自动化仓库管理系统

单元测试报告  
测试目标：

对于OutAndInServiceImpl类中的方法进行单元测试，确保它们能够正确地处理输入并返回预期的输出。

测试案例：

### **1、examineIn**

该方法中的所有逻辑分支和数据处理。

#### 测试路径：

1. 测试入库单不存在的情况，期望返回入库单不存在的错误信息。
2. 测试入库单存在但入库交接人不存在的情况，期望返回入库交接人不存在的错误信息。
3. 测试入库单存在且入库交接人存在的情况，期望更新入库单信息和包裹信息。
4. 测试入库单存在但没有可用的货架的情况，期望返回未找到可用货架的错误信息。

#### 测试代码如下：

import org.junit.jupiter.api.Test; import org.mockito.InjectMocks; import org.mockito.Mock; import org.mockito.MockitoAnnotations; import static org.mockito.Mockito.\*;

public class ExamineInTest {

@Mock

private InboundMapper inboundMapper;

@Mock

private WarehousepersonMapper warehousepersonMapper;

@InjectMocks

private ExamineInService examineInService;

@Test

public void testExamineInInboundNotExist() {

// 模拟入库单不存在的情况

when(inboundMapper.selectById(anyLong())).thenReturn(null);

// 调用被测试的方法

R result = examineInService.ExamineIn("123", new ExamineInParam());

// 验证返回结果是否符合预期

assertEquals("入库单不存在", result.getMsg());

}

@Test

public void testExamineInWarehousepersonNotExist() {

// 模拟入库单存在但入库交接人不存在的情况

Inbound inbound = new Inbound();

inbound.setInboundid(1L);

when(inboundMapper.selectById(anyLong())).thenReturn(inbound);

when(warehousepersonMapper.selectOne(any())).thenReturn(null);

// 调用被测试的方法

ExamineInParam param = new ExamineInParam();

param.setInID("1");

param.setInPeopleName("张三");

R result = examineInService.ExamineIn("123", param);

// 验证返回结果是否符合预期

assertEquals("入库交接人不存在", result.getMsg());

}

@Test

public void testExamineInSuccess() {

// 模拟入库单存在且入库交接人存在的情况

Inbound inbound = new Inbound();

inbound.setInboundid(1L);

when(inboundMapper.selectById(anyLong())).thenReturn(inbound);

when(warehousepersonMapper.selectOne(any())).thenReturn(new Warehouseperson());

when(examineInService.selectAvailableShelves()).thenReturn(new ArrayList<Shelf>());

// 调用被测试的方法

ExamineInParam param = new ExamineInParam();

param.setInID("1");

param.setInPeopleName("张三");

param.setInStatus("已入库");

R result = examineInService.ExamineIn("123", param);

// 验证返回结果是否符合预期

assertEquals("No available shelf", result.getMsg());

}

@Test

public void testExamineInNoAvailableShelf() {

// 模拟入库单存在但没有可用的货架的情况

Inbound inbound = new Inbound();

inbound.setInboundid(1L);

when(inboundMapper.selectById(anyLong())).thenReturn(inbound);

when(warehousepersonMapper.selectOne(any())).thenReturn(new Warehouseperson());

when(examineInService.selectAvailableShelves()).thenReturn(new ArrayList<Shelf>());

// 调用被测试的方法

ExamineInParam param = new ExamineInParam();

param.setInID("1");

param.setInPeopleName("张三");

param.setInStatus("已入库");

param.setParcelList(new ParcelList[]{new ParcelList()});

R result = examineInService.ExamineIn("123", param);

// 验证返回结果是否符合预期

assertEquals("No available shelf", result.getMsg());

}

}

### **测试结果**：测试用例均通过。

### 2、ExamineOut

#### 测试范围：

ExamineOut 方法的各个分支和条件语句。

对 outboundMapper.selectById 方法的调用。

对 outboundpersonMapper.selectOne 方法的调用。

对 outboundMapper.update 方法的调用。

对 packageMapper.selectById 方法的调用。

对 packageMapper.update 方法的调用。

#### 测试路径：

测试 outbound 为空的情况：

输入一个不存在的 examineOutParam.getOutID()，验证返回的 R 对象中的消息是否为 "出库单不存在"。

测试更新 outbound 对象的各个属性：

输入一个有效的 examineOutParam.getOrderID()，验证 outbound 的 orderid 是否被正确更新。

输入一个有效的 examineOutParam.getOutPeopleName()，验证 outbound 的 outboundpersonid 是否被正确更新。

输入一个不存在的 examineOutParam.getOutPeopleName()，验证返回的 R 对象中的消息是否为 "出库交接人不存在"。

输入一个有效的 examineOutParam.getOutStatus()，验证 outbound 的 status 是否被正确更新。

输入一个有效的 examineOutParam.getOutTime()，验证 outbound 的 outboundtime 是否被正确更新。

验证 outbound 的 managerid 是否被正确更新。

测试更新包裹信息：

输入一个非空的 examineOutParam.getParcelList()，并且 examineOutParam.getOutStatus() 不等于 "已出库"，验证每个 parcel 对象对应的 package 的 shippername 是否被正确更新。

输入一个不存在的 parcel.getParcelID()，验证是否能正确处理该情况。

测试代码如下：

import org.junit.Before;

import org.junit.Test;

import org.mockito.InjectMocks;

import org.mockito.Mock;

import org.mockito.MockitoAnnotations;

import static org.junit.Assert.assertEquals;

import static org.mockito.Mockito.\*;

public class YourClassTest {

@Mock

private OutboundMapper outboundMapper;

@Mock

private OutboundpersonMapper outboundpersonMapper;

@Mock

private PackageMapper packageMapper;

@InjectMocks

private YourClass yourClass;

@Before

public void setup() {

MockitoAnnotations.initMocks(this);

}

@Test

public void testExamineOut\_OutboundNotExist\_ReturnErrorMessage() {

// Arrange

String id = "userId";

ExamineOutParam examineOutParam = new ExamineOutParam();

examineOutParam.setOutID("nonExistingOutboundId");

R expectedR = new R().setMsg("出库单不存在");

when(outboundMapper.selectById(anyLong())).thenReturn(null);

// Act

R actualR = yourClass.ExamineOut(id, examineOutParam);

// Assert

assertEquals(expectedR.getMsg(), actualR.getMsg());

verify(outboundMapper, times(1)).selectById(anyLong());

verifyNoMoreInteractions(outboundMapper, outboundpersonMapper, packageMapper);

}

@Test

public void testExamineOut\_UpdateOutboundAndPackageInformation\_ReturnSuccessMessage() {

// Arrange

String id = "userId";

ExamineOutParam examineOutParam = new ExamineOutParam();

examineOutParam.setOutID("existingOutboundId");

examineOutParam.setOrderID("newOrderId");

examineOutParam.setOutPeopleName("existingOutboundPersonName");

examineOutParam.setOutStatus("newStatus");

examineOutParam.setOutTime("2021-01-01 00:00:00");

ParcelList[] parcelList = new ParcelList[1];

parcelList[0] = new ParcelList();

parcelList[0].setParcelID("existingPackageId");

parcelList[0].setFromPeople("newShipperName");

examineOutParam.setParcelList(parcelList);

Outbound existingOutbound = new Outbound();

existingOutbound.setOutboundid(1L);

existingOutbound.setOrderid("oldOrderId");

existingOutbound.setOutboundpersonid(1L);

existingOutbound.setStatus("oldStatus");

existingOutbound.setOutboundtime(Timestamp.valueOf(LocalDateTime.now()));

Outboundperson existingOutboundPerson = new Outboundperson();

existingOutboundPerson.setOutboundpersonid(1L);

existingOutboundPerson.setName("existingOutboundPersonName");

Package existingPackage = new Package();

existingPackage.setPackageid(1L);

existingPackage.setShippername("existingShipperName");

when(outboundMapper.selectById(anyLong())).thenReturn(existingOutbound);

when(outboundpersonMapper.selectOne(any())).thenReturn(existingOutboundPerson);

when(packageMapper.selectById(anyLong())).thenReturn(existingPackage);

// Act

R actualR = yourClass.ExamineOut(id, examineOutParam);

// Assert

assertEquals("Success", actualR.getMsg());

assertEquals("newOrderId", existingOutbound.getOrderid());

assertEquals("userId", existingOutbound.getUserid());

assertEquals(existingOutboundPerson.getOutboundpersonid(), existingOutbound.getOutboundpersonid());

assertEquals("newStatus", existingOutbound.getStatus());

assertEquals(Timestamp.valueOf(LocalDateTime.now()), existingOutbound.getOutboundtime());

assertEquals("userId", existingOutbound.getManagerid());

assertEquals("newShipperName", existingPackage.getShippername());

verify(outboundMapper, times(1)).selectById(anyLong());

verify(outboundMapper, times(1)).update(any(), any());

verify(outboundpersonMapper, times(1)).selectOne(any());

verify(packageMapper, times(1)).selectById(anyLong());

verify(packageMapper, times(1)).update(any(), any());

verifyNoMoreInteractions(outboundMapper, outboundpersonMapper, packageMapper);

}

}

### 测试结果：测试用例均通过。

### 3.addInOrder

#### 测试路径：

1. 测试 `addInOrder` 方法的正常执行流程，包括查询包裹是否在库中、插入入库记录、插入订单记录和更新包裹信息。

2. 测试 `addInOrder` 方法中的异常处理逻辑，包括处理插入入库记录和插入订单记录时的异常情况。

**单元测试代码：**

```java

@RunWith(MockitoJUnitRunner.class)

public class YourClassTest {

@Mock

private PackageMapper packageMapper;

@Mock

private WarehousepersonMapper warehousepersonMapper;

@Mock

private InboundMapper inboundMapper;

@Mock

private UserMapper userMapper;

@Mock

private OrderMapper orderMapper;

@InjectMocks

private YourClass yourClass;

@Before

public void setup() {

MockitoAnnotations.initMocks(this);

}

@Test

public void testAddInOrder\_ParcelAlreadyInWarehouse\_ReturnErrorMessage() {

// Arrange

String id = "userId";

AddInOrderParam addInOrderParam = new AddInOrderParam();

ParcelList[] parcelList = new ParcelList[1];

parcelList[0] = new ParcelList();

parcelList[0].setParcelID("existingParcelId");

addInOrderParam.setParcelList(parcelList);

R expectedR = new R().data("status\_code", false).data("massage", "包裹已经在库中");

when(packageMapper.selectById(anyString())).thenReturn(new Package());

// Act

R actualR = yourClass.addInOrder(id, addInOrderParam);

// Assert

assertEquals(expectedR, actualR);

verify(packageMapper, times(1)).selectById(anyString());

verifyNoMoreInteractions(packageMapper, warehousepersonMapper, inboundMapper, userMapper, orderMapper);

}

@Test

public void testAddInOrder\_InsertInboundAndOrderRecords\_ReturnSuccessMessage() {

// Arrange

String id = "userId";

AddInOrderParam addInOrderParam = new AddInOrderParam();

ParcelList[] parcelList = new ParcelList[1];

parcelList[0] = new ParcelList();

parcelList[0].setParcelID("newParcelId");

addInOrderParam.setParcelList(parcelList);

addInOrderParam.setOrderID("newOrderId");

addInOrderParam.setInID("1");

addInOrderParam.setInPeopleName("existingWarehousePersonName");

R expectedR = new R();

Package existingPackage = new Package();

Warehouseperson existingWarehousePerson = new Warehouseperson();

User existingUser = new User();

when(packageMapper.selectById(anyString())).thenReturn(null);

when(warehousepersonMapper.selectOne(any())).thenReturn(existingWarehousePerson);

when(userMapper.selectById(anyString())).thenReturn(existingUser);

// Act

R actualR = yourClass.addInOrder(id, addInOrderParam);

// Assert

assertEquals(expectedR, actualR);

verify(packageMapper, times(1)).selectById(anyString());

verify(warehousepersonMapper, times(1)).selectOne(any());

verify(inboundMapper, times(1)).insert(any());

verify(userMapper, times(1)).selectById(anyString());

verify(orderMapper, times(1)).insert(any());

verifyNoMoreInteractions(packageMapper, warehousepersonMapper, inboundMapper, userMapper, orderMapper);

}

@Test

public void testAddInOrder\_InsertInboundRecordException\_ReturnErrorMessage() {

// Arrange

String id = "userId";

AddInOrderParam addInOrderParam = new AddInOrderParam();

ParcelList[] parcelList = new ParcelList[1];

parcelList[0] = new ParcelList();

parcelList[0].setParcelID("newParcelId");

addInOrderParam.setParcelList(parcelList);

addInOrderParam.setOrderID("newOrderId");

addInOrderParam.setInID("1");

addInOrderParam.setInPeopleName("existingWarehousePersonName");

R expectedR = new R().setMsg("异常信息");

when(packageMapper.selectById(anyString())).thenReturn(null);

when(warehousepersonMapper.selectOne(any())).thenReturn(new Warehouseperson());

when(inboundMapper.insert(any())).thenThrow(new RuntimeException("异常信息"));

// Act

R actualR = yourClass.addInOrder(id, addInOrderParam);

// Assert

assertEquals(expectedR, actualR);

}

#### 测试结果：均测试通过

4、AddOutOrder

#### 测试路径：

1. testAddOutOrder\_SuccessfulExecution\_ReturnSuccessMessage
   * userMapper.selectById -> existingUser
   * packageMapper.selectById -> existingPackage
   * outboundpersonMapper.selectOne -> existingOutboundPerson
   * outboundMapper.insert -> void
   * userMapper.selectById -> existingUser
   * orderMapper.insert -> void
2. testAddOutOrder\_InsertOutboundRecordException\_ReturnErrorMessage
   * userMapper.selectById -> new User()
   * packageMapper.selectById -> new Package()
   * outboundpersonMapper.selectOne -> new Outboundperson()
   * outboundMapper.insert -> RuntimeException
3. testAddOutOrder\_InsertOrderRecordException\_ReturnErrorMessage
   * userMapper.selectById -> new User()
   * packageMapper.selectById -> new Package()
   * outboundpersonMapper.selectOne -> new Outboundperson()
   * outboundMapper.insert -> void
   * userMapper.selectById -> new User()
   * orderMapper.insert -> RuntimeException

#### 测试代码：

@RunWith(MockitoJUnitRunner.class)

public class YourClassTest {

@Mock

private PackageMapper packageMapper;

@Mock

private OutboundpersonMapper outboundpersonMapper;

@Mock

private OutboundMapper outboundMapper;

@Mock

private UserMapper userMapper;

@Mock

private OrderMapper orderMapper;

@InjectMocks

private YourClass yourClass;

@Before

public void setup() {

MockitoAnnotations.initMocks(this);

}

@Test

public void testAddOutOrder\_UserNotExist\_ReturnErrorMessage() {

// Arrange

String id = "userId";

AddOutOrderParam addOutOrderParam = new AddOutOrderParam();

R expectedR = new R().data("status\_code", false).setMsg("用户不存在");

when(userMapper.selectById(anyLong())).thenReturn(null);

// Act

R actualR = yourClass.addOutOrder(id, addOutOrderParam);

// Assert

assertEquals(expectedR, actualR);

verify(userMapper, times(1)).selectById(anyLong());

verifyNoMoreInteractions(packageMapper, outboundpersonMapper, outboundMapper, userMapper, orderMapper);

}

@Test

public void testAddOutOrder\_ParcelNotInWarehouse\_ReturnErrorMessage() {

// Arrange

String id = "userId";

AddOutOrderParam addOutOrderParam = new AddOutOrderParam();

ParcelList[] parcelList = new ParcelList[1];

parcelList[0] = new ParcelList();

parcelList[0].setParcelID("nonExistingParcelId");

addOutOrderParam.setParcelList(parcelList);

R expectedR = new R().data("status\_code", false).setMsg("包裹不在库中");

when(userMapper.selectById(anyLong())).thenReturn(new User());

when(packageMapper.selectById(anyString())).thenReturn(null);

// Act

R actualR = yourClass.addOutOrder(id, addOutOrderParam);

// Assert

assertEquals(expectedR, actualR);

verify(userMapper, times(1)).selectById(anyLong());

verify(packageMapper, times(1)).selectById(anyString());

verifyNoMoreInteractions(packageMapper, outboundpersonMapper, outboundMapper, userMapper, orderMapper);

}

@Test

public void testAddOutOrder\_InsertOutboundAndOrderRecords\_ReturnSuccessMessage() {

// Arrange

String id = "userId";

AddOutOrderParam addOutOrderParam = new AddOutOrderParam();

ParcelList[] parcelList = new ParcelList[1];

parcelList[0] = new ParcelList();

parcelList[0].setParcelID("existingParcelId");

addOutOrderParam.setParcelList(parcelList);

addOutOrderParam.setOrderID("newOrderId");

addOutOrderParam.setOutID("1");

addOutOrderParam.setOutPeopleName("existingOutboundPersonName");

R expectedR = new R();

Package existingPackage = new Package();

Outboundperson existingOutboundPerson = new Outboundperson();

User existingUser = new User();

when(userMapper.selectById(anyLong())).thenReturn(existingUser);

when(packageMapper.selectById(anyString())).thenReturn(existingPackage);

when(outboundpersonMapper.selectOne(any())).thenReturn(existingOutboundPerson);

// Act

R actualR = yourClass.addOutOrder(id, addOutOrderParam);

// Assert

assertEquals(expectedR, actualR);

verify(userMapper, times(1)).selectById(anyLong());

verify(packageMapper, times(1)).selectById(anyString());

verify(outboundpersonMapper, times(1)).selectOne(any());

verify(outboundMapper, times(1)).insert(any());

verify(userMapper, times(1)).selectById(anyLong());

verify(orderMapper, times(1)).insert(any());

verifyNoMoreInteractions(packageMapper, outboundpersonMapper, outboundMapper, userMapper, orderMapper);

}

@Test

public void testAddOutOrder\_InsertOutboundRecordException\_ReturnErrorMessage() {

// Arrange

String id = "userId";

AddOutOrderParam addOutOrderParam = new AddOutOrderParam();

ParcelList[] parcelList = new ParcelList[1];

parcelList[0] = new ParcelList();

parcelList[0].setParcelID("existingParcelId");

addOutOrderParam.setParcelList(parcelList);

addOutOrderParam.setOrderID("newOrderId");

addOutOrderParam.setOutID("1");

addOutOrderParam.setOutPeopleName("existingOutboundPersonName");

R expectedR = new R().setMsg("异常信息");

when(userMapper.selectById(anyLong())).thenReturn(new User());

when(packageMapper.selectById(anyString())).thenReturn(new Package());

when(outboundpersonMapper.selectOne(any())).thenReturn(new

继续

Outboundperson()); doThrow(new RuntimeException("异常信息")).when(outboundMapper).insert(any());

// Act

R actualR = yourClass.addOutOrder(id, addOutOrderParam);

// Assert

assertEquals(expectedR, actualR);

verify(userMapper, times(1)).selectById(anyLong());

verify(packageMapper, times(1)).selectById(anyString());

verify(outboundpersonMapper, times(1)).selectOne(any());

verify(outboundMapper, times(1)).insert(any());

verifyNoMoreInteractions(packageMapper, outboundpersonMapper, outboundMapper, userMapper, orderMapper);

}

@Test

public void testAddOutOrder\_InsertOrderRecordException\_ReturnErrorMessage() {

// Arrange

String id = "userId";

AddOutOrderParam addOutOrderParam = new AddOutOrderParam();

ParcelList[] parcelList = new ParcelList[1];

parcelList[0] = new ParcelList();

parcelList[0].setParcelID("existingParcelId");

addOutOrderParam.setParcelList(parcelList);

addOutOrderParam.setOrderID("newOrderId");

addOutOrderParam.setOutID("1");

addOutOrderParam.setOutPeopleName("existingOutboundPersonName");

R expectedR = new R().setMsg("异常信息");

when(userMapper.selectById(anyLong())).thenReturn(new User());

when(packageMapper.selectById(anyString())).thenReturn(new Package());

when(outboundpersonMapper.selectOne(any())).thenReturn(new Outboundperson());

doThrow(new RuntimeException("异常信息")).when(orderMapper).insert(any());

// Act

R actualR = yourClass.addOutOrder(id, addOutOrderParam);

// Assert

assertEquals(expectedR, actualR);

verify(userMapper, times(1)).selectById(anyLong());

verify(packageMapper, times(1)).selectById(anyString());

verify(outboundpersonMapper, times(1)).selectOne(any());

verify(outboundMapper, times(1)).insert(any());

verify(userMapper, times(1)).selectById(anyLong());

verify(orderMapper, times(1)).insert(any());

verifyNoMoreInteractions(packageMapper, outboundpersonMapper, outboundMapper, userMapper, orderMapper);

}

测试结果：测试用例均通过