Redis 如何保持和 MySQL 数据一致

1.MySQL持久化数据，Redis只读数据

redis在启动之后，从数据库加载数据。

读请求：

不要求强一致性的读请求，走redis，要求强一致性的直接从mysql读取

写请求：

数据首先都写到数据库，之后更新redis（先写redis再写mysql，如果写入失败事务回滚会造成redis中存在脏数据）

2.MySQL和Redis处理不同的数据类型

- MySQL处理实时性数据，例如金融数据、交易数据
- Redis处理实时性要求不高的数据，例如网站最热贴排行榜，好友列表等

在并发不高的情况下，读操作优先读取redis，不存在的话就去访问MySQL，并把读到的数据写回Redis中；写操作的话，直接写MySQL，成功后再写入Redis(可以在MySQL端定义CRUD触发器，在触发CRUD操作后写数据到Redis，也可以在Redis端解析binlog，再做相应的操作)

在并发高的情况下，读操作和上面一样，写操作是异步写，写入Redis后直接返回，然后定期写入MySQL

几个例子：

当更新数据时，如更新某商品的库存，当前商品的库存是100，现在要更新为99，先更新数据库更改成99，然后删除缓存，发现删除缓存失败了，这意味着数据库存的是99，而缓存是100，这导致数据库和缓存不一致。

解决方法：

这种情况应该是先删除缓存，然后在更新数据库，如果删除缓存失败，那就不要更新数据库，如果说删除缓存成功，而更新数据库失败，那查询的时候只是从数据库里查了旧的数据而已，这样就能保持数据库与缓存的一致性。

为什么是删除缓存，而不是更新缓存?

原因很简单，很多时候，在复杂点的缓存场景，缓存不单单是数据库中直接取出来的值。

比如可能更新了某个表的一个字段，然后其对应的缓存，是需要查询另外两个表的数据并进行运算，才能计算出缓存最新的值的。

另外更新缓存的代价有时候是很高的。是不是说，每次修改数据库的时候，都一定要将其对应的缓存更新一份？也许有的场景是这样，但是对于**比较复杂的缓存数据计算的场景**，就不是这样了。如果你频繁修改一个缓存涉及的多个表，缓存也频繁更新。但是问题在于，**这个缓存到底会不会被频繁访问到?**

举个栗子，一个缓存涉及的表的字段，在 1 分钟内就修改了 20 次，或者是 100 次，那么缓存更新 20 次、100 次；但是这个缓存在 1 分钟内只被读取了 1 次，有**大量的冷数据**。实际上，如果你只是删除缓存的话，那么在 1 分钟内，这个缓存不过就重新计算一次而已，开销大幅度降低。**用到缓存才去算缓存。**

其实删除缓存，而不是更新缓存，就是一个 lazy 计算的思想，不要每次都重新做复杂的计算，不管它会不会用到，而是让它到需要被使用的时候再重新计算。像 mybatis，hibernate，都有懒加载思想。查询一个部门，部门带了一个员工的 list，没有必要说每次查询部门，都里面的 1000 个员工的数据也同时查出来啊。80% 的情况，查这个部门，就只是要访问这个部门的信息就可以了。先查部门，同时要访问里面的员工，那么这个时候只有在你要访问里面的员工的时候，才会去数据库里面查询 1000 个员工。

在高并发的情况下，如果当删除完缓存的时候，这时去更新数据库，但还没有更新完，另外一个请求来查询数据，发现缓存里没有，就去数据库里查，还是以上面商品库存为例，如果数据库中产品的库存是100，那么查询到的库存是100，然后插入缓存，插入完缓存后，原来那个更新数据库的线程把数据库更新为了99，导致数据库与缓存不一致的情况

解决方法：

遇到这种情况，可以用队列的去解决这个问，创建几个队列，如20个，根据商品的ID去做hash值，然后对队列个数取摸，当有数据更新请求时，先把它丢到队列里去，当更新完后在从队列里去除，如果在更新的过程中，遇到以上场景，先去缓存里看下有没有数据，如果没有，可以先去队列里看是否有相同商品ID在做更新，如果有也把查询的请求发送到队列里去，然后同步等待缓存更新完成。

这里有一个优化点，如果发现队列里有一个查询请求了，那么就不要放新的查询操作进去了，用一个while（true）循环去查询缓存，循环个200MS左右，如果缓存里还没有则直接取数据库的旧数据，一般情况下是可以取到的。

在数据还未同步到「从库」的时候，由于cache miss去「从库」取到了未同步前的旧值

当「从库」完成同步的时候再额外做一次delete cache或者set cache的操作。

如此，虽说也没有100%解决短暂的数据不一致问题，但是已经将脏数据所存在的时长降到了最低（最终由主从同步的耗时决定），并且大大减少了无谓的资源消耗。

在高并发下要注意的问题：

1、读请求时长阻塞

由于读请求进行了非常轻度的异步化，所以一定要注意读超时的问题，每个读请求必须在超时时间范围内返回。

该解决方案，最大的风险点在于说，**可能数据更新很频繁**，导致队列中积压了大量更新操作在里面，然后**读请求会发生大量的超时**，最后导致大量的请求直接走数据库。务必通过一些模拟真实的测试，看看更新数据的频率是怎样的。

另外一点，因为一个队列中，可能会积压针对多个数据项的更新操作，因此需要根据自己的业务情况进行测试，可能需要**部署多个服务**，每个服务分摊一些数据的更新操作。如果一个内存队列里居然会挤压 100 个商品的库存修改操作，每隔库存修改操作要耗费 10ms 去完成，那么最后一个商品的读请求，可能等待 10 * 100 = 1000ms = 1s 后，才能得到数据，这个时候就导致**读请求的长时阻塞。**

一定要做根据实际业务系统的运行情况，去进行一些压力测试，和模拟线上环境，去看看最繁忙的时候，内存队列可能会挤压多少更新操作，可能会导致最后一个更新操作对应的读请求，会 hang 多少时间，如果读请求在 200ms 返回，如果你计算过后，哪怕是最繁忙的时候，积压 10 个更新操作，最多等待 200ms，那还可以的。

如果一个内存队列中可能积压的更新操作特别多，那么你就要**加机器**，让每个机器上部署的服务实例处理更少的数据，那么每个内存队列中积压的更新操作就会越少。

其实根据之前的项目经验，一般来说，数据的写频率是很低的，因此实际上正常来说，在队列中积压的更新操作应该是很少的。像这种针对读高并发、读缓存架构的项目，一般来说写请求是非常少的，每秒的 QPS 能到几百就不错了。

我们来**实际粗略测算一下**。

如果一秒有 500 的写操作，如果分成 5 个时间片，每 200ms 就 100 个写操作，放到 20 个内存队列中，每个内存队列，可能就积压 5 个写操作。每个写操作性能测试后，一般是在 20ms 左右就完成，那么针对每个内存队列的数据的读请求，也就最多 hang 一会儿，200ms 以内肯定能返回了。

经过刚才简单的测算，我们知道，单机支撑的写 QPS 在几百是没问题的，如果写 QPS 扩大了 10 倍，那么就扩容机器，扩容 10 倍的机器，每个机器 20 个队列。

2、请求并发量过高

这里还必须做好压力测试，确保恰巧碰上上述情况的时候，还有一个风险，就是突然间大量读请求会在几十毫秒的延时 hang 在服务上，看服务能不能扛的住，需要多少机器才能扛住最大的极限情况的峰值。

但是因为并不是所有的数据都在同一时间更新，缓存也不会同一时间失效，所以每次可能也就是少数数据的缓存失效了，然后那些数据对应的读请求过来，并发量应该也不会特别大。

3、多服务实例部署的请求路由

可能这个服务部署了多个实例，那么必须**保证**说，执行数据更新操作，以及执行缓存更新操作的请求，都通过 Nginx 服务器**路由到相同的服务实例上**。

比如说，对同一个商品的读写请求，全部路由到同一台机器上。可以自己去做服务间的按照某个请求参数的 hash 路由，也可以用 Nginx 的 hash 路由功能等等。可能这个服务部署了多个实例，那么必须保证说，执行数据更新操作，以及执行缓存更新操作的请求，都通过nginx服务器路由到相同的服务实例上

4、热点商品的路由问题，导致请求的倾斜

万一某个商品的读写请求特别高，全部打到相同的机器的相同的队列里面去了，可能会造成某台机器的压力过大。就是说，因为只有在商品数据更新的时候才会清空缓存，然后才会导致读写并发，所以其实要根据业务系统去看，如果更新频率不是太高的话，这个问题的影响并不是特别大，但是的确可能某些机器的负载会高一些。