Sundial——分布式定时任务系统

一. 设计背景

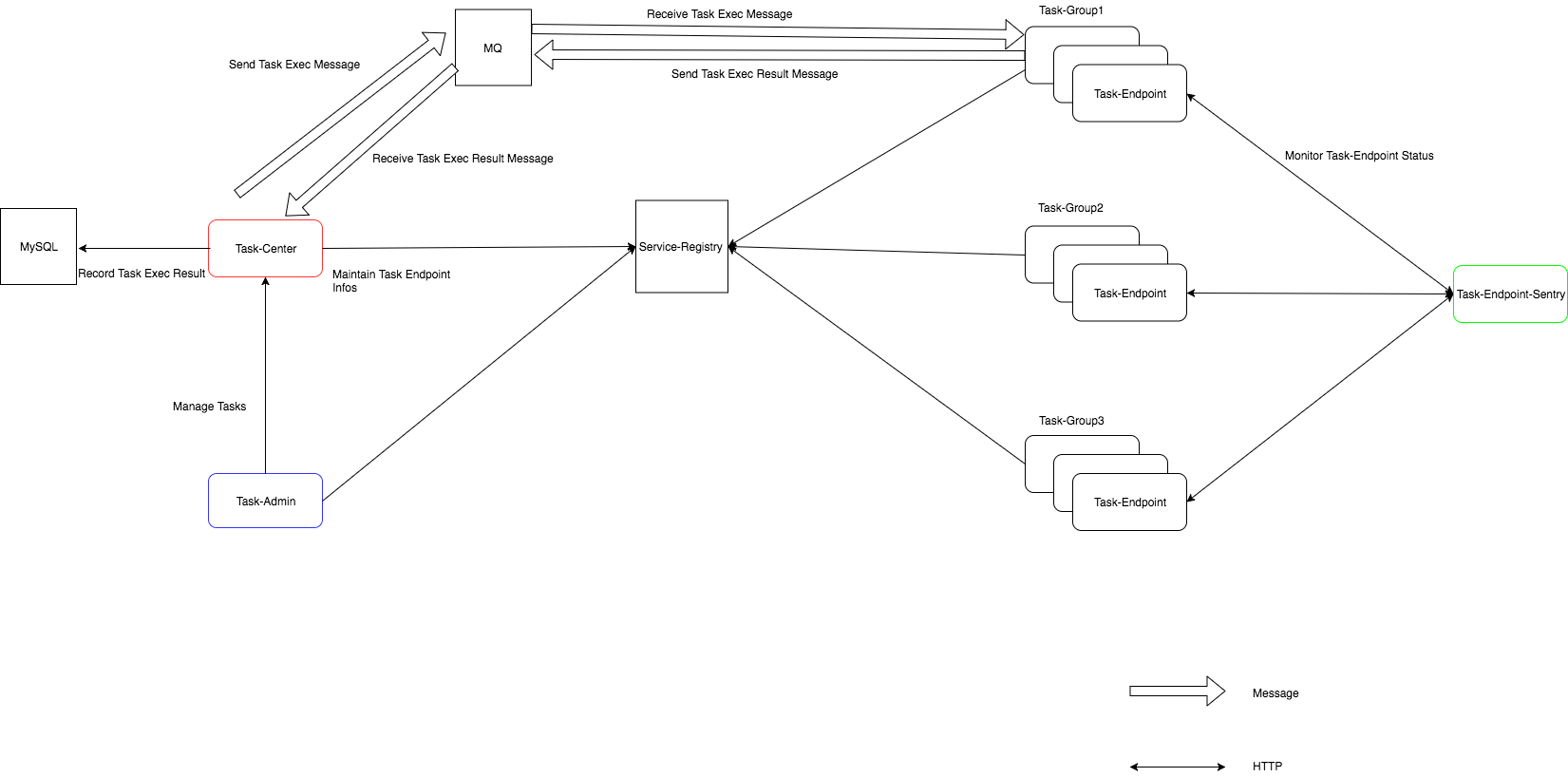
目前，系统中的定时任务大部分都通过部署环境配置，这种方式只能实现较简单的脚本式任务。而且各应用的定时任务独立配置和运行，不便于统一管理。因此，急需开发一套定时任务系统，统一管理所有应用的定时任务，以监控各任务的运行状态，并可以实现任务的动态添加与删除。

二. 待实现的功能(按照优先级排序)

1. 定时任务的统一管理和状态监控(P0)
2. 避免多容器部署情况下，同一个定时任务被不同的容器执行多次(P0)
3. 任务的动态创建与删除(P1)
4. 任务的持久化(P2)：不考虑定时任务的中断恢复，因此暂不处理持久化，只记录定时任务的执行状态与执行结果
5. 引入执行组的概念，同执行组以负载匀衡的方式执行同一任务，且支持执行节点的故障转移与警报(P2)

三. 系统架构

系统名称：Sundial——日晷，古代用来计时。



注：标记颜色的模块需开发。

四. 各模块功能介绍

1. Task-Center：定时任务中心，主要负责定时任务的调度。Task-Center通过一个注册表维护了所有Task-Endpoint的节点信息，并通过与Task-Endpoint的通信，调用Task-Endpoint执行任务。
2. Task-Endpoint：任务执行节点，负责实际定时任务的执行。执行结束后与Task-Center通信，发送执行结果。
3. Task-Endpoint-Sentry：任务哨兵，通过心跳，定时监控Task-Endpoint的状态，动态更新节点信息并同步到Task-Center；当节点故障时报警。
4. Task-Admin：定时任务管理后台。Task-Admin通过REST接口直接访问Task-Center，以实现定时任务的查看、动态创建与删除，图形化管理后台考虑后期单独开发。

五. 核心概念描述

1. 任务管理相关
   1. Task-Management-Authority：定时任务管理权限

|  |
| --- |
| public enum TaskManagementAuthority {  NULL(0), //无权限  READ(1), //可读  WRITE(2), //可写(包括可创建可删除)  EXECUTE(4), //可执行(可以手动触发定时任务的执行)  ; } |

* 1. Task-Management-Group：定时任务管理组，同一个管理组具有相同的任务管理权限

|  |
| --- |
| public class TaskManagementGroup {  private String id;  private String name; //管理组名称  private TaskManagementAuthority authority; //该管理组所具有的权限 } |

1. 任务执行相关
   1. ServiceMetaData：服务元信息，基于服务发现组件获取，记录服务名称和HA地址等信息

|  |
| --- |
| public class ServiceMetaData {  private String serviceName; //服务名称  private InetAddress haAddr; //HA地址 } |

* 1. Task-Execution-Group：任务执行组，在逻辑上对任务进行分组，同一个Task-Execution-Group内的实例执行相同逻辑的定时任务。

|  |
| --- |
| public class TaskExecutionGroup {  private String id;  private String name;  private ServiceMetaData serviceMetaData; //服务元信息  private Map<String,TaskEndpoint> taskEndpoints; //维护所有节点实例 } |

目前按照服务的Service-Name划分Task-Group，使用唯一标识Task-Execution-Group-Id。

* 1. Task-Endpoint：任务执行节点，负责实际定时任务的执行，一个Task-Group中包含多个相同逻辑的Task-Endpoint。在某个时间点，一个Task-Group最多只允许一个Task-Endpoint执行任务。

|  |
| --- |
| public class TaskEndpoint {  private String id;  private String name; } |

目前按照项目中的容器划分Task-Endpoint，使用唯一标识Task-Endpoint-Id。

1. TaskMetaData：定时任务元信息，记录定时任务所在的服务名、类名和方法名等；

|  |
| --- |
| public class TaskMetaData {  private String serviceName; //所在服务名  private String className; //所在类名  private String methodName; //所在方法名 } |

1. TaskExecutionContext：任务执行上下文，主要定义任务执行所需的参数等，在执行任务时传入

|  |
| --- |
| public class TaskExecutionContext {  private Object[] executionParams; //任务执行所需的参数  private String targetTaskEndpointId; //执行任务的目标TaskEndpointId } |

1. TaskDefinition：定时任务定义，封装对定时任务的配置信息

|  |
| --- |
| public interface TaskDefinition {} |

目前仅支持注解形式的任务定义

|  |
| --- |
| //基于注解的定时任务定义 public class AnnotationTaskDefinition implements TaskDefinition{  private AnnotationMetadata annotationMetadata; //注解元信息 } |

1. Task：定时任务实例

|  |
| --- |
| public class Task {  private String name;  private String key; //唯一标识  private TaskMetaData metaData; //任务元数据 } |

1. TaskCommand：任务命令，封装Task-Center和Task-Endpoint的通信信息，如ExecutionTaskCommand、RecordTaskResultCommand等

|  |
| --- |
| public interface TaskCommand {} |

|  |
| --- |
| //Task-Center发往Task-Endpoint，执行定时任务  public class ExecutionTaskCommand implements TaskCommand{  private Task targetTask; //待执行的定时任务  private TaskExecutionContext taskExecutionContext; //定时任务执行的上下文信息 } |

|  |
| --- |
| //Task-Endpoint发往Task-Center，定时任务完成，记录执行结果  public class RecordTaskResultCommand implements TaskCommand{  private Task completedTask; //执行完成的task  private TaskExecutionResult result; //执行结果 } |

1. TaskExecutionResult：任务执行结果

|  |
| --- |
| public class TaskExecutionResult {  private int code; //状态码    @Nullable  private String message; //信息  @Nullable  private Error error; //异常 } |

1. TaskExecutionStatus：任务执行状态

|  |
| --- |
| public enum TaskExecutionStatus {  REGISTED(0), //注册完成，未执行  RUNNING(1), //执行中  COMPLETED(2), //执行完成  CANCELED(3), //被终止  ; } |

六. 执行过程抽象

1. 本地任务解析

目前定时任务支持注解式配置，在Task-Endpoint启动时扫描自定义注解，解析定时任务

|  |
| --- |
| //任务解析器，解析程序里配置的定时任务  public interface TaskDefinitionResolver {  //解析定时任务  Task resolveTask(TaskDefinition taskDefinition); } |

目前仅支持注解类型的配置解析

1. 远程任务注册

定时任务解析后，需要统一向远程Task-Center执行注册

|  |
| --- |
| //任务注册器，将解析好的定时任务注册到Task-Center  public interface TaskRegister {  //注册定时任务  boolean registerTasks(List<Task> tasks); } |

1. 远程任务调度

任务注册到Task-Center后，统一交给RemoteTaskDispatcher进行管理和调度

|  |
| --- |
| //任务调度器，在Task-Center端对所有定时任务进行调度，并可动态增加、删除任务  public interface RemoteTaskScheduler {  //添加任务  boolean addTask(Task task);  //删除任务  boolean removeTask(Task task); } |

目前考虑使用类似Spring线程池对定时任务进行调度，后期可能进行优化

1. 任务命令分发器

封装Task-Center与Task-Endpoint的通信细节，发送执行定时任务的请求并接收定时任务的执行结果。可能的通信方式有REST和MQ，目前考虑实现基于MQ的方式。

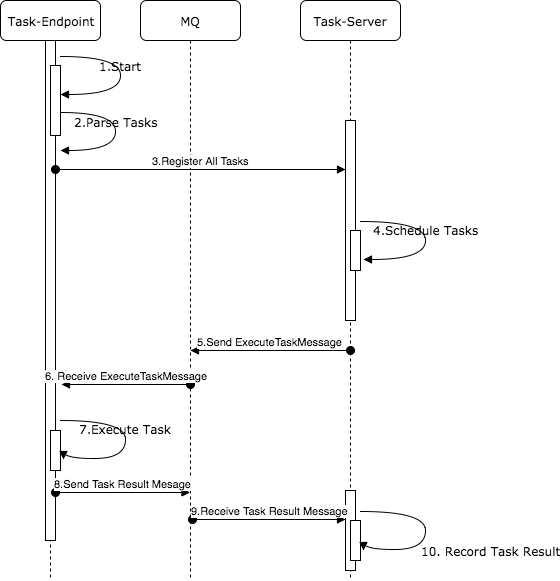
|  |
| --- |
| //任务命令分发器，主要处理Task-Center和Task-Endpoint的通信  public interface TaskCommandDispatcher {  //Task-Center → Task-Endpoint 执行定时任务  boolean sendExecutionTaskCommand(ExecutionTaskCommand command);  //Task-Endpoint → Task-Center 定时任务执行结果  boolean sendRecordTaskResultCommand(RecordTaskResultCommand command);  //发送通用TaskCommand  boolean sendTaskCommand(TaskCommand command); } |

1. 任务执行

Task-Endpoint本地使用TaskExecutor进行实际任务的执行

|  |
| --- |
| public interface LocalTaskExecutor {  //任务执行前置处理  TaskExecutionResult preTask(Task task, TaskExecutionContext context);   //执行任务  TaskExecutionResult executeTask(Task task, TaskExecutionContext context);   //任务执行的后置处理  TaskExecutionResult postTask(Task task, TaskExecutionContext context); } |

七. 定时任务执行流程



1. Task-Endpoint：容器启动
2. Task-Endpoint：扫描项目中定义的定时任务
3. Task-Endpoint：向Task-Center注册定时任务
4. Task-Center：对定时任务进行调度
5. Task-Center：向MQ发送执行定时任务的消息
6. Task-Endpoint：接收到执行任务的消息
7. Task-Endpoint：执行定时任务
8. Task-Endpoint：向MQ发送定时任务执行结果消息
9. Task-Center：从MQ接受消息
10. Task-Center：记录任务的执行结果

八. 技术选型

1. 服务的注册与发现：

基于Consul服务发现组件进行服务的注册与发现；通过Consul获取所有的Task-Group信息，并交于Task-Center进行维护。

1. 任务的解析与注册：

开发定时任务注解和解析器，支持注解式的任务配置；当Task-Endpoint启动时，解析项目中定义的所有定时任务，将其注册到Task-Center中。

1. 任务的调度与执行：

Task-Center内部使用线程池，完成所有定时任务的调度。当要执行某个任务时，Task-Center向MQ发送执行任务的消息。选择一个Task-Endpoint执行指定的定时任务。接口会传入执行任务所需的所有数据以及上下文信息，Task-Endpoint在执行任务后，对于需要持久化的定时任务，保存到Task-Endpoint对应的Redis中。

1. 定时任务管理后台：

Task-Admin与Task-Center通过REST接口交互，实现定时任务的统一管理、动态注册与删除。

九. 实现细节

1. Task-Center与Task-Endpoint的通信方式：
   1. Rest方式：基于Consul服务发现和HAProxy，直接访问Task-Group所在服务的HA地址，由HA处理负载均衡。

优点：可以直接利用HAProxy的least conn负载均衡策略，无需额外实现。

缺点：强依赖于HAProxy；业务方需要暴露URL，存在安全隐患。

* 1. MQ方式：基于现有的ActiveMQ，定时任务的执行和结果记录通过消息的方式进行通信。

优点：纯异步方式，可以实现任务调度与任务执行的解耦；业务方仅需注册一个Topic订阅消息即可。

缺点：Task-Center无法实时获取任务的执行状态和执行结果。

目前考虑使用MQ的通信方式。

1. 执行状态的记录

目前考虑将每个任务实例的执行状态同步记录到MySQL中，方便后期进行统一查询管理。

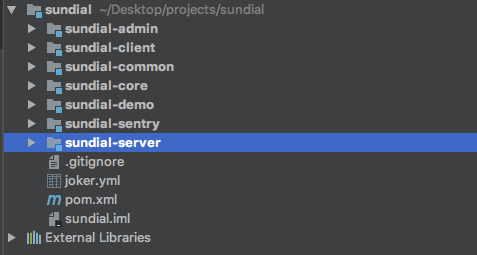
1. 任务的终止

后台提供手动终止任务的功能接口，一期暂不实现。

1. 任务的执行方式

大部分定时任务都是通过在业务里进行配置，通过Task-Center调度去执行的。同时，后台支持任务的手动创建与执行，即在后台输入任务的执行逻辑和所需参数，直接通过MQ发送TaskCommand通知Task-Endpoint去执行任务。

十. 工程结构



sundial-common：一些公用的常量、配置等；

sundial-core：定时任务的核心功能，包括自定义注解、处理器、通信组件等；

sundial-server：定时任务服务端模块，主要实现Task-Center的功能；

sundial-client：第三方系统接入所需要依赖的模块；

sundial-sentry：监控模块；

sundial-demo：示例代码；

sundial-admin：后台管理模块

第三方应用(Task-Endpoint)需要依赖sundial-client包

Task-Center服务需要依赖sundial-server包

后台管理应用依赖sundial-demo包。

十一. 数据库表设计

系统的表结构比较简单，目前仅涉及一张表，记录所有注册的定时任务的执行状态。

|  |
| --- |
| -- 定时任务执行状态表  CREATE TABLE `task\_execution\_status` (  `id` BIGINT(20) NOT NULL AUTO\_INCREMENT,  `task\_name` VARCHAR(64) NOT NULL COMMENT '任务名称',  `task\_key` VARCHAR(32) NOT NULL COMMENT '任务唯一key',  `service\_name` VARCHAR(64) NOT NULL COMMENT '任务所在的service-name',  `management\_group\_name` VARCHAR(64) NOT NULL COMMENT '任务所在的管理组名称',  `execution\_status` VARCHAR(32) NOT NULL COMMENT '任务执行状态',  `last\_execution\_result` VARCHAR(32) NOT NULL COMMENT '上一次执行结果',  `last\_start\_time` DATETIME NOT NULL COMMENT '上一次开始结束的时间',  `last\_end\_time` DATETIME NOT NULL COMMENT '上一次执行结束的时间',  `created\_at` TIMESTAMP NULL DEFAULT CURRENT\_TIMESTAMP COMMENT '创建时间',  `updated\_at` TIMESTAMP NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP COMMENT '更新时间',  PRIMARY KEY (`id`),  UNIQUE KEY `idx\_task\_key` (`task\_key`) ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='定时任务执行状态表'; |

目前暂时考虑记录以上核心字段，随着开发过程再进行扩展。