看完這篇，你還不能理解'數據庫架構'？ 趁早回家吧

糸糸人物 朱小廝的博客 今天

點擊上方“ 朱小廝的博客”，選擇“設為星標”

回复” 資料“獲取新整理的1TB資料

來源：http://rrd.me/ep46N

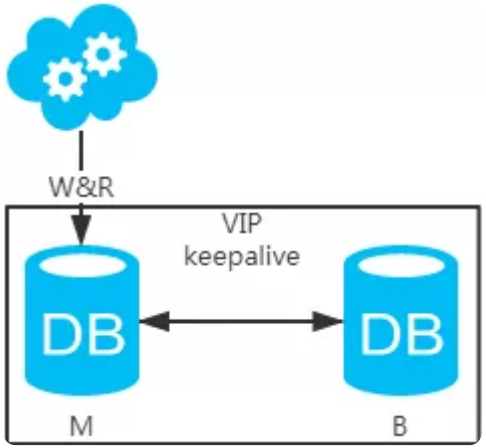
一、數據庫架構原則

1. 高可用

2. 高性能
3. 一致性
4. 擴展性

二、常見的架構方案

方案一：主備架構，只有主庫提供讀寫服務，備庫冗餘作故障轉移用




```
jdbc:mysql://vip:3306/xxdb
```

1、**高可用分析：**高可用，主庫掛了，keepalive（只是一種工具）會自動切換到備庫。這個過程對業務層是透明的，無需修改代碼或配置。

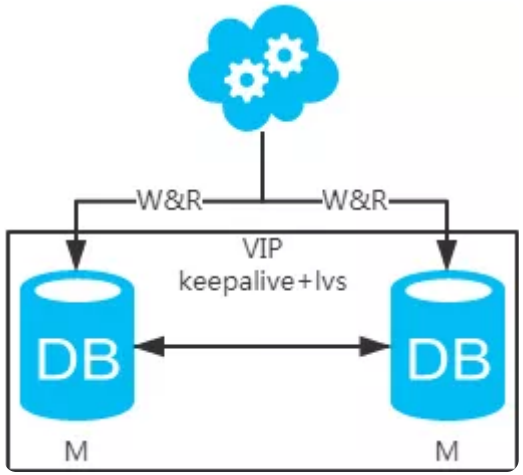
2、**高性能分析：**讀寫都操作主庫，很容易產生瓶頸。大部分互聯網應用讀多寫少，讀會先成為瓶頸，進而影響寫性能。另外，備庫只是單純的備份，資源利用率50%，這點方案二可解決。

3、**一致性分析：**讀寫都操作主庫，不存在數據一致性問題。

4、**擴展性分析：**無法通過加從庫來擴展讀性能，進而提高整體性能。

5、**可落地分析：**兩點影響落地使用。第一，性能一般，這點可以通過建立高效的索引和引入緩存來增加讀性能，進而提高性能。這也是通用的方案。第二，擴展性差，這點可以通過**分庫分錶**來擴展。

方案二：雙主架構，兩個主庫同時提供服務，負載均衡




```
jdbc:mysql://vip:3306/xxdb
```

1、**高可用分析：**高可用，一個主庫掛了，不影響另一台主庫提供服務。這個過程對業務層是透明的，無需修改代碼或配置。

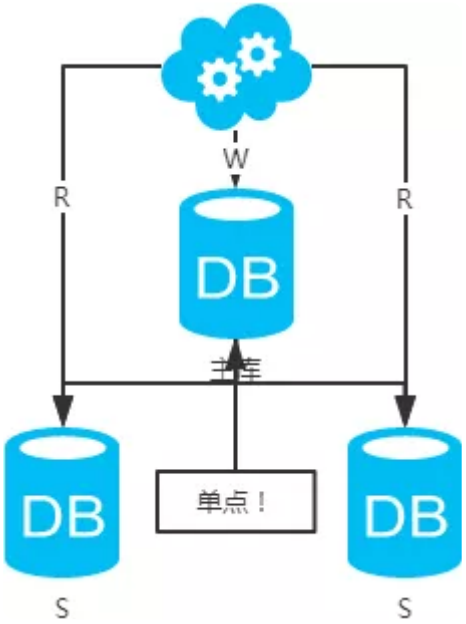
2、**高性能分析：**讀寫性能相比於方案一都得到提升，提升一倍。

3、**一致性分析：**存在數據一致性問題。請看，**一致性解決方案**。

4、**擴展性分析：**當然可以擴展成三主循環，但筆者不建議（會多一層**數據同步**，這樣同步的時間會更長）。如果非得在數據庫架構層面擴展的話，擴展為方案四。

5、**可落地分析：**兩點影響落地使用。第一，數據一致性問題，**一致性解決方案**可解決問題。第二，主鍵衝突問題，ID統一地由分佈式ID生成服務來生成可解決問題。

方案三：主從架構，一主多從，讀寫分離




```
jdbc:mysql://master-ip:3306/xxdb
jdbc:mysql://slave1-ip:3306/xxdb
jdbc:mysql://slave2-ip:3306/xxdb
```

1、**高可用分析：**主庫單點，從庫高可用。一旦主庫掛了，寫服務也就無法提供。

2、**高性能分析：**大部分互聯網應用讀多寫少，讀會先成為瓶頸，進而影響整體性能。讀的性能提高了，整體性能也提高了。另外，主庫可以不用索引，線上從庫和線下從庫也可以建立不同的索引（線上從庫如果有多個還是要建立相同的索引，不然得不償失；線下從庫是平時開發人員排查線上問題時查的庫，可以建更多的索引）。

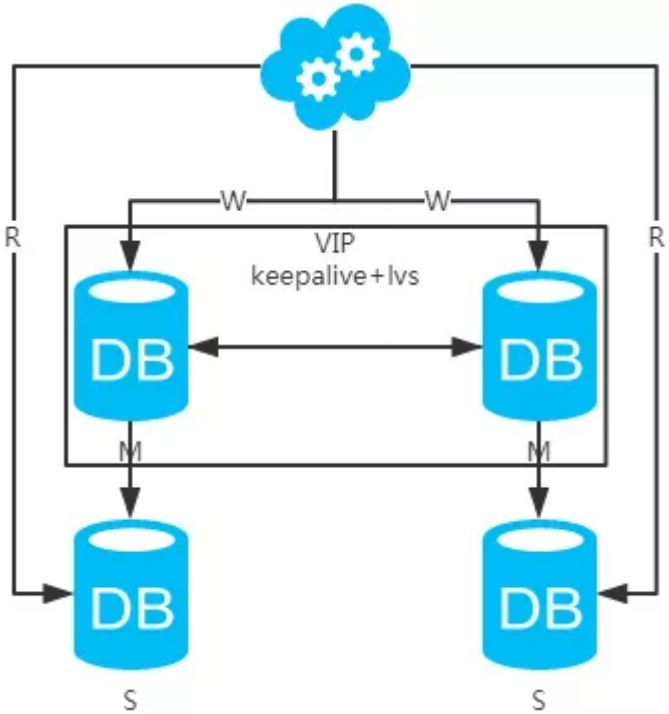
3、**一致性分析**：存在數據一致性問題。請看，**一致性解決方案**。

4、**擴展性分析**：可以通過加從庫來擴展讀性能，進而提高整體性能。（帶來的問題是，從庫越多需要從主庫拉取binlog日誌的端就越多，進而影響主庫的性能，並且**數據同步**完成的時間也會更長）

5、**可落地分析**：兩點影響落地使用。第一，數據一致性問題，**一致性解決方案**可解決問題。第二，主庫單點問題，筆者暫時沒想到很好的解決方案。

注：思考一個問題，一台從庫掛了會怎樣？讀寫分離之讀的負載均衡策略怎麼容錯？

方案四：雙主+主從架構，看似完美的方案




```
jdbc:mysql://vip:3306/xxdb
```

```
jdbc:mysql://slave1-ip:3306/xxdb
jdbc:mysql://slave2-ip:3306/xxdb
```

1、**高可用分析**：高可用。

2、**高性能分析**：高性能。

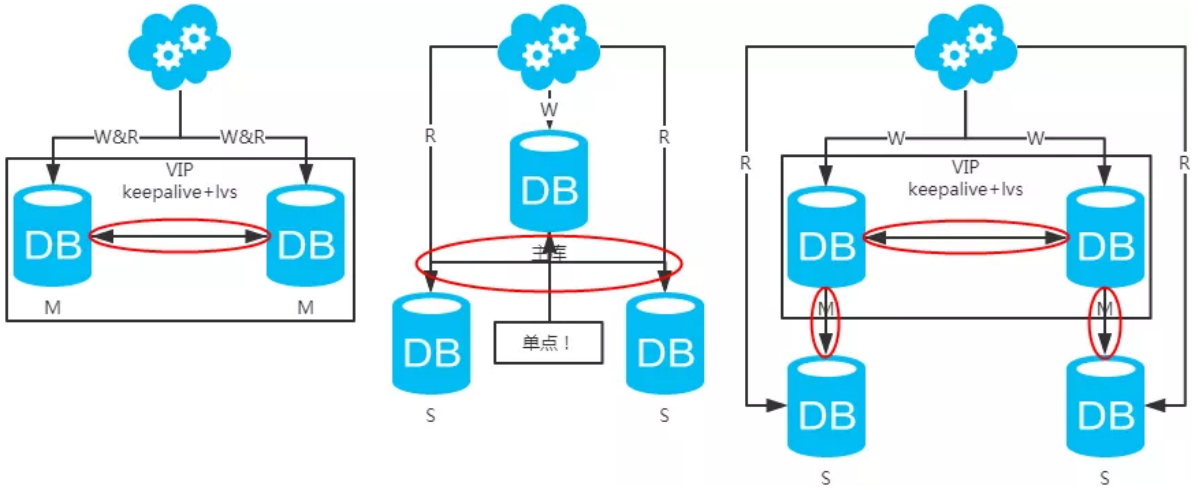
3、**一致性分析**：存在數據一致性問題。請看，**一致性解決方案**。

4、**擴展性分析**：可以通過加從庫來擴展讀性能，進而提高整體性能。（帶來的問題**同方案二**）

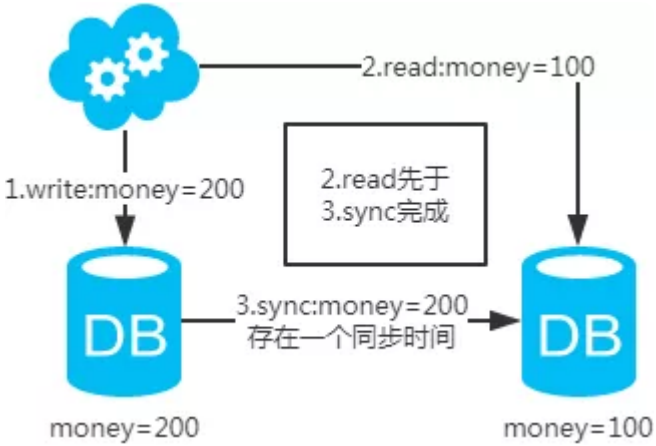
5、**可落地分析**：**同方案二**，但**數據同步**又多了一層，數據延遲更嚴重。

三、一致性解決方案

第一類：主庫和從庫一致性解決方案



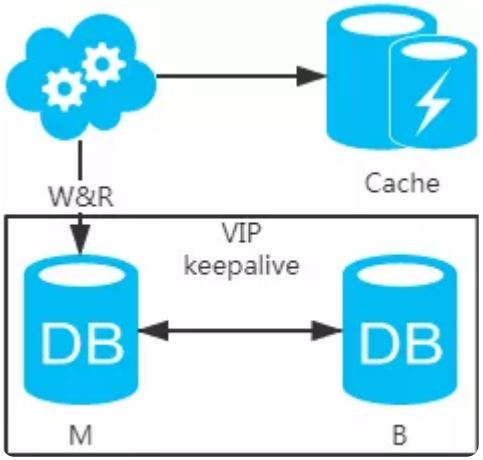
注：圖中圈出的是數據同步的地方，數據同步（從庫從主庫拉取binlog日誌，再執行一遍）是需要時間的，這個同步時間內主庫和從庫的數據會存在不一致的情況。如果同步過程中有讀請求，那麼讀到的就是從庫中的老數據。如下圖。



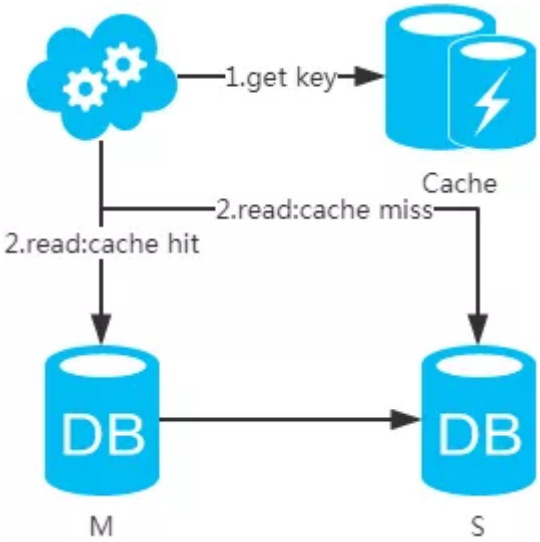
既然知道了數據不一致性產生的原因，有下面幾個解決方案供參考：

1、直接忽略，如果業務允許延時存在，那麼就不去管它。

2、強制讀主，採用**主備架構**方案，讀寫都走主庫。用緩存來擴展數據庫讀性能。有一點需要知道：如果緩存掛了，可能會產生雪崩現象，不過一般分佈式緩存都是高可用的。

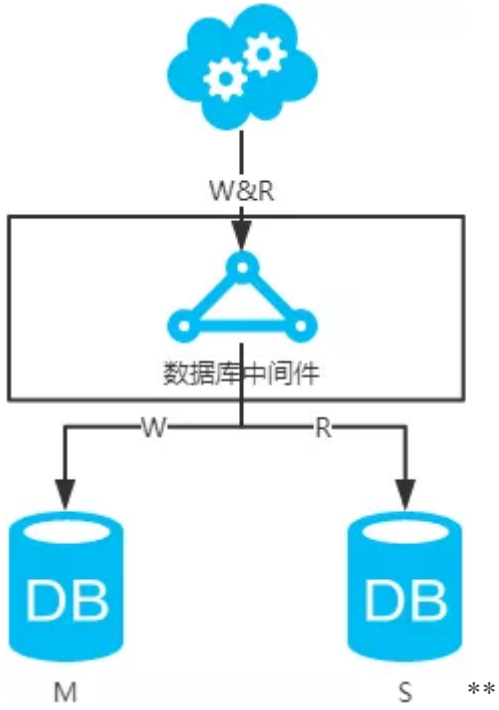


3、選擇讀主，寫操作時根據庫+表+業務特徵生成一個key放到Cache裡並設置超時時間（大於等於主從數據同步時間）。讀請求時，同樣的方式生成key先去查Cache，再判斷是否命中。若命中，則讀主庫，否則讀從庫。代價是多了一次緩存讀寫，基本可以忽略。

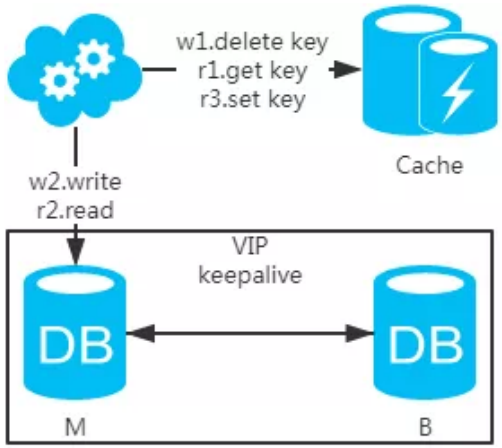


4、半同步複製，等主從同步完成，寫請求才返回。就是大家常說的“半同步複製”semi-sync。這可以利用數據庫原生功能，實現比較簡單。代價是寫請求時延增長，吞吐量降低。

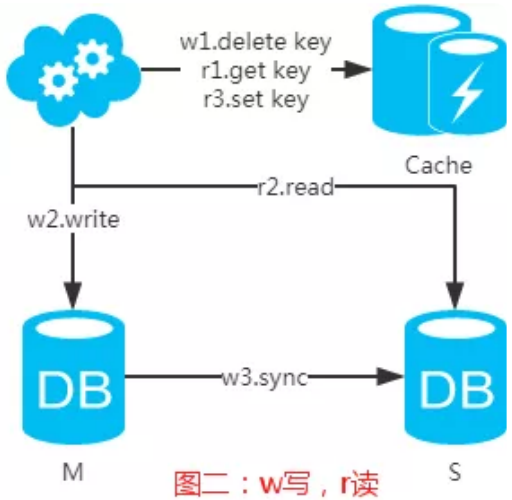
5、數據庫中間件，引入開源（mycat等）或自研的數據庫中間層。個人理解，思路同**選擇讀主**。數據庫中間件的成本比較高，並且還多引入了一層。**



第二類：**DB**和緩存一致性解決方案



图一：w写，r读



图二：w写，r读

先來看一下常用的緩存使用方式：

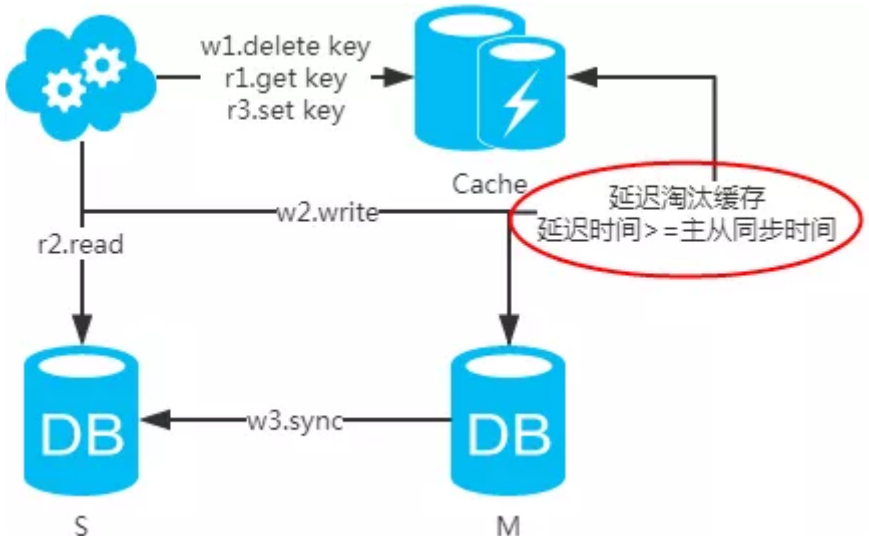
第一步：淘汰緩存；

第二步：寫入數據庫；

第三步：讀取緩存？返回：讀取數據庫；

第四步：讀取數據庫後寫入緩存。

注：如果按照這種方式，圖一，不會產生DB和緩存不一致問題；圖二，會產生DB和緩存不一致問題，即4.read先於3.sync執行。如果不做處理，緩存裡的數據可能一直是臟數據。解決方式如下：



注：設置緩存時，一定要加上有效時間，以防延時淘汰緩存失敗的情況！

四、個人的一些見解

1、架構演變

1、架構演變一：方案一-> 方案一+分庫分錶-> 方案二+分庫分錶-> 方案四+分庫分錶；

2、架構演變二：方案一-> 方案一+分庫分錶-> 方案三+分庫分錶-> 方案四+分庫分錶；

3、架構演變三：方案一-> 方案二-> 方案四-> 方案四+分庫分錶；

4、架構演變四：方案一-> 方案三-> 方案四-> 方案四+分庫分錶；

2、個人見解

1、加緩存和索引是通用的提升數據庫性能的方式；

2、分庫分錶帶來的好處是巨大的，但同樣也會帶來一些問題，詳見前日推文。

3、不管是主備+分庫分錶還是主從+讀寫分離+分庫分錶，都要考慮具體的業務場景。絕大部分的數據庫架構還是採用方案一和方案一+分庫分錶，只有極少部分用方案三+讀寫分離+分庫分錶。另外，阿里雲提供的數據庫雲服務也都是主備方案，要想主從+讀寫分離需要二次架構。

4、記住一句話：不考慮業務場景的架構都是耍流氓。

想知道更多？掃描下面的二維碼關注我

朕已閱

閱讀原文