# 如何高效处理数据

## 数据计算中间件

润乾软件@2018

# 目录页

CONTENTS


- 01 数据计算中间件
- 02 应用场景
- 03 特征与标准
- 04 相关技术
- 05 集算器esProc
- 06 应用案例

# 过渡页

CONTENTS


01 数据计算中间件

02 应用场景

03 特征与标准

04 相关技术

05 集算器esProc

06 应用案例

# 中间件回顾

## 什么是中间件

中间件是一种独立的系统软件或服务程序，分布式应用软件借助这种软件在不同的技术之间共享资源，中间件位于客户机服务器的操作系统之上，管理计算资源和网络通信。[IDC]

## 中间件特性

- 满足大量应用的需要
- 运行于多种硬件和OS平台
- 支持分布计算，提供跨网络、硬件和OS平台的透明性的应用或服务的交互
- 支持标准的协议
- 支持标准的接口

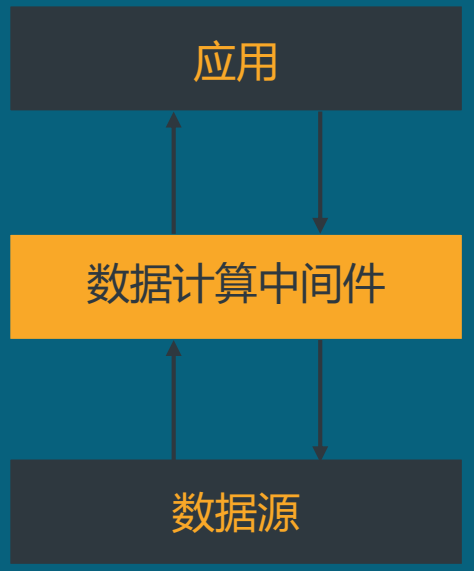
# 中间件分类

| | |
|---|---|
| 01 | 通信处理中间件 |
| 02 | 网络中间件 |
| 03 | 安全中间件 |
| 04 | WEB服务器中间件 |
| 05 | 事务处理中间件 |
| 06 | 数据存取管理中间件 |
| 07 | 数据计算中间件 |

# 什么是数据计算中间件

**DCM**

位于数据源和应用之间提供通用计算服务的中间件称为“**数据计算中间件**”，简称：**DCM**(Data Computing Middleware)



# DCM适应症状

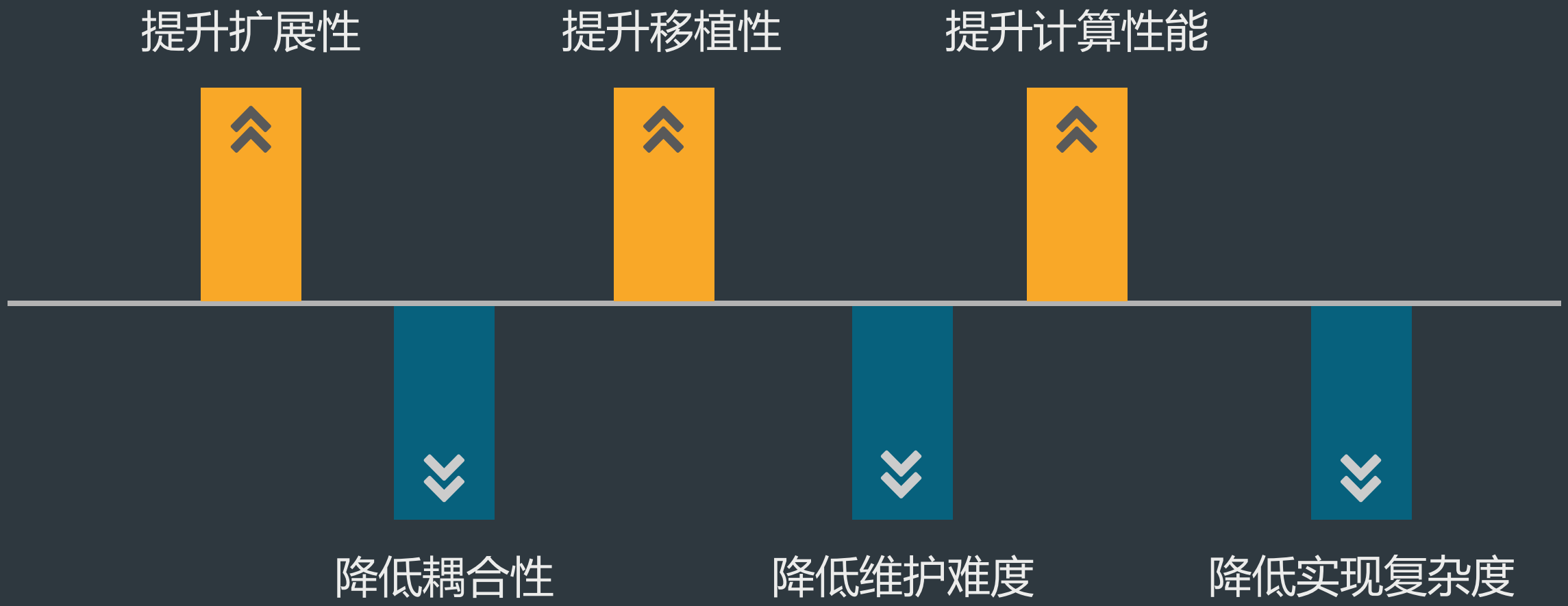
| | |
|---|---|
| 1 | 报表开发没完没了，新的不断加，老的总要改 |
| 2 | 查询与分析响应太慢，BI系统体验差 |
| 3 | 数据库资源总不够用，需要扩容 |
| 4 | ETL时间过长，影响业务进程 |
| 5 | 应用耦合性过强，无法拆分扩容 |
| 6 | 前人的代码怎么看也看不明白，维护累死人 |
| 7 | 新需求业务提完一直催，想快速实现 |

# DCM适应症状

| | |
|---|---|
| 8 | 多应用系统数据难以整合沟通 |
| 9 | 网络、日志等库外数据处理效率太低 |
| 10 | Hadoop性能功能不过关，还要指望传统数据库 |
| 11 | MySQL没有窗口函数，复杂计算难实现 |
| 12 | NoSQL做不了JOIN，无法实现复杂计算 |
| 13 | 直接用Java计算，编码维护难，库外代码满天飞 |
| 14 | 存储过程调试繁琐，编译执行，存在安全隐患 |

**技术团队很受伤**

# 为什么需要DCM



# 为什么需要DCM

## 提升开发效率

### 算法实现效率 -> 编码

- 明明知道算法逻辑，为啥用SQL咋就这么难？
- SQL是命令式编程，中间结果不可复用

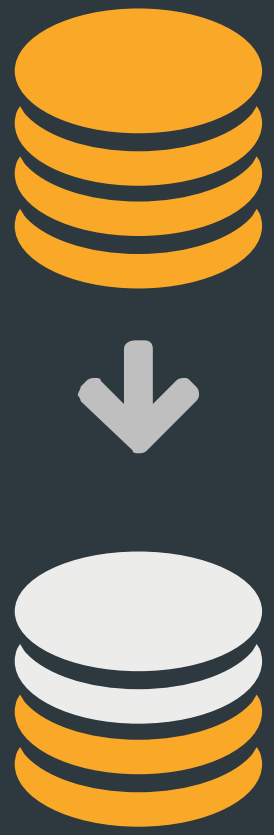
### 程序维护效率 -> 运维

- 存储过程是用来做高性能数据处理，不是用来做数据查询
- 别让复杂查询，变的不可维护

# 为什么需要DCM

## 减轻数据源（库）负担

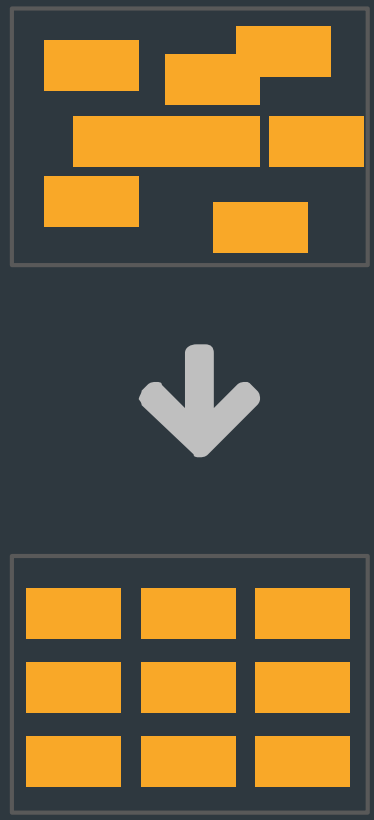
- 访问负担 -> 缓存
- 计算负担 -> 计算引擎
- 管理负担 -> 优化工具



# 为什么需要DCM

## 优化应用结构

- 应用与应用解耦
- 应用与数据源解耦
- 异构数据源混合使用



# 为什么需要DCM

## 提升计算性能

- 单线程计算能力
- 多线程计算能力
- 分布式计算能力

# 过渡页

CONTENTS


01 数据计算中间件

02 应用场景

03 特征与标准

04 相关技术

05 集算器esProc

06 应用案例

# 应用场景

01 报表应用数据准备

02 BI系统数据引擎

03 数据计算前置机

04 ETL数据处理

05 库外存储过程

06 边缘计算服务

07 可编程数据网关

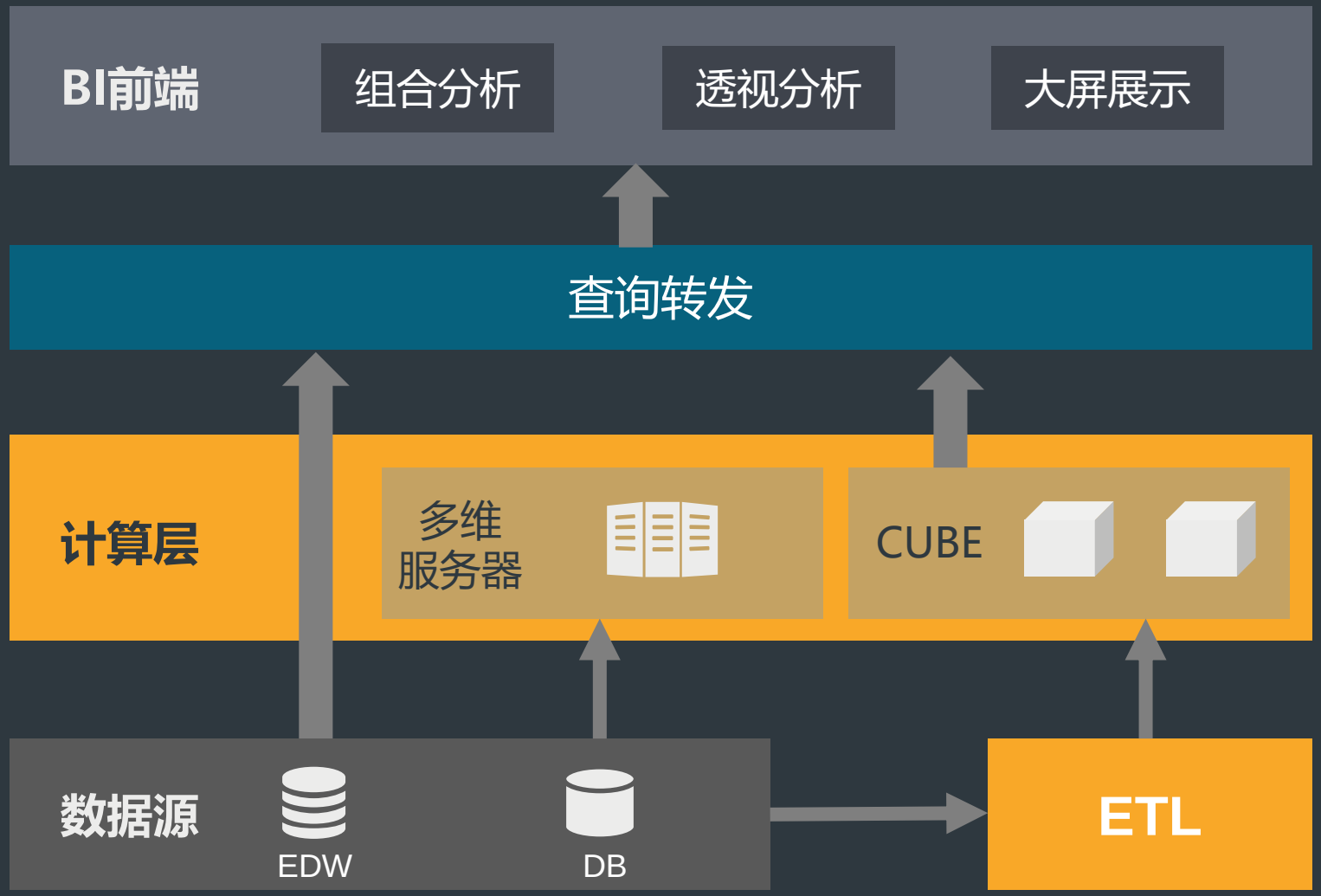
08 嵌入式数据计算

# 报表应用数据准备


## 计算层优点

- 提高报表开发效率
- 降低报表与应用耦合度
- 降低报表与数据源耦合度
- 提升报表计算性能

# BI系统数据引擎




## 数据引擎优点

- OLAP引擎负载过高
- CUBE建设过于复杂
- 冷热数据混合查询
- 数据计算智能分布

# 数据计算前置机


**共享计算前置机**

node1　node2　nodex

**数据源**

DB/DW　FileSystem　HDFS　其他数据源

## 数据前置机优点

- 降低应用自身耦合性
- 降低应用与数据源耦合性
- 提升系统扩展性
- 灵活选择共享和嵌入前置

# ETL数据处理

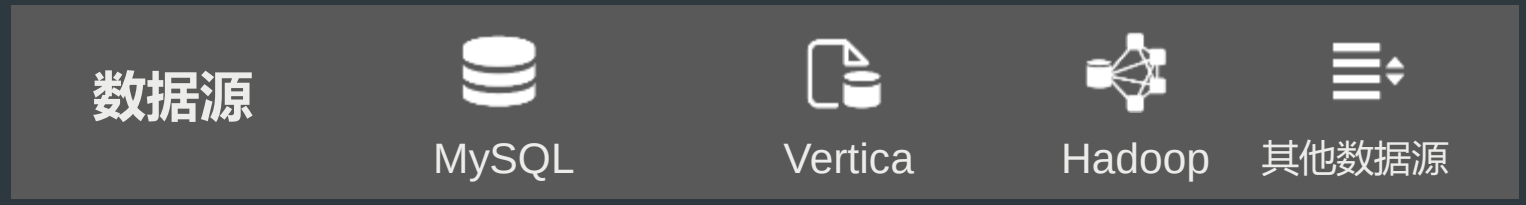

## 嵌入ETL数据处理优点

- 缩短时间窗口占用
- 提高处理性能
- 实现简单灵活
- 允许多源混合处理

# 库外存储过程

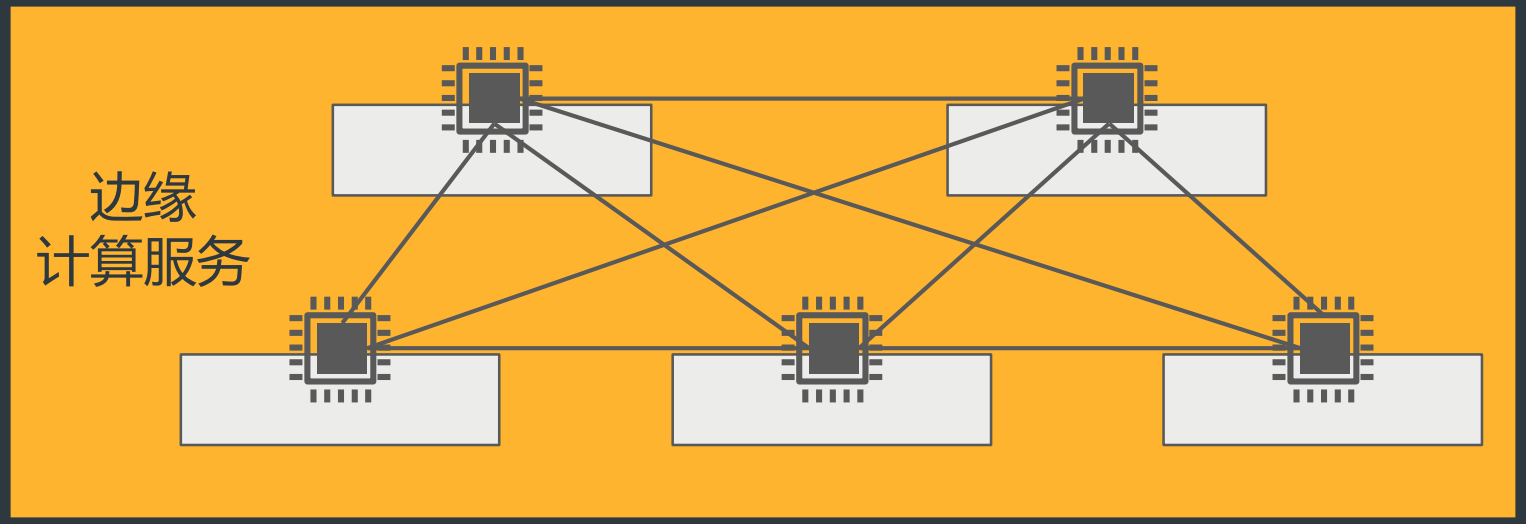

## 库外存储过程优点

- 继承存过过程化、集合计算优点
- 实现跨库计算
- 方便调试，便于移植
- 应用与数据源解耦



# 边缘计算服务

端到端业务




智能网关

智能资产

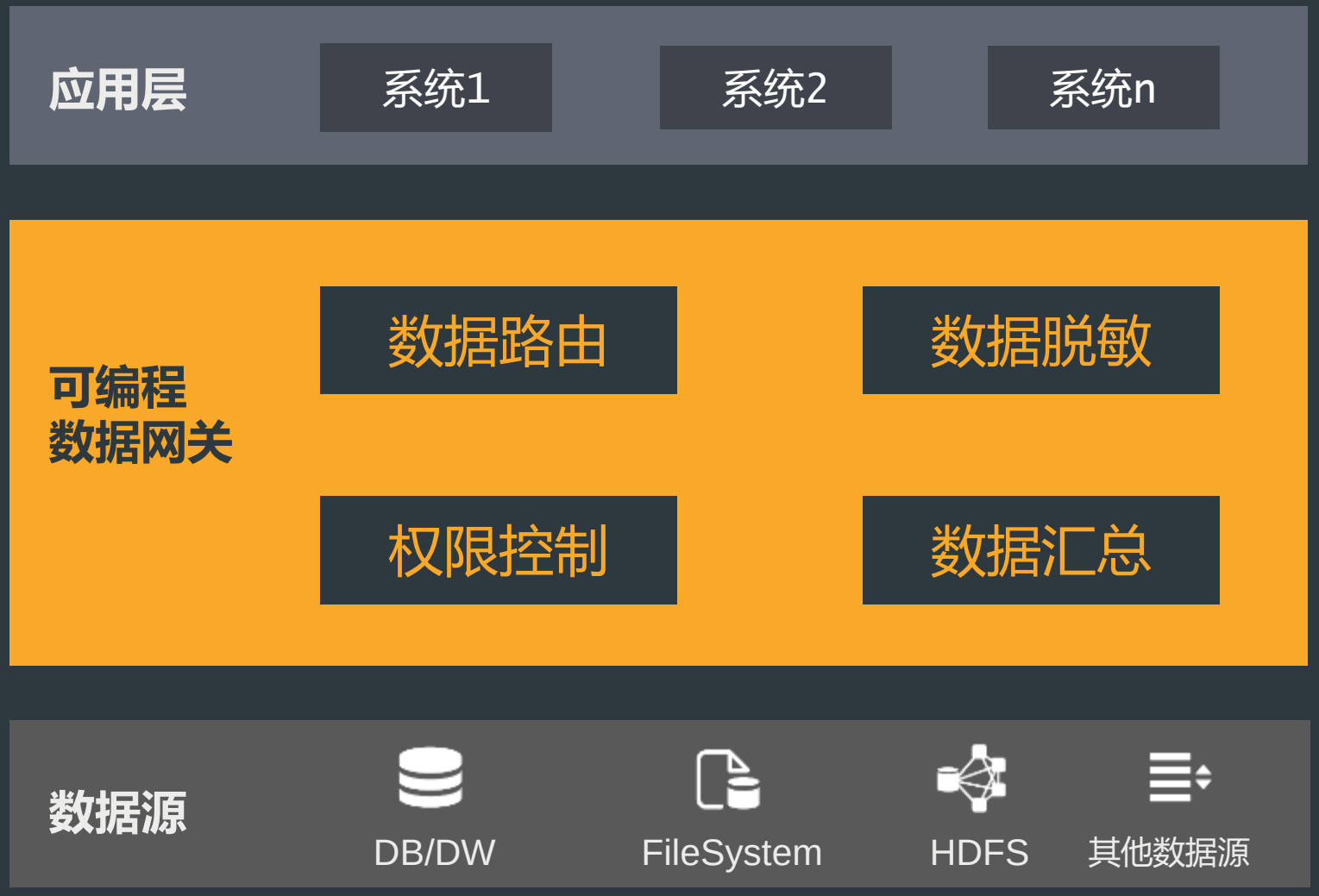
存储

网络

## 边缘计算服务优点

- 充分发挥集合计算能力
- 业务实现简单
- 提高可维护性
- 支持独立计算服务器

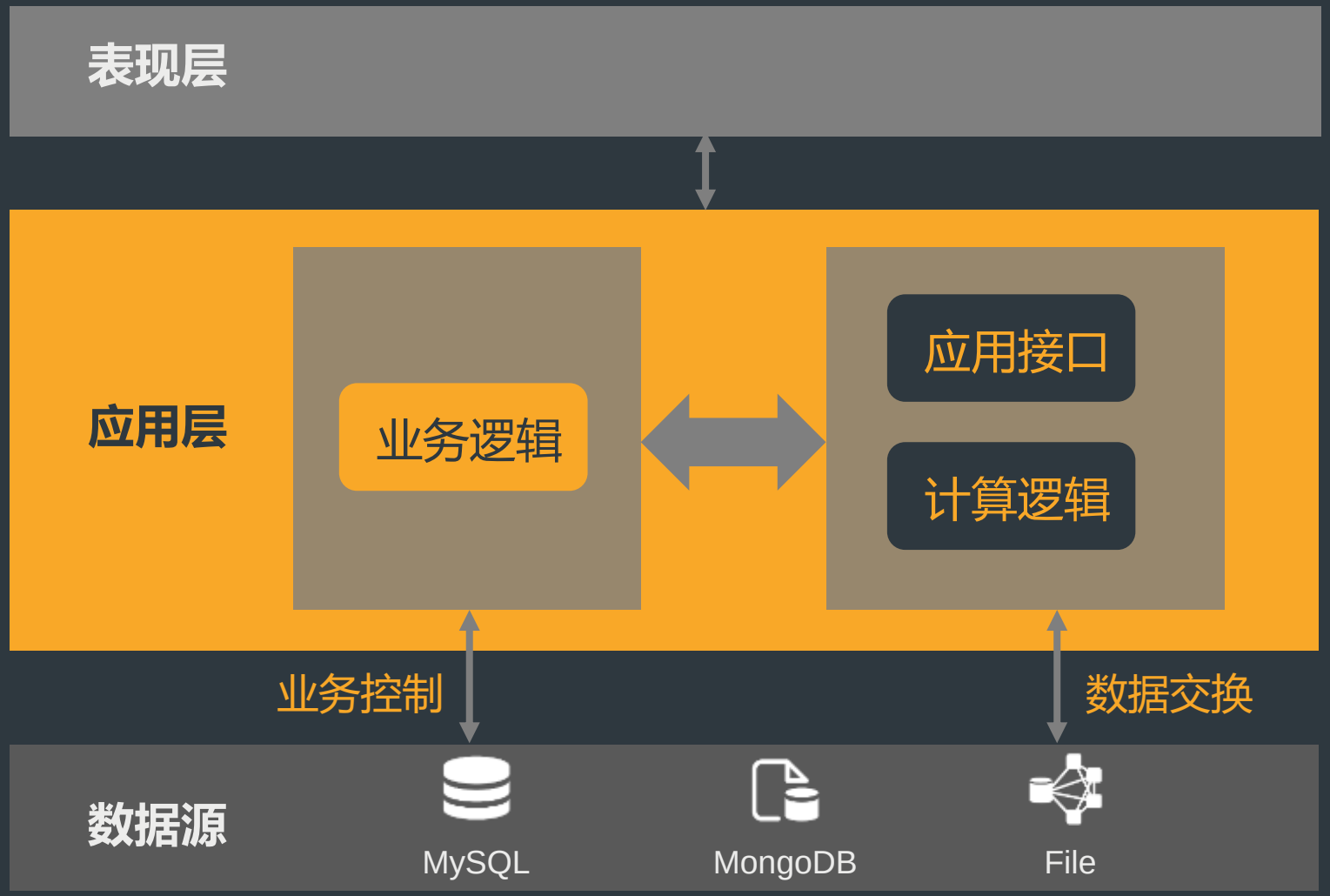
# 可编程数据网关




## 可编程数据网关优点

- 支持嵌入，可结合已有网关集成使用
- 可编程逻辑控制灵活
- 同一任务可路由至不同源并在网关中进行分支归并

# 嵌入式数据计算




## 嵌入式数据计算优点

- 可为应用或其他服务提供统一计算输出
- 增强系统扩展性
- 增强可维护性
- 支持热部署

# 过渡页

CONTENTS


01 数据计算中间件

02 应用场景

03 特征与标准

04 相关技术

05 集算器esProc

06 应用案例

# 数据计算中间件特征

CHEASE

- Compatible
- Hot-deploy
- Efficient
- Agile
- Scalable
- Embeddable

# 数据计算中间件标准

## CHEASE

| C | H | E | A | S | E |
|---|---|---|---|---|---|
| 兼容性 | 热部署 | 高性能 | 敏捷性 | 扩展性 | 集成性 |
| Compatible | Hot-deploy | Efficient | Agile | Scalable | Embeddable |

# 过渡页

CONTENTS


01 数据计算中间件

02 应用场景

03 特征与标准

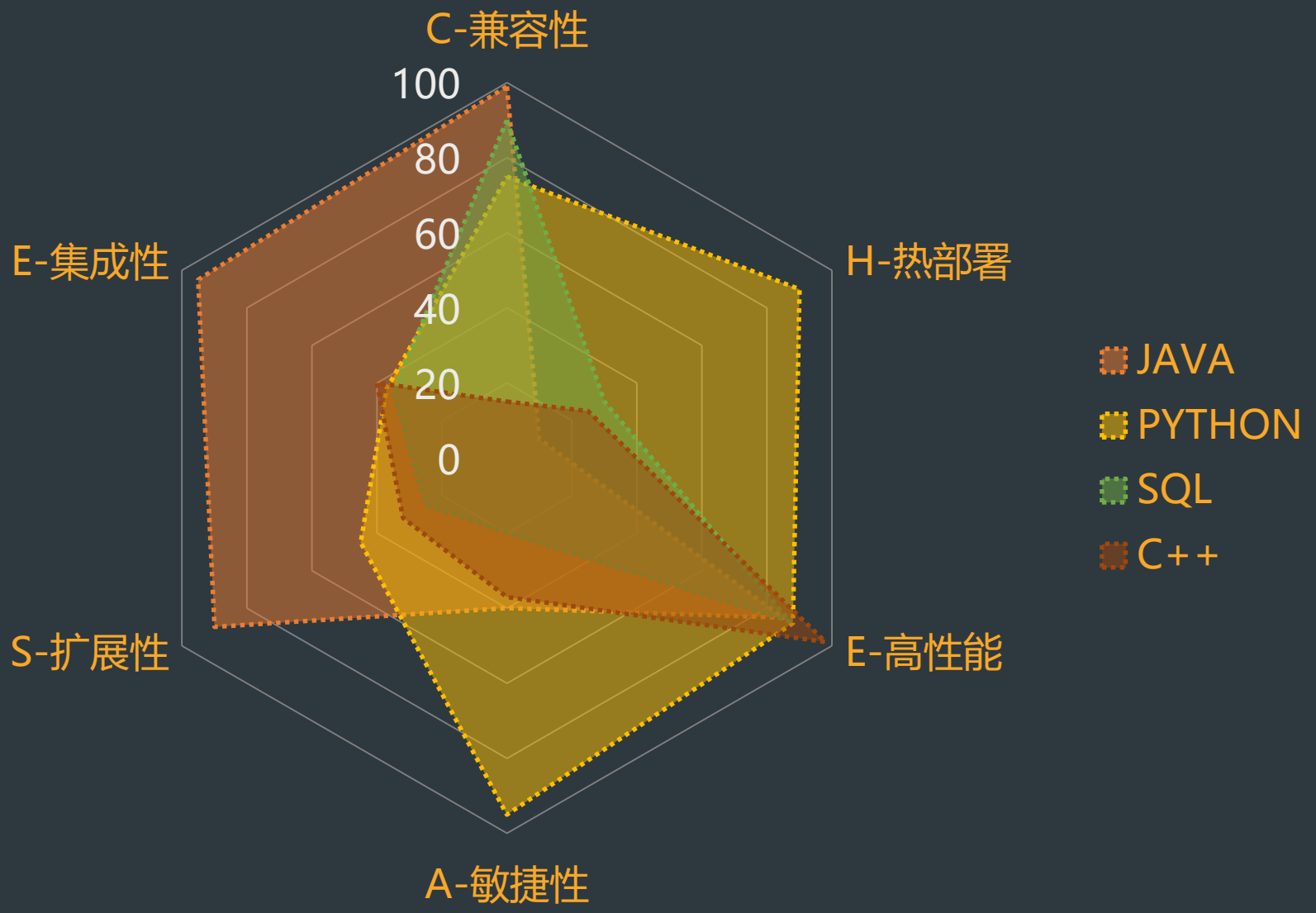
04 相关技术

05 集算器esProc

06 应用案例

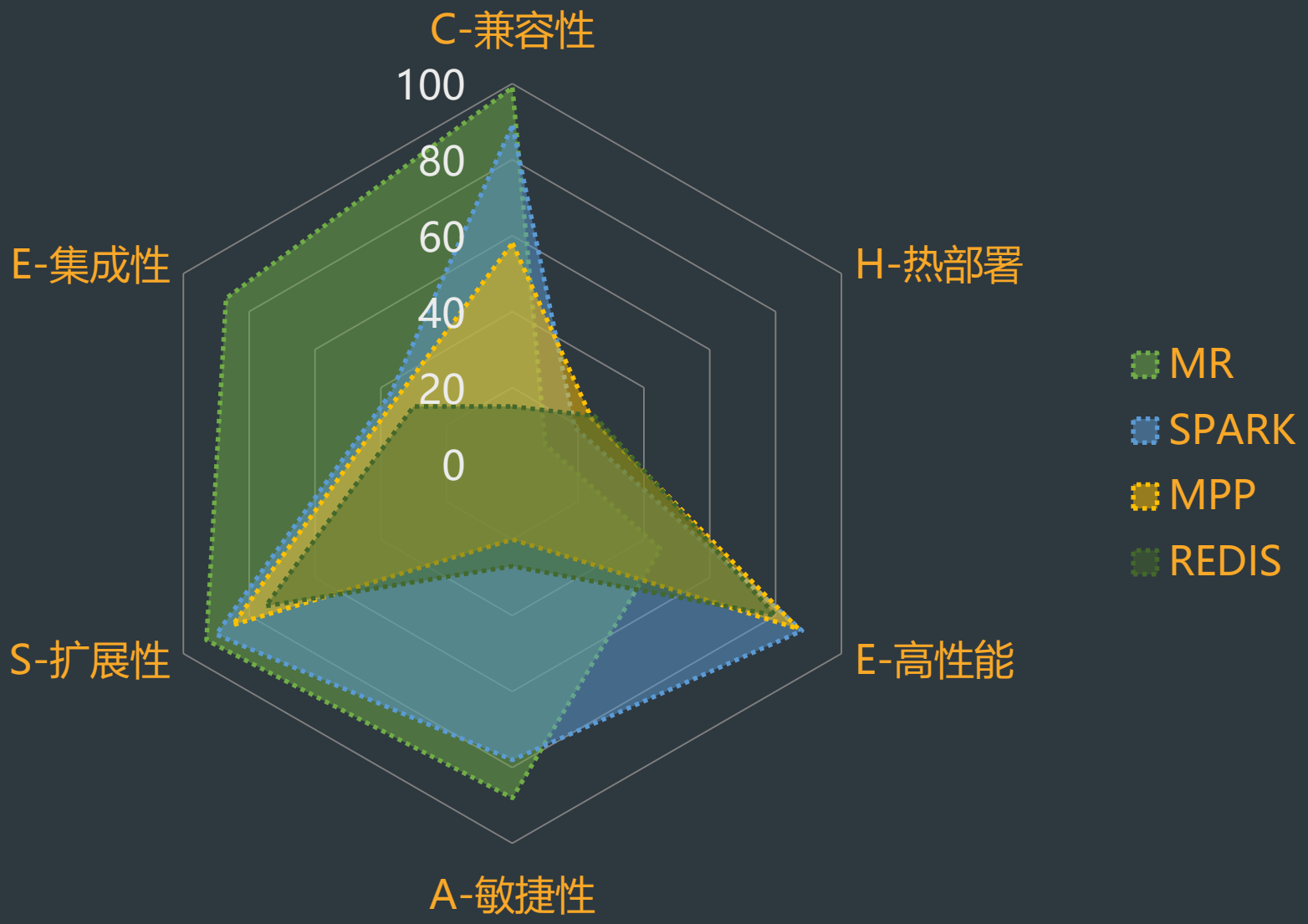
# 语言

| | C 兼容性 | H 热部署 | E 高性能 | A 敏捷性 | S 扩展性 | E 集成性 |
|---|---|---|---|---|---|---|
| JAVA | | X | | X | | |
| SQL | | X | | X | X | X |
| Python | | | X | | X | X |
| C++ | X | X | | X | X | X |



# 技术

| | C 兼容性 | H 热部署 | E 高性能 | A 敏捷性 | S 扩展性 | E 集成性 |
|---|---|---|---|---|---|---|
| MR | | X | X | X | | |
| Spark | | X | | | | X |
| MPP | | X | | X | | X |
| Redis | X | X | | X | | X |



# 过渡页

CONTENTS


01 数据计算中间件

02 应用场景

03 特征与标准

04 相关技术

05 集算器esProc

06 应用案例

# SPL与集算器

| C | H | E | A | S | E |
| --- | --- | --- | --- | --- | --- |
| 兼容性 | 热部署 | 高性能 | 敏捷性 | 扩展性 | 集成性 |

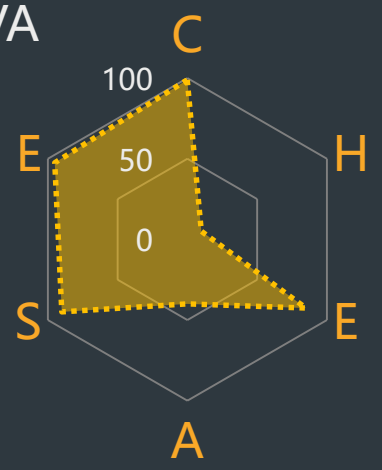
| JAVA开发 | 解释执行 | 并行计算 | 敏捷语法 | 分布式环境 | 中间件定位 |
| --- | --- | --- | --- | --- | --- |

SPL
(Structured Process Language)

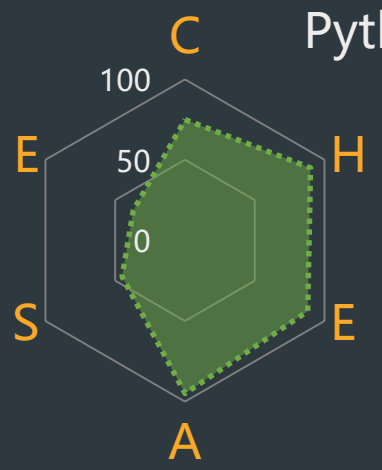
集算器 esProc

# SPL

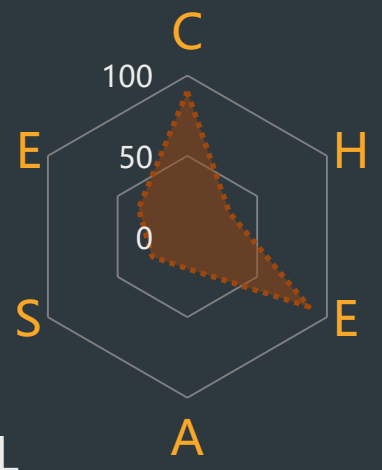
JAVA



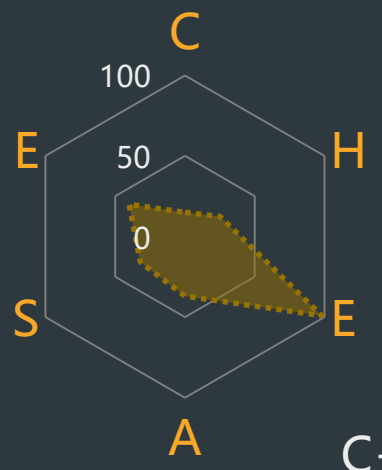

Python

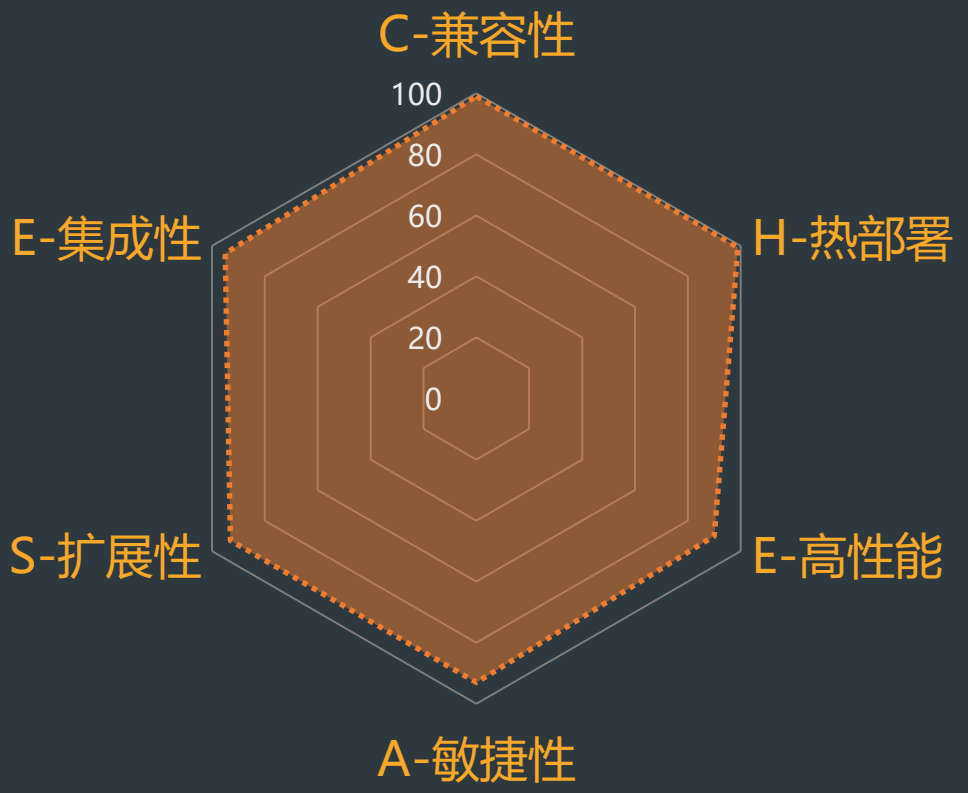



SQL



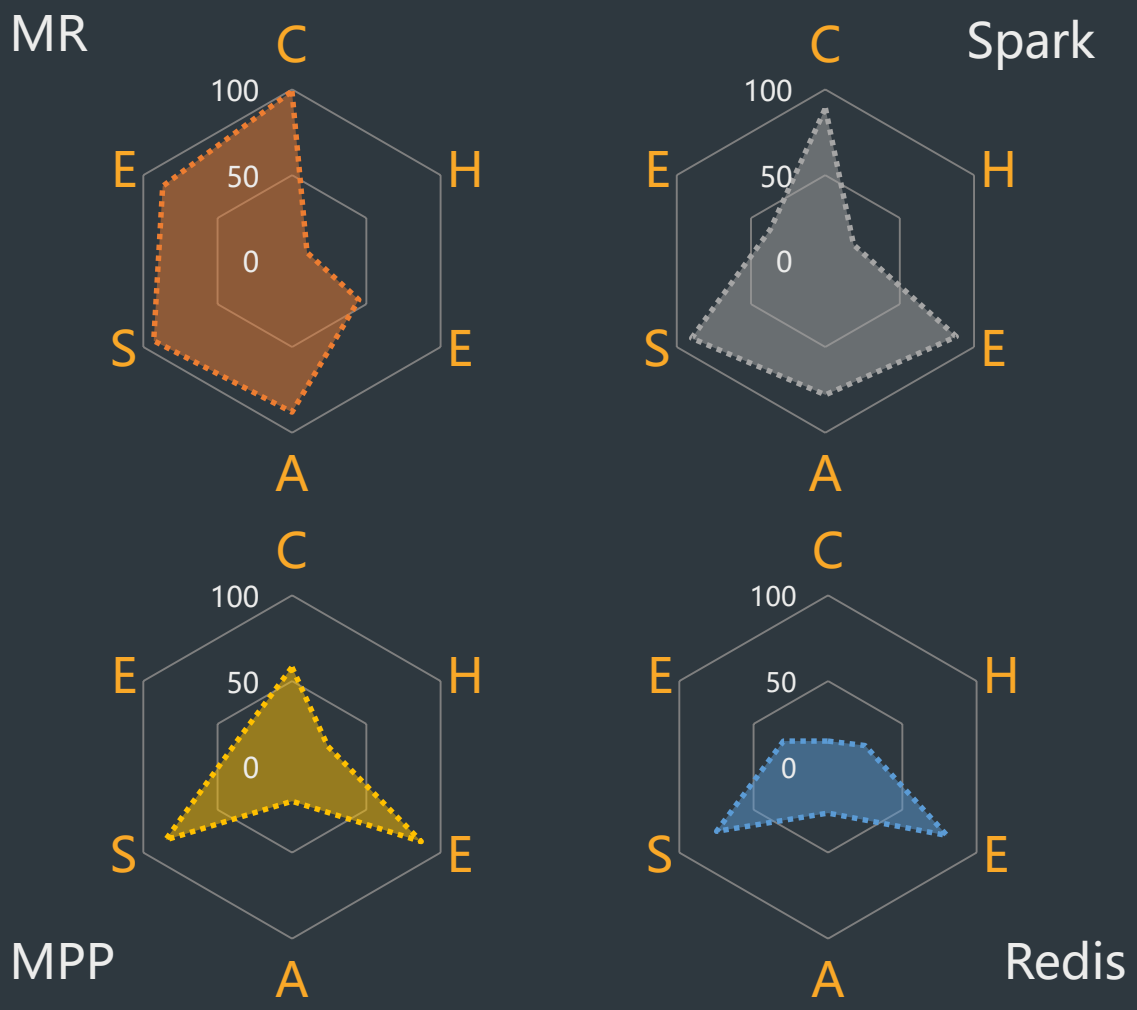

C++

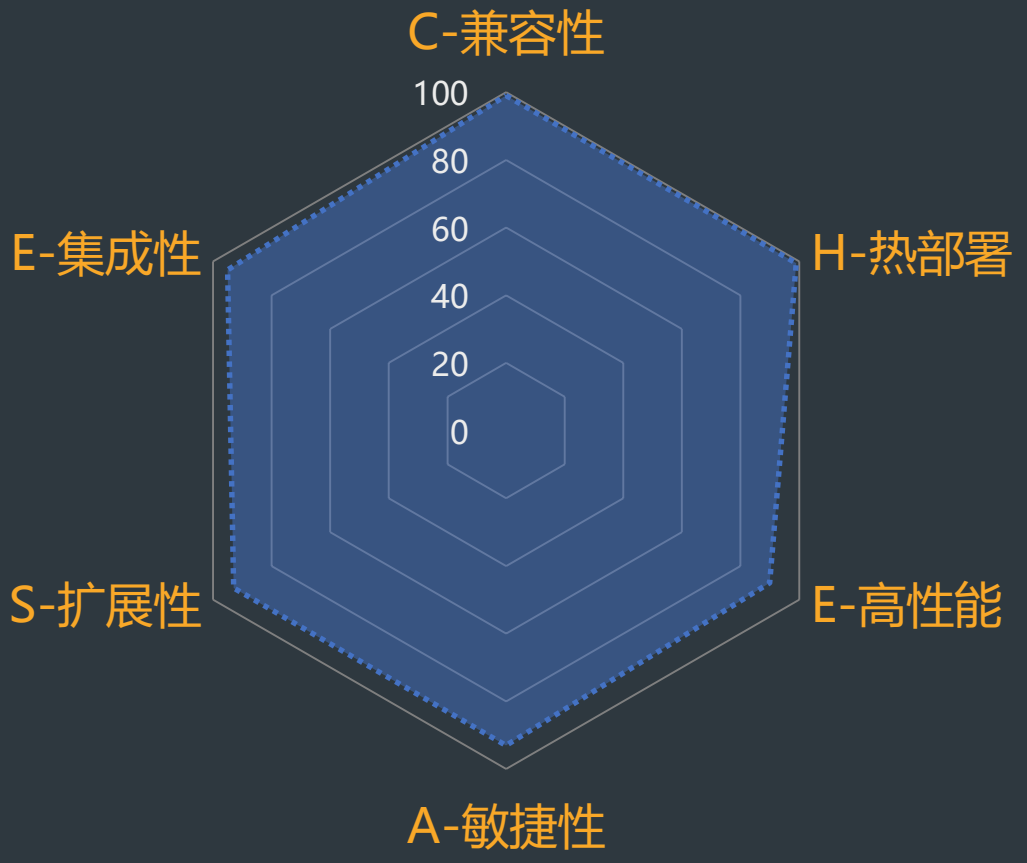



SPL

# 集算器esProc


MR
C
H
E
S
A
100
50
0
Spark
Redis
MPP


C-兼容性
H-热部署
E-高性能
A-敏捷性
S-扩展性
E-集成性
100
80
60
40
20
0
esProc

# 敏捷语法

## 某支股票最长连续涨了多少交易日？

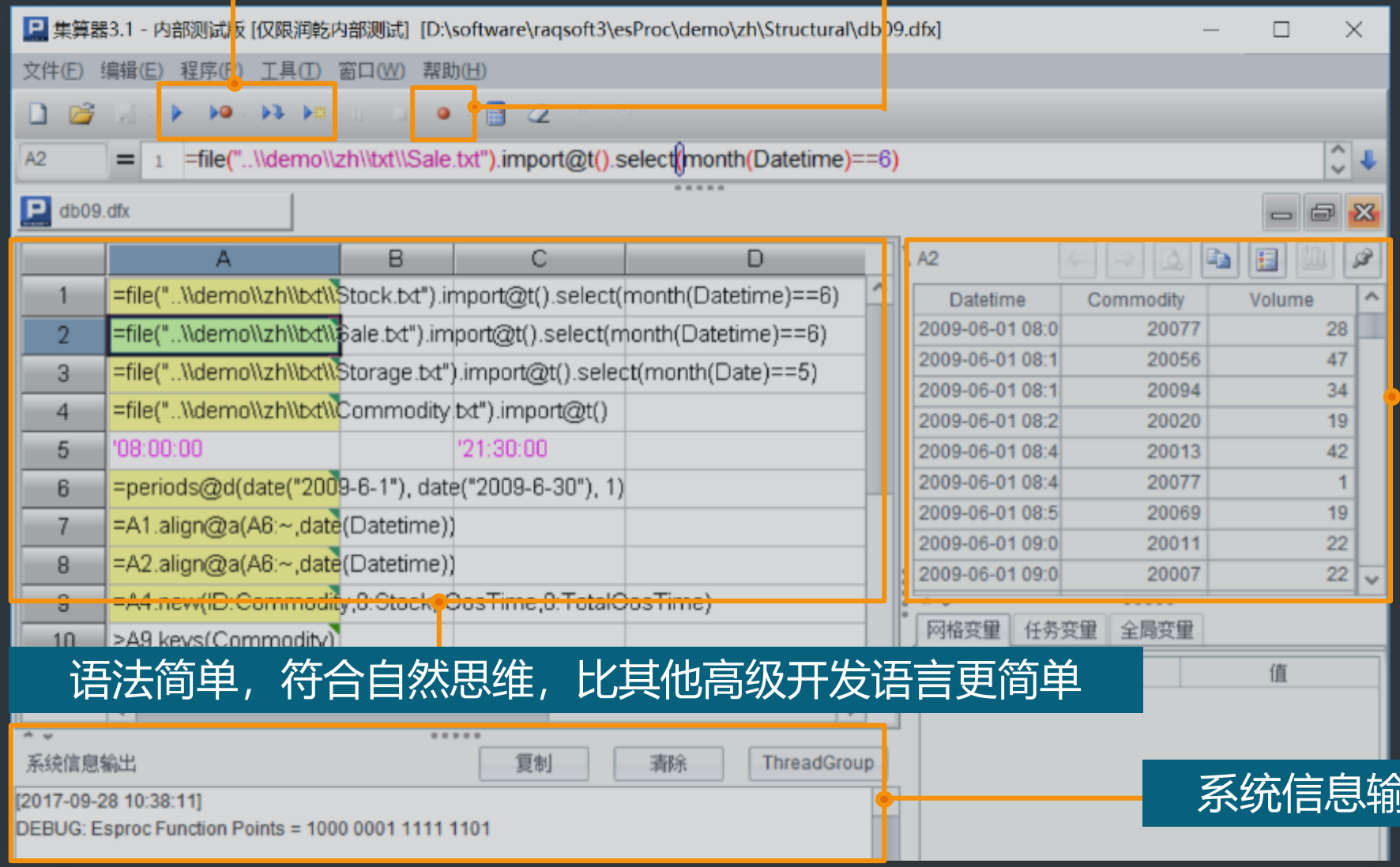
```
1  select max(连续日数)-1
2  from (select count(*) 连续日数
3        from (select sum(涨跌标志) over(order by 交易日) 不涨日数
4              from (select 交易日,
5                    case when 收盘价>lag(收盘价) over(order by 交易日)
6                    then 0 else 1 end 涨跌标志
7              from 股价表) )
8        group by 不涨日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

| | 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)) |

# 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**

# 集合运算

集算器作为集合化的程序语言，支持lamada语法和动态数据结构

| | A | |
|---|---|---|
| 1 | =file(“D.csv”).import@tc(name,sex,age) | |
| 2 | =A1.select(sex==”M”&&age>=25) | 过滤 |
| 3 | =A2.sort(name) | 排序 |
| 4 | =A2.groups(sex;avg(age):age) | 分组汇总，产生新数据结构 |
| 5 | =A2.id(left(name,1)) | 唯一值 |

# 过程计算

集算器具备良好的离散性，有效支持分步骤的过程计算

| | A | |
|---|---|---|
| 1 | =file(“D.csv”).import@tc() | 读入表数据 |
| 2 | =file(“P.txt”).import@t(id,area) | |
| 3 | =A1.switch(aid,A2:aid) | 关联，建立外键引用 |
| 4 | =A3.select(aid.area==”Beijing”) | 用外键引用记录的字段过滤 |

# 有序计算

在集合化和离散性配合下，可轻松完成有序计算

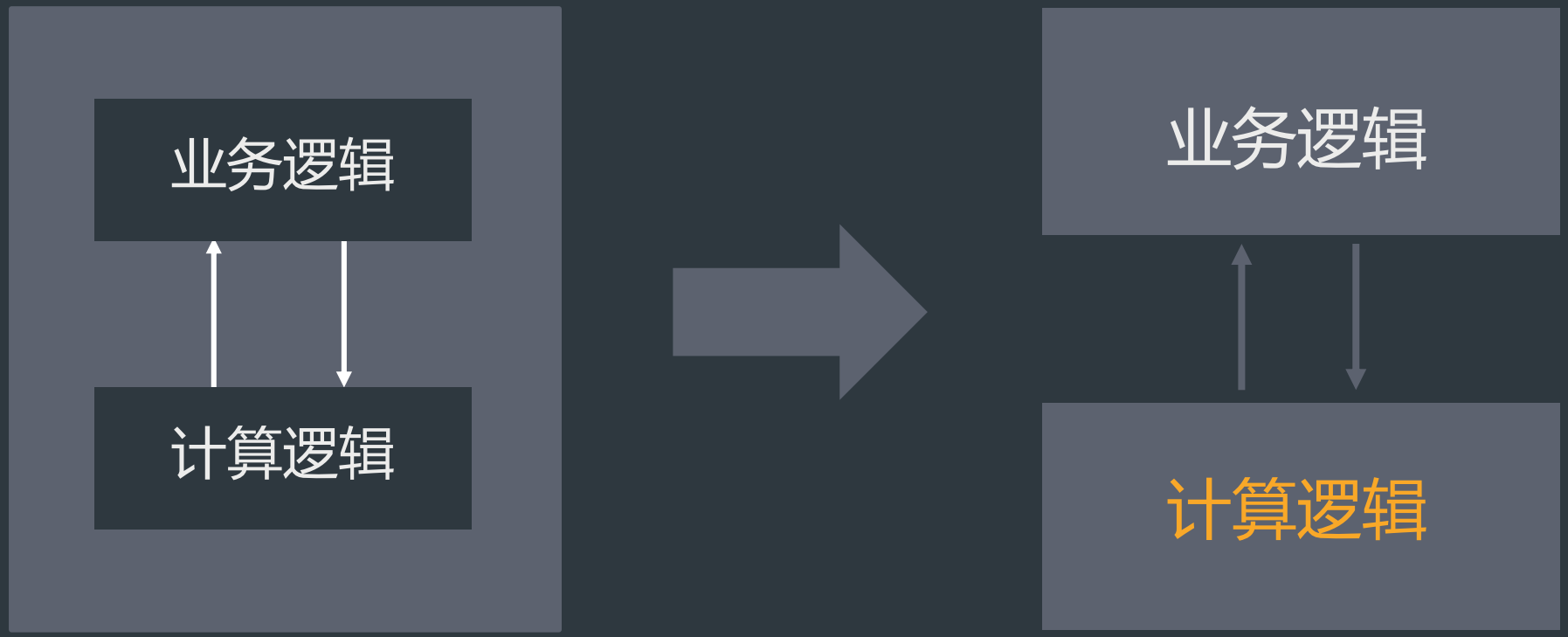
| | A | |
|---|---|---|
| 1 | =db.query("select * from S order by prod,month") | 读入表数据 |
| 2 | =A1.select(if(prod==prod[-1],sales/sales[-1])>1.1) | 销量比上月多10%记录 |

# 热切换

集算器脚本解释执行，支持不停机热切换

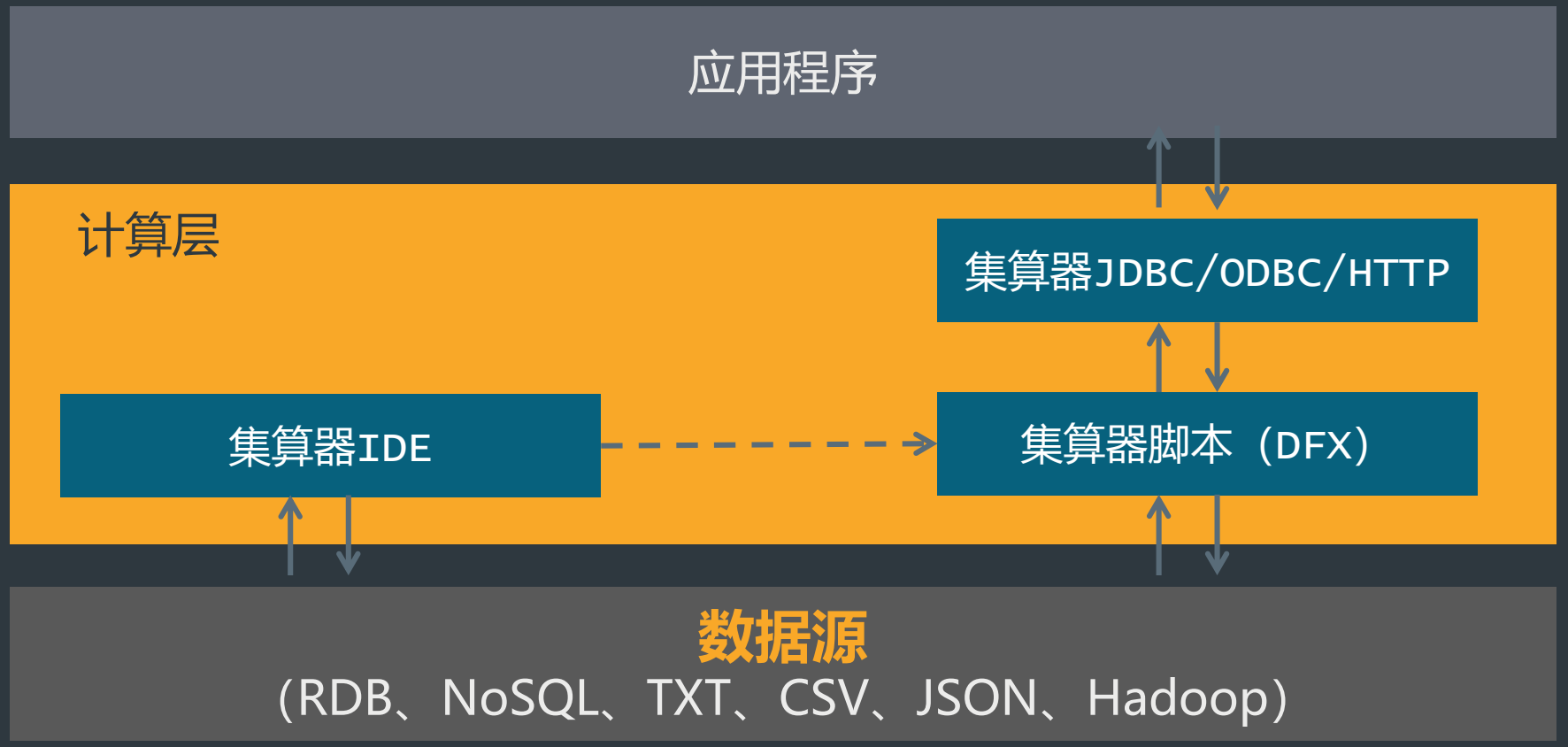
# 低耦合

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

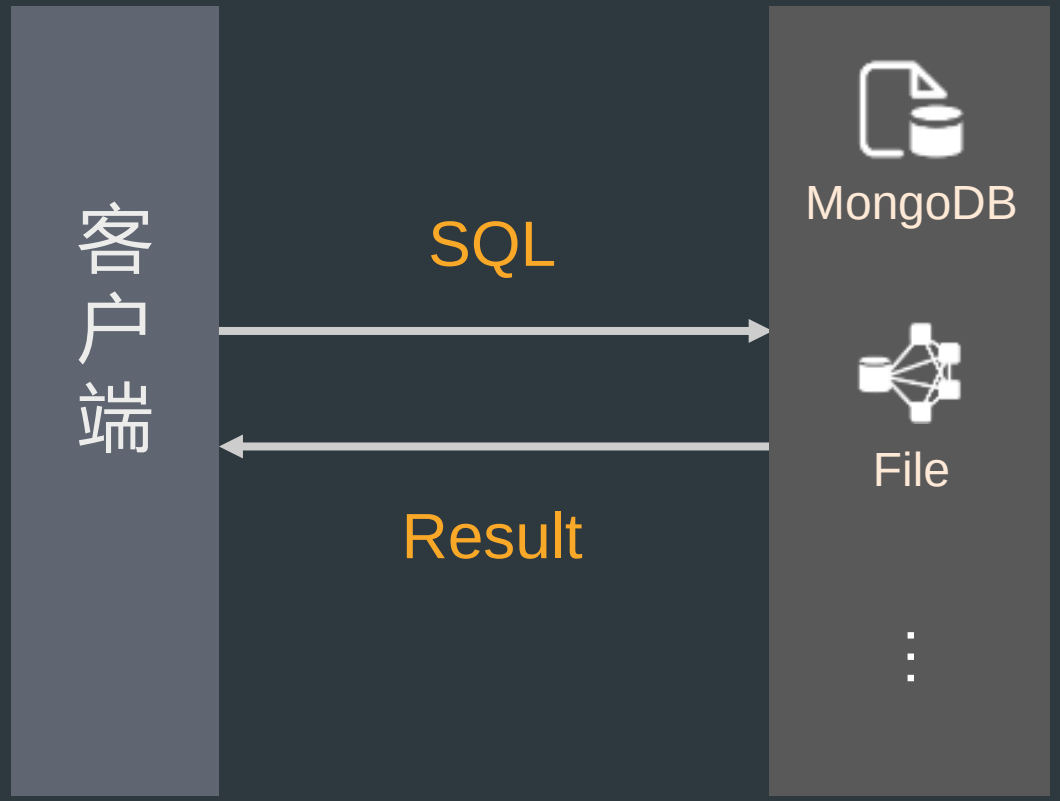
# 外部数据接口

- 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



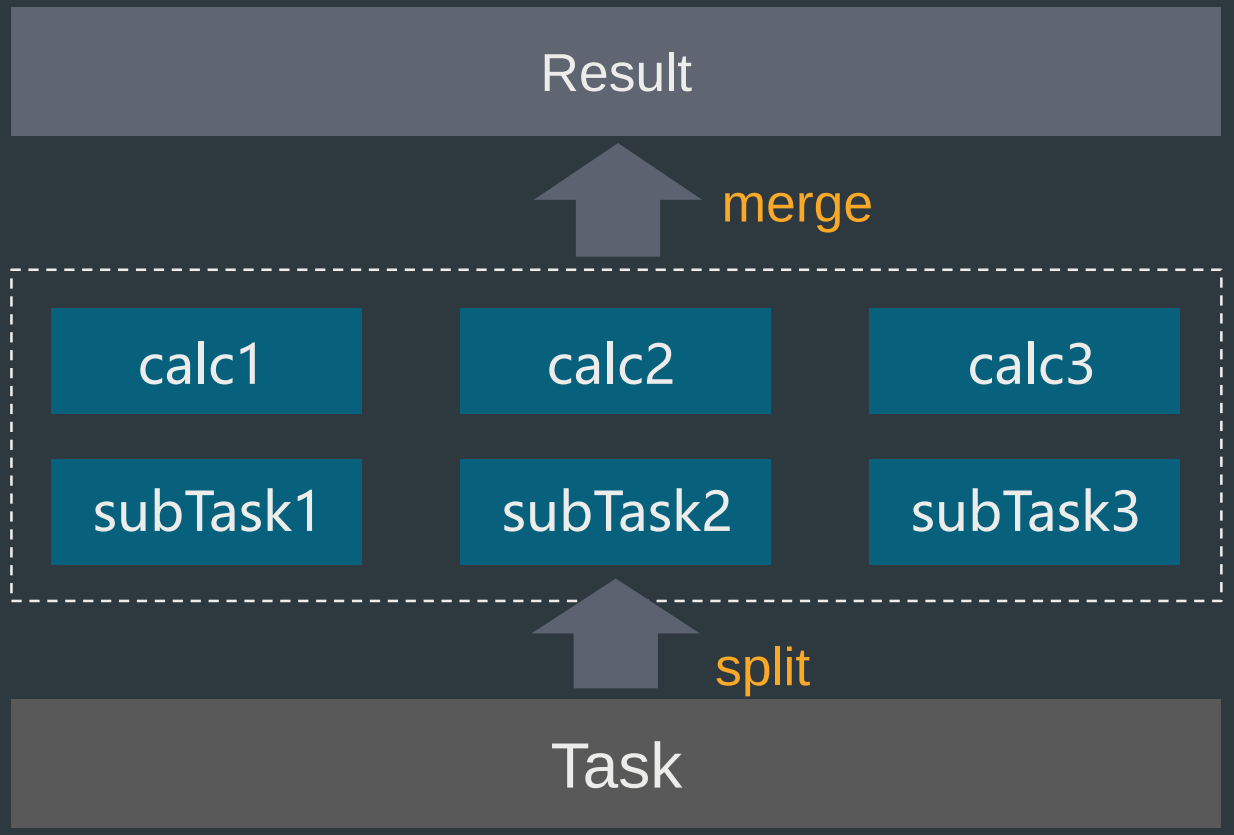

**集算器赋予**

**NoSQL和文件**

**SQL查询能力**

# 多线程并行

方便地针对单任务实施多线程计算



# 数据缓存

## 提供多样缓存数据存储方式

- 压缩数据
- 自由列存
- 多层序号主键
- 多层复式表
- 缓存索引

# T+0查询

## 目前T+0在线计算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

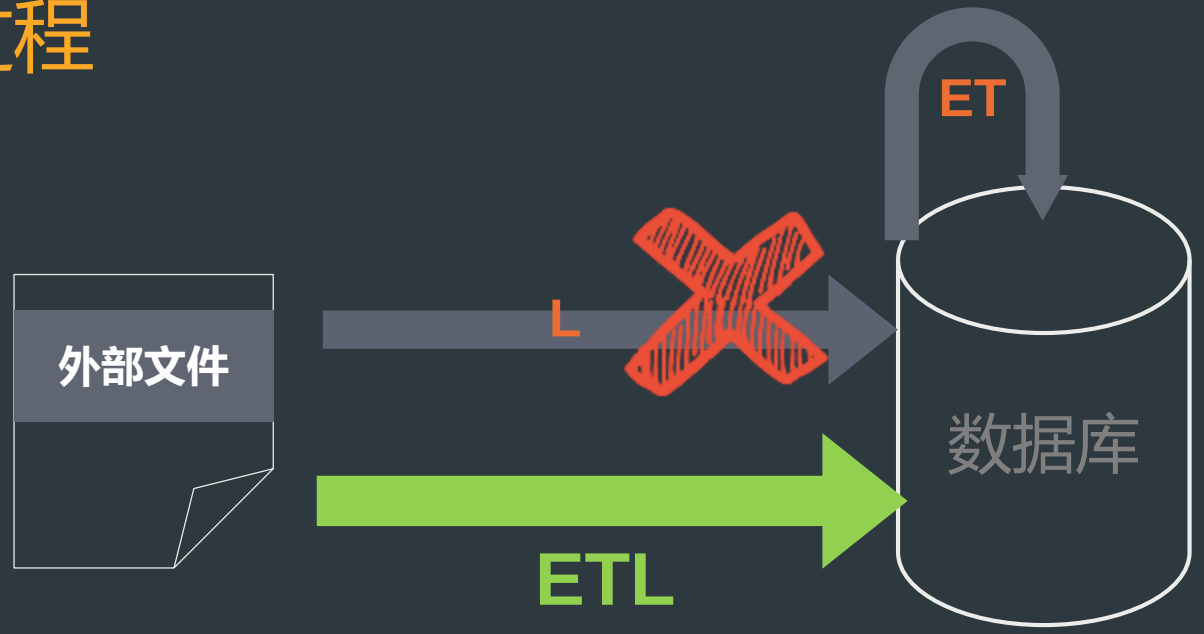
# ETL数据处理

## 缺乏库外计算能力扰乱ETL过程

ETL？ ELT？ LET？

加大数据库负担

## 库外计算实现合理的ETL



# 应用解耦

使用JAVA做处理在线计算会导致与应用耦合度过高

## 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**

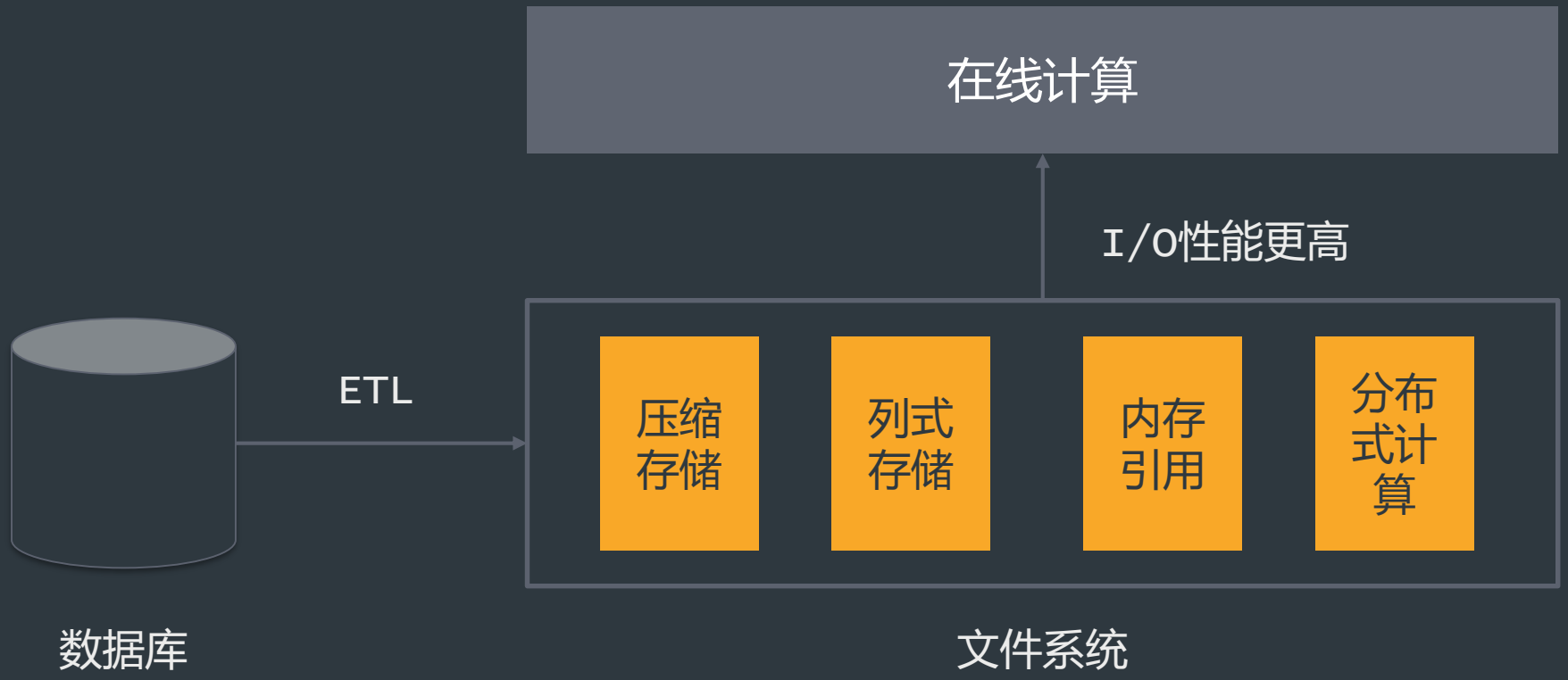
集算器脚本文件可以单独维护，方便模块化

**容易热切换**

集算器是解释执行的语言，很容易做到热切换

# 数据库解耦

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




分布式
计算
横向扩容
支持横向扩展scale-out
无中心
无中心避免单点失效
负载均衡
与容错
提供集群负载均衡与容错机制
外存数据
分布
提供自由的外存数据冗余方案
内存数据
分布
提供备胎式冗余方案
集群维表
支持大维表集群

# 过渡页

CONTENTS


01 数据计算中间件

02 应用场景

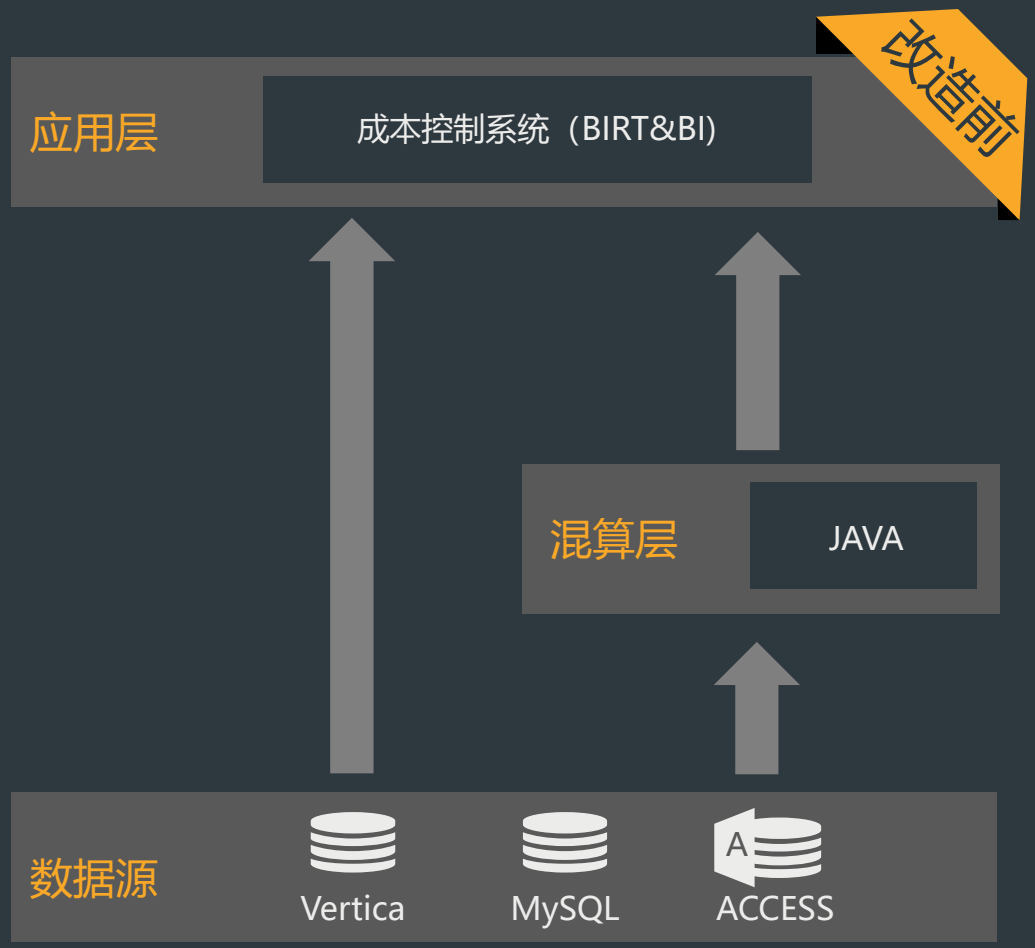
03 特征与标准

04 相关技术

05 集算器esProc

06 应用案例

# 某保险公司-库外存储过程





**尽量用单源，万不得已多源用JAVA硬编码**

**使Vertica支持存储过程，方便跨源计算**

# 应用效果

## 用户评价

“The best use for us is to pass parameters to the Vertica database.”

“ Each cell becomes a data array that are easy to use, compare and manipulate. It is very logical and you have made it user friendly.”

## 应用效果

Vertica与MySQL等跨源混算

过程化算法实施简单，友好

BIRT数据准备简单高效

# 某大型银行-CUBE重构



**旧系统维护困难、跑批时间长、查询性能差**

**并行计算、文件数据优化结构，提升性能**

# 应用效果

## 用户评价

集算器-仓库版 很好解决了高并发访问、大数据量计算造成的系统响应时间过长的问题！用集算器将高频次热点数据前置，构建数据计算中间层，可以说是最佳解决方案，在很多场景下要优于价值百万的数据库产品！

### ETL时间

8小时 -> 2小时内

### 架构变化

单机 -> 分布式

### 开发与性能

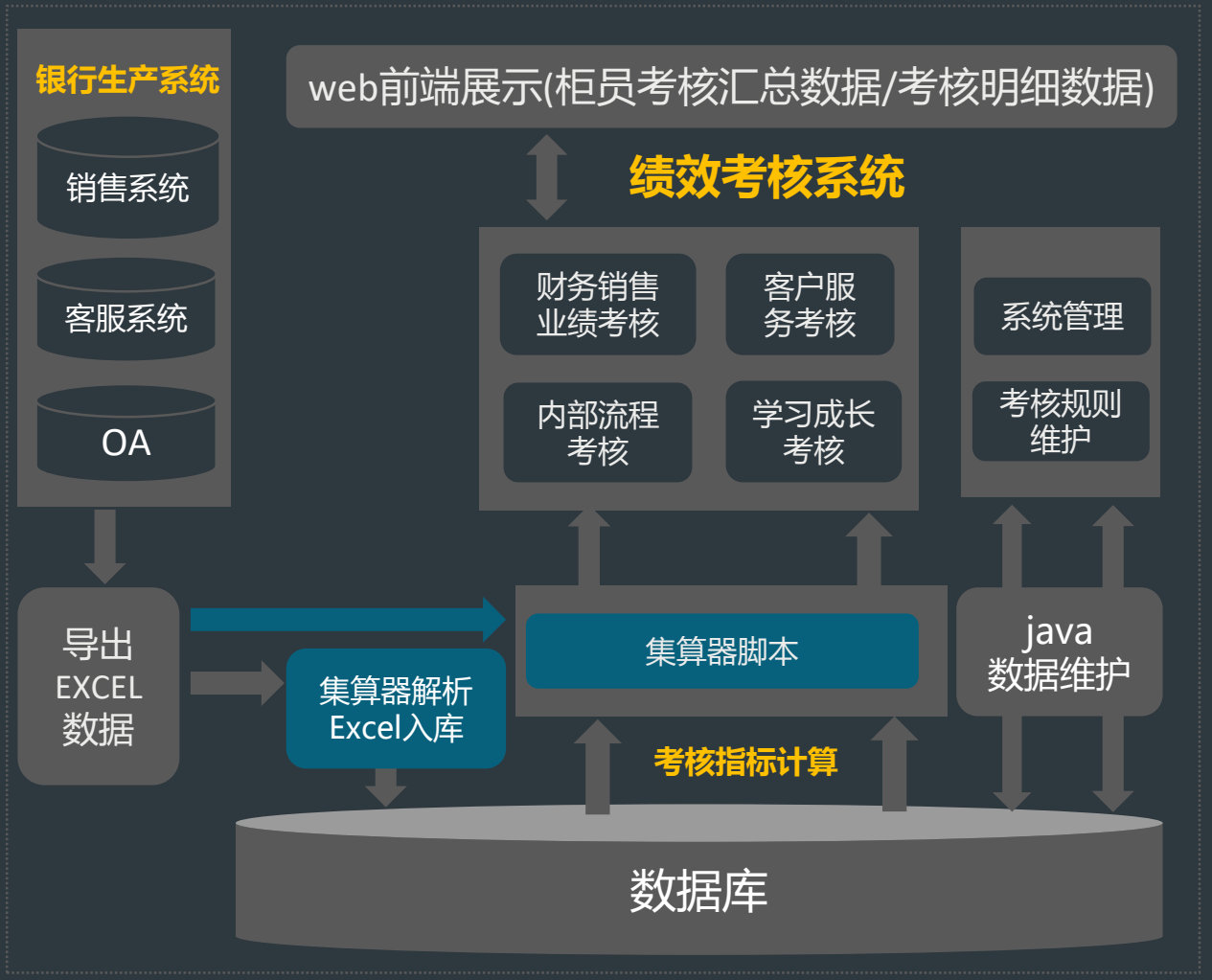
代码量：3500行->80行

性　能：420秒->3秒

| 指标 | 旧方式 | 新方式 | 提升 |
| --- | --- | --- | --- |
| 代码量 | 3500行 | 80行 | 43倍 |
| 工作量 | 未知 | 7人天 | -- |
| 执行性能 | 150-420s | 1-3s | 140倍 |
| 可维护性 | 困难 | 容易 | 质变 |
| 可优化性 | 困难 | 容易 | 质变 |

# 某国有银行-计算中间件


银行生产系统
销售系统
客服系统
OA
web前端展示(柜员考核汇总数据/考核明细数据)
绩效考核系统
财务销售业绩考核
客户服务考核
内部流程考核
学习成长考核
系统管理
考核规则维护
导出EXCEL数据
java程序
java程序
存储过程
java数据维护
考核指标计算
数据库


银行生产系统
销售系统
客服系统
OA
web前端展示(柜员考核汇总数据/考核明细数据)
绩效考核系统
财务销售业绩考核
客户服务考核
内部流程考核
学习成长考核
系统管理
考核规则维护
导出EXCEL数据
集算器解析Excel入库
集算器脚本
java数据维护
考核指标计算
数据库

# 应用效果

## 降低开发成本

全部44种Excel解析由原来的30人天下降到6人天

## 提升开发效率

每种Excel解析代码量由100行变为3行

## 降低维护成本

除代码量简短易维护外；脚本热部署即修改即生效

| 指标 | JAVA编程 | 集算器 | 提升 |
| --- | --- | --- | --- |
| 代码量 | 4400行 | 132行 | 33倍 |
| 工作量 | 30人天 | 6人天 | 5倍 |
| 可维护性 | -- | 容易 | 质变 |
| 可优化性 | -- | 容易 | 质变 |

# 某大型产权交易所-数据集市

改造前

业务接入层

应用层 查询 报表

数据源 ORA ORA

新系统

业务接入层

应用层 手工导出Excel

数据源 ORA ORA

老系统

**新老系统数据格式、规范不同导致无法互通**

改造后

业务接入层

应用层 查询 报表 自动生成Excel

集算器DM

标准应用接口

分布式计算节点

统一数据存储（集算器数据文件）

统一数据采集 – 集算器ETL

数据源 ORA ORA

新系统

数据源 ORA ORA

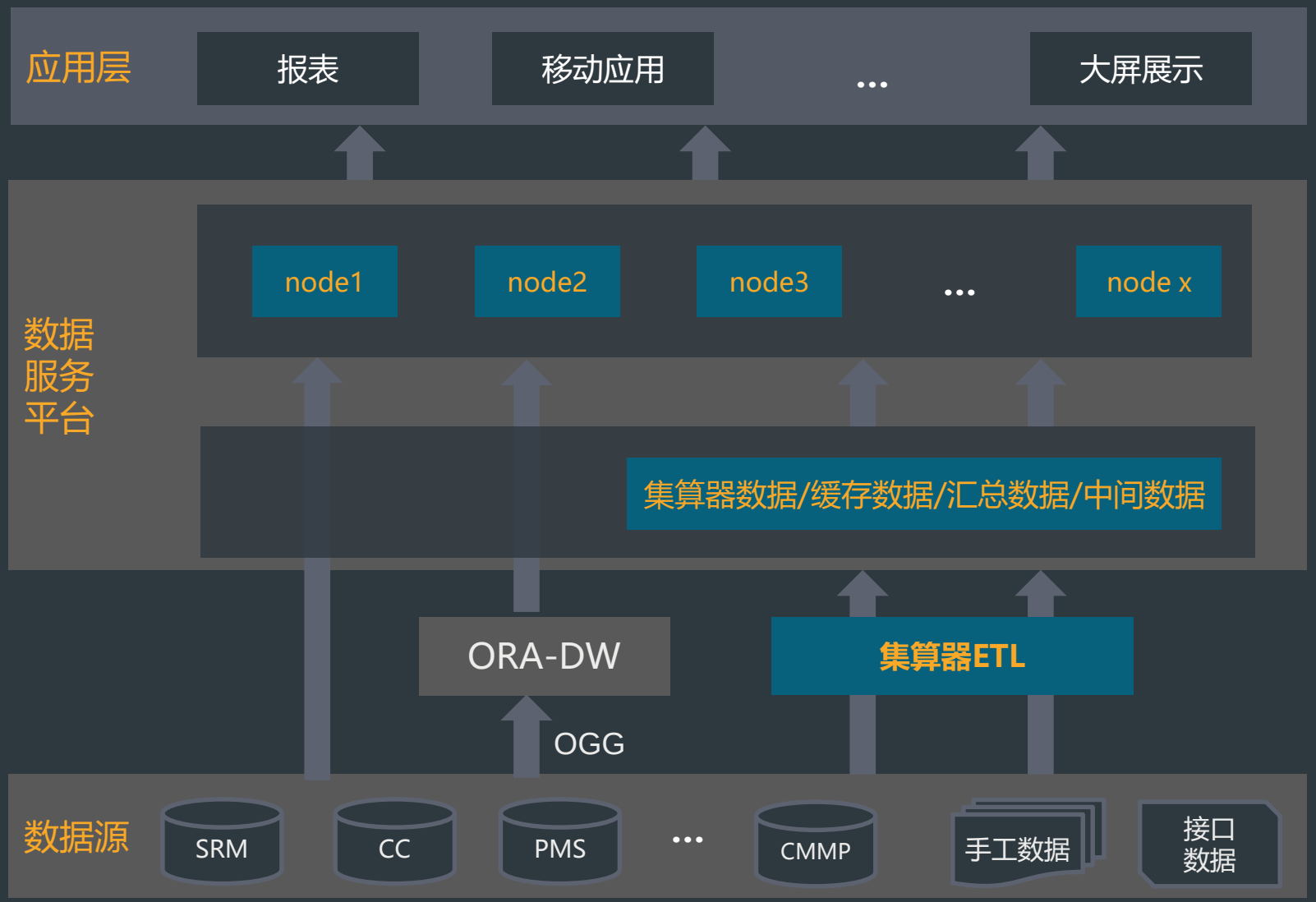
老系统

**实现系统间数据互通，统一管理，计算复用**

# 应用效果

| 异构源数据清洗 | 文件系统存储 | 计算逻辑复用 | 易于横向扩容 |
| --- | --- | --- | --- |
| 多源混算基于多个异构源同时ETL<br>通过十几个函数即完成复杂清洗计算 | 使用文件系统统一存储，节约了（关系）数据库的成本，且计算性能更高 | 一个业务逻辑一处编码实现，避免多个应用对于同一个指标计算结果不同的问题 | 数据集市中的集算器节点可以方便的横向扩展，支持数据量的不断增长 |

# 某航空-数据服务平台


应用层
报表
移动应用
...
大屏展示
数据
服务
平台
node1
node2
node3
...
node x
集算器数据/缓存数据/汇总数据/中间数据
ORA-DW
集算器ETL
OGG
数据源
SRM
CC
PMS
...
CMMP
手工数据
接口
数据

# 应用效果

应用直接基于集算器开发，屏蔽底层数据源之间的差异，规范了数据管理

统一数据管理

基于底层异构数据源进行混合计算，无需先进行数据同步

多源混合计算

将历史数据归档到文件系统，节约了数据库成本，可直接基于文件数据分析

历史数据归档

基于高效算法与并行计算机制，将数据计算性能提升了5-20倍

提升计算性能

横向扩展为未来航空公司数据的对接提供了足够的横向扩容能力

支持横向扩容

# 金融业-订阅系统


XX门户系统
应用层
门户应用
大数据可视化
固定报表
报表订阅
自助报表
系统管理
... ...
数据系统层
集算器分布式计算引擎
批量订阅报表计算
数据转换接口
集算器数据文件
Excel数据文件
记录日志信息
Control-M -> Perl脚本
命令调用
调度引擎
DTS数据处理
数据状态验证
数据加工就绪
ODS数据处理
数据文件校验
FTP推送数据文件（txt,verf）
数据源
DTS（SQLServer）
ODS数据文件
其它数据

# THANKS

创新技术 推动应用进步