public override bool CreateToMJob(IAutoUnit unit, string RobotAction, int Position)

{

try

{

var jobList = \_dataAccess.GetMissionByTarget(unit.MachineID);

//如果该机台存在任何正在来料的任务

var toMJob = jobList.FirstOrDefault(x => x.MCSMissionType == MCSMissionType.LToM || x.MCSMissionType == MCSMissionType.MToM);

if (toMJob == null)

{

string missionStop = unit.GetPropertyValue(PropertyDes.MissionStop);

if (!string.IsNullOrEmpty(missionStop) && missionStop == "true")

{

aLogger.Info($"[{unit.MachineName}]停止前置任务生成 取消创建任务", "MissionCreateControl");

return false;

}

var newJob = new DtoJob

{

MissionID = Guid.NewGuid().ToString(),

Target = unit.MachineID,

MachineName = unit.MachineName,

CreateTime = DateTime.Now,

MissionStatus = !string.IsNullOrEmpty(unit.GetShelfOnMachine()) ? string.Empty : MissionStatus.Create,

Priority = unit.GetPropertyValue(PropertyDes.Priority),

MCSMissionType = MCSMissionType.LToM,

Area = unit.GetPropertyValue(PropertyDes.Scope),

IsCallback = false

};

AddTrackAction(newJob, unit, RobotAction, Position);//在任务中添加轨道动作

AddJobType(newJob, unit);//在任务中添加任务类型

AddPodType(newJob);//在任务中添加货架类型

\_dataAccess.AddMission(newJob);

aLogger.Info($"[{unit.MachineName}]已创建任务:{newJob.MissionID}", "CreateJob");

Task.Run(() =>

{

\_machineManager.ModifyMachineTrackStatus(unit.MachineID, RobotAction, "T");

});

}

else

{

if (string.IsNullOrEmpty(toMJob.MissionStatus))

{

// 接驳台没有绑定货架

if (string.IsNullOrEmpty(unit.GetShelfOnMachine()))

{

aLogger.Info($"[{unit.MachineName}]更新任务[{toMJob.MissionID}]状态:Create", "MissionStatus");

\_dataAccess.UpdateMissionStatus(toMJob.MissionID, MissionStatus.Create);

}

else

{

if (string.IsNullOrEmpty(toMJob.Reason))

{

SendMissionStatus(toMJob.Target, "2");

\_dataAccess.UpdateMissionReason(toMJob.MissionID, $"接驳台有货架[{unit.GetShelfOnMachine()}]");

}

}

}

}

}

catch (Exception ex)

{

aLogger.Info($"[{unit.MachineName}]报错{ex.StackTrace},{ex.Message}", "CreateJob");

}

return false;

}

public override bool CreateJob(TrackMessage message, IAutoUnit unit)

{

try

{

//设备ID

var machineID = message.MachineID;

//轨道上层还是下层

var position = message.Position;

//保存接驳台屏蔽记录

//SaveDockingShieldHistory(unit,message.IsShield);

var warning = GetWarning(message, unit);

if (!string.IsNullOrEmpty(warning))

aLogger.Info($"[{unit.MachineName}]warning:{warning}", "CreateJob");

var jobList = \_dataAccess.GetMissionByTarget(unit.MachineID);

//如果该机台存在任何正在来料的任务

var toMJob = jobList.FirstOrDefault(x => x.MCSMissionType == MCSMissionType.LToM || x.MCSMissionType == MCSMissionType.MToM);

if (toMJob == null)

{

string missionStop = unit.GetPropertyValue(PropertyDes.MissionStop);

if (!string.IsNullOrEmpty(missionStop) && missionStop == "true")

{

aLogger.Info($"[{unit.MachineName}]停止前置任务生成 取消创建任务", "MissionCreateControl");

return false;

}

//更新接驳台允许搬来/搬离状态

ReferenceMachinesStatus(message, unit);

var action = string.Empty;

if (position == 1)

{

//第一层轨道动作

action = unit.GetPropertyValue(PropertyDes.FirstLayer);

}

else if (position == 2)

{

//第二层轨道动作

action = unit.GetPropertyValue(PropertyDes.SecondLayer);

}

var newJob = new DtoJob

{

MissionID = Guid.NewGuid().ToString(),

Target = unit.MachineID,

MachineName = unit.MachineName,

CreateTime = DateTime.Now,

MissionStatus = !string.IsNullOrEmpty(warning) ? string.Empty : MissionStatus.Create,

Priority = unit.GetPropertyValue(PropertyDes.Priority),

MCSMissionType = MCSMissionType.LToM,

Area = unit.GetPropertyValue(PropertyDes.Scope),

IsCallback = false,

BoatCode= message.BoatCode

};

AddTrackAction(newJob, unit, action, position);//在任务中添加轨道动作

AddJobType(newJob, unit);//在任务中添加任务类型

AddPodType(newJob);//在任务中添加货架类型

AddPodStatus(newJob);

\_dataAccess.AddMission(newJob);

aLogger.Info($"[{unit.MachineName}]已创建任务:{newJob.MissionID}", "CreateJob");

}

else

{

//任务存在状态

if (string.IsNullOrEmpty(toMJob.MissionStatus))

{

// 满足Release 任务的条件，改变任务状态为Create

if (string.IsNullOrEmpty(warning))

{

aLogger.Info($"[{unit.MachineName}],更新任务[{toMJob.MissionID}]状态:Create", "CreateJob");

\_dataAccess.UpdateMissionStatus(toMJob.MissionID, MissionStatus.Create);

}

else

{

if (toMJob.Reason != warning)

{

var info = $"接驳台正在对接货架[{unit.GetShelfOnMachine()}]";

if (warning == info) // 向接驳台发送报警信息

{

//SendMissionStatus(toMJob.Target, "2");

}

\_dataAccess.UpdateMissionReason(toMJob.MissionID, warning);

aLogger.Info(warning, "CreateJob");

}

}

}

//任务无状态

else

{

//更新接驳台允许搬来/搬离状态

ReferenceMachinesStatus(message, unit);

}

}

}

catch (Exception ex)

{

aLogger.Info($"[{unit.MachineName}]报错{ex.StackTrace},{ex.Message}", "CreateJob");

}

return false;

}

/// <summary>

/// 预调度处理

/// </summary>

/// <param name="message"></param>

/// <param name="unit"></param>

public override void CheckPreScheduling(TrackMessage message, IAutoUnit currentUnit)

{

try

{

//未设置阈值视为关闭

string limit = currentUnit.GetPropertyValue(PropertyDes.PreSchedulingLimit);

if (string.IsNullOrEmpty(limit))

return;

//如果已经发起过预调度

string PreScheduling = currentUnit.GetPropertyValue(PropertyDes.PreScheduling);

if (PreScheduling == "true")

return;

////如果该机台存在一个完成的前置任务 表示等待回调

//DtoJob dtoJob = \_dataAccess.GetMissionByTarget(currentUnit.MachineID).Where(x => (x.MCSMissionType == MCSMissionType.LToM || x.MCSMissionType == MCSMissionType.MToM) && x.MissionStatus == MissionStatus.Finish).FirstOrDefault();

//if (dtoJob == null)

//{

// aLogger.Info($"[{currentUnit.MachineName}]预调度分析,不存在前置任务 继续处理其它信号 阈值:{limit},实时轨道数量:{message.Count}", "PreScheduling");

// return;

//}

//查询当前任务的回调任务

var dtoJob\_callBack = \_dataAccess.GetMissionByTarget(currentUnit.MachineID).FirstOrDefault(x => x.IsCallback);

//已经产生了回调任务并且已经被接收

if (dtoJob\_callBack != null && (dtoJob\_callBack.MissionStatus == MissionStatus.Accepted || dtoJob\_callBack.MissionStatus == MissionStatus.Outbin || dtoJob\_callBack.MissionStatus == MissionStatus.Finish))

{

aLogger.Info($"[{currentUnit.MachineName}]存在回调任务且已开始执行", "PreScheduling");

return;

}

//预调度触发阈值

var preSchedulingLimit = Convert.ToInt32(limit);

//倒计时(秒)

var PreSchedulingCountdown = currentUnit.GetPropertyValue(PropertyDes.PreSchedulingCountdown);

var MissionType = currentUnit.GetPropertyValue(PropertyDes.MissionType);

aLogger.Info($"[{currentUnit.MachineName}]预调度分析,阈值:{preSchedulingLimit},实时轨道数量:{message.Count}", "PreScheduling");

//达到阈值 进行预调度

if (preSchedulingLimit <= message.Count)

{

aLogger.Info($"[{currentUnit.MachineName}]预调度分析,阈值:{preSchedulingLimit},实时轨道数量:{message.Count} 提前呼叫AGV", "PreScheduling");

List<HKMachineInfo> hKMachineInfos = new List<HKMachineInfo>()

{

new HKMachineInfo()

{

taskCountDown =PreSchedulingCountdown,

webCode=currentUnit.WbCode,

taskTyp=MissionType

}

};

HKsendMachineInfo hKsendMachineInfo = new HKsendMachineInfo()

{

reqCode = Guid.NewGuid().ToString("N"),

interfaceName = "sendMachineInfo",

webCodeList = hKMachineInfos

};

\_rcsClient.PreScheduling(hKsendMachineInfo);

//修改预调度属性为true 表示已经生成过预调度 不在生成(会在回调任务start时复位)

currentUnit.AddOrUpdateProperty(PropertyDes.PreScheduling, "true");

}

}

catch (Exception ex)

{

aLogger.Info($"{currentUnit.MachineName}检测是否需要预调度时发生异常{ex}", "PreScheduling");

}

}

/// <summary>

/// 接驳台完成信号处理

/// </summary>

/// <param name="unit"></param>

/// <param name="Position"></param>

public override void ContinueJob(IAutoUnit unit, int Position)

{

try

{

// 基础验证

//急停验证

var mcsControl = unit.GetPropertyValue(PropertyDes.MCSControlStatus);

if (!string.IsNullOrEmpty(mcsControl) && Convert.ToBoolean(mcsControl))

{

aLogger.Info($"[{unit.MachineName}]已经被屏蔽!", "ActionFinish");

return;

}

string callBackMissionStop = unit.GetPropertyValue(PropertyDes.CallBackMissionStop);

if (!string.IsNullOrEmpty(callBackMissionStop) && callBackMissionStop == "true")

{

aLogger.Info($"[{unit.MachineName}]停止回调任务生成 取消创建任务", "MissionCreateControl");

return;

}

var emergencyStop = unit.GetPropertyValue(PropertyDes.EmergencyStop);

if (!string.IsNullOrEmpty(emergencyStop) && emergencyStop == "true")

{

aLogger.Info($"[{unit.MachineName}]已经急停 触发完成信号无效 暂不产生回调任务", "ActionFinish");

return;

}

var podName = unit.GetShelfOnMachine();

if (string.IsNullOrEmpty(podName))

{

aLogger.Info($"[{unit.MachineName}]未绑定货架,无法产生搬离任务!", "ActionFinish");

return;

}

//已经产生了将货架搬离的任务

var job = \_dataAccess.GetMissionByTarget(unit.MachineID).FirstOrDefault(x => x.MCSMissionType == MCSMissionType.MToM || x.MCSMissionType == MCSMissionType.MToL);

if (job != null && job.PodCode == podName)

{

aLogger.Info($"[{unit.MachineName}]忽略完成信号,[{job.MissionID}]已经产生搬离[{podName}]的任务!", "ActionFinish");

return;

}

var job1 = \_dataAccess.GetMissionByPodcode(podName);

if (job1 != null && job1.PodCode == podName)

{

aLogger.Info($"[{unit.MachineName}]忽略完成信号,[{job1.MissionID}]已经产生搬离[{podName}]的任务!", "ActionFinish");

return;

}

// 若已经给了货架到位信号

var DtoPod = \_inventoryManager.GetPodLocationByName(podName);

if (DtoPod.Coordinate == unit.WbCode)// 货架已到位

{

unit.SetGoControlStatus("1");//变更搬离信号为可以搬离

}

// 根据完成信号，处理货架里的物料(9-15修改为路径决策后给货架添加包号)

//UpdatePackageInPod(unit);// 更新货架里的物料信息

//上料完成清空包号

ClearPackageInPod(unit);

//给货架重新赋值所在工艺段和循环

AddProcessAndScope(podName, unit);

// 防止同一个waitinghandle任务被2个完成信号锁定，导致任务路径被覆盖

lock (locker)

{

// 创建搬离任务

CreateMToJob(unit);

}

}

catch (Exception ex)

{

aLogger.Info($"{unit.MachineName}继续任务发生异常{ex}", "ActionFinish");

}

}

/// <summary>

/// 回调任务检索下工艺段MToM任务

/// </summary>

internal DtoJob GetWaitingforprocessJob(IAutoUnit unit, string action1, string action2)

{

try

{

DtoJob job\_waitingforProcess = null;

var mode = unit.GetPropertyValue(PropertyDes.MachineMode);

if (action1 == PropertyDes.GET || action2 == PropertyDes.GET)

{

if (mode != MachineMode.SpecialMode)// 设备处在正常模式

{

//1 判断toMachines是否存在put任务

//查询当前设备对应的上料设备组

var toMachines = unit.GetPropertyValue(PropertyDes.ToMachines);

string closeMachines = unit.GetPropertyValue(PropertyDes.CloseTargetMachines);

List<string> closeMachineList = closeMachines.Split(',').Where(x => !string.IsNullOrEmpty(x)).Distinct().ToList();

var notSpecialModeMachines = new List<string>();

var notSpecialModeMachines\_close = new List<string>();

if (!string.IsNullOrEmpty(toMachines))

{

string tomiachineMembers = \_dataAccess.GetGroupMembersByGroupName(toMachines);

var toMachineGroupMembers = tomiachineMembers.Split(',').Where(x => !string.IsNullOrEmpty(x)).ToList();

foreach (var machine in toMachineGroupMembers)

{

var targetUnit = \_machineManager.GetUnitByMachineID(machine);

string machineMode = targetUnit.proDict.ContainsKey(PropertyDes.MachineMode) ? targetUnit.proDict[PropertyDes.MachineMode] : MachineMode.Production;

if (machineMode != MachineMode.SpecialMode)// && targetUnit.GetMoveOffStatus() == "1")

{

var assStop = targetUnit.GetPropertyValue(PropertyDes.EmergencyStop);

if (!string.IsNullOrEmpty(assStop) && Convert.ToBoolean(assStop)) continue;// 关联设备处于急停状态，不能生成任务送到关联设备

notSpecialModeMachines.Add(machine);

}

}

}

if (closeMachineList.Count>0)

{

foreach (var machine in closeMachineList)

{

var targetUnit = \_machineManager.GetUnitByMachineID(machine);

string machineMode = targetUnit.proDict.ContainsKey(PropertyDes.MachineMode) ? targetUnit.proDict[PropertyDes.MachineMode] : MachineMode.Production;

if (machineMode != MachineMode.SpecialMode)// && targetUnit.GetMoveOffStatus() == "1")

{

var assStop = targetUnit.GetPropertyValue(PropertyDes.EmergencyStop);

if (!string.IsNullOrEmpty(assStop) && Convert.ToBoolean(assStop)) continue;// 关联设备处于急停状态，不能生成任务送到关联设备

notSpecialModeMachines\_close.Add(machine);

}

}

}

job\_waitingforProcess = \_dataAccess.GetWaitingForProcessPUTJob(notSpecialModeMachines,notSpecialModeMachines\_close);

}

}

else

{

//查询当前机台的关联机台 如果其对应的上料

var associatedMachine = unit.GetPropertyValue(PropertyDes.AssociatedMachine);

var assUnit = \_machineManager.GetUnitByMachineID(associatedMachine);

var assStop = assUnit.GetPropertyValue(PropertyDes.EmergencyStop);

if (!string.IsNullOrEmpty(assStop) && Convert.ToBoolean(assStop))

{

// 关联设备处于急停状态，不能生成任务送到关联设备

}

else

{

//关联设备无货架且允许搬来

if (string.IsNullOrEmpty(assUnit.GetShelfOnMachine())&& assUnit.GetGoControlStatus() == "1")

{

List<string> machines = new List<string>{associatedMachine};

job\_waitingforProcess = \_dataAccess.GetWaitingForProcessGetJob(machines);

}

}

}

return job\_waitingforProcess;

}

catch (Exception ex)

{

aLogger.Info($"回调添加路径检索下工艺段MToM发生异常{ex}", "AddRoutes");

return null;

}

}

/// <summary>

/// 为任务分配路径

/// </summary>

/// <param name="job"></param>

/// <param name="unit"></param>

/// <returns></returns>

public override DtoJob AddRoutes(DtoJob job, IAutoUnit unit)

{

try

{

string action = "";

var reason = string.Empty;

if (job.Action1 == PropertyDes.PUT || job.Action2 == PropertyDes.PUT)

action = PropertyDes.PUT;

else if (job.Action1 == PropertyDes.GET || job.Action2 == PropertyDes.GET)

action = PropertyDes.GET;

var machineMode = unit.GetPropertyValue(PropertyDes.MachineMode);

if (string.IsNullOrEmpty(machineMode)||machineMode==MachineMode.Production)

job.MachineMode = MachineMode.Production;

else

job.MachineMode = MachineMode.SpecialMode;

aLogger.Info($"任务[{job.MissionID}]开始分配路径,[{unit.MachineName}]任务类型[{job.MCSMissionType}]", "AddRoutes");

string mcshilpath = "";

//验证机台是否定线

if (string.IsNullOrEmpty(unit.GetPropertyValue(PropertyDes.Domain)))

{

aLogger.Info($"接驳台未定线!", "AddRoutes");

\_dataAccess.UpdateMissionReason(job.MissionID, $"接驳台未定线!");

return job;

}

if (job.MCSMissionType == MCSMissionType.MToL)

{

//回调任务以MergeAction为准

var mergeAction = unit.GetPropertyValue(PropertyDes.MergeAction);

if (mergeAction == PropertyDes.GET)

action = PropertyDes.GET;

//aLogger.Info($"[{job.MissionID}]开始添加路径", "AddRoutes");

mcshilpath = GetCallbackPath(unit, action, job);

if (string.IsNullOrEmpty(mcshilpath))

{

//GET任务CallBack无库位可用 验证下工艺段是否存在WatiForProcess任务 存在直接处理下工艺段的任务

if (action == PropertyDes.GET)

{

if (CheckPutCallbackJobMToMJob(job, unit, job.Action1, job.Action2))

return job;

}

reason = $"下工序无库位且下工艺段无待料机台";

if (job.Reason != reason)

{

aLogger.Info($"[{job.MissionID}]下工序无库位", "AddRoutes");

\_dataAccess.UpdateMissionReason(job.MissionID, reason);

SendMissionStatus(job.Target, "5");

}

return job;

}

}

else

{

var onlyInventory\_property = unit.GetPropertyValue(PropertyDes.OnlyInventory);

var onlyInventory = Convert.ToBoolean(string.IsNullOrEmpty(onlyInventory\_property) ? "false" : onlyInventory\_property);

//"onlyInventory\_property"属性为MToM开关

var paths = new List<MCSHKPositionInfo>();

//todo 查询任务机器人需要经过的路径

//如果是put任务 上料自动化要料

if (action == PropertyDes.PUT)

{

//上料任务验证是否限速

string speedControlCheckResultMessage = "";

bool speedControlCheck = CheckSpeedControl(unit, job,out speedControlCheckResultMessage);

if (speedControlCheck)

{

\_dataAccess.UpdateMissionReason(job.MissionID, speedControlCheckResultMessage);

return job;

}

onlyInventory = machineMode == MachineMode.SpecialMode || onlyInventory;

var start = GetPutStartLocationWithMode(unit, job);

if (start == null || start.ID == 0)

{

if (onlyInventory)

{

reason = $"只能从库区取料但无合适库位";

if (reason != job.Reason)

{

aLogger.Info($"[{job.MissionID}]只能从库区取料但无合适库位", "AddRoutes");

\_dataAccess.UpdateMissionReason(job.MissionID, reason);

SendMissionStatus(job.Target, "1");

}

return job;

}

//寻找源工艺段有无MTOL任务如果存在 修改终点到当前机台 处理完成

if (JobGraftingHandle(job, unit))

return job;

paths = new List<MCSHKPositionInfo>(){

//起点暂不确定等待GET条件触发后替换坐标点位

new MCSHKPositionInfo()

{

positionCode = "WAITING",

type="00",

positionType=HKPostionCodeType.Machine

},

new MCSHKPositionInfo()

{

positionCode = unit.WbCode,

type="00",

positionType=HKPostionCodeType.Machine,

action1=job.Action1,

action2=job.Action2

}

};

//归为待处理的PUT任务 等某个机台下料完成后 继续处理

job.MissionStatus = MissionStatus.WaitingForProcess;

mcshilpath = JsonConvert.SerializeObject(paths);

job.HKPositionInfos = JsonConvert.DeserializeObject<List<HKPositionInfo>>(mcshilpath);

job.Paths = mcshilpath;

job.MCSMissionType = MCSMissionType.MToM;

\_dataAccess.UpdatePayload(job);//更新任务上做的更改

if (reason != job.Reason)

{

\_dataAccess.UpdateMissionReason(job.MissionID, reason);

SendMissionStatus(job.Target, "1");

}

aLogger.Info($"任务[{job.MissionID}]设置为MTOM 起点待定.", "AddRoutes");

return job;

}

//起点明确

else

{

//找到目标带料货架 获取PUT任务终点

paths = new List<MCSHKPositionInfo>()

{

new MCSHKPositionInfo()

{

positionCode = start.Coordinate,

type="00",

inventoryShelveId = start.ShelvesID,

inventoryLocationId=start.ID,

positionType=HKPostionCodeType.InventoryLocation,

action1="",

action2=""

},

new MCSHKPositionInfo()

{

positionCode = unit.WbCode,

type="00",

positionType=HKPostionCodeType.Machine,

action1=job.Action1,

action2=job.Action2

}

};

//将库位占用 注: 任务完成后需要接触占用 否则无法继续使用

\_inventoryManager.InventoryLocationOccupy(start.ID, job.MissionID);

//解除该货架与机台的优先去向关系

aLogger.Info($"{start.ShelvesCode}解除与机台的优先去向关系", "InventoryManager");

\_inventoryManager.CancelPriorityMachineRelation(start.ShelvesID);

//清除库位的历史库区

\_inventoryManager.CancelHistoryAreaRelation(start.ID);

job.PodCode = start.ShelvesCode;

job.MCSMissionType = MCSMissionType.LToM;

//job.PodStatus =

}

}

//如果是get任务 下料自动化出料

else if (action == PropertyDes.GET)

{

var fromZone = unit.GetPropertyValue(PropertyDes.FromBuffers);

var closeLocations = unit.GetPropertyValue(PropertyDes.CloseLocation);

var start = \_inventoryManager.RecommendGetStartInventoryLocation(job.Target, fromZone, closeLocations);

//没有可用库位

if (start == null || start.ID == 0)

{

//如果可以从机台直接取货架

if (onlyInventory)

{

reason = $"只能从库区取料但无合适库位";

if (reason != job.Reason)

{

aLogger.Info($"[{job.MissionID}]只能从库区取料但无合适库位", "AddRoutes");

\_dataAccess.UpdateMissionReason(job.MissionID, reason);

SendMissionStatus(job.Target, "1");

}

return job;

}

//查询对应的机台是否存在未完成的任务(表示存在货架 可以稍作等待)

var associatedMachine = unit.GetPropertyValue(PropertyDes.AssociatedMachine);

var associatedMachinePutJob = \_dataAccess.GetToMMissionByMachineId(associatedMachine);

var associatedUnit = \_machineManager.GetUnitByMachineID(associatedMachine);

var podName = associatedUnit.GetShelfOnMachine();

if (!string.IsNullOrEmpty(podName))

{

paths = new List<MCSHKPositionInfo>(){

//起点暂不确定等待GET条件触发后替换坐标点位

new MCSHKPositionInfo()

{

positionCode = "WAITING",

type="00",

positionType=HKPostionCodeType.Machine

},

new MCSHKPositionInfo()

{

positionCode = unit.WbCode,

type="00",

positionType=HKPostionCodeType.Machine,

action1=job.Action1,

action2=job.Action2

}

};

job.MissionStatus = MissionStatus.WaitingForProcess;

mcshilpath = JsonConvert.SerializeObject(paths);

job.HKPositionInfos = JsonConvert.DeserializeObject<List<HKPositionInfo>>(mcshilpath);

job.Paths = mcshilpath;

job.MCSMissionType = MCSMissionType.MToM;

\_dataAccess.UpdatePayload(job);//更新任务上做的更改

reason = $"无可用库位,任务状态变为WaitforProcess 等待关联机台上料完成后直接送往该机台";

if (reason != job.Reason)

{

aLogger.Info($"任务{job.MissionID} 无可用库位 将任务类型修改为WaitforProcess 等待对应机台上料完成后直接送往该机台", "AddRoutes");

\_dataAccess.UpdateMissionReason(job.MissionID, reason);

SendMissionStatus(job.Target, "1");

}

aLogger.Info($"任务[{job.MissionID}]设置为MTOM 起点待定.", "AddRoutes");

return job;

}

else

{

reason = $"无合适库位且无关联设备";

if (reason != job.Reason)

{

aLogger.Info($"[{job.MissionID}]无合适库位且无关联设备", "AddRoutes");

\_dataAccess.UpdateMissionReason(job.MissionID, reason);

SendMissionStatus(job.Target, "1");

}

return job;

}

}

else

{

paths = new List<MCSHKPositionInfo>()

{

new MCSHKPositionInfo()

{

positionCode = start.Coordinate,

inventoryShelveId = start.ShelvesID,

inventoryLocationId=start.ID,

positionType = HKPostionCodeType.InventoryLocation,

type="00"

},

new MCSHKPositionInfo()

{

positionCode = unit.WbCode,

positionType = HKPostionCodeType.Machine,

type="00",

action1=job.Action1,

action2=job.Action2

}

};

//将库位占用 注: 任务完成后需要接触占用 否则无法继续使用

\_inventoryManager.InventoryLocationOccupy(start.ID, job.MissionID);

//解除该货架与机台的优先去向关系

aLogger.Info($"{start.ShelvesCode}解除与机台的优先去向关系", "InventoryManager");

\_inventoryManager.CancelPriorityMachineRelation(start.ShelvesID);

//清除库位的历史库区

\_inventoryManager.CancelHistoryAreaRelation(start.ID);

job.PodCode = start.ShelvesCode;

job.MCSMissionType = MCSMissionType.LToM;

}

}

mcshilpath = JsonConvert.SerializeObject(paths);

}

job.HKPositionInfos = JsonConvert.DeserializeObject<List<HKPositionInfo>>(mcshilpath);

job.Paths = mcshilpath;

reason = $"成功分配路径:{job.HKPositionInfos[0].positionCode}-{job.HKPositionInfos[1].positionCode}, 货架号为[{job.PodCode}]";

\_dataAccess.UpdateMissionReason(job.MissionID, reason);

if ((job.MCSMissionType == MCSMissionType.MToL))

{

//路径添加后给货架赋包号信息

AddPackageInPod(unit);

}

aLogger.Info($"[{job.MissionID}],{reason}", "AddRoutes");

return job;

}

catch (Exception ex)

{

var reason = $"任务[{job.MissionID}]分配路径发生异常:{ex}";

aLogger.Info(reason, "AddRoutes");

\_dataAccess.UpdateMissionReason(job.MissionID, reason);

return job;

}

}

/// <summary>

/// 查询指向当前设备的临近设备

/// </summary>

/// <returns></returns>

private string GetFromCloseMachines(string machineId) {

try

{

string result = "";

List<DtoMachineProperty> allProp = \_dataAccess.GetPropertiesByName(PropertyDes.CloseTargetMachines);

foreach (var item in allProp)

{

if (item.Value.Contains(machineId))

{

result += $"{item.MachineID},";

}

}

return result;

}

catch (Exception ex)

{

return "";

}

}

/// <summary>

/// 任务移植

/// </summary>

/// <param name="jobInfo"></param>

/// <param name="unit"></param>

/// <returns></returns>

public override bool JobGraftingHandle(DtoJob jobInfo, IAutoUnit unit) {

try

{

//aLogger.Info($"[{unit.MachineName}]-[{jobInfo.MissionID}]上料时无料可用尝试寻找对应工艺段的MToL任务并移植", "JobGraftingHandle");

//源设备组

string fromMachines = unit.GetPropertyValue(PropertyDes.FromMachines);

string changeEndPointLimit = unit.GetPropertyValue(PropertyDes.ChangeEndPointLimit);

//未配置不做逻辑处理

if (string.IsNullOrEmpty(changeEndPointLimit)|| changeEndPointLimit=="0")

return false;

int limit = Convert.ToInt32(changeEndPointLimit);

DtoJob sourceJob = null;

string members = \_dataAccess.GetGroupMembersByGroupName(fromMachines);

aLogger.Info($"[{unit.MachineName}]-[{jobInfo.MissionID}]设置的移植超时阈值为{limit} 源设备组成员为{members}", "JobGraftingHandle");

List<string> membersList = members.Split(',').ToList();

if (membersList.Count == 0)

return false;

//查询源设备的送往库区的任务 状态为发送成功 被接收 走出储位（非特殊模式任务）

List<DtoJob> jobsOfFromMachines = \_dataAccess.GetFromMachinesJobs(membersList).Where(x=>x.MachineMode==MachineMode.Production).ToList();

//查询outBin的任务

List<DtoJob> jobsOfOutBin = jobsOfFromMachines.Where(x => x.MissionStatus == MissionStatus.Outbin && DateTime.Now.Subtract(x.StartTime).Seconds <= limit).ToList();

if (jobsOfOutBin.Count > 0)

sourceJob = jobsOfOutBin.OrderBy(x => DateTime.Now.Subtract(x.StartTime).Seconds).FirstOrDefault();

//仍为空 再次查询其它状态的任务

if (sourceJob == null)

{

List<DtoJob> jobsOfAcceptAndSend = jobsOfFromMachines.Where(x => x.MissionStatus != MissionStatus.Outbin).ToList();

sourceJob = jobsOfOutBin.OrderBy(x => x.CreateTime).FirstOrDefault();

}

//修改终点

if (sourceJob!=null)

{

aLogger.Info($"[{unit.MachineName}]-[{jobInfo.MissionID}]找到目标可移植任务[{sourceJob.MissionID}] 任务状态[{sourceJob.MissionStatus}] 任务原路径:[{sourceJob.Paths}] 货架号[{sourceJob.PodCode}]", "JobGraftingHandle");

List<MCSHKPositionInfo> pathInfo = JsonConvert.DeserializeObject<List<MCSHKPositionInfo>>(sourceJob.Paths);

MCSHKPositionInfo start = pathInfo.First();

if (string.IsNullOrEmpty(start.positionCode))

return false;

string[] newPathArray = new string[2];

newPathArray[0] = start.positionCode;

newPathArray[1] = unit.WbCode;

var requestInfo = new HKMissionChangeDestination()

{

reqCode = Guid.NewGuid().ToString("N"),

interfaceName = "changeTaskDest",

podCode = sourceJob.PodCode,

positionPath = newPathArray

};

aLogger.Info($"[{unit.MachineName}]-[{jobInfo.MissionID}]向RCS发起修改终点 入参:{JsonConvert.SerializeObject(requestInfo)}", "JobGraftingHandle");

var strReply = \_rcsClient.SendChangeDestination(requestInfo);

aLogger.Info($"[{unit.MachineName}]-[{jobInfo.MissionID}]向RCS发起修改终点 出参:{JsonConvert.SerializeObject(strReply)}", "JobGraftingHandle");

//修改成功

if (strReply.code== "0")

{

//删除原任务

//修改sourceJob的任务类型和任务信息 （改为MToM 路径改为目标机台）

\_dataAccess.DeleteMission(jobInfo.MissionID);

//取消占用

\_dataAccess.CancelOccupy(sourceJob.MissionID);

DtoShelvesResponseDTO dtoShelvesResponseDTO = \_dataAccess.GetPodByName(sourceJob.PodCode);

List<MCSHKPositionInfo> newPathInfo = new List<MCSHKPositionInfo>();

newPathInfo.Add(start);

var action1 = unit.GetPropertyValue(PropertyDes.FirstLayer);

var action2 = unit.GetPropertyValue(PropertyDes.SecondLayer);

MCSHKPositionInfo newEnd = new MCSHKPositionInfo() {

positionType = HKPostionCodeType.Machine,

inventoryShelveId= dtoShelvesResponseDTO.ID,

positionCode = unit.WbCode,

action1= action1,

action2=action2

};

newPathInfo.Add(newEnd);

//修改源任务信息

\_dataAccess.UpdateMissionInfoAndType(sourceJob.MissionID,MCSMissionType.MToM,JsonConvert.SerializeObject(newPathInfo),$"任务途中修改终点为设备[{unit.MachineName}]",unit.MachineID,unit.MachineName);

DtoJob afterChange = \_dataAccess.GetMissionByID(sourceJob.MissionID);

//给当前机台发送信号

InformAboutJobSend(unit, afterChange);

//发送小车号

SendMissionStatus(unit.MachineID, afterChange.RobotName);

//关闭允许搬走

if (unit.GetGoControlStatus() == "1")

{

//设置为不允许搬走

unit.SetGoControlStatus("0");

}

aLogger.Info($"[{unit.MachineName}]-[{jobInfo.MissionID}]终点修改成功 删除任务,取消任务{afterChange.MissionID}对库位的占用 给设备[{unit.MachineName}]发送货架号和小车号", "JobGraftingHandle");

return true;

}

else

{

//修改失败 放弃修改

return false;

}

}

return false;

}

catch (Exception ex)

{

aLogger.Info($"库区无料获取MTOL任务尝试修改终点时发生异常:{ex}", "JobGraftingHandle");

return false;

}

}

/// <summary>

/// put回调任务无路径时检查下工艺段MToM任务 继续执行

/// </summary>

private bool CheckPutCallbackJobMToMJob(DtoJob currentJob, IAutoUnit unit, string action1, string action2)

{

try

{

var podCode = unit.GetShelfOnMachine();

//获取下工艺段WatingforProcess任务

DtoJob nextProcessMToMjob = GetWaitingforprocessJob(unit, action1, action2);

if (nextProcessMToMjob != null)

{

aLogger.Info($"GET任务[{currentJob.MissionID}]回调无路径,发现待料任务[{nextProcessMToMjob.MissionID}] 完成当前任务并处理待料任务", "CheckPutCallbackJobMToMJob");

//完成当前任务

FinishJob(currentJob);

//处理下工艺段的MToM任务

nextProcessMToMjob.Paths = nextProcessMToMjob.Paths.Replace("WAITING", unit.WbCode);

nextProcessMToMjob.PodCode = podCode;

nextProcessMToMjob.MissionStatus = MissionStatus.MESProcessed;

\_dataAccess.UpdatePayload(nextProcessMToMjob);//更新任务上做的更改

//路径添加后给货架赋包号信息

UpdatePackageInPod(unit);

return true;

}

else

{

//aLogger.Info($"GET任务[{currentJob.MissionID}]回调无路径未发现待料任务", "CheckPutCallbackJobMToMJob");

return false;

}

}

catch (Exception ex)

{

aLogger.Info($"GET回调任务无库位可用时验证目标设备组MToM任务过程发生异常：{ex}", "CheckPutCallbackJobMToMJob");

return false;

}

}

private string GetCallbackPath(IAutoUnit currendMachine, string RobotAction, DtoJob job)

{

var paths = new List<MCSHKPositionInfo>()

{

new MCSHKPositionInfo()

{

positionCode = currendMachine.WbCode,

type="00",

positionType=HKPostionCodeType.Machine

}

};

var toZone = currendMachine.GetPropertyValue(PropertyDes.ToBuffers);

//设备模式

var machineMode = currendMachine.GetPropertyValue(PropertyDes.MachineMode);

//如果是下料并且设备正在洗花篮模式

//if (RobotAction == PropertyDes.GET && machineMode == MachineMode.CleanBasket)

//{

// //获取该机台对应的上料机

// var assMachine = currendMachine.GetPropertyValue(PropertyDes.AssociatedMachine);

// if (!string.IsNullOrEmpty(assMachine))

// {

// var assMachie = \_machineManager.GetUnitByMachineID(assMachine);

// //源库区使用目标库区 清洗自己产生的空花篮

// toZone = assMachie.GetPropertyValue(PropertyDes.FromBuffers);

// }

// else

// {

// job.Reason = $"回调任务{job.MissionID}{currendMachine.MachineName}进入花篮清洗模式 但是未配置AssociatedMachine属性(关联设备)";

// aLogger.Info(job.Reason, "AddRoutes");

// }

//}

if (RobotAction == PropertyDes.GET && machineMode == MachineMode.SpecialMode)

{

//获取到该机台清线模式下专用的库区

toZone = currendMachine.GetPropertyValue(PropertyDes.FromBuffers\_clearLine);

}

//如果是PUT 送料完成 寻找关联设备的fromBuffer作为空货架库区

else if (RobotAction == PropertyDes.PUT)

{

//找到上料机台的关联机台 也就是其下料机台

var assMachines = currendMachine.GetPropertyValue(PropertyDes.AssociatedMachine);

IAutoUnit assMachineInfo = \_machineManager.GetUnitByMachineID(assMachines);

toZone = assMachineInfo.GetPropertyValue(PropertyDes.FromBuffers);

}

//如果是下料完成 寻找下个工艺段的FromMachines作为Tobuffers

else if (RobotAction == PropertyDes.GET)

{

//找到目标设备组 以其中某一台的专属库区作为ToBuffers

var toMachines = currendMachine.GetPropertyValue(PropertyDes.ToMachines);

List<string> tomachineList = \_dataAccess.GetGroupMembersByGroupName(toMachines).Split(',').ToList();

string machineId = tomachineList.FirstOrDefault();

//下工艺段无设备

if (string.IsNullOrEmpty(machineId) || string.IsNullOrEmpty(toMachines))

{

//tobuffers仍然为空

if (string.IsNullOrEmpty(currendMachine.GetPropertyValue(PropertyDes.ToBuffers))) {

var reason = $"任务[{job.MissionID}]分配路径失败,未找到下工艺段设备(检查定线).且未设置默认库区";

aLogger.Info(reason, "AddRoutes");

\_dataAccess.UpdateMissionReason(job.MissionID, reason);

return "";

}

//使用本设备设置的tobuffers

toZone = currendMachine.GetPropertyValue(PropertyDes.ToBuffers);

}

else

{

IAutoUnit nextMachine = \_machineManager.GetUnitByMachineID(machineId);

toZone = nextMachine.GetPropertyValue(PropertyDes.FromBuffers);

}

}

List<string> toBufferList = toZone.Split(',').Where(x => !string.IsNullOrEmpty(x)).ToList();

var isDry = currendMachine.GetPropertyValue(PropertyDes.IsDryProcess);

var isDryProcess = Convert.ToBoolean(string.IsNullOrEmpty(isDry) ? "false" : isDry);

//按定线寻找目标机台的就近库位

List<string> closeLocaitons = GetCloseMachinesCloseLocationList(currendMachine);

//按库区优先级推荐库位

DtoInventoryLocationResponse endpointLocation = \_inventoryManager.RecommendEmptyInventoryLocationsByPriority(currendMachine.MachineID, toBufferList, closeLocaitons, isDryProcess);

if (endpointLocation != null && endpointLocation.ID != 0)

{

\_inventoryManager.InventoryLocationOccupy(endpointLocation.ID, job.MissionID);

MCSHKPositionInfo endPoint = new MCSHKPositionInfo()

{

positionCode = endpointLocation.Coordinate,

type = "00",

inventoryShelveId = endpointLocation.ShelvesID,

inventoryLocationId = endpointLocation.ID,

positionType = HKPostionCodeType.InventoryLocation

};

paths.Add(endPoint);

string mcshilpath = JsonConvert.SerializeObject(paths);

return mcshilpath;

}

else

return "";

}

/// <summary>

/// 根据设备模式来获取开始库区

/// </summary>

/// <param name="unit"></param>

/// <param name="job"></param>

/// <returns></returns>

private DtoInventoryLocationResponse GetPutStartLocationWithMode(IAutoUnit unit, DtoJob job)

{

DtoInventoryLocationResponse start = null;

// 设备模式

var machineMode = unit.GetPropertyValue(PropertyDes.MachineMode);

var closeLocations = unit.GetPropertyValue(PropertyDes.CloseLocation);

string closeMachines = GetFromCloseMachines(unit.MachineID);

//备用设备

var spareMachines = unit.GetPropertyValue(PropertyDes.SpareMachines);

//分流设备

var shuntMachines = unit.GetPropertyValue(PropertyDes.ShuntMachines);

// 特殊模式

// 若库区有，则从库区取空花篮

// 若库区没有，从当前循环的出空花篮设备取空花篮货架

if (machineMode == MachineMode.SpecialMode)

{

// 源库区使用特殊模式专用源库区 只能从库区取货架

var startBufferZone = unit.GetPropertyValue(PropertyDes.FromBuffers\_clearLine);

var domains = \_dataAccess.GetLinesByStatus(LineDes.Active).Select(x => x.LineName).ToList();

var cycleID = unit.GetPropertyValue(PropertyDes.Scope);

var processName = \_machineManager.GetOutEmptyProcessByCycle(cycleID);

foreach (var domain in domains)

{

var fromMachines = $"{domain}\_{processName}\_{unit.MachineType}";// 特殊模式下,应该拿到的包号

start = RecommendPutJobStartLocation(job.Target, fromMachines, startBufferZone, machineMode, closeLocations, closeMachines, 0, spareMachines, shuntMachines);

if (start != null && start.ID != 0) break;

}

}

else // 正常生产模式

{

var startBufferZone = unit.GetPropertyValue(PropertyDes.FromBuffers);

// 定线时赋值的源设备组名称

var fromMachines = unit.GetPropertyValue(PropertyDes.FromMachines);

var coolingTime = unit.GetPropertyValue(PropertyDes.CoolingTime);

int timeCount = Convert.ToInt32(string.IsNullOrEmpty(coolingTime) ? "0" : coolingTime);

start = RecommendPutJobStartLocation(job.Target, fromMachines, startBufferZone, machineMode, closeLocations, closeMachines, timeCount, spareMachines, shuntMachines);

}

return start;

}

/// <summary>

/// 获取分流设备的配置分流阈值

/// </summary>

/// <returns></returns>

private Dictionary<string, int> GetShuntMachineConfigInfo(string shuntMachines) {

Dictionary<string, int> result = new Dictionary<string, int>();

try

{

List<string> shuntMachineList = shuntMachines.Split(',').Where(x => !string.IsNullOrEmpty(x)).ToList();

List<DtoProcess> allProcessConfig = \_dataAccess.GetAllProcesses();

List<DtoMachineGroup> allMachinesGroups = \_dataAccess.GetGroups();

foreach (var item in shuntMachineList)

{

//获取设备信息

IAutoUnit fromMachine = \_machineManager.GetUnitByMachineID(item);

//1设备的目标机台数量

string processName = fromMachine.GetPropertyValue(PropertyDes.ProcessSegment);

DtoProcess process = allProcessConfig.FirstOrDefault(x=>x.ProcessName==processName);

if (process == null)

continue;

//源机台所在的设备组

DtoMachineGroup fromMacinesGroup = allMachinesGroups.Where(x => x.Members.Contains(fromMachine.MachineID)).FirstOrDefault();

if (fromMacinesGroup == null)

continue;

//源机台的目标设备组

string toMachines = fromMachine.GetPropertyValue(PropertyDes.ToMachines);

DtoMachineGroup toMacinesGroup = allMachinesGroups.Where(x => x.GroupName== toMachines).FirstOrDefault();

if (toMacinesGroup == null)

continue;

//源机台设备组成员

List<string> machineGroupMemebers = fromMacinesGroup.Members.Split(',').Where(x => !string.IsNullOrEmpty(x)).ToList();

//源机台目标设备组成员

List<string> toMachineGroupMemebers = toMacinesGroup.Members.Split(',').Where(x => !string.IsNullOrEmpty(x)).ToList();

//配置的分流基数

double shuntLimit = process.ShuntLimit==0?2: process.ShuntLimit;

//2023 03 07客户要求改为目标机台数量\*基数

//int wipCount = (int)Math.Ceiling(toMachineGroupMemebers.Count()/ machineGroupMemebers.Count()\* shuntLimit);

int wipCount = (int)Math.Ceiling(toMachineGroupMemebers.Count()\* shuntLimit);

result.TryAdd(item,wipCount);

}

return result;

}

catch (Exception ex)

{

aLogger.Info($"获取分流配置发生异常:{ex}", "GetShuntMachineConfigInfo");

return result;

}

}

/// <summary>

/// RCS回调处理分发

/// </summary>

/// <param name="info"></param>

/// <param name="unit"></param>

public override void MissionCallBack(HKMissionCallback info, IAutoUnit unit)

{

var currentJob = \_dataAccess.GetMissionByID(info.taskCode);

//if (currentJob == null) // 查看是否有提前结束的ML任务

//{

// EndMLHandler(info);// 处理历史任务

// aLogger.Info($"{info.taskCode}发起回调但任务不存在", "RCSCallBackHandle");

// return;

//}

aLogger.Info($"{info.taskCode}RCS发起动作[{info.method}]回调-任务类型[{currentJob.MCSMissionType}],动作执行前任务状态[{currentJob.MissionStatus}],是否为回调[{currentJob.IsCallback}]", "RCSCallBackHandle");

aLogger.Info($"RCS入参:{JsonConvert.SerializeObject(info)}", "RCSMissionCallBackData");

if (info.method == "start") StartHandler(info, unit, currentJob);

if (info.method == "outbin") OutBinHandler(info, unit);

if (info.method == "end") EndHandler(info, unit, currentJob);

}

private void StartHandler(HKMissionCallback info, IAutoUnit unit, DtoJob currentJob)

{//todo 给接驳台发送任务开始信号

if (currentJob.MissionStatus != MissionStatus.SendSuccess && currentJob.MissionStatus != MissionStatus.Accepted)

{

aLogger.Info($"[{currentJob.MissionID}]重复start信号,忽略!", "JobInform");

return;

}

SendMissionStatus(unit.MachineID, info.robotCode);

if (unit.GetGoControlStatus() == "1")

{

//设置为不允许搬来

unit.SetGoControlStatus("0");

}

//if (currentJob.MCSMissionType != MCSMissionType.LToL)

//{

// if (currentJob.MCSMissionType == MCSMissionType.LToM || currentJob.MCSMissionType == MCSMissionType.MToM)

// {

// unit.BindShelfAndMachine(currentJob.PodCode);// 将此货架绑定到当前机台

// }

//}

//如果是回调任务开始启动

if (currentJob.IsCallback)

{

string PreSchedulingLimit = unit.GetPropertyValue(PropertyDes.PreScheduling);

if (!string.IsNullOrEmpty(PreSchedulingLimit) && PreSchedulingLimit == "true")

{

aLogger.Info($"{info.taskCode}start {currentJob.Target}预调度开关[PreScheduling]为[true] 恢复为[false]", "PreScheduling");

unit.AddOrUpdateProperty(PropertyDes.PreScheduling, "false");

}

}

//修改任务状态

\_dataAccess.UpdateMission(info.taskCode, info.robotCode, MissionStatus.Accepted, DateTime.Now);

}

public override void OutBinHandler(HKMissionCallback info, IAutoUnit unit)

{

try

{

\_dataAccess.UpdateMissionStatus(info.taskCode, MissionStatus.Outbin);

var currentJob = \_dataAccess.GetMissionByID(info.taskCode);

// 如果 MTO 从机台走出

if (currentJob.MCSMissionType == MCSMissionType.MToL || currentJob.MCSMissionType == MCSMissionType.MToM)

{

var flagAndVlaue\_CurrentJob = new Dictionary<string, object>();

var flagAndVlaue\_ParentJob = new Dictionary<string, object>();

//aLogger.Info($"[{currentJob.MCSMissionType}]任务:{info.taskCode} - 走出起点 删除前置任务", "JobInform");

//修改货架位置为空

//\_dataAccess.UpdatePodCoordinate(currentJob.PodCode, "");

\_inventoryManager.UpdateLocationOfPod(currentJob.PodCode, string.Empty);

string sourceMachine = unit.MachineName;

var machineInfo = \_dataAccess.GetMachineById(currentJob.Source);

//MTOM任务给任务的源设备添加记录

if (currentJob.MCSMissionType == MCSMissionType.MToM&& machineInfo!=null)

{

sourceMachine = machineInfo?.MachineName;

}

\_dataAccess.AddShelfTrace(new DtoShelfTrace()

{

ShelfName = currentJob.PodCode,

Description = "搬离接驳台",

Position = sourceMachine,

ProductionMode = unit.GetPropertyValue(PropertyDes.MachineMode),

});

if (currentJob.MCSMissionType == MCSMissionType.MToL)

{

unit.BindShelfAndMachine(string.Empty);

//设置为不允许搬走

unit.SetGoControlStatus("0");

//FinishJob(currentJob);//提前结束此任务

flagAndVlaue\_CurrentJob.Add("comeorgo", 2);

flagAndVlaue\_CurrentJob.Add("ShelvesCode", "0");

}

if (currentJob.MCSMissionType == MCSMissionType.MToM)

{

var paths = JsonConvert.DeserializeObject<List<MCSHKPositionMsg>>(currentJob.Paths);

var lastUnit = \_machineManager.GetUnitByPositionCode(paths.First().positionCode);

lastUnit.BindShelfAndMachine(string.Empty);// 解除货架与对接机构的绑定

//设置为不允许搬走

lastUnit.SetGoControlStatus("0");

// 给前置任务的设备发送搬离信号，清空货架号

flagAndVlaue\_ParentJob.Add("comeorgo", 2);

flagAndVlaue\_ParentJob.Add("ShelvesCode", "0");

WriteValueList(lastUnit.MachineID, flagAndVlaue\_ParentJob, "RCS回调处理", info.taskCode);

// 当前任务给搬来信号

//flagAndVlaue\_CurrentJob.Add("comeorgo", 1);

}

//将对应信号写给接驳台

WriteValueList(currentJob.Target, flagAndVlaue\_CurrentJob, "RCS回调处理", info.taskCode);

}

else// 如果是LTO从库位走出

{

aLogger.Info($"任务:{currentJob.MissionID}走出储位,入参：{JsonConvert.SerializeObject(info)},开始解绑库位、货架 并解除起点的任务占用", "Debug\_0624");

var mCSHKPositionInfos = JsonConvert.DeserializeObject<List<MCSHKPositionMsg>>(currentJob.Paths);

//库位、货架关系解绑

\_inventoryManager.UnboundLocationAndPod(mCSHKPositionInfos.First().inventoryLocationId);

// 解除任务占用库位

\_inventoryManager.CancelOccupy(currentJob.MissionID, mCSHKPositionInfos.First().inventoryLocationId);

}

}

catch (Exception ex)

{

aLogger.Info($"发生异常{ex}", "JobInform");

}

}

/// <summary>

/// 小车到达终点

/// </summary>

/// <param name="info"></param>

/// <param name="unit"></param>

/// <param name="currentJob"></param>

private void EndHandler(HKMissionCallback info, IAutoUnit unit, DtoJob currentJob)

{

if (currentJob.MCSMissionType == MCSMissionType.LToL)

{

LToLENDHandle(info, unit, currentJob);

return;

}

List<MCSHKPositionInfo> mCSHKPositionInfos = JsonConvert.DeserializeObject<List<MCSHKPositionInfo>>(currentJob.Paths);

MCSHKPositionInfo current = mCSHKPositionInfos.FirstOrDefault(x => x.positionCode == info.currentPositionCode);

//aLogger.Info($"[{unit.MachineName}],{info.taskCode},hik返回的位置:{info.currentPositionCode}", "JobInform");

//通知对接机构，小车或货架已到位，可以卸货或者装货

if (current.positionType == HKPostionCodeType.Machine)

{

//aLogger.Info($"{info.taskCode} - end - 到达最后一站[机台] 修改货架位置编码 修改任务状态", "JobInform");

var dtoShelvesResponseDTO = \_dataAccess.GetPodByName(info.podCode);

if (dtoShelvesResponseDTO != null)

{

//完成任务时，将货架绑定到设备上

unit.BindShelfAndMachine(currentJob.PodCode);// 将此货架绑定到当前机台

// 货架的坐标变更为接驳台的坐标，标识货架已经到了对接口

\_inventoryManager.UpdateLocationOfPod(dtoShelvesResponseDTO.ShelvesName, unit.WbCode);

\_dataAccess.AddShelfTrace(new DtoShelfTrace()

{

ShelfName = dtoShelvesResponseDTO.ShelvesName,

Description = "绑定到接驳台",

Position = unit.MachineName,

ProductionMode = unit.GetPropertyValue(PropertyDes.MachineMode)

});

}

var flagAndVlaue = new Dictionary<string, object>();

if (!string.IsNullOrEmpty(current.action1))

flagAndVlaue.Add("Ready1", current.action1 == "PUT" ? "1" : "2");

if (!string.IsNullOrEmpty(current.action2))

flagAndVlaue.Add("Ready2", current.action2 == "PUT" ? "1" : "2");

flagAndVlaue.Add("comeorgo", 1);

WriteValueList(currentJob.Target, flagAndVlaue, "RCS回调处理", currentJob.MissionID);

//更新货架所在的循环和工艺段

AddProcessAndScope(currentJob.PodCode, unit);

//清空货架的指定机台信息

\_inventoryManager.CancelPriorityMachineRelation(currentJob.PodCode);

}

//最后一站是库位

else if (current.positionType == HKPostionCodeType.InventoryLocation)

{

//绑定货架、库存 修改其状态

\_inventoryManager.BindLocationAndShelf(info.podCode, current.inventoryLocationId);

//到达库位同步货架的物料类型 0满货架满蓝 1空货架无花篮 2满货架空蓝 3未知

int materialType = !string.IsNullOrEmpty(currentJob.PodStatus) ? Convert.ToInt32(currentJob.PodStatus) :3;

\_dataAccess.UpdateShelvesMaterialType(info.podCode, materialType);

}

// 结束任务

FinishJob(currentJob);

//涉及库位、货架状态解除占用(通常end的库位一定是终点)

\_inventoryManager.CancelOccupy(currentJob.MissionID);

}

/// <summary>

/// 库区二次调度任务RCS回调处理

/// </summary>

/// <param name="info"></param>

/// <param name="unit"></param>

/// <param name="currentJob"></param>

private void LToLENDHandle(HKMissionCallback info, IAutoUnit unit, DtoJob currentJob)

{

aLogger.Info($"LTOL任务:{currentJob.MissionID}走到终点,入参：{JsonConvert.SerializeObject(info)},开始绑定库位、货架 并解除终点和货架的任务占用", "Debug\_0624");

//var pathList = JsonConvert.DeserializeObject<List<HKPositionInfo>>(currentJob.Paths);

List<MCSHKPositionInfo> mCSHKPositionInfos = JsonConvert.DeserializeObject<List<MCSHKPositionInfo>>(currentJob.Paths);

MCSHKPositionInfo current = mCSHKPositionInfos.FirstOrDefault(x => x.positionCode == info.currentPositionCode);

//到达最后一站

aLogger.Info($"{info.taskCode} - end - 到达最后一站 操作库存状态 修改任务状态", "RCSCallBackHandle");

string podcode = info.podCode;

if (string.IsNullOrEmpty(podcode))

{

var podInfo = \_dataAccess.GetShvlveByID(current.inventoryShelveId);

podcode = podInfo.ShelvesName;

}

//绑定货架、库存 修改其状态(继承货架的incomingTime)

\_dataAccess.InvertoryLocationBindingPod(podcode, current.inventoryLocationId, false);

//修改任务状态

currentJob.MissionStatus = MissionStatus.Finish;

//插入历史记录

\_dataAccess.InsertHistoryJob(currentJob);

//删除任务

\_dataAccess.DeleteMission(info.taskCode);

//涉及库位、货架状态解除占用(通常end的库位一定是终点)

\_dataAccess.InventoryLocationCancelOccupy(currentJob.MissionID);

}

/// <summary>

/// 处理等待来货架的任务 查看是否有库位满足条件

/// </summary>

/// <param name="unit"></param>

/// <param name="dtoJob"></param>

public override void WaitingforProcessHandle(IAutoUnit unit, DtoJob job)

{

try

{

//aLogger.Info($"设备{unit.MachineName}尝试处理waitingforProcess任务{job.MissionID} ", "WaitingforProcessHandle");

if (string.IsNullOrEmpty(job.Paths)) return;

var machineMode = unit.GetPropertyValue(PropertyDes.MachineMode);

var coolingTime = unit.GetPropertyValue(PropertyDes.CoolingTime);

int timeCount = Convert.ToInt32(string.IsNullOrEmpty(coolingTime) ? "0" : coolingTime);

var closeLocations = unit.GetPropertyValue(PropertyDes.CloseLocation);

//备用设备

var spareMachines = unit.GetPropertyValue(PropertyDes.SpareMachines);

//分流设备

var shuntMachines = unit.GetPropertyValue(PropertyDes.ShuntMachines);

string closeMachines = GetFromCloseMachines(unit.MachineID);

string fromBuffers = unit.GetPropertyValue(PropertyDes.FromBuffers);

string reason = "";

bool hasStart = false;

DtoInventoryLocationResponse start = new DtoInventoryLocationResponse();

MCSHKPositionInfo startpont = new MCSHKPositionInfo();

var fromMachines = unit.GetPropertyValue(PropertyDes.FromMachines);

//string members = \_dataAccess.GetGroupMembersByGroupName(fromMachines);

//List<string> machineList = members.Split(',').ToList();

//如果是PUT

if (job.Action1 == PropertyDes.PUT || job.Action2 == PropertyDes.PUT)

{

// 无源设备组名称(未定线?)

if (string.IsNullOrEmpty(fromMachines))

{

reason = $"库位推荐失败,该机台未定线(FromMachines属性值为空) 且无法直接从设备取料";

if (job.Reason != reason)

{

aLogger.Info($"任务{job.MissionID}-{reason}", "WaitingforProcessHandle");

}

}

else

{

//重新获取PUT任务库位

start = RecommendPutJobStartLocation(job.Target, fromMachines, fromBuffers, machineMode, closeLocations, closeMachines, timeCount, spareMachines, shuntMachines);

if (start != null && start.ID != 0)

{

job.MCSMissionType = MCSMissionType.LToM;

job.PodCode = start.ShelvesCode;

startpont = new MCSHKPositionInfo

{

positionType = HKPostionCodeType.InventoryLocation,

positionCode = start.Coordinate,

inventoryLocationId = start.ID,

inventoryShelveId = start.ShelvesInfo.ID,

type = "00"

};

hasStart = true;

}

}

}

//如果是GET

else

{

//查询关联设备

string assMachine = unit.GetPropertyValue(PropertyDes.AssociatedMachine);

var assUnit = \_machineManager.GetUnitByMachineID(assMachine);

//默认打开 false为关闭

bool keepWaiting = !string.IsNullOrEmpty(assUnit.GetPropertyValue(PropertyDes.KeepWaiting)) && Convert.ToBoolean(assUnit.GetPropertyValue(PropertyDes.KeepWaiting));

//如果配置了关联设备并且关联设备为死等模式

if (!string.IsNullOrEmpty(assMachine) && assUnit != null && keepWaiting)

{

//aLogger.Info($"[{unit.MachineName}],[{job.MissionID}] 设置为keepWaiting模式 等待对应的下料口下料完成", "WaitingforProcessHandle");

var podCode = assUnit.GetShelfOnMachine();

var moveoff = assUnit.GetGoControlStatus();

//有货架且允许搬走

if (!string.IsNullOrEmpty(podCode) && moveoff == "1")

{

aLogger.Info($"[{unit.MachineName}],[{job.MissionID}] 发现关联设备绑定的货架[{podCode}]", "WaitingforProcessHandle");

job.MCSMissionType = MCSMissionType.MToM;

job.PodCode = podCode;

job.Source = assUnit.MachineID;

startpont = new MCSHKPositionInfo

{

positionType = HKPostionCodeType.Machine,

positionCode = assUnit.WbCode,

type = "00"

};

hasStart = true;

//把对应设备的状态改为完成

//\_dataAccess.UpdateMission(dtoJob.MissionID, "", MissionStatus.Finish, DateTime.Now);

//aLogger.Info($"设备[{unit.MachineName}]尝试处理waitingforProcess任务{job.MissionID} 发现关联设备{assMachine}", "WaitingforProcessHandle\_KeepWaiting");

}

else

{

reason = $"关联设备[{assUnit.MachineName}]未绑定货架或者货架未到位!";

if (reason != job.Reason)

{

aLogger.Info($"[{unit.MachineName}],[{job.MissionID}]:{reason},关联设备[{assUnit.MachineName}]为keepwaiting", "WaitingforProcessHandle");

}

}

}

else

{

//aLogger.Info($"[{unit.MachineName}]未配置关联设备或配置错误 尝试从库位取货架", "WaitingforProcessHandle");

start = \_inventoryManager.RecommendGetStartInventoryLocation(job.Target, fromBuffers);

if (start != null && start.ID != 0)

{

job.MCSMissionType = MCSMissionType.LToM;

job.PodCode = start.ShelvesCode;

startpont = new MCSHKPositionInfo

{

positionType = HKPostionCodeType.InventoryLocation,

positionCode = start.Coordinate,

inventoryLocationId = start.ID,

inventoryShelveId = start.ShelvesInfo.ID,

type = "00"

};

hasStart = true;

}

}

}

if (hasStart)

{

List<MCSHKPositionInfo> oldpaths = JsonConvert.DeserializeObject<List<MCSHKPositionInfo>>(job.Paths);

MCSHKPositionInfo endpoint = oldpaths.Last();

List<MCSHKPositionInfo> newPath = new List<MCSHKPositionInfo>

{

startpont,

endpoint

};

//如果起点是库位

if (startpont.positionType == HKPostionCodeType.InventoryLocation)

{

//将库位占用 注: 任务完成后需要接触占用 否则无法继续使用

\_inventoryManager.InventoryLocationOccupy(start.ID, job.MissionID);

}

//记录任务处理时设备模式

if (string.IsNullOrEmpty(machineMode) || machineMode == MachineMode.Production)

job.MachineMode = MachineMode.Production;

else

job.MachineMode = MachineMode.SpecialMode;

job.Paths = JsonConvert.SerializeObject(newPath);

job.MissionStatus = MissionStatus.MESProcessed;

//添加包号

job = AddPayloads(job);

\_dataAccess.UpdatePayload(job);// 更新任务上做的更改

//aLogger.Info($"[{unit.MachineName}],[{job.MissionID}]结束", "WaitingforProcessHandle");

aLogger.Info($"{start.ShelvesCode}解除与机台的优先去向关系", "WaitingforProcessHandle");

\_inventoryManager.CancelPriorityMachineRelation(start.ShelvesID);

//清除库位的历史库区

\_inventoryManager.CancelHistoryAreaRelation(start.ID);

aLogger.Info($"[{unit.MachineName}],[{job.MissionID}]找到路径 {startpont.positionCode}-{endpoint.positionCode},运输[{job.PodCode}]", "WaitingforProcessHandle");

}

//无可用起点

else

{

//reason = $"无合适库位";

if (reason != job.Reason) // 发送无合适库位提醒到接驳台

{

SendMissionStatus(job.Target, "1");

aLogger.Info($"[{unit.MachineName}],[{job.MissionID}]仍未找到合适库位", "WaitingforProcessHandle");

}

//检索源工艺段的MTOL任务 并修改其终点

JobGraftingHandle(job, unit);

}

if (reason != job.Reason) // 原因变化需要更新

{

\_dataAccess.UpdateMissionReason(job.MissionID, reason);

}

}

catch (Exception ex)

{

aLogger.Info($"尝试处理任务发生异常{ex}", "WaitingforProcessHandle");

}

}

/// <summary>

/// 删除任务

/// 1.未分配路径的任务，未绑定货架，直接删除

/// 2.已分配路径和绑定货架，但是未发送给rcs,需要解除库位和货架的占用

/// 3.已发送给rcs的任务，需要解除库位和货架的占用，同时下线货架，解除设备与货架的绑定

/// </summary>

/// <param name="currentJob"></param>

public override ResponseResult DeleteJob(DtoJob currentJob)

{

try

{

var reason = string.Empty;

var result = false;

if (string.IsNullOrEmpty(currentJob.MissionStatus))

{

return new ResponseResult()

{

result = result,

message = reason

};

}

//var oldMissionStatus = currentJob.MissionStatus;

// 任务已发送给RCS,需要同步取消命令

if (currentJob.MissionStatus != MissionStatus.Create || currentJob.MissionStatus != MissionStatus.MESProcessed

|| currentJob.MissionStatus != MissionStatus.WaitingForProcess)

{

var reply = SendCancelToRcs(currentJob.MissionID);//通知rcs取消

if (reply != null)

{

if (reply.code == "0" || reply.code == "1")

{

if (currentJob.MCSMissionType != MCSMissionType.LToL) SendCancelToDC(currentJob.Target);

var unit = \_machineManager.GetUnitByMachineID(currentJob.Target);

//var pod = \_dataAccess.GetPodByName(currentJob.PodCode);

//var offResult = offLine(unit, pod);//删除任务同时将货架下线

string postionCode = unit.WbCode;

HKMissionReply rcsBindResult = new HKMissionReply();

//上料任务解绑库位

if (currentJob.MCSMissionType == MCSMissionType.LToM || currentJob.MCSMissionType == MCSMissionType.MToM)

{

if (!string.IsNullOrEmpty(currentJob.Paths))

{

List<MCSHKPositionInfo> mCSHKPositionInfos = JsonConvert.DeserializeObject<List<MCSHKPositionInfo>>(currentJob.Paths);

MCSHKPositionInfo start = mCSHKPositionInfos.First();

postionCode = start.positionCode;

rcsBindResult = BindOrUnbindShelves(postionCode, currentJob.PodCode, 0);

}

//通知接驳台

SendCancelToDC(currentJob.Target);

//未走出起点 通知起点机台取消

if (currentJob.MCSMissionType == MCSMissionType.MToM)

{

DtoMachine sourceMachine = \_dataAccess.GetMachineByWbCode(postionCode);

//未走出起点发送取消信号

if ((currentJob.MissionStatus == MissionStatus.Accepted || currentJob.MissionStatus == MissionStatus.MESProcessed || currentJob.MissionStatus == MissionStatus.SendSuccess) && sourceMachine!=null)

SendCancelToDC(sourceMachine.MachineID);

}

}

if (rcsBindResult.code == "1")

{

aLogger.Info($"[{unit.MachineName}]下线时向RCS解绑货架失败 下线失败", "SyncPodWithRCS");

result = false;

reason = GetEnumDescription(DockingStationResultType.RCSUnBindShelvesFail);

}

else

{

var dtoPod = \_inventoryManager.GetPodLocationByName(currentJob.PodCode);

\_inventoryManager.CancelOccupy(currentJob.MissionID);

\_machineManager.UnoundMachineAndShelf(currentJob.PodCode);//删除货架与设备的绑定

//设置为不允许搬走

unit.SetGoControlStatus("0");

// 若下线成功，解除当前货架与库位的关系

\_inventoryManager.OffLineShelf(dtoPod.ID, ShelvesStatus.OffLine);//下线货架

aLogger.Info($"下线[{dtoPod.ShelvesName}],货架坐标为[{dtoPod.Coordinate}]", "InventoryManager");

\_dataAccess.AddShelfTrace(new DtoShelfTrace()

{

ShelfName = dtoPod.ShelvesName,

OnlineStatus = ShelvesStatus.OffLine.ToString(),

Description = "下线货架,清空货架坐标",

Position = currentJob.MachineName,

ProductionMode = unit.GetPropertyValue(PropertyDes.MachineMode)

});

// 防止当货架上线成功之后，货架又变为可用

//if (dtoPod.InventoryLocation != null && !string.IsNullOrEmpty(dtoPod.InventoryLocation.LocationName))

//{

// \_inventoryManager.UnboundLocationAndPod(dtoPod.InventoryLocation?.LocationName);

//}

aLogger.Info($"手动删除任务:{currentJob.MissionID}", "DeleteJob");

aLogger.Info($"释放任务[{currentJob.MissionID}]占用的货架和库位!", "DeleteJob");

currentJob.MissionStatus = MissionStatus.ManualDelete;

\_dataAccess.InsertHistoryJob(currentJob);

\_dataAccess.DeleteMission(currentJob.MissionID);

result = true;

}

}

else

{

reason = reply.message;

}

}

else

{

reason = "与RCS通讯异常，无法删除任务!";

}

}

return new ResponseResult()

{

result = result,

message = reason

};

//// 若是ML或者LL任务可直接删除，或未发给rcs

//if (oldMissionStatus != MissionStatus.Create || oldMissionStatus != MissionStatus.WaitingForProcess)

//{

// // 只解除绑定，不清除库位信息

// // 清除任务对货架和库位的占用

//}

}

catch (Exception ex)

{

aLogger.Info($"删除任务发生异常{ex}", "DeleteJob");

aLogger.Info($"任务信息:{JsonConvert.SerializeObject(currentJob)}", "DeleteJob");

return new ResponseResult()

{

result = false,

message = "接口发生异常"

};

}

}

/// <summary>

/// 发送改变任务终点命令

/// </summary>

public override void ChangeDestination(DtoJob job)

{

try

{

string[] newPathArray;

List<MCSHKPositionMsg> newPaths;

if (job.MCSMissionType == MCSMissionType.LToM)

{

newPaths = GetFourthRoute(job);

newPathArray = new string[] { newPaths[0].positionCode, newPaths[1].positionCode };

}

else

{

return;

}

var request1 = new HKMissionChangeDestination()

{

reqCode = Guid.NewGuid().ToString("N"),

interfaceName = "changeTaskDest",

podCode = job.PodCode,

positionPath = newPathArray

};

var strReply = \_rcsClient.SendChangeDestination(request1);

aLogger.Info($"取消任务[{job.MissionID}]回复:{strReply.code},{strReply.message}", "CancelMission");

//改变成功

if (strReply.code == "0")

{

ConfirmCancelJob(job, newPaths);

}

else if (strReply.code == "1")

{

job.MissionStatus = MissionStatus.Cancel;

//在创建阶段，直接删除任务

\_dataAccess.InsertHistoryJob(job);

\_dataAccess.DeleteMission(job.MissionID);

\_inventoryManager.CancelOccupy(job.MissionID);

aLogger.Info($"RCS已取消任务,删除任务:{job.MissionID}!", "CancelMission");

}

else

{

\_dataAccess.UpdateMissionStatus(job.MissionID, MissionStatus.Cancel);

aLogger.Info($"取消任务[{job.MissionID}]失败,恢复任务状态为Cancel", "CancelMission");

}

}

catch (Exception ex)

{

aLogger.Info($"取消任务[{job.MissionID}]报错:{ex.Message},{ex.StackTrace}", "CancelMission");

}

}

public override void ShelvesOnlineOrOffline(IAutoUnit unit, int operationCode)

{

DockingStationResult result = new DockingStationResult();

try

{

aLogger.Info($"[{unit.MachineName}]设备上下线:{GetOnLineOperationMode(operationCode)}", "OnLineOrOffLine");

//开始上/下线

result = ShelvesOnLineOrOffLineByAutomationl(unit, (InventoryOperationType)operationCode);

aLogger.Info($"[{unit.MachineName}]设备上下线结果:{JsonConvert.SerializeObject(result)}", "OnLineOrOffLine");

}

catch (Exception ex)

{

aLogger.Info($"[{unit.MachineName}]设备上下线({operationCode})发生异常,{ex.StackTrace}", "OnLineOrOffLine");

result.Code = DockingStationResultType.MCSException;

result.Message = GetEnumDescription(DockingStationResultType.MCSException);

}

finally

{

//WriteResult(unit.MachineID, result);

//结果写给接驳台

Dictionary<string, object> flagAndValues = new Dictionary<string, object>();

flagAndValues.Add("OnlineOrOfflineResult", (int)result.Code);

WriteValueList(unit.MachineID, flagAndValues, "货架上下线");

}

}

public override DockingStationResult ShelvesOnlineOrOfflineManual(IAutoUnit unit, string shelvesName, int operationCode)

{

DockingStationResult result = new DockingStationResult();

try

{

aLogger.Info($"[{unit.MachineName}]页面上下线:{GetOnLineOperationMode(operationCode)}", "OnLineOrOffLine");

//开始上/下线

result = ShelvesOnLineOrOffLineByManual(unit, (InventoryOperationType)operationCode, shelvesName);

aLogger.Info($"[{unit.MachineName}]页面上下线:{JsonConvert.SerializeObject(result)}", "OnLineOrOffLine");

}

catch (Exception ex)

{

aLogger.Info($"[{unit.MachineName}]页面上下线({operationCode})发生异常,{ex.StackTrace}", "OnLineOrOffLine");

result.Code = DockingStationResultType.MCSException;

result.Message = GetEnumDescription(DockingStationResultType.MCSException);

}

return result;

}

/// <summary>

/// 货架自动上下线

/// </summary>

/// <param name="shelvesId"></param>

/// <param name="shelvesOperationType"></param>

/// <returns></returns>

public DockingStationResult ShelvesOnLineOrOffLineByAutomationl(IAutoUnit unit, InventoryOperationType shelvesOperationType)

{

DockingStationResult result = new DockingStationResult() { Result = false };

//未知的操作类型

if (shelvesOperationType != InventoryOperationType.ShelvesOffLineByAutomation && shelvesOperationType != InventoryOperationType.ShelvesOnLineByAutomationBeforeConvey && shelvesOperationType != InventoryOperationType.ShelvesOnLineByAutomationAfterConveyed)

{

result.Code = DockingStationResultType.UnknowActionCode;

result.Message = GetEnumDescription(DockingStationResultType.UnknowActionCode);

return result;

}

//读取该机台的货架号

string shelvesCode = ReadShevlesCode(unit.MachineID);

//货架读取失败

if (string.IsNullOrEmpty(shelvesCode) && shelvesCode != "0")

{

result.Code = DockingStationResultType.ShelvesReadFail;

result.Message = GetEnumDescription(DockingStationResultType.ShelvesReadFail);

return result;

}

DtoShelvesResponseDTO shelvesInfo = \_dataAccess.GetPodByName(shelvesCode);

if (shelvesInfo.Status == (int)ShelvesStatus.Banned)

{

result.Code = DockingStationResultType.ShelvesIsBanned;

result.Message = GetEnumDescription(DockingStationResultType.ShelvesIsBanned);

return result;

}

//货架未注册

if (shelvesInfo == null || shelvesInfo.ID == 0)

{

result.Code = DockingStationResultType.ShelvesUnregistered;

result.Message = GetEnumDescription(DockingStationResultType.ShelvesUnregistered);

return result;

}

//货架被占用

if (shelvesInfo.IsOccupy)

{

result.Code = DockingStationResultType.ShevlesIsOccupy;

result.Message = GetEnumDescription(DockingStationResultType.ShevlesIsOccupy);

return result;

}

//获取MCS原始屏蔽状态

string mcsControlStatus = unit.GetPropertyValue(PropertyDes.MCSControlStatus);

//屏蔽 避免产生任务

unit.AddOrUpdateProperty(PropertyDes.MCSControlStatus, "true");

//下线请求

if (shelvesOperationType == InventoryOperationType.ShelvesOffLineByAutomation)

{

unit.AddOrUpdateProperty(PropertyDes.MCSControlStatus, mcsControlStatus);

return offLine(unit, shelvesInfo);

}

if (shelvesOperationType == InventoryOperationType.ShelvesOnLineByAutomationAfterConveyed || shelvesOperationType == InventoryOperationType.ShelvesOnLineByAutomationBeforeConvey)

{

unit.AddOrUpdateProperty(PropertyDes.MCSControlStatus, mcsControlStatus);

return onLine(unit, shelvesInfo, shelvesOperationType);

}

//恢复原始状态

unit.AddOrUpdateProperty(PropertyDes.MCSControlStatus, mcsControlStatus);

return result;

}

/// <summary>

/// 货架下线

/// </summary>

/// <param name="unit"></param>

/// <param name="dtoShelvesResponseDTO"></param>

/// <returns></returns>

private DockingStationResult offLine(IAutoUnit unit, DtoShelvesResponseDTO dtoShelvesResponseDTO)

{

aLogger.Info($"[{unit.MachineName}]货架号:[{dtoShelvesResponseDTO.ShelvesName}]开始下线", "OnLineOrOffLine");

DockingStationResult result = new DockingStationResult();

if (unit.GetShelfOnMachine() != dtoShelvesResponseDTO.ShelvesName)

{

result.Result = false;

result.Code = DockingStationResultType.CoordinateIsWrong;

result.Message = GetEnumDescription(DockingStationResultType.CoordinateIsWrong);

aLogger.Info($"[{unit.MachineName}]货架号:[{dtoShelvesResponseDTO.ShelvesName}]货架号与机台不匹配!", "OnLineOrOffLine");

return result;

}

//重复下线

if (dtoShelvesResponseDTO.Status == (int)ShelvesStatus.OffLine)

{

result.Result = false;

result.Code = DockingStationResultType.ReOffLine;

result.Message = GetEnumDescription(DockingStationResultType.ReOffLine);

aLogger.Info($"[{unit.MachineName}]货架号:[{dtoShelvesResponseDTO.ShelvesName}]重复下线!", "OnLineOrOffLine");

return result;

}

var jobList = \_dataAccess.GetMissionByTarget(unit.MachineID).Where(x => !string.IsNullOrEmpty(x.MissionStatus)&&x.PodCode== dtoShelvesResponseDTO.ShelvesName);

// 当前机台存在发送给rcs的任务

if (jobList.Any(x => x.MissionStatus != MissionStatus.Create && x.MissionStatus != MissionStatus.WaitingForProcess))

{

result.Result = false;

result.Code = DockingStationResultType.OfflineOperationFailHasJob;

result.Message = GetEnumDescription(DockingStationResultType.OfflineOperationFailHasJob);

aLogger.Info($"[{unit.MachineName}]货架号:[{dtoShelvesResponseDTO.ShelvesName}]已经存在任务{JsonConvert.SerializeObject(jobList.Select(x => x.MissionID))}!", "OnLineOrOffLine");

return result;

}

var notSendJob = jobList.FirstOrDefault(x => x.MissionStatus == MissionStatus.Create || x.MissionStatus == MissionStatus.WaitingForProcess);

if (notSendJob != null)

{

aLogger.Info($"[{unit.MachineName}]与任务[{notSendJob.MissionID}]货架号:[{dtoShelvesResponseDTO.ShelvesName}]相同,删除任务,完成下线", "OnLineOrOffLine");

notSendJob.ErrorCode = "下线导致的直接完成并删除";

aLogger.Info($"解除任务[{notSendJob.MissionID}]占用的所有库位、货架资源", "OnLineOrOffLine");

//解除任务占用

\_dataAccess.InventoryLocationCancelOccupy(notSendJob.MissionID);

//插入历史 删除该任务

\_dataAccess.InsertHistoryJob(notSendJob);

aLogger.Info($"[{notSendJob.MachineName}]执行[{notSendJob.MissionID}],EMJobHistory表插入!", "OnLineOrOffLine");

\_dataAccess.DeleteMission(notSendJob.MissionID);

aLogger.Info($"[{notSendJob.MachineName}]执行[{notSendJob.MissionID}],EMJob表任务删除", "OnLineOrOffLine");

}

var job = \_dataAccess.GetMissionByPodcode(dtoShelvesResponseDTO.ShelvesName);//此时可以查到任务的货架，任务已经绑定货架，需要删除或者取消

if (job != null && job.MCSMissionType == MCSMissionType.MToM)

{

aLogger.Info($"[{unit.MachineName}]货架号:[{dtoShelvesResponseDTO.ShelvesName}]已经存在MM任务!", "OnLineOrOffLine");

result.Result = false;

result.Code = DockingStationResultType.OfflineOperationFailHasJob;

result.Message = GetEnumDescription(DockingStationResultType.OfflineOperationFailHasJob);

return result;

}

// 发送给RCS货架库位解绑消息

var rcsBindResult = BindOrUnbindShelves(unit.WbCode, dtoShelvesResponseDTO.ShelvesName, 0);

if (rcsBindResult.code == "1")

{

aLogger.Info($"[{unit.MachineName}]下线时向RCS解绑货架失败 下线失败", "SyncPodWithRCS");

result.Result = false;

result.Code = DockingStationResultType.RCSUnBindShelvesFail;

result.Message = GetEnumDescription(DockingStationResultType.RCSUnBindShelvesFail);

return result;

}

result.Result = true;

result.Code = DockingStationResultType.OfflineSuccess;

result.Message = GetEnumDescription(DockingStationResultType.OfflineSuccess);

// 解除接驳台和货架的绑定

//unit.BindShelfAndMachine(string.Empty);

\_machineManager.UnoundMachineAndShelf(dtoShelvesResponseDTO.ShelvesName);

//设置为不允许搬走

unit.SetGoControlStatus("0");

\_inventoryManager.OffLineShelf(dtoShelvesResponseDTO.ID, ShelvesStatus.OffLine);

\_dataAccess.AddShelfTrace(new DtoShelfTrace()

{

ShelfName = dtoShelvesResponseDTO.ShelvesName,

OnlineStatus = ShelvesStatus.OffLine.ToString(),

Description = "下线货架,清空货架位置",

Position = unit.MachineName,

ProductionMode = unit.GetPropertyValue(PropertyDes.MachineMode)

});

// 若下线成功，解除当前货架与库位的关系

// 防止当货架上线成功之后，货架又变为可用

if (dtoShelvesResponseDTO.InventoryLocation != null && !string.IsNullOrEmpty(dtoShelvesResponseDTO.InventoryLocation.LocationName))

{

\_inventoryManager.UnboundLocationAndPod(dtoShelvesResponseDTO.InventoryLocation?.LocationName);

}

\_dataAccess.AddDockingOperationLog(unit.MachineID, unit.MachineName, $"下线货架[{dtoShelvesResponseDTO.ShelvesName}]");

aLogger.Info($"[{unit.MachineName}]货架号:[{dtoShelvesResponseDTO.ShelvesName}]完成下线", "OnLineOrOffLine");

return result;

}

/// <summary>

/// 货架上线

/// </summary>

/// <param name="unit"></param>

/// <param name="shelvesInfo"></param>

/// <returns></returns>

private DockingStationResult onLine(IAutoUnit unit, DtoShelvesResponseDTO shelvesInfo, InventoryOperationType shelvesOperationType)

{

DockingStationResult result = new DockingStationResult();

aLogger.Info($"[{unit.MachineName}]货架号:[{shelvesInfo.ShelvesName}]开始上线", "OnLineOrOffLine");

string machineScope = unit.GetPropertyValue(PropertyDes.Scope);

if (!string.IsNullOrEmpty(shelvesInfo.SetScope)&& machineScope!= shelvesInfo.SetScope)

{

aLogger.Info($"[{unit.MachineName}]货架号:[{shelvesInfo.ShelvesName}({shelvesInfo.SetScope})]不属于设备所在循环{machineScope}", "OnLineOrOffLine");

DockingStationResultType resultType = DockingStationResultType.ShelvesNotInCurrentArea;

result.Result = false;

if (shelvesInfo.SetScope=="1")

resultType = DockingStationResultType.ShelvesNotInCurrentArea\_1;

else if (shelvesInfo.SetScope == "2")

resultType = DockingStationResultType.ShelvesNotInCurrentArea\_2;

else if (shelvesInfo.SetScope == "3")

resultType = DockingStationResultType.ShelvesNotInCurrentArea\_3;

else if (shelvesInfo.SetScope == "4")

resultType = DockingStationResultType.ShelvesNotInCurrentArea\_4;

result.Code = resultType;

result.Message = GetEnumDescription(resultType);

return result;

}

//重复上线

if (shelvesInfo.Status == (int)ShelvesStatus.Online)

{

aLogger.Info($"[{unit.MachineName}]货架号:[{shelvesInfo.ShelvesName}]重复上线", "OnLineOrOffLine");

result.Result = false;

result.Code = DockingStationResultType.ReOnLine;

result.Message = GetEnumDescription(DockingStationResultType.ReOnLine);

return result;

}

var podName = unit.GetShelfOnMachine();

if (!string.IsNullOrEmpty(podName))

{

aLogger.Info($"[{unit.MachineName}]存在货架号[{podName}],[{shelvesInfo.ShelvesName}]无法在此上线!", "OnLineOrOffLine");

result.Result = false;

result.Code = DockingStationResultType.HasOnlineShelves;

result.Message = GetEnumDescription(DockingStationResultType.HasOnlineShelves);

return result;

}

// 机台存在任务其它搬来货架任务

DtoJob currentToMJobJob = \_dataAccess.GetMissionByTarget(unit.MachineID)

.FirstOrDefault(x => x.MCSMissionType != MCSMissionType.MToL && x.MCSMissionType != MCSMissionType.LToL && !string.IsNullOrEmpty(x.MissionStatus));

if (currentToMJobJob != null)

{

//如果存在未分配路径的前置任务

if (currentToMJobJob.MissionStatus == MissionStatus.WaitingForProcess || currentToMJobJob.MissionStatus == MissionStatus.Create)

{

\_dataAccess.DeleteMission(currentToMJobJob.MissionID);

aLogger.Info($"删除未开始任务[{currentToMJobJob.MissionID}]", "OnLineOrOffLine");

}

//任务已经开始执行 无法上线

else

{

aLogger.Info($"[{unit.MachineName}]货架号:[{shelvesInfo.ShelvesName}]已存在任务[{currentToMJobJob.MissionID}]，无法下线!", "OnLineOrOffLine");

result.Result = false;

result.Code = DockingStationResultType.OnlineOperationFailHasJob;

result.Message = GetEnumDescription(DockingStationResultType.OnlineOperationFailHasJob);

return result;

}

}

// 向RCS绑定货架

var rcsBindResult = BindOrUnbindShelves(unit.WbCode, shelvesInfo.ShelvesName, 1);

if (rcsBindResult.code == "1")

{

aLogger.Info($"[{unit.MachineName}]上线时向RCS绑定货架失败 上线失败", "SyncPodWithRCS");

result.Result = false;

result.Code = DockingStationResultType.RCSBindShelvesFail;

result.Message = GetEnumDescription(DockingStationResultType.RCSBindShelvesFail);

return result;

}

#region 成功后的处理

//上线成功

result.Result = true;

result.Code = DockingStationResultType.OnlineSuccess;

result.Message = GetEnumDescription(DockingStationResultType.OnlineSuccess);

//修改货架状态

\_dataAccess.UpdateShvlvesStatus(shelvesInfo.ID, ShelvesStatus.Online);

//将货架号写给接驳台

var flagAndVlaue = new Dictionary<string, object>();

flagAndVlaue.Add("ShelvesCode", shelvesInfo.ShelvesName.Substring(3, 3));

//绑定新位置

unit.BindShelfAndMachine(shelvesInfo.ShelvesName);

\_inventoryManager.UpdateLocationOfPod(shelvesInfo.ShelvesName, unit.WbCode);

//插入货架操作历史

\_inventoryManager.InsertInventoryOperationHistory(shelvesInfo.ID, shelvesOperationType, InventoryObjectType.Shelves, GetEnumDescription(shelvesOperationType), "Robot");

string ShelfTraceMark = "(传篮后)";

//如果是传篮前上线需要发送轨道动作给接驳台

if (shelvesOperationType == InventoryOperationType.ShelvesOnLineByAutomationBeforeConvey || shelvesOperationType == InventoryOperationType.ShelvesOnLineByManualBeforeConvey)

{

ShelfTraceMark = "(传篮前)";

string action1 = unit.GetPropertyValue(PropertyDes.FirstLayer);

string action2 = unit.GetPropertyValue(PropertyDes.SecondLayer);

//向接驳台发送动作

//SendOrderToDcs(unit.MachineID, action1, action2);

if (!string.IsNullOrEmpty(action1))

flagAndVlaue.Add("Ready1", action1 == "PUT" ? "1" : "2");

if (!string.IsNullOrEmpty(action2))

flagAndVlaue.Add("Ready2", action2 == "PUT" ? "1" : "2");

}

//记录接驳台操作历史

\_dataAccess.AddDockingOperationLog(unit.MachineID, unit.MachineName, $"上线货架[{shelvesInfo.ShelvesName}]{ShelfTraceMark}");

\_dataAccess.AddShelfTrace(new DtoShelfTrace()// 记录货架流转历史

{

ShelfName = shelvesInfo.ShelvesName,

OnlineStatus = ShelvesStatus.Online.ToString(),

Description = $"上线货架,绑定货架到接驳台{ShelfTraceMark}",

Position = unit.MachineName,

ProductionMode = unit.GetPropertyValue(PropertyDes.MachineMode)

});

#endregion

WriteValueList(unit.MachineID, flagAndVlaue, "货架上下线");

aLogger.Info($"[{unit.MachineName}]货架号:[{shelvesInfo.ShelvesName}] 完成上线", "OnLineOrOffLine");

return result;

}

/// <summary>

/// 推荐带舟货架

/// </summary>

/// <param name="machineId"></param>

/// <param name="fromBuffersList"></param>

/// <param name="closeLocation"></param>

/// <returns></returns>

public DtoInventoryLocationResponse RecommendBoatShelvesInventoryLocation(string machineId,string fromBuffersList, string closeLocation) {

try

{

//指定设备的货架 > 切换过库区的货架 > 专属库区 > 超时库区 > 普通库区 > 全场库区

//获取全部可用库位

List<DtoInventoryLocationResponse> allLocation = \_dataAccess.GetAllInventoryFullShevlesLocation();

//验证货架的空满状态 获取带花篮的 并且货架状态在线的库位

allLocation = CheckFullShelvesStatus(allLocation);

////如果设置了冷却时间 进行过滤

//if (coolingTime > 0)

// allLocation = CheckCoolingTime(allLocation, coolingTime);

//承载过滤后的库位

List<DtoInventoryLocationResponse> areaInventoryLocationList = new List<DtoInventoryLocationResponse>();

//获取优先去向机台为当前设备的货架

areaInventoryLocationList = CheckPriorityMachine(allLocation, machineId);

//如果未获取到指定了该机台的货架 剔除存在优先去向的货架[以防指定了机台的货架去往其它设备]

if (areaInventoryLocationList.Count == 0)

allLocation = allLocation.Where(x => string.IsNullOrEmpty(x.ShelvesInfo.PriorityMachine)).ToList();

//获取专属库区物料

//库区组 包含所有可去库区 根据优先级排序 最高优先级为专属库区

List<string> areaNameList = fromBuffersList.Split(',').ToList();

var InventoryAreaList = \_dataAccess.GetInventoryAreaByNameList(areaNameList);

//设置了库区

if (areaInventoryLocationList.Count == 0 && InventoryAreaList.Count > 0)

{

List<int> areaIdList = InventoryAreaList.Select(x => x.ID).Distinct().ToList();

//（1.1）

//存在专属库区

//根据库位ID过滤 获取高优先级的库位

if (InventoryAreaList.Select(x => x.Priority).Distinct().Count() > 1)

{

//存在专属库区组 先从组内最高优先级库区取料

DtoInventoryAreaResponse highestArea = InventoryAreaList.OrderBy(x => x.Priority).FirstOrDefault();

//获取最高优先级的库位

var highestAreaInfo = \_dataAccess.GetInventoryAreaByName(highestArea.InventoryAreaName);

//次级优先级库位

var lowerAreaList = InventoryAreaList.Where(x => x.ID != highestAreaInfo.ID).ToList();

areaInventoryLocationList = allLocation.Where(x => x.AreaId == highestAreaInfo.ID && !string.IsNullOrEmpty(x.ShelvesInfo.PayLoad)).ToList();

//专属库区没有可用库位

if (areaInventoryLocationList.Count == 0)

{

//仍然无料 从其它优先级库区再次检索库位

if (areaInventoryLocationList.Count == 0)

areaInventoryLocationList = allLocation.Where(x => lowerAreaList.Select(x => x.ID).Contains(x.AreaId) && !string.IsNullOrEmpty(x.ShelvesInfo.PayLoad)).ToList();

}

}

//未设置专属库区

else

{

//（1.2）

//获取设置的全部库区库位

areaInventoryLocationList = allLocation.Where(x => areaIdList.Contains(x.AreaId) && !string.IsNullOrEmpty(x.ShelvesInfo.PayLoad)).ToList();

}

}

List<string> closeLocations = closeLocation.Split(',').Where(x => !string.IsNullOrEmpty(x)).ToList();

DtoInventoryLocationResponse targetLocation = CheckCloseLocationsAndFIFO(areaInventoryLocationList, closeLocations, 0);

if (targetLocation != null && targetLocation.ID != 0)

return targetLocation;

aLogger.Info($"[{machineId}],取料库区集合[{fromBuffersList}]未找到合适库位或合适货架", "JobProcessed");

return null;

}

catch (Exception ex)

{

aLogger.Info($"推荐PUT任务起点库位发生异常,{ex.Message} 入参:{machineId}", "JobProcessed");

return null;

}

}

/// <summary>

/// 二次调度（库区内发生）

/// </summary>

private void InventoryScheduleInArea()

{

try

{

//注释: 系统默认权重为999的库位为最高权重库位 当最高权限库位存在空的库位时,并且其它库位有料 其余库位应向该库位调度

var allInventoryArea = \_dataAccess.GetAllInventoryArea().ToList();

foreach (var item in allInventoryArea)

{

//获取当前库区的所有库位

var inventoryLocationsOfArea = \_dataAccess.GetInventoryLocationByAreaId(item.ID);

//查询权重为999的空库位

var highestEmptyLocations = inventoryLocationsOfArea.Where(x => x.Priority == 999 && x.InventoryStatus\_Enum == (int)InventoryLocationStatus.Empty && x.IsOccupy == 0 && x.IsEable != "Disable").ToList(); //查询不为空的其它权重库位

//0表示关闭 1表示不分空满全搬 2表示只搬运空货架 3表示只搬运满货架（其它无效配置默认全搬）

if (\_mcsVariable.InventorySchedule == 2)

{

inventoryLocationsOfArea = inventoryLocationsOfArea.Where(x => x.ShelvesInfo.InventoryStatus == (int)ShelvesInventoryStatus.Empty).ToList();

}

else if (\_mcsVariable.InventorySchedule == 3)

{

inventoryLocationsOfArea = inventoryLocationsOfArea.Where(x => x.ShelvesInfo.InventoryStatus == (int)ShelvesInventoryStatus.Full).ToList();

}

var otherFullLocations = inventoryLocationsOfArea.Where(x => x.Priority != 999 && x.InventoryStatus\_Enum == (int)InventoryLocationStatus.Full && x.IsOccupy == 0 && x.IsEable != "Disable").ToList();

//查询当前库区的最高权限库位

if (highestEmptyLocations.Count > 0 && otherFullLocations.Count > 0)

{

aLogger.Info($"库区{item.InventoryAreaName}发起二次调度 \r\n 999权重空库位信息：{JsonConvert.SerializeObject(highestEmptyLocations)},\r\n 其它权重库位{JsonConvert.SerializeObject(otherFullLocations)} \r\n", "InventorySchedule");

//生成库位调用任务 每次处理一个

//随机处理

var tolocation = highestEmptyLocations.OrderBy(x=>x.Priority).FirstOrDefault();

var location\_to = \_dataAccess.GetInventoryLocationByID(tolocation.ID);

//先进先出 获取最早来料的库位

var fromlocatopn = otherFullLocations.OrderBy(x => x.IncomingTime).FirstOrDefault();

//最后验证当前需要调度的库位是否以及被处理/占用

var location\_from = \_dataAccess.GetInventoryLocationByID(fromlocatopn.ID);

if (location\_from.IsOccupy == 1 || location\_to.IsOccupy == 1)

{

continue;

}

CreateInventoryScheduleJob(fromlocatopn, tolocation, item.RCSMissionType);

}

}

}

catch (Exception ex)

{

aLogger.Info($"创建库区二次调度任务发生异常{ex}", "InventorySchedule");

}

}

#region 环境监测小车定时任务

/// <summary>

/// 删除定时任务

/// </summary>

/// <param name="key"></param>

public async void DeleteSchedulerJob(string key)

{

try

{

\_scheduler = await \_schedulerFactory.GetScheduler();

//删除任务

var jobKey = new JobKey(key);

if (await \_scheduler.CheckExists(jobKey))

{

await \_scheduler.DeleteJob(jobKey);

}

}

catch (Exception ex)

{

aLogger.Info($"删除环境监测小车任务发生异常{ex}", "SendEnvironmentalMission");

}

}

public void InitAgvJob() {

try

{

List<DtoEnvironmentalMission> jobs = \_dataAccess.GetAllEnvironmentalMissions();

foreach (var item in jobs)

{

if (item.Status == 0)

continue;

StartJob(item);

}

}

catch (Exception ex)

{

aLogger.Info($"初始化环境监测小车任务发生异常{ex}", "SendEnvironmentalMission");

}

}

/// <summary>

/// 创建普通定时任务

/// </summary>

/// <param name="agvCode"></param>

/// <param name="missionType"></param>

/// <param name="startTime"></param>

/// <param name="intervalTime"></param>

public async void RunAgvJob(string agvCode, string missionType, string positionCode, DateTime startTime, int intervalTime)

{

try

{

\_scheduler = await \_schedulerFactory.GetScheduler();

await \_scheduler.Start();

var jobKey = new JobKey($"{agvCode}-{missionType}");

if (await \_scheduler.CheckExists(jobKey))

{

await \_scheduler.DeleteJob(jobKey);

}

//获取本地时区偏移量

var offset = TimeZoneInfo.Local.BaseUtcOffset;

//转成世界时(去除偏移量)

var jobStartTime = DateTimeOffset.MinValue;

//如果设置的开始时间小于当前时间 设为当前时间

if (startTime < DateTime.Now)

jobStartTime = DateTime.UtcNow;

else

jobStartTime = new DateTimeOffset(startTime);

aLogger.Info($"jobStartTime{jobStartTime}", "SendEnvironmentalMission");

var trigger = TriggerBuilder.Create()

.WithSimpleSchedule(x => x.WithIntervalInMinutes(intervalTime).RepeatForever())

.StartAt(jobStartTime)

.Build();

//创建作业实例

SendEnvironmentalMission.\_rcsClient = \_rcsClient;

var job = JobBuilder.Create<SendEnvironmentalMission>()

.WithIdentity($"{agvCode}-{missionType}")

.UsingJobData("agvCode", agvCode)

.UsingJobData("missionType", missionType)

.UsingJobData("positionCode", positionCode)

.Build();

await \_scheduler.ScheduleJob(job, trigger);

}

catch (Exception ex)

{

aLogger.Info($"启动任务发生异常{ex}", "SendEnvironmentalMission");

}

}

/// <summary>

/// 创建CORN表达式任务

/// </summary>

/// <param name="agvCode"></param>

/// <param name="missionType"></param>

/// <param name="cornExpression"></param>

public async void RunAgvJobWithCorn(string agvCode, string missionType, string positionCode, string cornExpression)

{

try

{

\_scheduler = await \_schedulerFactory.GetScheduler();

await \_scheduler.Start();

var trigger = TriggerBuilder.Create()

.WithCronSchedule(cornExpression)

.Build();

//创建作业实例

DeleteSchedulerJob($"{agvCode}-{missionType}");

SendEnvironmentalMission.\_rcsClient = \_rcsClient;

var job = JobBuilder.Create<SendEnvironmentalMission>()

.WithIdentity($"{agvCode}-{missionType}")

.UsingJobData("agvCode", agvCode)

.UsingJobData("missionType", missionType)

.UsingJobData("positionCode", positionCode)

.Build();

await \_scheduler.ScheduleJob(job, trigger);

}

catch (Exception ex)

{

aLogger.Info($"启动任务发生异常{ex}", "SendEnvironmentalMission");

}

}

public ResponseResult AddEnvironmentalMission(DtoEnvironmentalMission request) {

ResponseResult result = new ResponseResult();

try

{

result.result = true;

result.message = "success";

request.Status = 0;

DtoEnvironmentalMission dtoEnvironmentalMission = \_dataAccess.GetEnvironmentalMissionByCodeAndMissionType(request.AgvCode,request.MissionType);

if (dtoEnvironmentalMission == null) {

\_dataAccess.AddEnvironmentalMission(request);

StartJob(request);

}

else

{

result.result = false;

result.message = "AGV编码和任务编码不可组合重复";

}

}

catch (Exception ex)

{

result.result = false;

result.message = "新增失败 发生异常";

aLogger.Info($"新增环境监测小车任务发生异常{ex}", "SendEnvironmentalMission");

}

return result;

}

public ResponseResult UpdateEnvironmentalMission(DtoEnvironmentalMission request) {

ResponseResult result = new ResponseResult();

try

{

result.result = true;

result.message = "success";

DtoEnvironmentalMission old = \_dataAccess.GetEnvironmentalMissionByid(request.id);

//如果是启动状态 无法修改

if (old.Status == 1)

{

result.result = false;

result.message = "定时任务(注:任务指后台任务非RCS任务)执行中无法修改,请关闭后重试";

return result;

}

if (old.AgvCode!=request.AgvCode|| old.MissionType!=request.MissionType)

{

DtoEnvironmentalMission other = \_dataAccess.GetEnvironmentalMissionByCodeAndMissionType(request.AgvCode,request.MissionType);

if (other!=null)

{

result.result = false;

result.message = "AGV编码和任务编码不可组合重复";

return result;

}

}

\_dataAccess.UpdateEnvironmentalMission(request);

}

catch (Exception ex)

{

result.result = false;

result.message = "新增失败 发生异常";

aLogger.Info($"修改环境监测小车任务发生异常{ex}", "SendEnvironmentalMission");

}

return result;

}

public List<DtoEnvironmentalMission> GetEnvironmentalMissions() {

List<DtoEnvironmentalMission> res = new List<DtoEnvironmentalMission>();

try

{

res = \_dataAccess.GetAllEnvironmentalMissions();

return res;

}

catch (Exception ex)

{

aLogger.Info($"查询环境监测小车任务发生异常{ex}", "SendEnvironmentalMission");

return res;

}

}

public ResponseResult DeleteEnvironmentalMission(int id) {

ResponseResult result = new ResponseResult();

try

{

result.result = true;

result.message = "success";

DtoEnvironmentalMission environmentalMission = \_dataAccess.GetEnvironmentalMissionByid(id);

if (environmentalMission!=null)

{

DeleteSchedulerJob($"{environmentalMission.AgvCode}-{environmentalMission.MissionType}");

\_dataAccess.DeleteEnvironmentalMission(id);

}

}

catch (Exception ex)

{

result.result = false;

aLogger.Info($"删除环境监测小车任务发生异常{ex}", "SendEnvironmentalMission");

}

return result;

}

public ResponseResult StartEnvironmentalMission(int id) {

ResponseResult result = new ResponseResult();

try

{

result.result = true;

result.message = "success";

//修改状态

DtoEnvironmentalMission environmentalMission = \_dataAccess.GetEnvironmentalMissionByid(id);

environmentalMission.Status = 1;

\_dataAccess.UpdateEnvironmentalMissionStatus(environmentalMission);

//创建后台任务

StartJob(environmentalMission);

}

catch (Exception ex)

{

result.result = false;

result.message = "发生异常";

aLogger.Info($"启动环境监测小车任务发生异常{ex}", "SendEnvironmentalMission");

}

return result;

}

public ResponseResult StopEnvironmentalMission(int id)

{

ResponseResult result = new ResponseResult();

try

{

result.result = true;

result.message = "success";

//修改状态

DtoEnvironmentalMission environmentalMission = \_dataAccess.GetEnvironmentalMissionByid(id);

environmentalMission.Status = 0;

\_dataAccess.UpdateEnvironmentalMissionStatus(environmentalMission);

//删除后台任务

DeleteSchedulerJob($"{environmentalMission.AgvCode}-{environmentalMission.MissionType}");

}

catch (Exception ex)

{

result.result = false;

result.message = "发生异常";

aLogger.Info($"启动环境监测小车任务发生异常{ex}", "SendEnvironmentalMission");

}

return result;

}

private void StartJob(DtoEnvironmentalMission request) {

if (request.IsCorn)

{

//启动CORN表达式定时器

RunAgvJobWithCorn(request.AgvCode, request.MissionType, request.PositionCode, request.Expression);

}

else

{

//启动常规定时任务

RunAgvJob(request.AgvCode, request.MissionType, request.PositionCode, request.JobStartTime, request.IntervalTime);

}

}

public string GetSchduleJobList()

{

return JsonConvert.SerializeObject(\_scheduler.GetCurrentlyExecutingJobs());

}

#endregion

private void CreateInventoryScheduleJob(DtoInventoryLocationResponse fromlocation, DtoInventoryLocationResponse tolocation, string RcsMissionType = "")

{

try

{

if (fromlocation == null) return;

if (tolocation == null) return;

aLogger.Info($"开始创建任务 \r\n from：{JsonConvert.SerializeObject(fromlocation)},\r\n to{JsonConvert.SerializeObject(tolocation)} \r\n", "InventorySchedule");

var paths = new List<MCSHKPositionMsg>(){

//起点暂不确定等待GET条件触发后替换坐标点位

new MCSHKPositionMsg()

{

positionCode =fromlocation.Coordinate,

type="00",

positionType=HKPostionCodeType.InventoryLocation,

inventoryLocationId=fromlocation.ID,

inventoryShelveId=fromlocation.ShelvesInfo.ID,

locationName = fromlocation.LocationName,

podName = fromlocation.ShelvesInfo.ShelvesName

},

new MCSHKPositionMsg()

{

positionCode = tolocation.Coordinate,

type="00",

positionType=HKPostionCodeType.InventoryLocation,

inventoryLocationId=tolocation.ID,

locationName = tolocation.LocationName,

inventoryShelveId=fromlocation.ShelvesInfo.ID,

podName = fromlocation.ShelvesInfo.ShelvesName

}

};

string mcshilpath = JsonConvert.SerializeObject(paths);

aLogger.Info($"任务路径{mcshilpath}", "InventorySchedule");

var newJob = new DtoJob

{

MissionID = Guid.NewGuid().ToString(),

Target = string.IsNullOrEmpty(fromlocation.ShelvesInfo.Source)? \_mcsVariable.DefaultMachineId : fromlocation.ShelvesInfo.Source,

CreateTime = DateTime.Now,

MissionStatus = MissionStatus.MESProcessed,

Priority = "5",

MachineName = fromlocation.LocationName,

WbCode = fromlocation.Coordinate,

PodCode = fromlocation.ShelvesCode,

Paths = mcshilpath,

MCSMissionType = MCSMissionType.LToL,

MissionType = string.IsNullOrEmpty(RcsMissionType) ? \_mcsVariable.InventoryScheduleMissionType : RcsMissionType

};

\_dataAccess.AddMission(newJob);

//占用库位

\_dataAccess.InventoryLocationOccupy(fromlocation.ID, newJob.MissionID);

\_dataAccess.InventoryLocationOccupy(tolocation.ID, newJob.MissionID);

aLogger.Info($"任务创建完成后库位占用", "InventorySchedule");

}

catch (Exception ex)

{

aLogger.Info($"创建库区二次调度任务发生异常{ex}", "InventorySchedule");

}

}

/// <summary>

/// 获取所有库位物料信息

/// </summary>

/// <returns></returns>

public List<DtoStoreInfo> GetAllStoreInfos()

{

var bufferInfoList = new List<DtoStoreInfo>();

try

{

var allAreas = \_dataAccess.GetAllInventoryArea();

var allPoints = \_dataAccess.GetAllInventoryLocation();

var machines = \_dataAccess.GetMachines();

//var allPods = \_dataAccess.GetAllShvlves();

Dictionary<string, int> wipDic = GetMachineWIPTime();

foreach (var area in allAreas)

{

var allPointsInArea = allPoints.Where(x => x.InventoryArea.ID == area.ID).ToList();

var pointInfoList = new List<PointInfo>();

int fullCount = allPointsInArea.Where(x => x.InventoryStatus\_Enum == (int)InventoryLocationStatus.Full).Count();

foreach (var point in allPointsInArea)

{

//获取该库位货架对应的超时时间

var pointInfo = new PointInfo()

{

ShelvesName = point.ShelvesInfo?.ShelvesName,

Source = string.IsNullOrEmpty(point.ShelvesInfo?.Source) ? "" : machines.Where(x => x.MachineID == point.ShelvesInfo?.Source).First()?.MachineName,

IncomingTime = point.IncomingTime,

LocationName = point.LocationName,

Payload = point.ShelvesInfo?.PayLoad,

MaterialType= (point.ShelvesInfo == null|| point.ShelvesInfo.ID==0) ? null:point.ShelvesInfo?.MaterialType,

IsTimeout = false,

isOccupy = point.IsOccupy,

Coordinate = point.Coordinate

};

if (point.ShelvesInfo != null && !string.IsNullOrEmpty(point.ShelvesInfo.Source))

{

int wipTime = wipDic[point.ShelvesInfo.Source];

if (wipTime != 0)

{ //计算超时

pointInfo.IsTimeout = point.IncomingTime.Year > 1 && wipTime > 0 ? (point.IncomingTime.AddMinutes(wipTime) < DateTime.Now) : false;

}

}

pointInfoList.Add(pointInfo);

}

bufferInfoList.Add(new DtoStoreInfo()

{

InventoryAreaName = $"{area.InventoryAreaName}({allPointsInArea.Count}/{fullCount})",

CreateTime = area.CreateTime,

ID = area.ID,

PointInfos = pointInfoList.OrderBy(x=>x.LocationName).ToList()

});

}

}

catch (Exception ex)

{

aLogger.Info($"{ex.StackTrace},{ex.Message}", "InventoryManager");

}

return bufferInfoList.OrderBy(x => x.CreateTime).ToList();

}

/// </summary>

/// <param name="machineId">设备id</param>

/// <param name="fromBuffersList">源库区集合</param>

/// <param name="fromMachines">源设备组</param>

/// <param name="timeOut">上工艺段超时时间</param>

/// <param name="coolingTime">冷却时间</param>

/// <param name="closeLocation">临近库位</param>

/// <param name="closeMachies">临近设备</param>

/// <param name="basketType">花篮类型</param>

/// <param name="spareMachines">备用设备</param>

/// <param name="shuntMachines">分流设备</param>

/// <param name="machineMode">设备模式</param>

/// <param name="timeOutControl">超时控制上料</param>

/// <returns></returns>

public DtoInventoryLocationResponse RecommendFullShelvesInventoryLocation(string machineId, string fromBuffersList, string fromMachines, int timeOut, int coolingTime, string closeLocation,

string closeMachies,string basketType,string spareMachines,Dictionary<string,int> shuntMachines, string machineMode = "",int timeOutControl=0)

{

try

{

//指定设备的货架 > 切换过库区的货架 >临近超时物料> 专属库区>临近设备的物料 > 超时库区 > 普通库区 > 全场库区

//获取全部可用库位

List<DtoInventoryLocationResponse> allLocation = \_dataAccess.GetAllInventoryFullShevlesLocation();

//从全部库位获取 产出充足的分流机台的货架

List<DtoInventoryLocationResponse> shuntLocations = new List<DtoInventoryLocationResponse>();

if (machineMode!=MachineMode.SpecialMode)

shuntLocations= GetShuntMachineMaterial(allLocation, shuntMachines);

//验证货架的空满状态 获取带花篮的 并且货架状态在线的库位

allLocation = CheckFullShelvesStatus(allLocation);

if (basketType == "Full")

//验证包号

allLocation = CheckSourceMachine(allLocation, fromMachines);

else

//验证包号(剔除Line)

allLocation = CheckSourceMachineWithoutLine(allLocation, fromMachines);

//将分流机台的物料添加到库位集合中

allLocation.AddRange(shuntLocations);

//工艺段设置超时不可取料 剔除超时物料

if (timeOutControl>0)

allLocation = allLocation.Where(x => DateTime.Now.Subtract(x.IncomingTime).TotalSeconds < timeOutControl).ToList();

//如果设置了冷却时间 进行过滤

if (coolingTime > 0)

allLocation = CheckCoolingTime(allLocation, coolingTime);

//承载过滤后的库位

List<DtoInventoryLocationResponse> areaInventoryLocationList = new List<DtoInventoryLocationResponse>();

//获取优先去向机台为当前设备的货架

areaInventoryLocationList = CheckPriorityMachine(allLocation,machineId);

//如果未获取到指定了该机台的货架 剔除存在优先去向的货架[以防指定了机台的货架去往其它设备]

if (areaInventoryLocationList.Count == 0)

allLocation = allLocation.Where(x => string.IsNullOrEmpty(x.ShelvesInfo.PriorityMachine)).ToList();

List<string> closeMachineList = closeMachies.Split(',').Where(x => !string.IsNullOrEmpty(x)).ToList();

//获取专属库区物料

//库区组 包含所有可去库区 根据优先级排序 最高优先级为专属库区

List<string> areaNameList = fromBuffersList.Split(',').ToList();

var InventoryAreaList = \_dataAccess.GetInventoryAreaByNameList(areaNameList);

//未获取到绑定设备的货架 尝试获取切换了库区的库位(切换过库区的带货架库位 货架优先消耗)

if (areaInventoryLocationList.Count == 0)

areaInventoryLocationList = CheckChangeInventoryAreaLocations(allLocation, areaInventoryLocationList, InventoryAreaList, fromMachines);

//设置了库区

if (areaInventoryLocationList.Count==0 && InventoryAreaList.Count > 0)

{

List<int> areaIdList = InventoryAreaList.Select(x => x.ID).Distinct().ToList();

//如果是满花篮 需要验证超时时间

if (basketType == "Full")

{

//0 先去检索临近超时的物料 无视专属或者普通库区

//接近超时的物料

List<DtoInventoryLocationResponse> willTimeOutShelves = allLocation.Where(x => areaIdList.Contains(x.AreaId) && DateTime.Now.Subtract(x.IncomingTime).TotalSeconds > (timeOut \* 0.8)).ToList();

if (willTimeOutShelves.Count > 0)

areaInventoryLocationList = willTimeOutShelves;

}

//物料均未接近超时 优先获取专属库区的物料

if (areaInventoryLocationList.Count == 0)

{

//（1.1）

//存在专属库区

//根据库位ID过滤 获取高优先级的库位

if (InventoryAreaList.Select(x => x.Priority).Distinct().Count() > 1)

{

//存在专属库区组 先从组内最高优先级库区取料

DtoInventoryAreaResponse highestArea = InventoryAreaList.OrderBy(x => x.Priority).FirstOrDefault();

//获取最高优先级的库位

var highestAreaInfo = \_dataAccess.GetInventoryAreaByName(highestArea.InventoryAreaName);

//次级优先级库位

var lowerAreaList = InventoryAreaList.Where(x => x.ID != highestAreaInfo.ID).ToList();

//专属库区内临近设备的库位

//(2.1) 检索临近设备的物料(定线优先级)

areaInventoryLocationList = allLocation.Where(x => x.AreaId == highestAreaInfo.ID && !string.IsNullOrEmpty(x.ShelvesInfo.PayLoad)&& closeMachineList.Contains(x.ShelvesInfo.Source)).ToList();

//未获取到临近设备的物料 尝试获取其它设备的多余物料

//获取其它设备的物料时保证其对应设备正在对接货架或者在制数量大于1

if (areaInventoryLocationList.Count==0)

areaInventoryLocationList = CheckSameProcessMachineMaterial(allLocation,new List<int> { highestAreaInfo.ID },machineId);

////(2.2)专属库区全部库位(0927注释 未能获取到其它机台的)

//if (areaInventoryLocationList.Count == 0)

// areaInventoryLocationList = allLocation.Where(x => x.AreaId == highestAreaInfo.ID && !string.IsNullOrEmpty(x.ShelvesInfo.PayLoad)).ToList();

//专属库区没有可用库位

if (areaInventoryLocationList.Count == 0)

{

////（3）检索超时料

//if (timeOut > 0)

// areaInventoryLocationList = allLocation.Where(x => DateTime.Now.Subtract(x.IncomingTime).TotalSeconds > timeOut).ToList();

////(4)仍然无料 从其它优先级库区再次检索库位

//if (areaInventoryLocationList.Count == 0)

// areaInventoryLocationList = allLocation.Where(x => lowerAreaList.Select(x => x.ID).Contains(x.AreaId) && !string.IsNullOrEmpty(x.ShelvesInfo.PayLoad)).ToList();

//检索定线优先级机台物料

areaInventoryLocationList = allLocation.Where(x => lowerAreaList.Select(x => x.ID).Contains(x.AreaId) && !string.IsNullOrEmpty(x.ShelvesInfo.PayLoad) && closeMachineList.Contains(x.ShelvesInfo.Source)).ToList();

//检索其它定线优先级的冗余物料

if (areaInventoryLocationList.Count == 0)

areaInventoryLocationList = CheckSameProcessMachineMaterial(allLocation, lowerAreaList.Select(x => x.ID).ToList(), machineId);

}

}

//未设置专属库区

else

{

//1 获取临近设备的物料

areaInventoryLocationList = allLocation.Where(x => areaIdList.Contains(x.AreaId) && !string.IsNullOrEmpty(x.ShelvesInfo.PayLoad) && closeMachineList.Contains(x.ShelvesInfo.Source)).ToList();

//获取其它机台的冗余物料

if (areaInventoryLocationList.Count == 0)

areaInventoryLocationList = CheckSameProcessMachineMaterial(allLocation, areaIdList, machineId);

////获取全部可用物料

//if (areaInventoryLocationList.Count == 0)

// areaInventoryLocationList = allLocation.Where(x => areaIdList.Contains(x.AreaId) && !string.IsNullOrEmpty(x.ShelvesInfo.PayLoad)).ToList();

}

}

}

////（2.2）检索全场超时料

//if (areaInventoryLocationList.Count == 0 && timeOut > 0)

// areaInventoryLocationList = allLocation.Where(x => DateTime.Now.Subtract(x.IncomingTime).TotalSeconds > timeOut).ToList();

////(4)

////仍未获取到库位 检索全厂库位

//if (areaInventoryLocationList.Count == 0)

// areaInventoryLocationList = allLocation.Where(x => !string.IsNullOrEmpty(x.ShelvesInfo.PayLoad)).ToList();

List<string> spareList = spareMachines.Split(',').Where(x => !string.IsNullOrEmpty(x)).ToList();

//无物料可用 非特殊模式下 检索备用设备的物料

if (machineMode!=MachineMode.SpecialMode&& areaInventoryLocationList.Count==0&& spareList.Count>0)

{

//备选设备的物料

List<DtoInventoryLocationResponse> spareMachinesShelves = \_dataAccess.GetInventoryFullShevlesLocationByMachineIdList(spareList);

if (spareMachinesShelves.Count>0)

{

areaInventoryLocationList = spareMachinesShelves;

aLogger.Info($"[{machineId}],本库区无料可用,从备选设备[{spareMachines}]发现可用物料", "JobProcessed");

}

}

//对应库区组均无可用物料 检索全厂库位

//验证货架状态和FIFO

//优先获取近距离库位

//获取近距离库位

List<string> closeLocations = closeLocation.Split(',').Where(x => !string.IsNullOrEmpty(x)).ToList();

DtoInventoryLocationResponse targetLocation = CheckCloseLocationsAndFIFO(areaInventoryLocationList, closeLocations, timeOut);

if (targetLocation != null && targetLocation.ID != 0)

return targetLocation;

aLogger.Info($"[{machineId}],取料库区集合[{fromBuffersList}]设备组来源:[{fromMachines}]上一工艺段超时时间:[{timeOut}] 设备模式:{machineMode} 未找到合适库位或合适货架", "JobProcessed");

return null;

}

catch (Exception ex)

{

aLogger.Info($"推荐PUT任务起点库位发生异常,{ex.Message} 入参:{machineId}", "JobProcessed");

return null;

}

}

/// <summary>

/// 检查同工艺段设备的剩余物料

/// </summary>

/// <param name="allLocation"></param>

/// <returns></returns>

private List<DtoInventoryLocationResponse> CheckSameProcessMachineMaterial(List<DtoInventoryLocationResponse> allLocation,List<int> areaId,string machineid){

try

{

if (allLocation.Count == 0)

return allLocation;

aLogger.Info($"设备[{machineid}]获取定线优先级库位 可选库位:[{JsonConvert.SerializeObject(allLocation.Select(x => x.LocationName))}]", "CheckSameProcessMachineMaterial");

List<DtoInventoryLocationResponse> result = new List<DtoInventoryLocationResponse>();

List<string> propNameList = new List<string>();

propNameList.Add(PropertyDes.ProcessSegment);

propNameList.Add(PropertyDes.BindingShelf);

propNameList.Add(PropertyDes.CloseTargetMachines);

//查询全部设备属性

List<DtoMachineProperty> allPropertys = \_dataAccess.GetMachinePropertieListByNameList(propNameList);

//List<DtoMachineProperty> allPropertys = \_dataAccess.GetMachineProperties();

//1获取同工艺段的设备id

List<DtoMachineProperty> propList = allPropertys.Where(x=>x.MachineID== machineid).ToList();

string process = propList.Where(x => x.Name == PropertyDes.ProcessSegment).FirstOrDefault().Value;

//查询同工艺段的所有设备

List<DtoMachineProperty> processMachines = allPropertys.Where(x=>x.Name==PropertyDes.ProcessSegment &&x.Value==process).ToList();

List<string> machineIdList = processMachines.Select(x => x.MachineID).Distinct().ToList();

//查询同工艺段内未绑定货架的设备

List<DtoMachineProperty> processMachinesWithShelves = allPropertys.Where(x=>x.Name==PropertyDes.BindingShelf&& machineIdList.Contains(x.MachineID)).ToList();

List<string> emptyMachines = processMachinesWithShelves.Where(x => string.IsNullOrEmpty(x.Value)).Select(x => x.MachineID).ToList();

List<string> fullMachines = processMachinesWithShelves.Where(x => !string.IsNullOrEmpty(x.Value)).Select(x => x.MachineID).ToList();

List<string> cantUseLocations = new List<string>();

List<DtoMachineProperty> sourceMachinesProps = allPropertys.Where(x => x.Name == PropertyDes.CloseTargetMachines).ToList();

//查询所有指向未绑定货架的设备的上工艺段机台

foreach (var item in emptyMachines)

{

//所有指向该设备的设备id

List<string> sourceMachies = sourceMachinesProps.Where(x => x.Value.Contains(item)).Select(x => x.MachineID).Distinct().ToList();

foreach (var sourceMachine in sourceMachies)

{

List<DtoInventoryLocationResponse> wip = allLocation.Where(x => x.ShelvesInfo.Source == sourceMachine).ToList();

//如果对应的在制数量小于2表示不可送至其它机台 此库位不可用

if (wip.Count()<2)

{

cantUseLocations.AddRange(wip.Select(x=>x.LocationName).ToList());

aLogger.Info($"设备[{machineid}]上料时尝试获取同工艺段设备[{item}(空)]的临近设备物料,排除库位:[{JsonConvert.SerializeObject(wip.Select(x=>x.LocationName))}]", "CheckSameProcessMachineMaterial");

}

}

}

foreach (var item in fullMachines)

{

//所有指向该设备的设备id

List<string> sourceMachies = sourceMachinesProps.Where(x => x.Value.Contains(item)).Select(x => x.MachineID).Distinct().ToList();

foreach (var sourceMachine in sourceMachies)

{

List<DtoInventoryLocationResponse> wip = allLocation.Where(x => x.ShelvesInfo.Source == sourceMachine).ToList();

//如果对应的在制数量大于1表示不可送至其它机台 此库位不可用

if (wip.Count() >1)

{

cantUseLocations.AddRange(wip.Select(x => x.LocationName).ToList());

aLogger.Info($"设备[{machineid}]上料时尝试获取同工艺段设备[{item}(非空)]的临近设备物料,排除库位:[{JsonConvert.SerializeObject(wip.Select(x => x.LocationName))}]", "CheckSameProcessMachineMaterial");

}

}

}

aLogger.Info($"设备[{machineid}]获取定线优先级库位 不可选库位:[{JsonConvert.SerializeObject(cantUseLocations)}] \r\n \r\n", "CheckSameProcessMachineMaterial");

//获取不可用库位

result = allLocation.Where(x => areaId.Contains(x.AreaId)&& !string.IsNullOrEmpty(x.ShelvesInfo.PayLoad)&& !cantUseLocations.Contains(x.LocationName)).ToList();

return result;

}

catch (Exception ex)

{

aLogger.Info($"定线优先级验证发生异常{ex}", "CheckSameProcessMachineMaterial");

return allLocation;

}

}

/// <summary>

/// 检索切换过库区的库位

/// </summary>

/// <returns></returns>

private List<DtoInventoryLocationResponse> CheckChangeInventoryAreaLocations(List<DtoInventoryLocationResponse> allLocation, List<DtoInventoryLocationResponse> areaInventoryLocationList,List<DtoInventoryAreaResponse> areaList,string fromMachines)

{

try

{

//优先级不同(符合二次调度)

if (areaList.Select(x => x.Priority).Distinct().Count() > 1)

{

//专属库区

DtoInventoryAreaResponse highestArea = areaList.OrderBy(x => x.Priority).FirstOrDefault();

//二次调度不可用

bool inventorySchduleUnable = InventorySchduleUnable(highestArea,fromMachines);

//二次调度不可用时 直接获取所有库位中的切换过库区的库位

if (inventorySchduleUnable)

{

areaInventoryLocationList = allLocation.Where(x => x.HistoryAreaId != 0 ).ToList();

}

//二次调度可用时只获取专属库区的物料 避免和二次调度发生冲突

else

{

//查询不符合二次调度的库区组(组内库区的优先级一致) 此类库区组内的库区中的库位可以优先使用

List<DtoInventoryAreaGroupResponse> areaGroup = \_dataAccess.GetAllInventoryAreaGroup();

List<int> areaIdList = new List<int>();

foreach (var item in areaGroup)

{

if (item.InventoryAreaList.Count == 0)

continue;

//库区组内的库区优先级不一致 只取优先级最高的库区

if (item.InventoryAreaList.Select(x => x.Priority).Distinct().Count() > 1)

{

//只取优先级最高的库区

areaIdList.Add(item.InventoryAreaList.OrderBy(x => x.Priority).FirstOrDefault().ID);

}

else

{

areaIdList.Add(item.InventoryAreaList.FirstOrDefault().ID);

}

}

//获取到所有的库区ID后去重

areaIdList = areaIdList.Distinct().ToList();

//未获取到绑定设备的货架 尝试获取切换了库区的库位(切换过库区的带货架库位 货架优先消耗)

//得到不满足二次调度的库区后 获取该库区内的库位

areaInventoryLocationList = allLocation.Where(x => x.HistoryAreaId != 0 && areaIdList.Contains(x.HistoryAreaId)).ToList();

}

}

else

{

areaInventoryLocationList = allLocation.Where(x => x.HistoryAreaId != 0).ToList();

}

}

catch (Exception ex)

{

aLogger.Info($"获取切换库区库位发生异常{ex}", "CheckChangeInventoryAreaLocations");

}

return areaInventoryLocationList;

//List<DtoInventoryAreaGroupResponse> GetAllInventoryAreaGroup

}

private bool InventorySchduleUnable(DtoInventoryAreaResponse highestArea,string fromMachines) {

try

{

//各个设备组所需库位数量

Dictionary<string, int> machineGroupLocationsCount = new Dictionary<string, int>();

#region 查询专属库区的所有设备组和成员键值对

//1 获取该库区的目标机台 (fromBuffers为库区名称)的所有设备

List<DtoMachineProperty> propList = \_dataAccess.GetMachinePropertieListByNameList(new List<string>() { PropertyDes.FromBuffers, PropertyDes.FromMachines, PropertyDes.MCSControlStatus });

//查询fromBuffers为库区名称的设备id

List<string> machineList = propList.Where(x => x.Name == PropertyDes.FromBuffers && x.Value.Contains(highestArea.InventoryAreaName)).Select(x => x.MachineID).Distinct().ToList();

//过滤掉屏蔽的设备

List<string> shildMachineIdList = propList.Where(x => x.Name == PropertyDes.MCSControlStatus && x.Value == "true").Select(x => x.MachineID).Distinct().ToList();

//2 查询该设备的FromMachines 用作包号过滤

List<string> fromMachinesList = propList.Where(x => !string.IsNullOrEmpty(x.Value) && x.Name == PropertyDes.FromMachines && machineList.Contains(x.MachineID)).Select(x => x.Value).Distinct().ToList();

//3查询现有的LTOL任务 用作获取某种产品的在途货架

List<DtoJob> ltolJobList = \_dataAccess.GetLToLJobs();

//所有设备组的成员总数

double membersTotalCount = propList.Where(x => !shildMachineIdList.Contains(x.MachineID) && fromMachinesList.Contains(x.Value) && x.Name == PropertyDes.FromMachines).Select(x => x.MachineID).Distinct().Count();

//专属库区的全部库位

List<DtoInventoryLocationResponse> highestAreaLocations = \_dataAccess.GetInventoryLocationByAreaId(highestArea.ID);

//专属库区的库位总数

double total = highestAreaLocations.Where(x=>x.IsEable != "Disable").Count();

//当前设备组的成员数量

double membersCount = propList.Where(x => !shildMachineIdList.Contains(x.MachineID) && x.Value == fromMachines && x.Name == PropertyDes.FromMachines).Select(x => x.MachineID).Distinct().Count();

double memberRate = Math.Round(membersCount / membersTotalCount, 4);

//根据占比算出应得库位数量

int needLoactionCount = 0;

//成员占比大于0.5则向下取整

if (memberRate > 0.5)

needLoactionCount = (int)Math.Floor(Math.Round(total \* memberRate, 2));

//向上取整

else

needLoactionCount = (int)Math.Ceiling(Math.Round(total \* memberRate, 2));

//获取专属库区已存在的物料数量和在途的任务数量

int fullLoctionCount = CheckSourceMachine(highestAreaLocations,fromMachines).Count();

//获取设备组的在途任务

List<DtoJob> onRoadJobs = CheckSourceMachineForJob(ltolJobList, fromMachines);

int currentCount = fullLoctionCount + onRoadJobs.Count();

//存在的数量大于需要的数量

if (currentCount > needLoactionCount)

return true;

else

return false;

#endregion

}

catch (Exception ex)

{

aLogger.Info($"获取切换库区库位验证二次调度是否可用时发生异常,{ex.Message}", "JobProcessed");

return true;

}

}