计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2019,28(11):271−275 [doi: 10.15888/j.cnki.csa.007165]
©中国科学院软件研究所版权所有.
E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

微服务架构中的服务质量保障研究①

佟业新, 曲新奎

(中航信移动科技有限公司, 北京 100029)

摘　要: 微服务架构逐渐成为构建复杂业务系统时的主流架构, 但同时系统架构变得更加复杂, 服务质量的保障成为一个关键的问题. 本文围绕微服务架构下服务质量保障展开研究, 通过熔断、降级、多路调用等方式解决服务调用过程中出现的各种问题. 提出了微服务调用参数的计算模型, 以实时的业务和设备监控数据为基础, 对服务调用参数的精细化计算, 并支持在线的动态参数调整. 通过优雅停机和流量预热方法, 解决了服务重启过程中出现的质量抖动问题. 本文所提出的方法和模型, 在实际应用中得到了充分验证, 响应时间、失败率等指标大幅降低, 使微服务架构下的系统更加稳定可靠的运行.

关键词: 微服务; 服务质量; 稳定和可靠

引用格式: 佟业新,曲新奎.微服务架构中的服务质量保障研究.计算机系统应用,2019,28(11):271–275. http://www.c-s-a.org.cn/1003-3254/7165.html

Research on Quality of Service Guarantee in Micro-Service Architecture

TONG Ye-Xin, QU Xin-Kui

(Travelsky Mobile Technology Ltd., Beijing 100029, China)

Abstract: Micro-service architecture has gradually become the mainstream architecture for building complex business systems, but at the same time, the system architecture has become more complex, and the quality of service assurance becomes a key issue. This study focuses on the quality of service assurance under the micro-service architecture, and solves various problems in the process of service invocation by fusing, degrading and multi-channel invocation. Based on real-time monitoring data of business and equipment, a calculation model of service invocation parameters is proposed, which can refine the calculation of service invocation parameters and support online dynamic parameter adjustment. Through the elegant shutdown and flow preheating method, the quality jitter problem in the process of service restart is solved. The method and model proposed in this study have been fully validated in practical application. The response time and failure rate are greatly reduced, which makes the system run more stably and reliably under the micro-service architecture.

Key words: micro-service; service quality; stable and reliable

随着业务的发展, 企业规模扩大, 单体的架构已经无法满足企业对系统的需求, 需要根据组织的实际情况进行相应的架构调整[1]. 采用微服务架构的系统是由一系列协同工作且独立的服务组成[2], 与传统的的单体架构相比, 微服务架构具有很多的优点, 如灵活性更高, 具有更高的弹性和扩展性, 各个模块独立开发部署, 互不影响等.

同时也带来了一些问题, 采用微服务架构的应用属于分布式系统, 复杂性较高. 在实际应用过程中, 会碰到如下的一些问题, 包括服务调用链条较长, 存在重

① 收稿时间: 2019-04-23; 修改时间: 2019-05-20; 采用时间: 2019-05-27; csa 在线出版时间: 2019-11-06

© 中国科学院软件研究所 http://www.c-s-a.org.cn

复资源调用问题; 对异常时协议的容错处理不足, 被依赖的服务出现异常时, 协议的失败率波动很大鲁棒性差; 对一些服务依赖程度较高, 缺少降级策略; 服务调用的参数定义的不合理, 少量的服务调用导致整体服务不可用.

鉴于上述问题, 与单体架构相比, 在实际的应用中, 需要有更多的技术手段如分布式追踪、熔断隔离、实时监控等来保证微服务架构可靠性[3]. 为了保证系统的可靠性, 文中制定了质量评价指标, 围绕这些指标, 文中通过多线程并发调用以及分布式缓存的方法实现缩短响应服务时间的目的. 为了提升服务的可靠性, 从服务调用失败处理, 熔断隔离保护以及无损上线三个方面进行了研究, 并提出了熔断隔离参数的计算模型, 以对服务调用过程中的熔断隔离参数进精细化计算, 以便更加合理的调配多个服务对线程池等资源的使用. 这些研究在实际的项目中进行应用, 在服务质量提升方面起到了很好的效果.

1 服务质量保障方案

本文中服务质量的保障, 重点围绕缩短服务响应时间和提升服务可靠性展开, 同时需要提供实时的服务质量监控及评估和改进, 以达到服务质量的持续提升, 并在服务质量评价指标方面达到一定的标准.

在实际的生产运行中, 服务运行的速度、成败等因素直接影响到用户的体验, 因此将响应时间以及失败率作为本文重点关注服务质量评价指标进行定义. 同时系统的处理量会对响应时间以及失败率的指标产生影响.

在文中提出熔断隔离参数的计算模型, 以实际生产中的文中提到的关键质量指标为基础进行设计, 由于生产数据每时每刻都在变化, 为了应对这种情况, 需要进行实时数据的收集, 并基于模型进行实时计算, 并将计算结果在线推送给业务方, 进行在线生效变更.

1.1 质量评价指标

质量评价指标用于对系统的实际运行情况进行量化, 以实现对服务质量的监控以及评估[4]. 本文针对服务质量的保障, 重点关注如下几个指标:

1) 平均响应时间 (*avg*): 在一段时间内所有请求响应时间的平均值, 平均响应时间越小, 说明处理速度越快, 服务效率越高.

2) 95 线响应时间 (95*Line*): 在一段时间内 95% 请求响应时间的最大值.

3) 99 线响应时间 (99*Line*): 在一段时间内 99% 请求响应时间的最大值.

4) 失败率: 在一段时间内失败次数与总请求量的比值, 失败率越高说明服务可靠性越低.

5) *QPS*: 在一段时间内的服务器每秒所处理的查询量.

1.2 缩短服务响应时间

1.2.1 多线程并发调用

策略一. 多线程调用, 静态托底

(1) 应用场景: 需要对多个不同数据源的服务进行调用, 并对资源进行聚合, 需要在有限时间内完成.

(2) 策略描述: 多个服务通过多线程同时发起请求, 对返回的结果进行聚合. 设定响应时间的阈值, 对于超过阈值的服务, 对服务进行丢弃, 并对该服务中需要的数据进行容错处理.

(3) 优点: 解决了多服务并行调用带来的性能问题, 在服务质量得到保证的同时在规定时间内返回内容.

(4) 应用案例: 航旅纵横 APP 中的首页, 需要以服务推荐的形式为不同的业务提供运营位置, 进行功能展示. 各个服务需要按照要求提供服务, 由服务推荐模块进行服务聚合, 此处采用了该策略, 在规定时间内不满足性能要求的服务, 将无法获得推荐, 并保证了性能正常的服务可以正常展示. 调用过程如图 1 所示.

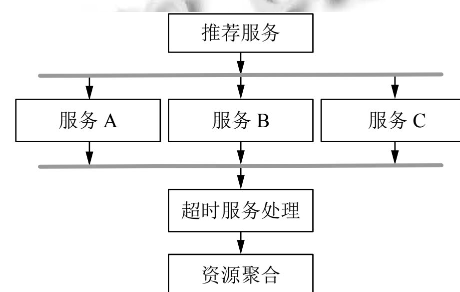

图 1 多线程调用过程

策略二. 多路保证, 优先选用

(1) 应用场景: 可用性和响应时间有极高要求的服务, 同时存在多个提供相似数据源的服务.

(2) 策略描述: 对于一次服务请求, 多个独立服务提供支持, 在请求处理过程中, 多路并发调用, 优先选用先返回的结果;

(3) 优点: 多路保证系统的稳定性; 以最先返回的结果优先, 一定程度上, 提高了响应时间;

© 中国科学院软件研究所 http://www.c-s-a.org.cn

(4) 缺点: 多路调用, 资源的消耗会根据同时调用的服务数量而倍数增长;

(5) 案例: 在航旅纵横 APP 中为旅客提供高可用的安检服务, 响应时间需要在 100 ms 以下, 并达到容错高可用能力, 考虑服务压力以及系统资源的情况下, 往往使用速度较快的服务代替资源紧张的服务.

1.2.2 分布式缓存

内存的访问性能明显优于磁盘. 把数据放入内存中, 可以提供更快的读取效率. 但在互联网业务的场景下, 将所有数据数据都装入内存, 显然是不明智的[5].

同时大部分的业务场景下, 80% 的访问量都集中在 20% 的热数据上 (适用二八原则). 因此, 通过引入缓存组件, 将高频访问的数据, 放入缓存中, 可以大大提高系统整体的承载能力, 提高响应速度, 提高用户体验. 原有单层 DB 的数据存储结构, 也变为 Cache+DB 的结构, 如图 2 所示.

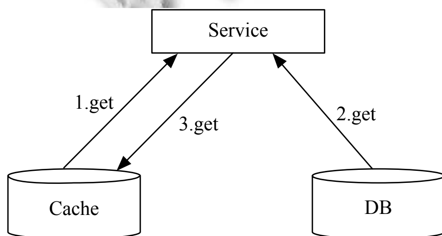

图 2 Cache+DB 结构图

在数据层引入缓存, 有以下几个好处:

(1) 提升数据读取速度;

(2) 提升系统扩展能力, 通过扩展缓存, 提升系统承载能力;

(3) 降低存储成本, Cache+DB 的方式可以承担原有需要多台 DB 才能承担的请求量, 节省机器成本.

根据业务场景, 通常缓存有以下几种使用方式:

(1) 懒汉式 (读时触发): 写入 DB 后, 然后把相关的数据也写入 Cache;

(2) 饥饿式 (写时触发): 先查询 DB 里的数据, 然后把相关的数据写入 Cache;

(3) 定期刷新: 适合周期性的跑数据的任务, 或者列表型的数据, 而且不要求绝对实时性;

(4) 实时刷新: 通过监听数据表的实时变更, 旁路推送到应用, 并进行缓存的实时更新.

1.3 提升服务可靠性

提升服务质量的另外一个重要任务是提升服务的可靠性. 重点对服务调用失败以及流量异常的处理方式进行了研究.

1.3.1 服务调用失败处理

针对调用失败有 Failfast 和 Failover 两种处理方式.

(1) Failfast: 快速失败只发起一次调用, 失败立即报错, 现在服务调用方的默认调用方式;

(2) Failover: 失败自动切换, 当出现失败, 重试其它服务器. 此种情况也支持, 慎用, 在某个服务提供方出现故障后, 会放大压力.

在实际应用中, Failfast 和 Failover 两种方式都进行了实现, 失败的处理方式通过分布式配置中心进行配置, 默认使用 Failfast. 如果各个业务有个性化需求, 可配置为 Failover 模式, 并设置重试次数的配置 key FailoverRetryCount, 即可实时切换为该模式.

1.3.2 熔断隔离保护

1.3.2.1 熔断隔离介绍

服务熔断, 类似于电路中的“保险丝”, 是指由于某些原因 (如用户大量请求、程序 Bug、硬件故障等) 造成目标服务调用慢或者有大量超时, 此时停用对该服务的调用, 等到服务情况好转时再恢复调用[6].

线程池隔离就是舱壁隔离模式, 货船通常利用舱壁将不同的船舱隔离起来, 防止一个船舱进水对其他船舱的影响, 这种资源隔离减少风险的方式就是舱壁隔离模式的一种应用场景. 该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, 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.

1.3.2.2 熔断隔离参数计算

在实际应用中, 可以引入 Hystrix 组件, 该组件采用线程池隔离机制, 实现对系统对稳定性保护. 通过隔离服务之间的依赖关系, 限制对其中任何一个的并发访问.

客户端请求会在单独的线程中执行, 执行依赖代码的线程与请求线程分离, 这样在某个依赖调用执行时间很长时, 请求线程可以自由控制离开的时间. Hystrix 通过为每个依赖服务分配独立的线程池进行资源隔离, 当某个服务不可用时, 即使为该服务独立分配的线程都处于等待状态, 也不会影响其他服务的调用.

服务启用线程池和熔断功能, 需要配置线程池大小和超时时间. 参数的准确程度会直接影响到熔断隔离

© 中国科学院软件研究所 http://www.c-s-a.org.cn

离的实际效果. 为了提升准确性, 结合现有的应用监控数据, 设计了服务调用参数的计算公式. 如公式 (1) 所示, 该公式用于计算被调用服务需要配置的线程数:

$$s(thread) = 95Line \div 1000 \times \max(QPS) \times (1+m) \quad (1)$$

其中, $s(thread)$ 代表应用所依赖服务的线程数, m 代表计算的冗余度, 95*Line* 表示某一天的 95 线响应时间, max(*QPS*) 表示某一天的 *QPS* 峰值. 通过该公式计算的数值, 具有一定的冗余度, 可以保证在 *QPS* 达到峰值, 服务响应时间达到 95*Line* 时, 线程数可以足够使用.

如式 (2) 所示, 用于计算服务的调用超时时间:

$$s(timeout) = 99Line \times (1+n) \quad (2)$$

其中, $s(timeout)$ 代表应用所依赖服务的超时时间, 使用 99*Line* 作为时间基准, n 为时间冗余度, 通过调整 n 的数值, 保证 99% 以上的请求不会因为超时而导致失败.

两个式子中计算的数值在极端情况下可能存在一定的冲突, 通过 95*Line* 作为基数计算的线程数, 如果使用 99*Line* 作为基数作为超时时间, 全部超时的情况下, 线程肯定不够用. 但实际中, 绝大多数请求都在平均响应时间上下, 同时通过应用监控系统及时发现调用的波动情况, 可以将响应时间控制在一定范围内, 保证了服务的稳定运行.

图 3 的参数计算流程展示了基于计算模型从数据收集到最终应用的全过程.

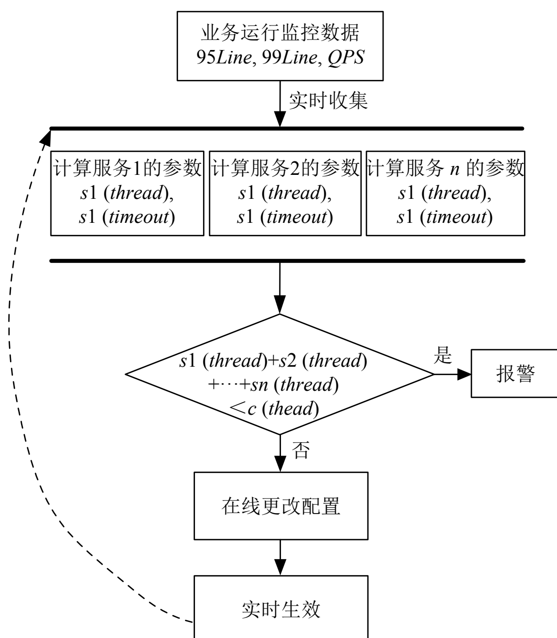

图 3　参数计算流程

(1) 从生产系统中实时收集到业务运行监控数据, 主要包括模型计算所需要的 95*Line*, 99*Line* 以及 *QPS*.

(2) 参数计算模型使用收集到的数据进行并行计算, 得到每个应用所依赖服务所需的线程数 $s(thread)$ 以及超时时间 $s(timeout)$.

(3) 考虑到系统性能问题, 设定所有被调用服务的线程数 $s(thread)$ 之和需要小于系统所在容器如 Tomcat 所设定的线程数 $c(thead)$, 如果超出了 $c(thread)$ 则进行系统报警, 提醒当前所需线程数总和超过了最大容量, 需要进行干预.

(4) 如果被调用服务的线程数 $s(thread)$ 之和小于 $c(thread)$, 则在线更改配置并实时生效.

(5) 生效以后, 进入下一个计算周期.

在进行参数计算的过程中, 考虑到实时收集数据数值的波动, 计算所用数值采用一段时间的数据, 在实际应用中为每隔 1 分钟会基于前 1 分钟的数值进行计算, 并将计算的结果根据上述流程进行计算和应用, 整个过程可以在秒级完成, 以保证系统安全, 及时应对流量突变.

1.3.3　无损上线

1.3.3.1　优雅停机

在应用重启过程中, 需要保证正在处理的请求能正常结束, 并且在停机前主动通知调用方不在继续向将要停机的服务器发送新请求, 待所有请求处理完毕之后, 进行应用的重启. 流程如图 4 所示.

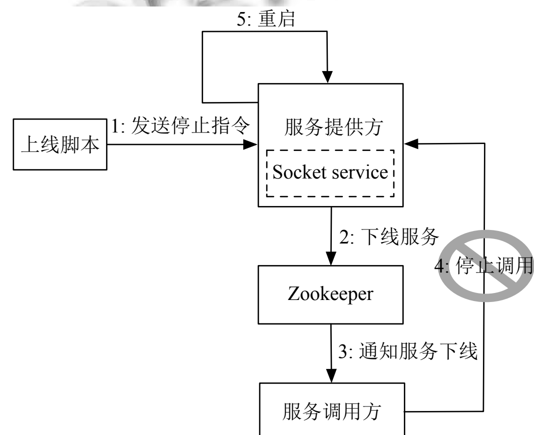

图 4　优雅停机流程

通过这样的流程, 避免了在应用重启或者上线过程中的请求失败, 提高服务质量. 在航旅微服务体系中, 主要通过在 Web 容器中添加 Socket 服务接收停止指令, 在收到停止指令后, 主动在 Zookeeper 上下线该服

© 中国科学院软件研究所　http://www.c-s-a.org.cn

务器对应的所有相关服务, 利用 Zookeeper 的实时通知机制, 通知到所有调用方停止向这台机器发送新的请求, 并保证处理完所有已经接收的请求后, 进行 Web 容器的重启.

1.3.3.2 流量预热

在 Web 容器启动结束之后, 端口已经暴露成功, 但是部分应用会进行基础数据的预热加载, 而且 JVM 也需要预热加载相关的类, 所以在 Web 容器启动后, 并不完全具备承载高并发请求的能力, 这种请求下如果有大量流量涌入, 轻则部分请求响应超时, 重则拖垮整个应用. 针对此类问题, 在航旅微服务体系中, 增加了启动预热模块, 针对某个服务的某台机器重启之后, 根据服务注册的时间, 前 5 分钟之内 (可在分布式配置中心进行动态调整), 服务调方会根据启动时间和这台机器的具体权重, 计算出一个相对合理的预热权重, *QPS* 会在 5 分钟内逐步递增到正常速度. 预热过程中的权重计算如式 (3) .

$$w = (now - registerTime) \div timeWindow \times weight \quad (3)$$

其中, *w* 代表预热权重, *now* 表示当前请求时间, *registerTime* 表示服务重新注册时间, *timeWindow* 表示预热时间窗口, *weight* 表示当前机器权重.

2 应用情况

通过本文研究的内容在航旅纵横中进行了实际应用, 通过近一年的时间, 对航旅纵横的 200 多个应用, 1000 多个服务质量的改造, 通过文中的研究成果提升服务的响应时间, 对协议调用链条中的响应时间、线程数进行计算和个性化设置, 更大程度的保障协议的成功率和系统稳定性.

如表 1 和表 2 所示, 针对航旅纵横 APP 中的 3 个核心业务航班动态、值机、行程改造前和改造后的数据进行了对比:

如表 1 和表 2 中数据所示, 改造前与改造后的数据相比, 3 个业务的服务可靠性大幅提升, 失败率降低了至少一个数量级, 绝大多数都降到了 0.001% 以下. 响应时间也有了较大的提高, 其中航班动态以信息查询为主, 由提供数据源的多个服务聚合而成, 改造后响平均响应时间和 95 线缩短了 1/2, 值机响应时间缩短接近 1/3.

表 1 改造前

业务	*QPS*	*avg* (ms)	95*Line*(ms)	失败率 (%)
航班动态	260	160	462	0.02
值机	59	215	785	0.03

表 2 改造后

业务	*QPS*	*avg* (ms)	95*Line*(ms)	失败率 (%)
航班动态	320	80	200	0.0001
值机	60	150	525	0.001

3 总结

本文在围绕微服务架构中的服务质量保障展开研究, 定义了服务质量的评价指标. 围绕评价指标, 提出了缩短响应时间以及提升服务可靠性的方法, 并在业务系统中进行了实际应用, 取得了不错的效果. 通过本文的研究内容, 形成了一套提升服务质量的标准方法, 研发人员只需要遵循该方法, 便可以开发出高质量的服务, 可以将更多的精力投入到实际业务的研发中, 大大提升了企业的研发效率.

参考文献

1 Conway ME. How do committees invent? Datamation, 1968, 14(4): 28–31.

2 Fowler M, Lewis J. Microservices: A definition of this new architectural term. https://www.martinfowler.com/articles/microservices.html, 2014-03-25.

3 Newman S. 微服务设计. 崔力强, 张骏, 译. 北京: 人民邮电出版社, 2016.

4 David B. Netflix hystrix-latency and fault tolerance for complex distributed systems. https://www.infoq.com/news/2012/12/netflix-hystrix-fault-tolerance. [2012-12-21].

5 崔解宾. 分布式内存缓存技术在数据处理平台中的研究与应用[硕士学位论文]. 北京: 北京邮电大学, 2015.

6 徐康明. 基于微服务架构的服务发现与服务可靠性的研究[硕士学位论文]. 北京: 北京邮电大学, 2018.

© 中国科学院软件研究所 http://www.c-s-a.org.cn