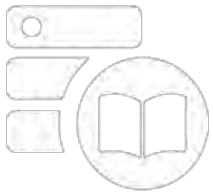
Gdevops

全球敏捷运维峰会

数据过程化处理的敏捷方法

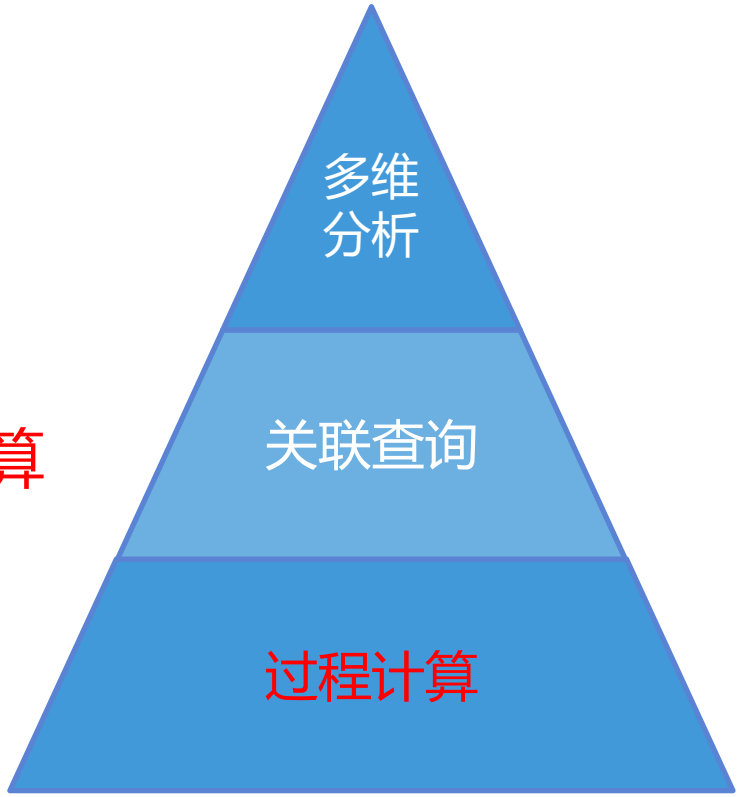
演讲人：张政宏

目录
contents



数据需求的三个层次

- 许多工具能解决好多维分析
- 少量工具能部分处理关联查询
- 用户的需求点大部分在过程计算



过程计算

- 过程计算的普遍性
  - *销售额占前一半的大客户都有哪些？*
  - *这个月内连涨3天的股票，第4天还继续上涨的比率有多大？*
  - *哪些半年不出单的客户在更换了销售人员后半年就出单了？*
- 最复杂的计算是过程计算
  - SQL不提倡分步计算
  - 多维分析、关联查询都不能解决过程计算

自助之外，需技术人员协助

- 技术人员协助的必要性
  - 抽象算法
  - 数据量大，无法导出
  - 数据源杂，不一定是数据库，难以做关联查询
  - 数据集太多，都导出太繁琐

处理技术方面的常见问题

数据处理操作化

文本先入库后计算

ETL变成了LET

SQL难写

封闭体系、常规数据类型、欠缺过程处理

存储过程耦合、难调试、难管理

高级语言的问题

Java缺结构化类库；代码冗长、不可复用

动态结构、描述能力弱；工作量大

工程转化的难题

数据库、Hadoop；体系结构臃肿

工程语言和开源工具；可集成性差

做到敏捷需要考虑的要点

工具选择和工程转化

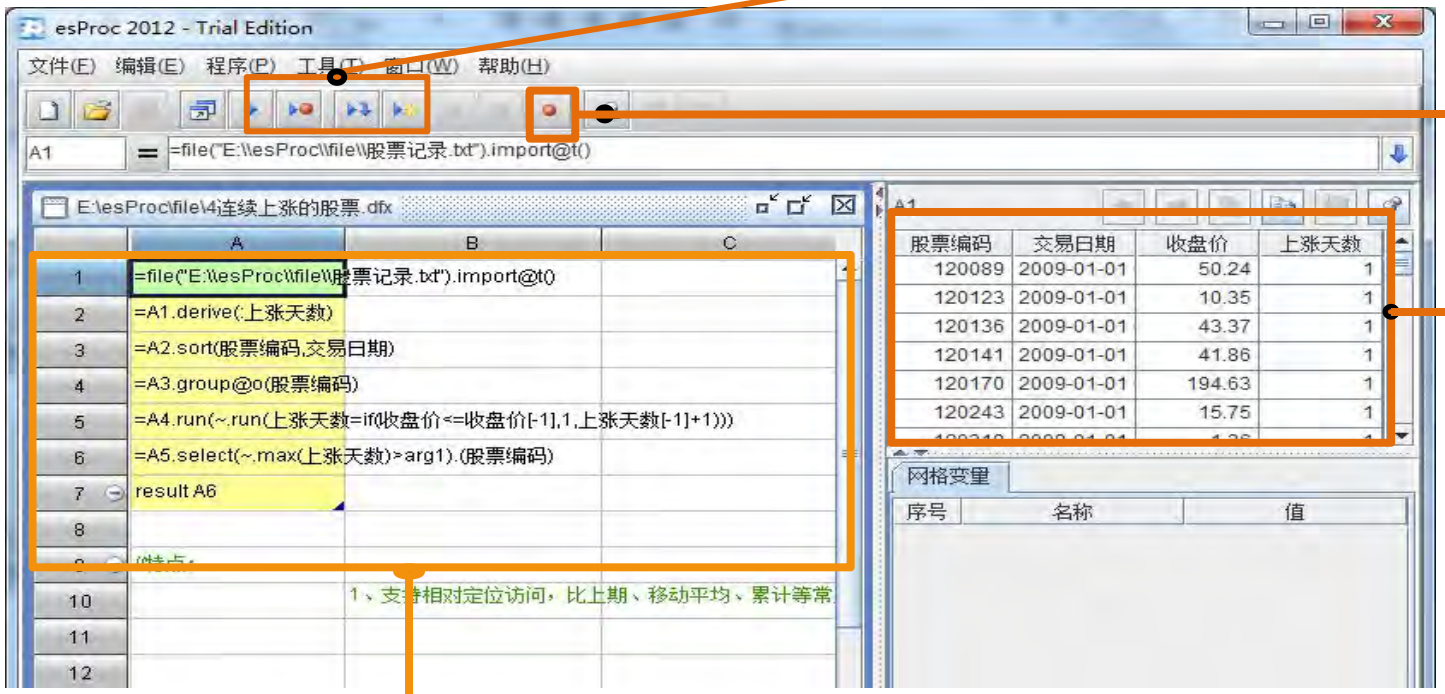
数据模型和算法

处理规模和性能

开发环境即装即用，配置简单，功能完善

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

网格式代码编写风格，比R、perl和python更直观；可命令行调用，可定时计算

丰富的外部数据接口

- 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

简洁易懂的代码语法体系

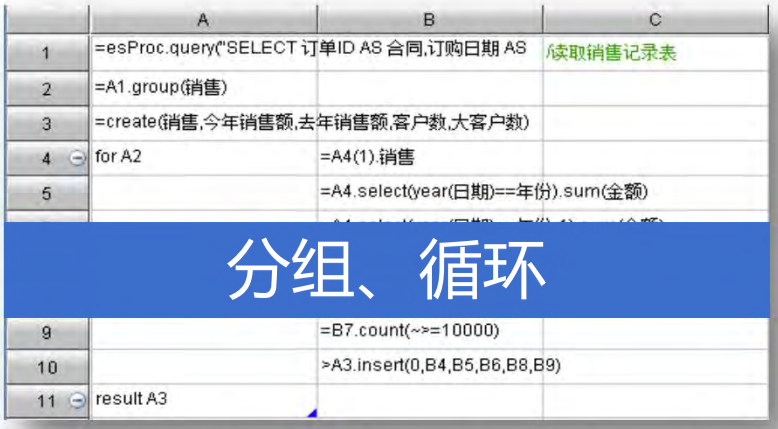
- 设计目标就是为了解决复杂过程运算

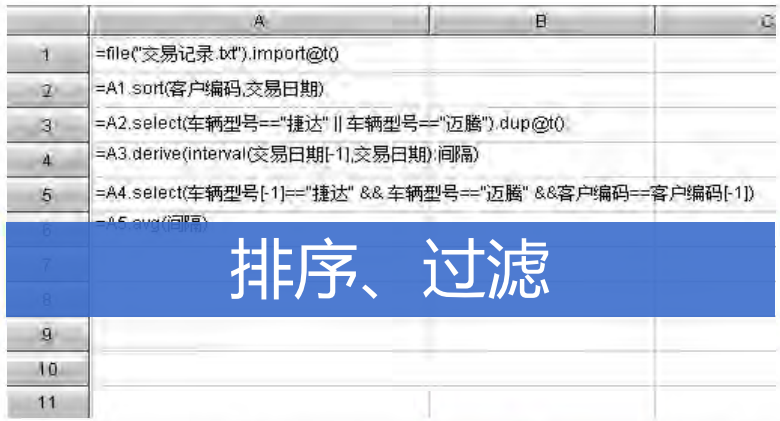
	A	B	C	D	E	F
1	=esProc.query("SELECT 订单ID AS 合同,订购日期 AS 日期,客户 ,订单金额 AS 金额,员工ID AS 销售 FROM 销售记录表 WHERE year(订购日期)=? OR year(订购日					
2	=esProc.query("select * from 员工表")					
3	>A1.run(销售=A2.select@1(编号:A1.销售))		/字段值 是记录			
4	=A1.group(销售)					
5	=create(销售,今年销售额,去年销售额,增长率,客户数,大客户数,大客户占比)					
6	for A4	=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()	/客户数			
12		=B10.count(~>=10000)	/大客户数			
13		=B12/B11	/大客户占比			
14		>A5.insert(0,B6,B7,B8,B9,B11,B12,B13)				
15	result A5					

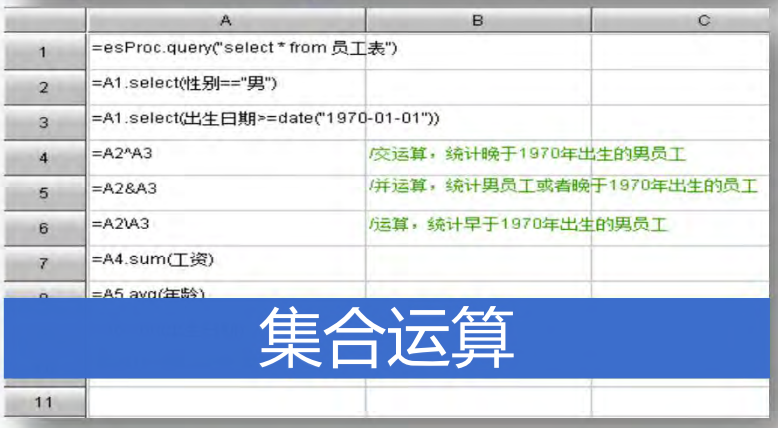
天然分步、层次清晰、直接引用单元格名无需定义变量

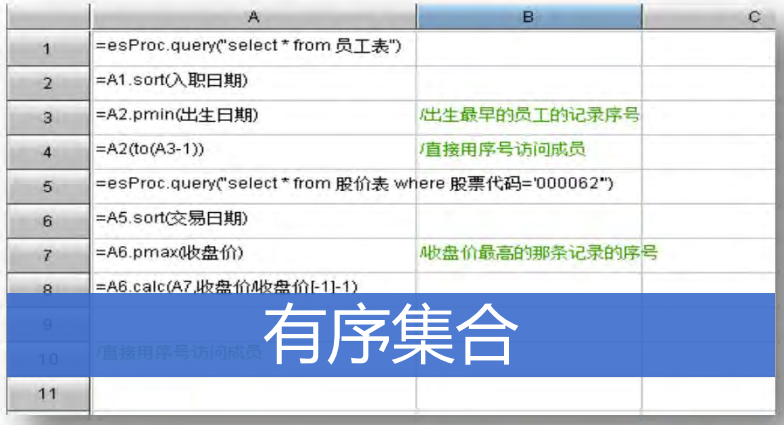
专业的结构化类库

- 专门针对结构化数据表设计









集算器与SQL对比

- 销售额占前一半的大客户

SQL

```
1  SELECT CUSTOMER, AMOUNT, SUM_AMOUNT
2  FROM (SELECT CUSTOMER, AMOUNT,
3        SUM(AMOUNT) OVER(ORDER BY AMOUNT DESC) SUM_AMOUNT
4        FROM (SELECT CUSTOMER, SUM(AMOUNT) AMOUNT
5              FROM ORDERS GROUP BY CUSTOMER))
6              WHERE 2 * SUM_AMOUNT < (SELECT
SUM(AMOUNT) TOTAL FROM ORDERS)
```

集算脚本

	A
1	=orders.groups(CUSTOMER;sum(AMOUNT):AMOUNT).sort(AMOUNT:-1)
2	=A1.derive(SUM_AMOUNT).run(SUM_AMOUNT=AMOUNT+SUM_AMOUNT[-1])
3	=A2.select(SUM_AMOUNT<=A1.sum(AMOUNT)/2)

与SQL这类本身就支持结构化计算的语言相比，集算器的语法完善了对分步计算、集合化、有序计算和对象引用等几方面的支持；对于日期和字串等运算，集算器也比大部分SQL提供了更丰富的方法。

SQL：单一数据库内

- 开发环境部署简单，存储过程调试麻烦
- 几乎没有外部数据接口，只能访问库内数据
- 语法体系对过程计算支持得很不好
- 复杂有序和集合的计算很麻烦
- 有大数据透明化计算能力，但过程复杂时性能差
- 存储过程可管理性较差

Python：外部或跨库数据/过程计算

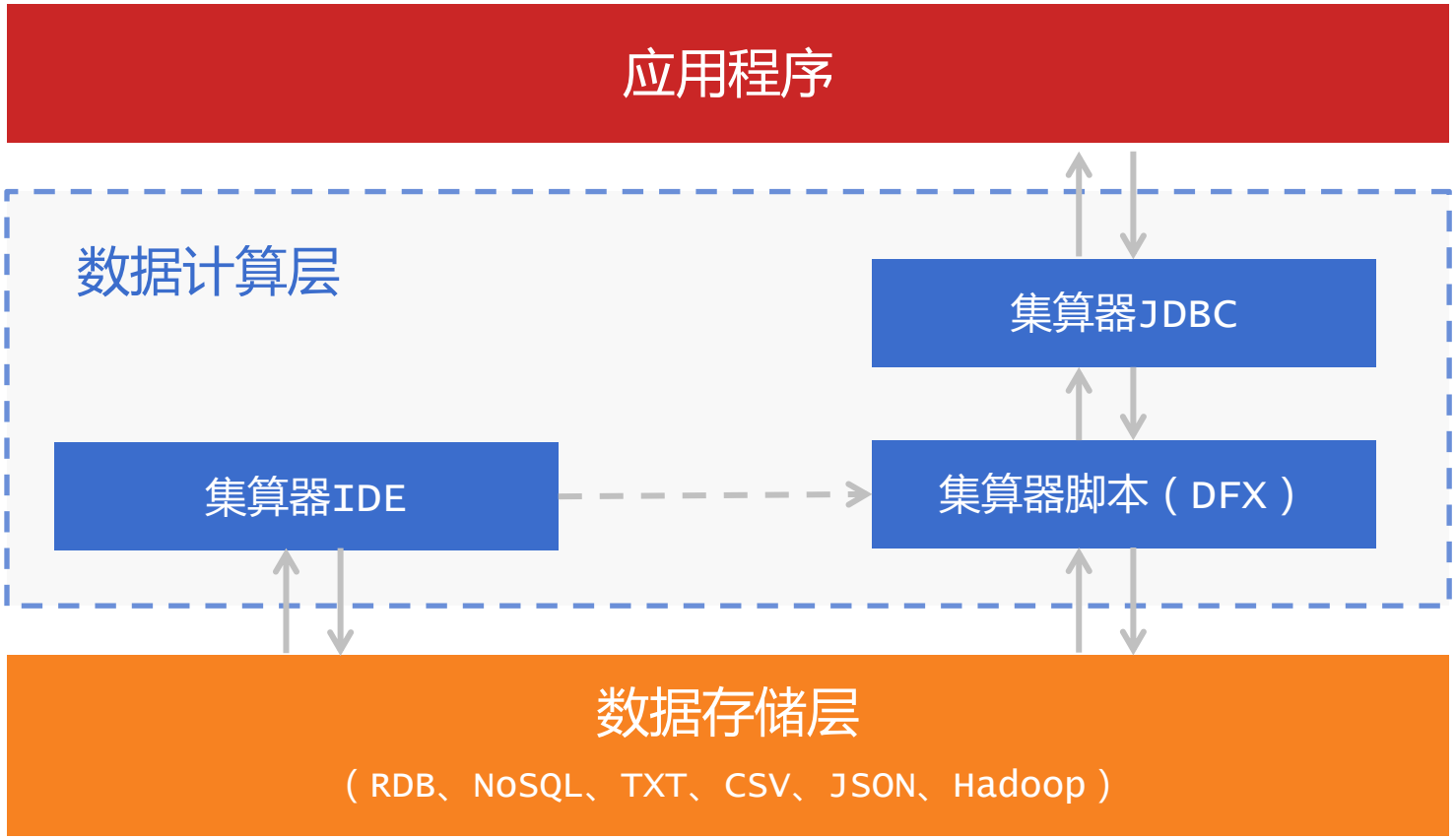
- 开发调试较简单
- 外部数据接口丰富，但开源包安装麻烦
- 语法体系不是专为结构化数据计算设计
- Pandas类库对于复杂运算支持得不够
- 缺乏自有的大数据方案
- 代码易于管理，但难于集成

脚本选择

	可管理性	开发效率	环境配置
SQL	好	低	易
存储过程	差	低	难
Python/R	好	中	中
Java	差	低	难
集算器	**好**	**高**	**易**

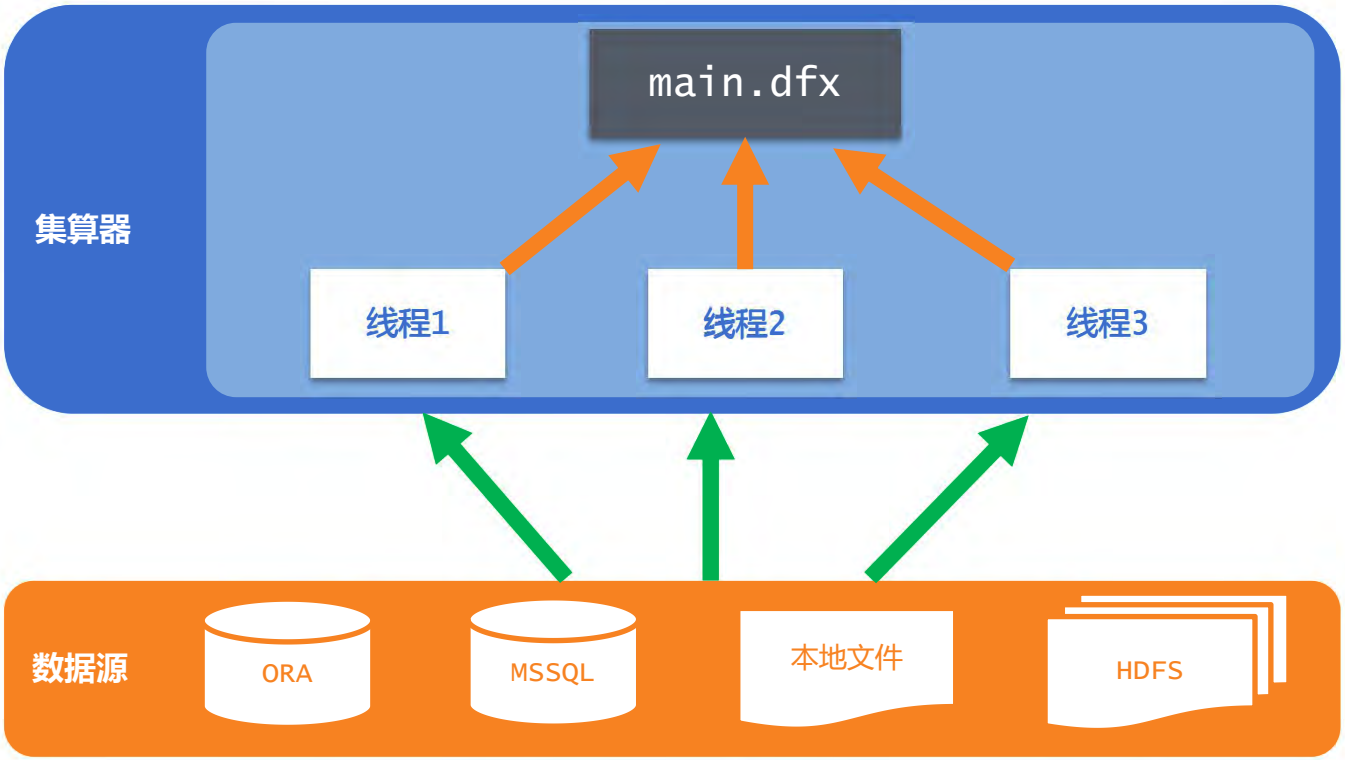
集成性：工程转化

- 应用无缝集成，代码易于管理



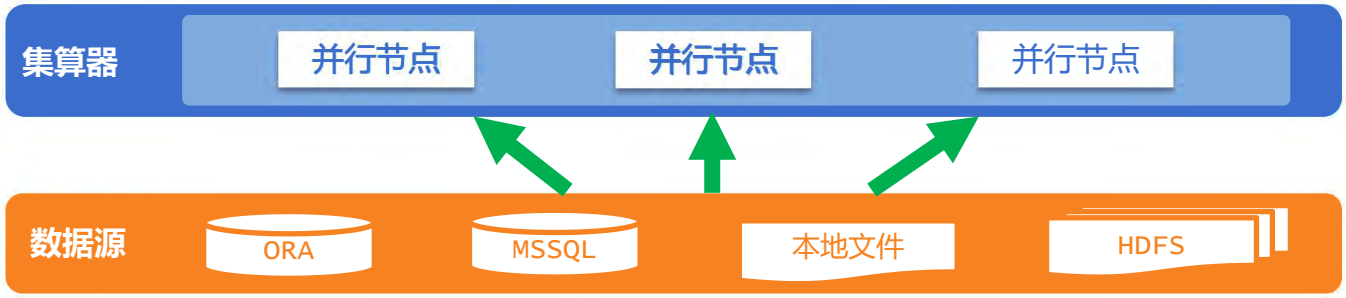
单机模式

多线程并行



多机模式

多机并行



共享数据源方式：计算分布实现，数据共享读取

	A	B	
1	=4.(“192.168.0.”/(10+~)/”:1234”)		节点机列表，4个
2	fork to(8);A1		到节点机上执行，分成8个任务
3		=hdfsfile(“hdfs:\\192.168.0.1\persons.txt”)	HDFS上的文件
4		=B3.cursor@t(;A2:8)	分段游标
5		=B4.select(gender==’M’).groups(;count(1):C)	过滤并计数
6	=A2.conj().sum(C)		汇总结果

做到敏捷需要考虑的要点

工具选择和工程转化

数据模型和算法

处理规模和性能

集算器计算模型

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

分组子集

计算任务：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

计算任务：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;
```

主子表

计算任务：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
```

有序分组

计算任务：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) 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)
```

分组有序计算

计算任务：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
```

做到敏捷需要考虑的要点

工具选择和工程转化

数据模型和算法

处理规模和性能

单机大数据工具集

1 遍历技术

2 连接解决

3 存储格式

4 使用索引

5 分段并行

遍历技术 - 延迟游标

游标概念

流式读入数据，每次仅计算一小部分

延迟计算

在游标上定义运算，返回结果仍然是游标，可再定义运算

不立即计算，最终一次性遍历和计算

	A	B
1	=file(“data.txt”).cursor@t()	/创建游标
2	=A1.select(product==“1”)	/过滤
3	=A2.derive(quantity*price:amount)	/计算列
4	=A3.sum(amount)	/实际计算

遍历技术 - 遍历复用

外存计算优化方向是减少访问量

可复用的遍历减少外存访问量

	A	
1	=file(“data.txt”).cursor()	
2	=channel().groups(;count(1))	配置同步计算
3	>A1.push(A2)	绑定
4	=A1.sortx(key)	排序，遍历过程中处理绑定计算
5	=A2.result().#1	取出绑定计算的结果，即总记录数
6	=A4.skip((A5-1)\2).fetch@x(2-A5%2).avg(key)	取出中位数记录并计算中位数

一次遍历可返回多个分组结果

遍历技术 – 有序游标

针对已有序的数据可一次遍历实现大结果集分组运算，减少外存交换

	A	
1	=file(“data.txt”).cursor@t()	
2	=A1.groupx@o(uid;count(1),max(login))	

复杂处理需要读出到程序内存中再处理

有序游标有效减少查找和遍历数量

	A	B	C
1	=file(“user.dat”).cursor@b()		/按用户id排序的源文件
2	for A1;id	…	/从游标中循环读入数据，每次读出一组id相同
3		…	/处理计算该组数据

遍历技术 - 聚合理解

从一个集合计算出一个单值或另一个集合都可理解为聚合

高复杂度的排序问题转换为低复杂度的遍历问题

	A	
1	=file(“data.txt”).cursor@t()	
2	=A1.groups(;top(10,amount))	金额在前10名的订单
3	=A1.groups(area;top(10,amount))	每个地区金额在前10名的订单

连接解决 - 区分JOIN!

1

外键维表1:N

指针化

序号化

2

同维表1:1

有序归并

3

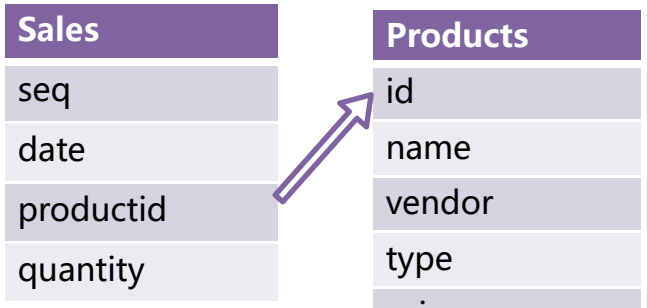
主子表1:N

有序归并

连接解决 - 外键指针化

外键需要随机小量频繁访问

内存指针查找大幅提高性能



	Java指针连接	Oracle
单表无连接	0.57s	0.623s
五表外键连接	2.3s	5.1s

	A	
1	=file(“Products.txt”).import()	读入商品列表
2	=file(“Sales.txt”).import()	读入销售记录
3	>A2.switch(productid,A1:id)	建立指针式连接，把商品编号转换成指针
4	=A2.sum(quantity*productid.price)	计算销售金额，用指针方式引用商品单价

连接解决 - 外键序号化

序号化相当于外存指针化

	A	
1	=file(“Products.txt”).import()	读入商品列表
2	=file(“Sales.txt”).cursor()	根据已序号化的销售记录建立游标
3	=A2.switch(productid,A1:#)	用序号定位建立连接指针，准备遍历
4	=A3.groups(;sum(quantity*productid.price))	计算结果

不需要再计算Hash值和比较

连接解决 - 有序归并

同维表和主子表连接可以先排序后变成有序归并

追加数据的再排序也仍然是低成本的归并计算

	A	
1	=file(“Order.txt”).cursor@t()	订单游标，按订单id排序
2	=file(“Detail.txt”).cursor@t()	订单明细游标，也按订单id排序
3	=joinx(A1:O,id;A2:D,id)	有序归并连接，仍返回游标
4	=A3.groups(O.area;sum(D.amount))	按地区分组汇总金额，地区字段在主表中，金额字段在明细子表中

存储格式 - 压缩二进制

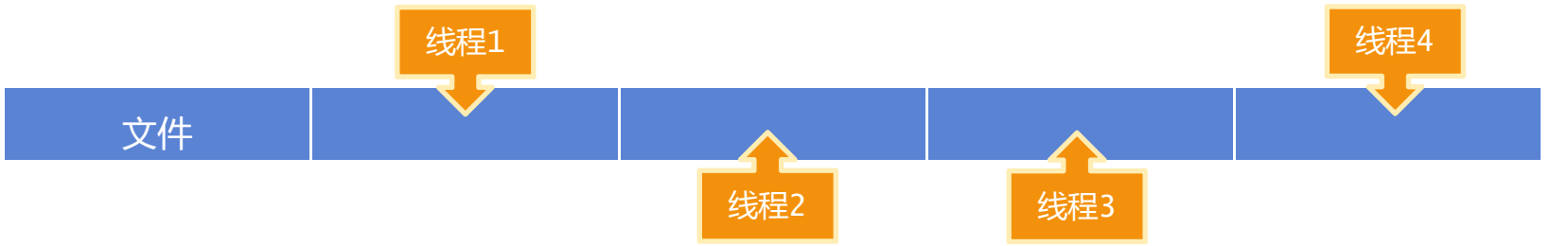
- 数据类型已存入，无须解析
- 轻量级压缩
  - 减少硬盘空间
  - 很少占用CPU时间
- 泛型存储，允许集合数据
- 可追加

使用索引

- 有序对分查找
  - 有序数据提供对分查找，快速定位
- 普通定位索引
  - 按键值找到数据
  - 两段式索引提高维护性能
- 片状索引
  - 连续记录的索引

分段并行 - 文本分段

并行处理需要将数据文件分段，每个线程处理一段



按行分段 ✗　　简单按字节分段 ✗　　去头补尾的字节分段 ✓

文本并行解析

	A	
1	=file(“data.txt”).cursor@tm(amount)	/定义并行取数的游标（并行）
2	=A1.groups(;sum(amount):amount)	/遍历游标汇总amount（串行）

分段并行 - 倍增分段

分段数足够大，以适应变化的并行数

每段记录数相对平均

数据追加时不必重写所有数据，保持连续性

	A	B	
1	=file(“data.bin”)		
2	fork 4	=A1.cursor@b(amount;A2:4)	
3		=B2.groups(;sum(amount):a)	
4	=A2.conj().sum(a)		

追加前	追加后
1	1
2	
3	2
4	
5	3
6	
7	4
8	
…	
1023	512
1024	
	513
	…

分段并行 - 有序对位分段

有序数据的分段点要落在组边界上才能分段并行

	A	
1	=file(“userlog.txt”).cursor@t()	原始数据游标
2	=A1.sortx(id)	按id排序
3	>file(“userlog.dat”).export@z(A2;id)	写成按id分段的文件

同维表、主子表需同步分段

	A	
1	=file(“Order.txt”).cursor@t()	
2	=A1.sortx(id)	按id排序
3	>file(“Order.dat”).export@z(A2;id)	写成按id分段的文件
4	=file(“Detail.txt”).cursor@t()	
5	=A4.sortx(id)	按id排序
6	>file(“Detail.dat”).export@z(A2;id,file(“Order.dat”),id)	和Order.dat同步按id分段

应用场景和优势

- 计算临时需求
- 报表后端计算
- 轻量级大数据解决方案（略）

计算临时需求

临时需求的特征：

- 以（准）结构化数据为主
- 大量涉及多样性的外部数据源
- 随意性，需求不可预测
- 常常涉及多步骤的过程计算
- 只做一次，缺乏直接可复用性
- 必要时可能转变成日常计算

计算临时需求

工具选择要点：

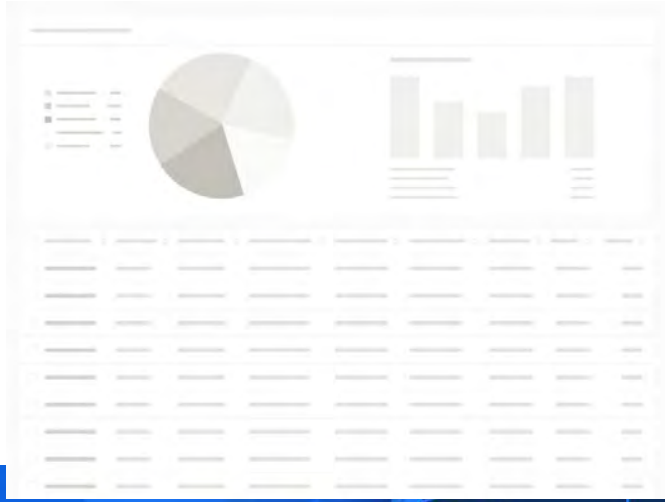
- 环境使用简单
- 开发快捷便利
- 学习低成本
- 应用可集成

报表问题

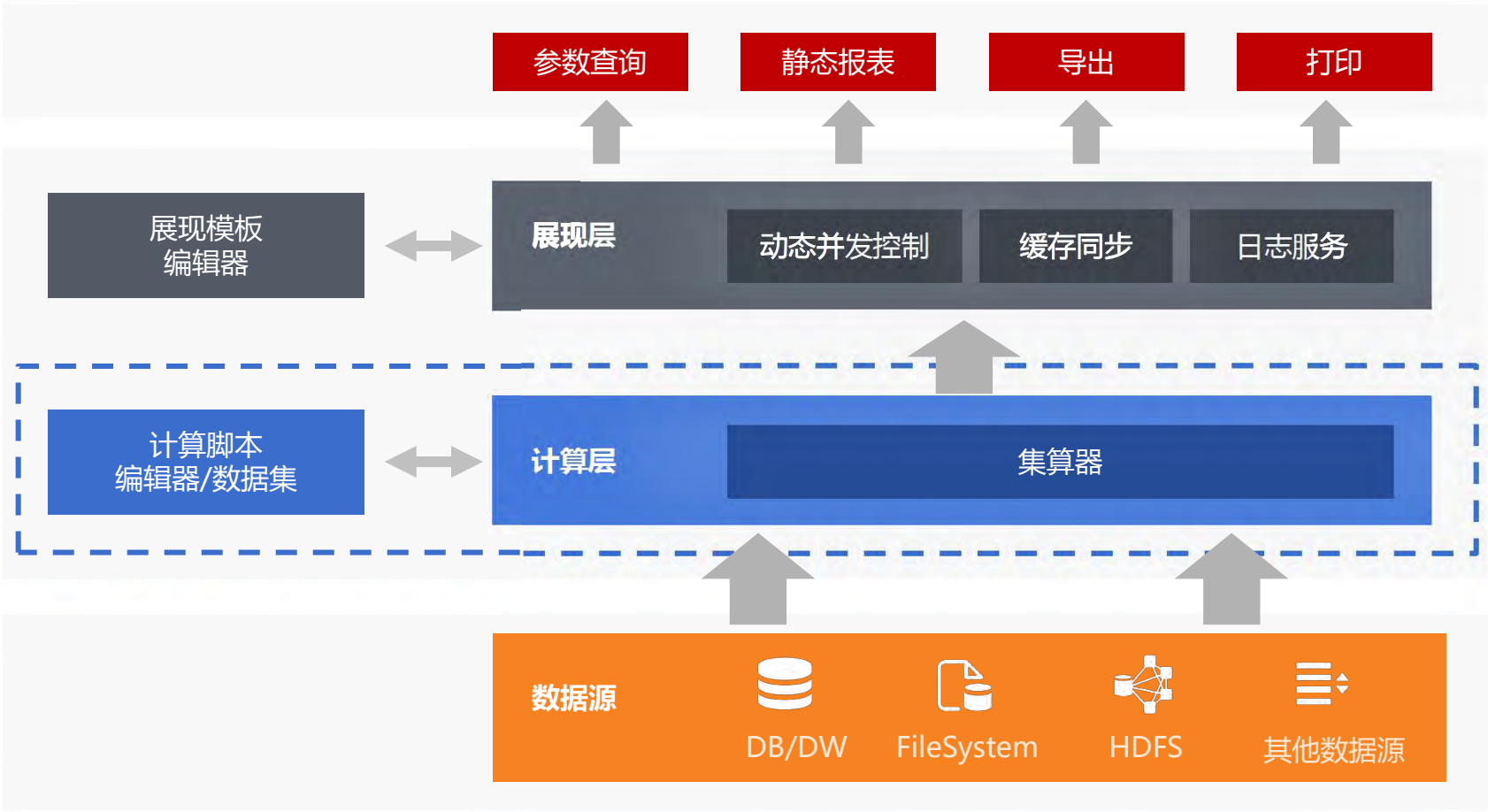
- 报表是获取数据的重要手段
  - 许多业务用户只会看报表
- 报表业务的不稳定是常态
  - 要建立长期应对机制
- 报表的困难点在于数据源
  - 占用大量开发时间

报表模块/平台的目标

- 应对报表的业务不稳定性
  - 与应用程序脱耦
  - 提高开发效率
  - 降低人员要求



引入报表数据计算层


参数查询
静态报表
导出
打印
展现模板
编辑器
展现层
动态并发控制
缓存同步
日志服务
计算脚本
编辑器/数据集
计算层
集算器
数据源
DB/DW
FileSystem
HDFS
其他数据源

集算器的优势

- 应对报表的业务不稳定性，集算器的手段和达到的目标

减少存储过程

存储过程的目的

数据整理

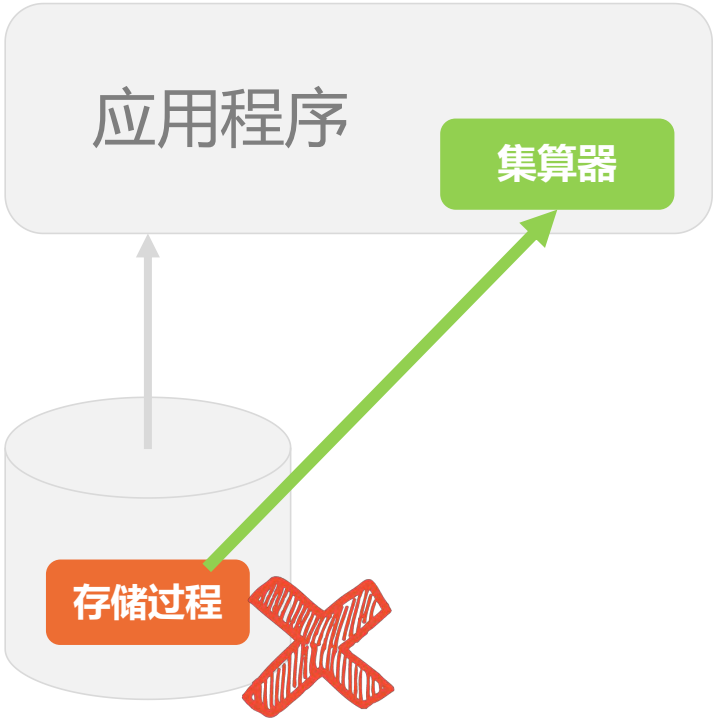
呈现准备

存储过程的问题

应用内与应用间耦合

安全性与易管理性

库外计算替代存储过程



减少冗余中间表

中间表的由来

运算复杂或数据量大

再次计算的能力

中间表的问题

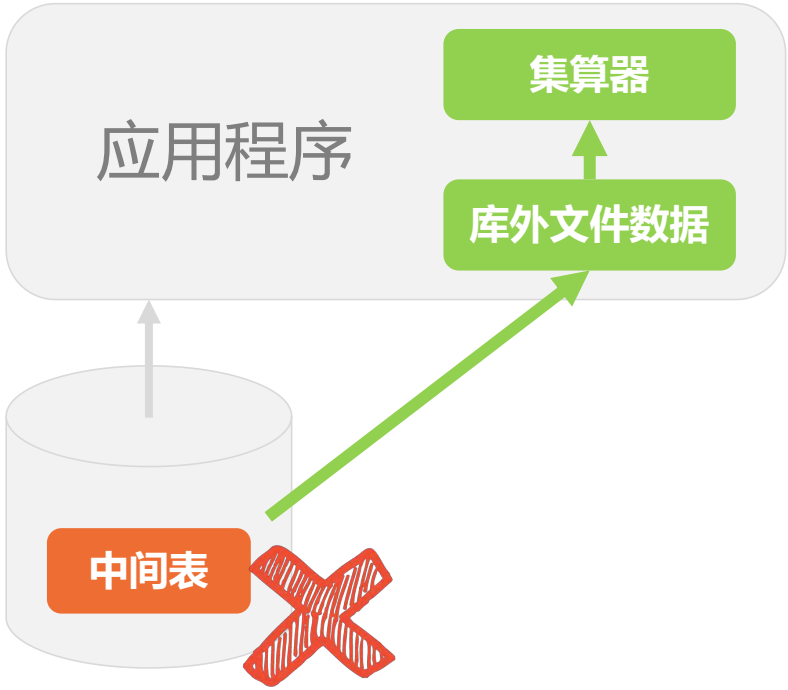
数量众多占用数据库资源

线性结构导致管理困难

库外计算将中间数据外置

计算不依赖于数据库，其它能力不需要

绑定应用、树状结构、易于管理



优化执行路径

复杂SQL的执行路径难以控制

库外计算优化SQL执行路径

自由控制执行步骤

部分运算移至库外进行

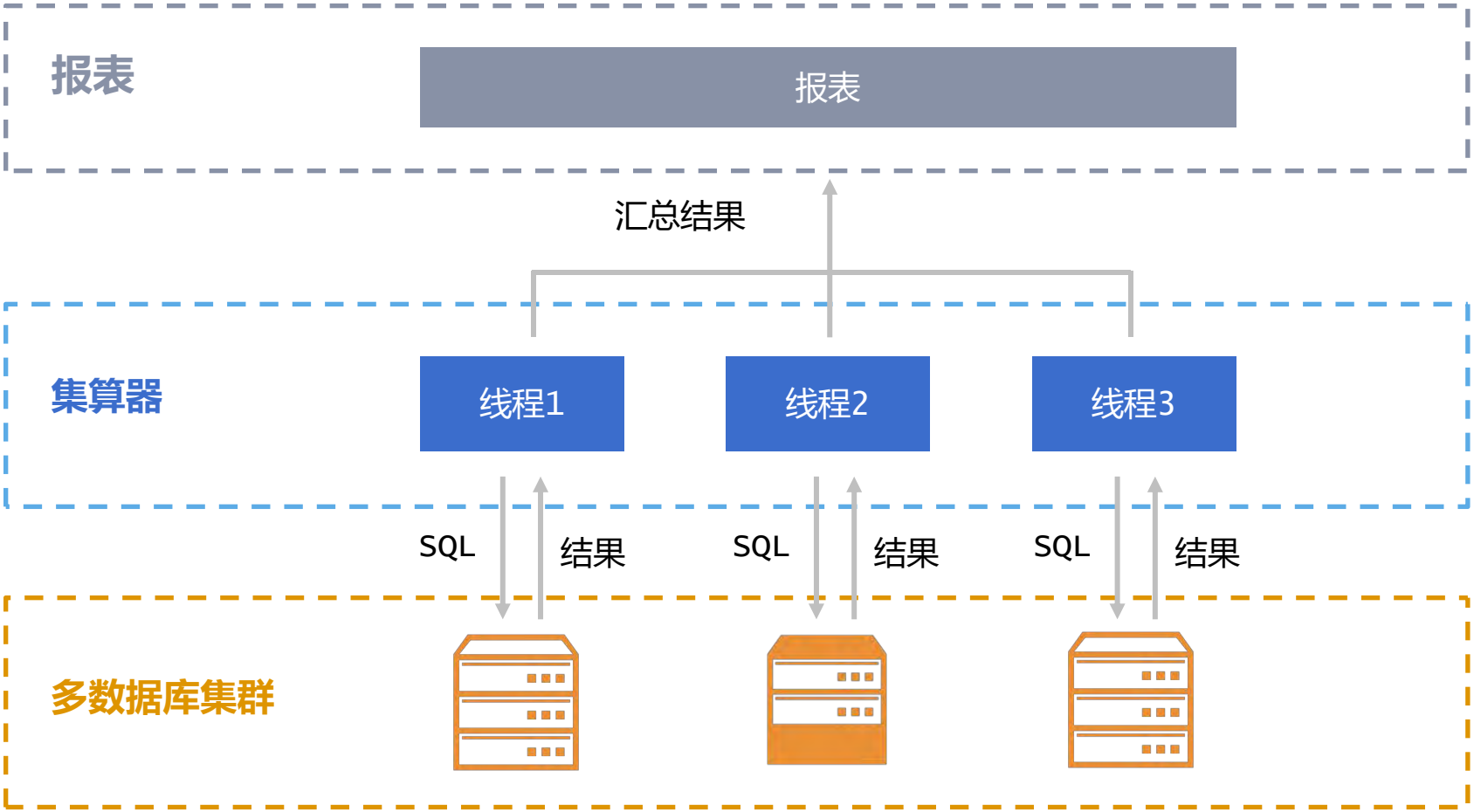
并行取数

JDBC性能瓶颈

计算引擎多线程取数

	A	B	C
1	fork 4	=connect(db)	/分4线程
2		=B1.query@x(“select * from T where part=?”,A1)	/分别取每一段
3	=A1.conj()		/合并结果

多数据源计算汇总



T+0报表查询

T+0问题

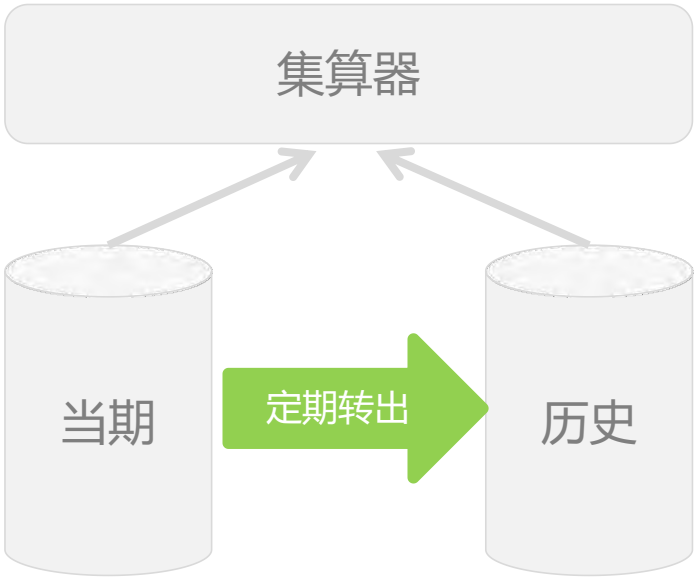
交易一致性要求关系数据库

历史与当期同库，数据量太大

历史与当期异库，跨库计算困难

库外计算实现并行跨库计算

历史数据还可文件化



扫码关注

《数据蒋堂》公众号

- ✓ 技术干货分享
- ✓ 每周一期
- ✓ 微信直播交流

Gdevops

全球敏捷运维峰会

THANK YOU！