SPL

Structure Process Language

润乾软件@2018

目录页

CONTENTS

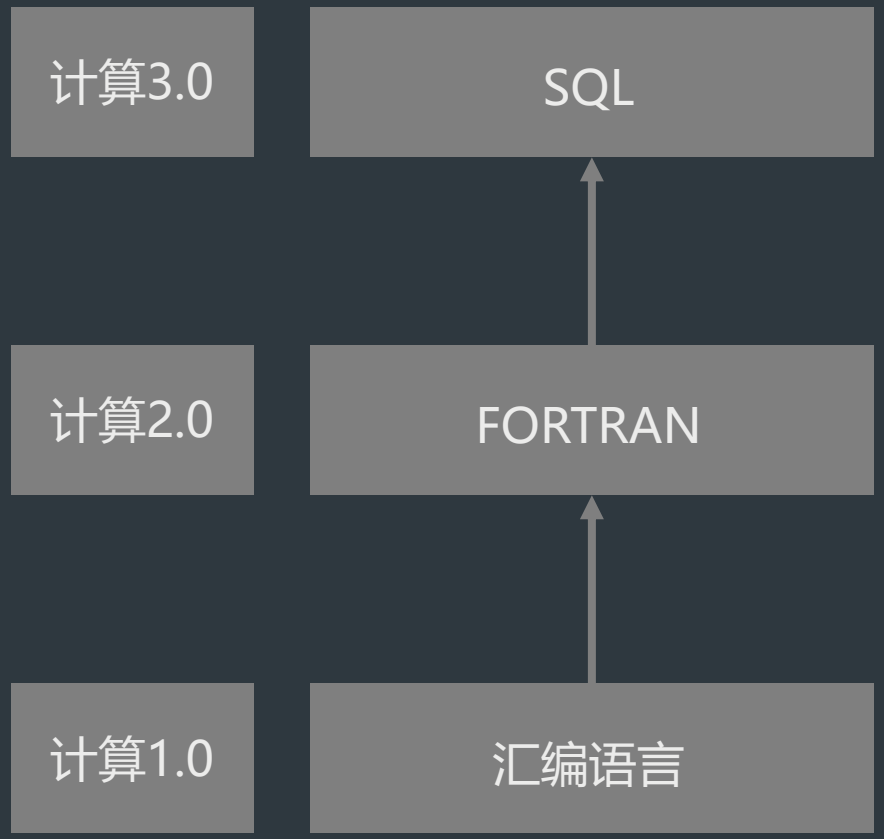
结构化数据计算

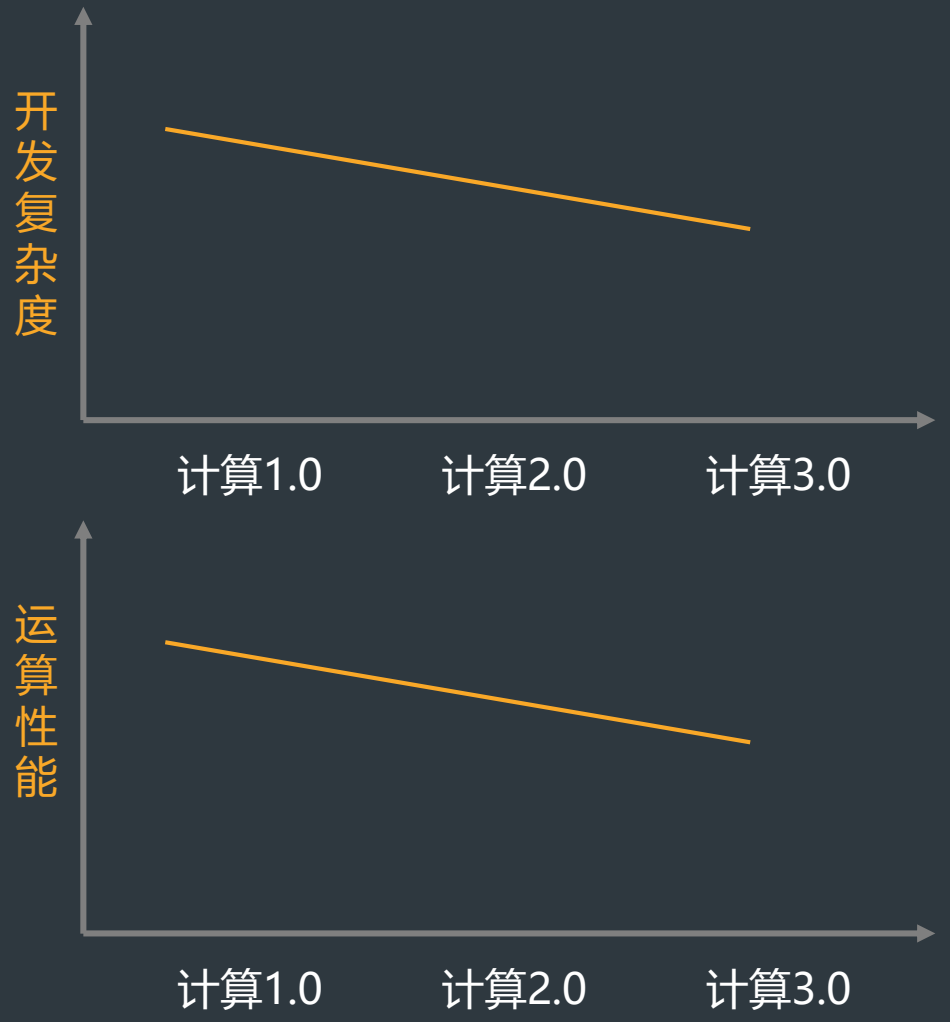
结构化数据仍是现代数据分析中最重要的内容

数据分析（计算）的目的是寻找事物之间的关联，而事物是由其属性决定的，技术上表现为结构化数据


计算3.0-SQL
计算2.0-Fortran
计算1.0-汇编语言
数据计算进化史

数据计算进化史


计算3.0
SQL
计算2.0
FORTRAN
计算1.0
汇编语言


开发复杂度
计算1.0
计算2.0
计算3.0
运算性能
计算1.0
计算2.0
计算3.0

3.0时代数据计算难题

开发难

复杂SQL实现难

前人代码维护难

库外计算实施难

多系统数据整合难

应用耦合性过高

MySQL无窗口函数

运算慢

查询分析响应慢

ETL时间长

存储过程执行慢

Hadoop性能低

库外数据处理慢

数据库资源不够用

原因分析

理论模型限制

SQL的理论模型是诞生于上世纪60年代的关系代数体系，该体系下实施计算难度非常大

不提倡分步运算

SQL语法很像英语，不提供分步机制，无过程性导致算法实现、维护困难

不支持有序集合

SQL不支持有序集合，涉及有序计算实现困难甚至实现不了，而有序计算在结构化数据计算中很常见

缺乏离散性

离散性允许记录游离于集合之外参与计算，SQL离散性差导致资源浪费，无法修改原记录

集合化不彻底

SQL不具备离散性，导致集合化不彻底，从而难以直接利用分组子集的内容

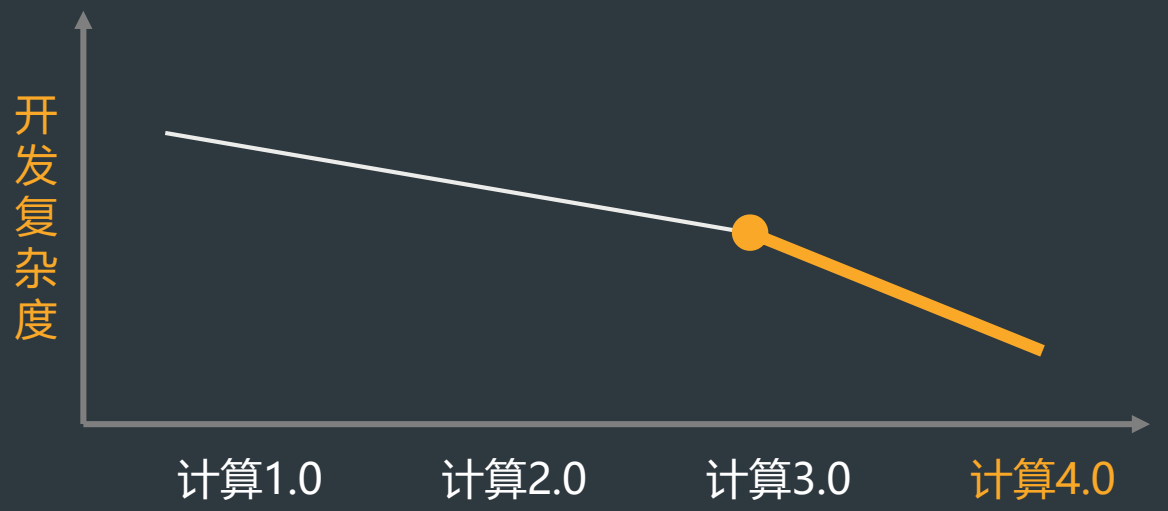
关联定义简单

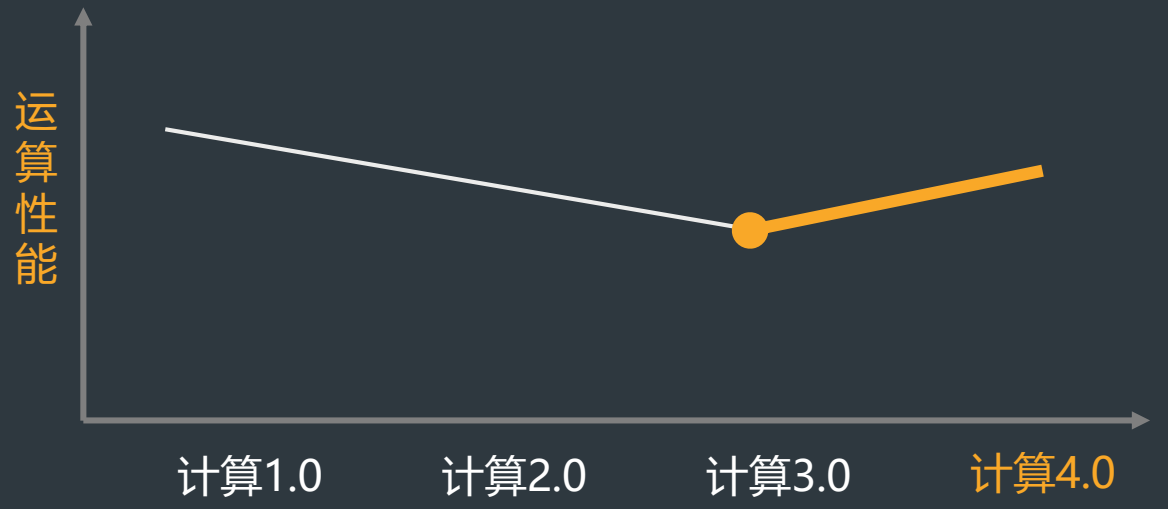
SQL对关联的定义过于简单，相当于加法，但在特定的情况下使用乘法更加高效，而非都用加法描述


计算4.0-SPL
计算3.0-SQL
计算2.0-Fortran
计算1.0-汇编语言
数据计算新时代

数据计算4.0


计算4.0
SPL
计算3.0
SQL
计算2.0
FORTRAN
计算1.0
汇编语言


开发复杂度
计算1.0
计算2.0
计算3.0
计算4.0


运算性能
计算1.0
计算2.0
计算3.0
计算4.0

目录页

CONTENTS

SPL

Structured Process Language

面向过程的结构化数据计算语言

SPL理论模型

离散数据集模型作为SPL的全新数学模型，全面改善了SQL在计算3.0时代的各种弊端，是计算4.0时代的重要理论基础

目录页

CONTENTS

集合化

离散性

深度集合化

有序性

SPL特性

集合化

集合化

结构化数据总是批量形式

集合运算库・lambda语法・动态数据结构

SQL是集合化的语言

WHERE,ORDER BY,GROUP

INTERSECT,UNION,MINUS

离散性

集合成员可以游离在集合外存在

独立运算或及其它游离成员再组合新集合运算

SQL缺乏离散性

只有单行记录的表，没有游离记录

计算结果总是新产生，和原记录无关

离散性应用

人员表计算张三和李四的年龄差和收入差-SPL

	A	
1	=employee.select@1(name=="张三")	
2	=employee.select@1(name=="李四")	
3	=A1.age-A2.age	
4	=A1.salary-A2.salary	

SQL的思路

人员表计算张三和李四的年龄差和收入差-SQL

```
1  SELECT (SELECT age FROM employee WHERE name='张三')
2      - (SELECT age FROM employee WHERE name='李四') FROM dual
3  SELECT (SELECT salary FROM employee WHERE name='张三')
4      - (SELECT salary FROM employee WHERE name='李四') FROM dual
```

离散性应用

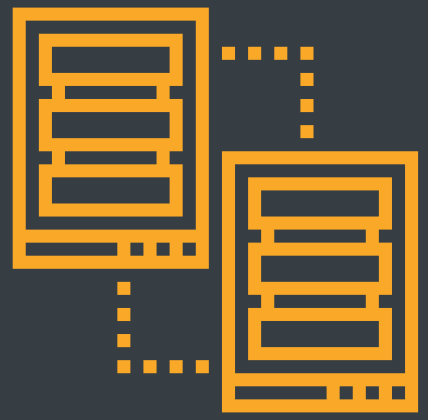
指定条件更新数据：业绩在前10%的代理商再奖励5%业绩

	A	
1	=agent.sort@z(amount).to(agent.len()*0.1)	取前10%
2	=A1.run(amount=amount*1.05)	奖励销售额

引用式外键：选出北京地区的交易记录

	A	
1	>交易记录.switch(地区,地区表:编号)	建立外键引用
2	=交易记录.select(地区.名称=="北京")	用外键引用记录的字段过滤

深度集合化



彻底的集合化需要离散性的支持

允许游离成员组成新集合

分组子集 • 非常规聚合 • 主子表

分组子集（一）

找出总分500分以上的学生的各科成绩记录

	A	
1	=成绩表.group(学生).select(~.sum(成绩)>=500).conj()	

有离散性后才能还原分组运算的本来面目

SQL没有游离记录构成的显式集合，无法保持分组结果，强迫聚合后要两次扫描并连接

```
1 WITH T AS
2 (SELECT 学生 FROM 成绩表 GROUP BY 学生 HAVING SUM(成绩)>500)
3 SELECT TT.* FROM T LEFT JOIN 成绩表 TT on T.学生=TT.学生
```

分组子集（二）

用户在最后一次登录前三天内的登录次数

	A
1	=登录表. group(uid;~.max(logtime):last,~.count(interval(logtime,last)<=3):num)

针对分组子集的聚合运算较复杂时难以用简单聚合式写出，保留分组子集再结合分步计算则很容易

SQL不能保持子集，要用子查询在原集上附加信息，导致多次计算

```
1  WITH T AS
2     (SELECT uid,max(logtime) last FROM 登录表 GROUP BY uid)
3  SELECT T. uid,T.last,count(TT.logtime)
4     FROM T LEFT JOIN 登录表 TT ON T.uid=TT.uid
5     WHERE T.last-TT.logtime<=3 GROUP BY T.uid,T.last
```

非常规聚合（一）

列出用户首次登录的记录

	A	
1	=登录表.group(uid).(~.minp(logtime))	

聚合运算不一定总是SUM/COUNT这些，还可以理解为取出某个成员

有离散性时可以简单针对分组子集实施这种聚合

```
1  SELECT * FROM
2      (SELECT RANK() OVER(PARTITION BY uid ORDER BY logtime) rk, T.* FROM 登录表 T) TT
3  WHERE TT.rk=1
```

非常规聚合（二）

列出每个用户最近一次登录间隔

	A	
1	=登录表.groups(uid;top(2,-logtime))	最后2个登录记录
2	=A1.new(uid,#2(1).logtime-#2(2).logtime:间隔)	计算间隔

聚合函数返回值不一定是单值，也可以返回一个集合

彻底的集合化后很容易针对分组子集实施返回集合的聚合运算

```
1  WITH T AS
2      (SELECT RANK() OVER(PARTITION BY uid ORDER BY logtime DESC) rk, T.*  FROM 登录表 T)
3  SELECT uid,(SELECT TT.logtime FROM TT where TT.uid=TTT.uid and TT.rk=1)
4      -(SELET TT.logtim FROM TT WHERE TT.uid=TTT.uid and TT.rk=2) 间隔
5  FROM 登录表 TTT GROUP BY uid
```

主子表

由订单明细计算金额

	A	
1	=订单表.derive(订单明细.select(编号==订单表.编号):明细)	建立子表集合字段
2	=A1.new(编号,客户,明细.sum(单价*数量):金额)	计算订单金额

字段取值也可以是个集合，从而轻松描述主子表，适应于多层结构数据

SQL没有显式集合数据，没有离散性也不能引用记录，要JOIN后再GROUP

```
1  SELECT订单表.编号,订单表.客户,SUM(订单明细.价格)
2      FROM 订单表
3  LEFT JOIN 订单明细 ON 订单表.编码=订单明细.编号
4  GROUP BY 订单表.编号,订单表.客户
```

有序性

有序计算是集合化与离散性的结合物

运算不仅与数据本身有关，还和数据所在位置有关

跨行引用·有序分组·位置利用

有序计算

关系代数延用了数学上的无序集合概念

早期SQL要生成序号后JOIN才能实现有限的有序计算

计算股票的涨幅

```
1 WITH T AS
2       (SELECT rownum 序号,交易日,收盘价
3       FROM ( SELECT * FROM 股票 ORDER BY 交易日) )
4 SELECT T1.交易日,T1.收盘价-T2.收盘价
5 FROM T T1 JOIN T T2 ON T1.序号=T2.序号+1
```

SQL2003标准增加了窗口函数方便生成序号并引用相邻成员

```
1 SELECT  交易日,收盘价-LAG(收盘价) OVER (ORDER BY 交易日) FROM 股票
```

跨行引用（一）

由订单明细计算金额

	A
1	=销售表.sort(产品,月份)
2	=A1.select(if(产品==产品[-1],销售量/销售量[-1])>1.1 && 销售额/销售额[-1])>1.1))

集合有序后可直接提供跨行引用的方案

SQL窗口函数必须使用子查询才能实现跨行计算，多个跨行项要分别使用窗口函数

```
1  WITH T AS
2      (SELECT 销售量/LAG(销售量) OVER(PARTITION BY 产品 ORDER BY 月份) r1
3      (SELECT 销售额/LAG(销售额) OVER(PARTITION BY 产品 ORDER BY 月份) r2, A.*, FROM 销售表 A)
4  SELECT * FROM T WHERE r1>1.1 AND r2>1.1
```

跨行引用（二）

计算每月前后各一个月的销售额移动平均值

	A
1	=销售表.sort(月份).derive(销售额{-1,1}.avg():移动平均)

有序集合上容易提供跨行集合引用方案

SQL窗口函数只有简单的跨行引用，涉及集合要用成员去拼

```
1  WITH B AS
2      (SELECT LAG(销售额) OVER (ORDER BY 月份) f1, LEAD(销售额) OVER (ORDER BY 月份) f2,  A.* FROM 销售表 A)
3      SELECT 月份,销售额,
4      (NVL(f1,0)+NVL(f2,0)+销售额)/(DECODE(f1,NULLl,0,1)+DECODE(f2,NULL,0,1)+1) 移动平均
5  FROM B
```

有序分组（一）

一批婴儿中连续出生的同性别人数超过5人的有多少组

	A
1	=婴儿表.sort(生日).group@o(性别).count(~.len()>=5)

分组可能和次序相关，不只是等值分组

有序集合上可以定义与次序有关的分组，考察值变化时即产生新组

```
1  SELECT COUNT(*) FROM
2      (SELECT 改变次数 FROM
3          (SELECT SUM(改变标志) OVER ( ORDER BY 生日) 改变次数 FROM
4          (SELECT CASE WHEN 性别=LAG(性别) OVER (ORDER BY 生日) THEN 0 ELSE 1 END 改变标志  FROM 婴儿表) )
5      GROUP BY 改变次数 HAVING COUNT(*)>=5)
```

有序分组（二）

一支股票最长连续上涨了多少天

	A
1	=股票.sort(交易日).group@i(收盘价<收盘价[-1]).max(~.len())

另一种和次序有关的分组，条件成立时产生新组

```
1  SELECT max(连续日数)-1 FROM
2      (SELECT count(*) 连续日数 FROM
3          (SELECT SUM(涨跌标志) OVER ( ORDER BY 交易日) 不涨日数 FROM
4          ( SELECT 交易日,
5              CASE WHEN  收盘价>LAG(收盘价) OVER( ORDER BY 交易日 THEN 0 ELSE 1 END 涨跌标志
6          FROM 股票 ))
7      GROUP BY 不涨日数)
```

混合情况

找出连续上涨三天的股票

	A
1	=股票.sort(交易日).group(代码)
2	=A1.select((a=0,~.pselect(a=if(收盘价>收盘价[-1],a+1,0):3))>0).(代码)

分组子集与有序计算的组合

```
1  WITH A AS
2      (SELECT 代码,交易日, 收盘价-LAG(收盘价) OVER (PARITITION BY 代码 ORDER BY 涨幅) FROM 股票)
3  B AS
4      (SELECT 代码,
5          CASE WHEN 涨幅>0 AND
6              LAG(涨幅) OVER (PARTITION BY 代码 ORDER BY 交易日) >0  AND
7              LAG(涨幅,2) OVER PARTITION BY 代码 ORDER BY 交易日) >0
8              THEN 1 ELSE 0 END 三天连涨标志  FROM A)
9  SELECT distinct 代码 FROM B WHERE 三天连涨标志=1
```

位置利用（一）

计算一组商品价格的中位数

	A
1	=价格表.sort(价格).([(价格表.len()+1)\2,价格表.len()\2+1]).avg()

有序集合可以直接用序号访问成员

SQL的无序集合需要人为制造出序号，不分步运算加剧了困难

```
1  WITH N AS (SELECT COUNT(1) FROM 价格表)
2  SELECT AVERGE(价格) FROM
3      (SELECT 价格,ROW_NUMBER() OVER (ORDER BY 价格) r FROM 价格表 ) T
4      WHERE r=TRUNC((N+1)/2) OR r=TRUNC(N/2)+1)
```

位置利用（二）

某股票股价最高的那三天的平均涨幅

	A
1	=股票.sort(交易日)
2	=A1.calc(A1.ptop(3,-收盘价),收盘价-收盘价[-1]).avg()

有序集合可以提供丰富的按位置访问机制

无序集合不能利用位置访问相邻成员，计算量增大，描述复杂度提高

```
1  SELECT AVG(涨幅) FROM
2      ( SELECT 交易日, 收盘价-LAG(收盘价) OVER ( ORDER BY 交易日) 涨幅 FROM 股价表
3          WHERE 交易日 IN
4              (SELECT 交易日 FROM
5                  (SELECT 交易日, ROW_NUMBER() OVER(ORDER BY 收盘价 DESC) 排名 FROM 股票)
6      WHERE 排名<=3 )
```

结构化数据计算总结

离散性与集合化的有效结合

集合化是批量计算的基本能力

离散计算也不可或缺

离散性支持更彻底的集合化

离散性产生有序集合运算

离散集合模型 = 集合运算 + 游离成员

=> 彻底集合化/有序集合

目录页

CONTENTS

集算器

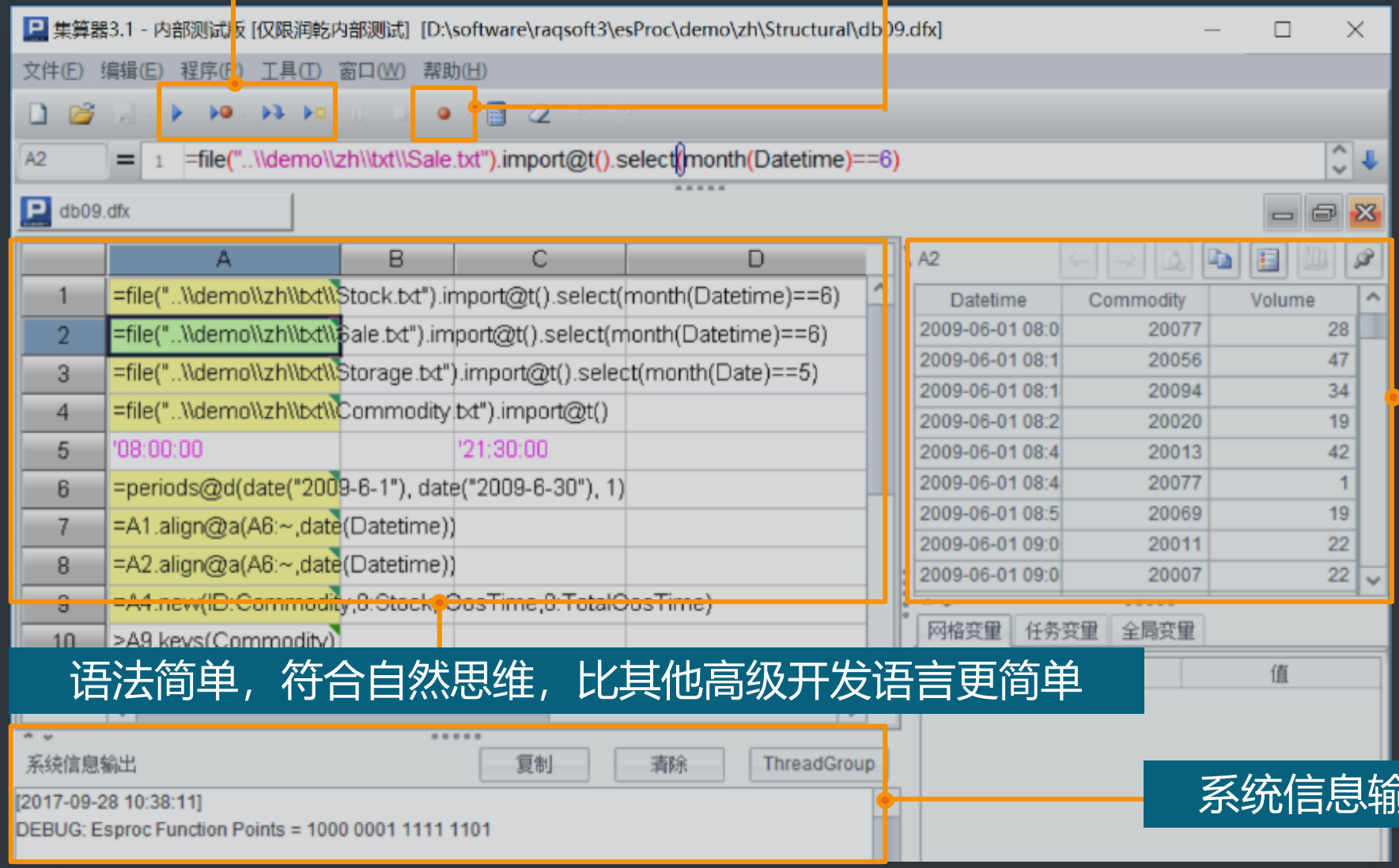
集算器是离散数据集的工程实现产品

SPL作为集算器的形式化语言

开发环境

执行、调试执行、单步执行

设置断点



网格结果所见即所得，易于调试；方便引用中间结果

语法简单，符合自然思维，比其他高级开发语言更简单

系统信息输出，异常随时查看

专门设计的语法体系

特别适合复杂过程运算

	A	B	C	D	E	F
1	=esProc.query("SELECT 订单ID AS 合同,订购日期 AS 日期,客户 ,订单金额 AS 金额,员工ID AS 销售 FROM 销售记录表 WHERE year(订购日期)=? OR year(订购日					
2	=esProc.query("select * from 员工表")					
3	>A1.run(销售=A2.select@1(编号:A1.销售))		/字段值 是记录			
4	=A1.group(销售)					
5	=create(销售,今年销售额,去年销售额,增长率,客户数,大客户数,大客户占比)					
6	for A4	=A6(1).销售.姓名				
7		=A6.select(year(日期)==年份).sum(金额)		/今年销售额		
8		=A6.select(year(日期)==年份-1).sum(金额)		/去年销售额		
9		=B8/B7-1	/增长率			
10		=A6.group(客户).(~.sum(金额))				
11		=B10.count()				
12		=B10.count(~>1000)				
13		=B12/B11				
14		>A5.insert(0,B6,B7,B8,B9,B11,B12,B13)				
15	result A5					

天然分步、层次清晰、直接引用单元格名无需定义变量

丰富的运算类库

专门针对结构化数据表设计

	A	B	C
1	=esProc.query("SELECT 订单ID AS 合同,订购日期 AS		/读取销售记录表
2	=A1.group(销售)		
3	=create(销售,今年销售额,去年销售额,客户数,大客户数)		
4	for A2	=A4(1).销售	
5		=A4.select(year(日期)==年份).sum(金额)	
6		=A4.select(year(日期)==年份-1).sum(金额)	
7		=A4.group(year(日期)).(sum(金额))	
8		=B7.count()	
9		=B7.count(~>=10000)	

分组、循环

	A	B	C
1	=esProc.query("select * from 员工表")		
2	=A1.select(性别=="男")		
3	=A1.select(出生日期>=date("1970-01-01"))		
4	=A2^A3	/交运算，统计晚于1970年出生的男员工	
5	=A2&A3	/并运算，统计男员工或者晚于1970年出生的员工	
6	=A2\A3	/运算，统计早于1970年出生的男员工	
7	=A4.sum(工资)		
8	=A5.avg(年龄)		
9	=A6.sort(出生日期)		
10	/集合作为基本数据类型可参与运算		
11			

集合运算

	A	B	C
1	=file("交易记录.txt").import@t()		
2	=A1.sort(客户编码,交易日期)		
3	=A2.select(车辆型号=="捷达" || 车辆型号=="迈腾").dup@t()		
4	=A3.derive(interval(交易日期[-1],交易日期):间隔)		
5	=A4.select(车辆型号[-1]=="捷达" && 车辆型号=="迈腾" &&客户编码==客户编码[-1])		
6	=A5.avg(间隔)		
7			
8			
9			

排序、过滤

	A	B	C
1	=esProc.query("select * from 员工表")		
2	=A1.sort(入职日期)		
3	=A2.pmin(出生日期)	/出生最早的员工的记录序号	
4	=A2(to(A3-1))	/直接用序号访问成员	
5	=esProc.query("select * from 股价表 where 股票代码='000062'")		
6	=A5.sort(交易日期)		
7	=A6.pmax(收盘价)	/收盘价最高的那条记录的序号	
8	=A6.calc(A7,收盘价/收盘价[-1]-1)		
9			
10	/直接用序号访问成员		
11			

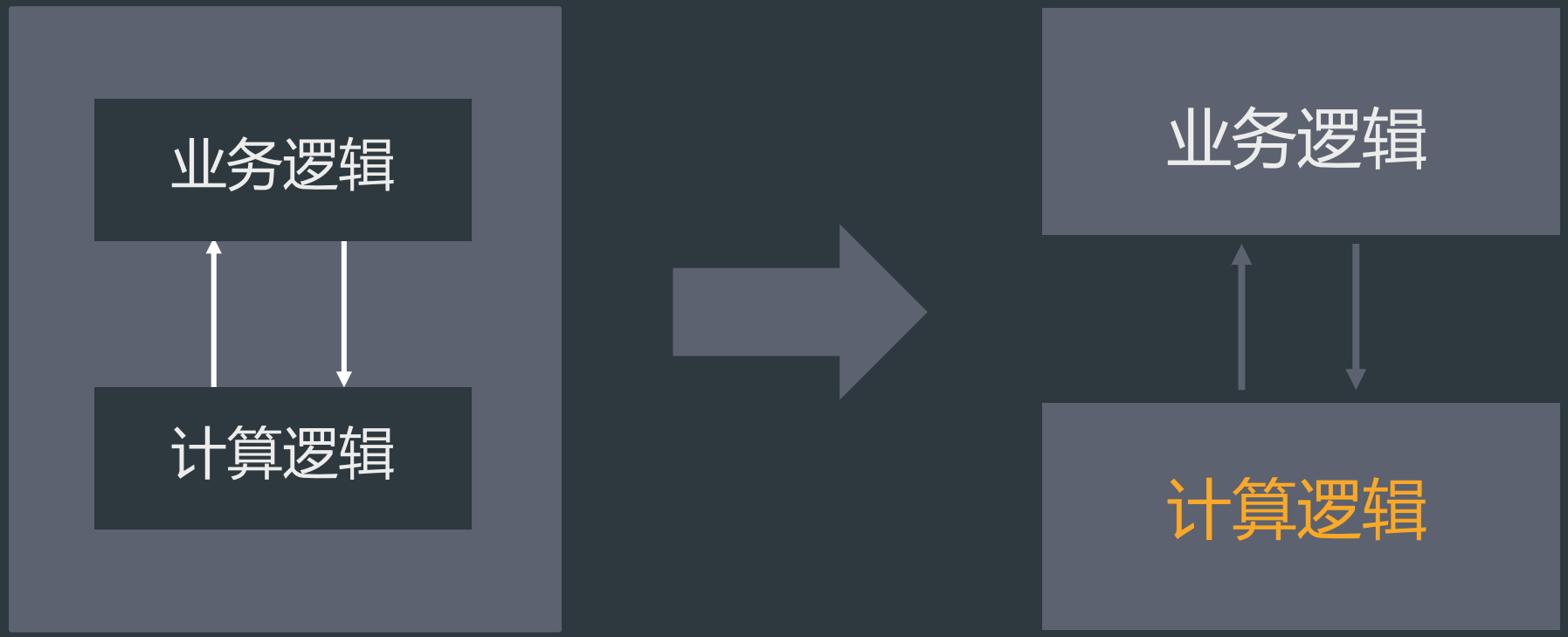
有序集合

热切换

集算器脚本解释执行，支持不停机热切换

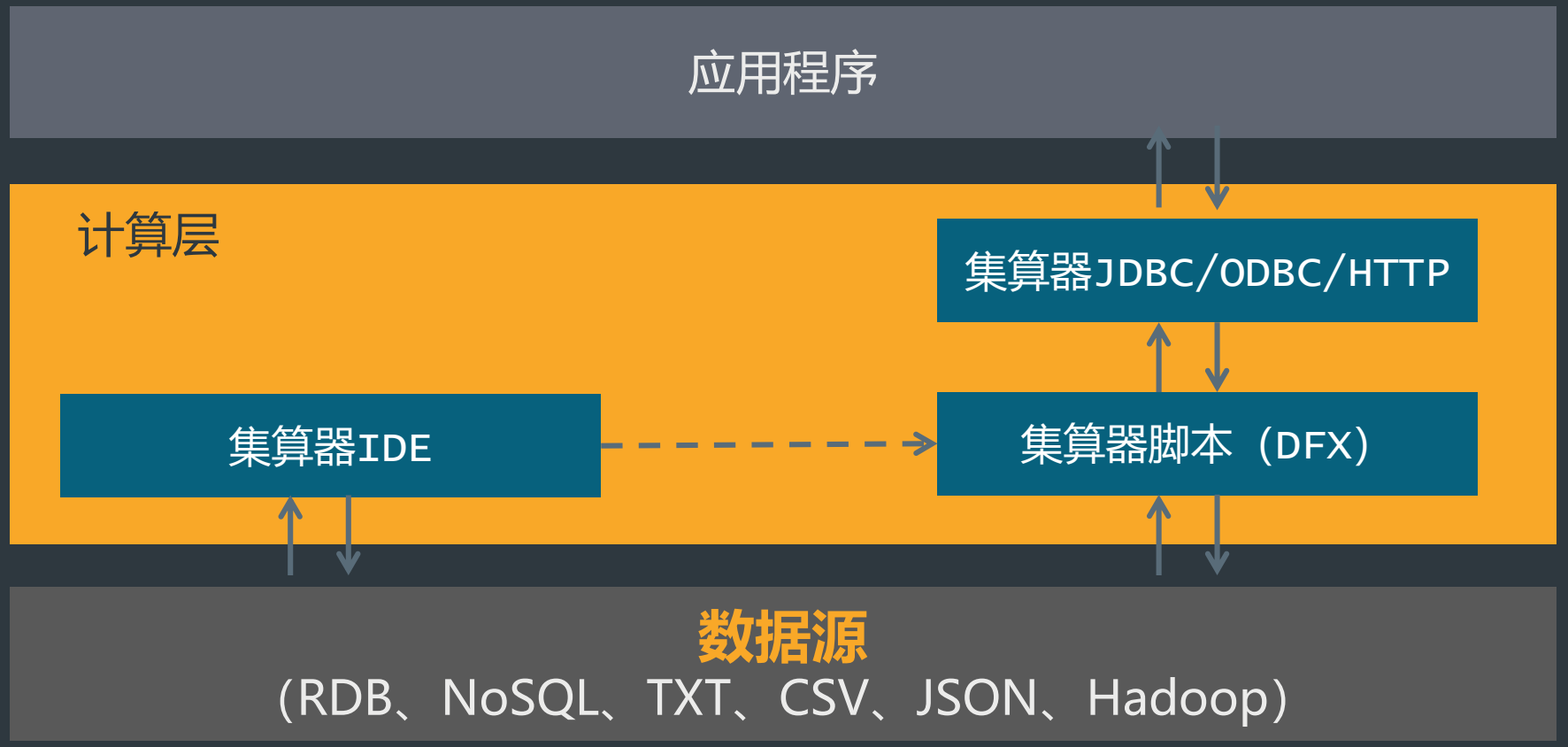
低耦合

计算逻辑脚本单独维护，方便模块化



集成性

集算器使用JAVA开发，提供标准应用接口可无缝集成到应用中



多样性数据源

直接使用多个数据源混合计算，无需后台先将数据统一（ETL）后再计算

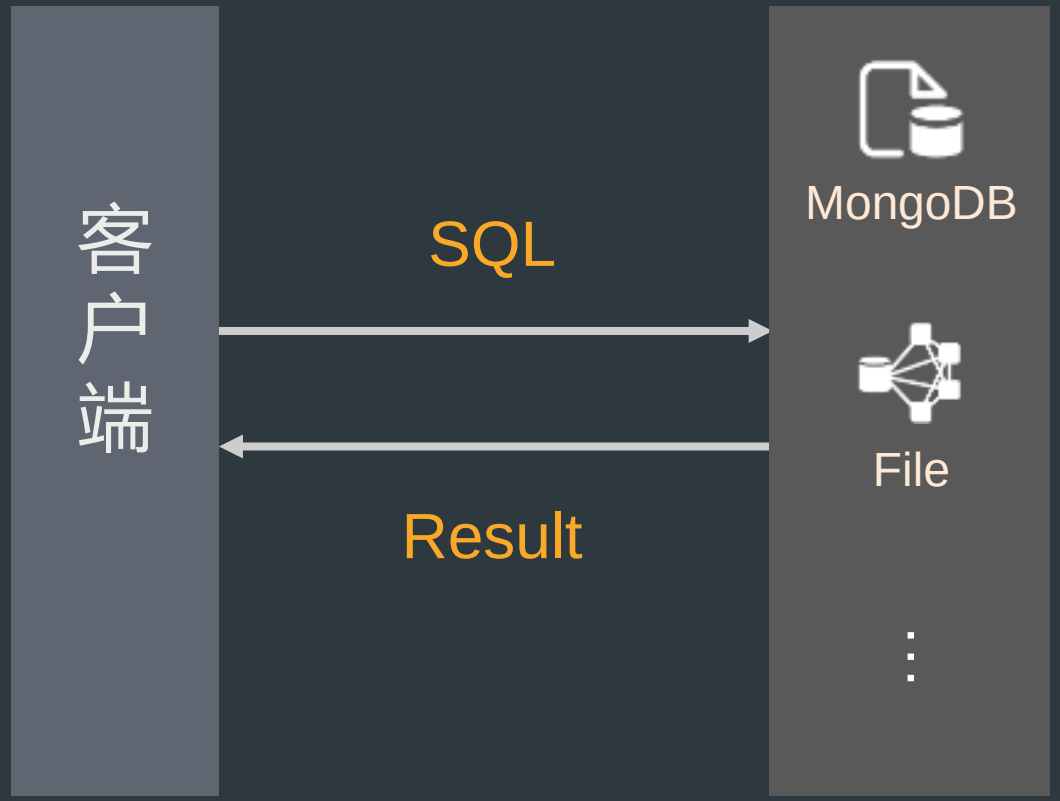
外部数据接口

- RDB：Oracle,DB2,MS SQL,MySQL,PG,....
- TXT/CSV，JSON/XML，EXCEL
- Hadoop：HDFS，HIVE，HBASE
- MongoDB，REDIS，...
- HTTP、ALI-OTS
- ...

内置接口，即装即用

文件SQL查询

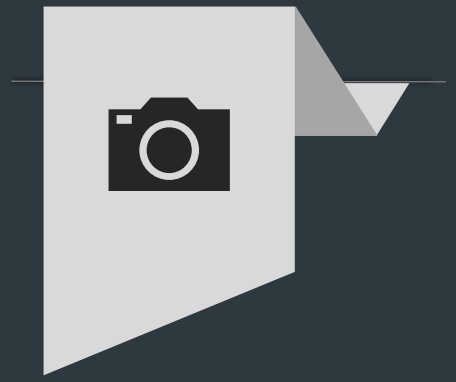
针对MongoDB和文件等使用SQL进行查询



集算器赋予NoSQL和文件SQL查询能力

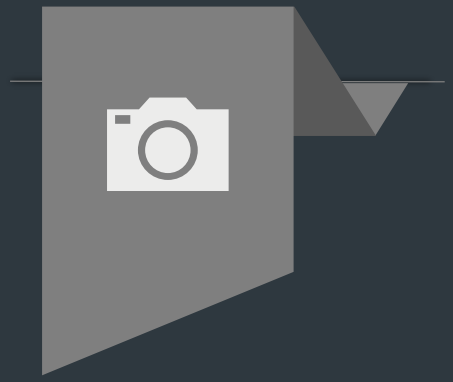
目录页

CONTENTS



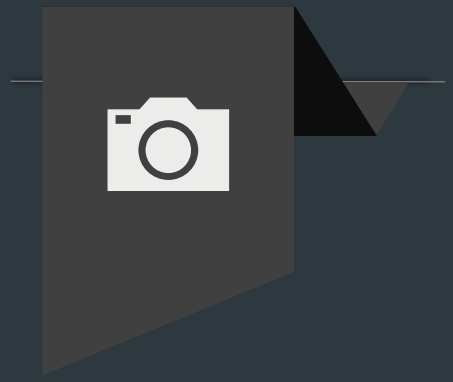
数据计算中间件

可集成到应用中作为数据计算层，为应用提供数据支持



挖掘前数据整理

整理数据挖掘前数据，以满足数据挖掘需要



临时计算

应对临时性查询和提数需求

桌面分析

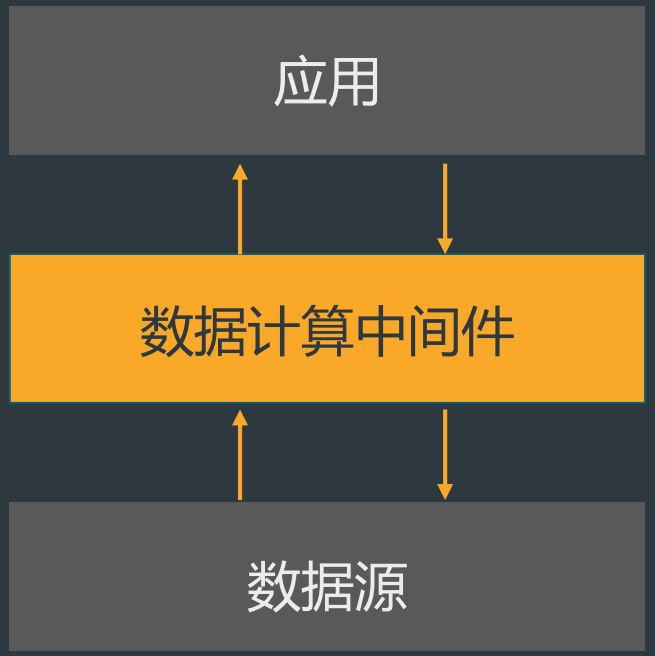
单机桌面数据分析，使用更灵活

应用场景

数据计算中间件

位于数据源和应用之间提供通用计算服务，DCM (Data Computing Middleware)

有效分担数据源压力、降低耦合性、提供开放计算能力

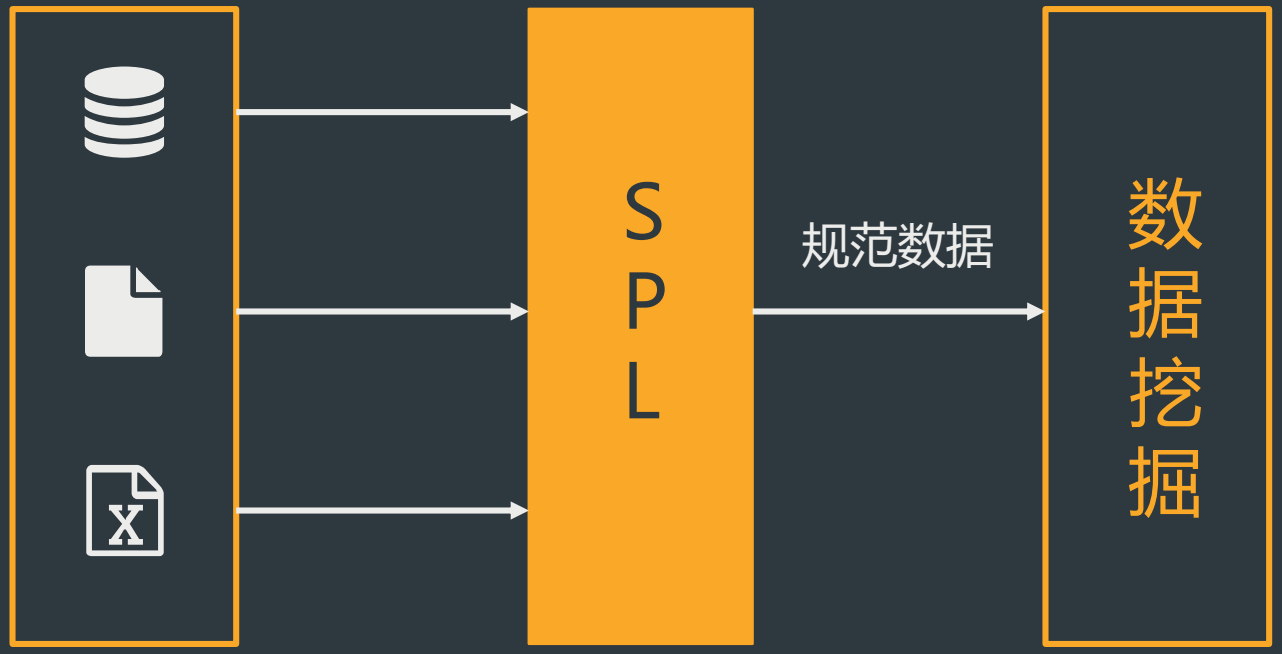


*更多内容请参考<数据计算中间件.pptx>

挖掘前数据整理

数据整理通常占数据挖掘工作量一半以上

SPL提供开放、灵活简单的整理方式



临时计算

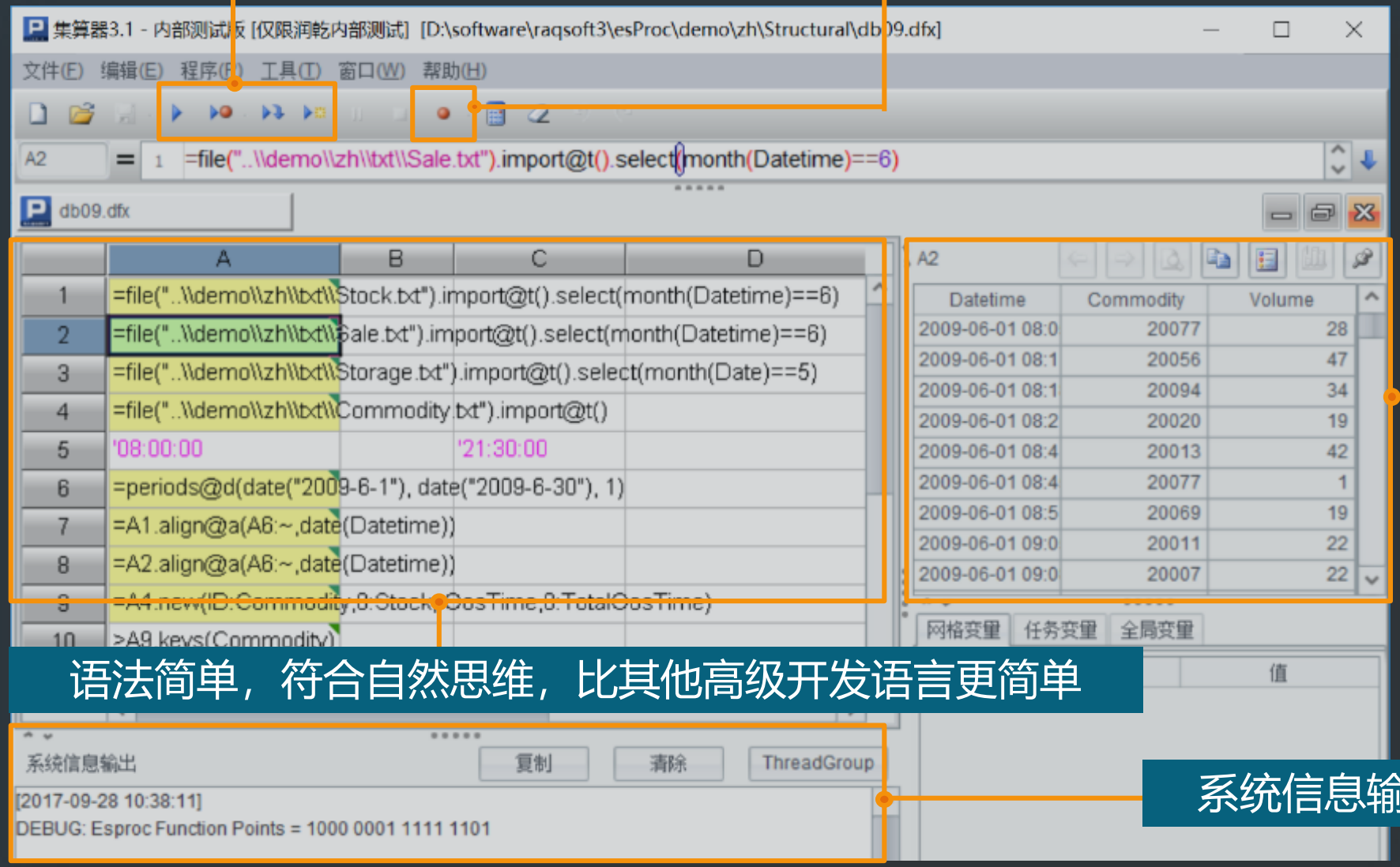
SPL开放计算能力可以很好满足临时性计算需求

- 应对业务部门的临时取数需求
- 即时进行的（外部）数据分析与考察
- 有业务规则的测试数据生成
- 大数据计算的优化方案实验
- 非日常（外部）数据清洗与入库

桌面分析

执行、调试执行、单步执行

设置断点



网格结果所见即所得，易于调试；方便引用中间结果

语法简单，符合自然思维，比其他高级开发语言更简单

系统信息输出，异常随时查看

目录页

CONTENTS

计算代码举例

文本计算

无结构计算

结构化计算

类文本数据

数据库计算

分组

序运算

字串与日期

无结构运算-文本解析（一）

实际需求

文本T.txt的行内数据项由不确定数量的空格分隔开：

2010-8-13　991003 3166.63 3332.57　3166.63　3295.11

2010-8-10　991003 3116.31 3182.66　3084.2　3140.2

……

现在要计算每行最后四项数据的平均值列表。

	A
1	=file("T.txt").read@n().(~.split@tp(" ").to(-4).avg())

read@n()将文本读入成字串集合，split@t(" ")将字串按不定数量的空白符拆成子串集合，@p将自动解析成合适的数据类型以便进一步计算（这里计算平均）。

无结构运算-结构化（一）

实际需求

日志S.log中每3行构成一段完整信息

需要将其解析成结构化数据后再保存到T.txt

	A	B	
1	=file("S.log").read@n()		
2	=create(…)		建立目标结果集
3	for A1.group((#-1)\3)	…	按行号分组，每3行一个单位
…		…	从A3（这3行）中解析出字段值
…		>A2.insert(…)	插入到目标结果集
…	>file("T.txt").export(A2)		写出结果

有了按行号分组的机制，就可以用循环每次处理一组数据，简化难度。 显然，更简单的单行情况是其特例

无结构运算-查找统计（一）

实际需求

grep是常用命令，但有些OS没有，用程序中实现也不简单。

当前目录下有很多文本文件

找出含有指定单词的文件，并列出所在行内容及行号

	A
1	=directory@p(“*.txt”)
2	=A1.conj(file(~).read@n().(if(pos(~,"xxx"),[A1.~,#,~].string()))).select(~))

集算器提供了**文件系统的遍历功能**，结合文本计算能力，只要两句代码就能完成。

结构化运算-读入控制（一）

实际需求

D.csv(逗号分隔)文件中有多列,每列都有标题

需要读入4列:name,sex,age,phone；其中phone是全数字但必须读成字符串类型

	A	
1	=file(“D.csv”).import@tc(name,sex,age,phone:string)	

import函数有丰富的参数和选项控制是否有标题、分隔符、读入的列及其数据类型，大部分结构化文本的读入可以一句完成。

相当于把文本当作数据库表读入

结构化运算-常规运算（一）

实际需求

从文本D.csv中找出男25岁以上、女23岁以上的人：

1) 按姓名排序列出；2) 按性别分组计算平均年龄； 3) 列出所有出现过的姓氏（不考虑复姓）。

	A	
1	=file(“D.csv”).import@tc(name,sex,age)	
2	=A1.select(sex==“男”&& age>=25||sex==“女”&& age>=23)	过滤
3	=A2.sort(name)	排序
4	=A2.groups(sex;avg(age):age)	分组汇总
5	=A2.id(left(name,1))	唯一值

集算器提供了丰富的结构化计算功能。

一定程度上可以将文本作为数据库表进行运算，无数据库也能获得类似SQL的计算能力。

结构化运算-文件对比（一）

实际需求

文本文件T1.txt和T2.txt都有id列

找出T1.txt和T2.txt中共有的id值

找出T1.txt中有，T2.txt中没有的id值

	A	
1	=file(“T1.txt”).import@ti(id)	@i结果只有一列时返回成序列
2	=file(“T2.txt”).import@ti(id)	
3	=A1^A2	交集，即T1和T2共有的值
4	=A1\A2	差集，即T1中有而T2中没有的值

比对列值直接用集合交、差运算即可。

类文本数据-json

实际需求

Java有足够多的类库用于解析和生成json，但缺乏后续计算能力。
集算器支持多层结构数据，可以不丧失信息地将json解析成可计算的内存数据表进一步处理。

```
{
“order”:[
        {
        “client”:”北京润乾软件”,
        “date”:”2015-6-23”,
        “item” : [
                {“product”:”HP笔记本”,“number”:4,“price”:3200},
                {“product”:”DELL服务器”,“number”:1,“price”:22100}]
        },…]
}
```

Json数据要写入数据库中：
order表，结构为：orderid,client,date；
orderdetail表，结构为：orderid,seq,product,number,price
其中：orderid和seq按顺序生成即可。

类文本数据-json

```
{
“order”:[
        {
        “client”:”北京润乾软件”,
        “date”:”2015-6-23”,
        “item” : [
                {“product”:”HP笔记本”,“number”:4,“price”:3200},
                {“product”:”DELL服务器
”,“number”:1,“price”:22100}]
        },…]
}
```

	A	
1	=file(“data.json”).read().import@j().order	
2	=A1.new(#:orderid,client,date)	
3	=A1.news(item;A1.#:orderid,#:seq,product,number,price)	
4	>db.update@i(A2,order)	
5	>db.update@i(A3,ordedetail)	

类文本数据-Excel

实际需求

Excel相当于已经结构化的文本。Java有能解析xls的开源类库（如poi），功能强大但过于底层，开发复杂度高。

集算器封装了poi，可读入xls转成二维数据表再进一步运算。

- position.xls保存“点”的位置，range.xls保存“范围”的起止位置
- 对position中的每个“点”在range中找到包含这个“点”的第一个“范围”
- 将“点”和对应的“范围”写到result.xls中。后。

range.xls		
range	start	stop
Range1	4561	6321
Range2	9842	11253
…		

position.xls	
Point	position
point1	5213
point2	10254
…	

类文本数据-Excel

range.xls		
range	start	stop
Range1	4561	6321
Range2	9842	11253
…		

position.xls	
Point	position
point1	5213
point2	10254
…	

	A
1	=file(“range.xls”).importxls@t()
2	=file(“position.xls”).importxls@t()
3	=A2.derive((t=A1.select@1(position>=start&&position<=stop)).range:range, t.start:start,t.stop:stop)
4	=file(“result.xls”).exportxls(A3)

集算器读入xls后可以充分利用已有的计算能力。

用Excel自带的VBA只能硬编码实现JOIN，非常繁琐，有时不得不导入数据库来做。

动态列-列间统计

实际需求

体育测验表结构为：姓名、短跑、长跑、跳远、铅球、...；成绩分为优秀、良好、及格、不及格四档，现在要**统计各档次在所有项目上的人数合计**。

	A	
1	=db.query("select * from 测验表")	
2	=A1.conj(~.array().to(2,))	从第2字段的各项目成绩合并起来
3	=A2.groups(~:成绩;count(1):数量)	分组汇总

集算器可以将多列作为一个集合，方便处理列数不确定的情况。

动态列-转置计算

实际需求

帐户状态变化表T，需要输出《**月帐户状态报表》：指定月份帐户每日的状态，若当日无记录，则延用前一日的状态。

SQL困难

虽然是静态转置，但列数较多且有规律，完全静态写出很麻烦。

转置过程中涉及到列间计算，即使有pivot语句用SQL也难以写出

帐户状态变化表T

序号	帐户	状态	发生日期
1	A	透支	2014-1-4
2	A	正常	2014-1-8
3	A	挂失	2014-3-21
…			

****月帐户状态报表**

帐户	1	2	3	4	5	6	7	8	9	…	31
A				透支	透支	透支	透支	正常	正常	…	正常
…											

动态列-转置计算

序号	帐户	状态	发生日期
1	A	透支	2014-1-4
2	A	正常	2014-1-8
3	A	挂失	2014-3-21
…			

帐户	1	2	3	4	5	6	7	8	9	…	31
A				透支	透支	透支	透支	正常	正常	…	正常
…											

	A	B
1	=db.query("select * from T where year(日期)=? and month(日期)=?",2014,1)	
2	=create(帐户,${to(31).concat@c()})	
3	for A1.group(帐户)	=31.(null)
4		>A3.run(B3(day(日期))=状态)
5		>B3.run(~=ifn(~,~[-1])
6		>A2.record(A3.帐户|B3)
7	return A2	

转置标准流程：A2中先用宏生成目标结果集，再在A3-B6的循环中将数据变换后插入到结果集

分组-非等值分组（一）按段分组

实际需求

按段分组：如成绩段（优秀、良好...），年龄段（青年、中年...）等。

集算器penum函数可返回枚举条件的序号：

["?<60","?>=60&&?<75", "?>=75&&?<90", "?>=90"].penum(成绩)

连续分段可以用pseg函数更简单地获得分段序号：[60,75,90].pseg(成绩)

上述条件和分段都是普通数组，可作为参数传递进来，长度不限制

基于分段号即可将枚举分组和按段分组转变成普通的等值分组

分组-非等值分组（二）对齐

实际需求

分组时要按指定的次序而不是数据本身排序，比如中国省份排列时一般要将北京放在第一个。

- 集算器提供了align@s函数专门用于对齐排序：

  T.align@s(["北京","河北","山东",...],地区)

- 将表T以字段地区按指定的次序排序
- 排序依据是个普通数据，可以作为参数传递

分组-逆分组

实际需求

分期付款表结构为：编号、总金额、起始日、总期数；

要将每笔贷款拆分成多期记录，结构为：编号、期数、还款日、金额。总金额将简单地平均分配到每一期，一期为一个月。

	A
1	=db.query("select * from 分期付款表")
2	=A1.news(总期数;编号,~:期数,after@m(起始日,~-1):还款日,总金额/总期数:金额)

news：计算序列字段值合并生成新序表，一行变多行。

字串与日期-字串（一）

实际需求

拼串任务：设有**学生表**结构为：班级、姓名、性别；

要按班级分组后将男生和女生分别列成**逗号分隔的串**，串成员按姓名排序

	A
1	=db.query("select * from 学生表")
2	=A1.group(班级; ~.select(性别=="男").(姓名).sort().concat@c():男生, ~.select(性别=="女").(姓名).sort().concat@c():女生)

集算器有集合数据，无需专门针对分组的拼串函数，可以随意组合出各种运算。

字串与日期-日期（一）

实际需求

设有旅行记录表结构为：人员、开始日、结束日、…；

现在要统计这批记录中哪5天正在旅行的人数最多。

	A
1	=db.query("select 开始日,结束日 from 旅行表")
2	=A1.conj(periods(开始日,结束日)).groups(~:日期,count(1):人次)
3	=A2.sort(人次:-1).to(5)

集算器提供日期拆分函数periods，完成这个问题就很轻松。

THANKS

创新技术 推动应用进步