促进软件开发领域知识与创新的传播

InfoQueue

QCon
全球软件开发大会

[上海] 2015年10月15-17日

ArchSummit
全球架构师峰会

[北京] 2015年12月18日-19日

关注InfoQ官方微信
及时获取ArchSummit演讲视频信息

泛前端、交易云与金融电商

- 传统券商的互联网技术之路

梁启鸿
董事总经理 | 首席架构师
广发证券 IT

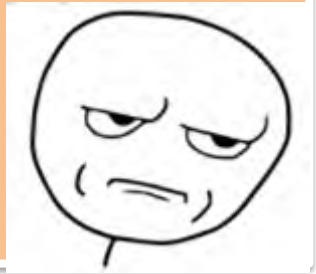
本土证券行业软件技术的悖论

IT技术落后

- 券商几乎没有严格意义上的软件R&D（研发）
- 行业开发工程师人员占比极低
- 一切围绕关系型数据库（RDBMS）、或者索性说Oracle为核心的技术系统
- 基于封闭技术的第三方中间件（技术古老、理念落后）
- Oracle+J2EE是部分工程师对技术世界的全部想象
- 语言：C/C++、Java、Stored Procedure、shell script
- “互联网业务就是建网站”
- “什么叫用户体验？”

本质追求技术创新

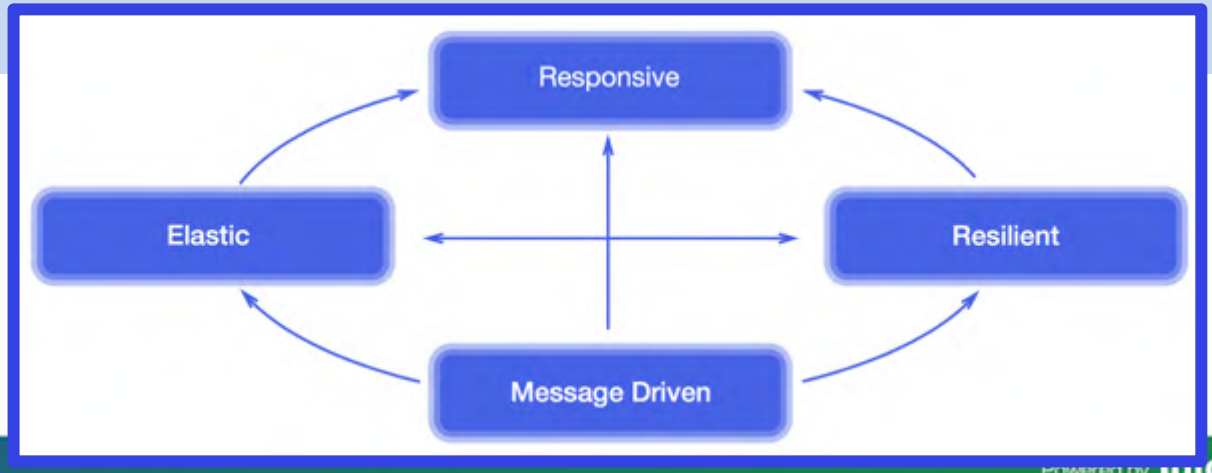
- 华尔街是最追逐前沿技术的地方之一
- 极速交易、量化交易、高频交易、实时风控对技术性能的极致追求
- 软件系统可靠性、高可用性、快速扩容能力要求超越绝大部分其他行业
- 业务创新100%依赖IT – 新一代金融机构就是IT机构、大数据公司、云服务提供者
- 可是…



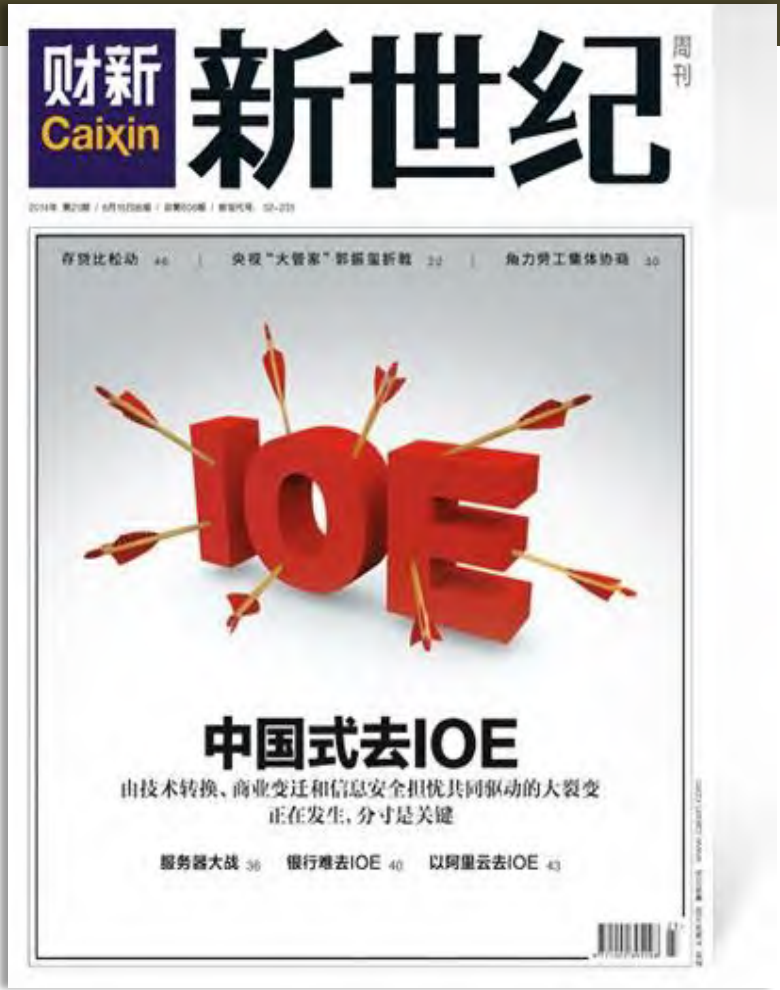
我们 - Next-gen FinTech R&D

We Are Reactive

- 2013年从金融电商平台研发开始
- 致力于用最前沿的互联网开源技术，打造创新型 FinTech “Startup”
- 互联网技术基因、金融软件深度技术研究
- 与IT运维联合支持峰值2500亿日交易
- “Macbook + 函数/脚本语言 + Git” – 工具虽小，体现文化
- “Bleeding Edge”（对于传统金融业而言）：2013年开始大规模采用 AngularJS、Docker、Node.js...
- 各种 OO Design Patterns、Architecture Patterns、Cloud Patterns的践行者



“去IOE” – 广发新金融技术Stack



- “互联网规模”（Internet-scale）要求成本可控的大规模分布式系统部署
- “互联网金融”：既然在互联网上，就要采用互联网基因的技术
- 财新网2014年6月16日周刊捅破了“去IOE”窗户纸
- “由技术转换、商业变迁和信息安全担忧共同驱动的大裂变正在发生”

mongoDB · docker · redis · nodeJS · Go Programming Language · hadoop · APACHE HBASE · Spark · elasticsearch. · ANGULARJS by Google · kafka · kubernetes · Flume hadoop

我们干了什么

广发证券互联网创新2年回顾

平台数据

Platform DATA

6,000,000

手机证券总用户超600万
5月份
单月新增用户超70万

600万 总用户数

73.6万 5月份单月增长用户数

$100亿—500亿

易淘金电商平台，2014年产品年销量过100亿，15年目标500亿

17.6亿 7月8日单日销量

46.5亿 2015年4月份销量

$ 7,000万

创造的产品二级转让市场“淘金市场”
单日转让记录突破7000万

7283万 4.17单日完成转让

8.2亿 5月份完成转让

金融电商类 – 易淘金平台、淘金市场、理财账户

上线10个月达到100亿

2015年预期超过500亿

4月份销量近50亿、单日销售量超过15亿

单日转让记录7200万

社会化平台类 – 证券界“网店”、“嘀嘀打车”、“社交晒股” apps

（屏幕图片均为真实App截图）

泛交易终端类 – 手机交易、页面交易、客户端交易

证券界的

“手游”

“页游”

“端

（屏幕图片均为真实App截图）

经营驾驶舱APP

电商运营数据随时随地一手掌控

开户监控分析
Account Dashboard

领导驾驶舱
Management Dashboard

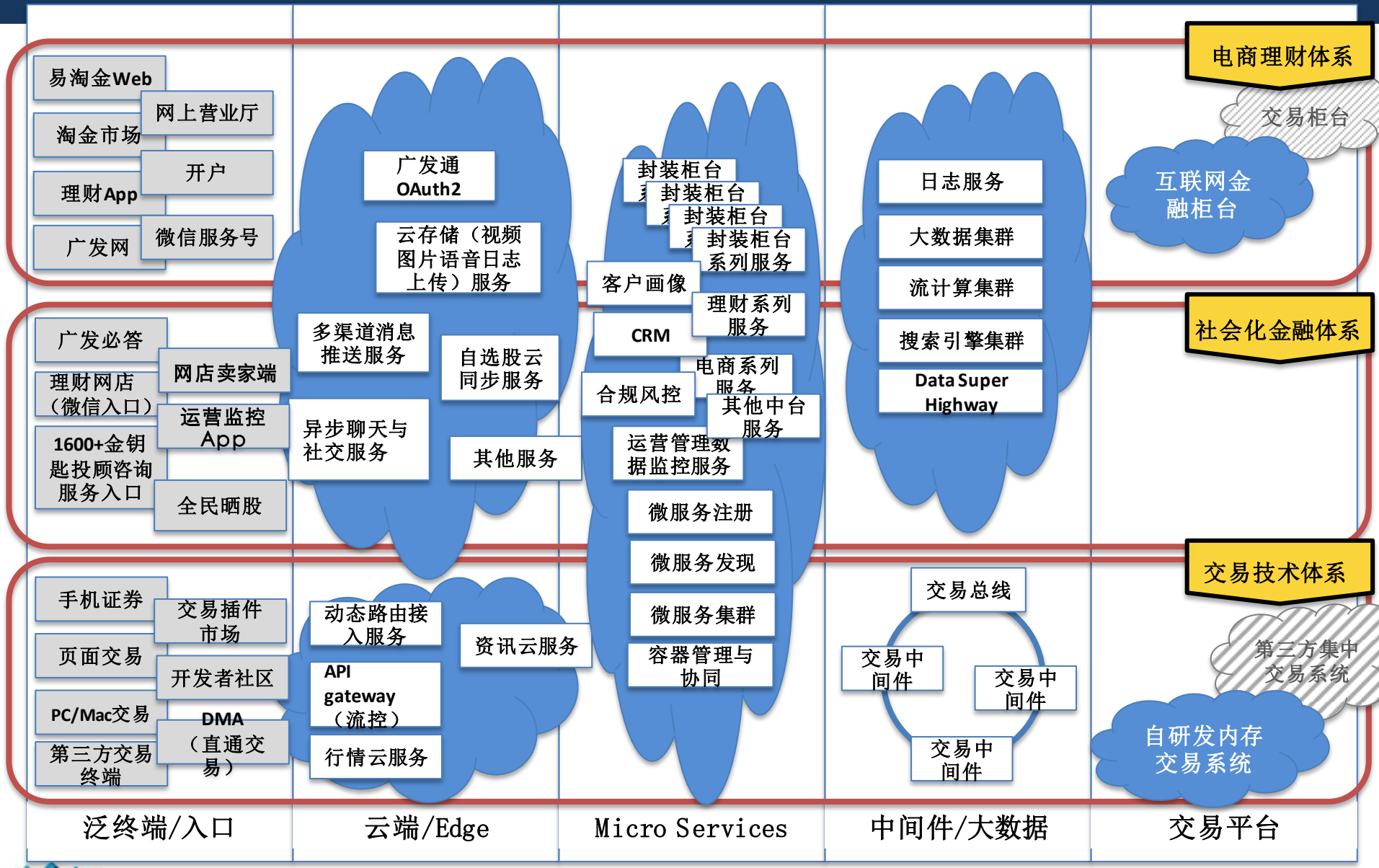
（屏幕图片均为真实App截图）

云计算类 – Open Trading 交易云

- ❖ FIX国际标准协议
- ❖ C、Python、Go、Java语言SDK
- ❖ 极速接入

- ✓ 券商唯一交易云平台
- ✓ 打造金融业的开发者社区
- ✓ “API经济”
- ✓ 对接互联网合作伙伴
- ✓ 服务传统DMA客户
- ✓ 仿真交易平台开放给一切量化交易开发者

广发证券互联网金融技术体系全景



金融电商背后的技术

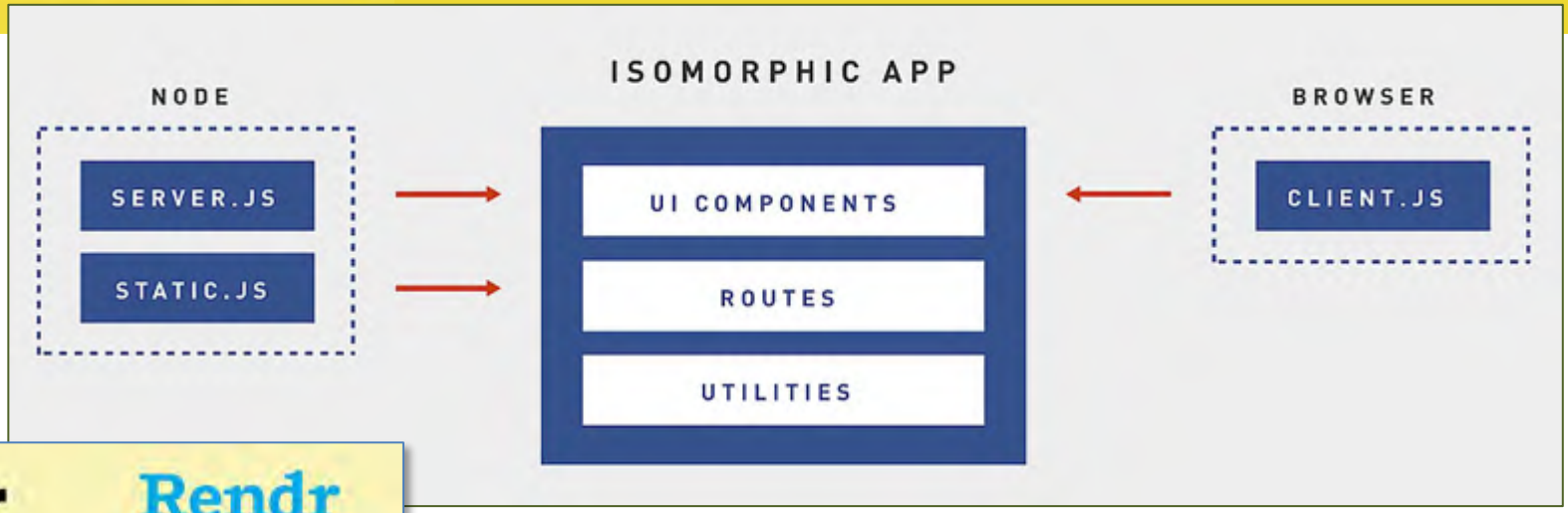
MEAN Stack + Micro Services + 大数据

MEAN Stack + Ionic

- MongoDB、Express、AngularJS、Node.js – 全栈 JavaScript
- 服务器端
  - PHP? 我们不是它的粉丝
  - Java ? 太Verbose的语言，JSON和POJO还要来回转换
  - 我们喜欢Node.js – 事件驱动、异步编程、高并发、与客户端语言及数据对象同构
- 数据存储
  - MongoDB – JSON格式存储中间结果、灵活、不仅是简单的缓存（复杂查询、Aggregator、MapReduce）
  - 无传统“SQL注入”类攻击之虞；采用最佳实践防范“服务器端JavaScript注入”攻击
- AngularJS - 依赖注入、双向绑定、强大开源社区支持
- Ionic + AngularJS 支持 “移动先行” 策略
- ES6 Now with Babel

“全栈同构”

- 解决符合中国国情的浏览器兼容性问题
- 优化“互联网最后一公里”的SPA下载
- 支持搜索引擎索引



Meteor Rendr

LoopBack React

mootools lazojs

- 移动端：AngularJS + Ionic
- Web端：
  - Koa + Flux + React?
  - Meteor?
  - Derby、Rendr、Flatiron… ?

mongoDB 实践

- 我们喜欢的
  - 模型设计简单灵活 – 符合电商产品数据模型EAV/稀疏矩阵的特点
  - 原生支持数据集复制与分片，实现高可用和分片技术架构相对简单直接
  - 天然符合“全栈同构”技术体系
  - 一些小特性比较实用。例如TTL索引，GEO索引等
  - 内置GridFS，是一个不错的分布式文件系统

- 让我们掉坑里的
  - 金融数据要求严谨，schema-less的数据库迫使应用程序贯彻的强数据类型的schema（并且保证多个应用程序遵循），增加应用程序负担
  - 2.x版本里锁的粒度太粗，一个数据集忘记加索引，导致整个数据库巨大性能问题
  - 查询不够灵活，对于稍微复杂的查询一般都需要用到map-reduce、aggregate才可以满足
  - 3.0以前，批量删除性能比较差；由于每个文档都保存字段名，大多数情况，占用的空间比关系型数据库都要大

注意

1、日常运维的时候，要经常留意日志。找出那些耗时比较长的语句，进行优化，否则可能导致整个数据库不可用
2、默认是不启用用户认证，生产中的机器需要注意
3、在适当的场合，最好在应用层进行建模。过于灵活的代价可能是难以维护，特别是核心的业务。

Micro Services Architecture – 金融服务中后台“救星”

- 券商传统系统（legacy）支持互联网业务时，需封装各种服务（Request/Response、Pub/Sub）
- 绝大部分封装服务只考虑提供“接口”以对接Web层，简单粗暴
- 传统系统本身的架构设计不是为高并发的互联网应用特点而设计，几乎无法优化扩容
- 服务数量巨大无法管理
- 历史遗留系统由五花八门的语言写成

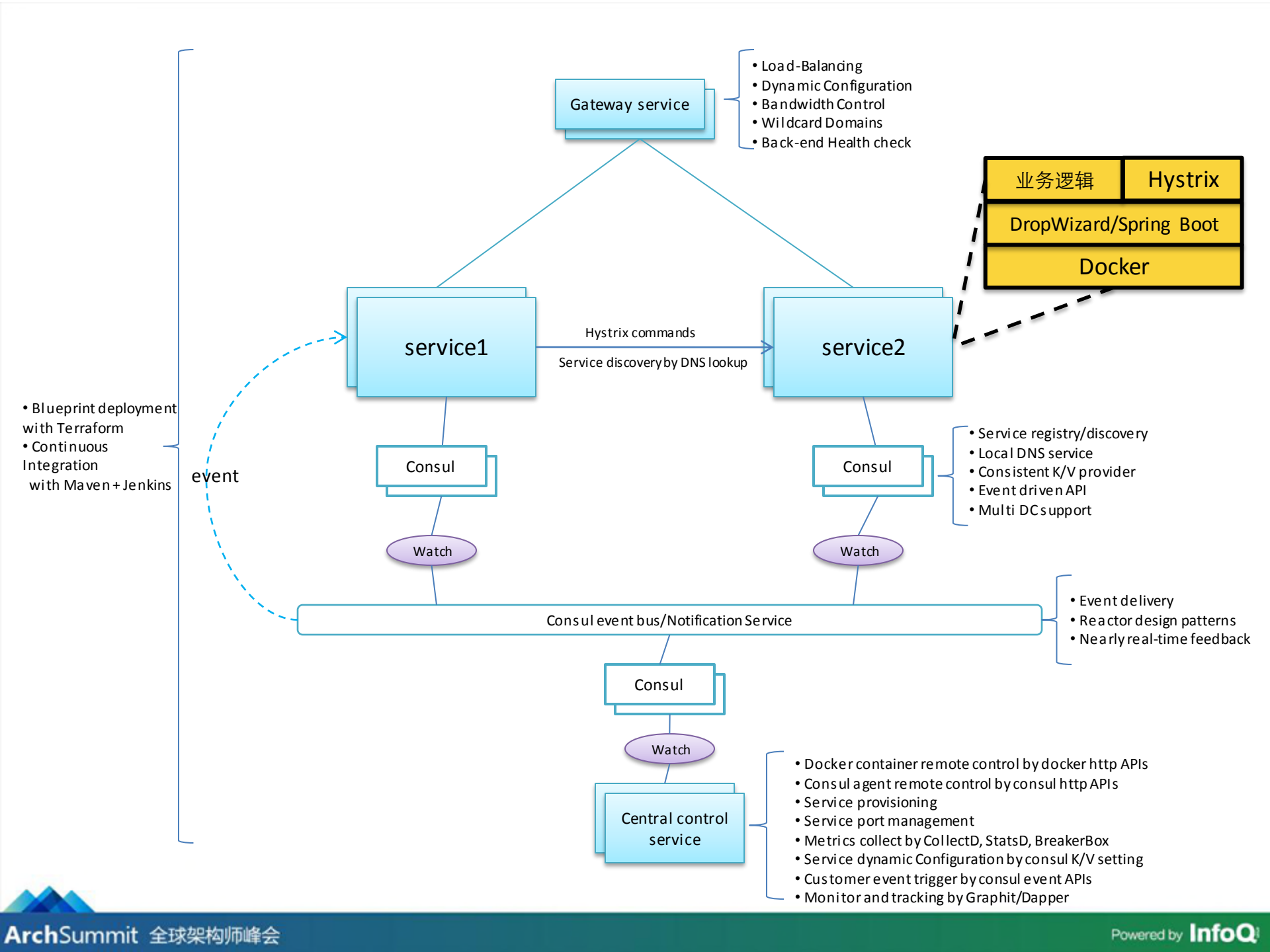
问题

- Polyglot（混合编程）
- Hybrid Persistence（混合存储），“加厚”服务层，缓冲对旧系统压力
- 大量服务通过统一服务注册、服务发现机制来管理
- 利用健壮性（Resiliency）架构范式（Architecture Patterns）提高分布式架构下服务依赖、调用的健壮性
- 利用“容器化”获得水平扩容、部署自动化能力
- 平台Performance Metrics监控

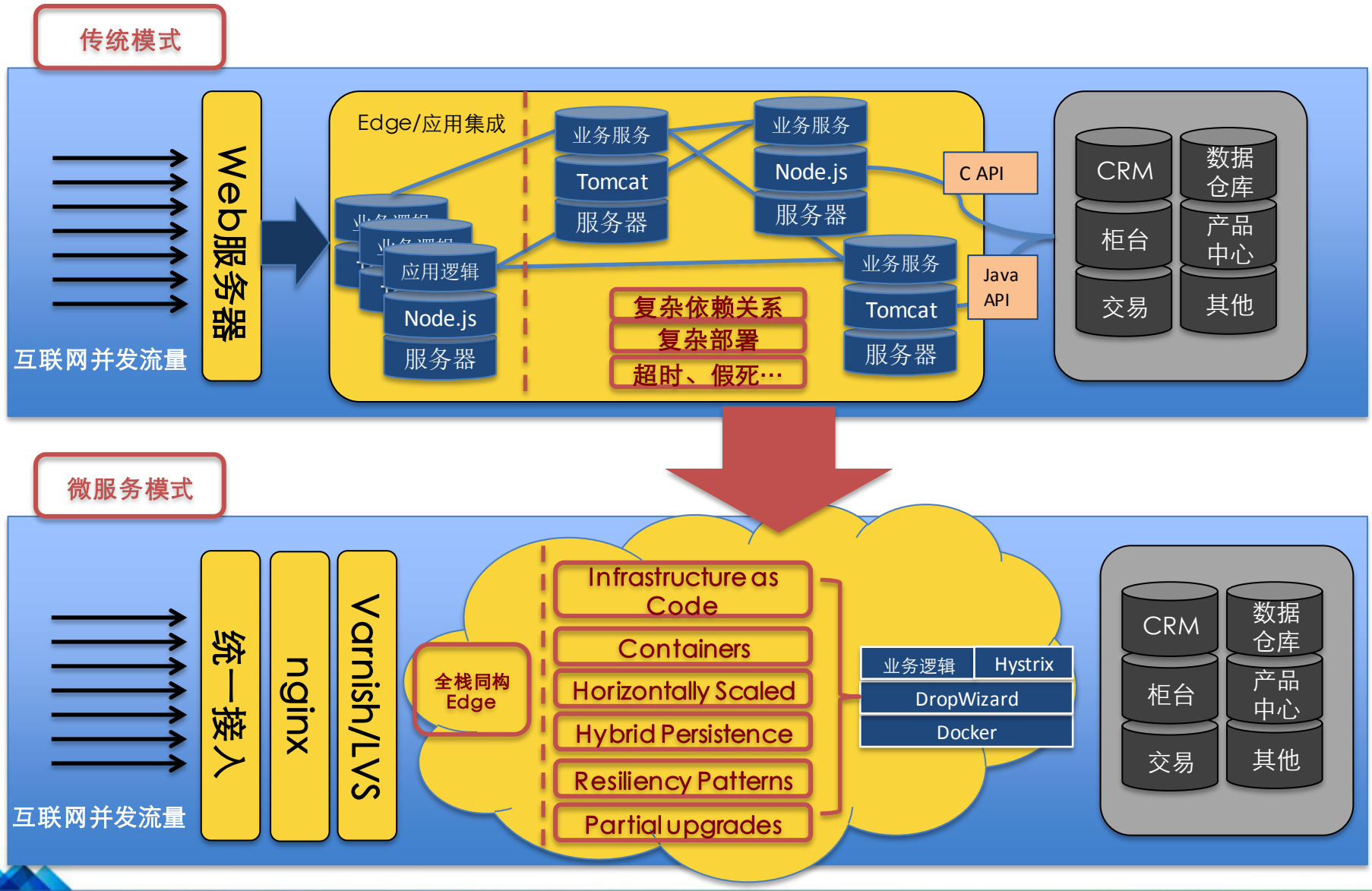
方案

Consumer-driven contract test	Pact、Pact-jvm			
API Doc & Pub	Swagger.io		Slate	
Service Discovery	Consul	Curator	SmartStack	Etcd
Service framework	DropWizard		Spring Boot	
Resiliency	Netflix Hystrix		Spring Cloud	
Async event-driven programming	ReactiveX			
Container + Orchestration	Docker + Kubernetes + Mesos + Open vswitch/Weave + Terraform			

广发电商中后台微服务API

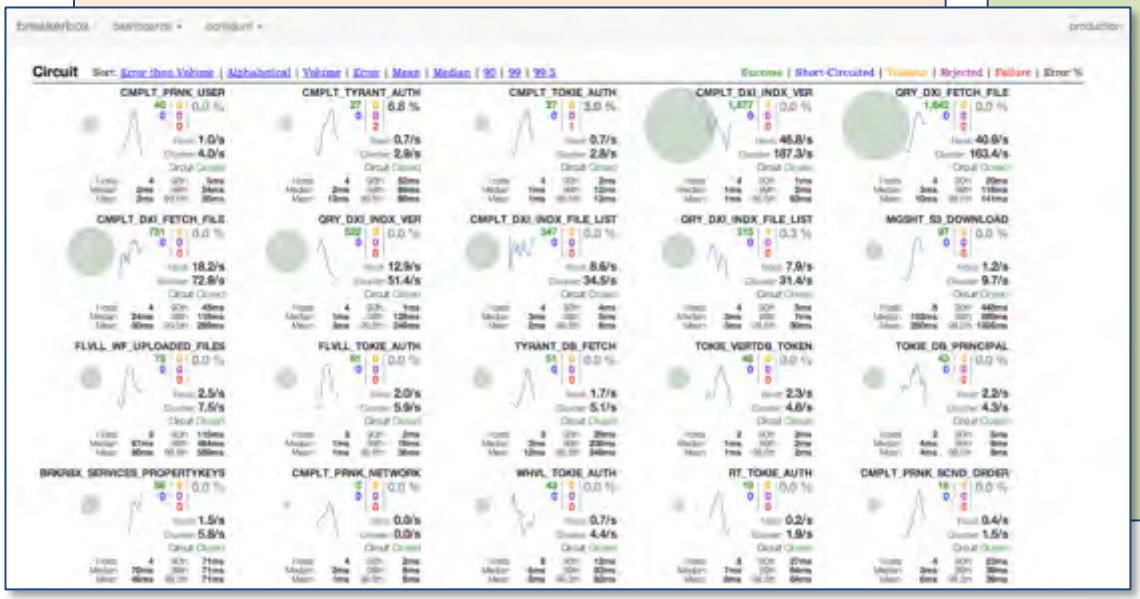

Gateway service
• Load-Balancing
• Dynamic Configuration
• Bandwidth Control
• Wildcard Domains
• Back-end Health check
业务逻辑
Hystrix
DropWizard/Spring Boot
Docker
service1
Hystrix commands
Service discovery by DNS lookup
service2
• Blueprint deployment with Terraform
• Continuous Integration with Maven + Jenkins
event
Consul
Consul
• Service registry/discovery
• Local DNS service
• Consistent K/V provider
• Event driven API
• Multi DC support
Watch
Watch
Consul event bus/Notification Service
• Event delivery
• Reactor design patterns
• Nearly real-time feedback
Consul
Watch
Central control service
• Docker container remote control by docker http APIs
• Consul agent remote control by consul http APIs
• Service provisioning
• Service port management
• Metrics collect by CollectD, StatsD, BreakerBox
• Service dynamic Configuration by consul K/V setting
• Customer event trigger by consul event APIs
• Monitor and tracking by Graphit/Dapper

微服务让券商后台封装服务在互联网链路上“健壮化”



Micro Services要素

- 框架
  - 服务动态配置（Dynamic Configuration）
  - 指标监控（Application Metrics）
  - 仪表板工具支持（Dashboard）
  - 日志（Logging）
  - 运维工具箱（Operation tool box）



- 健壮性架构范式（Resiliency Patterns）
  - Circuit Breaker
  - Intelligent routing
  - Micro proxy
  - Control bus
  - One-time token
  - Global lock

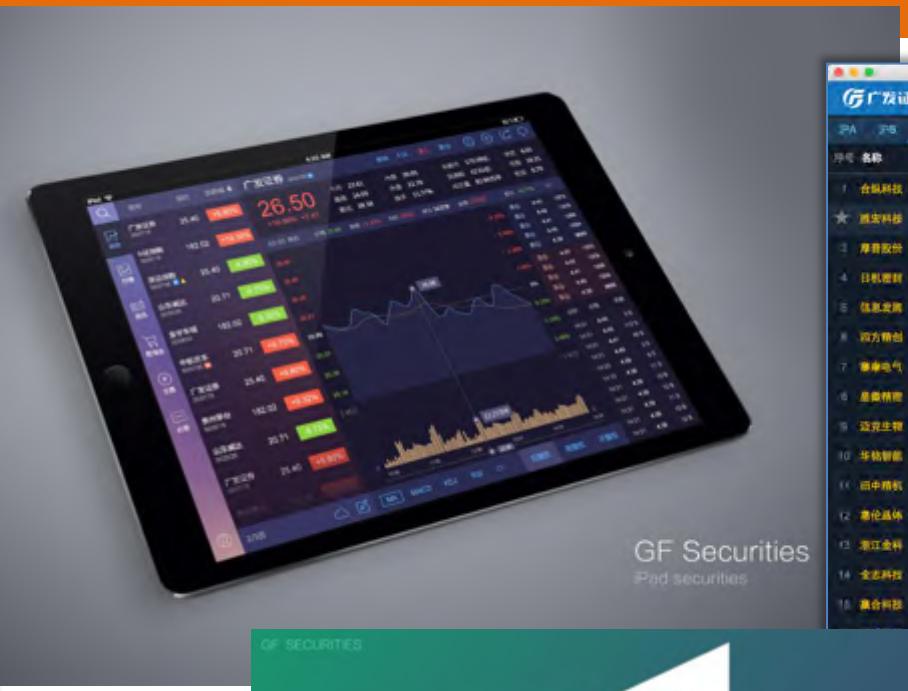
Leadership election
Distributed sessions
Cluster state
Re-try
Time-out
Fail-fast
More…

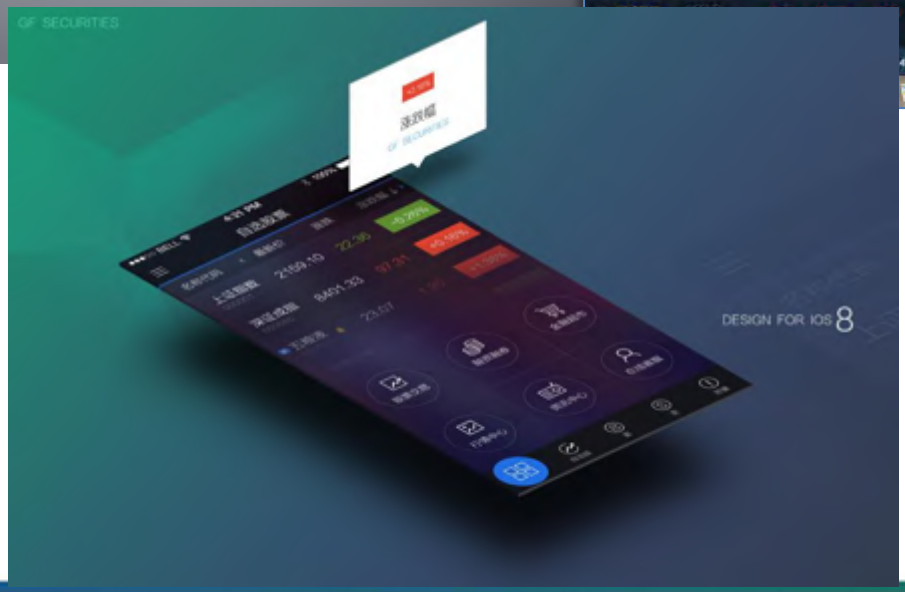
DropWizard

广发泛交易终端架构

容器化、依赖注入、DDD

泛终端 – 手机、平板、Web、PC/Mac交易终端

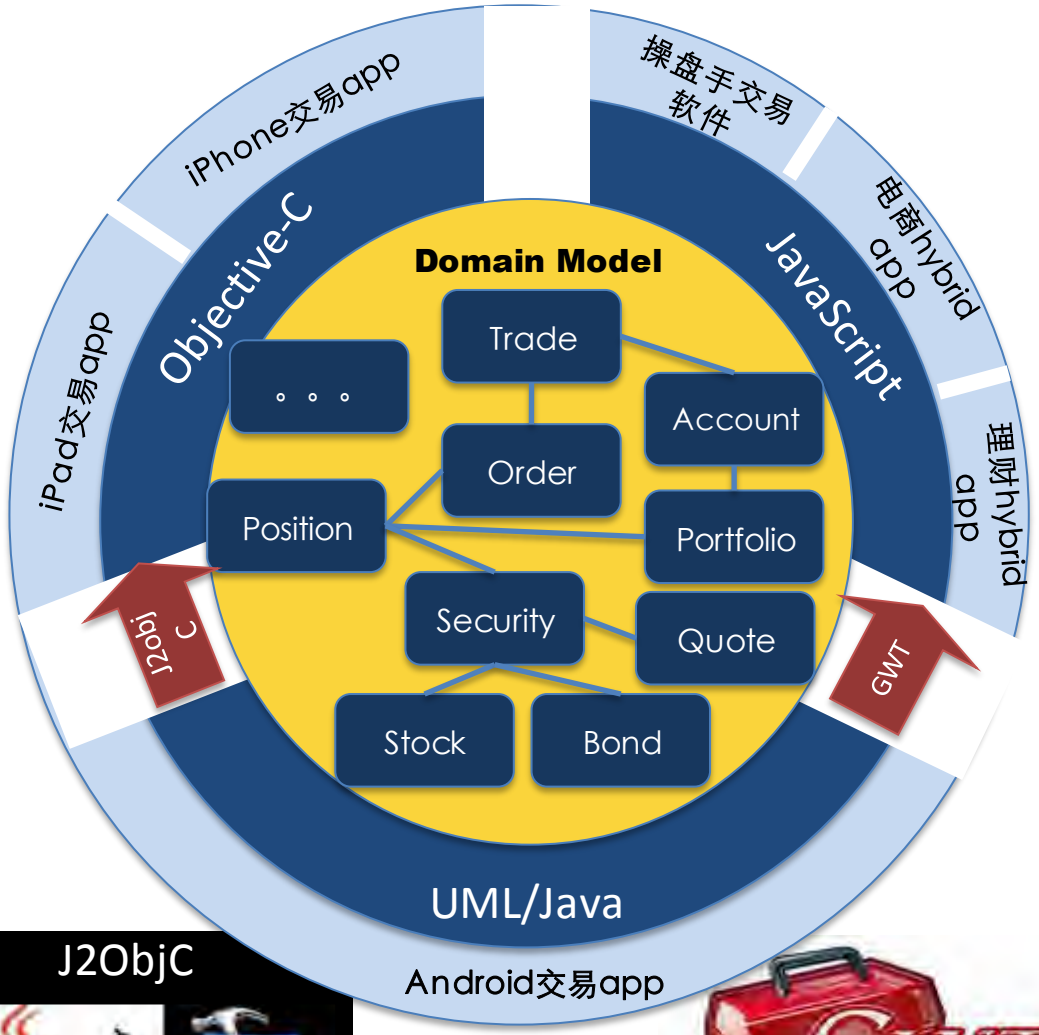




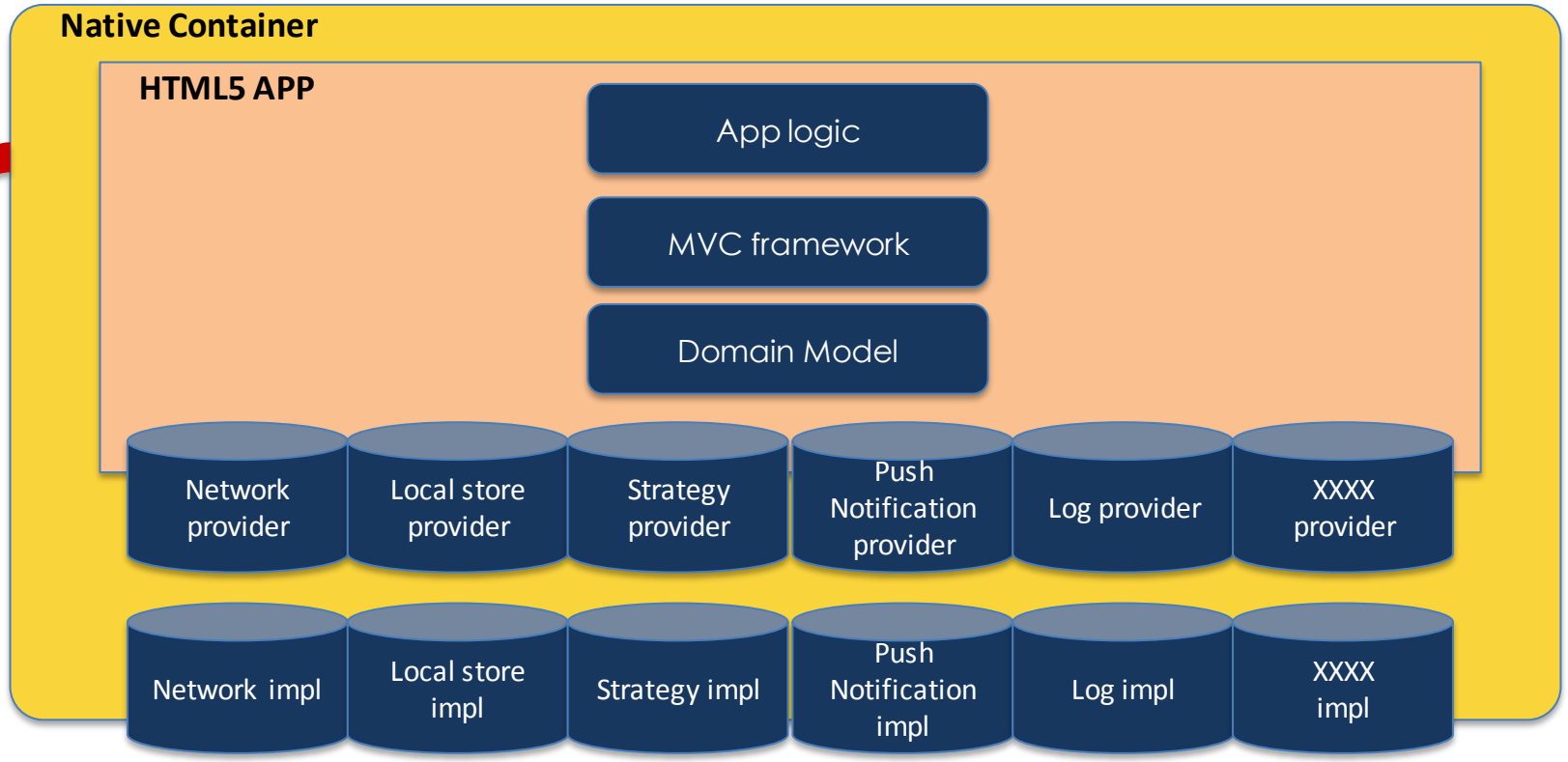
- 证券投资者使用终端的多样性
- 全终端覆盖的开发挑战
- 多终端协同的必要性

DDD – 域模型驱动开发

- **挑战**
  - 非常复杂的业务领域
  - 基本业务元素在多终端、多系统的反复应用
  - 新业务基于相对稳定的基础业务元素的扩展
  - 掌握核心业务知识的工程师很少
- 解决方法
  - 用良好的面向对象设计对业务领域建模（Domain Model）
  - 广泛采用设计模式（Design Patterns）贯彻“松散耦合”原则
  - 既懂技术又懂业务的专家掌控核心“域模型”
  - “域模型”跨语言、跨平台、跨系统通用
  - “域模型”独立可测试、扩展
  - 应用层工程师使用域模型开发应用



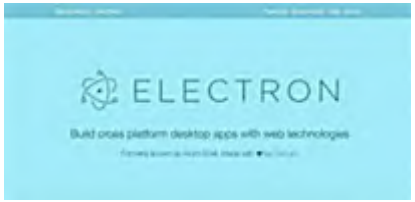
容器化与依赖注入



“古代”浏览器

“现代”浏览器

Chrome app

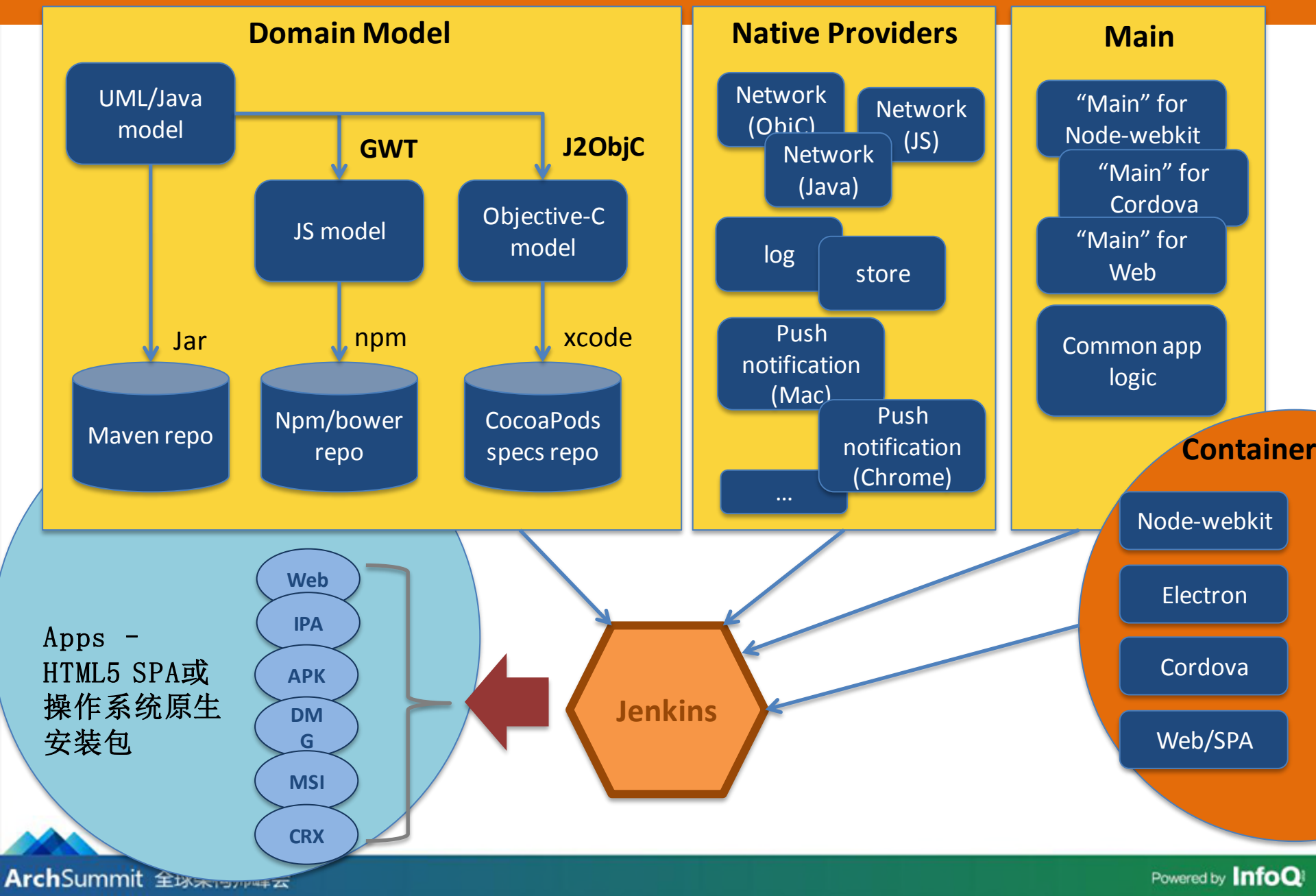


Github Electron

同一个HTML5应用在不同容器的性能优化

	标准浏览器	Chrome	Chrome App	Electron Node-webkit Cordova/Crosswalk
本地存储	浏览器内，小	浏览器内、indexDB	多种技术选项	操作系统支持的任何技术
网络连接	不是所有都支持WebSocket	可用WebSocket、QUIC	可用WebSocket、QUIC	WebSocket、TCP/UDP、anything
协议解码	效率低	效率低	可用原生插件	可用C/C++/Java
图形渲染	不一定支持硬件加速	Canvas、硬件加速、Chrome实验扩展	Canvas、硬件加速、Chrome实验扩展	Canvas、硬件加速
安全沙箱	浏览器安全沙箱	浏览器安全沙箱	NaCL	调用操作系统服务
消息推送	不支持	GCM	GCM、自定义	自定义、Mac OSX消息中心、Windows消息中心
桌面控制	无	无	Chrome app launcher	操作系统桌面应用
优化空间	HTML5范围内	利用Chrome特性（SPDY、Pre-render、QUIC…）	除Chrome所有特性外，再加PNaCL，近原生应用性能级别	近原生应用性能级别
安装方式	页面加载	页面加载	Web store	安装包、App store

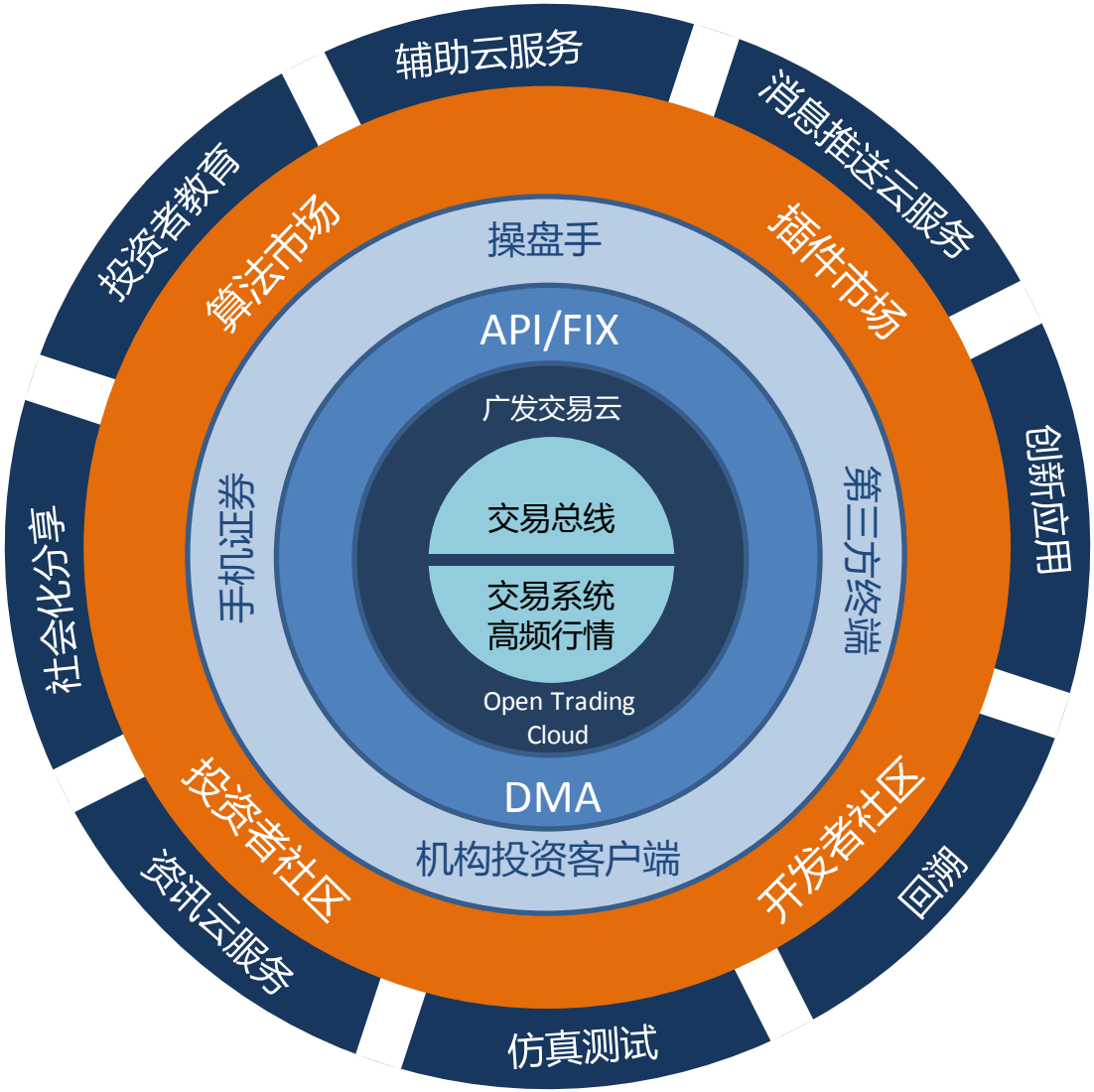
“抓中药”式构建、打包、集成


Domain Model
UML/Java model
GWT
J2ObjC
JS model
Objective-C model
Jar
npm
xcode
Maven repo
Npm/bower repo
CocoaPods specs repo
Native Providers
Network (ObjC)
Network (JS)
Network (Java)
log
store
Push notification (Mac)
Push notification (Chrome)
...
Main
“Main” for Node-webkit
“Main” for Cordova
“Main” for Web
Common app logic
Container
Node-webkit
Electron
Cordova
Web/SPA
Jenkins
Apps - HTML5 SPA或操作系统原生安装包
Web
IPA
APK
DMG
MSI
CRX

Open Trading – 广发交易云

行情与交易开放API、开发者社区

“API经济” – 云服务打造行业生态圈



开发者社区
- 机构投资者
- 互联网合作公司
- 工程师里的投资者
- 投资者里的程序员

主要技术
- API Gateway
- Time-series Database
- 基于PGM组播的消息总线
- Docker+Kubernetes
- FIX协议实现
- Java+Go
- Dapper-like分布式系统跟踪技术
- Disruptor
- 大量分布式架构设计模式

HPC 交易中间件 – Mechanical Sympathy

存储	延迟
DRAM	~65ns
MC	~20ns
L3 cache	~15ns
L2 cache	~3ns
L1 cache	~1ns
Register	<1ns

* Martin Thompson QCon SF 2010 数据

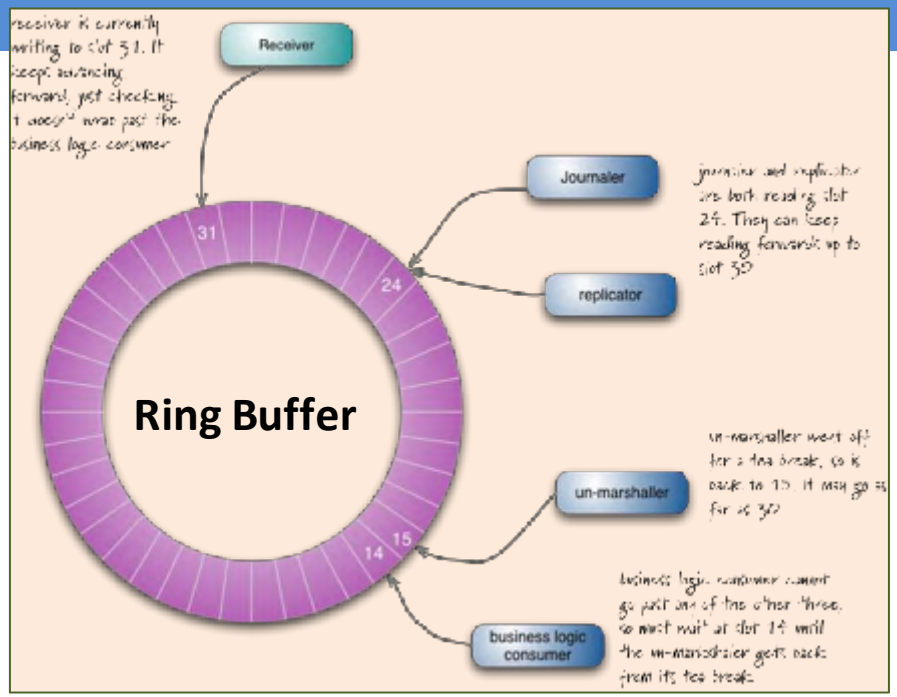
交易系统中间件的挑战：

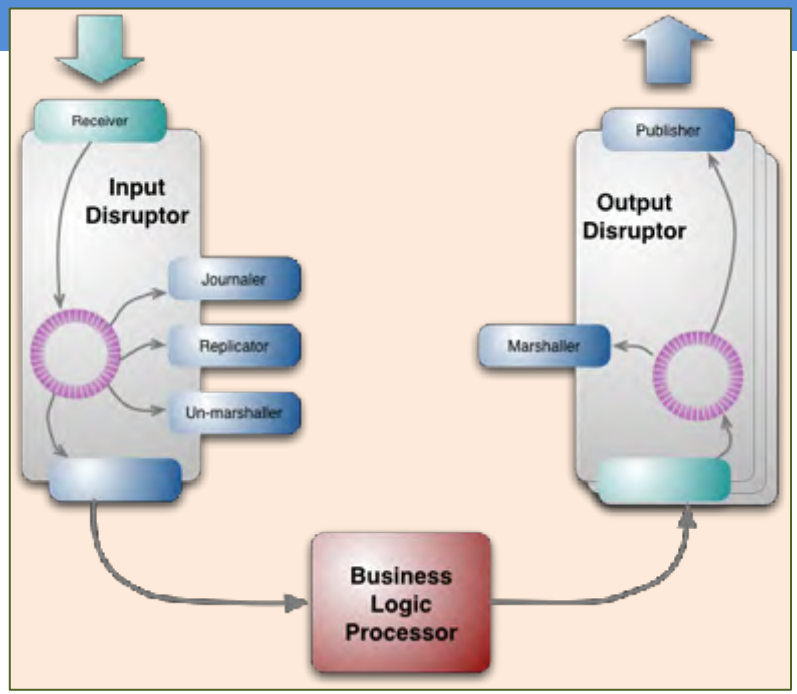
- 超低延迟
- 超高并发
- 分片与水平扩容难
- 都是事务型操作

基于Java的分布式系统，比C++开发效率高、相对易维护，也可以实现HPC - Mechanical Sympathy是关键

- 利用高性能网络技术
  - Java Sockets Direct Protocol (SDP) 支持 Infiniband RDMA
  - Solarfare OpenOnLoad TCP kernel bypassing
  - ZMQ OpenPGM extension 让我们获得PGM可靠多播
- 正确使用存储
  - 传统硬盘读写也可以很快，只要是顺序的（避免随机访问）- Kafka证明其既不丢失数据又保证高性能，系统架构简单，非C/C++技术亦可达成
  - SSD用于随机访问，大幅提高RDBMS和MongoDB等性能
- 内存与线程管理
  - 避免cache hit miss - HPC算法里最大性能损耗
  - 慎用多线程锁技术
  - Queue作为线程间通讯手段有各种复杂问题，Java里的Queue还是导致垃圾收集的源头
  - GC影响交易系统性能，需要避免或可控

交易总线 - LMAX Disruptor + PGM + 仲裁/排队/去重



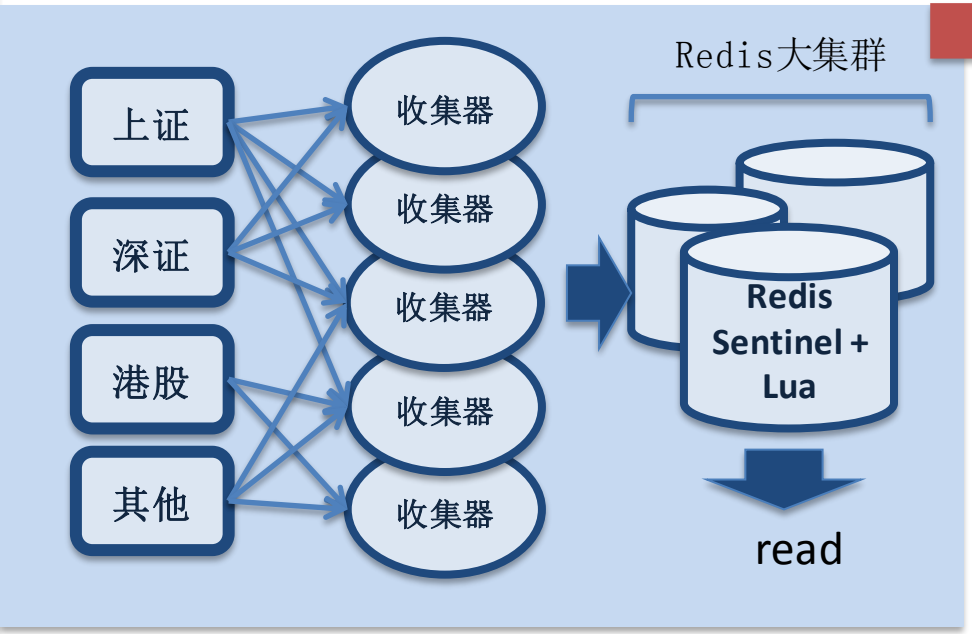


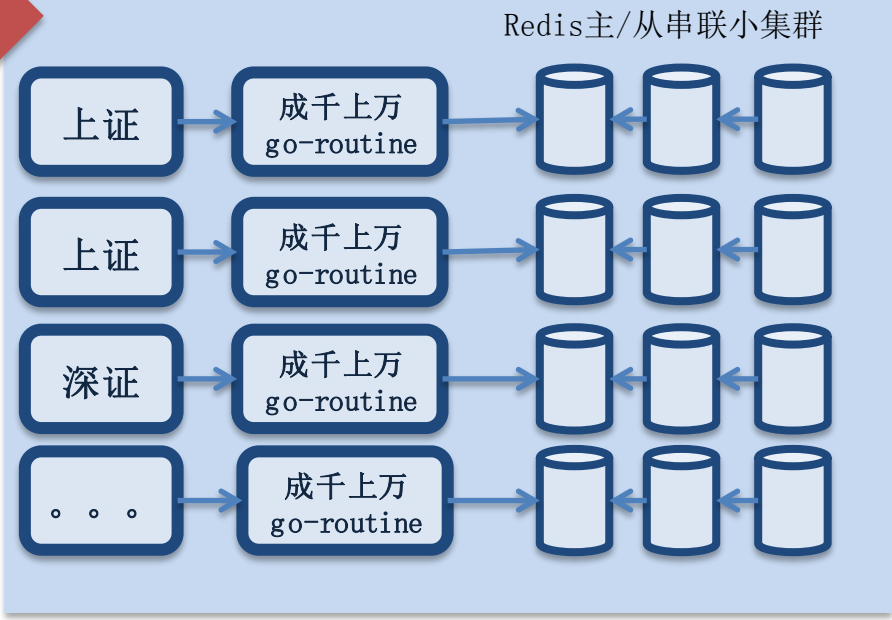
（插图来自Martin Fowler 关于LMAX Disruptor的文章）

- ✓ HPC最大性能挑战之一 Cache hit missed（缓存不命中）由神奇的LMAX Disruptor解决，单线程无锁、GC可控，避免线程间Queue低效
- ✓ 交易总线节点高可用、“瞬间切换”由PGM可靠多播同步数据实现
- ✓ 冗余数据不能报盘到交易所 - 基于Event Sourcing、Competing Consumers、Leader Election、CQRS、Queue-based load leveling、Pipes-n-filters等实现消息流水线、排队机、仲裁机、去重

行情指标计算 – 挪计算还是挪数据？

1. 分时(交易价格和交易量在划分到一个小的时间区间得出的连续统计点)
2. K线(5、10、15、30、60分钟、日、周、月、季、年)
3. 分笔(对交易所每笔成交数据的展现)
4. 排行(涨跌幅、开盘价、最新价成交量等等)
5. 其他指标(内外盘、换手率、市盈率等等)





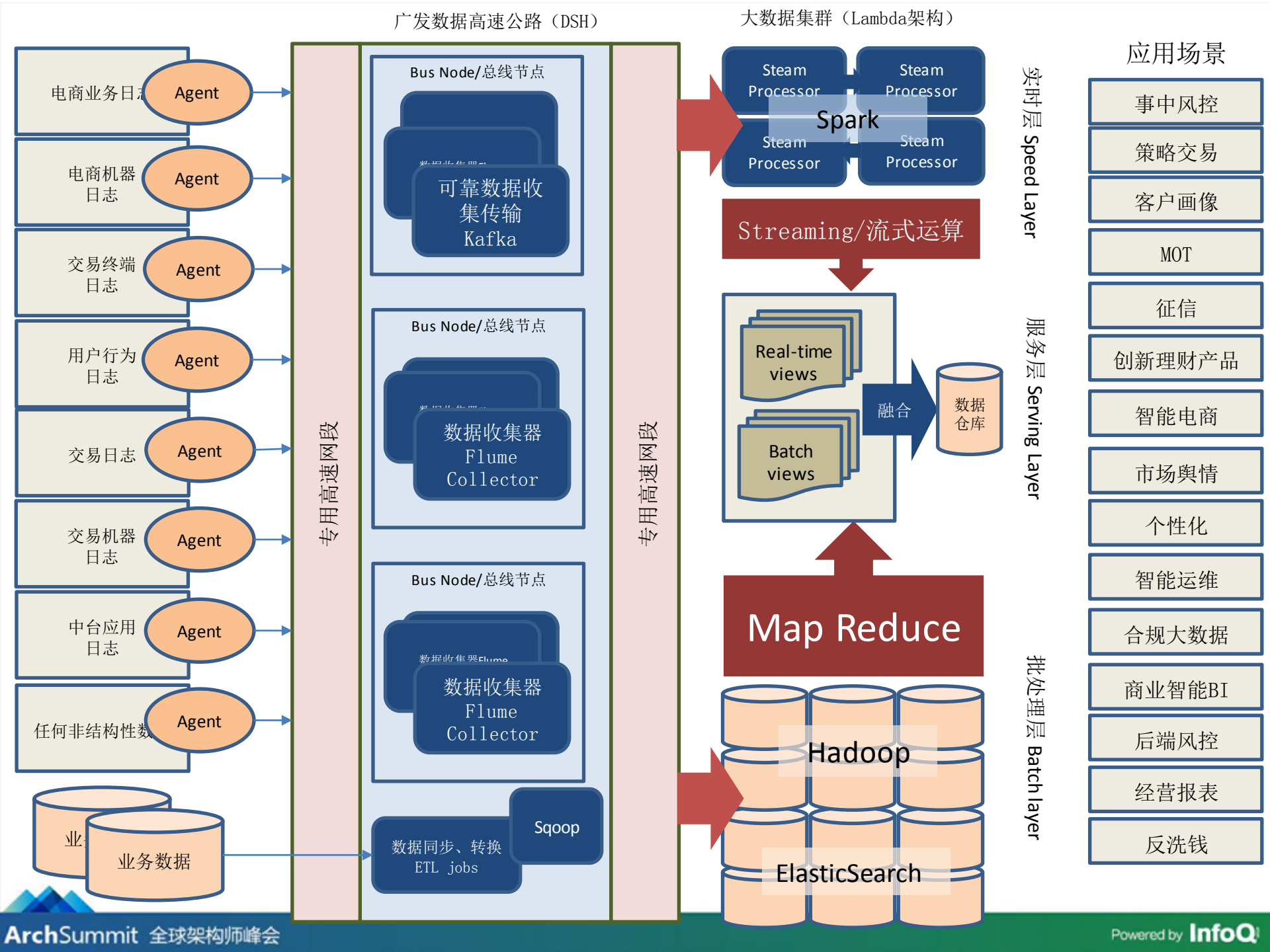
- Move Compute to Data：Lua在Redis分片内就地计算
- 减少10倍数据交换量
- 单线程Redis内Lua计算导致查询阻塞
- 减少阻塞要加大Redis分片数量
- 资源消耗、数据修正困难、迁移不便

- Move Data to Compute：数据挪到Go服务作计算
- 一个股票对应一个go-routine，大规模并行
- 市场独立不分片
- Redis主从读写分离
- 运维简单、扩展性好、数据故障修复简单

λ architecture

大数据无处不在

（屏幕图片均为真实App截图）


广发数据高速公路（DSH）
大数据集群（Lambda架构）
电商业务日志
Agent
电商机器日志
Agent
交易终端日志
Agent
用户行为日志
Agent
交易日志
Agent
交易机器日志
Agent
中台应用日志
Agent
任何非结构性数
Agent
业务数据
专用高速网段
Bus Node/总线节点
可靠数据收集传输 Kafka
Bus Node/总线节点
数据收集器 Flume Collector
Bus Node/总线节点
数据收集器 Flume Collector
数据同步、转换 ETL jobs
Sqoop
专用高速网段
Steam Processor
Steam Processor
Spark
Steam Processor
Steam Processor
Streaming/流式运算
Real-time views
Batch views
融合
数据仓库
Map Reduce
Hadoop
ElasticSearch
实时层 Speed Layer
服务层 Serving Layer
批处理层 Batch layer
应用场景
事中风控
策略交易
客户画像
MOT
征信
创新理财产品
智能电商
市场舆情
个性化
智能运维
合规大数据
商业智能BI
后端风控
经营报表
反洗钱

DevOPS – 运维工具与理念

“measure anything, measure everything” - Etsy

“天量交易” + “互联网化” + “国际化” 的三大挑战

- 互联网业务的挑战
  - 24x7
  - 高并发
  - 部署扩容响应快
- 屡创新高的交易量挑战
  - 更丰富、更复杂金融产品
  - 黑天鹅事件频发
  - T+0终会来临
- 国际化早晚到来
  - 多交易市场
  - 24小时滚动清算
  - 停机时间窗口小

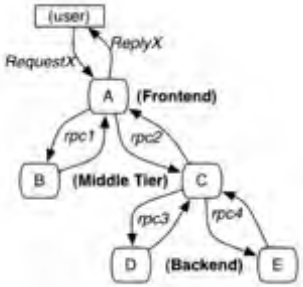
行业运维的典型现状

- ✓ 人肉监控、人肉部署
- ✓ 开发者（商）对机器日志的忽（bu）视（dong）
- ✓ 割裂的监控点，靠人肉挖掘日志恢复“案发现场”（bug scene）
- ✓ 多数限于硬盘、网络、CPU之类的“硬指标”监控，缺乏业务语义的应用监控-发现故障有时晚于客户
- ✓ 扩容能力弱 - 知识结构和财务制度无法支持弹性设备资源池（云）
- ✓ 系统性能优化往往限于数据库

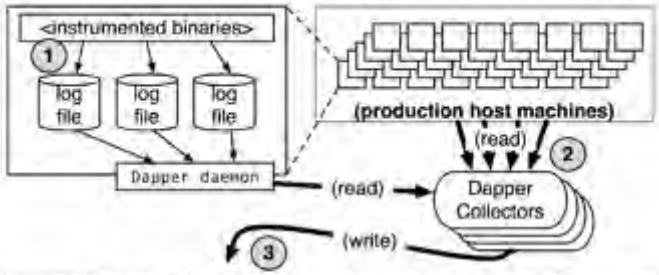
我们需要 “Operational Intelligence”

“不懂运维的架构师不是好程序员”

性能跟踪



基于Google Dapper论文的低消耗、应用级透明、分布式系统跟踪技术内置于一切



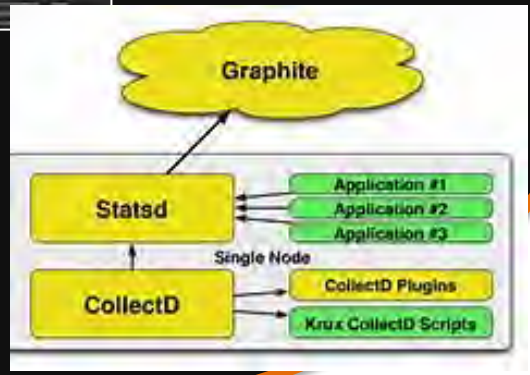
trace id	span 12	span 23	span 34	span 45	span 56	...
123456	nil	nil	<data>	<data>	nil	...
246802	<data>	nil	nil	nil	<data>	...
357913	nil	<data>	nil	nil	nil	...
...	...	...	...	...	...	...

(Central Bigtable repository for trace data)

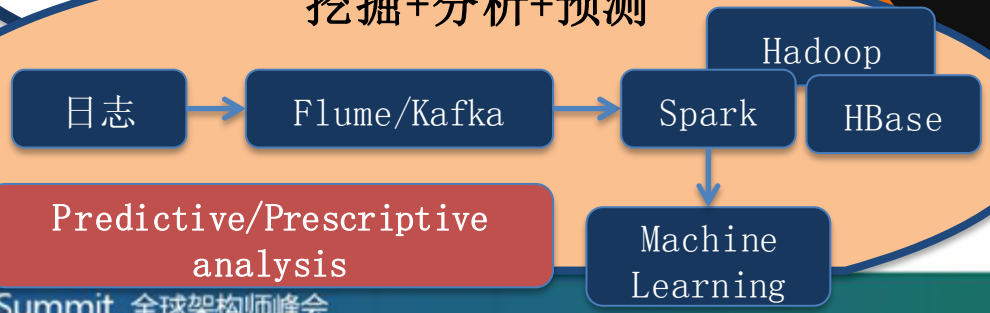
收集+监控+预警

Log Analysis Open Source con Logstash Elasticsearch & Kibana

- Statsd
- Graphite
- Cabot
- InfluxDB



挖掘+分析+预测

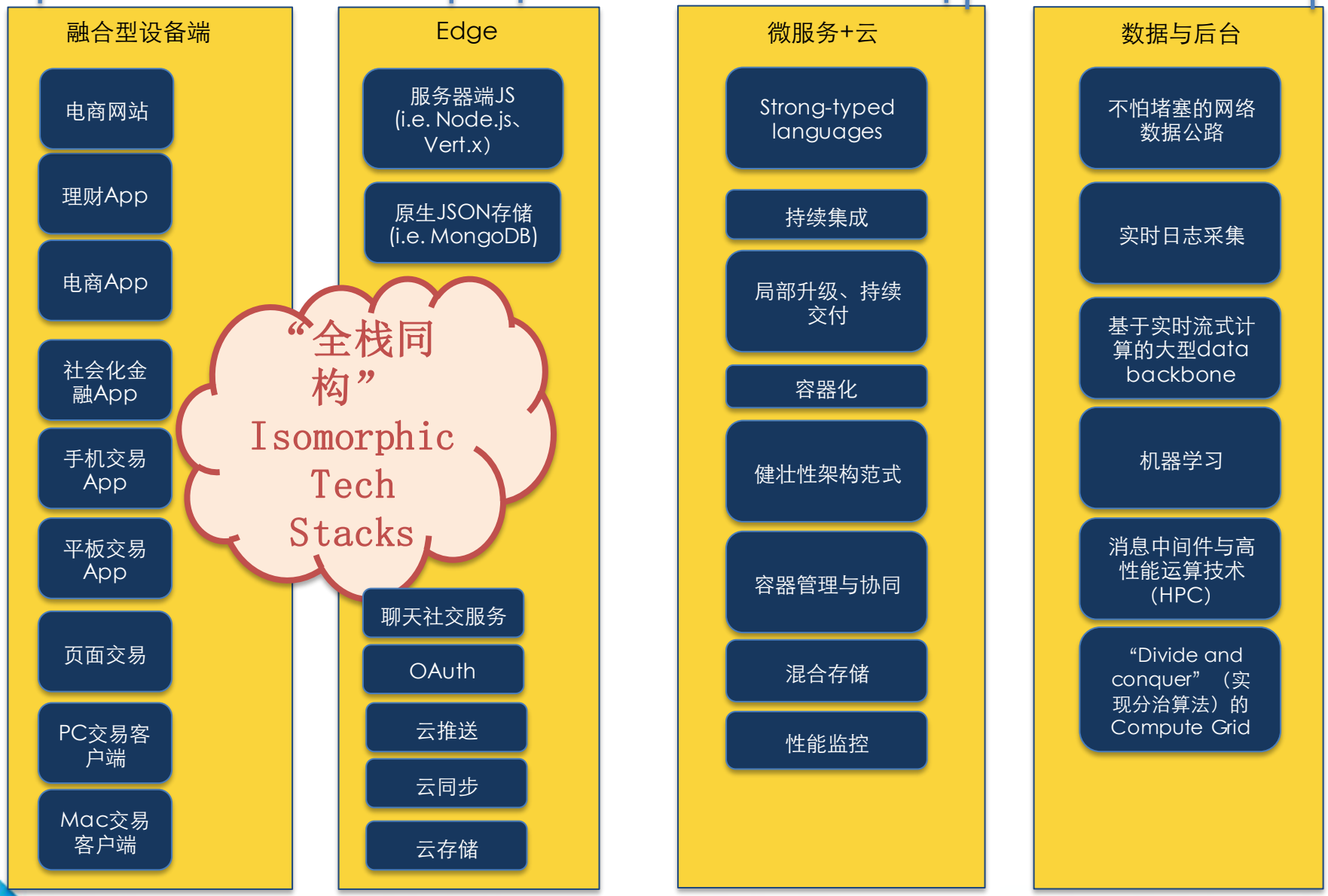


一张图的总结

1 考虑用户体验（浏览器兼容性、SPA下载）+SEO

2 边缘服务与端技术同构，微服务严谨健壮

3 核心技术引擎支持创新



developer.gf.com.cn