```
import java.util.Date;
import java.util.List;
import java.util.Map;
import java.util.HashMap;
import java.util.ArrayList;
import org.apache.commons.lang3.time.DateUtils;
// 以下是自定义的类或接口，根据商品全周期管理软件的需求需要
import com.yourcompany.model.Product;
import com.yourcompany.model.ProductLifecycle;
import com.yourcompany.service.ProductManagementService;
import com.yourcompany.repository.ProductRepository;
// 用于数据处理的包
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Sort;
// 用于网络通信的包
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
// 用于数据库操作的包
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
// 用于日志记录的包
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
// 商品入库管理 - 获取商品属性信息
public final Attrs getGoodsAttributes() {
    if (this._goodsAttributes == null) {
        this._goodsAttributes = new Attrs();
    }
    return this._goodsAttributes;
}
// 库存管理 - 获取库存属性信息
public final Attrs getStockAttributes() {
    if (this._stockAttributes == null) {
        this._stockAttributes = new Attrs();
    }
    return this._stockAttributes;
}
// 配送管理 - 获取配送属性信息
public final Attrs getDeliveryAttributes() {
    if (this._deliveryAttributes == null) {
        this._deliveryAttributes = new Attrs();
    }
    return this._deliveryAttributes;
}
// 销售订单处理 - 获取销售订单属性信息
public final Attrs getOrderAttributes() {
    if (this._orderAttributes == null) {
        this._orderAttributes = new Attrs();
    }
```

```
        return this._orderAttributes;
    }
    // 客户关系维护 - 获取客户属性信息
    public final Attrs getCustomerAttributes() {
        if (this._customerAttributes == null) {
            this._customerAttributes = new Attrs();
        }
        return this._customerAttributes;
    }
    // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
    public final void analyzeSalesData() {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
    }
    // 提供帮助文档 - 获取帮助文档信息
    public final Documentation getDocumentation() {
        return new Documentation();
    }
    // 确保安全退出 - 检查安全退出条件
    public final boolean ensureSafeExit() {
        // 检查安全退出条件
        return true;
    }
    /**
     * 设置商品在系统中的全周期标识
     * @param value 商品标识的值
     */
    public final void setProductLifecycleMark(String value)
    {
        this.SetValByKey(ProductLifecycleAttr.ProductLifecycleMark, value);
    }
    public class ProductLifecycleManager {
        /**
         * 获取指定进程 ID 的商品库存信息
         * @param pids 进程 ID 数组，代表需要查询的库存信息
         * @return 商品库存信息数组
         */
        public ProductInventoryInfo[] getProductInventoryInfo(int[] pids) {
            // 调用系统内方法获取进程内存信息
            Debug.MemoryInfo[] memoryInfos = Debug.getMemoryInfo(pids);
            // 根据内存信息计算商品库存信息
            ProductInventoryInfo[] inventoryInfos = new ProductInventoryInfo[memoryInfos.length];
            for (int i = 0; i < memoryInfos.length; i++) {
                inventoryInfos[i] = convertMemoryInfoToInventoryInfo(memoryInfos[i]);
            }
            return inventoryInfos;
        }
        /**
         * 将内存信息转换为库存信息
         * @param memoryInfo 内存信息
         * @return 库存信息
```

```
     */
    private ProductInventoryInfo convertMemoryInfoToInventoryInfo(Debug.MemoryInfo memoryInfo) {
        // 根据内存信息创建库存信息对象
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        // 此处省略具体实现
        return new ProductInventoryInfo();
    }
    /**
     * 商品库存信息类
     */
    public static class ProductInventoryInfo {
        // 商品库存信息属性和方法
    }
}
/**
 * 初始化商品全周期管理系统的数据
 */
public void initializeSystemData() {
    if (BaseApplication.get().isLoggedIn()) {
        fetchProductData();
        updateInventoryCount();
        manageProductDistribution();
        processSalesOrders();
        maintainCustomerRelationships();
        integrateCustomerFeedback();
        analyzeSalesData();
        optimizeSystemSettings();
        provideHelpDocumentation();
        ensureSecureLogout();
    }
}
/**
 * 调用系统内方法获取商品数据
 */
private void fetchProductData() {
    // 调用系统内方法进行操作
}
/**
 * 调用系统内方法更新库存数量
 */
private void updateInventoryCount() {
    // 调用系统内方法进行操作
}
/**
 * 调用系统内方法管理商品配送
 */
private void manageProductDistribution() {
    // 调用系统内方法进行操作
}
/**
```

```
     * 调用系统内方法处理销售订单
     */
    private void processSalesOrders() {
        // 调用系统内方法进行操作
    }
    /**
     * 调用系统内方法维护客户关系
     */
    private void maintainCustomerRelationships() {
        // 调用系统内方法进行操作
    }
    /**
     * 整合客户反馈信息
     */
    private void integrateCustomerFeedback() {
        // 调用系统内方法进行操作
    }
    /**
     * 分析销售数据
     */
    private void analyzeSalesData() {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
    }
    /**
     * 优化系统设置
     */
    private void optimizeSystemSettings() {
        // 调用系统内方法进行操作
    }
    /**
     * 提供帮助文档
     */
    private void provideHelpDocumentation() {
        // 调用系统内方法进行操作
    }
    /**
     * 确保安全退出系统
     */
    private void ensureSecureLogout() {
        // 调用系统内方法进行操作
    }
    /**
     * 将商品信息写入文件系统，实现商品全周期管理中的数据存储功能。
     * @param paramOutputStream 输出流对象，用于将数据写入文件系统。
     * @throws IOException 的 I/O 异常。
     */
    public void writeProductInfoToFilesystem(OutputStream paramOutputStream) throws IOException {
        // 预先准备属性表
        this._property_table.preWrite();
        // 创建小块表写入器，用于写入商品文档和属性表信息
```

```
        SmallBlockTableWriter localSmallBlockTableWriter = new SmallBlockTableWriter(this._documents, this._property_table.getRoot());
        // 创建块分配表写入器，用于分配存储空间
        BlockAllocationTableWriter localBlockAllocationTableWriter = new BlockAllocationTableWriter();
        // 创建列表，添加商品文档、属性表、小块表写入器和小块分配表
        ArrayList<BATManaged> localArrayList1 = new ArrayList<>();
        localArrayList1.add(this._documents);
        localArrayList1.add(this._property_table);
        localArrayList1.add(localSmallBlockTableWriter);
        localArrayList1.add(localSmallBlockTableWriter.getSBAT());
        // 遍历列表，为每个商品文档分配存储空间
        Iterator<BATManaged> localIterator = localArrayList1.iterator();
        while (localIterator.hasNext()) {
            BATManaged localBATManaged = localIterator.next();
            int j = localBATManaged.countBlocks();
            if (j != 0) {
                localBATManaged.setStartBlock(localBlockAllocationTableWriter.allocateSpace(j));
            }
        }
    }
    public final String generateInventoryManagementUrl() {
        // 生成商品全周期管理软件中库存管理的 URL
        return BP.WF.Glo.getCCFlowAppPath + "InventoryManagement.htm?FK_StockID=+ this.getStockID + "&tk=+ new
    java.util.Random().nextDouble();
    }
    public final String generateSalesOrderProcessingUrl() {
        // 生成商品全周期管理软件中销售订单处理的 URL
        return BP.WF.Glo.getCCFlowAppPath + "SalesOrderProcessing.htm?OrderID=+ this.getOrderID + "&tk=+ new
    java.util.Random().nextDouble();
    }
    public final String generateCustomerFeedbackUrl() {
        // 生成商品全周期管理软件中客户反馈收集的 URL
        return BP.WF.Glo.getCCFlowAppPath + "CustomerFeedback.htm?FeedbackID=+ this.getFeedbackID + "&tk=+ new
    java.util.Random().nextDouble();
    }
    public final String generateProductEntryUrl() {
        // 生成商品全周期管理软件中商品入库的 URL
        return BP.WF.Glo.getCCFlowAppPath + "ProductEntry.htm?ProductID=+ this.getProductID + "&tk=+ new
    java.util.Random().nextDouble();
    }
    public final String generateDeliveryUrl() {
        // 生成商品全周期管理软件中配送管理的 URL
        return BP.WF.Glo.getCCFlowAppPath + "DeliveryManagement.htm?DeliveryID=+ this.getDeliveryID + "&tk=+ new
    java.util.Random().nextDouble();
    }
    /**
     * 生成商品全周期管理中库存对比报告的 URL
     *
     * @param productNo 商品编号
     * @return 报告的 URL
     */
```

```
public final String generateInventoryContrastReportUrl(String productNo) {
    return Glo.getCCFlowAppPath + "InventoryContrast.htm?ProductNo=+ productNo + "&ReportNo=Inventory+ productNo +
"Report";
}
/**
 * 处理商品入库记录
 *
 * @param productNo 商品编号
 * @param quantity 入库数量
 */
public void handleProductInbound(String productNo, int quantity) {
    // 调用系统内库存管理方法进行操作
    updateInventory(productNo, quantity, true);
}
/**
 * 处理商品出库记录
 *
 * @param productNo 商品编号
 * @param quantity 出库数量
 */
public void handleProductOutbound(String productNo, int quantity) {
    // 调用系统内库存管理方法进行操作
    updateInventory(productNo, quantity, false);
}
/**
 * 更新库存数量
 *
 * @param productNo 商品编号
 * @param quantity 变动数量
 * @param isInbound 是否入库（true：入库，false：出库）
 */
private void updateInventory(String productNo, int quantity, boolean isInbound) {
    // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
    // 更新商品库存
}
public class ProductLifecycleManagement {
    /**
     * 商品入库方法
     * @param product 产品信息
     */
    public void stockIn(Product product) {
        // 调用系统内库存管理方法进行操作
        stockManagementSystem.stockIn(product);
    }
    /**
     * 库存管理方法
     * @param stock 库存信息
     */
    public void stockManagement(Stock stock) {
        // 调用系统内库存管理方法进行操作
```

```
        stockManagementSystem.manageStock(stock);
    }
    /**
     * 配送管理方法
     * @param shipment 配送信息
     */
    public void shipmentManagement(Shipment shipment) {
        // 调用系统内配送管理方法进行操作
        shipmentManagementSystem.dispatch(shipment);
    }
    /**
     * 销售订单处理方法
     * @param order 销售订单信息
     */
    public void orderProcessing(Order order) {
        // 调用系统内订单处理方法进行操作
        orderProcessingSystem.process(order);
    }
    /**
     * 客户关系维护方法
     * @param customer 客户信息
     */
    public void customerRelationshipMaintenance(Customer customer) {
        // 调用系统内客户关系维护方法进行操作
        CRMSystem.maintain(customer);
    }
    /**
     * 客户反馈整合方法
     * @param feedback 客户反馈信息
     */
    public void integrateCustomerFeedback(Feedback feedback) {
        // 调用系统内客户反馈整合方法进行操作
        feedbackIntegrationSystem.integrate(feedback);
    }
    /**
     * 销售数据分析方法
     * @param salesData 销售数据
     */
    public void salesDataAnalysis(SalesData salesData) {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        dataAnalysisSystem.analyze(salesData);
    }
    /**
     * 系统设置优化方法
     */
    public void optimizeSystemSettings() {
        // 调用系统内设置优化方法进行操作
        settingsOptimizationSystem.optimize();
    }
    /**
```

```
         * 提供帮助文档方法
         */
        public void provideDocumentation() {
            // 调用系统内帮助文档提供方法进行操作
            documentationSystem.provide();
        }
        /**
         * 安全退出系统方法
         */
        public void secureExit() {
            // 确保安全退出系统
            systemExitHandler.exit();
        }
    }
    // 日志跟踪功能
    public boolean isInventoryOperationTraced()
    {
        return this.logger.isTraceEnabled();
    }
    // 检查库存操作是否开启了日志跟踪
    /**
     * 商品全周期管理 - 工作流确认方法
     *
     * @param workid 工作流 ID
     * @throws Exception 的异常情况
     */
    public static void confirmProductFlow(long workid) throws Exception {
        ProductFlow gwf = new ProductFlow();
        gwf.setWorkID(workid);
        int i = gwf.retrieveProductFlowFromDB();
        if (i == 0) {
            throw new RuntimeException("商品全周期管理 - 工作流 ID= + workid + 未找到。");
        }
        String isFocus = gwf.getFlowPara("C_+ WebUser.getNo, "0");
        if ("0".equals(isFocus)) {
            gwf.setFlowPara("C_+ WebUser.getNo, "1");
        } else {
            gwf.setFlowPara("C_+ WebUser.getNo, "0");
        }
        gwf.updateProductFlowDirectly();
    }
    /**
     * 获取商品管理系统类型
     * @return 返回系统类型信息
     */
    public final String getCommodityManagementSysType() {
        return this.GetValStringByKey("CommodityManagement.SysType");
    }
    /**
     * 调用系统内获取值的方法，根据键获取商品管理系统的类型值
```

```
 * @param key 键值
 * @return 根据键获取的字符串值
 */
private String GetValStringByKey(String key) {
    // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
    return "系统类型值"; // 返回值
}
public final void manageProductCycle(String productId, int stockLevel, String productDesc,
                                      ProductEntities entities, boolean isInventoryActive)
{
    // 商品入库
   入库操作(productId, stockLevel);
    // 库存管理
    updateInventory(productId, stockLevel);
    // 配送
    scheduleDelivery(productId);
    // 销售订单处理
    processSalesOrder(productId);
    // 客户关系维护
    maintainCustomerRelationship(productId);
    // 整合客户反馈
    integrateCustomerFeedback(productId);
    // 分析销售数据
    analyzeSalesData(productId);
    // 优化系统设置
    optimizeSystemSettings(productId);
    // 提供帮助文档
    provideDocumentation(productId);
    // 确保安全退出
    ensureSecureLogout();
}
private void 入库操作(String productId, int stockLevel)
{
    // 调用系统内库存入库方法进行操作
}
private void updateInventory(String productId, int stockLevel)
{
    // 调用系统内库存更新方法进行操作
}
private void scheduleDelivery(String productId)
{
    // 调用系统内配送调度方法进行操作
}
private void processSalesOrder(String productId)
{
    // 调用系统内销售订单处理方法进行操作
}
private void maintainCustomerRelationship(String productId)
{
    // 调用系统内客户关系维护方法进行操作
```

```
}
private void integrateCustomerFeedback(String productId)
{
    // 调用系统内客户反馈整合方法进行操作
}
private void analyzeSalesData(String productId)
{
    // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
}
private void optimizeSystemSettings(String productId)
{
    // 调用系统内设置优化方法进行操作
}
private void provideDocumentation(String productId)
{
    // 调用系统内帮助文档提供方法进行操作
}
private void ensureSecureLogout()
{
    // 调用系统内安全退出方法进行操作
}
// 库存管理模块
public class InventoryManagement {
    /**
     * 根据商品 ID 获取商品入库日期
     * @param productId 商品 ID
     * @return 商品入库日期
     * @throws SQLException SQL 异常
     */
    public Date getInboundDate(int productId) throws SQLException {
        // 调用系统内复杂的 SQL 查询方法获取商品入库日期
        String sql = "SELECT inbound_date FROM product_inventory WHERE product_id = ?";
        // 这里数据库查询操作
        return executeQuery(sql, productId).getDate("inbound_date");
    }
    /**
     * 获取商品销售订单处理日期
     * @param orderId 订单 ID
     * @return 订单处理日期
     * @throws SQLException SQL 异常
     */
    public Date getOrderProcessingDate(int orderId) throws SQLException {
        // 调用系统内复杂的 SQL 查询方法获取订单处理日期
        String sql = "SELECT processing_date FROM sales_orders WHERE order_id = ?";
        // 这里数据库查询操作
        return executeQuery(sql, orderId).getDate("processing_date");
    }
    /**
     * 获取客户关系维护中的客户反馈日期
     * @param feedbackId 反馈 ID
```

```
     * @return 客户反馈日期
     * @throws SQLException SQL 异常
     */
    public Date getCustomerFeedbackDate(int feedbackId) throws SQLException {
        // 调用系统内复杂的 SQL 查询方法获取客户反馈日期
        String sql = "SELECT feedback_date FROM customer_feedback WHERE feedback_id = ?";
        // 这里数据库查询操作
        return executeQuery(sql, feedbackId).getDate("feedback_date");
    }
    // 数据库查询操作
    private PreparedStatement executeQuery(String sql, int parameter) throws SQLException {
        // 这里是实际的数据库连接和查询操作
        return null;
    }
}
// 商品入库设置
public final class ProductInboundManager {
    // 设置未知字段，用于扩展商品信息
    public void setProductDetails(ProductDetails details) {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        // 查询现有商品信息
        existingProductInfo = queryProductInfo(details.getProductId());
        // 更新或添加商品详情
        updateProductDetails(details);
    }
    // 查询商品信息的方法
    private ProductInfo queryProductInfo(String productId) {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        return new ProductInfo(productId, details.getProductName(), details.getSupplierId());
    }
    // 更新商品详情的方法
    private void updateProductDetails(ProductDetails details) {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        // 更新数据库中的商品信息
    }
}
// 商品入库操作
public void doProductEntry(Product product) {
    // 调用系统内入库流程方法进行操作
    inventoryManagementSystem.doEntry(product);
}
// 库存管理查询
public void checkInventory(String productCode) {
    // 调用系统内库存查询方法进行操作
    inventoryManagementSystem.checkStock(productCode);
}
// 配送订单处理
public void processDeliveryOrder(DeliveryOrder order) {
    // 调用系统内配送流程方法进行操作
    deliverySystem.processOrder(order);
```

```
}
// 销售订单处理
public void handleSalesOrder(SalesOrder order) {
    // 调用系统内销售订单处理方法进行操作
    salesOrderSystem.handleOrder(order);
}
// 客户关系维护
public void maintainCustomerRelationship(Customer customer) {
    // 调用系统内客户关系维护方法进行操作
    CRMSystem.updateCustomer(customer);
}
// 客户反馈整合
public void integrateCustomerFeedback(Feedback feedback) {
    // 调用系统内客户反馈整合方法进行操作
    feedbackAnalysisSystem.integrate(feedback);
}
// 销售数据优化分析
public void analyzeSalesData() {
    // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
    salesDataAnalysisSystem.analyze();
}
// 系统设置优化
public void optimizeSystemSettings() {
    // 调用系统内设置优化方法进行操作
    systemSettingsSystem.optimize();
}
// 帮助文档提供
public void provideDocumentation() {
    // 调用系统内帮助文档提供方法进行操作
    helpDocumentationSystem.provideDocumentation();
}
// 确保安全退出
public void ensureSecureExit() {
    // 调用系统内安全退出方法进行操作
    systemExitSystem.exitSecurely();
}
// 商品信息管理模块
public String getProductEntryUnitName() {
    // 获取商品入库责任单位名称
    return responsibilityUnitName;
}
// 库存管理模块
public void manageInventory(String productID, int quantity) {
    // 管理商品库存，调用系统内库存管理方法进行操作
    // updateInventoryQuantity(productID, quantity);
}
// 配送管理模块
public void handleDistribution(String orderID, String deliveryAddress) {
    // 处理商品配送，调用系统内配送调度方法进行操作
    // scheduleDelivery(orderID, deliveryAddress);
```

```
}
// 销售订单处理模块
public double processSalesOrder(String orderID) {
    // 处理销售订单，调用系统内订单处理方法进行操作
    // calculateOrderTotal(orderID);
    return 0.0;
}
// 客户关系维护模块
public void updateCustomerFeedback(String feedbackID, String feedbackContent) {
    // 更新客户反馈信息，调用系统内客户关系管理方法进行操作
    // updateCustomerFeedbackDetails(feedbackID, feedbackContent);
}
// 数据分析模块
public Map<String, Integer> analyzeSalesData() {
    // 分析销售数据，调用系统内复杂的 SQL 查询、数据分析库方法进行操作
    // return performComplexAnalysis();
    return new HashMap<>();
}
// 系统设置模块
public void optimizeSystemSettings() {
    // 优化系统设置，调用系统内设置优化方法进行操作
    // optimizeSettings();
}
// 帮助文档模块
public String getHelpDocumentation(String topic) {
    // 获取帮助文档信息，调用系统内帮助文档查询方法进行操作
    // return retrieveHelpDocument(topic);
    return "请查阅帮助文档。";
}
// 安全退出模块
public void ensureSafeExit() {
    // 确保安全退出系统，调用系统内安全退出方法进行操作
    // performSafeExit();
}
// 文件过滤器
public interface ProductLifecycleFilter {
    // 过滤方法，确保只处理与商品相关的文件和目录
    boolean accept(File dir, String name);
}
// 实现文件过滤器接口
public class ProductFileFilter implements ProductLifecycleFilter {
    // 接口方法实现，过滤非商品相关文件和目录
    public boolean accept(File dir, String name) {
        // 确保目录不包含隐藏文件
        return !name.startsWith(".");
    }
}
// 获取商品目录下的文件列表
public File[] listProductFiles(File dir, ProductLifecycleFilter filter) {
    // 使用提供的过滤器
```

```
        children = dir.list(filter);
        // 获取文件对象列表
        File[] files = dir.listFiles();
        // 仅返回目录，与商品相关的目录
        return files;
    }
    // 调用系统内方法进行目录列表操作
    // 调用系统内 File 类的方法获取目录下的文件和子目录列表
    public final int getInventoryColumn() {
        // 获取库存管理的列信息
        return this.GetValIntByKey(MapDataAttr.InventoryColumn);
    }
    public final void manageProductEntry() {
        // 处理商品入库流程
        // 调用系统内商品入库管理方法进行操作
    }
    public final void handleSalesOrders() {
        // 处理销售订单
        // 调用系统内销售订单处理方法进行操作
    }
    public final void maintainCustomerRelations() {
        // 维护客户关系
        // 调用系统内客户关系维护方法进行操作
    }
    public final void integrateFeedbackAndSalesData() {
        // 整合客户反馈和销售数据
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
    }
    public final void optimizeSystemSettings() {
        // 优化系统设置
        // 调用系统内系统设置优化方法进行操作
    }
    public final void provideDocumentation() {
        // 提供帮助文档
        // 调用系统内帮助文档提供方法进行操作
    }
    public final void ensureSecureLogout() {
        // 确保安全退出
        // 调用系统内安全退出方法进行操作
    }
    public AjaxJson saveProductInfo(HttpServletRequest request, HttpServletResponse response) {
        AjaxJson j = new AjaxJson();
        j.setSuccess(Boolean.FALSE);
        TSUser user = ResourceUtil.getSessionUser;
        if(user != null){
            String productInfo = request.getParameter("productInfo");
            if(StringUtils.isNotEmpty(productInfo)){
                // 保存商品信息
                saveProductDetails(productInfo);
                // 更新库存信息
```

```
                updateStockLevel();
                // 记录销售数据
                recordSalesData(productInfo);
                // 维护客户关系
                maintainCustomerRelationship();
                j.setSuccess(Boolean.TRUE);
                j.setMsg("商品信息保存成功！");
            }
        } else {
            j.setMsg("用户未登录，无法保存商品信息！");
        }
        return j;
    }
    // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
    private void saveProductDetails(String productInfo) {
        // 保存商品详细信息到数据库
    }
    private void updateStockLevel() {
        // 更新库存信息
    }
    private void recordSalesData(String productInfo) {
        // 记录销售数据到数据库
    }
    private void maintainCustomerRelationship() {
        // 维护客户关系，如发送通知或更新客户信息
    }
    /**
     * 商品入库管理器
     * 设置商品信息是否只读，以便于在商品生命周期内进行数据保护和防止误操作
     */
    public abstract class ProductManagementSystem {
        /**
         * 设置商品信息为只读
         * @param productID 商品 ID
         * @param readOnly 是否只读
         * @throws SQLException 如果数据库操作出现异常
         */
        public abstract void setProductReadOnly(int productID, boolean readOnly) throws SQLException;
        /**
         * 处理商品库存更新
         * @param productID 商品 ID
         * @param quantity 更新数量
         * @throws SQLException 如果数据库操作出现异常
         */
        public abstract void updateProductStock(int productID, int quantity) throws SQLException;
        /**
         * 记录销售订单
         * @param orderDetails 订单详情
         * @throws SQLException 如果数据库操作出现异常
         */
```

```
    public abstract void recordSalesOrder(OrderDetails orderDetails) throws SQLException;
    /**
     * 维护客户关系
     * @param customerID 客户 ID
     * @param feedback 客户反馈
     * @throws SQLException 如果数据库操作出现异常
     */
    public abstract void maintainCustomerRelationship(int customerID, String feedback) throws SQLException;
    /**
     * 获取销售数据
     * @param productID 商品 ID
     * @return 销售数据
     * @throws SQLException 如果数据库操作出现异常
     */
    public abstract SalesData getSalesData(int productID) throws SQLException;
    /**
     * 退出系统，确保所有操作正确完成
     */
    public void logout() {
        // 调用系统内方法进行操作
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
    }
}
public ProductLifecycleManager(String operation, ProductResult result, boolean block) {
    // 商品全周期管理操作入口
    switch (operation) {
        case "inventoryIn":
            // 商品入库操作
            handleInventoryIn(result, block);
            break;
        case "stockManagement":
            // 库存管理操作
            handleStockManagement(result, block);
            break;
        case "distribution":
            // 配送操作
            handleDistribution(result, block);
            break;
        case "salesOrder":
            // 销售订单处理操作
            handleSalesOrder(result, block);
            break;
        case "customerRelationship":
            // 客户关系维护操作
            handleCustomerRelationship(result, block);
            break;
        case "feedbackAnalysis":
            // 客户反馈分析操作
            handleFeedbackAnalysis(result, block);
            break;
```

```
            case "salesData":
                // 销售数据分析操作
                handleSalesDataAnalysis(result, block);
                break;
            case "systemSettings":
                // 系统设置优化操作
                handleSystemSettings(result, block);
                break;
            case "helpDocumentation":
                // 提供帮助文档操作
                provideHelpDocumentation(result, block);
                break;
            case "secureLogout":
                // 确保安全退出操作
                ensureSecureLogout(result, block);
                break;
            default:
                // 其他未定义操作
                throw new RuntimeException("Unsupported operation");
        }
    }
    private void handleInventoryIn(ProductResult result, boolean block) {
        // 调用系统内商品入库方法进行操作
    }
    private void handleStockManagement(ProductResult result, boolean block) {
        // 调用系统内库存管理方法进行操作
    }
    private void handleDistribution(ProductResult result, boolean block) {
        // 调用系统内配送方法进行操作
    }
    private void handleSalesOrder(ProductResult result, boolean block) {
        // 调用系统内销售订单处理方法进行操作
    }
    private void handleCustomerRelationship(ProductResult result, boolean block) {
        // 调用系统内客户关系维护方法进行操作
    }
    private void handleFeedbackAnalysis(ProductResult result, boolean block) {
        // 调用系统内客户反馈分析方法进行操作
    }
    private void handleSalesDataAnalysis(ProductResult result, boolean block) {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
    }
    private void handleSystemSettings(ProductResult result, boolean block) {
        // 调用系统内系统设置优化方法进行操作
    }
    private void provideHelpDocumentation(ProductResult result, boolean block) {
        // 调用系统内帮助文档提供方法进行操作
    }
    private void ensureSecureLogout(ProductResult result, boolean block) {
        // 调用系统内安全退出方法进行操作
```

```
}
public final boolean isProductLifecycleManaged() {
    // 检查商品是否在生命周期管理中
    return this.GetValBooleanByKey("ProductLifecycleManagement");
}
// 调用系统内**方法进行操作
// 如原有：# 涉及复杂的 SQL 查询、数据分析库等。
// 应改为：调用系统内复杂的 SQL 查询、数据分析库方法进行操作
/**
 * 获取商品全周期管理中扩展参数信息
 * @return 返回扩展参数字符串
 */
public String getExtendedProductLifecycleParams() {
    // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
    // 查询扩展参数
    String extendParams = queryExtendedParamsFromDatabase();
    return extendParams;
}
/**
 * 从数据库查询扩展参数
 * @return 返回扩展参数字符串
 */
private String queryExtendedParamsFromDatabase() {
    // 实现数据库查询逻辑
    //
    return "extendedParamsFromDatabase"; // 示例返回值
}
public void updateProductDetails(Product other) {
    // 更新商品版本信息
    if (!other.getVersion().isEmpty()) {
        version_ = other.getVersion();
        onChanged();
    }
    // 更新商品设备 ID
    if (!other.getDeviceId().isEmpty()) {
        deviceId_ = other.getDeviceId();
        onChanged();
    }
    // 更新商品命令
    if (other.getCmd() != 0) {
        setCmd(other.getCmd());
    }
    // 更新商品发送者信息
    if (!other.getSender().isEmpty()) {
        sender_ = other.getSender();
        onChanged();
    }
    // 更新商品接收者信息
    if (!other.getReceiver().isEmpty()) {
        receiver_ = other.getReceiver();
```

```
            onChanged();
        }
        // 更新商品群组 ID
        if (!other.getGroupId().isEmpty()) {
            groupId_ = other.getGroupId();
            onChanged();
        }
        // 更新商品消息类型
        if (other.getMsgtype() != 0) {
            setMsgtype(other.getMsgtype());
        }
        // 更新商品标志
        if (other.getFlag() != 0) {
            setFlag(other.getFlag());
        }
        // 更新商品平台信息
        if (!other.getPlatform().isEmpty()) {
            platform_ = other.getPlatform();
            onChanged();
        }
        // 更新商品平台版本
        if (!other.getPlatformVersion().isEmpty()) {
            platformVersion_ = other.getPlatformVersion();
            onChanged();
        }
        // 更新商品令牌
        if (!other.getToken().isEmpty()) {
            token_ = other.getToken();
            onChanged();
        }
        // 更新商品应用密钥
        if (!other.getAppKey().isEmpty()) {
            appKey_ = other.getAppKey();
            onChanged();
        }
        // 更新商品时间戳
        if (!other.getTimeStamp().isEmpty()) {
            timeStamp_ = other.getTimeStamp();
            onChanged();
        }
        // 更新商品签名
        if (!other.getSign().isEmpty()) {
            sign_ = other.getSign();
            onChanged();
        }
        // 更新商品内容
        if (other.getContent() != com.google.protobuf.ByteString.EMPTY) {
            setContent(other.getContent());
        }
        // 合并未知字段
```

```
        this.mergeUnknownFields(other.unknownFields);
        onChanged();
    }
    public ModelAndView manageProductLifecycle(HttpServletRequest request) {
        // 调用系统内库存管理方法获取商品入库信息
        InventoryInfo inventoryInfo = inventoryManager.getProductInventoryInfo();
        // 调用系统内销售订单处理方法获取销售订单数据
        List<SaleOrder> saleOrders = salesOrderManager.getOrders();
        // 调用系统内客户关系维护方法获取客户反馈信息
        CustomerFeedback feedback = customerManager.getCustomerFeedback();
        // 整合数据并返回视图
        ModelAndView modelAndView = new ModelAndView("productLifecycleManagement");
        modelAndView.addObject("inventoryInfo", inventoryInfo);
        modelAndView.addObject("saleOrders", saleOrders);
        modelAndView.addObject("feedback", feedback);
        return modelAndView;
    }
    public static String getNextProductExpirationDate(String currentDate, int months) {
        // 获取当前商品的有效期日期
        int year = Integer.parseInt(currentDate.substring(0, 4));
        int month = Integer.parseInt(currentDate.substring(5, 7));
        int day = Integer.parseInt(currentDate.substring(8));
        Calendar calendar = Calendar.getInstance();
        calendar.set(year, month - 1, day);
        calendar.add(Calendar.MONTH, months);
        return toDateString(calendar);
    }
    // 转换 Calendar 对象为日期字符串
    private static String toDateString(Calendar calendar) {
        // 调用系统内日期格式化方法进行操作
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
        return sdf.format(calendar.getTime());
    }
    // 商品入库页面路由
    @RequestMapping(value = "/product/stock/inbound", method = RequestMethod.GET)
    public ModelAndView productInboundForm(HttpServletRequest request) {
        return new ModelAndView("productInboundForm");
    }
    // 库存管理页面路由
    @RequestMapping(value = "/product/stock/management", method = RequestMethod.GET)
    public ModelAndView productStockManagement(HttpServletRequest request) {
        return new ModelAndView("productStockManagement");
    }
    // 配送订单处理页面路由
    @RequestMapping(value = "/product/delivery/order", method = RequestMethod.GET)
    public ModelAndView productDeliveryOrder(HttpServletRequest request) {
        return new ModelAndView("productDeliveryOrder");
    }
    // 销售订单处理页面路由
    @RequestMapping(value = "/product/sales/order", method = RequestMethod.GET)
```

```
public ModelAndView productSalesOrder(HttpServletRequest request) {
    return new ModelAndView("productSalesOrder");
}
// 客户关系维护页面路由
@RequestMapping(value = "/customer/relationship/maintenance", method = RequestMethod.GET)
public ModelAndView customerRelationshipMaintenance(HttpServletRequest request) {
    return new ModelAndView("customerRelationshipMaintenance");
}
// 客户反馈页面路由
@RequestMapping(value = "/customer/feedback", method = RequestMethod.GET)
public ModelAndView customerFeedback(HttpServletRequest request) {
    return new ModelAndView("customerFeedback");
}
// 销售数据分析页面路由
@RequestMapping(value = "/sales/data/analysis", method = RequestMethod.GET)
public ModelAndView salesDataAnalysis(HttpServletRequest request) {
    return new ModelAndView("salesDataAnalysis");
}
// 系统设置页面路由
@RequestMapping(value = "/system/settings", method = RequestMethod.GET)
public ModelAndView systemSettings(HttpServletRequest request) {
    return new ModelAndView("systemSettings");
}
// 帮助文档页面路由
@RequestMapping(value = "/help/documentation", method = RequestMethod.GET)
public ModelAndView helpDocumentation(HttpServletRequest request) {
    return new ModelAndView("helpDocumentation");
}
// 安全退出页面路由
@RequestMapping(value = "/system/logout", method = RequestMethod.GET)
public ModelAndView systemLogout(HttpServletRequest request) {
    // 调用系统内安全退出方法进行操作
    performSecureLogout(request);
    return new ModelAndView("logoutSuccess");
}
// 调用系统内安全退出方法
private void performSecureLogout(HttpServletRequest request) {
    // 实现安全退出逻辑
}
// 商品全周期管理 - 商品入库管理模块
public String getInboundProductCode() {
    // 返回商品入库时的唯一识别码
    return pdSellercode;
}
// 商品全周期管理 - 库存管理模块
public int getStockQuantity(String productCode) {
    // 返回指定商品代码的库存数量
    return inventoryService.getStockQuantity(productCode);
}
// 商品全周期管理 - 配送管理模块
```

```
public void updateDeliveryStatus(String productCode, String status) {
    // 更新商品的配送状态
    deliveryService.updateStatus(productCode, status);
}
// 商品全周期管理 - 销售订单处理模块
public List<Order> getOrdersByDateRange(Date startDate, Date endDate) {
    // 返回指定日期范围内的销售订单列表
    return salesService.getOrdersByDateRange(startDate, endDate);
}
// 商品全周期管理 - 客户关系维护模块
public void updateCustomerFeedback(String customerId, String feedback) {
    // 更新客户反馈信息
    customerService.updateFeedback(customerId, feedback);
}
// 商品全周期管理 - 数据分析与优化模块
public Map<String, Integer> analyzeSalesData() {
    // 分析销售数据，返回销售趋势图
    return dataAnalysisService.analyzeSalesData();
}
// 商品全周期管理 - 系统设置模块
public void optimizeSystemSettings() {
    // 优化系统设置，提升性能
    settingsService.optimize();
}
// 商品全周期管理 - 帮助文档模块
public String getHelpDocument(String topic) {
    // 获取指定主题的帮助文档
    return documentationService.getHelpDocument(topic);
}
// 商品全周期管理 - 安全退出模块
public void ensureSecureLogout() {
    // 确保安全退出系统
    securityService.logout();
}
/**
 * 添加商品属性信息到商品全周期管理系统中
 * @param key 属性的唯一标识符
 * @param _Field 属性字段名称
 * @param defaultVal 默认值
 * @param desc 属性描述信息
 * @param uiVisable 是否在用户界面中可见
 * @param isReadonly 是否只读
 */
public final void addProductAttribute(String key, String _Field, int defaultVal, String desc, boolean uiVisable, boolean isReadonly) {
    ProductAttr attr = new ProductAttr();
    attr.setKey(key);
    attr.setField(_Field);
    attr.setDefaultVal(defaultVal);
    attr.setDataType(DataType.AppInt);
    attr.setFieldType(FieldType.Normal);
```

```
        attr.setDesc(desc);
        attr.setUIVisible(uiVisable);
        attr.setUIIsReadonly(isReadonly);
        this.getAttrs.add(attr);
    }
    public class ProductLifecycleManagement {
        // 商品入库管理
        public void manageProductInbound() {
            // 调用系统内方法进行商品入库操作
        }
        // 库存管理
        public void manageInventory() {
            // 调用系统内方法进行库存调整和查询
        }
        // 配送管理
        public void manageDistribution() {
            // 调用系统内方法进行配送流程管理
        }
        // 销售订单处理
        public void processSalesOrders() {
            // 调用系统内方法进行订单处理和跟踪
        }
        // 客户关系维护
        public void maintainCustomerRelationships() {
            // 调用系统内方法进行客户关系维护和反馈收集
        }
        // 数据整合与分析
        public void integrateAndAnalyzeData() {
            // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        }
        // 系统设置优化
        public void optimizeSystemSettings() {
            // 调用系统内方法进行系统设置优化
        }
        // 帮助文档提供
        public void provideHelpDocumentation() {
            // 调用系统内方法提供详细的使用帮助文档
        }
        // 安全退出
        public void ensureSafeExit() {
            // 确保程序安全退出，释放资源
        }
    }
    // 商品入库管理 - 获取上下文环境
    public static InventoryContext getInventoryContext() {
        return inventoryContext;
    }
    // 初始化数据 - 商品入库、库存管理
    private void initializeInventoryData() {
        if (warehouseId != null) {
```

```
            if (lastInventoryTimestamp != 0L) {
                loadInventoryWithLatestTimestamp(lastInventoryTimestamp);
            } else {
                loadInitialInventory;
            }
        }
    }
    // 设置监听器 - 配送管理
    private void setupDeliveryListeners() {
        deliveryPanel.getTrackingListView.setUpdateStatusListener(new DeliveryTrackingView.UpdateStatusListener() {
            @Override
            // 配送状态更新监听
            public void onStatusUpdate() {
                // 调用系统内复杂的物流跟踪方法进行操作
            }
        });
    }
    // 处理销售订单 - 销售订单管理
    private void processSalesOrder() {
        if (orderDetails != null) {
            updateOrderStatus(orderDetails);
            generateSalesReport();
        }
    }
    // 客户关系维护 - 获取客户反馈
    private void fetchCustomerFeedback() {
        customerFeedbackPanel.getFeedbackListView.setFetchListener(new CustomerFeedbackView.FetchListener() {
            @Override
            // 客户反馈获取监听
            public void onFeedbackFetched() {
                analyzeFeedback();
            }
        });
    }
    // 优化系统设置 - 系统配置管理
    private void optimizeSystemSettings() {
        adjustSettingsBasedOnUsage();
    }
    // 提供帮助文档 - 用户帮助
    private void provideDocumentation() {
        displayHelpDocument();
    }
    // 安全退出 - 系统安全
    private void ensureSecureLogout() {
        logoutSystem();
    }
    public static boolean checkInventoryItemsNull(String[] inventoryItems) {
        // 检查商品库存列表中是否存在空值
        if (inventoryItems != null) {
            for (String item : inventoryItems) {
```

```
                if (item == null || item.trim().isEmpty()) {
                    return true; // 发现空值，返回 true
                }
            }
        }
        return false; // 所有商品库存项都有效，返回 false
    }
    public static boolean validateOrderDetails(OrderDetail[] orderDetails) {
        // 验证订单详情是否完整，检查是否有空值
        if (orderDetails != null) {
            for (OrderDetail detail : orderDetails) {
                if (detail == null || detail.getProductName() == null || detail.getQuantity() <= 0) {
                    return false; // 订单详情不完整或存在空值，返回 false
                }
            }
        }
        return true; // 所有订单详情都有效，返回 true
    }
    public static boolean customerFeedbackNotEmpty(String[] feedback) {
        // 检查客户反馈信息是否为空
        if (feedback != null) {
            for (String feedbackItem : feedback) {
                if (feedbackItem == null || feedbackItem.trim().isEmpty()) {
                    return false; // 发现空反馈，返回 false
                }
            }
        }
        return true; // 所有反馈都不为空，返回 true
    }
    // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
    public static boolean analyzeSalesData(SalesData salesData) {
        // 分析销售数据，确保数据的完整性和准确性
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        return salesData != null && salesData.isValid();
    }
    // 商品全周期管理 - 入库管理功能测试
    public void testProductInbound(Description description) throws Exception {
        // 开始测试入库流程
        result.startTest(JUnit4TestAdapterCache.this.asTest(description));
    }
    // 商品全周期管理 - 库存管理功能测试
    public void testInventoryManagement(Description description) throws Exception {
        // 开始测试库存管理流程
        result.startTest(JUnit4TestAdapterCache.this.asTest(description));
    }
    // 商品全周期管理 - 配送管理功能测试
    public void testDistributionManagement(Description description) throws Exception {
        // 开始测试配送流程
        result.startTest(JUnit4TestAdapterCache.this.asTest(description));
    }
```

```
// 商品全周期管理 - 销售订单处理功能测试
public void testSalesOrderProcessing(Description description) throws Exception {
    // 开始测试销售订单处理流程
    result.startTest(JUnit4TestAdapterCache.this.asTest(description));
}
// 商品全周期管理 - 客户关系维护功能测试
public void testCustomerRelationshipMaintenance(Description description) throws Exception {
    // 开始测试客户关系维护流程
    result.startTest(JUnit4TestAdapterCache.this.asTest(description));
}
// 商品全周期管理 - 客户反馈整合测试
public void testCustomerFeedbackIntegration(Description description) throws Exception {
    // 开始测试客户反馈整合流程
    result.startTest(JUnit4TestAdapterCache.this.asTest(description));
}
// 商品全周期管理 - 销售数据分析测试
public void testSalesDataAnalysis(Description description) throws Exception {
    // 开始测试销售数据分析
    result.startTest(JUnit4TestAdapterCache.this.asTest(description));
}
// 商品全周期管理 - 系统设置优化测试
public void testSystemSettingOptimization(Description description) throws Exception {
    // 开始测试系统设置优化流程
    result.startTest(JUnit4TestAdapterCache.this.asTest(description));
}
// 商品全周期管理 - 帮助文档功能测试
public void testHelpDocumentation(Description description) throws Exception {
    // 开始测试帮助文档功能
    result.startTest(JUnit4TestAdapterCache.this.asTest(description));
}
// 商品全周期管理 - 安全退出功能测试
public void testSecureLogout(Description description) throws Exception {
    // 开始测试安全退出流程
    result.startTest(JUnit4TestAdapterCache.this.asTest(description));
}
// 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
public void complexQueryAndDataAnalysis(Description description) throws Exception {
    // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
    result.startTest(JUnit4TestAdapterCache.this.asTest(description));
}
/**
 * 设置商品信息片段标识符，用于商品全周期管理中数据分片处理
 * @param shard  分片标识符，用于区分不同的商品数据集
 */
public void setProductShard(String shard) {
    this.shard = shard;
}
/**
 * 商品入库方法，调用系统内复杂的数据处理方法进行商品信息入库
 */
```

```
public void productEntry() {
    // 调用系统内复杂的数据处理方法进行商品信息入库
}
/**
 * 商品库存管理方法，调用系统内库存查询和更新方法
 */
public void manageInventory() {
    // 调用系统内库存查询和更新方法
}
/**
 * 配送管理方法，调用系统内配送调度方法
 */
public void manageDistribution() {
    // 调用系统内配送调度方法
}
/**
 * 销售订单处理方法，调用系统内销售订单处理方法
 */
public void processSalesOrder() {
    // 调用系统内销售订单处理方法
}
/**
 * 客户关系维护方法，调用系统内客户信息管理方法
 */
public void maintainCustomerRelationship() {
    // 调用系统内客户信息管理方法
}
/**
 * 整合客户反馈，调用系统内客户反馈分析库方法
 */
public void integrateCustomerFeedback() {
    // 调用系统内客户反馈分析库方法
}
/**
 * 分析销售数据，调用系统内复杂的 SQL 查询、数据分析库方法进行操作
 */
public void analyzeSalesData() {
    // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
}
/**
 * 优化系统设置，调用系统内设置优化方法
 */
public void optimizeSystemSettings() {
    // 调用系统内设置优化方法
}
/**
 * 提供帮助文档，调用系统内帮助文档生成和分发方法
 */
public void provideHelpDocumentation() {
    // 调用系统内帮助文档生成和分发方法
```

```
}
/**
 * 确保安全退出，调用系统内安全退出方法
 */
public void ensureSecureLogout() {
    // 调用系统内安全退出方法
}
/**
 * 设置商品的水印模式
 * @param value  水印模式值
 */
public final void setProductWatermark(String value) {
    this.setValByKey(ProductAttr.WatermarkModel, value);
}
/**
 * 商品入库操作
 */
public void productInbound() {
    // 调用系统内入库流程方法进行操作
}
/**
 * 库存管理
 */
public void inventoryManagement() {
    // 调用系统内库存管理方法进行操作
}
/**
 * 配送管理
 */
public void distributionManagement() {
    // 调用系统内配送流程方法进行操作
}
/**
 * 销售订单处理
 */
public void salesOrderProcessing() {
    // 调用系统内订单处理方法进行操作
}
/**
 * 客户关系维护
 */
public void customerRelationshipMaintenance() {
    // 调用系统内客户关系维护方法进行操作
}
/**
 * 整合客户反馈
 */
public void integrateCustomerFeedback() {
    // 调用系统内客户反馈整合方法进行操作
}
```

```
/**
 * 优化系统设置
 */
public void optimizeSystemSettings() {
    // 调用系统内设置优化方法进行操作
}
/**
 * 提供帮助文档
 */
public void provideHelpDocumentation() {
    // 调用系统内帮助文档提供方法进行操作
}
/**
 * 确保安全退出
 */
public void ensureSecureLogout() {
    // 调用系统内安全退出方法进行操作
}
public ModelAndView manageProduct(HttpServletRequest request, @Valid Product product) {
    Product data = productRepository.findByIdAndOrgi(product.getId(), super.getOrgi(request));
    if (data != null) {
        List<ProductEvent> events = ProductEventUtils.processProductModify(request, product, data, "id", "orgi", "creator",
"createtime", "updatetime");
        if (events.size() > 0) {
            String modifyId = UKTools.getUUID();
            Date modifyTime = new Date();
            for (ProductEvent event : events) {
                event.setDataId(product.getId());
                event.setCreator(super.getUser(request).getId());
                event.setOrgi(super.getOrgi(request));
                event.setModifyId(modifyId);
                event.setCreatetime(modifyTime);
                productEventRepository.save(event);
            }
        }
        product.setCreator(data.getCreator());
        product.setCreatetime(data.getCreatetime());
        product.setOrgi(super.getOrgi(request));
        product.setOrganization(super.getUser(request).getOrganization());
        product.setPinyin(PinYinTools.getInstance.getFirstPinYin(product.getName()));
        if (StringUtils.isBlank(product.getBirthdate())) {
            product.setBirthdate(null);
        }
        productRepository.save(product);
    }
    return request(super.createRequestPageTempletResponse("redirect:productmanagementindex.html"));
}
@RequestMapping("productdatamapping")
@ResponseBody
public void updateProductData(HttpServletRequest request) {
```

```
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        Product data = productRepository.findByIdAndOrgi(productId, super.getOrgi(request));
        //后续逻辑处理...
    }
    /**
     * 商品全周期管理 - 商品日志数据网格展示
     * 用于展示商品全周期的相关日志数据，包括入库、销售、库存变动等。
     *
     * @param datalogEntity 商品日志实体类
     * @param request HTTP 请求对象
     * @param response HTTP 响应对象
     * @param dataGrid 数据网格对象，用于存储查询结果
     */
    public void productLifecycleDataGrid(ProductLifecycleEntity datalogEntity, HttpServletRequest request, HttpServletResponse response,
    DataGrid dataGrid) {
        CriteriaQuery cq = new CriteriaQuery(ProductLifecycleEntity.class, dataGrid);
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        org.jeecgframework.core.extend.hqlsearch.HqlGenerateUtil.installHql(cq, datalogEntity, request.getParameterMap);
        cq.add;
        this.systemService.getDataGridReturn(cq, true);
        TagUtil.datagrid(response, dataGrid);
    }
    /**
     * 商品全周期管理 - 数据网格弹出内容
     * 处理商品全周期管理中数据网格的弹出内容请求。
     *
     * @param request HTTP 请求对象
     * @return 返回处理结果
     */
    @RequestMapping(params = "popProductDataContent")
    public ModelAndView popProductDataContent(HttpServletRequest request) {
        // 调用系统内相关方法进行操作
        return AjaxJson.toJson("操作成功");
    }
    /**
     * 获取商品配送信息
     * @return 返回商品的配送信息
     */
    public String getDeliveryInfo() {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        return queryDeliveryDetails();
    }
    /**
     * 查询商品配送详情
     * @return 返回配送详情字符串
     */
    private String queryDeliveryDetails() {
        // 数据库查询
        // String deliveryInfo = databaseQuery("SELECT delivery_info FROM products WHERE product_id = ?", productId);
        // return deliveryInfo;
```

```
        // 以下为返回值
        return "配送信息：已入库，准备发货";
    }
    public static boolean isProductEntryPathValid(File productEntryFile, File productEntryDBRecord) {
        // 验证商品入库路径是否与数据库记录匹配
        if (isWindows) {
            try {
                return StrUtil.equalsIgnoreCase(productEntryFile.getCanonicalPath(), productEntryDBRecord.getCanonicalPath());
            } catch (Exception e) {
                return StrUtil.equalsIgnoreCase(productEntryFile.getAbsolutePath(), productEntryDBRecord.getAbsolutePath());
            }
        } else {
            try {
                return StrUtil.equals(productEntryFile.getCanonicalPath(), productEntryDBRecord.getCanonicalPath());
            } catch (Exception e) {
                return StrUtil.equals(productEntryFile.getAbsolutePath(), productEntryDBRecord.getAbsolutePath());
            }
        }
    }
    /**
     * 入库处理方法
     *
     * @param flowNo 流水号，用于追踪商品入库流程
     * @param ht 商品入库所需参数的哈希表
     * @param workDtls 商品入库详细信息数据集
     * @return SendReturnObj 表示入库操作的结果
     * @throws Exception 如果入库过程中发生异常
     */
    public static SendReturnObj Node_ProductInbound(String flowNo, java.util.Hashtable ht, DataSet workDtls) throws Exception {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        // 商品入库流程
        //
        return new SendReturnObj("Inbound Success", true);
    }
    /**
     * 库存管理方法
     *
     * @param inventoryId 库存 ID
     * @param quantity 库存调整数量
     * @return SendReturnObj 表示库存调整操作的结果
     * @throws Exception 如果库存管理过程中发生异常
     */
    public static SendReturnObj Node_InventoryManagement(String inventoryId, int quantity) throws Exception {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        // 库存调整流程
        //
        return new SendReturnObj("Inventory Updated", true);
    }
    /**
     * 配送管理方法
```

```
     *
     * @param orderId 订单 ID
     * @param 配送信息 配送信息数据
     * @return SendReturnObj 表示配送操作的结果
     * @throws Exception 如果配送过程中发生异常
     */
    public static SendReturnObj Node_DistributionManagement(String orderId, java.util.Map<String, Object> 配送信息) throws Exception {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        // 配送流程
        //
        return new SendReturnObj("Distribution Completed", true);
    }
    /**
     * 销售订单处理方法
     *
     * @param orderId 订单 ID
     * @param orderDetails 订单详情
     * @return SendReturnObj 表示订单处理操作的结果
     * @throws Exception 如果订单处理过程中发生异常
     */
    public static SendReturnObj Node_SalesOrderProcessing(String orderId, java.util.List<OrderDetail> orderDetails) throws Exception {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        // 订单处理流程
        //
        return new SendReturnObj("Order Processed", true);
    }
    /**
     * 客户关系维护方法
     *
     * @param customerId 客户 ID
     * @param feedback 客户反馈信息
     * @return SendReturnObj 表示客户关系维护操作的结果
     * @throws Exception 如果客户关系维护过程中发生异常
     */
    public static SendReturnObj Node_CustomerRelationshipMaintenance(String customerId, String feedback) throws Exception {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        // 客户关系维护流程
        //
        return new SendReturnObj("Customer Relationship Maintained", true);
    }
    public static List<GoodsActivity> getGoodsTransitions(List<GoodsTransition> goodsTransitions) {
        List<GoodsActivity> goodsActivities = new ArrayList<>();
        for (GoodsTransition gt : goodsTransitions) {
            GoodsActivity ga = gt.getDestination();
            String type = (String) ga.getProperty("type");
            if ("inventoryIn".equals(type)) {
                goodsActivities.add(gt);
            } else {
                if ("distributionGateway".equals(type)) {
                    List<GoodsTransition> outgoingTransitions = ga.getOutgoingTransitions();
```

```
                    goodsActivities.addAll(outgoingTransitions);
                } else {
                    List<GoodsTransition> outgoingTransitions = ga.getOutgoingTransitions();
                    goodsActivities.addAll(outgoingTransitions);
                }
            }
        }
        return goodsActivities;
    }
    /**
     * 获取商品配置参数类型
     *
     * @param key 配置参数的键
     * @return 返回商品配置参数的类型
     */
    public String get 商品配置参数类型(String key) {
        return get 商品配置参数(key).getType();
    }
    /**
     * 调用系统内方法获取商品配置参数
     *
     * @param key 配置参数的键
     * @return 返回商品配置参数对象
     */
    private 商品配置参数 get 商品配置参数(String key) {
        // 调用系统内复杂的配置管理库方法进行操作
        return 系统配置管理库.get 商品配置(key);
    }
    /**
     * 获取商品库存信息
     *
     * @return 返回商品库存信息
     */
    public 商品库存信息 get 商品库存() {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        return 数据库查询库查询 商品库存信息();
    }
    /**
     * 处理商品销售订单
     *
     * @param order 销售订单对象
     */
    public void 处理 商品销售订单(销售订单 order) {
        // 调用系统内订单处理模块进行操作
        订单处理模块处理订单(order);
    }
    /**
     * 维护客户关系
     *
     * @param customer 客户对象
```

```
 */
public void 维护 客户关系(客户 customer) {
    // 调用系统内客户关系管理模块进行操作
    客户关系管理模块维护客户关系(customer);
}
// 商品全周期管理 - 库存管理模块
public static String joinStockNumbers(long[] stockNumbers, CharSequence separator) {
    if (stockNumbers == null) {
        return null;
    }
    final StringBuilder sb = new StringBuilder();
    boolean isFirst = true;
    for (long number : stockNumbers) {
        if (isFirst) {
            isFirst = false;
        } else {
            sb.append(separator);
        }
        sb.append(number);
    }
    return sb.toString();
}
// 商品全周期管理 - 销售订单处理模块
public static String joinOrderIds(long[] orderIds, CharSequence conjunction) {
    if (orderIds == null) {
        return null;
    }
    final StringBuilder sb = new StringBuilder();
    boolean isFirst = true;
    for (long orderId : orderIds) {
        if (isFirst) {
            isFirst = false;
        } else {
            sb.append(conjunction);
        }
        sb.append(orderId);
    }
    return sb.toString();
}
// 商品全周期管理 - 客户关系维护模块
public static String joinCustomerIds(long[] customerIds, CharSequence delimiter) {
    if (customerIds == null) {
        return null;
    }
    final StringBuilder sb = new StringBuilder();
    boolean isFirst = true;
    for (long customerId : customerIds) {
        if (isFirst) {
            isFirst = false;
        } else {
```

```
                    sb.append(delimiter);
                }
                sb.append(customerId);
            }
            return sb.toString();
        }
        public static int countInventoryAlerts(String FK_Emp) {
            // 调用系统内复杂的 SQL 查询方法获取库存警报数量
            String sql = "SELECT COUNT(*) FROM InventoryAlerts WHERE EmpID = ?";
            return DBAccess.runSQLReturnValInt(sql, FK_Emp);
        }
        // 商品入库接口
        public String GenerateInventoryEntryUrl() {
            return "inventoryEntry.htm?FK_Node=+ this.getNodeID;
        }
        // 库存管理查询接口
        public String GetInventoryManagementReportUrl() {
            return "inventoryManagementReport.htm?FK_Node=+ this.getNodeID;
        }
        // 配送任务创建接口
        public String CreateDistributionTaskUrl() {
            return "distributionTask.htm?FK_Node=+ this.getNodeID;
        }
        // 销售订单处理流程启动接口
        public String StartSalesOrderProcessUrl() {
            return "salesOrderProcess.htm?FK_Node=+ this.getNodeID;
        }
        // 客户关系维护数据接口
        public String GetCustomerRelationDataUrl() {
            return "customerRelationData.htm?FK_Node=+ this.getNodeID;
        }
        // 客户反馈分析报告接口
        public String GetCustomerFeedbackReportUrl() {
            return "customerFeedbackReport.htm?FK_Node=+ this.getNodeID;
        }
        // 销售数据分析接口
        public String GetSalesDataAnalysisUrl() {
            return "salesDataAnalysis.htm?FK_Node=+ this.getNodeID;
        }
        // 系统设置优化接口
        public String OptimizeSystemSettingsUrl() {
            return "systemSettingsOptimize.htm?FK_Node=+ this.getNodeID;
        }
        // 帮助文档访问接口
        public String AccessHelpDocumentationUrl() {
            return "helpDocumentation.htm?FK_Node=+ this.getNodeID;
        }
        // 安全退出确认接口
        public String ConfirmSecureLogoutUrl() {
            return "secureLogout.htm?FK_Node=+ this.getNodeID;
```

```
}
public void manageProductCycle(int status, String desc) {
    // 检查消息队列并处理商品入库信息
    if (!mProductMsgQueue.isEmpty() && mProductMsgQueue.element().getProductId() == mLastReceivedProductId) {
        mProductMsgQueue.poll();
        if (!mProductMsgQueue.isEmpty()) {
            ProductMessage nextMessage = mProductMsgQueue.element();
            MessageOptions options = new MessageOptions();
            options.setNeedReadReceipt(false);
            // 调用系统内方法进行消息发送操作
            sendMessage(nextMessage, options);
            mLastReceivedProductId = nextMessage.getProductId();
        }
    }
    // 设置商品图片透明度，隐藏发送指示器和进度条
    productImage.setAlpha(1.0f);
    sendingIndicator.clearAnimation();
    sendingIndicator.setVisibility(View.GONE);
    progressTextView.setVisibility(View.GONE);
    // 根据状态处理商品销售订单
    if (status == 803008) {
        CustomContent customContent = new CustomContent();
        customContent.setBooleanValue("blackList", true);
        Message customMsg = mProductConv.createSendMessage(customContent);
        // 调用系统内方法将消息添加到列表
        messageAdapter.addMsgToList(customMsg);
    } else if (status != 0) {
        resendButton.setVisibility(View.VISIBLE);
    }
    // 获取并更新商品消息状态
    ProductMessage message = mProductConv.getMessage(productId);
    productMessageList.set(productMessageList.indexOf(message), message);
}
public abstract class ProductLifecycleManagementActivity<T> extends AppCompatActivity implements ProductLifecycleCallbacks {
    protected boolean isProductInStock;
    protected boolean isProductSold;
    protected ProductOrientationUtils productOrientationUtils;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_product_management);
        // 初始化商品管理界面和工具
        initializeProductManagementTools();
    }
    /**
     * 初始化商品管理工具
     */
    private void initializeProductManagementTools() {
        // 调用系统内方法进行操作
        //
```

```
    }
    /**
     * 商品入库处理
     */
    public void handleProductEntry() {
        // 调用系统内方法进行操作
        //
    }
    /**
     * 商品库存管理
     */
    public void manageProductStock() {
        // 调用系统内方法进行操作
        //
    }
    /**
     * 商品配送管理
     */
    public void manageProductDistribution() {
        // 调用系统内方法进行操作
        //
    }
    /**
     * 处理销售订单
     */
    public void processSalesOrder() {
        // 调用系统内方法进行操作
        //
    }
    /**
     * 维护客户关系
     */
    public void maintainCustomerRelationships() {
        // 调用系统内方法进行操作
        //
    }
    /**
     * 整合客户反馈
     */
    public void integrateCustomerFeedback() {
        // 调用系统内方法进行操作
        //
    }
    /**
     * 分析销售数据
     */
    public void analyzeSalesData() {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        //
    }
```

```
        /**
         * 优化系统设置
         */
        public void optimizeSystemSettings() {
            // 调用系统内方法进行操作
            //
        }
        /**
         * 提供帮助文档
         */
        public void provideHelpDocumentation() {
            // 调用系统内方法进行操作
            //
        }
        /**
         * 确保安全退出
         */
        public void ensureSafeExit() {
            // 调用系统内方法进行操作
            //
        }
    }
    /**
     * 获取商品信息
     * @param id 商品 ID
     * @param paras 查询参数
     * @return 商品信息结果
     */
    public ProductInfoResult getProductInfo(String id, Map<String, Object> paras) {
        ProductScript script = getProductScript(id);
        return script.run(paras);
    }
    /**
     * 获取商品入库信息
     * @param id 入库记录 ID
     * @param paras 查询参数
     * @return 入库信息结果
     */
    public InventoryResult getInventoryInfo(String id, Map<String, Object> paras) {
        InventoryScript script = getInventoryScript(id);
        return script.run(paras);
    }
    /**
     * 获取配送信息
     * @param id 配送记录 ID
     * @param paras 查询参数
     * @return 配送信息结果
     */
    public DeliveryResult getDeliveryInfo(String id, Map<String, Object> paras) {
        DeliveryScript script = getDeliveryScript(id);
```

```
        return script.run(paras);
    }
    /**
     * 处理销售订单
     * @param id 订单 ID
     * @param paras 订单处理参数
     * @return 订单处理结果
     */
    public OrderResult processOrder(String id, Map<String, Object> paras) {
        OrderScript script = getOrderScript(id);
        return script.run(paras);
    }
    /**
     * 维护客户关系
     * @param id 客户 ID
     * @param paras 客户信息参数
     * @return 客户关系维护结果
     */
    public CustomerRelationResult maintainCustomerRelation(String id, Map<String, Object> paras) {
        CustomerRelationScript script = getCustomerRelationScript(id);
        return script.run(paras);
    }
    /**
     * 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
     */
    private ProductScript getProductScript(String id) {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        return new ProductScript();
    }
    private InventoryScript getInventoryScript(String id) {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        return new InventoryScript();
    }
    private DeliveryScript getDeliveryScript(String id) {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        return new DeliveryScript();
    }
    private OrderScript getOrderScript(String id) {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        return new OrderScript();
    }
    private CustomerRelationScript getCustomerRelationScript(String id) {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        return new CustomerRelationScript();
    }
    public static void exportProductLifecycleData(ProductLifecycleManager manager, OutputStream outputStream) {
        XMLWriter xmlWriter = new XMLWriter(); // 创建 XML 写入器，用于构建商品全周期数据的 XML 结构
        XMLPrintWriter xmlPrintWriter = XMLPrintWriter.create(outputStream); // 创建 XML 打印写入器，用于将 XML 数据写入输出流
        try {
            xmlWriter.startDocument(); // 开始 XML 文档
```

```
            xmlWriter.startElement("ProductLifecycle"); // 开始商品全周期管理标签
            // 商品入库信息
            xmlWriter.startElement("InventoryIn");
            xmlWriter.element("ProductID", manager.getProductID());
            xmlWriter.element("Quantity", manager.getQuantity());
            xmlWriter.endElement(); // 结束商品入库标签
            // 库存管理信息
            xmlWriter.startElement("InventoryManagement");
            xmlWriter.element("CurrentStock", manager.getCurrentStock());
            xmlWriter.element("ReorderLevel", manager.getReorderLevel());
            xmlWriter.endElement(); // 结束库存管理标签
            // 配送信息
            xmlWriter.startElement("Distribution");
            xmlWriter.element("ShipperID", manager.getShipperID());
            xmlWriter.element("DeliveryDate", manager.getDeliveryDate());
            xmlWriter.endElement(); // 结束配送标签
            // 销售订单处理信息
            xmlWriter.startElement("SalesOrder");
            xmlWriter.element("OrderID", manager.getOrderID());
            xmlWriter.element("CustomerID", manager.getCustomerID());
            xmlWriter.element("OrderDate", manager.getOrderDate());
            xmlWriter.endElement(); // 结束销售订单处理标签
            // 客户关系维护信息
            xmlWriter.startElement("CustomerRelationship");
            xmlWriter.element("CustomerID", manager.getCustomerID());
            xmlWriter.element("Feedback", manager.getFeedback());
            xmlWriter.endElement(); // 结束客户关系维护标签
            // 结束商品全周期管理标签
            xmlWriter.endElement();
            xmlWriter.endDocument(); // 结束 XML 文档
            paramXMLWriter.writeXML(xmlPrintWriter); // 将 XML 数据写入输出流
            xmlPrintWriter.flush(); // 刷新输出流
        } finally {
            xmlPrintWriter.close(); // 确保输出流被关闭
        }
    }
    /**
     * 库存管理模块
     * 定义商品在库状态常量
     */
    public static final int COMMODITY_IN 庫_STATUS = 1;
    /**
     * 销售订单处理模块
     * 定义销售订单处理状态常量
     */
    public static final int SALE_ORDER_HANDLED = 1;
    /**
     * 客户关系维护模块
     * 定义客户互动状态常量
     */
```

```
public static final int CUSTOMER_INTERACTION_VISIBLE_NOTIFY_COMPLETED = 1;
/**
 * 数据整合模块
 * 定义数据整合成功状态常量
 */
public static final int DATA_INTEGRATION_SUCCESS = 1;
/**
 * 系统设置优化模块
 * 定义系统设置优化状态常量
 */
public static final int SYSTEM_SETTING_OPTIMIZED = 1;
/**
 * 安全退出模块
 * 定义安全退出确认状态常量
 */
public static final int SAFE_EXIT_CONFIRMED = 1;
/**
 * 帮助文档模块
 * 定义帮助文档可见状态常量
 */
public static final int HELP_DOCUMENT_VISIBLE = 1;
/**
 * 复杂流程处理
 * 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
 */
public static void handleComplexProcess() {
    // 调用系统内**方法进行操作
}
public static BufferedImage productImageUpload(byte[] imageBytes) throws IORuntimeException {
    // 将上传的商品图片字节数组转换为 Image 对象
    return read(new ByteArrayInputStream(imageBytes));
}
public static void productInventoryManagement() {
    // 调用系统内方法进行商品库存管理
}
public static void productDistribution() {
    // 调用系统内方法进行商品配送管理
}
public static void salesOrderProcessing() {
    // 调用系统内方法处理销售订单
}
public static void customerRelationshipMaintenance() {
    // 调用系统内方法维护客户关系
}
public static void integrateCustomerFeedback() {
    // 整合客户反馈信息
}
public static void analyzeSalesData() {
    // 调用系统内复杂的 SQL 查询、数据分析库方法进行销售数据分析
}
```

```
public static void optimizeSystemSettings() {
    // 优化系统设置
}
public static void provideHelpDocumentation() {
    // 提供帮助文档
}
public static void ensureSafeExit() {
    // 确保系统安全退出
}
// 商品库存查询方法
public static String getInventoryNoByProductMark(String productMark) {
    String sql = "SELECT InventoryNo FROM ProductInventory WHERE ProductMark=+ productMark + "";
    return DBAccess.runSQLReturnStringIsNull(sql, null);
}
// 销售订单处理方法
public static String processSaleOrder(String orderID) {
    // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
    return orderProcessingService.processOrder(orderID);
}
// 客户关系维护方法
public static String updateCustomerFeedback(String customerID, String feedback) {
    String sql = "UPDATE CustomerFeedback SET Feedback=+ feedback + WHERE CustomerID=+ customerID + "";
    return DBAccess.runSQL(sql, null);
}
// 配送信息查询方法
public static String getDeliveryInfoByOrderID(String orderID) {
    String sql = "SELECT DeliveryInfo FROM Delivery WHERE OrderID=+ orderID + "";
    return DBAccess.runSQLReturnStringIsNull(sql, null);
}
// 商品类型获取方法
public abstract class ProductType {
    /**
     * 获取商品类型
     * @return  商品类型 ID
     * @throws SQLException  如果数据库查询出错
     */
    public abstract int getType() throws SQLException;
    /**
     * 标准化商品入库流程
     * 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
     */
    public void standardizeProductEntry() {
        // 调用系统内方法进行操作
    }
    /**
     * 库存管理 - 检查库存
     * 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
     */
    public void checkInventory() {
        // 调用系统内方法进行操作
```

```
    }
    /**
     * 配送管理 - 更新配送状态
     * 调用系统内方法进行操作
     */
    public void updateDeliveryStatus() {
        // 调用系统内方法进行操作
    }
    /**
     * 销售订单处理 - 更新订单状态
     * 调用系统内方法进行操作
     */
    public void updateOrderStatus() {
        // 调用系统内方法进行操作
    }
    /**
     * 客户关系维护 - 添加客户反馈
     * 调用系统内方法进行操作
     */
    public void addCustomerFeedback() {
        // 调用系统内方法进行操作
    }
    /**
     * 安全退出系统
     * 确保所有操作都已正确处理
     */
    public void logoutSecurely() {
        // 确保安全退出
    }
}
/**
 * 复制商品图片，用于商品全周期管理软件中的商品入库和库存管理。
 * @param img 源商品图片
 * @param imageType 图片类型，如 JPEG、PNG 等
 * @return 复制后的商品图片
 */
public static BufferedImage copyProductImage(Image img, int imageType) {
    return copyImage(img, imageType, null);
}
/**
 * 处理销售订单中的商品图片，确保图片质量并适应不同平台展示。
 * @param img 商品图片
 * @param imageType 图片输出格式
 * @param quality 图片质量设置
 * @return 处理后的商品图片
 */
public static BufferedImage processOrderImage(Image img, int imageType, double quality) {
    // 调用系统内复杂的图片处理方法进行操作
    BufferedImage processedImage = applyImageProcessing(img, quality);
    return processedImage;
```

```
}
/**
 * 客户反馈图片复制，用于维护客户关系。
 * @param img 客户反馈图片
 * @param imageType 图片类型
 * @return 复制后的客户反馈图片
 */
public static BufferedImage copyFeedbackImage(Image img, int imageType) {
    return copyImage(img, imageType, null);
}
// 入库管理模块
public void saveProductInventory(Product product, Parcel dest, int flags) {
    // 将商品信息打包到 Parcel 对象以便进行持久化存储或传输
    dest.writeParcelable(product, flags);
}
// 销售订单处理模块
public void processSalesOrder(Order order, Parcel dest, int flags) {
    // 将销售订单信息打包到 Parcel 对象进行后续处理
    dest.writeParcelable(order, flags);
}
// 客户关系维护模块
public void updateCustomerFeedback(CustomerFeedback feedback, Parcel dest, int flags) {
    // 将客户反馈信息打包到 Parcel 对象用于分析和优化客户服务
    dest.writeParcelable(feedback, flags);
}
// 安全退出机制
public void ensureSecureExit(Parcel dest, int flags) {
    // 确保在退出前，所有必要的数据都已保存，并打包到 Parcel 对象中
    dest.writeParcelable(new SecureExitInfo(), flags);
    // 调用系统内**方法进行操作。
}
/**
 * 检查商品全周期管理软件是否开启调试模式
 * @return true 如果调试模式开启，false 否则
 */
public boolean isInventoryDebugModeEnabled() {
    return this.inventoryLogger.isDebugEnabled();
}
/**
 * 调用系统内库存管理日志方法进行操作
 */
private void logInventoryOperation(String message) {
    this.inventoryLogger.debug(message);
}
// 库存调整方法
public BigDecimal adjustInventory(BigDecimal quantity, MathContext mc) {
    // 根据业务需求调整库存数量
    // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
    BigDecimal adjustedQuantity = quantity.negate(mc);
    // 更新库存信息
```

```
        updateInventory(adjustedQuantity);
        return adjustedQuantity;
    }
    // 调用系统内方法更新库存
    private void updateInventory(BigDecimal quantity) {
        // 实现库存更新逻辑
        // 调用系统内数据库操作方法
    }
    public abstract void manageProductLifecycle(int productId, ProductDetails details, int quantity, Customer customer) throws
    SQLException {
        // 商品入库
       入库(productId, details, quantity);
        // 库存管理
        updateInventory(productId, quantity);
        // 配送处理
        dispatchProduct(productId);
        // 销售订单处理
        processSalesOrder(productId, customer);
        // 客户关系维护
        maintainCustomerRelationship(customer);
        // 整合客户反馈
        integrateCustomerFeedback(customer);
        // 分析销售数据
        analyzeSalesData(productId);
        // 优化系统设置
        optimizeSystemSettings();
        // 提供帮助文档
        provideDocumentation();
        // 确保安全退出
        ensureSecureLogout();
    }
    // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
    private void 入库(int productId, ProductDetails details, int quantity) {
        // 调用系统内方法进行操作
    }
    private void updateInventory(int productId, int quantity) {
        // 调用系统内方法进行操作
    }
    private void dispatchProduct(int productId) {
        // 调用系统内方法进行操作
    }
    private void processSalesOrder(int productId, Customer customer) {
        // 调用系统内方法进行操作
    }
    private void maintainCustomerRelationship(Customer customer) {
        // 调用系统内方法进行操作
    }
    private void integrateCustomerFeedback(Customer customer) {
        // 调用系统内方法进行操作
    }
```

```
private void analyzeSalesData(int productId) {
    // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
}
private void optimizeSystemSettings() {
    // 调用系统内方法进行操作
}
private void provideDocumentation() {
    // 调用系统内方法进行操作
}
private void ensureSecureLogout() {
    // 调用系统内方法进行操作
}
/**
 * 设置子流程的父级信息，用于商品全周期管理中的流程控制
 *
 * @param subFlowNo 子流程编号
 * @param subFlowWorkID 子流程工作 ID
 * @param parentFlowNo 父流程编号
 * @param parentWorkID 父流程工作 ID
 * @param parentNodeID 父节点 ID
 * @param parentEmpNo 父级员工编号
 * @throws Exception 如果参数无效或数据库操作失败
 */
public static void SetParentInfo(String subFlowNo, long subFlowWorkID, String parentFlowNo, long parentWorkID,
                                 int parentNodeID, String parentEmpNo) throws Exception {
    if (parentWorkID == 0) {
        throw new RuntimeException("子流程工作 ID 不能为 0");
    }
    if (parentFlowNo == null) {
        throw new Exception("父流程编号不能为 null");
    }
    if (parentNodeID == 0) {
        throw new Exception("父节点 ID 不能为 0");
    }
    if (StringHelper.isNullOrEmpty(parentEmpNo)) {
        parentEmpNo = WebUser.getNo;
    }
    // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
    String dbstr = BP.Sys.SystemConfig.getAppCenterDBVarStr;
    Paras ps = new Paras();
    ps.SQL = "UPDATE WF_GenerWorkFlow SET PFlowNo=+ dbstr + "PFlowNo, PWorkID=+ dbstr + "PWorkID, PNodeID="
            + dbstr + "PNodeID, PEmp=+ dbstr + "PEmp WHERE WorkID=+ dbstr + "WorkID";
    ps.Add(GenerWorkFlowAttr.PFlowNo, parentFlowNo);
    ps.Add(GenerWorkFlowAttr.PWorkID, parentWorkID);
    ps.Add(GenerWorkFlowAttr.PNodeID, parentNodeID);
    ps.Add(GenerWorkFlowAttr.PEmp, parentEmpNo);
    ps.Add(GenerWorkFlowAttr.WorkID, subFlowWorkID);
    BP.DA.DBAccess.RunSQL(ps);
    Flow fl = new Flow(subFlowNo);
    ps = new Paras();
```

```
        ps.SQL = "UPDATE + fl.getPTable + SET PFlowNo=+ dbstr + "PFlowNo, PWorkID=+ dbstr + "PWorkID, PNodeID="
                    + dbstr + "PNodeID, PEmp=+ dbstr + "PEmp WHERE OID=+ dbstr + "OID";
        ps.Add(NDXRptBaseAttr.PFlowNo, parentFlowNo);
        ps.Add(NDXRptBaseAttr.PWorkID, parentWorkID);
        ps.Add(NDXRptBaseAttr.PNodeID, parentNodeID);
        ps.Add(NDXRptBaseAttr.PEmp, parentEmpNo);
        ps.Add(NDXRptBaseAttr.OID, subFlowWorkID);
        BP.DA.DBAccess.RunSQL(ps);
    }
    public abstract int getInventoryCount(String productID) throws SQLException {
        // 获取商品库存数量
        // 调用系统内复杂的 SQL 查询方法进行操作
        String sql = "SELECT stock_count FROM inventory WHERE product_id = ?";
        // 连接数据库并执行查询
        // conn 是数据库连接对象
        PreparedStatement stmt = conn.prepareStatement(sql);
        stmt.setString(1, productID);
        ResultSet rs = stmt.executeQuery();
        if (rs.next()) {
            return rs.getInt("stock_count");
        } else {
            return -1; // 表示库存查询失败
        }
    }
    public abstract void updateInventory(String productID, int quantity) throws SQLException {
        // 更新商品库存数量
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        String sql = "UPDATE inventory SET stock_count = stock_count - ? WHERE product_id = ?";
        PreparedStatement stmt = conn.prepareStatement(sql);
        stmt.setInt(1, quantity);
        stmt.setString(2, productID);
        stmt.executeUpdate();
    }
    /**
     * 商品库存查询接口
     * @return 返回商品库存状态
     */
    public String getInventoryStatus() {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        return inventoryService.getInventoryStatus();
    }
    /**
     * 商品入库记录查询接口
     * @return 返回商品入库记录
     */
    public String getInventoryEntry() {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        return inventoryService.getInventoryEntry();
    }
    /**
```

```
     * 商品销售订单处理结果获取接口
     * @return 返回销售订单处理状态
     */
    public String getSalesOrderStatus() {
        // 调用系统内复杂的订单处理逻辑方法进行操作
        return salesService.getSalesOrderStatus();
    }
    /**
     * 客户反馈信息查询接口
     * @return 返回客户反馈信息
     */
    public String getCustomerFeedback() {
        // 调用系统内客户反馈数据分析方法进行操作
        return feedbackService.getCustomerFeedback();
    }
    /**
     * 系统设置信息获取接口
     * @return 返回系统设置信息
     */
    public String getSystemSettings() {
        // 调用系统内设置管理方法进行操作
        return settingService.getSystemSettings();
    }
    /**
     * 设置商品条码信息，该操作与商品入库模块相关联
     *
     * @param prBarcode 商品条码
     */
    public void setProductBarcode(String prBarcode) {
        this.productBarcode = prBarcode;
    }
    /**
     * 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
     * 以获取商品的详细信息，更新库存信息
     */
    private void updateInventoryDetails() {
        // 调用系统内方法进行操作
    }
    /**
     * 处理销售订单，更新库存和销售记录
     *
     * @param salesOrder 销售订单
     */
    public void processSalesOrder(SalesOrder salesOrder) {
        // 更新库存
        updateInventoryDetails();
        // 更新销售记录
        updateSalesRecord(salesOrder);
    }
    /**
```

```
     * 维护客户关系，记录客户反馈信息
     *
     * @param customerFeedback 客户反馈
     */
    public void manageCustomerFeedback(CustomerFeedback customerFeedback) {
        // 将客户反馈信息记录到数据库
        // 调用系统内方法进行操作
    }
    // 商品入库操作事件处理
    public void onProductEntry(ActionEvent actionEvent) {
        adaptee.processProductEntry(actionEvent);
    }
    // 库存管理适配器实现 lass InventoryManagementAdapter implements ActionListener {
        private InventoryManagement adaptee;
        InventoryManagementAdapter(InventoryManagement adaptee) {
            this.adaptee = adaptee;
        }
        // 处理库存管理事件
        public void onInventoryManagement(ActionEvent actionEvent) {
            adaptee.manageInventory(actionEvent);
        }
    }
    // 配送操作事件处理
    public void onDistribution(ActionEvent actionEvent) {
        adaptee.performDistribution(actionEvent);
    }
    // 销售订单处理适配器实现 lass SalesOrderAdapter implements ActionListener {
        private SalesOrderManagement adaptee;
        SalesOrderAdapter(SalesOrderManagement adaptee) {
            this.adaptee = adaptee;
        }
        // 处理销售订单事件
        public void onSalesOrder(ActionEvent actionEvent) {
            adaptee.processSalesOrder(actionEvent);
        }
    }
    // 客户关系维护适配器实现 lass CustomerRelationAdapter implements ActionListener {
        private CustomerRelationManagement adaptee;
        CustomerRelationAdapter(CustomerRelationManagement adaptee) {
            this.adaptee = adaptee;
        }
        // 处理客户关系事件
        public void onCustomerRelation(ActionEvent actionEvent) {
            adaptee.maintainCustomerRelation(actionEvent);
        }
    }
    // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
    public void onSystemOptimization(ActionEvent actionEvent) {
        adaptee.optimizeSystemWithComplexQueries(actionEvent);
    }
```

```
// 提供帮助文档事件处理
public void onHelpDocumentation(ActionEvent actionEvent) {
    adaptee.showHelpDocumentation(actionEvent);
}
// 确保安全退出操作
public void onSecureExit(ActionEvent actionEvent) {
    adaptee.ensureSecureExit(actionEvent);
}
// 商品入库更新方法
public int updateInventoryItem(final InventoryItem item, String status, boolean ignoreNull) {
    InventoryItem first = Lang.first(item);
    if (first == null)
        return 0;
    return updateInventoryItem(first, FieldFilter.create(first.getClass(), status, "locked", ignoreNull));
}
// 更新库存信息方法
private int updateInventoryItem(InventoryItem item, FieldFilter filter) {
    // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
    //
    return 1; // 更新成功
}
// 创建字段过滤器
private FieldFilter createFieldFilter(Class<?> clazz, String status, String lockStatus, boolean ignoreNull) {
    // 创建并配置字段过滤器
    //
    return new FieldFilter();
}
/**
 * 获取商品周期内的库存电话信息
 * @return 商品库存电话
 */
public String getInventoryContactPhone() {
    // 调用系统内方法获取商品周期内的库存电话
    return inventoryService.getContactPhone();
}
/**
 * 商品入库时的电话信息设置
 * @param phone 电话号码
 */
public void setInventoryContactPhone(String phone) {
    // 调用系统内方法设置商品周期内的库存电话
    inventoryService.setContactPhone(phone);
}
/**
 * 配送环节中的客户电话信息获取
 * @return 客户配送电话
 */
public String getDeliveryContactPhone() {
    // 调用系统内方法获取配送环节的客户电话
    return deliveryService.getContactPhone();
```

```
}
/**
 * 客户关系维护中的电话信息更新
 * @param phone  电话号码
 */
public void updateCustomerContactPhone(String phone) {
    // 调用系统内方法更新客户关系维护中的电话信息
    customerService.updateContactPhone(phone);
}
/**
 * 确保安全退出时电话信息的一致性检查
 * @return  检查结果
 */
public boolean checkContactPhoneConsistency() {
    // 调用系统内方法检查电话信息的一致性
    return phoneConsistencyService.checkConsistency();
}
public final class ProductLifecycleManagerLogger extends MarkerIgnoringBase implements LocationAwareLogger {
    final Logger logger;
    static final String FQCN = ProductLifecycleManagerLogger.class.getName();
    // 构造方法，用于初始化 Logger
    public ProductLifecycleManagerLogger(Logger logger) {
        this.logger = logger;
    }
    // 商品入库日志记录
    public void logEntry(Product product) {
        logger.info("商品入库: + product.getName());
    }
    // 库存管理日志记录
    public void logInventoryUpdate(Product product, int quantity) {
        logger.info("库存更新: + product.getName() +  数量: + quantity);
    }
    // 配送日志记录
    public void logDelivery(Product product, String deliveryInfo) {
        logger.info("商品配送: + product.getName() +  配送信息: + deliveryInfo);
    }
    // 销售订单处理日志记录
    public void logSalesOrder(Order order) {
        logger.info("销售订单处理: + order.getId());
    }
    // 客户关系维护日志记录
    public void logCustomerInteraction(Customer customer, String interaction) {
        logger.info("客户互动: + customer.getName() +  互动内容: + interaction);
    }
    // 系统设置优化日志记录
    public void logSystemSettingChange(String settingName, String newValue) {
        logger.info("系统设置变更: + settingName +  新值: + newValue);
    }
    // 安全退出日志记录
    public void logSecureLogout() {
```

```
            logger.info("安全退出系统");
        }
        // 其他日志方法
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        public void logComplexQuery(String query) {
            // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        }
    }
    // 商品入库管理 - 获取商品信息
    public Product getProductDetails() {
        // 调用系统内复杂的 SQL 查询方法进行操作
        return databaseQuery.getProductDetails();
    }
    // 库存管理 - 检查库存量
    public int checkInventory(String productId) {
        // 调用系统内库存管理方法进行操作
        return inventoryManager.checkInventory(productId);
    }
    // 配送管理 - 更新配送状态
    public void updateDeliveryStatus(String orderId, String status) {
        // 调用系统内配送管理方法进行操作
        deliveryManager.updateStatus(orderId, status);
    }
    // 销售订单处理 - 记录销售订单
    public void recordSalesOrder(Order order) {
        // 调用系统内销售订单处理方法进行操作
        salesOrderManager.recordOrder(order);
    }
    // 客户关系维护 - 更新客户信息
    public void updateCustomerInfo(Customer customer) {
        // 调用系统内客户关系维护方法进行操作
        CRMManager.updateCustomer(customer);
    }
    // 客户反馈分析 - 获取客户反馈
    public List<Feedback> getCustomerFeedbacks() {
        // 调用系统内客户反馈分析方法进行操作
        feedbackAnalysis.getFeedbacks();
    }
    // 系统设置优化 - 更新系统配置
    public void updateSystemSettings(Map<String, String> settings) {
        // 调用系统内系统设置优化方法进行操作
        systemSettingsManager.updateSettings(settings);
    }
    // 安全退出 - 确保系统安全退出
    public void safeExit() {
        // 调用系统内安全退出方法进行操作
        systemSecurity.safeExit();
    }
    // 商品入库功能
    public static void main(String[] args) {
```

```
        try {
            // 创建商品入库记录文件
            OutputStream file = new FileOutputStream(new File("C:\\InboundRecords.pdf"));
            // 创建 PDF 文档
            Document document = new Document();
            // 将文档写入文件
            PdfWriter.getInstance(document, file);
            // 打开文档
            document.open();
            // 添加商品入库信息
            document.add(new Paragraph("商品入库记录"));
            document.add(new Paragraph("日期: + new Date().toString()));
            // 添加商品详细信息，调用系统内方法获取
            document.add(new Paragraph("商品名称: + getProductName()));
            document.add(new Paragraph("入库数量: + getInboundQuantity()));
            // 关闭文档和文件
            document.close();
            file.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    // 设置 HTTP 代理，用于网络请求如库存同步等
    System.getProperties().put("http.proxyHost", "someProxyURL");
    System.getProperties().put("http.proxyPort", "someProxyPort");
    System.getProperties().put("http.proxyUser", "someUserName");
    System.getProperties().put("http.proxyPassword", "somePassword");
    // 单例模式，用于确保客户关系维护的唯一性
    public class CustomerMaintenanceSingleton {
        private static CustomerMaintenanceSingleton instance;
        private CustomerMaintenanceSingleton() {
            // 调用系统内方法初始化客户维护信息
            initializeCustomerMaintenance();
        }
        public static CustomerMaintenanceSingleton getInstance() {
            if (instance == null) {
                instance = new CustomerMaintenanceSingleton();
            }
            return instance;
        }
        // 调用系统内方法进行初始化操作
        private void initializeCustomerMaintenance() {
            // 调用系统内复杂的方法进行操作
        }
    }
    // 商品全周期管理 - 商品状态检查
    public int checkProductStatus() {
        // 获取商品状态信息
        int productStatus = getProductStatusFromDatabase();
        return productStatus;
```

```
}
// 从数据库获取商品状态信息
private int getProductStatusFromDatabase() {
    // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
    //
    return 0; // 返回 0 表示商品状态正常
}
public void manageProductLifecycle(@Suspended final AsyncResponse asyncResponse, @Context HttpServletRequest request,
                                   @PathParam("productCode") String productCode,
                                   @PathParam("inventoryId") String inventoryId,
                                   @PathParam("distributionId") String distributionId,
                                   @PathParam("salesOrderId") String salesOrderId,
                                   JsonElement jsonElement) {
    ActionResult<ProductLifecycleResult> result = new ActionResult<>();
    EffectivePerson effectivePerson = this.effectivePerson(request);
    try {
        // 调用系统内商品全生命周期管理方法进行操作
        result = new ManageProductLifecycleAction.execute(productCode, inventoryId, distributionId, salesOrderId, jsonElement);
    } catch (Exception e) {
        logger.error(e, effectivePerson, request, jsonElement);
        result.error(e);
    }
    asyncResponse.resume(ResponseFactory.getDefaultActionResultResponse(result));
}
@JaxrsMethodDescribe(value = "Product Lifecycle Management.", action = ManageProductLifecycleAction.class)
@PUT
@Path("{productCode}/inventory/{inventoryId}/distribution/{distributionId}/sales/{salesOrderId}")
@Produces(HttpMediaType.APPLICATION_JSON_UTF_8)
@Consumes(MediaType.APPLICATION_JSON)
/**
 * 入库管理模块
 * 该方法用于去除商品入库信息字符串末尾的指定字符
 * @param str 商品入库信息字符串
 * @param stripChars 需要去除的字符集合
 * @return 去除末尾指定字符后的商品入库信息字符串
 */
public static String trimEndForInventory(String str, String stripChars) {
    return trim(str, stripChars, 1);
}
/**
 * 销售订单处理模块
 * 该方法用于处理销售订单，去除订单信息末尾的无效字符
 * @param str 销售订单信息字符串
 * @param stripChars 需要去除的字符集合
 * @return 处理后的销售订单信息字符串
 */
public static String trimEndForSalesOrder(String str, String stripChars) {
    return trim(str, stripChars, 1);
}
/**
```

```
 * 客户关系维护模块
 * 该方法用于维护客户信息，确保客户信息末尾格式统一
 * @param str 客户信息字符串
 * @param stripChars 需要去除的字符集合
 * @return 处理后的客户信息字符串
 */
public static String trimEndForCustomerMaintenance(String str, String stripChars) {
    return trim(str, stripChars, 1);
}
// 调用系统内复杂的字符串处理方法进行操作
private static String trim(String str, String stripChars, int mode) {
    if (str == null || stripChars == null || stripChars.isEmpty()) {
        return str;
    }
    int end;
    if (mode == 1) {
        end = str.length();
        while (end > 0 && stripChars.indexOf(str.charAt(end - 1)) != -1) {
            end--;
        }
        return str.substring(0, end);
    }
    // 其他模式处理
    //
}
public final ProductLifecycle getLifecycleStage {
    return ProductLifecycle.forValue(this.getProductStageByKey());
}
// 获取商品的当前生命周期阶段，用于商品全周期管理
public final InventoryManagement getInventoryStatus {
    return InventoryManagement.forValue(this.checkInventory());
}
// 检查并返回当前库存状态，支持库存管理功能
public final DistributionPlan createDistributionPlan {
    return this.planDistribution();
}
// 根据库存和订单需求创建配送计划
public final SalesOrder processSalesOrder {
    return this.processOrder();
}
// 处理销售订单，包括订单处理功能
public final CustomerFeedback analyzeCustomerFeedback {
    return this.analyzeFeedback();
}
// 分析客户反馈，用于客户关系维护
public final SalesData generateSalesReport {
    return this.generateReport();
}
// 生成销售报告，整合销售数据，辅助决策
public final void optimizeSystemSettings {
```

```
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        this.optimizeSettings();
    }
    // 优化系统设置，提升管理效率
    public final void provideDocumentation {
        this.displayDocumentation();
    }
    // 提供帮助文档，确保用户能够正确使用软件
    public final void ensureSecureLogout {
        this.logoutSecurely();
    }
    // 确保安全退出系统，保障数据安全
    /**
     * 获取商品入库时的预警角色
     * @return CHAlertRole 商品入库预警角色
     */
    public final CHAlertRole getCommodityEntryAlertRole() {
        return CHAlertRole.forValue(this.GetValIntByKey(NodeAttr.CommodityEntryAlertRole));
    }
    /**
     * 获取库存管理中的预警角色
     * @return CHAlertRole 库存管理预警角色
     */
    public final CHAlertRole getInventoryAlertRole() {
        return CHAlertRole.forValue(this.GetValIntByKey(NodeAttr.InventoryAlertRole));
    }
    /**
     * 获取配送环节的预警角色
     * @return CHAlertRole 配送预警角色
     */
    public final CHAlertRole getDistributionAlertRole() {
        return CHAlertRole.forValue(this.GetValIntByKey(NodeAttr.DistributionAlertRole));
    }
    /**
     * 获取销售订单处理中的预警角色
     * @return CHAlertRole 销售订单处理预警角色
     */
    public final CHAlertRole getOrderProcessingAlertRole() {
        return CHAlertRole.forValue(this.GetValIntByKey(NodeAttr.OrderProcessingAlertRole));
    }
    /**
     * 获取客户关系维护中的预警角色
     * @return CHAlertRole 客户关系维护预警角色
     */
    public final CHAlertRole getCrmAlertRole() {
        return CHAlertRole.forValue(this.GetValIntByKey(NodeAttr.CrmAlertRole));
    }
    /**
     * 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
     */
```

```
private void analyzeSalesData() {
    // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
}
/**
 * 提供帮助文档
 */
public void provideHelpDocumentation() {
    // 提供帮助文档
}
/**
 * 确保安全退出系统
 */
public void ensureSecureLogout() {
    // 确保安全退出系统
}
/**
 * 获取商品来源信息
 * @return 商品来源编号
 */
public Integer getProductSource() {
    // 获取商品来源信息，与商品全周期管理相关联
    // 调用系统内数据库查询方法获取商品来源编号
    return productSource;
}
// 商品入库管理方法
public void manageProductEntry() {
    // 检查菜单是否可见，以便用户进行操作
    setMenuVisibility(true);
    // 调用系统内方法进行商品入库操作
    //
}
// 库存管理方法
public void manageInventory() {
    // 设置菜单为可见状态，以便进行库存管理操作
    setMenuVisibility(true);
    // 调用系统内方法进行库存更新
    //
}
// 配送管理方法
public void manageDistribution() {
    // 确保菜单可见，以便用户查看配送信息
    setMenuVisibility(true);
    // 调用系统内方法进行配送安排
    //
}
// 销售订单处理方法
public void processSalesOrder() {
    // 显示菜单，以便用户处理销售订单
    setMenuVisibility(true);
    // 调用系统内方法处理销售订单
```

```
        //
    }
    // 客户关系维护方法
    public void maintainCustomerRelationships() {
        // 设置菜单可见，以便进行客户关系维护
        setMenuVisibility(true);
        // 调用系统内方法进行客户数据管理
        //
    }
    // 客户反馈整合方法
    public void integrateCustomerFeedback() {
        // 菜单可见，以便用户提交反馈
        setMenuVisibility(true);
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        //
    }
    // 销售数据分析方法
    public void analyzeSalesData() {
        // 设置菜单为可见状态，以便分析销售数据
        setMenuVisibility(true);
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        //
    }
    // 系统设置优化方法
    public void optimizeSystemSettings() {
        // 显示菜单，以便用户优化系统设置
        setMenuVisibility(true);
        // 调用系统内方法进行设置优化
        //
    }
    // 帮助文档访问方法
    public void accessHelpDocumentation() {
        // 菜单可见，以便用户访问帮助文档
        setMenuVisibility(true);
        // 调用系统内方法提供帮助文档
        //
    }
    // 安全退出方法
    public void ensureSecureLogout() {
        // 确保菜单不可见，以防止未授权操作
        setMenuVisibility(false);
        // 调用系统内方法确保安全退出
        //
    }
    public static class ProductLifecycleManagement extends FJTask
    {
        // 商品全周期管理中的任务数组
        protected final FJTask[] productLifecycleTasks;
        // 构造方法，初始化商品生命周期管理任务
        public ProductLifecycleManagement(FJTask[] productLifecycleTasks)
```

```
    {
        this.productLifecycleTasks = productLifecycleTasks;
    }
    // 商品入库管理方法
    public void manageProductInbound()
    {
        // 调用系统内方法进行商品入库操作
        //
    }
    // 库存管理方法
    public void manageInventory()
    {
        // 调用系统内方法进行库存管理操作
        //
    }
    // 配送管理方法
    public void manageDistribution()
    {
        // 调用系统内方法进行配送管理操作
        //
    }
    // 销售订单处理方法
    public void processSalesOrders()
    {
        // 调用系统内方法进行销售订单处理
        //
    }
    // 客户关系维护方法
    public void maintainCustomerRelationships()
    {
        // 调用系统内方法进行客户关系维护
        //
    }
    // 整合客户反馈
    public void integrateCustomerFeedback()
    {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行客户反馈整合
        //
    }
    // 分析销售数据
    public void analyzeSalesData()
    {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行销售数据分析
        //
    }
    // 优化系统设置
    public void optimizeSystemSettings()
    {
        // 调用系统内方法进行系统设置优化
        //
```

```
        }
        // 提供帮助文档
        public void provideDocumentation()
        {
            // 调用系统内方法提供帮助文档
            //
        }
        // 确保安全退出
        public void ensureSecureLogout()
        {
            // 调用系统内方法确保安全退出
            //
        }
    }
    public static void verifyProductInventory(double expectedStock, double actualStock) {
        // 验证商品库存数量是否正确
        if (Math.abs(expectedStock - actualStock) > acceptableDelta) {
            throw new AssertionError("库存数量不符，预期: + expectedStock +  实际: + actualStock);
        }
    }
    public static void manageProductEntry(double quantity) {
        // 管理商品入库
        // 调用系统内方法进行库存更新操作
        updateInventoryAfterEntry(quantity);
    }
    public static void trackProductDistribution(double quantity) {
        // 跟踪商品配送
        // 调用系统内方法进行配送记录
        recordDistribution(quantity);
    }
    public static void processSalesOrder(double quantity) {
        // 处理销售订单
        // 调用系统内方法进行订单处理
        handleOrder(quantity);
    }
    public static void maintainCustomerRelationships(double customerSatisfaction) {
        // 维护客户关系
        // 调用系统内方法进行客户满意度分析
        analyzeCustomerSatisfaction(customerSatisfaction);
    }
    public static void provideDocumentation() {
        // 提供帮助文档
        // 调用系统内方法进行文档生成
        generateDocumentation();
    }
    public static void ensureSecureExit() {
        // 确保安全退出
        // 调用系统内方法进行资源释放和状态保存
        performSecureExit();
    }
```

```
public class ProductLifecycleController extends Handler {
    private static final String DATABASE_CONNECTION = "jdbc:mysql://localhost:3306/product_management";
    private final Logger log = LoggerFactory.getLogger(ProductLifecycleController.class);
    // 处理商品入库请求
    @RequestMapping(value = "product/inventory/add", name = "addProduct", type = "dm", subtype = "inventory")
    public ResponseEntity<?> addProductToInventory(@RequestBody Product product) {
        // 调用系统内方法进行商品入库操作
        inventoryService.addProduct(product);
        return ResponseEntity.ok("Product added to inventory successfully.");
    }
    // 处理库存管理请求
    @RequestMapping(value = "product/inventory", name = "manageInventory", type = "dm", subtype = "inventory")
    public ResponseEntity<?> manageInventory() {
        // 调用系统内方法进行库存管理操作
        inventoryService.manageInventory();
        return ResponseEntity.ok("Inventory managed successfully.");
    }
    // 处理商品配送请求
    @RequestMapping(value = "product/delivery", name = "handleDelivery", type = "dm", subtype = "distribution")
    public ResponseEntity<?> handleProductDelivery(@RequestBody DeliveryOrder deliveryOrder) {
        // 调用系统内方法进行商品配送操作
        distributionService.handleDelivery(deliveryOrder);
        return ResponseEntity.ok("Product delivery processed successfully.");
    }
    // 处理销售订单请求
    @RequestMapping(value = "order/sales", name = "processSalesOrder", type = "dm", subtype = "sales")
    public ResponseEntity<?> processSalesOrder(@RequestBody SalesOrder salesOrder) {
        // 调用系统内方法进行销售订单处理
        salesService.processOrder(salesOrder);
        return ResponseEntity.ok("Sales order processed successfully.");
    }
    // 处理客户关系维护请求
    @RequestMapping(value = "customer/maintenance", name = "maintainCustomerRelationship", type = "dm", subtype = "customer")
    public ResponseEntity<?> maintainCustomerRelationship(@RequestBody Customer customer) {
        // 调用系统内方法进行客户关系维护操作
        customerService.maintainCustomerRelationship(customer);
        return ResponseEntity.ok("Customer relationship maintained successfully.");
    }
    // 整合客户反馈
    @RequestMapping(value = "feedback/integrate", name = "integrateFeedback", type = "dm", subtype = "feedback")
    public ResponseEntity<?> integrateCustomerFeedback(@RequestBody Feedback feedback) {
        // 调用系统内方法进行客户反馈整合操作
        feedbackService.integrateFeedback(feedback);
        return ResponseEntity.ok("Customer feedback integrated successfully.");
    }
    // 优化系统设置
    @RequestMapping(value = "system/optimization", name = "optimizeSystemSettings", type = "dm", subtype = "settings")
    public ResponseEntity<?> optimizeSystemSettings(@RequestBody Settings settings) {
        // 调用系统内方法进行系统设置优化操作
        settingsService.optimizeSettings(settings);
```

```
            return ResponseEntity.ok("System settings optimized successfully.");
        }
        // 提供帮助文档
        @RequestMapping(value = "help/documentation", name = "provideDocumentation", type = "dm", subtype = "docs")
        public ResponseEntity<?> provideDocumentation() {
            // 调用系统内方法提供帮助文档
            documentationService.provideDocumentation();
            return ResponseEntity.ok("Documentation provided successfully.");
        }
        // 确保安全退出
        @RequestMapping(value = "exit/safe", name = "ensureSafeExit", type = "dm", subtype = "exit")
        public ResponseEntity<?> ensureSafeExit() {
            // 调用系统内方法确保安全退出
            systemService.ensureSafeExit();
            return ResponseEntity.ok("System exited safely.");
        }
    }
    /**
     * 商品库存除法运算，用于库存调整和计算。
     * @param v1  库存数量 1
     * @param v2  库存数量 2
     * @param scale  保留小数位数
     * @param roundingMode  四舍五入模式
     * @return  计算后的库存数量
     */
    public static double calculateInventoryDivision(double v1, double v2, int scale, RoundingMode roundingMode) {
        return div(Double.toString(v1), Double.toString(v2), scale, roundingMode).doubleValue;
    }
    /**
     * 初始化商品信息节点，用于商品全周期管理软件中商品入库流程
     * @return  商品信息节点的 JSON 字符串
     * @throws Exception  初始化过程中出现的异常
     */
    public String ProductNode_Init() throws Exception {
        BP.WF.Node nd = new BP.WF.Node();
        nd.setNodeID(this.getFK_Node);
        nd.RetrieveFromDBSources();
        return nd.ToJson();
    }
    /**
     * 管理商品入库信息，调用系统内复杂的 SQL 查询、数据分析库方法进行操作
     */
    public void ManageProductEntry() {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        //
    }
    /**
     * 商品库存管理，实现库存的增删改查
     */
    public void ManageProductInventory() {
```

```
        // 实现库存管理逻辑
        //
    }
    /**
     * 商品配送管理，包括配送计划、执行和跟踪
     */
    public void ManageProductDistribution() {
        // 实现配送管理逻辑
        //
    }
    /**
     * 销售订单处理，包括订单接收、审核和发货
     */
    public void ProcessSalesOrder() {
        // 实现销售订单处理逻辑
        //
    }
    /**
     * 客户关系维护，包括客户信息管理、客户反馈收集
     */
    public void MaintainCustomerRelationship() {
        // 实现客户关系维护逻辑
        //
    }
    /**
     * 集成客户反馈和销售数据，优化系统设置
     */
    public void IntegrateFeedbackAndSalesData() {
        // 实现数据集成和系统设置优化逻辑
        //
    }
    /**
     * 提供帮助文档，帮助用户更好地使用商品全周期管理软件
     */
    public void ProvideHelpDocumentation() {
        // 提供帮助文档逻辑
        //
    }
    /**
     * 确保系统安全退出，防止数据丢失或损坏
     */
    public void EnsureSafeExit() {
        // 实现安全退出逻辑
        //
    }
    public class ProductLifecycleManagement {
        // 商品入库操作标志
        public static final int INVENTORY_IN = 1;
        // 库存管理操作标志
        public static final int INVENTORY_MANAGEMENT = 2;
```

```
    // 配送操作标志
    public static final int DELIVERY = 3;
    // 销售订单处理操作标志
    public static final int SALE_ORDER_PROCESSING = 4;
    // 客户关系维护操作标志
    public static final int CUSTOMER_RELATIONSHIP_MAINTENANCE = 5;
    // 客户反馈处理操作标志
    public static final int CUSTOMER_FEEDBACK = 6;
    // 销售数据分析操作标志
    public static final int SALE_DATA_ANALYSIS = 7;
    // 系统设置优化操作标志
    public static final int SYSTEM_SETTING_OPTIMIZATION = 8;
    // 帮助文档访问操作标志
    public static final int HELP_DOCUMENT_ACCESS = 9;
    // 安全退出操作标志
    public static final int SAFE_EXIT = 10;
    // 标准化作业流程方法
    public static void standardizeOperation() {
        // 调用系统内方法进行操作
    }
    // 提升管理效率方法
    public static void enhanceManagementEfficiency() {
        // 调用系统内方法进行操作
    }
    // 增强竞争力方法
    public static void enhanceCompetition() {
        // 调用系统内方法进行操作
    }
    // 智慧决策支持方法
    public static void supportWisdomDecisionMaking() {
        // 调用系统内方法进行操作
    }
}
/**
 * 获取商品库存数量
 * 该方法用于获取当前商品在库存中的数量
 */
public abstract int getInventoryCount();
/**
 * 更新商品入库记录
 * 该方法用于记录商品入库信息，包括商品 ID、数量、入库时间等
 */
public void updateInventoryEntry(int productId, int quantity, Date entryDate);
/**
 * 处理销售订单
 * 该方法用于处理客户销售订单，包括更新库存数量、记录销售数据等
 */
public void processSalesOrder(int orderId, int productId, int quantitySold);
/**
 * 维护客户关系
```

```
     * 该方法用于记录和管理客户反馈信息，以便于后续分析和改进
     */
    public void recordCustomerFeedback(int customerId, String feedback);
    /**
     * 提供系统帮助文档
     * 该方法用于生成并返回系统帮助文档，帮助用户了解和使用软件功能
     */
    public String getHelpDocumentation();
    /**
     * 初始化商品入库信息界面
     * @return 返回商品入库信息的 JSON 字符串
     * @throws Exception 如果初始化过程中发生异常
     */
    public String InitProductInboundInfo() throws Exception {
        DataTable inboundInfo = GetProductInboundFields();
        inboundInfo.TableName = "inboundInfo";
        return Json.ToJson(inboundInfo);
    }
    /**
     * 获取商品入库字段信息
     * @return 返回商品入库字段信息的 DataTable 对象
     */
    private DataTable GetProductInboundFields() {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        return MapData.GetFieldsOfPTableMode2(getFK_ProductInbound);
    }
    /**
     * 发送商品全周期管理相关的电子邮件
     * @param mailAddress 收件人邮箱地址
     * @param emilTitle 邮件标题
     * @param emailBody 邮件内容
     * @param msgType 消息类型
     * @param msgGroupFlag 消息分组标志
     * @param sender 发件人
     * @param msgPK 消息主键
     * @param sendToEmpNo 发送给的员工编号
     */
    public static void sendProductLifecycleEmail(String mailAddress, String emilTitle, String emailBody, String msgType,
                                                  String msgGroupFlag, String sender, String msgPK, String sendToEmpNo) {
        // 调用系统内方法进行操作
        Port_SendEmail(mailAddress, emilTitle, emailBody, msgType, msgGroupFlag, sender, msgPK, sendToEmpNo);
    }
    // 商品活动数量统计
    public int getNumActivities() {
        // 调用系统内商品活动管理方法获取活动数量
        int activitiesCount = system.getActivitiesCount();
        // 更新活动数量统计信息
        numActivities = activitiesCount;
        return numActivities;
    }
```

```
/**
 * 获取商品全周期管理中异常信息的方法
 * 该方法用于在商品全周期管理过程当发生异常时，获取并返回异常的具体信息
 */
public String getExceptionMessage() {
    try {
        // 调用系统内异常处理方法进行操作
        String message = thrownException().getMessage();
        return message;
    } catch (Exception e) {
        // 调用系统内异常处理方法进行操作
        return "系统错误：" + e.getMessage();
    }
}
/**
 * 获取抛出的异常对象
 * 该方法用于获取当前操作中抛出的异常对象
 */
private Exception thrownException() {
    // 调用系统内异常捕获方法进行操作
    return new Exception("未知的错误");
}
/**
 * 确保系统在异常退出时的安全操作
 * 该方法确保在发生异常时，系统能够安全退出，并保留必要的数据和状态
 */
public void ensureSafeExit() {
    // 调用系统内安全退出方法进行操作
    //
}
public Result manageProductLifeCycle(ProductRequest request, InventoryManagementSystem system)
{
    // 添加监听器以记录操作历史
    system.getHistoryListener().addListener(this.fHistory.listener());
    // 运行商品生命周期流程
    return system.runProductLifecycle(sortRequest(request).getRunner());
}
// 对请求进行排序处理，适配商品全周期管理需求
private SortedProductRequest sortRequest(ProductRequest request)
{
    // 调用系统内复杂的排序方法进行操作
    return system.sortProductRequest(request);
}
// 管理库存
private void manageInventory(Product product)
{
    // 调用系统内库存管理方法进行操作
    system.manageInventory(product);
}
// 处理销售订单
```

```
private void processSalesOrder(SalesOrder order)
{
    // 调用系统内销售订单处理方法进行操作
    system.processSalesOrder(order);
}
// 维护客户关系
private void maintainCustomerRelationship(Customer customer)
{
    // 调用系统内客户关系维护方法进行操作
    system.maintainCustomerRelationship(customer);
}
// 整合销售数据
private void integrateSalesData()
{
    // 调用系统内数据分析库方法进行操作
    system.integrateSalesData();
}
// 提供帮助文档
private void provideDocumentation()
{
    // 调用系统内帮助文档方法进行操作
    system.provideDocumentation();
}
// 确保安全退出系统
private void ensureSecureLogout()
{
    // 调用系统内安全退出方法进行操作
    system.ensureSecureLogout();
}
public boolean manageProductLifecycle() {
    // 验证商品是否为合成商品，并处理相关逻辑
    if (isSynthetic()) {
        throw new RuntimeException("商品管理异常：商品为合成商品，需要特殊处理。");
    }
    // 商品入库操作
    handleProductEntry();
    // 库存管理
    manageInventory();
    // 配送流程
    dispatchProducts();
    // 销售订单处理
    processSalesOrders();
    // 客户关系维护
    maintainCustomerRelationships();
    // 整合客户反馈
    integrateCustomerFeedback();
    // 分析销售数据
    analyzeSalesData();
    // 优化系统设置
    optimizeSystemSettings();
```

```
        // 提供帮助文档
        provideDocumentation();
        // 确保安全退出
        ensureSecureLogout();
        return true;
    }
    // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
    private void handleProductEntry() {
        // 调用系统内方法进行商品入库操作
    }
    // 调用系统内方法进行库存管理
    private void manageInventory() {
        // 调用系统内方法进行库存管理
    }
    // 调用系统内方法进行配送流程
    private void dispatchProducts() {
        // 调用系统内方法进行配送流程
    }
    // 调用系统内方法进行销售订单处理
    private void processSalesOrders() {
        // 调用系统内方法进行销售订单处理
    }
    // 调用系统内方法进行客户关系维护
    private void maintainCustomerRelationships() {
        // 调用系统内方法进行客户关系维护
    }
    // 调用系统内方法进行客户反馈整合
    private void integrateCustomerFeedback() {
        // 调用系统内方法进行客户反馈整合
    }
    // 调用系统内方法进行销售数据分析
    private void analyzeSalesData() {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
    }
    // 调用系统内方法进行系统设置优化
    private void optimizeSystemSettings() {
        // 调用系统内方法进行系统设置优化
    }
    // 提供帮助文档
    private void provideDocumentation() {
        // 提供帮助文档
    }
    // 确保安全退出
    private void ensureSecureLogout() {
        // 确保安全退出
    }
    public void updateProductInfo(int productId, ProductInfo paramProductInfo) throws SQLException {
        // 更新商品信息的时间戳
        DatabaseError.throwSqlException(76, "updateProductInfoTimestamp");
        // 初始化返回访问器，准备更新操作
```

```
        if (this.returnAccessors != null) return;
        this.returnAccessors = paramOracleReturnResultSet.returnAccessors;
        switch (this.autoKeyType) {
            case 0:
                initProductMetaDataKeyFlag();
                break;
            case 1:
            case 2:
                initProductMetaDataColumnIndexes();
                break;
        }
    }
    void initProductMetaDataColumnIndexes() throws SQLException {
        for (int j = 0; j < this.returnAccessors.length; j++) {
            Accessor localAccessor = this.returnAccessors[j];
            int i = this.columnIndexes[j] - 1;
            localAccessor.columnName = this.tableColumnNames[i];
            localAccessor.describeType = this.tableColumnTypes[i];
            localAccessor.describeMaxLength = this.tableMaxLengths[i];
            localAccessor.nullable = this.tableNullables[i];
            localAccessor.precision = this.tablePrecisions[i];
            localAccessor.scale = this.tablePrecisions[i];
            localAccessor.formOfUse = this.tableFormOfUses[i];
        }
    }
    void initProductMetaDataKeyFlag() throws SQLException {
        this.returnAccessors[0].columnName = "PRODUCT_ID";
        this.returnAccessors[0].describeType = 104;
        this.returnAccessors[0].describeMaxLength = 4;
        this.returnAccessors[0].nullable = true;
        this.returnAccessors[0].precision = 0;
        this.returnAccessors[0].scale = 0;
        this.returnAccessors[0].formOfUse = 0;
    }
    static final boolean isInsertProductSqlStmt(String paramString) throws SQLException {
        if (paramString == null) {
            DatabaseError.throwSqlException(68);
        }
        return paramString.trim().toUpperCase().startsWith("INSERT");
    }
    public static class ProductLifecycleManagementTask extends FJTask
    {
        // 商品全周期管理任务
        protected final Runnable runnable;
        // 商品入库处理
        public void handleProductInbound() {
            // 调用系统内方法进行操作
            // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        }
        // 库存管理操作
```

```
    public void manageInventory() {
        // 实现库存管理逻辑
    }
    // 配送流程控制
    public void controlDistribution() {
        // 配送流程控制逻辑
    }
    // 销售订单处理
    public void processSalesOrders() {
        // 销售订单处理逻辑
    }
    // 客户关系维护
    public void maintainCustomerRelationships() {
        // 客户关系维护逻辑
    }
    // 整合客户反馈
    public void integrateCustomerFeedback() {
        // 客户反馈整合逻辑
    }
    // 分析销售数据
    public void analyzeSalesData() {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
    }
    // 优化系统设置
    public void optimizeSystemSettings() {
        // 系统设置优化逻辑
    }
    // 提供帮助文档
    public void provideHelpDocumentation() {
        // 帮助文档提供逻辑
    }
    // 确保安全退出
    public void ensureSecureLogout() {
        // 安全退出逻辑
    }
}
/**
 * 获取商品全周期管理中所有节点的列表
 * @return 返回商品全周期管理中的节点列表
 */
public List<ProductNode> getAllProductNodes() {
    List<ProductInfoNode> infoNodes = getProductInfoNodes();
    int len = infoNodes.size();
    List<ProductNode> nodes = new ArrayList<ProductNode>(len);
    if (len > 0) {
        for (ProductInfoNode infoNode : infoNodes) {
            nodes.add(infoNode.getProductNode());
        }
    }
    return nodes;
```

```
}
/**
 * 获取商品信息节点，用于商品全周期管理
 * @return 返回商品信息节点列表
 */
private List<ProductInfoNode> getProductInfoNodes() {
    // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
    return database.queryProductInfoNodes();
}
/**
 * 商品信息节点类，包含商品节点信息
 */lass ProductInfoNode {
    private ProductNode productNode;
    public ProductNode getProductNode() {
        return productNode;
    }
}
/**
 * 商品节点类，表示商品在管理过程中的一个节点
 */lass ProductNode {
    // 节点相关的属性和方法
}
/**
 * 设置商品描述信息，该信息将在商品全周期管理中用于展示和记录。
 * @param prDescription 商品描述信息对象
 */
public void updateProductDescription(ProductDescription prDescription) {
    // 保存商品描述信息，以备后续的商品入库、库存管理、销售订单处理等功能使用
    this.productDescription = prDescription;
}
/**
 * 商品描述信息类，包含商品描述的相关属性。
 */lass ProductDescription {
    private String description; // 商品描述文本
    // 其他相关属性和方法...
    // 构造函数、getters 和 setters...
}
// 商品库存检查方法
public final boolean isProductInStock(Product product) {
    return this.contains(product.getId());
}
// 调用系统内复杂的 SQL 查询方法进行操作
private boolean contains(Long productId) {
    // 检查商品是否存在于库存中
    return inventoryRepository.existsById(productId);
}
/**
 * 解析商品状态字符串
 * 将字符串转换为商品状态（启用/禁用），适用于商品库存状态等场景。
 *
```

```
     * @param s  商品状态字符串
     * @return  商品状态，true 表示启用，false 表示禁用
     */
    public static boolean parseProductStatus(String s) {
        if (null == s || s.length() == 0)
            return false;
        if (s.length() > 5)
            return true;
        if ("0".equals(s))
            return false;
        s = s.toLowerCase();
        return !"false".equals(s) && !"off".equals(s) && !"no".equals(s);
    }
    public void manageProductInventory(View v) {
        // 检查商品入库信息
        checkProductEntry();
        // 管理库存状态
        updateInventoryStatus();
        // 配送商品至指定地点
        dispatchProduct();
        // 处理销售订单
        processSalesOrder();
        // 维护客户关系信息
        updateCustomerRelationship();
        // 整合客户反馈
        integrateCustomerFeedback();
        // 分析销售数据
        analyzeSalesData();
        // 优化系统设置
        optimizeSystemSettings();
        // 提供帮助文档访问
        provideHelpDocumentation();
        // 确保安全退出系统
        ensureSafeLogout();
    }
    private void checkProductEntry() {
        // 调用系统内商品入库验证方法进行操作
    }
    private void updateInventoryStatus() {
        // 更新库存状态的逻辑
    }
    private void dispatchProduct() {
        // 配送商品的逻辑
    }
    private void processSalesOrder() {
        // 处理销售订单的逻辑
    }
    private void updateCustomerRelationship() {
        // 更新客户关系的逻辑
    }
```

```
private void integrateCustomerFeedback() {
    // 整合客户反馈的逻辑
}
private void analyzeSalesData() {
    // 分析销售数据的逻辑
}
private void optimizeSystemSettings() {
    // 优化系统设置的逻辑
}
private void provideHelpDocumentation() {
    // 提供帮助文档的逻辑
}
private void ensureSafeLogout() {
    // 确保安全退出的逻辑
}
/**
 * 计算指定商品键的库存总量
 * @param key 商品键
 * @return 商品库存总量
 */
public final double CalculateInventorySumByKey(String key)
{
    double sum = 0d;
    for (Product product : inventoryManager.getProducts())
    {
        sum += product.getValDoubleByKey(key);
    }
    return sum;
}
/**
 * 调用系统内库存管理方法获取商品列表
 */
private List<Product> getProducts()
{
    // 调用系统内库存管理方法进行操作
}
/**
 * 获取商品键对应的值
 * @param key 商品键
 * @return 商品值
 */
public double getValDoubleByKey(String key)
{
    // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
}
public final String manageProductLifecycle(String doType, ProductNode currNode, ProductEntity en, String atPara, SendReturnObjs objs)
        throws Exception {
    // 调用系统内商品全周期管理流程方法进行操作
    return manageProductLifecycle(doType, currNode, en, atPara, objs, null, null);
}
```

```
// ProductNode: 表示当前在商品全周期管理中的节点，如入库、库存、配送等
// ProductEntity: 表示商品的基本信息实体，包括商品 ID、名称、库存数量等
// atPara: 用于传递额外的参数，如订单信息、客户反馈等
// SendReturnObjs: 表示发送返回对象，用于处理流程中的结果和异常
// 商品入库模块
public class ProductInboundManager {
    // 商品入库方法
    public void inboundProduct(Product product) {
        // 调用系统内商品入库逻辑进行操作
        //
    }
}
// 库存管理模块
public class InventoryManager {
    // 库存更新方法
    public void updateInventory(Product product, int quantity) {
        // 调用系统内库存更新逻辑进行操作
        //
    }
}
// 配送管理模块
public class DistributionManager {
    // 配送商品方法
    public void distributeProduct(Product product) {
        // 调用系统内配送逻辑进行操作
        //
    }
}
// 销售订单处理模块
public class SalesOrderProcessor {
    // 处理销售订单方法
    public void processSalesOrder(SalesOrder order) {
        // 调用系统内销售订单处理逻辑进行操作
        //
    }
}
// 客户关系维护模块
public class CustomerRelationshipManager {
    // 更新客户信息方法
    public void updateCustomerInfo(Customer customer) {
        // 调用系统内客户信息更新逻辑进行操作
        //
    }
}
// 数据分析模块
public class DataAnalysis {
    // 分析销售数据方法
    public void analyzeSalesData() {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        //
```

```
    }
}
// 帮助文档模块
public class HelpDocumentation {
    // 提供帮助文档方法
    public void provideDocumentation(String topic) {
        // 显示或生成相关帮助文档
        //
    }
}
// 安全退出模块
public class SecureLogout {
    // 安全退出系统方法
    public void logoutSystem() {
        // 确保所有操作都已保存，并安全退出系统
        //
    }
}
/**
 * 设置商品计划编码，该编码用于标识商品的计划信息。
 * @param prPlanercode 商品的计划编码对象
 */
public void setProductPlanerCode(Object prPlanercode) {
    this.productPlanerCode = prPlanercode;
}
/**
 * 商品入库操作，将商品信息登记到系统。
 */
public void manageProductInbound() {
    // 调用系统内方法进行操作。
}
/**
 * 库存管理，更新库存信息。
 * @param quantity 更新后的库存数量
 */
public void manageInventory(int quantity) {
    // 更新库存数量
}
/**
 * 配送管理，安排商品配送。
 * @param deliveryDetails 配送详细信息
 */
public void manageDelivery(DeliveryDetails deliveryDetails) {
    // 安排商品配送
}
/**
 * 销售订单处理，处理客户的订单请求。
 * @param orderDetails 订单详细信息
 */
public void processSalesOrder(OrderDetails orderDetails) {
```

```
        // 处理销售订单
    }
    /**
     * 客户关系维护，更新客户信息。
     * @param customerInfo 客户信息对象
     */
    public void maintainCustomerRelationship(CustomerInfo customerInfo) {
        // 更新客户信息
    }
    /**
     * 客户反馈处理，收集和分析客户反馈。
     * @param feedback 客户反馈信息
     */
    public void handleCustomerFeedback(Feedback feedback) {
        // 处理客户反馈
    }
    /**
     * 数据分析，整合销售数据进行分析。
     * @param salesData 销售数据
     */
    public void analyzeSalesData(SalesData salesData) {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
    }
    /**
     * 系统设置优化，根据反馈调整系统设置。
     */
    public void optimizeSystemSettings() {
        // 优化系统设置
    }
    /**
     * 提供帮助文档，帮助用户了解和使用系统。
     */
    public void provideHelpDocumentation() {
        // 提供帮助文档
    }
    /**
     * 确保安全退出系统，进行必要的清理和验证操作。
     */
    public void ensureSafeLogout() {
        // 确保安全退出
    }
    public boolean isProductInCycle(Class<?> type) {
        // 检查产品类型是否与当前类型相同当前类型是否可以转换为指定类型
        if (productType == type || type.isAssignableFrom(productType))
            return true;
        // 检查是否为基本数据类型及其包装类型
        if (productType.isPrimitive() && type.isPrimitive()) {
            if (productType.isPrimitiveNumber() && Mirror.me(type).isPrimitiveNumber)
                return true;
        }
```

```
        try {
            // 尝试获取包装类是否匹配
            return Mirror.me(type).getWrapperClass == this.getWrapperClass;
        } catch (Exception e) {
            // 捕获异常
        }
        return false;
    }
    // 更新商品信息方法
    public static String updateProductInfo(String GName, String GAddress, int GMId, float GInPrice, float GOutPrice, int GId) {
        Connection conn = GetConnection.getConn();
        String sql = "UPDATE Products SET ProductName=?, Address=?, ManufacturerId=?, PurchasePrice=?, SalePrice=? WHERE ProductId=?";
        PreparedStatement st = null;
        int set = 0;
        try {
            st = conn.prepareStatement(sql);
            st.setString(1, GName);
            st.setString(2, GAddress);
            st.setInt(3, GMId);
            st.setFloat(4, GInPrice);
            st.setFloat(5, GOutPrice);
            st.setInt(6, GId);
            set = st.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                if (st != null) {
                    st.close();
                }
                if (conn != null) {
                    conn.close();
                }
            } catch (SQLException ex) {
                ex.printStackTrace();
            }
        }
        if (set <= 0) {
            return "Update failed";
        } else {
            return "Update successful";
        }
    }
    // 连接获取方法
    package com.erp.warehouse.management;
    import java.sql.Connection;
    import java.sql.DriverManager;
    import java.sql.SQLException;
    public class GetConnection {
```

```
        public static Connection getConn() {
            // 调用系统内复杂的 SQL 连接、数据库配置方法进行操作
            String url = "jdbc:mysql://localhost:3306/warehouse_db";
            String user = "root";
            String password = "password";
            Connection conn = null;
            try {
                conn = DriverManager.getConnection(url, user, password);
            } catch (SQLException e) {
                e.printStackTrace();
            }
            return conn;
        }
    }
    public ProductLifecycle(String productID) throws Exception {
        this.setProductID(productID);
        if (SystemConfig.getIsDebug) {
            int i = this.RetrieveProductInfo;
            if (i == 0) {
                throw new RuntimeException("商品信息未找到");
            }
        } else {
            this.RetrieveProductInfo;
        }
    }
    @Override
    protected boolean beforeProductUpdateInsertAction throws Exception {
        this.setProductEventEntity(BP.WF.Glo.GetProductEventEntityStringByProductMark(this.getProductMark, this.getProductID));
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        DBAccess.RunSQL("UPDATE Product_Info SET ProductName=+ this.getProductName + WHERE ProductID=+ this.getProductID +
    "");
        DBAccess.RunSQL("UPDATE Product_Stock SET StockName=+ this.getProductName + WHERE ProductID=+ this.getProductID + "");
        return super.beforeProductUpdateInsertAction;
    }
    /**
     * 获取商品的库存状态，商品全周期管理中的库存查询功能
     * @param month 商品入库月份
     * @param day 商品入库日
     * @return 返回商品的库存状态
     */
    public static String getInventoryStatus(int month, int day) {
        return InventoryStatus.getInventoryStatus(month, day);
    }
    public ProductLifecycleDocumentPath getParent()
    {
        int i = this.components.length - 1;
        if (i < 0)
        {
            return null;
        }
```

```
        ProductLifecycleDocumentPath localProductLifecycleDocumentPath = new ProductLifecycleDocumentPath(null);
        localProductLifecycleDocumentPath.components = new String[i];
        System.arraycopy(this.components, 0, localProductLifecycleDocumentPath.components, 0, i);
        return localProductLifecycleDocumentPath;
    }
    // 获取商品全周期管理软件中的商品信息数据库界面
    public final FrmProductCycleDBs getFrmProductCycleDBs() throws Exception
    {
        Object tempVar = this.GetRefObject("FrmProductCycleDBs");
        FrmProductCycleDBs obj = (FrmProductCycleDBs)((tempVar instanceof FrmProductCycleDBs) ? tempVar : null);
        if (obj == null)
        {
            obj = new FrmProductCycleDBs(this.getNo);
            this.SetRefObject("FrmProductCycleDBs", obj);
        }
        return obj;
    }
    // 商品入库操作
    public void handleProductEntry() throws Exception
    {
        FrmProductCycleDBs db = getFrmProductCycleDBs();
        // 调用系统内复杂的 SQL 查询方法进行商品入库操作
        db.performEntry();
    }
    // 库存管理操作
    public void handleInventoryManagement() throws Exception
    {
        FrmProductCycleDBs db = getFrmProductCycleDBs();
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行库存管理操作
        db.manageInventory();
    }
    // 配送操作
    public void handleDistribution() throws Exception
    {
        FrmProductCycleDBs db = getFrmProductCycleDBs();
        // 调用系统内复杂的物流管理方法进行配送操作
        db.performDistribution();
    }
    // 销售订单处理操作
    public void handleSalesOrder() throws Exception
    {
        FrmProductCycleDBs db = getFrmProductCycleDBs();
        // 调用系统内复杂的订单处理方法进行销售订单处理
        db.processSalesOrder();
    }
    // 客户关系维护操作
    public void handleCustomerRelationship() throws Exception
    {
        FrmProductCycleDBs db = getFrmProductCycleDBs();
        // 调用系统内客户关系管理方法进行客户关系维护
```

```
        db.maintainCustomerRelationship();
    }
    // 整合客户反馈
    public void integrateCustomerFeedback() throws Exception
    {
        FrmProductCycleDBs db = getFrmProductCycleDBs();
        // 调用系统内数据分析库方法进行客户反馈整合
        db.integrateFeedback();
    }
    // 优化系统设置
    public void optimizeSystemSettings() throws Exception
    {
        FrmProductCycleDBs db = getFrmProductCycleDBs();
        // 调用系统内配置优化方法进行系统设置优化
        db.optimizeSettings();
    }
    // 提供帮助文档
    public void provideHelpDocumentation() throws Exception
    {
        FrmProductCycleDBs db = getFrmProductCycleDBs();
        // 调用系统内文档管理方法提供帮助文档
        db.provideDocumentation();
    }
    // 确保安全退出
    public void ensureSecureLogout() throws Exception
    {
        FrmProductCycleDBs db = getFrmProductCycleDBs();
        // 调用系统内安全退出方法确保安全退出
        db.ensureSecureLogout();
    }
    public static void manageProductLifecycle(long productId) {
        // 检查是否启用任务池
        if (!Glo.getIsEnableTaskPool) {
            throw new RuntimeException("任务池未启用");
        }
        // 获取商品信息
        GenerWorkFlow productInfo = getProductInfo(productId);
        if (productInfo.getTaskSta() == TaskSta.None) {
            throw new RuntimeException("商品信息未设置");
        }
        if (productInfo.getTaskSta() == TaskSta.Sharing) {
            throw new RuntimeException("商品信息正在共享中");
        }
        // 更新商品入库状态
        updateProductStatus(productId, "InStock");
        // 获取待处理任务列表
        DataTable toDoTasks = getToDosForProduct(productId);
        if (toDoTasks == null || toDoTasks.Rows.size == 0) {
            throw new RuntimeException("无待处理任务");
        }
```

```
        // 处理待处理任务
        String toDoEmps = getToDoEmps(toDoTasks);
        updateTodoEmps(productId, toDoEmps, toDoTasks.Rows.size);
        // 记录跟踪信息
        logTrackInfo(productInfo.getFK_Flow(), productInfo.getFK_Node(), productInfo.getNodeName(), productId);
    }
    private static GenerWorkFlow getProductInfo(long productId) {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        return new GenerWorkFlow(productId);
    }
    private static void updateProductStatus(long productId, String status) {
        // 更新商品状态，如入库、出库等
        // 调用系统内**方法进行操作
    }
    private static DataTable getToDosForProduct(long productId) {
        // 获取商品相关的待处理任务
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        return new DataTable();
    }
    private static String getToDoEmps(DataTable toDoTasks) {
        // 构建待办任务人员信息
        String toDoEmps = "";
        for (DataRow dr : toDoTasks.Rows) {
            toDoEmps += String.format("%1$s,%2$s", dr.getValue("FK_Emp").toString(), dr.getValue("FK_EmpText").toString()) + ";";
        }
        return toDoEmps;
    }
    private static void updateTodoEmps(long productId, String toDoEmps, int toDoEmpsNum) {
        // 更新待办任务人员信息和数量
        // 调用系统内**方法进行操作
    }
    private static void logTrackInfo(String fkFlow, String fkNode, String nodeName, long workId) {
        // 记录商品生命周期中的跟踪信息
        // 调用系统内**方法进行操作
    }
    public void manageProductLifecycle() {
        // 商品入库处理
        handleProductEntry();
        // 库存管理
        manageInventory();
        // 配送流程
        dispatchProducts();
        // 销售订单处理
        processSalesOrders();
        // 客户关系维护
        maintainCustomerRelationships();
        // 整合客户反馈
        integrateCustomerFeedback();
        // 分析销售数据
        analyzeSalesData();
```

```
        // 优化系统设置
        optimizeSystemSettings();
        // 提供帮助文档
        provideHelpDocumentation();
        // 确保安全退出
        ensureSecureLogout();
    }
    // 商品入库处理
    private void handleProductEntry() {
        // 调用系统内方法进行操作
    }
    // 库存管理
    private void manageInventory() {
        // 调用系统内方法进行操作
    }
    // 配送流程
    private void dispatchProducts() {
        // 调用系统内方法进行操作
    }
    // 销售订单处理
    private void processSalesOrders() {
        // 调用系统内方法进行操作
    }
    // 客户关系维护
    private void maintainCustomerRelationships() {
        // 调用系统内方法进行操作
    }
    // 整合客户反馈
    private void integrateCustomerFeedback() {
        // 调用系统内方法进行操作
    }
    // 分析销售数据
    private void analyzeSalesData() {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
    }
    // 优化系统设置
    private void optimizeSystemSettings() {
        // 调用系统内方法进行操作
    }
    // 提供帮助文档
    private void provideHelpDocumentation() {
        // 调用系统内方法进行操作
    }
    // 确保安全退出
    private void ensureSecureLogout() {
        // 调用系统内方法进行操作
    }
    public ModelAndView productManagement(ModelMap map, HttpServletRequest request, @Valid String q, @Valid String category,
    @Valid String productId) {
        // 获取商品信息
```

```
        ProductInfo productInfo = productRepository.findByQueryAndOrgi(null, super.getOrgi(request), UKTools.getWeekStartTime, null, false);
        map.addAttribute("productList", productRepository.findProductList(q, new PageRequest(super.getP(request), super.getPs(request))));
        // 设置周期性筛选
        FilterCriteria filterCriteria = new FilterCriteria();
        filterCriteria.setWeek(true);
        map.put("filterCriteria", UKTools.encode(UKTools.toJson(filterCriteria)));
        // 处理销售订单
        OrderProcess orderProcess = orderService.processOrder(productId);
        // 维护客户关系
        CustomerRelation customerRelation = customerService.updateCustomerFeedback(q);
        // 确保安全退出
        super.safeLogout(request);
        // 返回商品管理页面
        return request(super.createAppsTempletResponse("productManagementTemplate"));
    }
    @RequestMapping("product")
    @Menu(type = "product", subtype = "management", name = "management")
    /**
     * 获取商品全周期管理中的整体进度
     * @return  整体进度值
     */
    public Integer getFullCycleProgress() {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        //
        return overallProgress;
    }
    /**
     * 更新商品全周期管理中的整体进度
     * @param progress  新的整体进度值
     */
    public void updateFullCycleProgress(Integer progress) {
        // 更新数据库中的整体进度信息
        //
        this.overallProgress = progress;
    }
    /**
     * 管理商品入库信息
     * @param inventoryItem  入库的商品信息
     */
    public void manageInventoryEntry(InventoryItem inventoryItem) {
        // 商品入库逻辑处理
        //
    }
    /**
     * 管理商品库存
     * @param inventory  库存信息
     */
    public void manageInventory(Inventory inventory) {
        // 库存管理逻辑处理
```

```
        //
    }
    /**
     * 管理商品配送
     * @param shipment 配送信息
     */
    public void manageShipment(Shipment shipment) {
        // 配送管理逻辑处理
        //
    }
    /**
     * 处理销售订单
     * @param salesOrder 销售订单信息
     */
    public void processSalesOrder(SalesOrder salesOrder) {
        // 销售订单处理逻辑
        //
    }
    /**
     * 维护客户关系
     * @param customer 客户信息
     */
    public void maintainCustomerRelationship(Customer customer) {
        // 客户关系维护逻辑
        //
    }
    /**
     * 商品描述列名称常量
     * 用于数据库中商品描述字段的标识
     */
    public static final String COLUMN_DESCRIPTION = "product_description";
    /**
     * 商品入库处理 - 记录商品描述信息
     * 调用系统内数据库操作方法进行数据插入
     */
    public static void recordProductDescription(String productId, String description) {
        // 调用系统内数据库操作方法进行数据插入
        // database.insert("products", COLUMN_DESCRIPTION, description, productId);
    }
    /**
     * 库存管理 - 更新商品描述信息
     * 调用系统内数据库操作方法进行数据更新
     */
    public static void updateProductDescription(String productId, String newDescription) {
        // 调用系统内数据库操作方法进行数据更新
        // database.update("products", COLUMN_DESCRIPTION, newDescription, "id", productId);
    }
    /**
     * 销售订单处理 - 检索商品描述信息
     * 调用系统内数据库查询方法获取商品描述
```

```
 */
public static String getProductDescription(String productId) {
    // 调用系统内数据库查询方法获取商品描述
    // return database.query("products", COLUMN_DESCRIPTION, "id", productId);
}
/**
 * 客户关系维护 - 更新客户反馈中的商品描述
 * 调用系统内数据库操作方法更新客户反馈记录
 */
public static void updateFeedbackProductDescription(String feedbackId, String newDescription) {
    // 调用系统内数据库操作方法更新客户反馈记录
    // database.update("feedbacks", COLUMN_DESCRIPTION, newDescription, "id", feedbackId);
}
/**
 * 安全退出前检查 - 确保所有商品描述信息已正确保存
 * 调用系统内数据验证和保存方法
 */
public static boolean checkAndSaveProductDescriptions() {
    // 调用系统内数据验证和保存方法
    // return database.checkAndSaveAllProductDescriptions();
}
public final String fetchProductLifecycleParameters() {
    // 获取商品全周期管理中的批次参数
    String str = this.GetValStringByKey(NodeAttr.ProductLifecycleParas);
    // 替换系统配置中的 SDK 服务主机地址
    str = str.replace("@SDKFromServHost", BP.Sys.SystemConfig.getAppSettings.get("SDKFromServHost").toString());
    return str;
}
/**
 * 获取商品数组与指定商品数组最匹配的索引位置。
 * 该方法用于在商品全周期管理软件帮助找到商品库存中与订单商品最接近的索引。
 *
 * @param inventory 商品库存数组，代表所有商品。
 * @param orderItem 订单中的商品数组，代表需要查找的商品。
 * @param maxDistance 最大允许的匹配误差范围。
 * @param tolerance 实际允许的误差范围。
 * @return 最匹配的商品数组在库存中的索引位置，如果没有找到匹配项，则返回-1。
 */
public static int findClosestInventoryIndex(float[] inventory, float[] orderItem, int maxDistance, float tolerance) {
    // 调用系统内复杂的算法方法进行操作
    return arrayLastIndexOf(inventory, orderItem, maxDistance, tolerance);
}
/**
 * 根据商品流水号和列名构建查询条件，用于商品全周期管理中的库存查询。
 * @param flowNo 商品流水号，代表商品的唯一标识。
 * @param colName 数据库列名，通常与商品信息相关。
 * @return 构建好的查询条件字符串。
 */
public static String buildProductQueryCondition(String flowNo, String colName) {
    flowNo = String.valueOf(Integer.parseInt(flowNo));
```

```
        String len = BP.Sys.SystemConfig.getAppCenterDBLengthStr;
        return (+ colName + LIKE PD+ flowNo + '% AND + len + '(+ colName + ')='
                + ((new String("PD")).length + flowNo.length() + 2) + ') OR (+ colName + = PD+ flowNo
                + "Rpt ) OR (+ colName + LIKE PD+ flowNo + '__Dtl% AND + len + '(+ colName + ')>'
                + ((new String("PD")).length + flowNo.length() + 2 + (new String("Dtl")).length()) + ')";
    }
    // 商品入库管理 - 设置商品类别名称
    public void setProductCategoryName(String productCategoryName) {
        this.productCategoryName = productCategoryName;
    }
    // 库存管理 - 更新库存信息
    public void updateInventory(String productId, int quantity) {
        // 调用系统内库存管理方法进行操作
        inventoryManagement.update(productId, quantity);
    }
    // 配送管理 - 设置配送信息
    public void setDeliveryInfo(String orderId, DeliveryDetails details) {
        this.deliveryInfo = details;
        // 调用系统内配送调度方法进行操作
        deliveryScheduling.schedule(this.deliveryInfo);
    }
    // 销售订单处理 - 记录销售订单
    public void recordSalesOrder(SalesOrder order) {
        // 调用系统内销售订单处理方法进行操作
        salesOrderProcessing.process(order);
    }
    // 客户关系维护 - 更新客户信息
    public void updateCustomerInfo(String customerId, CustomerDetails details) {
        this.customerInfo = details;
        // 调用系统内客户关系维护方法进行操作
        customerRelationshipManagement.update(customerId, this.customerInfo);
    }
    // 系统设置优化 - 保存系统配置
    public void saveSystemSettings(Settings settings) {
        this.systemSettings = settings;
        // 调用系统内设置保存方法进行操作
        systemConfiguration.save(this.systemSettings);
    }
    // 帮助文档提供 - 更新帮助文档
    public void updateHelpDocument(String topic, String content) {
        this.helpDocuments.put(topic, content);
        // 调用系统内帮助文档更新方法进行操作
        helpDocumentation.update(this.helpDocuments);
    }
    // 安全退出系统
    public void logout() {
        // 确保所有操作已完成，并安全退出
        inventoryManagement.commit();
        salesOrderProcessing.commit();
        customerRelationshipManagement.commit();
```

```
        systemConfiguration.commit();
        // 系统安全退出
        System.exit(0);
    }
    /**
     * 商品入库管理 Fragment，用于处理商品入库流程
     */
    public class ProductEntryFragment extends Fragment {
        /**
         * 根据标签查找商品入库 Fragment
         *
         * @param paramString 标签参数，用于定位具体的 Fragment 实例
         * @return 返回对应的商品入库 Fragment 实例
         */
        public abstract ProductEntryFragment findProductEntryFragmentByTag(String paramString);
        /**
         * 商品库存管理 Fragment，用于处理商品库存相关操作
         */
        public class ProductInventoryFragment extends Fragment {
            /**
             * 根据标签查找商品库存 Fragment
             *
             * @param paramString 标签参数，用于定位具体的 Fragment 实例
             * @return 返回对应的商品库存 Fragment 实例
             */
            public abstract ProductInventoryFragment findProductInventoryFragmentByTag(String paramString);
        }
        /**
         * 商品配送管理 Fragment，用于处理商品配送流程
         */
        public class ProductDistributionFragment extends Fragment {
            /**
             * 根据标签查找商品配送 Fragment
             *
             * @param paramString 标签参数，用于定位具体的 Fragment 实例
             * @return 返回对应的商品配送 Fragment 实例
             */
            public abstract ProductDistributionFragment findProductDistributionFragmentByTag(String paramString);
        }
        /**
         * 销售订单处理 Fragment，用于处理销售订单相关操作
         */
        public class SalesOrderFragment extends Fragment {
            /**
             * 根据标签查找销售订单 Fragment
             *
             * @param paramString 标签参数，用于定位具体的 Fragment 实例
             * @return 返回对应的销售订单 Fragment 实例
             */
            public abstract SalesOrderFragment findSalesOrderFragmentByTag(String paramString);
```

```
    }
    /**
     * 客户关系维护 Fragment，用于维护客户关系
     */
    public class CustomerRelationFragment extends Fragment {
        /**
         * 根据标签查找客户关系维护 Fragment
         *
         * @param paramString 标签参数，用于定位具体的 Fragment 实例
         * @return 返回对应的客户关系维护 Fragment 实例
         */
        public abstract CustomerRelationFragment findCustomerRelationFragmentByTag(String paramString);
    }
    /**
     * 客户反馈处理 Fragment，用于处理客户反馈
     */
    public class CustomerFeedbackFragment extends Fragment {
        /**
         * 根据标签查找客户反馈 Fragment
         *
         * @param paramString 标签参数，用于定位具体的 Fragment 实例
         * @return 返回对应的客户反馈 Fragment 实例
         */
        public abstract CustomerFeedbackFragment findCustomerFeedbackFragmentByTag(String paramString);
    }
    /**
     * 销售数据分析 Fragment，用于分析销售数据
     */
    public class SalesDataAnalysisFragment extends Fragment {
        /**
         * 根据标签查找销售数据分析 Fragment
         *
         * @param paramString 标签参数，用于定位具体的 Fragment 实例
         * @return 返回对应的销售数据分析 Fragment 实例
         */
        public abstract SalesDataAnalysisFragment findSalesDataAnalysisFragmentByTag(String paramString);
    }
    /**
     * 系统设置 Fragment，用于优化系统设置
     */
    public class SystemSettingFragment extends Fragment {
        /**
         * 根据标签查找系统设置 Fragment
         *
         * @param paramString 标签参数，用于定位具体的 Fragment 实例
         * @return 返回对应的系统设置 Fragment 实例
         */
        public abstract SystemSettingFragment findSystemSettingFragmentByTag(String paramString);
    }
    /**
```

```
     * 帮助文档 Fragment，提供帮助文档
     */
    public class HelpDocumentFragment extends Fragment {
        /**
         * 根据标签查找帮助文档 Fragment
         *
         * @param paramString  标签参数，用于定位具体的 Fragment 实例
         * @return  返回对应的帮助文档 Fragment 实例
         */
        public abstract HelpDocumentFragment findHelpDocumentFragmentByTag(String paramString);
    }
    /**
     * 确保安全退出系统
     */
    public void ensureSafeExit() {
        // 调用系统内安全退出方法进行操作
    }
}
// 商品入库功能实现
public int checkInProduct(Product product) {
    // 检查商品信息完整性
    if (product == null || product.getProductName() == null || product.getQuantity() <= 0) {
        throw new IllegalArgumentException("商品信息不完整");
    }
    // 调用系统内库存管理模块进行商品入库
    InventoryManager inventoryManager = new InventoryManager();
    inventoryManager.addProduct(product);
    // 返回商品入库后的库存量
    return inventoryManager.getProductQuantity(product);
}
// 库存管理功能实现
public int manageInventory(Product product, int quantity) {
    // 调用系统内库存管理模块调整库存
    InventoryManager inventoryManager = new InventoryManager();
    inventoryManager.adjustInventory(product, quantity);
    // 返回调整后的库存量
    return inventoryManager.getProductQuantity(product);
}
// 配送管理功能实现
public void dispatchProduct(Product product) {
    // 调用系统内配送管理模块进行商品配送
    DispatchManager dispatchManager = new DispatchManager();
    dispatchManager.dispatch(product);
}
// 销售订单处理功能实现
public void processSalesOrder(SalesOrder order) {
    // 检查订单有效性
    if (order == null || order.getOrderItems() == null) {
        throw new IllegalArgumentException("订单信息不完整");
    }
```

```
        // 调用系统内销售管理模块处理订单
        SalesManager salesManager = new SalesManager();
        salesManager.processOrder(order);
    }
    // 客户关系维护功能实现
    public void maintainCustomerRelationship(Customer customer) {
        // 调用系统内客户关系管理模块维护客户关系
        CRMManager crmManager = new CRMManager();
        crmManager.maintainCustomer(customer);
    }
    // 系统设置优化
    public void optimizeSystemSettings() {
        // 调用系统内设置管理模块优化系统设置
        SettingManager settingManager = new SettingManager();
        settingManager.optimizeSettings();
    }
    // 安全退出功能实现
    public void safeExit() {
        // 调用系统内安全模块进行安全退出
        SecurityManager securityManager = new SecurityManager();
        securityManager.safeExit();
    }
    public static Date calculateExpiryDate(Date productionDate, int shelfLifeDays, int maxHours, int maxMinutes) {
        // 根据生产日期和保质期计算商品过期日期
        Date expiryDate = AddDayHoursSpan(productionDate, shelfLifeDays, maxHours, maxMinutes);
        // 检查并确保计算出的过期日期有效
        if (expiryDate != null) {
            // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
            // 查询商品库存，并更新库存过期商品的记录
            updateInventoryForExpiredProduct(expiryDate);
        }
        return expiryDate;
    }
    private static void updateInventoryForExpiredProduct(Date expiryDate) {
        // 更新库存系统中过期商品的记录
        // 此处调用系统内复杂的 SQL 查询、数据分析库方法进行操作
    }
    /**
     * 版本号管理类
     * 该类负责管理商品全周期管理软件的版本信息
     */
    public class ProductLifecycleManagementSoftwareVersion {
        /**
         * 获取当前软件版本号
         * @return  返回软件的版本号
         */
        public static int getVersion() {
            return Ver;
        }
        /**
```

```
     * 主方法，启动软件时调用
     * @param args 启动参数
     */
    public static void main(String[] args) {
        int version = getVersion();
        System.out.println("商品全周期管理软件版本： + version);
        // 商品入库流程启动
        startInventoryInProcess();
        // 库存管理流程启动
        startInventoryManagementProcess();
        // 配送流程启动
        startDistributionProcess();
        // 销售订单处理流程启动
        startSalesOrderProcess();
        // 客户关系维护流程启动
        startCustomerRelationshipProcess();
        // 系统设置优化
        optimizeSystemSettings();
        // 提供帮助文档
        provideHelpDocumentation();
        // 确保安全退出
        ensureSafeExit();
    }
    /**
     * 启动商品入库流程
     */
    private static void startInventoryInProcess() {
        // 调用系统内复杂的入库操作方法进行操作
        performComplexInventoryInOperation();
    }
    /**
     * 启动库存管理流程
     */
    private static void startInventoryManagementProcess() {
        // 调用系统内复杂的库存管理操作方法进行操作
        performComplexInventoryManagementOperation();
    }
    /**
     * 启动配送流程
     */
    private static void startDistributionProcess() {
        // 调用系统内复杂的配送操作方法进行操作
        performComplexDistributionOperation();
    }
    /**
     * 启动销售订单处理流程
     */
    private static void startSalesOrderProcess() {
        // 调用系统内复杂的订单处理操作方法进行操作
        performComplexOrderProcessingOperation();
```

```
    }
    /**
     * 启动客户关系维护流程
     */
    private static void startCustomerRelationshipProcess() {
        // 调用系统内复杂的客户关系维护操作方法进行操作
        performComplexCustomerRelationshipMaintenanceOperation();
    }
    /**
     * 优化系统设置
     */
    private static void optimizeSystemSettings() {
        // 调用系统内优化设置的方法进行操作
        performSystemSettingsOptimization();
    }
    /**
     * 提供帮助文档
     */
    private static void provideHelpDocumentation() {
        // 调用系统内帮助文档生成和提供的方法进行操作
        performHelpDocumentationOperation();
    }
    /**
     * 确保安全退出
     */
    private static void ensureSafeExit() {
        // 调用系统内安全退出操作的方法进行操作
        performSafeExitOperation();
    }
}
// 商品入库时间比较方法
public static int compareProductEntryTime(String entryTime1, String entryTime2) {
    if (entryTime1 == null || entryTime2 == null)
        return 0;
    java.util.Date entryTime1Date = convertEntryTimeToDateTime(entryTime1);
    java.util.Date entryTime2Date = convertEntryTimeToDateTime(entryTime2);
    return compareDates(entryTime1Date, entryTime2Date);
}
// 将字符串时间转换为日期时间对象
private static java.util.Date convertEntryTimeToDateTime(String timeStr) {
    return toDate(timeStr, "00:00:00");
}
// 比较两个日期时间
private static int compareDates(java.util.Date date1, java.util.Date date2) {
    return date1.compareTo(date2);
}
// 商品入库管理 - 更新商品元数据
public ModelAndView updateProductMetadata(HttpServletRequest request, @PathVariable String orgi, @PathVariable String productID,
@Valid ProductMetadata meta) throws Exception {
    ModelAndView view = request(new ResponseData(new
```

```
StringBuffer().append("redirect:").append(orgi).append("/product/edit").append(productID).append(".html?msgcode=S_DM_80010110").
toString()), orgi);
        ProductMetadata existingMeta = (ProductMetadata) super.getService.getIObjectByPK(ProductMetadata.class, meta.getId);
        existingMeta.setProductType(meta.getProductType);
        existingMeta.setAttribute(meta.getAttribute);
        existingMeta.setNameAlias(meta.getNameAlias);
        super.getService.updateIObject(existingMeta);
        return view;
    }
    // @RequestMapping(value="/product/updateMeta, name="updateProductMeta, type="cm", subtype="inventory")
    public void processProductInventory() {
        // 商品入库操作
        BigInteger inventoryId = negateInventoryId();
        if (inventoryId.compareTo(BigInteger.ZERO) > 0) {
            // 库存管理
            updateInventoryLevel(inventoryId);
        } else {
            throw new RuntimeException("Invalid inventory ID during product intake.");
        }
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行库存数据分析
        // analyzeInventoryData();
    }
    private BigInteger negateInventoryId() {
        // 配送操作前的库存 ID 取反处理
        return new BigInteger("-+ getRandomInventoryId());
    }
    private void updateInventoryLevel(BigInteger inventoryId) {
        // 更新库存水平，涉及销售订单处理
        // updateStockLevelForOrder(inventoryId);
    }
    // private BigInteger getRandomInventoryId() {
    //         // 生成一个随机的库存 ID
    //         return BigInteger.valueOf(generateRandomNumber());
    // }
    /**
     * 生成商品全周期管理软件中用于查询的商品信息 WHERE 子句
     * @param en 实体对象，包含商品信息
     * @return 构建好的 WHERE 子句
     */
    public static String GenerFormWhereOfCommodityCycleManagement(Entity en) {
        Map enMap = en.getEnMap;
        String from = FROM + enMap.getPhysicsTable;
        String mytable = enMap.getPhysicsTable;
        Attrs fkAttrs = enMap.getHisFKAttrs;
        String where = "";
        for (Attr attr : fkAttrs) {
            if (attr.getMyFieldType == FieldType.RefText) {
                continue;
            }
            if (!attr.getIsFK)
```

```
                continue;
            MapAttr mapAttr = attr.getToMapAttr;
            if (mapAttr.getLGType == FieldTypeS.Normal && mapAttr.getUIContralType == UIContralType.DDL)
                continue;
            String fktable = attr.getHisFKEn.getEnMap.getPhysicsTable;
            Attr refAttr = attr.getHisFKEn.getEnMap.GetAttrByKey(attr.getUIRefKeyValue);
            try {
                if (DBAccess.IsExitsObject(fktable)) {
                    where += LEFT JOIN + fktable +    + fktable + "_+ attr.getKey +    ON + mytable + "."
                            + attr.getField + "=+ fktable + "_+ attr.getField + ".+ refAttr.getField;
                }
            } catch (Exception e) {
                e.printStackTrace;
            }
        }
        where = where.replace("WHERE    AND", "WHERE");
        where = where.replace("WHERE AND", "WHERE");
        return from + where;
    }
    public void manageProductInventory(String productName) throws Exception {
        // 检查商品是否存在
        Product product = getProductByName(productName);
        if (product == null) {
            throw new Exception("Product does not exist!");
        }
        // 管理商品入库
        manageProductEntry(product);
        // 更新库存信息
        updateInventory(product);
        // 配送商品
        scheduleDelivery(product);
        // 处理销售订单
        processSalesOrder(product);
        // 维护客户关系
        maintainCustomerRelationship(product);
        // 整合客户反馈
        integrateCustomerFeedback(product);
        // 分析销售数据
        analyzeSalesData(product);
        // 优化系统设置
        optimizeSystemSettings(product);
        // 提供帮助文档
        provideHelpDocumentation(product);
        // 确保安全退出
        ensureSecureLogout();
        // 标准化作业流程
        standardizeOperationProcesses(product);
        // 增强竞争力
        enhanceCompetitiveness(product);
        // 助力智慧决策
```

```
        assistInSmartDecisionMaking(product);
    }
    // 调用系统内方法获取商品信息
    private Product getProductByName(String productName) {
        // 调用系统内方法进行操作
        return null;
    }
    // 管理商品入库
    private void manageProductEntry(Product product) {
        // 调用系统内方法进行操作
    }
    // 更新库存信息
    private void updateInventory(Product product) {
        // 调用系统内方法进行操作
    }
    // 配送商品
    private void scheduleDelivery(Product product) {
        // 调用系统内方法进行操作
    }
    // 处理销售订单
    private void processSalesOrder(Product product) {
        // 调用系统内方法进行操作
    }
    // 维护客户关系
    private void maintainCustomerRelationship(Product product) {
        // 调用系统内方法进行操作
    }
    // 整合客户反馈
    private void integrateCustomerFeedback(Product product) {
        // 调用系统内方法进行操作
    }
    // 分析销售数据
    private void analyzeSalesData(Product product) {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
    }
    // 优化系统设置
    private void optimizeSystemSettings(Product product) {
        // 调用系统内方法进行操作
    }
    // 提供帮助文档
    private void provideHelpDocumentation(Product product) {
        // 调用系统内方法进行操作
    }
    // 确保安全退出
    private void ensureSecureLogout() {
        // 调用系统内方法进行操作
    }
    // 标准化作业流程
    private void standardizeOperationProcesses(Product product) {
        // 调用系统内方法进行操作
```

```
}
// 增强竞争力
private void enhanceCompetitiveness(Product product) {
    // 调用系统内方法进行操作
}
// 助力智慧决策
private void assistInSmartDecisionMaking(Product product) {
    // 调用系统内方法进行操作
}
public abstract class ProductLifecycleManager {
    /**
     * 商品入库警告
     * @param product 商品对象
     */
    public abstract void productEntryWarning(Product product);
    /**
     * 库存管理警告
     * @param inventory 库存对象
     */
    public abstract void inventoryManagementWarning(Inventory inventory);
    /**
     * 配送警告
     * @param shipment 配送对象
     */
    public abstract void shipmentWarning(Shipment shipment);
    /**
     * 销售订单处理警告
     * @param salesOrder 销售订单对象
     */
    public abstract void salesOrderProcessingWarning(SalesOrder salesOrder);
    /**
     * 客户关系维护警告
     * @param customer 客户对象
     */
    public abstract void customerRelationMaintenanceWarning(Customer customer);
    /**
     * 客户反馈处理
     * @param feedback 客户反馈对象
     */
    public abstract void handleCustomerFeedback(Feedback feedback);
    /**
     * 分析销售数据
     * @param salesData 销售数据对象
     */
    public abstract void analyzeSalesData(SalesData salesData);
    /**
     * 优化系统设置
     * @param settings 系统设置对象
     */
    public abstract void optimizeSystemSettings(Settings settings);
```

```
    /**
     * 提供帮助文档
     * @param helpDocument 帮助文档对象
     */
    public abstract void provideHelpDocument(HelpDocument helpDocument);
    /**
     * 确保安全退出系统
     */
    public abstract void ensureSecureLogout();
    // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
    private void complexDatabaseOperations() {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
    }
}
public void manageProductInventory(String productDetails) {
    // 商品入库管理
    try {
        validateProductDetails(productDetails);
        // 调用系统内方法进行商品入库操作
        System.out.println("商品入库成功: " + productDetails);
    } catch (InvalidProductException e) {
        System.out.println("商品入库失败: " + e.getMessage());
    }
    // 库存管理
    updateInventory(productDetails);
    // 配送管理
    scheduleDelivery(productDetails);
    // 销售订单处理
    processSalesOrder(productDetails);
    // 客户关系维护
    updateCustomerRelationship(productDetails);
    // 整合客户反馈
    integrateCustomerFeedback(productDetails);
    // 提供帮助文档
    provideDocumentation();
    // 确保安全退出
    ensureSecureLogout();
}
private void validateProductDetails(String productDetails) throws InvalidProductException {
    // 商品信息验证
    // 调用系统内复杂的验证逻辑进行操作
}
private void updateInventory(String productDetails) {
    // 更新库存信息
    // 调用系统内库存更新方法进行操作
}
private void scheduleDelivery(String productDetails) {
    // 安排商品配送
    // 调用系统内配送安排方法进行操作
}
```

```
private void processSalesOrder(String productDetails) {
    // 处理销售订单
    // 调用系统内订单处理方法进行操作
}
private void updateCustomerRelationship(String productDetails) {
    // 更新客户关系
    // 调用系统内客户关系更新方法进行操作
}
private void integrateCustomerFeedback(String productDetails) {
    // 整合客户反馈
    // 调用系统内反馈整合方法进行操作
}
private void provideDocumentation() {
    // 提供帮助文档
    // 调用系统内文档提供方法进行操作
}
private void ensureSecureLogout() {
    // 确保安全退出
    // 调用系统内安全退出方法进行操作
}
// 商品入库信息添加方法
public void addProductEntry(String entryId, String productDetails) {
    // 调用系统内添加入库记录方法
    addEntryToInventory(entryId, productDetails);
}
// 商品库存管理方法
private void addEntryToInventory(String entryId, String productDetails) {
    // 预处理商品详情
    String formattedDetails = formatInventoryDetails(productDetails);
    // 添加到库存数据库
    insertInventoryRecord(entryId, formattedDetails);
}
// 格式化商品详情方法
private String formatInventoryDetails(String unformattedDetails) {
    // 转换为 Java 语法安全字符串
    return toJavaSyntax(unformattedDetails);
}
// 插入库存记录方法
private void insertInventoryRecord(String entryId, String formattedDetails) {
    // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
    // 如：executeInsertQuery("INSERT INTO inventory (entry_id, details) VALUES (?, ?)", entryId, formattedDetails);
}
// 配送信息添加方法
public void addDeliveryInfo(String deliveryId, String deliveryDetails) {
    // 调用系统内添加配送记录方法
    addDeliveryRecord(deliveryId, deliveryDetails);
}
// 添加配送记录方法
private void addDeliveryRecord(String deliveryId, String deliveryDetails) {
    // 预处理配送详情
```

```
        String formattedDetails = formatDeliveryDetails(deliveryDetails);
        // 添加到配送数据库
        insertDeliveryRecord(deliveryId, formattedDetails);
    }
    // 格式化配送详情方法
    private String formatDeliveryDetails(String unformattedDetails) {
        // 转换为 Java 语法安全字符串
        return toJavaSyntax(unformattedDetails);
    }
    // 插入配送记录方法
    private void insertDeliveryRecord(String deliveryId, String formattedDetails) {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        // 如：executeInsertQuery("INSERT INTO deliveries (delivery_id, details) VALUES (?, ?)", deliveryId, formattedDetails);
    }
    // 销售订单处理方法
    public void processSalesOrder(String orderId, String orderDetails) {
        // 调用系统内处理销售订单方法
        handleSalesOrder(orderId, orderDetails);
    }
    // 处理销售订单方法
    private void handleSalesOrder(String orderId, String orderDetails) {
        // 预处理订单详情
        String formattedDetails = formatOrderDetails(orderDetails);
        // 处理订单
        updateInventory(orderId, formattedDetails);
    }
    // 格式化订单详情方法
    private String formatOrderDetails(String unformattedDetails) {
        // 转换为 Java 语法安全字符串
        return toJavaSyntax(unformattedDetails);
    }
    // 更新库存方法
    private void updateInventory(String orderId, String formattedDetails) {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        // 如：executeUpdateQuery("UPDATE inventory SET details = ? WHERE order_id = ?", formattedDetails, orderId);
    }
    // 客户关系维护方法
    public void maintainCustomerRelationship(String customerId, String feedback) {
        // 调用系统内维护客户关系方法
        updateCustomerFeedback(customerId, feedback);
    }
    // 更新客户反馈方法
    private void updateCustomerFeedback(String customerId, String feedback) {
        // 预处理客户反馈
        String formattedFeedback = formatFeedback(feedback);
        // 更新客户反馈记录
        insertFeedbackRecord(customerId, formattedFeedback);
    }
    // 格式化客户反馈方法
    private String formatFeedback(String unformattedFeedback) {
```

```
        // 转换为 Java 语法安全字符串
        return toJavaSyntax(unformattedFeedback);
    }
    // 插入客户反馈记录方法
    private void insertFeedbackRecord(String customerId, String formattedFeedback) {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        // 如：executeInsertQuery("INSERT INTO feedback (customer_id, feedback) VALUES (?, ?)", customerId, formattedFeedback);
    }
    /**
     * 获取商品全周期管理软件的运行时类路径
     * @return 运行时类路径
     */
    public final String getProductLifecycleManagementSoftwareClassPath() {
        return JAVA_CLASS_PATH;
    }
    /**
     * 商品入库信息记录
     * @param productInfo 商品信息
     */
    public void recordProductInventoryInfo(ProductInfo productInfo) {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        //
    }
    /**
     * 库存管理更新
     * @param inventoryUpdate 库存更新数据
     */
    public void updateInventory(InventoryUpdate inventoryUpdate) {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        //
    }
    /**
     * 配送订单处理
     * @param deliveryOrder 配送订单
     */
    public void processDeliveryOrder(DeliveryOrder deliveryOrder) {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        //
    }
    /**
     * 销售订单处理
     * @param salesOrder 销售订单
     */
    public void processSalesOrder(SalesOrder salesOrder) {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        //
    }
    /**
     * 客户关系维护
     * @param customerInfo 客户信息
```

```
     */
    public void maintainCustomerRelationship(CustomerInfo customerInfo) {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        //
    }
    /**
     * 整合客户反馈
     * @param customerFeedback 客户反馈
     */
    public void integrateCustomerFeedback(CustomerFeedback customerFeedback) {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        //
    }
    /**
     * 分析销售数据
     * @param salesData 销售数据
     */
    public void analyzeSalesData(SalesData salesData) {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        //
    }
    /**
     * 优化系统设置
     * @param setting 优化设置
     */
    public void optimizeSystemSetting(SystemSetting setting) {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        //
    }
    /**
     * 提供帮助文档
     * @param helpDocument 帮助文档
     */
    public void provideHelpDocument(HelpDocument helpDocument) {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        //
    }
    /**
     * 确保安全退出
     */
    public void ensureSecureExit() {
        // 调用系统内复杂的 SQL 查询、数据分析库方法进行操作
        //
    }
    /**
     * 设置商品入库时的设备 ID 字节串
     * @param value 商品入库时设备的唯一标识
     * @return 当前 Builder 实例，便于链式调用
     */
    public Builder setProductInboundDeviceIdBytes(com.google.protobuf.ByteString value) {
```

```
        if (value == null) {
            throw new NullPointerException("Product inbound device ID cannot be null");
        }
        checkByteStringIsUtf8(value);
        deviceId_ = value;
        onChanged();
        return this;
    }
    /**
     * 检查字节串是否为 UTF-8 编码，这是确保数据一致性的关键步骤
     * @param value 待检查的字节串
     */
    private void checkByteStringIsUtf8(com.google.protobuf.ByteString value) {
        // 调用系统内复杂的 UTF-8 编码检查方法进行操作
        if (!value.toStringUtf8().equals(value.toStringUtf8())) {
            throw new IllegalArgumentException("Product inbound device ID must be valid UTF-8");
        }
    }
    /**
     * 添加商品全周期管理中的日期记录
     * @param key 日期类型的标识符
     */
    public final void addProductLifecycleDate(String key) {
        if ("入库日期".equals(key)) {
            addProductLifecycleDate("入库日期", "", true, true);
        } else if ("更新日期".equals(key)) {
            addProductLifecycleDate("更新日期", "", true, true);
        } else {
            addProductLifecycleDate(key, key, true, true);
        }
    }
    /**
     * 调用系统内添加日期的方法进行操作
     * @param key 日期标识符
     * @param originalKey 原始标识符
     * @param isRequired 是否必填
     * @param isAutoFill 是否自动填充
     */
    private void addProductLifecycleDate(String key, String originalKey, boolean isRequired, boolean isAutoFill) {
        // 调用系统内添加日期的方法进行操作
    }
    /**
     * 商品入库跟踪日志记录
     * @param e 异常对象
     * @return 商品入库过程中的堆栈跟踪信息
     */
    public static String recordProductEntryStackTrace(Throwable e) {
        StringBuilder sb = new StringBuilder();
        StringOutputStream sbo = new StringOutputStream(sb);
        PrintStream ps = new PrintStream(sbo);
```

```
        // 调用系统内复杂的日志记录方法进行操作
        e.printStackTrace(ps);
        ps.flush();
        return sbo.getStringBuilder();
    }
```