

【124智慧物料采购平台】开发文档v2.0

前言：可以完全将问题抽象成生产者消费者问题

完善了开发文档，重构了部分代码，修改了变量名称，可读性更高。

平台逻辑

1. 初始化地图阶段(readMapOk): 将所有平台状态信息读入 对于**复合生产型工作台** 我们会生成对应的生产任务 加入到任务队列 然后修改平台的标识位 **【是否发布任务】** 为true 防止重复发布任务 **【同时需要在Task数据结构中存上任务对应的平台】**
2. 每一帧的输入数据处理：**更新平台信息(readFrameOk)**: 更新机器人与平台信息。特别的，对于平台信息，我们需要针对复合型工作台 **【4-7】** 做更新，检查是否能再发布 **【生产型任务】**， **【fetch型任务】**。
 - 若产品格为1并且平台还没发布fetch型任务 则发布**fetch型任务**，由于platform数据结构中存储了 **【需要该产品的平台的队列】**，直接将队头元素
- 3.

任务队列逻辑

1. 优先级问题
 - fetch型任务优先级

优先级列表
Product

2. 任务分解具体细节

采用【懒分解策略】，比如初始任务队列为队头【7,4,5,6...】，对于这些复合任务，当机器人真正有空去领取任务的时候，才进行分解，减少每帧的处理任务。

任务队列分解策略
(懒分解)

1.
任务
7(curid=xxx)
├── 任务4(-1)
├── 任务5(-1)
└── 任务6(curid=-1)

4,5,6子任务入队，设置相应优先级
并且Task数据结构中修改
rootTaskPlatformId为父任务的
curTaskPlatformId

2.
任务
7(curid=xxx)
├── 任务4(xxxx)
├── 任务5(-1)
└── 任务6(curid=-1)
├── 任务1(-1)
└── 任务2(-1)

访问任务队列，发现队头任务还是复合任务，
因此继续分解，发现任务四curTaskPlatformId=-1,由于分解任务需要
子任务中存储父任务实际的平台编号，因此利用算法确定好4对应的平
台编号xxxx，然后在xxxx中存储rootId=xxx即可。此时可以分解任务
4，为1，2。
1中存储rootId=xxxx，2中存储rootId=xxxx。加入优先队列。
此时优先队列队头为原子任务，可以交付给机器人0，机器人拿到原子
任务，根据路径算法确定1的位置pS，然后任务中的
rootTaskPlatformId则是pE，
则Robot数据结构中更新pS, pE

Task数据结构
int rootTaskPlatformId; // 父任务的平台编号. -1为没有指定
int: curTaskPlatformId; // 当前任务的平台编号. -1为没有指定

Robot数据结构
int pS; // 买材料所需要去的平台 -1代表没有指定
int pE; // 卖材料所需要去的平台 -1代表没有指定
// -1为机器人空闲，否则则为pS | pE其中之一【这个字段只是为了兼容】
int targetPlatformIndex;
boolean status; // true为卖操作; false为买操作

考虑机器人的动作：对于任意阶段，机器人若是去买东西，那么一定要卖东西；机器人若是去卖东西，那么先前一定经过了买东西的阶段。
因此，可以理解机器人都是在做买卖，可以把买卖抽象成一次操作。每次确定一买一卖的目的地

机器人视角

每帧迭代4个可用的机器人

1.若机器人空闲【待分配】

查找任务队列

队头取任务，若为复合任务，则调用分解任务函数。

取到的任务类型：都是买卖。区别在于

- 任务没有指定卖材料的目的地：比如生产7号材料平台，待7号材料生产出来的时候，平台会发布一个fetch任务，这类型的任务就没有目的地，机器人取到此类任务，利用算法规划一个卖材料的目的地即可。
- 任务指定卖材料的目的地：比如生产7号材料分解出来的4号任务，该4号任务的数据结构Task内就存储了rootTaskPlatformId，即卖材料的目的地。

若任务队列为空

说明该地图没有复合型生产工作台。那么只需要考虑买1，2，3 往9号类型平台输送即可。

2.若机器人繁忙【可能需要更新相关数据结构】

若附近工作台id = targetPlatformIndex

若为买操作

数据结构

Platform数据结构增加的成员

```
private boolean isAssignProductTask; //是否发布生产任务
private boolean isAssignFetchTask; //是否发布取的任务
private boolean isChoosedForProduct; //是否被选择作为某些任务的父平台
private Queue<Integer> platformWhichNeedProductQueue; //需要本平台产品的平台，表现为一个队列，
按照请求该产品的顺序排队
```

Task数据结构增加的成员

```
// 成员变量
// curTaskPlatformId 完全等同于 platformIdForBuy
// rootTaskPlatformId 完全等同于 platformIdForSell
// 这样定义这是为了语义性更好
private boolean isAtomic; // 任务是否是原子任务【1, 2, 3】or 【Sell】
private int curTaskPlatformId; // 平台角度： 当前任务对应的平台id 【-1表示未指定】
private int rootTaskPlatformId; // 平台角度： 父任务对应的平台id 【-1表示未指定】
private int platformIdForBuy; // 机器人角度： 机器人领取任务后，需要去哪里买材料【-1表示未指定】
private int platformIdForSell; // 机器人角度： 机器人领取任务后，需要去哪里卖材料 【-1 表示未指定】
private int priority; // 任务优先级
private int taskNum; // 任务编号[1, 7]
```

Robot数据结构增加的成员

```
// 所有的操作都可以看成是一个买卖，没有单纯的买，也没有单纯的卖
// 比如当前机器人是买途，targetPlatformIndex就代表买途需要去的平台 nextTargetPlatformIndex就代表
卖途需要去的平台

private int nextTargetPlatformIndex; // 下一个目的地
private int targetPlatformIndex; // 目标工作台所在的数组的下标
```

优化思路-TODO

1. 领取任务问题，我们现在是遍历4个机器人，然后对于空闲的机器人，让其去**任务队列**领取任务。但是这样的
一个
坏处是**队头任务**分配给**当前遍历到的机器人**也许不是最优的。所以不妨换个角度思考，若机器人空闲，则取出
任务队列队头元素

2. 任务对应平台的确定算法。比如7号任务分解成了4, 5, 6号任务, 当要确定4号任务对应的平台时, 我们此前的算法是寻找距离7最近的4, 这样的坏处是有可能4号所需材料1, 2离4很远。
因此, 不妨在**初始化阶段**记录距离每一个4号平台最近的1, 2; **同样的**, 记录距离每一个5号平台最近的1, 3;
将距离4号平台最近的1, 2到该4号平台的距离之和记为权重
然后当**7号任务要确定4号任务的平台时**, 找到所有4号台, 找到 (**7号台到枚举到的4号台的距离 + 4号台的权重**) 最小的可用平台, 作为选中的平台。

3. 针对于单独的4, 5, 6, 即不是由7分解出来的4, 5, 6, 维护一个生产队列。因为单独的4, 5, 6发布的任务, 是没有rootTaskPlatformId属性的, 即**不知道生产出来的材料送去哪里**。所以我们可以将单独4, 5, 6发布的任务, 在其完成后, 放到**【生产队列】**中。如何利用这个生产队列? 例如某一时刻需要确定由7号任务分解而来的4号任务, 我们首先会在**【生产队列】**中查找, 若有, 则可直接在生产队列中领取4号成品材料。若没有, 则正常逻辑, 确定4号任务对应平台, 继续分解4号任务。

4. 最后帧数机器人不应该再接买卖**【此前实现过, 可以搬用】**