|  |  |  |
| --- | --- | --- |
| **部门/项目代码: YUN/XXXX** | **文档 ID: YUN003** | **Version:** ***V 0.02*** |

|  |  |
| --- | --- |
|  |  |

|  |
| --- |
| 描述:  技术规格文档适用于云事业部产品开发 |

|  |  |
| --- | --- |
| **文档名称:** | ***Mango-DAX计算任务***  ***技术规格文档***  ***Technical Specification***<Replace with organization name, process document name> |

|  |  |  |  |
| --- | --- | --- | --- |
| **状态** | 初稿 | 评审 | 发布 |
| **机密等级** | 外部使用 | 内部使用 | 限制使用 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **文档评审记录** | **部门** | **名字** | **版本** | **日期** |
| **起草** | 软件开发工程师 | 胡健 | V0.1 | 2017-11-16 |
| **评审** | 项目经理  验证负责人  质量负责人  部门经理 |  |  |  |
| **批准** | 昊沧管理层 |  |  |  |

|  |
| --- |
| **分发至** |
| 云事业部研发工程师 |

|  |  |  |  |
| --- | --- | --- | --- |
| **修订历史** | | | |
| **版本** | **日期** | **作者** | **修改内容** |
| 0.01 | 2018-4-9 | 胡健 | 创建文档 |
| 0.02 | 2018-4-13 | 胡健、魏海峰 |  |
| 0.03 | 2018-5-11 | 魏海峰 | 修改取前点算法描述 |
| 0.04 | 2018-5-15 | 魏海峰 | 补充计算任务依赖表结构和Web接口 |
| 0.04 | 2018-5-15 | 胡健 | 精简了计算任务的框架 |
|  |  |  |  |
|  |  |  |  |

目录

[1 介绍 4](#_Toc511133108)

[1.1 参考文档 4](#_Toc511133109)

[1.2 缩略语和术语 4](#_Toc511133110)

[1.3 范围 4](#_Toc511133111)

[2 依赖表结构 4](#_Toc511133112)

[3 详细设计 4](#_Toc511133113)

[3.1 任务调度 4](#_Toc511133114)

[3.2 数据服务 9](#_Toc511133115)

# 介绍

## 参考文档

|  |  |  |  |
| --- | --- | --- | --- |
| 参考文档 | | | |
| 名称 | 版本 | 作者 | 备注 |
| 农污产品需求说明书 | V0.016 | 刘宗永 |  |
| DAX\_2.0（计算任务）\_PS | V0.03 | 魏海峰，胡健 |  |
|  |  |  |  |

## 缩略语和术语

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## 范围

该文档只包含计算任务部分，产品整体架构和DAX其它部分架构请参考“MAX\_2.0\_PS”和”“DAX\_2.0\_PS”。

# 依赖表结构

## 字段属性

### 测点表（T\_MPOINT）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段** | **类型** | **空** | **默认** | **注释** |
| id (主键) | bigint(20) | 否 |  | 主键ID |
| site\_id(外键) | bigint(20) | 否 |  | 所属站点ID：外键关联站点信息表。 |
| mpoint\_id | varchar(20) | 否 |  | 测点编号：自动生成，例如：000001 |
| mpoint\_name | varchar(100) | 否 |  | 测点名称 |
| category\_id(外键) | bigint(20) | 否 |  | 数据分类ID：外键关联数据分类表。 |
| datasource | varchar(20) | 否 |  | 数据来源：AUTO 自动采集 INPUT 人工录入 CALC 数据计算。 |
| datype | varchar(20) | 否 |  | 信号类型：State 状态信号 Digtal 数值信号。 |
| unit | varchar(20) | 是 | NULL | 单位 |
| numtail | bigint(20) | 是 | NULL | 显示小数位：只能填写阿拉伯数字：0/1/2/3/4。 |
| enumvalue | varchar(100) | 是 | NULL | 枚举值：用于定义状态信号每个状态值的中文含义。配置文本形如：0,关；1，开 |
| upper\_range | decimal(30,6) | 是 | NULL | 测点数值范围上限 |
| lower\_range | decimal(30,6) | 是 | NULL | 测点数值范围下限 |
| slope | decimal(30,6) | 是 | NULL | 数据预处理斜率 |
| cardinality | decimal(30,6) | 是 | NULL | 基数 |
| magnification | decimal(30,6) | 是 | NULL | 倍率 |
| remarks | varchar(4000) | 是 | *NULL* | 备注 |
| create\_time | datetime | 否 |  | 创建时间 |
| create\_userid(外键) | bigint(20) | 否 |  | 创建人，外键关联用户表 |
| update\_time | datetime | 是 | *NULL* | 更新时间 |
| update\_userid(外键) | bigint(20) | 是 | *NULL* | 更新人，外键关联用户表 |
| delete\_flag | tinyint(1) | 否 | *0* | 标记数据是否被删除，0为未删除，1为已删除 |

### 数据分类表（T\_DATA\_CATEGORY）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段** | **类型** | **空** | **默认** | **注释** |
| id (主键) | bigint(20) | 否 |  | 主键ID |
| category\_name | varchar(100) | 否 |  | 分类名称：水质/水量/能耗/物耗/设备参数/运行参数，系统内置，不提供正删改查。 |
| unit | varchar(4000) | 是 | NULL | 可选单位：在测点管理中，选择对应数据分类的时候，只能选择我们配置的单位供选择。存储结构形如：mg/L,g/L 单位间通过逗号隔开。 |
| create\_time | datetime | 否 |  | 创建时间 |
| create\_userid(外键) | bigint(20) | 否 |  | 创建人，外键关联用户表 |
| update\_time | datetime | 是 | *NULL* | 更新时间 |
| update\_userid(外键) | bigint(20) | 是 | *NULL* | 更新人，外键关联用户表 |

### 计算任务表（T\_CALC\_TASK）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段** | **类型** | **空** | **默认** | **注释** |
| id (主键) | bigint(20) | 否 |  | 主键ID |
| taskname | varchar(100) | 否 |  | 计算任务名称 |
| siteid | bigint(20) | 否 |  | 任务所属厂站 |
| mpointId | bigint(20) | 否 |  | 结果存放测点 |
| startDT | datetime | 否 |  | 任务开始时间 |
| endDT | datetime | 是 |  | 任务截止时间 |
| cycle | bigint(20) | 否 |  | 周期外键 |
| formulaId | bigint(20) | 否 |  | 计算公式外键 |
| status | varchar(100) | 否 |  | ON：启用 OFF：停用，默认值：ON |
| excute\_info | varchar(100) | 否 |  | Normal：正常 Error：异常 Waitting: 待执行，默认 |
| create\_time | datetime | 否 |  | 创建时间 |
| create\_userid(外键) | bigint(20) | 否 |  | 创建人，外键关联用户表 |
| update\_time | datetime | 是 | *NULL* | 更新时间 |
| update\_userid(外键) | bigint(20) | 是 | *NULL* | 更新人，外键关联用户表 |
| remarks | varchar(4000) | 是 | *NULL* | 备注 |
| ShiftsType | BIGINT | 是 |  | 当前计算任务关联的班次类型 |
| Calc\_end | Datetime | 是 |  | 当前计算任务当前计算周期结束时间，用于解决五分钟计算一次的问题 |

### 计算公式表（T\_CALC\_FORMULA）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段** | **类型** | **空** | **默认** | **注释** |
| id (主键) | bigint(20) | 否 |  | 主键ID |
| fname | varchar(100) | 是 | NULL | 算法名称 |
| exp\_info | varchar(200) | 是 | NULL | 公式说明 |
| expression | varchar(200) | 否 |  | 算法公式 |
| ftype | varchar(50) | 否 |  | Simple：Simple:简单型计算 Complex：复杂型计算 Custom：自定义型计算。 |
| siteId | bigint(20) | 是 |  | 自定义类型的厂站外键 |

### 公式变量表（T\_ FORMULA\_ VARIABLE）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段** | **类型** | **空** | **默认** | **注释** |
| id (主键) | bigint(20) | 否 |  | 主键ID |
| code | varchar(100) | 否 |  | 参数code |
| var\_type | varchar(50) | 否 |  | Constant为常量，Variable为变量 |
| var\_name | varchar(100) | 是 |  | 参数名称 |
| formulaId | bigint(20) | 否 |  | 参数关联参数 |

### 计算任务参数表（T\_CALC\_PARM）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段** | **类型** | **空** | **默认** | **注释** |
| id (主键) | bigint(20) | 否 |  | 主键ID |
| taskId | bigint(20) | 否 |  | 计算任务外键ID |
| code | varchar(100) | 否 |  | 公式参数code |
| defaultVal | decimal(30,6) | 是 |  | 默认值 |
| mpointId | bigint(20) | 是 |  | 关联测点Id |

### 任务日志表（T\_CALC\_LOG）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段** | **类型** | **空** | **默认** | **注释** |
| taskid | bigint(20) | 否 |  | 计算任务ID |
| dataDT | datetime | 否 |  | 任务计算的数据时间 |
| finishDT | Datetime | 否 |  | 任务完成当前的时间 |
| excute\_val | decimal(30,6) | 是 |  | 完成后的结果数据值 |
| status | varchar(50) | 否 |  | Normal：正常 Error：异常 Waitting: 待执行 Recalced 重新计算过 |
| error | varchar(4000) | 是 |  | 异常信息 |

### 任务补算表（T\_CALC\_ ROLLBACK）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段** | **类型** | **空** | **默认** | **注释** |
| taskid | bigint(20) | 否 |  | 计算任务ID |
| dataDT | datetime | 否 |  | 任务计算的数据时间 |
| finishDT | Datetime | 否 |  | 任务完成当前的时间 |
| excute\_val | decimal(30,6) | 是 |  | 完成后的结果数据值 |
| status | varchar(50) | 否 |  | Normal：正常 Error：异常 Waitting: 待执行 Recalced 重新计算过 |
| error | varchar(4000) | 是 |  | 异常信息 |

### 任务记录班次信息表(T\_CALC\_SHIFT)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段 | 类型 | 空 | 默认 | 注释 |
| dataDT | dateTime | 否 |  | 计算任务计算的班次开始时间或者是保存时间点 |
| mpointid | Bigint(20) | 否 |  | 计算任务存放结果的测点id |
| shiftstype | Varchar(100) | 否 |  | 计算任务关联班次类型名称 |
| Shift | Varcher(100) | 否 |  | 当前计算结果所在的班次名称 |
| endDT | datetime | 否 |  | 计算任务当前周期数据的开始时间 |
| startDT | datetime | 否 |  | 计算任务当前周期数据的结束时间 |

# 详细设计

## 任务调度

### 生成任务队列表

计算任务和参数的物理表结构关系如下（红色背景表示任务有依赖）：



将任务之间的关联抽象出来，首先抽离无子任务关联的任务，如下：



通过A,D获取依赖A、D的任务，如下：



由于在level1中由D推导出的C已经存在，则不放在队列之中。再通过**level1**中的B、C获取依赖于B、C的任务，如下：



则任务依赖推导完毕。将之转换成任务队列表，如下：



最后通过从高到低**level**的去重，保留**level**值大的任务，去掉**level**值小的任务

代码如下：

**后续再补**

去重结果如下：



### 任务递进遍历

**假设当前时间为2018-1-1 9:00:00**，首先根据level低到高的顺序遍历，遍历**level0**



然后将A和D分别往前计算一个周期，遍历**level1**



此时A、D已经计算完，但是由于B依赖于A，所以B计算的前提是A的计算时间到达2017-5-1，故无法计算。则B跳过，进入**level2**



此时已经跳过B,但是C依赖于B，所以C计算的前提条件是B计算时间到达2018-1和D计算到2018-1；E依赖于B。



由于D先于A计算完成，则只计算A至2017-5-1，此时开始计算B，然后继续遍历。



此时A、B都已计算完成，则开始计算C，E，C计算完成，本轮周期遍历结束。

核心代码：

private void taskDecomposition() throws Exception {  
List<String> sends = new ArrayList<String>();  
while (sends.size() == 0) {  
 for (int i = BaseConfig.*RELTABLE*.size() - 1; i > -1; i--) {  
 List<CalcTask> tasks = BaseConfig.*RELTABLE*.get(i);  
 for (CalcTask task : tasks) {  
 // 如果当前任务从未计算过  
 if (task.getDatadt() == null) {  
 CoreUtil.*arrangeStartDT*(task);  
 }  
 // 检查任务依赖  
 if (!checkDepend(task)) {  
 continue;  
 }  
  
 // 向前前进一轮周期  
 if (CoreUtil.*getNextTime*(task)) {  
 /\*  
 \* 递进情况1:当任务的计算时间小于预期完成时间，则任务正常计算 正常累计，按预期完成时间提交状态  
 \*/  
 if (task.getFinishdt().getTime() >= task.getCalcDate()  
 .getSave().getTime()) {  
 // 每次执行之后就会加一  
 task.setExecuteCount(task.getExecuteCount() + 1);  
 ComputingUnit unit = CoreUtil.*task2Unit*(task);  
 sends.add(CoreUtil.*object2JSONStr*(unit));  
 } else{  
 /\*  
 \* 递进情况2:由于某个新添加的任务，导致整个队列的完成时间延迟。  
 \* 进而导致了某个任务周期较小的周期开始了下一个周期。 此时应保持周期较小任务的低延时性  
 \* 并对预期完成时间向后推移，以及计算数量推移。 保证超出预期时间的计算每经历一次运算都会提交状态  
 \*/  
 task.setFinishdt(task.getCalcDate().getSave());  
 task.setExecuteSum(task.getExecuteSum() + 1);  
 task.setExecuteCount(task.getExecuteCount() + 1);  
 ComputingUnit unit = CoreUtil.*task2Unit*(task);  
 sends.add(CoreUtil.*object2JSONStr*(unit));  
 }  
 }  
 }  
 }  
 // 如果一轮遍历完，无一个计算单元则证明所有的计算任务都计算到当前时间了  
 if (sends.size() == 0) {  
 break;  
 } else {  
 sendCount += sends.size();  
 // 记录发布的数值  
 zookeeperClient.getZclient().setData().forPath(  
 BaseConfig.*NUM\_PATH*,  
 new String(sends.size() + "").getBytes());  
 pKafkaProducer.send(sends);  
 logger.debug("The current poll generates a total of {} calculation units.",sends.size());  
 BaseConfig.*SEND\_STATUS* = false;  
 while (!BaseConfig.*SEND\_STATUS*) {  
 logger.debug("The last polled computing unit was not consumed. This polling is waiting. . .");  
 // 当前一轮未被消费完时，当前轮一直处于等待状态  
 Thread.*sleep*(BaseConfig.*POLLED\_SLEEP*);  
 }  
 sends.clear();  
 }  
}  
logger.debug(  
 "This round of calculation is completed, a total of {} calculation units are generated",  
 sendCount);  
 }

### 生成补算队列

假设数据库中存在以下任务



则任务分解步骤为：首先分析对A的依赖任务，则为B、C、E，所以补算A应该同时补算B、C、E，则任务列表中存在记录：



分析对B的依赖任务，有C、E，所以补算B时应该同时补算C、E，则任务列表中存在记录：



由于黄色部分重复，故移除前面的两条记录（保证高level任务后置计算），则结果为：



分析C的任务依赖，只有C本身，故将C添加至列表中，但是由于C重复，故删除前面的一条记录，结果为：



分析D的任务依赖，有D、C，所以补算D时应该同时补算C，但是由于前面的C-C-2017重复，则删除前面的记录，结果为：



添加E的任务，结果为：

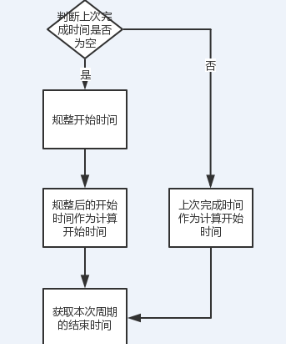


添加A的任务，删除重复记录结果为



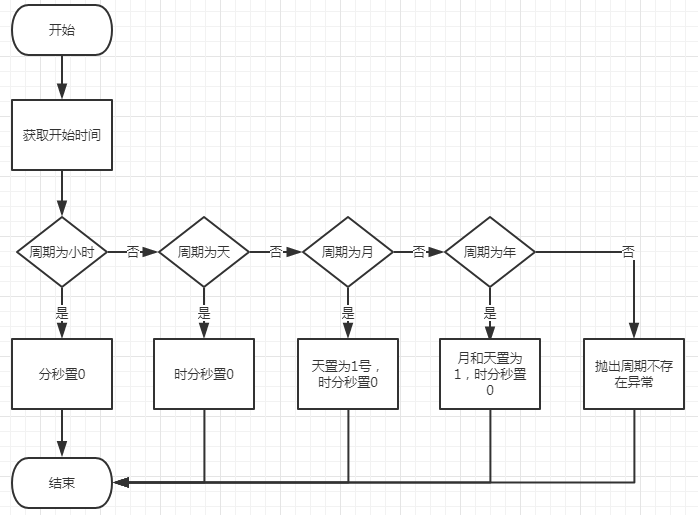
### 计算任务时间规整

在整个任务的逻辑中，计算任务规整是在任务开始计算前计算的，首先基于是否有上次完成时间来判断是否使用开始时间。



如果使用任务开始时间则在进行开始时间的的规整，保证周期的起始是在整点下进行的

具体任务规整的逻辑为（规整时间仅限于规整开始时间，此限制可以在计算任务创建的时候进行限制）



核心代码：

public static Date arrangeStartDT(Date start, Number cycle) {  
Calendar c = Calendar.*getInstance*();  
c.setTime(start);  
if (cycle.longValue() < Cycle.*DAY*.getCycle()) {  
 return new Date(start.getTime() / (cycle.longValue() \* 1000L)  
 \* cycle.longValue() \* 1000L);  
}  
if (cycle.longValue() == Cycle.*DAY*.getCycle()) {  
 *clearhour*(c);  
 return c.getTime();  
}  
// 时间是七天  
if (cycle.longValue() == 604800) {  
 if (c.get(Calendar.*DAY\_OF\_WEEK*) == 1) {  
 c.add(Calendar.*DAY\_OF\_MONTH*, -6);  
 *clearhour*(c);  
 return c.getTime();  
 } else {  
 c.add(Calendar.*DAY\_OF\_MONTH*,  
 -(c.get(Calendar.*DAY\_OF\_WEEK*) - 2));  
 *clearhour*(c);  
 return c.getTime();  
 }  
}  
// 时间是一个月  
if (cycle.longValue() == Cycle.*MONTH*.getCycle()) {  
 c.set(Calendar.*DAY\_OF\_MONTH*, 1);  
 *clearhour*(c);  
 return c.getTime();  
}  
// 时间是一个季度  
if (cycle.longValue() == 7776000L) {  
 if (c.get(Calendar.*MONTH*) <= 2) {  
 c.set(Calendar.*MONTH*, 0);  
 c.set(Calendar.*DAY\_OF\_MONTH*, 1);  
 *clearhour*(c);  
 }  
 if (c.get(Calendar.*MONTH*) > 2 && c.get(Calendar.*MONTH*) <= 5) {  
 c.set(Calendar.*MONTH*, 3);  
 c.set(Calendar.*DAY\_OF\_MONTH*, 1);  
 *clearhour*(c);  
 }  
 if (c.get(Calendar.*MONTH*) > 5 && c.get(Calendar.*MONTH*) <= 8) {  
 c.set(Calendar.*MONTH*, 6);  
 c.set(Calendar.*DAY\_OF\_MONTH*, 1);  
 *clearhour*(c);  
 }  
 if (c.get(Calendar.*MONTH*) > 8 && c.get(Calendar.*MONTH*) <= 11) {  
 c.set(Calendar.*MONTH*, 9);  
 c.set(Calendar.*DAY\_OF\_MONTH*, 1);  
 *clearhour*(c);  
 }  
 return c.getTime();  
}  
// 时间是一年  
if (cycle.longValue() == Cycle.*YEAR*.getCycle()) {  
 c.set(Calendar.*MONTH*, 0);  
 c.set(Calendar.*DAY\_OF\_MONTH*, 1);  
 *clearhour*(c);  
 return c.getTime();  
}  
return start;  
 }

### 计算任务分解

每一个计算任务存储在Kafka里面的格式为：

public class ComputingUnit {  
 private Long taskid;  
 private Long cycle;  
 private Date startDT;  
 private Date endDT;  
 private Date saveDT;  
 private Long mpointId;  
 private String expression;  
 private List<CalcParm> calcParms;

}

public class CalcParm {  
 private Long id;  
  
 private Long taskid;  
  
 private String code;  
  
 private BigDecimal defaultval;  
  
 private Long mpointid;  
   
 private String datasource;  
   
 private String point;  
  
 private String mpointName;

}

### 补算任务分解

每一个补算任务存储在Kafka里面的格式为：

1. {
2. "taskid": 111,
3. "cycle": 86400,
4. "datadt": "2017-4-1",
5. "expression": "Sum",
6. "type": "roll",
7. "parm": [
8. {
9. "code": "a",
10. "mpoint": 1
11. },
12. {
13. "code": "b",
14. "defaultValue": 0
15. }
16. ]
17. }

## 数据服务

### Web接口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **测点列表功能** | **方法** | **接口** | **参数** | **返回值** |
| 查询计算任务列表 | GET | api/calcTask/tasks | site=1,2&status=ON&execute=Normal&query=123&page=1&size=10 | {items:[{“calcno”:”000007”“id””:123,“taskName”:‘’asd”,“cycle”：“月”，“formula”：“差值”，”startDT”:”2017-8-7”,”status”:”ON”,”excuteInfo”:”Normal”,”type”:”Simple”}]} |
| 根据计算任务获取计算记录 | GET | api/calcTask/calclog/task | startDT=2017-7-1&endDT=2017-8-1&execute=Normal | {items:[{“dataDT”:”2017-8-1 07:00:00”,” finishDT”: ”2017-8-1 08:21:45”,”excuteVal”:12,”status”:” Normal”, “error”:”” }]，ntotal:20,etotal:0} |
| 添加补算任务 | POST | api/calcTask/rollback | { taskid :1223，status “Waitting”，dataDT:”2017-8-1”} | {items:”success”} |
| 删除计算任务 | DELETE | [api/mpoint/mpoints](http://dev3.haocang.com:8080/swagger-ui/index.html#!/27979288573164929702/deleteMpointUsingDELETE_2) | Ids=1,2,3 | {items:”success”} |
| 更改计算任务状态 | PUT | api/calcTask/tasks | Ids=1,2,3&status=ON | {items:”success”} |

说明：错误信息使用与mango项目中相同，在hader中以”x-error”:”XXXXException”形式返回

## 缓存设计

目前体量可不引入缓存，待续使用时在进行讨论。

## 结果收集

for (int i = BaseConfig.*RELTABLE*.size() - 1; i > -1; i--) {  
List<CalcTask> tasks = BaseConfig.*RELTABLE*.get(i);  
for (CalcTask task : tasks) {  
 if (task.getExecuteSum() != 0  
 && task.getExecuteCount() == task.getExecuteSum()) {  
 if(task.getId() == 5638){  
 System.*out*.println();  
 }  
 long count = calcLogService.getExecuteCountByTask(  
 task.getId(), task.getStart(),  
 task.getFinishdt());  
 if (count == task.getExecuteSum()) {  
 calcTaskService.updateStatus(task.getId(),  
 task.getFinishdt());  
 logger.info("Submit the task with id {} and modify the completion time to {}",task.getId(),task.getFinishdt());  
 }  
 }  
}  
 }

## 任务队列

采用Kafka实现，为了保证父子任务依赖关系，从低等级任务开始，先一层一层的将低等级任务到当前时间全部发送到kafka中，等待task模块计算完成之后在发送更高等级计算任务，只到遍历完全部等级的计算任务。

## 消费监控

采用Zookeeper实现，核心代码：

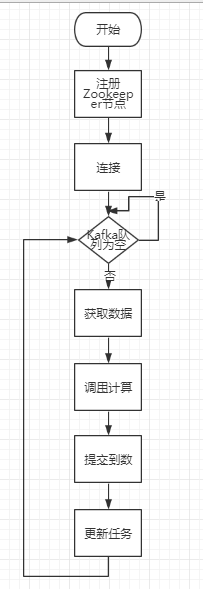
ZookeeperClient zclient = factory.getBean("zkClient",  
 ZookeeperClient.class);  
 CuratorFramework client = zclient.getZclient();  
 InterProcessSemaphoreMutex lock = new InterProcessSemaphoreMutex(  
 client, BaseConfig.*NUM\_PATH*);  
 if (lock.acquire(10, TimeUnit.*SECONDS*)) {  
String data = new String(client.getData().forPath(  
 BaseConfig.*NUM\_PATH*));  
String wdata = new String(new Long(data).longValue()  
 - 1 + "");  
client.setData().forPath(BaseConfig.*NUM\_PATH*,  
 wdata.getBytes());  
lock.release();  
 }

## 节点监控

采用Zookeeper实现，核心代码

final NodeCache watcher = new NodeCache(zclient, BaseConfig.*NUM\_PATH*);  
watcher.getListenable().addListener(new NodeCacheListener() {  
 @Override  
 public void nodeChanged() throws Exception {  
 if (watcher.getCurrentData() != null && watcher.getCurrentData().getData() != null) {  
 String data = new String(watcher.getCurrentData().getData());  
 if(new Long(data) == 0){  
 BaseConfig.*SEND\_STATUS* = true;  
 }  
 }  
 }  
});  
watcher.start();

## 计算节点



## 计算算法

### COD消减量

**COD削减量 = （进水COD-出水COD）× 进水流量**,其中进、出水浓度差的单位是mg/L，折合g/m3；进水流量单位是m3/h。

这样计算出的削减量单位是：g/h，即每小时的污染物削减量。

对每小时的污染物削减量进行累加，可得到每天的污染物削减量：

以此类推可计算出每月、每年的污染物削减量。

按上述方法计算的削减量单位是g，可除以1000折算成kg，或除以106折算成t。

如果算每天以及以上

**周期污染物削减量 = （周期均进水污染物浓度-周期均出水污染物浓度）× 周期累积处理水量**

### 氨氮消减量

**NH3-N削减量 = （进水NH3-N-出水NH3-N）× 进水流量**

**具体计算方式于COD消减量相同**

### 污染物综合消减量

**污染物综合削减量 = 0.6×COD削减量+0.4×NH3-N削减量，**用于统计综合污染物消减量

### COD合格率

现在默认检查频率为两小时，例周期为一天时，没两小时检查一次，即检查12次。然后根据上述公式计算合格率，保留四位小时即展示格式为XX.XX%，存储格式为0.XXXX

**COD合格率的范围最大值50**

### 氨氮合格率

计算方法类似COD合格率

**氨氮指标最大值5**

### PH合格率

计算方法类似COD合格率

**PH合格范围是6-9**

### BOD合格率

计算方法类似COD合格率

**BOD指标最大值10**

### SS合格率

计算方法类似COD合格率

**SS指标最大值10**

### TN合格率

计算方法类似COD合格率

**TN指标最大值15**

### TP合格率

计算方法类似COD合格率

**TP指最大值0.5**

### 水力负荷率

累积处理水量和设计处理水量的比值。

水力负荷率可按天、月、年为时间单位进行计算。

其中，设计处理水量 = 设计处理流量 × 时间单位（天，月，年）

### 小时差值

适用于**累计量测点**计算一个小时的增长量

### 去年同期同比

当前值与去年同一周期比较，计算公式为：**同比=(当前值 – 去年值)/去年值**

保留四位小时即展示格式为XX.XX%，存储格式为0.XXXX

*备注：该算法之前是为葛洲坝项目定制的算法，使用率低，重要度较低，暂不实现*

### 上月环比

当前值与上一个周期比较，计算公式为：**环比=(当前值 – 上周期值)/ 上周期值**

保留四位小时即展示格式为XX.XX%，存储格式为0.XXXX

*备注：该算法之前是为葛洲坝项目定制的算法，使用率低，重要度较低，暂不实现*

### 于去年平均值对比增减

当前值与去年平均值比较，计算公式为：**对比=(当前值 – 上周期值)/ 上周期值**

保留四位小时即展示格式为XX.XX%，存储格式为0.XXXX

*备注：该算法之前是为葛洲坝项目定制的算法，使用率低，重要度较低，暂不实现*

### 四则运算

四则运算主要支持用户自定义配置四则公式，并绑定公式参数对应的测点，支持加、减、乘、除、括号（英文括号）运算方式来编写公式并制定参数对应测点及周期，例如A+B需指定A和B对应的测点并制定周期

### 求和

将周期范围内的数据进行相加求和，例如：2018年4月1日一天的进水流量数据如下：

|  |  |
| --- | --- |
| 时间 | 值 |
| 2018-04-01 00:00:00 | 2401 |
| 2018-04-01 01:00:00 | 2315 |
| 2018-04-01 02:00:00 | 2364 |
| 2018-04-01 03:00:00 | 2932 |
| 2018-04-01 04:00:00 | 2138 |
| 2018-04-01 05:00:00 | 2613 |
| 2018-04-01 06:00:00 | 2098 |
| 2018-04-01 07:00:00 | 2693 |
| 2018-04-01 08:00:00 | 2611 |
| 2018-04-01 09:00:00 | 2167 |
| 2018-04-01 10:00:00 | 2426 |
| 2018-04-01 11:00:00 | 2890 |
| 2018-04-01 12:00:00 | 2648 |
| 2018-04-01 13:00:00 | 2699 |
| 2018-04-01 14:00:00 | 2471 |
| 2018-04-01 15:00:00 | 2149 |
| 2018-04-01 16:00:00 | 2235 |
| 2018-04-01 17:00:00 | 2078 |
| 2018-04-01 18:00:00 | 2479 |
| 2018-04-01 19:00:00 | 2356 |
| 2018-04-01 20:00:00 | 2585 |
| 2018-04-01 21:00:00 | 2244 |
| 2018-04-01 22:00:00 | 2753 |
| 2018-04-01 23:00:00 | 2506 |

将上述表格中的数据进行相加求和，得到58851的计算结果为2018年4月1日一天的进水累计流量。

### 平均值

将周期范围内的数据进行求平均值，例如：2018年4月1日一天的出水COD数据如下：

|  |  |
| --- | --- |
| 时间 | 值 |
| 2018-04-01 00:00:00 | 30.34 |
| 2018-04-01 02:00:00 | 35.64 |
| 2018-04-01 04:00:00 | 34.20 |
| 2018-04-01 06:00:00 | 35.51 |
| 2018-04-01 08:00:00 | 36.56 |
| 2018-04-01 10:00:00 | 34.64 |
| 2018-04-01 12:00:00 | 37.65 |
| 2018-04-01 14:00:00 | 37.39 |
| 2018-04-01 16:00:00 | 36.20 |
| 2018-04-01 18:00:00 | 30.19 |
| 2018-04-01 20:00:00 | 37.20 |
| 2018-04-01 22:00:00 | 30.78 |

将上述表格中的数据进行求平均值，得到34.69的计算结果，2018年4月1日一天的出水COD。

### 最大值

将周期范围内的数据进行求最大值，例如：2018年4月1日一天的出水COD数据如下：

|  |  |
| --- | --- |
| 时间 | 值 |
| 2018-04-01 00:00:00 | 30.34 |
| 2018-04-01 02:00:00 | 35.64 |
| 2018-04-01 04:00:00 | 34.20 |
| 2018-04-01 06:00:00 | 35.51 |
| 2018-04-01 08:00:00 | 36.56 |
| 2018-04-01 10:00:00 | 34.64 |
| 2018-04-01 12:00:00 | 37.65 |
| 2018-04-01 14:00:00 | 37.39 |
| 2018-04-01 16:00:00 | 36.20 |
| 2018-04-01 18:00:00 | 30.19 |
| 2018-04-01 20:00:00 | 37.20 |
| 2018-04-01 22:00:00 | 30.78 |

将上述表格中的数据进行求最大值，得到37.65的计算结果，2018年4月1日一天的最大的出水COD值。

### 最小值

将周期范围内的数据进行求最小值，例如：2018年4月1日一天的出水COD数据如下：

|  |  |
| --- | --- |
| 时间 | 值 |
| 2018-04-01 00:00:00 | 30.34 |
| 2018-04-01 02:00:00 | 35.64 |
| 2018-04-01 04:00:00 | 34.20 |
| 2018-04-01 06:00:00 | 35.51 |
| 2018-04-01 08:00:00 | 36.56 |
| 2018-04-01 10:00:00 | 34.64 |
| 2018-04-01 12:00:00 | 37.65 |
| 2018-04-01 14:00:00 | 37.39 |
| 2018-04-01 16:00:00 | 36.20 |
| 2018-04-01 18:00:00 | 30.19 |
| 2018-04-01 20:00:00 | 37.20 |
| 2018-04-01 22:00:00 | 30.78 |

将上述表格中的数据进行求最小值，得到30.19的计算结果，2018年4月1日一天的最小的出水COD值。

### 取前点

将距周期范围的开始时间最近的一个值作为计算结果，例如：计算周期为天，需要计算2018年4月1日的取前点值，现有数据如下：

|  |  |
| --- | --- |
| 时间 | 值 |
| 2018-03-31 23:59:00 | 35.64 |
| 2018-04-01 00:00:00 | 30.34 |
| 2018-04-01 00:01:00 | 35.64 |
| 2018-04-01 00:02:00 | 34.20 |
| 2018-04-01 00:03:00 | 35.51 |
| 2018-04-01 00:04:00 | 36.56 |
| 2018-04-01 00:05:00 | 34.64 |
| ...... | …… |
| 2018-04-01 23:55:00 | 37.39 |
| 2018-04-01 23:56:00 | 36.20 |
| 2018-04-01 23:57:00 | 30.19 |
| 2018-04-01 23:58:00 | 37.20 |
| 2018-04-01 23:59:00 | 30.78 |
| 2018-04-02 00:00:00 | 31.23 |

将上述表格中的数据进行取前点，得到结果为获取2018年4月1日前时间点最新的记录2018-03-31 23:59:00的35.64。

### 差值

将周期范围内数据的增长量作为计算结果，例如：计算周期为天，需要计算2018年4月1日的取前点值，现有数据如下：

|  |  |
| --- | --- |
| 时间 | 值 |
| 2018-03-31 23:59:00 | 8233192.214 |
| 2018-04-01 00:00:00 | 8233231.154 |
| 2018-04-01 00:01:00 | 8233312.683 |
| 2018-04-01 00:02:00 | 8233412.142 |
| 2018-04-01 00:03:00 | 8233547.142 |
| 2018-04-01 00:04:00 | 8233597.543 |
| 2018-04-01 00:05:00 | 8233663.632 |
| ...... | …… |
| 2018-04-01 23:55:00 | 8235863.632 |
| 2018-04-01 23:56:00 | 8235921.364 |
| 2018-04-01 23:57:00 | 8236044.643 |
| 2018-04-01 23:58:00 | 8236127.216 |
| 2018-04-01 23:59:00 | 8236208.117 |
| 2018-04-02 00:00:00 | 8236311.632 |

将上述表格中的数据进行差值，根据算法先取出周期内最新的时间点2018-04-01 23:59:00的值8236208.117，再取出离周期范围前面最近的时间点2018-03-31 23:59:00的值8233192.214。将两个值相减得到3015.903。

### 开机次数

根据设备的开停状态测点，统计设备在周期范围内的开机次数，例如：计算周期为1天的2018年4月1日的设备开机次数，现有数据如下：

|  |  |
| --- | --- |
| 时间 | 值 |
| 2018-03-31 23:59:00 | 0 |
| 2018-04-01 00:00:00 | 0 |
| 2018-04-01 00:01:00 | 0 |
| 2018-04-01 00:02:00 | 1 |
| 2018-04-01 00:03:00 | 1 |
| 2018-04-01 00:04:00 | 1 |
| 2018-04-01 00:05:00 | 1 |
| 2018-04-01 00:06:00~2018-04-01 23:54:00 | 1 |
| 2018-04-01 23:55:00 | 1 |
| 2018-04-01 23:56:00 | 0 |
| 2018-04-01 23:57:00 | 0 |
| 2018-04-01 23:58:00 | 0 |
| 2018-04-01 23:59:00 | 0 |
| 2018-04-02 00:00:00 | 0 |

将上述表格中的数据进行计算开机次数，获取2018年4月1日一天内的所有数据，进行轮询，对于数据由0变成1的情况，计开机次数一次，2018-04-01 00:01:00的值为0，2018-04-01 00:02:00的值为1，计开机次数一次。最终轮询完成所有数据，得到结果1。

### 停机次数

根据设备的开停状态测点，统计设备在周期范围内的停机次数，例如：计算周期为1天的2018年4月1日的设备停机次数，现有数据如下：

|  |  |
| --- | --- |
| 时间 | 值 |
| 2018-03-31 23:59:00 | 0 |
| 2018-04-01 00:00:00 | 0 |
| 2018-04-01 00:01:00 | 0 |
| 2018-04-01 00:02:00 | 1 |
| 2018-04-01 00:03:00 | 1 |
| 2018-04-01 00:04:00 | 1 |
| 2018-04-01 00:05:00 | 1 |
| 2018-04-01 00:06:00~2018-04-01 23:54:00 | 1 |
| 2018-04-01 23:55:00 | 1 |
| 2018-04-01 23:56:00 | 0 |
| 2018-04-01 23:57:00 | 0 |
| 2018-04-01 23:58:00 | 0 |
| 2018-04-01 23:59:00 | 0 |
| 2018-04-02 00:00:00 | 0 |

将上述表格中的数据进行计算停机次数，获取2018年4月1日一天内的所有数据，进行轮询，对于数据由1变成0的情况，计停机次数一次，2018-04-01 23:55:00的值为1，2018-04-01 23:56:00的值为0，计停机次数一次。最终轮询完成所有数据，得到结果1。

### 运行时间

根据设备的开停状态测点，统计设备在周期内的运行时间，例如：计算周期为1天的2018年4月1日的设备运行时间，现有数据如下：

|  |  |
| --- | --- |
| 时间 | 值 |
| 2018-03-31 23:59:00 | 0 |
| 2018-04-01 00:00:00 | 0 |
| 2018-04-01 00:01:00 | 0 |
| 2018-04-01 00:02:00 | 1 |
| 2018-04-01 00:03:00 | 1 |
| 2018-04-01 00:04:00 | 1 |
| 2018-04-01 00:05:00 | 1 |
| 2018-04-01 00:06:00~2018-04-01 23:54:00 | 1 |
| 2018-04-01 23:55:00 | 1 |
| 2018-04-01 23:56:00 | 0 |
| 2018-04-01 23:57:00 | 0 |
| 2018-04-01 23:58:00 | 0 |
| 2018-04-01 23:59:00 | 0 |
| 2018-04-02 00:00:00 | 0 |

将上述表格中的数据进行计算运行时间，获取2018年4月1日一天内的所有数据，进行轮询，把周期范围内值为1的时间进行累计，即：2018-04-01 23:55:00与2018-04-01 00:02:00的时间差54分钟。

### 停机时间

根据设备的开停状态测点，统计设备在周期内的停机时间，例如：计算周期为1天的2018年4月1日的设备运行时间，现有数据如下：

|  |  |
| --- | --- |
| 时间 | 值 |
| 2018-03-31 23:59:00 | 0 |
| 2018-04-01 00:00:00 | 0 |
| 2018-04-01 00:01:00 | 0 |
| 2018-04-01 00:02:00 | 1 |
| 2018-04-01 00:03:00 | 1 |
| 2018-04-01 00:04:00 | 1 |
| 2018-04-01 00:05:00 | 1 |
| 2018-04-01 00:06:00~2018-04-01 23:54:00 | 1 |
| 2018-04-01 23:55:00 | 1 |
| 2018-04-01 23:56:00 | 0 |
| 2018-04-01 23:57:00 | 0 |
| 2018-04-01 23:58:00 | 0 |
| 2018-04-01 23:59:00 | 0 |
| 2018-04-02 00:00:00 | 0 |

将上述表格中的数据进行计算运行时间，获取2018年4月1日一天内的所有数据，进行轮询，把周期范围内值为0的时间进行累计，即：2018-04-01 00:00:00与2018-04-01 00:01:00的时间差2分钟加上2018-04-01 23:56:00与2018-04-01 23:59:00的时间差4分钟共6分钟。

### 累加计算

|  |  |
| --- | --- |
| 2020-08-17 21:46:05 | 1158.523 |
| 2020-08-17 21:47:05 | 1158.523 |
| 2020-08-17 21:48:05 | 1158.523 |
| 2020-08-17 21:49:05 | 1158.523 |
| 2020-08-17 21:50:06 | 1158.523 |
| 2020-08-17 21:51:06 | 1158.523 |
| 2020-08-17 21:52:06 | 1158.523 |
| 2020-08-17 21:53:06 | 1158.523 |
| 2020-08-17 21:54:06 | 1158.523 |
| 2020-08-17 21:55:06 | 1160.746 |
| 2020-08-17 21:56:06 | 1160.746 |
| 2020-08-17 21:57:06 | 1160.746 |
| 2020-08-17 21:58:06 | 1160.746 |
| 2020-08-17 21:59:06 | 1160.746 |
| 2020-08-17 22:00:06 | 1160.746 |
| 2020-08-17 22:01:06 | 1160.746 |
| 2020-08-17 22:02:06 | 1160.746 |
| 2020-08-17 22:03:06 | 1160.746 |
| 2020-08-17 22:04:06 | 1160.746 |
| 2020-08-17 22:05:06 | 1160.746 |
| 2020-08-17 22:06:06 | 1160.746 |
| 2020-08-17 22:07:06 | 1160.746 |
| 2020-08-17 22:08:06 | 1160.746 |

如上数据，使用后一条数据减去前一条数据，然后进行累加，第一条数据不用减前一条，后一条数据不用被减

计算方式逻辑:

* 如果当前是重新计算或者班次类型为周期的计算，取时间段内的所有数据，如果当前时间段内没有数据，就直接返回异常结果
* 如果当前是正常计算，而且是第一次计算，获取当前时间段内的所有数据，如果当前计算周期是小时，就直接返回异常结果，如果当前周期是非小时的，由于是第一次进行计算，显示没有数据
* 如果当前是正常计算并且已经计算过了
  + 获取当前上一次计算的值
    - 上一次计算没有值，并且时间段内没有数据，认为当前数据异常无数据
    - （一定是非小时周期计算任务）上次计算有值，直接继承当前数据作为计算数据
  + 获取前点数据
    - 如果取值时间段(天，周，月)没有数值，返回null，不计算前点
    - 如果取值时间段存在数值，并且当前时间段结束点存在数值，返回null，避免一个数据被多次相减(其实也没有太大的关系)
    - 如果当前取值没有时间段
  + 如果当前测点时间段内只有一条数据，不进行计算并返回只有一条数据的异常

## 关于班次的计算任务

### 开始时间的计算

开始时间计算标记时间选择:

1. 如果当前时间没有计算过(既没有DataDT数据)，通过开始时间startDT计算当前周期的开始时间
2. 如果当前时间已经计算过(既存在DataDt 数据)，通过dataDT计算当前周期的开始时间

周期开始时间计算策略:

a.计算使用班次类型版本

1) 通过遍历计算任务之后，调用接口获取所有有效的班次类型版本以及对于的班次列表

2) 针对一个计算任务所关联的班次类型，将所有的版本按照时间顺序排序生成列表

3) 如果标记时间(如果dataDT不存在就选用startDT)比列表中的第一个时间还小，计算任务会从列表中第一个班次类型版本的更新时间开始计算，在计算任务开始时间到第一个班次类型版本的更新时间端之间不会进行计算。

4) 如果标记时间比列表中最后一个时间还大，选用最后一个班次类型版本的班次列表作为基础计算开始时间

b. 通过班次列表计算开始时间

1) 根据当前班次列表的开始时间进行排序

2) 如果标记时间小于班次列表的第一个班次的开始时间，开始时间就是第一个班次的结束时间后的半个小时

3) 如果标记时间大于班次列表的最后一个班次的开始时间，开始时间就是第二天的第一个班次的结束时间后的半个小时

4) 如果表示时间大于第i个班次的开始时间，小于等于第i+1的班次的开始时间，当前计算任务的开始时间就是第i+1班次的结束时间后的半个小时。

|  |  |  |
| --- | --- | --- |
| 班次名称 | 时间段 | 结束时间后半个小时 |
| 中班 | 3:00 -> 7:00 | 7:30 |
| 晚班 | 8:00 -> 12:00 | 12:30 |
| 白班 | 12:00->2:00 | 2：30 |

如上图(跨天，已排序)：

* 当前标记时间 0:00，开始时间是 7:30;
* 当前标记时间 3:00，开始时间是 7:30;
* 当前标记时间 8:30，开始时间是 12:30;
* 当前标记时间是 12:00, 开始时间是 第二天2:30;

### 关于班次类型的计算任务中数据库以及实体类中时间解释

数据库中的事件字段：

表t\_calc\_task:

1. startDT：当前计算任务的开始时间，通过页面设置
2. dataDT：当前已经计算的最后一个班次的结束时间
3. finishDT：当前计算一轮修改数据库t\_calc\_task表的时间

表t\_calc-log:

1. dataDT：当前班次的开始事件
2. finishDT：当前周期计算完成的录入数据库事件

实体类中的数据：

CalcTask:

1. startDT：数据表t\_calc\_task中的startDT；
2. start ：当前周期的开始计算时间；

CalcDate:

1. start： 当前班次的开始时间
2. end ：当前班次的结束时间
3. save：当前班次的开始时间，也是当前记录存储的时间点