集算器

创新大数据计算引擎

新一代内存计算引擎

润乾软件出品

当前内存计算技术 – 概述

- 未到企业级，仅适用较小数据和低并发应用
- 计算性能不足，无法应对高并发和实时查询

轻型内存数据库

SQLite

Key-Value缓存

redis

- 一般用作高速存取器，并不能作为独立数据库使用
- 数据类型少，几乎无计算能力，比如GROUP/JOIN，大多依靠硬编码

- 成本相当高昂，且扩容能力有限
- 复杂算法开发难度大，需编写UDF
- 性能尚可，但仍不能达到最优

商业内存数据库

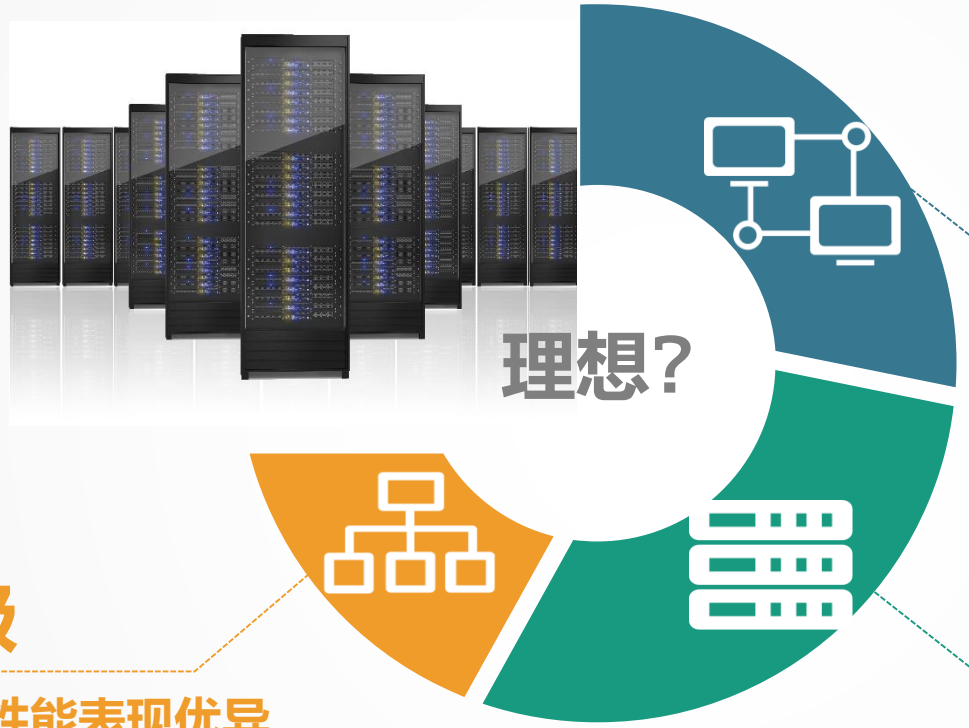
SAP HANA

内存计算框架

Spark

- 大规模甚至超大规模集群，体系沉重
- 因原生语言开发困难，接口逐步转向SQL，复杂算法仍然难写
- 内在机制和SQL体系导致性能不足

为什么需要内存计算？ – 更快！



高性能

- ✓ 一定要计算的够快
- ✓ 轻松应对实时查询
- ✓ 具备高并发的能力
- ✓ ...

易开发

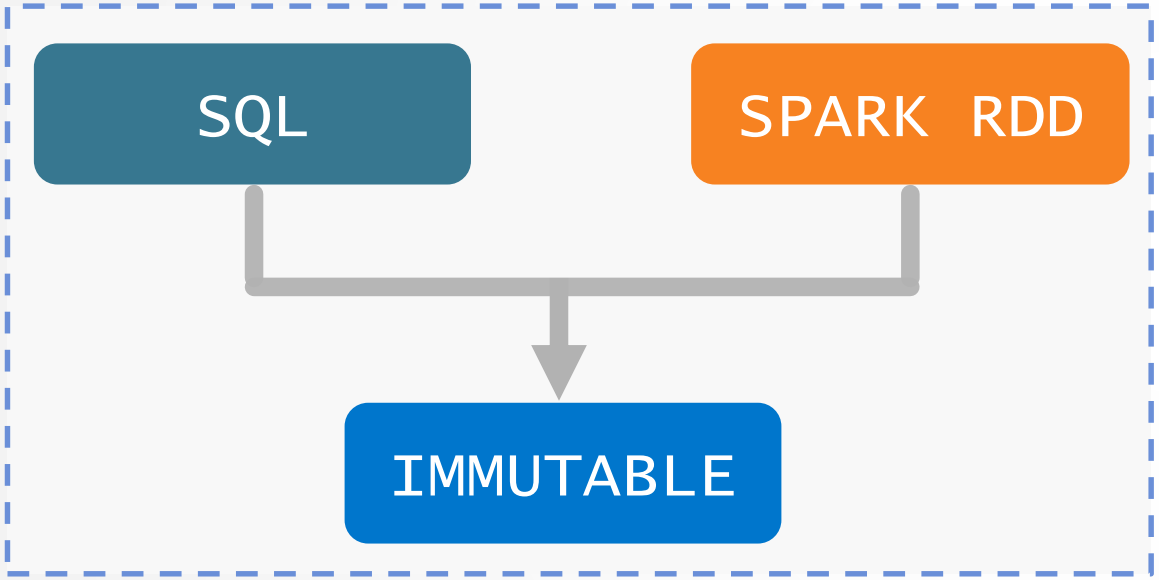
- ✓ 完善的开发和调试环境
- ✓ 提高开发效率
- ✓ ...

轻量级

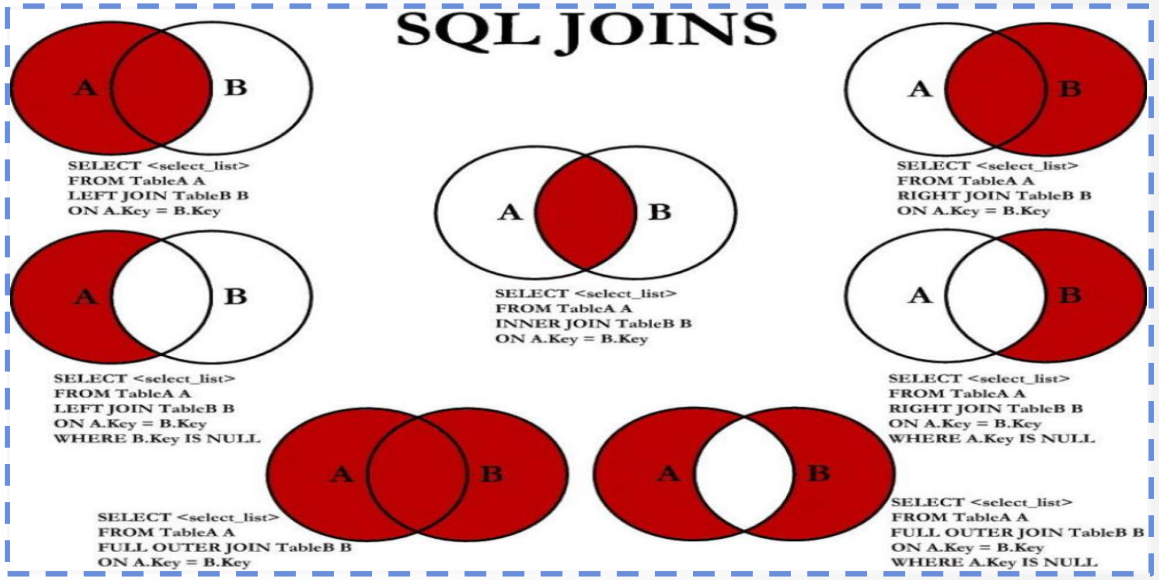
- ✓ 单机应用性能表现优异
- ✓ 中小规模集群
- ✓ 无缝应用集成
- ✓ ...

当前技术为什么不快?

SQL和SPARK RDD都是immutable风格，运算过程中会造成大量没必要的内存复制，降低运算性能！



SQL的JOIN运算不能利用内存预关联技术，执行过程中还要进行HASH计算和遍历比较，性能不佳！



局限性

- 数据一旦产生后不能修改！
- 各种运算等操作都会重新复制出新数据！

历史原因

- SQL发明于内存很小的时代，不能充分利用内存计算的特征！

高性能的内存计算 – SPL

为什么SPL会算得更快?

离散数据集模型

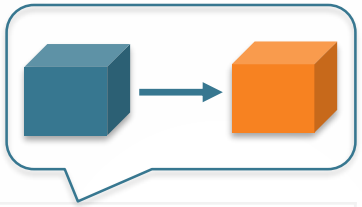
SPL

SPL高性能的原因

- ✓ 良好的**复用**、**引用**机制，减少数据复制；同样的内存，SPL能做**更多**的计算任务！
- ✓ 充分利用内存特性，外键可用指针实现**预关联**，JOIN可在常数时间内完成！
- ✓ 更多高性能优化算法：有序分组、归并关联、...

高性能应用举例 – SQL与SPL比较

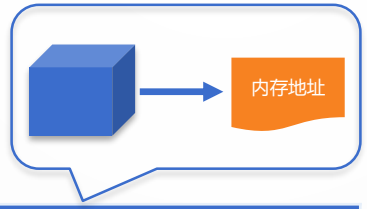
从合同表中筛选出1998年的销售记录



```
SELECT * FROM 合同表 WHERE YEAR(销售日期)=1998
```

SQL

查询过滤的结果总是新产生，这就会把数据再复制一遍，不仅消耗时间，而且占用更多内存！



	A
1	=合同表.select(year(销售日期)==1998)
2	

SPL

A.select()的计算结果，还是原成员构成的新集合，数据并没有复制，仅复制内存记录地址，速度更快且占用内存更小！

高性能应用举例 – 外键指针化

按照商品大类名称分组汇总订单金额

订单表	商品表	类别表
订单编号	商品编号	类别编号
商品编号	名称	大类名称
数量	类别编号	二级大类
…	单价	…
	…	

```
1  SELECT 类别表.大类名称,SUM(订单表.数量*商品表.单价) 订单金额
2  FROM 订单表,商品表,类别表
3  WHERE 订单表.商品编号=商品表.商品编号 AND 商品表.类别编号=类别表.类别编号
4  GROUP BY 类别表.大类名称
```

SQL

- SQL采用HASH JOIN算法解析关联，每次只能解析一个JOIN（须遍历一次数据），多表关联意味着**多次HASH计算和比对**，性能较差！而且书写繁复。
- ✓ SPL采用外键指针办法解析JOIN，对事实表**遍历一次**即可完成所有外键的内存**预关联**！

```
A
1  =订单表.groups(商品编号.类别编号.大类名称:大类名称;sum(数量*商品编号.单价):订单金额)
2  //计算订单金额,用.属性的方式引用到商品表的单价字段
```

SPL

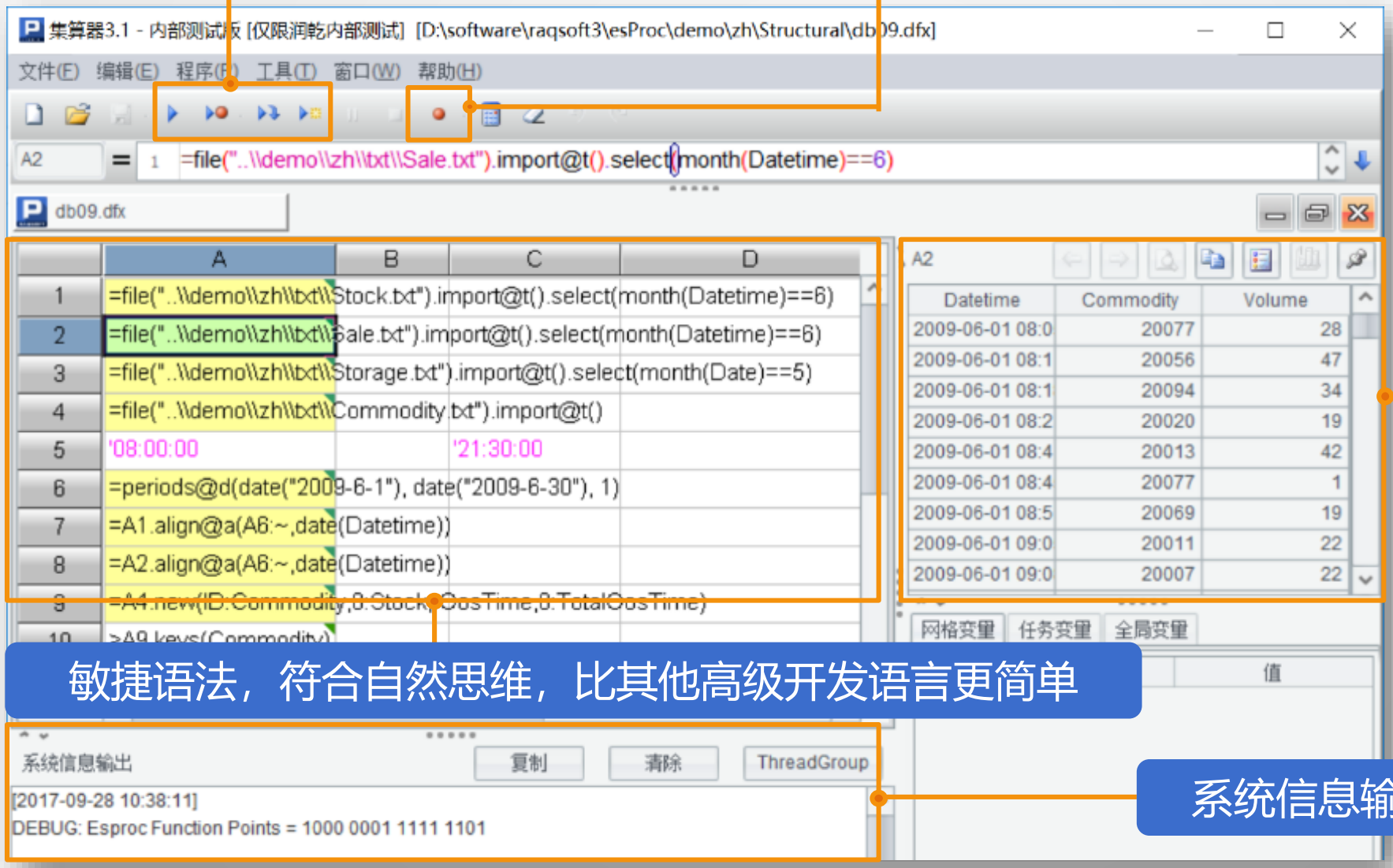
✓ 指针一旦建立即可**复用**，再次引用维表记录时，SPL**不再**做HASH计算，直接引用的性能更高且书写简单；而SQL不能做内存预关联，关联维表时还要**再次**计算HASH值并比对。

基于SQL的内存数据库也没法享受指针这一优势！

开箱即用的开发环境 – 集算器(SPL)

执行、调试执行、单步执行

设置断点

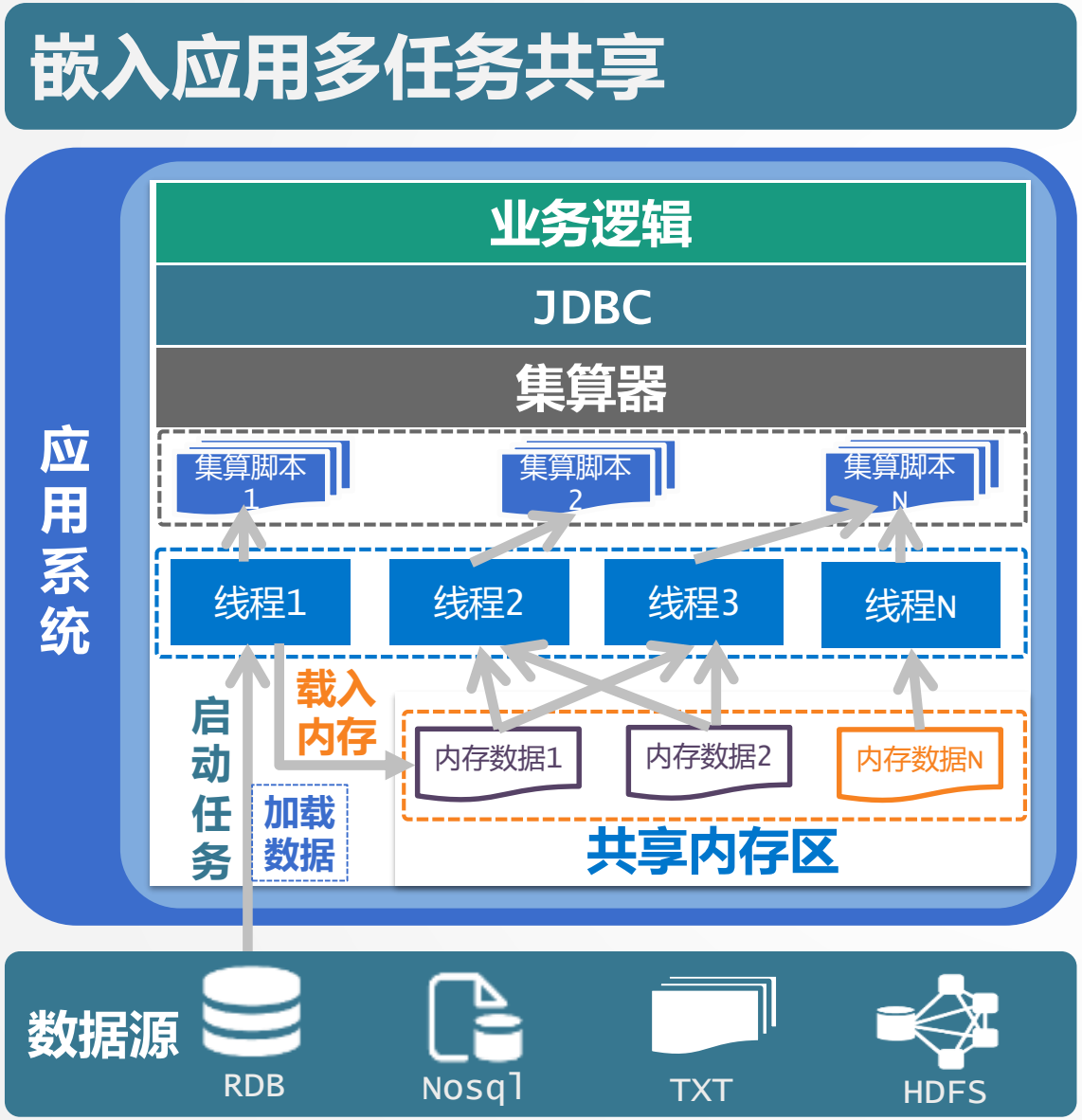


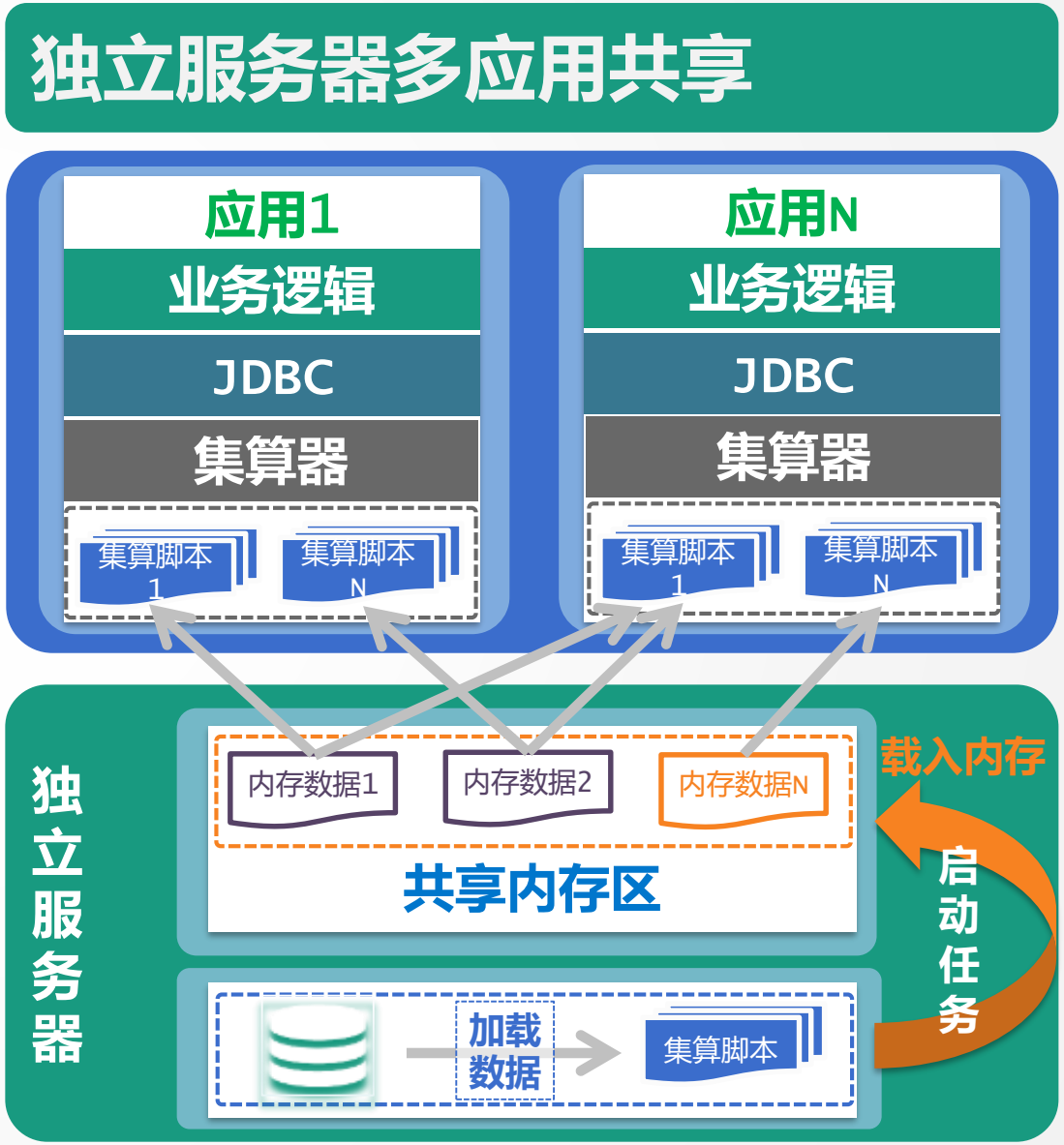
网格结果所见即所得，易于调试；方便引用中间结果！

敏捷语法，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

应用集成与管理体系 – 集算器（SPL）





集算器(SPL)作为内存计算引擎 – 总结

集算器（SPL）

高性能

离散数据集模型支持更高计算性能

轻量级

依赖较少JAR包就可以独立工作

易开发

开箱即用无须环境配置与代码引用

易集成

对外提供标准JAR包易于嵌入集成

创新技术

推动应用进步！