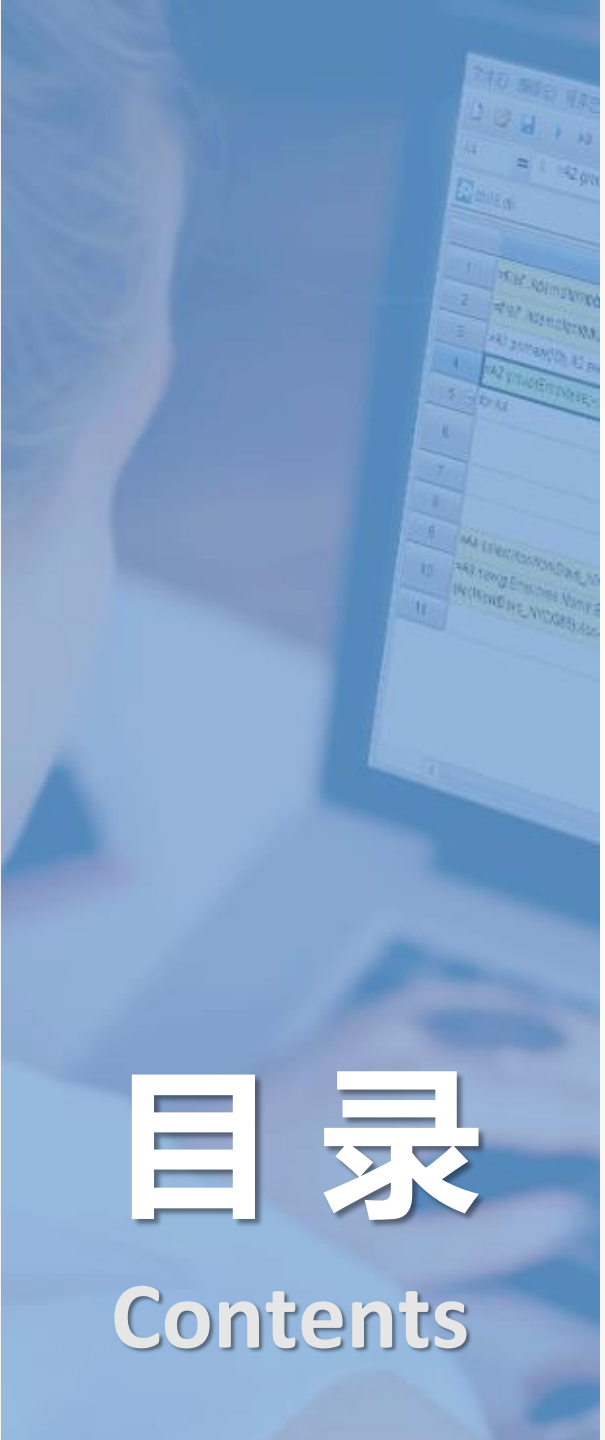
润乾
RAQSOFT

集算器：数据中台计算引擎

润乾软件出品



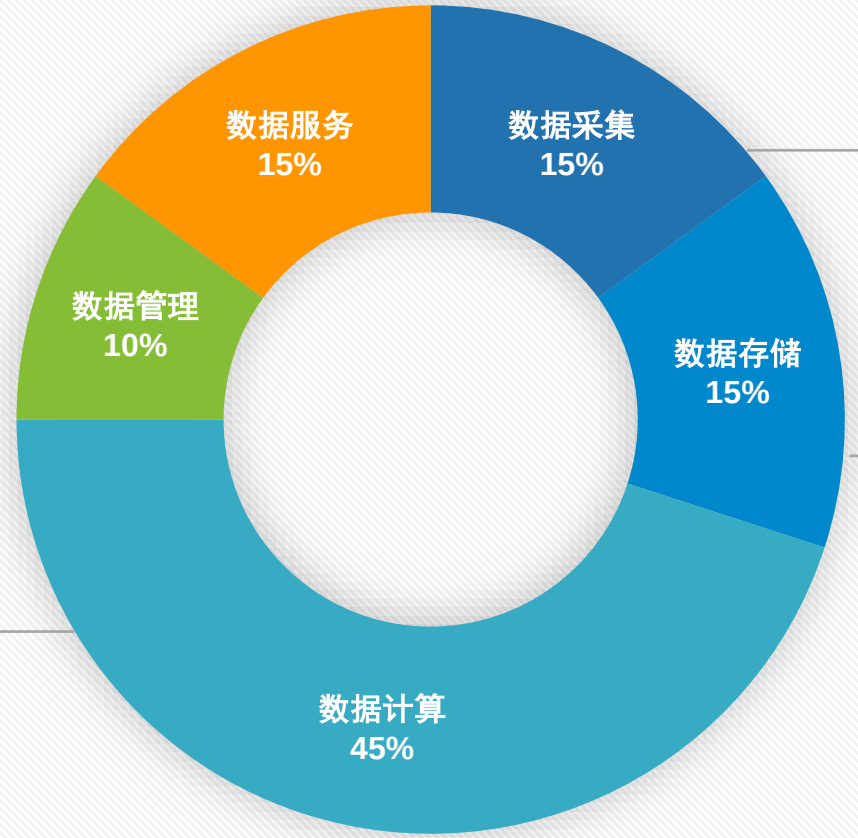
目录

Contents

数据中台回顾

从中台架构说起，一个典型的中台架构

各部分权重与主要实现技术

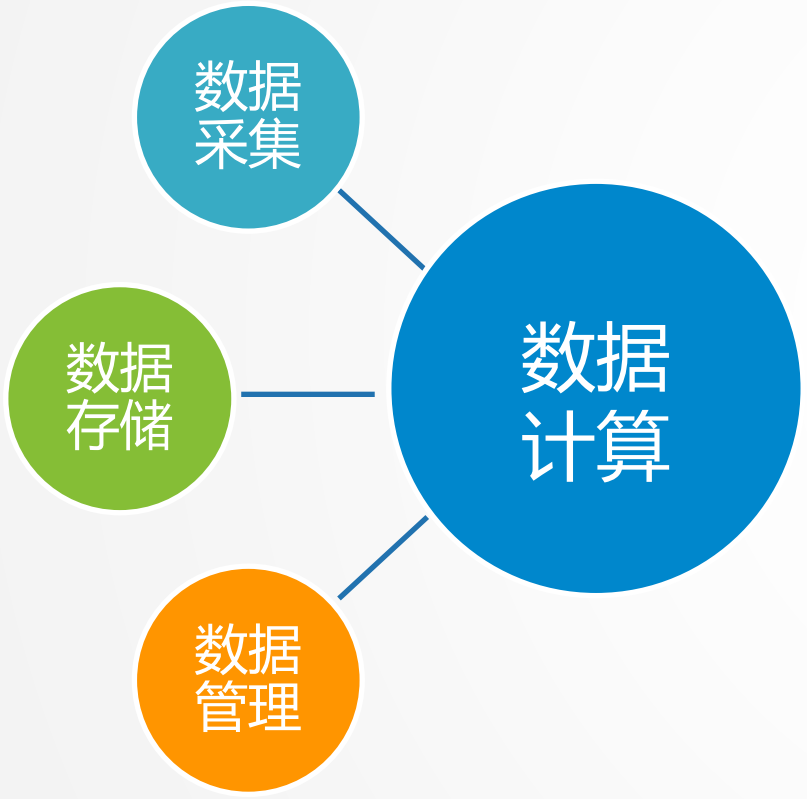


ETL工具，可使用如Kettle/Sqoop/Kafka/Flume

数据库与文件系统，可使用如MySQL/Hive/MongoDB/HDFS

计算引擎（框架），如MapReduce/Spark/Presto/Redis

数据计算的重要性



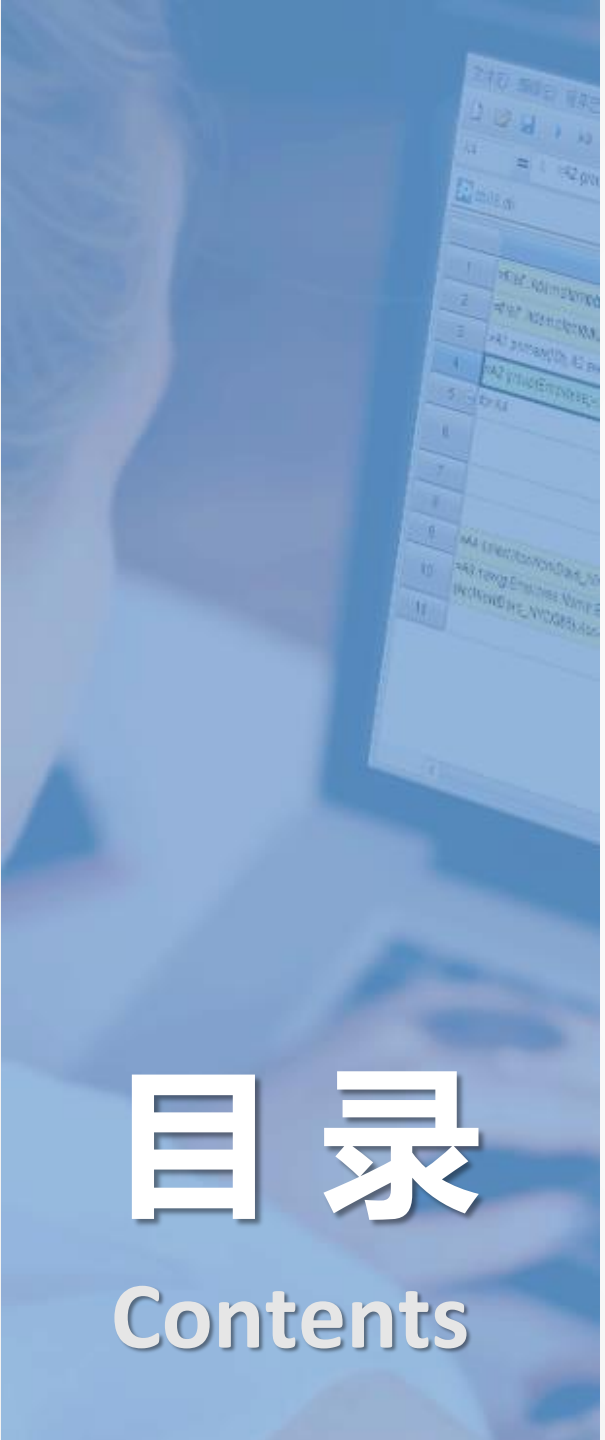
中台中其他部分都是为计算服务的

数据计算是中台的核心

- 数据计算是中台为前台提供数据服务的关键
- 数据计算伴随中台整个生命周期不断新增修改（业务驱动）
- 数据计算仍以结构化数据为主

聚焦关注点

结构化数据计算是数据计算的核心

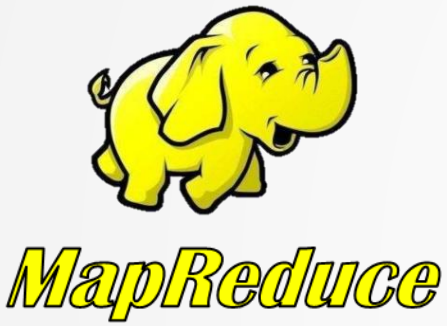


目录

Contents

当前实施数据计算有哪些技术

数据中台的数据计算采用一些主流的计算引擎（框架）

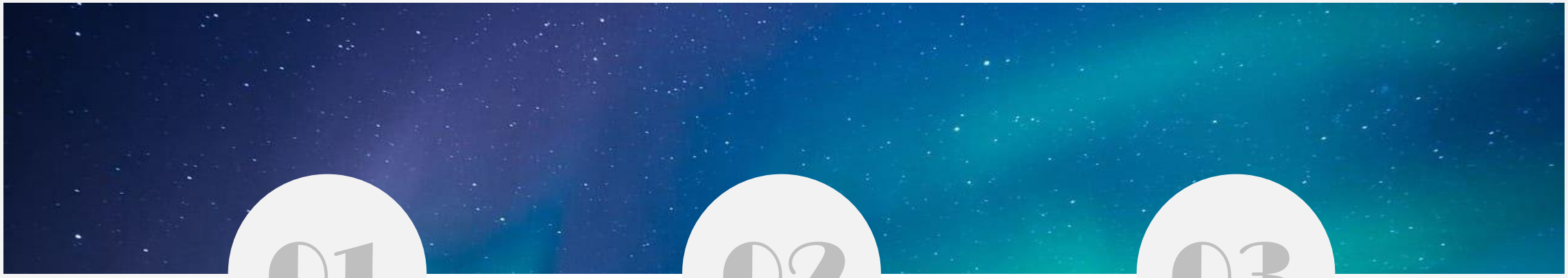


使用这些技术进行数据计算时，都需要编写大量硬编码
（以Java为主，少量使用SQL、Scala等）

*SQL本身就不适合编写中台计算中常见的过程计算，能提供完善SQL支持的持久化数据库过于沉重，也较少在中台使用；中台计算引擎提供的SQL能力更弱，难以支撑复杂业务逻辑，经常要写UDF辅助

*Scala对团队技能要求高，Spark架构沉重，体系封闭，资源消耗多，常常只用于某些专门场景

当前数据计算存在的问题



01

开发难

JAVA缺少结构化计算类库，计算实现复杂度高，实现成本高

02

性能难保障

算法性能取决于编码人员水平，在分布式算法实现和优化方面尤其难以保障

03

运维困难

代码冗长不利于修改维护，且JAVA不支持热切换，计算逻辑修改无法实时生效

举例：一段实现简单分组统计的Java代码

```java
public static void groupCount(){
    groupBy mainJava = new groupBy();
    List<MainBean> list = mainJava.initList();
    List<MainBean> mainList = new ArrayList<MainBean>();
    Map<String, MainBean> map = new HashMap<String, MainBean>();
    MainBean totalBean = null;
    String employeeId = null;
    for(MainBean mainBean : list) {
        employeeId = mainBean.getEmployeeId();
        if (!map.containsKey(employeeId)) {
            totalBean = new MainBean();
            totalBean.setEmployeeId(employeeId);
            totalBean.setBuyNum(mainBean.getBuyNum());
            totalBean.setGoodsPrice(mainBean.getGoodsPrice());
            totalBean.setTotalBuyNum(mainBean.getTotalBuyNum());
            totalBean.setFreight(mainBean.getFreight());
            totalBean.setTotalGoodsPrice(mainBean.getGoodsPrice() * mainBean.getBuyNum());
            totalBean.setTotalPrice(mainBean.getGoodsPrice() * mainBean.getBuyNum() + mainBean.getFreight());
            mainList.add(totalBean);
            map.put(employeeId, totalBean);
        } else {
            totalBean = map.get(employeeId);
            totalBean.setTotalBuyNum(totalBean.getTotalBuyNum() + mainBean.getTotalBuyNum());
            totalBean.setFreight(totalBean.getFreight() + mainBean.getFreight());
            totalBean.setTotalGoodsPrice(totalBean.getTotalGoodsPrice() + mainBean.getGoodsPrice() * mainBean.getBuyNum());
            totalBean.setTotalPrice(totalBean.getTotalPrice() +
                    mainBean.getGoodsPrice() * mainBean.getBuyNum() + mainBean.getFreight());
        }
    }
```

这么长的代码仅仅实现了单个字段分组单个字段汇总的功能

热切换的三种实现手段

JAVA动态编译技术

实现很复杂，实际应用中很少使用

借助容器

使用Docker等容器技术实现热切换

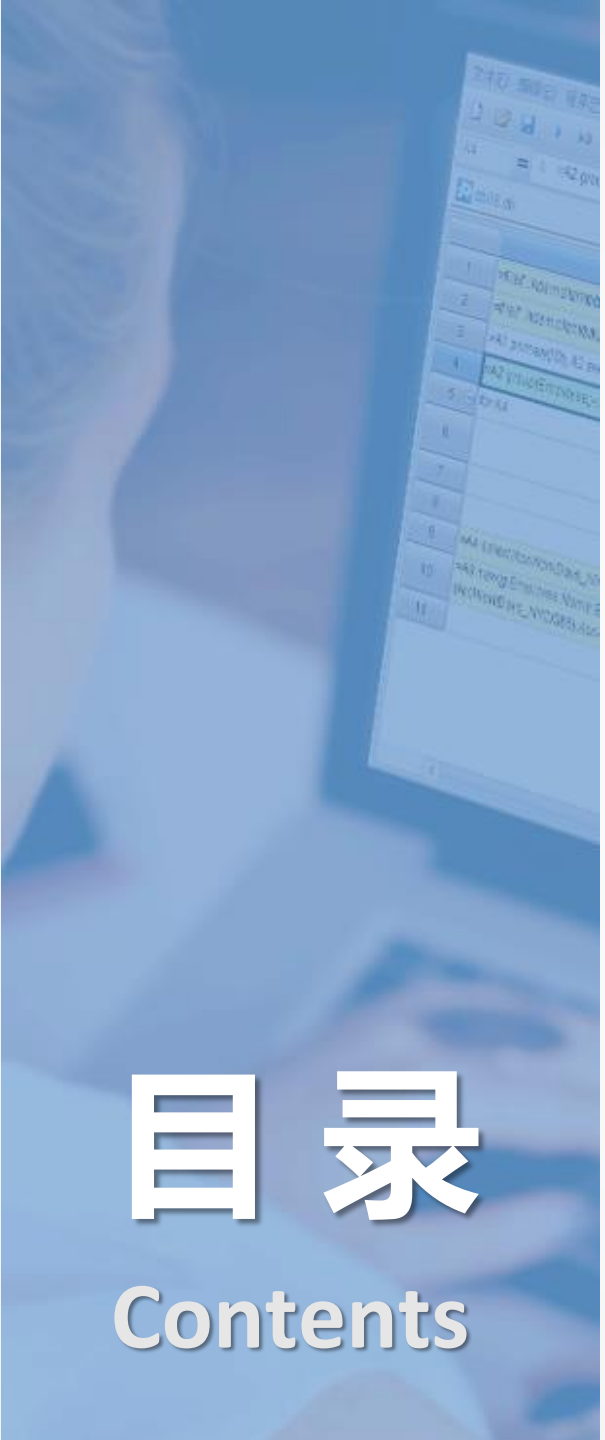
借助框架

使用SpringCloud等服务治理框架，配置多节点实现热切换

目前的三种实现方式都比较绕，使用起来也复杂笨重，缺少原生解决方案

理想的计算引擎特性

开发维护简单	计算类库丰富，算法实现简单，开发快； 支持热切换，算法修改实时生效
运算性能高效	提供高性能算法，支持分布式计算，允许横扩展计算能力
应用结构合理	提供多样性数据源支持，可以进行跨数据源混合计算； 提供标准数据输出接口，数据服务易封装 支持JAVA调用，可以与JAVA无缝结合使用

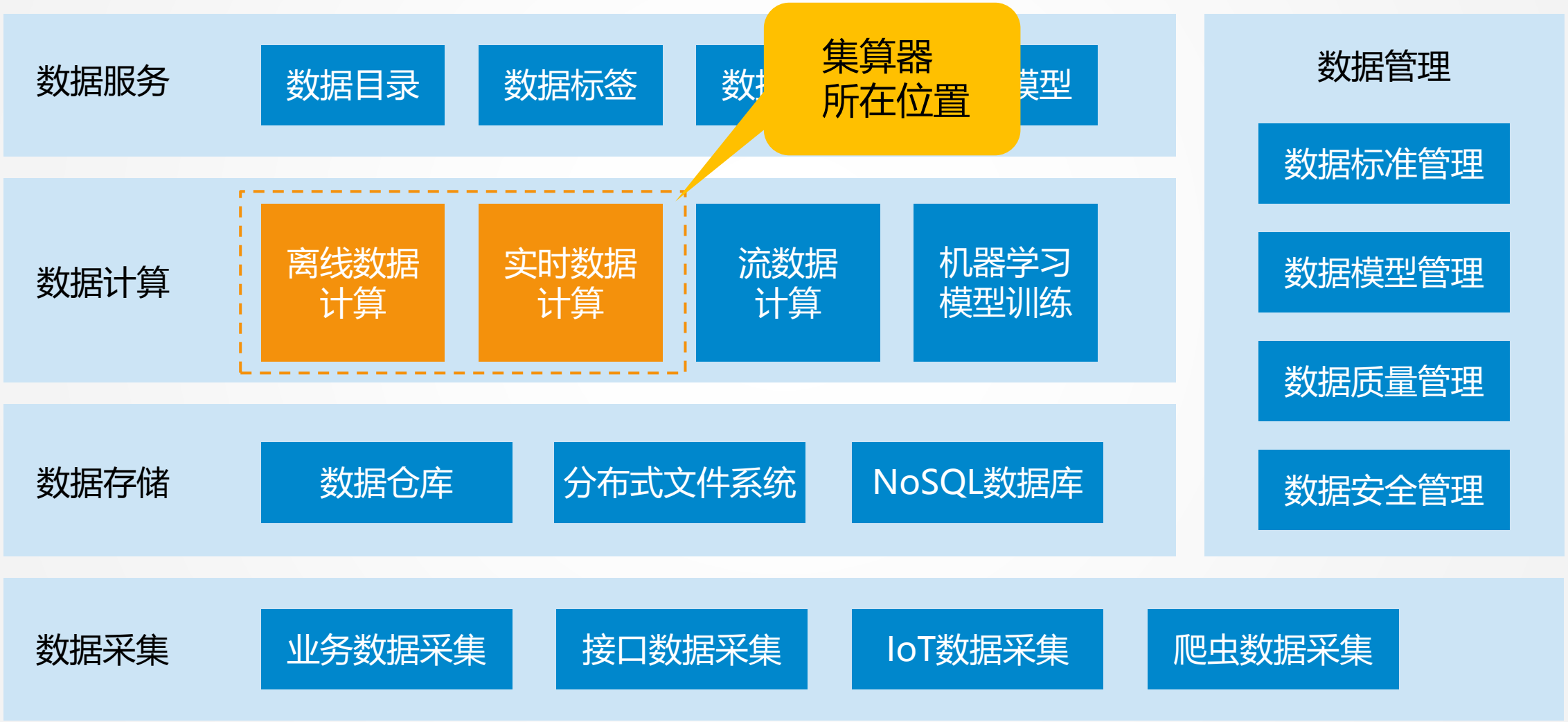


目录

Contents

集算器在数据中台中的位置

基于这样的背景，我们开发了专门用于中台数据处理的计算引擎-集算器



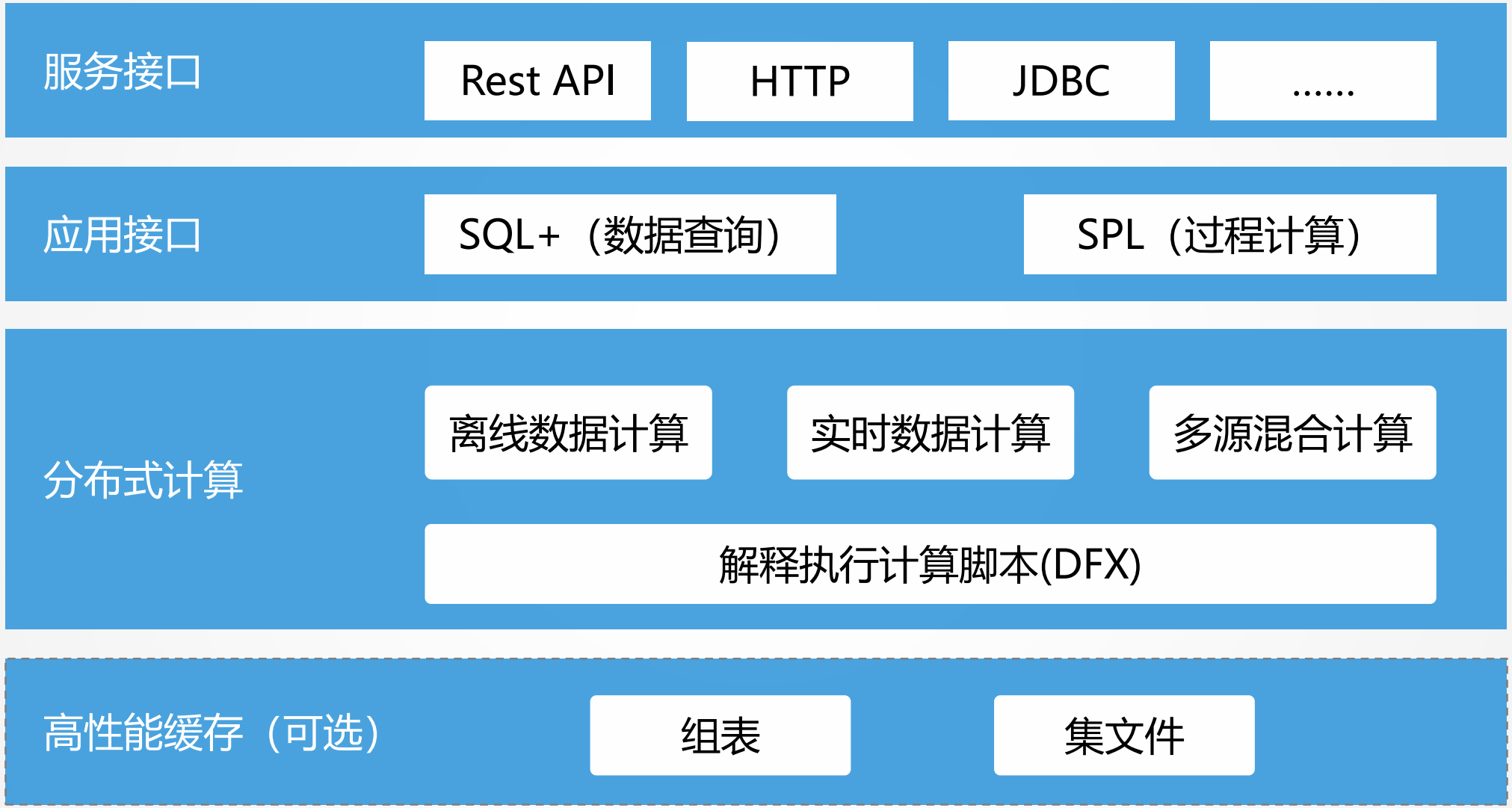
集算器是什么?

专门用于结构化数据计算的高性能

计算引擎

*集算器提供专有的计算语言SPL（Structure Process Language，面向过程的结构化数据计算语言）

集算器架构


服务接口
Rest API
HTTP
JDBC
.....
应用接口
SQL+（数据查询）
SPL（过程计算）
分布式计算
离线数据计算
实时数据计算
多源混合计算
解释执行计算脚本(DFX)
高性能缓存（可选）
组表
集文件

集算器特性

开发快

热切换

高性能

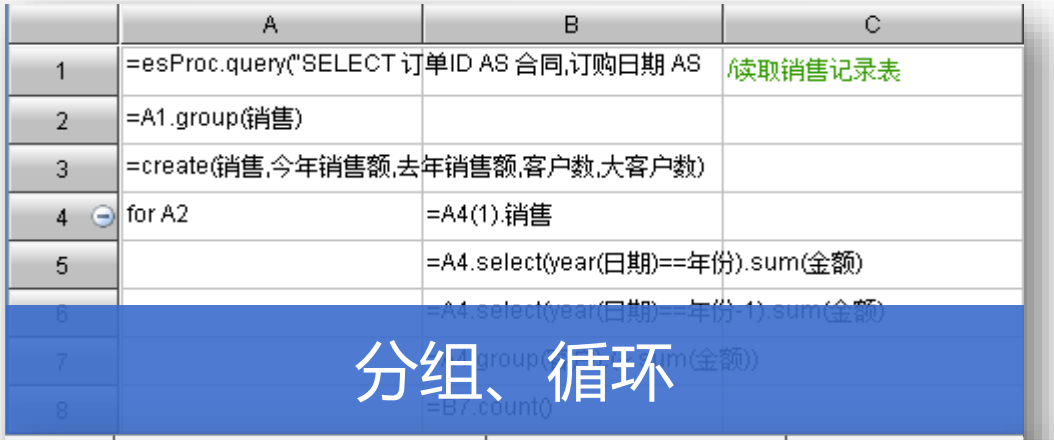
多源混算

多种数据输出接口

与JAVA无缝结合

丰富的运算类库

专门针对结构化数据设计，可完成各类复杂运算

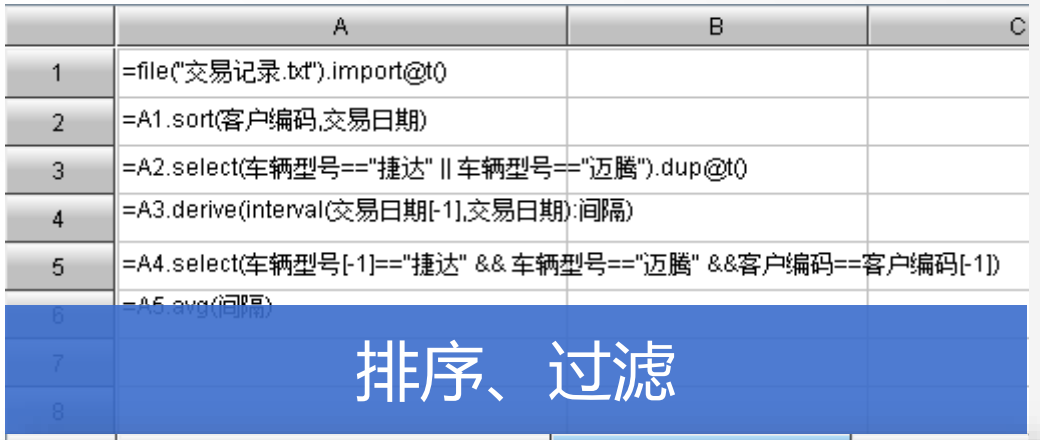


	A	B	C
1	=esProc.query("SELECT 订	单ID AS 合同,订购日期 AS	/读取销售记录表
2	=A1.group(销售)		
3	=create(销售,今年销售额,去	年销售额,客户数,大客户数)	
4	for A2	=A4(1).销售	
5		=A4.select(year(日期)==年份).sum(金额)	

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

相对JAVA代码更精简

相对使用JAVA实现数据计算，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，开发效率更高

写的更快更短

- 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

集算器SPL实现分组举例

这类基本运算还可以直接用SQL写：
SELECT name,count(name) FROM user/test/duty.xlsx GROUP BY name

常规分组

汇总每个人的值班天数

	A
1	=file(“/Users/test/duty.xlsx”).importxls@tx()
2	=A1.groups(name;count(name):count)

每组Top N

获取每个月、每个人、头三天的加班记录

	A
1	=file(“/Users/test/duty.xlsx”).importxls@tx()
2	=A1.groups(month(workday):mon,name;~.top(3):top3)

对位分组

按顺序分别列出使用Chinese、English、French 作为官方语言的国家数量

	A
1	=connect(“mysql”)
2	=A1.query@x(“select * from world.countrylanguage where isofficial= ‘T’ ”)
3	[Chinese,English,French]
4	=A2.align@a(A3,Language)
5	=A4.new(A3(#):name,~.len():cnt)

比SQL更易写

SPL甚至比SQL更易写易理解

举例：某支股票最长连续涨了多少交易日

```
1  select max(continuousDays)-1
2  from (select count(*) continuousDays
3        from (select sum(changeSign) over(order by tradeDate) unRiseDays
4              from (select tradeDate,
5                    case when closePrice>lag(closePrice) over(order by tradeDate)
6                    then 0 else 1 end changeSign
7                    from stock) )
8        group by unRiseDays)
```

SQL

	A
1	=stock.sort(tradeDate)
2	=0
3	=A1.max(A2=if(closePrice>closePrice[-1],A2+1,0))

SPL脚本

语法体系更容易描述人的自然思维！

思考：按照自然思维怎么做？

热切换

热切换

集算器脚本采用解释执行，服务修改可不停机热切换，即时修改即时生效

原生的热切换机制优于各种曲线实现手段

高性能

高效语法

结合SQL和JAVA的优点，即支持批量数据计算，也方便复杂过程计算，算法实施简单且更为高效

高性能存储

数据计算过程中可以使用集算器提供的私有数据存储格式，实现压缩、分段等机制提升读取运算效率，亦可作为数据缓存使用

分布式计算

集算器提供了单机多线程并行和多机分布式计算机制，支持横向扩展，进一步保障运算性能

高效语法

SQL+

在传统SQL的基础上，额外加入新的标记语法，用来描述数据特征等关键信息，从而弥补传统SQL描述能力不足的缺陷，大幅提高计算性能，并有效降低学习成本

SPL

面向多步骤的复杂计算，采用过程式编程，借助丰富的集合运算类库、离散性、深度集合化等特性，极大改善传统SQL或JAVA在实现复杂计算时难实现、难维护、性能低等情况

高性能算法引擎

遍历技术

- 延迟游标
- 聚合理解
- 有序游标
- 遍历复用
- 预过滤遍历

高效关联

- 外键指针化
- 外键序号化
- 有序归并
- 主子同维表一体化
- 单边HASH连接

高性能存储

- 有序压缩存储
- 自由列式存储
- 层次序号式定位
- 片状索引及缓存
- 倍增自由分段并行

分布式计算

- 抢先式负载均衡
- Fork-reduce
- 外存冗余式容错
- 内存备胎式容错
- 集群维表

高性能算法举例

聚合理解

	A	
1	=file(“data.ctx”).create().cursor()	
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

高复杂度的排序转换为低复杂度的聚合

遍历复用

	A	
1	=file(“order.ctx”).create().cursor()	准备遍历
2	=channel(A1).groups(product;count(1):N)	配置复用计算
3	=A1.groups(area;sum(amount):amount)	遍历，并获得分组结果
4	=A2.result()	取出复用运算的结果

一次遍历可返回多个结果集

高性能存储，可用于数据缓存

高性能数据存储

私有数据存储格式，集文件、组表

文件系统存储形式

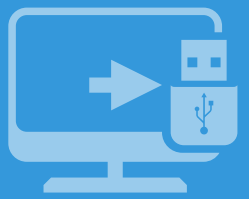
支持以树状目录方式按业务分类存储数据

集文件

倍增分段方式支持任意数量并行
自有高效压缩编码（减少空间；CPU占用少；安全）
泛型存储，允许集合数据

组表

行列混合存储
有序存储提高压缩率和定位性能
高效智能索引
倍增分段方式支持任意数量并行
主子表合一减少存储与关联
排号键值实现高效定位关联



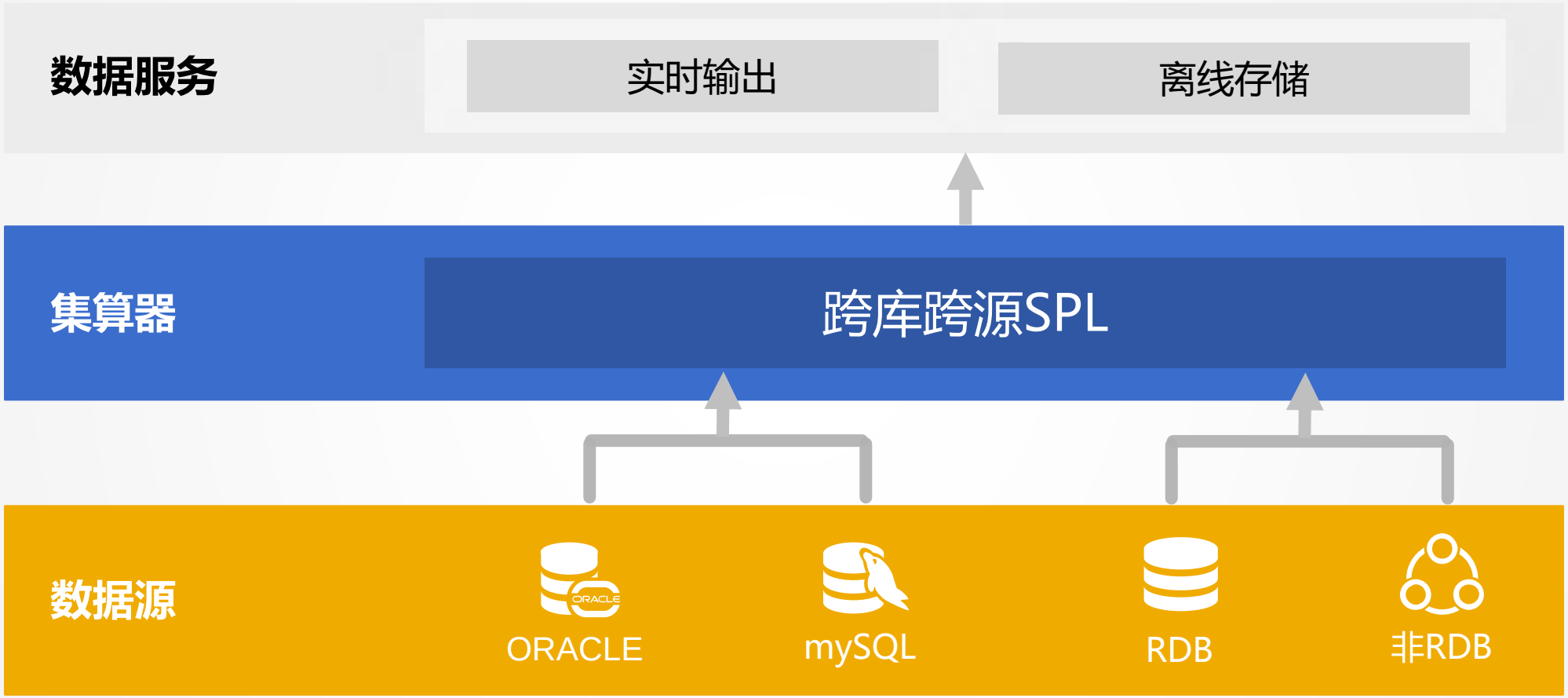
多样性数据源支持

JAVA计算能力难以胜任多样性数据源

集算器内置接口便捷访问

- 商用 RDBMS：Oracle、MS SQL Server、DB2、Informix
- 开源 RDBMS：MySQL、PostgreSQL
- 开源 NOSQL：MongoDB、Redis、Cassandra、ElasticSearch
- Hadoop家族：HDFS、HIVE、HBase
- 应用软件：SAP ECC、BW
- 文件：Excel、Json、XML、TXT
- 其他：Http Restful、Web Services、OLAP4j 、...

跨源混合计算支持



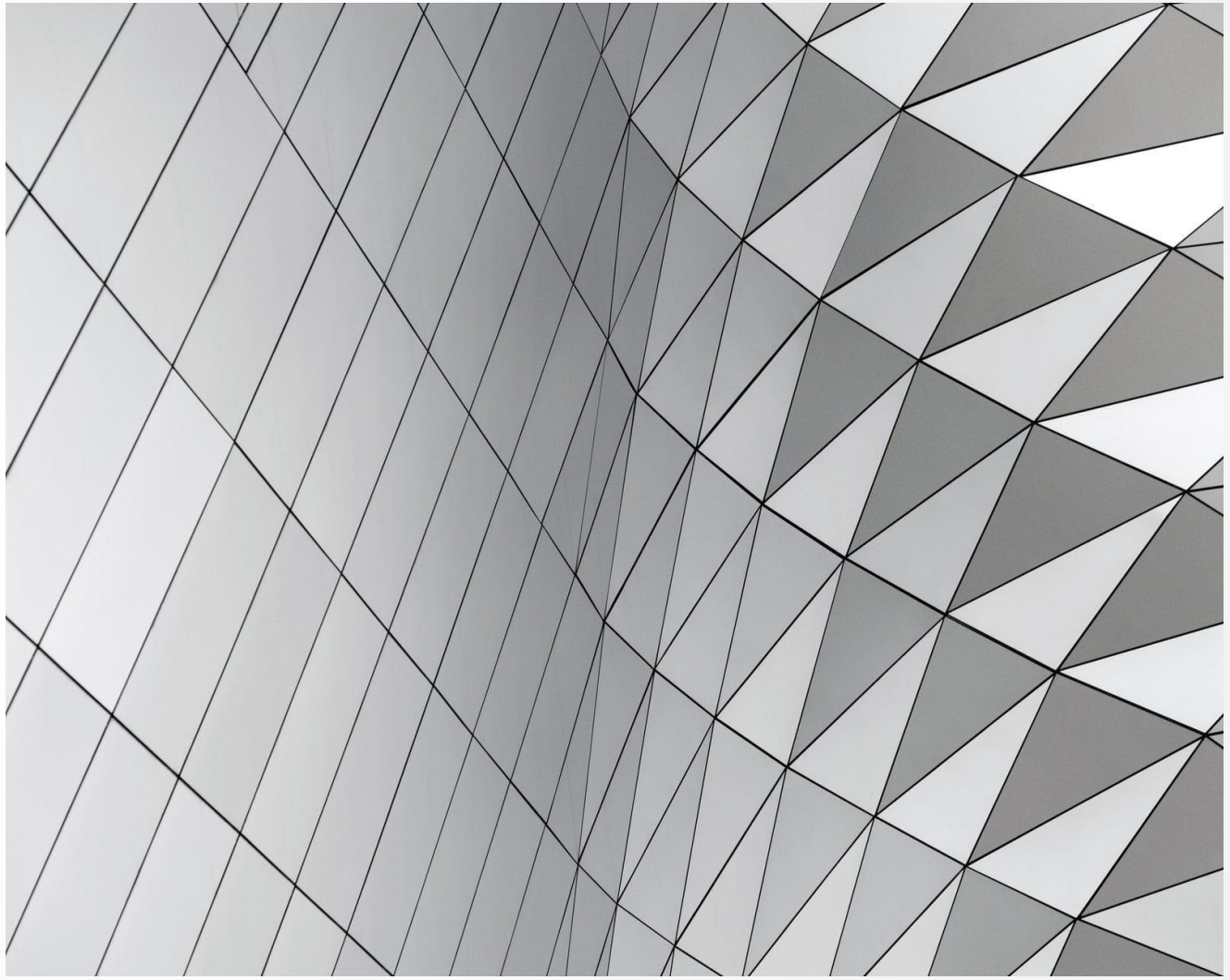
多样数据服务与接口

数据脱敏、权限控制

……

REST API/JDBC/ODBC/HTTP

……



与JAVA无缝结合

标准JAR包

与JAVA无缝结合使用

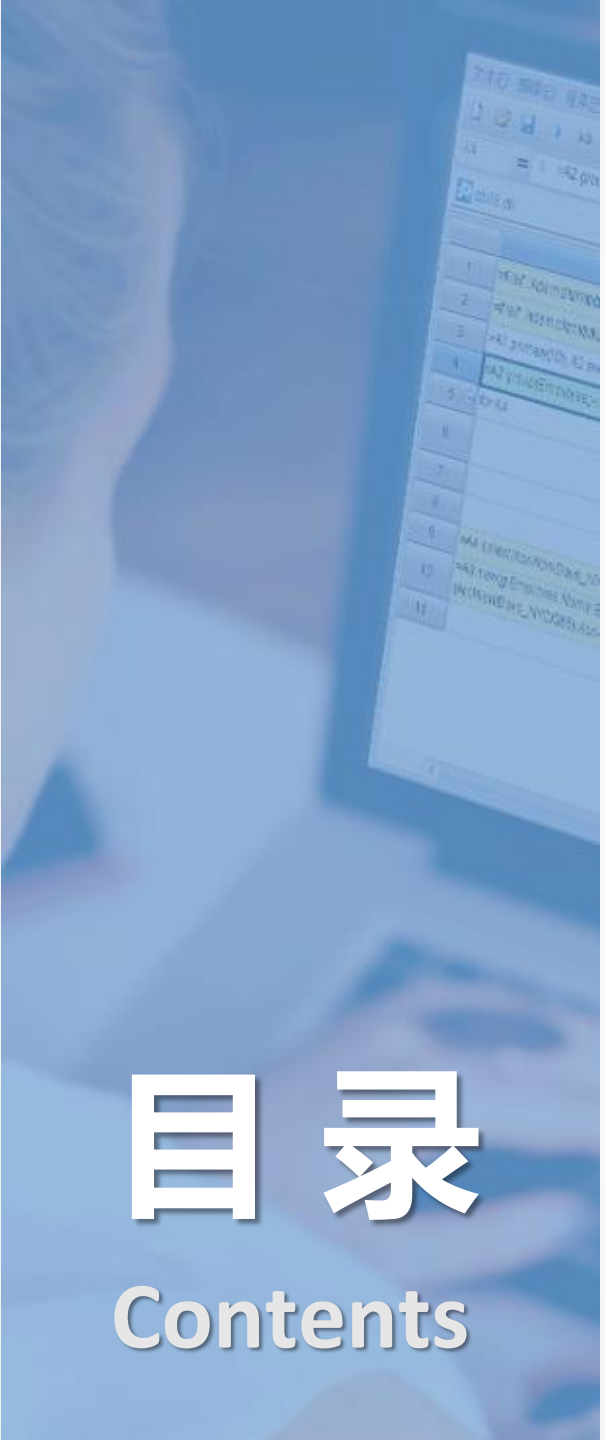
与JAVA无缝结合

Java代码

```
Connection con = null;
Class.forName("com.esproc.jdbc.InternalDriver");
con= DriverManager.getConnection("jdbc:esproc:local://");
//调用存储过程，其中CountName是dfx的文件名
st =(com. esproc.jdbc.InternalCStatement)con.prepareCall("call CountName()");
//执行存储过程
st.execute();
//获取结果集
ResultSet rs = st.getResultSet();
...
```

CountName即为SPL脚本名称

标准化调用接口



目录

Contents

总结：集算器对数据中台建设的意义

- 数据计算相对中台其他部分的特殊性在于其持续性，只要中台在运行，数据计算就会不断更新
- 利用集算器的特性可以大幅降低中台建设和运营成本

低成本