**上架商品→用户预订→商家查看意向购买人→确认交易→查看历史**

**1. 依赖配置（pom.xml 新增）**

<dependency>

<groupId>org.seleniumhq.selenium</groupId>

<artifactId>selenium-java</artifactId>

<version>4.15.0</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

**2. 测试代码**

**package** soft\_practice01;

**import** **static** org.junit.Assert.\*;

**import** org.junit.\*;

**import** org.springframework.mock.web.MockHttpServletRequest;

**import** org.springframework.mock.web.MockHttpServletResponse;

**import** org.springframework.mock.web.MockHttpSession;

**import** datebase.\*;

**import** Servlet.\*;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** javax.servlet.http.HttpServlet;

**import** java.sql.Connection;

**import** java.util.List;

**import** java.lang.reflect.Method;

**import** java.util.UUID;

**public** **class** Function\_Test {

**private** **static** Connection *con*;

**private** WorksDaoIplm worksDao;

**private** BuyerDaoimpl buyerDao;

// Servlet实例

**private** insert\_goodServlet insertGoodServlet = **new** insert\_goodServlet();

**private** Buyer\_message\_servlet buyerMessageServlet = **new** Buyer\_message\_servlet();

**private** BuyServlet buyServlet = **new** BuyServlet();

**private** modify\_goodStatusServlet modifyGoodServlet = **new** modify\_goodStatusServlet();

**private** HistoryProductsServlet historyServlet = **new** HistoryProductsServlet();

// 测试数据 - 使用UUID确保唯一性

**private** String testOrderId;

**private** **int** testWorkId;

**private** **final** String uniqueProductName = "流程测试商品\_" + UUID.*randomUUID*().toString().substring(0, 8);

@BeforeClass

**public** **static** **void** setUpBeforeClass() **throws** Exception {

// 初始化数据库连接并关闭自动提交（便于回滚）

*con* = DbUtil.*getCon*();

**if** (*con* == **null**) {

*fail*("数据库连接初始化失败");

}

*con*.setAutoCommit(**false**);

}

@AfterClass

**public** **static** **void** tearDownAfterClass() **throws** Exception {

// 测试结束后回滚并关闭连接

**if** (*con* != **null**) {

**try** {

*con*.rollback();

} **catch** (Exception e) {

e.printStackTrace();

} **finally** {

*con*.close();

}

}

}

@Before

**public** **void** setUp() **throws** Exception {

worksDao = **new** WorksDaoIplm();

buyerDao = **new** BuyerDaoimpl();

}

@After

**public** **void** tearDown() **throws** Exception {

// 每个测试方法结束后回滚，避免数据污染

**if** (*con* != **null**) {

*con*.rollback();

}

}

/\*\*

\* 完整业务流程测试：上架→预订→查看购买人→确认交易→查看历史

\*/

@Test

**public** **void** testCompleteTradeFlow() **throws** Exception {

// 1. 测试商品上架

testWorkId = testAddNewProductSuccess();

*assertTrue*("商品上架失败", testWorkId > 0);

// 2. 测试用户预订

testOrderId = testUserReservationSuccess();

*assertNotNull*("用户预订失败", testOrderId);

*assertTrue*("订单ID格式不正确", testOrderId.matches("^[A-Za-z]{2}\\d{5}$"));

// 3. 测试查看意向购买人

Buyer selectedBuyer = testViewInterestedBuyers();

*assertNotNull*("未查询到意向购买人", selectedBuyer);

*assertEquals*("购买人信息不匹配", "测试", selectedBuyer.getBuyer\_name());

// 4. 测试确认交易

**boolean** tradeSuccess = testConfirmTrade();

*assertTrue*("交易确认失败", tradeSuccess);

// 5. 测试查看历史记录

**boolean** historyExists = testCheckHistory();

*assertTrue*("历史记录未找到", historyExists);

}

/\*\*

\* 测试商品上架功能（正常场景）

\*/

**private** **int** testAddNewProductSuccess() **throws** Exception {

MockHttpServletRequest request = **new** MockHttpServletRequest();

MockHttpServletResponse response = **new** MockHttpServletResponse();

MockHttpSession session = **new** MockHttpSession();

request.setSession(session);

// 设置商品参数（使用唯一名称避免冲突）

request.setParameter("work\_status", "available");

request.setParameter("work\_name", uniqueProductName);

request.setParameter("work\_description", "用于完整流程测试的商品");

request.setParameter("work\_price", "299.99");

request.setParameter("work\_image", "flow\_test.jpg");

// 调用插入Servlet

callDoGet(insertGoodServlet, request, response);

// 验证响应状态

*assertEquals*("Servlet响应状态异常", 302, response.getStatus());

// 验证结果

List<Works> workList = (List<Works>) session.getAttribute("wklist");

*assertNotNull*("商品列表为空", workList);

**for** (Works work : workList) {

**if** (uniqueProductName.equals(work.getWork\_name())) {

**return** work.getId();

}

}

*fail*("未找到新上架的商品");

**return** -1;

}

/\*\*

\* 测试用户预订功能（正常场景）

\*/

**private** String testUserReservationSuccess() **throws** Exception {

MockHttpServletRequest request = **new** MockHttpServletRequest();

MockHttpServletResponse response = **new** MockHttpServletResponse();

request.setMethod("POST");

// 设置预订参数

request.setParameter("buyer\_name", "测试");

request.setParameter("buyer\_phonenumber", "13800138000");

request.setParameter("trading\_address", "测试地址123号");

request.setParameter("trading\_time", "2024-12-31T16:00");

// 调用预订Servlet的doPost方法

Method doPostMethod = Buyer\_message\_servlet.**class**.getDeclaredMethod("doPost",

HttpServletRequest.**class**, HttpServletResponse.**class**);

doPostMethod.setAccessible(**true**);

doPostMethod.invoke(buyerMessageServlet, request, response);

// 验证无错误信息

*assertNull*("预订过程出现错误", request.getAttribute("errorMessage"));

// 获取订单ID

String orderId = (String) request.getAttribute("order\_id");

*assertNotNull*("订单ID为空", orderId);

**return** orderId;

}

/\*\*

\* 测试查看意向购买人功能

\*/

**private** Buyer testViewInterestedBuyers() **throws** Exception {

MockHttpServletRequest request = **new** MockHttpServletRequest();

MockHttpServletResponse response = **new** MockHttpServletResponse();

MockHttpSession session = **new** MockHttpSession();

request.setSession(session);

// 调用购买Servlet查询购买人

request.setParameter("method", "trade");

request.setParameter("orderId", testOrderId);

callDoGet(buyServlet, request, response);

// 验证无错误信息

*assertNull*("查询购买人过程出现错误", request.getAttribute("errorMsg"));

// 验证结果

Buyer selectedBuyer = (Buyer) session.getAttribute("selected\_buyer");

*assertNotNull*("未找到对应的购买人", selectedBuyer);

**return** selectedBuyer;

}

/\*\*

\* 测试确认交易功能

\*/

**private** **boolean** testConfirmTrade() **throws** Exception {

MockHttpServletRequest request = **new** MockHttpServletRequest();

MockHttpServletResponse response = **new** MockHttpServletResponse();

MockHttpSession session = **new** MockHttpSession();

// 准备商品列表数据

List<Works> workList = worksDao.serachAll(*con*);

*assertNotNull*("商品列表查询失败", workList);

session.setAttribute("wklist", workList);

request.setSession(session);

// 设置修改参数（状态改为已售）

request.setParameter("work\_id", String.*valueOf*(testWorkId));

request.setParameter("work\_status1", "sold");

request.setParameter("work\_name1", uniqueProductName);

request.setParameter("work\_description1", "用于完整流程测试的商品");

request.setParameter("work\_price1", "299.99");

request.setParameter("work\_image1", "flow\_test.jpg");

// 调用修改状态Servlet

callDoGet(modifyGoodServlet, request, response);

// 验证响应状态

*assertEquals*("修改状态Servlet响应异常", 302, response.getStatus());

// 验证状态更新

List<Works> updatedList = (List<Works>) session.getAttribute("wklist");

*assertNotNull*("更新后的商品列表为空", updatedList);

**for** (Works work : updatedList) {

**if** (work.getId() == testWorkId) {

**return** "sold".equals(work.getWork\_status());

}

}

*fail*("未找到目标商品或状态未更新");

**return** **false**;

}

/\*\*

\* 测试查看历史记录功能

\*/

**private** **boolean** testCheckHistory() **throws** Exception {

MockHttpServletRequest request = **new** MockHttpServletRequest();

MockHttpServletResponse response = **new** MockHttpServletResponse();

request.setParameter("search", testOrderId);

// 调用历史记录Servlet

callDoGet(historyServlet, request, response);

// 验证历史记录

List<Order> orders = (List<Order>) request.getAttribute("orders");

*assertNotNull*("历史记录列表为空", orders);

*assertFalse*("历史记录列表为空", orders.isEmpty());

**for** (Order order : orders) {

**if** (testOrderId.equals(order.getOrderNumber())) {

**return** **true**;

}

}

*fail*("历史记录中未找到目标订单");

**return** **false**;

}

/\*\*

\* 反射工具方法：调用Servlet的protected doGet方法

\*/

**private** **void** callDoGet(HttpServlet servlet, HttpServletRequest request, HttpServletResponse response) **throws** Exception {

Method doGetMethod = HttpServlet.**class**.getDeclaredMethod("doGet",

HttpServletRequest.**class**, HttpServletResponse.**class**);

doGetMethod.setAccessible(**true**);

doGetMethod.invoke(servlet, request, response);

}

}

1. **环境准备**：
   * 安装 JDK 21 及 Maven
   * 下载 ChromeDriver 并配置到环境变量
   * 启动应用服务器（如 Tomcat），确保应用可访问
2. **执行命令**：

bash

# 运行所有测试

mvn test

# 运行指定测试类

mvn test -Dtest=ProductTest

1. **报告查看**：测试报告生成路径：target/surefire-reports/index.html