ArchSummit

挖财的Scala应用实践

宏江 2015.12

InfoQ

关于我

挖财资深架构师，从事软件开发有十多年
曾在阿里巴巴担任架构师，Ali-tomcat负责人
有丰富的企业开发和互联网架构经验
同时也是函数式编程爱好者，和Scala布道者
个人blog：http://hongjiang.info

1. 开发团队以及后端架构概况

团队状况

- 近一年多时间的技术人员规模

近200人

50人左右

- 挖财不是技术精英型团队，也不太可能走这种路线

挖财的后端技术

在线系统涉及的开源产品和框架：

- Nginx、Tomcat 、Spring-mvc/Play
- Springboot、Dubbo、Zookeeper、Spring、Mybatis
- MySql、Cobar、Hbase、Mongo、Redis/Memcached
- Kafka、Storm
- Zabbix、ELK

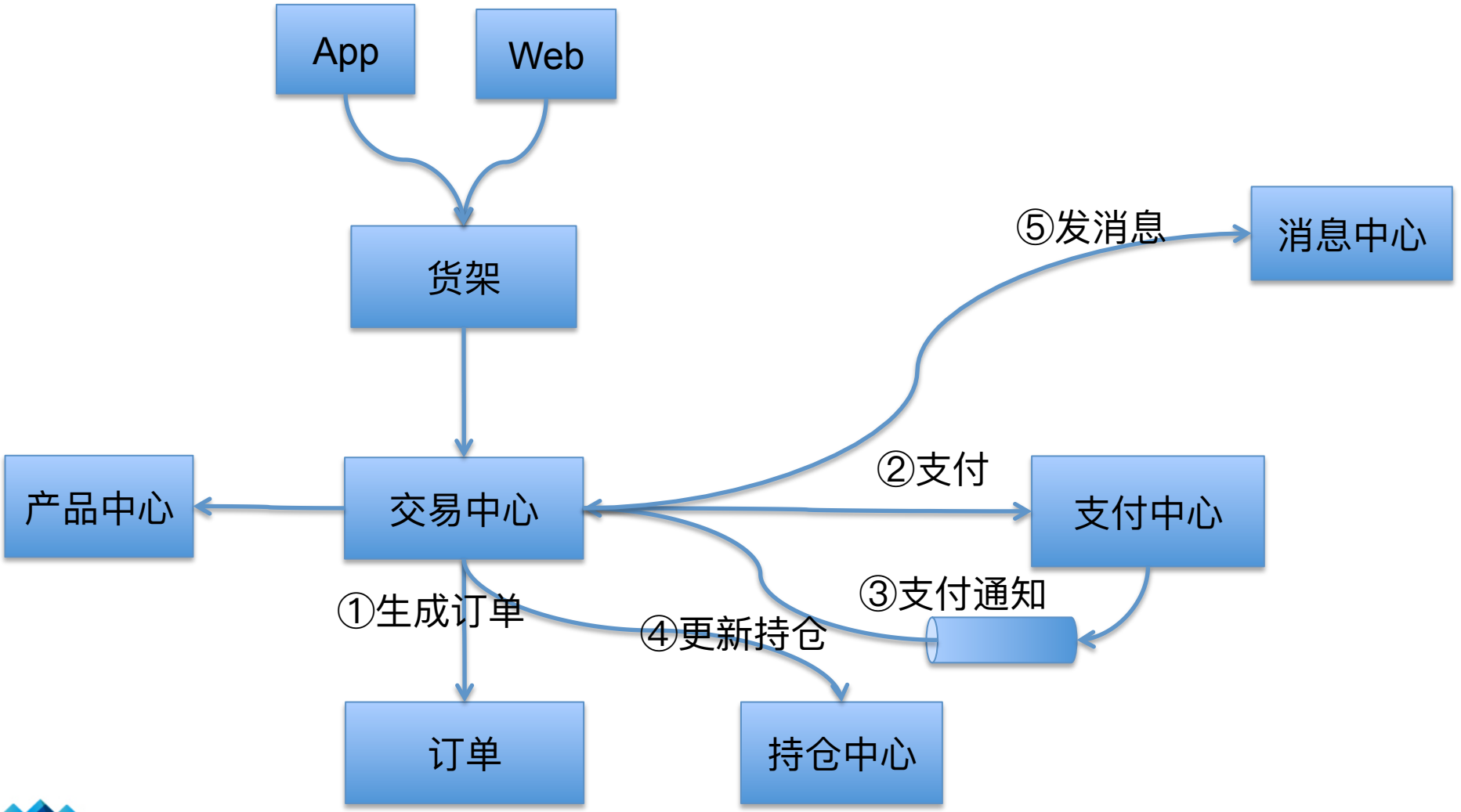
开发语言： Java/Scala, Go, C++, Python/Php

挖财的后端技术

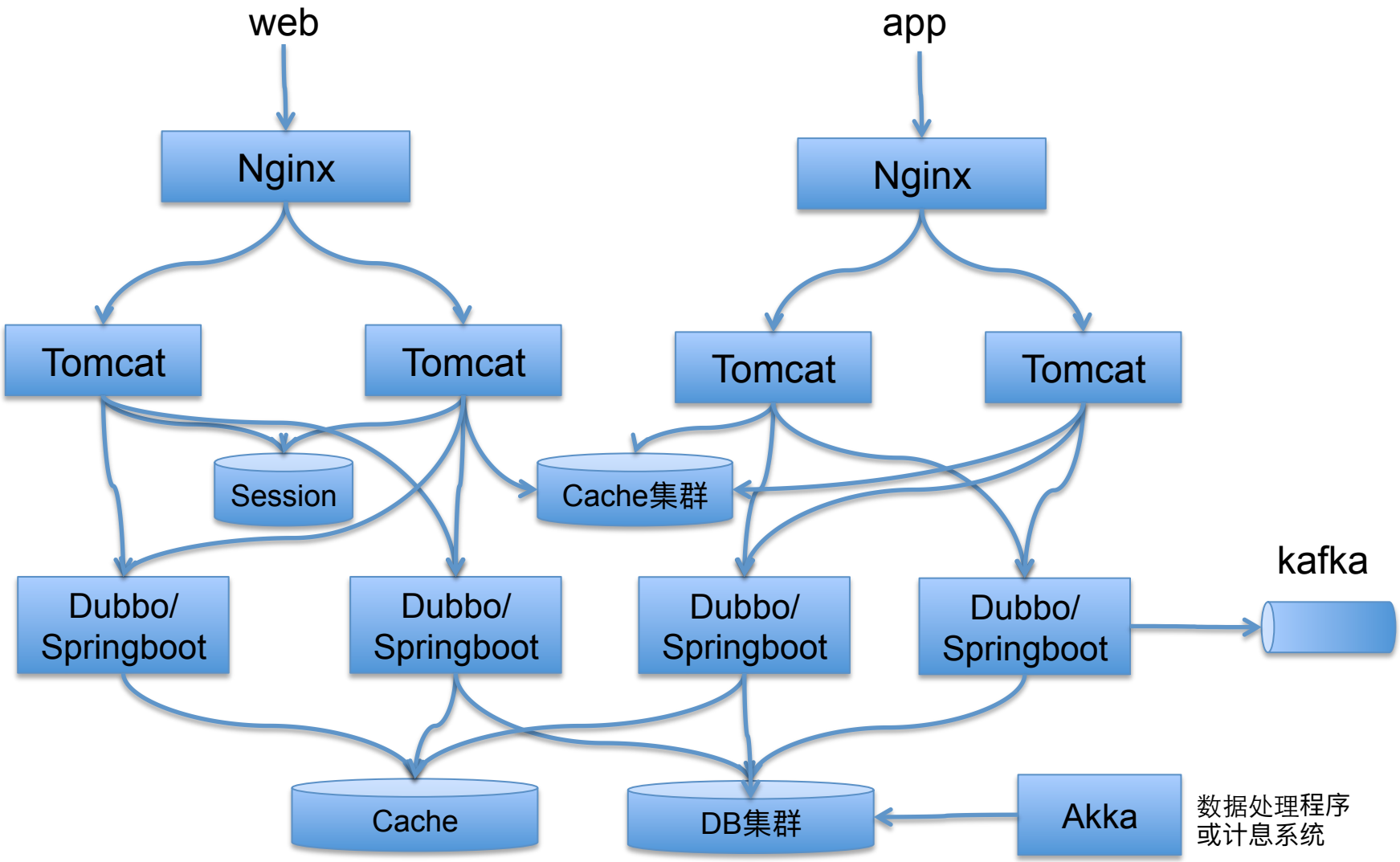
- 遗留问题，将要废弃的：
  - Mongo
  - Memcached
  - 在线业务的python程序
  - 架构决策基于自身的运维能力所考虑

技术架构概况

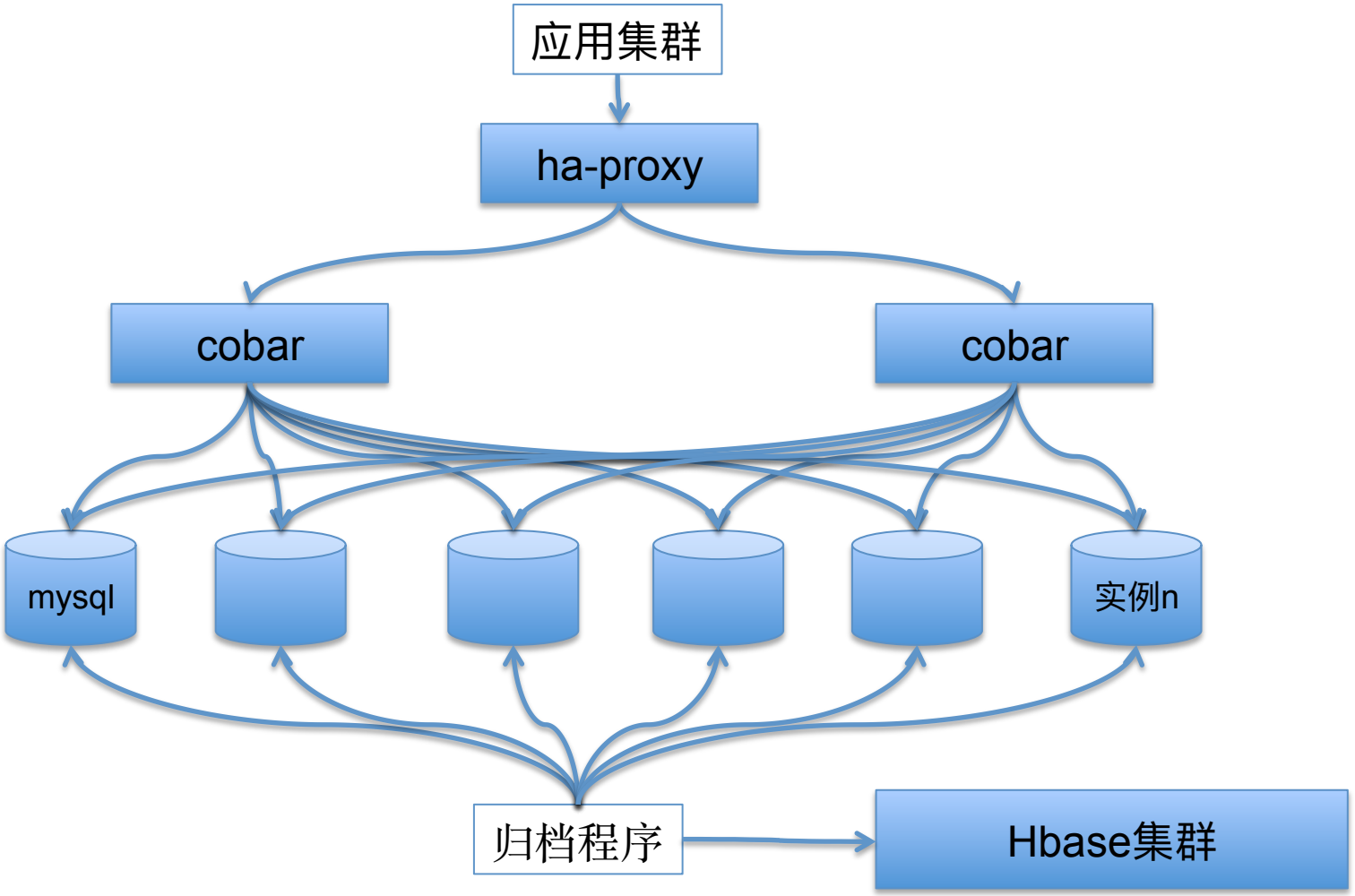
理财业务局部模块


App
Web
货架
⑤发消息
消息中心
产品中心
交易中心
②支付
支付中心
③支付通知
①生成订单
④更新持仓
订单
持仓中心

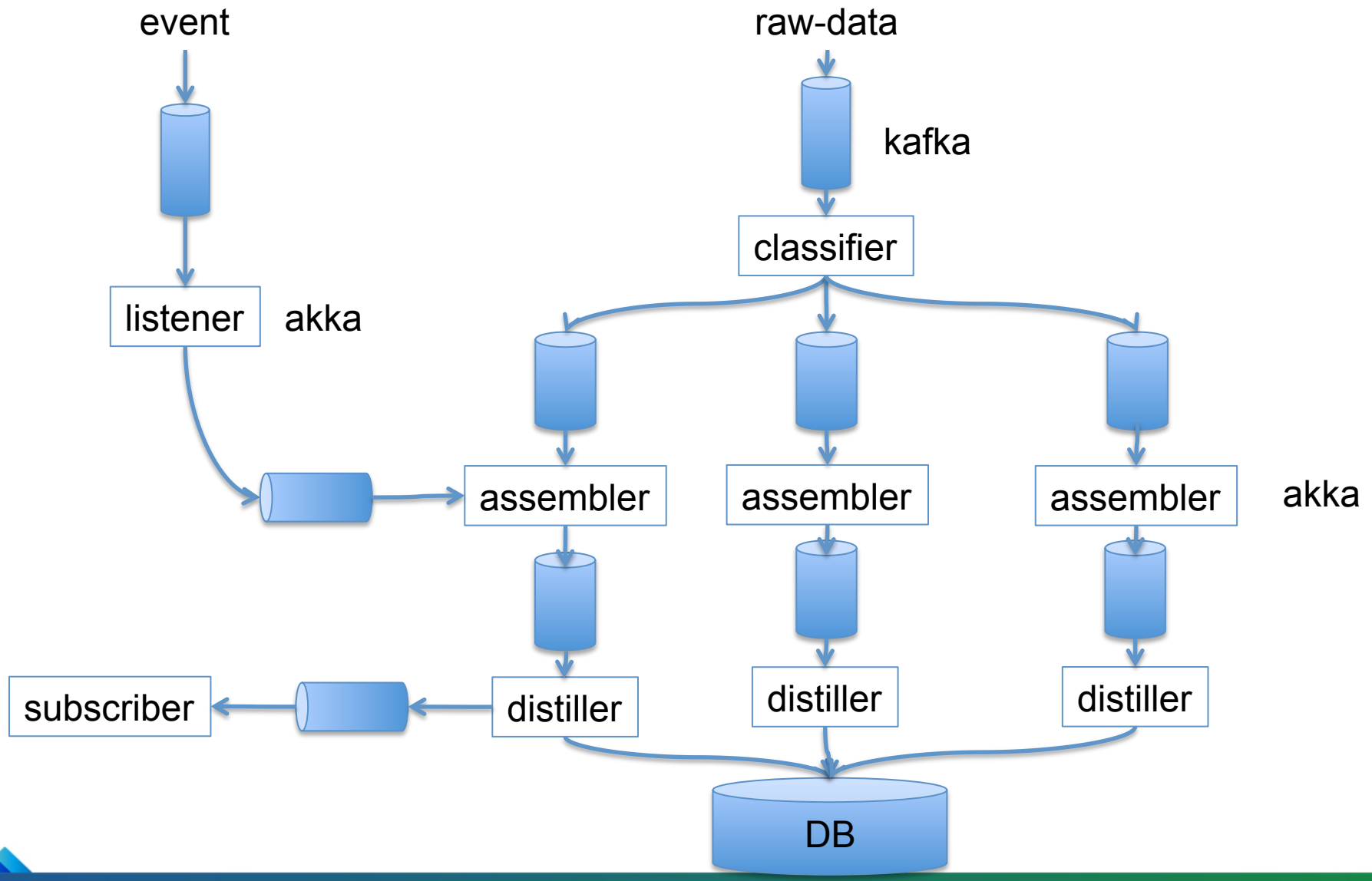
典型的技术架构


web
app
Nginx
Nginx
Tomcat
Tomcat
Tomcat
Tomcat
Session
Cache集群
kafka
Dubbo/
Springboot
Dubbo/
Springboot
Dubbo/
Springboot
Dubbo/
Springboot
Cache
DB集群
Akka
数据处理程序
或计息系统

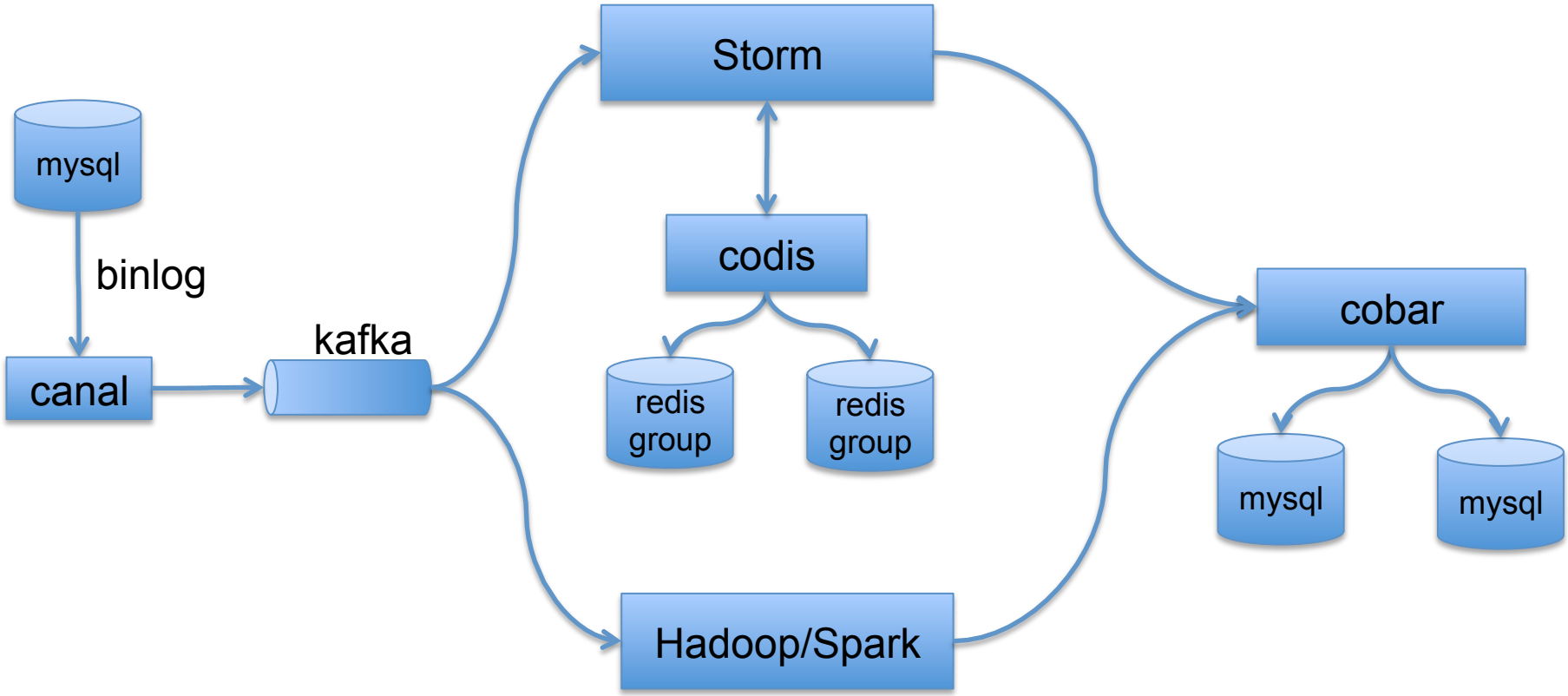
记账数据存储



自动记账数据处理


event
raw-data
kafka
classifier
listener
akka
assembler
assembler
assembler
akka
subscriber
distiller
distiller
distiller
DB

数据分析


Storm
mysql
binlog
codis
kafka
canal
redis group
redis group
cobar
mysql
mysql
Hadoop/Spark

2. Scala在挖财的用法

语言之争

通过反观过去半个世纪以来编程语言的进化方向，我认为编程语言绝对不会按照保罗.格雷厄姆所说，向着“小而干净”的方向进化。现在的编程语言，无论时功能上还是语法上都已经不是那样单纯了，虽然也曾经有人努力尝试将这些语言变得更小更简单，但包括保罗.格雷厄姆自己所设计的Arc在内，都决不能算是成功的尝试。

在我看来，编程语言的进化动机，不是工具和语言本身的简化，而是将通过这些工具和语言所得到的结果(解决方案)更简洁地表达出来。近半个世纪以来，编程语言不断提供愈发高度的抽象化特性，也正是为了达到这个目的。因此我们可以很自然地认为，这种趋势在将来也应该会继续保持。

松本行弘

选择Scala的理由和注意事项

理由：

大部分架构师都喜欢这门语言，并有实践经验

注意事项:

1) 不能单点，至少要有几个好基友都能hold住

2) 团队的意愿

3) 培养，不强迫

4) 谨慎使用Scala生态框架，不轻易改变Java主流框架

- Scala语言很好，但社区成熟度很差！

Scala的优点

更简洁、高效的表达

一些模式，Scala直接在语言层面支持：

- 1）singleton pattern → object
- 2）visitor pattern → pattern matching
- 3）facotry pattern → apply method
- 4）builder pattern → currying
- 5）dependency inject → cake pattern
- 6）immutable pattern → val
- 7）value object → case class
- ……

Scala的优点

静态类型也可以支持动态语言的特性：

鸭子类型 → pimp my library(implicit)

Scala 2.10 Dynamic Types……

函数式特性可以自定义程序的流控

C# using语句 → 借贷模式

强大的集合操作：

```
// 从一组流水中找出有转账关系(交易号相同)的
cashflowBuf.groupBy(_.transactionNo).filter(_._2.size == 2).values
// 找出相差不超过1的相邻元素
scala> val timeList = List(1,2,3,5,7,8)
scala> timeList.sliding(2,1).filter(e=>e(1)-e(0)<=1).foreach(println)
  List(1, 2)
  List(2, 3)
  List(7, 8)
```

Scala不好的地方

- 门槛
  - 函数式背景(大部分程序员不具备)
    - 例如 eta规约
  - 类型系统(带来安全性的同时也增加了复杂性)
    - 大量基于类型的模式(type class及其变种)
    - 协变、逆变、视界、路径依赖类型、type lambda等概念众多
    - Scala类型系统是图灵完备的，可实现SKI calculus

Scala不好的地方

- 灵活性在团队协作上的问题
  - 减少高阶特性
  - 克制，不要炫技
  - 约定编码风格
  - 利用编译器参数

Scala在挖财的使用方式

- 使用Scala语言，但不使用它的框架
  - (akka除外)
- 不改变前端的servlet编程模型
- 仍围绕Spring为中心

Scala在挖财的使用方式

- Scala 与 Spring, MyBatis 等主流框架
  - 无缝集成
  - @BeanProperty
  - case class 构造参数注意声明为var，并提供无参构造器
- Akka 与 Spring
  - 扩展
- Scala与Dubbo框架
  - 无缝
  - API/Interface 仍只用Java
  - Provider端对象序列化时Scala标准类库全部转为java，调用者可以不依赖Scala

Scala在挖财的使用方式

- Jdk8, Scala2.11
- REPL(要用好)
- maven或sbt，推荐用maven (人员水平参差不齐，maven已够用)
- 每个工程在开发时都要保证能以:
  mvn spring-boot:run 或 mvn tomcat7:run 方式运行起来
- 一些有用的编译参数:
  -explaintypes
  -Yno-adapted-args
  -Ywarn-dead-code
  -Ywarn-unused
  -Ywarn-unused-import

3. Cobar/Kafka/Akka等中间件产品在挖财的使用经验

Cobar

- 相对稳定，经过阿里的业务考验
- 个别可规避的小问题
- Hash sharding vs Range sharding

Kafka的使用经验

- 中等规模
  - 业务消息每天千万级
  - 日志消息每天几亿级
- 不仅仅是个消息系统
  - 严格的说也不是个消息系统
- 系统之间的万金油
  - 分布式数据管道，存储器，日志系统

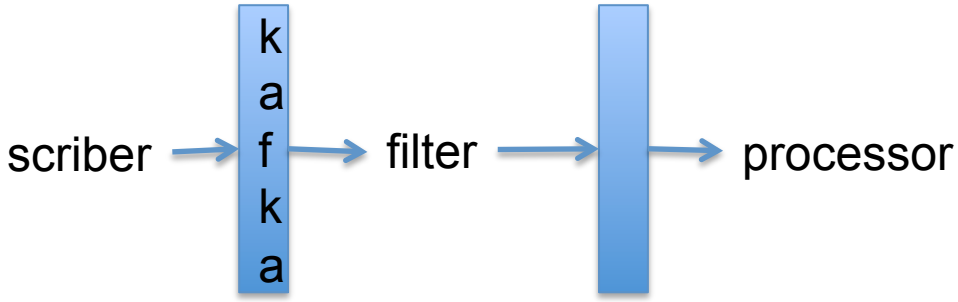
Kafka的使用经验

1) 作为分布式管道

单机版：

cat file | grep content | awk ‘{print $1}’

分布式版：

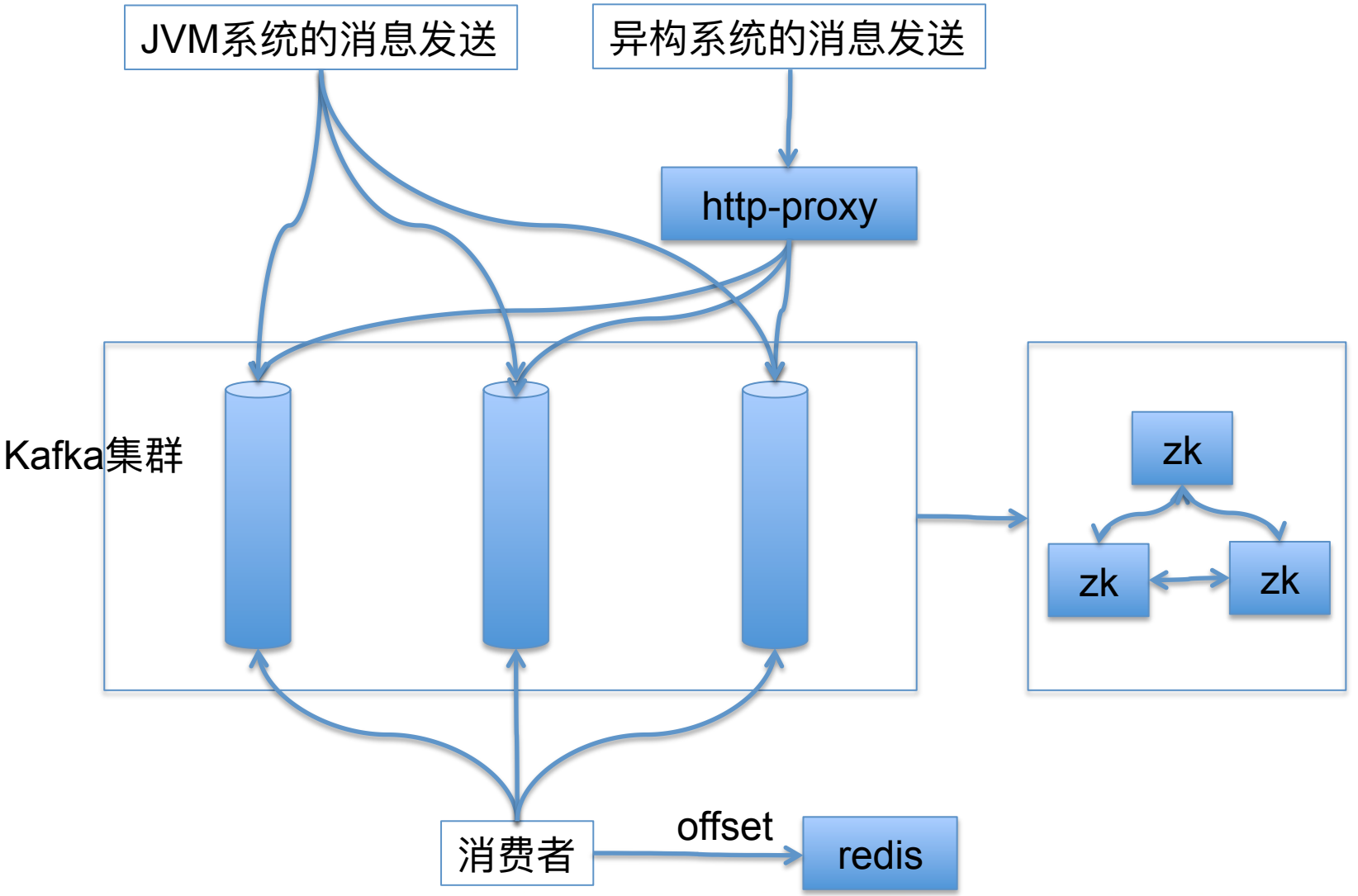


2) 作为存储器和Journal System

充分利用了磁盘和文件系统特性

《日志：每个软件工程师都应该知道的有关实时数据的统一抽象》

Kafka的使用经验


JVM系统的消息发送
异构系统的消息发送
http-proxy
Kafka集群
zk
zk
zk
消费者
offset
redis

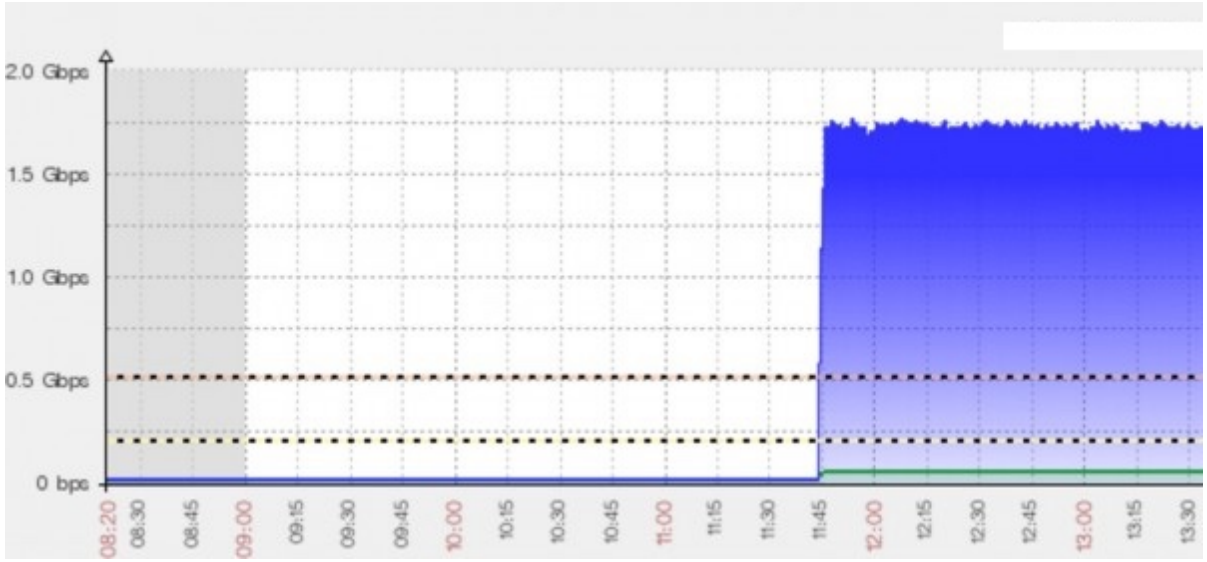
Kafka的使用经验

- 基于SimpleConsumer的封装
  - offset存于redis而非zookeeper
  - 允许一组消费端串行消费某一分区的消息

- 运维体会
  - 相对稳定，投入精力少（一个人10~20%精力）
  - 业务消息规模未达到亿级之前，可采取简单方式管理，topic只采用单分区

Kafka的使用经验

- 运维案例
  1）同步或异步都可能出现重复发送
  2）msgSize, replicaSize，fetchSize 的参数不一致问题
  3) 测试环境流量异常的问题

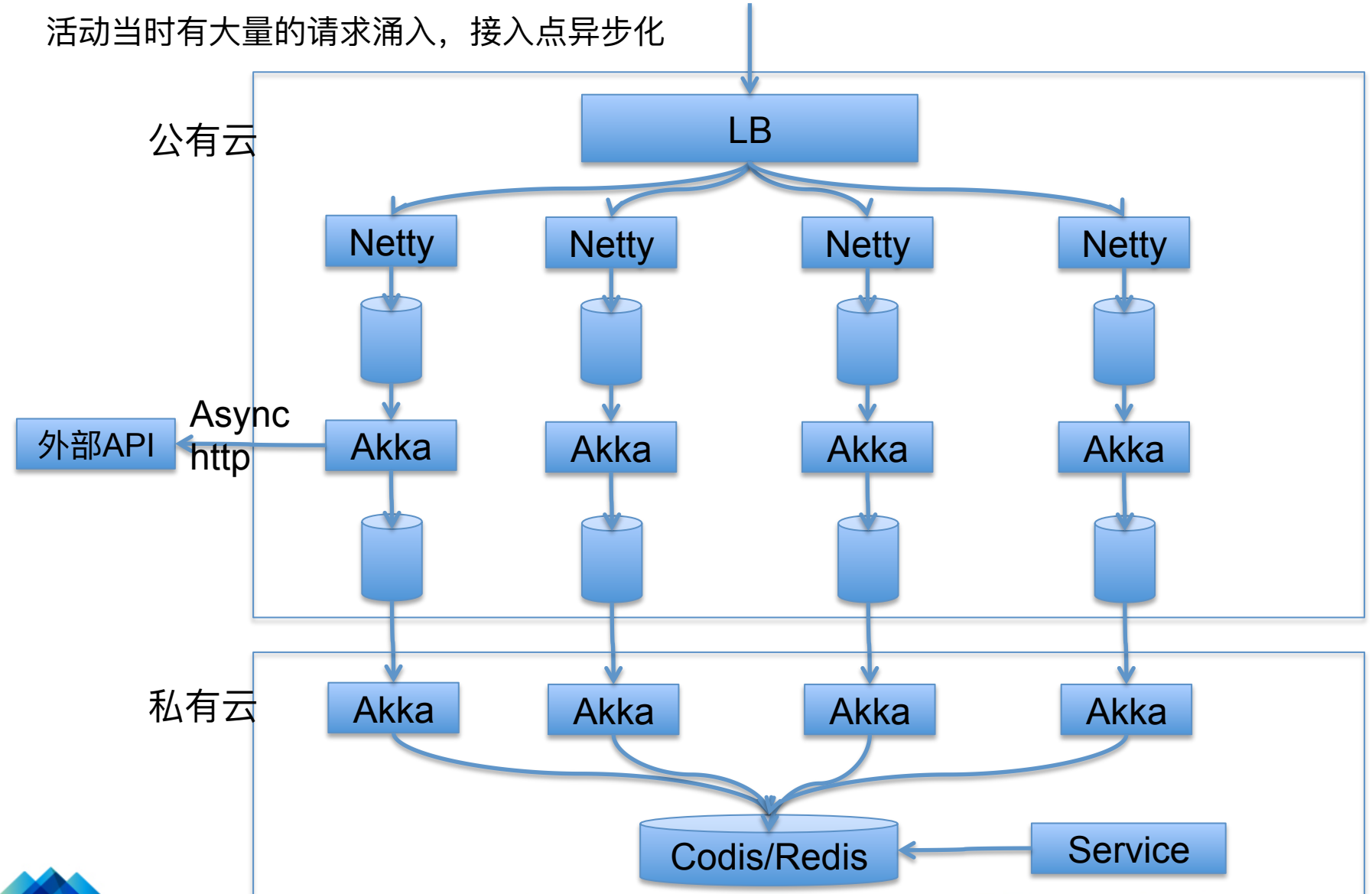


Akka的使用经验

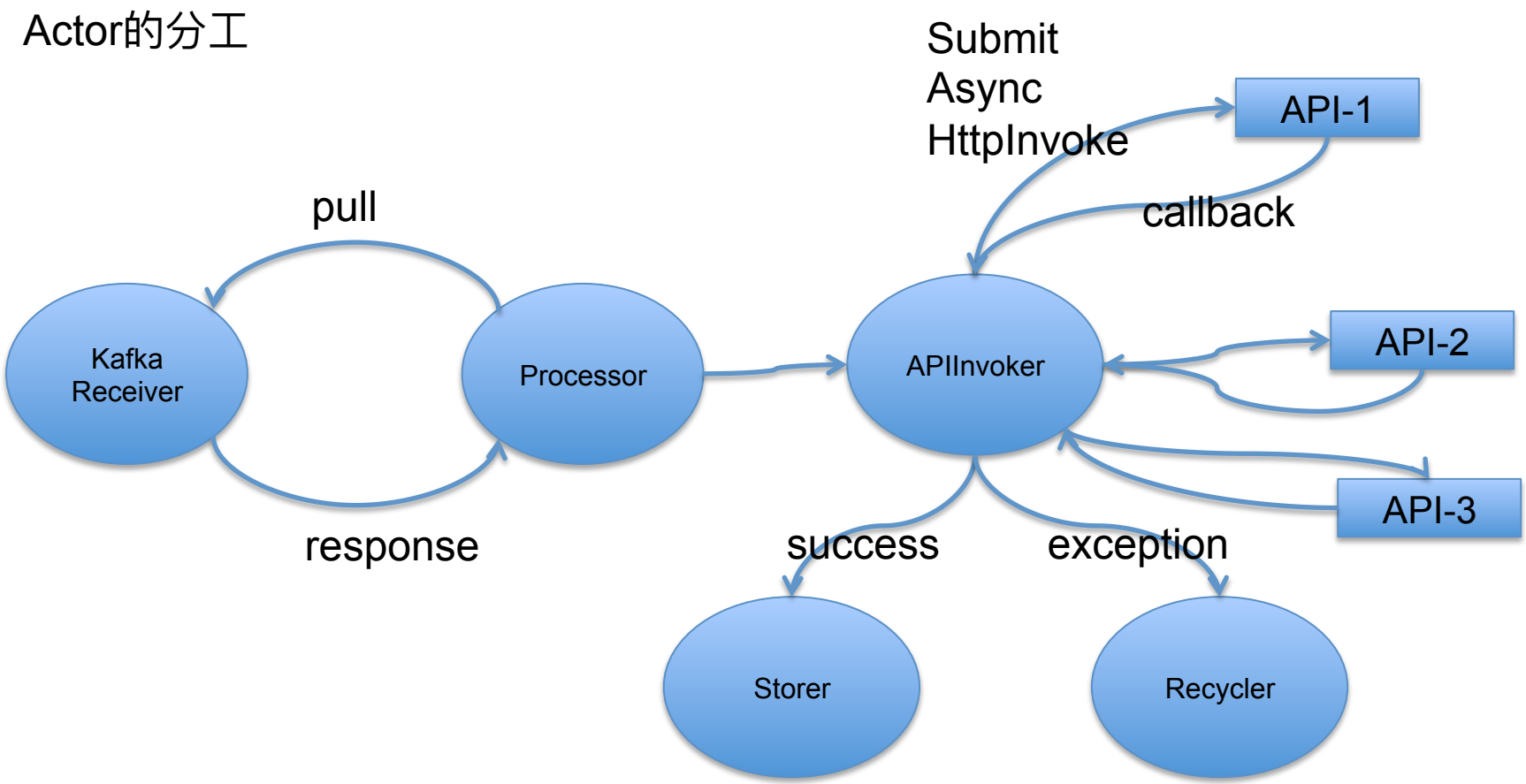
- 数据处理中大量使用
- 未使用持久化和集群等复杂特性
- 介绍一个简单的场景

基于混合云的引流

活动当时有大量的请求涌入，接入点异步化



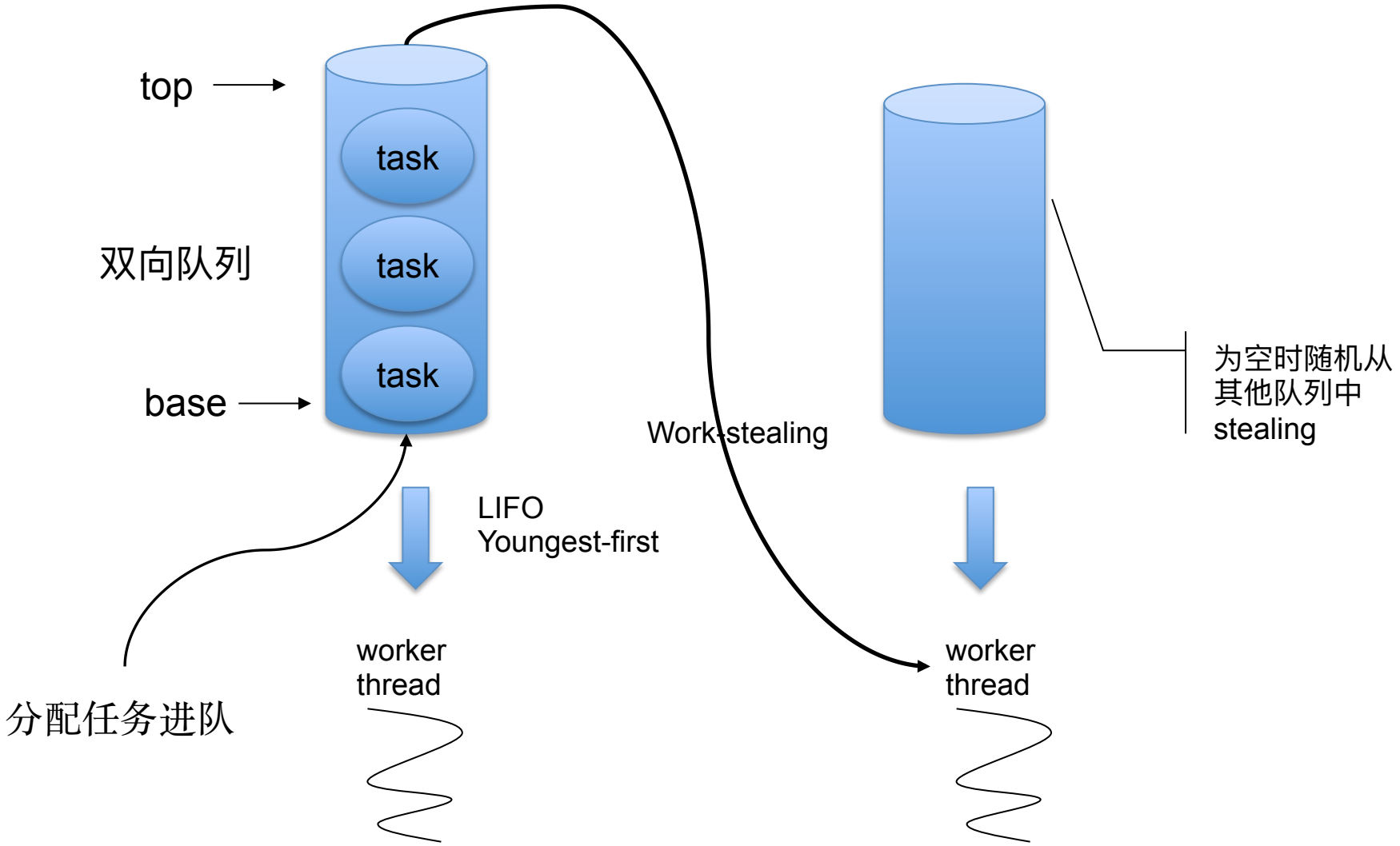
基于混合云的引流


Actor的分工
Submit
Async
HttpInvoke
API-1
callback
pull
Kafka
Receiver
Processor
APIInvoker
API-2
API-3
response
success
exception
Storer
Recycler

Akka的使用经验

- 无锁的世界
- 天然的扩展性
- 底层还是基于线程池的实现
- ForkJoin 效率很高，容易把CPU跑满

work-stealing （工作窃取）的实现



Akka的使用经验

- 与Kafka很搭
  - 分布式的管道与命令
- Akka 与 Spring 的整合
  - 对象/Actor 创建的问题
  - Akka Extension

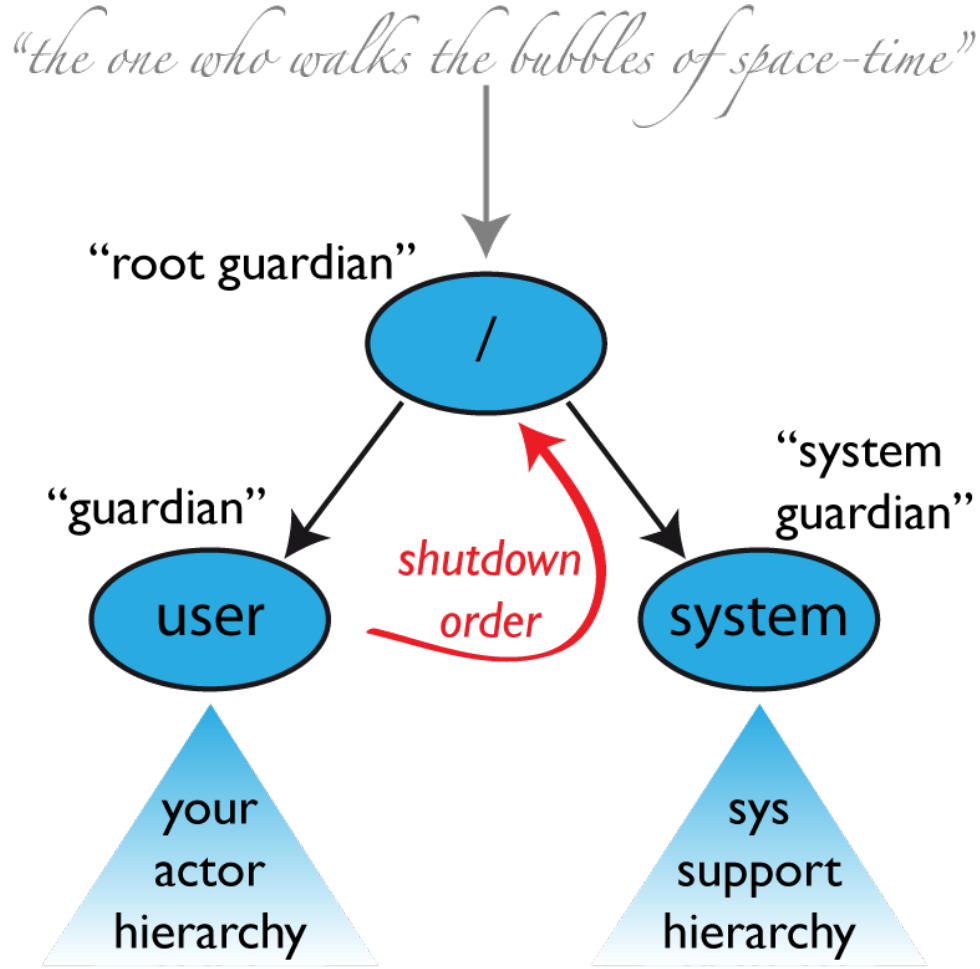
Akka的使用经验

- 实践小结
  - Actor职责尽可能单一
  - 避免阻塞
  - Supervisor和错误处理
  - Push vs Pull
  - WaterMark
  - 邮箱监控

Akka常用模式

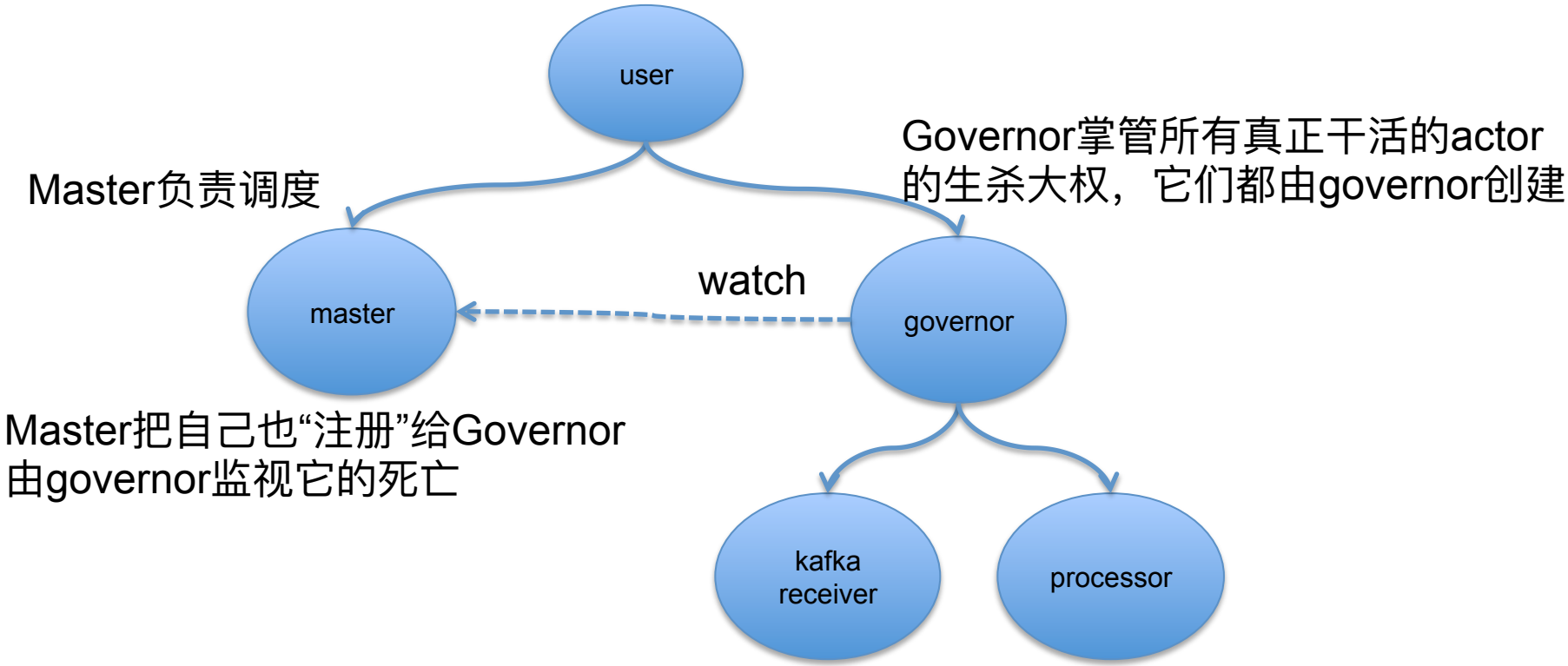
- 替身模式(cameo pattern)
  - Pattern.ask 就是一个例子
- Extra pattern
- Circuit Breaker
- WaterMark

优雅(顺序)关闭的问题


"the one who walks the bubbles of space-time"
"root guardian"
/
"guardian"
"system guardian"
shutdown order
user
system
your actor hierarchy
sys support hierarchy

优雅(顺序)关闭的问题

Governor：Terminator模式的变种



优雅(顺序)关闭的问题

提供一个 Governor的模板，实现顺序停止子Actor

```
abstract class ContextManager(signal: Semaphore) extends Actor {

 // 顺序的停止所有子actor
  protected def stopAll(kids: List[OrderedActorRef]): Future[Any] = {
    kids match {
      case first :: Nil =>
        gracefulStop(first.actor, stopTimeout).flatMap { _ => Future { AllDead } }
      case first :: rest =>
        gracefulStop(first.actor, stopTimeout).flatMap { _ => stopAll(rest) }
      case Nil =>
        Future { AllDead }
    }
  }
 ...
```

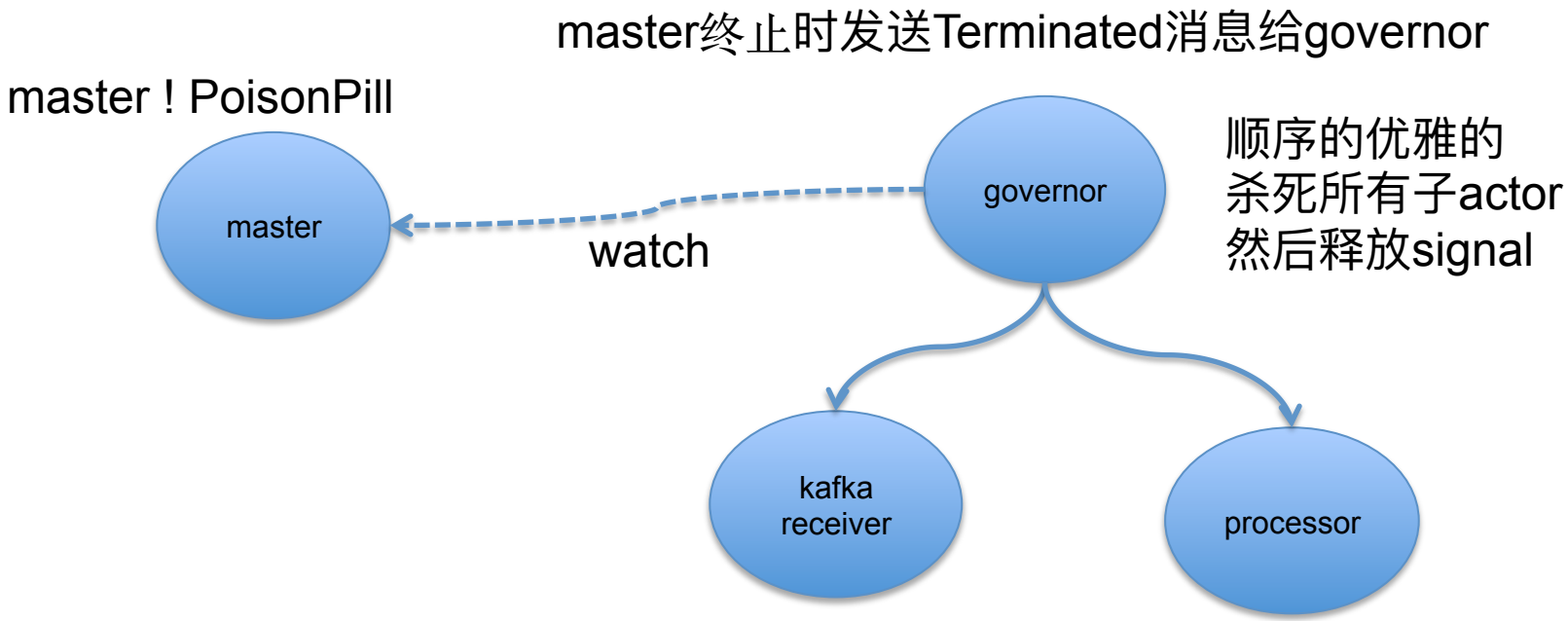
优雅(顺序)关闭的问题

实现 Governor 时，指定每个子Actor的关闭顺序，
比如下面在shutdown时先停止receiver再停止processor最后停止storer

```
class Governor(signal: Semaphore) extends ContextManager(signal) {
  override def createActors() {
    val ext = SpringExtension(context.system)
    // 1) receiver
    val receiver = context.actorOf(props(ext, "serialConsumer"), "kafkaReceiver”)
    setOrderNo(receiver, 0)
    // 2) processor
    val processor = context.actorOf(props(ext, "processor").withRouter(getProcessorRouter), "processor”)
    setOrderNo(processor, 1)
    // 3) storer
    val storer = context.actorOf(props(ext, "storer").withRouter(getStorerRouter), "storer”)
    setOrderNo(storer, 2)
  }
....
}
```

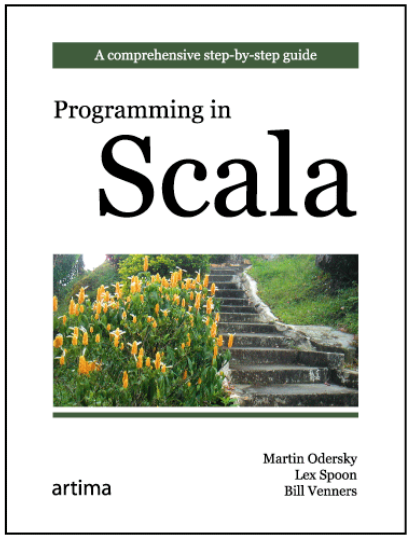
在系统关闭的逻辑里(shutdownhook)，master发给自己毒药丸，并等待 signal，signal满足后整个akka系统退出

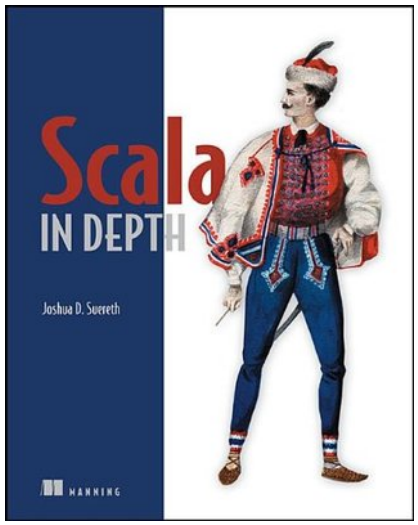
优雅(顺序)关闭的问题

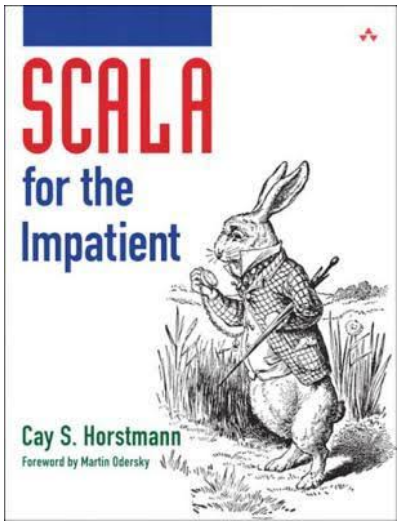

master终止时发送Terminated消息给governor
master ! PoisonPill
master
watch
governor
顺序的优雅的
杀死所有子actor
然后释放signal
kafka
receiver
processor

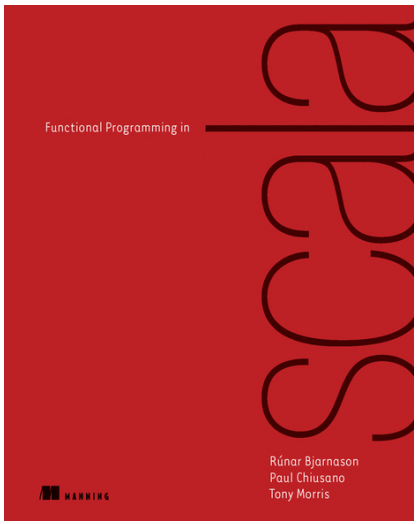
推荐书籍和相关阅读

都有中文版









《Akka in Production: our story》 by Evan Chan
《Gearpump：基于Akka的新一代流处理引擎》by 钟翔

Thanks!