文档说明

数据库表设计是自己设计的，然后用ai生成相应的代码。

用户行为记录功能说明文档

一、功能概述

目标：在不影响原有业务的前提下，后台统一记录关键用户行为（注册、登录、修改资料/密码/头像、播放歌曲等），便于后期统计分析、问题排查和审计。
实现方式：新增 `user_action_log` 表及对应 Entity/Service，并在相关业务 Service 中埋点记录。

二、数据表设计：`user_action_log`

表名：`user_action_log`

字段说明：

- **id**：`bigint`，主键，自增
- **user_id**：`int`，用户 ID（`consumer.id`）
- **action**：`varchar(64)`，动作类型编码，例如：
  - `REGISTER`
  - `LOGIN`
  - `UPDATE_PROFILE`
  - `UPDATE_PASSWORD`
  - `UPDATE_AVATAR`
  - `PLAY_SONG`
- **detail**：`varchar(255)`，动作详情文本（如“用户名登录成功”“播放歌曲：林俊杰-关键词(38)”）
- **ip**：`varchar(64)`，请求来源 IP
  - 本地开发环境常见值为 `0:0:0:0:0:0:0:1`（IPv6 回环，等价 `127.0.0.1`）
- **create_time**：`datetime`，创建时间（由 MyBatis-Plus 自动填充）
- **update_time**：`datetime`，更新时间（由 MyBatis-Plus 自动填充）

表结构在 `tp_music.sql` 中新增，导入 SQL 或执行迁移后自动创建。

三、核心代码结构

3.1 实体类 `UserActionLog`

- 位置：`model/domain/UserActionLog.java`
- 作用：映射 `user_action_log` 表，配合 MyBatis-Plus 的自动填充。
  关键字段：

```
@TableName("user_action_log")
public class UserActionLog {
    @TableId(type = IdType.AUTO)
    private Long id;
    private Integer userId;
    private String action;
    private String detail;
    private String ip;
    @TableField(fill = FieldFill.INSERT)
    private Date createTime;
    @TableField(fill = FieldFill.INSERT_UPDATE)
    private Date updateTime;
}
```

> `createTime` / `updateTime` 的填充由已有的 `MyMetaObjectHandler` 统一处理。

3.2 Mapper & Service

- **Mapper**：`mapper/UserActionLogMapper.java`
  继承 `BaseMapper<UserActionLog>`，无需额外方法。
- **Service 接口**：`service/UserActionLogService.java`

```
public interface UserActionLogService extends
IService<UserActionLog> {
    void recordAction(Integer userId, String action, String
detail, String ip);
}
```

- **Service 实现**：`service/impl/UserActionLogServiceImpl.java`
  封装一个简单的写库入口，空参数自动忽略：

```
    public void recordAction(Integer userId, String action, String
    detail, String ip) {
        if (userId == null || StringUtils.isBlank(action)) {
            return;
        }
        UserActionLog log = new UserActionLog();
        log.setUserId(userId);
        log.setAction(action);
        log.setDetail(detail);
        log.setIp(StringUtils.defaultIfBlank(ip, "UNKNOWN"));
        this.save(log);
    }
```

四、IP 获取工具：`IpUtils`

- 位置：`utils/IpUtils.java`
- 功能：从常见代理头（`X-Forwarded-For`、`X-Real-IP` 等）中解析客户端 IP，没有则回退 `request.getRemoteAddr()`。

```
public static String getClientIp() {
    RequestAttributes attributes =
RequestContextHolder.getRequestAttributes();
    if (!(attributes instanceof ServletRequestAttributes)) {
        return "UNKNOWN";
    }
    HttpServletRequest request = ((ServletRequestAttributes)
attributes).getRequest();
    for (String header : IP_HEADER_CANDIDATES) {
        String ipList = request.getHeader(header);
        if (StringUtils.isNotBlank(ipList) &&
!"unknown".equalsIgnoreCase(ipList)) {
            return extractFirstIp(ipList);
        }
    }
    String remoteAddr = request.getRemoteAddr();
    return StringUtils.defaultIfBlank(remoteAddr, "UNKNOWN");
}
```

> 在本地开发（浏览器直连本机后端）时，通常拿到 `0:0:0:0:0:0:0:1`。

五、埋点位置与行为说明

5.1 用户注册 / 信息修改 / 密码修改 / 头像修改

- 位置：`service/impl/ConsumerServiceImpl.java`
- 注入：

```
@Autowired
private UserActionLogService userActionLogService;

private static final String ACTION_REGISTER = "REGISTER";
private static final String ACTION_LOGIN = "LOGIN";
private static final String ACTION_UPDATE_PROFILE =
"UPDATE_PROFILE";
private static final String ACTION_UPDATE_PASSWORD =
"UPDATE_PASSWORD";
private static final String ACTION_UPDATE_AVATAR = "UPDATE_AVATAR";
```

- 注册成功：

```
if (consumerMapper.insert(consumer) > 0) {
    recordUserAction(consumer.getId(), ACTION_REGISTER, "用户注册成
功");
    return R.success("注册成功");
}
```

- 更新用户信息成功：

```
if (consumerMapper.updateById(consumer) > 0) {
    recordUserAction(consumer.getId(), ACTION_UPDATE_PROFILE, "更新
用户基本信息");
    return R.success("修改成功");
}
```

- 修改密码 / 邮件重置密码成功时写入 `UPDATE_PASSWORD`，更新头像成功时写入 `UPDATE_AVATAR`，逻辑类似。

- 统一记录方法：

```
private void recordUserAction(Integer userId, String action, String
detail) {
    if (userId == null) {
        return;
    }
    String ip = IpUtils.getClientIp();
    userActionLogService.recordAction(userId, action, detail, ip);
}
```

5.2 用户登录（用户名 / 邮箱）

- 用户名登录成功：

```
if (this.verityPasswd(username, password)) {
    session.setAttribute("username", username);
    QueryWrapper<Consumer> queryWrapper = new QueryWrapper<>();
    queryWrapper.eq("username", username);
    List<Consumer> consumerList =
consumerMapper.selectList(queryWrapper);
    if (!consumerList.isEmpty()) {
        recordUserAction(consumerList.get(0).getId(), ACTION_LOGIN,
"用户名登录成功");
    }
    return R.success("登录成功", consumerList);
}
```

- 邮箱登录成功：

```
if (this.verityPasswd(consumer1.getUsername(), password)) {
    session.setAttribute("username", consumer1.getUsername());
    Consumer consumer = new Consumer();
    consumer.setUsername(consumer1.getUsername());
    List<Consumer> consumerList = consumerMapper.selectList(new
QueryWrapper<>(consumer));
    if (!consumerList.isEmpty()) {
        recordUserAction(consumerList.get(0).getId(), ACTION_LOGIN,
"邮箱登录成功");
    }
    return R.success("登录成功", consumerList);
}
```

5.3 播放歌曲行为

5.3.1 控制层接口

- 位置：`controller/SongController.java`
- 新增接口：`POST /song/play`

```
@PostMapping("/song/play")
public R recordSongPlay(@RequestBody SongPlayRequest playRequest,
HttpSession session) {
    return songService.recordSongPlay(playRequest.getSongId(),
playRequest.getUserId(),
            session != null ? (String)
session.getAttribute("username") : null);
}
```

- 请求体 DTO：`SongPlayRequest`（`songId` 必填，`userId` 可选）

```
@Data
public class SongPlayRequest {
    private Integer songId;
    private Integer userId;
}
```

5.3.2 Service 逻辑

- 接口定义：`SongService.recordSongPlay(Integer songId, Integer userId, String sessionUsername)`
- 实现位置：`service/impl/SongServiceImpl.java`

```
public R recordSongPlay(Integer songId, Integer userId, String
sessionUsername) {
    if (songId == null) {
        return R.error("歌曲ID不能为空");
    }
    Song song = songMapper.selectById(songId);
    if (song == null) {
        return R.error("歌曲不存在");
    }
    Integer resolvedUserId = resolveUserId(userId, sessionUsername);
    if (resolvedUserId != null) {
        String detail = String.format("播放歌曲：%s(%d)",
song.getName(), songId);
        userActionLogService.recordAction(resolvedUserId,
ACTION_PLAY_SONG, detail, IpUtils.getClientIp());
    }
    return R.success("播放记录成功");
}
```

- 用户 ID 解析策略（匿名用户处理）：

```
private Integer resolveUserId(Integer userId, String
sessionUsername) {
    if (userId != null) {
        return userId;
    }
    if (StringUtils.isBlank(sessionUsername)) {
        return null;
    }
    QueryWrapper<Consumer> queryWrapper = new QueryWrapper<>();
    queryWrapper.eq("username", sessionUsername);
    Consumer consumer = consumerMapper.selectOne(queryWrapper);
    return consumer != null ? consumer.getId() : null;
}
```

> 说明：
>
> - 若前端传了 `userId`，直接使用；
> - 否则尝试从 session 中的用户名查找对应 `consumer`；
> - 若仍拿不到用户，则 不写入日志，避免大量匿名噪声数据。

六、前端/接口使用说明（概要）

1. 登录/注册/修改资料/密码/头像
   - 调用原有接口（`/user/add`、`/user/login/status`、`/user/update` 等）逻辑不变。
   - 后端在成功分支自动写入 `user_action_log`，前端无需额外改动。
2. 播放行为记录
   - 新增调用：在播放器开始播放时，调用：
     - URL：`POST /song/play`
     - Body 示例：

       ```
       { "songId": 38, "userId": 5 }
       ```

     - 若前端已维护登录用户 ID，建议总是传 `userId`，解析更直接。
3. 日志查看
   - 在 MySQL 中查询：

     ```sql
     SELECT * FROM user_action_log ORDER BY create_time DESC;
     ```

   - 可按 `user_id`、`action`、`create_time` 进行筛选统计。

七、行为与本地环境说明

- 本地开发时，IP 记录为 `0:0:0:0:0:0:0:1` 是正常现象，表示 IPv6 回环地址。
- 上线后在 nginx / 网关正确设置 `X-Forwarded-For`、`X-Real-IP` 等头部，`IpUtils` 会优先解析真实客户端 IP。

如你之后增加新的用户行为（比如“搜索歌曲”“收藏/取消收藏”等），可以直接在对应 Service 中注入 `UserActionLogService`，调用 `recordAction(userId, "ACTION_CODE", "描述...", IpUtils.getClientIp())` 即可复用现有能力。

搜索历史和收藏操作记录功能说明

功能概述

实现搜索历史和收藏操作的记录功能，具体包括：

1. 搜索历史记录

- 数据库表：`search_history`
  - 字段包括：id, user_id, keyword, search_type, create_time
  - 支持记录用户搜索的关键词和搜索类型（歌曲/歌单）
- 后端实现：
  - `SearchHistory` 实体类
  - `SearchHistoryMapper` 数据访问层
  - `SearchHistoryService` 服务层
  - `SearchHistoryController` 控制器
  - 在 `SongController` 和 `SongListController` 中自动记录搜索历史
- 前端实现：
  - 在 `SearchSong.vue` 和 `SearchSongList.vue` 中传递用户ID
  - 更新了 `api/index.ts` 中的搜索API调用

2. 收藏操作记录

- 使用现有表：`user_action_log`
  - 记录收藏和取消收藏操作
  - 包含操作类型、详细信息、用户IP等
- 后端实现：
  - 在 `CollectServiceImpl` 中添加收藏和取消收藏的操作日志记录
  - 使用 `UserActionLogService` 记录操作

数据库表创建

请执行以下SQL创建搜索历史表：

```sql
-- 搜索历史表
DROP TABLE IF EXISTS `search_history`;
CREATE TABLE `search_history` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `user_id` int(10) unsigned DEFAULT NULL COMMENT '用户ID，可为空表示未登录用户',
  `keyword` varchar(255) NOT NULL COMMENT '搜索关键词',
  `search_type` varchar(20) DEFAULT 'song' COMMENT '搜索类型：song-歌曲，songList-歌单',
  `create_time` datetime NOT NULL COMMENT '搜索时间',
  PRIMARY KEY (`id`),
  KEY `idx_user_id` (`user_id`),
  KEY `idx_create_time` (`create_time`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='搜索历史表';
```

SQL文件位置：`music-server/sql/search_history.sql`

API接口

搜索历史相关

1. 记录搜索历史
   - 路径：`POST /search/history/record`
   - 参数：userId（可选），keyword, searchType（可选，默认song）
2. 获取用户搜索历史
   - 路径：`GET /search/history/user`
   - 参数：userId, limit（可选，默认10）

收藏操作

收藏操作会自动记录到 `user_action_log` 表中，包括：

- ADD_COLLECT：添加收藏
- DELETE_COLLECT：取消收藏

文件清单

后端文件

1. `music-server/src/main/java/com/example/yin/model/domain/SearchHistory.java`
2. `music-server/src/main/java/com/example/yin/mapper/SearchHistoryMapper.java`

3. `music-server/src/main/java/com/example/yin/service/SearchHistoryService.java`
4. `music-server/src/main/java/com/example/yin/service/impl/SearchHistoryServiceImpl.java`
5. `music-server/src/main/java/com/example/yin/controller/SearchHistoryController.java`
6. `music-server/src/main/java/com/example/yin/controller/SongController.java` (已修改)
7. `music-server/src/main/java/com/example/yin/controller/SongListController.java` (已修改)
8. `music-server/src/main/java/com/example/yin/service/impl/CollectServiceImpl.java` (已修改)
9. `music-server/sql/search_history.sql`

前端文件

1. `music-client/src/api/index.ts` (已修改)
2. `music-client/src/views/search/SearchSong.vue` (已修改)
3. `music-client/src/views/search/SearchSongList.vue` (已修改)

使用说明

1. 执行数据库脚本：运行 `music-server/sql/search_history.sql` 创建搜索历史表
2. 搜索历史记录：
   - 用户在搜索时，如果已登录，系统会自动记录搜索关键词
   - 搜索类型会自动识别（歌曲搜索或歌单搜索）
3. 收藏操作记录：
   - 当用户添加或取消收藏时，系统会自动记录操作日志
   - 日志包含操作类型、详细信息、用户IP等
4. 查询搜索历史：
   - 可以通过API接口查询用户的搜索历史
   - 支持限制返回数量

注意事项

1. 搜索历史记录支持未登录用户（user_id可为null）
2. 收藏操作记录会自动获取用户IP地址
3. 所有操作日志记录是异步的，不会影响主要业务逻辑的性能