缓存与数据库不一致了，咋办？

华仔聊技术 2022-01-04 08:00

编者荐语：

维护者是19届双非本科毕业，曾在百度、携程、华为等大厂搬砖，专注Java生态各种业务及框架原理和实战，全是硬核干货。关注后，后台回复“面试”即可获取多年珍藏大厂求职大礼包。

以下文章来源于JavaEdge ，作者JavaEdge

JavaEdge

曾就职于百度、携程、华为等大厂，六年Java开发经验。CSDN博客专家，阿里云社区专家博主，华...

👇推荐大家关注一个公众号👇

JavaEdge

曾就职于百度、携程、华为等大厂，六年Java开发经验。CSDN博客专家，阿里云社区专家博主，华...

159篇原创内容

公众号

点击上方 "**JavaEdge**"关注, 星标或置顶一起成长

后台回复“**面试**”有惊喜礼包！

> 这是一个纷杂而无规则的世界，越想忘掉的事情，越难忘记。

正文

我们这行都很幽默，总说编程就是CV，自黑写程序大部分都是靠复制粘贴。实际上，

0 前言

只要使用Redis缓存，就必然存在缓存和DB数据一致性问题。若数据不一致，则业务应用从缓存读取的数据就不是最新数据，可能导致严重错误。如将商品库存缓存在Redis，若库存数量不对，则下单时就可能出错，这是难以接受的。

1 什么是缓存和DB的数据一致性?

一致性包含如下情况：

- 缓存有数据缓存的数据值需和DB相同
- 缓存无数据DB是最新值

不符合这两种情况的，都属于缓存和DB数据不一致。

2 缓存的读写模式

根据是否接收写请求，可将缓存分成读写缓存和只读缓存。

2.1 读写缓存

若要对数据进行增删改，需要在Cache进行。同时根据采取的写回策略，决定是否同步写回DB：

2.1.1 同步直写

写缓存时，也同步写数据库，缓存和数据库中的数据一致。

2.1.2 异步写回

写缓存时不同步写DB，等到数据从缓存中淘汰时，再写回DB。使用这种策略时，若数据还没有写回DB，缓存就发生故障，则此时，DB就没有最新数据了。

所以，对于读写缓存，要想保证缓存和DB数据一致，就要采用同步直写。若采用这种策略，就需同时更新缓存和DB。

所以，要在业务代码中使用事务，保证缓存和DB更新的原子性，即两者：

- 要么一起更新
- 要么都不更新，返回错误信息，进行重试

否则，我们无法实现同步直写。

有些场景下，我们对数据一致性要求不高，比如缓存的是电商商品的非关键属性或短视频的创建或修改时间等，则可以使用异步写回。

2.2 只读缓存

- 新增数据直接写DB
- 删改数据删改DB，删除只读缓存中的数据

这样应用后续再访问这些增删改的数据时，由于Cache无数据 =》缓存缺失。此时，再从DB把数据读入Cache，这样后续再访问数据时，直接读Cache。

下面我们针对只读缓存，看看具体会遇到哪些问题，又该如何解决。

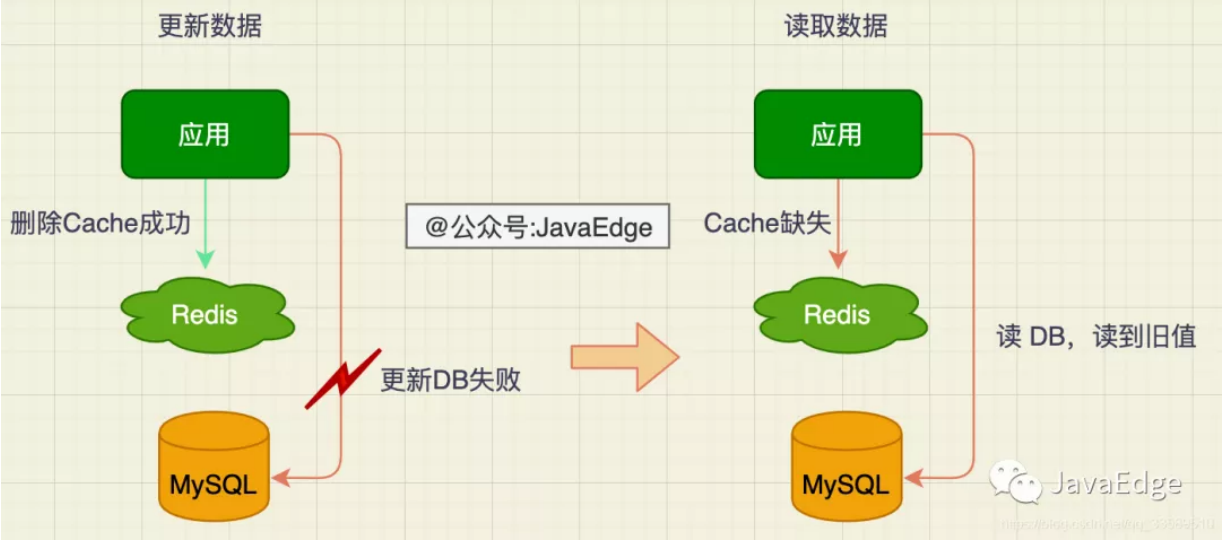
3 新增数据

数据直接写到DB，不操作Cache。此时，Cache本身无新增数据，而DB是最新值，所以，此时缓存和DB数据一致。

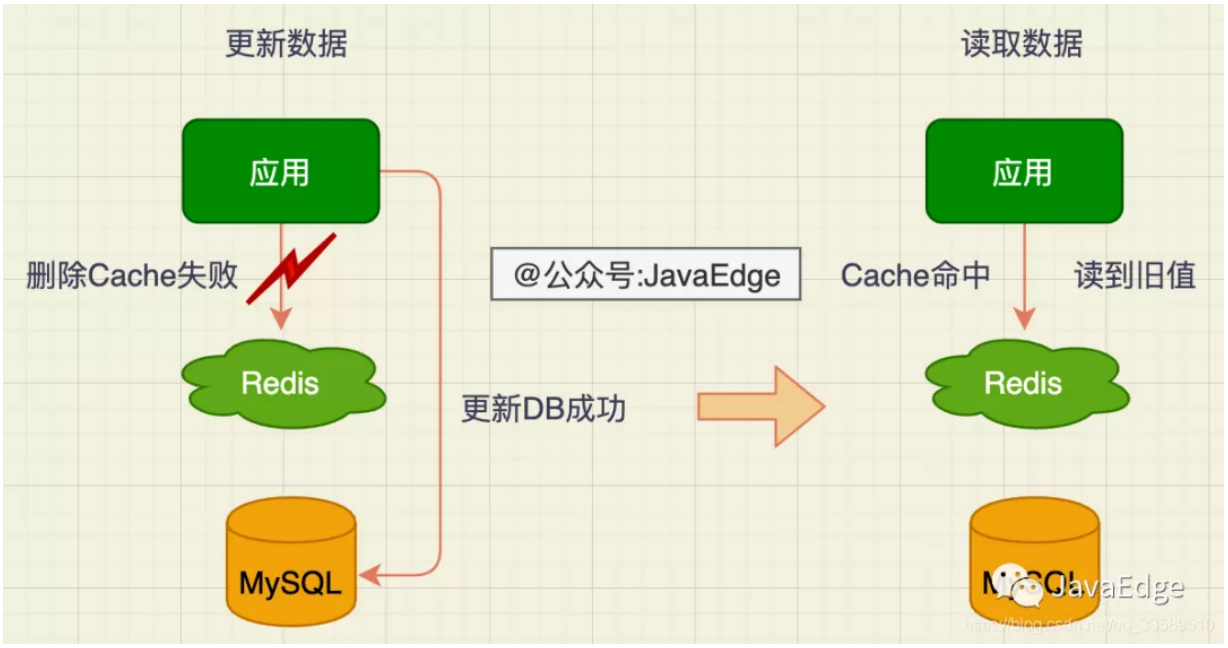
4 删改数据

此时应用既要更新DB，也要删除Cache。这俩操作若无法保证原子性，就可能出现数据不一致。

4.1 先删Cache，再更新DB



4.2 先更新DB，再删除Cache



综上，在更新DB和删除Cache时，无论这俩操作谁先执行，只要有一个操作失败了，就会导致客户端读到旧值。

那怎么办？好像怎么都会导致数据不一致？

5 数据不一致的解决方案

5.1 无并发

重试，将：

- 要删除的Cache值
- 或要更新的DB值

暂存到MQ。

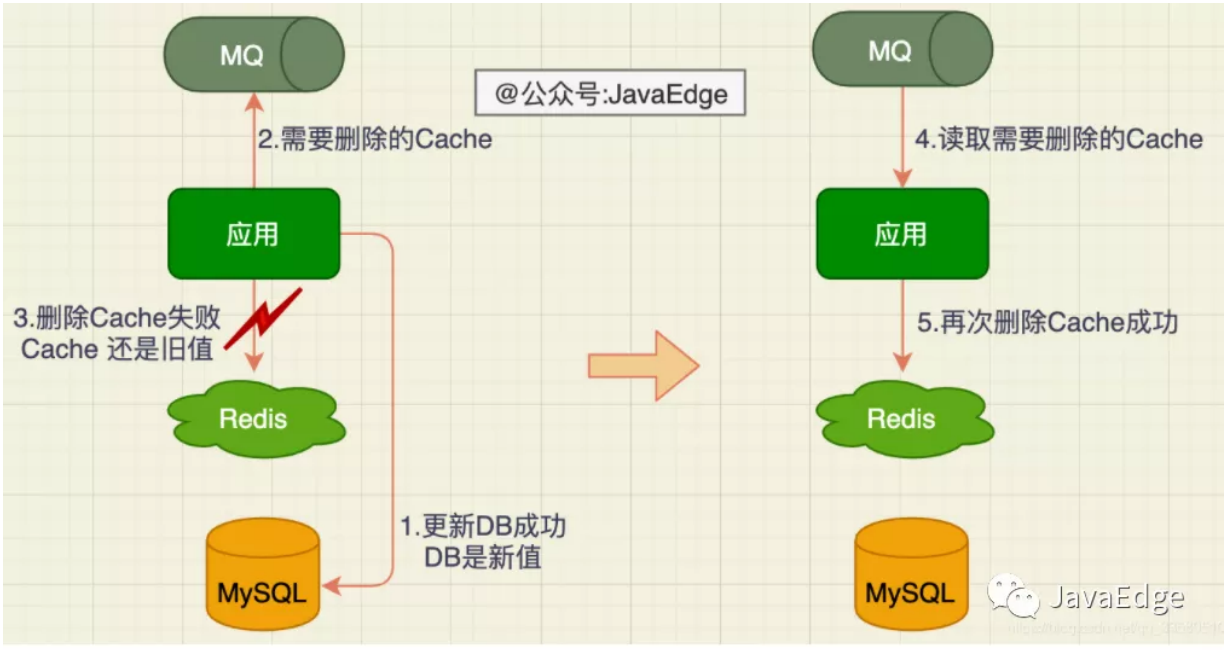
当应用删除Cache或更新DB：

- 成功把这些值从MQ去除，避免重复操作，这时即可保证DB、Cache数据一致性。
- 失败重试。从MQ重新读取这些值，然后再次进行删除或更新。若重试超过一定次数，还没成功，就向业务层发送报错信息。

在更新数据库和删除缓存值的过程中，其中一个操作失败了：

先更新DB，再删除缓存

- 若删除缓存失败，再次重试后删除成功



其它情况不再赘述。

即使这两个操作第一次执行时都没有失败，当有大量并发请求时，应用还是有可能读到不一致的数据。按不同的删除和更新顺序，分成两种情况来看

5.2 高并发

5.2.1 先删除Cache，再更新DB

假设条件：

- 时刻t1< t2 < t3
- 线程 T1、T2

	T1 更新操作	T2 查询操作	影响
t1	删除缓存X的缓存值		缓存X为空
t2		1. 读缓存，未命中 于是从DB读X，读到旧值 2.把读到的数据X的旧值写入Cache	1.T1尚未更新 DB，导致 T2 读到旧值 2.T2把旧值写入Cache，导致其它线程可能读到旧值
t3	更新DB中的X		Cache中是旧值，DB 是新值，最终二者不一致

这咋办？可以考虑如下解决方案：

延迟双删

T1更新完DB后，让它sleep一段时间，再删除Cache。

为何要 sleep 一段时间?

让T2能先从DB读数据，再把缺失数据写入Cache，然后，T1再进行删除。所以，T1 sleep的时间，就要大于T2读取数据再写入Cache的时间。

sleep 时间如何确定?

业务程序运行时，统计一下线程读数据和写缓存的操作时间，以此估算。确保读请求结束，写请求可删除读请求造成的缓存脏数据。

该策略还要考虑 Cache 和 DB 主从同步的耗时。最后写数据的休眠时间：则在读数据业务逻辑的耗时的基础上，加上几百ms即可。

这样，当其它线程读数据时，会发现Cache未命中，所以从DB读最新值。因为该方案会在第一次删除Cache后，延迟一段时间再删除，所以叫“延迟双删”。

```
cache.delKey(X)
db.update(X)
Thread.sleep(N)
cache.delKey(X)
```

设置缓存TTL

设置缓存TTL，是保证最终一致性的解决方案。所有写操作以DB为准，只要到达缓存TTL，则后面的读请求自然都会从DB读最新值，然后回填缓存。

结合【双删策略】+【缓存TTL设置】，这样最差情况就是在TTL时间内数据存在不一致，而且又增加写请求耗时。

该方案的缺点

操作完DB后，由于某原因删除Cache失败，此时可能出现数据不一致，需提供重试补偿方案：

方案一

1. 更新DB
2. Cache因某异常，删除失败（问题点）
3. 将待删除的K发送至MQ
4. 自己消费消息，获得待删除K
5. 重试删除操作，直到成功（解决问题）

该方案有个缺点，对业务代码侵入性太强，于是有方案二。方案二中，启动一个订阅程序去订阅DB的binlog，获得待操作的数据。在应用程序中，另起一段程序，获得这个订阅程序传来的信息，执行删除Cache操作。

方案二

1. 更新DB数据
2. DB会将操作信息写入binlog日志
3. 订阅程序提取出所需要的数据及K
4. 另起一段非业务代码，获得该信息
5. 尝试删除Cache操作，发现删除失败
6. 将这些信息发送至MQ
7. 重新从MQ获得该数据，重试删除操作

以上方案都是在业务中经常会碰到的场景，可据业务对数据一致性的要求选择具体方案。

5.2.2 先更新DB，再删除Cache

	T1	T2	问题
t1	删除 DB 的数据 X		
t2		读数据X，Cache命中， 从Cache读X，读到旧值	T1 尚未删除 Cache 导致 T2 读到 Cache 旧值
t3	删除 Cache的数据 X		

这种情况下，如果其他线程并发读缓存的请求不多，那么，就不会有很多请求读取到旧值。而且，线程A一般也会很快删除缓存值，这样一来，其他线程再次读取时，就会发生缓存缺失，进而从数据库中读取最新值。所以，这种情况对业务的影响较小。

至此，Cache和DB数据不一致的原因也都有了对应解决方案。

- 删除Cache或更新DB失败而导致数据不一致重试，确保删除或更新成功
- 在删除Cache、更新DB这两步操作中，有其他线程的并发读操作，导致其他线程读取到旧值延迟双删

绝大多数场景都会将Redis作为只读缓存：

- 既可以先删除缓存值再更新数据库
- 也可以先更新数据库再删除缓存

推荐优先使用先更新数据库再删除缓存：

- 先删除缓存值再更新数据库，有可能导致请求因缓存缺失而访问数据库，给数据库带来压力
- 如果业务应用中读取数据库和写缓存的时间不好估算，那么，延迟双删中的等待时间就不好设置

不过，当使用先更新数据库再删除缓存时，也有个地方需要注意，如果业务层要求必须读取一致的数据，那么，我们就需要在更新数据库时，先在Redis缓存客户端暂存并发读请求，等数据库更新完、缓存值删除后，再读取数据，从而保证数据一致性。

6 直接更新 Cache

在只读缓存中进行数据的删改操作时，需要在缓存中删除相应的缓存值。若此过程不是删除缓存，而是直接更新缓存，效果如何?

这种情况相当于把Redis当做读写缓存使用，删改操作同时操作DB、Cache。

6.1 无并发

先更新数据库，再更新缓存

若更新DB成功，但Cache更新失败，此时DB最新值，但缓存旧值，后续读请求会直接命中缓存，得到旧值。

先更新缓存，再更新数据库

如果更新缓存成功，但数据库更新失败：

- 缓存中是最新值
- 数据库中是旧值

后续读请求会直接命中缓存，但得到的是最新值，短期对业务影响不大。但一旦缓存过期或满容后被淘汰，读请求就会从数据库中重新加载旧值到缓存中，之后的读请求会从缓存中得到旧值，对业务产生影响。

针对这种其中一个操作可能失败的情况，类似只读缓存方案，也可使用重试。把第二步操作放入到MQ中，消费者从MQ取出消息，再更新缓存或数据库，成功后把消息从消息队列删除，否则进行重试，以此达到数据库和缓存的最终一致。

6.2 并发读写

也会产生不一致，分为以下4种双写场景。

> 双写模式下，更新DB有返回值，更新Redis的操作可放到更新DB返回后进行，通过数据库的行锁机制，可以避免更新DB是线程A，B，但更新Redis是线程B，A的情况。

先更新数据库，再更新缓存

写+读并发。线程A先更新数据库，之后线程B读取数据，此时线程B会命中缓存，读取到旧值，之后线程A更新缓存成功，后续的读请求会命中缓存得到最新值。

这时，线程A未更新完缓存之前，在这期间的读请求会短暂读到旧值，对业务短暂影响。

先更新缓存，再更新数据库

写+读并发。线程A先更新缓存成功，之后线程B读取数据，此时线程B命中缓存，读取到最新值后返回，之后线程A更新数据库成功。这种场景下，虽然线程A还未更新完数据库，数据库会与缓存存在短暂不一致，但在这之前进来的读请求都能直接命中缓存，获取到最新值，所以对业务没影响。

先更新数据库，再更新缓存

写+写并发。线程A和线程B同时更新同一条数据，更新数据库的顺序是先A后B，但更新缓存时顺序是先B后A，这会导致数据库和缓存的不一致。

先更新缓存，再更新数据库

写+写并发。与场景3类似，线程A和线程B同时更新同一条数据，更新缓存的顺序是先A后B，但是更新数据库的顺序是先B后A，这也会导致数据库和缓存的不一致。

场景1和2对业务影响较小，场景3和4会造成数据库和缓存不一致，影响较大。即读写缓存下，写+读并发对业务的影响较小，而写+写并发时，会造成数据库和缓存的不一致。

针对场景3、4解决方案：对于写请求，配合分布式锁。写请求进来时，针对同一资源的修改操作，先加分布式锁，这样同一时间只允许一个线程去更新DB和Cache，没有拿到锁的线程把操作放入到MQ，延时处理。这样保证多个线程操作同一资源的顺序性，以此保证一致性。

综上，使用读写缓存同时操作数据库和缓存时，因为其中一个操作失败导致不一致的问题，同样可以通过MQ重试解决。而在并发的场景下，读+写并发对业务没有影响或者影响较小，而写+写并发时需要配合分布式锁的使用，才能保证缓存和数据库的一致性。

另外，读写缓存模式由于会同时更新数据库和缓存：

- 优点缓存一直会有数据。若更新后立即访问，可直接命中缓存，能降低读请求对DB的压力（没有只读缓存的删除缓存导致缓存缺失和再加载的过程）

- 缺点若更新后的数据，之后很少再被访问到，会导致缓存中保留的不是最热数据，缓存利用率不高（只读缓存中保留的都是热数据）

所以读写缓存比较适合用于读写相当的业务场景。

7 总结

快为你的项目选择一套合适的方案吧~

欢迎加入后端**交流群**，关注本公众号添加我本人微信，邀请进群。

END

最近在准备面试BAT，特地整理了一份面试资料，覆盖Java核心技术、JVM、Java并发、SSM、微服务、数据库、数据结构等等。在这里，我为大家准备了一份2021年最新最全的互联网大厂Java面试经验总结。


7. 微服务

7.1.1. 服务注册发现

服务注册就是维护一个登记簿，它管理系统内所有的服务地址。当新的服务启动后，它会向登记簿交待自己的地址信息。服务的依赖方直接向登记簿要 Service Provider 地址就行了。当下用于服务注册的工具非常多 ZooKeeper，Consul，Etcd，还有 Netflix 家的 eureka 等。服务注册有两种形式：客户端注册和第三方注册。

7.1.1.1. 客户端注册（zookeeper）

客户端注册是服务自身要负责注册与注销的工作。当服务启动后向注册中心注册自身，当服务下线时注销自己。期间还需要和注册中心保持心跳。心跳不一定要客户端来做，也可以由注册中心负责（这个过程叫探活）。这种方式的缺点是注册工作与服务耦合在一起，不同语言都要实现一套注册逻辑。