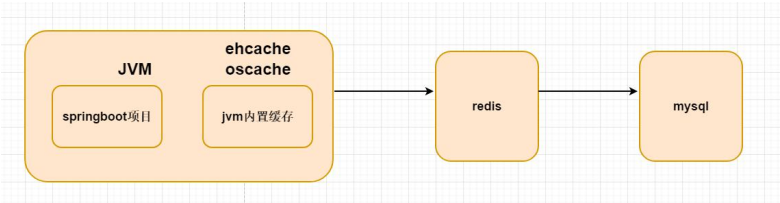
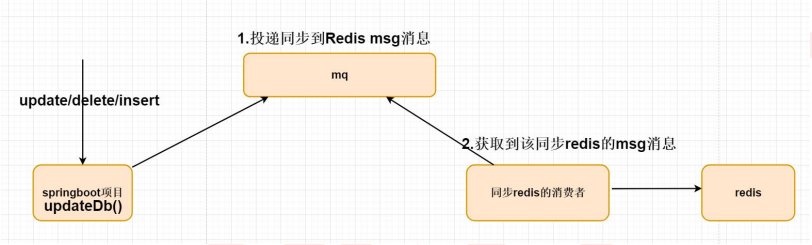
mysql 与 redis 如何保证数据一致性问题

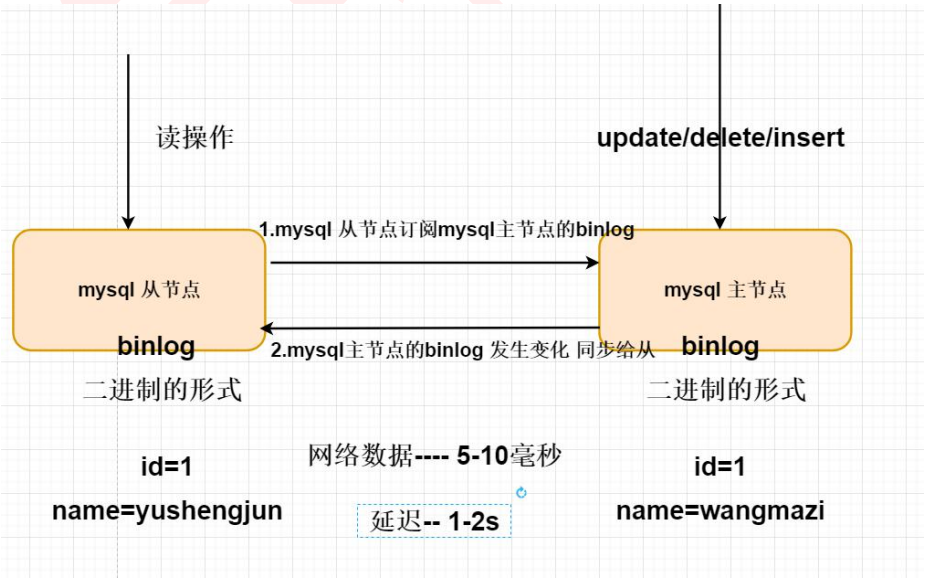
1. 更新 mysql 数据，在手动清除 Redis 缓存 ，在重新查询最新的数据同步到 Redis 中



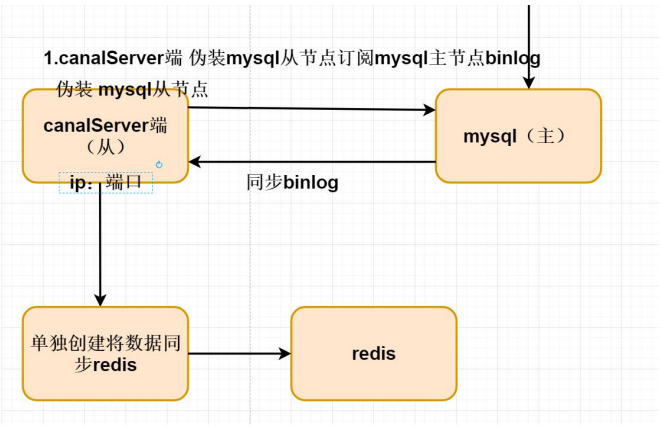
伪代码：
判断如果 redis 是为空的 则读取 mysql
同步到 redis 中
setRedis(key，mysql 数据)
2.更新 mysql 数据，在采用 mq 异步的形式 同步数据到 Redis 中；



缺点：延迟概率就比较大
优点：解耦
3. 基于订阅 mysql binlog 采用 mq 异步的形式将数据同步到 Redis 中；



4. 订阅 mysql binlog 文件 异步的形式同步到 redis 中（canal 框架）



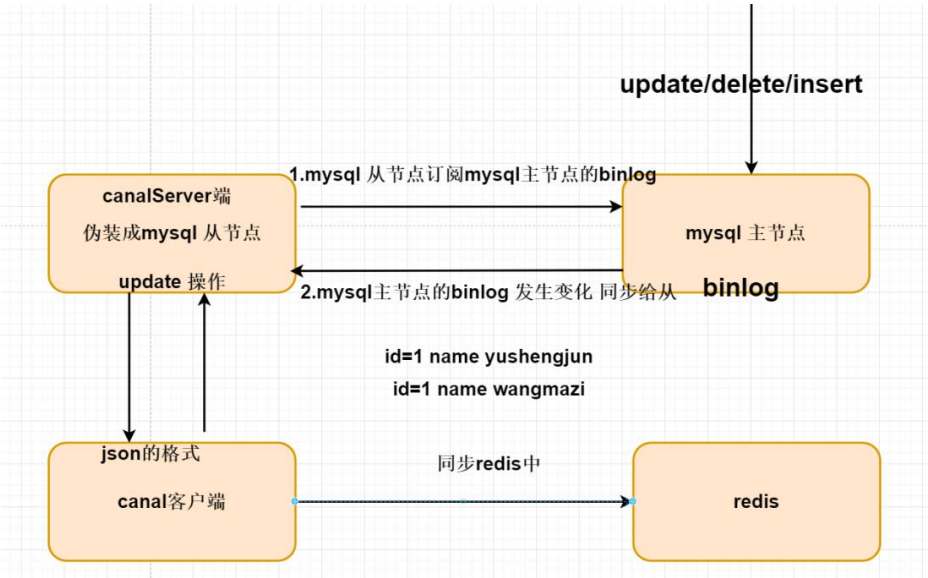
4. 延迟双删策略或者（双写）

mysql 与 redis 同步数据是否存在延迟呢？

数据同步过程中，会存在短暂的延迟，这属于正常的现象。
在分布式架构中很难实现数据强一致性
弱一致性： 主从之间数据允许不一致性；
强一致性： 主从之间数据必须一致性； 如果实现 成本是非常高，会设计到一些锁的技术，
最终一致性：短暂的数据延迟是允许的，但是最终数据是需要一致;
---在分布式中做数据同步需要经过网络传输的，网络传输数据需要一定的时间，所以数据短暂的延迟是允许的，但是最终数据一定达成一致。
延迟是很难避免的，优化 减少延迟的时间。
公司中数据 同步延迟 优化在 10-30 毫秒。

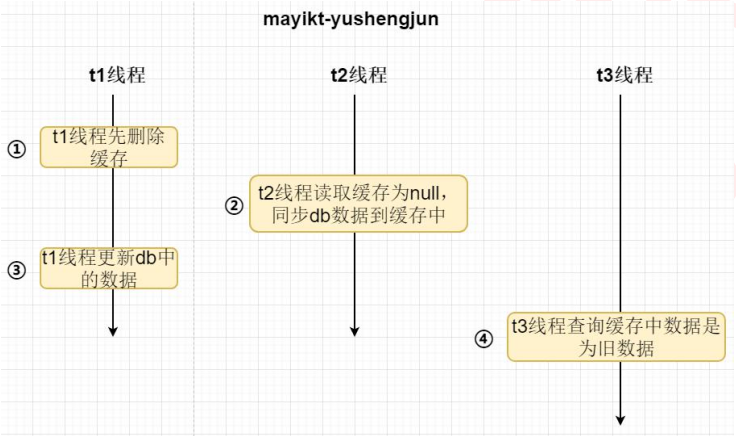
canal 解决 mysql 与 Redis 数据同步原理

1.canal 伪装成 mysql 从节点 订阅 mysql 主节点的 binlog 文件;
2. 当我们的 mysql 主节点 binlog 文件发生了变化，则将该 binlog 文件发送给 canal 服务器端;
3.canal 服务器端将该 binlog 文件二进制转化成 json 格式给 canal 客户端;
4.canal 客户端在将该数据同步到 Redis/ES;



说说延迟双删策略原理

1. 先删除缓存，在更新 DB 在高并发的情况下 其他线程可能读取缓存为 null 值，将 db 旧的数据在同步到缓存中。



1.t1 线程先删除缓存；
2.t2 线程读取缓存为 null，同步 db 数据到缓存中；
3.t1 线程更新 db 中的数据；
4.t3 线程查询缓存中数据是旧数据；
如何解决该问题：延迟双删策略
t1 线程更新完 DB 后，让它 sleep 一段时间，再删除缓存 ；

延迟双删缺点就是 第二次删除缓存时间很难控制---不推荐

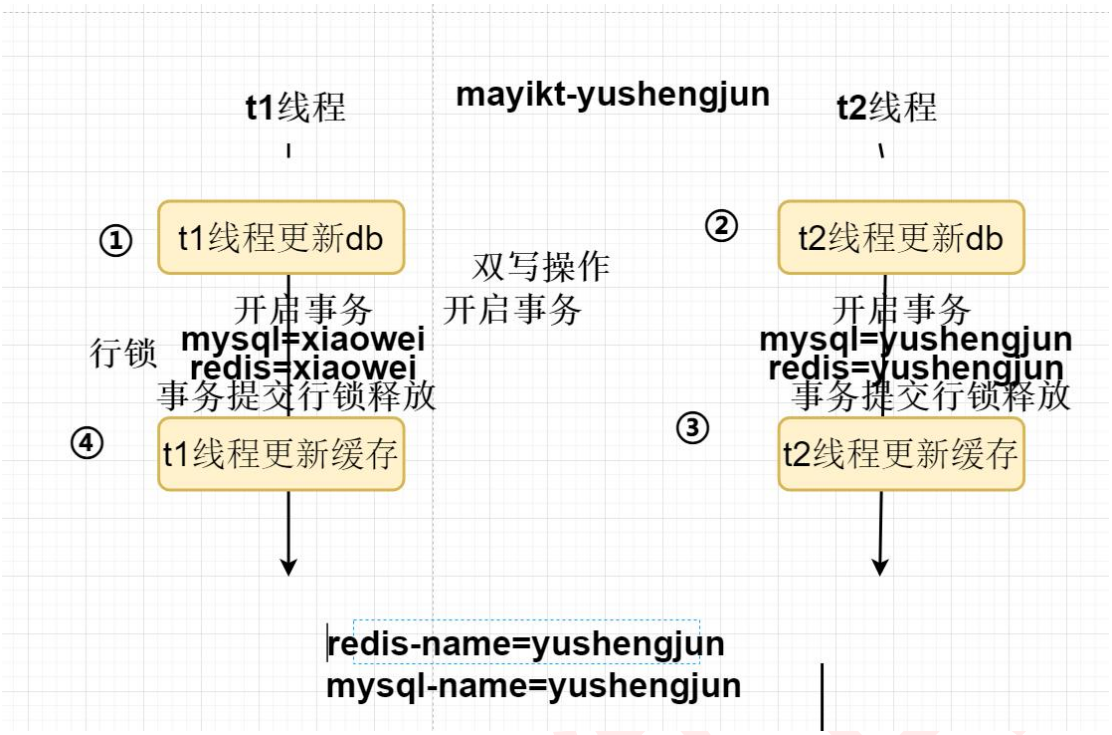
什么是如何双写一致性问题

1.更新完 db 后，同步更新 Redis； 更新 db 操作与更新 redis 操作 是一样的 不是 直接删除 Redis 缓存 key 俗称：双写

t1 和 t2 线程 同时对 db 数据做修改操作，同时需要更新 redis 缓存。
但是在同时更新 redis 缓存过程中 ，会存在顺序执行问题，导致数据不一致。

谈谈双写一致性的优缺点

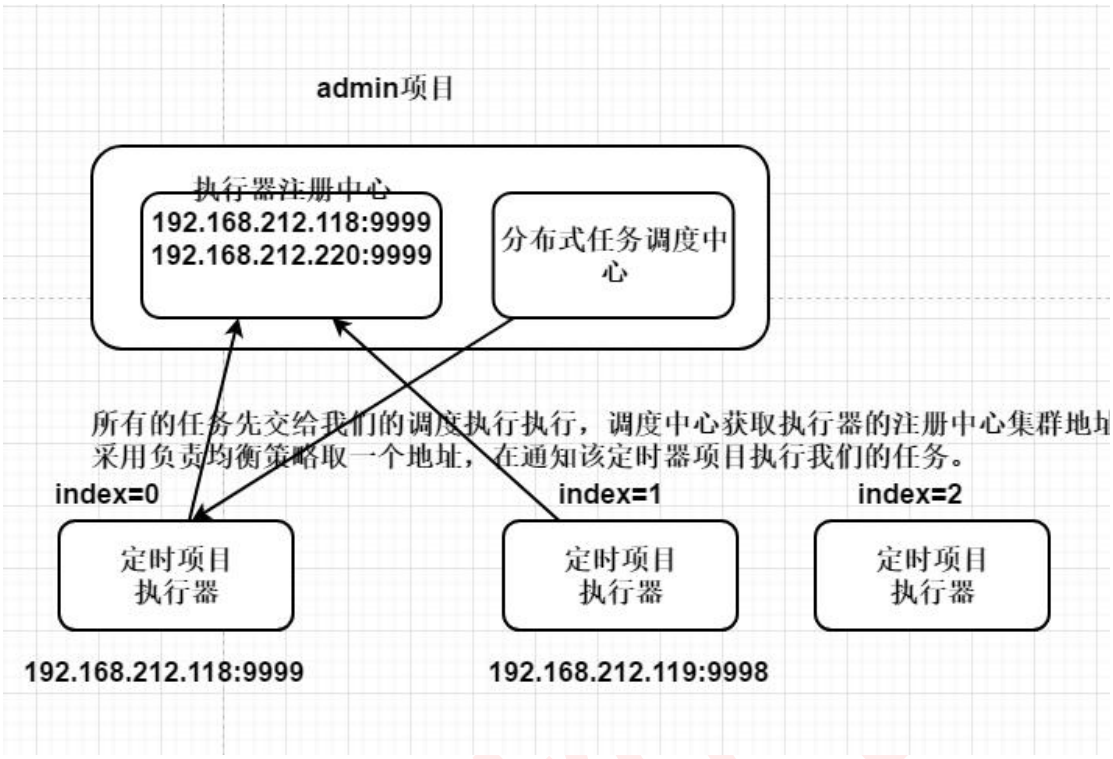
1.分布式锁解决多个线程同时执行双写业务逻辑，最终只会有一个获取到分布式锁线程才可以执行，没有获取到分布式锁线程则阻塞等待，这样确保线程执行双写 不会被其他线程干扰，但是效率非常低；
2.mysql 行锁实现 ，多个线程同时获取行锁，最终只会有一个线程获取行锁成功，
获取行锁成功的线程 如果更新 Redis 成功，就可以提交事务，如果更新 redis 失败
可以采用重试策略，重试多次更新到 redis 还是失败的话，直接回滚事务，同时释放行锁，行锁-----提交或者回滚事务。 但是效率非常低；不推荐



如何在集群中，保证我们的定时任务只会触发一次

1.将业务逻辑和定时任务逻辑完全分开部署，实现解耦、只对业务逻辑实现集群，不对我们的定时任务逻辑集群；---定时任务单机版本 缺点无法实现高可用的问题；
2.对我们 Jar 包加上一个开关，项目启动的时候读取该开关 如果为 true 的情况下则加载定时任务类，否则情况下就不加载该定时任务类；--效率非常低；
3.在数据库加上一个主键能够创建成功，则触发定时任务，否则就不触发定时任务， 高可用的问题；
4.分布式锁实实现，只要 jar 能够拿到分布式锁就能够执行定时任务，否则情况下不执行；
以上的方案都是属于规模比较小的项目，在微服务架构中应该采用分布式任务调度平台。
定时任务可以集群的执行 但是 重要条件。
每个定时任务 跑批数据范围 是不一样 ---可以跑批效率。

分布式任务调度平台架构原理



将定时任务代码与业务代码分开部署。
1.定时任务代码集群运行，注册到注册中心上，在通过调度中心触发定时任务，先去注册中心获取定时任务接口，在通过负载均衡算法 发送请求给具体定时任务执行定时任务。
2. 如果我们定时还是单机版本，执行定时任务效率很低，所有我们的定时还是会集群执行只是通过分片的方式。 也就是调度中心会给定时任务发送请求传递一个不同的值 例如定时任务 1 为 index 0 定时任务 2 为 index=1 定时任务 3 为 index=3 实际该 index 值就是定时任务注册到注册中心上存放的 index 位置，定时任务在根据不同的 index 值查询自己需要跑批的数据。

Xxljob 与 elastic-job

Xxljob 自己内部注册注册中心，而 elastic-job 是基于 zk 实现注册中心
整体思想是差不多的。

元原生架构、ddd 服务网格
Springboot 3.0 依赖 jdk？ 17
2025-2028

分布式架构---SOA 服务---微服务
----云原生 jdk17 jdk8 jdk25-30