集算器

创新大数据计算引擎

工业传感器标准化采集案例

润乾软件出品

项目背景

工业设备生成的数据，通常由实时数据库进行初级采集。但实时数据过于原始，无法被信息管理系统、商业智能、数据挖掘等应用直接使用，一般需要进行标准化二次采集，形成规范一致的结构化数据服务于上层应用，及时了解机组健康状况，以此降低维护成本，提高经济效益！

这里会有问题吗？

采集

实时库

麦杰

ETL

C# / Java

关系库

Mysql

查询

应用层

异常预测 | 停机分析 | 导出/打印

数据处理前后的字段含义说明

原始实时数据存储在麦杰数据库中，麦杰库没有JDBC接口，只有符合工业标准的OPC接口（OLE for Process Control）。通过该接口，C#/Java程序可获取某时间段内某风力发电机每秒（或每毫秒）的起停状态。数据处理前后的格式如下：

原始实时数据

```
DynamicData: time:2017-07-05 15:06:00.000,AV:1.0,AS:0
DynamicData: time:2017-07-05 15:35:00.000,AV:1.0,AS:0
DynamicData: time:2017-07-05 15:38:00.000,AV:4.0,AS:0
DynamicData: time:2017-07-05 15:44:00.000,AV:2.0,AS:0
DynamicData: time:2017-07-05 15:44:01.000,AV:2.0,AS:-32768
DynamicData: time:2017-07-05 15:53:00.000,AV:1.0,AS:0
DynamicData: time:2017-07-05 15:54:00.000,AV:1.0,AS:0
DynamicData: time:2017-07-05 15:59:00.000,AV:5.0,AS:0
DynamicData: time:2017-07-05 15:59:01.000,AV:5.0,AS:-32768
DynamicData: time:2017-07-05 15:59:02.000,AV:1.0,AS:0
…
```

代表时间点和该时间点的风机状态

AV等于1.0表示开机，等于其他值表示各种原因导致的停机

停机开始时刻（对应原始数据中的time字段）

停机结束时刻（即风机起动时刻，对应原始数据中的time字段）

停起间隔秒数（endtime - starttime）

停机状态（对应原始数据中的AV）

记录编号（自增型）

标准化数据

starttime	endtime	distance	alarmno	id	…
2017-06-27 19:00:00	2017-06-27 19:00:00	0	-1	653	…
2017-07-05 16:03:00	2017-07-05 16:02:00	60	2	654	…
2017-07-05 18:06:00	2017-07-05 18:03:00	180	4	655	…
2017-07-05 18:21:00	2017-07-05 18:20:00	60	5	656	…
2017-07-05 18:39:00	2017-07-05 18:32:00	420	10	657	…
…	…	…	…		…

数据处理前后的对应关系举例

1、连续的多个（至少一个）停机数据，简称为停机区；连续的多个（至少1个）起机数据，简称为起机区。原始数据一定是两区交替出现的形式，如图1白色为起机区，红色为停机区。

```
DynamicData: time:2017-07-05 15:06:00.000,AV:1.0,AS:0
DynamicData: time:2017-07-05 15:35:00.000,AV:1.0,AS:0
DynamicData: time:2017-07-05 15:38:00.000,AV:4.0,AS:0
DynamicData: time:2017-07-05 15:44:00.000,AV:2.0,AS:0
DynamicData: time:2017-07-05 15:44:01.000,AV:2.0,AS:-32768
DynamicData: time:2017-07-05 15:53:00.000,AV:1.0,AS:0
DynamicData: time:2017-07-05 15:54:00.000,AV:1.0,AS:0
DynamicData: time:2017-07-05 15:59:00.000,AV:5.0,AS:0
DynamicData: time:2017-07-05 15:59:01.000,AV:5.0,AS:-32768
DynamicData: time:2017-07-05 15:59:02.000,AV:1.0,AS:0
…
```

①

2、停机区和紧挨着的下一个起机区，合称为停起区。如图2框中就是停起区。

```
DynamicData: time:2017-07-05 15:06:00.000,AV:1.0,AS:0
DynamicData: time:2017-07-05 15:35:00.000,AV:1.0,AS:0

DynamicData: time:2017-07-05 15:38:00.000,AV:4.0,AS:0
DynamicData: time:2017-07-05 15:44:00.000,AV:2.0,AS:0
DynamicData: time:2017-07-05 15:44:01.000,AV:2.0,AS:-32768
DynamicData: time:2017-07-05 15:53:00.000,AV:1.0,AS:0
DynamicData: time:2017-07-05 15:54:00.000,AV:1.0,AS:0

DynamicData: time:2017-07-05 15:59:00.000,AV:5.0,AS:0
DynamicData: time:2017-07-05 15:59:01.000,AV:5.0,AS:-32768
DynamicData: time:2017-07-05 15:59:02.000,AV:1.0,AS:0
…
```

②

3、显然，停起区对应一条标准化数据，其中停机开始时刻为停机区的第一条，停机结束时刻（起动时刻）为起机区的第一条，这两条数据称为停起对。如图3打*号开头（粉色）所示：

```
DynamicData: time:2017-07-05 15:06:00.000,AV:1.0,AS:0
DynamicData: time:2017-07-05 15:35:00.000,AV:1.0,AS:0

*DynamicData: time:2017-07-05 15:38:00.000,AV:4.0,AS:0
DynamicData: time:2017-07-05 15:44:00.000,AV:2.0,AS:0
DynamicData: time:2017-07-05 15:44:01.000,AV:2.0,AS:-32768
*DynamicData: time:2017-07-05 15:53:00.000,AV:1.0,AS:0
DynamicData: time:2017-07-05 15:54:00.000,AV:1.0,AS:0

*DynamicData: time:2017-07-05 15:59:00.000,AV:5.0,AS:0
DynamicData: time:2017-07-05 15:59:01.000,AV:5.0,AS:-32768
*DynamicData: time:2017-07-05 15:59:02.000,AV:1.0,AS:0
…
```

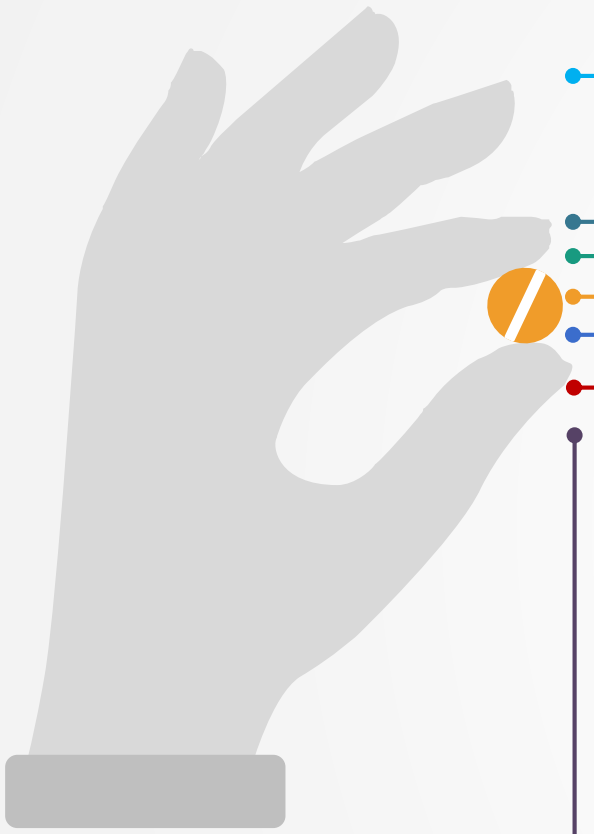
③

4、所以从原始数据中，只对应2条标准化数据（停起对），如图4

④

starttime	endtime	distance	alarmno
2017-07-05 15:38:00	2017-07-05 15:53:00	900	4.0
2017-07-05 15:59:00	2017-07-05 15:59:02	2	5.0
…	…	…	…

数据标准化过程



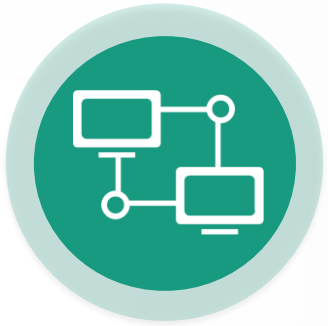
1. 从实时库取出字符串，转为结构化二维表，以便进行后续计算。这个算法可称为结构化。
2. 若关系型数据库为空，则新增一条初始数据，时间为8天前。
3. 当第一条数据为停机时，如果库中最新记录为半条（只有停机说明数据错误），则比对数据后更新库，如果库中最新记录为整条（说明数据正常），则追加数据。
4. 当第一条数据为起机时，需要追溯上一个小时或N个小时，直到找到对应的停机时刻，形成完整的停起对。这个算法称为追溯
5. 追溯得到的停起对，需要和数据库最新记录比较，如果是漏写的数据，则追加这条数据。本算法称为补漏。
6. 不能无限制追溯，如果追溯7天仍然找不到停机时刻，则将7天前作为停机时刻。并计算出所有的停起对。
7. 写库时，若记录以起为结尾，说明最后一个停起对是完整的，但若以停结尾，则说明最后一个停起对不完整（风机停了但一直没启动），此时需要向数据库插入半条记录（只有停），待下个小时采集时，再补上此记录的起机时刻
8. 当“两次停机间隔小于10分钟，则合并为一次停机”，即：把上一条记录的停机时刻作为合并后的停机时刻，把下一条记录的起机时刻作为合并后的起机时刻。本算法会将多条记录合并为少量记录，可称为“拉链”算法。

原算法存在的不足

由于实时库的特殊性，导致其标准化采集算法异常复杂，以前只能用C#/Java等高级语言实现，不仅设计周期长，且算法难以实现，难以维护，执行效率也不尽人意！

接口特殊

只提供API接口，访问很不方便，计算起来更是困难重重，无SQL可用，只能用JAVA/C#从底层实现，代码量极大，性能也难以保障。



算法复杂

其中追溯（算法4）、停起对（算法6）、拉链（算法8）都是很难实现的算法。以计算停起对为例，C#要195行代码，涉及大量嵌套循环以及难以理解的临时变量和条件关系。

频繁写库

拉链算法会频繁操作数据库，每次还要计算停起间隔，其他算法也会反复查询数据库，这对数据库造成巨大压力，稳定性变差。算法中之所以易发生漏写数据，就是因为频繁写库造成的。

易产生隐患

用C#实现追溯算法，只能用两个嵌套循环分别实现，但向上循环过于抽象复杂，导致无法算出起机时间，只能用第一条（一定是起机时间）代替。如果进行整月整年的统计，就会出现较大误差。

引入数据计算中间件

独立数据计算引擎，提供不依赖于数据库的计算能力，合理地实现 E、T、L的过程!

多数据源接口

集算器除了关系型数据，还提供了大量非关系型数据接口，可直接访问MongoDB、Excel、Hive、SPARK、redis、阿里云OTS，也包括麦杰等。

代码更精简

以计算停起对为例，C# 要195行代码，涉及了大量嵌套循环以及难以理解的临时变量和条件关系，而集算器只需3行。

统一入库口径

在算法中实现数据比较、拉链、查询等操作，最后批量生成SQL，只更新一次数据库，显著减轻数据库压力，原本C#中的漏写数据、数据错误不再发生。

稳定性更强

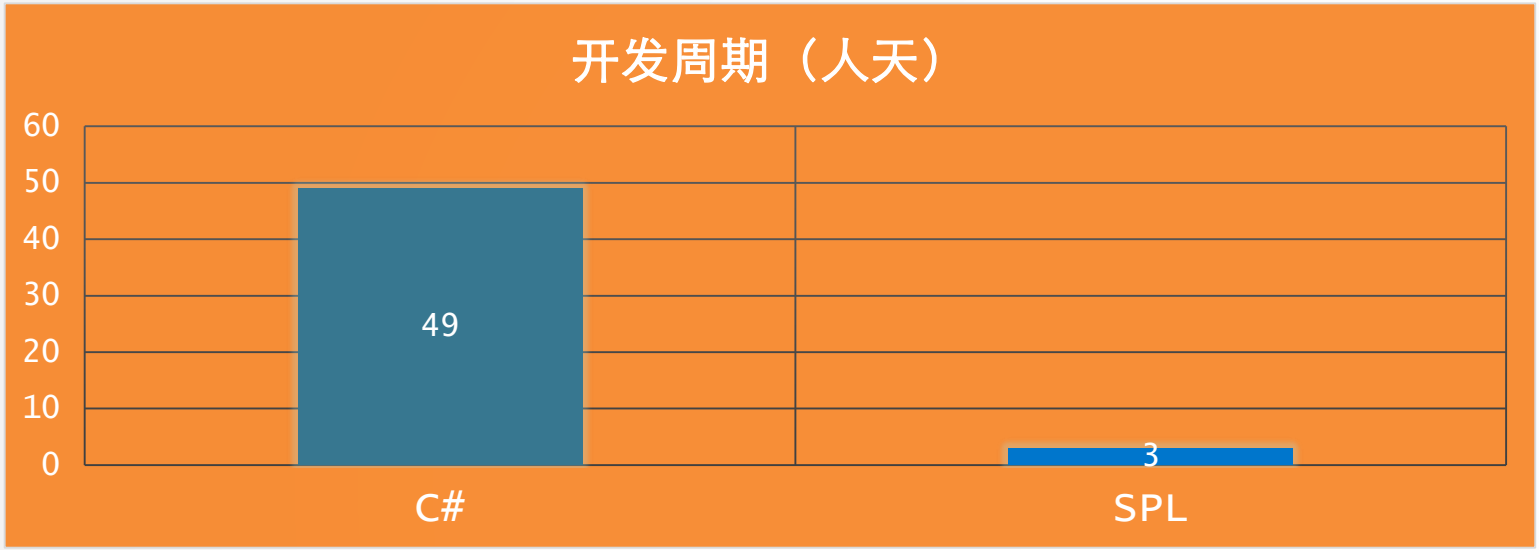
不依赖于数据库的库外计算能力；减少I/O操作，缩短时间窗口；且实现过程简单灵活

集算器完整代码示例

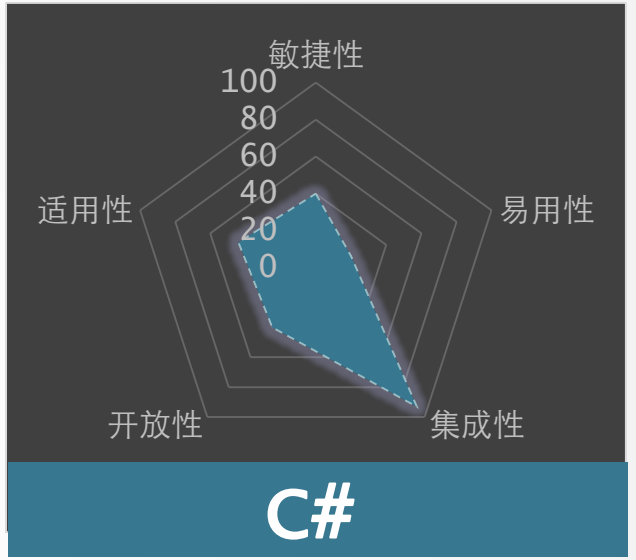
	A	B	C	D	E	F
1	=interval=3600	=nowhour=datetime("2017-07-05	=lasthour=elaps	=name="W3.NA.GD_N	/按库取表名，按表名取测点	/=invoke(GetMaijie.getNode
2	=invoke(GetMaijie.getData	=本麦杰	if 本麦	end	/空麦杰库处理	
3	=conn=connect("mysql")	=原始表=conn.query("select	=改后表=原始表.derive()		/初始化原始表和修改后的表	
4	if 原始表.len()==0	=改后表=改后表.record([停机=elapse@s(nowhour,-interval*24*8),起机=elapse@s(nowhour,-interval*24*8),interval@s(停机,起机),-1])				
5	if 本麦杰(1).AV==1.0	=tnowhour=nowhour	=tlasthour=lasthour		/回溯处理：从麦杰取前N个小时记录，找到最近的停，所有数据	
6		for	=tnowhour=elapse@s(tnowhour,-interval)			
7			=tlasthour=elapse@s(tlasthour,-interval)			
8			=invoke(GetMaijie.getData,name,string(tlasthour),string(tnowhour),0,1)			
9			=上麦杰=C8.new((t1=replace(~,"DynamicData:			
10			=上麦杰.pselect@z1(AV!=1.0 && AV[-1]==1.0)			
11			if C10==null	if	=本麦杰=本麦	break
12				else	=本麦杰=上麦杰|本麦杰	next
13			else	=本麦杰=上麦	break	
14	=本麦杰.group@i(AV!=1.0	=A14.(~.group@i(AV[-1]!=1.0 &&	=停起对		/变成起停对	
15	if 改后	end			/检查是否第二次误执行	
16						
17	if 改后	=改后表.m(-1).modify(停起对(1).bTime:startTime,停起对(1).AV:alarmno,interval@s(改			/库记录无起时补上	
18	if A5==true	if 停起对(1).eTime>改后表.m(-1).endTime	=改后表.record([停起对(1).eTime,停起对		/补上可能缺失的记录	
19	for 停起对	if interval@s(改后	=改后表=改后		/拉链并修改数据或新增	
20		else	=改后表.m(-1).modify(A19.bTime:startTime,interval@s(改后表.m(-1).endTime,A19.bTime):distance)			
21	=conn.update@l(改后表:原始表,alarm)				/实际更新库，可加事务处理	
22	=conn.close()					

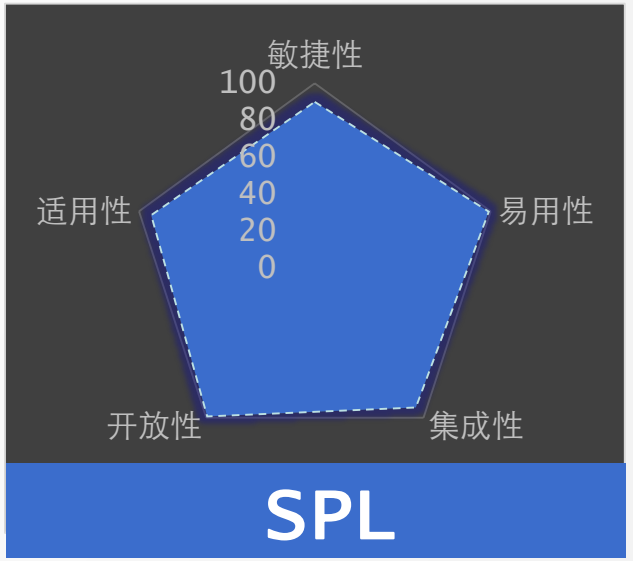
22行

实测：集算器开发效率在各方面的工作量评估



对比指标	C#	SPL	变化
代码行	2029	22	代码量减至1%
开发周期	49人天	3人天	工作量减至6%
执行性能	4.4秒（平均）	0.9秒（平均）	性能提升5倍
稳定性	较差	很稳定	提升稳定性
隐患	有	无	隐患消除





为什么集算器能有如此效率

独立计算引擎

不依赖于数据库的计算能力，拥有完整的敏捷语法体系

敏捷语法体系

语法简洁易学，更接近人的自然思维，特别适合做复杂过程计算

高性能计算

库外计算，为数据库减负，减少IO，缩短时间窗口

多样性数据源

丰富的多样性数据源接口，直接取数计算

创新技术
推动应用进步！


各设备故障分析
近7天设备的总故障率
近7天
1.53%
近1天
0.95%
近6小时
2.40%
全年故障综合分析
近7天故障情况
近7天故障次数前5的设备
服务器CPU使用率
今日
昨日
今天硬盘情况