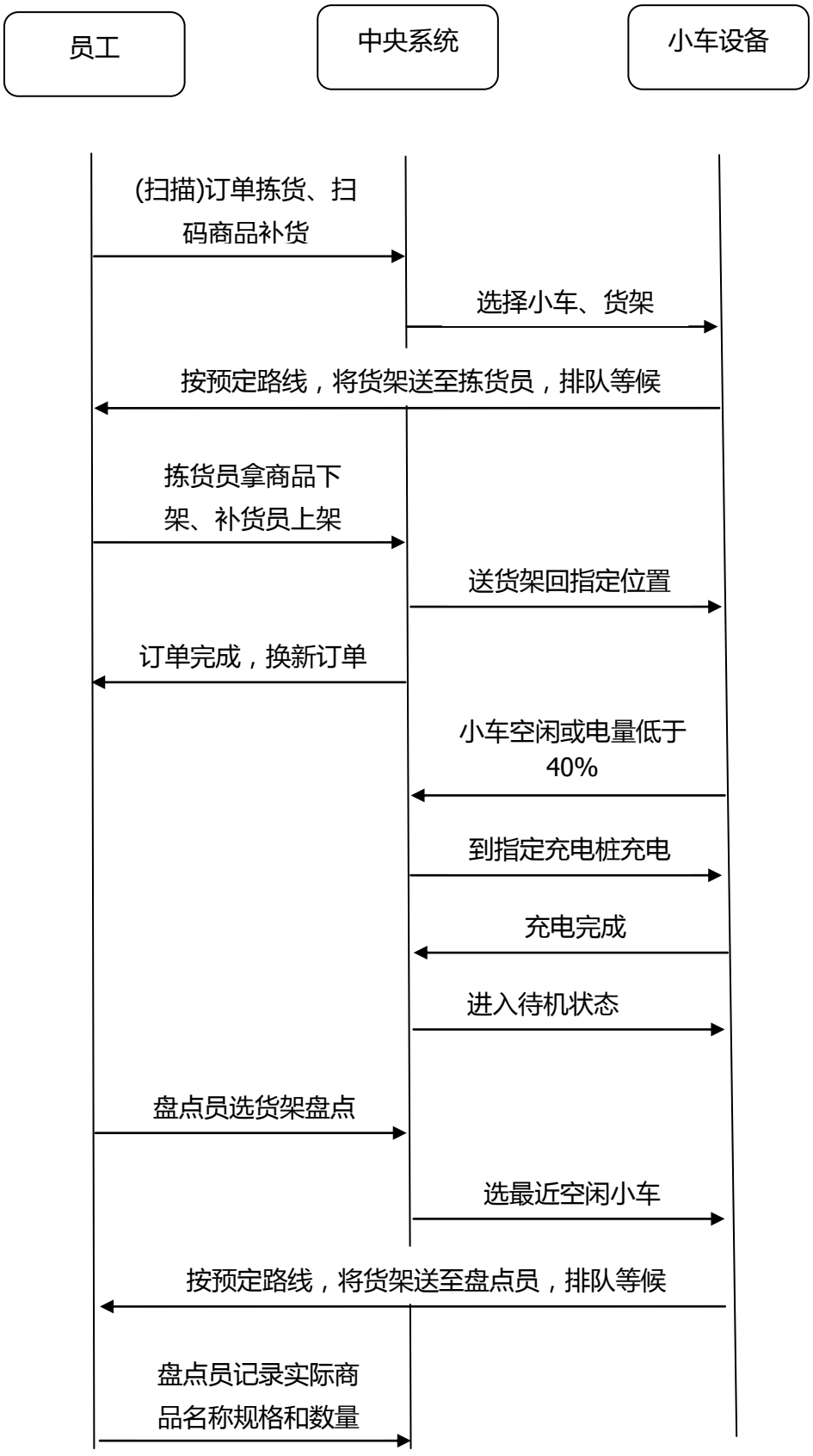
目录

功能概要设计

1. 产品上架
    a) 扫码仓库库位条码/直接手动输入库位编号（库区-通道-货架-层号-格号）
    b) 再依次扫码产品/直接录入产品 ID(按箱/组合上架产品暂不考虑)
2. 选择订单开始打包商品，绑定订单和盛放商品的容器，简称订单容器
3. 获取订单中的产品
    a) 选择小车设备（可能当前有多个设备可用）和目标货架（可能有多个货架上有对应商品）
    b) 分配设备取货（设备自行制定行走路线，给出相对坐标值，只需要向减少间距方向移动即可）
    c) 拣货员将商品从货架下，扫码/直接出库产品，然后拿至订单容器
    d) 设备归位货架（同 b，都属于在两点间通过设备移动货架）
4. 循环 2 中步骤，直至订单完成，换新订单
5. 安排设备充电

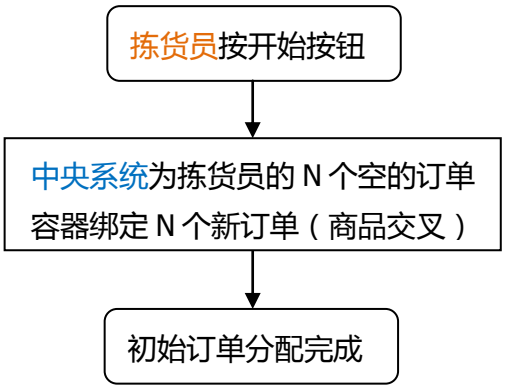
业务顺序图


员工
中央系统
小车设备
(扫描)订单拣货、扫码商品补货
选择小车、货架
按预定路线，将货架送至拣货员，排队等候
拣货员拿商品下架、补货员上架
送货架回指定位置
订单完成，换新订单
小车空闲或电量低于40%
到指定充电桩充电
充电完成
进入待机状态
盘点员选货架盘点
选最近空闲小车
按预定路线，将货架送至盘点员，排队等候
盘点员记录实际商品名称规格和数量

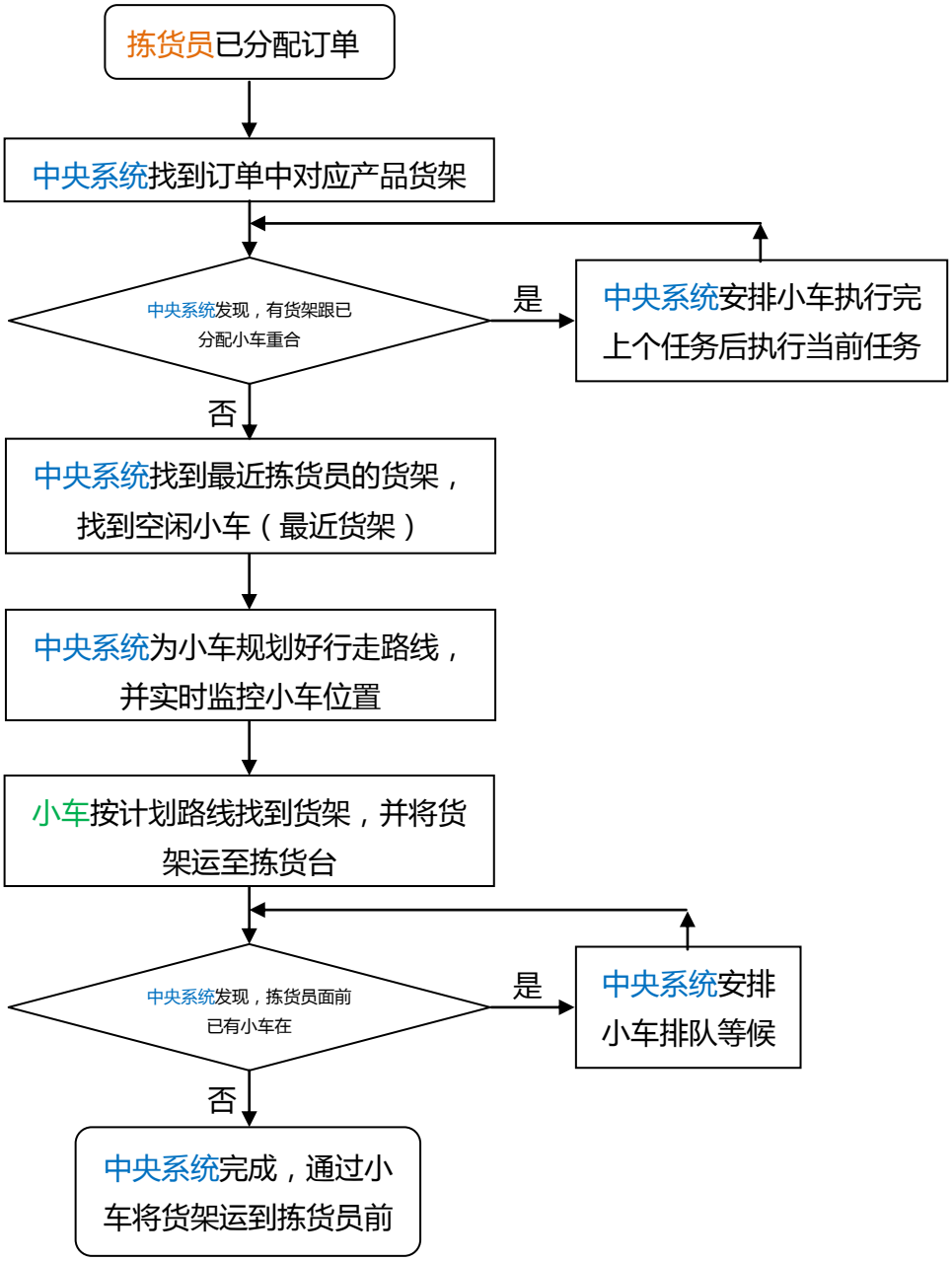
系统工作流程图

拣货（暂不考虑通过设备运送订单货架，仅通过人工选择订单容器）

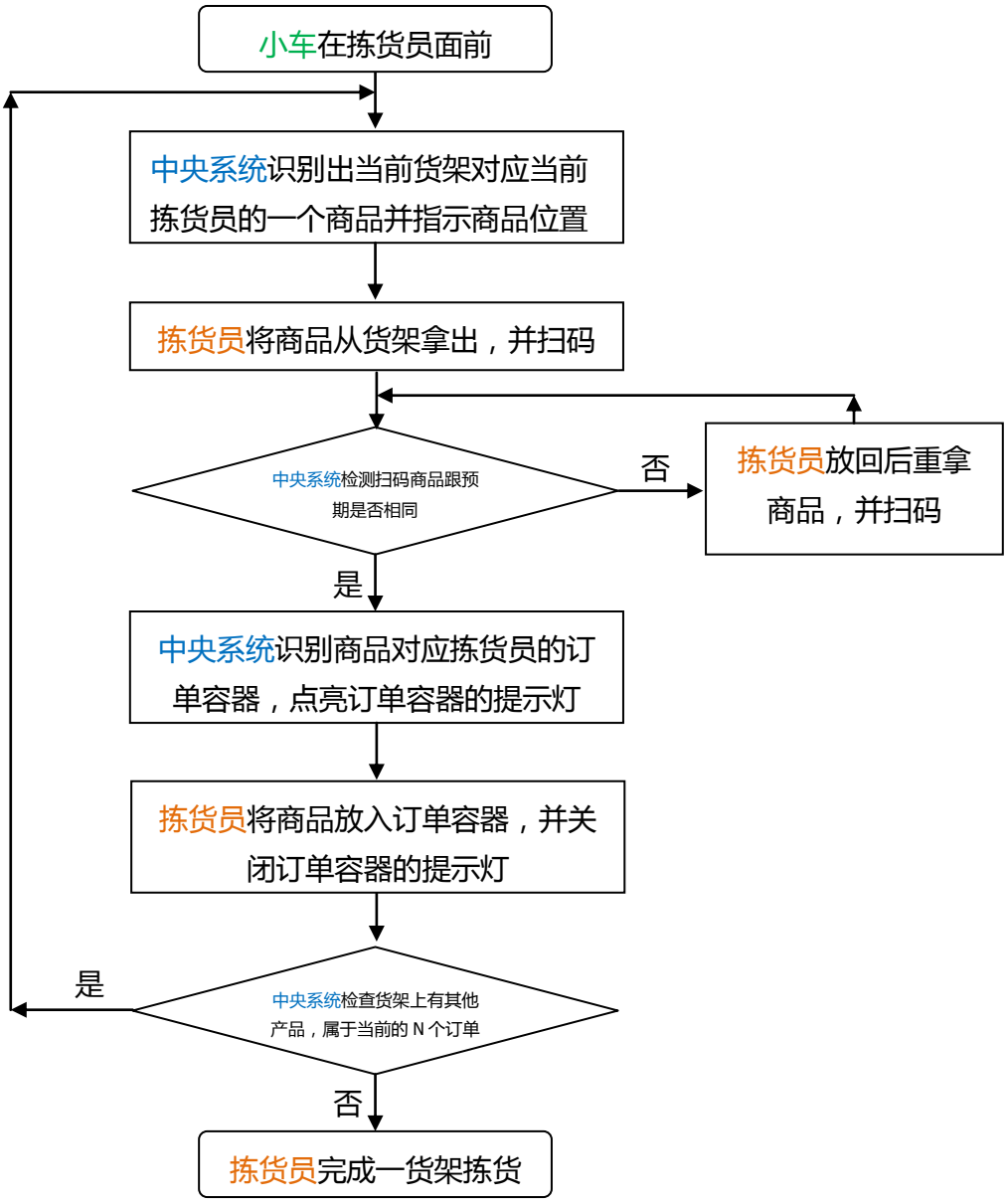
订单分配给拣货员



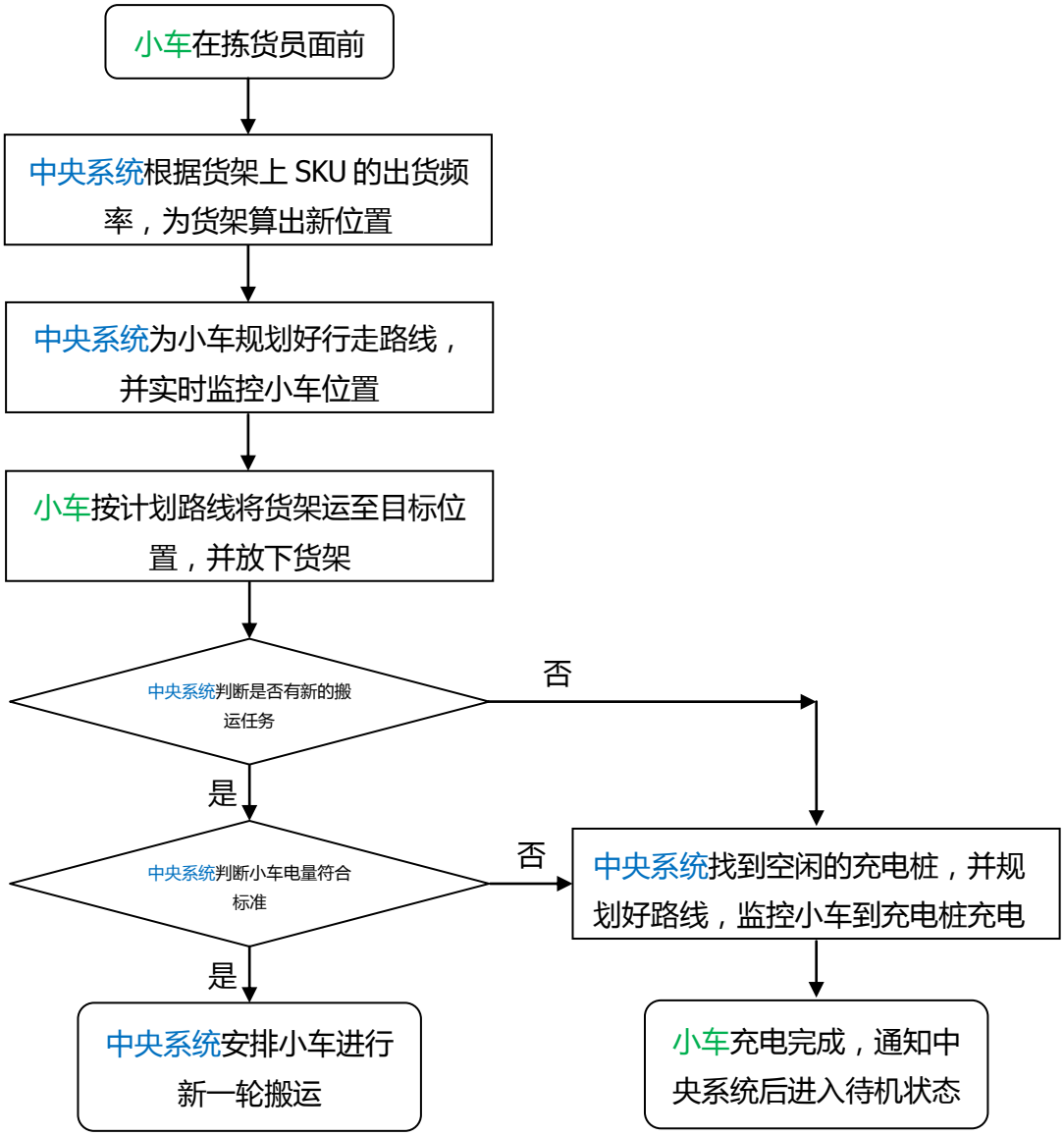
小车运货架到拣货员面前



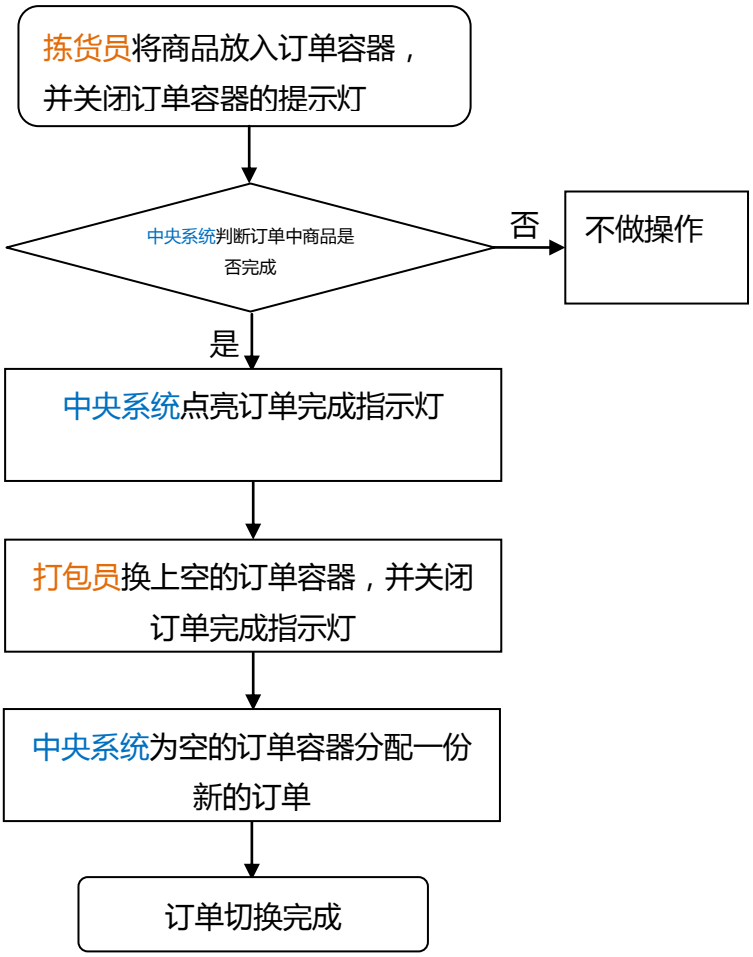
拣货员拣一个货架上的货



小车将货架运回仓储区


小车在拣货员面前
中央系统根据货架上 SKU 的出货频率，为货架算出新位置
中央系统为小车规划好行走路线，并实时监控小车位置
小车按计划路线将货架运至目标位置，并放下货架
中央系统判断是否有新的搬运任务
否
是
中央系统判断小车电量符合标准
否
中央系统找到空闲的充电桩，并规划好路线，监控小车到充电桩充电
是
中央系统安排小车进行新一轮搬运
小车充电完成，通知中央系统后进入待机状态

订单切换

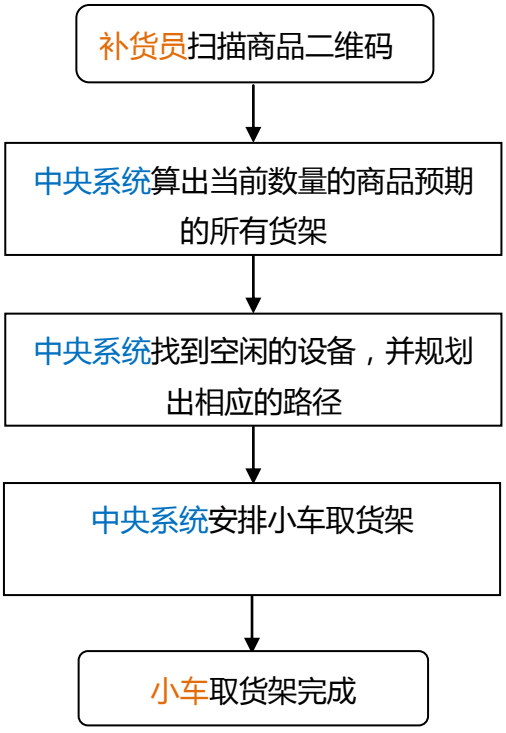

拣货员将商品放入订单容器，并关闭订单容器的提示灯
中央系统判断订单中商品是否完成
否
不做操作
是
中央系统点亮订单完成指示灯
打包员换上空的订单容器，并关闭订单完成指示灯
中央系统为空的订单容器分配一份新的订单
订单切换完成

补货

补货应该是先给出补货计划，补货计划是对于订单的预期以及当前库存量得到的，所以，补货商品是在库存中有预期的对应库位的。

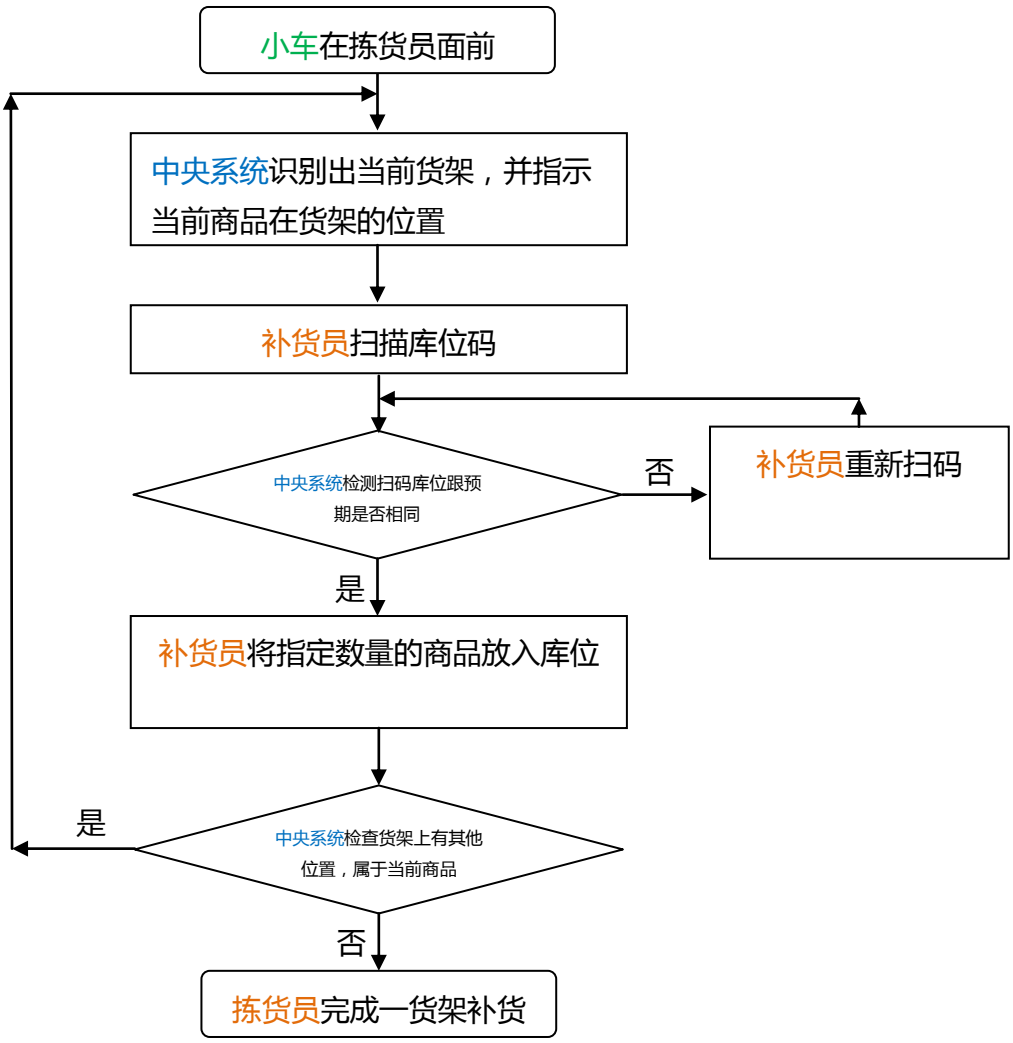
操作是拣货的逆向过程，区别有两点：1，定位货架的查询条件不同：当补货员扫描商品后，根据要补货的数量和货架自身空位量，以及最近一周跟补货商品的销售相似度，综合考虑来决定为哪个货架进行补货，从而选择哪个货架；而拣货需要是订单中的产品，所以只需要考虑，订单需要移动的货架数量、商品先进先出和距离拣货员的距离作为考虑因素。2，货架往返的执行结果不同：对于要补货的货架是上架商品，拣货是相反的下架商品。

补货员对商品扫码，小车取货架



小车运货架到补货员面前（同拣货）

补货员将商品放到货架



小车将货架运回仓储区（同拣货）

商品切换

补货员换个商品进行扫码

盘点

盘点操作跟拣货类似，两个区别分别是：1，对于货架的选择条件，此处是随机一个货架；2，没有对货物的上下架，仅检查核实，操作之后没有商品的增减。

按货架盘点

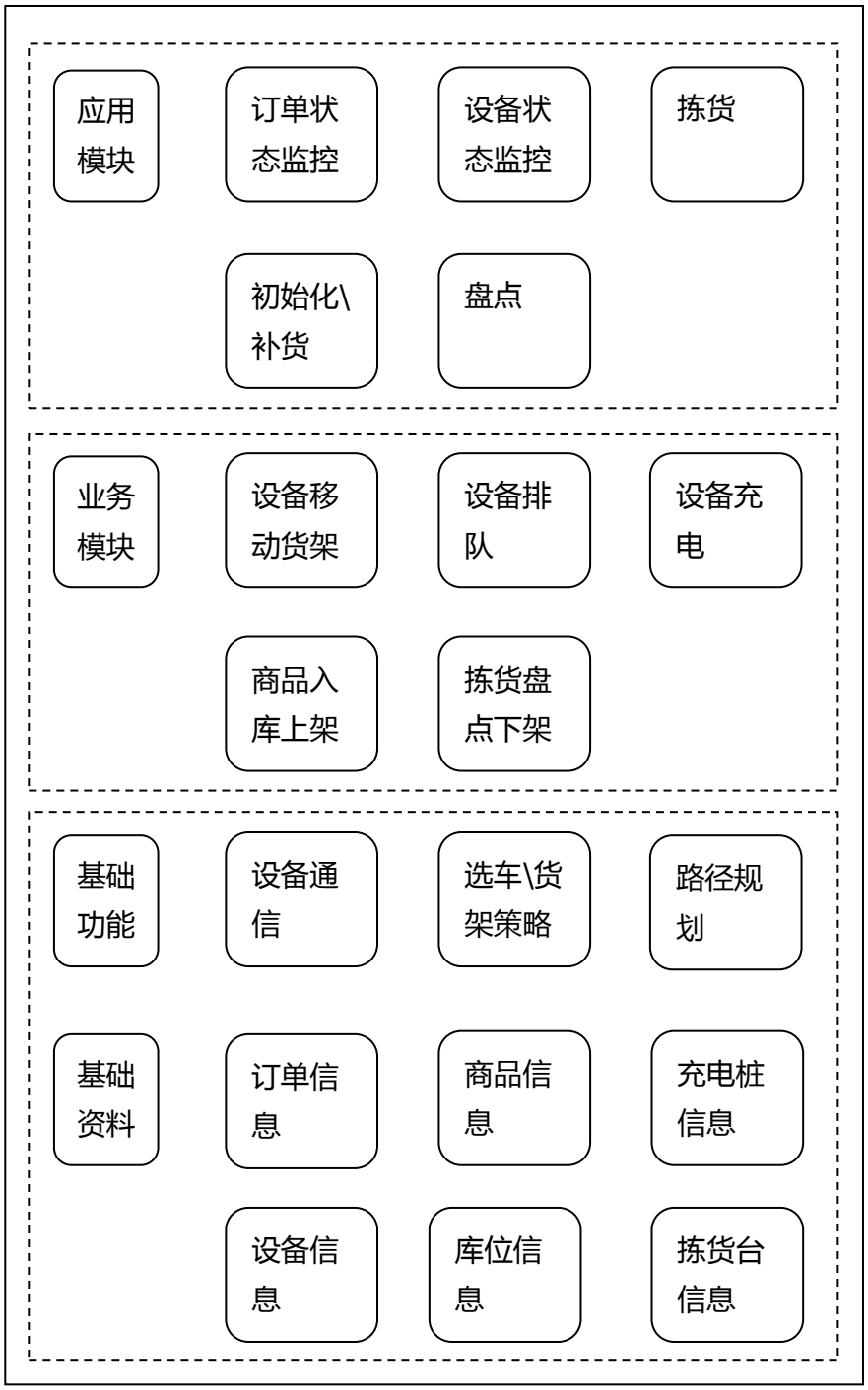
随机选中一台货架，给出系统中货架上的产品及数量，由盘点员确认对应的实际结果。

按商品盘点

随机选中一款商品，给出系统中的货架上的产品及数量，有盘点员确认对应的实际结果。

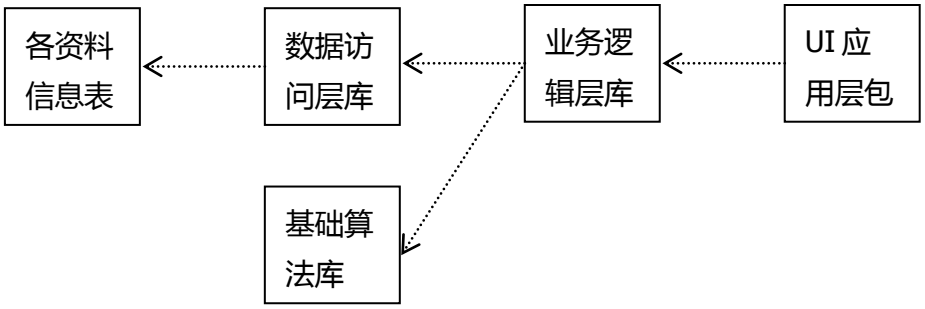
按拣货员/订单盘点

软件模块图

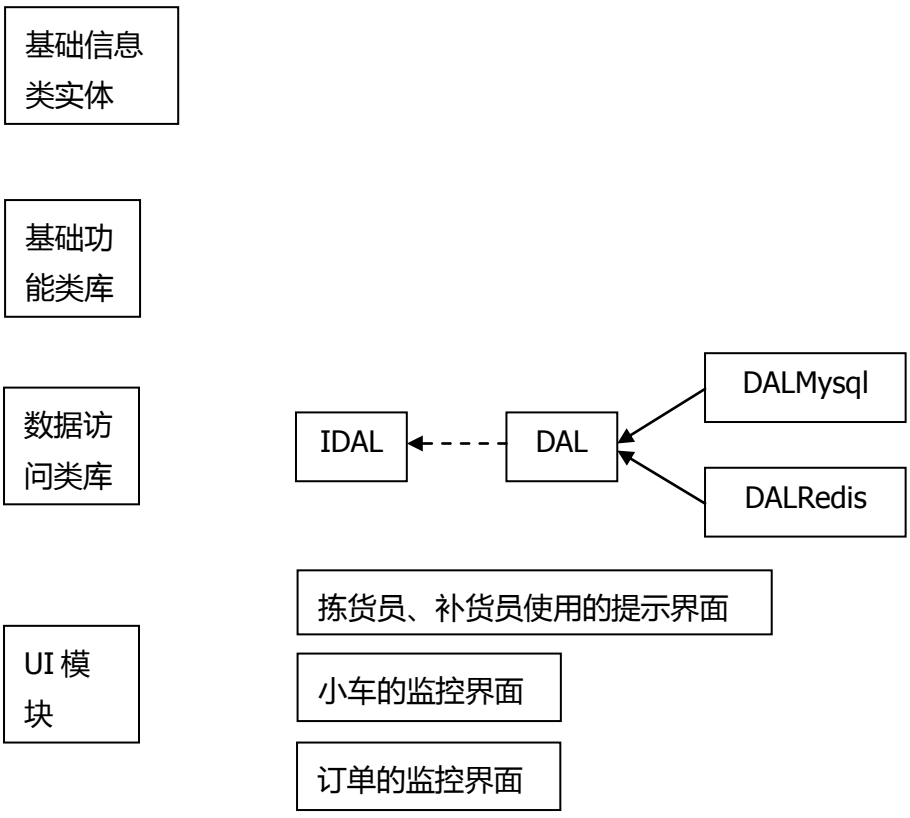

应用模块
订单状态监控
设备状态监控
拣货
初始化\补货
盘点
业务模块
设备移动货架
设备排队
设备充电
商品入库上架
拣货盘点下架
基础功能
设备通信
选车\货架策略
路径规划
基础资料
订单信息
商品信息
充电桩信息
设备信息
库位信息
拣货台信息

软件 UML 图

包图


各资料信息表
数据访问层库
业务逻辑层库
UI 应用层包
基础算法库

类图


基础信息类实体
基础功能类库
数据访问类库
IDAL
DAL
DALMysql
DALRedis
UI 模块
拣货员、补货员使用的提示界面
小车的监控界面
订单的监控界面

数据表结构

持久记录（用于初始化）

拣货员表
人员 ID：1
姓名：张三
性别：男
权限：11
职位：管理员
年龄：20
手机：150XXX
联系地址：南山蛇口

货架信息表
货架 ID：1
货架编码(6 位：仓库 2+行数 1+列数 2+层数 1)：02A271
坐标索引：2
位置历史：3，2
货架层数：4
货架面数：2
各面（分号分隔）每层格数：01020201; 01020301
类型（冗余字段：1 小 2 中 3 大 11 冷）：1

SKU 信息表
SKU ID：1
商品名称：水杯
库存数量：3
型号：300ml
颜色：红色
尺寸 (mm)：20*200*2000
重量(g)：200
备注信息：易碎

商品信息表（bak）

商品 ID(通过条码/ID 唯一定位)：1
条码编号：DE34553233
SKU ID：1
货架 ID：1
货架面号（最多 4 个面）：1
库位号（当前货架所有库位自下到上，自左到右编号）：2 {此库位仅用于显示给拣货员：若将库位，层号信息分别用数据记录表示，数据冗余不增加查询和计算效率}
商品名称及规格（用于显示）：水杯；红色 300ml
生产日期：2015-07-01
过期日期：2016-12-31
尺寸规格(mm)：20*200*2000
重量(g)：200
数量：2
上架时间（扫码/指派）：2016-07-01 10:51:50
出库时间（扫码/指派）：2016-07-10 10:51:50
商品状态（0 正常 3 部分出库 9 出库）：0

设备信息表
设备 ID：1
设备序号（可读编号）：S0012
状态（0 待命 3 工作中 8 充电中 9 故障）：1
绝对坐标 X，Y，Z：（2，3，1）
IP 地址：192.168.1.111
厂家：XX 有限公司
生产日期：2016-07-17
上线日期：2016-07-19
备注：哈哈哈

订单信息表
自增 ID：1
订单编号：SD0012
SKU 信息：1,2;4,1
商品总数：3
优先级：1

状态（0 未处理， 5 已处理，9 异常）：0
拣货台 ID：1
拣货台 ID：1
导入订单/下单时间：2016-07-15 10:10:10
开始拣货时间：2016-07-15 11:10:10
备注：哈哈哈

仓库内站点（补货/拣货台/充电桩）
序号 ID：1
可读序号：HT002
仓库 ID：1
IP 地址：192.168.1.10
位置索引：1
绝对坐标 X，Y，Z：（2，3，1）
类型（5 补货 3 拣货 2 充电桩）：2
状态（0 空闲 3 工作中 9 故障）：3
备注：

仓库内位置坐标表
位置 ID：1
仓库序号：2
绝对坐标 X，Y，Z：（2，3，1）
状态（0 正常 3 工作中 8 阻塞 9 故障）：0
类型（1 货架点 2 拣货台 3 补货台 4 充电桩 5 路交叉点）：1

仓库内路线表
序号 ID：1
仓库序号：2
位置 1 ID：1
位置 2 ID：2
类型（1 正向 2 反响 3 双向）：1
状态（0 正常 8 阻塞 9 故障）：0

实时状态记录（用于跟踪调试 - Mysql）

实时订单表 订单开始拣货时增加记录，每次录入商品更新记录
自增序号：1
订单 ID：5542144
商品总数：5
商品信息（SKU ID，数量）：1,2;3,1;4,1;6,1
拣货员 ID：3
拣货台 ID ：1
取货设备（冗余字段：设备编号）：1,2
取货商品（冗余字段：SKU ID，条码 ID，数量;）：1,1,1;1,2,1;3,3,1
已取数量：3
状态（0 未处理，3 拣货中，5 完成拣货，9 异常）：3
拣货备注：2016-07-20 10:10:10（2），2016-07-20 10:13:10（1）

实时拣货商品表 员工每次开始为一个订单拣货，增加一个订单的所有商品记录
自增序号：1
站台 ID:1
订单 ID：1
SKU ID：11
总数量：2
已完成数量：1
状态（0 未处理，3 处理中，5 已完成，9 异常）：3
最后更新时间：2016-07-01 13:50:10

实时移动货架表 给设备发布取货任务时增加记录，小车业务状态改变后更新状态
自增序号：1
货架 ID：1
设备 ID：1
商品数量：1
商品 ID（通过条码/ID 唯一定位）：1,2,3
SKU ID（同个 SKU 包含多个商品）：1,2,3
订单 ID：1,2,3
拣货台 ID：1,2,3
状态（3 取货中，5 已完成，9 异常）：3

开始取货时间：2016-07-01 13:50:10
到达货架时间：2016-07-01 13:59:10
搬起货架时间：2016-07-01 13:59:10
送达货架时间：2016-07-01 13:59:10
完成拣货时间：2016-07-01 13:59:10
送回货架时间：2016-07-01 13:59:10

实时设备位置表 设备连接后，每秒增加 5 条记录（前期先用 Mysql，吞吐支持不到时再独立）
自增序号：1
设备 ID：1
状态（0 候命中 1 取货中 2 运货中 3 电量低 4 充电中 11 故障 12 失联）：1
当前功能：1
起点位置索引：1
终点位置索引：2
当前位置索引：3
当前绝对位置 X，Y，Z： 1，2，3
IP 地址：192.168.1.111
当前时间：2016-07-01 13:50:10
备注：电量低时记录电量，取货中时记录商品名称，运货中时记录货架编号

日志、异常记录每步操作（用于历史备案 - 文档/Nosql）

通讯记录表
记录时间：2016-07-01 11:20:50
对象类型（1 设备 2 拣货员）：1
行为主体 ID：1
操作步骤（功能名称）：设备取货
日志内容（操作记录）：设备接收取货指令（XXX）

员工行为记录（拣货/补货/登入/登出）表 员工每次扫码拣货、补货后增加一条记录

自增序号：1
员工 ID：1
行为类型（1 拣货 2 补货 3 上班 4 下班）：1
商品 ID：11
SKU ID：11
所在站台 ID：22
操作时间：2016-07-01 13:50:10

异常记录表
记录时间：2016-07-01 11:20:50
对象类型（1 设备 2 拣货员）：1
行为主体 ID：1
操作步骤：设备取货
异常信息：发信息（XXX）获取设备（XXX）状态，响应超时

状态信息解释

动态状态 -- 行为结果（0 未处理，3 处理中，5 已完成，9 异常）

实时订单状态（0 未处理，3 拣货中，5 完成拣货，9 异常）

订单状态（0 未处理， 5 已处理，9 异常）

实时拣货商品状态（0 未处理，3 处理中，5 已完成，9 异常）

实时移动货架状态（3 取货中，5 已完成，9 异常）

静态状态 -- 自身状态（0 正常 3 工作中 5 已完成 8 阻塞 9 故障）

仓库内位置坐标表状态（0 正常 3 工作中 8 未工作 9 故障）

仓库内路线状态（0 正常 8 阻塞 9 故障）

仓库内站点状态（0 空闲 3 工作中 9 故障）

设备信息状态（0 待命 3 工作中 8 充电中 9 故障）

商品信息状态（0 正常 3 部分出库 5 出库)

其他状态

故障状态

通讯协议

通信协议格式

开始位	<
包数据高字节	0
包数据低字节	N1
保留位高字节	头文件属性
保留位低字节	头文件属性
功能码 1	0x01
功能 1 数据高字节	0
功能 1 数据低字节	N1
功能 2、3	功能保留（6 位）
功能码 4	0x04
功能 4 数据高字节	0
功能 4 数据低字节	N4
数据位 1	0xAF
数据位 2	0xAF
……	……
数据位 N1	0xAF
若有其他功能的数据则依次顺序排序，若没有则不保留位置（包括功能 1）	
校验码	0xAF
校验码	0xAF

头文件属性

属性名称	所在位置	参数说明
是否需要回复	低字节低位第 2 位 0000 0000 0000 0010	0：不需要回执 1：需要

功能码

功能名称	功能码	数据位数据格式
上位机：查询状态	0x10	
上位机：	0x11	

回执		
上位机：停止移动	0x12	
上位机：转向	0x13	顺 0/逆 1 旋转（1Byte），转动次数（1Byte）
上位机：安排充电	0x20	充电桩 ID（2 Byte），第一步 X（2 Byte），第一步 Y（2 Byte），第一步 Z（1 Byte），……,第 i 步 X（2 Byte），第 i 步 Y（2 Byte），第 i 步 Z（1 Byte）……，最后一步 X（2 Byte），最后一步 Y（2 Byte），最后一步 Z（1 Byte）
上位机：移动到位置等待	0x21	保留位（2Byte）,第一步 X（2 Byte），第一步 Y（2 Byte），第一步 Z（1 Byte），……,第 i 步 X（2 Byte），第 i 步 Y（2 Byte），第 i 步 Z（1 Byte）……，最后一步 X（2 Byte），最后一步 Y（2 Byte），最后一步 Z（1 Byte）
上位机：去找货架	0x22	货架 ID（2 位），第一步 X（2 Byte），第一步 Y（2 Byte），第一步 Z（1 Byte），……,第 i 步 X（2 Byte），第 i 步 Y（2 Byte），第 i 步 Z（1 Byte）……，最后一步 X（2 Byte），最后一步 Y（2 Byte），最后一步 Z（1 Byte）
上位机：运货架到拣货台	0x23	拣货台 ID（2 位），第一步 X（2 Byte），第一步 Y（2 Byte），第一步 Z（1 Byte），……,第 i 步 X（2 Byte），第 i 步 Y（2 Byte），第 i 步 Z（1 Byte）……，最后一步 X（2 Byte），最后一步 Y（2 Byte），最后一步 Z（1 Byte）
上位机：送回货架到仓储区	0x24	货架 ID（2 Byte），第一步 X（2 Byte），第一步 Y（2 Byte），第一步 Z（1 Byte），……,第 i 步 X（2 Byte），第 i 步 Y（2 Byte），第 i 步 Z（1 Byte）……，最后一步 X（2 Byte），最后一步 Y（2 Byte），最后一步 Z（1 Byte）
上位机：小车是否可以前行	0x25	状态（2Byte）
小车：当前状态/心跳	0x30	小车 ID（2 Byte），X 坐标（2 Byte），Y 坐标（2 Byte），Z 坐标（1 Byte）,当前状态（2Byte）,保留位（3Byte）
小车：电量低	0x31	电量（2 Byte）{百分比值），X 坐标（2 Byte），Y 坐标（2 Byte），Z 坐标（1 Byte）
小车：遇到障碍	0x32	障碍距离（2 Byte），X 坐标（2 Byte），Y 坐标（2 Byte），Z 坐标（1 Byte）
小车：	0x33	货物重量（2 Byte），X 坐标（2 Byte），Y 坐标（2 Byte），Z

超载		坐标（1 Byte）
小车：货物不稳	0x34	货物重量（2 Byte），X 坐标（2 Byte），Y 坐标（2 Byte），Z 坐标（1 Byte）
小车：未知异常	0x39	保留位（2Byte）,X 坐标（2 Byte），Y 坐标（2 Byte），Z 坐标（1 Byte）
小车：开始充电	0x40	小车 ID（2Byte）,X 坐标（2 Byte），Y 坐标（2 Byte），Z 坐标（1 Byte）
小车：找到货架并抬起	0x41	小车 ID（2Byte）,X 坐标（2 Byte），Y 坐标（2 Byte），Z 坐标（1 Byte）
小车：到拣货台	0x42	小车 ID（2Byte）,X 坐标（2 Byte），Y 坐标（2 Byte），Z 坐标（1 Byte）
小车：送回货架并放下	0x43	小车 ID（2Byte）,X 坐标（2 Byte），Y 坐标（2 Byte），Z 坐标（1 Byte）
小车：收到取货架命令	0x44	小车 ID（2Byte）
小车：问是否可以前行	0x45	小车 ID（2Byte）,拣货台 ID（2Byte），保留位（3Byte）
拣货员：当前状态	0x50	拣货员 ID（2Byte），拣货台 ID（2Byte），总可拣订单数（2Byte），空闲订单位（1Byte）
拣货员：请求订单	0x51	拣货员 ID（2 Byte），拣货台 ID（2Byte），订单数量（2Byte），保留位（1Byte）
拣货员：找到商品	0x52	拣货台 ID（2Byte），条码位数（2Byte），商品数量（2Byte），保留位（1Byte），商品条码
拣货员：商品放入订单分拣	0x53	货架 ID（2Byte），订单 ID（2Byte），商品 ID（2Byte），保留位（1Byte）
拣货员：完成拣货并结束	0x55	拣货台 ID（2Byte），拣货员 ID（2Byte），保留位（3Byte）
上位机：询问拣货台状态	0x60	

上位机：分配订单	0x61	保留位（2Byte），第一张订单 ID（2Byte）,第一订单商品总数（2Byte）,保留位（1Byte），……，第 i 张订单 ID（2Byte）,第 i 订单商品总数（2Byte）,保留位（1Byte），……, 最后一张订单 ID（2Byte）, 最后一订单商品总数（2Byte）,保留位（1Byte）
上位机：当前商品对应订单	0x62	订单 ID（2Byte），商品 ID（2Byte），Sku ID（2Byte），保留位（1Byte）
上位机：货架待拣商品对应信息	0x63	货架 ID（2Byte），商品库位（2Byte），商品 ID（2Byte）,货架继续 1 取货标志（1Byte），货架库位字节数（2Byte），商品名称字节数（2Byte），条码字节数（1Byte），货架库位类型，商品条码，商品名称
上位机：拣货结果	0x64	0 成 9 败（2Byte）,原因字节数（2Byte），保留位（3Byte），失败原因

路径规划

直接按坐标接近来决定行走方向，转弯越少越好

选择策略（移动总曼哈顿距离最短）

选订单：

初始订单时，按时间前 N 个(选择有更多共同商品，或者在相同货架的订单)；

换新订单时，最早订单（选择跟当前拣货员待拣商品更多重合的订单）

选货架：

拣货时，选择拥有更多待检订单商品、靠近拣货台的货架

补货时，选择跟对应 SKU 出货频率接近的货架，同时满足更接近（大于）当前商品尺寸空库位的货架

选小车：

最靠近货架的空闲小车

算法实现逻辑

绿色背景为当前方案

计算待选货架

Si：第 i 个货架，Gi：第 i 个商品 SKU，Gic：第 i 个商品有 c 个

1，当前拣货员 =》 所有订单

=》所有商品 SKU 及 数量

=》 所有商品货架 及 对应货架上商品数量

=》按商品 SKU 分类取得所有货架组合

=》计算每个商品满足数量的货架组合

=》货架原子集（用小单元组合）

=》从每个商品的组合中挑选一个重新组合，并去重

=》货架原子集（用小单元组合）

G1：S1+S2, S2+S3, S1+S3

G2：S1, S3

G3：S2, S3

G4：S2, S4

2，当前拣货员 =》 所有订单

=》所有商品 SKU 及 数量

=》 所有商品货架 及 对应货架上商品数量

=》所有货架组合

=》计算所有商品满足数量的货架组合

=》货架原子集（用小单元组合）

S1：G1c, G2c

S2：G1c, G3c, G4c

S3：G1c, G2c, G3c

S4：G4c

确定货架

获得货架集合

=》集合所有货架到拣货台的总距离

=》最小距离，相同距离用最少货架，货架数相同随机取

路径搜索

数据结构的复杂度

V：节点数；E：路径数；Degree(V)：节点维度（V 节点路径数）

结构名称	空间复杂度	时间复杂度		
		增加一条路径	相邻节点的路径	遍历一个节点的路径
路径	E	1	E	E
邻接矩阵	V^2	1	1	V
邻接链表	E+V	1	Degree(V)	Degree(V)

由于仓库的结构中每个节点仅跟附近节点有路径，大部分节点之间没有连接，使用邻接矩阵将太多空间闲置，路径的查询效率相对有点低，暂时考虑使用邻接链表实现。

下标	数据	节点位置	名称	边指针列表
1	22	22，33，11	AA	索引 权重 距离：2 1 1；索引 权重 距离：3 1 1；索引 权重 距离：4 1 1
2	33	22，43，11	AB	索引 权重 距离：1 1 1；索引 权重 距离：3 1 1
3	44	22，53，11	AC	索引 权重 距离：1 1 1；索引 权重 距离：2 1 1
4	55	32，53，11	CA	索引 权重 距离：1 1 1
5	45	33，53，11	CB	
6				

搜索算法

1. 深度优先搜索：沿一条路纵向搜索
2. 广度优先搜索：按层次横向搜索

最短路径算法

1. Dijkstar：从未访问节点中选择距离出发点最短路径的节点，直到所有节点被访，数据结构用邻接链表

=》结果是两点间的最短路径，若需要计算所有节点间距离也可以，则需要为每个节点都执行当前算法，整体时间复杂度跟 floyd 算法一致，但 floyd 算法的计算量小。

在未访问节点中寻找最短路径节点的算法逻辑：

初始代码逻辑：外层循环遍历未访问节点列表，内层循环遍历已访问节点列表，依次判断两个循环中节点间的最短距离，两个列表节点数和固定，时间复杂度为 O（n^2）=n*(n-m)，m,n 分别为未访问和全部节点数

```
int minIdx = 0, minDistance = Int32.MaxValue, tmpLen;
foreach (int item in UnkownList)
{
    foreach (KeyValuePair<int, int> visitor in VisitedList)
    {
        tmpLen = graph.CheckEdgeDistance(item, visitor.Key);
        if (tmpLen > 0 && tmpLen + visitor.Value < minDistance)
        {
            minIdx = item;
            minDistance = visitor.Value + tmpLen;
        }
    }
}
pathList.Add(graph.GetHeadNodeByID(minIdx));
VisitedList.Add(minIdx, minDistance);
UnkownList.Remove(minIdx);
```

优化后的逻辑：先在未访问节点列表中循环，找到跟起点建立联系节点的最短距离，再循环更新未访问节点中通过所选节点被缩短的距离，时间复杂度为 O（n）=2*m， m 为未访问节点数（未访问节点会越来越少）

```
int minIdx = 0, minDistance = Int32.MaxValue, tmpLen;
//先找到当前最小的距离值
foreach (KeyValuePair<int, int> item in UnkownList)
{
    if (item.Value > 0 && item.Value < minDistance)
    {
        minIdx = item.Key;
        minDistance = item.Value;
    }
}
pathList.Add(graph.GetHeadNodeByID(minIdx));
VisitedList.Add(minIdx, minDistance);
//再更新新节点减少的距离
foreach (KeyValuePair<int, int> item in UnkownList)
{
    tmpLen = graph.CheckEdgeDistance(item.Value, minIdx);
    if (tmpLen > 0 && (item.Value < 0 || tmpLen + minDistance < item.Value))
    {
        UnkownList[item.Key] = tmpLen + minDistance;
    }
}
```

2. Floyd-Warshall：计算每个节点（A）对任意两点间（X、Y）距离的贡献，若 XY 两点间距离因节点 A 而缩短，则用 A 作为 XY 间路线的一站，数据结构用邻接矩阵

   =》结果是任意两点间的最短距离，优点是一次计算一直使用，缺点是计算的时间复杂度较高（$O(V^3)$）

3. 最终算法（两种都实现，分别用于不同场景）

   a) 默认使用第二种算法，调度时只需知道起止点则不需要计算路径，可以直接使用初始计算结果

   b) 当设备遇到故障或者遇到堵塞时，需要重新规划路线时，将某个位置设为不可行，通过第一种方案重新计算路径。

生命周期

全局变量

AllMapPoints（地图上所有节点列表）：仅用于查找对应位置的坐标值，地图初始加载时实例化，资源在程序运行阶段一直没有释放，数据值也一直没有改变。

RealDevices（当前所有小车列表）：用于根据小车 ID 查看小车状态和位置，项目启动时实例化，资源没释放，小车 IP 值在发心跳时如果 IP 变了则更新，状态值在每次小车状态改变时都会改变（闲、忙、充电），位置绝对值随着小车心跳位置改变而改变也会修改数据表数据，完成某个任务达到一个节点时，小车位置索引会改变。每次启动时位置为上次结束时的位置，状态为未启用（程序启动时设置），未启用状态的小车处于隐藏状态。

RealShelves（当前所有货架列表）：用于根据货架 ID 查看货架位置，项目启动时实例化，资源没释放，由于货架拣完货后都放回了原来位置，并且没有对于小车搬走货架时没实时更新货架位置，所以暂时状态没做任何修改。

RealStation（当前所有拣货台/充电桩/补货台列表）：用于根据站台 ID 查看站台位置，项目启动时实例化，资源没释放，拣货台请求订单时，根据拣货台对应状态修改实时信息（包括 IP 和状态），没更新数据表记录，每次启动拣货台都默认为未启用状态（状态在 DB 中）。

ShelvesNeedToMove（待搬运的货架列表）：每次分配订单时确定商品的货架，确定了仓储区的货架时增加一条记录，当小车回复说已经收到了去取货架命令，则此时将对应货架从列表中删除，并加入移动中货架，当为货架分配小车时，修改小车状态为 Working，并将小车赋值给对应货架。

ShelvesMoving（移动中的货架）：小车回复收到取货架命令时增加记录，并将货架状态改为 PreWorking，货架运回仓储区时，将对应货架记录移除，到达和离开拣货台的状态分别被改为 Working 和 AfterWorking，当小车在一个拣货台拣完货需要过去其他拣货台时，会修改将当前拣货台更新为上个拣货台，当到达新拣货台时会将上个拣货台记录清空。

StationShelfProduct（当前货架和拣货台对应要拣货的商品）：分配订单后，确定拣货的货架时，若拣货台和货架唯一索引的记录没有时新增记录，若存在则增加其中的商品，拣货员将商品放入订单分拣箱时删除记录中的一个商品，当货架对应拣货台本次要拣货物拣完时，删除记录。

RealGraphTraffic（当前地图信息）：项目启动时初始化地图，不释放资源，实时增加、关闭、启用路线/节点时，更新对应模块的状态。

InteractQueue（当前数据包列表）：数据接收线程收到数据时记录加入队列，数据处理线程取走数据时记录拉出队列，资源不释放。

触发条件

分配订单给拣货台

1. 拣货台开始工作时，向系统申请初始化订单

2. 有新订单来临时，系统会询问所有拣货台是否可以接受新订单及对应数量

3. 打包员拿走（双击红色高亮）一个订单箱时，拣货台向系统申请一个订单

小车去拣货台

1. 仓储区有新货架需要运到拣货台，小车在仓储区抬起货架后
2. 小车在拣货台拣完货后，其它拣货台有当前货架上商品，从当前拣货台直接过去
3. 小车回去仓储路上，任意一个拣货台分配到新订单，其中商品在当前货架上有剩余，从实时位置直接过去对应拣货台

拣货台状态

1. 拣货员开始工作，从系统申请到订单，处于工作状态
2. 拣货员将商品放入订单箱后，所有订单箱完成拣货、或者双击完成订单箱当前没有新订单、或者初始状态当前没新订单，处于闲置状态
3. 拣货员结束工作并完成当前拣货后，或者拣货员开始工作前，处于未启用状态

小车状态

1. 启动但未发心跳连接系统前，未启用状态（主控看不到）
2. 首次发送心跳未分配新任务，或者放下货架未分配新任务，处于闲置状态
3. 安排到新任务但没抬起货架，处于空车忙碌状态
4. 抬起货架后，直到放下货架前，处于抬起小车的忙碌状态
5. 到充电桩开始充电后，处于充电状态

交互流程

主控系统运行，拣货台和小车程序运行（不分先后）

1. A 拣货台（开始）：告知系统开始工作，目的是申请一批（6 个）新的订单
2. B 小车（空闲心跳）：告知系统当前位置并激活小车
3. A 系统（回复拣货台）：按时间顺序找 6 个未处理订单，并将订单号和订单中商品总数发给拣货台，此处之后可以优化为寻找最相关的，或者商品跟拣货台最近的 6 个订单，若订单数量不足 6 个，则发送当前可以提供的数量，若无则发送空列表
    a) 订单分配时，会先从当前移动中的货架进行过滤
    b) 剩余订单商品再从待移动货架进行过滤
    c) 最后剩余未找到的商品才去仓储区货架中查找

d) 若发现当前移动货架已经在回去仓储区的路上了，则为其重新分配路线让其直接过去拣货台

e) 若其在过去拣货台路上或者正在其他拣货台拣货，则将当前拣货台要拣货商品放入该货架拣货队列中，其拣完货后会从拣货台直接过来当前拣货台

4. A 系统（发给小车）：寻找当前闲置或充电中的所有小车，并找到待移动货架中距离小车最近的货架位置，安排小车去取货架，若没有小车可用，则不做任何事情

5. B 系统（回复小车）：上面是在拣货台收到订单时安排小车去取货架，这是回复小车的心跳，查看有无需要搬运的货架

   a) 若有，则将搬运命令及货架位置对应的路径发给小车

   b) 若无，则不做任何处理

6. Ba 小车（忙碌心跳）：小车发送自己过去货架的实时位置形成路径

7. 小车（找到并抬起货架）：其实找到货架系统是可用识别的，但对于小车是否准备好出发无法判断，比如是否已经抬起货架，所以需要小车准备好出发去拣货台时，自己发消息给系统

8. 系统（回复小车）：找到当前货架对应要去的拣货台，并将路径拐点发给小车（包括起终点）

9. 小车（忙碌心跳）：小车自身按设定的速度（长度）将路径进行切割，并将切割后的位置发给系统，表示小车当前的位置

10. 小车（询问是否可以继续向前）：小车到达拣货台等待区域，会先询问系统是否可以继续向前

11. 系统（回复小车）：判断当前拣货台是否有其他小车已经过来或者正在拣货，如果没有则回复可以过来，如果存在则回复工作状态

12. 小车（重复 10-12）：是否可以继续向前，直到系统回复可以向前

13. 小车（忙碌心跳）：自己切割剩下的最后一段路，并将对应实时位置发给系统

14. 小车（到达拣货台）：这一步系统也是可以判断的，但无法判断小车是否已经准备好开始拣货，比如是否按收到的命令完成转体（指定的面朝向拣货员），所以需要小车发消息给拣货台

15. 系统（发给拣货台）：当前需要拣货商品所在货架库位，货架布局，商品名称和商品条码发给拣货台

16. 拣货台（显示商品）：绘制货架布局，并将商品所在库位高亮显示，显示商品名称并自动填充商品条码

17. A 拣货员（单击拣货）：拣货台发消息给系统，执行对当前商品进行拣货

18. A 系统（回复拣货台）：找到当前商品在当前拣货台对应的订单编号，发给当前拣货台，判断当前货架在当前拣货台是否还有其他商品需要被拣，若有则等待，若无则查看是否有其他拣货台的商品，若有则发消息给小车直接过去其他拣货台，若无则发消息给小车回仓储区

19. B 系统（发给小车）：若当前货架在当前拣货台没有待拣商品：

    a) 若当前货架在其它拣货台有待拣商品，则另个拣货台设置终点，算出两个拣货台之间的路径，则发给小车到拣货台命令同时附上对应路线

    b) 若没有则发给小车回仓储区命令，同时将货架初始在仓储区的位置作为终点，将两点之间的路径发给小车

20. A 拣货台（显示订单）：收到订单编号，跟当前订单箱对应的编号进行比对，将一直的订单箱高亮显示，若找不到订单编号，则提示拣货商品失败（若系统启动时存在商品拣货未完成可能会导致脏数据）

21. A 拣货员（单击订单箱）：单击高亮显示的订单箱，表示商品放入订单箱，则拣货台将对应订单拣货命令发给系统

22. A 系统（回复拣货台）：验证并修改订单拣货实时和 DB 数据，则发给拣货一个具有两命令的包，第一个命令时拣货结果，第二个命令时下个当前货架待拣商品的信息（名称，条码及库位）

23. A 拣货台：检查当前拣货商品是否是对应订单中最后一个商品

    a) 若已经完成订单，则订单箱红色高亮显示，表示订单完成

    b) 若否，则重复（16-22）直到 19，24，25 为标志的 B 时，小车就可以离开了

24. B 系统（回复拣货台）：验证并修改订单拣货实时和 DB 数据，如果一切正常，则发给拣货台拣货处理结果

25. B 拣货台（显示）：如果处理正常不显示，并将商品显示清空，若处理有错误则显示错误信息

26. Ba 小车（重复 9-25）：小车过去其他拣货台进行拣货，一直重复直到货架拣货完成，通过 Bb 回仓储区

27. Bb 小车（忙碌心跳）：回仓储区路上的小车，发送当前实时位置

28. Bb 系统（发给小车）：若有拣货台分到新订单的商品，在当前返回仓储区的货架上，则将对应拣货台设置终点，根据小车当前的实时位置作为起点，生成行走路线，作为去拣货台命令发给小车，然后小车重复（9-25），直到小车完成拣货再回去仓储区

29. Bb 小车（到达货架位置并放下货架）：小车到达货架时可以系统判断，但是否小车安全放下货架却无法知道，所以需要小车汇报

30. B 系统（发给小车）：小车放下货架则处于闲置状态，系统检查当前是否有新的待搬运货架

    a) 若有，则找到距离小车最近的货架，当前位置和目标货架作为起止点，将去取货架命令和对应路径发给小车，然后小车重复（6-29）

    b) 若无，则找到距离小车最近的闲置充电桩，当前位置和充电桩作为起止点，将去充电命令和对应路径发给小车

31. Bbb 小车（闲置心跳）：过去充电桩并实时发送心跳

32. Bbb 小车（到达充电桩）：告知系统

33. Bbb 系统（发给小车）：若小车过去充电桩路上或者已经在充电桩充电时，有新的订单来临需要搬运货架到拣货台，而当前小车是所有闲置小车中距离货架最近的，则将当前小车位置和目标货架作为起止点，发送去取货架命令和对应路径给小车，然后小车重复（6-29）

34. Aa 拣货员（双击订单箱）：红色高亮显示的订单箱表示订单已经拣货完成，双击表示打包员已经将订单箱拿走

35. Aa 拣货台（向系统申请订单）：打包员将完成拣货订单拿走后，拣货台自己触发去申请一个新订单

36. Aa 系统（回复拣货台）：系统读取当前未拣货订单，检查是否存在

    a) 若存在，则找一个订单（当前只是找了最老的），然后重复（3-35）

    b) 若不存在，则发空信息

待确定问题：

1. 对接原 OMS/WMS 数据（订单、产品）的方案

   关键点：

   - 订单信息：仅需要可以定位到商品即可（商品 ID），需要 OMS 提供接口获取订单对应的产品（通常商家可能不愿意提供订单的客户信息）
   - 产品所在库位信息：跟 WMS 系统绑定，或者用 WMS 中产品信息进行初始化
   - 库位、拣货台、充电桩及设备信息：独立导入/录入

2. 数据存储方案

   关键点：

   - 基础信息存储在持久数据库 Mysql 中（数据格式相对固定不考虑 Nosql，收费不考虑 Sqlserver，选择 Mysql）
   - 多个小车位置状态改变频繁，每秒更新 5 次小车位置，Mysql 可以胜任 100 台左右小车，若更多小车，则需要引入实时数据库 ExtremeDB（会有高并发不考虑内存数据库 Sqlite）

3. 对设备的路径规划及运动控制是设备独立实现，不采用中心调度

4. 入库/补货阶段如何通过商品二维码跟商品 SKU 做绑定