2019

面向自动仓储的智能调度仿真系统

祁佳薇 杨希希 胡纯浩

目录

第一章 系统需求规格说明书

1.1 引言

1.1.1 编写目的

需求分析与规约解决“做什么”的问题，输入用户原始需求，输出详细的需求规格说明书。

本面向自动仓储智能调度仿真系统的需求规格说明书（SRS）是在收集并分析仓库管理制度及仓库管理人员对于智能调度系统的需求后，为明确系统功能和非功能性需求、安排项目规划与进度、组织系统设计、实现与测试而撰写的，旨在于提高软件开发过程中的能见度，便于对整个系统开发过程的控制与管理。

1.1.2 预期读者和阅读建议

- 用户：进一步了解系统预期实现的功能，与分析人员再次讨论协商，及时变更信息
- 设计人员：进行系统设计
- 开发人员：根据需求规格说明书及后续设计工程输出的设计说明书进行系统实现
- 测试人员：编写测试用例，对系统功能和性能进行测试
- 项目经理：安排系统开发计划，进行项目管理

1.1.3 项目背景及意义

随着互联网+、物联网、移动互联网、大数据等新一代信息技术的发展，以智能化制造、两化深度融合为代表的新型制造方式终将引起工业界乃至社会发展的巨大变革。目前，制造业生产运输及物流业仓储物流等产业已经逐渐采用 AGV 自动运输来代替传统手工搬运和叉车搬运等方式，这些企业的生产物流订单大多具有周期短、批次多、单次运输量小、路径复杂等特点，因此，研发一套高效率、优化任务分配方案并能为多 AGV 规划出无冲突最短路径的 AGV 调度管理系统具有重要现实意义。

1.1.4 定义

用户（user）：一般指与系统直接交互的人员，这里指仓库负责管理的工人，可以通过监

视屏查看小车状态、发布命令（备料出料、送料等）。用户和客户一般不是同一类人

用例图（Use Case）：定义了系统的功能需求，是被称为参与者的外部用户所能观察到的系统功能模型图，展示了用例之间、用例与参与者之间的关系

时序图（Sequence Diagram）：一种 UML 行为图，通过描述对象之间发送消息的时间顺序显示多个对象之间的动态协作，每条消息对应状态机中引起转换的触发事件

类图（Class Diagram）：一种 UML 静态图，由模型元素（例如类、包）及之间关系组成

包图（Package Diagram）：一种 UML 结构图，模型是某一视点给定精度上对系统的完整描述，一个模型可以看作一个包，包图只显示包及其依赖关系，而不显示包内部的细节

1.1.5 参考资料

[1] 杨卫东 《2019 年软件工程课程项目——面向自动仓储的智能调度仿真系统》
[2] 钱乐秋 《软件工程（第 3 版）》 清华大学出版社
[3] 《GB/T 9385-2008 计算机软件需求规格说明规范》
[4] 《需求规格说明书（ISO 标准版）》
[5] 《需求规格说明书（RUP 版）》
[6] Alistair Cockburn 《编写有效用例（Writing Effective Use Cases）》 机械工业出版社

1.2 总体描述

1.2.1 用户特点

该系统的最终用户是仓库负责线上管理的管理员和仓库高层决策人员，具有较强的软件使用能力，能够快速熟悉绝大多数软件的基本操作和使用方法

系统的维护人员是计算机专业人员，对操作系统、数据库和网络维护的维护都很熟悉

1.2.2 目标与范围

“面向自动仓储的智能调度仿真系统”（简称“智能调度系统”）是一套操作简便、实用性强的仓库调度管理软件。给仓库管理人员和高层决策人员提供简单易懂的交互界面，包括两大功能模块：1）通过显示屏实时监视小车的各种状态信息来控制其执行仓库运输任务；2）通过按键输入任务，控制小车完成出料备料、送料、空货架调度、空货架回收

1.2.3 假定与约束

假定

- 可操作性：假定使用本系统的用户在经过一段时间熟悉之后，可以灵活地操作本网站来满足自己的需要。
- 用户支持：假定在本系统在开发的各个环节中得到用户的有效支持和配合。
- 技术支持：假定开发初期，小组成员充分认识本系统的需求，认真学好相关知识。开发过程中遇到技术问题，可以及时得到其他同学或者老师的指导与帮助。
- 人员配合：假定小组主要成员基本不会出现变动，并且在项目开发过程中不会因为突发情况的发生而导致项目成员无法正常参与开发工作。
- 时间限定：假定项目的截止时间不会提前。
- 需求限定：假定项目需求基本确定之后，不会有太大改变。

约束

- 人员约束： 团队成员均为复旦大学计算机专业大三学生，共 3 人。
- 管理约束：

  1. 本次开发，实行分工合作的模式进行。团队开发过程中按照进度表进行，开发过程中遇到的问题通过小组会议得到一致的解决。

  2. 小组成员首次合作，需要一个磨合过程，需要明确自身责任，分清各自的任务，互相配合，遇到问题项目经理必须能够有效进行协调，才能快速、有效地完成开发过程。
- 技术约束：

  1. 大部分小组成员在相关技术水平方面存在一定欠缺，缺乏相关项目经验。文档编制能力也有待提升。

  2. 小组成员在 UI 方面，能力有限，只有个别人员有前端开发能力。
- 时间约束： 本系统开发周期为三周，时间相对紧张。
- 其他约束： 由于在开发期间，小组成员处于期末季，还有其他科目的学习任务，将对项目进度造成不小的影响。

1.3 需求规定

- **用户原始需求描述**

- 面向自动仓储的智能调度系统是通过调度智能小车来为生产线配料，完成生产任务。
- 备料出料流程：工作人员（或其他自动化设备）在指定工位将待装配的工件及托盘放置在货架上，按下呼叫按钮，此时调度系统调度智能小车过来将分拣好的货架拉到缓存区。此后智能小车处于空闲状态可以执行后续任务。
- 送料流程：生产线体按下呼叫器发出叫料需求，调度系统指派小车去相应缓存库位托起货架送到生产线体呼叫处，满足线体生产，此后小车处于空闲状态可以执行后续任务。
- 空货架调度：缓存区按下空货架不足呼叫器时，调度系统指派处于空闲状态的小车到空货架区域拉取空货架到缓存区，放到指定位置后小车处于空闲状态可以执行后续任务。

- 空货架回收：线体产生空筐时，按下去空货架呼叫器，调度系统指派处于空闲状态的智能小车将空货架放回到空货架缓存区。充电：当智能小车处于空闲状态且电量不足时，调度系统指派智能小车到充电区域进行充电任务
- 产线任务单：生产线要完成的任务，包含时间、所需配料（配料分为 A、B、C 三种）
- 地图包含生产线等待区域（一个），配料缓存区域分为（A、B、C 个区域，分别存放三种零件）。不同区域通过铺有二维码的道路连接，建立地图模型，用于管理相关信息。
- 完成合理的智能调度算法、任务管理算法，并支持随时插入新的任务。
- 智能小车状态监测功能：通过界面可实时监测智能小车的各种状态信息，可用于控制其执行任务。
- 提供异常状态告警功能，需要处理的异常情况
  （1） 智能小车运动路径有偏移
  （2） 前方有障碍物，道路不通
  （3） 位置信息缺失
  （4） KIVA 心跳数据缺失
  （5） 智能小车的电量不足;
  （6） 智能小车任务过多
  （7） 智能小车任务途中抛锚

- **系统需求**

开发系统不是直接根据用户原始需求进行的，需要进行需求建模分析，形成系统需求，再依据系统需求进行设计实现。下面先列出表格、画出框图对系统需求进行总体概括，然后介绍需求建模的详细过程

表 1 系统需求

需求 ID	需求标题	需求内容	需求类型
SysR101	小车状态监控	用户可以通过显示屏实时监视小车状态信息	功能
SysR201	备料出料	从供料区运至缓存区	功能
SysR202	送料	从缓存区运至生产线	功能
SysR203	空货架调度	从空货架缓存区搬运空货架到缓存区	功能
SysR204	空货架回收	从生产线回收空货架到空货架缓存区	功能
SysR301	报警处理	充电/故障处理	功能
SysR401	性能好	精度、吞吐量、执行速度和时间	非功能
SysR402	运行用户体验好	页面布局、跳转	非功能
SysR403	系统稳定	可靠性、安全性	非功能

- **保证模型一致性**

在完成需求分析后首先要进行需求验证，由于需求的变化往往使系统的设计和实现也跟着改变，所以由需求问题引起系统做变更的成本比修改设计或代码错误的成本大得多，所以应及时更新验证需求。需求管理的方法是赋予每个需求唯一的标识符，如表格第一列所示，目的在于建立需求的跟踪表，每个跟踪表标示需求与其它需求或设计文档、代码、测试用例的不同版本间的关系。进行需求跟踪的目的是建立和维护从用户需求开始到测试完成之间的一致性和完整性。确保所有的实现是以用户需求为基础，所有的输出符合并且全面覆盖了用户需求。所以可以保证模型的一致性。

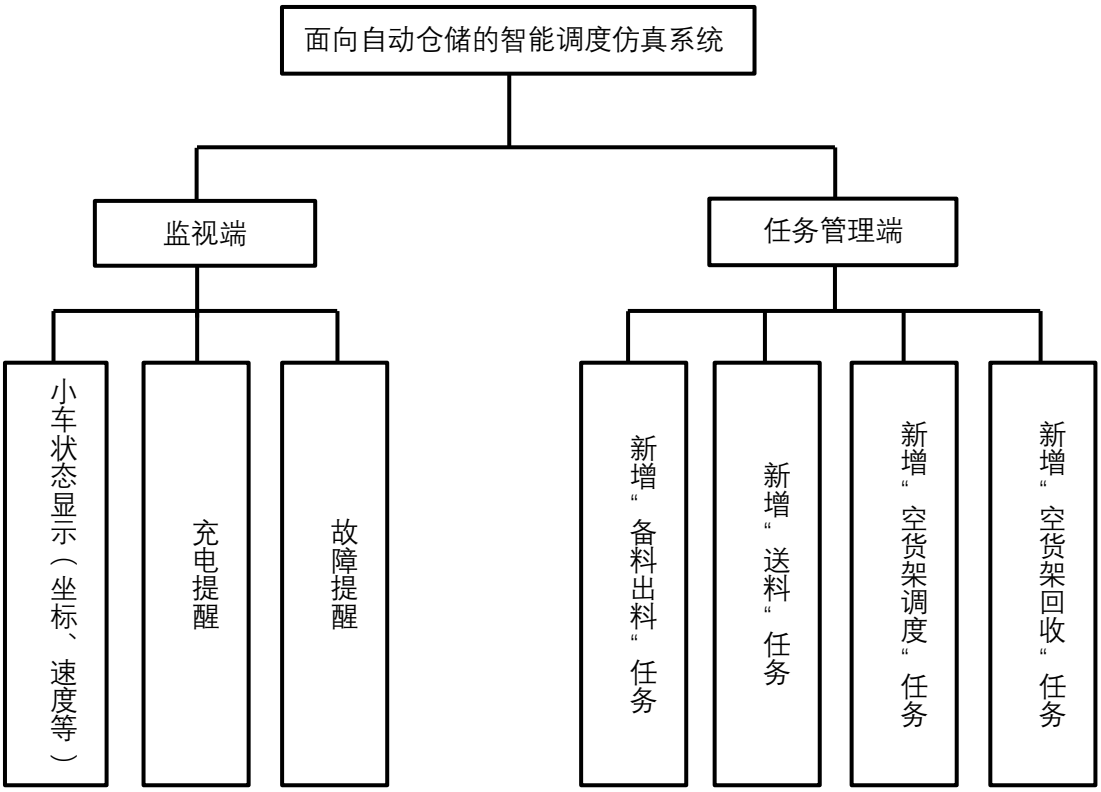


图 1 面向自动仓储的智能调度仿真系统结构图

1.3.1 功能性需求分析

1.3.1.1 用例建模

用况建模步骤如下:

1、识别执行者

执行者包括与系统交互的人或其他外部系统，本调度系统的执行者及其简单描述如下:

- 调度系统管理员：使用该系统的用户，可以监视小车和管理任务
- 高层决策人员：使用该系统的用户，可以进行仓库信息和地图信息的初始化
- 供料区系统：提供货物的工作人员或机器设备
- 生产线系统：需要货物完成生产单的工人或设备
- 缓存区系统：货物的中转站
- 空货架缓存区系统：空货架储存区
- 充电区系统：小车电量不足时的充电区
- 地图信息管理系统：记录地图信息的软件系统
- 故障处理系统：处理小车故障的硬件/软件系统

2、识别用况

- 仓库地图管理
- 仓库信息初始化
- 小车运行状态监测:
- 充电
- 故障处理
- 调度任务管理：插入、删除、修改任务（下面三个是子用况）
- 备料出料: 工作人员或其他自动化设备在指定工位将待装配的工件及托盘放置在货架上，按下呼叫按钮，调度小车将货架拉到缓冲区
- 送料：生产线按下呼叫按钮，调度小车去相应缓存库位将相应货架送到生产线呼叫处
- 空货架调度：缓存区空货架不足时小车从空货架区域搬运
- 空货架回收：生产线产生空货架时小车放回空货架缓存区

3、系统用例图

系统总体用例图如图所示

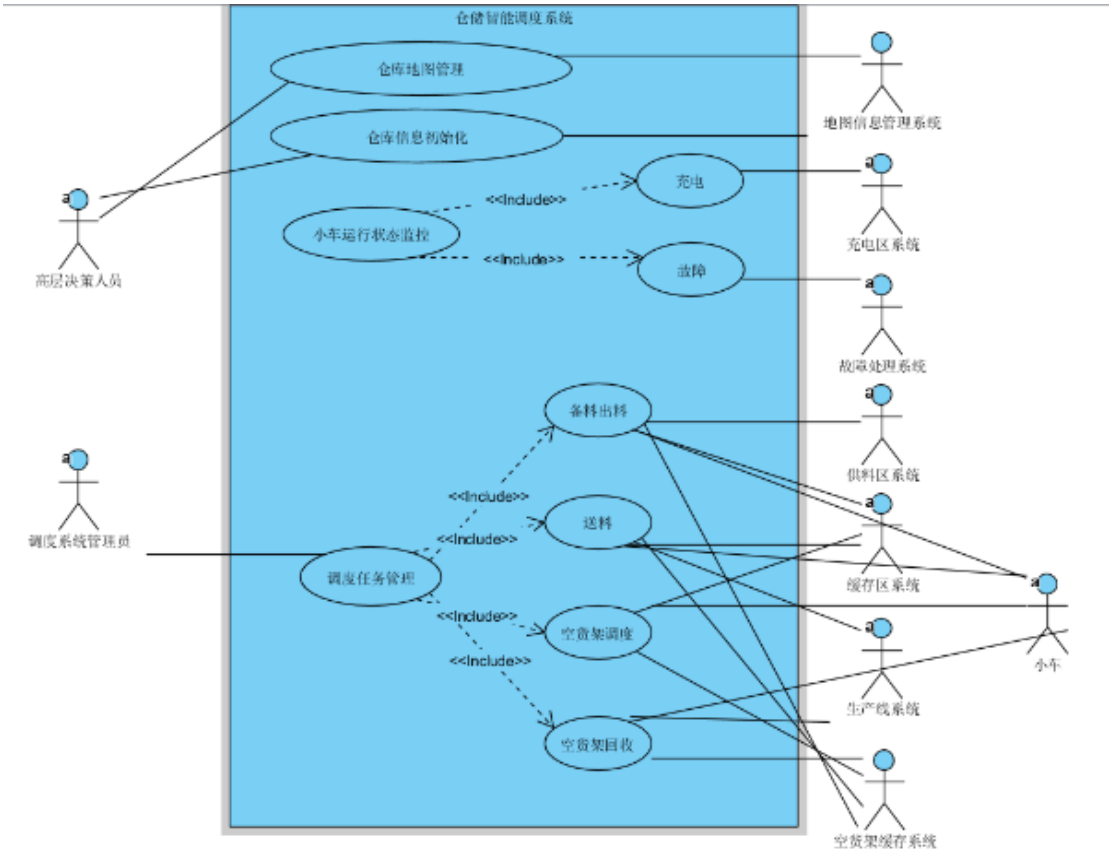


图 2 系统总体用例图

4、用例分析

针对关键用例进行详细文本描述（**RUP style**），并绘制出相应的动态模型——**活动图**

（1）备料出料

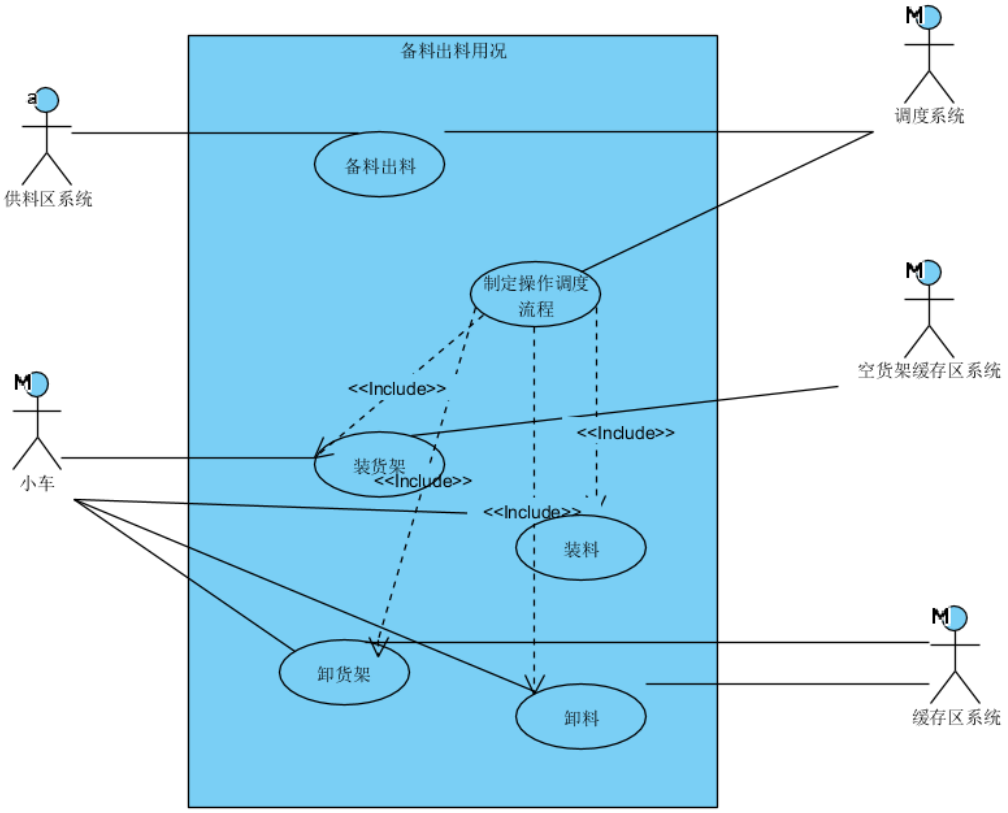


图 3 **备料出料用例图**

1. 用例名称：备料出料 1.1 简要描述 这个用例允许供货区工作人员（或其它自动化设备）呼叫小车进行备料出料。 1.2 参与执行者 供料区系统（主执行者） 小车、缓存区系统、空货架缓存区系统（次执行者） 1.3 触发事件 供料区工作人员（或其它自动化设备）按下呼叫按钮发出备料出料需求 2. 事件流程 2.1 基本流程 a. 当供料区工作人员（或其它自动化设备）在指定工位将待装配的工件及托盘放置在货架上，按下呼叫按钮时，用例开始执行 b. 调度系统选择优先级最高的小车进行调度；相同优先级的小车计算到供料点路径长度，选择时间开销最短的小车进行调度； c. 被选中的小车行驶至供料点 d. 小车装载分拣好的装有货物的货架

e. 调度系统对供料点到 A、B、C 三类缓存区的总路径进行规划

e. 小车依次行驶至三类缓存区

f. 小车卸载货物到缓料区的货架上

g. 小车进入空闲状态，用例结束

2.2 可选流程

a. 所有小车都正在行驶（执行任务），则需要等待有小车完成任务进入空闲状态

b. 在小车接收到任务后开始行驶之前，如果 A、B、C 三类缓存区没有空货架来盛放从供货区运来的货物，则需要调度小车先从空货架缓存区运送货架到缓存区

c. 任何时刻小车出现异常报警，都需要立刻终止任务，进入异常处理程序，调度系统调度另外的小车完成任务

3. 特殊需求

本用例无特殊需求

4. 前置条件

管理员登录系统，并选择新增任务功能

5. 后置条件

任务新增成功后写入任务表并进行调度分配和执行

6. 扩展点

无

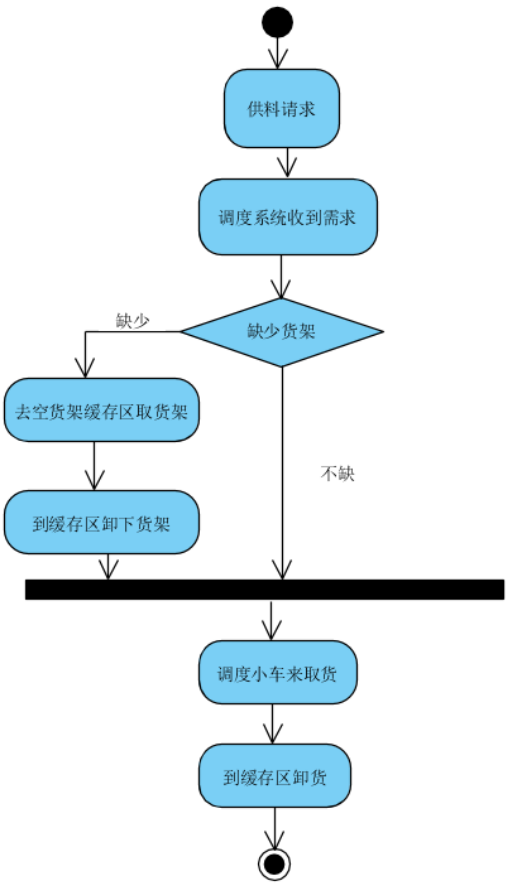


图 4 **备料出料活动图**

（2）送料

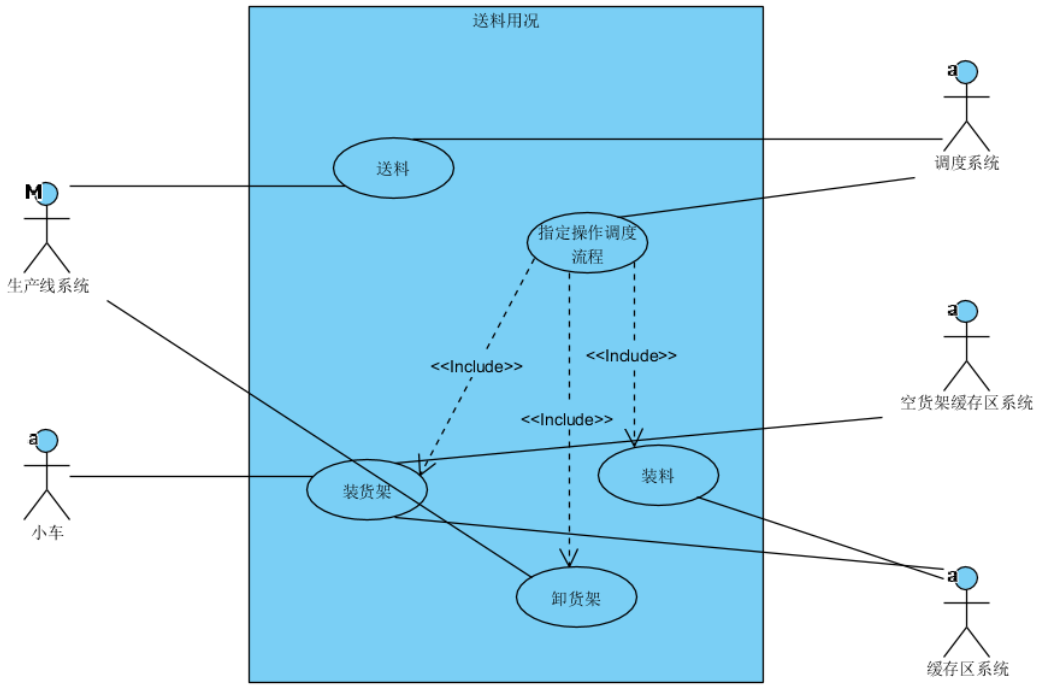


图 5 送料用例图

1. 用例名称：送料 1.1 简要描述 这个用例允许生产线体呼叫小车进行送料 1.2 参与执行者 生产线系统（主执行者） 小车、缓存区系统、空货架缓存区系统（次执行者） 1.3 触发事件 生产线按下呼叫按钮发出送料需求 2. 事件流程 2.1 基本流程 a. 当生产线体上某一位置按下来料呼叫器发出叫料需求时，用例开始执行 b. 调度系统选择优先级最高的小车进行调度；相同优先级的小车计算到货物所在缓存区地址的路径长度，选择时间开销最短的小车进行调度； c. 被选中的小车行驶至货物所在缓存区 d. 小车装载分拣好的装有货物的货架 e. 调度系统对从缓存区到叫料点的路径进行规划 e. 小车行驶至叫料点 f. 小车卸载装有货物的货架到生产线叫料点 g. 小车进入空闲状态，用例结束 2.2 可选流程

a. 所有小车都正在行驶（执行任务），则需要等待有小车完成任务进入空闲状态

b. 在小车接收到任务后开始行驶之前，如果 A、B、C 三类缓存区没有空货架来盛放一个架子上从缓存区取走要求数量的货物后剩下的货物，则需要调度小车先从空货架缓存区运送货架到缓存区

c. 任何时刻小车出现异常报警，都需要立刻终止任务，进入异常处理程序，调度系统调度另外的小车完成任务

3. 特殊需求

本用例无特殊需求

4. 前置条件

管理员登录系统，并选择新增任务功能

5. 后置条件

任务新增成功后写入任务表并进行调度分配和执行

6. 扩展点

无

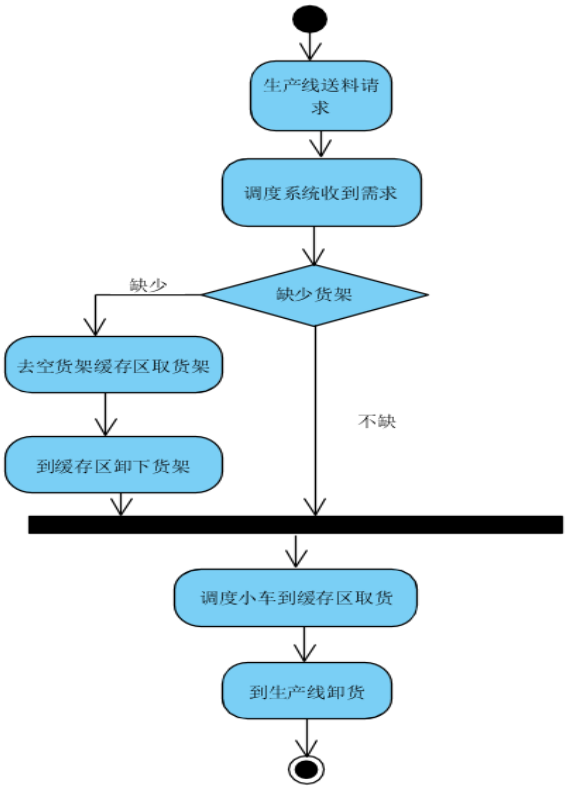


图 6 送料活动图

（3）空货架调度

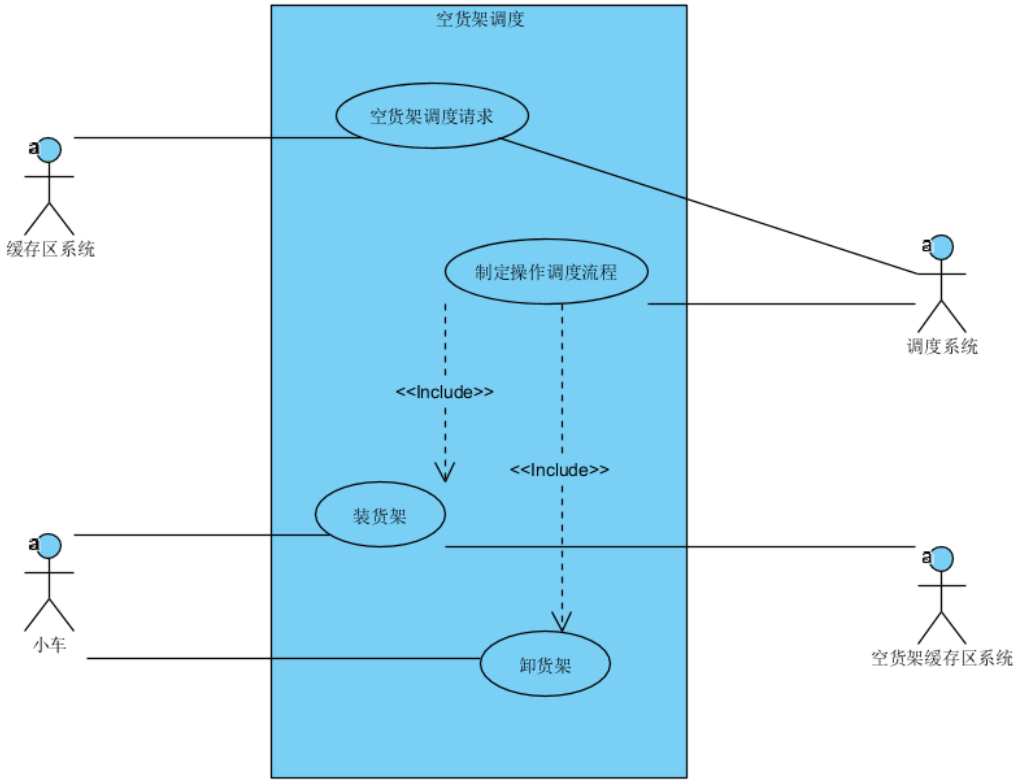


图 7 空货架调度用例图

1. 用例名称：空货架调度
   1.1 简要描述
      这个用例允许缓存区呼叫小车拉取空货架
   1.2 参与执行者
      缓存区系统（主执行者）
      小车、空货架缓存区系统（次执行者）
   1.3 触发事件
      缓存区系统按下空货架不足呼叫器
2. 事件流程
   2.1 基本流程
      a. 当缓存区系统按下空货架不足呼叫器时，用例开始执行
      b. 调度系统选择优先级最高的小车进行调度；相同优先级的小车计算到空货架缓存区的路径长度，选择时间开销最短的小车进行调度；
      c. 被选中的小车行驶至空货架缓存区
      d. 小车装载空货架
      e. 调度系统对空货架供货点到缓存区呼叫处的路径进行规划
      e. 小车行驶至缓存区呼叫处
      f. 小车卸载货架到缓存区呼叫处
      g. 小车进入空闲状态，用例结束
   2.2 可选流程

a. 所有小车都正在行驶（执行任务），则需要等待有小车完成任务进入空闲状态

b. 任何时刻小车出现异常报警，都需要立刻终止任务，进入异常处理程序，调度系统调度另外的小车完成任务

3. 特殊需求

本用例无特殊需求

4. 前置条件

管理员登录系统，并选择新增任务功能

5. 后置条件

任务新增成功后写入任务表并进行调度分配和执行

6. 扩展点

无

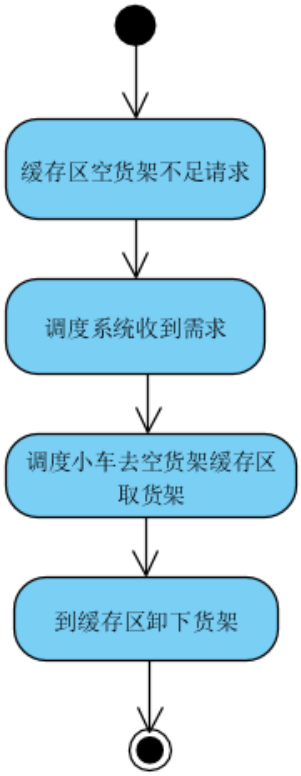


图 8 空货架调度活动图

（4）空货架回收

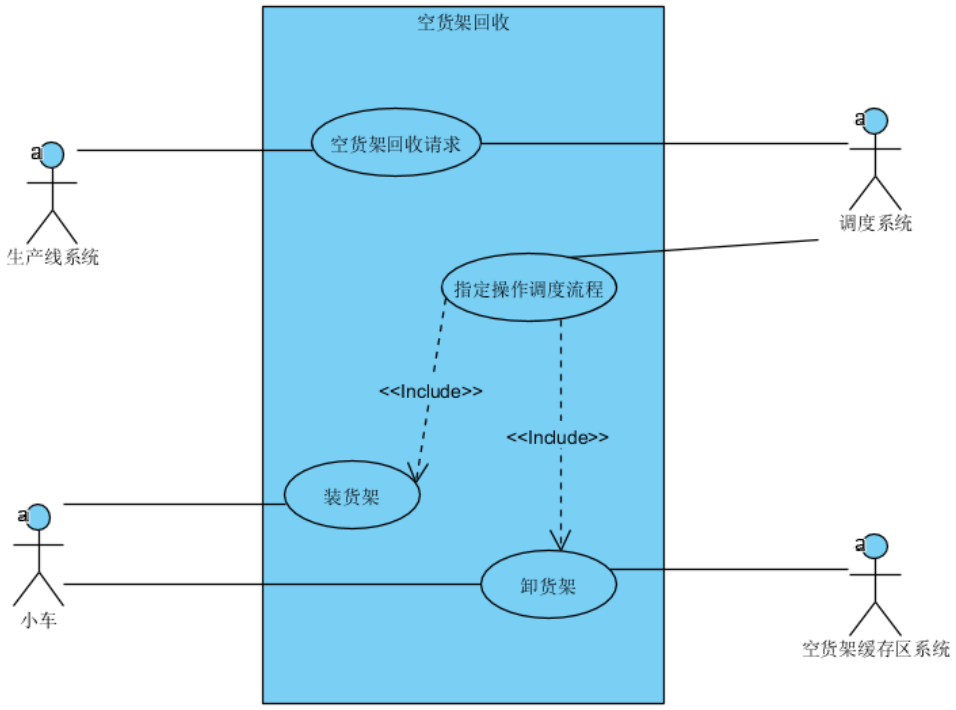


图 9 空货架回收用例图

1. 用例名称：空货架回收 1.1 简要描述 这个用例允许生产线呼叫小车进行空货架回收 1.2 参与执行者 生产线系统（主执行者） 小车、空货架缓存区系统（次执行者） 1.3 触发事件 生产线按下取空货架呼叫器 2. 事件流程 2.1 基本流程 a. 当生产线线体产生空架子时，按下取空货架呼叫器时，用例开始执行 b. 调度系统选择优先级最高的小车进行调度；相同优先级的小车计算到生产线呼叫处的路径长度，选择时间开销最短的小车进行调度； c. 被选中的小车行驶至生产线呼叫处 d. 小车装载空货架 e. 调度系统对生产线呼叫处到空货架缓存区的路径进行规划 e. 小车行驶至空货架缓存区 f. 小车卸载空货架到空货架缓存区 g. 小车进入空闲状态，用例结束 2.2 可选流程

a. 所有小车都正在行驶（执行任务），则需要等待有小车完成任务进入空闲状态

b. 任何时刻小车出现异常报警，都需要立刻终止任务，进入异常处理程序，调度系统调度另外的小车完成任务

3. 特殊需求

本用例无特殊需求

4. 前置条件

管理员登录系统，并选择新增任务功能

5. 后置条件

任务新增成功后写入任务表并进行调度分配和执行

6. 扩展点

无

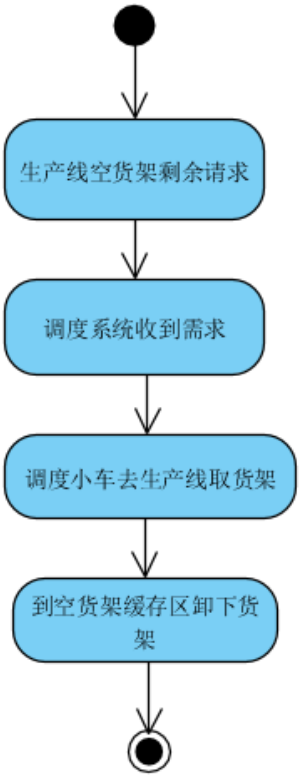


图 10 空货架回收活动图

（5）充电

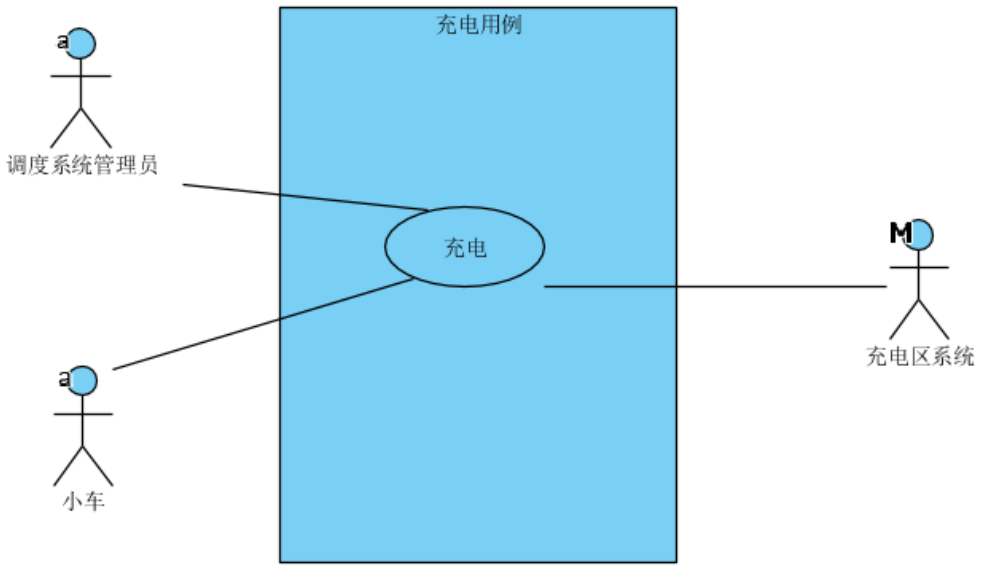


图 11 充电用例图

1. 用例名称：充电
   1.1 简要描述
      这个用例对处于空闲状态且电量不足的小车进行充电
   1.2 参与执行者
      调度系统管理员（主执行者）
      小车、充电区系统（次执行者）
   1.3 触发事件
      系统管理员监测到智能小车电量不足 10%
2. 事件流程
   2.1 基本流程
      a. 当调度系统管理员监测到智能小车电量不足 10%时，用例开始执行
      b. 当智能小车处于空闲状态时，调度系统对小车位置到充电区的路径进行规划
      c. 小车行驶至充电区
      d. 小车充满电后进入空闲状态，用例结束
   2.2 可选流程
      当电量不足 10%时，若小车正在执行任务，继续执行任务，然后前往充电区，如果中途电量变为 0，则报警异常，进入异常处理程序
3. 特殊需求
      本用例无特殊需求
4. 前置条件
      小车无故障，正处于空闲状态或任务执行状态
5. 后置条件
      无
6. 扩展点
      无

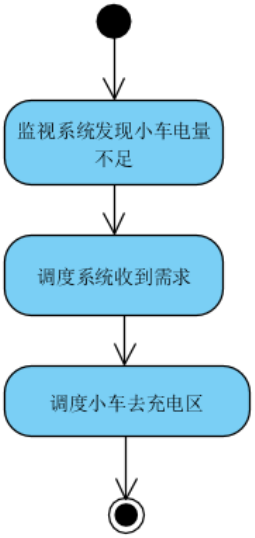


图 12 充电活动图

（6）小车运行状态监控

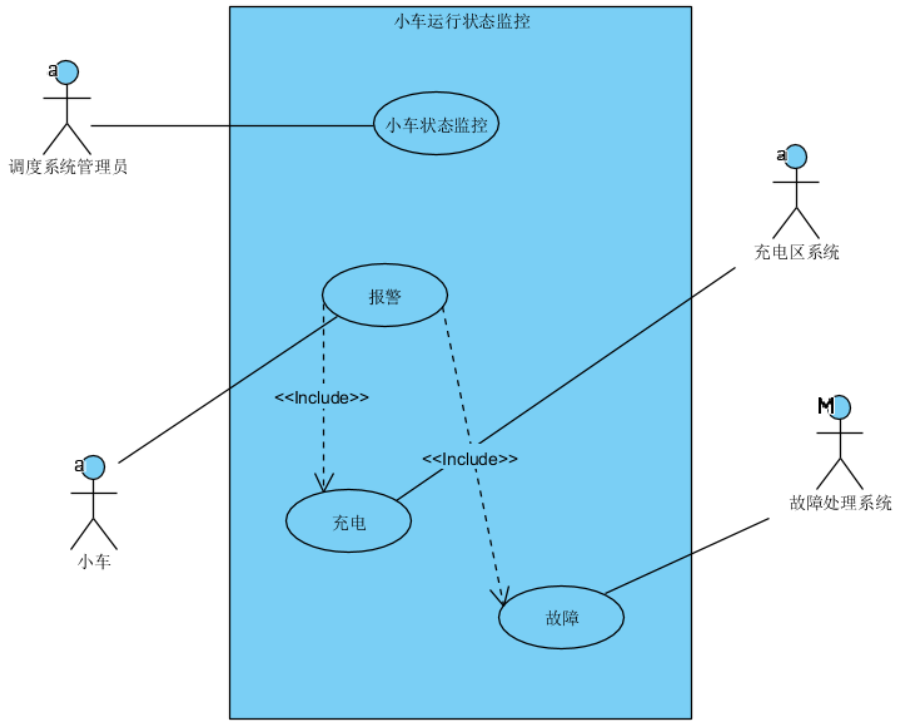


图 13 监视用例图

1. 用例名称：小车运行状态监控 1.1 简要描述 这个用例允许调度系统管理员对小车运行状态进行监控 1.2 参与执行者 调度系统管理员（主执行者）

小车、充电区系统、故障处理系统（次执行者）

1.3 触发事件

整个调度系统开启时

2. 事件流程

2.1 基本流程

a. 当调度系统开启时，用例开始执行

b. 当调度系统关闭时，用例结束

2.2 可选流程

a. 当小车电量不足时，进入充电程序

b. 当小车前方有障碍物（出现碰撞或道路不通）时，交由路径规划程序解决

c. 当小车运动路径有偏移时，交由故障处理系统解决

d. 当小车位置信息缺失时，交由故障处理系统解决

e. 当小车 KIVA 心跳数据缺失时，交由故障处理系统解决

f. 当小车任务过多时，交由故障处理系统解决

g. 当小车任务途中抛锚时，交由故障处理系统解决

3. 特殊需求

本用例无特殊需求

4. 前置条件

用户登录系统，并选择监视功能

5. 后置条件：无

6. 扩展点：无

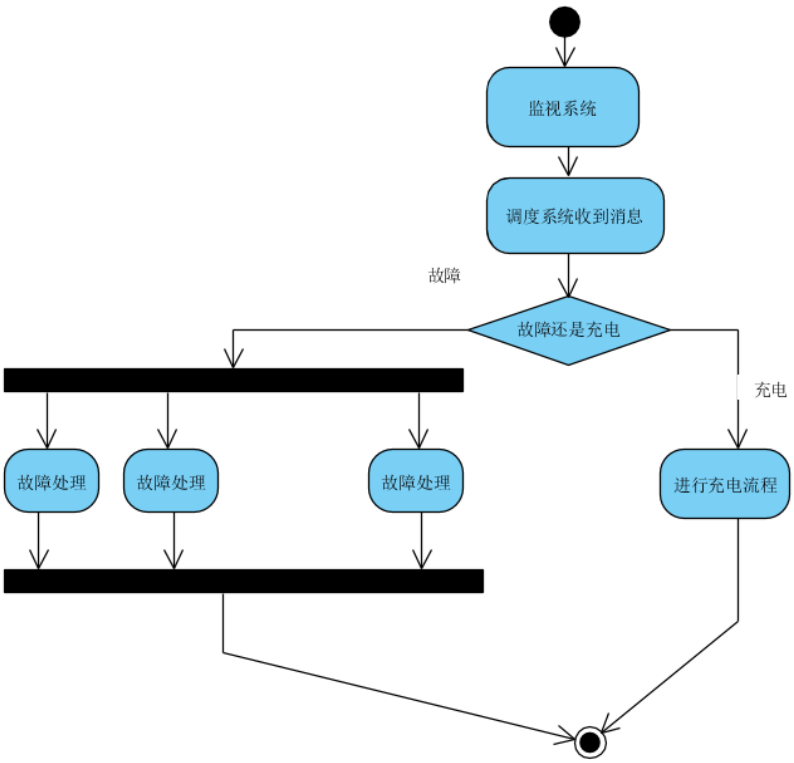


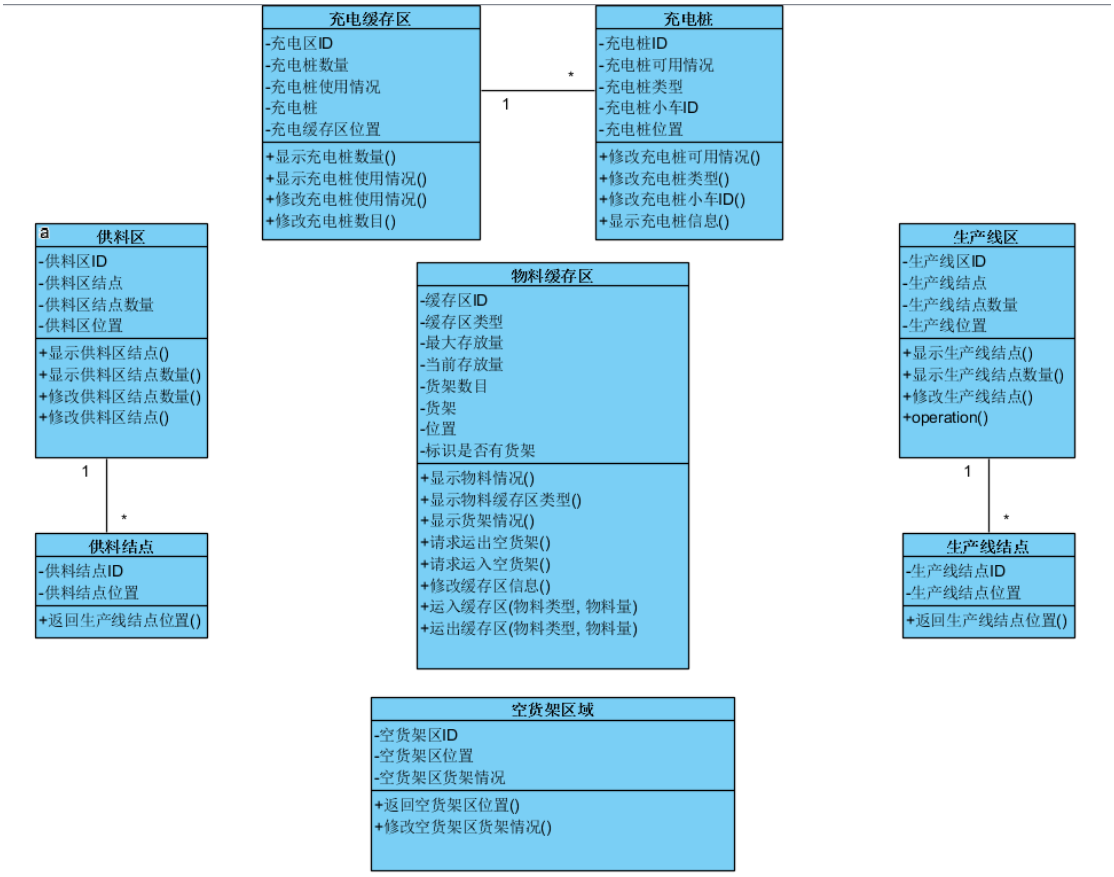
图 14 监视活动图

1.3.1.2 类模型

我们把类分为了几个不同的类型，分别是：地图类，实体类，人员类，系统类和规则类。下面会有对应的详细说明。

（1） 地图类

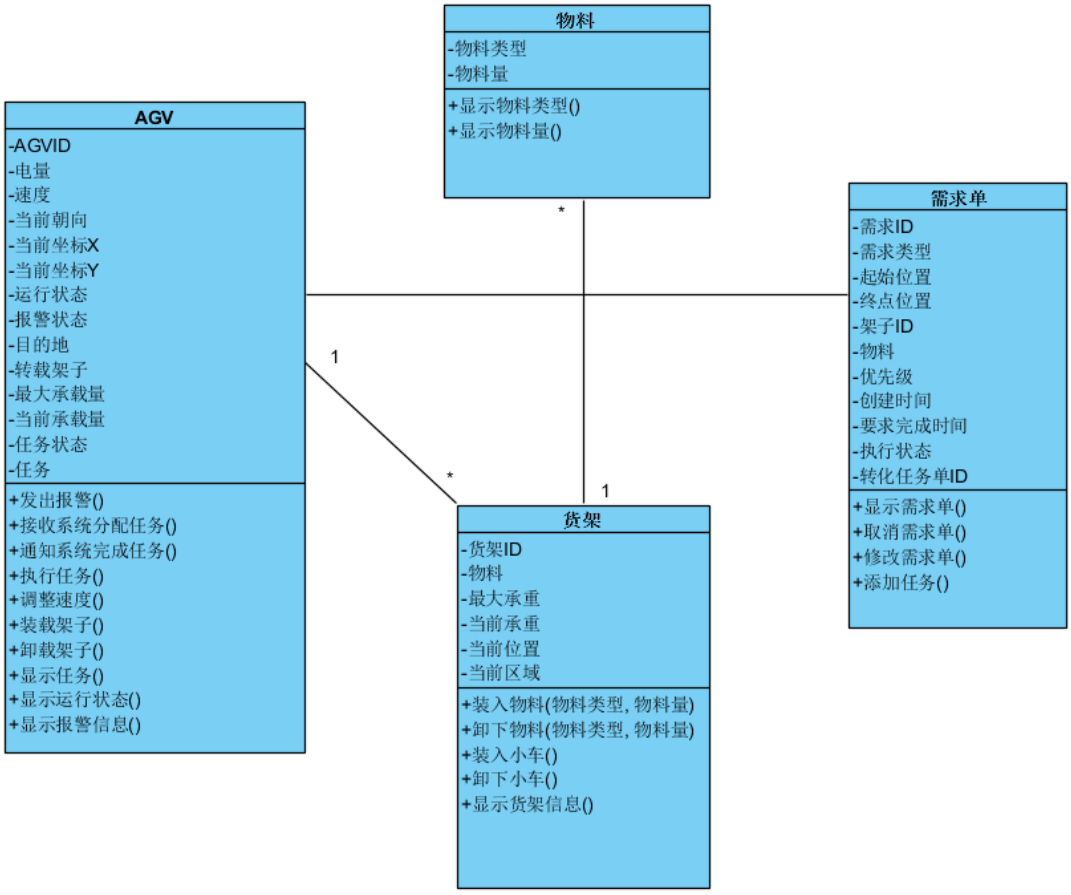
地图类包括了所有和工厂地区有关的类，元素都属于工厂中存在的区域或者设施。在地图上分区显示。这里包括了充电缓存区，充电桩，供料区、供料结点、物料缓存区、空货架区域、生产线区和生产线结点。其中的每一类都有位置属性作为表示，指定设备和区域在地图上的位置。



（2） 实体类

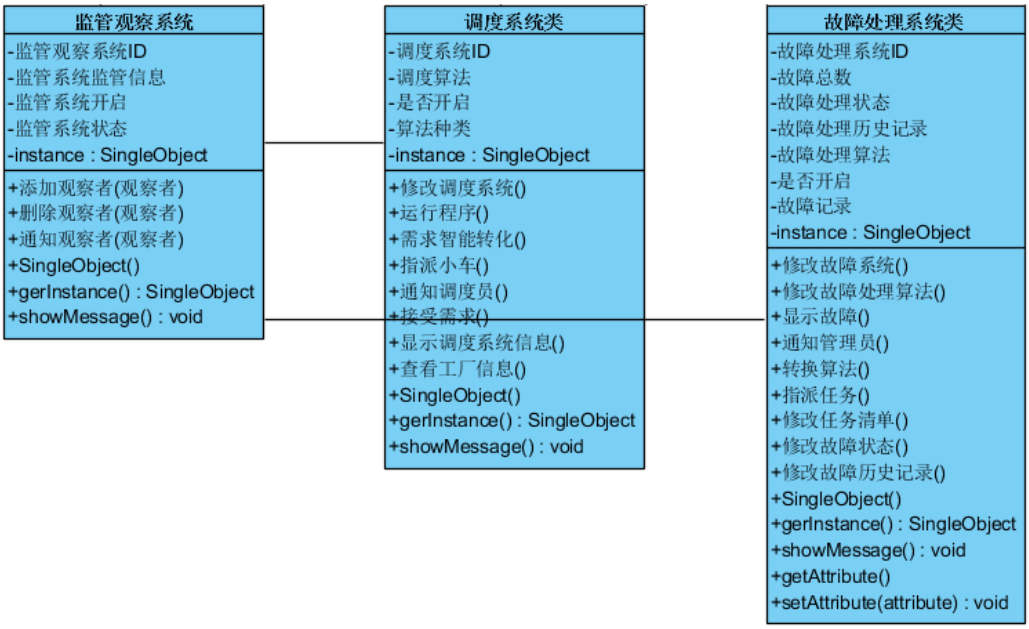
实体类主要是作为数据管理和业务逻辑处理层面上存在的类别；它们主要在分析阶段区分 实体类的主要职责是存储和管理系统内部的信息，它也可以有行为，甚至很复杂的行为，但这些行为必须与它所代表的实体对象密切相关。

实体类包含了系统中所有的实体，包括 AGV 类，物料类、需求单类和货架类。同时在这四个类中，详细定义了实体需要用到的操作。值得注意的是，我们使用方法来返回类中的属性信息，实现了属性和外界类的隔离，将一些重要的属性定为 private 属性，外界类无法直接修改属性，但是提供了一系列修改属性的方法，增加安全性。



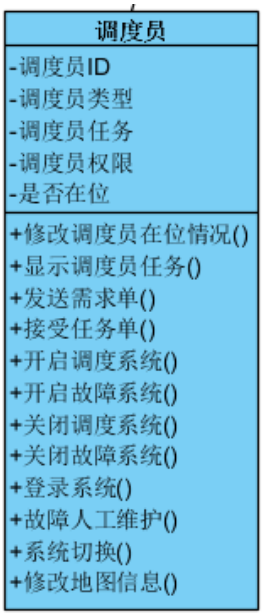
（3）系统类

将所有的系统类合为一个系统类，包含了监管观察系统，调度系统类和故障处理系统类。这里的系统是实现智能调度和智能异常处理的关键。包含智能调度算法和智能异常处理算法。可以解决工厂规划流程等一系列问题。可以被管理员开启，修改等。



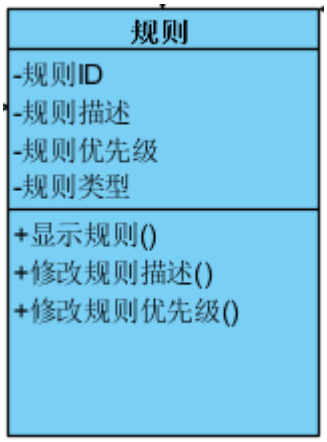
（4）人员类

人员类中有调度员类，将工厂中所有员工集成到调度员类中。调度员中分出很多不同的员工岗位，职能也不同。在调度员类中用调度员类型和调度员权限来区分。可以实现代码复用，优化管理模式。

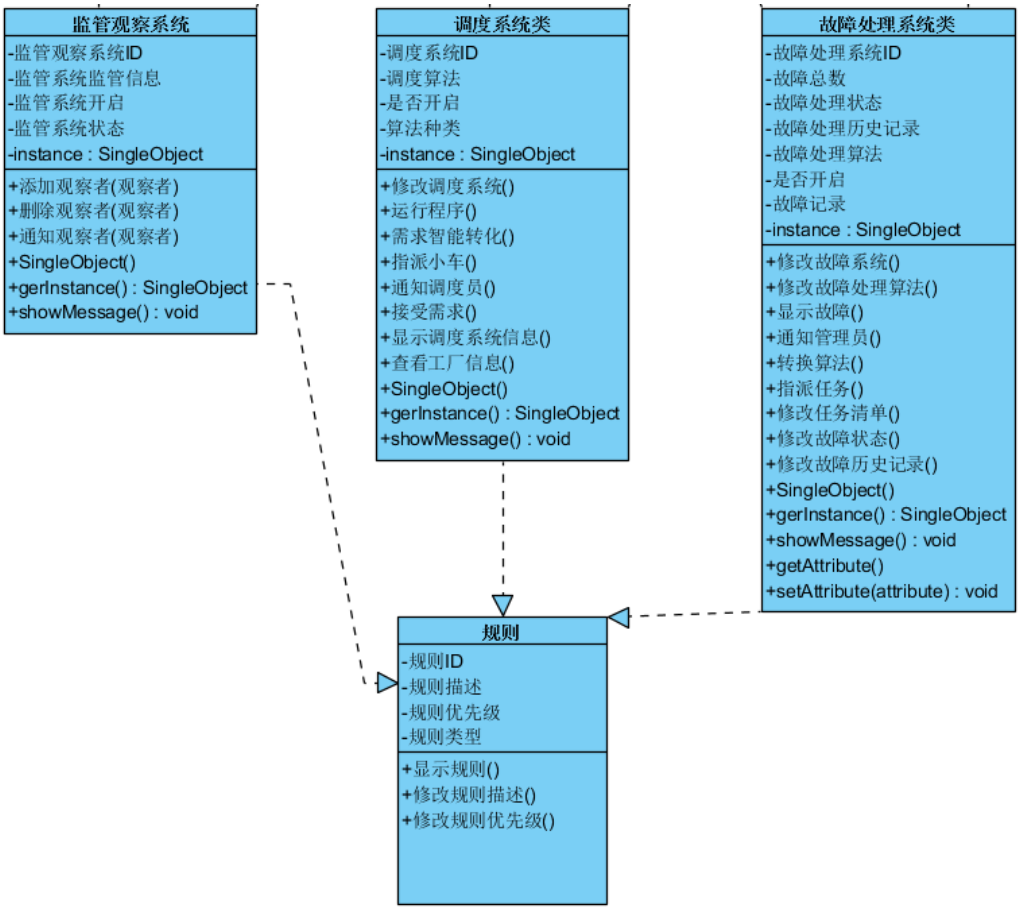


（5）规则类

规则类即集合了所有的规则，对于一个工厂，会制定很多规则，比如上下班时间，员工权限区分，AGV 规则等。通过规则 ID 和规则描述相区分。

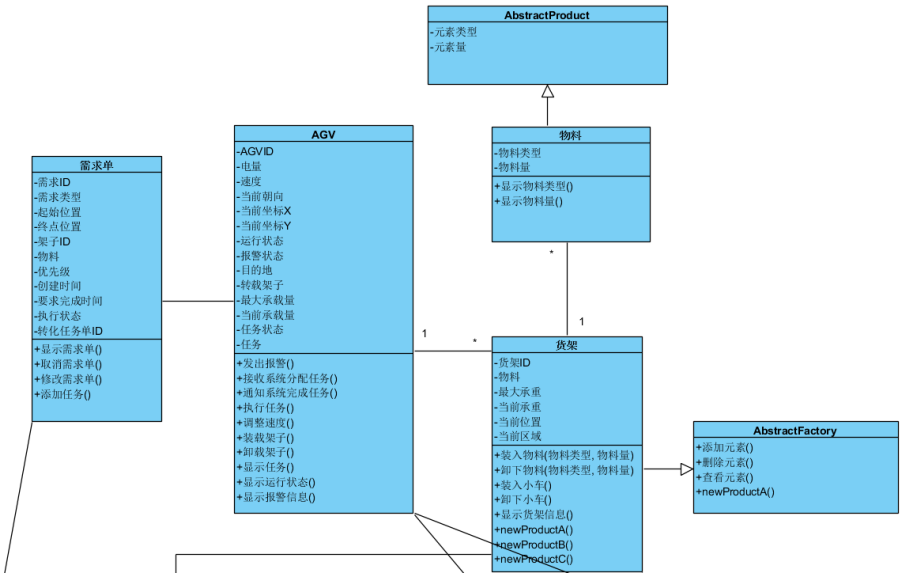


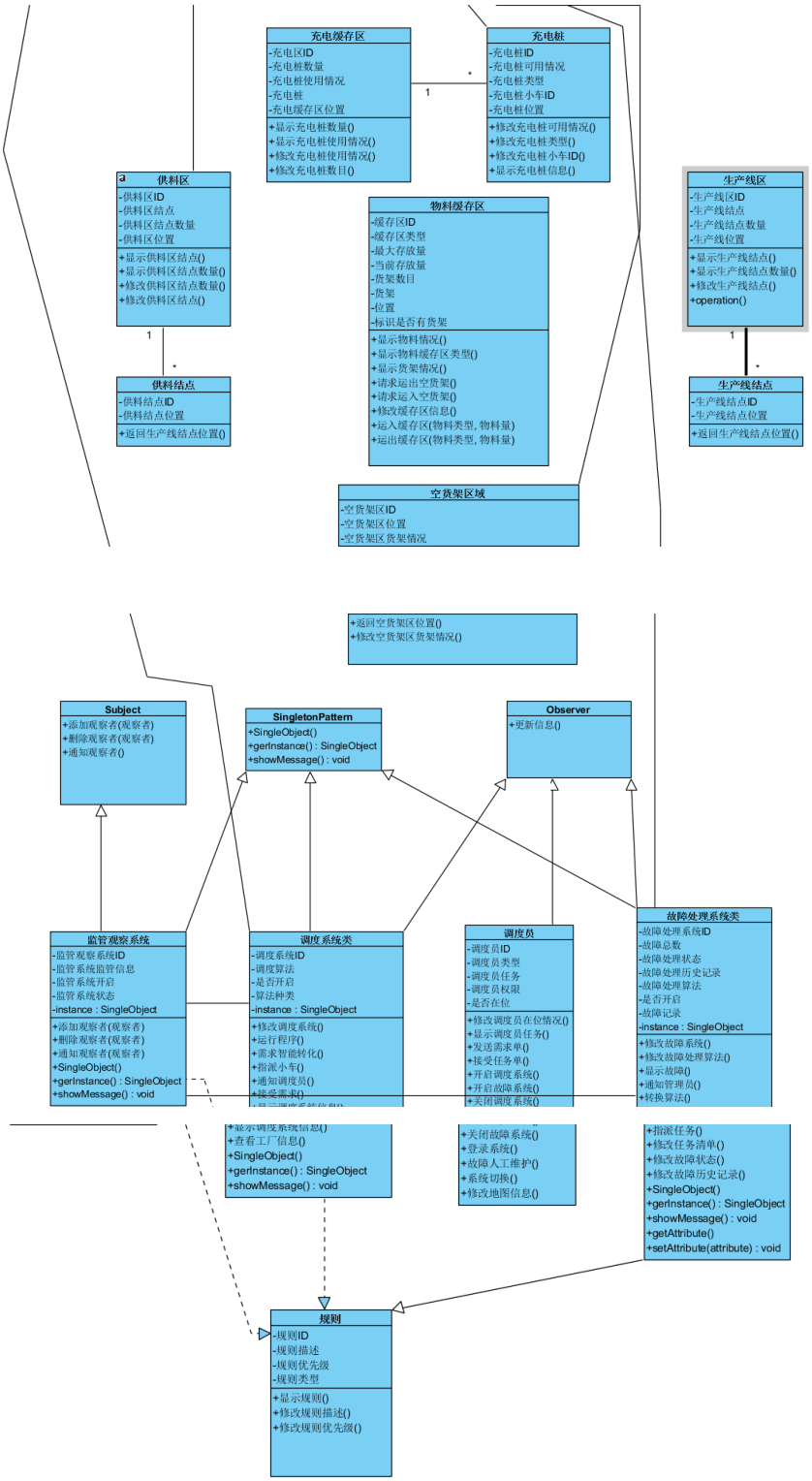
其中值得注意的是规则类的实现（Realization）：



（6）总图

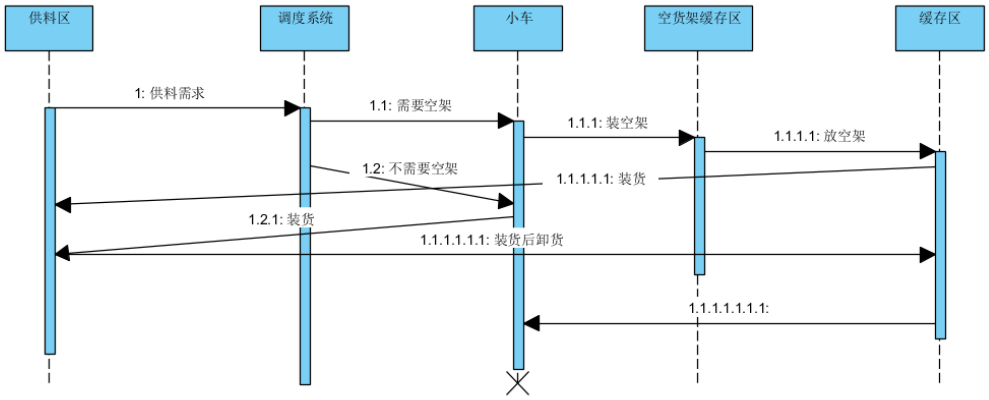
将各个类的关系定义之后，可以形成总图




充电缓存区
-充电区ID
-充电桩数量
-充电桩使用情况
-充电桩
-充电缓存区位置
+显示充电桩数量()
+显示充电桩使用情况()
+修改充电桩使用情况()
+修改充电桩数目()
充电桩
-充电桩ID
-充电桩可用情况
-充电桩类型
-充电桩小车ID
-充电桩位置
+修改充电桩可用情况()
+修改充电桩类型()
+修改充电桩小车ID()
+显示充电桩信息()
供料区
-供料区ID
-供料区结点
-供料区结点数量
-供料区位置
+显示供料区结点()
+显示供料区结点数量()
+修改供料区结点数量()
+修改供料区结点()
物料缓存区
-缓存区ID
-缓存区类型
-最大存放量
-当前存放量
-货架数目
-货架
-位置
-标识是否有货架
+显示物料情况()
+显示物料缓存区类型()
+显示货架情况()
+请求运出空货架()
+请求运入空货架()
+修改缓存区信息()
+运入缓存区(物料类型, 物料量)
+运出缓存区(物料类型, 物料量)
生产线区
-生产线区ID
-生产线结点
-生产线结点数量
-生产线位置
+显示生产线结点()
+显示生产线结点数量()
+修改生产线结点()
+operation()
供料结点
-供料结点ID
-供料结点位置
+返回生产线结点位置()
生产线结点
-生产线结点ID
-生产线结点位置
+返回生产线结点位置()
空货架区域
-空货架区ID
-空货架区位置
-空货架区货架情况
+返回空货架区位置()
+修改空货架区货架情况()
Subject
+添加观察者(观察者)
+删除观察者(观察者)
+通知观察者()
SingletonPattern
+SingleObject()
+gerInstance() : SingleObject
+showMessage() : void
Observer
+更新信息()
监管观察系统
-监管观察系统ID
-监管系统监管信息
-监管系统开启
-监管系统状态
-instance : SingleObject
+添加观察者(观察者)
+删除观察者(观察者)
+通知观察者(观察者)
+SingleObject()
+gerInstance() : SingleObject
+showMessage() : void
调度系统类
-调度系统ID
-调度算法
-是否开启
-算法种类
-instance : SingleObject
+修改调度系统()
+运行程序()
+需求智能转化()
+指派小车()
+通知调度员()
调度员
-调度员ID
-调度员类型
-调度员任务
-调度员权限
-是否在位
+修改调度员在位情况()
+显示调度员任务()
+发送需求单()
+接受任务单()
+开启调度系统()
+开启故障系统()
故障处理系统类
-故障处理系统ID
-故障总数
-故障处理状态
-故障处理历史记录
-故障处理算法
-是否开启
-故障记录
-instance : SingleObject
+修改故障系统()
+修改故障处理算法()
+显示故障()
+通知管理员()
+转换算法()
+查看工厂信息()
+SingleObject()
+gerInstance() : SingleObject
+showMessage() : void
+关闭故障系统()
+登录系统()
+故障人工维护()
+系统切换()
+修改地图信息()
+指派任务()
+修改任务清单()
+修改故障状态()
+修改故障历史记录()
+SingleObject()
+gerInstance() : SingleObject
+showMessage() : void
+getAttribute()
+setAttribute(attribute) : void
规则
-规则ID
-规则描述
-规则优先级
-规则类型
+显示规则()
+修改规则描述()
+修改规则优先级()

1.3.1.3 时序模型

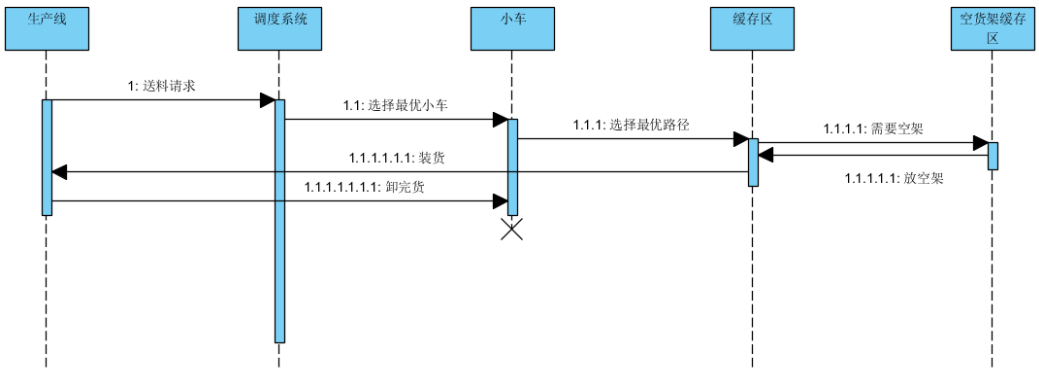
（1）备料供料



供料流程：

供料区完成生产配料，发出供料的需求。需求信号发送给调度系统，调度系统经过一系列的决策，调度和计算（在文章后续会详细阐述），计算出一个任务数组。任务数组中包含对空货架的操作和对物料的操作。调度系统判断缓存区是否需要空货架。若需要，则调度小车先去空货架缓存区拿货架，送到缓存区。若不需要，则直接调度小车来供料区装货。装完货后小车根据路径调度送料到缓存区。最后小车停在缓存区的对应位置，小车的状态转为空闲。

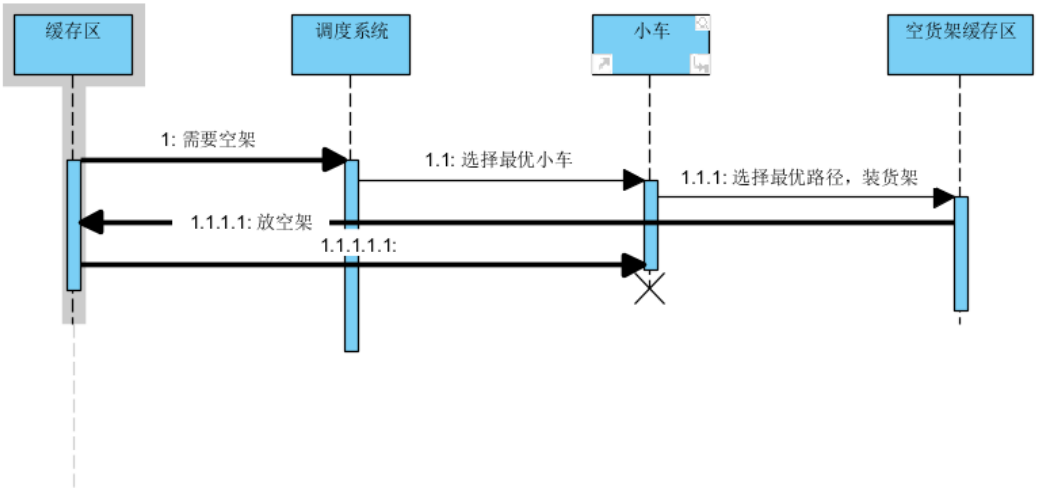
（2）送料



送料流程：

当生产线需要某种物料时，会发送一个送料请求的需求给调度系统。调度系统根据路径调度和任务管理，同样判断是否需要空货架。若需要，则调度一辆最优小车去空货架缓存区取空货架并放到缓存区。若不需要或已经放好空货架，则由调度系统根据调度原则和策略，选择小车和最优路径，由从缓存区到生产线的最优路径装货和卸货，最后小车进入空闲状态。

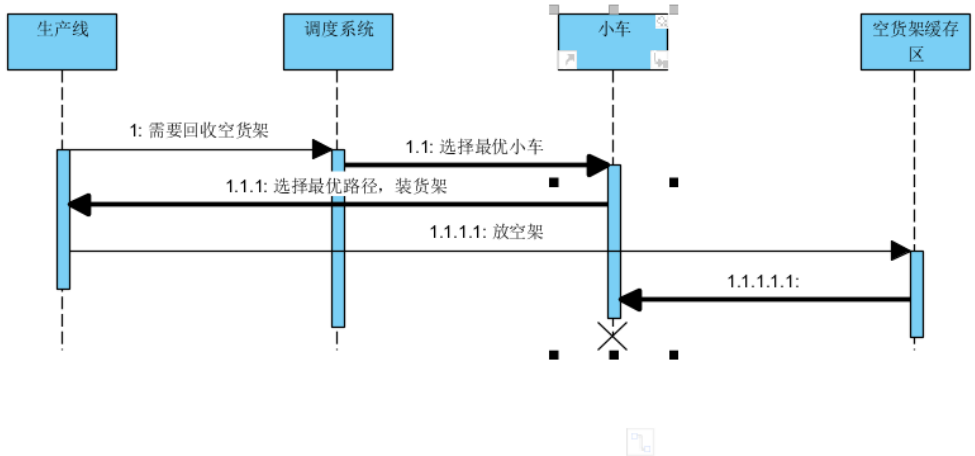
（3）空货架调度



空货架调度流程:

当计算出缓存区系统的空货架不足时发送一个需要空架的请求，调度系统根据调度算法和规则进行调度；选出最优的小车，计算出最优路径，被选中的小车行驶至空货架缓存区装货架小车装载空货架后调度系统对空货架供货点到缓存区呼叫处的路径进行规划。小车行驶至缓存区呼叫处，小车在缓存区对应位置卸载货架，小车进入空闲状态，流程结束

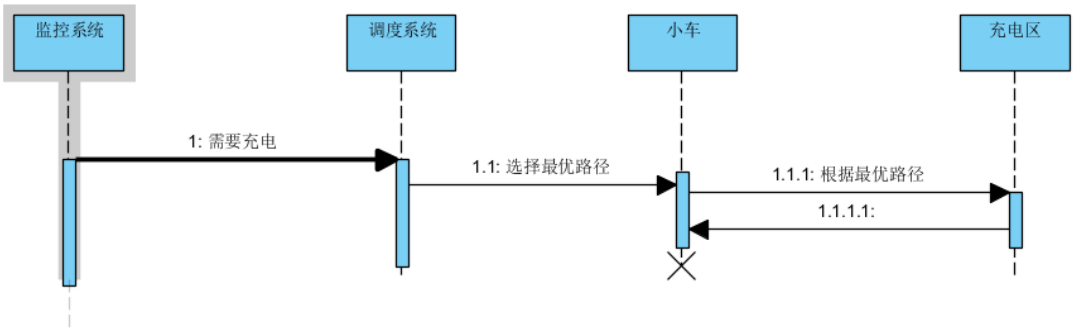
（4）空货架回收



空货架回收流程:

当生产线系统产生空架子时，会发送一个需要回收空货架请求，调度系统根据调度规则和调度策略，选择最优的小车进行调度，同时给出从生产线空货架处到空货架缓存区的最优路径。，被选中的小车行驶至生产线呼叫处，装载空货架。随后小车根据最优路径行驶至空货架缓存区，小车在空货架缓存区卸载空货架。最后小车进入空闲状态，流程结束

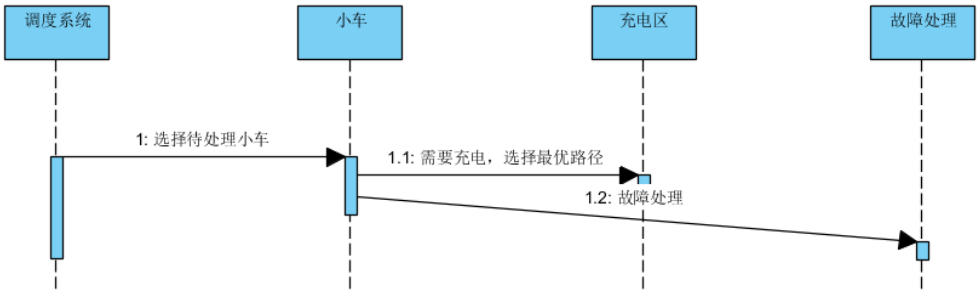
（5）充电



充电流程：

监控系统会实时将小车状态传达给调度系统，此时若小车电量不足，监控系统会给调度系统发送充电请求。调度系统根据调度策略和规则，为该小车计算出最优的路径。小车根据最优的路径安排小车到达充电区充电，此时小车进入空闲状态。流程结束。

（6）监控



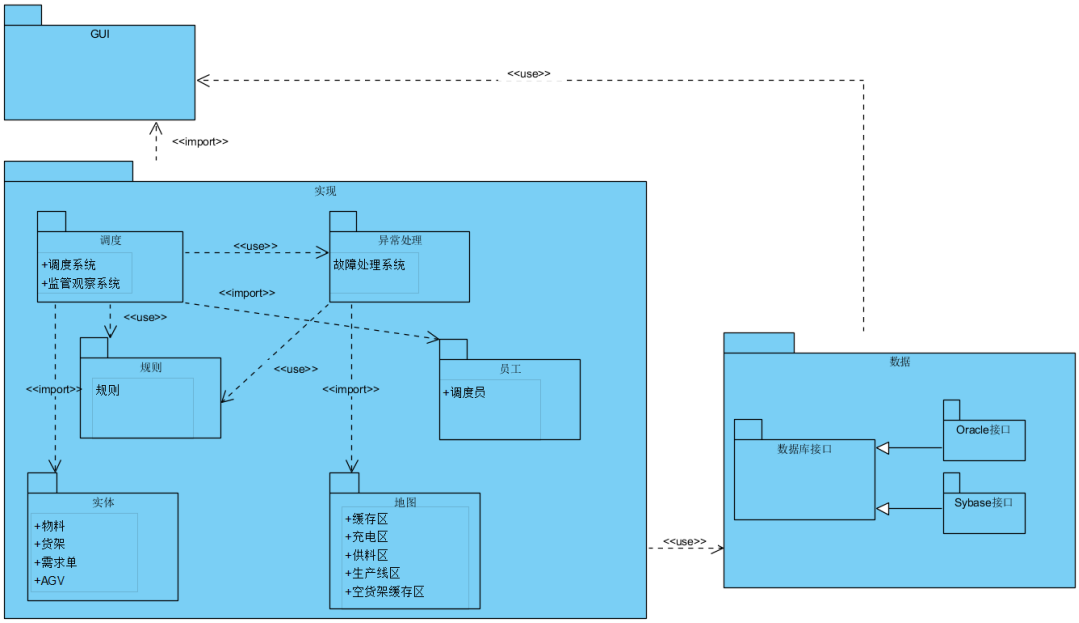
监控流程：

监控系统会实时将小车状态传达给调度系统，此时若小车电量不足，监控系统会通过上面的时序图安排小车去充电区充电。若是发生了故障，调度系统会根据故障处理的规则和策略，为对应小车选择故障处理的方式，故障处理的方式会在后续详细说明。

1.3.1.4 包模型

包图是在 UML 中用类似于文件夹的符号表示的模型元素的组合。系统中的每个元素都只能为一个包所有，一个包可嵌套在另一个包中。使用包图可以将相关元素归入一个系统。一个包中可包含附属包、图表或单个元素。

我们将包图分为三大类，分别是 GUI，实现和数据。其中 GUI 体现用户界面如何实现，实现是整个系统的核心，包括了工厂元素和所有操作，数据中包含处理数据的一系列方法。



1.3.2 性能需求

（1）精度

及时更新整理数据，保证仓库地图、货物、任务等的安全性和准确性；由于本系统中查询、增添、删除的关键字只能为元素 ID，不存在精确查找或是泛型查找的区别

（2）容量、吞吐量及时间特性要求

不同规模仓库对于供料区、生产线、缓存区等所需工人、机器、货架容量不同，需要灵活调整；根据对工厂仓库调度系统实际应用需求的调查，本系统应至少支持 10 路用户端并发访问和任务输入（调度系统管理员、决策人员），每日数据和信息的吞吐量保证在 1000 条以上，实时监测小车状态的更新时间应控制在 1 秒以内包括 1 秒，新任务布置的平均响应时间不大于 3 秒

（3）灵活性

当用户需求有改变时，比如运行环境的变化、角色及操作方式的变化，系统能够灵活配置，满足用户需要

1.3.3 运行需求

用户界面的设计首先保证在规定的输入输出下功能实现的完整性；其次保证用户体验度高，可以快速上手，能够在不需要太多培训情况下使用本系统：

（1）首页显示两大模块：小车状态实时监测、调度任务管理

（2）小车状态监视界面：显示整个仓库和小车的布局，要符合日常仓库实际情况

（3）调度任务管理界面：

备料出料输入：供料区呼叫点、供料类型和供料量

送料输入：生产线呼叫点、缺料类型和缺料量

空货架调度输入：缓存区缺少空货架位置、数量

空货架回收输入：生产线产生空货架位置、数量

四个功能的输出：小车的行驶路径、装载卸下货物架子通知等

（4）用户点击的层级越少越好，最好直接进入主题页面

（5）网页个页面在操作方式、布局及视觉风格上要统一

1.3.4 其它非功能性需求

（1）可靠性

在正常的网络环境和软硬件配置环境下，能够高性能灵活地不间断稳定运行，满足用户地功能性需求；在特殊情况下（火灾、地震等自然灾害），仓库智能调度系统需要保证不间断运行且不会出现故障；一旦出现意外情况，系统数据要做到不会损坏和丢失，系统应该自动保存，生成错误日志，不影响用户查看

（2）安全性

系统需要有自动报警提示功能，以防人工误操作造成系统数据错误或者造成数据匹配错误，该功能使客户端使用人员可以在第一时间发现错误并做出调整；仓库的数据属于机密信息需要保密，通信、程序、网络三级权限口令管理使网络系统必备功能

第二章 系统设计规格说明书

2.1 引言

2.1.1 编写目的

设计与规约解决“怎么做”的问题，输入需求规格说明书，输出设计规格说明书。

本面向自动仓储智能调度仿真系统的设计规格说明书（SDS）的编写目的在于，将上阶段对用户的需求分析升级为对实现这些需求的功能进行体系结构设计，将原本比较复杂的系统划分为一个个模块，来确定这些模块的功能以及模块间的调用关系，以供后续实现。

2.1.2 预期读者及阅读建议

1）开发人员：可以参考详细设计说明书编写程序

2）测试及维护人员人员：在发现系统问题时，可以参考详细设计说明书了解各个模块的实现，提出修改意见，完善系统设计

3）项目经理：根据设计规格说明书跟进项目进度并安排系统开发计划

2.1.3 定义

- 设计模式（Design Pattern）：设计模式是一套可以被反复使用的、多数人知晓的、经过分类编目的、代码设计经验的总结，使用设计模式是为了可重用代码、让代码更容易被他人理解并且提高代码的可靠性。
- 部件图（Component Diagram）：是一种用于物理体系结构建模的图，又称为构件图，描述的是在软件系统中遵从并实现一组接口的物理的、可替换的软件模块
- 部署图（Deployment Diagram）：也是一种用于体系结构建模的图，展示了运行时处理结点和在结点上生存的制品的配置

2.1.4 参考资料

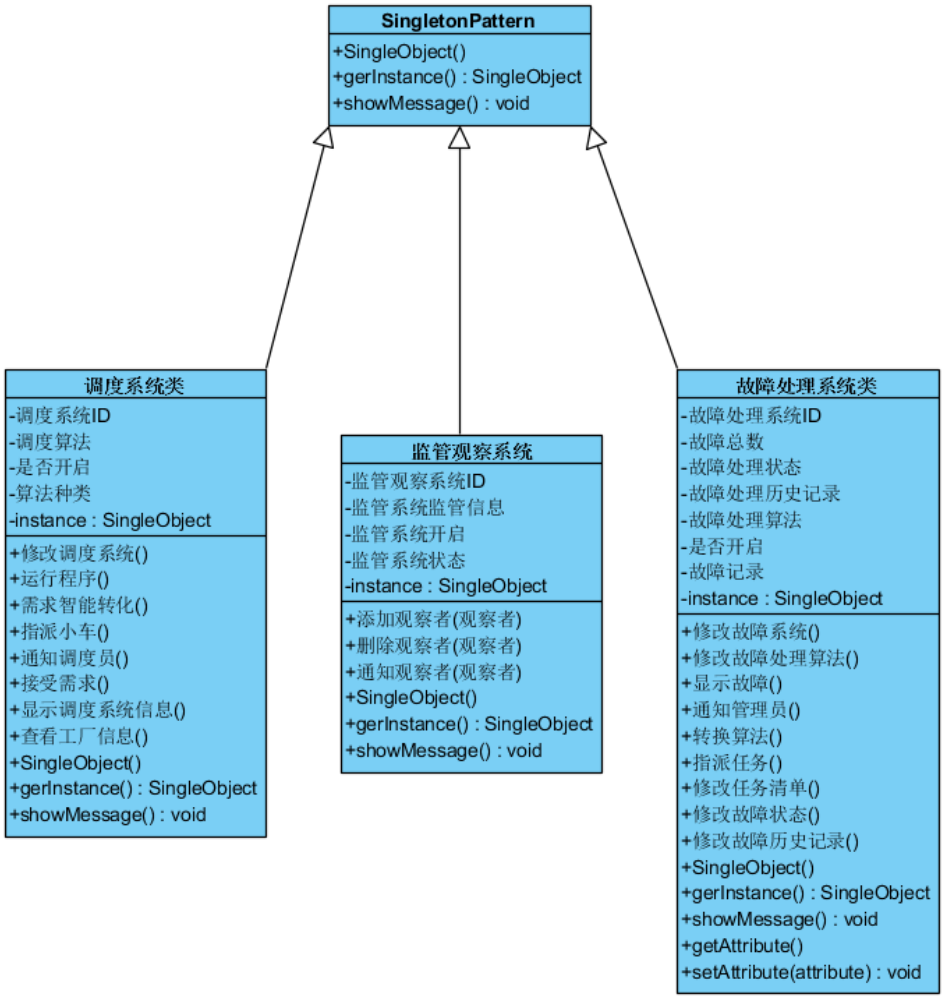
[1] 杨卫东 《2019 年软件工程课程项目——面向自动仓储的智能调度仿真系统》
[2] 钱乐秋 《软件工程（第 3 版)》 清华大学出版社
[3] 《GB8567-88 软件设计文档国家标准》
[4] Erich Gamma / Richard Helm / Ralph Johnson / John Vlissides 《设计模式：可复用面向对象软件的基础（Design Patterns: Elements of Reusable Object-Oriented Software)》 机械工业出版社

2.2 系统设计

2.2.1 设计模式介绍及完善包模型

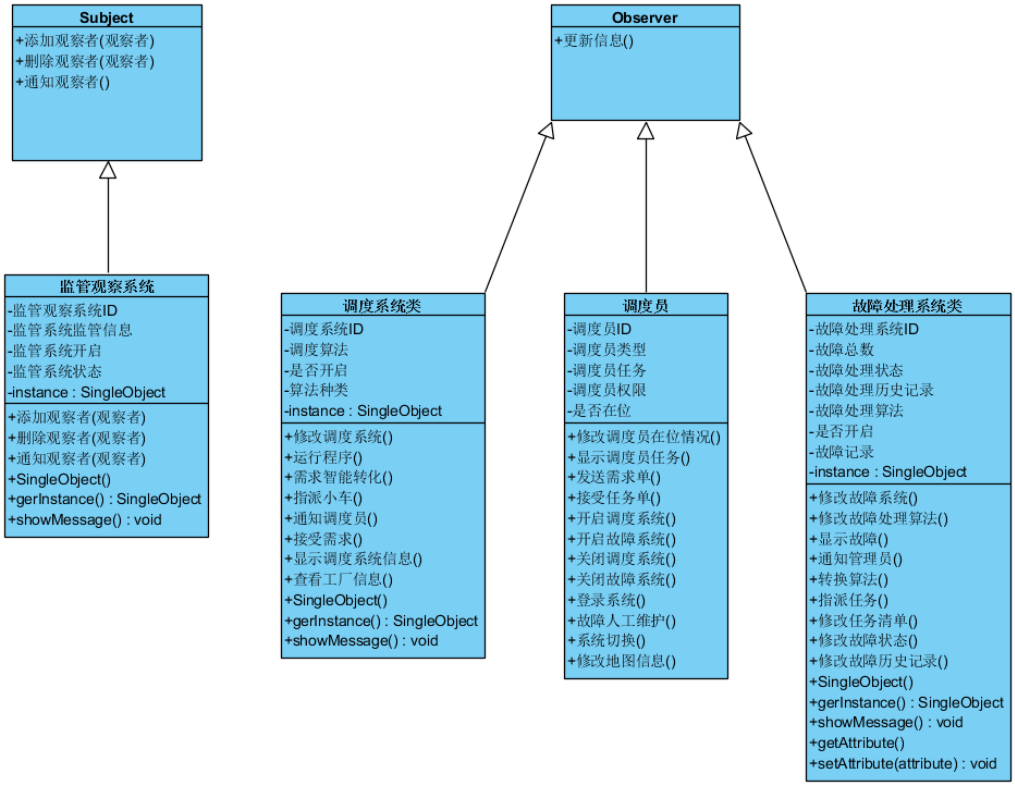
1. 设计模式
这里使用了三种：单例模式，观察者模式和抽象工厂模式。

（1）单例模式
由于调度系统类，监管观察系统和故障处理系统类都是单例模式，所以在类图中抽象成一个 SingletonPattern 抽象类，同时在三个系统类中，都有单例设计模式的方法和属性，比如单例的属性，得到单例的方法等等。



（2）观察者模式

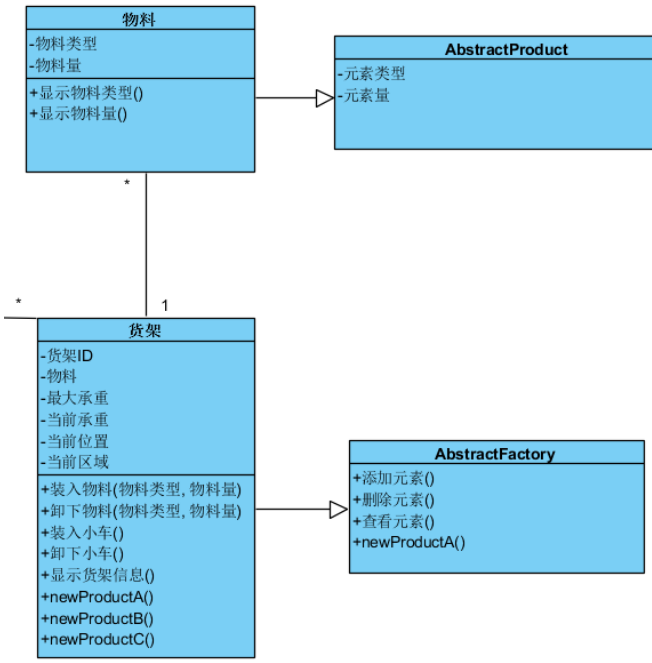
全局的监管观察类的功能是检测工厂所有的信息和情况，然后对于不同的信息会汇报通知给不同的系统类或者管理员类，所以这里是观察者模型。这里的 Observer 和 Subject 都是抽象类，提高代码复用，逻辑更清晰。



（3）抽象工厂模式

抽象工厂模式（Abstract Factory Pattern）是围绕一个超级工厂创建其他工厂。该超级工厂又称为其他工厂的工厂。这种类型的设计模式属于创建型模式，它提供了一种创建对象的最佳方式。在抽象工厂模式中，接口是负责创建一个相关对象的工厂，不需要显式指定它们的类。每个生成的工厂都能按照工厂模式提供对象。

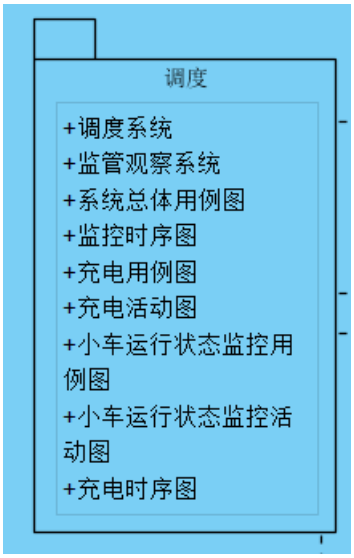
这里我们用货架类扩展抽象类抽象工厂，用物料类扩展抽象类抽象产品。



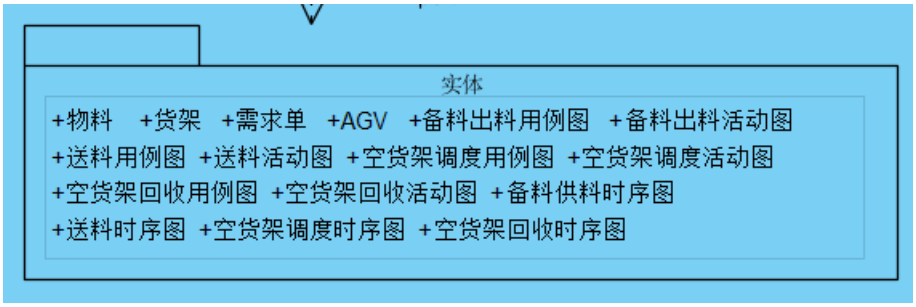
2. 包图的完善

包图的完善体现在对 GUI 的实现做了细化说明，对于实现部分加入了用例图，活动图和时序图的说明等。下面选取部分进行说明。

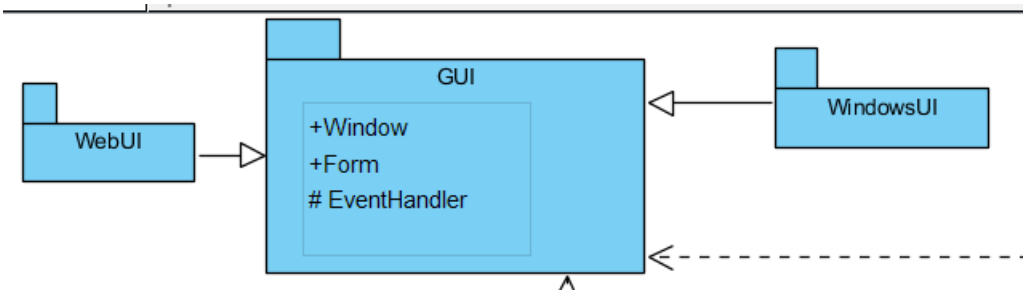
调度部分含有有关调度系统和监管系统的类图，用例图，时序图等。



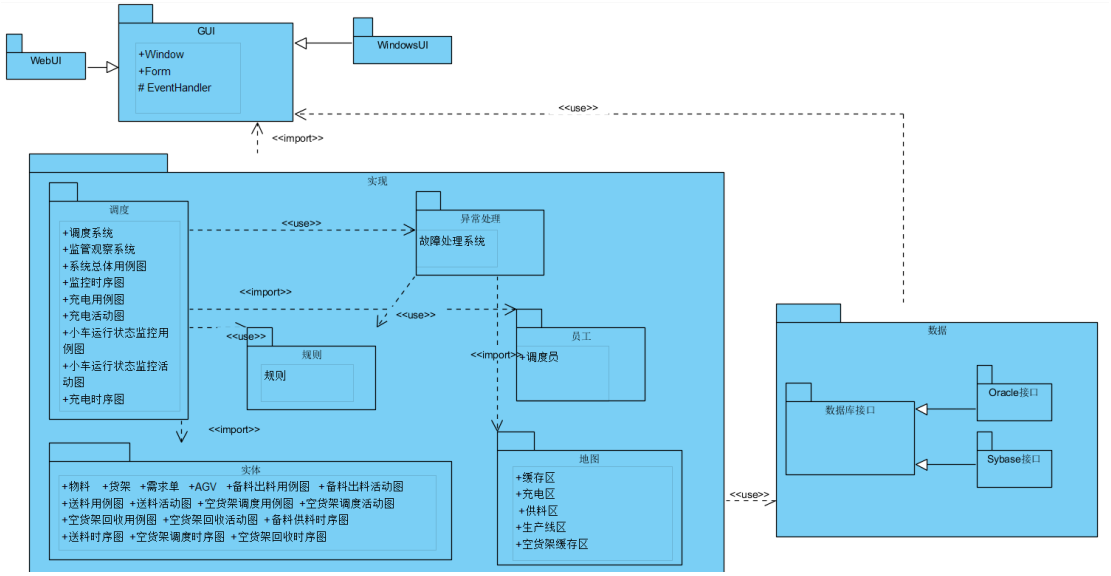
实体中包含各种实体类，同时含有实体所需要做的操作，用用例图，时序图等来表示。



GUI 可以被 WebUI 和 WindowsUI 来实现。

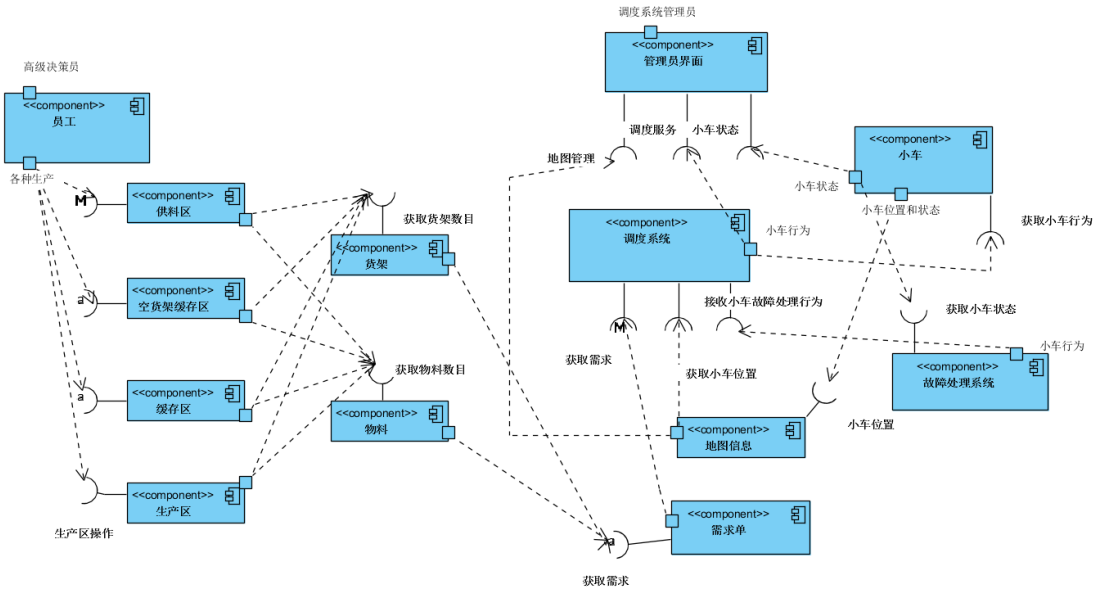


总图



2.2.2 体系结构设计

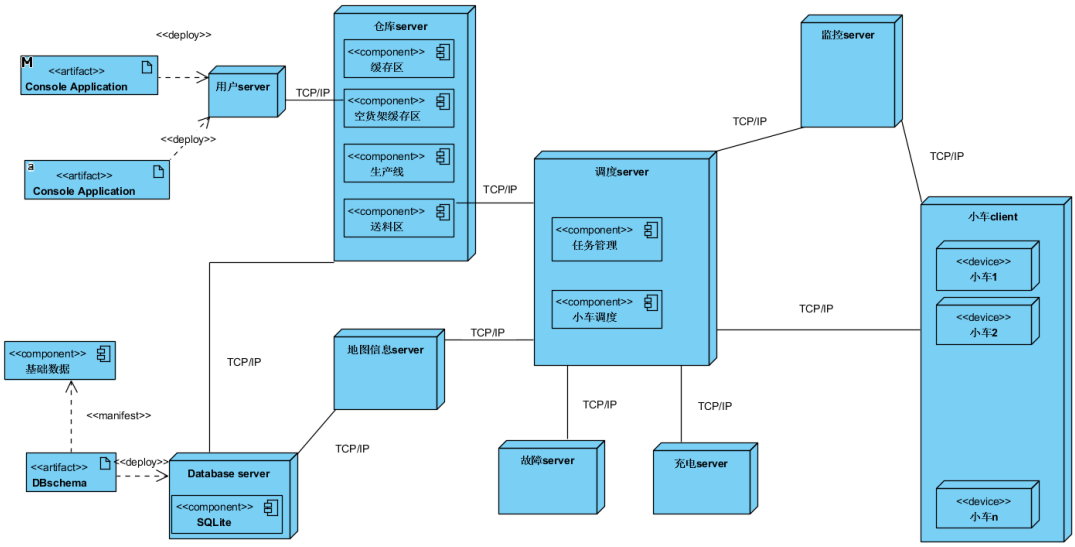
2.2.1.1 组件模型



构件图显示构件类型的定义、内部结构和依赖。构件是系统设计的模块化部分，给出各个接口的依赖，但是隐藏实现。在图的左半部分，是涉及到整个仓库生产的流程。员工或者任何自动化部件会和四大模块（供料区，空货架缓存区， 缓存区和生产区）有接口连接。这四大区域都会提供会给物料和货架接口提供数量和对应的种类。物料和货架接口将数量发给需求单接口。需求单部件将需求单提交给调度系统，调度系统综合地图信息，各个小车的状态来计算出一个任务列表，然后根据任务列表和任务管理算法去控制小车的行为，或者发

生故障时通过接口访问故障处理系统，根据故障处理系统返回的故障处理方案再调度小车的运行。而地图，调度和小车的信息都通过接口返回给管理员界面，系统的管理员可以看到全局的情况。

2.2.1.2 部署模型



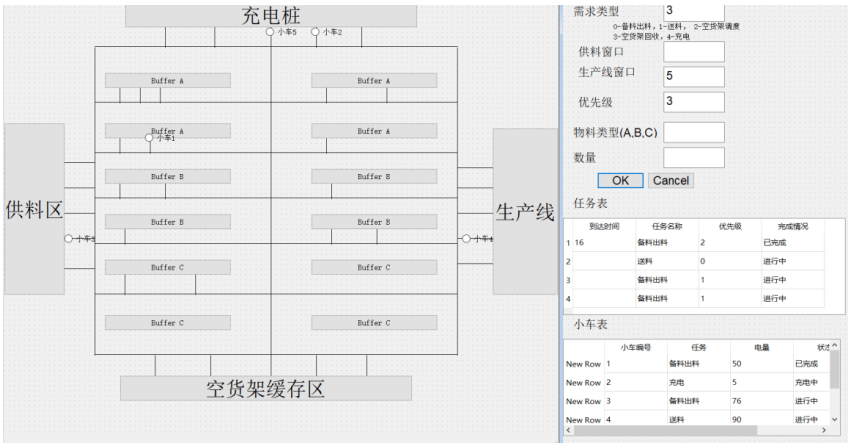
部署图(deployment diagram，配置图)是用来显示系统中软件和硬件的物理架构。从部署图中，您可以了解到软件和硬件组件之间的物理关系以及处理节点的组件分布情况。使用部署图可以显示运行时系统的结构，同时还传达构成应用程序的硬件和软件元素的配置和部署方式。

部署图使用了用户 server，仓库 server，调度 server，监控 server，地图信息 server，Database server，故障 server，充电 server 以及小车 client。对数据库模式做了 manifest 说明。用户 server 也细分成了前端的应用显示界面。小车 client 细分许多小车。这个 server，client 之间都用 TCP/IP 网络协议连接

2.2.2 用户界面设计

设计思想：我们想模拟一个控制台。可以一边显示整个仓库的地图概览，另一边可以进行添加任务的操作，还可以显示任务列表和小车列表

整体界面



左侧是整个仓库的地图设计，道路用细线绑定矩阵为 1 的点，不用区域绑定矩阵中的不同值
右侧是用户交互界面，可以通过输入添加任务，以及通过表格的形式查看任务列表和小车的状态情况
在用户交互界面中，可以输入任务的类型，对应供料区和生产线的窗口，任务优先级以及物料的类型
任务表中，会自动加载添加的已有的任务，显示出任务类型，优先级和任务的状态
小车表中，会记录所有编号的小车的情况，包括电量，编号，任务列表中当前执行的任务和任务状态。

2.2.3 详细算法设计

2.2.3.1 调度算法

在自动化仓储系统中，合理地规划路径对于小车的高效工作非常重要。这一部分也是调度系统的主要功能。由于我们采用矩阵的方式来表示地图，于是我们就可以将路径调度问题转化为图论的问题。一般来说，求单源最短路径有 BFS,DFS，Dijkstra 等基础算法。拓展的有 A*算法等等。以下将详细阐述将各种算法的适用场景，优劣比较，同时也给出自己设计的一个独特的算法。路径规划分为两部分，单台小车的最短路径规划，即无碰撞冲突。多台小车的路径调度，考虑了多种碰撞冲突。

1. 单台小车的最短路径规划（无碰撞冲突）

（1）BFS 算法

BFS 算法是用来寻找最短路径的最典型的算法。依靠队列的数据结构来实现，类似于树的层次遍历。在图中选定一个节点，依次访问该节点的全部邻接点，再依次遍历与其邻接点相邻的所有邻接点，直到找到目标节点。

伪代码:

```
Path = []
Q = Queue()
Q.push(startpoint)
While(1):
```

```
S = Q.pop()
For each direction:
    If map[s][direction] == 1              //代表有路
        Path.push(s+ map[s][direction])
    If map[s][direction] == terminal:              //找到了最短路
        Return s+ map[s][direction]
```

（2）经典 Dijkstra 路径规划算法

经典的 Dijkstra 算法是按路径长度递增的顺序来产生最短路径的算法，可以解决单源点的最短路径问题。它可以找出源点到终点的最短路径，其主要思想是，首先从源点至相邻节点寻找距离最短的一条路径，然后以路径的终点为中间节点在相邻节点中再次寻找，直至遍历所有节点。Dijkstra 算法的特点是所搜索到的最短路径中的 任意子路径均为最短路径，故可得到从源点到地图中任意目标节点的最短路径。

（3）A* 算法

A*算法是最有效的直接搜索算法之一。不同于 BFS 算法和 DFS 算法，A*算法是一种启发式算法。所谓启发式搜索，就是在当前搜索节点往下一步节点。选择时，通过启发函数来选取距离目标点代价最小的点作为下一步搜索节点而跳转。A*算法最大的好处是体现在有障碍物的图中，它会比前三个算法更快的找到最短路。

A*算法的启发函数，可以用以下公式表示：

$$f(n)=g(n)-h(n),\ h(n)<=h^*(n)$$

f(n)——节点 n 的估价函数

g(n)——从起始点到当前点的代价

h(n)——当前节点 n 到目标节点的启发估计代价值

h*(n)——从节点 n 到目标节点的实际最优代价值

当启发函数 h(n)<=h*(n)时，A*算法搜索的范围较广、点数较多、搜索效率较低，但能得到最优解。如果 h(n)=h*(n)时，则启发函数 h(n)就是最短距离，此时搜索效率最高。若 h(n)>=h*(n)，则此时搜索的范围变小、搜索点数变少、搜索效率较高，但不能保证搜索得到最优解。

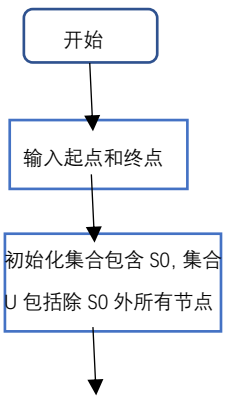
这里我使用的是最简单的曼哈顿距离

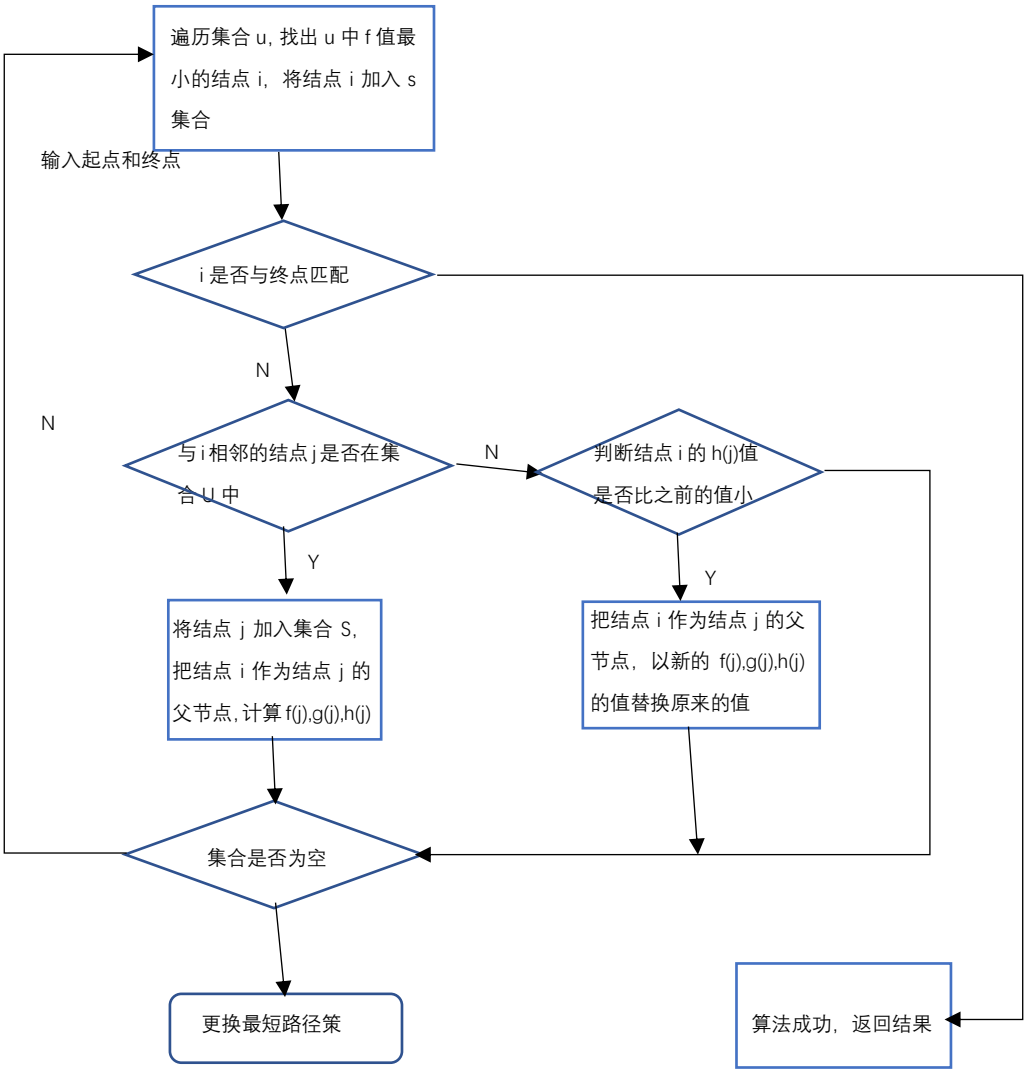
$$h(n) = K * (abs(x1-x2) + abs(y1-y2))$$

另外提供一个可供参考的启发函数——切比雪夫距离:

$$h(n) = K * \max (abs(x1-x2) + abs((y1-y2))$$

流程图如下所示：





（4）改进的基于最短时间的 Dijkstra 算法优化

由于仓储系统的地图是“矩形”的，所以任意两点间存在着多条长度相同的最短路径，而经典的 Dijkstra 算法在搜索时仅保存唯一的中间节点，最终只能找到一条最短路径。改进的 Dijkstra 算法将相同距离的节点均作为中间节点保存以备用，并以中间节点为起点在相邻节点中再次搜索，直至遍历至指定终点。通过多次迭代，找到所有相同距离的最短路径。

对于图 map，改进的 Dijkstra 算法如下：

(1) 初始化，将起点 v 标记加入 S 集合；

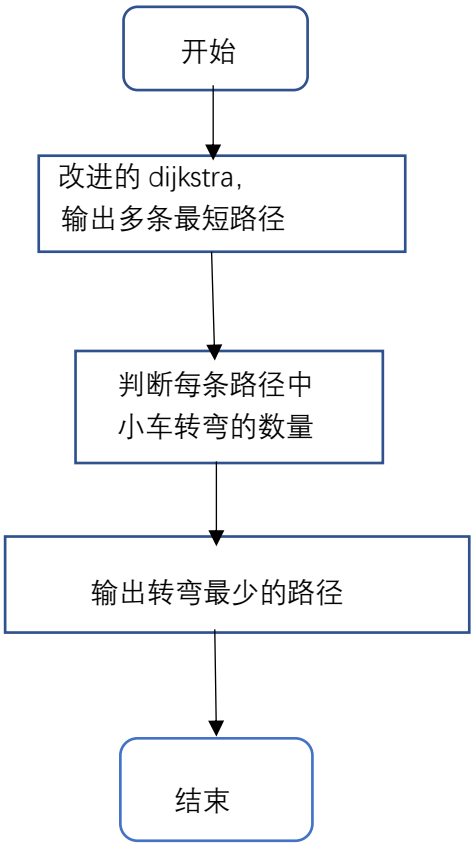
(2) 在 V-S 集合中遍历节点，选出所有与起点 v 一次可达的节点均作为备选中间节点；

(3) 在备选中间节点中选择编号最小的节点 i，加入 S 集合；

(4) 将 i 作为新的中间节点，同步骤(2)、(3)，在 V-S 中寻找与 i 一次可达的节点中选择编号最小的，更新源点 v 到节点 J 的距离，若 DIST(j)>DIST(i)+c(i，j)，即经过节点 i 比不经过节点 i 距离短，则修改 DIST(j)为 DIST(j)= DIST(i)+c(i，j)，同时把节点 j 加入 S 中。

(5) 重复(2)—(5) n — 1 次，当已遍历至指定终点，则 DIST(X)中储存了从源点到指定终点之间的最短路径。

当迭代结束，即所有最短路径搜索完毕，输出每条等距路径的起点、中间节点、终点。小车通过节点时，存在直行及转弯两种行驶方向。其中直行通过无需减速，用时可视为 0s；而转弯时，小车经过减速、转向 90 度、继续行进，用时大约 4s。显然，尽管小车行驶相同的距离时，路径中转弯越多用时越长。在小车的路径规划中，不仅需要考虑路径的长度，还要考虑行驶时间。因此，在最短路径搜索的基础上，计算每条路径的转弯个数，从而找到距离和时间均最短的路径，即转弯最少的路径。针对改进 Dijkstra 算法搜索到的所有路径，由式(3—4)判断是否存在转弯，并计算出每条路径的转弯个数，从而找到转弯最少的路径。另一种处理方式就是将一个转弯数当作一个很大的路径代价加到 c(i, j)中



（5）改进的 A*算法

同样考虑转弯的因素：小车转向过多，会浪费大量的时间。这里对启发函数进行和评估函数进行修改即可。首先，需要增大启发函数 h(n)在评估函数 f(n)中所占的比例，使得在路径规划过程中，启发信息的权重增大，路径规划算法会得到更准确的路径，这样可以通过在曼哈顿距离的计算中乘以一个系数来实现，公式如下：

$$h(n) = K1 * abs(x1 - x2) + K2 * abs(y1 - y2)$$

$$f(n) = g(n) + h(n) * s(n)$$

g(n)——从起始搜索点到当前节点的代价

h(n)——基于曼哈顿距离的启发函数

s(n)——转弯权值函数

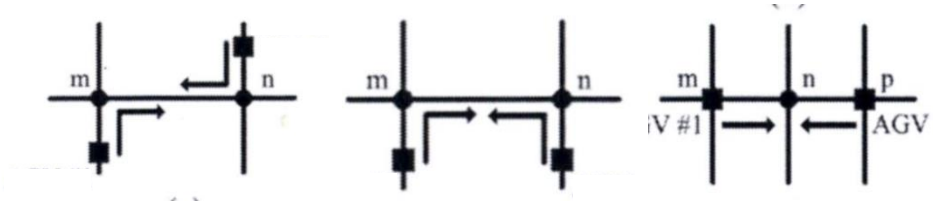
2. 多台小车的最短路径规划（有碰撞冲突）

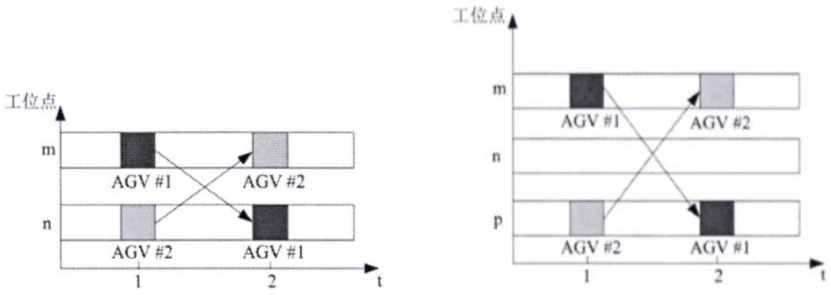
当多个小车同时行驶，难免会出现同时占用同一路径或工位点的情况，因此可能会出现诸如碰撞或死锁之类的交通问题。有效的多小车无冲突路径规划策略既可以提高路径灵活性、空间利用率和安全性，也可以降低运营成本。

首先要做的事就是冲突检测。当每一辆小车出发时，都要进行一下冲突检测。冲突检测是小车两两迭代进行的，所以下面详细讨论两台小车之间的冲突类型，然后统一讨论一下各个冲突类型对应的策略。

（1）相对冲突

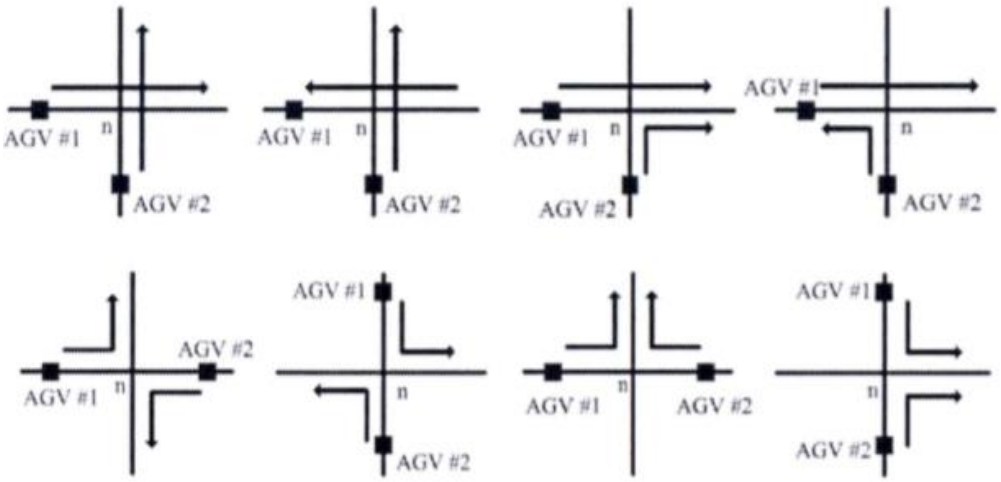
相对冲突发生在两台小车同时行驶在同一路径上，或两台小车同时经过同一工位点，但方向相反时。时间窗模型如下图所示。时间窗由许多带颜色的方块组成，每个方块代表当前小车所占用的缓存区工位点，这些工位点组成了一条小车的路径。每两个方块之间的空闲时间窗可供其他小车使用。

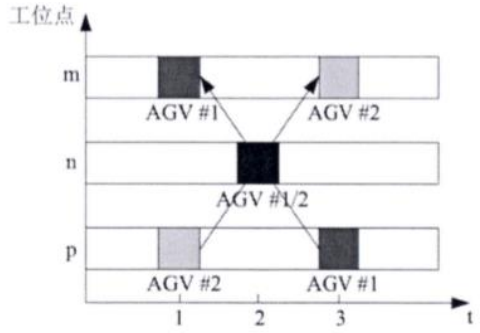




（2）十字交叉冲突

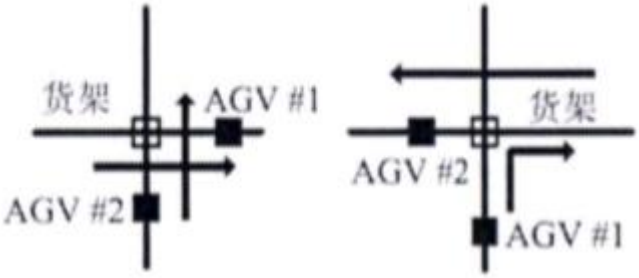
十字交叉冲突发生在两台小车在十字路口争夺缓存区工位点资源的时候。它的时间窗模型如图所示，每个方块代表当前小车所占用的工位点，这些工位点组成了一条小车的路径。每两个方块之间的空闲时间窗可供其它小车使用。

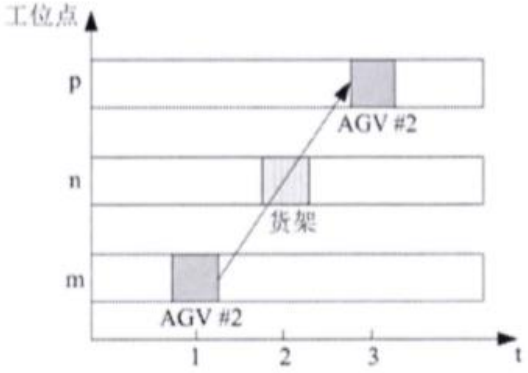




（3）缓存区占用冲突

缓存区占用冲突发生在以下情况下：有一辆小车从送料区运料至某个缓存区，然后停留在缓存区变成了空闲状态。此时，生产线发出一个从缓存区送料的请求，而此时依据调度算法，可能调度了一辆其他的小车来执行该任务。此时，缓存区发生占用冲突，因为后来的小车没法进去该缓存区取货。





基本冲突解决策略：

对于某些冲突类型，如相对冲突，传统的等待策略并不能有效的解决小车之间的冲突，甚至会导致多小车系统的死锁和其他系统故障。冲突解决策略对于自动仓储小车系统的安全性和效率提高非常重要。采取合适的策略是冲突解决的核心问题，直接影响着 AGV 系统的性能。根据冲突的类型，我们提出三种解决方案：

- 策略(a)：等待。在冲突点前一定距离处等待，或者在进入冲突点或冲突路径前减速；
- 策略(b)：更改路径。在计算路径时将冲突点及相连的路径从地图中删除，重新规划后到达冲突点的 AGV 的路径；
- 策略(c)：重新下达任务。

策略(a)是小车 2 停在冲突的节点的上游路径上，等待预定的时间，直到小车 1 离开。也就是说，小车 2 的路线中的节点的顺序不变，而所有节点的时间安排或小车 2 通过该节点的时刻将发生改变。

等待时间的设置如下：

1)对于十字交叉冲突，小车 2 需要等待至少 3 个时间单位；

2)对于相对冲突，小车 2 需要等待至少 15 个时间单位，直到小车 1 离开冲突路径段。

策略(b)是改变小车 2 的路线。如果小车 2 的备选路径与小车 1 的备选路径存在冲突，服务器将重新定义环境地图，将冲突的部分删除，然后利用改进的 Dijkstra 算法重新规划小车 2 的最短路径。环境地图的修改方法如下：

1)对于十字交叉冲突、节点占用冲突，冲突节点及所有与其相邻的路径将被删 除；

2)对于相对冲突，冲突的路径将被删除。 在解决冲突时，服务器会实时比较小车采用二者的行驶时间，也就是说，策略(a)和策略(b)都会被检查，用时最少的将被作为最终的解决策略。

然而，如果发生缓存区占用冲突，只能采用策略(c)解决冲突,也就是放弃调度策略，直接调用停在缓存区上的小车。

2.2.3.2 任务管理算法

首先我们明确在该调度系统中的任务定义：

备料出料	小车当前位置→空货架缓存区→缓存区
	小车当前位置→供料区→缓存区
送料	小车当前位置→空货架缓存区→缓存区
	小车当前位置→缓存区→生产线
空货架调度	小车当前位置→空货架缓存区→缓存区
空货架回收	小车当前位置→生产线→空货架缓存区
非用户输入的任务	小车当前位置→充电区
	故障处理
	碰撞冲突解决
	小车状态调整（停靠、运行、速度、转向）

用户在交互界面可以直接添加和管理的任务如上图第一列，但是这些任务可以被分为子任务如上图第二列，分配给不同的小车并行处理，例如备料出料任务，从空货架缓存区取空货架和从供料区取装有货物的货架可以同时进行，分配给两个小车，分别选择相应最优解，两个子任务最优时间取 max 值，即是大任务的执行时间。

如上图最后一行，除了用户输入的任务外，还有调度过程中调度系统产生的需要小车完成的任务，例如碰撞处理、故障处理。

最后问题转换成多任务多处理器的调度问题：本文采用整数编码，所需调度任务数为 n，可调度车辆数为 m。将所有任务编号为 0, 1, 2, …, n-1，车辆编号为 0, 1, 2, …, m-1。任务管理算法分为任务分配和任务排序两部分，前者决定任务分配给哪个小车，后者决定小车任务列表的执行顺序。

一般任务调度算法有 2 种：一种是累积一定任务后的统一调度，称为静态调度；另一种是调度安排随着新任务的进入而改变，称为动态调度。因为仓库生产过程中需要面对新任务的插入、路径冲突、突发故障等问题，所以调度环境是动态的，一些系统信息是随时间变化的，所以这里的任务管理算法是一种动态调度问题。对于这种问题，通常需要算法的求解速度快、效率高，并且能够满足生产运输中实时变化的任务调度，我们讨论出如下的一下智能小车任务调度原则，方便实际系统实现时选择。

（1）优先级原则

为了确保多个任务下发到多个小车后能够正常运行，必须按照任务的种类与下发时间赋予每个任务一个优先级，下发时间相同的情况下优先选择优先级高的任务，同一优先级任务优先选择分配时间早的任务。

按照常识及调查结果，有以下优先级评定规则：

故障处理任务和运行途中的碰撞处理任务是优先级最高的任务，此种情形一旦出现，在任务队列中排在最前位置；其次是小车充电任务；再次是运输任务和状态调整任务（速度、停靠/运行、转向等），停靠的优先级最低

（2）规划路径最短原则

规划路径最短原则是在对任务进行调度时，调度系统对分配的所有任务规划出相应的路径方案，然后计算各个方案中所有 AGV 总的行驶路径，选取总路径最短的一个规划方案。此种方法采用最短行驶路径优先的原则，保证所有任务中各 AGV 行驶路径是最短的，使得总体车辆行驶距离最短，但是这种调度方式忽略了 AGV 之间可能发生的冲突问题，从而使得任务的执行时间反而更长。基于行驶路径最短原则的调度方法主要有最短行驶距离优先方法、最近工作站点优先方法和最近车辆优先方法等。其目标函数公式如下面公式所示：

$$\min F = \sum_{i=1}^{n} (\mu 1 * L1 + \mu 2 * L2)$$

式中：

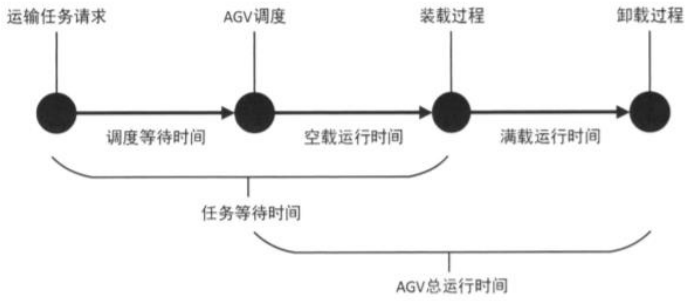
n——AGV 小车数量

μ1、μ2——AGV 分别为空载和负载时的运行系数

L1、L2——AGV 分别为空载和负载时的运行距离

一个任务的完成时间分为如下几部分：

调度等待时间为从调度系统分配任务，到某小车解收并由系统为其规划出路径的时间；空载运行时间为小车未负载时运行时间；满载运行时间为小车有负载时运行时间；后两者相加为小车运行总时间



（3）等待时间最短原则

该原则是指，在任务调度中，执行任务的 AGV 会出现因冲突而等待或在等待任务下发而空闲的情况，此种原则目的是使得每个任务的总等待时间最短。等待时间即为冲突等待时间与空闲等待时间之和，冲突等待时间是指 AGV 执行任务时遇到冲突时的等待时间，空闲等待时间是指 AGV 系统所要求的任务调度系统用于发出 AGV 所需的任务、任务等待时间

其目标函数如下公式：

$$\min F = \sum_{i=1}^{n} (\mu 1 * T1 + \mu 2 * T2)$$

式中：

n——AGV 小车数量

μ1、μ2——AGV 分别为空载和负载时的运行系数

T1、T2——AGV 分别为空载和负载时的运行时间

（4）工作队列最优原则

调度系统将任务分配到任务队列中，而 AGV 工作队列最优原则是指调度系统将任务下发到多个任务队列中，使得调度系统总运输量最大，此种原则主要包括最大化出站队列长度方法、最小化等待出站队列空问等。

其目标函数如下公式：

$$\max F = \sum_{i=1}^{n} \sum_{j=1}^{m} \omega\, ij$$

式中：

n——AGV 小车数量

m——位点的数量

ω_{ij}——第 i 辆 AGV 在第 j 个位点的负载

（5）复合指标调度原则

复合指标调度将多个任务调度指标进行复合考虑，将不同的指标根据 AGV 调度系统的需求，赋予其不同的优先级。本文采用复合调度指标进行调度，主要考虑 AGV 的行驶距离和 AGV 利用率，通过空载、负载、距离等系数动态调整各个指标的权重分配。

其目标函数如下公式：

$$\mathrm{F} = \min\{(\mu 1 * L1 + \mu 2 * L2\ ) + \mu 3 * W\}$$

式中：

n——AGV 小车数量

μ1、μ2——AGV 分别为空载和负载时的运行系数

L1、L2——AGV 分别为空载和负载时的运行距离

μ3——调度效率系数

W——小车利用率

任务的生命周期

本文设计调度任务的状态一共有 8 种，分别为初始状态(RAW)、激活状态（ACTIVE)、可分配状态(DISPATCHABLE)、不可分配(切谖 0UTABLE)、执行状态(BEING PROCESSED)、取消状态(WITHDRAWN)、失败状态(FAILED)和完成状态(FINISHED)。

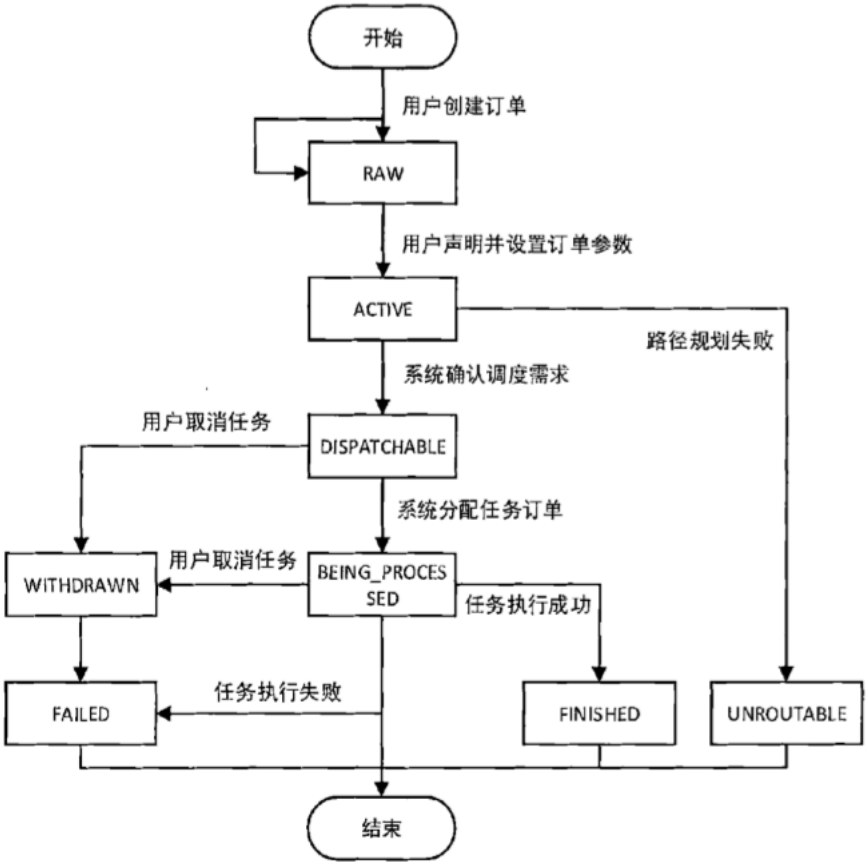


图 3-3 任务生命周期状态机模型

2.2.3.3 故障处理算法

2.2.3.3.1 故障诊断的概述

故障诊断是一个活跃的研究方向。故障的定义为系统的特性或变量偏离了正常的范围，广义的讲为系统表现出不期望的特性的任何异常现象。根据现有的故障诊断方法，可以将故障诊断方法分为基于信号处理的方法、基于知识的方法以及基于模型的方法。

（1）基于信号处理的方法

基于信号处理的思想是：利用处理设备，将采集而来的信号，进行变化、综合、估计与识别处理，达到检测出故障的问题。该方法不需要对象的准确模型，实时性强。常

用的有傅里叶分析法、小波变换、谱谱分析法、相关分析法。

（2） 基于知识的方法

基于知识的故障诊断方法是以知识为基础，不需要了解系统的定量数学模型，利用处理方法与概念上的知识化，使得故障诊断具有“智能的”特点。该类的方法包括模型推理的方法、专家系统的方法、神经网络以及小波神经网络结合的方法。模型故障诊断是在建立模糊规则与模糊函数的条件下，通过模糊推理实现的。专家系统方法是根据专家的经验、知识以及统计故障相关信息的知识，在计算机里设计程序库，用于故障诊断推理，从而实现诊断故障。通过在故障诊断的过程中，在线的与实际健康模型进行对比，根据不同的结果，能够诊断出系统故障。

（3）基于模型的方法

对于基于解析模型的故障诊断方法的原理是：在系统的定性或定量模型为已知的前提条件下，根据系统的结构，求解系统的输入/输出，系统的状态量以及参数之间的关系模型。该类型的方法有：状态估计方法、参数估计方法、等价空间方法。基于解析模型的方法，根据系统的不同，分为线性方法与非线性方法。在这三种方法中，状态估计法是研究的比较多的方法，经过归纳总结，基于模型的方法可以归纳为四类：基于非线性的方法、基于滤波器的方法、微分几何方、自适应学习方法。

2.2.3.3.2 功能需求

该系统目标用于工厂物流搬运中。对于AGV其主要任务是将货架从一个地方转移到另外一个地方，在该过程中，AGV必须具备如下的功能需求：

(1) 能够沿着预定的路径进行自主巡航，并保持较高的精度；

(2) 能够自动判断工位信息，在不同的工位之间完成行驶和停靠，并且具有一定的定位精度；

(3) 具有故障检测功能，能够对系统本身以及周围导致系统出现不期望的运行的情况进行检测；

(4) 能够自动卸载货架，不需人为去进行操作；

(5) 适应多种用户需求，应具备手动和自动两种工作模式；

(6) 具备安全警报功能，包含视觉以及语音提示功能；

(7) AGV能够沿着路径向终点运动。

2.2.3.3.3 故障诊断功能

我们的 AGV 调度系统有一个全局的监管观察系统，这个系统可以查看整个工厂的所有的信息，包括小车的运行状态，小车的报警状态等等。同时，如果工厂中有实体对象需要向监管观察系统报告信息，监管观察系统也可以即使接收到信息。对于不同的信息，监管观察系统

会相应的给出不同的解决方案，同时实现兼顾全局的智能优化的解决方案。

以下将详细讲解小车的异常处理，小车在出现异常情况后，会自动修改自己的报警状态为相应的状态，即 0、1、2、3、4 分别对应了无报警、障碍物报警、电机报警、低电量报警、急停报警，加上小车其他属性的信息，都会在小车出现故障的时候报给监管观察系统，之后监管观察系统会针对不同的故障得到优化智能的解决方案，安排小车进行后续的维修或者顶替操作等等。

我们在小车出现异常时，可以将其不同的异常分为以下几个类型的异常情况，便于我们的处理和更好的分析、得到解决方案。为确保 AGV 在执行任务过程中顺利的完成任务。需对 AGV 可能出现的故障进行分析。

（1） 智能小车运动路径有偏移
（2） 前方有障碍物，道路不通
（3） 位置信息缺失
（4） KIVA 心跳数据缺失
（5） 智能小车的电量不足;
（6） 智能小车任务过多
（7） 智能小车任务途中抛锚

对 AGV 可能出现的故障分析，将上面的实际的故障类型抽象出来，可以分为以下几类:

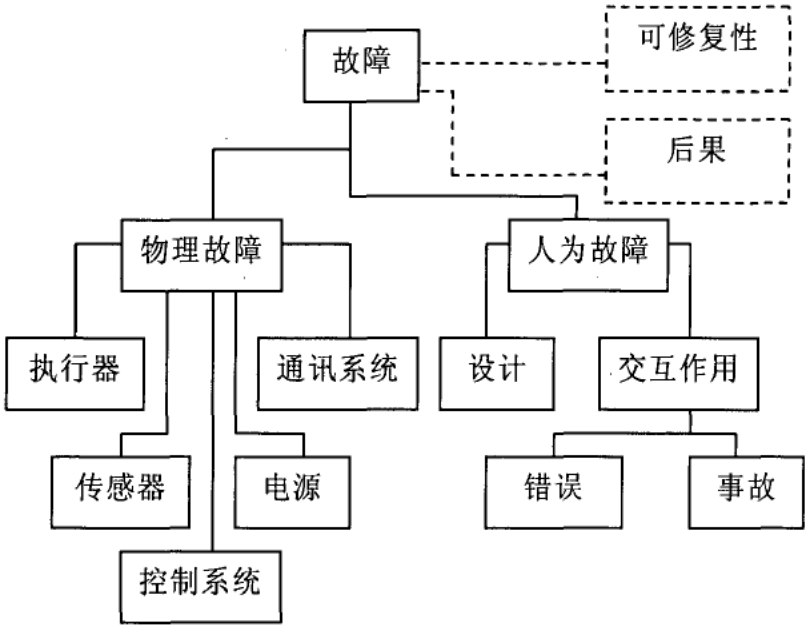


图 故障诊断类型图

（1）执行器类故障

执行器类故障是指移动机器的执行部件在运行的过程中，因为某些系统的内部或者外部原

因导致执行器无法完成指定工作任务的情况。
行走电机与转向电机的故障指行走电机与转向电机无法执行启动与停止等相应的操作的情况。可能产生的原因为在于电机的驱动器或者电机本身损坏等。

（2）电源故障

作为AGV的供电系统，电源为其重要的部分。在AGV的运行过程中，由于能源的消耗，电源发生的故障分为两类：电源低，电源严重偏低。对于电源偏低的定义为当电量与充满电量时的比为15%以下时。电源严重偏低是当电量与充满电量时的比为5%以下时。在电源偏低时AGV应该尽快去充电场所充电，对于严重偏低应该告知系统进行相应的处理，否则会导致阻碍交通。对于电路检测需要添加对电流与电压检测的传感器。

（3）通讯系统故障

通讯系统故障是指工控机与外部器件之间的通讯系统不能够正常获取或发送数据的情况。通讯系统的正常是系统能够正常运行的前提，在通讯系统不正常的情况下机器人应当采用停止且采用相应的通知等措施。工控机系统中通讯的接口有串口、USB接口以及与中央控制机相连接的无线接口。对通讯功能故障设定为两种故障形式：完全丧失功能、部分丧失功能。部分丧失功能故障是指在通信的过程中，将不能接收到的数据频率大于5Hz的情况；完全丧失功能故障指完全无法收到数据信息。

（4）控制系统故障

控制系统故障是指机器人在运行的过程中无法满足控制系统的指标要求。对于本机器人来说，有脱轨、取货故障等情况。
取货故障指在叉臂进行货物的装载或运送过程中时出现的叉臂无法叉入栈板、运行过程中的货物脱落、货物虚取、虚放货物的情况。在叉送货物的时候可能出现控制系统没有叉到栈板而出现的认为已经装载货物的虚取货行为。

（5）传感器故障

传感器故障泛指传感器在检测过程中由于本身传感器的原因或者传感器结构连接部分不紧固所造成的数据与真实值的所产生的偏差已经不能够满足系统的性能要求的故障。这些传感器对于定位单元相当重要。

（6）碰撞类

AGV运行的过程中，场地的有限性会导致其运行的过程中可能出现碰撞到人或者物品的情况，这将会造成人身财产安全事故。因此，在AGV上需要安装许许多多的测距传感器。

（7）其他故障

在以上介绍的故障中，还有没有包含在其中的故障，包括系统温度过高等情况。

2.2.3.3.4 故障诊断的性能需求

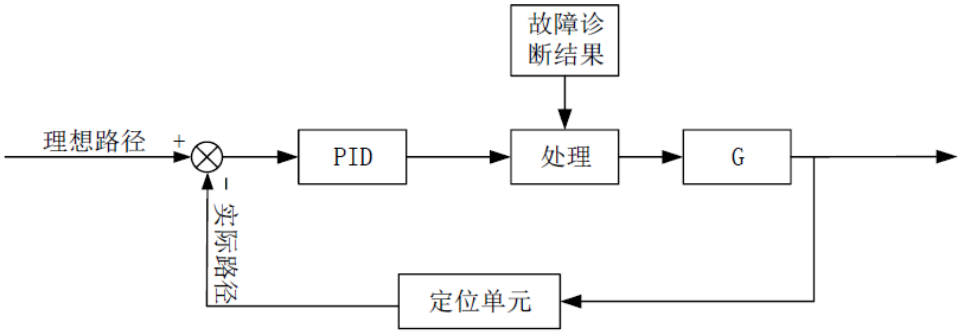
故障诊断的主要性能指标有早期的检测的灵敏度、故障检测的及时性以及故障的误报率和漏报率，该性能指标主要是对传感器故障诊断提出。早期的检测灵敏度系统对只有“微小”的故障相关信息的情况下，检测出故障的能力。故障检测的及时性指当故障发生时，故障诊断系统能够检测到故障所需要的时间的评判标准。

2.2.3.3.5 异常诊断与解决

下面将针对以上七种异常情况进行异常诊断与解决。

（1） 智能小车运动路径有偏移

智能小车运动路径有偏移属于控制系统故障。AGV 在按照指定的轨迹运行的过程如图 所示。



图

发生小车运动路径有偏移的情况的时候，我们的监管观察系统是可以及时发现的。因为小车在进行移动的时候，我们的调度系统已经对其制定了要走的路径，同时对于这个要走的路径上的每一个点都已经存在数组里，赋值给了小车的 path 属性。所以在小车移动的每一步，小车的一个函数都会自动检测是否在之前预计的路径上。如果是正常的，那就不需要其他操作；如果不是，那么说明出现了异常——小车运动路径有偏移。

假设小车得到异常——小车运动路径有偏移，那么他会调用一个函数，立即通报监管观察系统，通知监管观察系统异常的类型，小车的 ID，小车的位置，周围的工厂的情况。这个接口就不在这里展示。

监管观察系统得到异常后，通过小车的当前位置信息，和小车当前任务的 path 信息，计算出小车从当前位置到 path 最短路径 p1，记录下这个 p1，发送给小车，同时修改小车当前任务，即暂停当前任务。同时修改任务，令小车当前任务为恢复正常路

径。当小车恢复到正常路径之后，继续之前的任务，即任务被恢复。

综合起来，处理该异常可以简要分为以下几个步骤。

1) 小车检测到小车运动路径偏移异常
2) 汇报给监管观察系统
3) 监管系统计算得到复位最短路径
4) 记录异常信息
5) 强制小车进入异常情况处理
6) 小车按照给定的路径，从当前的偏移位置行驶到给定的正规位置
7) 小车传送信号给监管观察系统，报告已经完成异常处理
8) 监管观察系统强制小车恢复之前的任务状态，继续执行任务

*对于异常信息我们记录下来，之后会统一判断原因，找到异常偏移原因。

(2) 前方有障碍物，道路不通

此情景出现在场景异常的情况，也就是显示场景和系统存储的场景不同。因为 AGV 会得到调度系统给出的路线，会按照系统存储的地图去寻找最短路径，得到的路径指示在正常情况下是不会出现前方有障碍物的，除非有突发情况，地图中当时加入了其他的物品或者设备，导致道路不通或者有障碍物。

当小车遇到障碍物时，会立刻停止行驶，同时将信息汇报给中央的监管观察系统。中央系统会根据传来的异常信息，得知地图上有障碍物。此时监管观察系统会并发执行以下两个流程。

1) 向高层管理人员汇报，得到高层人员的帮助，比如将障碍物移除或者将障碍物输入地图类中，使得系统可以获取障碍物的信息。此后规划路线对于该故障将不再出错。将此故障加入案例记录，形成一个 case，之后走 case 的流程，等待 case 的解决，解决后加入历史纪录。
2) 为小车重新规划路线，根据小车传感器得知小车可以走的方向，前提是小车不会撞到任何障碍物。根据终点的方位，不断尝试靠近，直到到达终点。同时有可能由于障碍物完全堵塞所有道路，导致小车无法到达终点。这时会提高该异常的优先级，使其尽快被管理人员处理。

小车会根据自身传感器的信息和中央的系统给的指令，继续探索路径，直到到达终点。

(3) 位置信息缺失

位置信息缺失有几种原因，其中最大的可能是 GPS 功能失效，还有可能是位置传感器或者网络效率低。最终会导致监管观察系统无法得到小车的位置信息。

当监管观察系统发现有小车出现位置信息缺失时，会自动生成位置信息缺失异常单，进行处理。处理流程如下：

1) 强制小车停止行驶，即停在原地
2) 获取小车当前任务，获取小车有信息交换的终止时间和最后一段的位置信息
3) 通过任务信息和最后一段的位置信息，预测小车可能出现的位置范围。
4) 将异常处理信息优先级提高
5) 派去紧急维修人员确认位置信息缺失原因
6) 维修人员维修结束，报告系统
7) 位置信息可以正常返回，中央监管观察系统可以得到信息，两方建立正常的连接
8) 退出异常处理流程
9) 恢复小车任务

(4) KIVA 心跳数据缺失

心跳信号是每隔一段时间向互联的另一方发送一个很小的数据包，通过对方回复情况判断互联的双方之间的通讯链路是否已经断开的方法。心跳信号是为了确认一个事实——互联的双方在长时间没有通讯的情况下是否都还在线，或者说存在于互联的双方之间的通讯链路是否已经断开。而不是网上有些人所谓的“用来保持连接”，“用来维持长连接”。连接一旦建立，只能被异常或正常的断开，而不会因为没有数据传输而断开的，所以没有什么长连接的概念，更不需要用发送数据的方式来保持连接。

当 KIVA 心跳数据缺失的时候，可以判断出大概率是网络原因，这里需要分类讨论。分为大批量 KIVA 心跳数据缺失与小部分 KIVA 心跳数据缺失。

1) 大批量 KIVA 心跳数据缺失：此时可以判断是系统原因或者网络原因的大规模出错。建立异常处理的任务单，将优先级提到最高，直接报告给最高层管理人员。此时系统讲同步检测自身系统所有模块，检测是否为系统错误，如果有错误，讲记录错误模块，一起提交给管理人员。进行网络测试，检测是否为网络原因导致 KIVA 心跳数据缺失。
2) 小部分 KIVA 心跳数据缺失：可能是网络原因，或者小车出现故障。锁定心跳数据缺失小车 ID，之后先尝试与小车进行数据交换，如果没有通过网络检测，说明可能是小车硬件出现故障；如果通过网络检测，说明之前心跳缺失属于偶发状态，会自动纠正。网速存在偶然的异常。硬件错误将会建立新的异常任务单，交给维修人员处理。

(5) 智能小车的电量不足；

智能小车的电量不足分为两种情况，一个是任务执行中的电量不足；一个是无任务小车的电量不足。下面分开讨论。

1) 任务执行中小车电量不足：
当任务执行的时候，小车有可能电量不足，无法继续任务。在我们的调度系统中，每一次分配任务会先计算可能的耗电量，计算耗电量通过要行驶的距离，不同车型，要载的重量等等。所以当预计的耗电量不准确时，小车行驶耗电过多才会导致任务执行中小车电量不足。

发现小车电量不足时，小车已经无法继续执行任务。此时的处理流程如下。

1） 小车向监管观察系统报告自己电量不足，无法继续执行任务
2） 监管系统创建异常处理单
3） 监管系统寻找替代车辆，寻找最近的空闲车辆
4） 为选定的空闲车辆规划替代任务，包括路线
5） 向选定的空闲车辆发送替代任务
6） 选定的空闲车辆行驶到电量不足小车处
7） 电量不足的小车与选定的空闲车进行架子转移
8） 监管观察系统向电量不足的小车发送充电指令，电量不足的小车行驶如充电区充电
9） 将剩余任务发送给选定的空闲小车，小车继续执行任务

2） 无任务小车电量不足：

小车没有任务并且没电的时候，这里的异常处理会更简单，即指派小车行驶至充电区充电。监管观察系统为小车规划充电最短路径，然后发送指令给小车，小车行驶到充电区，之后充电。充电完成后发送完成信息给监管观察系统，告知监管观察系统该小车充电完成并且汇报电量。

3） 无任务小车电量不满

小车没有任务的时候，即使是电量没有报警，当监管观察系统检测发现有有大量小车没有任务，也会生成指挥小车去充电去充电的指令。

这是为了当小车有任务的时候会有更多的电，为了均衡任务发送时间不均衡的问题。

这里的异常处理和第二个差不多，即指派小车行驶至充电区充电。监管观察系统为小车规划充电最短路径，然后发送指令给小车，小车行驶到充电区充电。充电完成后发送完成信息给监管观察系统，告知监管观察系统该小车充电完成并且汇报电量。

（6） 智能小车任务过多

对于每个小车存在一个算法，对于小车接受的任务，计算全部完成时间的时间长度，将完成时间与预定的最大任务时间比较，如果超过了最大任务时间，将会自动归为智能小车任务过多异常。

当小车的算法计算出产生了智能小车任务过多异常后，小车会主动向监管观察系统报告，说明任务过多等情况。当监管观察系统收到该信息之后，将会进行以下流程。

1) 监管观察系统得到小车任务过多异常信息。
2) 检查所有小车的任务信息，得到是否有小车分担任务的小车清单
3) 如果存在小车分担任务，则将任务过多小车的任务一半分出，逐个给可交代任务的小车交代任务。如果不存在小车分担义务，则汇报给高层人员，归类为安排任务不合理情况，记录到历史纪录中。等待高层人员对任务分配做出优化。比如安排更多的小车。

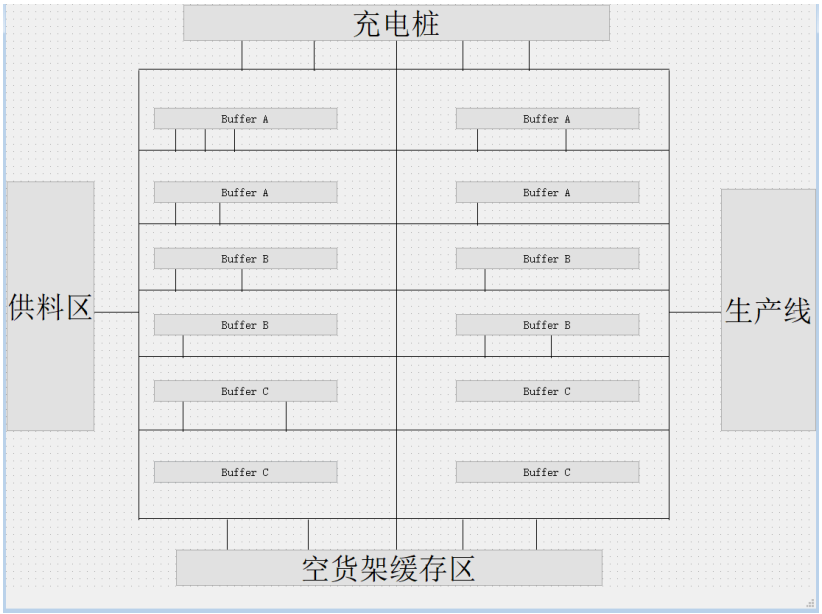
(7) 智能小车任务途中抛锚

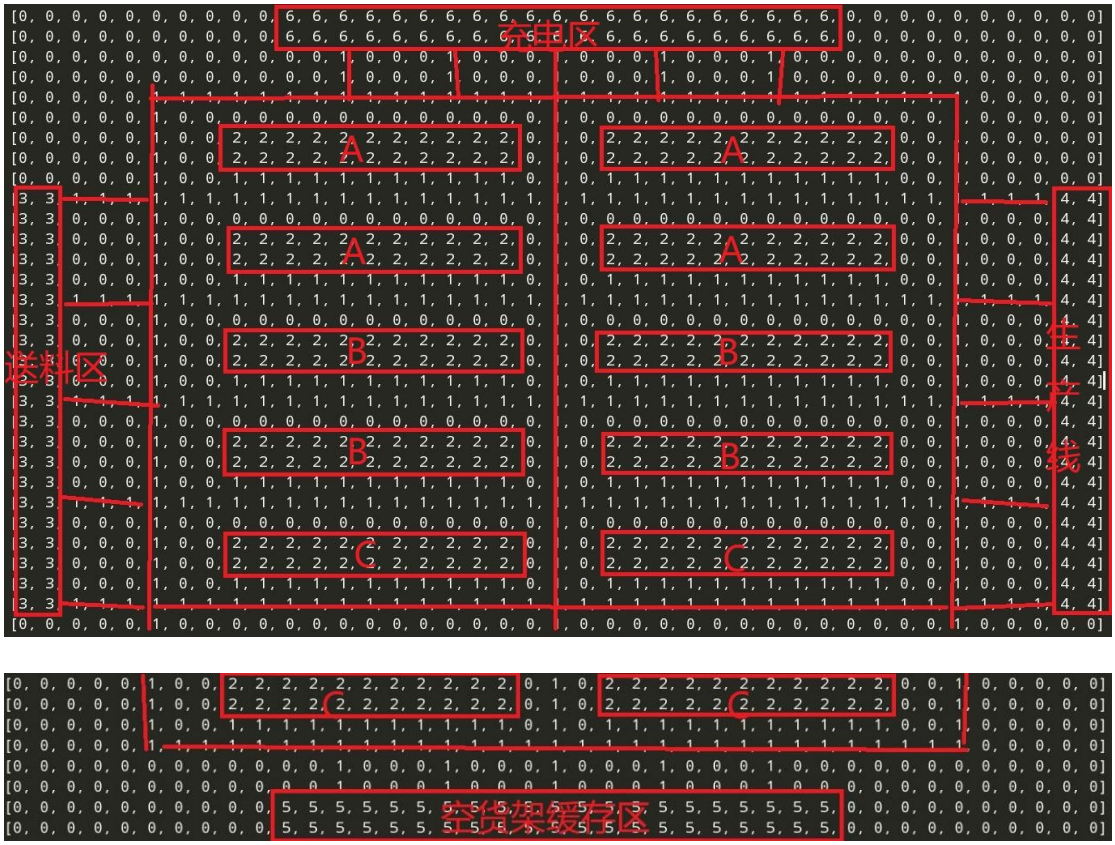
智能小车任务途中抛锚，小车会主动向监管观察系统报告抛锚信息，如位置等。

1) 小车向监管观察系统报告小车任务途中抛锚，无法继续执行任务
2) 监管系统创建异常处理单
3) 监管系统寻找替代车辆，寻找最近的空闲车辆
4) 为选定的空闲车辆规划替代任务，包括路线
5) 向选定的空闲车辆发送替代任务
6) 选定的空闲车辆行驶到电量不足小车处
7) 小车任务途中抛锚的小车与选定的空闲车进行架子转移
8) 监管观察系统向维修系统发送维修命令，维修人员前去维修
9) 将剩余任务发送给选定的空闲小车，小车继续执行任务

2.2.4 详细的地图编辑管理和数据设计

本系统为精确、直观地表述小车在执行任务过程中地货物货架装卸点、充电点等元素，设计开发了基于用户界面模式的仓库电子地图。仓储环境地图用拓扑法（即网格法）建立。如下图所示，地图分为 6 部分：小车行驶区（即路径）、缓存区、供料区、生产线、空货架缓存区，以及充电区。





1）小车行驶区：位于其它各区之间，在小车 AGV 行驶区是路径规划的主要区域，本系统主要关注点也集中在该区域；

2）缓存区：位于地图中心，占据面积最大，参与多个系统用例，用于暂时存储来自供料区的货物，在生产线呼叫时提供货物，在货物调度过程中会出现缺少货架的情况，需要小车从空货架缓存区调度货架；

3）供料区：左边的供料区有多个工作人员（或其它自动化设备），可以呼叫小车将货物移至缓存区；

4）生产线：右边的生产线时整个产线任务的最后一步，由多个位点构成，可以呼叫小车从缓存区拿取 A/B/C 货物；在生产完成后，之前装载货物的货架回收至空货架缓存区；

5）空货架缓存区：下面的空货架缓存区为缓存区提供货架，从生产线回收货架；

6）充电区：当小车的电量小于 10%时，需要到充电区充电

针对栅格地图，每个站点占据一个栅格，大小均相同。站点分为三种：货位点、路径点和无效点。货位点可放置货架，小车可运行至货架底部，也可长时间停留，是装货卸货充电的地方；通道点不能放置货架，小车可以通过该点或短暂停留，不可长时间停留，但可转向；无效点是图中四个角位置调度系统不会安排小车行驶的位置。任何两个相邻通道点、通道点与货位点可以直线行驶，但是相邻货位点之间不可通行，需要经由通道；同一时刻相邻两位点只能有一个小车通过，同一位点出现超过一个小车即发生碰撞

每个位点都由 QR 二维码进行标记，每个二维码储存它所标志位点的 ID 和空间坐标，用来确定智能小车的当前位置和方向。当小车识别到二维码信息后，将当前的时间和该位点的编号通过通讯系统传至上层系统，用于实时监测系统的显示和调度系统核对，即做出继续向前或转弯的反应，按照调度系统安排的路径行驶，或者因为碰撞等原因调整路线。

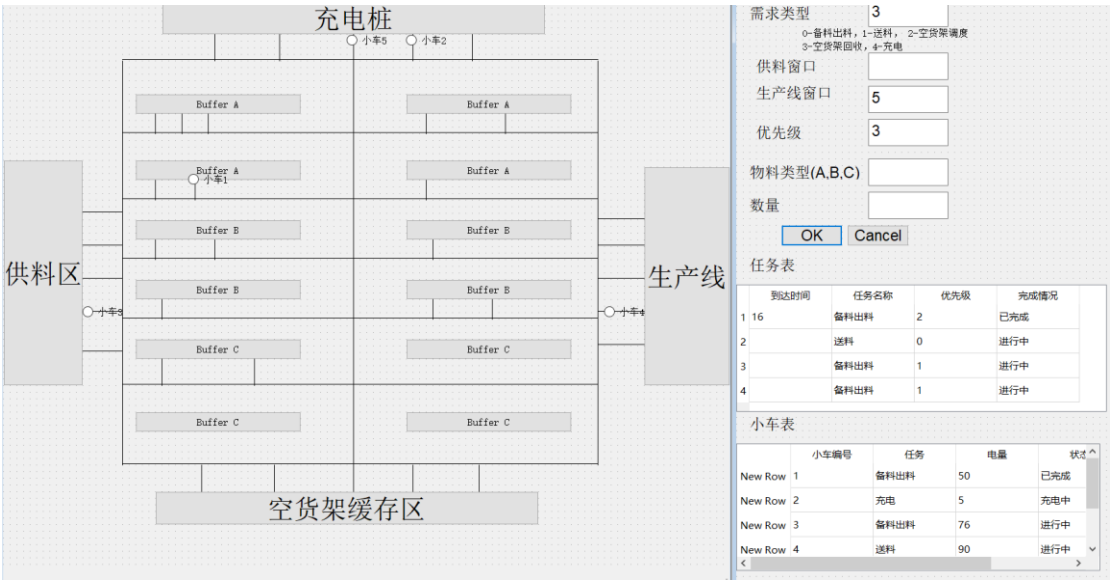
第三章 系统实现与测试

3.1 总体描述

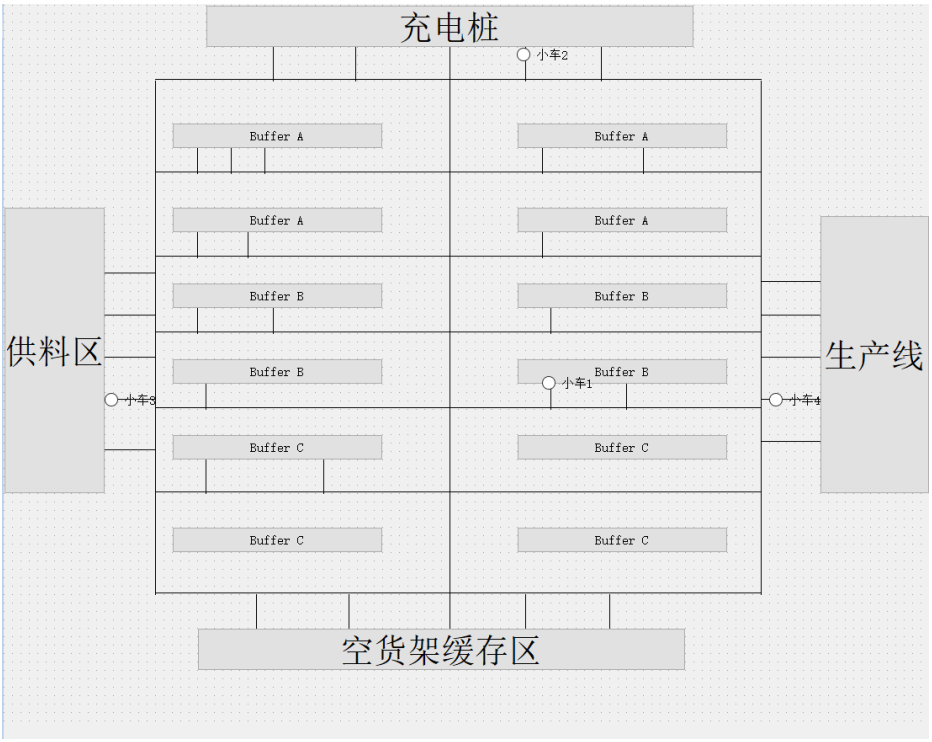
3.1.1 运行环境

所使用的框架是 Python3+pyqt 进行数据绑定和前端页面的展示。这里为了演示，有一些数据已经预先设置好

3.1.1 整体界面

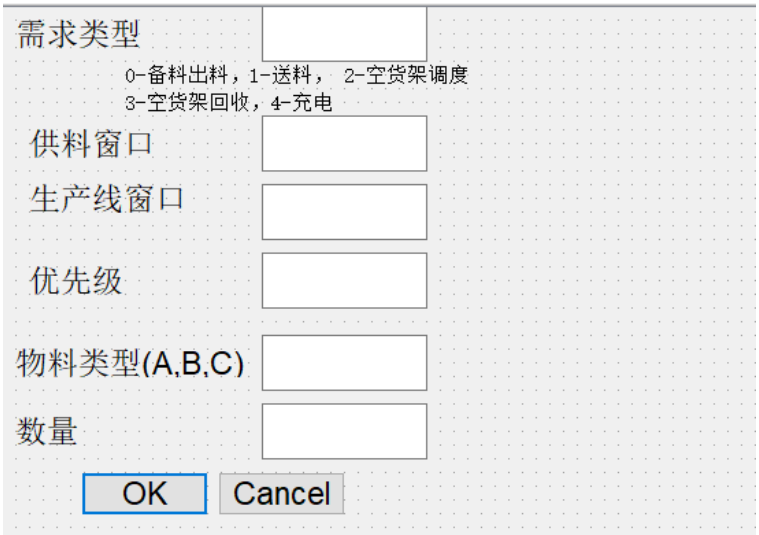


左侧是整个仓库的地图设计，道路用细线绑定矩阵为 1，不用区域绑定矩阵中的不同值
右侧是用户交互界面，可以通过输入添加任务，以及通过表格的形式查看任务列表和小车的状态情况
初始设置地图如下



小车 1 已经完成 id 为 1 的备料出料任务
小车 2 正在充电
小车 3 正在进行 id 为 3 的备料出料任务
小车 4 正在进行 id 为 2 的送料任务

从这里可以添加任务



显示任务和小车情况

任务表

	到达时间	任务名称	优先级	完成情况
1	4	备料出料	2	完成
2		送料	0	进行中
3		备料出料	1	进行中
4				

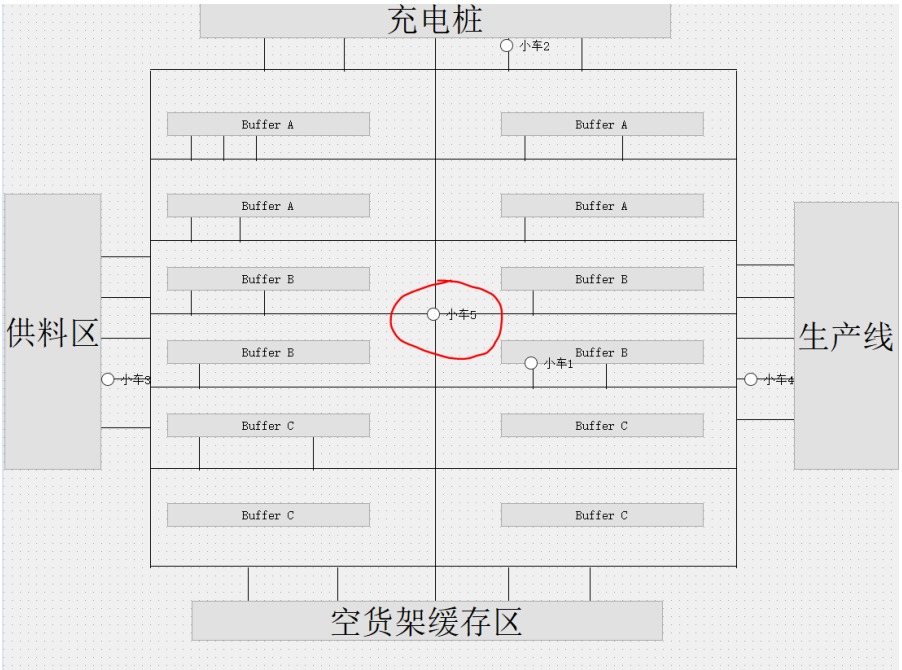
小车表

	小车编号	任务	电量	状态
New Row	1	备料出料	80	已完成
New Row	2	充电	5	充电中
New Row	3	备料出料	76	进行中
New Row	4	送料	90	进行中

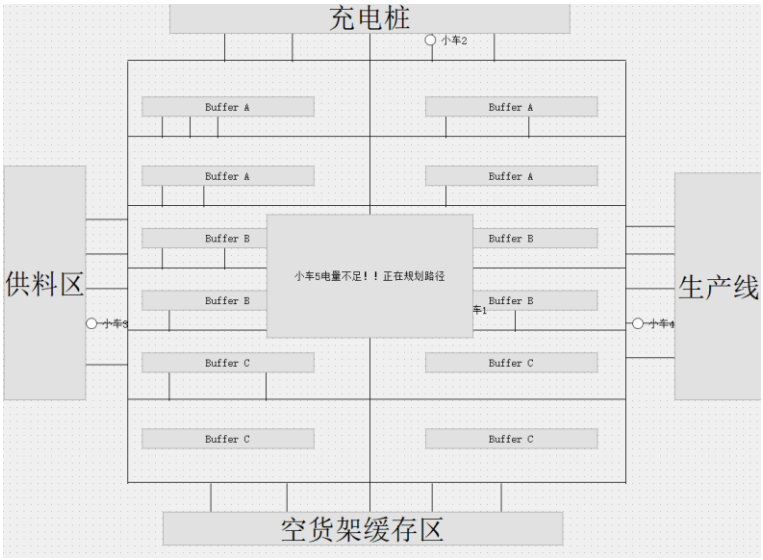
3.2 实现需求测试

3.2.1 充电测试

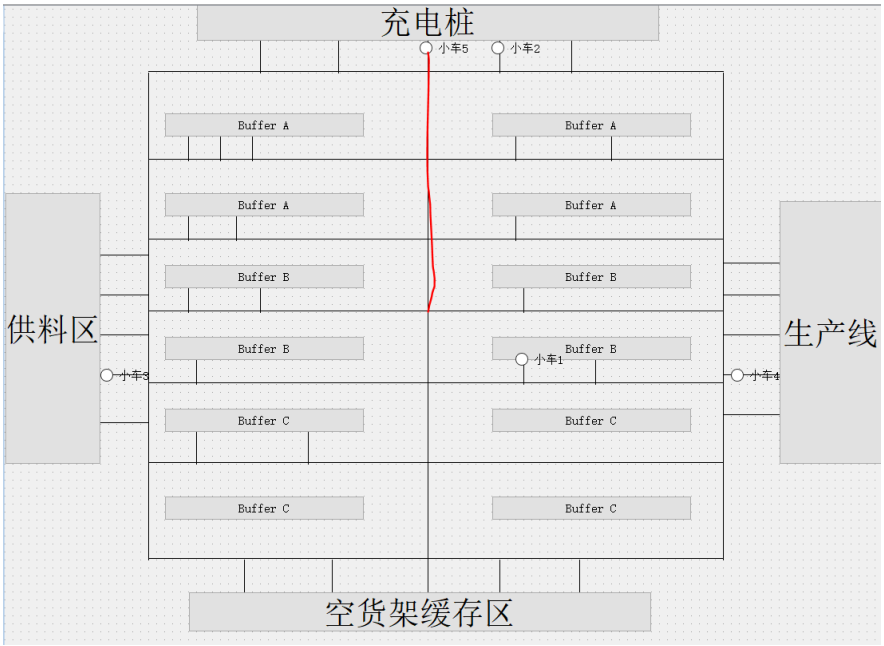
现在简单地模拟一个小车电量不足的情况：
假设小车 5 在当前位置电量不足



系统会跳出弹窗

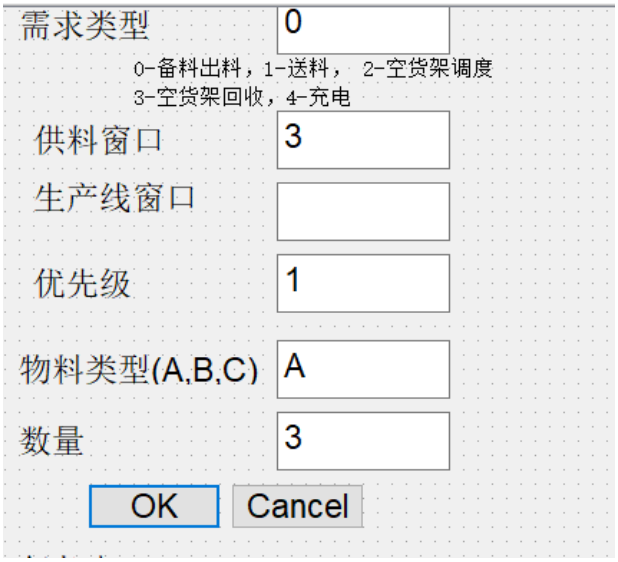


小车选择了最短路径去充电，红线表示了小车的运行轨迹。



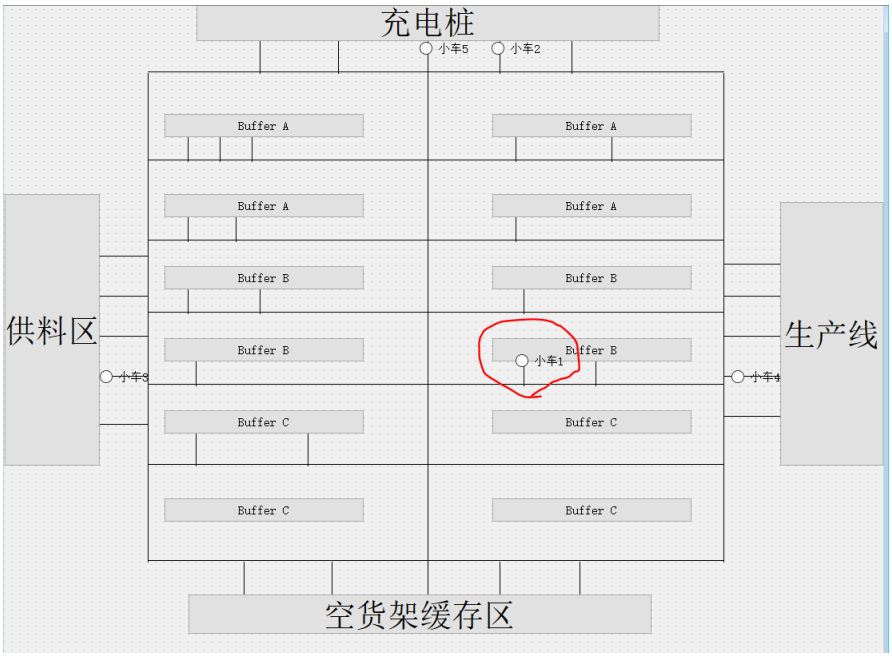
结果：验证了充电需求的正确性。

3.2.2 调度测试

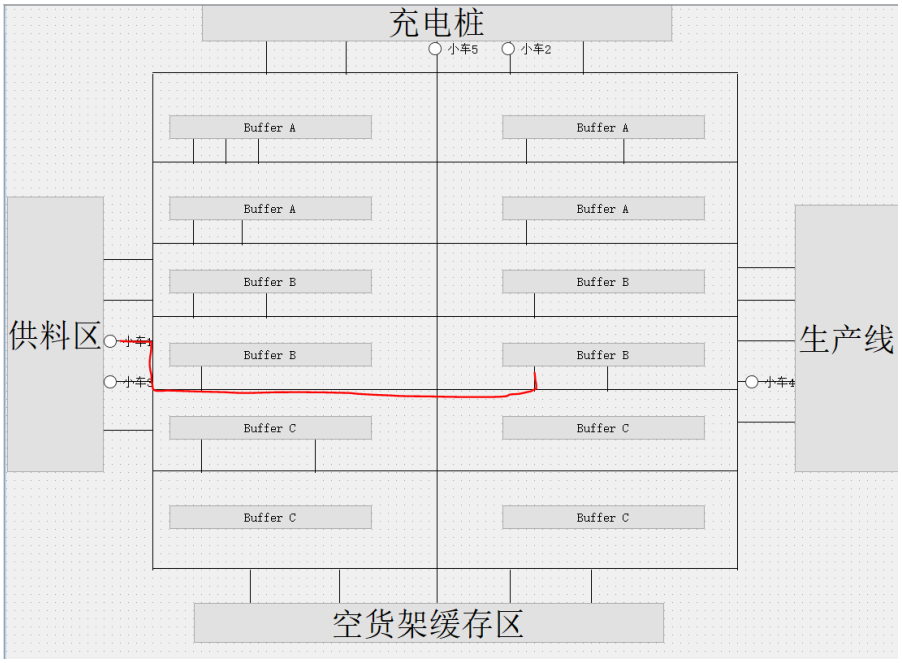


从右侧输入框可以输入添加的任务，对应不同任务可以将相应输出空出来，系统会自动检测此时在任务表中会增加一条条目

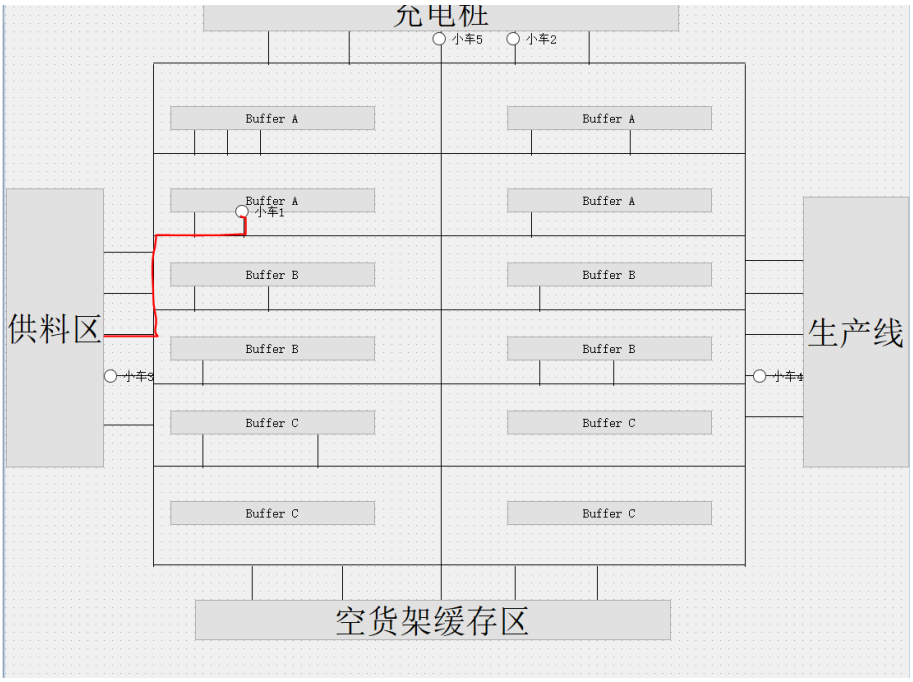
由仓库地图可知，此时只有小车 1 是空闲的，于是安排它去供料区



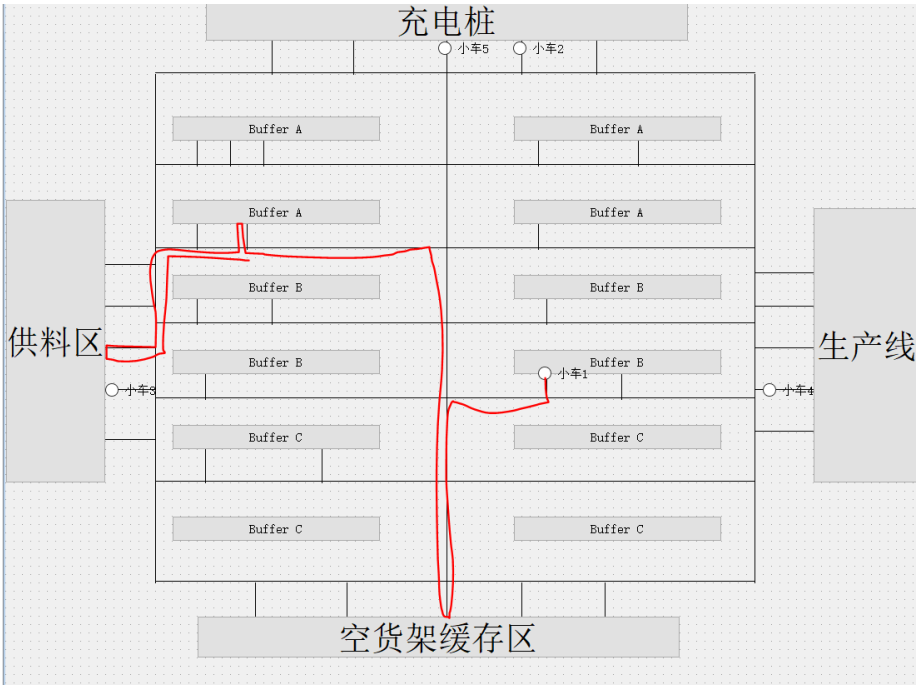
如下所示，走的是最短路径。小车直接去对应的供料区取物料



然后它将送到缓存区，走的也是最短路径。



另一种情况是需要先需要去空货架缓存区取空货架，然后再执行取物料操作。路径如图下所示，也是最短路径



可以看到小车 1 的情况

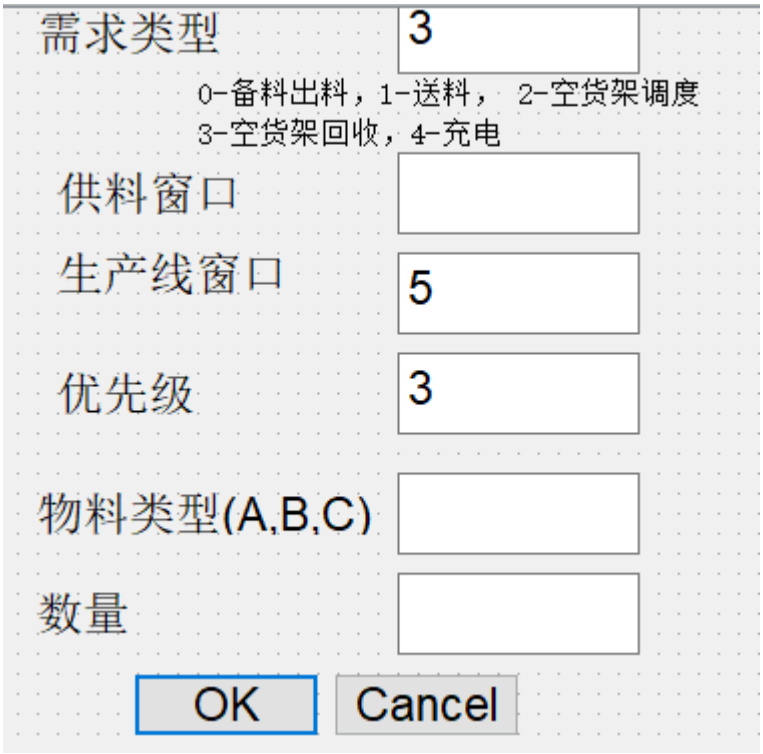
电量会减少，状态会改变

以上情况说明添加任务之后是按照最优的小车和最优的路径来完成调度的，验证了调度的完整和正确

3.2.3 任务管理测试

这里和之前理论分析的假设不同，在这里，优先级是用户输入的，便于实验方便

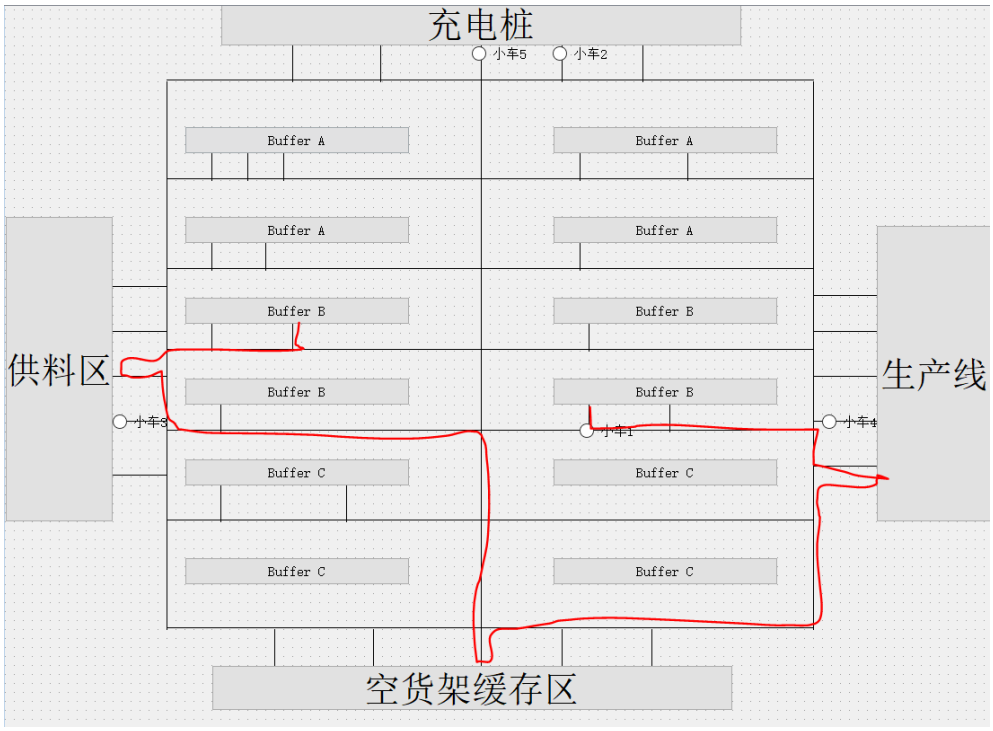
假设在前面的情况下，插入一个优先级更高的空货架回收任务，那么小车就会先到生产线回收空货架，再进行别的任务。



此时任务表显示

任务表

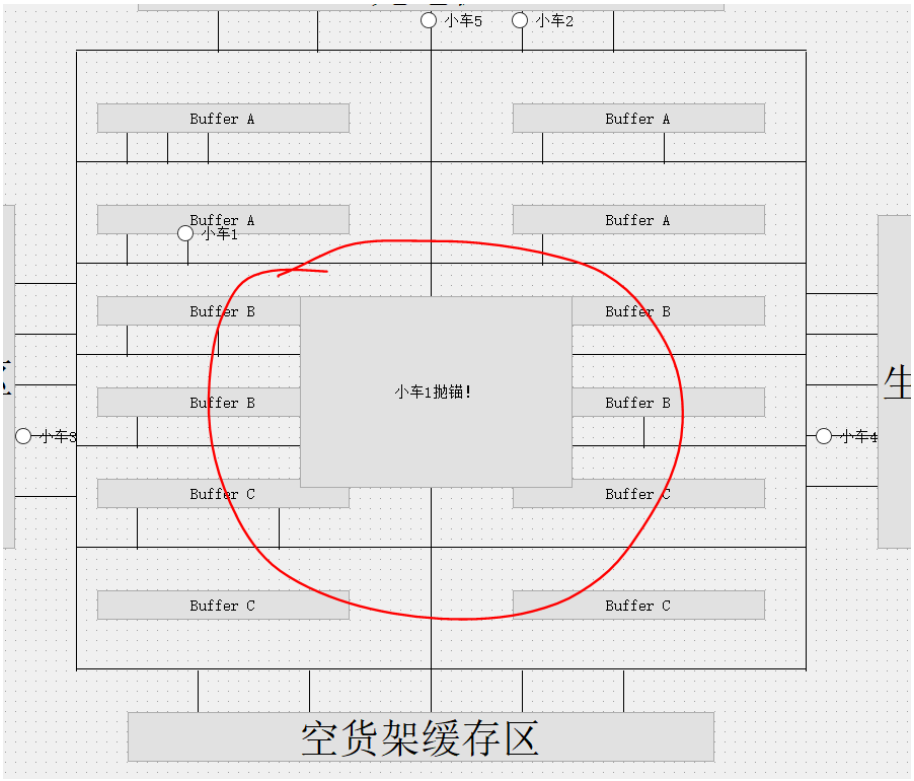
	到达时间	任务名称	优先级	完成情况
1		空货架回收	3	进行中
2		送料	0	进行中
3		备料出料	1	进行中
4		备料出料	1	进行中



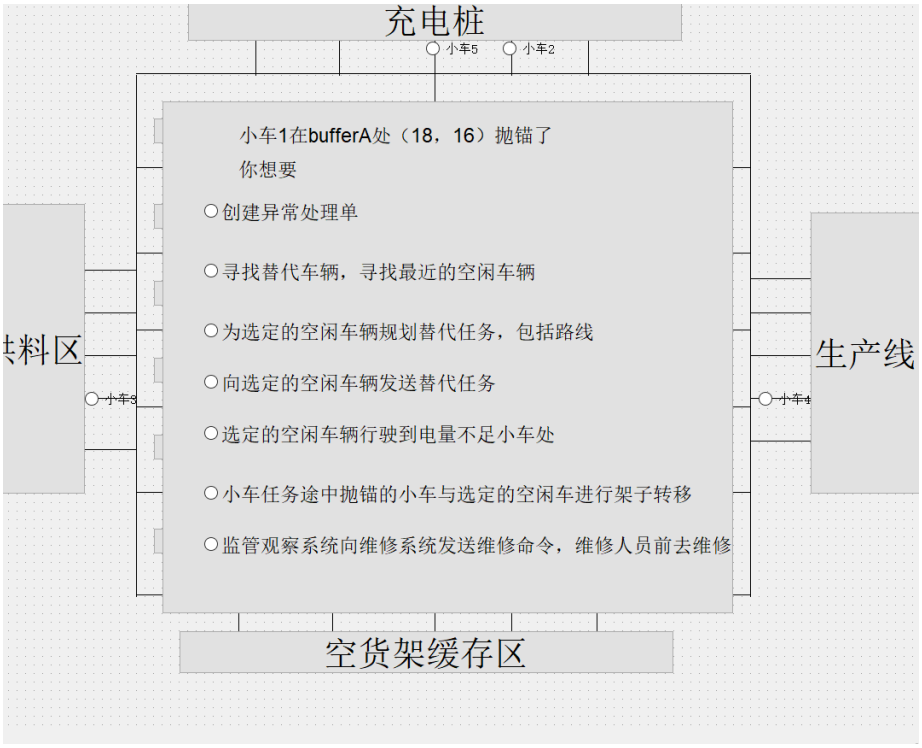
可以看到，同一个小车虽然先被分配了备料出料的任务，但是却先执行了优先级更高的空货架回收任务，可以验证是存在任务管理机制的。

3.2.4 故障测试

这里人工的方法假设小车 1 抛锚了，会输出错误信息



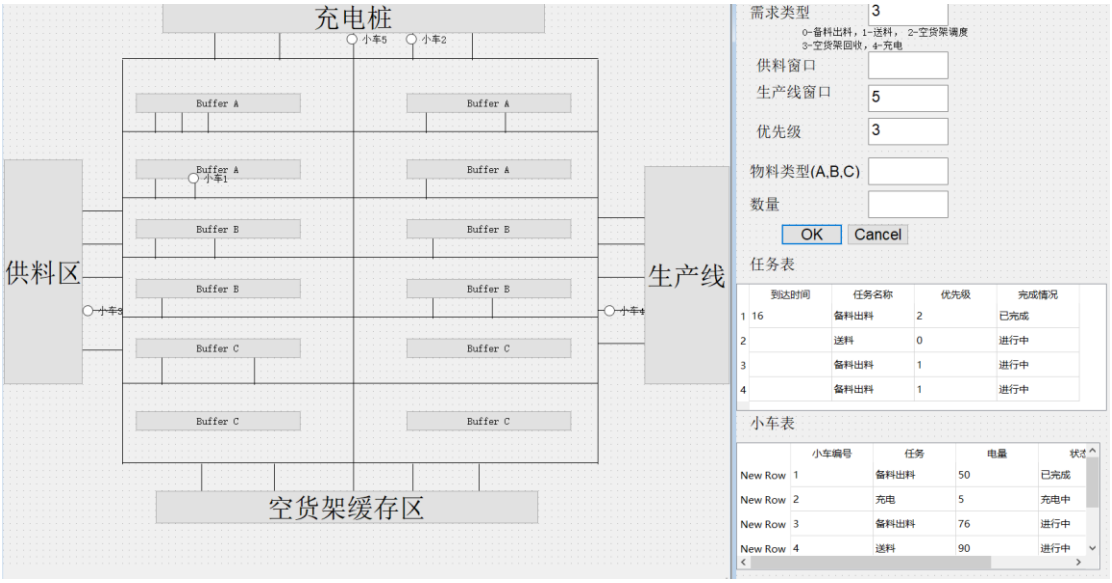
然后会输出预设的各种选项



这里因为只是简便实现，所以只是输出各种勾选框，并不做相应处理。

3.3 总结

以上测试可以简要验证所有基础功能的完整性
整体界面回顾:



第四章 总结

4.1 小组分工

总的来说，我们小组三位成员都对此次项目非常积极，大家合理分工，都很认真高效的完成任务，保证了高质量和高完成度的项目作业。除了老师规定的画图与设计，我们还完成了实现算法的代码编写，完成了部分功能，实现了可视化界面，进行了测试用例的设计，进行测试与分析等待。这都是我们合理分工，小组成员非常积极完成的成果。我们小组每个人合作紧密，相互帮助，每个人的工作量大致相同。下面是三位成员的详细分工。

祁佳薇：需求规格说明书，用例模型（包括详细文本描述），设计规格说明书，实现代码的编写（scheduleClass.py，statClass,py），任务管理系统的算法设计，地图设计，文档综合整理

杨希希：类模型，包模型，部署模型，完善包模型，设计模式运用，实现代码的编写（scheduleClass.py，statClass.py，main.py），异常处理系统的设计与实现，测试用例的设计

胡纯浩：时序模型（时序图、活动图），组件模型，用户界面设计，实现代码的编写（scheduleClass.py，map.py），小车调度系统的设计与实现，测试用例的测试与分析

4.2 总结感悟

通过这次的项目，我们将软件工程课程中学到的各种知识点灵活的运用到实战项目中。这次加深了我们对各个方面的认知，也从了解程度升级到了运用程度，受益匪浅。比如学会了对各种面向对象的各种图的绘制，加深了对面向对象的理解；将各种设计模式融合在设计的方案中，提高了设计的抽象层度和可用度；学习了难度较大的调度算法，自己设计处调度系统，任务管理系统和异常处理系统；编写出可以实现部分功能的代码；对代码进行测试与分析等等。

在这次的小组合作中，各个成员也收获了一起开发软件系统的团队精神，每个人都积极参与，合理分工，互帮互助。我们为这个项目付出了相当多的时间与精力，最后超额完成的成果也是令我们很满意的。