



## Solomon【肖老师】

海量数据 高并发 亿级用户 架构师 技术VP 大厂

🎵 产品集群 *PRODUCT CLUSTER*

[查看更多](#)



麦田音乐



太合麦田



海螺音乐



大石版权



千千音乐



太合乐人



秀动



Lava熔岩音乐



亚神音乐

## 讲师介绍

### Solomon(肖老师)

- 前百度环境音乐CTO&架构师、全球海量专利数据项目负责人
- 中国机械出版社签约作家,《深入理解Dubbo工业级架构设计》图书的作者
  - 目前市面上唯一一本完整的架构设计书籍
- 11年互联网大厂经验,在互联网音乐、电商、大数据、高并发、数据挖掘、云架构、虚拟化、容器化等领域有丰富的实战经验。
- 擅长于通信协议、微服务架构、框架设计、消息队列、服务治理等领域

# 第一天

---

## 问题探索

如果单机单体架构能够支持亿级别并发量，还需要分布式架构吗？如果需要你觉得作用的范围是什么？

分析源码有什么用？为源码而源码？还是向往一个高度？

服务治理在什么场景中会被使用

服务治理之有哪些手段

## 课程范围

1. Dubbo 画像
2. dubbo市场需求与前景
3. dubbo在企业项目中的地位
4. dubbo生态圈
5. 微服务治理在企业项目中的地位
6. 微服务治理之服务限流
7. 微服务治理之服务降级
8. 微服务治理之服务容错
9. 微服务治理之路由规划

## 课程目标

- 何为服务治理？
- 服务治理的手段有哪些？
- 服务治理的实现原理如何实现？

## 课程收获

1. 阿里系面试关注的点
2. 微服务治理运作核心原理
3. 微服务治理手段与场景
4. 微服务服务降级、容错、限流工作机制等流程

## 聚焦

---

微服务之服务治理**思想**

## 传统的服务调用方式

---

- http-----使用权重
- rest api
- webservice-----xfire
- rmi
- rpc -----解决什么问题

备注：跨进程通信

# Dubbo 市场需求分析

---

- Netflix 所有组件闭源
- Netflix 被spring cloud alibaba取代
- Dubbo 已成为apacha 顶级项目
- Dubbo 开源生态逐渐完善
  - spirng-clould-alibaba-dubbo
  - spring-clould-alibaba-seata
  - spring-clould-alibaba-sentinel-zuul
  - spring-clould-alibaba-alicloud-sms
  - spring-clould-alibaba-nacos【替代zookeeper】
  - spring-clould-alibaba-sentinel
  - spring-clould-stream-binder-rocketmq

## Dubbo RPC在企业中的地位




---

- 阿里巴巴
  - dubbo---范围广
  - **hsf**
  - **sofa-rpc**--阿里金服必备
- 当当网
  - **dubbox**
- 滴滴出行
- 去哪儿
- 中国电信
- 中国工商银行
- 海尔
- 二次开发（开源自研）
- .....
- **EDAS AWS**（云架构师,）

<https://github.com/apache/dubbo/issues/1012>

## Dubbo Github 基本信息【规模之大】

---

 Used by ▾	642	 Watch ▾	3,211	 Star	26,837	 Fork	17,747
---	-----	---	-------	--	--------	--	--------



# 今天 DUBBO 的扩展

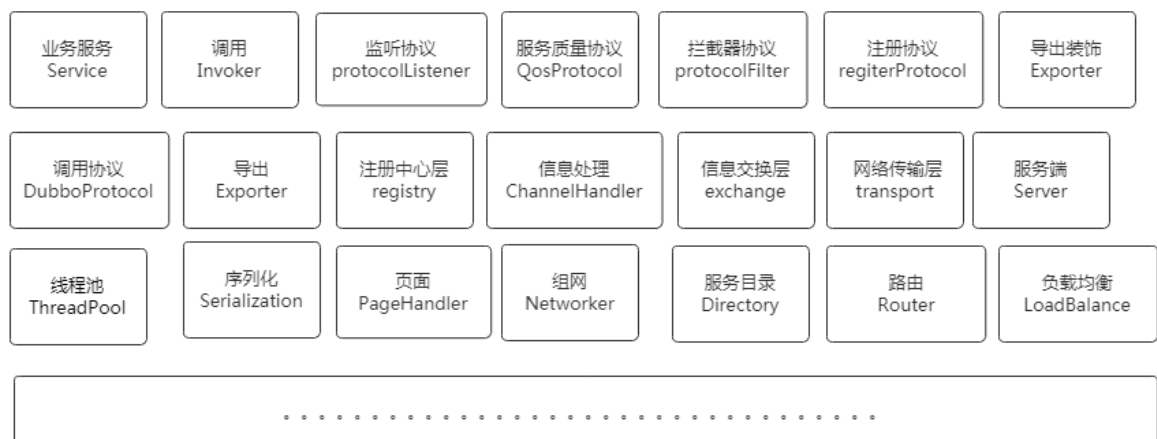


## DUBBO 3.0 路线图

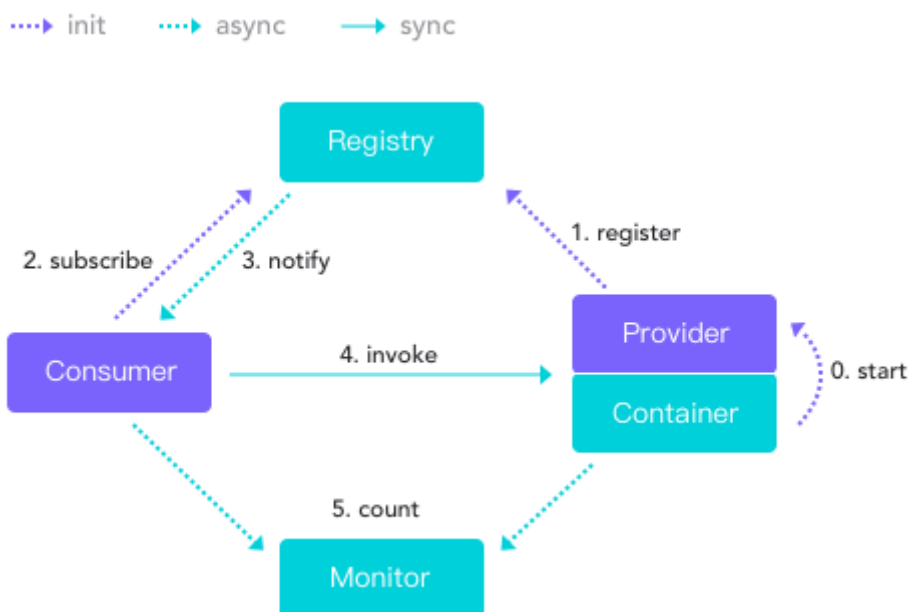
定义协议



## Dubbo 画像

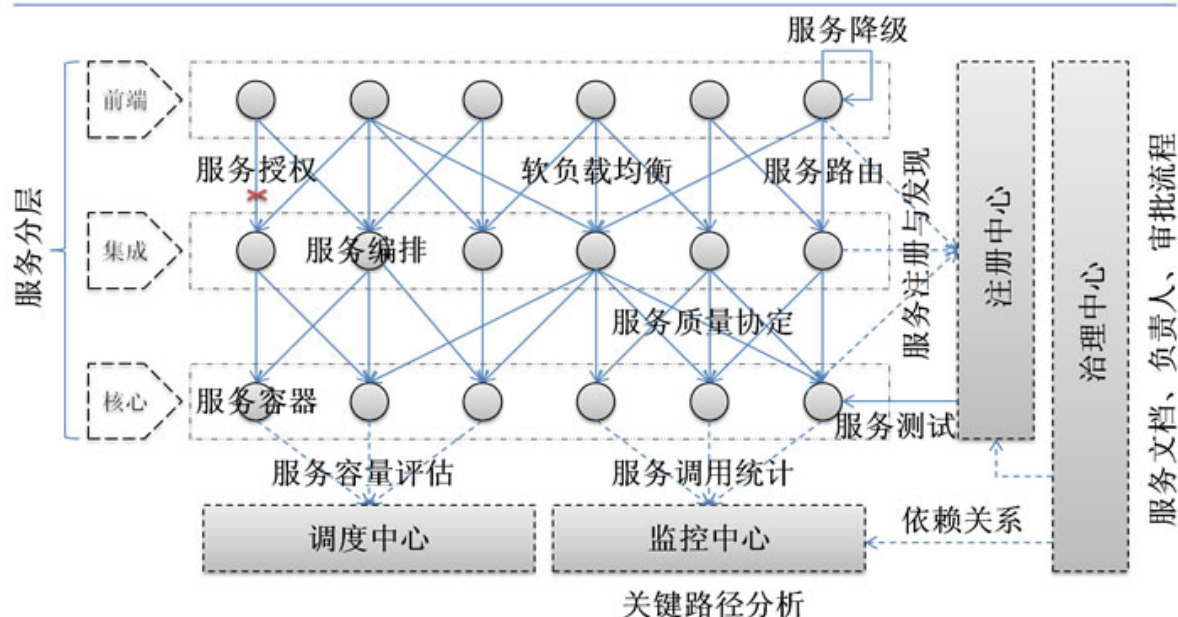






## 服务治理的本质

### Dubbo服务治理



## 服务治理（人员体系管理）

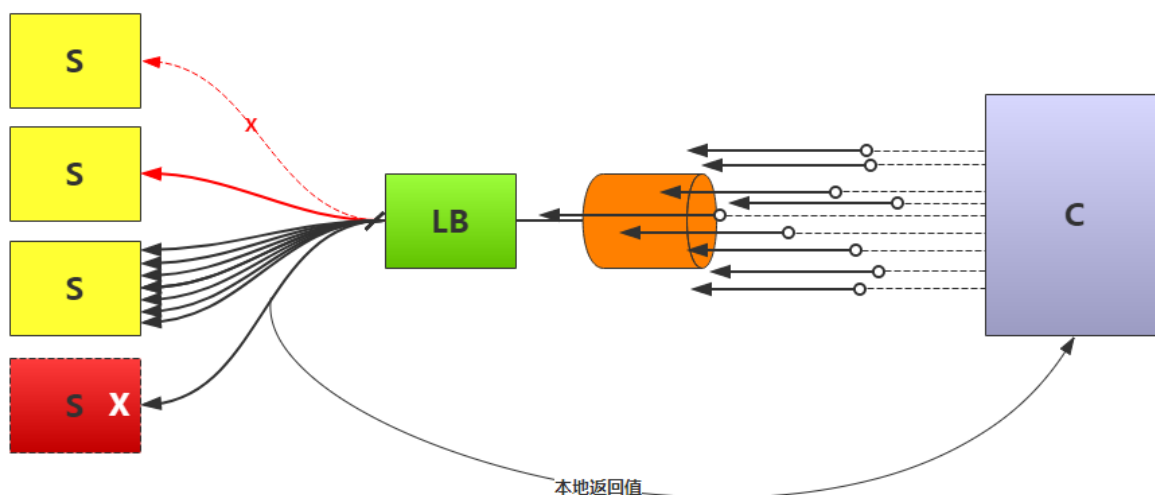
### • 为何需要治理

1. 在大规模服务化之前，应用可能只是通过 RMI 或 Hessian 等工具，简单的暴露和引用远程服务，通过配置服务的 URL 地址进行调用，通过 F5 等硬件进行负载均衡。
2. 当服务越来越多时，服务 URL 配置管理变得非常困难，F5 硬件负载均衡器的单点压力也越来越大。此时需要一个服务注册中心，动态地注册和发现服务，使服务的位置透明。并通过在消费方获取服务提供方地址列表，实现软负载均衡和 Failover，降低对 F5 硬件负载均衡器的依赖，也能减少部分成本。
3. 当进一步发展，服务间依赖关系变得错综复杂，甚至分不清哪个应用要在哪个应用之前启动，架构师都不能完整的描述应用的架构关系。这时，需要自动画出应用间的依赖关系图，以帮助架构师理清关系

4. 接着，服务的调用量越来越大，服务的容量问题就暴露出来，这个服务需要多少机器支撑？  
什么时候该加机器？为了解决这些问题，第一步，要将服务现在每天的调用量，响应时间，都统计出来，作为容量规划的参考指标。其次，要可以动态调整权重，在线上，将某台机器的权重一直加大，并在加大的过程中记录响应时间的变化，直到响应时间到达阈值，记录此时的访问量，再以此访问量乘以机器数反推总容量

- 治理的手段
  - hystrix
  - sentinel
  - Micro governance
- 热门的服务治理技术原理
  - 限流指标
    - QPS
    - TPS
    - 并发连接数
    - 系统负载
  - 隔离
    - 线程池隔离
    - 指标隔离
  - 熔断降级
    - 失败率降级
    - 平均响应时间降级

## 整体流程图（流量塑形）



## 服务限流原理讲解

限流和熔断是处理并发的两大利器，客户端熔断，服务端限流

Dubbo本地伪装

### 限流的主要作用

保护应用，防止雪崩（重试与超时积压）。每个应用都有自己处理请求的上限，一旦应用承受过多请求，首先会对正在处理中的请求造成影响，如果更严重，对上下游也会造成雪崩效应

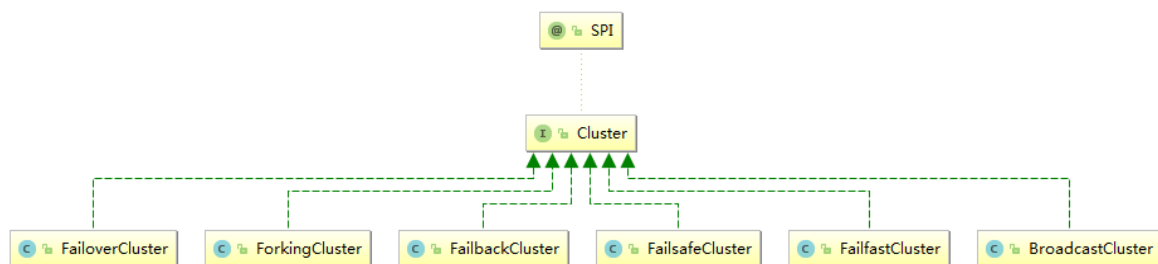


- 客户端限流
  - 信号量限流（通过统计的方式）
  - 连接数限流 (socket->tcp)
- 服务端限流
  - 线程池限流 (隔离手段)
  - 信号量限流 (非隔离手段)
  - 接收数限流 (socket->tcp)

## 服务容错原理讲解

服务容错就是保证服务客户端的高可靠调用（HA）

### 整体类结构图

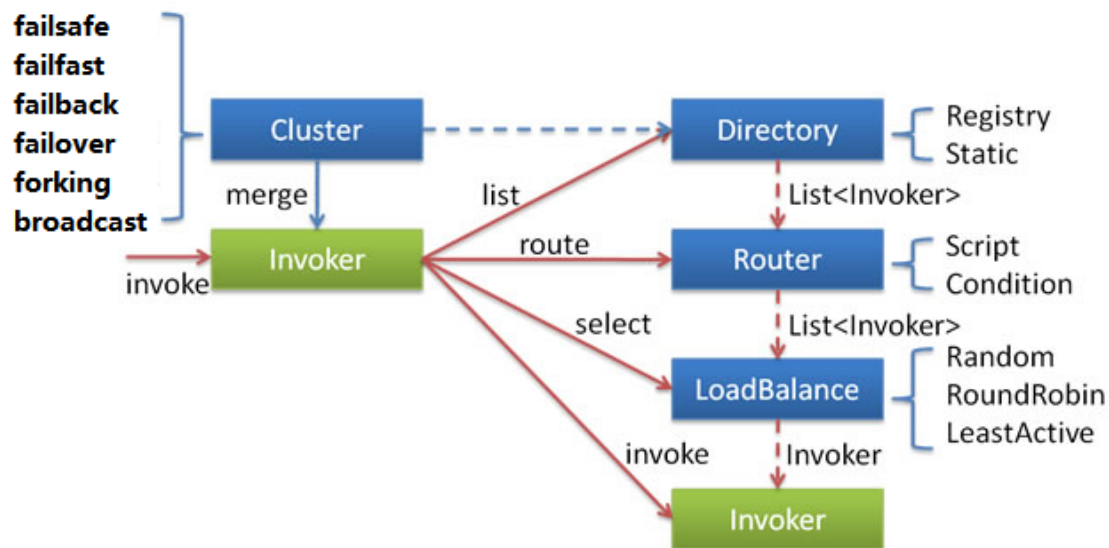


### 容错的主要作用

通过容错机制,缓和服务单点调用导致失败的问题

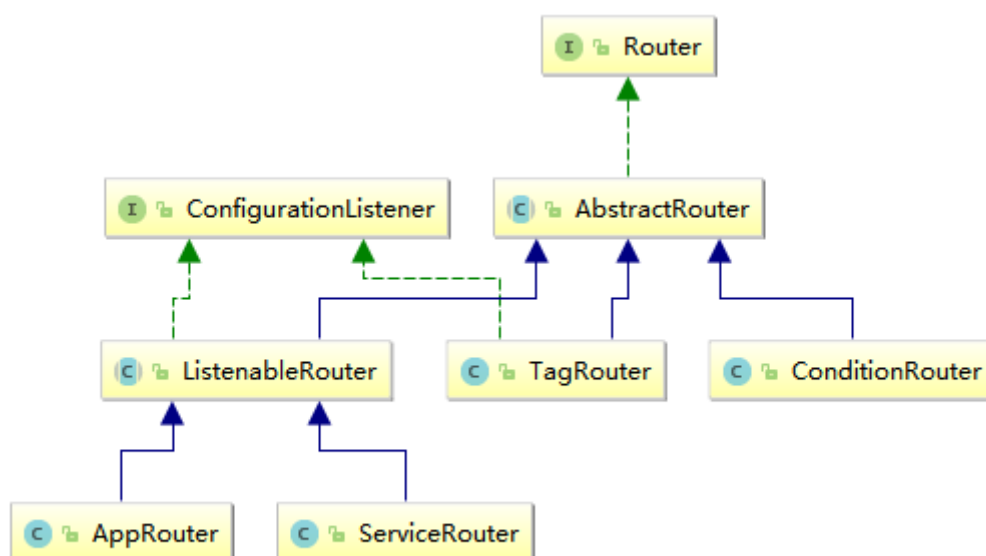
### 容错策略

- **failfast** 快速失败，只发起一次调用，失败立即报错【非幂等性的写操作】
- **failback** 失败之后，后台记录，隔一段时间，定时再次调用【必须成功调用服务】
  - 调用补偿机制（增量过程）
- **failover** 失败自动切换【默认】retries="3"
- **failsafe** 出现异常时，返回空结果, 直接忽略，记录失败错误
- **forking** 并行调用多个服务器，只要一个成功即返回【实时性要求较高的读操作】预缓存这种热标记
- **broadcast** 广播调用所有提供者，逐个调用，任意一台报错则报错【服务同步数据】跨同Server下的 instances



## 路由规则

路由规则，决定一次 dubbo 服务调用的目标服务器，分为条件路由规则和脚本路由规则，并且支持可扩展



### 标签路由设置规则

```
# governance-tagrouter-provider应用增加了两个标签分组tag1和tag2
# tag1包含一个实例 127.0.0.1:20880
# tag2包含一个实例 127.0.0.1:20881
---
force: false           #当路由结果为空时，是否强制执行
runtime: true          #是否在每次调用时执行路由规则
enabled: true          #当前路由规则是否生效
key: governance-tagrouter-provider #key取值为application名称
tags:
```

```

- name: tag1                                #标签名称
  addresses: ["127.0.0.1:20880","127.0.0.3:20880"]  #当前标签包含的实例列表
- name: tag2
  addresses: ["127.0.0.1:20881"]
...

```

uc.tag-router

<dubbo:application name="uc">

/dubbo/config/uc/tag-router [追加节点值]

### 静态路由打标

```

<!-- 提供者标签设置 -->
<dubbo:provider tag="tag1"/>
<!-- 单服务标签设置 -->
<dubbo:service tag="tag1"/>

```

### RPC上下文动态路由打标

```

RpcContext.getContext().setAttachment(Constants.REQUEST_TAG_KEY, "tag1");

```

### Java参数路由打标

```

java -jar xxx-provider.jar -Ddubbo.provider.tag={the tag you want, may come from
OS ENV}

```

### 条件路由设置规则

```

#应用粒度-----
# app1的消费者只能消费所有端口为20880的服务实例
# app2的消费者只能消费所有端口为20881的服务实例
---
scope: application
force: true                #当路由结果为空时，是否强制执行
runtime: true              #是否在每次调用时执行路由规则
enabled: true              #当前路由规则是否生效
key: governance-conditionrouter-consumer    #key取值为application名称
conditions:
- application=app1 => address=*:20880
- application=app2 => address=*:20881
...
#服务粒度
# DemoService的sayHello方法只能消费所有端口为20880的服务实例
# DemoService的sayHi方法只能消费所有端口为20881的服务实例
---
scope: service
force: true
runtime: true
enabled: true

```

```
key: org.apache.dubbo.samples.governance.api.DemoService #key取值为[{group}:]
{service}[:{version}]的组合
conditions:
  - method=sayHello => address=*:20880
  - method=sayHi => address=*:20881
...
```