CrowdOS Kernel 技术文档

v1.0.1

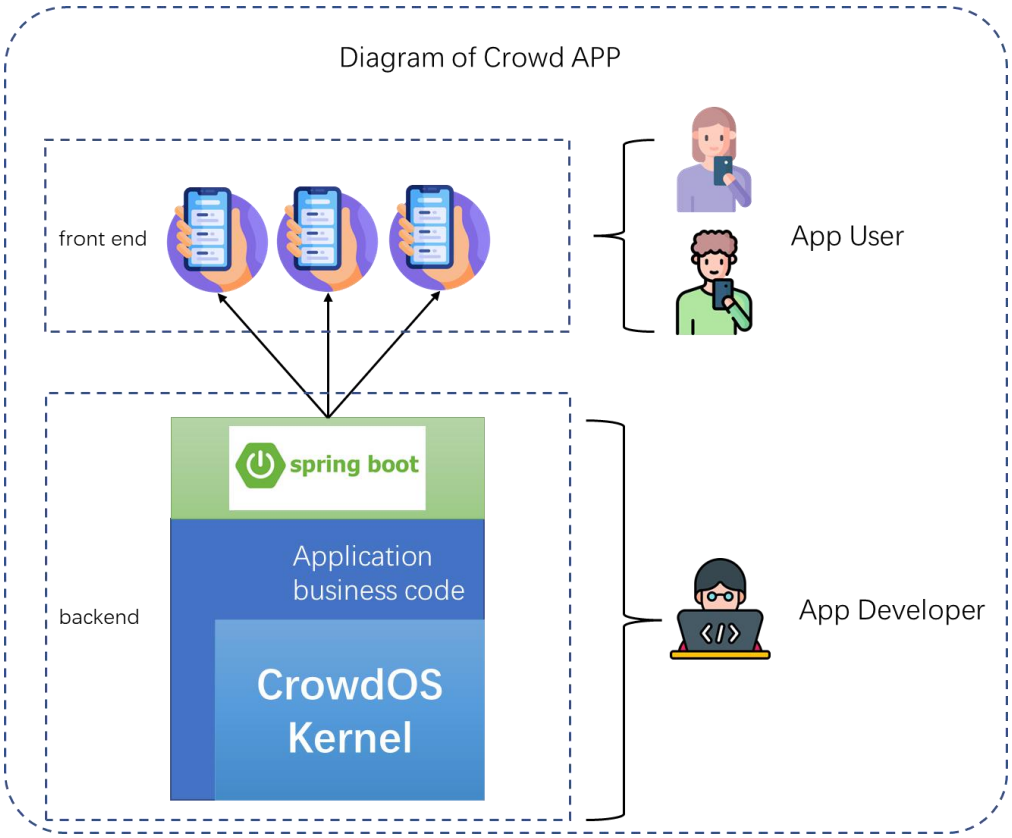
骆艺轩

目录

一、 Kernel 整体架构

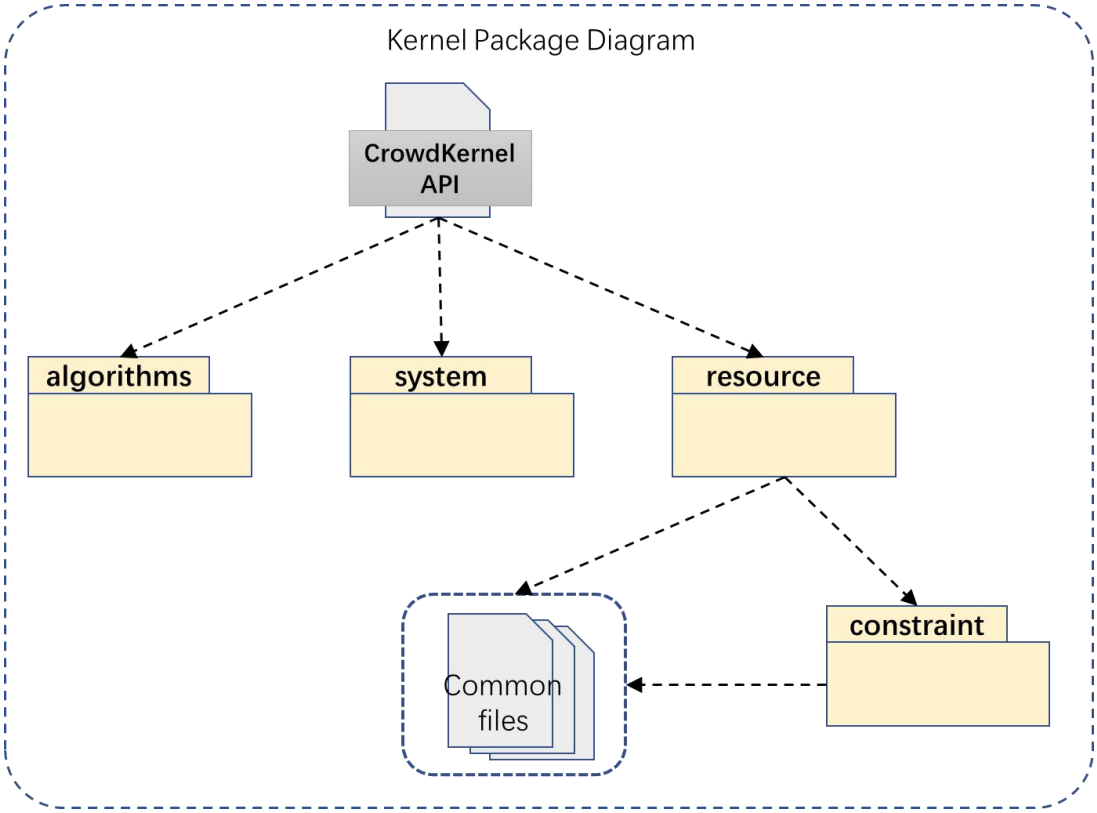
1. CrowdOS Kernel 与其他框架

CrowdOS 目标在于提升群智应用的构建效率，降低群智应用的使用门槛。目前 CrowdOS 使用 CrowdOS Kernel 和 SpringBoot 等 Web 框架快速开发基于群智感知功能的群智 APP。一个使用 CrowdOS Kernel 开发的群智 APP 的框架图如下：



2. CrowdOS Kernel 内部架构

CrowdOS Kernel 提供了支撑群智功能的核心实现，目前 v1.0.1 的内核实现包图如下：



开发者使用通过 *CrowdKernel* 接口（具体实现为 *Kernel.java*）与内核交互。CrowdOS Kernel 会自动管理系统中的所有任务与参与者（*system* 包）以及任务分配与调度流程（*Scheduler.java*）。系统提供了~~丰富~~的群智算法实现以及自定算法实现接口（*algorithms* 包）。

二、 constraint 与 resource 包

1. 设计逻辑

群智感知应用中关键的两个因素是任务与完成任务的参与者。在 CrowdOS Kernel 中任务被定义为包含一组**约束**（*constraint* 包）条件的开发者自定义**任务**（*resource* 包），参与者为包含一组能力的开发

者自定义**参与者**（*resource* 包）。

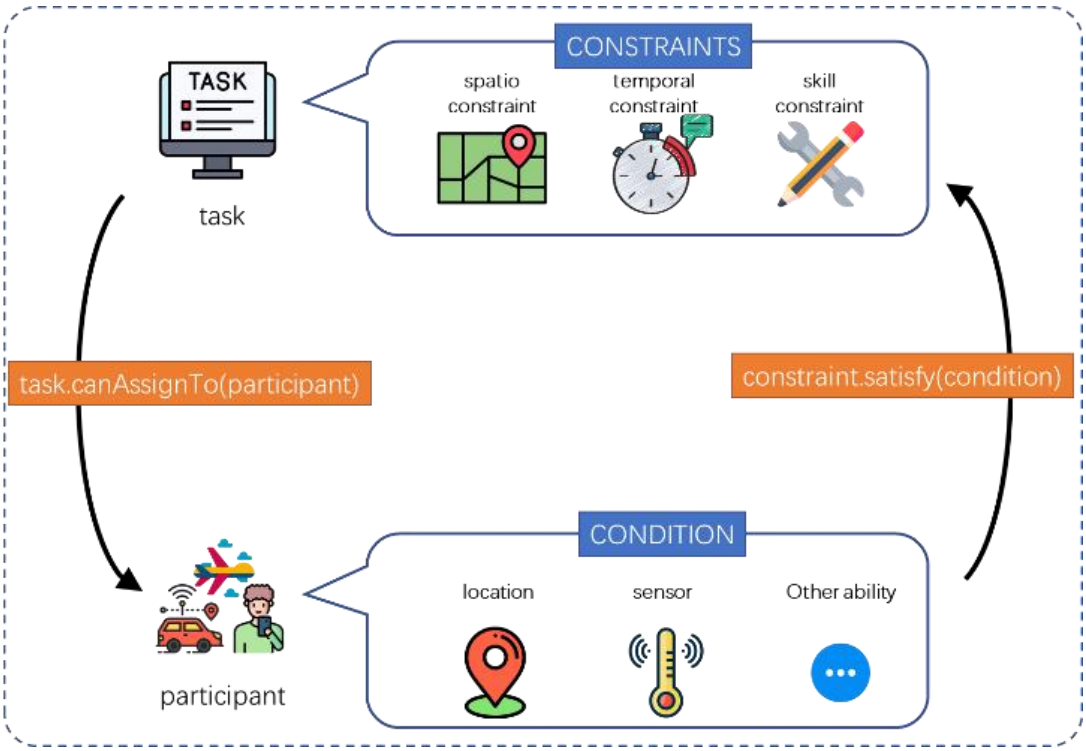
任务 *Task*（实际情况为实现 *Task* 接口的程序员自定义实体类）通过方法 *canAssignTo()*，检测该任务能否分配给某个参与者 *Participant*（实际情况为实现 *Participant* 接口的程序员自定义实体类）。

每一个约束（实际情况为实现 *Constraint* 接口的程序员自定义实体类）通过方法 *satisfy()*，检测某条件（实际情况为实现 *Condition* 接口的程序员自义定的实体类）是否满足自身要求。

任务能否分配给某个参与者的充分必要条件为：任务的所有约束（*Constraint*）条件都要被该参与者所具备的能力（*Ability*）满足（在目前的术语中，参与者的能力（*Ability*）与满足约束的条件（*Condition*）是同一个意思）。在具体实现中，方法 *canAssignTo()*会做两个检查：

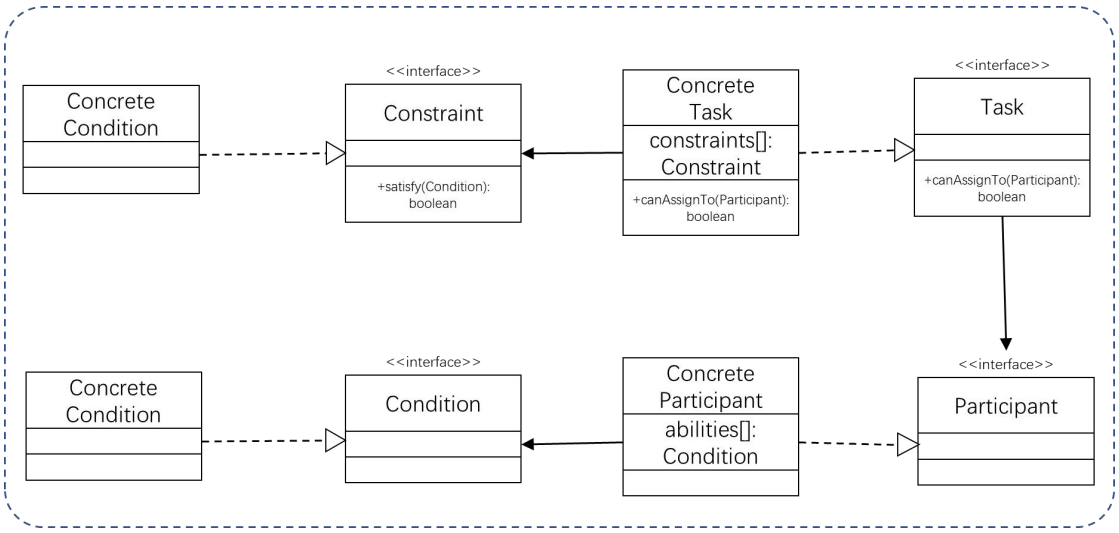
1. 检查对于任务要求的某约束类型，该参与者是否具体满足该类型约束的能力。例如，任务要求在某一区域执行任务，此时需要坚持参与者是否提供了自己的 GPS 信息。
2. 检查参与者所具有的某个能力，是否是能满足该约束的条件。

例如，检查参与者的 GPS 位置信息是否在任务要求的范围内。设计逻辑图如下所示:



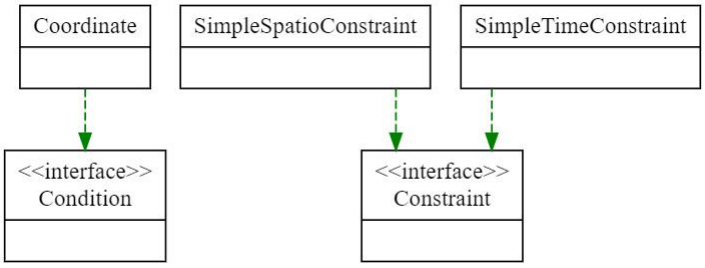
2. 设计模式

constraint 包与 resource 包中的类实现了一种 Double Dispatch 模式（双分派的另一个常用的实现是 Visitor 模式）。同时使用了一些反射技术优化了代码实现。具体类图如下：

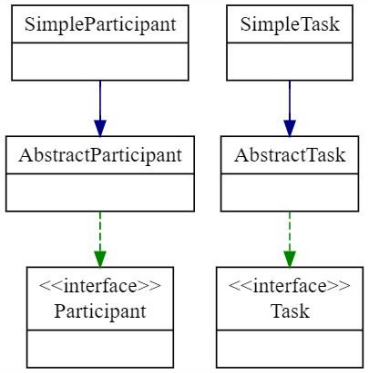


3. 其他内容

constraint 包中包含了构建任务的 *Constraint* 接口和构建参与者的 *Condition* 接口，并且提供了一部分实现简单实现。此外，在 constraint.wrapper 包中，提供了基础类型的 *Condition* 版本（此处的逻辑与 JAVA 包装类相同）。



resource 包中除过关键的 *Task* 和 *Participant* 接口外，还提供了对应的 Abstract Class 分别为 *AbstractTask* 和 *AbstractParticipant*。当程序员开发自定义任务和参与者时，只要继承对应的抽象基类，而不必从实现基础接口开始（此处的设计逻辑与 JAVA Container 部分设计思路相同）。此外还提供了一些示例性的实体类。



三、 system 包

1. 设计逻辑

system 包中的 SystemResourceCollection 管理了系统中的所有实体。目前，系统实体包括 TaskPool、ParticipantPool、AlgoContainer 和 Schedule。系统实体需要继承 Resource 接口，实现 getHandler()方法。出于对系统实体的保护，所有访问系统实体的操作都应该通过 getHandler()方法，该方法返回某一个具体实体的处理句柄。其他包在访问系统实体时，必须通过实体句柄。系统实体句柄 SystemResourceHandler<T>提供了两类访问方式：

1. T getResourceView()：访问该实体的不可更改视图；

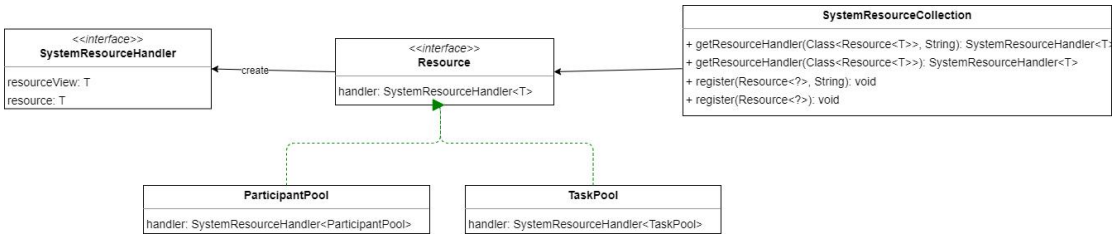
2. T getResource()：访问该实体本身。

system 包对其他包（或代码）做出了一个约定：当其他包（或其他代码）使用 getResourceView()时，system 包保证系统功能正常（例如，调度系统、任务池管理、参与者池管理等）；当使用 getResource()时，system 包不提供该保证。

基于 system 包提供的保证，其他包可以放心的通过系统实体句柄实现自己的功能。例如，在 algorithms 包中，算法的实现需要访问系统实体的各项信息，因此该包中所有操作仅可使用 getResourceView()。

2. 设计模式

system 包实现了一种类似于迭代模式的保证。具体的 UML 类图如下：



四、 algorithms 包

1. 设计逻辑

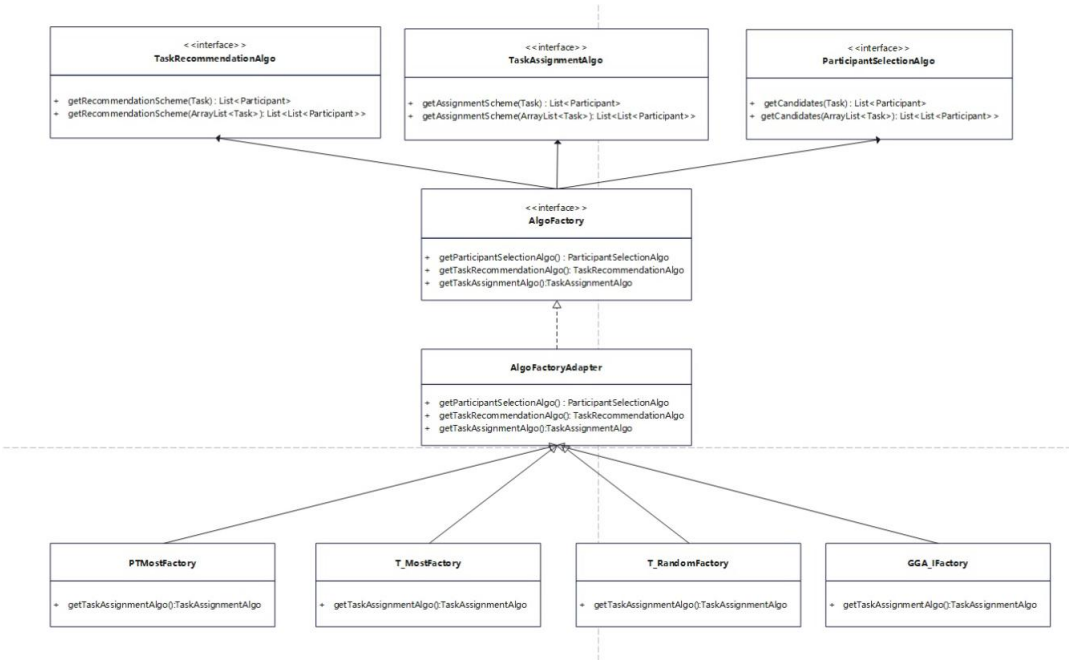
algorithms 包定义了系统中所使用的群智感知相关算法，目前提供了任务分配、任务推荐和参与者选择算法接口。algorithms 包实现了算法与系统流程的解耦。在实现 algorithms 包中算法的过程中，基于 system 包提供的保证，可以解耦算法流程与系统流程，确保系统流程的稳定运行。

2. 设计模式

algorithms 包涉及的设计模式主要有两种，分别为：

1. algorithms 自身使用了工程模式。每一类算法工厂产出一类特定的算法实现。目前每个算法工厂需要分别实现任务分配、任务推荐和参与者选择算法；
2. algorithms 与 Scheduler 交互时，algorithms 以模板模式的形式嵌入 Scheduler。

algorithms 包的 UML 类图如下：



3. 算法说明

algorithms 包中提供了四种经典的任务分配算法，分别为 T_Most、PT_Most、T_Random、GGA_I。四种算法对应的算法工厂均继承自算法适配器 AlgoFactoryAdapter，每种算法都可支持但任务分配和多任务分配。

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

算法适配器 AlgoFactoryAdapter 实现接口 AlgoFactory，为系统提供了默认任务分配、任务推荐和参与者选择算法，若不进行算法选择，

则系统提供其中的默认算法实现，具体算法的接入可通过继承算法适配器 AlgoFactoryAdapter 实现。

五、 CrowdKernel 系统接口与实现

1. CrowdKernel 接口

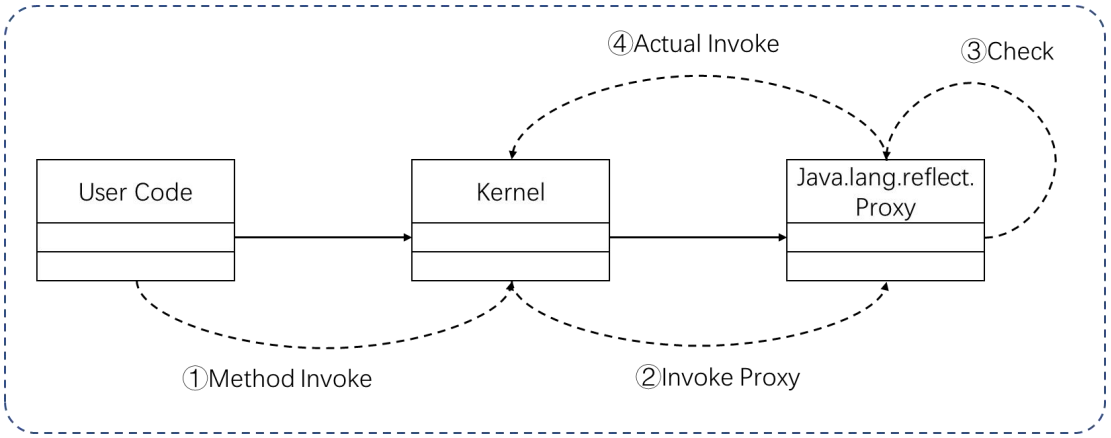
接口 CrowdKernel 定义了程序员与内核功能交互的接口。目前 CrowdKernel 定义了如下功能：

返回值	原型	含义
boolean	isInitialed();	返回内核是否初始化
void	algoSelect(String name);	选择需要的算法
void	initial(Object...args);	以参数初始化内核
void	initial();	初始化内核
SystemResourceCollection	getSystemResourceCollection();	获取系统实体集合对象
boolean	submitTask(Task task);	提交任务
List<Task>	getTasks();	获取系统中的任务
List<Participant>	getTaskAssignmentScheme(Task task);	获取任务的分配结果
List<List<Participant>>	getTaskAssignmentScheme(ArrayList<Task> tasks);	获取多任务的分配结果
List<Participant>	getTaskRecommendationScheme(Task task);	获取任务的推荐结果
List<List<Participant>>	getTaskRecommendationScheme(ArrayList<Task tasks);	获取多任务的推荐结果
List<Participant>	getTaskParticipantSelectionResult(Task task);	获取参与者选择结果
List<List<Participant>>	getTaskParticipantSelectionResult(ArrayList<Task tasks);	获取多任务的参与者选择结果
List<Participant>	getParticipants();	获取系统中的参与者

2. Kernel 实现

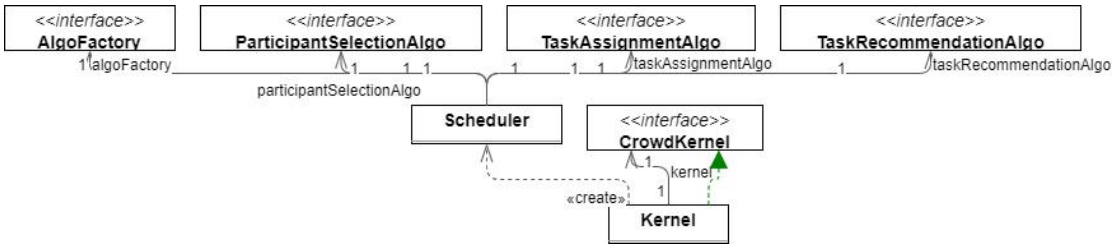
Kernel.java 是接口 CrowdKernel 的具体实现。Kernel 主要负责管理系统中的各类实体，并提供一些接口调用保护。目前使用 JAVA 代理模式接口完成了方法调用安全，主要检查在调用方法前，内核需要

完成初始化。具体的 UML 视图如下：



3. Scheduler

Scheduler 实体负责系统的任务分配与任务调度流程，其依赖于不同算法工厂来实现不同的调度效果。Scheduler 与其他系统实体关系为：



六、 可分解的与分解器

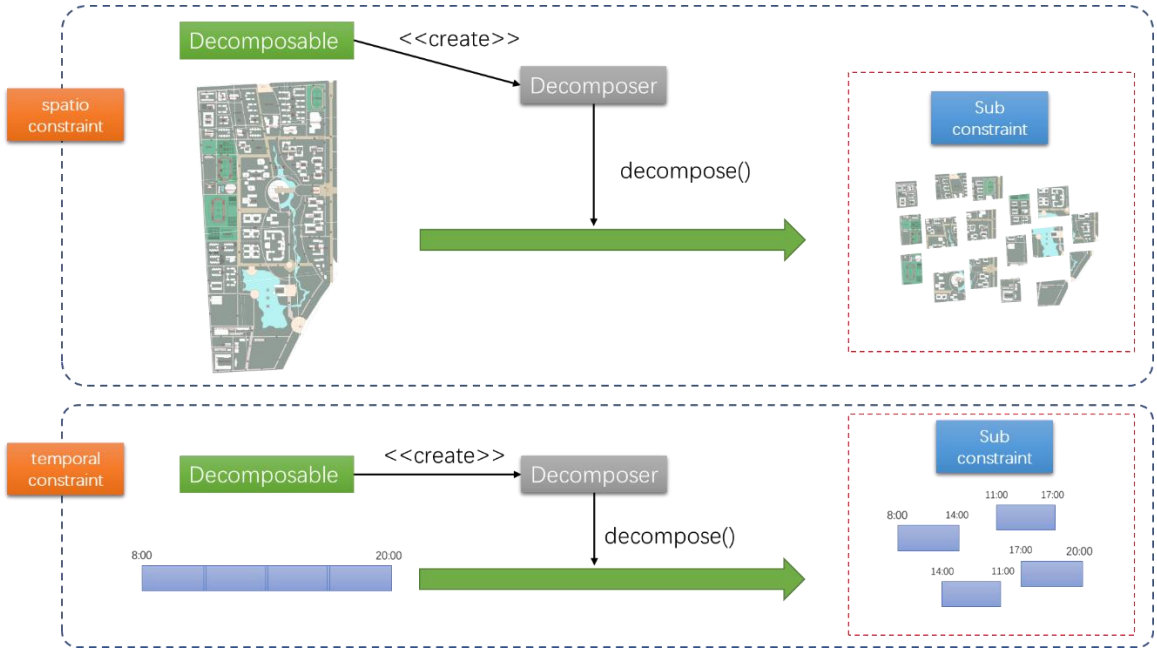
1. 设计逻辑

在群智感知中往往需要将一个大任务分解为多个子任务。分解步骤有：

1. 按一定规则将任务的所有约束分解为子约束，最终获得一个多个子约束集合的集合。

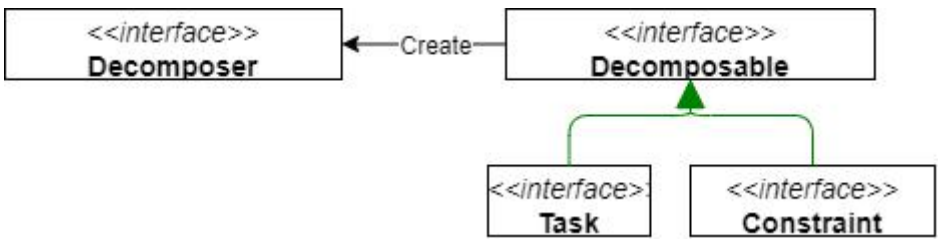
2. 将多个子约束集合进行笛卡尔乘积，每一个乘积结果为一个子任务。

例如，有一个环境监测任务，任务要求监测西北工业大学长安校区内8:00-20:00 校园内噪声情况。对于该任务，有两个约束条件：1.为空间约束（西北工业大学长安校区）；2. 为时间约束（8:00-20:00）。第一步需要将空间约束分解小的区域、将时间约束分解为小的时间段。第二步将两个子约束结合进行笛卡尔积，最终得到子任务。



2. 设计模式

Decomposable 和 Decomposer 的设计模式与 Iterable 和 Iterator 的设计模式类似。具体的 UML 类图如下：



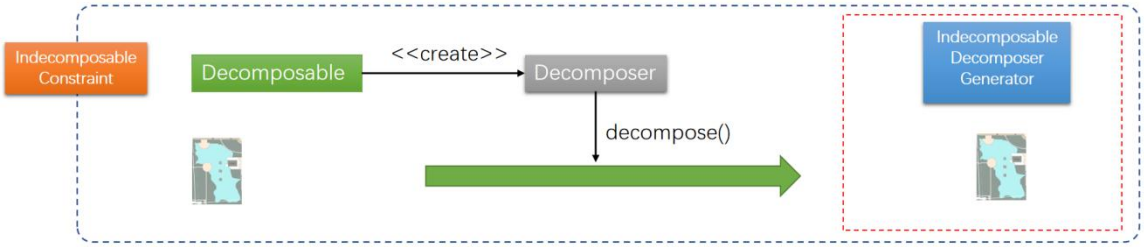
七、 分解器生成器

1. 设计逻辑

在群智感知任务中仍然存在许多不可分解的任务，其在系统中不需要被分解为多个子任务。我们针对此种情况使用反射技术更新了相应生成器的默认行为。大致步骤如下：

1. 获取不可分解约束的 Class 对象以及相应构造参数

2. 使用 Class 对象生成对应分解器构造器（IndecomposableDecomposerGenerator）



2. 设计模式与形式

其设计模式依赖于Decomposable 和Decomposer。

其设计形式如下所述：

IndecomposableDecomposerGenerator 设计靠近上层应用开发人员，使用反射技术获取Class对象并创建相应生成器。IndecomposableConstraint 应为开发使用人员具体修改。

八、 隐私保护模块的接口与实现

1. 设计逻辑

在群智感知中，隐私保护是保障任务安全运行和鼓励参与者执行任务的必要手段。而参与者的位置隐私尤为重要。

因此，在Constraint包中，提供了PrivacyConstraint包作为具有参与者位置隐私保护功能的约束包。其中Paillier类提供对Coordinate类型的位置进行加密和解密的方法。EncryptCoordinate是对加密位置的定义，包含加密后的经度和纬度。SimpleEncryptSpatioConstraint是简单的加密空间约束，提供了使用密文下的位置进行任务的匹配过程。

2. 设计模式

其设计模式依赖于Condition和Constraint.具体的UML类图如下所示：