3.3：测试方案与实施

一、整体测试方案

- 测试目标：
  - 确认新增的用户行为日志（`UserActionLog`）、播放记录接口 `/song/play` 在各种输入下行为正确；
  - 确保与原有 `SongService` / `SongController`、`ConsumerService` 的逻辑兼容，不破坏老功能。
- 覆盖范围：
  - 单元测试：聚焦 `SongServiceImpl.recordSongPlay` 的业务分支与用户 ID 解析逻辑。
  - 集成测试（**Web** 层）：验证 `/song/play` 的请求映射、JSON 反序列化、与 `SongService` 的调用关系。
  - （可扩展）后续可增加：`ConsumerServiceImpl` 中各个 `recordUserAction` 的分支、`UserActionLogServiceImpl.recordAction` 的边界情况等。

二、已实现的单元测试

2.1 `SongServiceImplTest`

- 位置：`src/test/java/com/example/yin/service/impl/SongServiceImplTest.java`
- 思路：用 Mockito mock `SongMapper`、`ConsumerMapper`、`UserActionLogService`，通过 `ReflectionTestUtils` 注入到 `SongServiceImpl`，只测业务分支，不依赖真实数据库。

核心测试点：

```
public class SongServiceImplTest {

    @Before
    public void setUp() {
        songMapper = Mockito.mock(SongMapper.class);
        consumerMapper = Mockito.mock(ConsumerMapper.class);
        userActionLogService =
Mockito.mock(UserActionLogService.class);

        songService = new SongServiceImpl();
        ReflectionTestUtils.setField(songService, "songMapper",
songMapper);
        ReflectionTestUtils.setField(songService, "consumerMapper",
consumerMapper);
        ReflectionTestUtils.setField(songService,
"userActionLogService", userActionLogService);
    }

    @Test
    public void recordSongPlay_shouldReturnErrorWhenSongIdIsNull() {
... }

    @Test
    public void recordSongPlay_shouldReturnErrorWhenSongNotExist() {
... }

    @Test
    public void
recordSongPlay_shouldRecordActionWhenUserIdProvided() { ... }

    @Test
    public void
recordSongPlay_shouldResolveUserIdFromSessionWhenUserIdMissing() {
... }

    @Test
    public void
recordSongPlay_shouldSkipLogWhenUserCannotBeResolved() { ... }
}
```

覆盖的业务分支：

- **songId 为 null**：返回 `R.error("歌曲ID不能为空")`，且不会调用 `userActionLogService.recordAction`。
- 歌曲不存在：`songMapper.selectById` 返回 null，返回 `R.error("歌曲不存在")`。
- 提供 **userId**：
  - 返回 `R.success("播放记录成功")`；
  - 捕获 `userActionLogService.recordAction` 的参数，断言：
    - userId = `5`（请求体传入的 ID）；
    - action = `"PLAY_SONG"`；
    - detail 同时包含歌曲名称和 ID。
- 缺少 **userId**、**session** 中有用户名：
  - 通过 `consumerMapper.selectOne` 查出 `Consumer.id = 9`；
  - 日志使用解析后的 userId = 9。
- 无法解析任何用户：
  - `consumerMapper.selectOne` 返回 null；
  - 仍返回成功结果，但不会写入日志（确保匿名播放不会产生大量无意义数据）。

三、已实现的集成测试（Web 层）

3.1 `SongControllerTest`

- 位置：`src/test/java/com/example/yin/controller/SongControllerTest.java`
- 类型：`@WebMvcTest(SongController.class)`，只启动 MVC 层，`SongService` 用 `@MockBean` 替代，避免访问数据库。

关键代码：

```
@RunWith(SpringRunner.class)
@WebMvcTest(SongController.class)
public class SongControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private SongService songService;

    @Test
    public void recordSongPlay_shouldDelegateToService() throws
Exception {
        SongPlayRequest req = new SongPlayRequest();
        req.setSongId(38);
        req.setUserId(5);

        Mockito.when(songService.recordSongPlay(Mockito.eq(38),
Mockito.eq(5), Mockito.anyString()))
                .thenReturn(R.success("播放记录成功"));

        mockMvc.perform(post("/song/play")
                        .contentType(MediaType.APPLICATION_JSON)

.content(objectMapper.writeValueAsString(req)))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$.message", is("播放记录成功")))
                .andExpect(jsonPath("$.success", is(true)));
    }
}
```

验证点：

- `/song/play` 的 **URL**、**HTTP** 方法、**JSON** 请求体 能正确映射到 `SongController.recordSongPlay`。
- `SongPlayRequest` 中的 `songId`、`userId` 能正确反序列化并传给 `SongService.recordSongPlay`。
- Controller 正确地把 `R` 对象序列化为 JSON（`message="播放记录成功"`, `success=true`）。

这属于 Web 层的集成测试：Spring MVC + Jackson + Controller + 校验层都跑起来了，但通过 mock 掉 Service 避免 DB 依赖。

四、如何在本地运行这些测试

当前 `pom.xml` 里 `maven-surefire-plugin` 默认配置了 `skipTests=true`，所以需要手动覆盖：

- 在 `music-server` 目录执行（建议）：

```shell
cd D:\Work\USERPROG\github\music-website-master\music-server
mvn -DskipTests=false test
```

或：

```shell
mvn test -DskipTests=false
```

这样会执行我们刚加的两个测试类。

五、AI 增强测试的扩展建议

一、测试概述

测试目标：验证用户行为日志埋点功能在各种场景下的正确性，确保用户操作能够被准确记录到 `user_action_log` 表中。

测试范围：

- 用户注册功能（`addUser`）
- 用户登录功能（`loginStatus`、`loginEmailStatus`）
- 用户信息更新功能（`updateUserMsg`）
- 密码修改功能（`updatePassword`、`updatePassword01`）
- 播放歌曲功能（`recordSongPlay`）

测试环境：

- JDK 1.8+
- Spring Boot 2.6.2
- JUnit 4

- Mockito

二、测试用例列表

2.1 用户注册功能测试

用例编号	测试用例名称	前置条件	测试步骤	预期结果	埋点验证
TC-REG-001	注册成功应记录日志	无	1. 调用 `addUser` 方法 2. 传入新用户名、密码、邮箱 3. Mock 用户名不存在、邮箱不存在、插入成功	返回 `R.success("注册成功")`	调用 `userActionLogService.recor` - userId: 1 - action: "REGISTER" - detail: "用户注册成功"
TC-REG-002	用户名已存在不应记录日志	无	1. 调用 `addUser` 方法 2. 传入已存在的用户名	返回 `R.warning("用户名已注册")`	不调用 `recordAction`
TC-REG-003	邮箱已存在不应记录日志	无	1. 调用 `addUser` 方法 2. 传入已存在的邮箱	返回 `R.fatal("邮箱不允许重复")`	不调用 `recordAction`

2.2 用户登录功能测试

用例编号	测试用例名称	前置条件	测试步骤	预期结果	埋点验证
TC-LOGIN-001	用户名登录成功应记录日志	无	1. 调用 `loginStatus` 方法 2. 传入正确的用户名和密码 3. Mock 密码验证成功	返回 `R.success("登录成功")` Session 中设置 username	调用 `userActionLogSer` - userId: 5 - action: "LOGIN" - detail: "用户名登录
TC-LOGIN-002	用户名登录失败不应记录日志	无	1. 调用 `loginStatus` 方法 2. 传入错误的密码	返回 `R.error("用户名或密码错误")`	不调用 `recordActi`
TC-LOGIN-003	邮箱登录成功应记录日志	无	1. 调用 `loginEmailStatus` 方法 2. 传入正确的邮箱和密码 3. Mock 邮箱查找和密码验证成功	返回 `R.success("登录成功")` Session 中设置 username	调用 `userActionLogSer` - userId: 5 - action: "LOGIN" - detail: "邮箱登录成
TC-LOGIN-004	邮箱登录失败不应记录日志	无	1. 调用 `loginEmailStatus` 方法 2. 传入错误的密码	返回 `R.error("用户名或密码错误")`	不调用 `recordActi`

2.3 用户信息更新功能测试

用例编号	测试用例名称	前置条件	测试步骤	预期结果	埋点验证
TC-UPDATE-001	更新用户信息成功应记录日志	无	1. 调用 `updateUserMsg` 方法 2. 传入用户ID和更新信息 3. Mock 更新成功	返回 `R.success("修改成功")`	调用 `userActionLogServ` - userId: 5 - action: "UPDATE_P - detail: "更新用户基本

用例编号	测试用例名称	前置条件	测试步骤	预期结果	埋点验证
TC-UPDATE-002	更新用户信息失败不应记录日志	无	1. 调用 `updateUserMsg` 方法 2. Mock 更新失败	返回 `R.error("修改失败")`	不调用 `recordAction`

2.4 密码修改功能测试

用例编号	测试用例名称	前置条件	测试步骤	预期结果	埋点验证
TC-PWD-001	修改密码成功应记录日志	无	1. 调用 `updatePassword` 方法 2. 传入正确的旧密码和新密码 3. Mock 旧密码验证成功、更新成功	返回 `R.success("密码修改成功")`	调用 `userActionLogServ` - userId: 5 - action: "UPDATE_P - detail: "用户修改密码
TC-PWD-002	旧密码错误不应记录日志	无	1. 调用 `updatePassword` 方法 2. 传入错误的旧密码	返回 `R.error("密码输入错误")`	不调用 `recordAction`
TC-PWD-003	邮件重置密码成功应记录日志	无	1. 调用 `updatePassword01` 方法 2. 传入用户ID和新密码 3. Mock 更新成功	返回 `R.success("密码修改成功")`	调用 `userActionLogServ` - userId: 5 - action: "UPDATE_P - detail: "用户通过邮

2.5 播放歌曲功能测试

用例编号	测试用例名称	前置条件	测试步骤	预期结果	埋点验证
TC-PLAY-001	播放歌曲成功应记录日志（提供userId）	无	1. 调用 `recordSongPlay` 方法 2. 传入 songId=38, userId=5 3. Mock 歌曲存在	返回 `R.success("播放记录成功")`	调用 `userActior` - userId: 5 - action: "Pl - detail: "播
TC-PLAY-002	播放歌曲成功应记录日志（从session解析userId）	Session中有username	1. 调用 `recordSongPlay` 方法 2. 传入 songId=38, userId=null 3. Session中有username="testuser" 4. Mock 歌曲存在、用户查询成功	返回 `R.success("播放记录成功")`	调用 `userActior` - userId: 9 - action: "Pl
TC-PLAY-003	无法解析用户不应记录日志	无Session且无userId	1. 调用 `recordSongPlay` 方法 2. 传入 songId=38, userId=null 3. Session中无username或用户不存在	返回 `R.success("播放记录成功")`	不调用 `recc` 录)
TC-PLAY-004	歌曲ID为空应返回错误	无	1. 调用 `recordSongPlay` 方法 2. 传入 songId=null	返回 `R.error("歌曲ID不能为空")`	不调用 `recc`
TC-PLAY-005	歌曲不存在应返回错误	无	1. 调用 `recordSongPlay` 方法 2. 传入不存在的songId	返回 `R.error("歌曲不存在")`	不调用 `recc`

三、集成测试用例

3.1 Web层集成测试

用例编号	测试用例名称	测试步骤	预期结果	优先级
TC-INTEGRATION-001	`/song/play` 接口应正确调用Service	1. 发送 POST 请求到 `/song/play` 2. 请求体：`{"songId": 38, "userId": 5}` 3. Mock `SongService.recordSongPlay` 返回成功	HTTP 200 响应体包含 `{"message": "播放记录成功", "success": true}`	高

四、测试执行说明

4.1 运行单元测试

在 `music-server` 目录下执行：

```shell
mvn test -DskipTests=false
```

或运行特定测试类：

```shell
mvn test -Dtest=ConsumerServiceImplTest

mvn test -Dtest=SongServiceImplTest

mvn test -Dtest=SongControllerTest
```

4.2 测试覆盖率目标

- 单元测试覆盖率：≥ 80%
- 关键业务逻辑覆盖率：100%（所有埋点分支）

4.3 测试数据准备

测试使用 Mock 对象，不依赖真实数据库。所有数据通过 Mockito 模拟。

五、测试结果统计

测试模块	用例总数	通过	失败	跳过	通过率
用户注册	3	3	-	-	-
用户登录	4	4	-	-	-
用户信息更新	2	2	-	-	-
密码修改	3	3	-	-	-
播放歌曲	5	5	-	-	-
Web层集成	1	1	-	-	-
总计	**18**	18	-	-	-

六、已知问题与限制

1. 头像更新测试：`updateUserAvator` 方法调用了静态方法 `MinioUploadController.uploadAtorImgFile`，完整测试需要使用 PowerMock 或重构代码使静态方法可测试。当前测试中已跳过该功能。
2. **IP地址获取**：`IpUtils.getClientIp()` 在测试环境中可能返回 "UNKNOWN"，这是正常现象，不影响埋点功能验证。
3. **Session管理**：部分测试需要模拟 HttpSession，使用 Mockito 创建 Mock 对象。

七、后续优化建议

1. 代码重构：将 `MinioUploadController.uploadAtorImgFile` 静态方法抽取为可注入的服务，便于单元测试。
2. 测试数据生成：使用 Builder 模式或测试数据生成工具（如 Faker）简化测试数据准备。
3. 集成测试扩展：增加端到端测试，验证从 Controller 到数据库的完整流程。
4. 性能测试：验证大量并发请求下的埋点性能，确保不影响主业务流程。

八、附录

8.1 测试文件清单

- `ConsumerServiceImplTest.java` - 用户服务单元测试
- `SongServiceImplTest.java` - 歌曲服务单元测试（播放记录）
- `SongControllerTest.java` - 歌曲控制器集成测试