CrowdOS 算法库

1.CrowdOS 现有算法

CrowdOS 内核现有四种任务分配算法，分别为：T_Most、PT_Most、T_Random、GGA_I。具体代码可查看内核代码中的 algorithms 部分：

CrowdOS\kernel\src\main\java\cn\crowdos\kernel\algorithms

在内核中使用的默认算法为“DefaultAlgo”，也就是简单的默认实现，如果需要使用上面四种经典任务分配算法，则需要使用“algoSelect”函数来进行算法选择。

2.算法库相关接口

Kernel 中的 algorithms 包定义了系统中所使用的群智感知相关算法，目前提供了任务分配、任务推荐和参与者选择算法接口，分别为 TaskAssignmentAlgo、TaskRecommendationAlgo、ParticipantSelectionAlgo。algorithms 包使用了工厂模式，每一类算法工厂产出一类特定的算法实现，每一个算法工厂都提供了上述三种算法。

接口 AlgoFactory 定义了内核中使用的所有算法的接口，目前定义了三个功能：

返回值	原型	含义
TaskAssignmentAlgo	getTaskAssignmentAlgo();	返回任务分配算法
TaskRecommendationAlgo	getTaskRecommendationAlgo();	返回任务推荐列表算法
ParticipantSelectionAlgo	getParticipantSelectionAlgo();	返回参与者选择算法

TaskAssignmentAlgo、TaskRecommendationAlgo、ParticipantSelectionAlgo 三个接口的定义类似，以 TaskAssignmentAlgo 为例，接口 TaskAssignmentAlgo 定义了任务分配算法的功能接口，包含单任务分配和多任务分配：

返回值	原型	含义
List<Participant>	getTaskAssignmentScheme(Task task);	获取任务分配的参与者列表
List<List<Participant>>	getTaskAssignmentScheme(ArrtList<Task> task);	返回每个任务分配的参与者列表的列表

3. 算法库中的算法接入

Kernel 中的 algorithms 包采用了工厂模式，即每一类工厂产出一类特定的算法实现，因此在 Kernel 中接入新算法时需要创建一个新的算法工厂，在 Kernel 中提供了一个算法适配器 AlgoFactoryAdapter，其中提供了基本的任务分配、任务推荐和参与者选择的三种实现，因此新建的算法工厂可通过继承 AlgoFactoryAdapter 实现。以包含 PT_Most 任务分配算法的算法工厂 PTMostFactory 为例，首先得到此时的系统资源：

```
1 usage  LHY
public class PTMostFactory extends AlgoFactoryAdapter {

    1 usage  LHY
    public PTMostFactory(SystemResourceCollection resourceCollection) {
        super(resourceCollection);
    }
```

然后开始复写任务分配算法，在任务分配算法中存在单任务分配算法的多任务分配算法，在进行算法增加时系统可提供多任务分配算法的实现，但不提供单任务分配算法的实现，因此参赛者在接入新算法时必须至少自行提供单任务分配的实现。以单任务分配的为例，先得到资源池中需要的资源变量：

```
    ParticipantPool participants = resourceCollection.getResourceHandler(ParticipantPool.class).getResource

    int taskNum = 1;

    List<Constraint> taskLocation = task.constraints().stream()
            .filter(constraint -> constraint instanceof POIConstraint)
            .collect(Collectors.toList());
    if (taskLocation.size() != 1) {
        return null;
    }
    List<Participant> candidate = participants.stream()
            .filter(task::canAssignTo)
            .collect(Collectors.toList());
    int workerNum = candidate.size();
```

然后通过提前计算得到新接入算法需要的参数：

```
List<Participant> candidate = participants.stream()
        .filter(task::canAssignTo)
        .collect(Collectors.toList());
int workerNum = candidate.size();


Coordinate tLocation = (Coordinate) taskLocation.get(0);
double[][] distanceMatrix = new double[workerNum][taskNum];
for (int i = 0; i < candidate.size(); i++) {
    Participant worker = candidate.get(i);
    Coordinate wLocation = (Coordinate) worker.getAbility(Coordinate.class);
    distanceMatrix[i][0] = wLocation.euclideanDistance(tLocation);
}

double[][] taskDistanceMatrix = new double[][]{{1}};
```

然后创建已经准备好的算法实例，接入参数得到任务的分配结果：

```
// create algorithm instance
PT_Most pt_most = new PT_Most(workerNum, taskNum, distanceMatrix, taskDistanceMatrix, new int[]{1},  q: 1);
pt_most.taskAssign();

// parser result
List<Participant> assignmentScheme = new ArrayList<>();
pt_most.getAssignMap().keySet().forEach(participantId -> assignmentScheme.add(participants.get(participantId)));
return assignmentScheme;
```

最后在 kernel 中注册自己新接入的算法，则算法接入完成。

```
public void initial(Object...args){
    systemResourceCollection = new SystemResourceCollection();
    try {
        systemResourceCollection.register(new TaskPool());
        systemResourceCollection.register(new ParticipantPool());
        systemResourceCollection.register(new AlgoContainer(new AlgoFactoryAdapter(systemResourceCollection)), resourceName: "DefaultAlgo");
        systemResourceCollection.register(new Scheduler(systemResourceCollection));
        systemResourceCollection.register(new MissionHistory());
        systemResourceCollection.register(new TrustBasedIncentiveImpl());
        systemResourceCollection.register(new AlgoContainer(new PTMostFactory(systemResourceCollection)), resourceName: "PTMost");
        systemResourceCollection.register(new AlgoContainer(new T_MostFactory(systemResourceCollection)), resourceName: "T_Most");
        systemResourceCollection.register(new AlgoContainer(new T_RandomFactory(systemResourceCollection)), resourceName: "T_Random");
        systemResourceCollection.register(new AlgoContainer(new GGA_IFactory(systemResourceCollection)), resourceName: "GGA_I");
    } catch (DuplicateResourceNameException e) {
        throw new RuntimeException(e);
    }
    initialed = true;
}
```

4. 内核中算法使用

想要在内核中使用所新增的算法，首先需要在内核初始化时注册新增的算法工厂，并赋予其独特有代表性的命名，如此将算法工厂加入到系统资源中：

```
systemResourceCollection.register(new AlgoContainer(new PTMostFactory(systemResourceCollection)), resourceName: "PTMost");
```

在内核中提供算法选择功能 algoSelect(String name)，可在其中选择需要的算法，其通过在系统的调度器 Scheduler 中选择需要的算法工厂，进而选择需要的算法。