链滴

[转]服务治理过程演进

作者：skyesx
原文链接：https://ld246.com/article/1467197345547
来源网站：链滴

转自 梁飞的博客 http://javatar.iteye.com/blog/1345073

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，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。

(1) 当服务来越多时，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。 br />
此时需要一个服务注册中心，动态的注册和发现服务，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硬件负载均衡器依赖，也能减少部分成本。

(2) 当进一步发展，服务间依赖关系变得错踪复，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，以帮助架构师理清理关系。

 (3) 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？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，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。

(4) 规模继扩大，应用之间不再是扁平的对应关系，开始分层，比如核心数据层，业务集成层等，就算没有出现环依赖，也不允许从低层向高层依赖，以免后续被逼循环依赖。

这时，需在注册中心定义架构体系，列明有哪些层的定义，每个服务暴露或引用时，都必须声明自己应用属于一层，这样注册中心能更快的发现架构的腐化现象。

(5) 服务多了，沟通成也开始上升，调某个服务失败该找谁？服务的参数都有什么约定？

这时就要登记每个服务都是谁负责的，并建立一个服务的文档库，方便检索。

(6) 慢一些敏感数据也都服务化了，安全问题开始变得重要，谁能调该服务？如何授权？

这样的服务可能需要一个密码，访问时需带着此密码，但如果用密码，要改密码时，就会很方便，所有的消费方都要改，所以动态生成令牌(Token)可能会更好，提供方将令牌告之注册中心，注册中心决定是否告之消费方，这样就能在注册中心页面上做复杂的授权模型。

(7) 就算是不敏感的服务，也不是能任意调用，比如某服务突然多了一个消费者，这个消费者的请量直接把服务给拖跨了，其它消费者跟着一起故障。

首先服务提供方需要控，当流程超标时，能拒绝部分请求，进行自我保护。

其次，消费者上线前和提供者定《服务质量等级协定(SLA)》，SLA包括消费者承诺每天调用量，请求数据量，提供方承诺响应时间出错率等，将SLA记录在监控中心，定时与监控数据对比，超标则报警。

(8) 虽然有SLA约定，如果不能控制，就只是君子协定，如何确保服务质量？

 如：一个应用很重要，一个不那么重要，它们调用同一个服务，这个服务就应该向重要应用倾斜，而是一视同仁，当支撑不住时，应限制不重要应用的访问，保障重要应用的可用，如何做到这一点呢。时，就需要服务路由，控制不同应用访问不同机器，比如：
应用分离：
consumer.application = foo => provider.host = 1,2,3
consumer.application != foo => provider.host = 5,6
读写分离：
method.name = find*,get* => provider.host = 1,2,3
 method.name != find*,get* => provider.host = 5,6

(9) 服务上线后需要验证服务是否可用，但因防火墙的限制，线下是不能访问线上服务的，不得不先写好一个测试Man，然后放到线上去执行，非常麻烦，并且容易忘记验证。

所以线上需要一个自动运行的验证程序，用户只需在界面上填上要验证的服务方法，以及参数值和期望的返回值，有一个服务提供者上线时，将自动运行该用例，并将运行结果发邮件通知负责人。

(10) 服务应用和Web应用是有区别的，它是一个后台Daemon程序，不需要Tomcat之类的We容器。但因公司之前以Web应用为主，规范都是按Web应用的，所以不得不把服务跑在一个根本用上的Web容器里，而搭一个这样的Web工程也非常费事。

所以需要实现个非Web的容器，只需简单的Main加载Spring配置即可，并提供Maven模板工程，只需mvn dubbo generate 即可创建一个五脏俱全的服务应用。

(11) 开发服务的人越来越多更注重开发效率，IDE的集成支持必不可少。

通过插件，可以在Eclipse中接运行服务，提供方可以直接填入测试数据测试服务，消费方可以直接Mock服务不依赖提供方开发。

(12) 因为暴露服务很简单，服务的上线越来越随意，有时候负责服务化的构师都不知道有人上线了某个服务，使得线上服务鱼龙混杂，甚至出现重复的服务，而服务下线比上还困难。

需要一个新服务上线审批流程，必须经过服务化的架构师审批过，才可以上线。

而服务下线时，应先标识为过时，然后通知调用方尽快修改调用，直没有人调此服务，才能下线。

(13) 因服务接口设计的经验一直在慢慢的积过程中，很多接口并不能一促而蹴，在修改的过程中，如何保证兼容性，怎么判断是否兼容？另外，

深层次的，业务行为兼容吗？

可以根据使用的协议类型，分析接口及领域型的变更是否兼容，比如：对比加减字段，方法签名等。

而业务上，可能需要基于自回归测试用例，形成Technology Compatibility Kit (TCK)，确保兼容升级。

(14) 随着服务的不停升级，总有些意想不到的事发生，比如cache写错了导致内存溢出，故障不可避，每次核心服务一挂，影响一大片，人心慌慌，如何控制故障的影响面？服务是否可以功能降级？或资源劣化？

应用间声明依赖强度，哪些功能强依赖，哪些弱依赖，然后基依赖强度，计算出影响面，并定期测试复查，加强关键路径上的服务的优化和容错，清理不该在关键径上的服务。

提供容错Mock数据，Mock数据也应可以在注册中心在运行时动态下发当某服务不可用时，用Mock数据代替，可以减少故障的发生，比如某验权服务，当验权服务全部挂后，直接返回false表示没有权限，并打印Error日志报警。

另外，前端的页面也应采用ortal进行降级，当该Portal获取不到数据时，直接隐藏，或替换为其它模块展示，并提供功能开关，人工干预是否展示，或限制多少流量可以展示。

(15) 当已有很多小服务，能就需要组合多个小服务的大服务，为此，不得不增加一个中间层，暴露一个新服务，里面分别调其小服务，这样的新服务业务逻辑少，却带来很多开发工作量。

此时，需要个服务编排引擎，内置简单的流程引擎，只需用XML或DSL声明如何聚合服务，注册中心可以直接下给消费者执行聚合逻辑，或者部署通用的编排服务器，所有请求有编排服务器转发。

<trong>(16) 并不是所有服务的访问量都大，很多的服务都只有一丁点访问量，却需要部署两台提供务的机器，进行HA互备，如何减少浪费的机器。

此时可能需要让服务容支持在一台机器上部署多个应用，可以用多JVM隔离，也可以用ClassLoader隔离。

<srong>(17) 多个应用如果不是一个团队开发的，部署在一台机器上，很有可以误操作，停掉了别人的务。

所以需要实现自动部署，所有的部署都无需人工干扰，最好是一键式署。

(18) 机器总是的闲时和忙时，或者冗余机器防灾，如何提高机器的利率？

即然已经可以自动部署了，那根据监控数据，就可以实现资源调度，据应用的压力情况，自动添加机器并部署。

如果你的应用是国际化的，有中文站，美站之类，因为时差，美国站的机器晚上闲的时候，可能正是中文站的白天忙时，可以通过资源调度，时段自动调配和部署双方应用。

按关键词归纳为：

1. 服务注册与发现

2. 软负载均衡与容错

3. 服务监控与统计
 br />4. 服务容量评估

5. 服务上线审批

6. 服务下线通知

 7. 服务负责人

8. 服务文档

9. 服务路由

10. 服务编排 <b />
11. 服务黑白名单

12. 服务权限控制

13. 服务依赖关系

14. 服务分层架构

15. 服务调用链跟踪

16. 故障传导分析

17. 服务降级

18. 服务等级协定

19. 服务自动测试

 />20. 服务伪装容错

21. 服务兼容性检测

22. 服务使用情况报告

23. 服务权重动态调整

24. 服务负载均衡调整

25. 服务映射

26. 服务模板工程

27. 服务开发IDE

28. 服务健康检测

29. 服务容器

30. 服务自动部署

31. 服务资源调度</p>