Java整体知识架构详解-之分布式架构

- 分布式架构设计
  - SOA架构和微服务架构
    - SOA架构
      - 面向服务的架构，依赖ESB企业级总线
    - 微服务架构
      - 是对SOA的升级，将业务系统彻底的服务化组件化，依赖于网关分配，充分利用协调资源
  - CAP理论
    - C（Consistency）：一致性，表示所有节点上的数据必须保持同步
    - A（Availability）：可用性，所有请求必定会得到响应，但不保证正确性
    - P（Partition tolerance）：分区容错性，系统应该持续提供服务，即使系统内部有服务挂了，不会彻底死机
    - cap理论指出一个分布式系统不可能同时满足一致性、可用性和分区容错性，这三个要求只能同时满足其中两项
  - BASE理论
    - 概述
      - BASE全称Basically Available（基本可用），Soft state（软状态）和Eventually consistent（最终一致性)
    - Basically Available（基本可用）
      - 在分布式系统出现故障时，允许瞬时部分可用性
        - 比如数据库分片，其中一片挂了，剩余仍可以使用
        - 比如当大流量访问时，只允许部分用户访问，其它用户降级处理
    - Soft state（软状态）
      - 表示数据存在中间状态，允许在不同节点的数据副本在同步过程中存在延迟
        - 比如支付过程，待支付，支付中，支付成功，支付失败，那么支付中就是个中间状态，在成功之后同步状态存在一定延迟
    - Eventually consistent（最终一致性）
      - 表示所有数据副本在一段时间的同步后最终达到一致的状态
  - 高可用设计
    - 避免单点故障
      - 负载均衡技术
        - failover快速失败
        - 选址DNS
        - 硬件负载
          - f5等
        - 软件负载

- lvs
- nginx
- HAProxy
- 去中心化的软件负载
  - gossip（redis-cluster使用）
  - gossip（redis-cluster使用）
- 热备（Linux HA）
- 多机房（同城灾备、异地灾备）

- 应用高可用
  - 故障监控，自动预警
    - CPU、内存监控
    - 链路监控
    - 日志监控
  - 应用容错设计、自我保护
    - 服务降级、限流、快速失败等
  - 数据量
    - 数据分片
    - 读写分离
- 可伸缩设计
  - 垂直伸缩
    - 比如业务垂直拆分成多个微服务，优先考虑
  - 水平伸缩
    - 比如数据库单表数据量过大，根据业务条件水平拆表到多个表
  - 提升硬件性能
  - 提升硬件性能
  - 增加服务器
- CDN
  - 加速静态内容访问速度
- 灰度发布
  - 应用的分批发布

- 分布式架构解决方案
  - 分布式锁
    - redis
      - 实现方式
        - 流行用setnx方法，设置成功则获取锁，redis可以设置过期时间
      - 缺点
        - 第一是在setnx方法之后，设过期时间之前redis宕机仍然可能死锁，且过期时间不好定；第二redis通常都是主从部署，可以看出redis是AP模型，而分布式锁需要实现CP模型，明显是不可实现的，具体表现在，如果在主节点A获取到了锁，此时还没同步到从节点B，主节点宕机，那么B从节点会变成主节点，此时其他服务是又可以获取分布式锁的

- zookeeper
  - 实现方式
    - zk通过添加同名节点成功者获取锁的方式实现，使用它的临时节点方式，可以在一个会话结束后自动释放节点，也就不存在需要定时释放的问题，哪怕程序异常断开了连接，节点没了会话维持也会自动失效
  - 缺点
    - zk的强一致性使得在zk节点越多的情况下，获取分布式锁越慢，且在存在leader机器宕机的时候，zk的ZAB一致性协议会使zk集群暂时无法提供服务，直到选出leader，并恢复数据
  - 优点
    - zk可以创建顺序节点，使用zk的watch机制，可以实现公平锁
- 数据库
  - 通过表主键判断，插入成功获取锁，缺点明显，数据库IO是比较消耗性能的，不可重入，不可定时释放

- 全局ID生成方案
  - snowflow
    - 优点
      - 速度快，直接内存操作生成
      - 不需要依赖第三方接口
      - 实现简单
    - 缺点
      - 只能趋势递增（考虑如果是订单，可以通过订单id来推测订单量）
      - 依赖机器时间，如果机器时间回拨可能导致生成id重复
  - Leaf美团点评分布式ID生成系统
    - Leaf-segment数据库方案
      - 每次去数据库获取一个号段的值，用完再去获取
    - Leaf-snowflake
      - 解决时钟回拨问题
  - uid-generator百度
    - 每次取回一批id自己用

- 调用链监控
  - 概述
    - 随着微服务调用链越来越复杂，急需一个调用链监控工具来进行监控和查找问题，google首先发布了一篇调用链论文Dapper，后续有人根据该论文推出了一系列开源项目
    - 随着开源调用链产品增多，为了多语言结合不同调用链产品的兼容性，一个开源组织对调用链进行了规范，称为OpenTracing协议
  - 方案
    - Dapper，google的调用链实现，未开源
    - zipkin
      - 开源的调用链实现工具，现在已是SpringCloud的一个组件，支持开源协议OpenTracing

- jaeger
  - 也是国外一个开源软件
- skywalking
  - 国内的开源组件，非侵入字节码增强方式
- pinpoint
  - 非侵入字节码增强方式进行调用链监控，不需要埋点
- cat
  - 点评开发的一款链路追踪，具有更多的表报聚合数据展示

- session共享
  - 使用nginx的ip_hash策略保证同一个用户始终访问同一个服务器
  - 使用数据库存放session，可以保证session持久化，不过在微服务情况下，各个微服务有自己的数据库这就不好用了
  - cookie存储session，这样每次请求都会携带session信息，消耗资源，不推荐
  - redis存储session，可以使用spring-session框架，它重写了httpSession

- 分布式架构中间件
  - 分布式消息通信
    - RabbitMQ
      - 四种通信模式
        - topic
          - 主题订阅模式，消息经由交换机通过routing key绑定到队列，需要注意**一条消息只能消费一次**
        - fanout
          - 广播模式，消息经由交换机直接发向和交换机绑定的所有队列，routing key无效
        - direct
          - 直连模式，这种模式也可以说是种默认模式，当你连交换机都懒得创建的时候，这种模式使用的是默认交换机，它和fanout模式的区别除了交换机不同，就是当向该交换机发送一条消息时，与它绑定的队列只有一个能消费到
        - header
          - 这种几乎用不到，它是通过请求信息中，请求头包含的信息进行消息区分
    - Kafka
      - 高吞吐低延迟mq，适用用大数据量的流式传输，如日志传输，用户活动跟踪，运营指标等
    - ActiveMQ
      - apache开源mq中间件，消息堆积能力不足，一般不考虑
    - RocketMQ
      - 阿里开源的mq中间件，支持事务
      - 组成
        - NameServer
          - 无状态节点，存储注册信息，可集群部署
        - Broker
          - 存储消息队列

- Producer
  - 生产者
- Consumer
  - 消费者

- Redis分布式缓存
  - 基本数据类型
    - string
      - 常用kv结构，用来缓存数据
    - hash
      - 将信息凝聚在一个key中，可以进行统一的删除，缺点是设置过期时间只能在key上，不能对key对应的数据设置过期；优点是避免了命名冲突，减少内存消耗
    - set
      - 无重复数据的集合，可用作抽奖，存放点赞的人，可以进行交集，并集，差集的计算；可以用来实现共同关注的人，可能认识的人等功能；甚至可以用来对账
    - zset
      - 有序的无重复数据集合，可以用在时事热点信息的排序上
    - list
      - 可以用作消息列表使用
  - memcache VS redis
    - memcache适合以KV的缓存为主，数据量，并发业务量大的场景，它支持多线程操作，充分利用CPU，但缓存不能持久化
    - Redis拥有丰富的数据类型，可以做到缓存持久化，宕机更容易数据恢复，在单线程的效率比memcache快
  - 部署三种模式
    - 单机
      - 适用于可以接受缓存穿透的场景，即去掉缓存也不影响业务正常进行的应用
    - 哨兵
      - 哨兵模式提供了主从的监控，当主服务宕机后，从服务能快速切换成主服务，适合缓存高可用场景，缺点是不易扩容
    - 集群
      - 适合于大数据量的高可用缓存场景，集群模式内存和QPS不受限于单机，可用于横向扩展
      - 通信采用gossip协议，请求逐步打到各个节点上，有一定延迟，但集群压力比较小
  - 持久化
    - 持久化类型
      - AOF
        - 备份方式
          - 几乎实时的备份，通常设置1秒备份一次
        - 恢复方式
          - 通过记录命令重复执行进行恢复数据，效率较低，但数据比较全

- RDB
  - 备份方式
    - 每隔一段时间用子线程进行全量备份
  - 恢复方式
    - 直接数据的传输覆盖，数据不全
- 过期策略
  - 定时删除
    - 概述
      - 对于设置了expire过期时间的key，到了key的过期时间会立即删除
    - 优点
      - 保证内存尽快释放，保证过期的准时性
    - 缺点
      - 若过期key很多，删除这些key会比较消耗CPU时间
      - 每设置一个过期时间会创建一个定时器，大量定时器会比较影响性能
  - 定期删除
    - 概述
      - 对于到期的key，不会立即删除，根据定期策略设置的时间，到了之后会检查key是否过期，若过期则删除
    - 优点
      - 通过限制删除操作的时长和频率，来减少对CPU时间的占用
    - 缺点
      - 会造成一定的内存占用
      - 会造成一定的内存占用
  - 惰性删除
    - 概述
      - 对过期的key不做删除处理，当再次通过key来获取值的时候，会检查key是否过期，过期则删除
    - 优点
      - 删除操作只发生在通过key取值的时候，占用CPU较少
    - 缺点
      - 可能比定期删除还要占用内存，若大量数据缓存而不再调用，可能导致内存溢出
      - 可能比定期删除还要占用内存，若大量数据缓存而不再调用，可能导致内存溢出
- 淘汰策略
  - volatile-lru
    - 从**已设置过期时间**的数据集中挑选**最近最少使用的数据淘汰**
  - volatile-ttl
    - 从**已设置过期时间**的数据集中挑选**将要过期的数据淘汰**
  - volatile-random
    - 从**时间已设置过期**的数据集中**任意选择数据淘汰**
  - allkeys-lru

- 从数据集中挑选**最近最少使用淘汰**
- allkeys-random
  - 从数据集中**任意选择数据淘汰**
- no-enviction
  - 不淘汰数据

- Redis Cluster数据分部算法Hash slot（虚拟桶）
  - 虚拟桶是一种取模和一致性哈希折中的办法，直接取模会导致数据和节点紧密关联，缺乏灵活扩展；一致性hash在扩容或缩容情况下，部分数据需要重新计算节点
  - 优点
    - 扩容直接分离一部分槽给新的机器就能达到扩容效果
  - 缺点
    - 槽个数是固定的，需要根据实际情况预先定下槽的数量，redis cluster槽数量是16384个
- 常见问题
  - 缓存穿透
    - 概述
      - 大量请求同一个数据，由于数据不存在，请求都访问数据库，导致数据库崩掉
    - 解决办法
      - 缓存不存在时，数据库返回null值也存入缓存，之后直接通过缓存返回null的处理办法
  - 缓存雪崩
    - 概述
      - 大量缓存在同一时间失效，请求都打到数据库，导致数据库崩掉
    - 解决办法
      - 并发压力大通过加锁或队列，当缓存失效时，对某个key只允许一条线程访问，其它等待
      - 并发压力大通过加锁或队列，当缓存失效时，对某个key只允许一条线程访问，其它等待
      - 缓存失效时间设置不同，可以在一个时间范围乘个随机数，尽量分布均匀
      - 加二级缓存，二级缓存失效时间大于一级缓存，当一级缓存失效，二级缓存可以起到作用
      - 如果能预计到某个时间点会有大量并发操作，可以设计手动reload缓存
  - 缓存预热
    - 系统启动时对热点数据进行缓存主动加载

- 数据存储
  - MySQL
    - mysql主从复制与读写分离
    - mysql+keepalived实现双主高可用
    - mysql分库分表
    - 数据库中间件Mycat
      - 分库分表利器

- zookeeper
  - 概述
    - zookeeper之所以流行，很大程度上得益于国内dubbo使用的是zookeeper作为其注册中心，然而zookeeper是个CP模型架构，我们的服务更多时候需要实现的是AP模型，也就是高可用，它是天然相冲的；由于这些原因，阿里现在也推出了nacos注册中心用来替代zookeeper成为dubbo的注册中心
  - 功能
    - 配置管理服务
    - 分布式协调中心
    - 分布式锁
    - 发布订阅
    - leader选举
  - 概念和使用详解
- nginx
- 配置中心
  - Spring Cloud Config
    - Spring Cloud 大家庭的一员，使用git托管配置文件，没有管理界面，若要实现自动更新需要加上mq、bus，实现复杂依赖多，还不好用（本人用过，后来弃了改用apollo^_^）
  - Apollo
    - 携程开发的配置管理中心，只依赖数据库，较为流行的配置中心
  - Nacos
    - 阿里最近加大力度推广的新开源配置中心，和dubbo配合较好，也支持SpringCloud，可能会是以后的流行趋势
  - disconf
    - 百度的开源配置中心，似乎逐渐没落，依赖较多
  - diamond
    - 淘宝曾经的配置中心，不维护了，pass