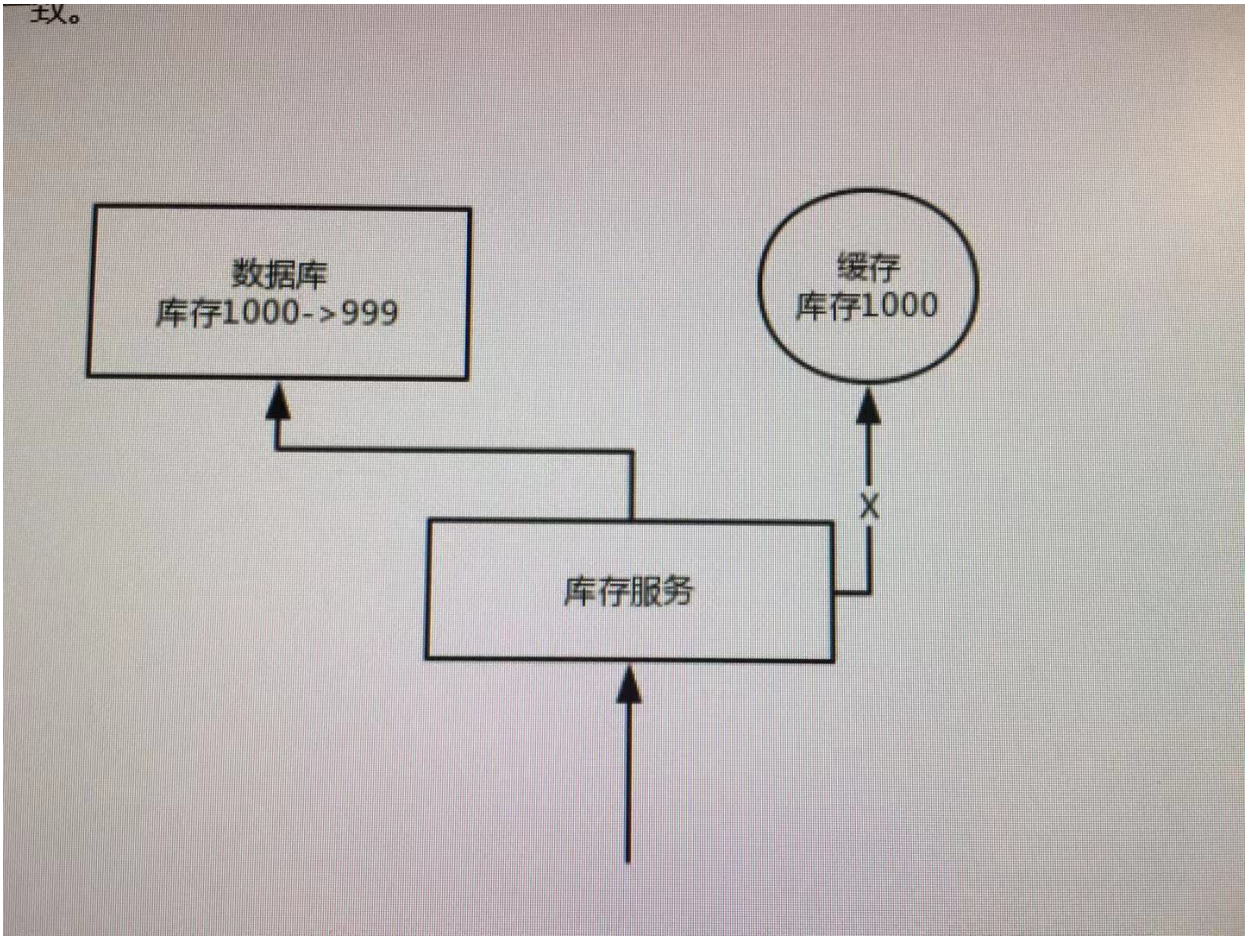
一般来说，如果允许缓存可以稍微的跟数据库偶尔不一致的情况，也就是说如果系统不是严格要求“缓存+数据库”必须保持一致性的话，最好不要做这个方案，即“读请求和写请求串行化”，串行到一个内存队列里去。
串行化可以保证一定不会出现不一致的情况，但是它也会导致系统的吞吐量大幅度降低，用比正常情况下多几倍的机器去支撑线程的一个请求。

Cache Aside Pattern
最经典的缓存+数据库读写的模式，就是cache aside pattern。
读的时候，先读缓存，缓存没有的话，就读数据库，然后取出数据后放入缓存，同时返回响应。
更新的时候，先更新数据库，然后再删除缓存。

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？
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。

最初级的缓存不一致问题及解决方案：
问题：先修改数据库，再删除缓存。如果删除缓存失败了，那么会导致数据库中是新数据，缓存中是旧数据，数据就出现了不一致。



解决思路：先删除缓存，再修改数据库。如果数据库修改失败了，那么数据库中是旧数据，缓存中是空的，那么数据不会不一致。因为读的时候没有缓存，则读数据库中就数据，然后更新到缓存中。

比较复杂的数据不一致问题分析：

数据发生了变更，先删除了缓存，然后要去修改数据库，此时还没修改。一个请求过来，去读缓存，发现缓存空了，去查询数据库，查到了修改前的旧数据，放到了缓存中。随后数据变更的程序完成了数据库的修改。完了，数据库和缓存中的数据不一样了。

为什么上亿流量高并发场景下，缓存会出现这个问题？？

只有在对一个数据在并发进行读写的时候，才可能会出现这个问题。其实如果说你的并发量很低的话，特别是读并发低，每天访问量就1次万次，那么很少的情况下，会出现刚才描述的那种不一致的场景。但是问题是，如果每天是上亿的流量，每秒的并发读是几万，每秒只要有数据更新的请求，就可能会出现上述的数据库+缓存不一致的情况。

解决方案如下：

更新数据的时候，根据数据的唯一标识，将操作路由之后，发送到一个jvm内部队列中。读取数据的时候，如果发现数据不在缓存中，那么将重新读取数据+更新缓存的操作，根据唯

一标识路由之后，也发送同一个jvm内部队列之中。
一个队列对应一个工作线程，每个工作线程串行拿到对应的操作，然后一条一条的执行。这样的话，一个数据变更的操作，先删除缓存，然后再去更新数据库，但是还没完成更新。此时如果一个读请求过来，读到了空的缓存，那么可以先将缓存更新的请求发送到队列中，此时会在队列中积压，然后同步等待缓存更新完成。
这里有一个优化点，一个队列中，其实多个更新缓存请求串在一起是没意义的，因此可以做过滤，如果发现队列中已经有一个更新缓存的请求了，那么就不用再放个更新请求操作进去了，直接等待前面的更新操作请求完成即可。

待那个队列对应的工作线程完成了上一个操作的数据库的修改之后，才会去执行下一个操作，也就是缓存更新的操作，此时会从数据库中读取最新的值，然后写入缓存中。
如果请求还在等待时间范围内，不断轮询发现可以取到值了，那么就直接返回；如果请求等待的时间超过一定时长，那么这一次直接从数据库中读取当前的旧值。

高并发的场景下，该解决方案要注意的问题：
　　读请求长时阻塞。
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。
如果一个内存队列中可能积压的更新操作特别多，那么就要加机器，让每个机器上部署的服务实例处理更少的操作，那么每个内存队列中积压的更新操作就会越少。