服务框架实践与探索

阿里巴巴(B2B) 技术部
钱霄（@shawnqianx）
2011/10/23

Alibaba.com
Global trade starts here.™

Overview

- 承载每天10亿次的调用
- 管理超过1000个的服务
- 部署在阿里巴巴整个站点

提纲

- 应用开发的挑战
- 服务框架的演进
- 一些总结分享
- 问答交流

应用开发技术的变迁

- Alibaba B2B的Web应用
  - 1999/2000年 使用Perl CGI开发
  - 2001年开始 改用Java技术
    - 2001年Servlet/JSP开发
    - 2002年Java EE技术
    - 2003年 基于Turbine MVC框架开发
    - 2004年 使用轻量级容器
    - 2005年 自制的WebX框架成为应用开发的首选

应用结构的变化

- 发展初期：规模小，JEE技术管用
- 高速成长：膨胀，巨无霸应用开始出现
- 寻求变革：拆 分应用，独立服务
- 持续优化：聚 合服务，管控治理

挑战

- 业务不断发展，应用规模日趋庞大
  - 巨型应用的开发维护成本高，部署效率降低
  - 应用数量膨胀，数据库连接数变高
- 访问量逐年攀升，服务器数不断增加
  - 数据连接增加，数据库压力增大
  - 网络流量增加，负载均衡设备压力增大
- 对性能，可靠性的要求越来越高

对策

- 拆分
  - 对巨型系统进行梳理，垂直拆分成多个独立的Web系统。
- 剥离
  - 抽取共用的服务，提供远程调用接口，与应用共生
- 独立
  - 甄别核心的服务，独立搭建集群，提供专门服务。
- 均衡
  - 减少专业负载均衡设备使用，应用自行支持分布式调用/调度。

通讯

- 进程内 → 进程间
- 节点内 → 节点间
- RPC 是一切的基础

远程调用的变化

- EJB@Alibaba B2B的年代
  - 享受容器级的db事务连接池等服务，及透明的分布式调用
- RPC@Alibaba B2B
  - RMI/Hessian
  - XML-RPC/WS
- 定制的框架
  - Dubbo

重新造轮子？

- 需要吗？
  - 不仅仅是RPC
    - LB/FailOver/Routing/QoS等功能，是达成治理所必需的，但一般的开源方案少有提供。
  - 稳定性/兼容性考虑
    - 开源方案并不完美，用好她们，付出的代价也不低。
  - 集团作战需要规范
    - 大量的应用并存，大规模的开发团队，需要统一的规范指引。

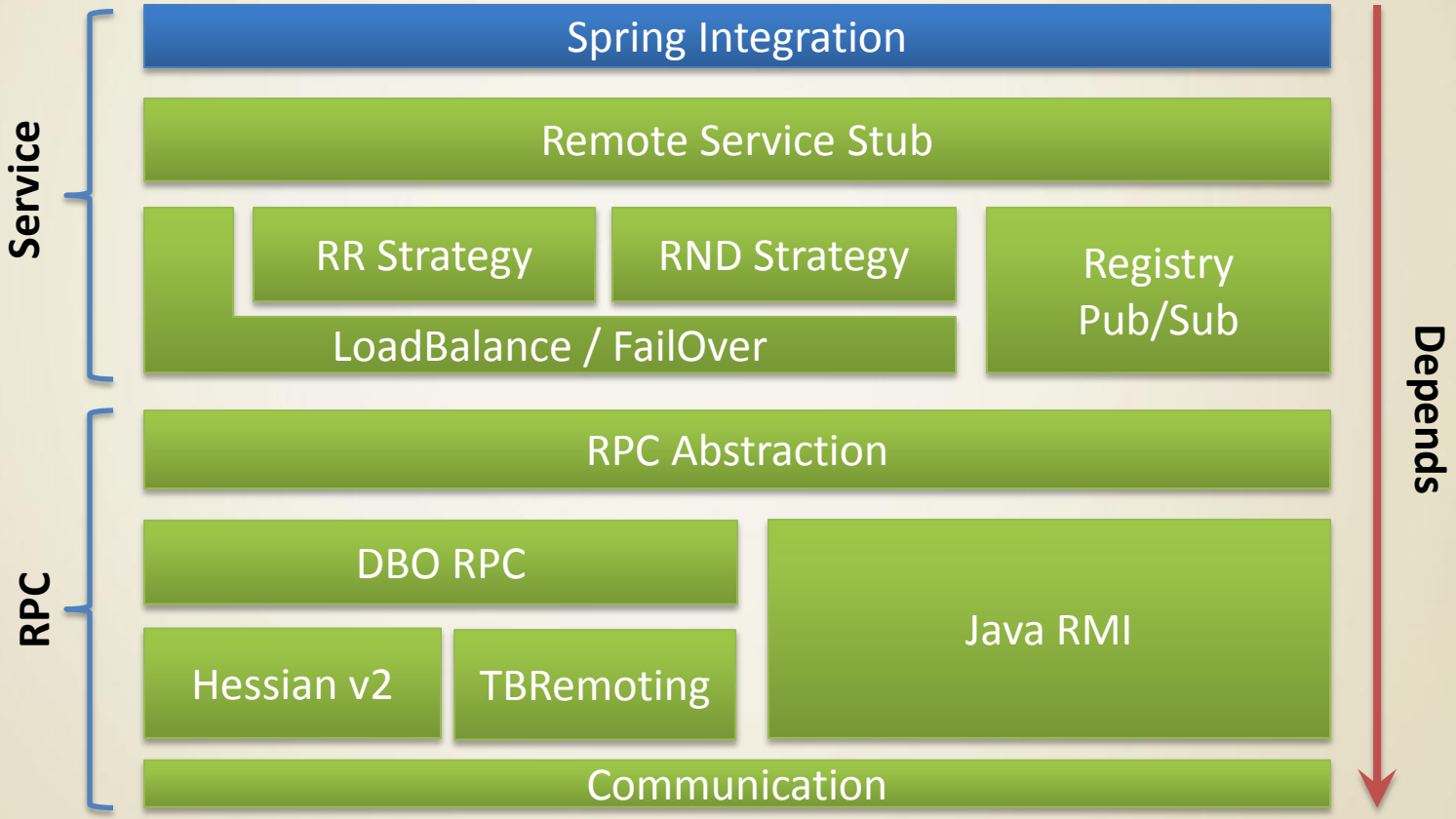
初始目标

- 零入侵
- 高性能
- 高可靠/适应高并发的环境
- 模块化设计
- 从底层支持服务化

实践

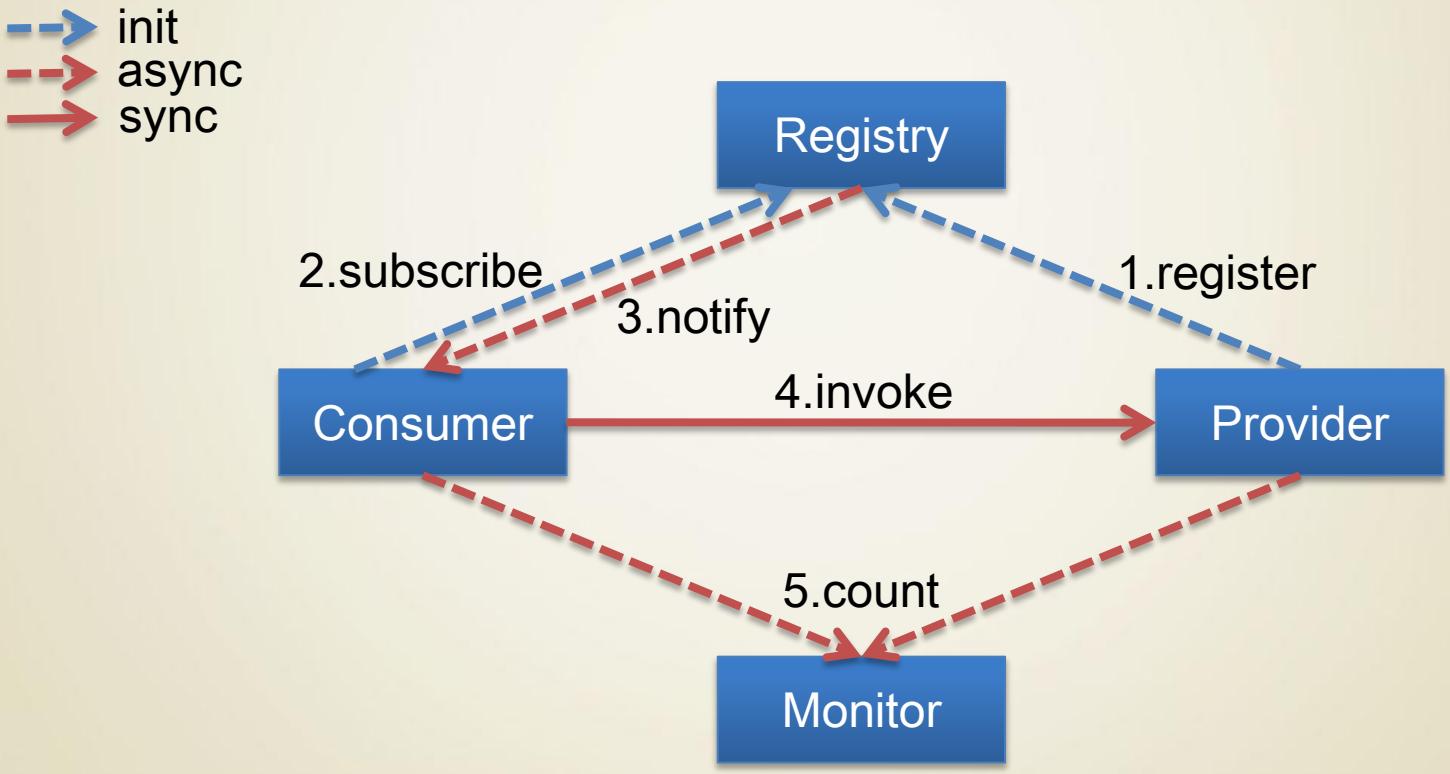
- **最初的尝试 - Dubbo 0.9**

重点1 – 核心功能抽象



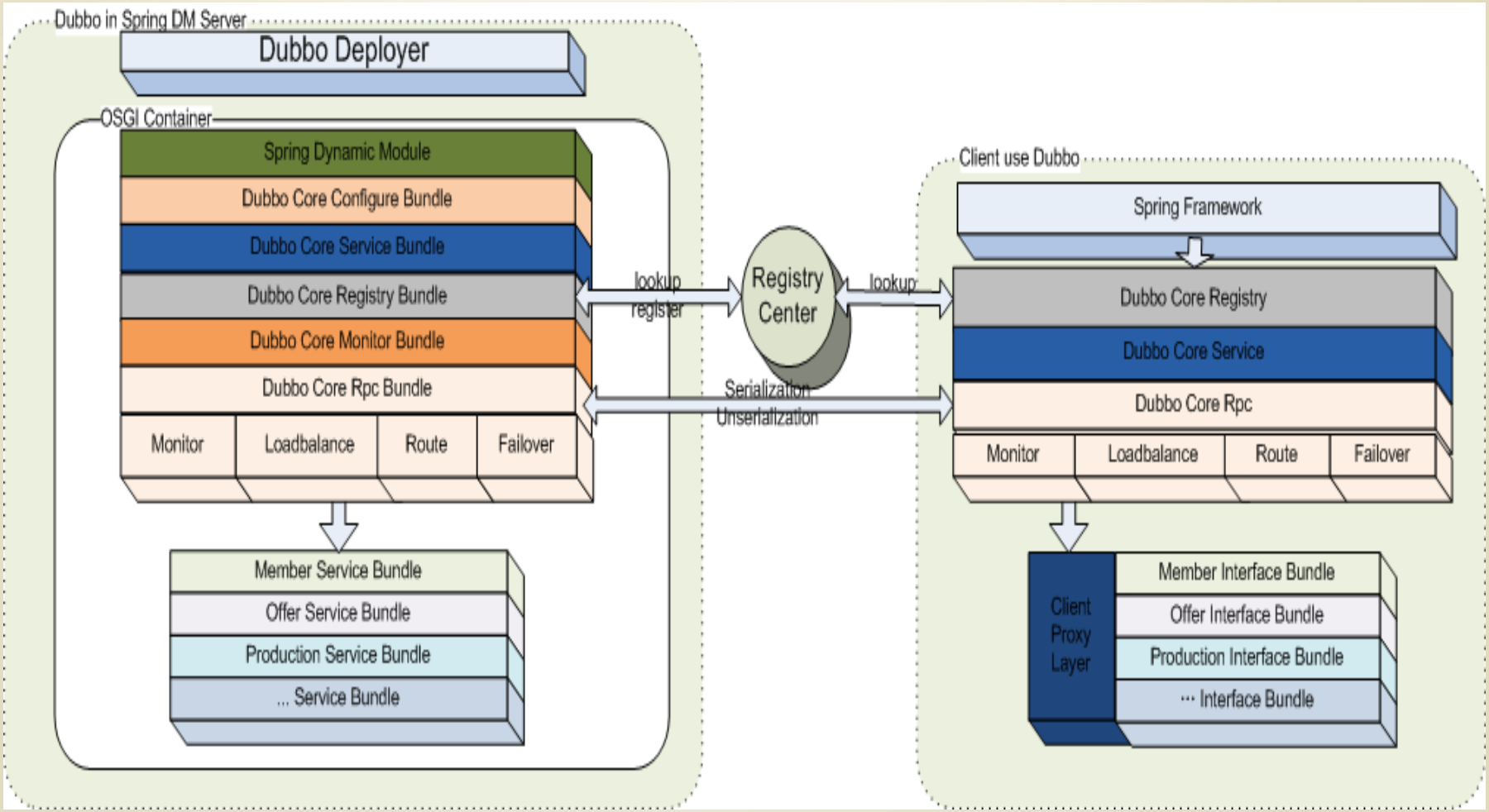
示意图

重点2 -软负载


init
async
sync
Registry
2.subscribe
3.notify
1.register
Consumer
4.invoke
Provider
5.count
Monitor

示意图

重点3 – OSGi化？


Dubbo in Spring DM Server
Dubbo Deployer
OSGI Container
Spring Dynamic Module
Dubbo Core Configure Bundle
Dubbo Core Service Bundle
Dubbo Core Registry Bundle
Dubbo Core Monitor Bundle
Dubbo Core Rpc Bundle
Monitor
Loadbalance
Route
Failover
Member Service Bundle
Offer Service Bundle
Production Service Bundle
... Service Bundle
lookup
register
Registry Center
lookup
Serialization
Unserialization
Client use Dubbo
Spring Framework
Dubbo Core Registry
Dubbo Core Service
Dubbo Core Rpc
Monitor
Loadbalance
Route
Failover
Client Proxy Layer
Member Interface Bundle
Offer Interface Bundle
Production Interface Bundle
··· Interface Bundle

取舍

- Dubbo 0.9 的选择
  - Spring Bean XML配置方式来暴露/注册服务
  - 用内部的TB-Remoting作为通讯框架
  - 用Hessian v2作为首选的序列化方式
  - 用Spring-DM/OSGi作为模块化的基础
  - 简易的服务注册中心，支持订阅推送
- 放弃
  - 多协议/多通讯框架/异步调用...

三个月后...

逐渐成长 – 1.0

- Dubbo1.0 版本
  - 放弃对Spring-DM/OSGi的支持
  - 增加独立的服务管理中心，提供初步的服务治理能力。
  - 调用数据的监控与展示

关于OSGi

Spring-DM Server的一些不适应：

1.遗留应用的迁移，需要付出很高代价
2.为了处理OSGi/non-OSGi不同的情况，框架代码变复杂。
3.针对ClassLoading的问题的特殊处理，非常不优雅。
4.使用DMServer后，被框架Bundle与业务Bundle的互相依赖及启动顺序问题所困扰，未能妥善解决。
5.构建及调试不够完善。

幸亏...

- Spring-DM隔离了OSGi的API
- 设计初期有预留伏笔，框架模块没有完全切换到OSGi Bundle风格

- 3天时间脱离DM Server，使用Spring完成Bootstrap.

野蛮生长...

- Dubbo 1.0迅速推广，覆盖了大部分的关键应用
- 成为B2B内部服务调用的首选实现
- 遭遇第一次大规模故障
- 支持热线打爆

新的要求...

- 被要求支持更多的使用场景
  - 支持专用服务协议的调用(memcached etc...)
  - 多种模式的异步调用
- 要求完善的监控
  - 服务状态/性能/调用规模等多方面多维度的统计及分析...
- 治理功能
  - 服务分组
  - 流量分离
  - ...

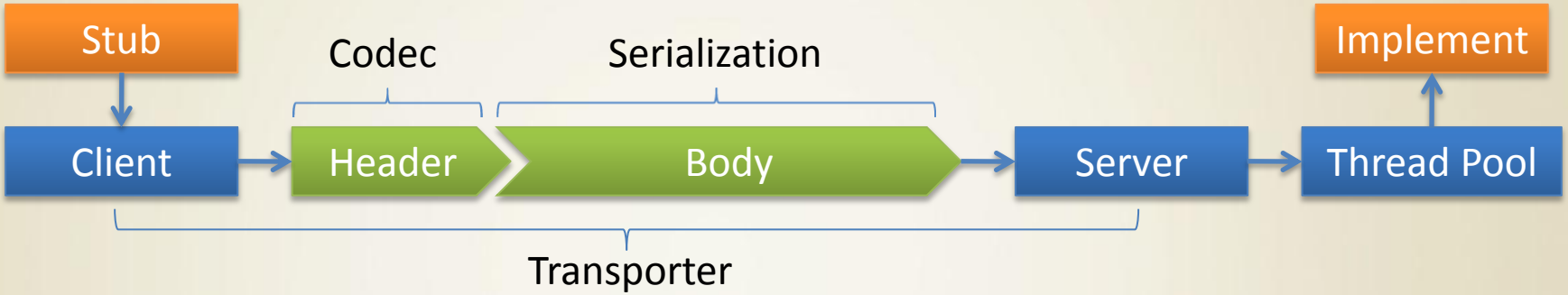
新的挑战…

- 如何抵抗功能膨胀？
  - “通用”==“难用”，如何取舍？
  - 精力有限，团队忙不过来了！
- 如何抵抗架构的衰退？
  - 新需求的加入…
  - 新人的加入…

新的旅程 – 2.x

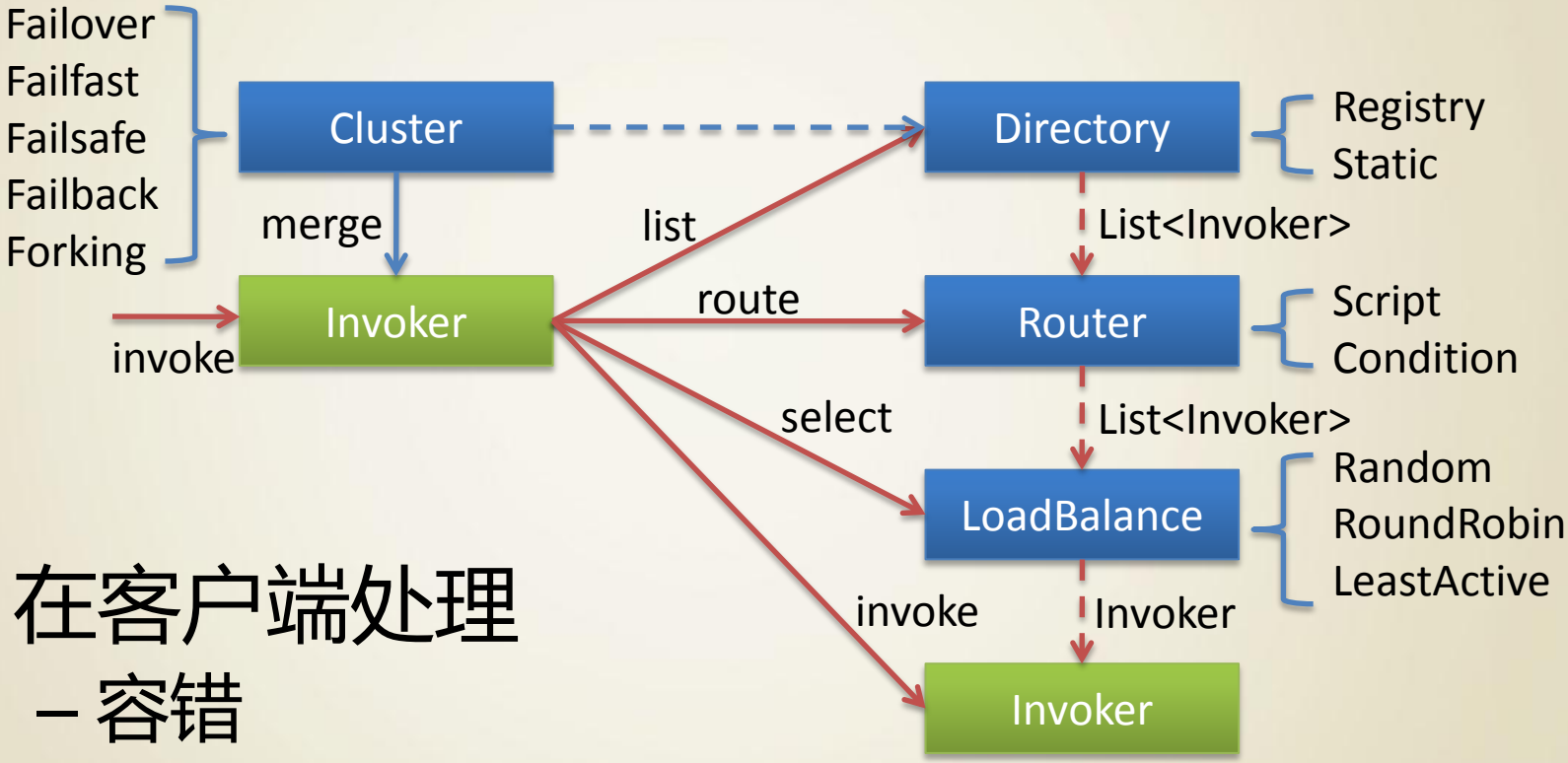
- Dubbo 2 – 重构
  - 对RPC框架的重新审视
  - 对模块化机制的重构
    - 以JDK SPI机制替代原有的Spring Bean组装
  - 扩展扩展扩展
    - 支持更多的通讯框架(Mina/Netty/Grizzly…)
    - 支持更多的序列化方式(Hessian/JSON/PB…)
    - 支持更多的远程调用协议(DBO/RMI/WS)
    - 完整的异步调用支持
  - 服务注册中心持续增强
    - 分组/路由/QoS/监控

重点1 – 协议优化



- Transporter
  - mina, netty, grizzy…
- Serialization
  - dubbo, hessian2, java, json…
- ThreadPool
  - fixed, cached

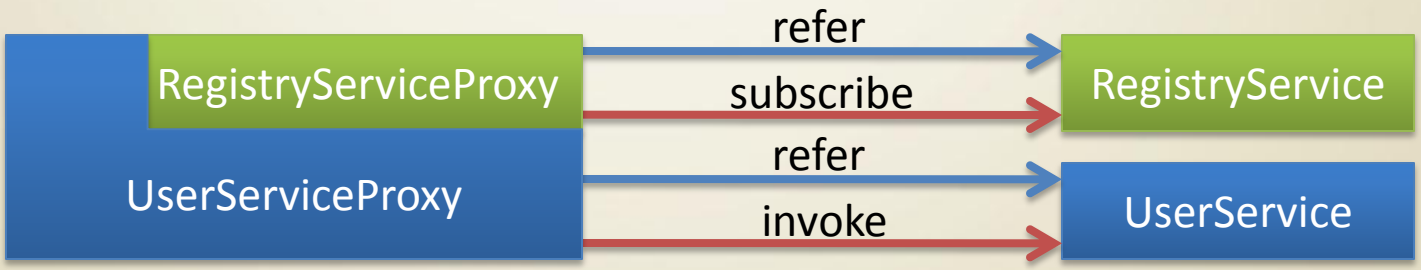
重点2 – 负载均衡增强



- 在客户端处理
  - 容错
  - 路由
  - 软负载均衡

重点3 - Dogfooding

- 注册中心和监控中心也是普通RPC服务
  - 为此，需支持回调操作：
    - 生成回调参数的反向代理：
    - subscribe(URL url, NotifyListener listener)
    - listener.notify(list)



重点4 – 更多调用方式

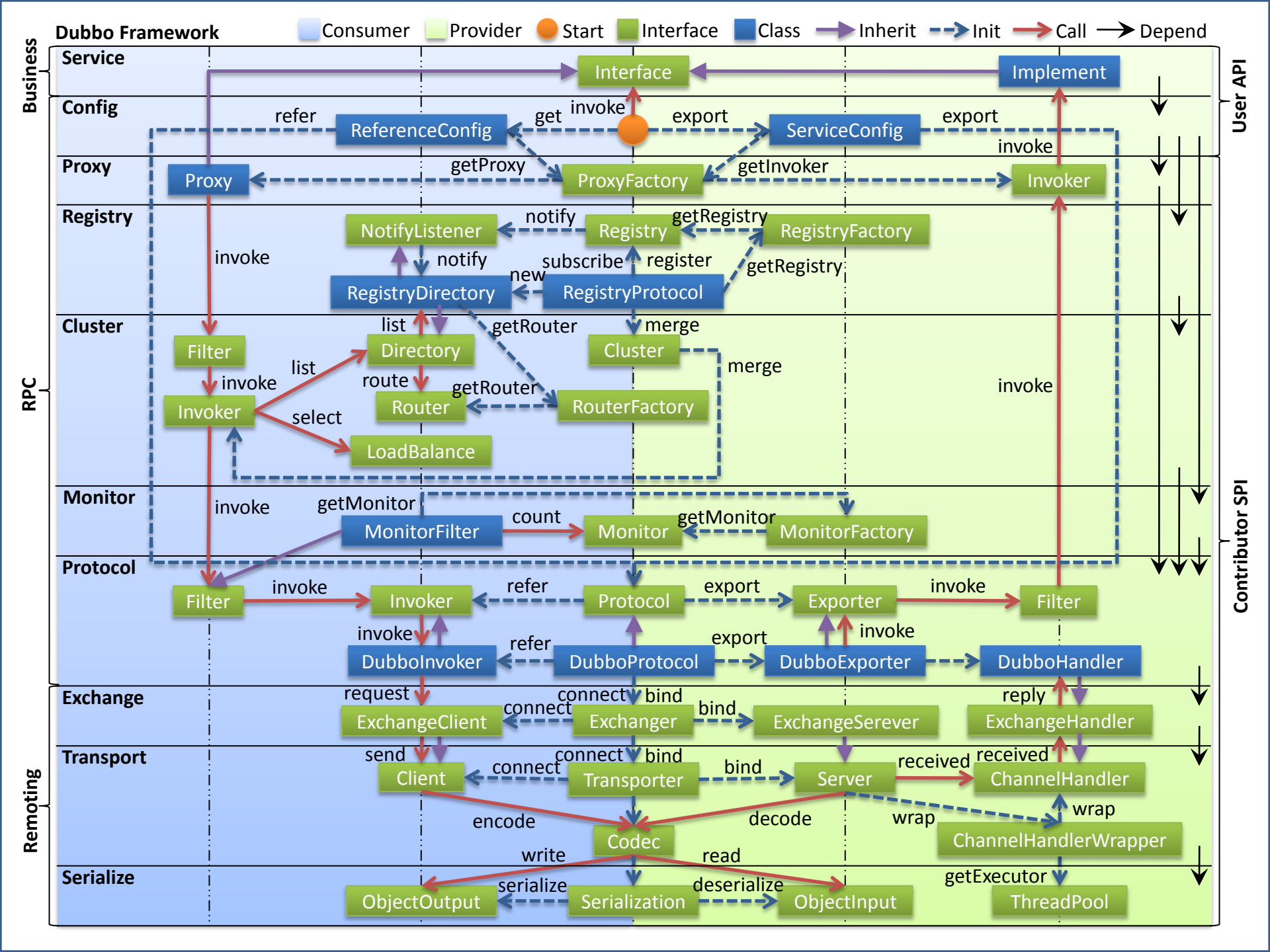
- 完整的异步调用支持
  - 基于内部的消息中间件，实现可靠的异步调用
- 并行调用(Fork/Join)
  - 利用API，应用可以同时发起多个远程请求
  - 虽然比较简单，但的确管用！

重点5 – 插件机制调整

- 简化插件机制
  - 基于JDK的SPI机制扩展
  - 不再依赖Spring
- 区分API/SPI
  - API给使用者。
  - SPI给扩展者

其他增强

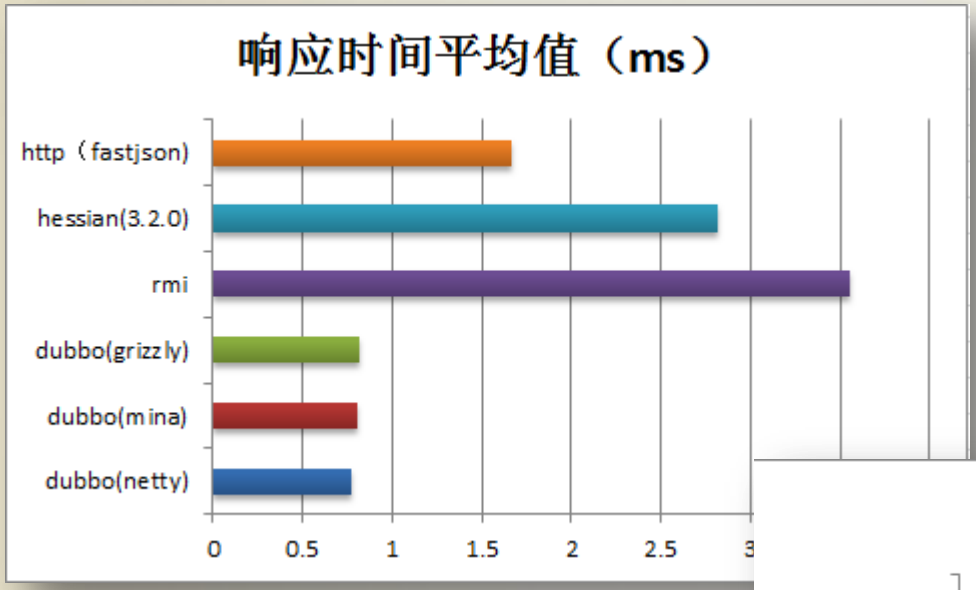
- 远程调用的本地短路
  - 允许：缓存或远端故障时，本地短路
- 调用的Cookie传递
  - 某些隐式传参的场合（鉴权等）
- 诊断功能
  - 自带远程调试诊断功能 (Diagnosing via Telnet)

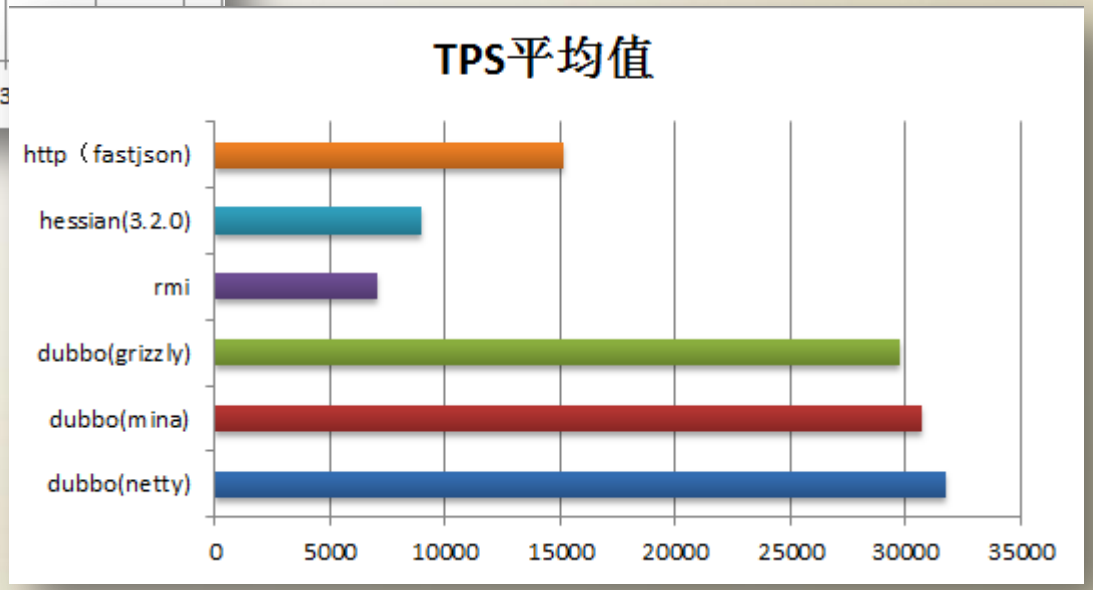

Dubbo Framework
Consumer
Provider
Start
Interface
Class
Inherit
Init
Call
Depend
Business
RPC
Remoting
User API
Contributor SPI
Service
Config
Proxy
Registry
Cluster
Monitor
Protocol
Exchange
Transport
Serialize
Interface
Implement
refer
ReferenceConfig
get
invoke
export
ServiceConfig
export
invoke
getProxy
Proxy
ProxyFactory
getInvoker
Invoker
NotifyListener
notify
Registry
getRegistry
RegistryFactory
invoke
notify
subscribe
register
getRegistry
new
RegistryDirectory
RegistryProtocol
list
getRouter
merge
Filter
Directory
Cluster
merge
invoke
list
route
invoke
Invoker
Router
getRouter
RouterFactory
select
LoadBalance
invoke
getMonitor
count
Monitor
getMonitor
MonitorFactory
MonitorFilter
invoke
refer
export
invoke
Filter
Invoker
Protocol
Exporter
Filter
invoke
export
invoke
refer
DubboInvoker
DubboProtocol
DubboExporter
DubboHandler
request
connect
bind
reply
ExchangeClient
connect
Exchanger
bind
ExchangeSerever
ExchangeHandler
send
connect
bind
received
received
Client
connect
Transporter
bind
Server
ChannelHandler
wrap
encode
decode
wrap
Codec
ChannelHandlerWrapper
write
read
getExecutor
serialize
deserialize
ObjectOutput
Serialization
ObjectInput
ThreadPool

Dubbo 2 一些数据

- 部署范围：
  - 运行在200+个产品中
  - 为1000+个服务提供支持
  - 涉及数千台服务器
- 繁忙程度：
  - 最繁忙单个应用： 4亿次/天
  - 累计：10亿次/天

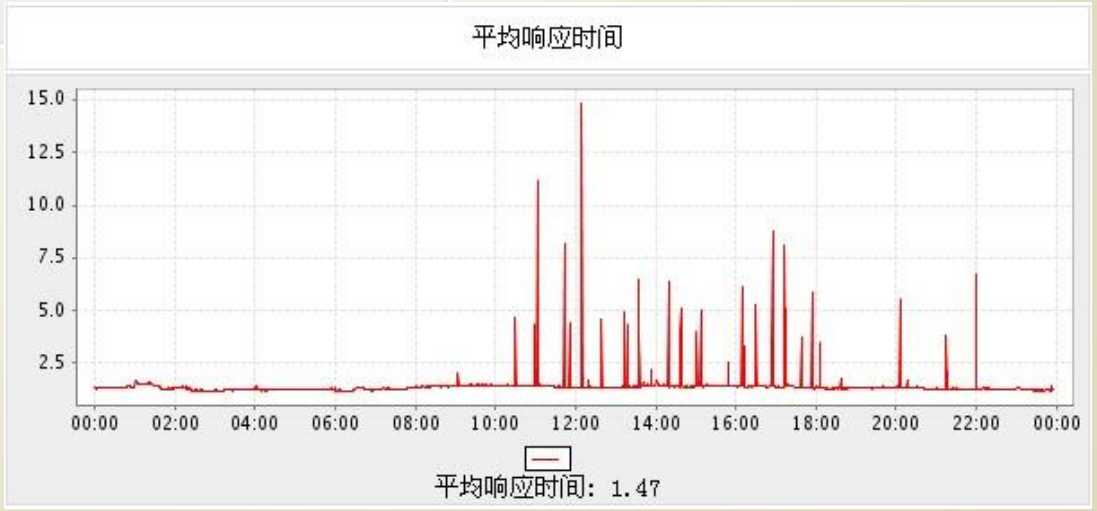
Dubbo2 性能数据





CPU : E5520 @ 2.27GHz *2
内存: 24G
网卡: 1G
OS: RedHat EL 6.1
Linux Kernel: 2.6.32-131.0.15.el6.x86_64

某服务调用情况



Credit to TB LogStat ☺

后续方向 – 我们在路上

- 服务治理
  - 服务的版本管理
  - 优雅升降级
- 资源管理
  - 服务容器及服务自动部署
  - 统一管理集群资源
- 开发阶段增强
  - IDE支持

一些总结

- 框架的入侵性
  - 支持Spring的Bean配置 - 包括业务Service Bean的暴露及框架的运行时参数配置
  - 也支持API编程方式的暴露服务，及API配置框架的运行时参数。
  - 远程服务的特性 决定了不可能完全无入侵

一些总结 – 续

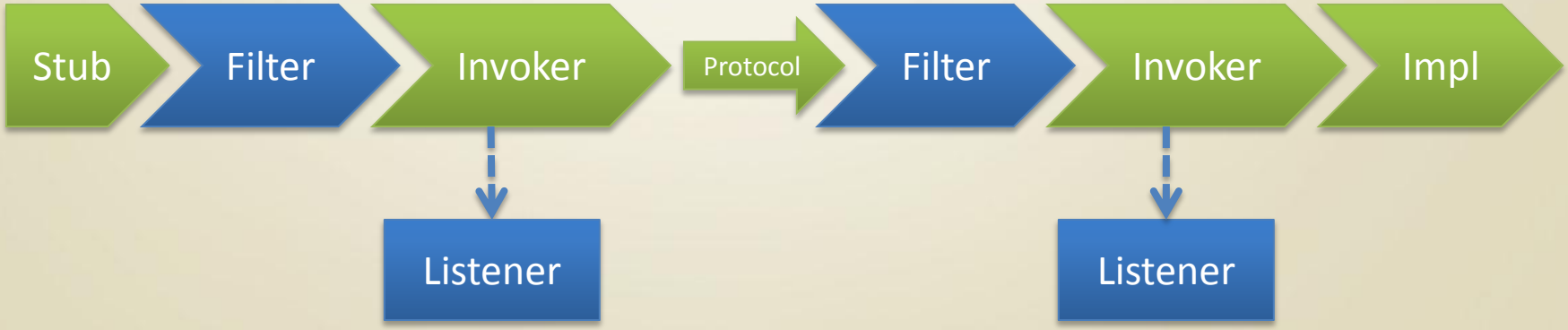
- 框架的可配置性
  - 约定优于配置
    - Convention over Configuration
  - 配置方式对等
    - XML == Java

一些总结 – 续

- 框架的扩展性
  - 微内核设计风格，框架由一个内核及一系列核心插件完成。
  - 平等对待第三方的SPI扩展。
    - 第三方扩展可以替代核心组件,实现关键的功能
  - 区分API与SPI
    - API面向使用者，SPI面向扩展者

一些总结 – 续

- 模块间的解耦
  - 事先拦截
    - 在关键环节点允许配置类似ServletFilter的强类型的拦截器。
  - 事后通知
    - 允许注册消息监听器，框架在执行关键操作后，回调用户代码。



一些总结 – 三方库

- 三方库的采纳
  - 严格控制三方库依赖的规模
    - 传递依赖会对应用程序的依赖管理造成很大的负担
    - 核心代码尽可能少的依赖三方库。
    - 必须考虑三方库不同版本的冲突问题。
  - 隔离三方库的不稳定/不兼容
    - 内联

一些总结 – 性能优化

- 性能及优化
  - 环境优化
    - 升级Linux内核开启 ReceivePacketSteering，测试表明包处理吞吐量提升明显
    - JVM GC tuning

IRQ Balancing

```
#cat /proc/interrupts
           CPU0       CPU1       CPU2       CPU3       CPU4       CPU5
256: 1644210921          0          0          0          0          0  Dynamic-irq  timer0
257:    9340150          0          0          0          0          0  Dynamic-irq  resched0
258:         35          0          0          0          0          0  Dynamic-irq  callfunc0
259:        598          0          0          0          0          0  Dynamic-irq  xenbus
260:          0    4320359          0          0          0          0  Dynamic-irq  resched1
261:          0         90          0          0          0          0  Dynamic-irq  callfunc1
262:          0 1194128577          0          0          0          0  Dynamic-irq  timer1
263:          0          0    4213713          0          0          0  Dynamic-irq  resched2
264:          0          0         94          0          0          0  Dynamic-irq  callfunc2
265:          0          0  875607694          0          0          0  Dynamic-irq  timer2
266:          0          0          0    4430842          0          0  Dynamic-irq  resched3
267:          0          0          0         90          0          0  Dynamic-irq  callfunc3
268:          0          0          0 1026707982          0          0  Dynamic-irq  timer3
269:          0          0          0          0    3961141          0  Dynamic-irq  resched4
270:          0          0          0          0         94          0  Dynamic-irq  callfunc4
271:          0          0          0          0 1013003895          0  Dynamic-irq  timer4
272:          0          0          0          0          0    3884153  Dynamic-irq  resched5
273:          0          0          0          0          0         94  Dynamic-irq  callfunc5
274:          0          0          0          0          0  908640884  Dynamic-irq  timer5
275:          0          0          0          0          0          0    4507576  Dynamic-irq  resched6
276:          0          0          0          0          0          0         93  Dynamic-irq  callfunc6
277:          0          0          0          0          0          0  960676489  Dynamic-irq  timer6
278:          0          0          0          0          0          0          0    4049357  Dynamic-irq  resched7
279:          0          0          0          0          0          0          0         68  Dynamic-irq  callfunc7
280:          0          0          0          0          0          0          0 1000477658  Dynamic-irq  timer7
281:          2          0          0          0          0          0          0          0  Dynamic-irq  xencons
282:       2912          0         50          0        105          0          0          0  Dynamic-irq  xenfb
283:          0          0          0          0          0          0          0          0  Dynamic-irq  xenkbd
284:   19062748      13108        284          0          0        385          0          0  Dynamic-irq  blkif
285:         64          0          0          0          0          0          0          0  Dynamic-irq  blkif
286:   65852646          0          0          0          0          0          0          0  Dynamic-irq  eth0
NMI:          0          0          0          0          0          0          0          0
LOC:          0          0          0          0          0          0          0          0
ERR:          0
MIS:          0
```

```
#cat /proc/irq/286/smp_affinity
00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000001
```

一些总结 – 优化 续

- 性能及优化
  - 代码优化
    - 锁粒度细化
    - 善用无锁数据结构(Lock-free data structure)
    - 使用对SMP优化的同步机制(Java Concurrent Lib)

一些总结 – 优化 续

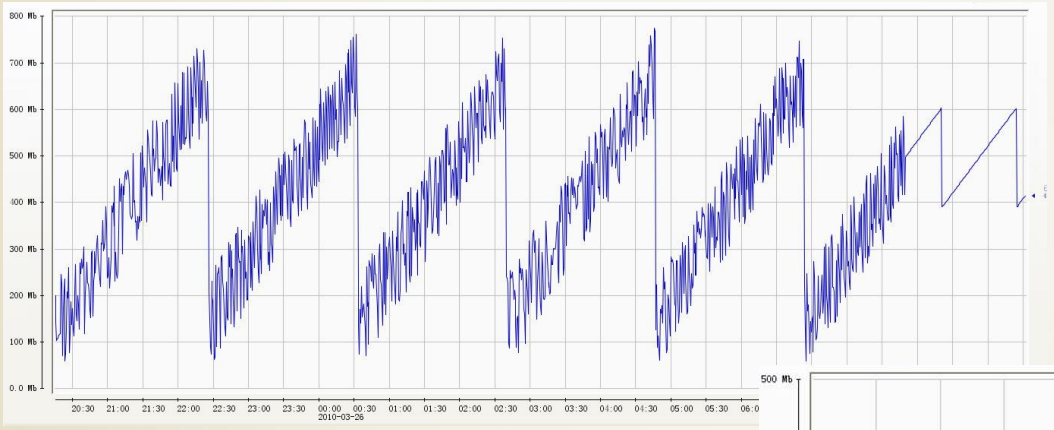
- 考量性价比，避免过度优化
  - 莫钻牛角尖，充分够用即可。
    - 拿90分容易，拿99分难
    - 比应用足够快，就够了。
  - 优化，通常意味着牺牲未来的可能性。

一些总结 – NIO框架

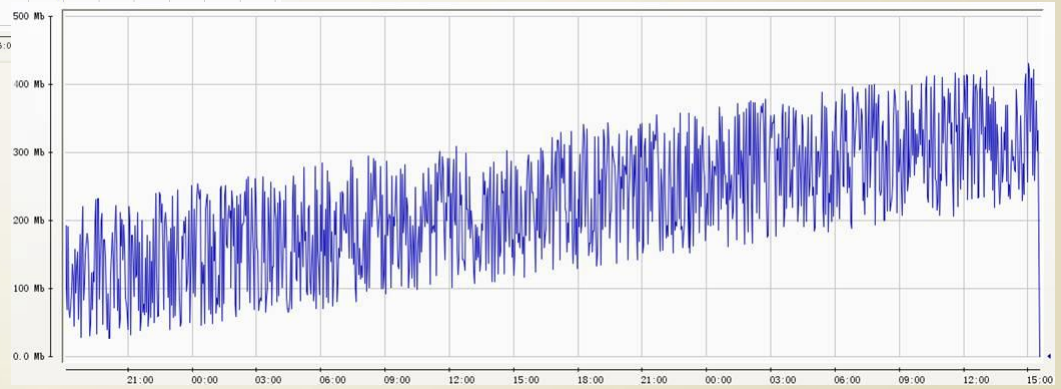
- 支持多个NIO框架是挑战
  - Mina/Netty的差异只会在细节中体现
    - 内存使用表现上的差异
    - 适配Codec和Serialization的行为差异
    - 线程处理上的差异：Netty一次请求派发两个事件，导致需两倍线程处理

Minas vs. Netty Memory

Mina



Netty




```
num     #instances         #bytes  class name
----------------------------------------------
   1:      18985140      607524480  java.util.concurrent.ConcurrentLinkedQueue$Node
```

一些总结 – 线程模型

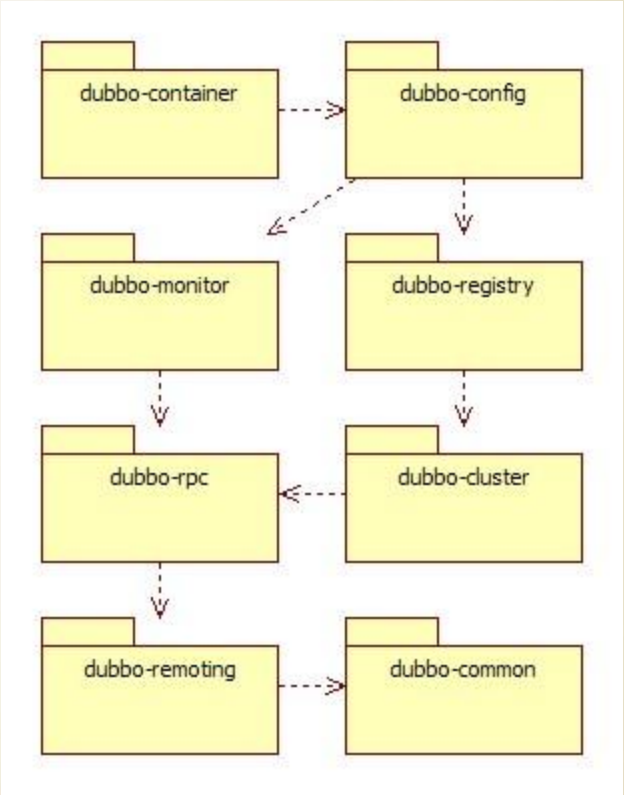
- 线程模型选择权留给应用
  - IO线程池与业务线程池的隔离
  - 固定线程数 vs 可变线程数

更多分享 尽在技术博客...

- 实现的健壮性
- 防痴呆设计
- 泛化式扩展与组合式扩展
- 常见但易忽略的编码细节
- 负载均衡扩展接口重构

真正的分享 - **开源！**

- Dubbo2框架核心以Apache v2协议开源
  - 久经考验的代码
  - 符合开源社区口味的开发流程
  - 完善的单元测试
  - 必要的文档



开源模块一览

OPEN SESAME

Edit

9 Added by kimi Lv, last edited by 梁飞 on Oct 21, 2011 (view change)

DUBBO || Download || User Guide || Developer Guide || FAQ || Release Notes || Roadmap || Issue Tracking || Community ||

Dubbo Overview

Serving 1,000+ services with 1,000,000,000+ invocations everyday, Dubbo becomes the key part of Alibaba's SOA solutio alibaba.com family:

Alibaba.com Global trade starts here.™ 阿里巴巴 1688.com AliExpress by Alibaba.com Alibado.com 阿里学院 阿里巴巴金融 Alibaba Financial 良无限

So, What is Dubbo?

Dubbo ['dʌbəʊ] is a service framework enpowers applications with service import/export capability with high performance RPC.

It's composed of three kernel parts:

- **Remoting:** a network communication framework provides sync-over-async and request-response messaging.
- **RPC:** a remote procedure call abstraction with load-balancing/failover/clustering capabilities.
- **Registry:** a service directory framework for service registration and service event publish/subscription

Dubbo can:

- Integrate different types of RPC solutions(RMI/Hessian...) with unified behavior by the abstraction layer of RPC
- Support out-of-box, plug-able load balancing and fault tolerance strategies.
- Achieve graceful service upgrade/downgrade with service registry.

One minute quick start:

Export a service

资源

- 访问 http://code.alibabatech.com/
  - 了解更多关于 Alibaba B2B 开源项目的信息
- Blog http://code.alibabatech.com/blog/
- Follow Us @dubbo

Credit & Thanks to

- The Dubbo Team
- PupaQian
- BlueDavy & the HSF Team
- ……

Q & A

End