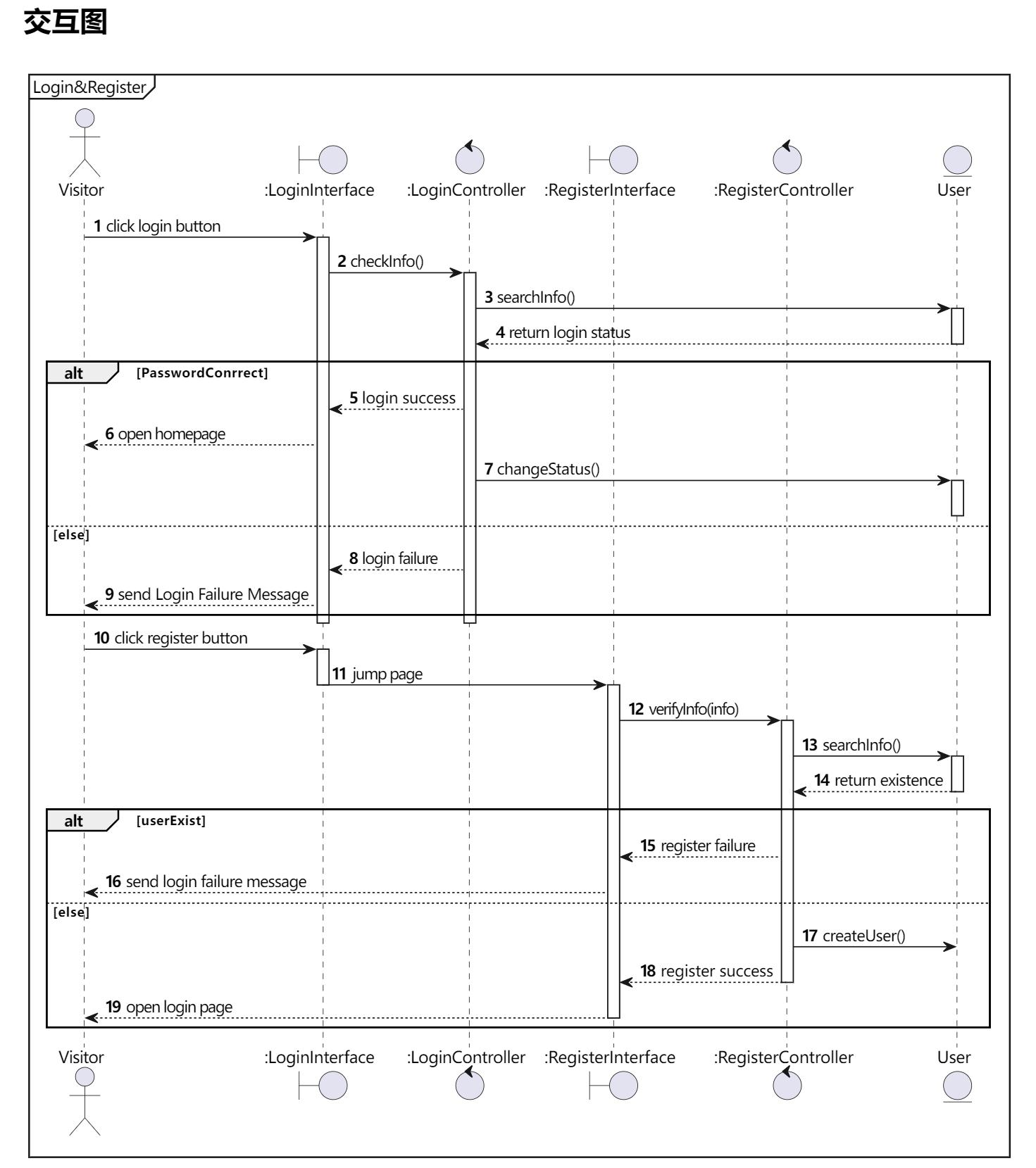
这就是解释器模式在我们的音乐推荐系统中的一个应用实例。通过使用解释器模式，我们可以将复杂的查询解析逻辑抽象化，使得系统更加灵活和可扩展。同时，解释器模式也使得我们的代码更加清晰和易于理解，提高了代码的可读性和可维护性。

## **用例的实现**

### 用户登录

类图与交互图如下：





实现步骤如下

1. 用户在登录表单中输入用户名和密码。
2. 登录界面会先对信息进行输入级别的检查，即系统会拒绝格式不合法的登录信息（如空用户名和空密码）。
3. 之后，信息将被解析为 JSON 格式。登录请求将被发送到 Spring MVC。
4. 同时，带有登录信息的 JSON 文件将通过 REST API 发送到数据库，同时向数据库发送相应的查询请求。
5. 数据库中的用户表存储了用户的所有信息。数据库查询用户表，查找给定信息是否在用户表中。如果存在，则将成功消息返回给用户管理子系统，即可以在数据库中找到对应的用户名和密码。
6. 在得到查询结果后，我们的系统也会启动一个验证过程来检查用户帐户是否被允许登录。身份验证过程将由 Sa-Token 处理。通过调用StpUtil.getUserByToken方法，会返回用户的token是否合法。
7. 如果证明token是合法的（如果出现用户封禁等情况，其token会被设置为非法），会生成一个新的登录token，发送到登录子系统，并保存到本地。
8. 每当用户在我们系统中切换网页时，Sa-Token（通过 REST API）将在登录状态级别（操作由 StpUtil.getLoginIdByToken 方法提供支持）检查帐户的token，以防系统崩溃或其他错误情况。具体来说，当用户的登录状态受到系统故障影响时，token将被禁用，系统将拒绝用户继续使用该服务。

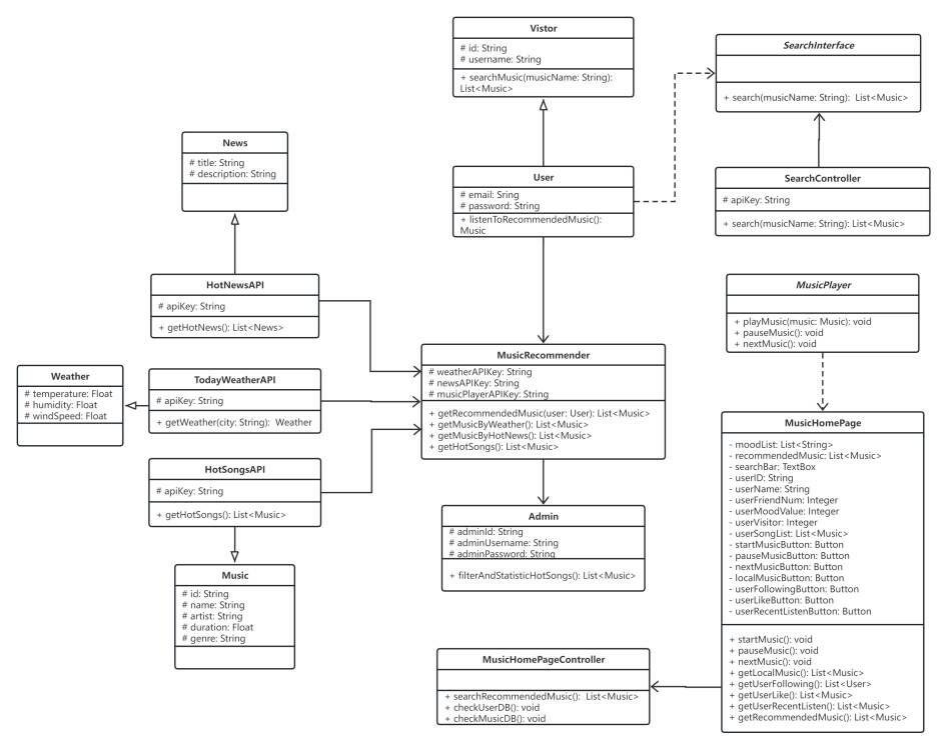
下面的代码展示了如何在Spring MVC中处理登录请求，并使用Sa-Token进行身份验证。

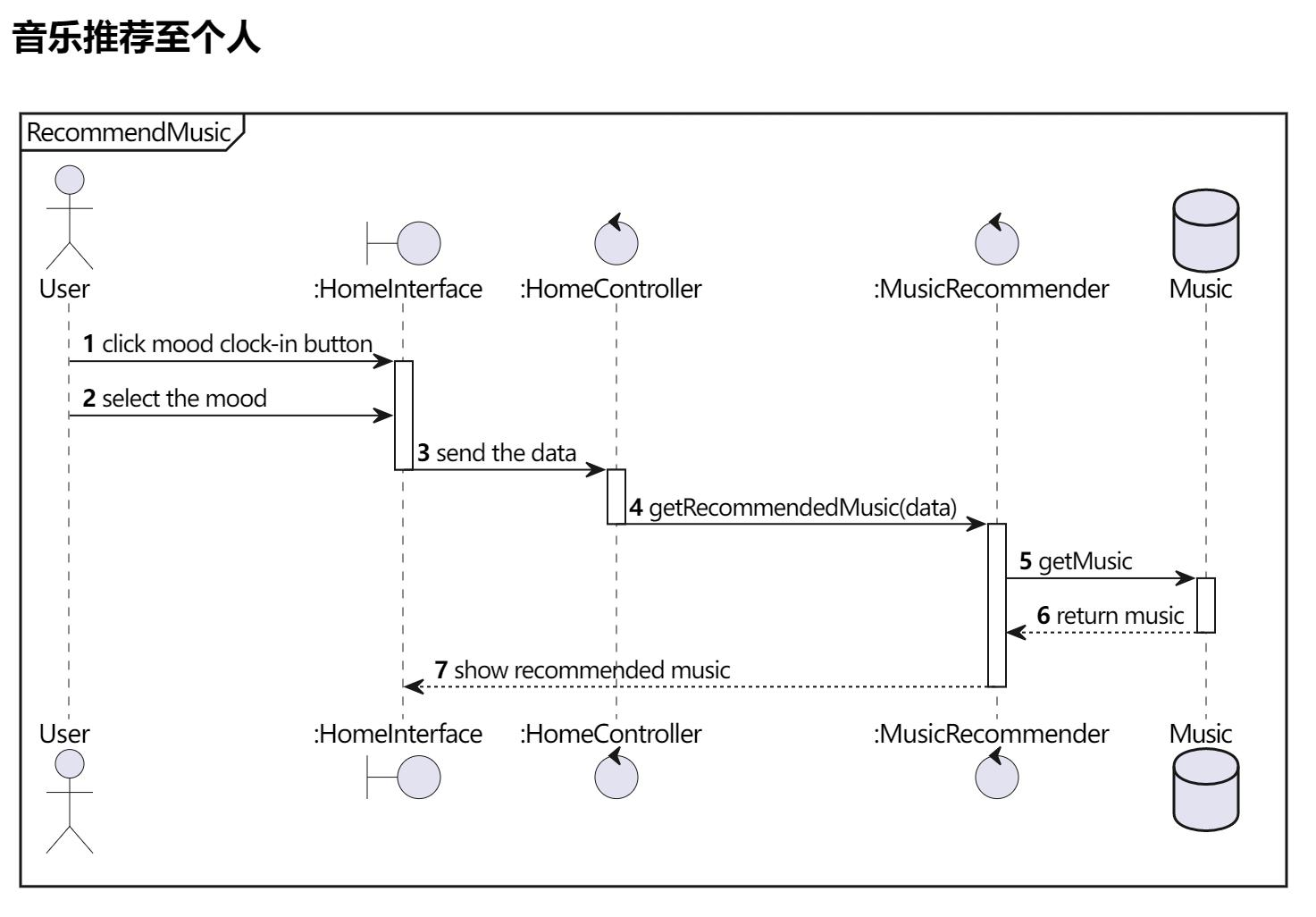
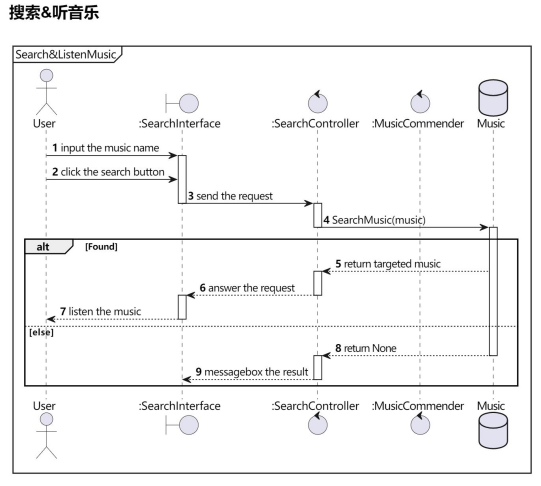
// LoginController.java  
public class LoginController {  
 private UserService userService;  
 public String login(String username, String password) {  
 User user = userService.findByUsernameAndPassword(username, password);  
 if (user != null) {  
 // 登录成功，使用Sa-Token进行身份验证  
 loginWithSaToken(user.getId());  
 return "redirect to home page";  
 } else {  
 // 登录失败  
 return "redirect to login page";  
 }  
 }  
}  
// UserService.java  
public class UserService {  
 private UserRepository userRepository;  
 public User findByUsernameAndPassword(String username, String password) {  
 // 查询数据库，查找用户名和密码匹配的用户  
 return userRepository.findByUsernameAndPassword(username, password);  
 }  
}  
// UserRepository.java  
public interface UserRepository {  
 User findByUsernameAndPassword(String username, String password);  
}

在这个示例中，我们首先在LoginController中定义了一个login方法，这个方法接收用户名和密码作为参数。然后，我们调用UserService的findByUsernameAndPassword方法，这个方法会查询数据库，查找用户名和密码匹配的用户。如果找到了匹配的用户，我们就使用Sa-Token的login方法进行身份验证，然后重定向到主页。如果没有找到匹配的用户，我们就重定向到登录页。

### 音乐推荐

其类图与交互图如下：





实现步骤如下：

1. 用户在搜索界面输入音乐名称并点击搜索按钮。这个过程涉及到前端的JavaScript和HTML技术，可能使用到的库包括React或Vue.js来构建用户界面。
2. 搜索界面将请求发送到搜索控制器。这个过程涉及到Spring MVC的控制器和视图模型，使用HTTP协议传输用户的请求。
3. 搜索控制器在音乐数据库中搜索音乐。这个过程涉及到REST API的使用，通过REST API，我们可以从网易云的API获取音乐数据，可能使用到的技术包括JSON数据格式和HTTP请求方法。
4. 如果找到了目标音乐，音乐数据库将目标音乐返回给搜索控制器，搜索控制器回答请求，搜索界面让用户听音乐。这个过程涉及到Spring MVC的数据绑定和视图渲染，可能使用到的技术包括Thymeleaf或JSP。
5. 如果没有找到目标音乐，音乐数据库返回None，搜索控制器在搜索界面显示结果消息框。这个过程涉及到Spring MVC的异常处理和消息提示，可能使用到的技术包括Spring的错误处理机制和Bootstrap的消息提示组件。
6. 用户在主页界面点击心情打卡按钮并选择心情。这个过程涉及到前端的JavaScript和HTML技术，可能使用到的库包括React或Vue.js来构建用户界面。
7. 主页界面将数据发送到主页控制器。这个过程涉及到Spring MVC的控制器和视图模型，使用HTTP协议传输用户的请求。
8. 主页控制器获取音乐推荐器推荐的音乐。这个过程涉及到机器学习算法，用于根据用户的心情推荐音乐，可能使用到的技术包括Python的Scikit-learn库或TensorFlow库。
9. 音乐推荐器从音乐数据库获取音乐，音乐数据库返回音乐给音乐推荐器。这个过程涉及到REST API的使用，通过REST API，我们可以从网易云的API获取音乐数据，可能使用到的技术包括JSON数据格式和HTTP请求方法。
10. 音乐推荐器在主页界面显示推荐的音乐。这个过程涉及到Spring MVC的数据绑定和视图渲染，可能使用到的技术包括Thymeleaf或JSP。

其对应的伪代码如下：

// 用户在搜索界面输入音乐名称并点击搜索按钮  
String musicName = userInput();  
// 搜索界面将请求发送到搜索控制器  
SearchController searchController = new SearchController();  
// 搜索控制器在音乐数据库中搜索音乐  
Music music = searchController.searchMusic(musicName);  
// 如果找到了目标音乐  
if (music != null) {  
 // 搜索控制器回答请求，搜索界面让用户听音乐  
 playMusic(music);  
} else {  
 // 搜索控制器在搜索界面显示结果消息框  
 showMessage("Music not found.");  
}  
// 用户在主页界面点击心情打卡按钮并选择心情  
String mood = userInput();  
// 主页界面将数据发送到主页控制器  
HomeController homeController = new HomeController();  
// 主页控制器获取音乐推荐器推荐的音乐  
Music recommendedMusic = homeController.getRecommendedMusic(mood);  
// 音乐推荐器在主页界面显示推荐的音乐  
showRecommendedMusic(recommendedMusic);