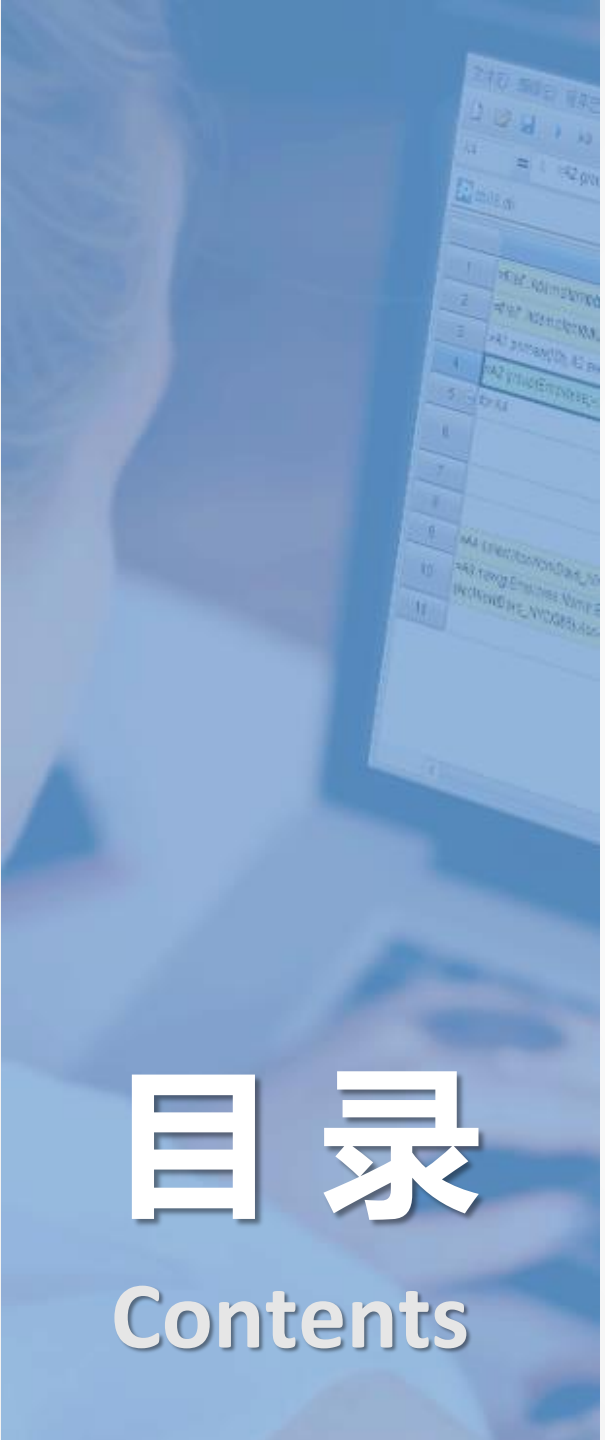
集算器

创新大数据计算引擎

润乾报表优化引擎

润乾软件出品



目录

Contents

现状分析

报表业务稳定性差，开发没完没了

这是常态，要试图适应而不能解决，建立长期应对机制

自助报表并不管用

只能解决10%-20%的需求

零编码制作报表只是一句口号

只限于呈现环节

报表开发中的二八原则

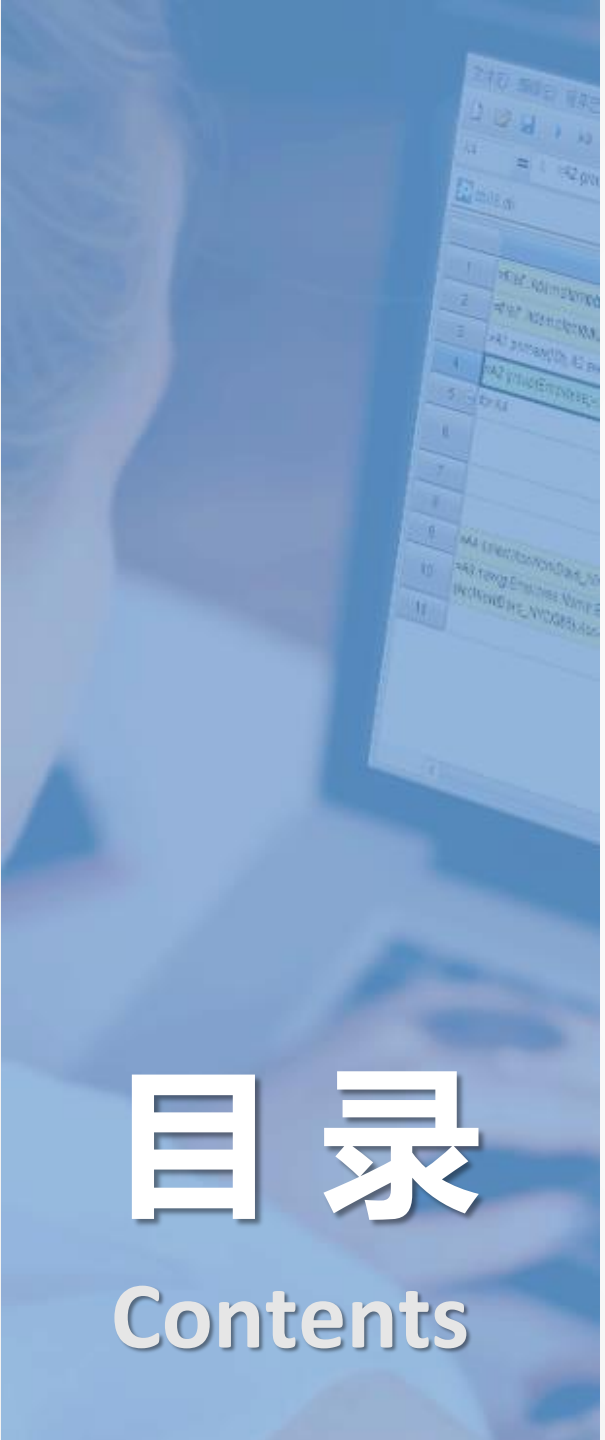
复杂报表在数量上只占二成，但工作量却占八成

现状分析

- ✓ 成熟的报表工具已经能解决呈现环节问题
- ✓ 报表开发更多的困难在数据源上
- ✓ 大多数性能问题也是数据源造成的或者需要由数据源解决
- ✓ 好的数据源处理机制还能优化应用结构

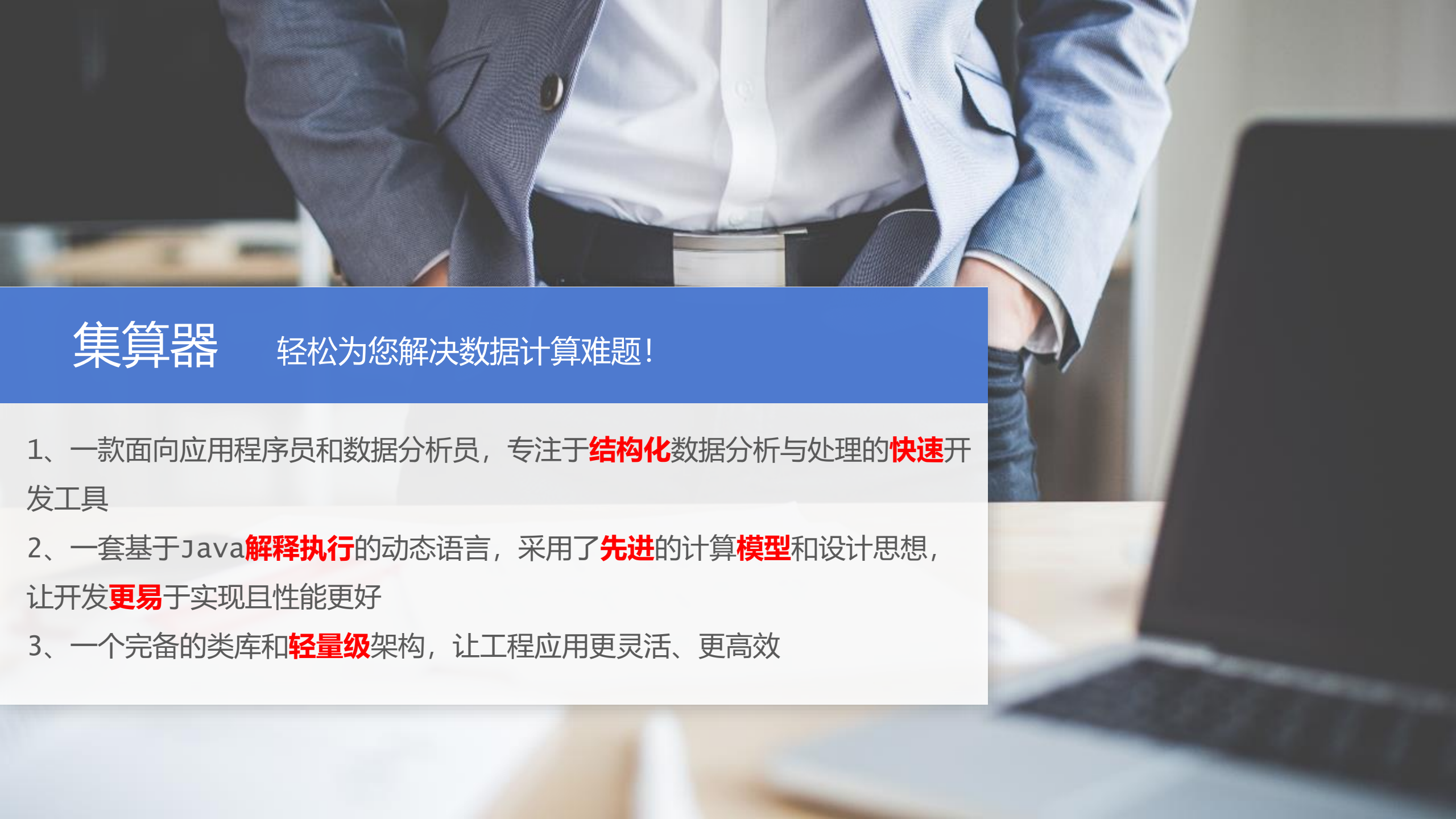
现状分析-具体场景

场景	描述与举例
报表特殊格式	报表工具无法实现，子表横向插入主表，多层动态表，横向分栏，补足空行
动态图形/水印/签章	普通报表工具能实现静态图形/水印/签章，实现动态需要自定义类，调试、扩展、维护均不方便
动态数据源/集	根据传参动态连接数据库，动态拼接SQL并对结果集容量进行一定控制
报表过程/关联计算	利用大量隐藏格保存中间结果，变相的/很费力的实现复杂的过程式计算，且性能体验很差
多样性数据源	关系型数据库，文本/文件数据类型，Hadoop，NoSQL，HTTP、ALI-OTS，...
自定义数据集	单纯SQL/存储过程一次逻辑算不完，需要借助JAVA二次加工或反复几轮调用混算后的结果集
大量存储过程/中间表	为业务查询而准备，报表应用与数据库耦合度高、管理混乱、数据库I/O瓶颈，扩容成本高
报表慢/内存溢出	日积月累，数据量上来了，报表访问越用越慢，经常内存溢出，甚至导致服务器宕机
多源跨库计算	实现跨多种类型数据源查询困难，且不分同/异构。比如：Oracle+MSSQL、数据库+Excel
T+0混合计算	要求数据库具有跨库运算能力，但实施复杂度较高，性能较低；当数据库类型不同时难以实现
大清单列表	清单式千万行大报表展现/打印/导出，用户体验差，甚至内存溢出，采用数据库分页方式性能差
离线ETL辅助	自己编码太累，专业etl工具来干又过于浪费，并且有一定的局限性、不够灵活、成本过高等
报表数据准备困难	报表工具解决展现层，但计算层还需编程(Java/复杂SQL/存储过程等)，且需专业程序员
应用耦合度高	为报表数据计算编写的代码，和应用系统的其他部分耦合在一起。一旦调整，且需专业程序员
SQL/存储过程难写	较复杂的计算逻辑，SQL/存储过程难写、难调试，低成本维护人员没办法自主完成
中间件	是否要求具有良好的可管理性、集成性、扩展性、开放性?



目录

Contents



集算器

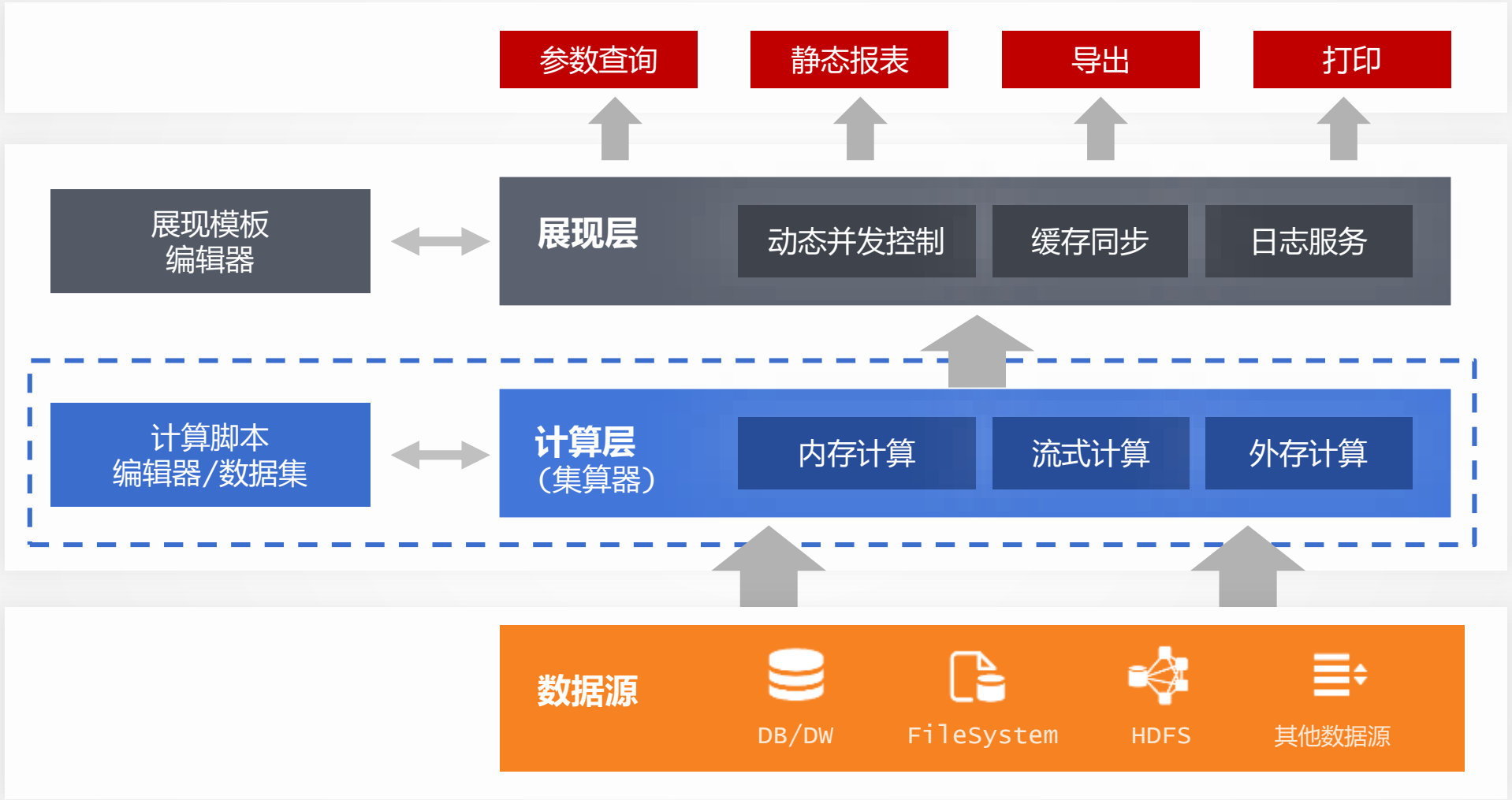
轻松为您解决数据计算难题！

1、一款面向应用程序员和数据分析员，专注于**结构化**数据分析与处理的**快速**开发工具

2、一套基于Java**解释执行**的动态语言，采用了**先进**的计算**模型**和设计思想，让开发**更易**于实现且性能更好

3、一个完备的类库和**轻量级**架构，让工程应用更灵活、更高效

引入数据计算层-集算器


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)
内存计算
流式计算
外存计算
数据源
DB/DW
FileSystem
HDFS
其他数据源

引入数据计算层-工具化、独立化

开发工具化

1、不仅呈现层要工具化，报表数据计算层也要工具化，降低对开发人员要求。

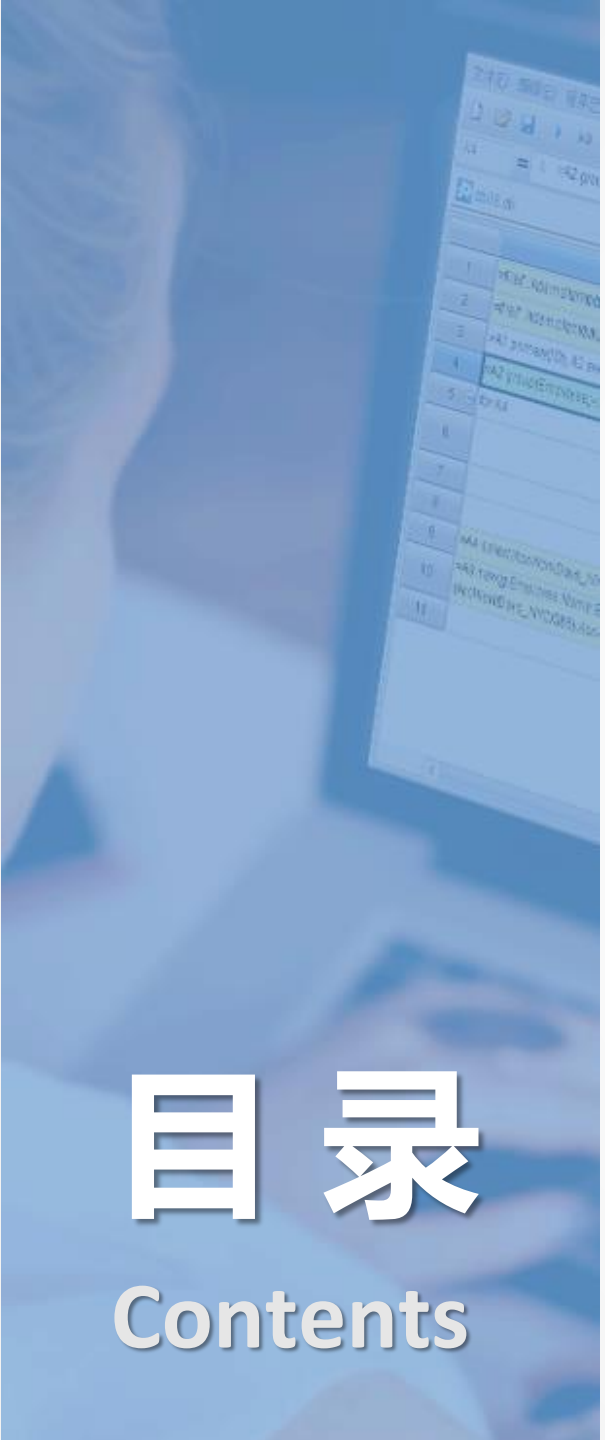
2、不必做复杂的环境配置(数据源等)，可编写简单代码实现复杂计算。

3、多样性数据源(比如EXCEL/文本)，也必须支持简单脚本计算。

模块独立化

1、数据计算层也要和呈现层一样，完全和应用系统解耦合，实现独立维护。

2、报表需求变更和新增时，仅修改报表模块就可以了，不会影响应用系统其他部分。



目录

Contents

核心优势

降低开发难度

提高运算性能

优化应用结构

大数据报表

过程计算

报表采用直观的状态式计算

不合适进行过程计算

过程计算在复杂报表中很普遍

采用隐藏格

采用Java或存储过程

过程计算举例

销售额占前一半的大客户

	A	B	C
1	客户	金额	=ds.sum(金额)/2
2	=ds.select(客户)	=ds.金额	=C2[-1]+B2
3	大客户数量	=count(B2{C2[-1]<C1})	
4	平均销售额	=avg(B2{C2[-1]<C1})	

隐藏格：C列要隐藏，且在第2行设置显示条件为C2[-1]<C1。不能用更简单地C2<=C1，否则刚跨过一半销售额的那一行会出错

条件会被计算多次，还要用到格集过滤等报表工具特有能力

舍位平衡

四舍五入后明细与合计不一致，需要根据合计重新舍入明细

集算器实现

先准备出数据集

	A	B
1	=db.query(“select 客户,金额 from 客户销售 order by 金额 desc”)	
2	=A1.sum(金额)/2	=0
3	=A1.pselect((B1+=金额)>=A2)	return A1.to(A3)

再做报表就很简单

	A	B
1	客户	金额
2	=ds.select(客户)	=ds.金额
3	大客户数量	=ds.count()
4	平均销售额	=ds.avg(金额)

多了一步，但更为清晰；

任何报表工具都可以完成

比JAVA的优势

集算器采用集合化语法，代码要比没有直接提供结构化计算的JAVA更加短小

写的更快更短

- 集算器基于Java提供了更高层的类库和方法

容易理解和排错

- 伪实代码的比例大约是只有1:1.5，大多数报表数据准备算法可以在一个屏幕内显示出来
- 一个页面内能看到更多代码，能更完整地理解代码的含义与排错

比SQL/存储过程的优势

销售额占前一半的大客户

```
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)
```

分步计算，调试开发更方便

离散性支持下实现更彻底的集合运算和有序运算

多样性数据源支持

报表工具的计算能力和容量都难以胜任多样性数据源

计算层处理多样性数据源

无需入库，减少工作步骤

支持多层数据格式，减少开发工作量

动态数据源/集

动态数据源

根据参数决定连接的数据库　`${pds}.query("select * from T where F=?",pF)`

动态数据集

动态SQL，需要程序逻辑来拼接

	A	
1	=sums.array().("sum("+~+") as "+~).string()	/把a,b变成sum(a) as a,sum(b) as b
2	=db.query("select G,"+A1+" from T group by G")	

结果集容量控制

	A	B	
1	=db.cursor("select * from T")	=A1.fetch(1000)	
2	if B1.fetch@0(1)	>B1.insert(0,"继续")	/未完成则插入标记
3	>A1.close()	return B1	

特殊格式

解决报表工具不支持的格式

- 横向分栏

员工号	姓名	部门	员工号	姓名	部门	员工号	姓名	部门
1	Rebecca	R&D	2	Ashley	Finance	3	Rachel	Sales
4	Emily	HR	5	Ashley	R&D	6	Matthew	Sales
7	Alexis	Sales	8	Megan	Marketing	9	Victoria	HR
10	Ryan	R&D	11	Jacob	Sales	12	Jessica	Sales
13	Daniel	Finance	14	Alyssa	Sales	15	Alexis	Sales
16	Christopher	Production	17	Hannah	Marketing	18	Jonathan	Administration

	A	B	C
1	=db.query("select a,b,c from T ")		
2	=A1.step(3,1)	=A1.step(3,2)|[null]	=A1.step(3,3)|[null]
3	=A2.derive(B2(#).a:a2,B2(#).b:b2,B2(#).c:c2,C2(#).a:a3,C2(#).b:b3,C2(#).c:c3)		

- 补足空行

	A	
1	=db.query("select * from T")	
2	=pn-A1.len()%pn	/计算应补的行数
3	=A1|if(A2!=pn,A2*[null])	/补充空行数后的结果集

核心优势

降低开发难度

提高运算性能

优化应用结构

大数据报表

报表的性能问题

大部分是数据源导致的

报表本身一般是个小数据任务

数据进入报表之前就很慢，　优化报表环节意义不大

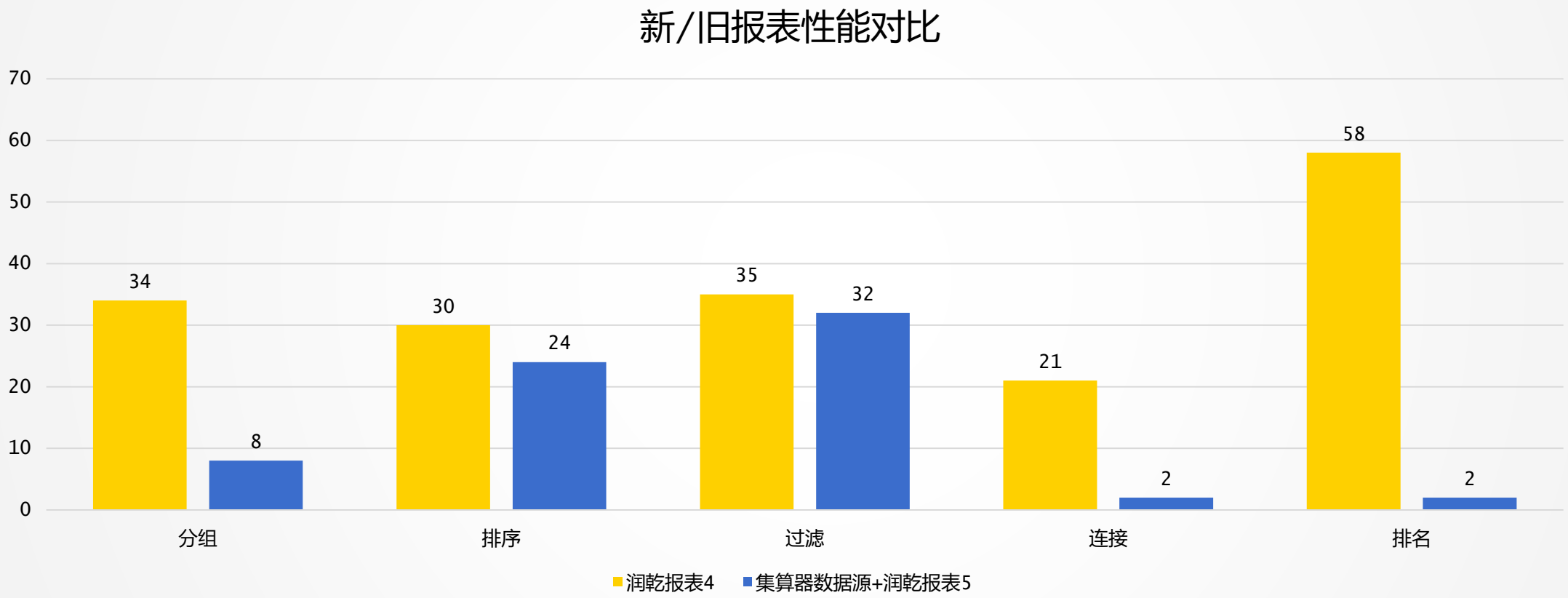
报表环节的性能也可由数据源环节解决

采用不同的计算方式

减少隐藏格

数据传输、可控缓存、内存共享

替代报表计算 – 高效算法



集算器中多数据源对齐采用Hash算法，优于报表中的遍历方式；复杂度由平方级降为线性级

这个优势可以服务于任何报表工具

多源分片实例

报表中有4个数据集，在报表单元格中完成关联，运行时间3700秒

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1(标)	POS交易情况表																
2(标)																	
3(标)	="　　会计日期:"+string(datetime(begin_date,"yyyyMMdd"),"yyyy-MM-dd")+" 至 "+string(datetime(end_date,"yyyyMMdd"),"yyyy-MM-dd")+"　　单位:元"																
4(头)		我行POS机商户交易			他行卡在我行POS机交易			我行卡在所有POS交易			我行卡在他行POS机消费				我行卡在我行POS机交易		
5(头)	机构名称	交易笔数	交易金额	手续费收入	交易笔数	交易金额	手续费收入	交易笔数	交易金额	手续费收入	消费笔数	消费金额	手续费收入		交易笔数	交易金额	手续费收入
6	=ds2.select(CM_OPUN_C	=POS_BH.sum	=POS_BH.sum(BHAMT	=G6+Q6	=POS_BH.sum	=POS_BH.sum	=POS_	=K6+O6	=L6+P6	=M6+Q6	=POS_TH.sum	=POS_TH.sum(AMT,	=N6-Q6	=ds3.select	=POS_BH.sum	=POS_	=POS_BH.select
7	合计	=sum(B6{})	=sum(C6{})	=sum(D6{})	=sum(E6{})	=sum(F6{})	=sum(G6{})	=sum(H6{})	=sum(I6{})	=sum(J6{})	=sum(K6{})	=sum(L6{})	=sum(M6{})		=sum(O6{})	=sum(P6{})	=sum(Q6{})
8(脚)			制表:			复核:											

4个结果集在集算器脚本中关联后，将结果返回给报表展现，运行时间105秒，获得35.2倍性能提升

15	>ds2.switch(C1,A14:CR_MGBR)
16	>ds2.switch(C2,B14:CR_OPUN_COD)
17	>ds2.switch(C3,C14:CR_OPUN_COD)
18	=ds2.new(CM_OPUN_COD,CM_OPUN_FLNM_CHN,~.C1.BHCNT_THCNT:BHCNT_THCNT,~.C1.BHAMT_THAMT:BHAMT_THAMT,~.C1.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1(标)	POS交易情况表															
2(标)																
3(标)	="　　会计日期:"+string(datetime(begin_date,"yyyyMMdd"),"yyyy-MM-dd")+" 至 "+string(datetime(end_date,"yyyyMMdd"),"yyyy-MM-dd")+"　　单位:元"															
4(头)		我行POS机商户交易			他行卡在我行POS机交易			我行卡在所有POS交易			我行卡在他行POS机消费			我行卡在我行POS机交易		
5(头)	机构名称	交易笔数	交易金额	手续费收入	交易笔数	交易金额	手续费收入	交易笔数	交易金额	手续费收入	消费笔数	消费金额	手续费收入	交易笔数	交易金额	手续费收入
6	=ds1.select(CM_OPUN_C	=ds1.BHCNT_	=ds1.BHAMT_THAMT	=G6+P6	=ds1.THCNT	=ds1.THAMT	=ds1.SHFY_	=K6+N6	=L6+O6	=M6+P6	=ds1.NUM	=ds1.AMT	=ds1.SHFY-P6	=ds1.BHCNT	=ds1.BHAMT	=ds1.SHFY_BH1*0.9
7	合计	=sum(B6{})	=sum(C6{})	=sum(D6{})	=sum(E6{})	=sum(F6{})	=sum(G6{})	=sum(H6{})	=sum(I6{})	=sum(J6{})	=sum(K6{})	=sum(L6{})	=sum(M6{})	=sum(N6{})	=sum(O6{})	=sum(P6{})
8(脚)			制表:			复核:										

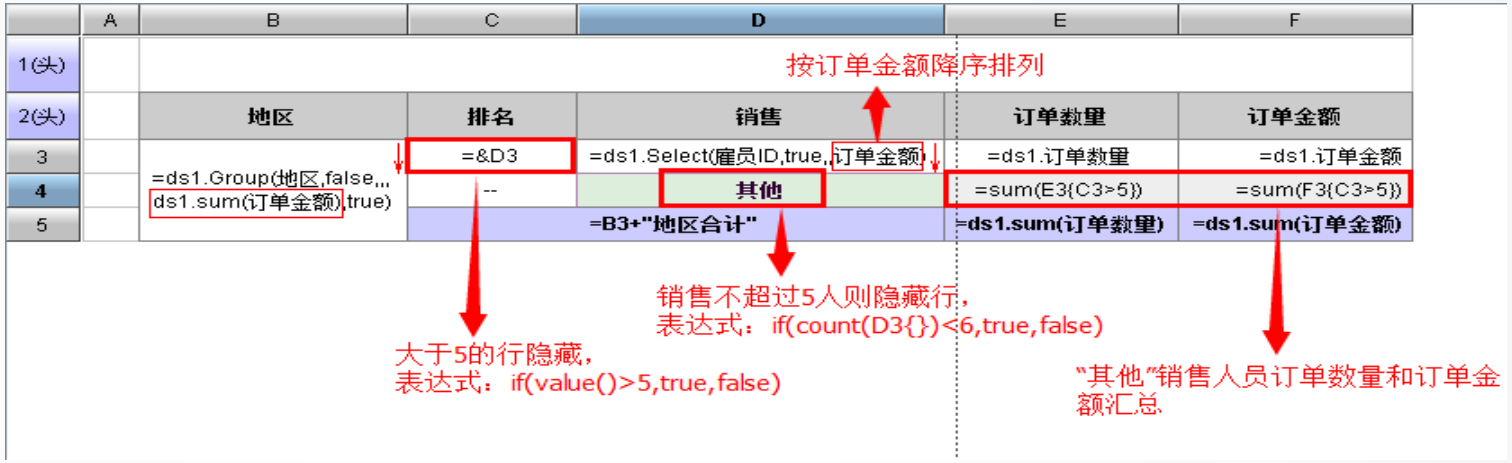
替代报表计算 – 减少隐藏格

隐藏格作用	用于保存中间结果
隐藏格缺点	单元格带有展现属性会占用过多内存，出现性能瓶颈
改进方法	✓ 在独立的计算层中准备数据，方便复用中间结果 ✓ 不涉及隐藏格和外观属性，内存使用效率更高

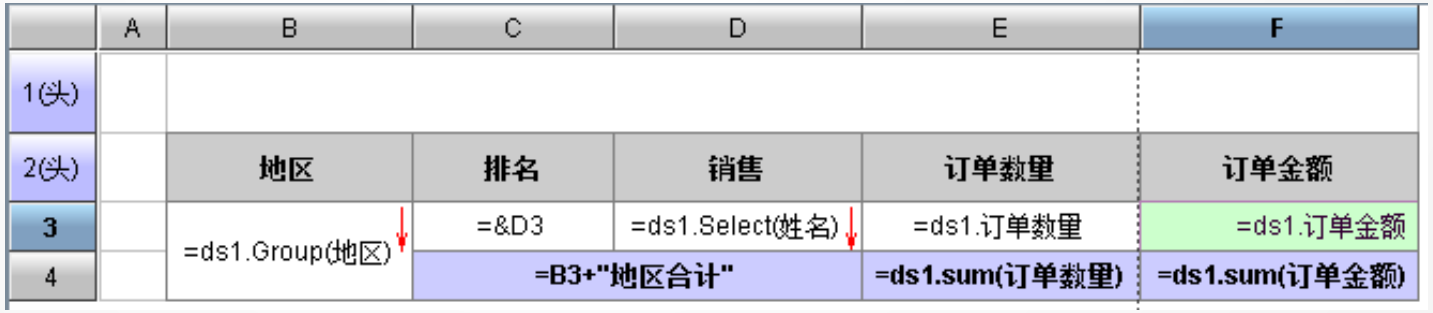
减少隐藏格实例

统计各地区前五的销售业绩，第六名以后全部归并为其他

单纯报表工具实现



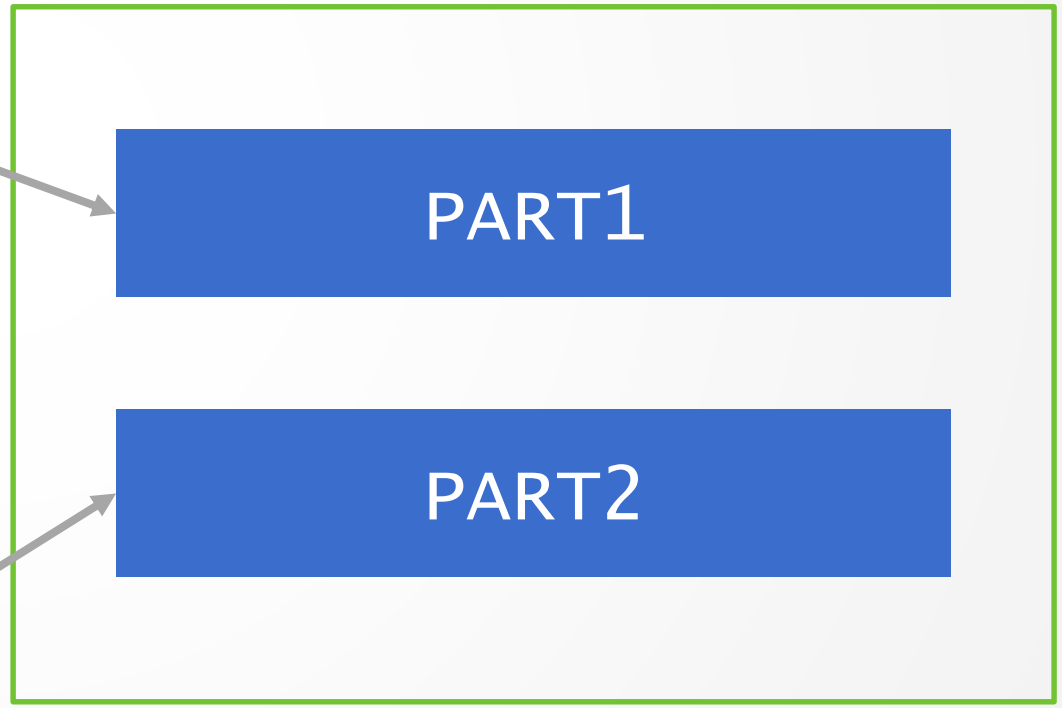
集算器+报表工具实现



控制SQL执行路径

- 数据库的透明性为用户带来方便的同时，使得优化SQL执行路径非常困难
- 集算器就可以自由控制执行路径，部分运算可以移出数据库实施，方便完成性能优化

```
1   select t.*
2    from (select *
3            from (select syb.org_abbreviation as syb,
4                         max(xmb.org_abbreviation) as
5  xmb,
                         sub.org_subjection_id as sub_id,
                  .....
298  group by l.requisition_id,
299                       l.note,
300                       l.requisition_type,
                  .....
364  left join cpl_rwdmx_view ve
365            on ve.requisition_id = a.req_id) t
```



SQL - 442秒

集算器+SQL - 41秒

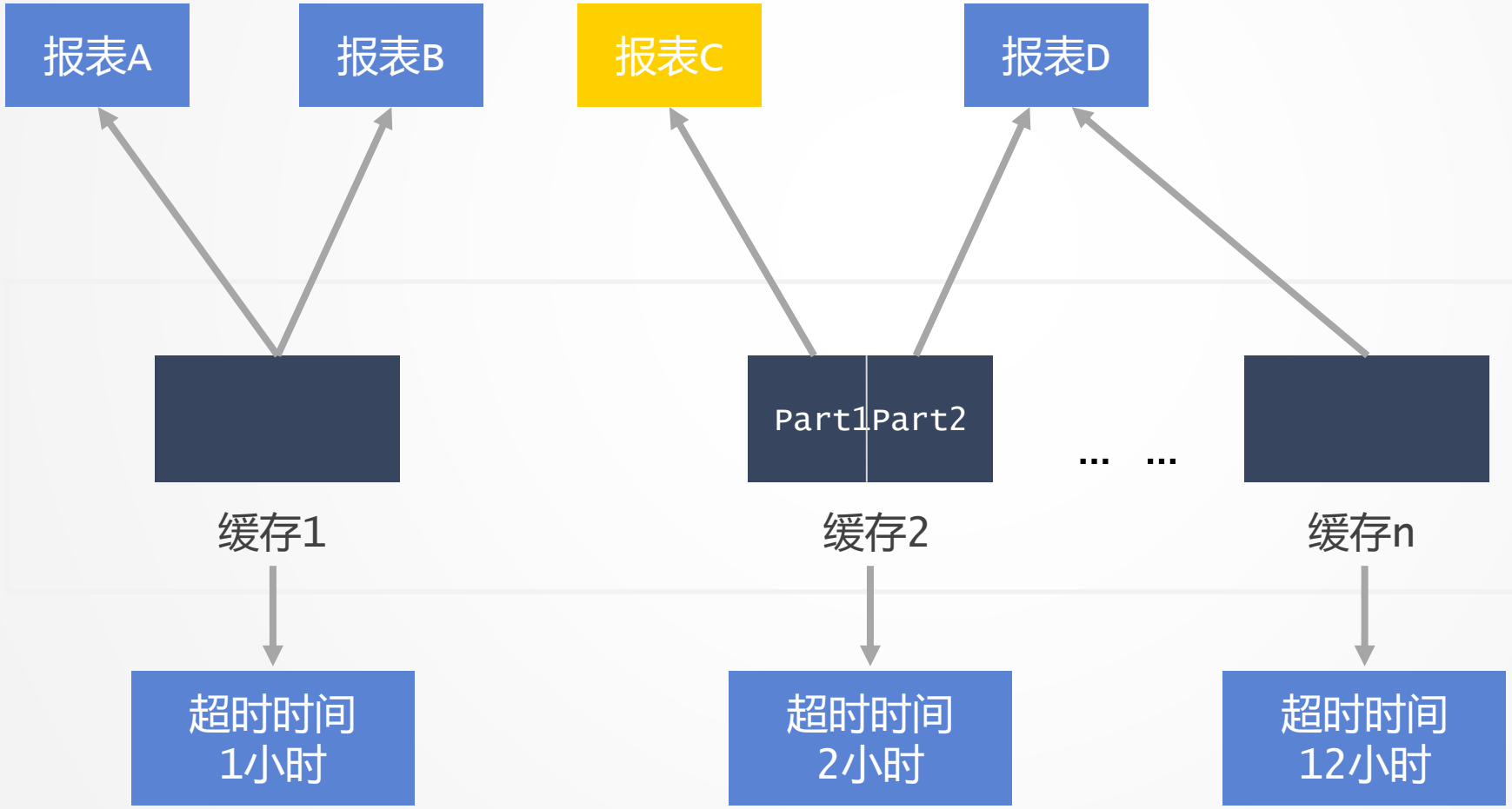
并行取数

数据库JDBC性能较差，报表性能又严重依赖于取数环节；集算器可以采用多线程并行的方式同时建立多个数据库连接从数据库分段取数，可以获得数倍性能提升

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

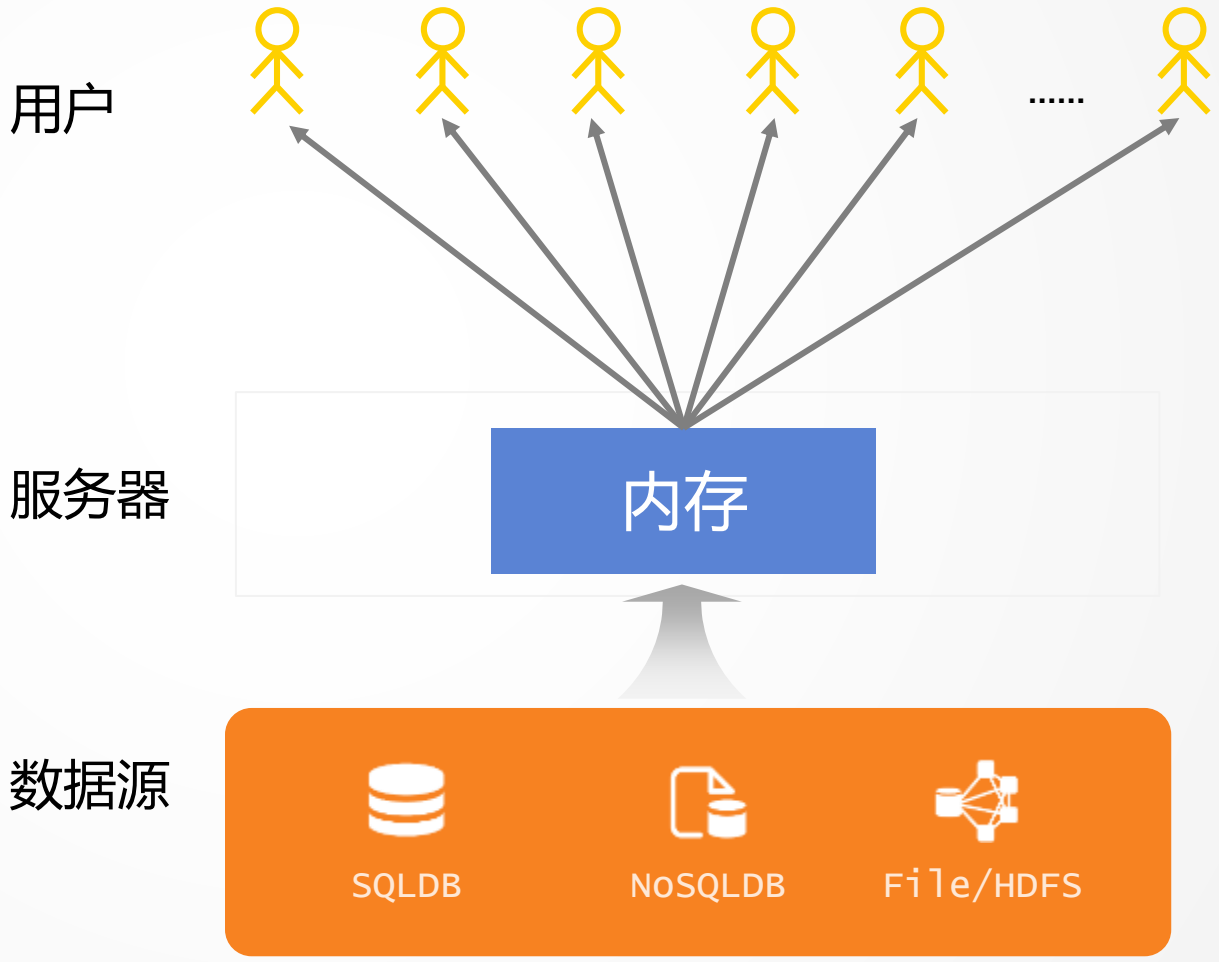
可控缓存

集算器可以实现报表的**部分缓存**、多个报表之间**缓存复用**、以及不同缓存的**不同生存周期**



内存共享

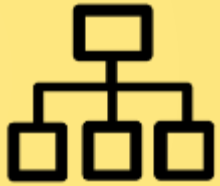
对于**高并发**报表，可以利用内存共享机制，性能更高，而且更方便并行计算



核心优势

降低开发难度

提高运算性能



优化应用结构

大数据报表

解释执行降低应用耦合度

使用JAVA和集算器准备报表数据源会有以下不同：

JAVA

模块化困难

Java程序必须和主应用一起编译打包，耦合度高

难以热切换

使用Java编写的报表数据准备算法有修改后会导致整个应用重新编译部署，很难做到热切换。

集算器

模块化简单

集算器脚本文件可以和报表模板一起管理维护，从而使报表功能模块化

容易热切换

集算器是解释执行的语言，很容易做到热切换

算法外置减少存储过程

采用存储过程实现数据准备算法，会造成报表与数据库的耦合问题

存储过程和报表的存放位置不同，导致对应难度很大

存储过程修改需要分配相应的数据库权限，存在安全隐患

存储过程容易被其他应用使用，造成多个应用间的耦合

使用集算器替代存储过程完成报表数据准备，会极大减少存储过程，算法外置后与报表模板一起存放管理，完全归属于应用本身，降低报表与应用其他部分或其他应用的耦合

数据外置减少中间表

由于数据量或计算复杂度原因，经常需要在数据库中创建中间表，中间表会带来如下问题：

数据库管理混乱

各个应用的累积大量中间表存储在线性结构的数据库中造成数据库管理混乱

数据库资源浪费

不用的中间表，仍然需要相应ETL过程向其更新数据，浪费数据库资源

将中间表数据外置存放到文件系统，便于管理，而且通过集算器获得计算能力，可以获得更高效的IO性能和计算能力，充分减少数据库中间表，梳理数据库结构

直接使用多数据源及跨库计算

集算器直接使用各类型数据源可以带来以下好处：

1、无须入库，体系结构更精简，减少中间表

2、直接取数，实时性好，减少不一致风险

3、充分利用各数据源固有的优势

核心优势

降低开发难度

提高运算性能

优化应用结构

大数据报表

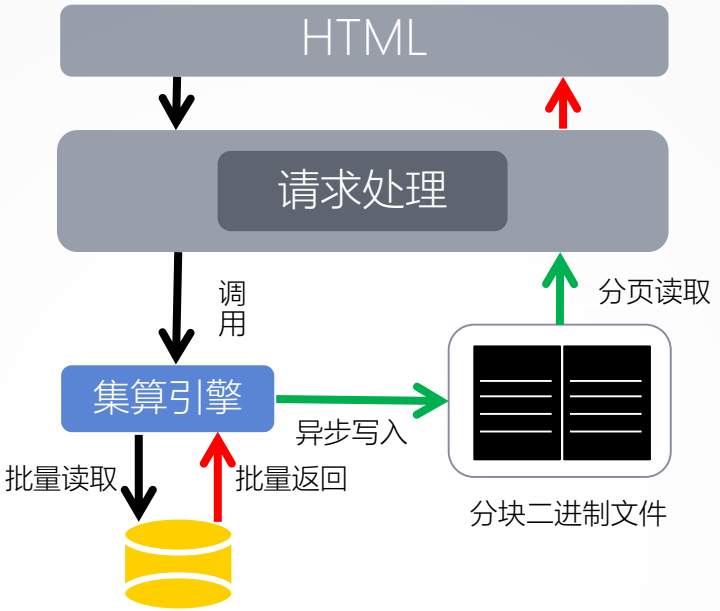
大清单报表

传统报表模式

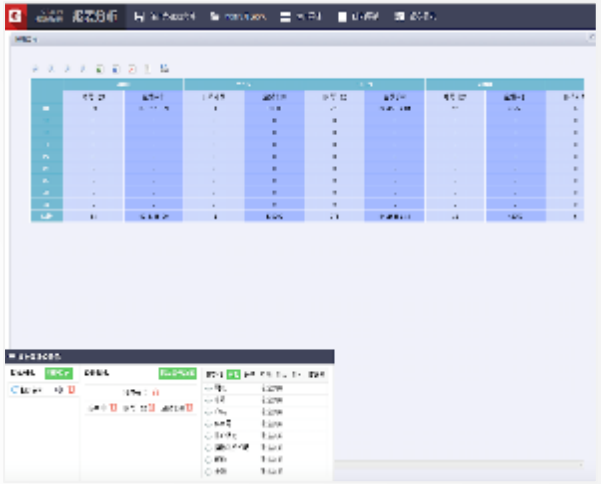
全部取出后再呈现，用户体验恶劣

全内存计算的报表可能溢出

由数据库实现分页取出，性能很差



基于大报表的在线分析

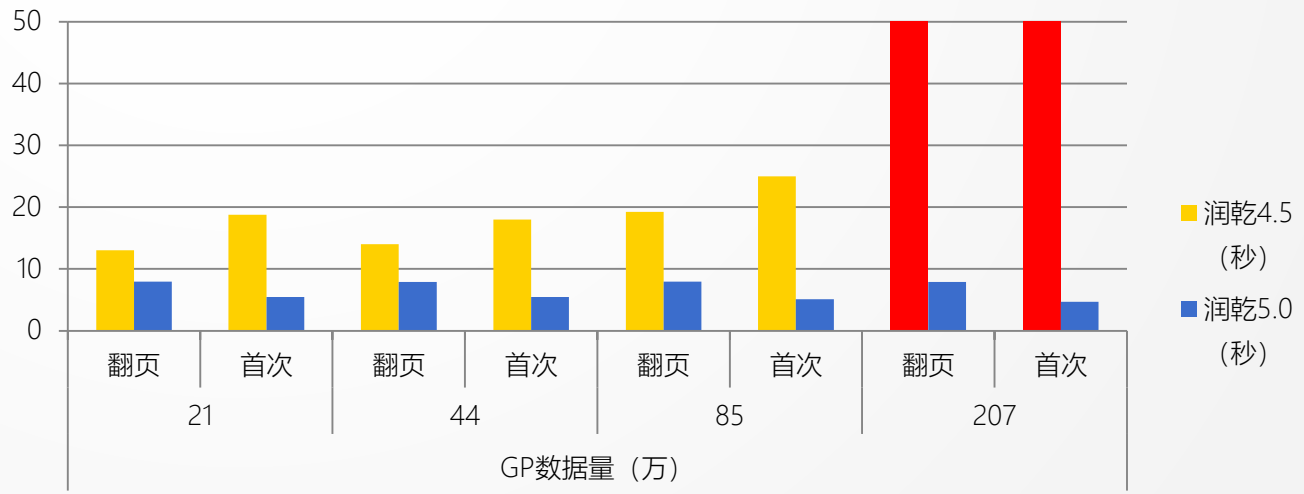


数据计算层实现分页取出

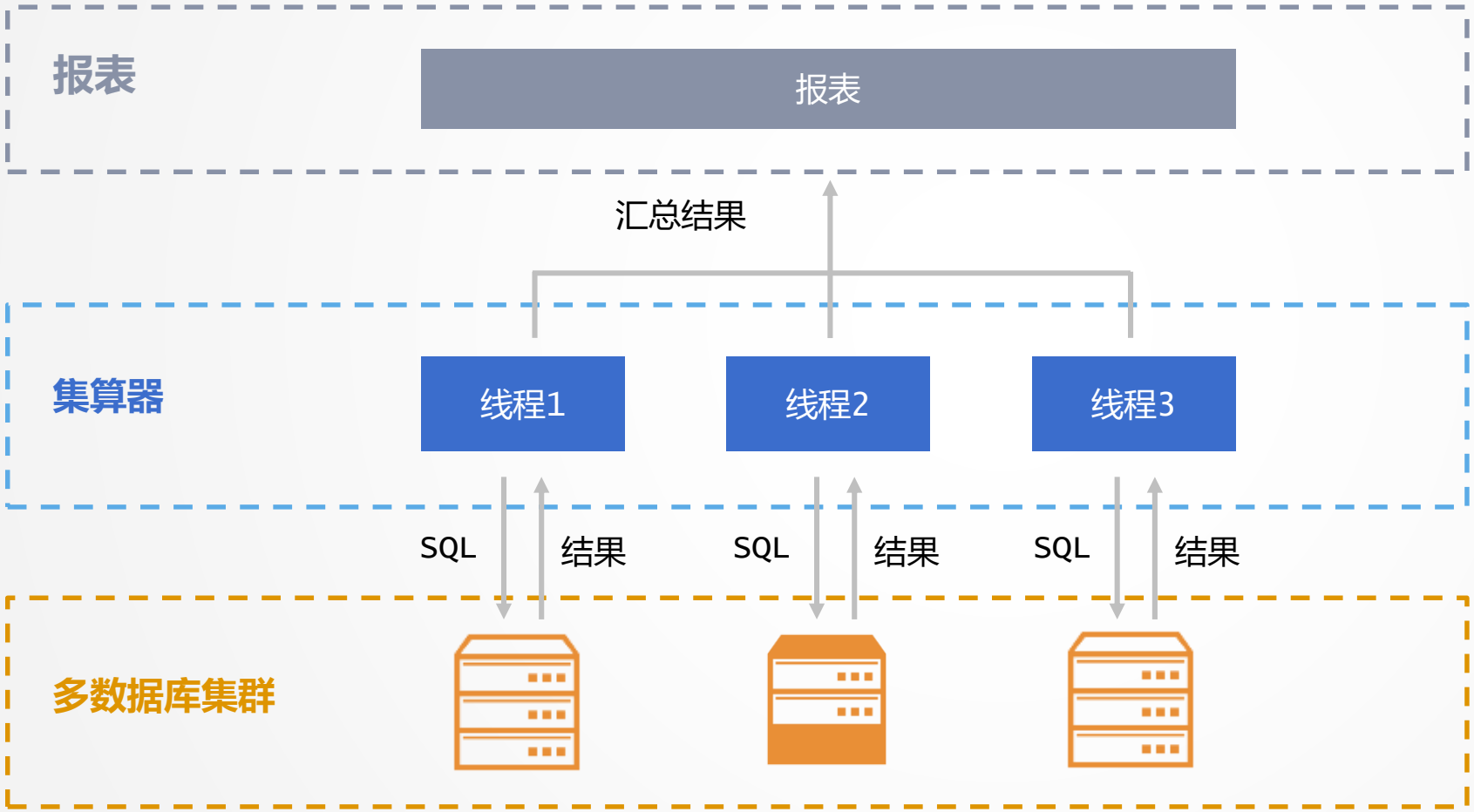
异步线程边取数边呈现，快速响应

本地缓存，按页随机取数

不依赖于数据库的分布取出能力



分库计算汇总



集算器可以基于同构或异构数据库集群进行结果汇总，为报表输出汇总后的结果集

混合运算实现T+0报表

目前T+0报表常用方式与存在问题：

① 历史和当期数据同库存储

大量的历史数据会导致高昂的数据库成本（存储成本和性能成本）

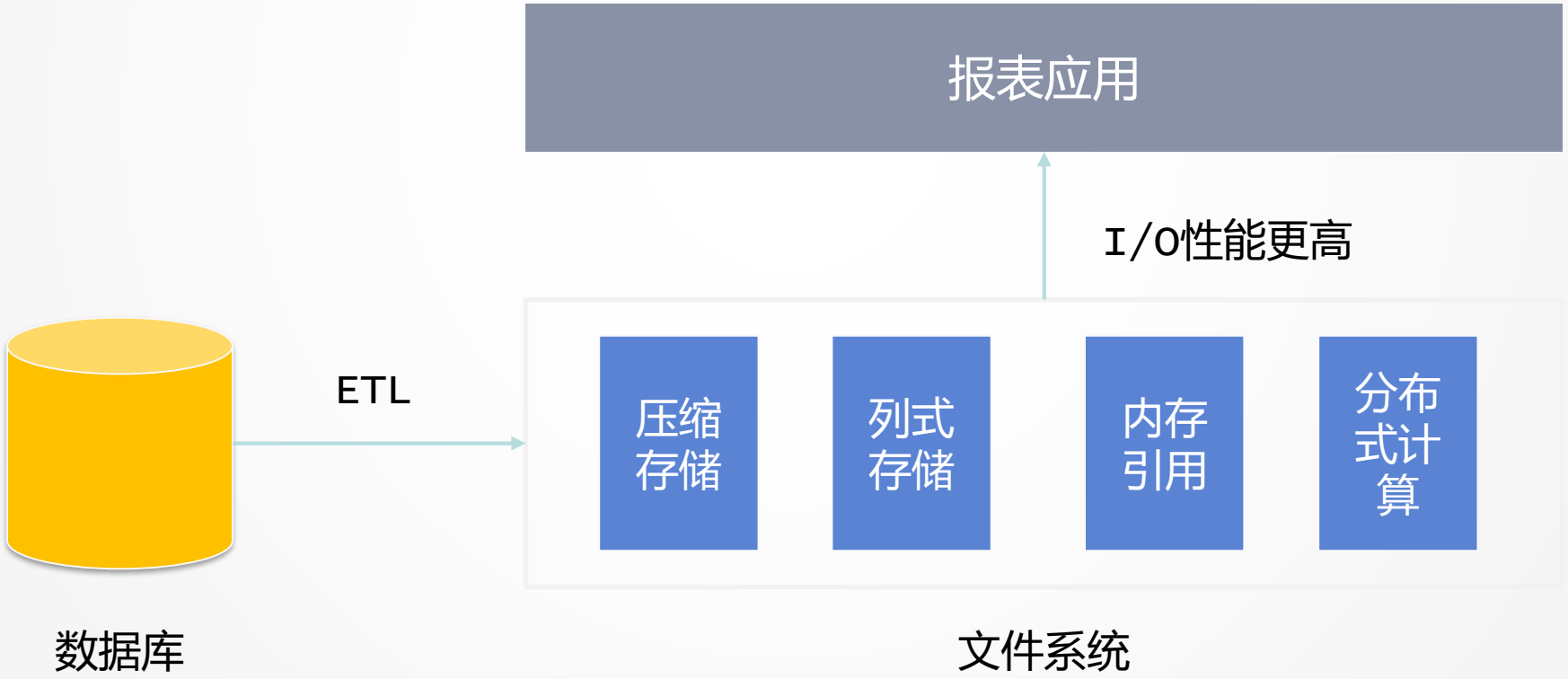
② 历史和当期数据分库存储

需要数据库具有跨库运算能力，但实施复杂度较高，性能较低；当数据库类型不同时难以实现

- 集算器可以基于多个异构数据库完成报表T+0查询；
- 还可以将历史数据存放到IO性能更佳的文件系统中采用集群运算获得更高性能和更低成本

数据外置与并行计算

文件系统IO性能高于数据库，还可以使用压缩数据存储、列式存储、内存记录引用，以及分布式集群等获得更高性能，减轻数据库负担



Hadoop数据源

Hadoop计算能力尚不完善

开发复杂

经常把计算后的数据再导入关系数据库

使用集算器实现Hadoop上报表计算

开发更轻松

支持集群并行计算，比Hadoop原生方法性能更好

无需导入数据库

集算器实现数据计算层-总结

应对报表的业务不稳定性，集算器的手段和达到的目标

高效率

更丰富的语法和类库

低耦合

报表模板与算法一体化

工具化

报表开发彻底独立化与应用解耦

热切换

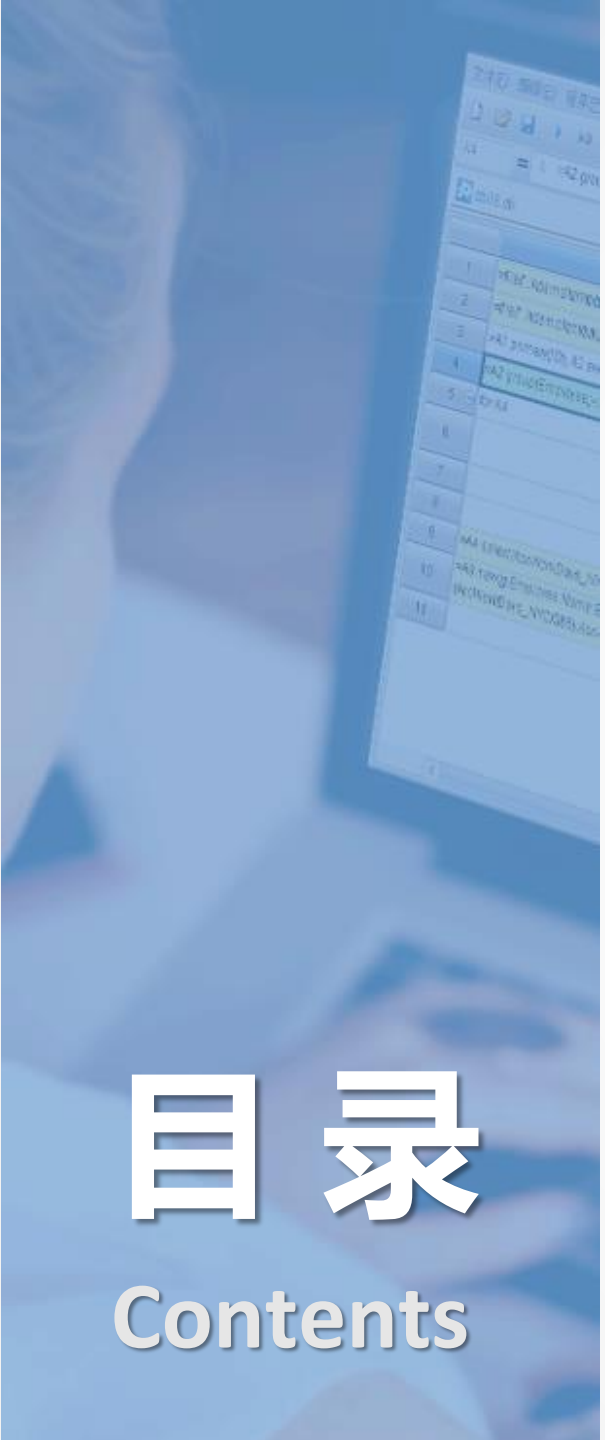
解释执行无须重启应用

易开发

无须环境配置与应用层代码引用

低成本

非专业程序员自主完成

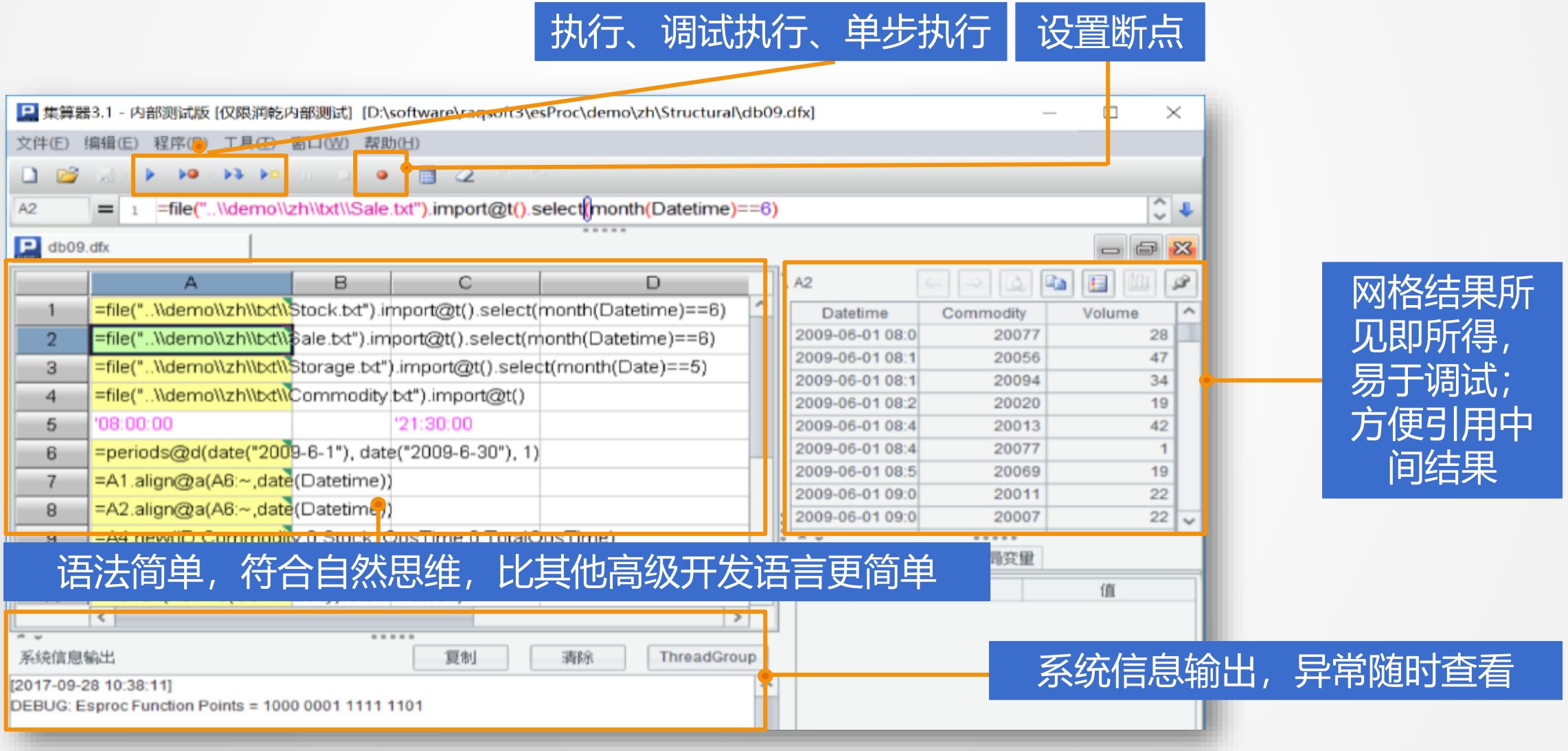


目录

Contents

集算器–开发环境

即装即用，调试功能完善



集算器–面向过程计算

完整的循环分支控制

	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(~>=A5[B9])	/大客户数			
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

集算器–敏捷语法体系

某支股票最长连续涨了多少交易日

```
1  select max(连续日数)
2  from (select count(*) 连续日数
3        from (select sum(涨跌标志) over(order by 交易日) 不涨日数
4              from (select 交易日,
5                    case when 收盘价>lag(收盘价) over(order by 交易日)
6                    then 0 else 1 end 涨跌标志
7              from 股价表) )
8        group by 不涨日数)
```

SQL

	A
1	=股价表.sort(交易日)
2	=0
3	=A1.max(A2=if(收盘价>收盘价[-1],A2+1,0))

集算脚本

思考：按照自然思维怎么做?

语法体系更容易描述人的思路

数据模型不限制高效算法实现

集算器–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)	

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)		

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])		

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	

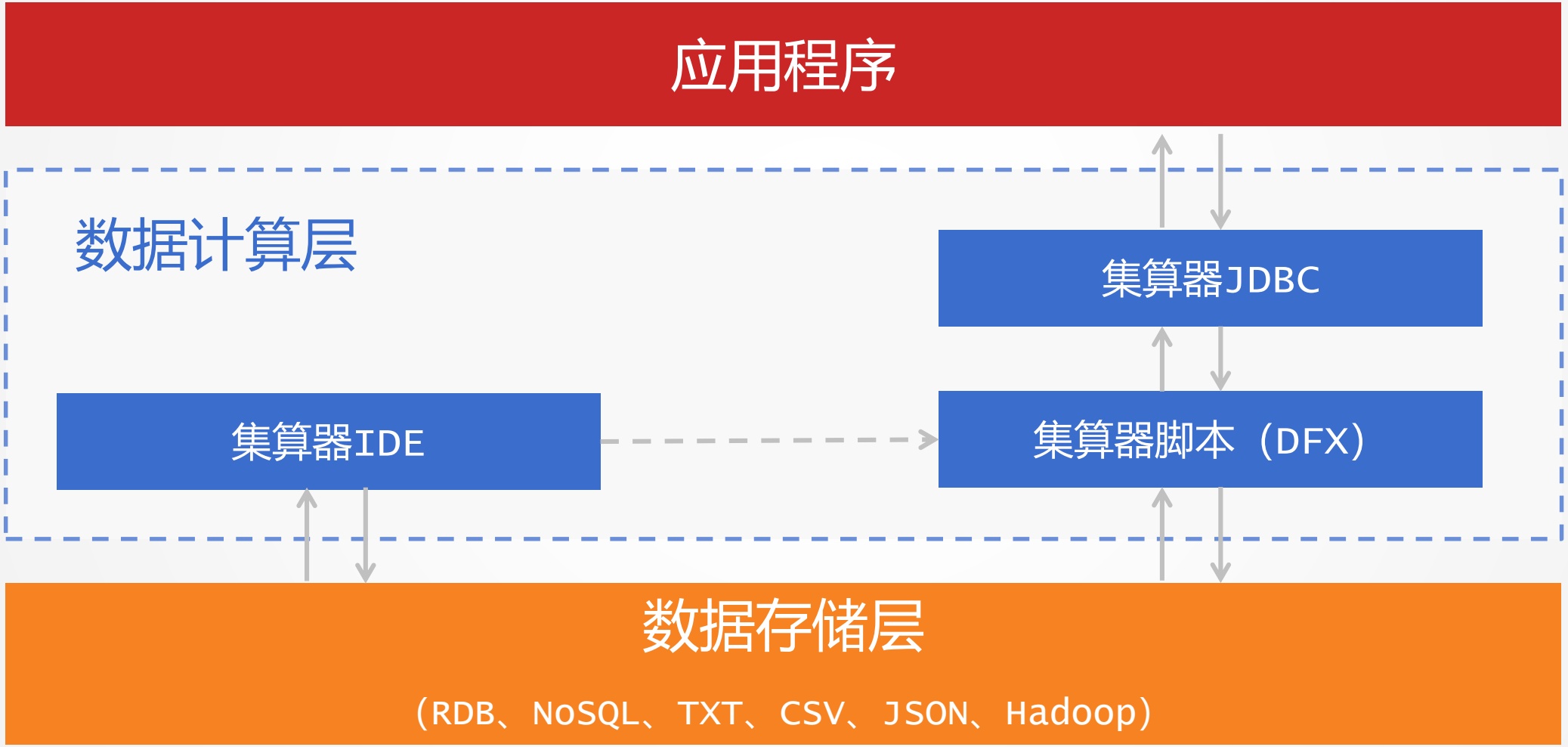
有序集合

集算器–多样性数据源接口

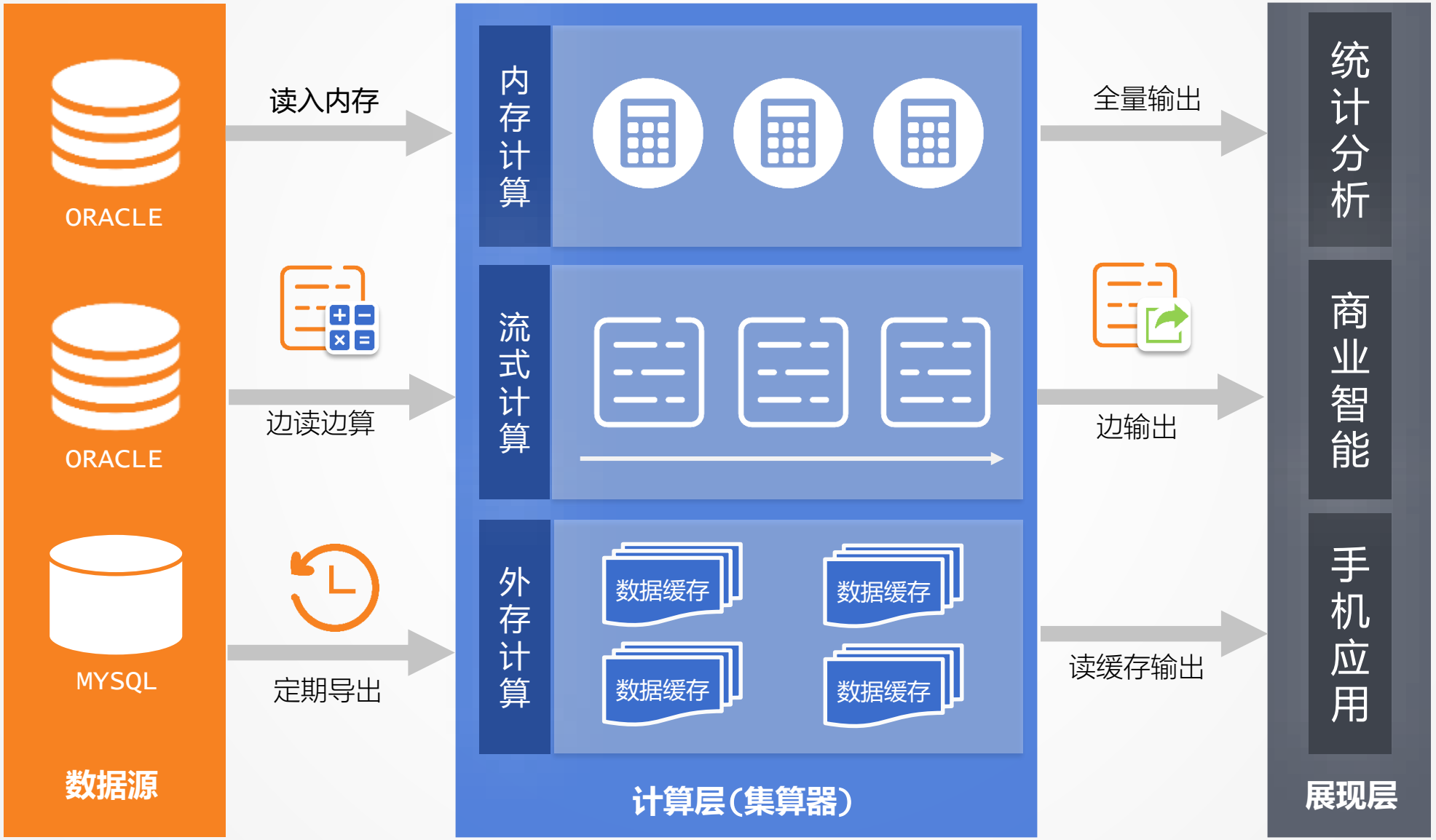
- 高效二进制压缩文件、列式存储
- RDB：Oracle,DB2,MS SQL,MySQL,PG,....
- TXT/CSV，JSON/XML，EXCEL
- Hadoop：HDFS，HIVE，HBASE
- MongoDB，REDIS，...
- HTTP、ALI-OTS
- ... ...
- 内置接口，即装即用

集算器–集成与管理体系

应用无缝集成，代码易于管理



集算器–数据流向模式


ORACLE
ORACLE
MYSQL
数据源
读入内存
边读边算
定期导出
内存计算
流式计算
外存计算
数据缓存
数据缓存
数据缓存
数据缓存
计算层(集算器)
全量输出
边输出
读缓存输出
统计分析
商业智能
手机应用
展现层

创新技术

推动应用进步！