如何保证缓存与数据库的一致性

比较经典的做法就是：缓存 + 数据库读写模式

1，**查询的时候**：先读缓存，缓存没有读数据库，然后取出数据库的数据，放入缓存，返回响应。

2，**更新的时候**：先更新数据库，然后在删除缓存。

更新的时候为什么是删除缓存，而不是更新缓存?

一个比较耗时缓存计算场景，如果你频繁修改某个表的数据，每次都涉这个缓存更新，但是这个缓存在这段时间内都不会被用到。

例子：一个缓存涉及的表字段，在1分钟被修改20次，那么缓存也更新20次，但是这个缓存在1分钟内，只有被读取到1次，有大量的冷数据。

实际上，如果你删除了这个缓存的话，那么在1分钟内内，只有被读取这个缓存的时候计算1次，并且写入缓存。开销大幅度降低，用到才去缓存，就是懒加载思想。

更新缓存 VS 淘汰缓存

更新缓存：数据不但写入数据库，还会写入缓存
淘汰缓存：数据只会写入数据库，不会写入缓存，只会把缓存删除掉

更新缓存的优点：缓存的命中率高，不会因为缓存没有去查询数据库
淘汰缓存的优点：简单，直接删除缓存啊

那么到底是更新缓存，还是淘汰缓存?

取决于**“更新缓存的复杂度”**，比如更新缓存的内容需要复杂查询计算，那么淘汰缓存就更适合

先操作数据库 vs 先操作缓存

当写操作发生时，假设淘汰缓存作为对缓存通用的处理方式，那么又面临着两种选择

1、先写数据库，再淘汰缓存
2、写淘汰缓存，再写数据库

怎么选择 先淘汰缓存，再写数据库？

对于一个不能保证事务性的操作时，一定涉及 “那个任务先做，那个任务后做” 的问题，解决这个问题的方向是：

如果出现数据不一致，谁先做对业务的影响小，就先执行谁

1、假设先写数据库，再淘汰缓存：第一步写数据库操作成功，第二步淘汰缓存失败，则会出现 db 中是新数据，cache 中是旧数据，数据不一致了，**这是属于原子性被破坏，导致的不一致**

2、假设先淘汰缓存，在写数据库：第一步淘汰储存成功，第二部写数据库失败，则会出现数据库中是旧的，缓存是空的，只会引起一次 `Cache miss` 【这就是所谓的**数据未命中**“miss”】（就是缓存没有数据时候，重新查询数据库，写入缓存）

结论：数据库和缓存的操作顺序，是很清楚的：先淘汰缓存，再写数据库。

缓存架构的优化

上述的架构，有一个**缺点**：**业务需要同时关注cache 和DB** ,有两种常见的方案，，一种是主流方案，一种非主流的方案。

主流的优化方案，服务化：

加入一个服务层，像上级提供数据访问接口，屏蔽底层数据存储的细节，这样业务线就不需要关注我的数据来自db还是cache

非主流的的优化方案，异步缓存更新：

业务线所有的写操作都是走数据，所有的读操作都是走总缓存，这样的话，需要一个异步的工具来做数据库和缓存之间的同步

1、要有一个 init cache 的过程，将需要缓存的数据全量写入cache
2、如果DB有些的操作，异步更新程序读取binlong 更新cache

a）业务线读取cache，一定能够命中（很短的时间，可能有脏数据），无需关注数据库

b)业务线写DB,cache 中能得到异步更新，无需关注缓存

这样大大的简化了业务线的调用逻辑，缺点就是，如果缓存的数据业务比较复杂，async-update (异步更新)的逻辑可能也会复杂

缓存架构的结论强调

1，淘汰缓存是一种通用的处理方式

2，先淘汰缓存，在写数据库的顺序是毋庸置疑的
3，服务化是向业务发展屏蔽底层数据库与缓存复杂性的一种通用方式

为什么缓存 和 数据库会不一致

在分布式情况下，数据的读写都是并发的，上游有多个应用，一个应用多个部署，对于一个数据进行读写，在数据库层面的并发并不能保证是顺序执行，也就是说并发的时候，后发出的请求可能先到先完成（读出脏数据）

栗子：

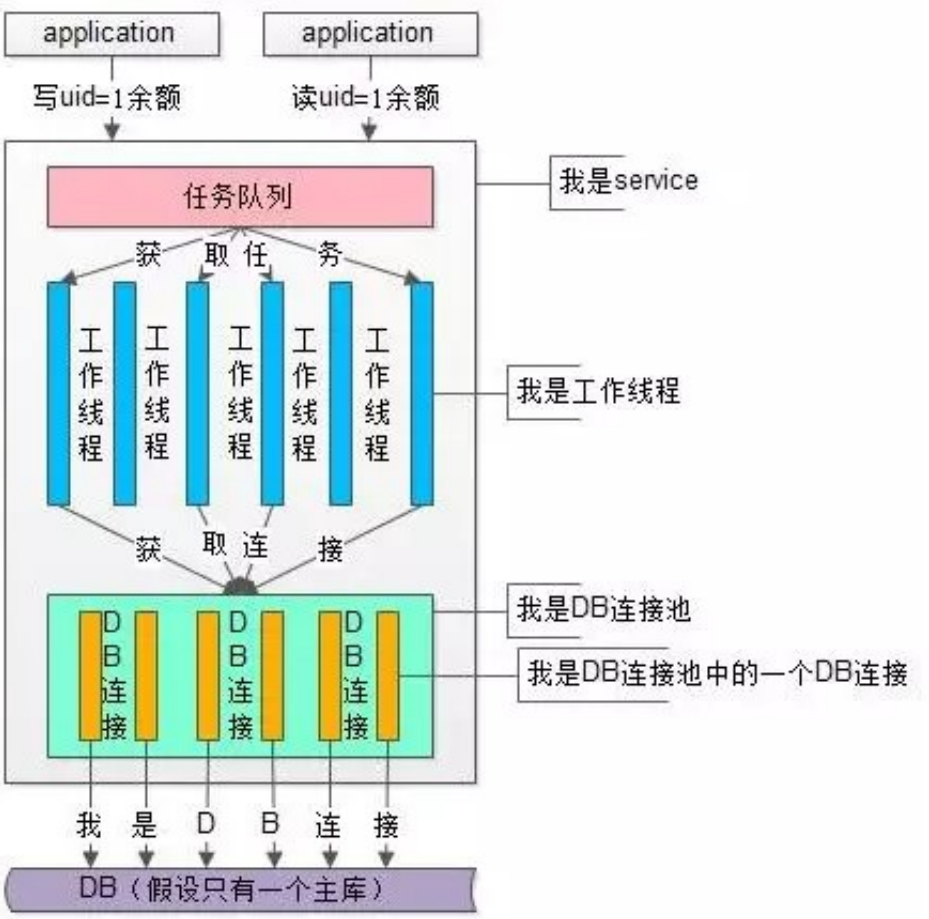
1，并发情况下，A 发出写库请求，此时A 先淘汰 `cache`。
2，A 的写库操作还未完成，这时 B 读请求的过来了，发现 `cache` 是空的，读出了一个脏数据放到 `cache`
3，在数据库层面，后发出的请求先完成操作，读出了脏数据，缓存和数据库的数据就发生了不一致。

不一致的优化思路

能否做到先发出的请求先执行完成呢？，常见的思路，就是“串行化”

栗子：

在一个服务中，并发的多个读写SQL一般是怎么执行的



1，service 的上游是多个业务应用，上游发起请求对一个数据并发的进行读写操作，上图中，并发的进行了 一个uuid = 1 的余额修改（写）操作，与 uuid 的余额查询（读）操作

2，service 的下游是数据库DB 假设读写一个DB

3，中间是服务层 service 他又分成了这么几个部分

- 最上层是任务队列
- 中间是工作线程，每个工作线程完成实际的工作任务，**典型的工作任务是通过数据库连接池读写数据**库
- 最下层是数据库连接池，所有的 SQL 语句都是通过数据的连接池发往数据执行的

工作线程的典型工作流是这样的：

```
void work_thread_routine(){
Task t = TaskQueue.pop(); // 获取任务
// 任务逻辑处理，生成sql语句
DBConnection c = CPool.GetDBConnection(); // 从DB连接池获取一个DB连接
c.execSQL(sql); // 通过DB连接执行sql语句
```

```
CPool.PutDBConnection(c); // 将DB连接放回DB连接池
}
```

提问：任务队列已经做了任务队列串行化的工作，能否保证任务不并发执行?

答案：不行，因为

1）1个服务有多个工作线程，串行弹出的任务会被并行执行
2）1个服务有多个数据库连接，每个工作线程获取不同的数据库连接都会在DB层并发执行

提问：假设服务只部署一份，能否保证任我游不并发执行?

答案：不行，原因同上

提问：假设1个服务只有一条数据库连接，能否保证任务不并发执行?

答案：不行，因为

1）1个服务只有1条数据库连接，只能保证一个服务在一个服务器上的请求是串行的
2）因为服务是分布式的，多个服务上的请求在数据库层面，仍可能是并发执行

提问：假设服务只有1份，且只有1条数据库连接，能否保证任务不并发执行?

答案：可以。全局看来请求都是串行执行的，吞吐量很低，并且服务无法保证可用性

解决方案：让数据库的访问能“串行化”就行

其实不需要全局的请求串行化，而只需要 “让同一个数据的访问能串行化” 就行

在一个服务内，如何做到“让同一个数据的访问串行化”，只需要“让同一个数据的访问通过同一条DB连接执行”就行。

如何做到 “让同一个数据的访问通过同一条DB连接执行”，只需要“在DB连接池层面稍微修改，按数据取连接即可”

获取DB 连接的

```
获取DB连接的CPool.GetDBConnection()【返回任何一个可用DB连接】改为

CPool.GetDBConnection(longid)【返回id取模相关联的DB连接】
```

这个修改的好处就是：

1，简单，只需要修改DB连接池实现，以及DB 连接获取处

2，连接池的修改不需要关注业务，传入的ID 是什么含义连接池不关注，直接按照id取模返回DB连接即可

3，可以适用于多种业务场景，取用户数据业务传入 user-id 取连接，取订单数据业务传入 order-id 取连接即可

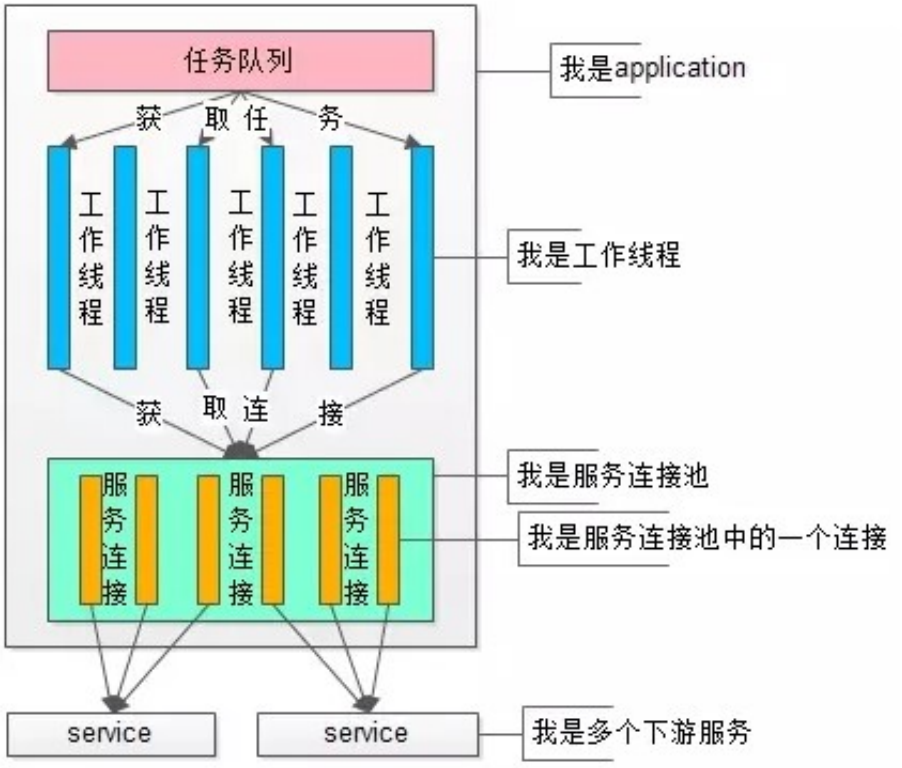
这样的话，就能够保证同一个数据例如 uid 在数据库层面的执行一定是串行的

多服务部署，上述方案就不可用

服务部署了多份，上述方案只能保证同一个数据在同一个服务上访问，在数据库层面是串行执行

实际上服务是分布式部署的，在全局范围的访问仍然是并发的，怎么解决呢？能不能做到，同一数据的访问落到同一个服务器上呢？

同一数据的访问落到同一个服务器上？



上图是一个业务应用的上下游及服务内部详细展开，细节如下：

（1）业务应用的上游不确定是啥，可能是直接是http请求，可能也是一个服务的上游调用

（2）业务应用的下游是多个服务service

（3）中间是业务应用，它又分为了这么几个部分

（3.1）最上层是任务队列【或许web-server例如tomcat帮你干了这个事情了】

（3.2）中间是工作线程【或许web-server的工作线程或者cgi工作线程帮你干了线程分派这个事情了】，每个工作线程完成实际的业务任务，典型的工作任务是通过服务连接池进行RPC调用

（3.3）最下层是服务连接池，所有的RPC调用都是通过服务连接池往下游服务去发包执行的

工作线程的典型工作流是这样的：

```
voidwork_thread_routine(){
Task t = TaskQueue.pop(); // 获取任务
// 任务逻辑处理，组成一个网络包packet，调用下游RPC接口
ServiceConnection c = CPool.GetServiceConnection(); // 从Service连接池获取一个Service连接
c.Send(packet); // 通过Service连接发送报文执行RPC请求
CPool.PutServiceConnection(c); // 将Service连接放回Service连接池
}
```

似曾相识吧？没错，只要对服务连接池进行少量改动：

获取Service连接的CPool.GetServiceConnection()【返回任何一个可用Service连接】改为

CPool.GetServiceConnection(longid)【返回id取模相关联的Service连接】

这样的话，就能够保证同一个数据例如uid的请求落到同一个服务Service上。

总结，改造连接池，解决数据不一致的问题

由于数据库层面的读写并发，引发的数据库与缓存数据不一致的问题，（本质就是后发生的请求先返回），可以通过小的改动解决

1）修改服务 Service 连接池，id 取模选取服务连接，能够保证 同一个数据的读写都落在同一个后端服务上

2）修改数据库的连接池，id取模选取DB连接，能够保证同一个数据的读写在数据库层面是串行的

提问：取模是否会影响服务的可用性

答案：不会，当有下游服务器挂掉的时候，服务连接池能够检测到连接的可用性，取模时把不可用的服务连接排除掉

提问：取模访问服务与取模访问DB,是否会影响各连接上的请求的负载均衡?

答案：不会，只要访问数据访问的id 是均衡的，从全局来看，由于id 取模获取各连接的概率也是均衡的，既是负载均衡的

数据库主从不一致，怎么解?

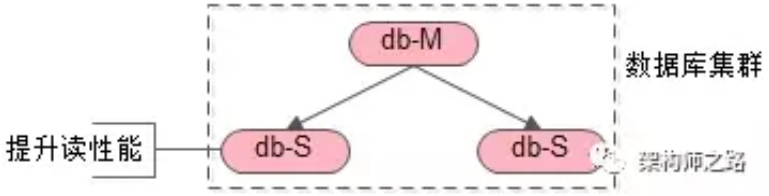
提问：要是数据库的架构做了主从同步，读写分离

写请求写主库，读请求读从库也有可能导致缓存中进入脏数据呀，这种情况怎么解决呢（读写请求根本不落在同一个DB上，并且读写DB有同步时延）？

在聊数据库与缓存一致性问题之前，先聊聊**数据库**主库与从库的一致性问题。

问：常见的数据库集群架构如何？

答：一主多从，主从同步，读写分离

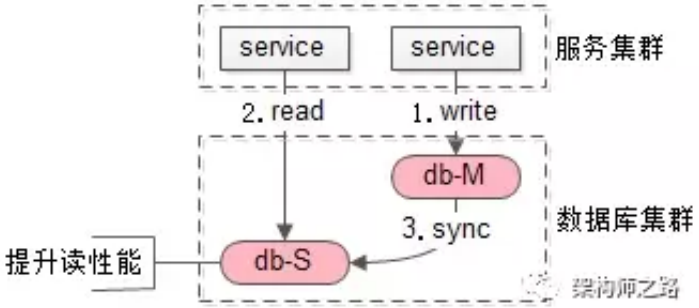


1，一个主库提供写服务
2，多个从库提供读服务，可以增加从库提升读性能
3，主从之间同步数据

任何方案不要忘了本心，加从库的本心，是提升读性能的

为什么会出现不一致？

答：主从同步有延迟，这时在延迟间读从库，可能读到不一致的数据。



1，服务发起了一个写的请求
2，服务又发起了一个读请求，此时同步未完成，读到了一个不一致的脏数据
3，数据库主从同步最后才完成

任何数据冗余，必将引发一致性的问题

如何避免这种主从延时导致的不一致？

常见的方法有这么几种：忽略短时间的不一致，强制读主，选择读主

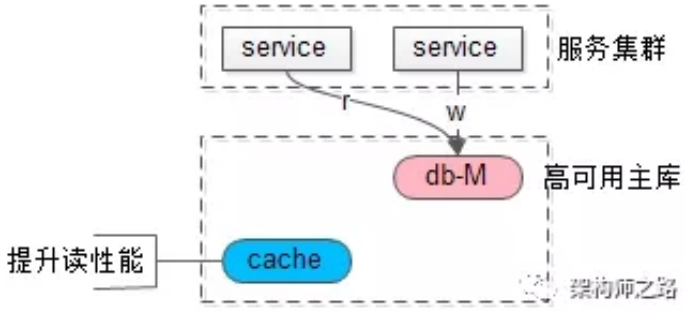
方案一：忽略

任何脱离业务的架构设计都是耍流氓，绝大部分业务，例如：百度搜索，淘宝订单，QQ消息，58帖子都允许短时间不一致。

如果业务能接受，最推崇此法。

如果业务能够接受，别把系统架构搞得太复杂。

方案二：强制读主



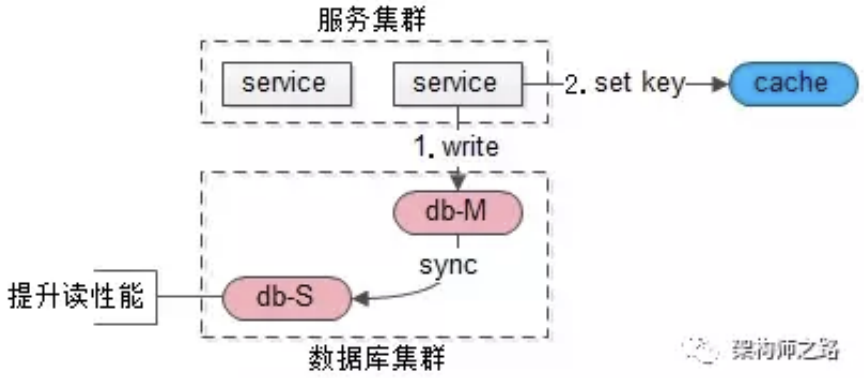
1，使用一个高可用的主库提供数据库服务
2，读和写都落在主库上
3，采用缓存来提升系统读性能

方案：选择性读主

强制读主过于粗暴，毕竟只有少量的写请求，很短时间，可能读取到脏数据

有没有可能实现，只有这一段时间，可能读到从库脏数据的读请求读主，平时读从呢？

可以利用一个缓存记录必须读主的数据

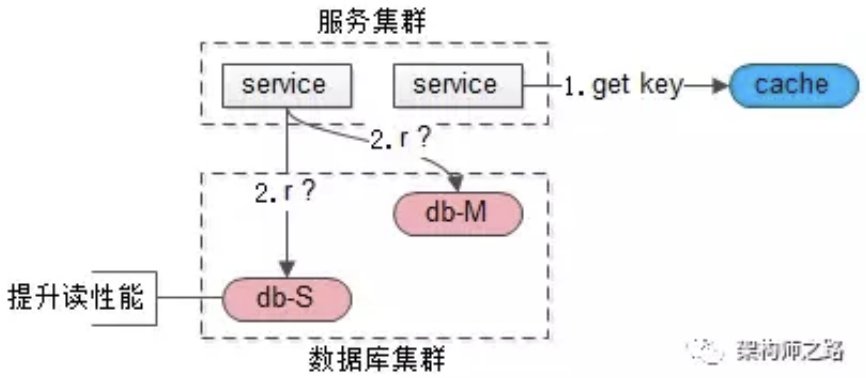


如上图，当写请求发生时：

1，写主库

2，将那个库，那个表，那个主键三个信息拼接一个key设置cache里，这条记录的超时时间，设置为“主从同步延迟”

```
key的格式为“db:table:PK”，假设主从延时为1s，这个key的cache超时时间也为1s。
```



如上图，当读请求发生时：

这是要读那个库，那个表，那个主键的数据呢，也将这三个信息拼接一个key,到cache里去查询

1，cache 里有这个key,说明1s内刚发生过写请求，数据库主从同步可能没有完成，此时就应该去主库查询
2，cache 里没有这个key,说明最近没有发生过写请求，此时就可以去从库查询，以此保证读到的数据一定不是不一致的脏数据

总结：数据库主库和从库不一致，常见有这么几种优化方案：

1，业务可以接受，系统不优化
2，强制读主，高可用主库，用缓存提高读性能
3，在cache里记录那些记录发生过写请求，来路由读主还是读从