# Clockwork设计文档

## 1.设计目标

完成一个通用调度系统，作为数据ETL过程的平台。

第一阶段：实现调度后台，依靠数据库作为持久化，并为RP定制界面的最小集合，驱动后台。

第二阶段：配套一个通用界面，方便移植。

## 2.架构设计

### 2.1 调度后台架构

由于调度本身是典型的daemon应用，考虑采用c/s架构分离引擎和控制。

Server端为调度引擎，负责维护一个待调度队列，接受client对调度队列的操作，调度配置信息持久化的操作由界面完成。

Client端作为操作端，对服务器端的计时器线程池发出控制指令。

由于Client端有可能被部署于不同服务器的网页前端调用，c/s端的通信采用socket。

但考虑研究和学习domain socket的需求，对于c/s同处一台服务器的情况，client可选用domain socket进行通信。

Socket/domain socket的选择由client调用方决定。

#### 2.1.1 服务器端架构

服务器端以多线程形式存在，主要线程如下：

##### 2.1.1.1 主线程

主线程主要负责：

* 生成计时器线程，并维护线程池

将每天24小时按常量INTERVAL\_HOUR分成若干个区块，以START\_HOUR为起点将调度时间符合该区块内的任务以生成计时器线程。

* 初始化socket，接收到socket请求时，生成通信线程

##### 2.1.1.2 计时器线程

计时器线程用来在触发时刻执行任务，并控制任务的错误处理（重新执行次数），以及任务结果的输出（重定向stdout,stderr至本地文件系统）。

##### 2.1.1.3 通信线程

针对client发来的信息，在确认权限的情况下，对线程池进行相应的操作，目前支持的操作如下：

|  |  |
| --- | --- |
| 操作 | 服务器响应 |
| 立即执行任务 | 立即执行该任务 |
| 取消一个任务 | 取消一个正在执行的任务 |
| 加入调度 | 计算该任务调度时间，并根据结果决定是否启动计时器线程 |
| 删除调度 | 如果该任务当前处在计时器线程池中，终止 |
| 重加载调度 | 先执行“删除调度”操作，再执行“加入调度”操作 |
| 查询服务器状态 | 返回当前等待和运行的任务计数 |

#### 2.1.2 客户端架构

客户端支持独立命令行启动，也开放API被其它模块调用（主要为网页后台）。

## 3 主要数据结构

### 3.1 工作（Job）

工作包括以下主要属性：

|  |  |
| --- | --- |
| 属性名 | 定义 |
| Id | 唯一标示 |
| 工作时间配置 | 类似crontab格式的时间配置字符串 |
| 命令 | 须执行的shell命令字符串 |
| 重试次数 | 重试次数 |

命令字符串：

* ${param\_name} 代表参数，在执行时需要被正确的字符串所替代
* 每个参数拥有参数名称，参数默认值，默认值支持预置函数

预置函数：

|  |  |
| --- | --- |
| 保留字符 | 需支持的操作 |
| #{TODAY} | 使用yyyymmdd格式转换为今日 |
| #{YESTERDAY} | 使用yyyymmdd格式转换为昨日 |

### 3.2 任务（Task）

根据工作（Job）定义，实例化得到任务。

任务包括以下主要属性：

|  |  |
| --- | --- |
| 属性名 | 定义 |
| Id | 唯一标示 |
| 执行时间 | 确定的时间点 |
| 命令内容 | 替换完参数的命令字符串 |
| 状态 | 该任务的状态 |

状态值有：

|  |  |
| --- | --- |
| 状态 | 定义 |
| 未执行 | 已进入队列，未执行 |
| 执行中 | 正在执行 |
| 执行成功 | 已执行成功 |
| 执行失败 | 执行失败 |

任务的标准输出和标准错误将输出至每个任务的特定的日志文件中