微服务熔断与隔离

微服务 (/lib/tag/微服务) 分布式系统 (/lib/tag/分布式系统) 2016-03-03 14:47:48 发布

来自： https://yq.aliyun.com/articles/7443 (https://yq.aliyun.com/articles/7443)

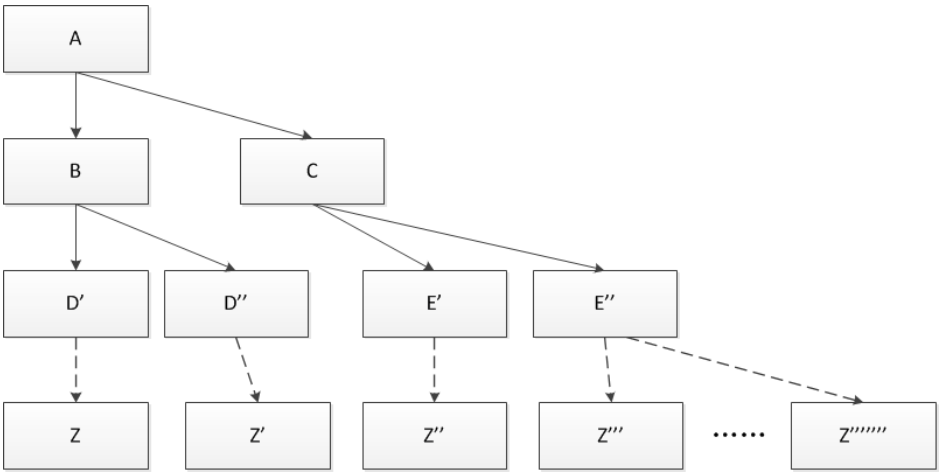
微服务近年来很火很热，相关的文章汗牛充栋，关于架构设计本文就不作叙述了，只谈谈在分布式服务的容错方面怎么做。

1 什么是微服务

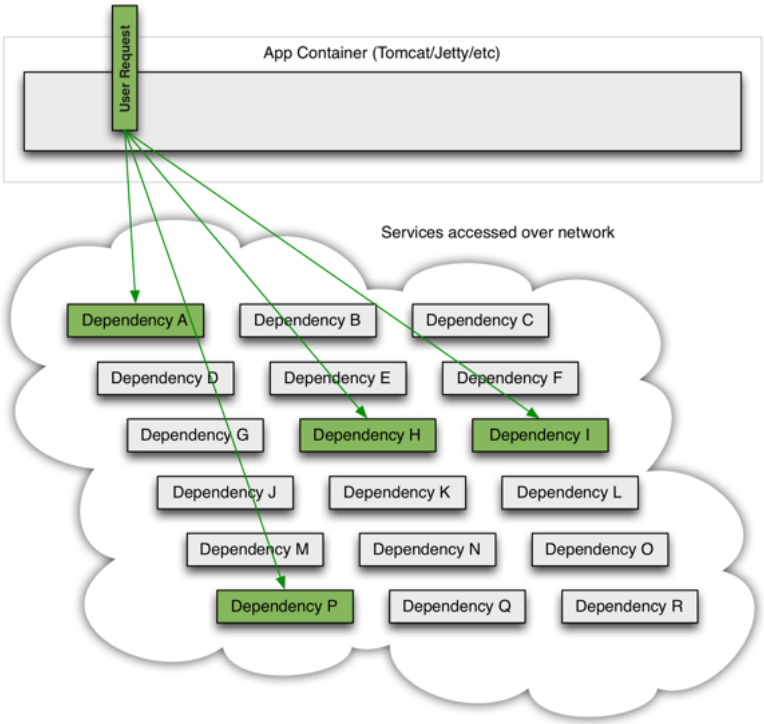
对于微服务，我们可以简单的理解成对一个服务解耦，以降低业务系统的复杂性，将服务系统中的功能进行拆分成多个轻量的子服务，各个自服务间通过RPC实现服务间的关联，这样做的好处是将业务简单化，每个子服务可以有自己独立的编程语言，模式等且能够独立维护，独立部署，功能复用。

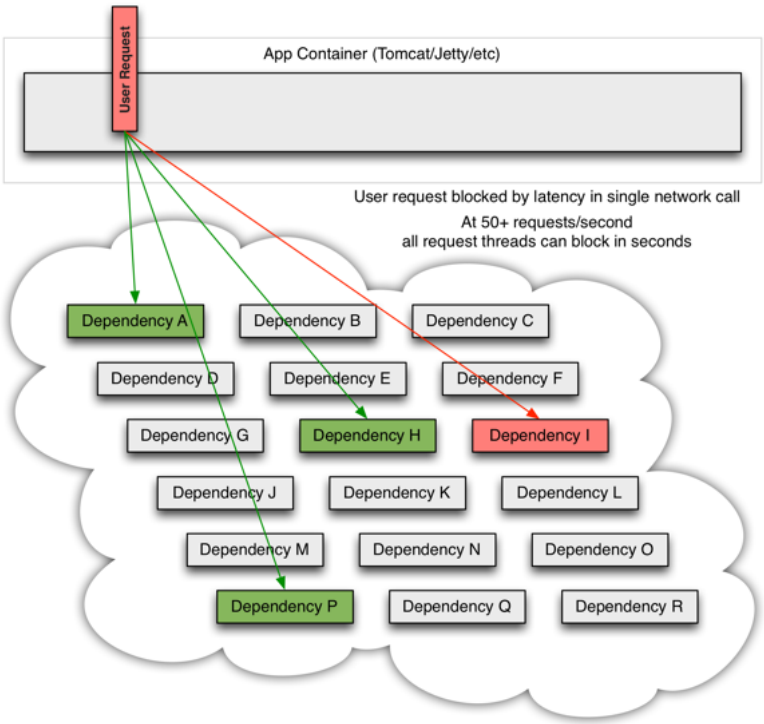
2 为什么需要做服务隔离与熔断

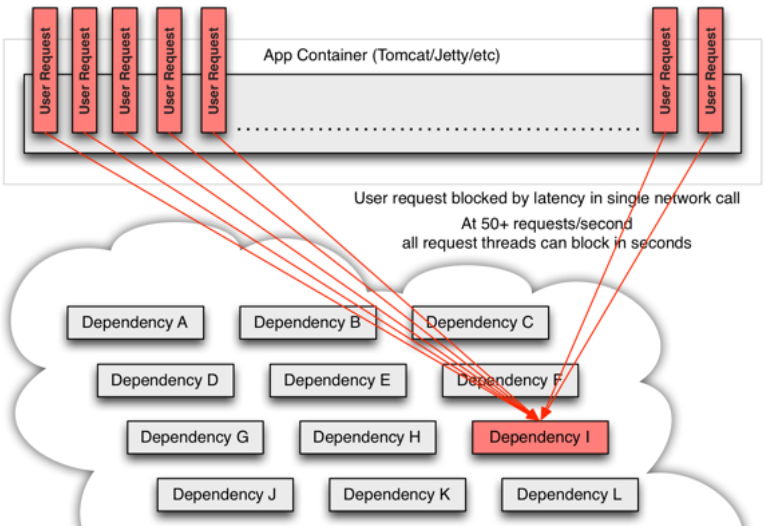
由于微服务间通过RPC来进行数据交换，所以我们可以做一个假设：在IO型服务中，假设服务A依赖服务B和服务C，而B服务和C服务有可能继续依赖其他的服务， 继续下去会使得调用链路过长，技术上称1->N扇出。如果在A的链路上某个或几个被调用的子服务不可用或延迟较高，则会导致调用A服务的请求被堵住，堵住的请求会消耗占用掉系统的线程、io等资源，当该类请求越来越多，占用的计算机资源越来越多的时候，会导致系统瓶颈出现，造成其他的请求同样不可用，最终导致业务系统崩溃，又称：雪崩效应。

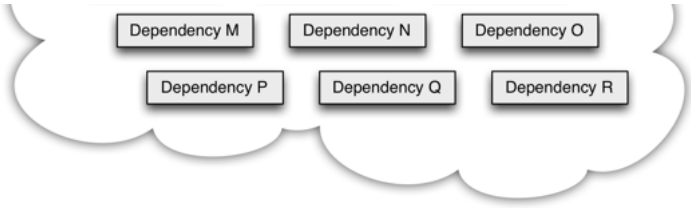


1->N扇形


User Request
App Container (Tomcat/Jetty/etc)
Services accessed over network
Dependency A
Dependency B
Dependency C
Dependency D
Dependency E
Dependency F
Dependency G
Dependency H
Dependency I
Dependency J
Dependency K
Dependency L
Dependency M
Dependency N
Dependency O
Dependency P
Dependency Q
Dependency R


User Request
App Container (Tomcat/Jetty/etc)
User request blocked by latency in single network call
At 50+ requests/second
all request threads can block in seconds
Dependency A
Dependency B
Dependency C
Dependency D
Dependency E
Dependency F
Dependency G
Dependency H
Dependency I
Dependency J
Dependency K
Dependency L
Dependency M
Dependency N
Dependency O
Dependency P
Dependency Q
Dependency R


User Request
App Container (Tomcat/Jetty/etc)
User request blocked by latency in single network call
At 50+ requests/second
all request threads can block in seconds
Dependency A
Dependency B
Dependency C
Dependency D
Dependency E
Dependency F
Dependency G
Dependency H
Dependency I
Dependency J
Dependency K
Dependency L



雪崩效应

3 服务雪崩的原因

（1）某几个机器故障：例如机器的硬驱动引起的错误，或者一些特定的机器上出现一些的bug（如，内存中断或者死锁）。

（2）服务器负载发生变化：某些时候服务会因为用户行为造成请求无法及时处理从而导致雪崩，例如阿里的双十一活动，若没有提前增加机器预估流量则会造服务器压力会骤然增大二挂掉。

（3）人为因素：比如代码中的路径在某个时候出现bug

4 解决或缓解服务雪崩的方案

一般情况对于服务依赖的保护主要有3中解决方案：

（1）熔断模式：这种模式主要是参考电路熔断，如果一条线路电压过高，保险丝会熔断，防止火灾。放到我们的系统中，如果某个目标服务调用慢或者有大量超时，此时，熔断该服务的调用，对于后续调用请求，不在继续调用目标服务，直接返回，快速释放资源。如果目标服务情况好转则恢复调用。

（2）隔离模式：这种模式就像对系统请求按类型划分成一个个小岛的一样，当某个小岛被火少光了，不会影响到其他的小岛。例如可以对不同类型的请求使用线程池来资源隔离，每种类型的请求互不影响，如果一种类型的请求线程资源耗尽，则对后续的该类型请求直接返回，不再调用后续资源。这种模式使用场景非常多，例如将一个服务拆开，对于重要的服务使用单独服务器来部署，再或者公司最近推广的多中心。

（3）限流模式：上述的熔断模式和隔离模式都属于出错后的容错处理机制，而限流模式则可以称为预防模式。限流模式主要是提前对各个类型的请求设置最高的QPS阈值，若高于设置的阈值则对该请求直接返回，不再调用后续资源。这种模式不能解决服务依赖的问题，只能解决系统整体资源分配问题，因为没有被限流的请求依然有可能造成雪崩效应。

5 熔断设计

在熔断的设计主要参考了hystrix的做法。其中最重要的是三个模块：熔断请求判断算法、熔断恢复机制、熔断报警

（1）熔断请求判断机制算法：使用无锁循环队列计数，每个熔断器默认维护10个bucket，每1秒一个bucket，每个blucket记录请求的成功、失败、超时、拒绝的状态，默认错误超过50%且10秒内超过20个请求进行中断拦截。

（2）熔断恢复：对于被熔断的请求，每隔5s允许部分请求通过，若请求都是健康的（RT<250ms）则对请求健康恢复。

（3）熔断报警：对于熔断的请求打日志，异常请求超过某些设定则报警

6 隔离设计

隔离的方式一般使用两种

（1）线程池隔离模式：使用一个线程池来存储当前的请求，线程池对请求作处理，设置任务返回处理超时时间，堆积的请求堆积入线程池队列。这种方式需要为每个依赖的服务申请线程池，有一定的资源消耗，好处是可以应对突发流量（流量洪峰来临时，处理不完可将数据存储到线程池队里慢慢处理）

（2）信号量隔离模式：使用一个原子计数器（或信号量）来记录当前有多少个线程在运行，请求来先判断计数器的数值，若超过设置的最大线程个数则丢弃改类型的新请求，若不超过则执行计数操作请求来计数器+1，请求返回计数器-1。这种方式是严格的控制线程且立即返回模式，无法应对突发流量（流量洪峰来临时，处理的线程超过数量，其他的请求会直接返回，不继续去请求依赖的服务）

7 超时机制设计

超时分两种，一种是请求的等待超时，一种是请求运行超时。

等待超时：在任务入队列时设置任务入队列时间，并判断队头的任务入队列时间是否大于超时时间，超过则丢弃任务。

运行超时：直接可使用线程池提供的get方法

8 隔离与熔断代码实现

后续会放到github上

9 性能损耗测试

由于存在计数统计和线程切换等的开销，所以对每个请求会有一定的性能损耗，测试结果表明在线程池隔离模式中，平均一个请求的损耗在0.5ms以内。

测试方法：顺序请求，记录业务运行时间和隔离器运行业务的时间，请求数量500次。

变量解释：

单个请求耗时：为业务的运行时间（使用Thread.sleep()模拟）；

隔离消耗=请求总用时-业务用时;

隔离评价消耗=隔离消耗/请求次数/

测试时间统计（单位ms）：

单个请求耗时	请求总用时	业务用时	隔离消耗	隔离平均消耗
1	586	510	76	0.152
5	2637	2514	124	0.248
10	5248	5136	112	0.024
50	25261	25111	150	0.3
100	50265	50130	135	0.27
200	100657	100284	373	0.746

10 参考

在设计和实现的过程中参考了一些现有的设计和一些文章：

1、Hystrix官方文档：https://github.com/Netflix/Hystrix/wiki

2、Hystrix使用与分析：http://hot66hot.iteye.com/blog/2155036

3、Facebook文章：http://queue.acm.org/detail.cfm?id=2839461

4、Facebook文章：http://queue.acm.org/detail.cfm?id=2209336

4、 分布式服务容错模式和实践： http://www.atatech.org/articles/31559

同类热门经验

1. 使用Java快速入门RPC框架 - Thrift (/lib/view/open1341798273463.html)
2. Spring提供的用于访问Rest服务的客户端：RestTemplate实践 (/lib/view/open1436018677419.html)
3. C# 创建和调用 WebService (/lib/view/open1330408551889.html)
4. REST 入门介绍 (/lib/view/open1336819781369.html)
5. Java Web 服务性能优化实践 (/lib/view/open1356048814526.html)
6. 理解REST软件建构设计理念 (/lib/view/open1343635725774.html)

阅读目录

相关文档 — 更多 (http://www.open-open.com/doc)

- 分布式数据库服务器系统的可信化研究.pdf (http://www.open-open.com/doc/view/cb83887379594c67bb2983654b9ceca7)
- CORBA 事件服务在分布式远程诊断系统中的应用研究.pdf (http://www.open-open.com/doc/view/22704553b3bf47e7a9f68f6b45a26405)
- 分布式系统概念与设计（第3版）.pdf (http://www.open-open.com/doc/view/e9c2634bfb704c20a8a1ba3d9c7c230a)
- 微服务的场景化应用-刘永峰.pdf (http://www.open-open.com/doc/view/e3366116ec544559aa33699e432f1c24)
- 分布式存储与web服务伸缩性.pdf (http://www.open-open.com/doc/view/383a926f00424e04a30ce2e50fab8a95)
- 《分布式系统》教学大纲.doc (http://www.open-open.com/doc/view/f5f1ec2059bf44e6bb509508504cf042)
- 淘宝分布式服务框架.pdf (http://www.open-open.com/doc/view/e7c4ebd8be77492ba55f93deede74a96)
- 云计算核心技术架构与实现.pdf (http://www.open-open.com/doc/view/836c29fb7f494208afeb787d8805cfa8)
- chapter 7 分布式系统中容错技术.ppt (http://www.open-open.com/doc/view/6e66f24902e84e70a6eba42d7db7146c)
- 大规模分布式存储系统：原理解析与架构实战.pdf (http://www.open-open.com/doc/view/f541d57d0ea5488d9d7672804f21ce6d)
- 分布式java应用.pdf (http://www.open-open.com/doc/view/6f8ad25c2a6841c8bc32e3b76bfdb073)
- 分布式采集系统总体设计方案.doc (http://www.open-open.com/doc/view/677e953b922c418380981bb42eb0a828)
- 分布式平台微服务架构演化实践.pdf (http://www.open-open.com/doc/view/c15c9ed0b598413c99d7cb2f632b60ac)
- 分布式计算开源框架 Hadoop 入门实践.pdf (http://www.open-open.com/doc/view/345725af71f3418198d391273468a682)
- 分布式系统概念与设计(第3版).pdf (http://www.open-open.com/doc/view/acaa58c7a8144d44af9054fc008d3ffa)
- 分布式文件系统tfs架构演进.pdf (http://www.open-open.com/doc/view/be16e660a09b4c5296a971f014d30424)
- Benchmark Factory 使用指导书.docx (http://www.open-open.com/doc/view/d9068fe11cbe4ec69eddfbeecb0ccf55)
- Benchmark Factory 使用指导书.docx (http://www.open-open.com/doc/view/e9aaf960d266430e91f58dc1af67cb60)
- 从Paxos到ZooKeeper：分布式一致性原理与实践.pdf (http://www.open-open.com/doc/view/434c5807d65647e1b14e73642e4b7828)
- 基于CORBA的分布式系统的设计.pdf (http://www.open-open.com/doc/view/578009e823cb40538488b6a815abd965)

相关经验 — 更多 (http://www.open-open.com/lib)

- 从单体架构迁移到微服务，8个关键的思考、实践和经验 (/lib/view/open1470917801705.html)
- 一篇文章带你了解Cloud Native (/lib/view/open1447420363069.html)
- 简述 Microservices（微服务） (/lib/view/open1455947863589.html)
- 微服务架构宜缓行 (/lib/view/open1434464403755.html)
- 基于容器云的微服务架构实践 (/lib/view/open1437459743662.html)
- Android系统架构之微服务架构 (/lib/view/open1433314468807.html)
- Android系统架构之微服务架构 (/lib/view/open1434008066004.html)
- （译）面向性能的微服务 (/lib/view/open1460989472558.html)
- 微服务与持续交付 (/lib/view/open1451440541448.html)
- 容器识别与访问管理技术：Lightwave (/lib/view/open1429576653510.html)
- 使用容器构建微服务架构 (/lib/view/open1422868144986.html)
- 使用容器技术来建立一个微服务架构 (/lib/view/open1437985519175.html)
- 微服务实践（五）：微服务的事件驱动数据管理 (/lib/view/open1452060266526.html)
- 微服务架构实践总结 (/lib/view/open1437749771568.html)

相关讨论 — 更多 (http://www.open-open.com/solution)

- 分布式与集群的区别 (http://www.open-open.com/solution/view/1410837958539)
- 百度 网页搜索部_系统架构高级工程师 (http://www.open-open.com/solution/view/1376722885803)
- 处理遗留系统 (http://www.open-open.com/solution/view/1338261747448)
- 高性能分布式数据库系统 OceanBase (http://www.open-open.com/solution/view/1319457574140)
- 那些年，追过的开源软件和技术 (http://www.open-open.com/solution/view/1425959150201)
- Windows平台分布式架构实践 - 负载均衡 (http://www.open-open.com/solution/view/1405087737029)
- 北京知名公司招聘：高级、资深java 应用/系统架构师 (http://www.open-open.com/solution/view/1380164957758)