途牛系统架构演化实践

赵国光 - 途牛首席架构师 技术委员会负责人

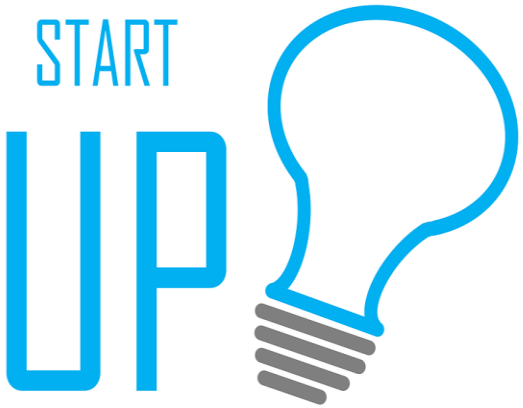
内容

1.途牛业务与系统整体介绍

2.系统演化过程中的经验总结

3.架构实例

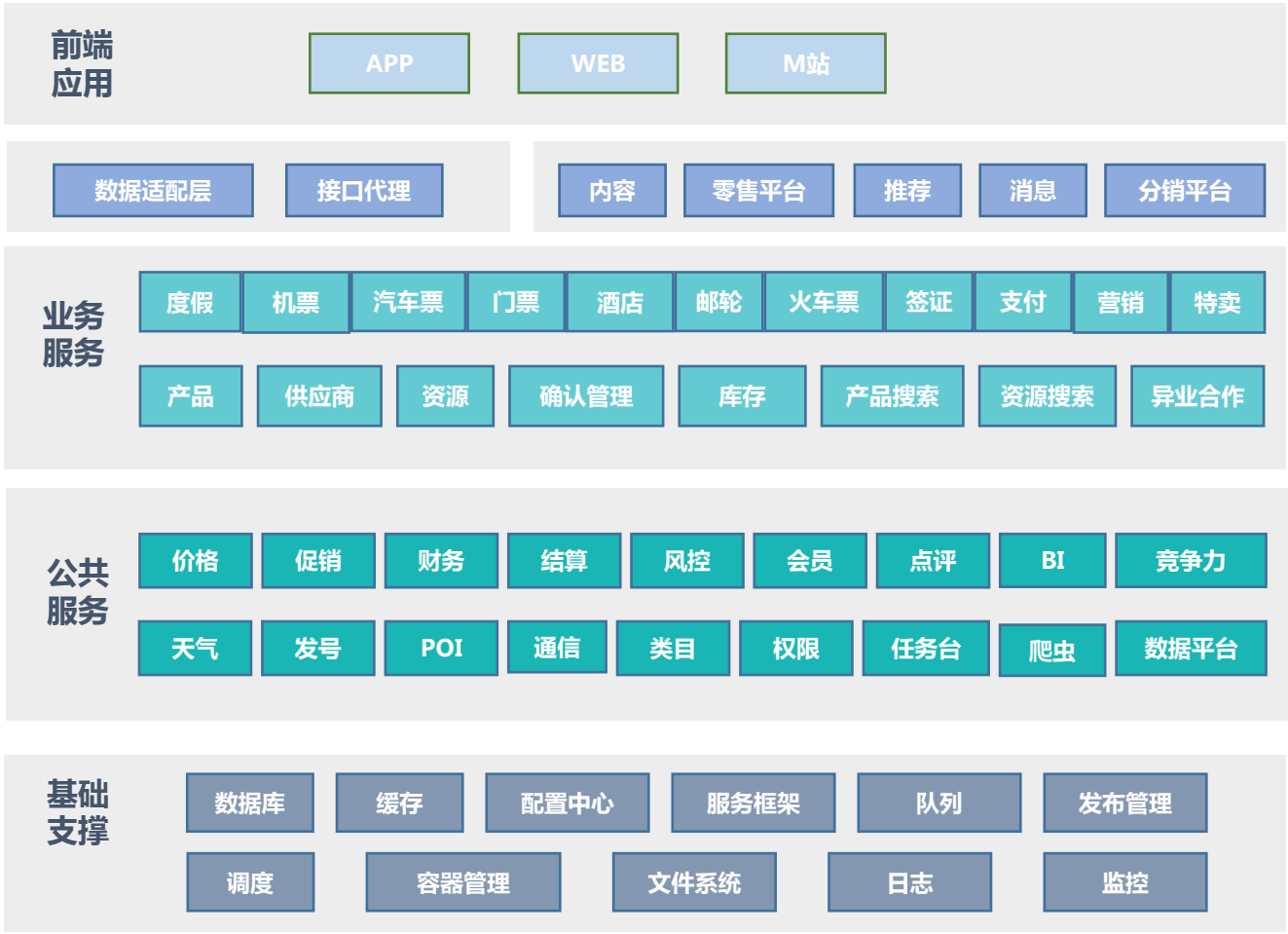
4.架构师的角色



途牛业务与系统整体介绍

- 产品构成复杂
- 价格变化频率高，价格策略复杂
- 对接多供应商，售卖场景复杂
- 品类多，业务形式多样
- 售卖到使用时间段长，期间变化多
- 同质化

途牛业务与系统整体介绍



系统演化过程中的经验总结

- 垂直架构
- 微服务化过程
- 业务系统的演化
- 技术架构

垂直架构

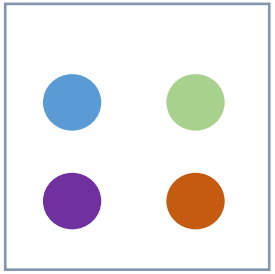
- 重复建设
- 系统间数据难以打通、资源共享
- 缺少业务沉淀

微服务化过程

- 服务化抽象
- 服务管理的负担
- 职能细分
- 数据完整性和一致性

服务化技术框架 ≠ 服务化

解耦， 复用



微服务化过程

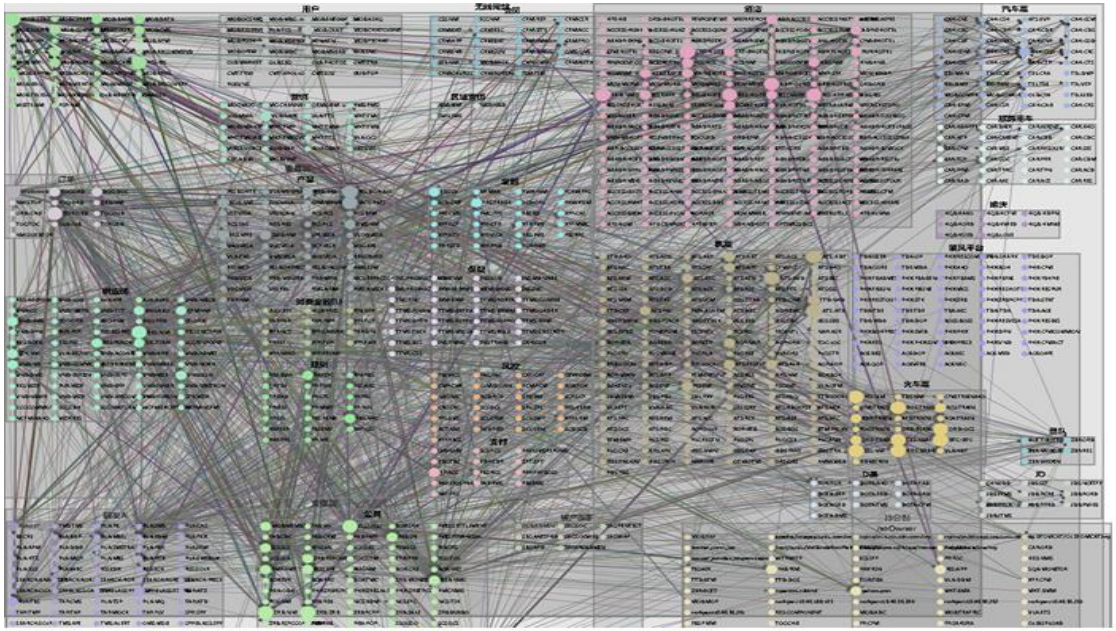
- 服务化抽象
- 服务管理的负担
- 职能细分
- 数据完整性和一致性

“服务化不是无成本的”

微服务化过程

- 服务管理
- 问题排查
- 沟通协同

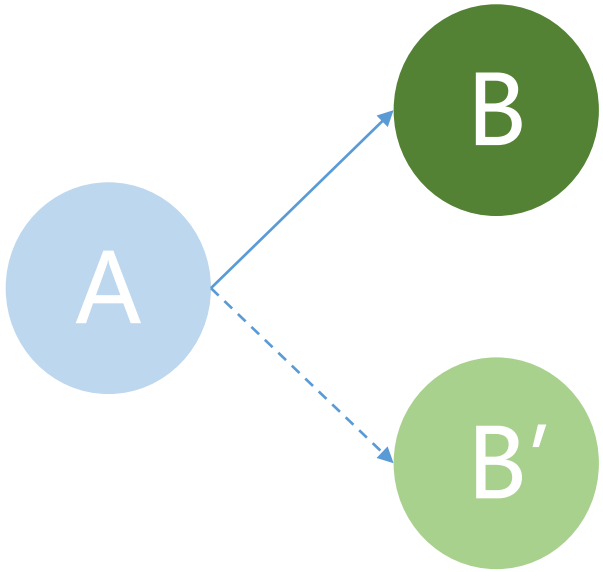
“服务化不是无成本的”



微服务化过程

- 服务化抽象
- 服务管理的负担
- 职能细分
- 数据完整性和一致性

重复

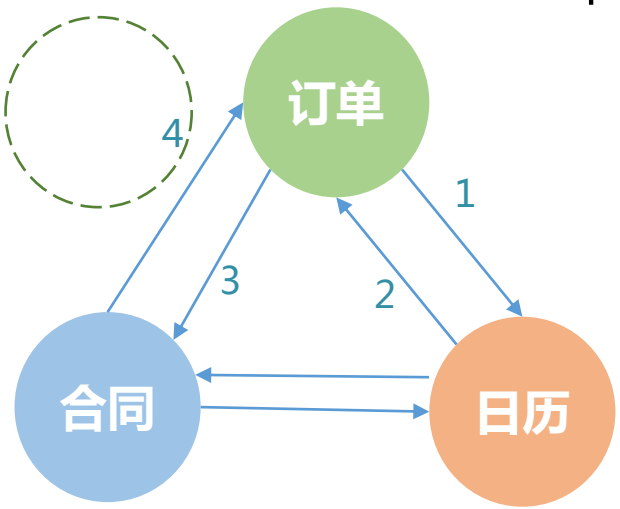


隔离 *vs* 重复

微服务化过程

- 服务化抽象
- 服务管理的负担
- 职能细分
- 数据完整性和一致性

职责分散



节假日核损：

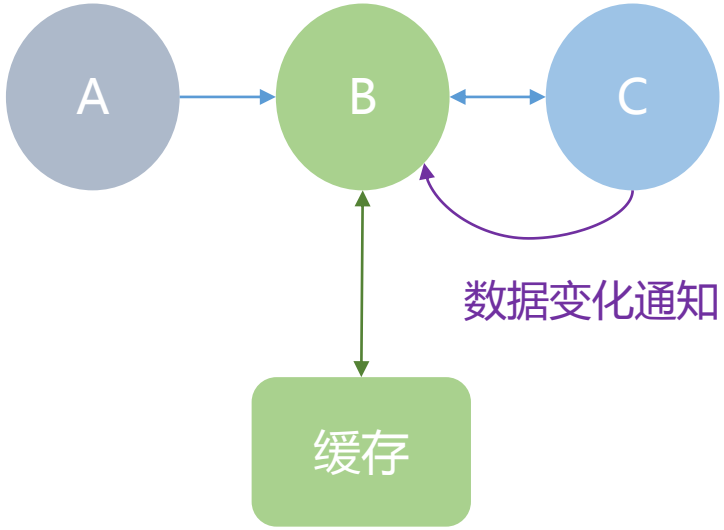
- *节假日？*
- *出哪份合同？*

合同只是**数据**，“按规则出对应的合同”才是**职责**

微服务化过程

- 服务化抽象
- 服务管理的负担
- 职能细分
- 数据完整性和一致性

依赖关系复杂



微服务化原则

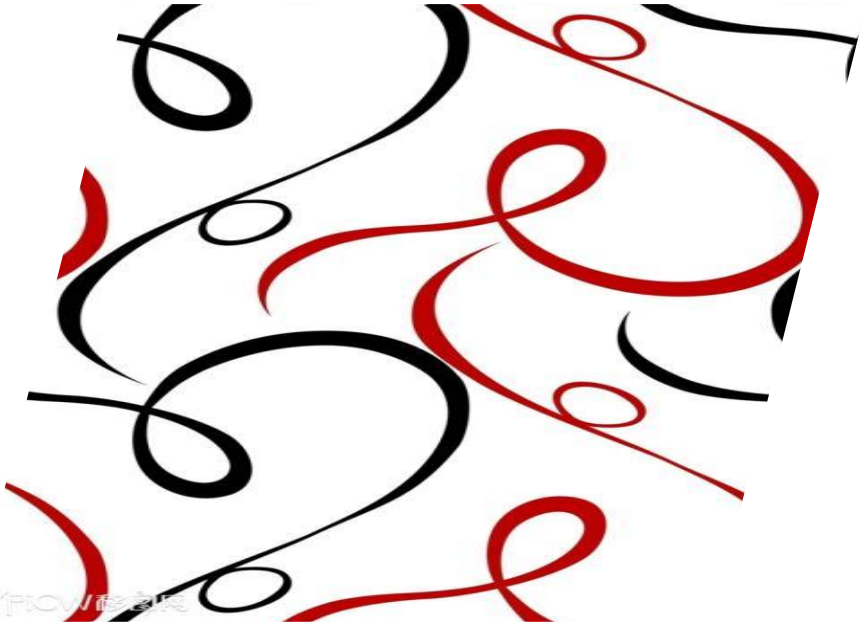
*In short, the **microservice** architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies.*

-- ***Martin Fowler***

- 面向业务，围绕领域模型
- 隐藏实现细节
- 聚焦用户与API
- 去中心化
- 独立、自动化部署

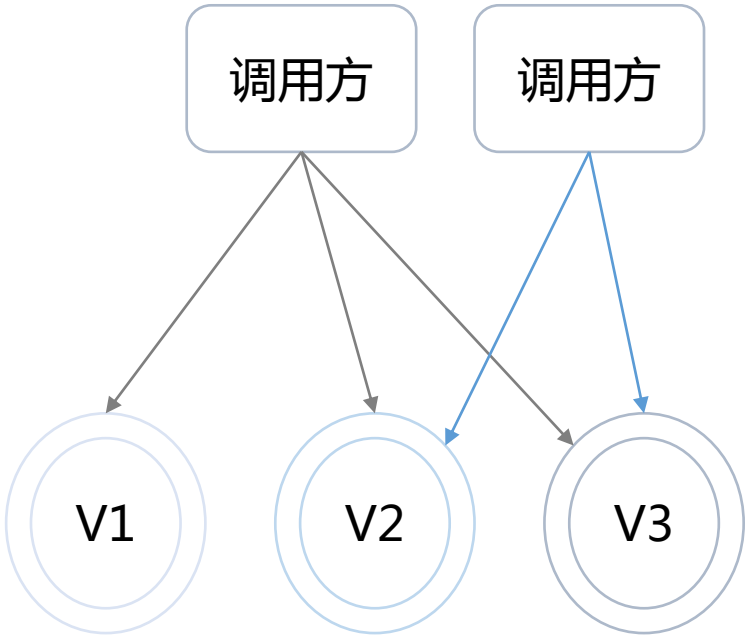
业务系统的演化

- 问题域&解决方案域
- 过渡系统
- 流程标准化



业务系统的演化

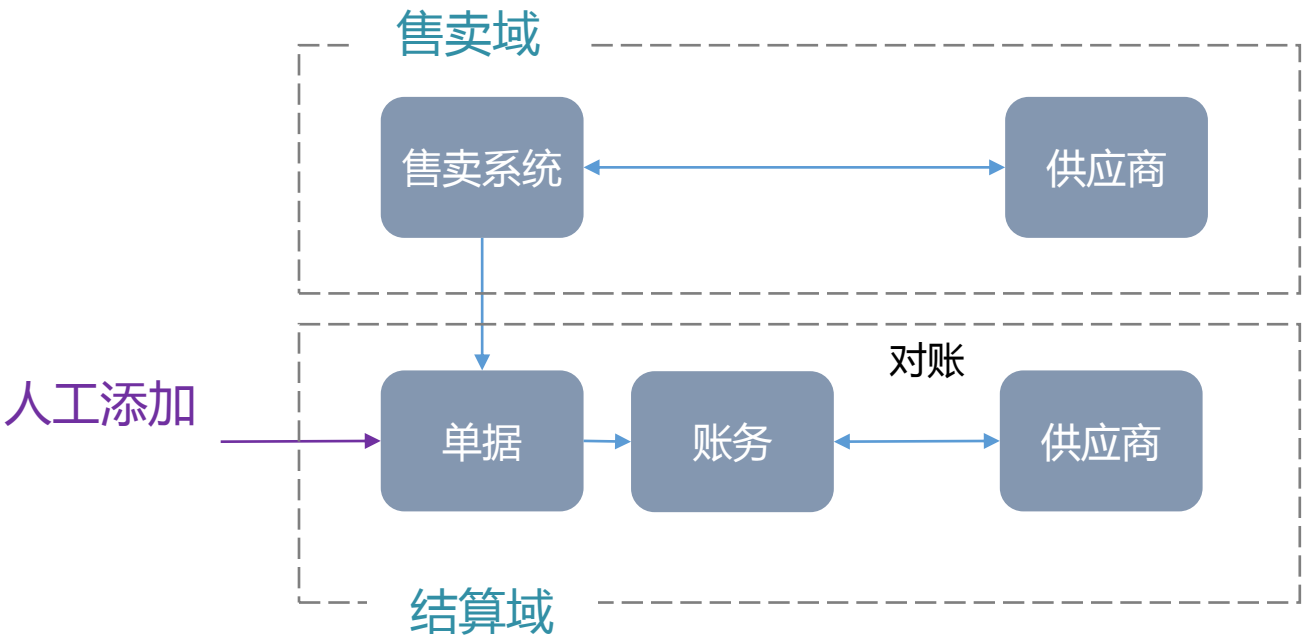
- 问题域&解决方案域
- 过渡系统
- 流程标准化



业务系统的演化

- 问题域&解决方案域
- 过渡系统
- 流程标准化

一时取巧会得到两样东西：
短期的便捷与长期的拖累



技术架构

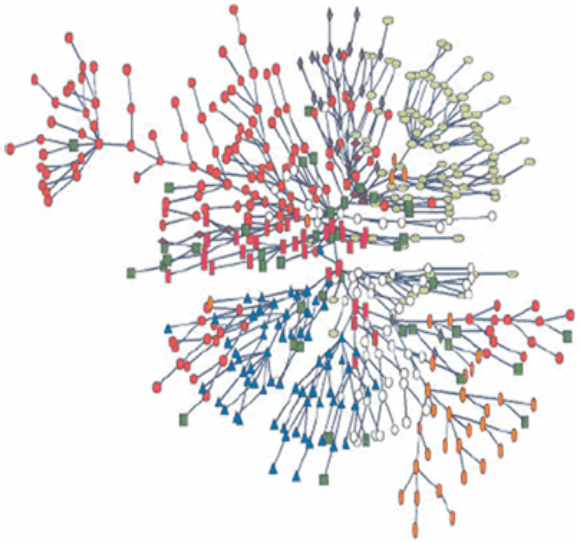
- 稍领先于业务架构（*服务治理、监控、技术支撑*）
- 避免重复建设小轮子

架构实例

不同场景下企业应用面对的复杂性是不同的：

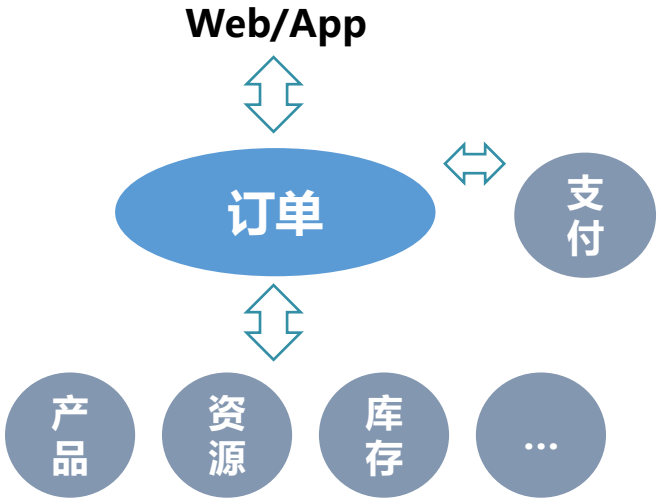
- 大用户**量**，大流量，高并发
- 复杂的业务**逻辑**
- 快速增加**变**化功能

trade-off



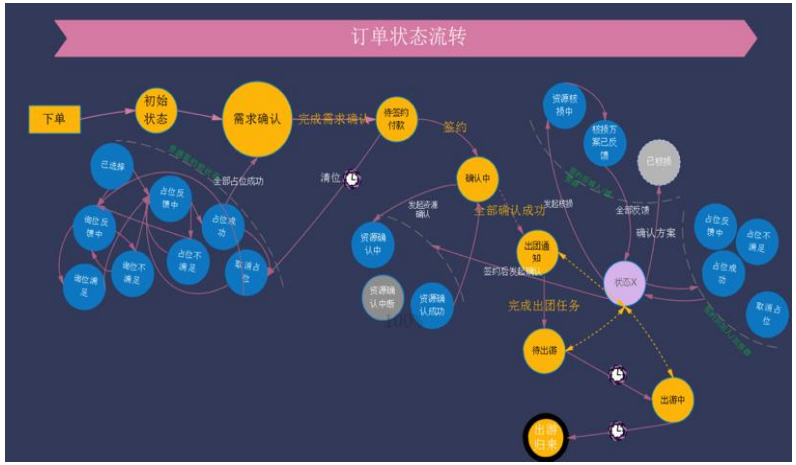
应用架构实例：订单平台项目

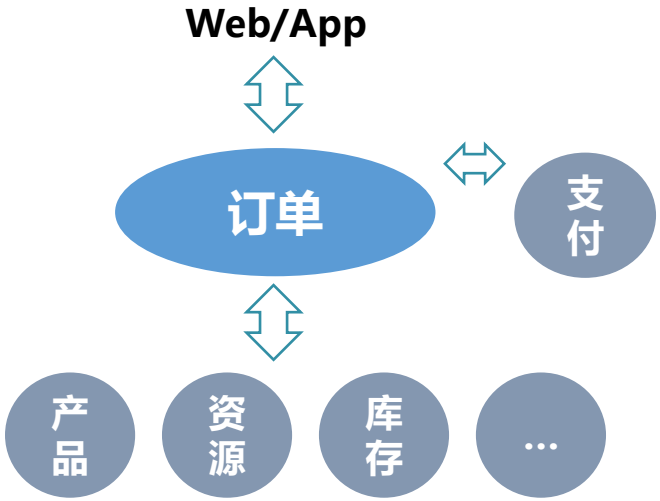
- 不同品类订单：
  - 人机交互逻辑不同
  - 状态流转*不完全*相同
- 资源类型多，每种资源的处理方式不同



应用架构实例：订单平台项目

- 不同品类订单：
  - 人机交互逻辑不同
  - 状态流转*不完全*相同
- 资源类型多，每种资源的处理方式不同

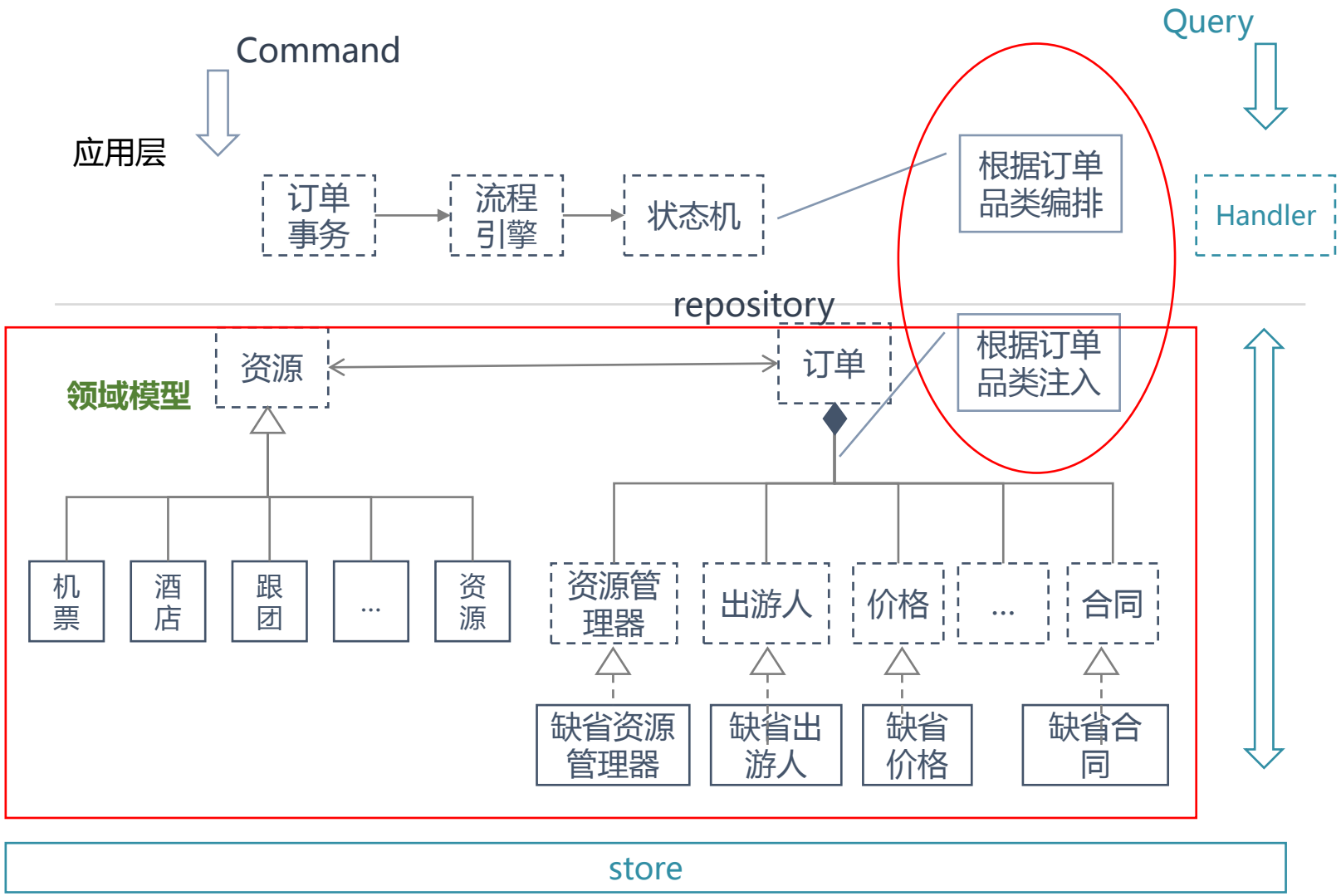




应用架构实例：订单平台项目

- 不同品类订单：
  - 人机交互逻辑不同
  - 状态流转*不完全*相同
- 资源类型多，每种资源的处理方式不同

应用架构实例：订单平台项目　软件架构



应用架构实例：订单平台项目 案例总结

- 领域模型（***DDD***）
- 业务隔离

应用架构实例：价格管理中心

5 ➡ 250000 ➡ 6,250,000 ➡ 187,500,000

三亚五日游

Day1	Day2	Day3	Day4	Day5
飞机酒店	沙滩公园酒店	岛酒店	岛景点酒店	飞机

机票：5家航空公司，每家3个航班

酒店：10家酒店，每家10个房型

景点：15家供应商提供门票

Day1去Day5返回的**所有资源组合**中价格最低

出发去三亚游玩的城市可以有300个

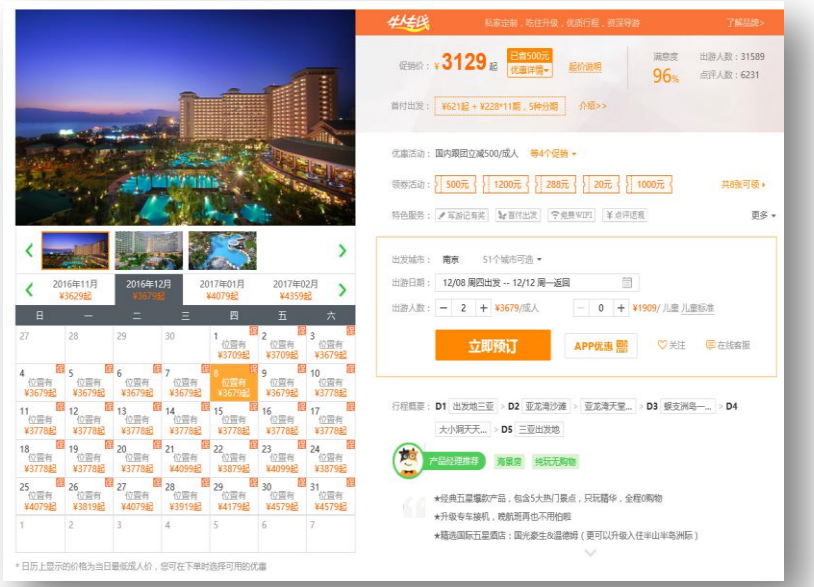
每家酒店计算Day1到Day4**连续**4天有空房间并且总价最低

每一天的资源价格都不一样，最长预售期180天！

每个预定城市的促销规则可以不一样

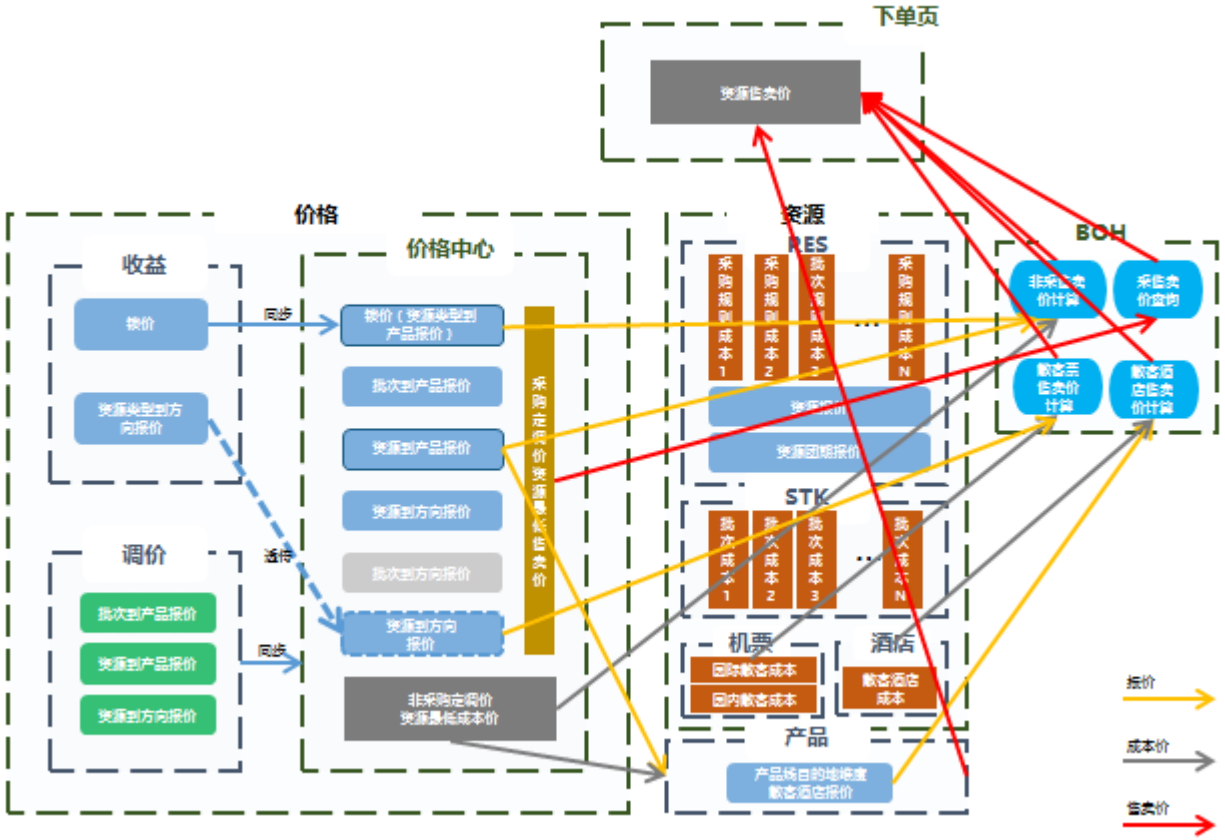
同一个资源**不同批次**的成本价/利润规则都可以不一样

每种批次在每个产品上的售卖价格都可以不一样



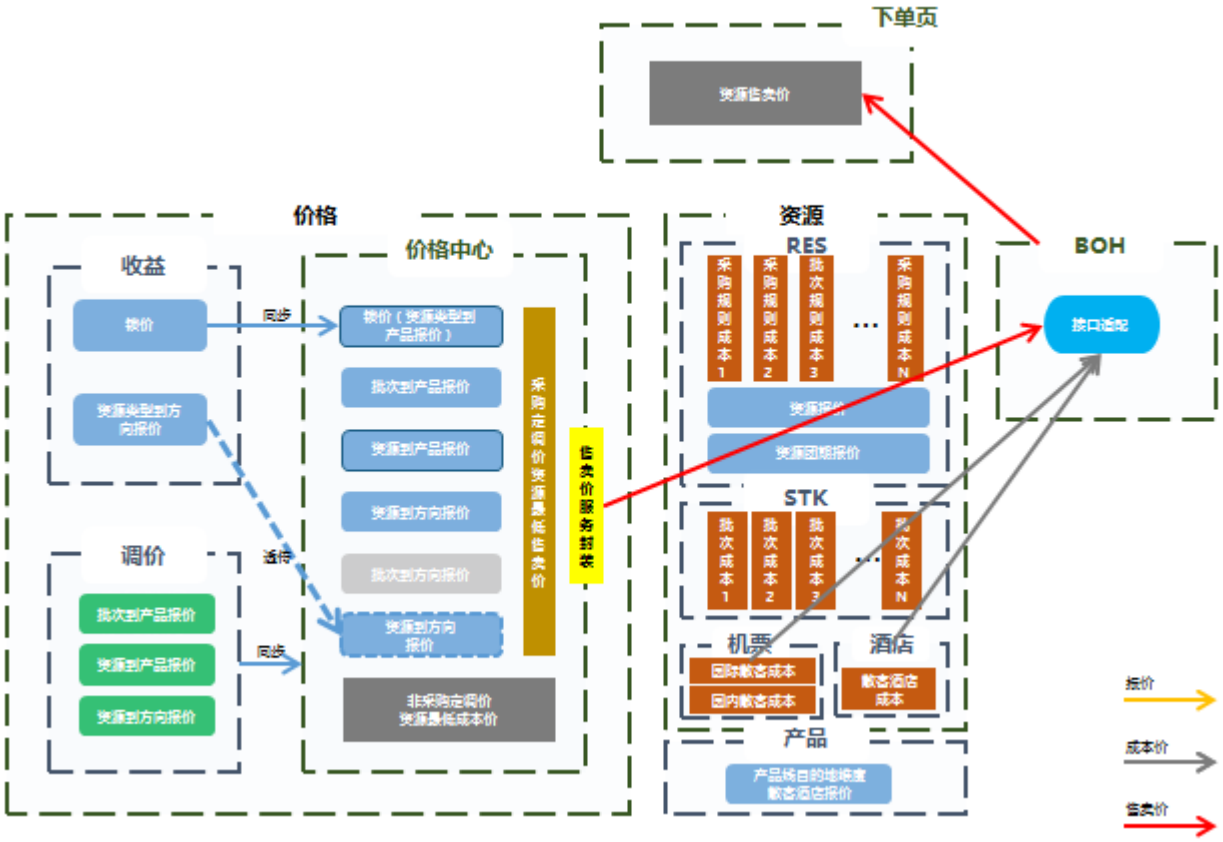
应用架构实例：价格管理中心

随着功能不断增加，出现系统职责越来越分散

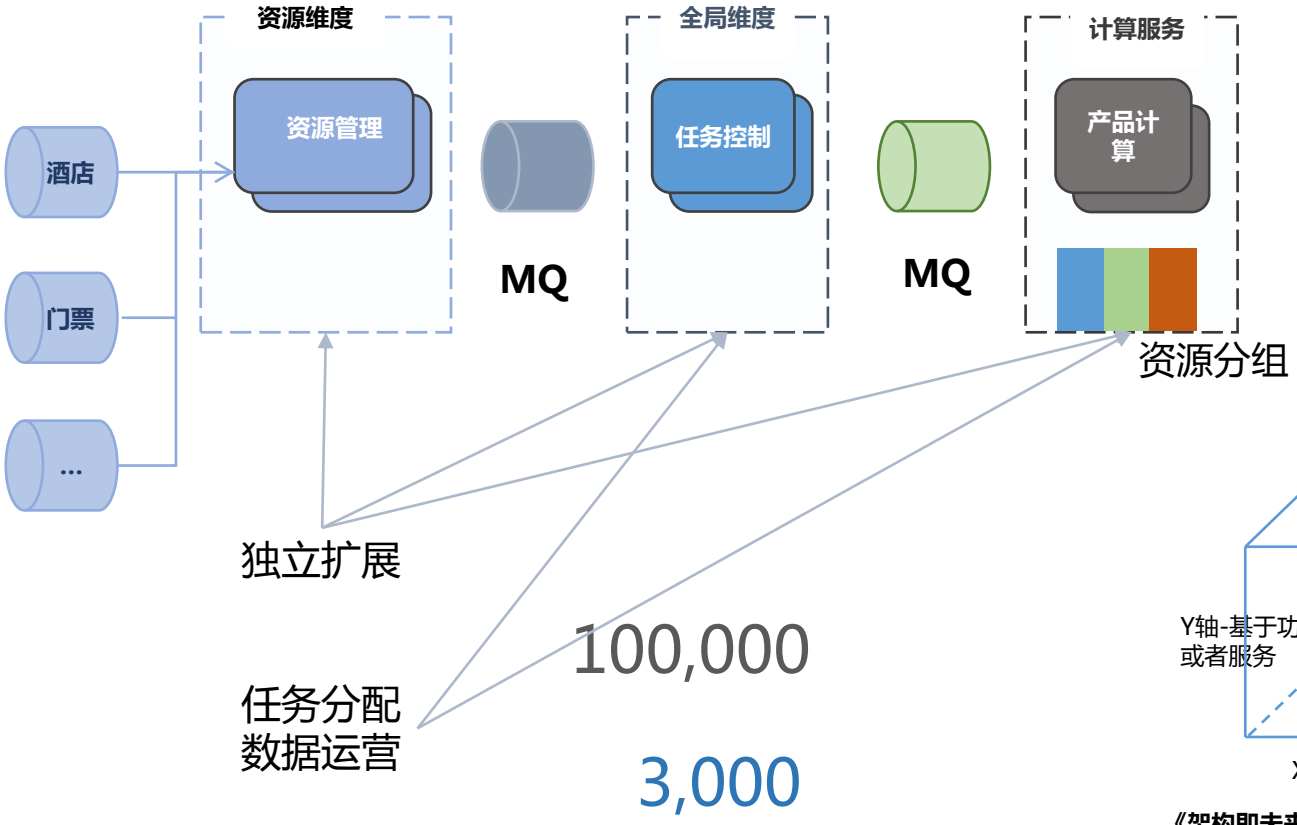


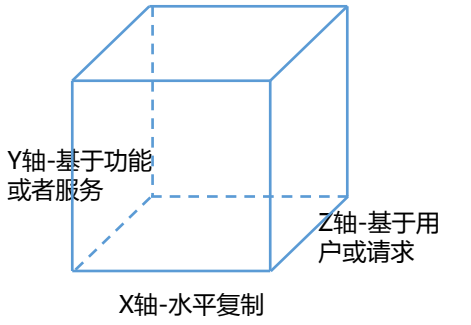
应用架构实例：价格管理中心

服务封装



应用架构实例：价格管理中心





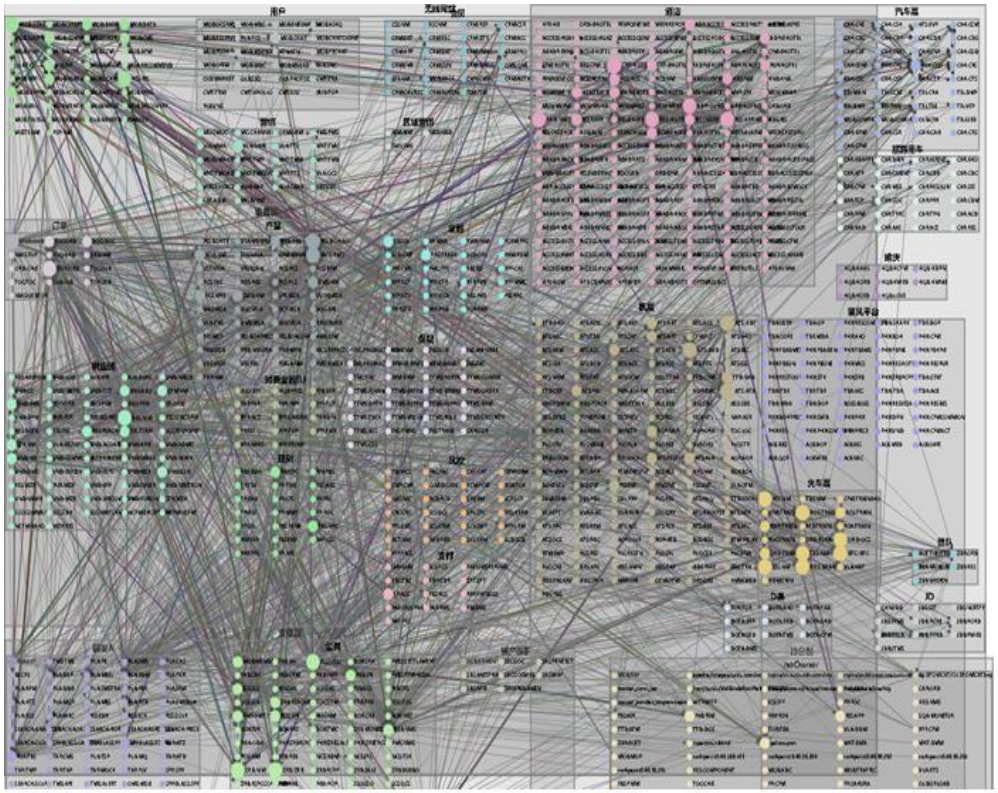
《架构即未来》THE ART OF SCALABILITY

应用架构实例：价格管理中心　案例总结

- 服务化封装，降低周边系统的负担和复杂度
- 能力扩展（*扩展立方体、异步、无状态*）
- 资源分配优化，业务闭环，数据自运营

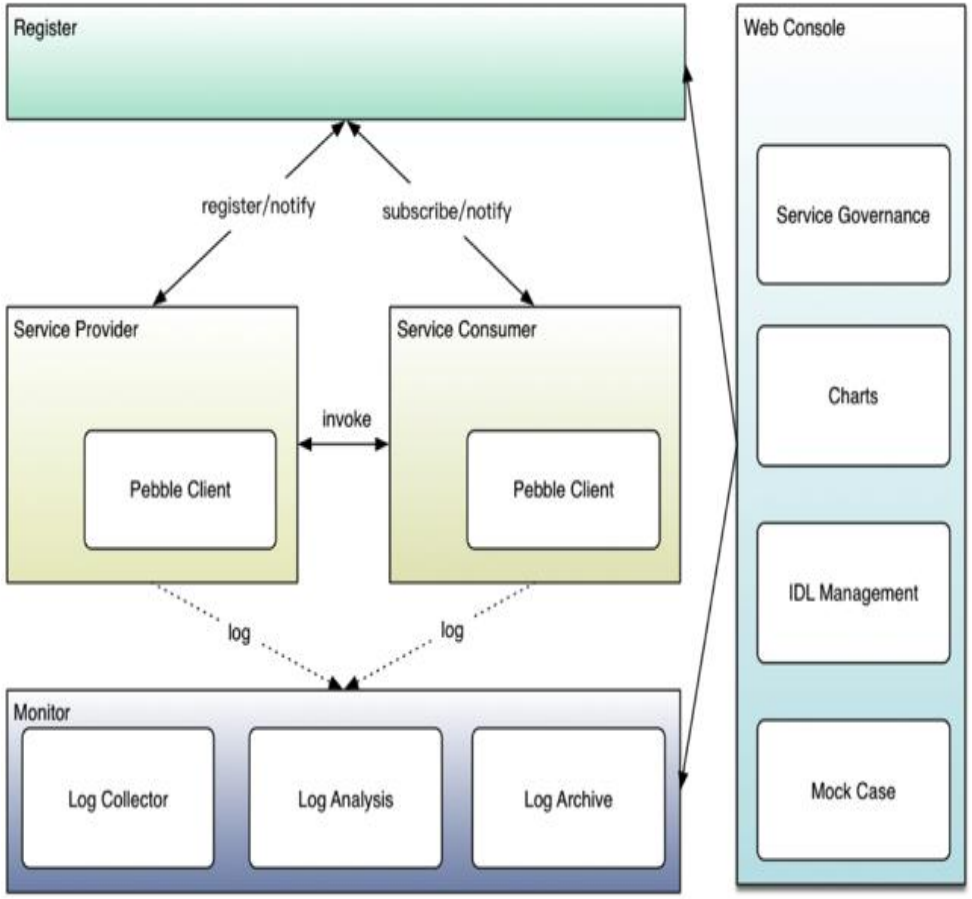
技术架构实例：分布式服务框架 *Pebble*

- 服务治理
  - *注册与订阅*
  - *路由规则*
  - *限流*
  - *熔断*
- 服务接口协议
- 服务监控与度量
- 测试

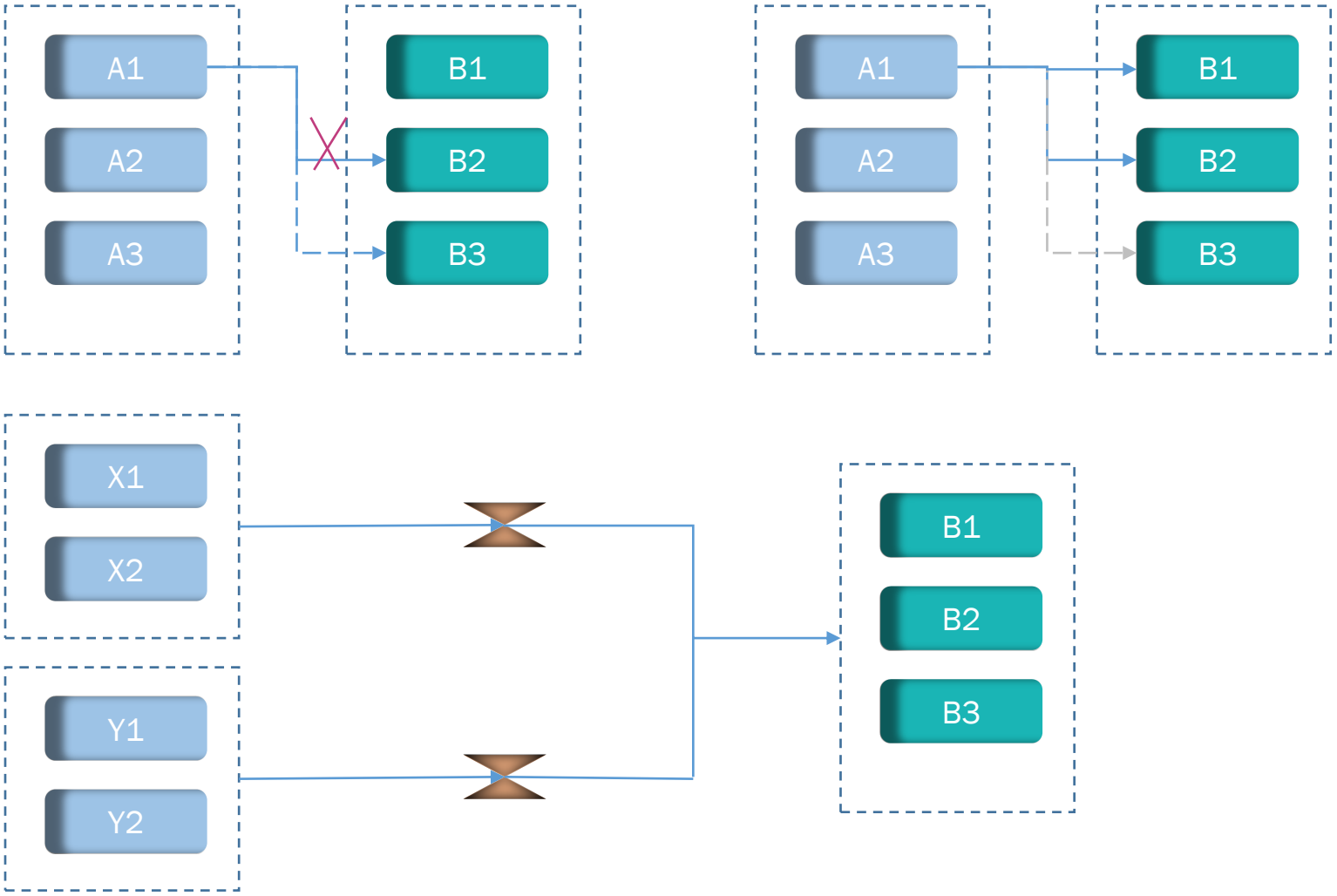


技术架构实例：分布式服务框架 *Pebble*

- 服务治理
  - *注册与订阅*
  - *路由规则*
  - *限流*
  - *熔断*
- 服务接口协议
- 服务监控与度量
- 测试

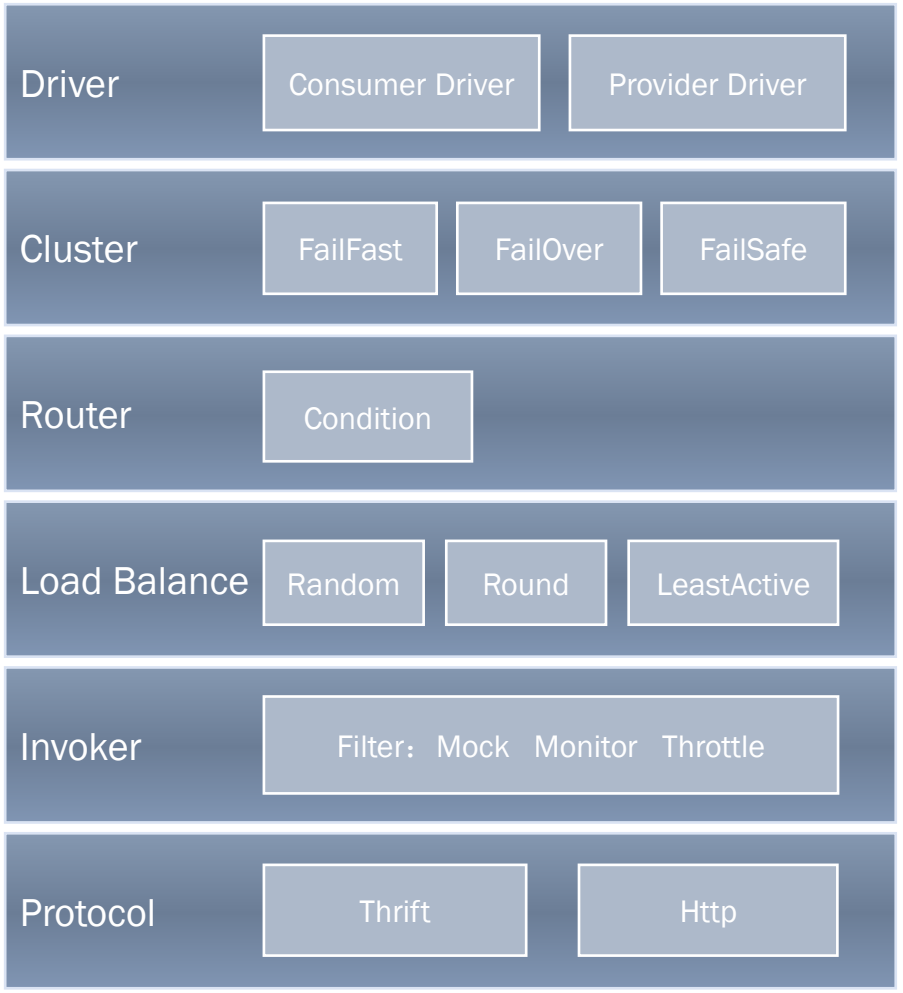


技术架构实例：分布式服务框架 *Pebble*



技术架构实例：分布式服务框架 *Pebble*

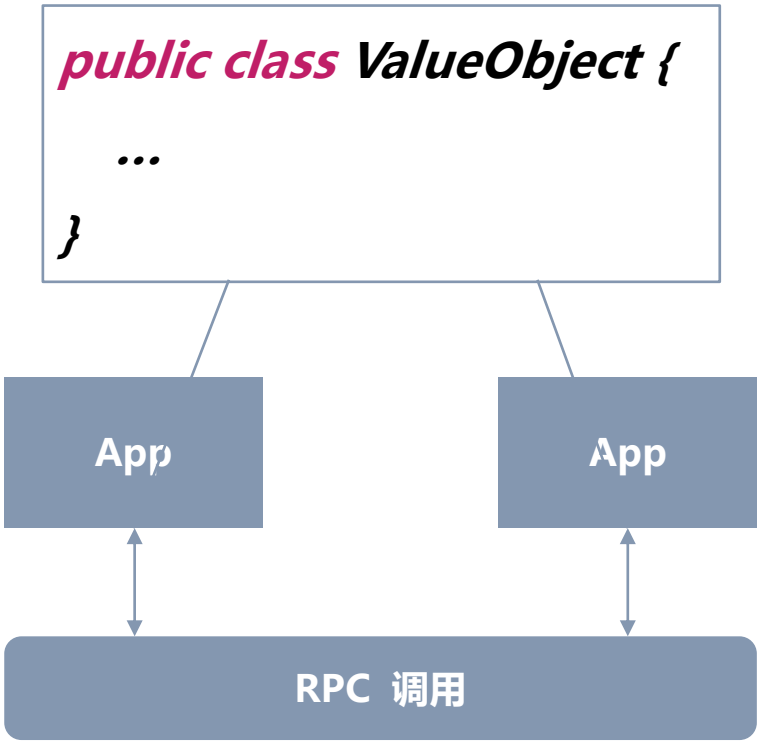
软件架构与扩展性



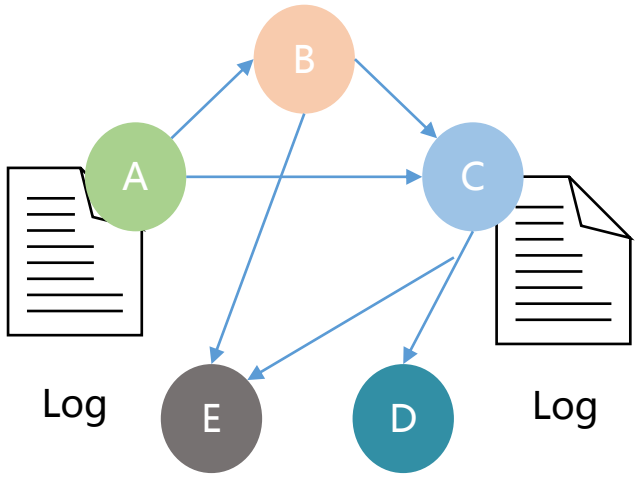
技术架构实例：分布式服务框架 *Pebble*

Interface Definition Language

统一接口定义

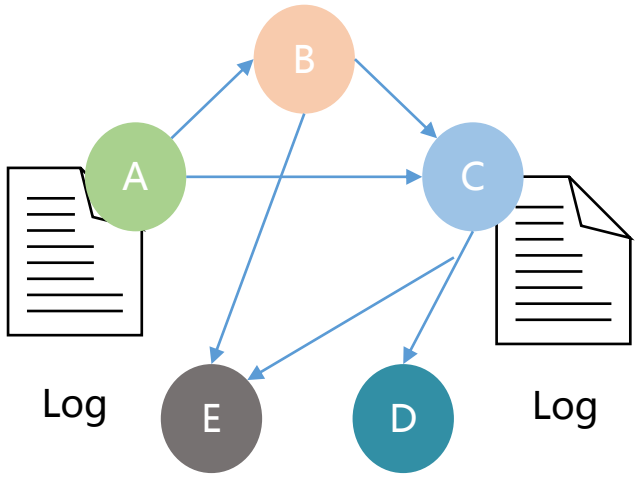


技术架构实例：分布式跟踪系统 *X-Ray*



- 到底失败在哪
- 各系统的日志孤立，无法查找
- 一次业务链经过了哪些系统
- 哪个系统慢
- 容量规划

技术架构实例：分布式跟踪系统 *X-Ray*



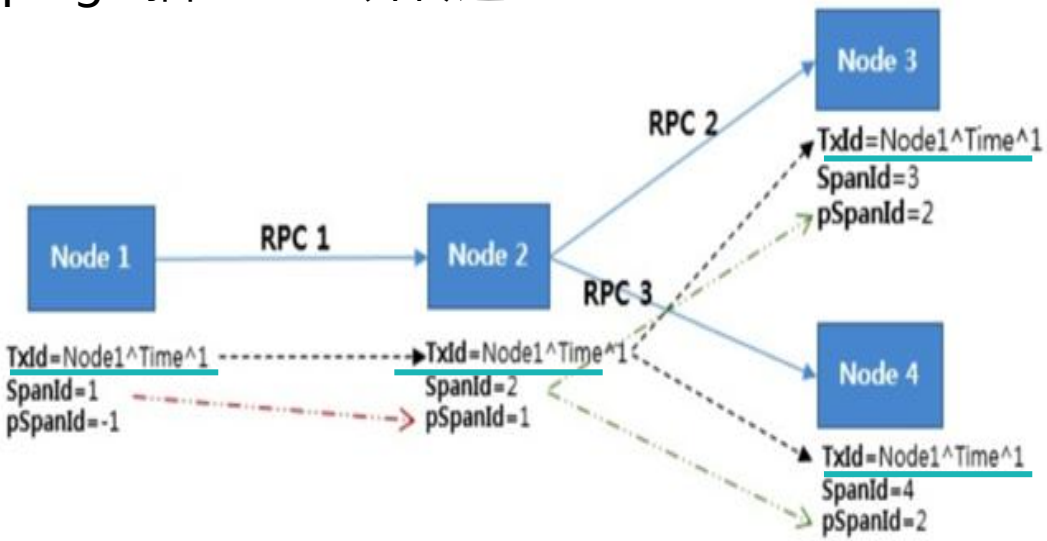
目标：

- 串接业务，精准告警
- 业务链跟踪与分析
- 系统运行情况分析
- 非入侵性

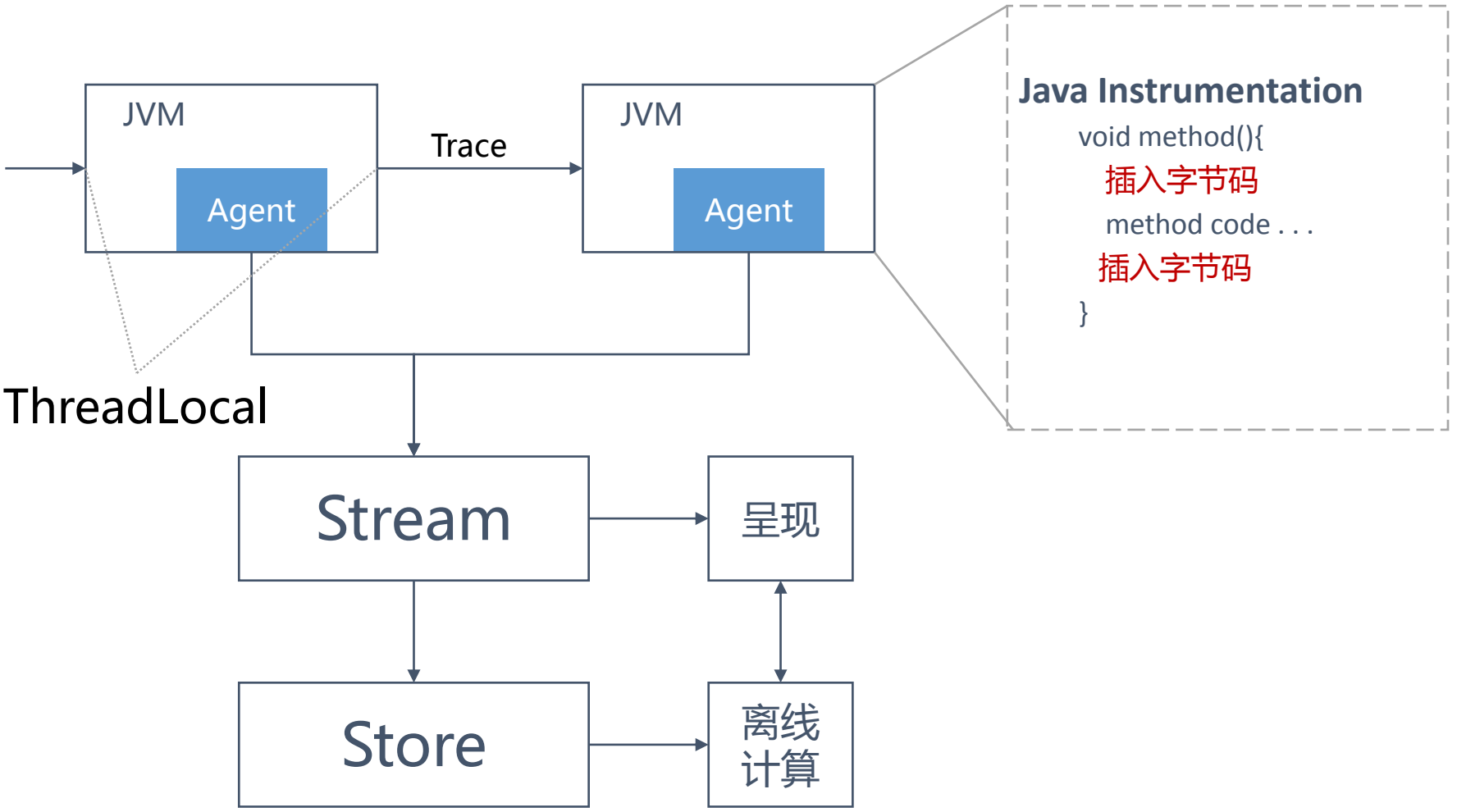
X-Ray 实现原理

Pinpoint，基于Google Dapper论文

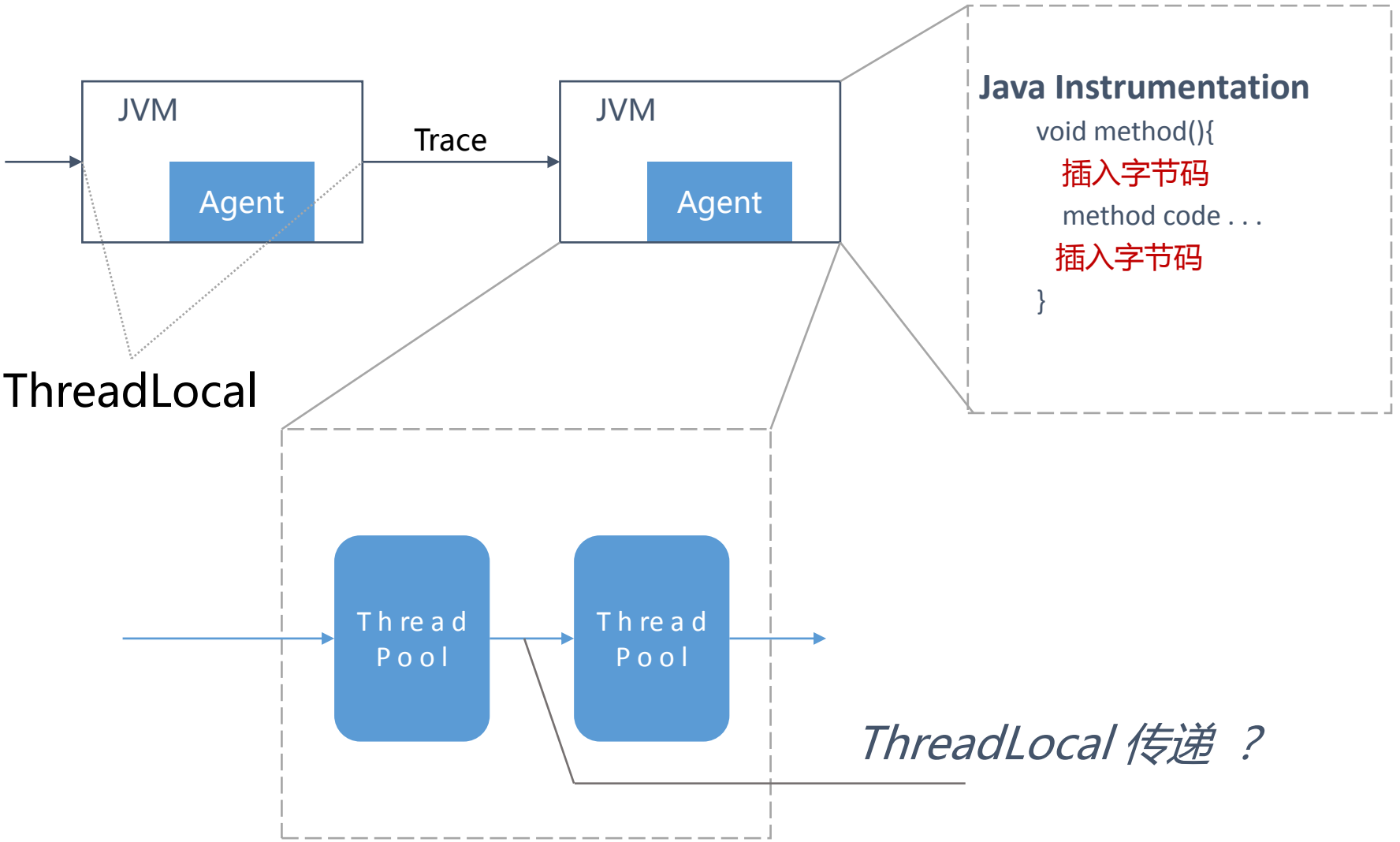
- Plugin生成Trace
- 在应用内通过ThreadLocal保存
- 在网络协议上再通过plugin插入Trace并传递



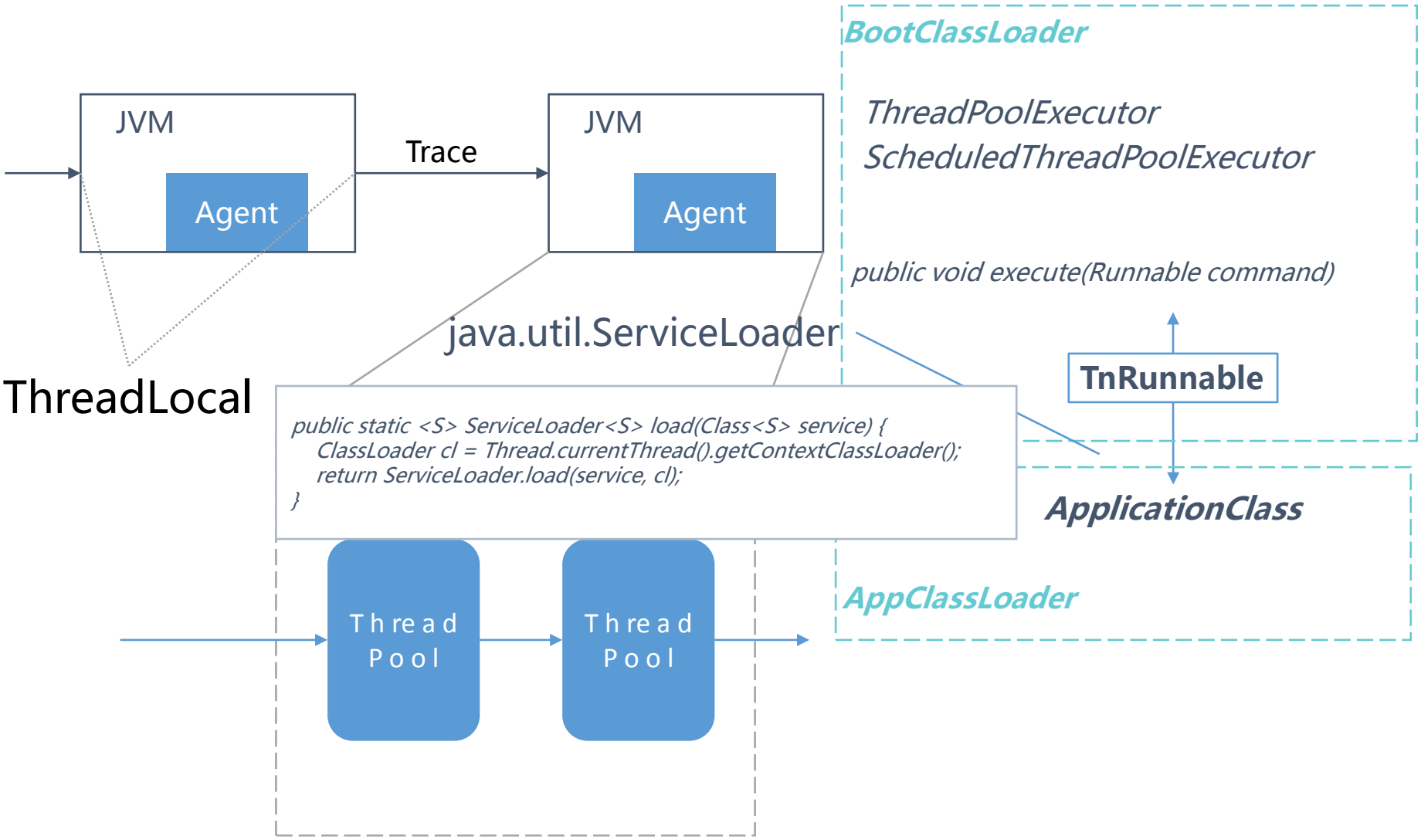
X-Ray 实现原理


JVM
Agent
Trace
JVM
Agent
Java Instrumentation
void method(){
插入字节码
method code . . .
插入字节码
}
ThreadLocal
Stream
呈现
Store
离线
计算

X-Ray 实现原理


JVM
Agent
Trace
JVM
Agent
Java Instrumentation
void method(){
插入字节码
method code . . .
插入字节码
}
ThreadLocal
Thread Pool
Thread Pool
ThreadLocal 传递 ？

X-Ray 实现原理

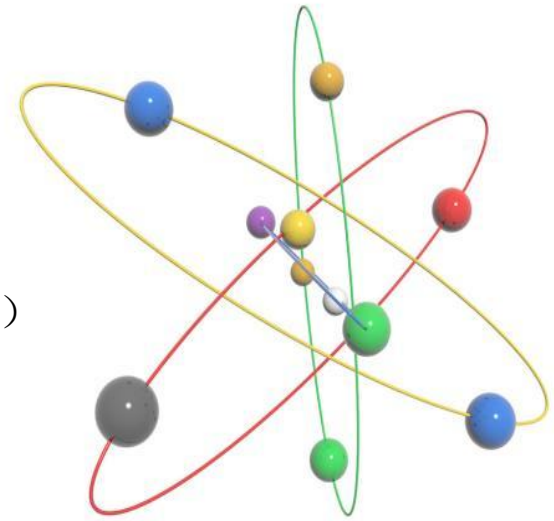

BootClassLoader
ThreadPoolExecutor
ScheduledThreadPoolExecutor
public void execute(Runnable command)
JVM
Agent
Trace
JVM
Agent
java.util.ServiceLoader
ThreadLocal
TnRunnable
public static <S> ServiceLoader<S> load(Class<S> service) {
ClassLoader cl = Thread.currentThread().getContextClassLoader();
return ServiceLoader.load(service, cl);
}
ApplicationClass
AppClassLoader
Thread Pool
Thread Pool

架构设计目标

- 运行表现：高可用、高性能、高并发、响应时间、安全
- 可伸缩性：XYZ扩展
- 可维护性：可复用、低耦合、低沟通成本
- 易于应对业务变化

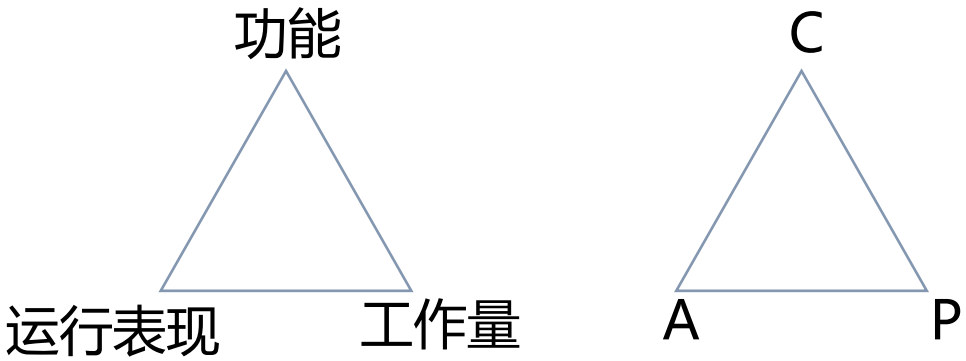
总的架构原则

- 备份原则（N+1， 非单点）
- 能力可扩展（Design-Implement-Deploy)
- 异步设计
- 无状态设计
- 单一职责
- 最小知识原则
- 最小依赖（最低强依赖）
- 闭环与完整性（系统内业务闭环、数据完整、职责完整）
- 隔离原则
  - 核心业务与非核心隔离
  - 易变与稳定隔离
  - 业务间隔离
- 服务可治理（限流、降级、超时、熔断）
- 回滚与向下兼容
- 可监控



架构师的角色

- 理解业务
- 定义问题，并解决
- 关注人的因素，沟通
- 权衡



总结

1.途牛业务与系统整体介绍

2.系统演化过程中的经验总结

3.架构实例

4.架构师的角色

Thanks

赵国光