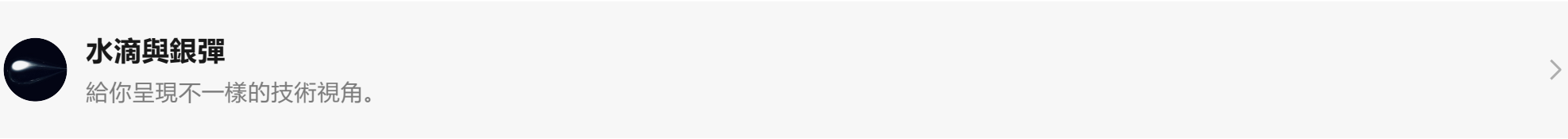
緩存和數據庫一致性問題，看這篇就夠了

三太子敖丙 今天

以下文章來源於水滴與銀彈



> 閱讀本文大約需要10 分鐘。

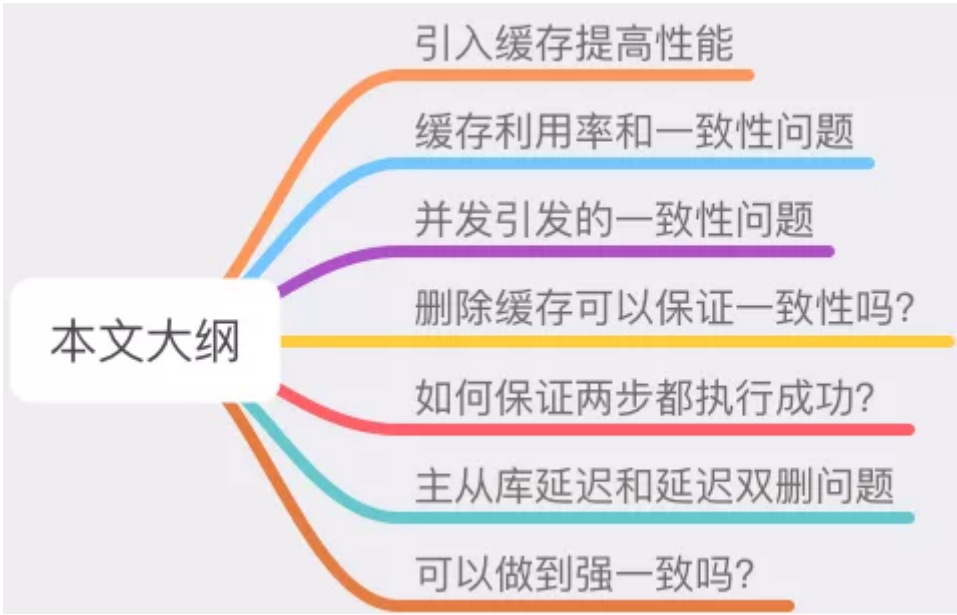
如何保證緩存和數據庫一致性，這是一個老生常談的話題了。

但很多人對這個問題，依舊有很多疑惑：

- 到底是更新緩存還是刪緩存？
- 到底選擇先更新數據庫，再刪除緩存，還是先刪除緩存，再更新數據庫？
- 為什麼要引入消息隊列保證一致性？
- 延遲雙刪會有什麼問題？到底要不要用？
- ...

這篇文章，我們就來把這些問題講清楚。

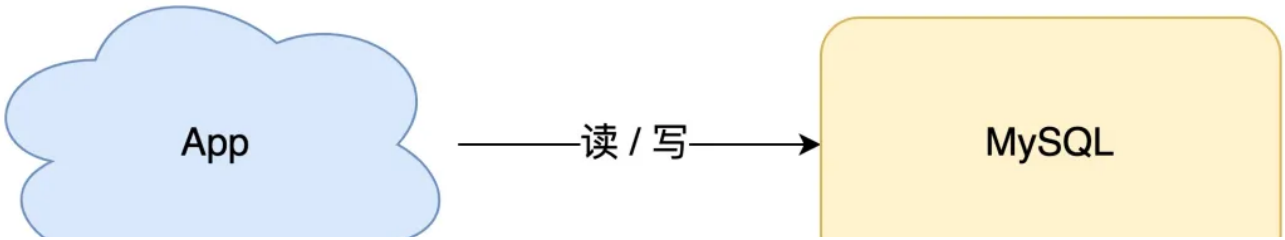
這篇文章乾貨很多，希望你可以耐心讀完。



引入緩存提高性能

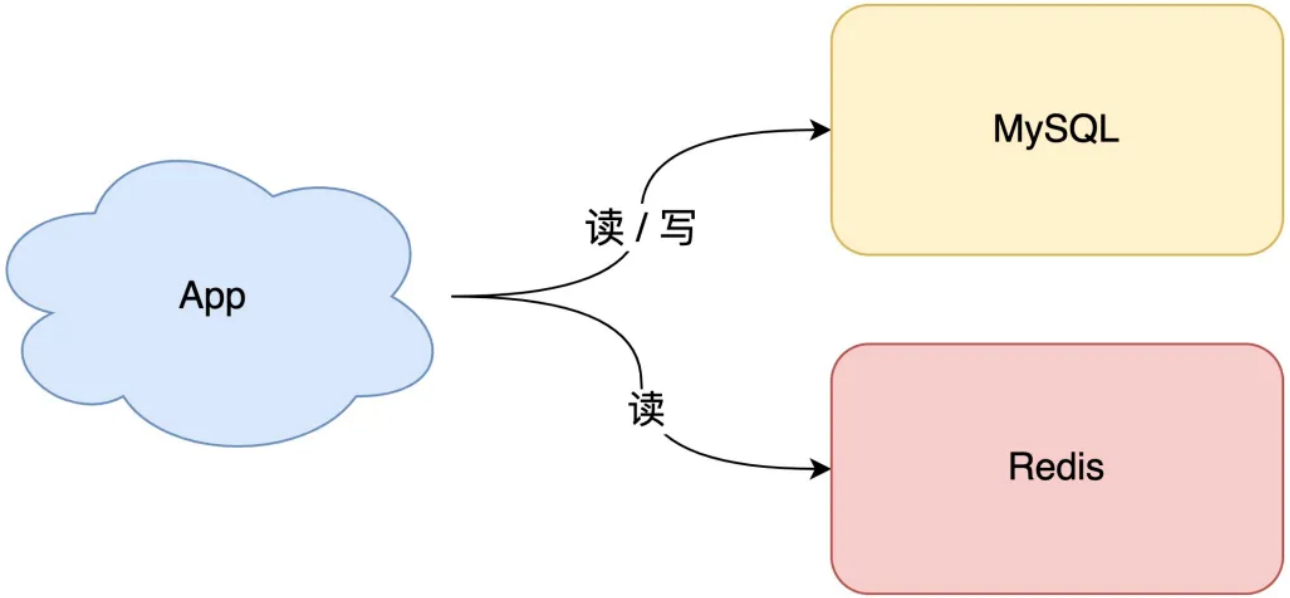
我們從最簡單的場景開始講起。

如果你的業務處於起步階段，流量非常小，那無論是讀請求還是寫請求，直接操作數據庫即可，這時你的架構模型是這樣的：



但隨著業務量的增長，你的項目請求量越來越大，這時如果每次都從數據庫中讀數據，那肯定會有性能問題。

這個階段通常的做法是，引入「緩存」來提高讀性能，架構模型就變成了這樣：



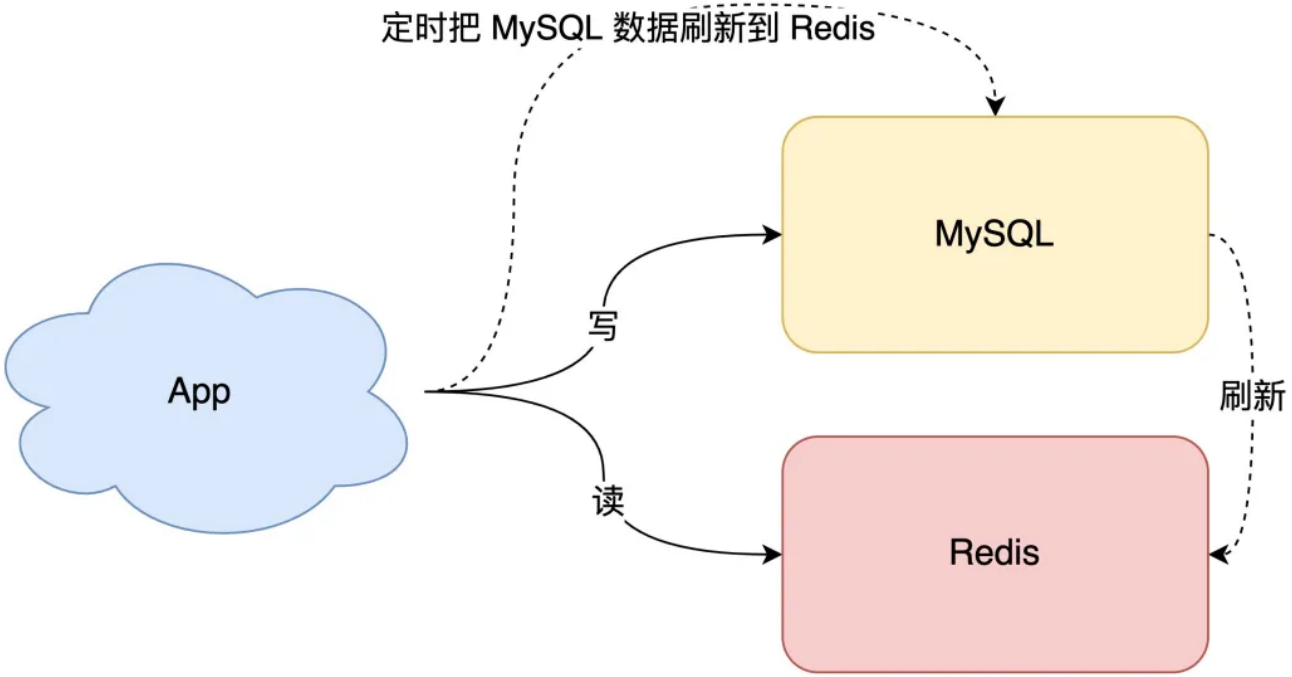
當下優秀的緩存中間件，當屬Redis 莫屬，它不僅性能非常高，還提供了很多友好的數據類型，可以很好地滿足我們的業務需求。

但引入緩存之後，你就會面臨一個問題：

最簡單直接的方案是「全量數據刷到緩存中」：

- 數據庫的數據，全量刷入緩存（不設置失效時間）

- 寫請求只更新數據庫，不更新緩存
- 啟動一個定時任務，定時把數據庫的數據，更新到緩存中



這個方案的優點是，所有讀請求都可以直接「命中」緩存，不需要再查數據庫，性能非常高。

但缺點也很明顯，有2 個問題：

1. **緩存利用率低**
2. **數據不一致**

所以，這種方案一般更適合業務「體量小」，且對數據一致性要求不高的業務場景。

那如果我們的業務體量很大，怎麼解決這2 個問題呢？

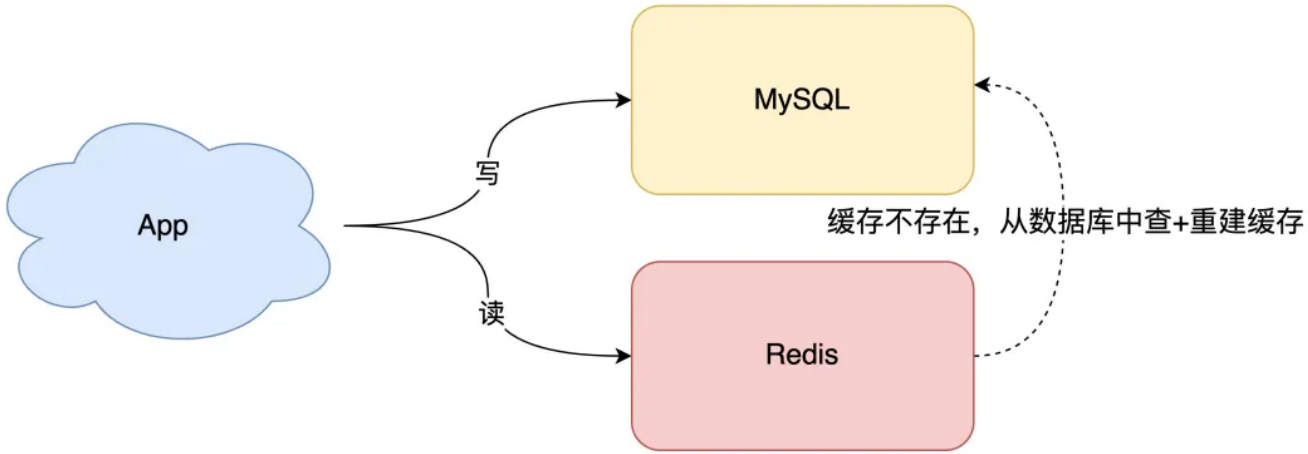
緩存利用率和一致性問題

先來看第一個問題，如何提高緩存利用率？

想要緩存利用率「最大化」，我們很容易想到的方案是，緩存中只保留最近訪問的「熱數據」。但具體要怎麼做呢？

我們可以這樣優化：

- 寫請求依舊只寫數據庫
- 讀請求先讀緩存，如果緩存不存在，則從數據庫讀取，並重建緩存
- 同時，寫入緩存中的數據，都設置失效時間



這樣一來，緩存中不經常訪問的數據，隨著時間的推移，都會逐漸「過期」淘汰掉，最終緩存中保留的，都是經常被訪問的「熱數據」，緩存利用率得以最大化。

再來看數據一致性問題。

要想保證緩存和數據庫「實時」一致，那就不能再用定時任務刷新緩存了。

所以，當數據發生更新時，我們不僅要操作數據庫，還要一併操作緩存。具體操作就是，修改一條數據時，不僅要更新數據庫，也要連帶緩存一起更新。

但數據庫和緩存都更新，又存在先後問題，那對應的方案就有2 個：

1. 先更新緩存，後更新數據庫
2. 先更新數據庫，後更新緩存

哪個方案更好呢？

先不考慮並發問題，正常情況下，無論誰先誰後，都可以讓兩者保持一致，但現在我們需要重點考慮「異常」情況。

因為操作分為兩步，那麼就很有可能存在「第一步成功、第二步失敗」的情況發生。

這2 種方案我們一個個來分析。

1) 先更新緩存，後更新數據庫

如果緩存更新成功了，但數據庫更新失敗，那麼此時緩存中是最新值，但數據庫中是「舊值」。

雖然此時讀請求可以命中緩存，拿到正確的值，但是，一旦緩存「失效」，就會從數據庫中讀取到「舊值」，重建緩存也是這個舊值。

這時用戶會發現自己之前修改的數據又「變回去」了，對業務造成影響。

2) 先更新數據庫，後更新緩存

如果數據庫更新成功了，但緩存更新失敗，那麼此時數據庫中是最新值，緩存中是「舊值」。

之後的讀請求讀到的都是舊數據，只有當緩存「失效」後，才能從數據庫中得到正確的值。

這時用戶會發現，自己剛剛修改了數據，但卻看不到變更，一段時間過後，數據才變更過來，對業務也會有影響。

可見，無論誰先誰後，但凡後者發生異常，就會對業務造成影響。那怎麼解決這個問題呢？

別急，後面我會詳細給出對應的解決方案。

我們繼續分析，除了操作失敗問題，還有什麼場景會影響數據一致性？

這裡我們還需要重點關注：

並發引發的一致性問題

假設我們採用「先更新數據庫，再更新緩存」的方案，並且兩步都可以「成功執行」的前提下，如果存在並發，情況會是怎樣的呢？

有線程A 和線程B 兩個線程，需要更新「同一條」數據，會發生這樣的場景：

1. 線程A 更新數據庫（X = 1）

2. 線程B 更新數據庫（X = 2）
3. 線程B 更新緩存（X = 2）
4. 線程A 更新緩存（X = 1）

最終X 的值在緩存中是1，在數據庫中是2，發生不一致。

也就是說，A 雖然先於B 發生，但B 操作數據庫和緩存的時間，卻要比A 的時間短，執行時序發生「錯亂」，最終這條數據結果是不符合預期的。

> 同樣地，採用「先更新緩存，再更新數據庫」的方案，也會有類似問題，這裡不再詳述。

除此之外，我們從「緩存利用率」的角度來評估這個方案，也是不太推薦的。

這是因為每次數據發生變更，都「無腦」更新緩存，但是緩存中的數據不一定會被「馬上讀取」，這就會導致緩存中可能存放了很多不常訪問的數據，浪費緩存資源。

而且很多情況下，寫到緩存中的值，並不是與數據庫中的值一一對應的，很有可能是先查詢數據庫，再經過一系列「計算」得出一個值，才把這個值才寫到緩存中。

由此可見，這種「更新數據庫+ 更新緩存」的方案，不僅緩存利用率不高，還會造成機器性能的浪費。

所以此時我們需要考慮另外一種方案：

刪除緩存可以保證一致性嗎？

刪除緩存對應的方案也有2 種：

1. 先刪除緩存，後更新數據庫
2. 先更新數據庫，後刪除緩存

經過前面的分析我們已經得知，但凡「第二步」操作失敗，都會導致數據不一致。

這裡我不再詳述具體場景，你可以按照前面的思路推演一下，就可以看到依舊存在數據不一致的情況。

這裡我們重點來看「並發」問題。

1) 先刪除緩存，後更新數據庫

如果有2 個線程要並發「讀寫」數據，可能會發生以下場景：

1. 線程A 要更新X = 2（原值X = 1）
2. 線程A 先刪除緩存
3. 線程B 讀緩存，發現不存在，從數據庫中讀取到舊值（X = 1）
4. 線程A 將新值寫入數據庫（X = 2）
5. 線程B 將舊值寫入緩存（X = 1）

最終X 的值在緩存中是1（舊值），在數據庫中是2（新值），發生不一致。

可見，先刪除緩存，後更新數據庫，當發生「讀+寫」並發時，還是存在數據不一致的情況。

2) 先更新數據庫，後刪除緩存

依舊是2 個線程並發「讀寫」數據：

1. 緩存中X 不存在（數據庫X = 1）
2. 線程 A 讀取數據庫，得到舊值（X = 1）
3. 線程B 更新數據庫（X = 2）
4. 線程B 刪除緩存
5. 線程A 將舊值寫入緩存（X = 1）

最終X 的值在緩存中是1（舊值），在數據庫中是2（新值），也發生不一致。

這種情況「理論」來說是可能發生的，但實際真的有可能發生嗎？

其實概率「很低」，這是因為它必須滿足3 個條件：

1. 緩存剛好已失效
2. 讀請求+ 寫請求並發
3. 更新數據庫+ 刪除緩存的時間（步驟3-4），要比讀數據庫+ 寫緩存時間短（步驟2 和5）

仔細想一下，條件3 發生的概率其實是非常低的。

因為寫數據庫一般會先「加鎖」，所以寫數據庫，通常是要比讀數據庫的時間更長的。

這麼來看，「先更新數據庫+ 再刪除緩存」的方案，是可以保證數據一致性的。

所以，我們應該採用這種方案，來操作數據庫和緩存。

好，解決了並發問題，我們繼續來看前面遺留的，

如何保證兩步都執行成功?

前面我們分析到，無論是更新緩存還是刪除緩存，只要第二步發生失敗，那麼就會導致數據庫和緩存不一致。

保證第二步成功執行，就是解決問題的關鍵。

想一下，程序在執行過程中發生異常，最簡單的解決辦法是什麼?

答案是:

是的，其實這裡我們也可以這樣做。

無論是先操作緩存，還是先操作數據庫，但凡後者執行失敗了，我們就可以發起重試，盡可能地去做「補償」。

那這是不是意味著，只要執行失敗，我們「無腦重試」就可以了呢?

答案是否定的。現實情況往往沒有想的這麼簡單，失敗後立即重試的問題在於:

- 立即重試很大概率「還會失敗」
- 「重試次數」設置多少才合理?
- 重試會一直「佔用」這個線程資源，無法服務其它客戶端請求

看到了麼，雖然我們想通過重試的方式解決問題，但這種「同步」重試的方案依舊不嚴謹。

那更好的方案應該怎麼做？

答案是：什麼是異步重試？

其實就是把重試請求寫到「消息隊列」中，然後由專門的消費者來重試，直到成功。

或者更直接的做法，為了避免第二步執行失敗，我們可以把操作緩存這一步，直接放到消息隊列中，由消費者來操作緩存。

到這裡你可能會問，寫消息隊列也有可能會失敗啊？而且，引入消息隊列，這又增加了更多的維護成本，這樣做值得嗎？

這個問題很好，但我們思考這樣一個問題：如果在執行失敗的線程中一直重試，還沒等執行成功，此時如果項目「重啟」了，那這次重試請求也就「丟失」了，那這條數據就一直不一致了。

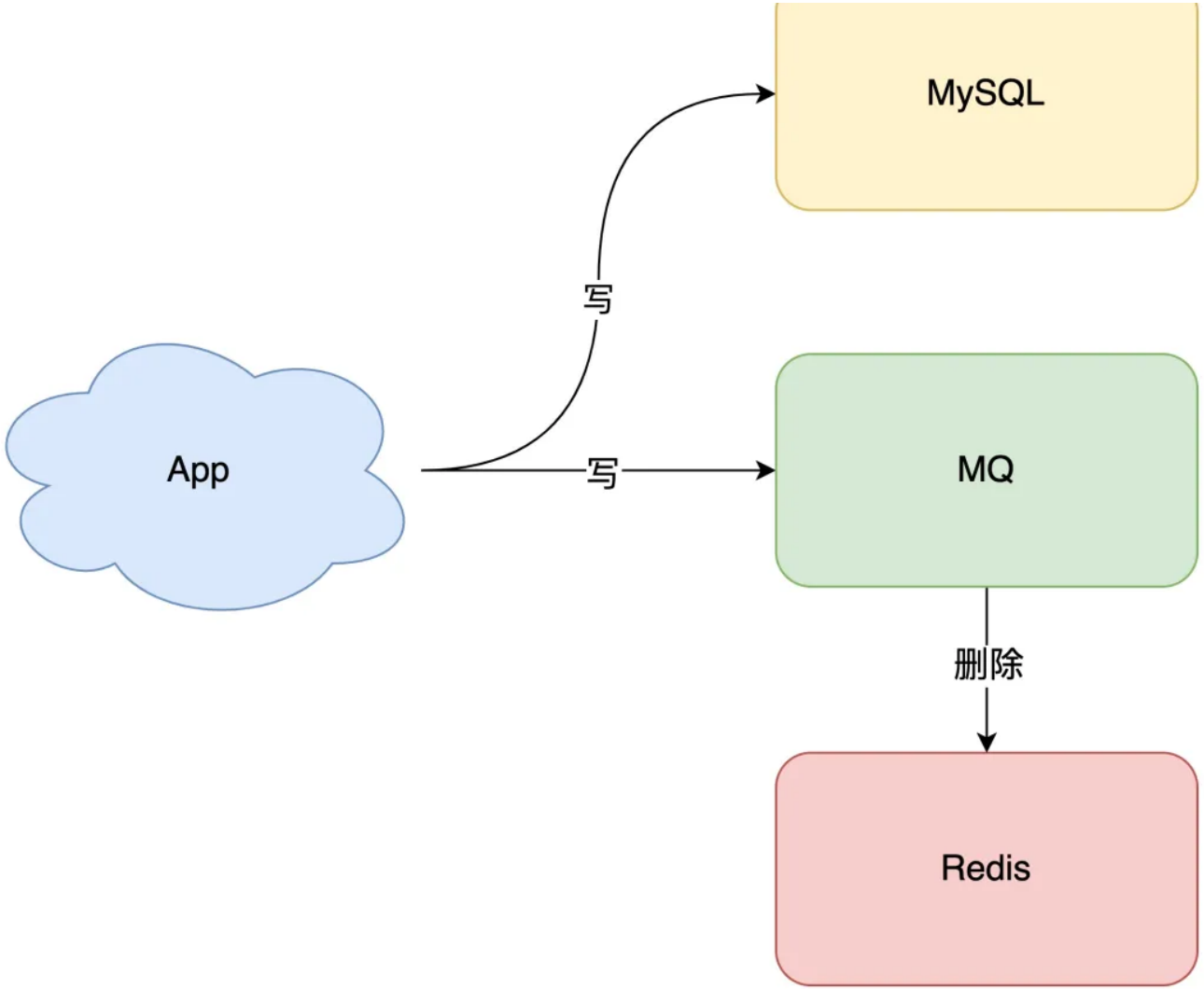
所以，這裡我們必須把重試或第二步操作放到另一個「服務」中，這個服務用「消息隊列」最為合適。這是因為消息隊列的特性，正好符合我們的需求：

- **消息隊列保證可靠性**
- **消息隊列保證消息成功投遞**

至於寫隊列失敗和消息隊列的維護成本問題：

- **寫隊列失敗**
- **維護成本**

所以，引入消息隊列來解決這個問題，是比較合適的。這時架構模型就變成了這樣：



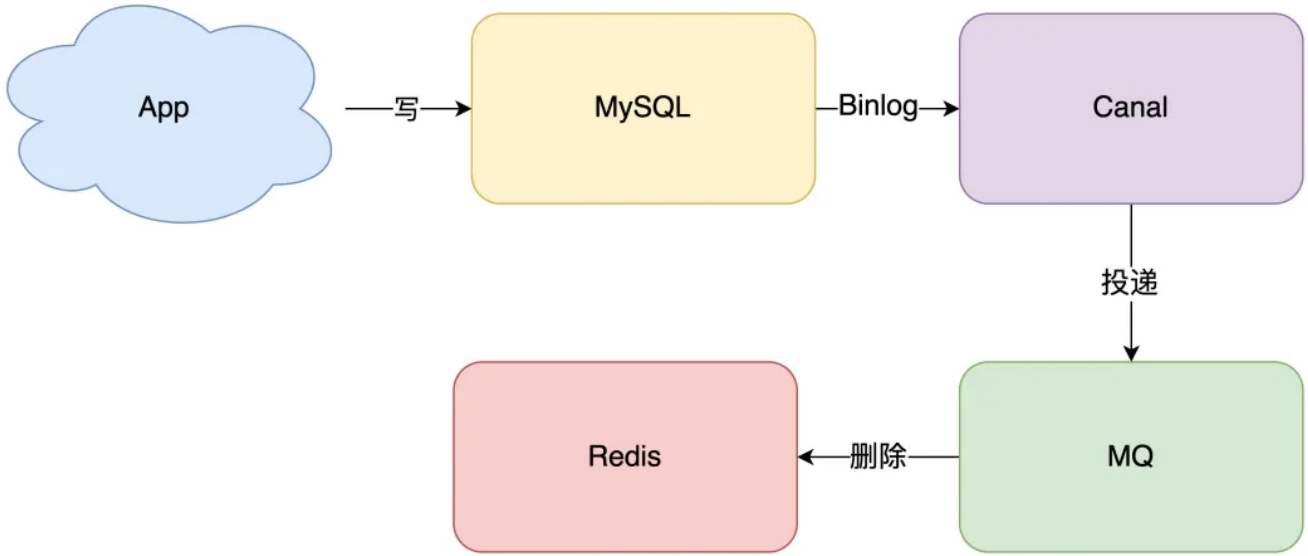
那如果你確實不想在應用中去寫消息隊列，是否有更簡單的方案，同時又可以保證一致性呢？

方案還是有的，這就是近幾年比較流行的解決方案：

具體來講就是，我們的業務應用在修改數據時，「只需」修改數據庫，無需操作緩存。

那什麼時候操作緩存呢？這就和數據庫的「變更日誌」有關了。

拿MySQL 舉例，當一條數據發生修改時，MySQL 就會產生一條變更日誌（Binlog），我們可以訂閱這個日誌，拿到具體操作的數據，然後再根據這條數據，去刪除對應的緩存。



訂閱變更日誌，目前也有了比較成熟的開源中間件，例如阿里的canal，使用這種方案的優點在於：

- **無需考慮寫消息隊列失敗情況**
- **自動投遞到下游隊列**

當然，與此同時，我們需要投入精力去維護canal 的高可用和穩定性。

> 如果你有留意觀察很多數據庫的特性，就會發現其實很多數據庫都逐漸開始提供「訂閱變更日誌」的功能了，相信不遠的將來，我們就不用通過中間件來拉取日誌，自己寫程序就可以訂閱變更日誌了，這樣可以進一步簡化流程。

至此，我們可以得出結論，想要保證數據庫和緩存一致性，

主從庫延遲和延遲雙刪問題

到這裡，還有2 個問題，是我們沒有重點分析過的。

第一個問題

這裡我再把例子拿過來讓你複習一下：

2 個線程要並發「讀寫」數據，可能會發生以下場景：

1. 線程A 要更新X = 2（原值X = 1）
2. 線程A 先刪除緩存
3. 線程B 讀緩存，發現不存在，從數據庫中讀取到舊值（X = 1）
4. 線程A 將新值寫入數據庫（X = 2）
5. 線程B 將舊值寫入緩存（X = 1）

最終X 的值在緩存中是1（舊值），在數據庫中是2（新值），發生不一致。

第二個問題

在「先更新數據庫，再刪除緩存」方案下，「讀寫分離+ 主從庫延遲」其實也會導致不一致：

1. 線程A 更新主庫X = 2（原值X = 1）
2. 線程A 刪除緩存
3. 線程B 查詢緩存，沒有命中，查詢「從庫」得到舊值（從庫X = 1）
4. 從庫「同步」完成（主從庫X = 2）
5. 線程B 將「舊值」寫入緩存（X = 1）

最終X 的值在緩存中是1（舊值），在主從庫中是2（新值），也發生不一致。

看到了麼？這2 個問題的核心在於：

那怎麼解決這類問題呢？

最有效的辦法就是，

但是，不能立即刪，而是需要「延遲刪」，這就是業界給出的方案：

按照延時雙刪策略，這2 個問題的解決方案是這樣的：

解決第一個問題

解決第二個問題

這兩個方案的目的，都是為了把緩存清掉，這樣一來，下次就可以從數據庫讀取到最新值，寫入緩存。

但問題來了，這個「延遲刪除」緩存，延遲時間到底設置要多久呢?

- 問題1：延遲時間要大於「主從復制」的延遲時間
- 問題2：延遲時間要大於線程B 讀取數據庫+ 寫入緩存的時間

但是，

很多時候，我們都是憑藉經驗大致估算這個延遲時間，例如延遲1-5s，只能盡可能地降低不一致的概率。

所以你看，採用這種方案，也只是盡可能保證一致性而已，極端情況下，還是有可能發生不一致。

所以實際使用中，我還是建議你採用「先更新數據庫，再刪除緩存」的方案，同時，要盡可能地保證「主從復制」不要有太大延遲，降低出問題的概率。

可以做到強一致嗎?

看到這裡你可能會想，這些方案還是不夠完美，我就想讓緩存和數據庫「強一致」，到底能不能做到呢?

其實很難。

要想做到強一致，最常見的方案是2PC、3PC、Paxos、Raft 這類一致性協議，但它們的性能往往比較差，而且這些方案也比較複雜，還要考慮各種容錯問題。

相反，這時我們換個角度思考一下，我們引入緩存的目的是什麼?

沒錯，

一旦我們決定使用緩存，那必然要面臨一致性問題。性能和一致性就像天平的兩端，無法做到都滿足要求。

而且，就拿我們前面講到的方案來說，當操作數據庫和緩存完成之前，只要有其它請求可以進來，都有可能查到「中間狀態」的數據。

所以如果非要追求強一致，那必須要求所有更新操作完成之前期間，不能有「任何請求」進來。

雖然我們可以通過加「分佈鎖」的方式來實現，但我們要付出的代價，很可能會超過引入緩存帶來的性能提升。

所以，既然決定使用緩存，就必須容忍「一致性」問題，我們只能盡可能地去降低問題出現的概率。

同時我們也要知道，緩存都是有「失效時間」的，就算在這期間存在短期不一致，我們依舊有失效時間來兜底，這樣也能達到最終一致。

總結

好了，總結一下這篇文章的重點。

1、想要提高應用的性能，可以引入「緩存」來解決

2、引入緩存後，需要考慮緩存和數據庫一致性問題，可選的方案有：「更新數據庫+ 更新緩存」、「更新數據庫+ 刪除緩存」

3、更新數據庫+ 更新緩存方案，在「並發」場景下無法保證緩存和數據一致性，且存在「緩存資源浪費」和「機器性能浪費」的情況發生

4、在更新數據庫+ 刪除緩存的方案中，「先刪除緩存，再更新數據庫」在「並發」場景下依舊有數據不一致問題，解決方案是「延遲雙刪」，但這個延遲時間很難評估，所以推薦用「先更新數據庫，再刪除緩存」的方案

5、在「先更新數據庫，再刪除緩存」方案下，為了保證兩步都成功執行，需配合「消息隊列」或「訂閱變更日誌」的方案來做，本質是通過「重試」的方式保證數據一致性

6、在「先更新數據庫，再刪除緩存」方案下，「讀寫分離+ 主從庫延遲」也會導致緩存和數據庫不一致，緩解此問題的方案是「延遲雙刪」，憑藉經驗發送「延遲消息」到隊列中，延遲刪除緩存，同時也要控制主從庫延遲，盡可能降低不一致發生的概率

後記

本以為這個老生常談的話題，寫起來很好寫，沒想到在寫的過程中，還是挖到了

在這裡我也分享4 點心得給你：

1、性能和一致性不能同時滿足，為了性能考慮，通常會採用「最終一致性」的方案

2、掌握緩存和數據庫一致性問題，核心問題有3 點：緩存利用率、並發、緩存+ 數據庫一起成功問題

3、失敗場景下要保證一致性，常見手段就是「重試」，同步重試會影響吞吐量，所以通常會採用異步重試的方案

4、訂閱變更日誌的思想，本質是把權威數據源（例如MySQL）當做leader 副本，讓其它異質系統（例如Redis / Elasticsearch）成為它的follower 副本，通過同步變更日誌的方式，保證leader 和follower 之間保持一致

很多一致性問題，都會採用這些方案來解決，希望我的這些心得對你有所啟發。

丙丙
我的心冰冰的

2篇原創內容

公眾號

喜歡此內容的人還喜歡

分佈式數據庫-應用場景與對比測試（1）

白鱔的洞穴

航母級基因功能預測數據庫

醫學數據庫百科

Netflix：提升視頻編碼工作流效率

媒礦工廠