大家好，我是楼仔！

今天写的这个主题内容，其实非常基础，但是**作为高并发非常重要的几个场景，绝对绕不开**，估计大家面试时，也经常会遇到。

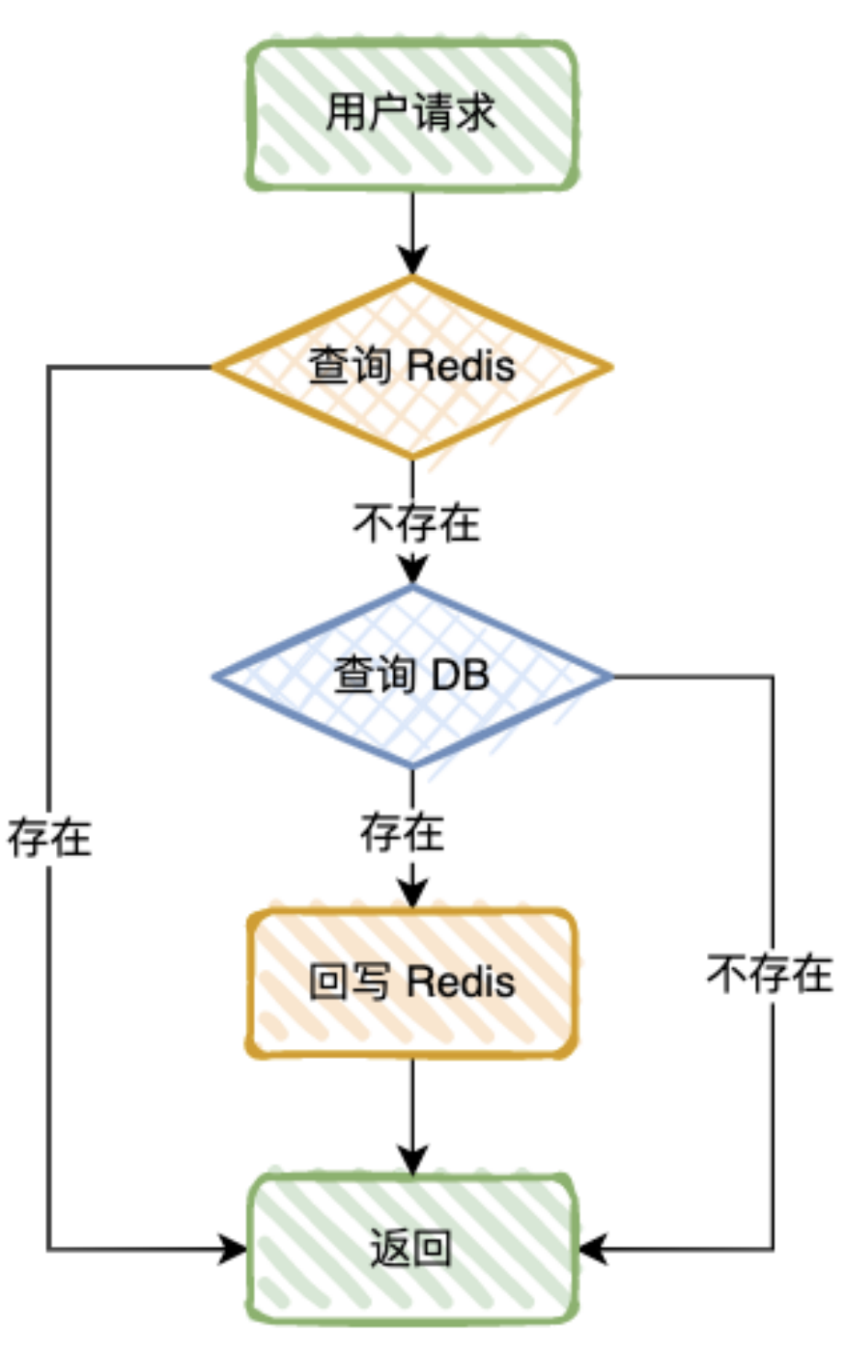
这个主题的文章，网上非常多，本来想直接转载一篇，但是感觉没有合适的，要么文章不够精炼，要么就是精简过头，所以还是自己写一篇吧。

内容虽然基础，但我还是秉承以往的写作风格，参考众多优秀的博客后，打算写一篇能通俗易懂，又不失全面的文章。

前言

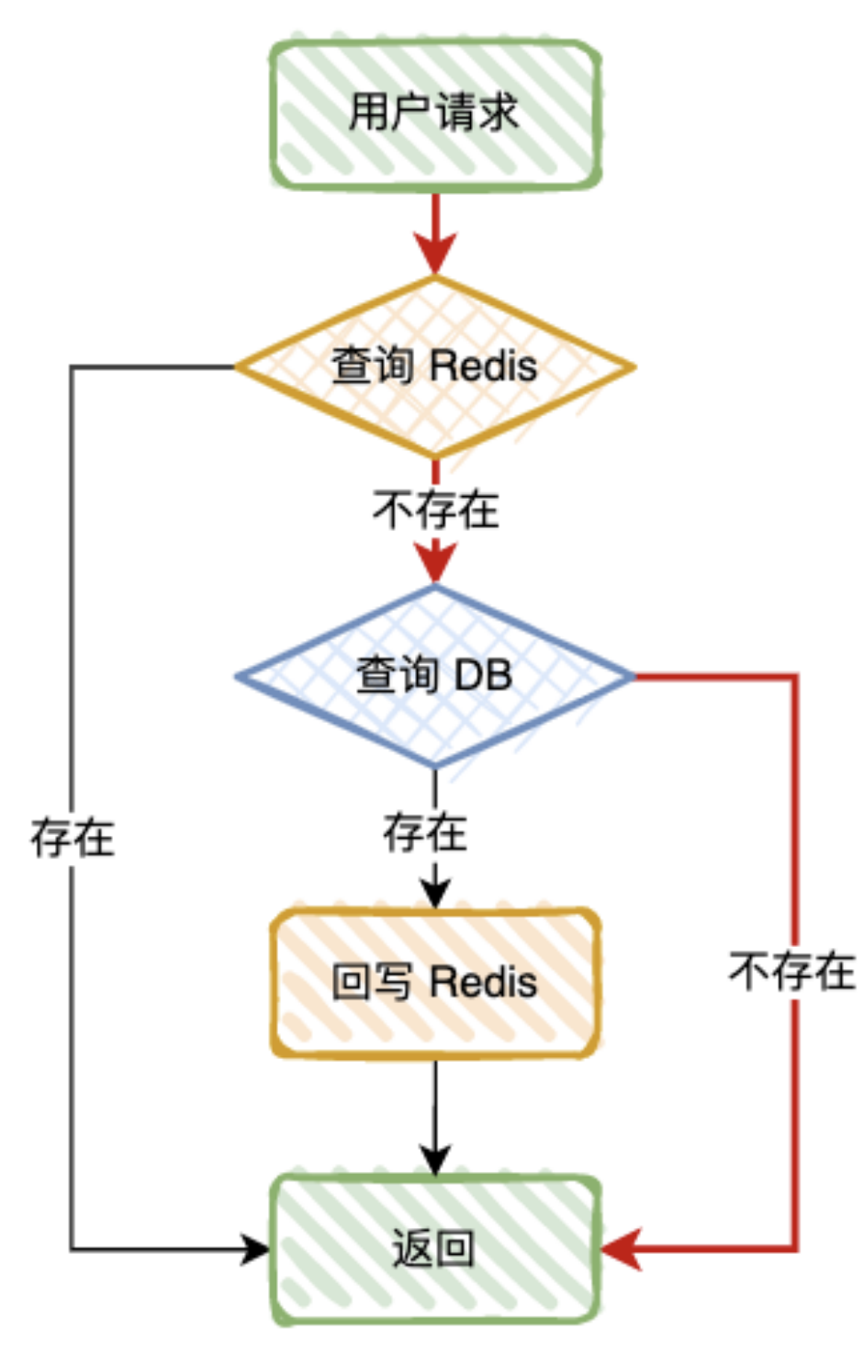
我们先看一下正常情况的查询过程：

- 先查询 Redis，如果查询成功，直接返回，查询不存在，去查询 DB；
- 如果 DB 查询成功，数据回写 Redis，查询不存在，直接返回。



缓存穿透

> 定义：当查询数据库和缓存都无数据时，因为数据库查询无数据，出于容错考虑，不会将结果保存到缓存中，因此**每次请求都会去查询数据库**，这种情况就叫做缓存穿透。



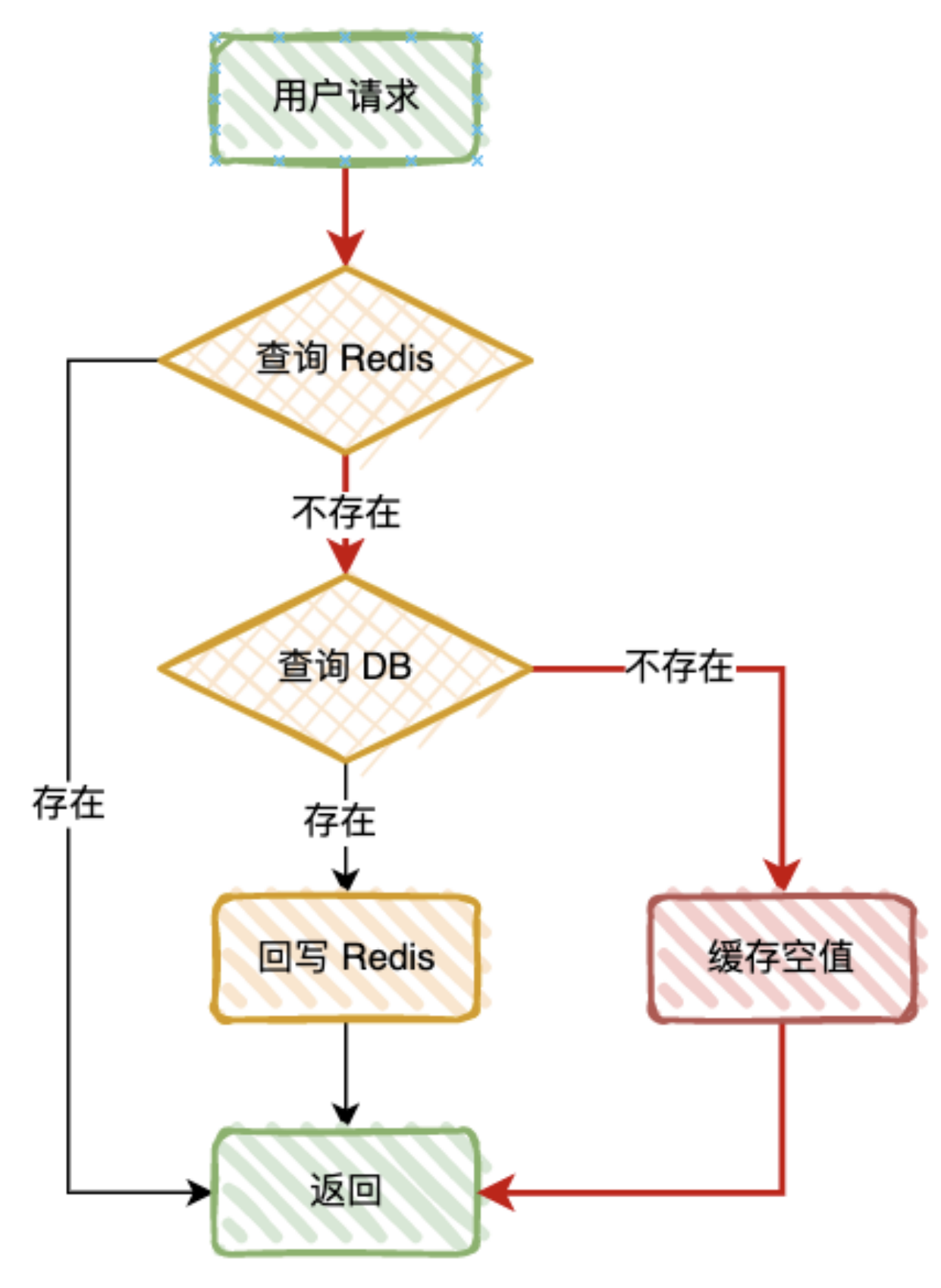
红色的线条，就是缓存穿透的场景，**当查询的 Key 在缓存和 DB 中都不存在时，就会出现这种情况。**

可以想象一下，比如有个接口需要查询商品信息，如果有恶意用户模拟不存在的商品 ID 发起请求，瞬间并发量很高，估计你的 DB 会直接挂掉。

可能大家第一反应就是对入参进行正则校验，过滤掉无效请求，对！这个没错，那有没有其它更好的方案呢？

缓存空值

当我们从数据库中查询到空值时，**我们可以向缓存中回种一个空值**，为了避免缓存被长时间占用，需要给这个空值加一个比较短的过期时间，例如 3~5 分钟。



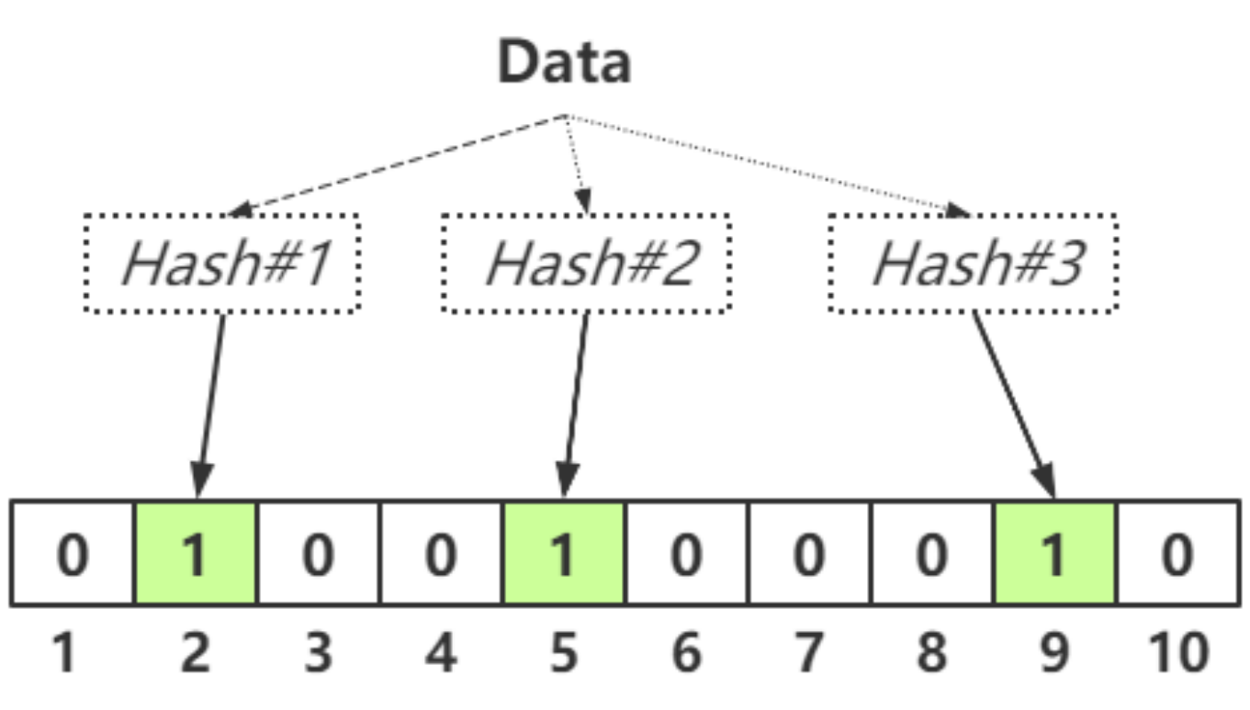
不过这个方案有个问题，当大量无效请求穿透过来时，缓存内就会有有大量的空值缓存，如果缓存空间被占满了，还会因剔除掉一些已经被缓存的用户信息，反而会造成缓存命中率的下降，**所以这个方案，需要评估缓存容量。**

如果缓存空值不可取，这时你可以考虑使用布隆过滤器。

布隆过滤器

布隆过滤器是由一个可变长度为 N 的二进制数组与一组数量可变 M 的哈希函数构成，说的简单粗暴一点，就是一个 Hash Map。

原理相当简单：比如元素 key=#3，假如通过 Hash 算法得到一个为 9 的值，就存在这个 Hash Map 的第 9 位元素中，通过标记 1 标识该位已经有数据，如下图所示，0 是无数据，1 是有数据。



所以通过该方法，会得到一个结论：**在 Hash Map 中，标记的数据，不一定存在，但是没有标记的数据，肯定不存在。**

为什么“标记的数据，不一定存在”呢？**因为 Hash 冲突！**

比如 Hash Map 的长度为 100，但是你有 101 个请求，假如你运气好到爆，这 100 个请求刚好均匀打在长度为 100 的 Hash Map 中，此时你的 Hash Map 已经全部标记为 1。

当第 101 个请求过来时，就 100% 出现 Hash 冲突，虽然我没有请求过，但是得到的标记却为 1，导致布隆过滤器没有拦截。

如果需要减少误判，可以增加 Hash Map 的长度，并选择却分度更高的 Hash 函数，比如多次对 key 进行 hash。

除了 Hash 冲突，**布隆过滤器其实会带来一个致命的问题：布隆过滤器更新失败。**

比如有一个商品 ID 第一次请求，当 DB 中存在时，需要在 Hash Map 中标记一下，但是由于网络原因，导致标记失败，那么下次这个商品 ID 重新发起请求时，请求会被布隆过滤器拦截，比如这个是双11的爆款商品库存，明明有 10W 件商品，你却提示库存不存在，领导可能会说“明天你可以不用来了”。

所以如果使用布隆过滤器，在对 Hash Map 进行数据更新时，需要保证这个数据能 100% 更新成功，可以通过异步、重试的方式，所以这个方案有一定的实现成本和风险。

缓存击穿

> 定义：**某个热点缓存在某一时刻恰好失效**，然后此时刚好有大量的并发请求，此时这些请求将会给数据库造成巨大的压力，这种情况就叫做缓存击穿。

这个其实和“缓存穿透”流程图一样，只是这个的出发点是“某个热点缓存在某一时刻恰好失效”，比如某个非常热门的爆款商品，缓存突然失效，流量直接全部打到 DB，**造成某一时刻数据库请求量过大，更强调瞬时性。**

解决问题的方法主要有 2 种：

1. **分布式锁**：只有拿到锁的第一个线程去请求数据库，然后插入缓存，当然每次拿到锁的时候都要去查询一下缓存有没有，这种在高并发场景下，个人不太建议用分布式锁，会影响查询效率；
2. **设置永不过期**：对于某些热点缓存，我们可以设置永不过期，这样就能保证缓存的稳定性，但需要注意在数据更改之后，要及时更新此热点缓存，不然就会造成查询结果的误差，比如热门商品，都先预热到数据库，后续再下线掉。

网上还有“缓存续期”的方式，比如缓存 30 分钟失效，可以搞个定时任务，每 20 分钟跑一次，感觉这种方式不伦不类，仅供大家参考。

缓存雪崩

> 定义：**在短时间内有大量缓存同时过期，导致大量的请求直接查询数据库**，从而对数据库造成了巨大的压力，严重情况下可能会导致数据库宕机的情况叫做缓存雪崩。

如果说“缓存击穿”是单兵反抗，那“缓存雪崩”就是集体起义了，那什么情况会出现缓存雪崩呢？

1. 短时间内有大量缓存同时过期；
2. 缓存服务宕机，导致某一时刻发生大规模的缓存失效。

那么有哪些解决方案呢？

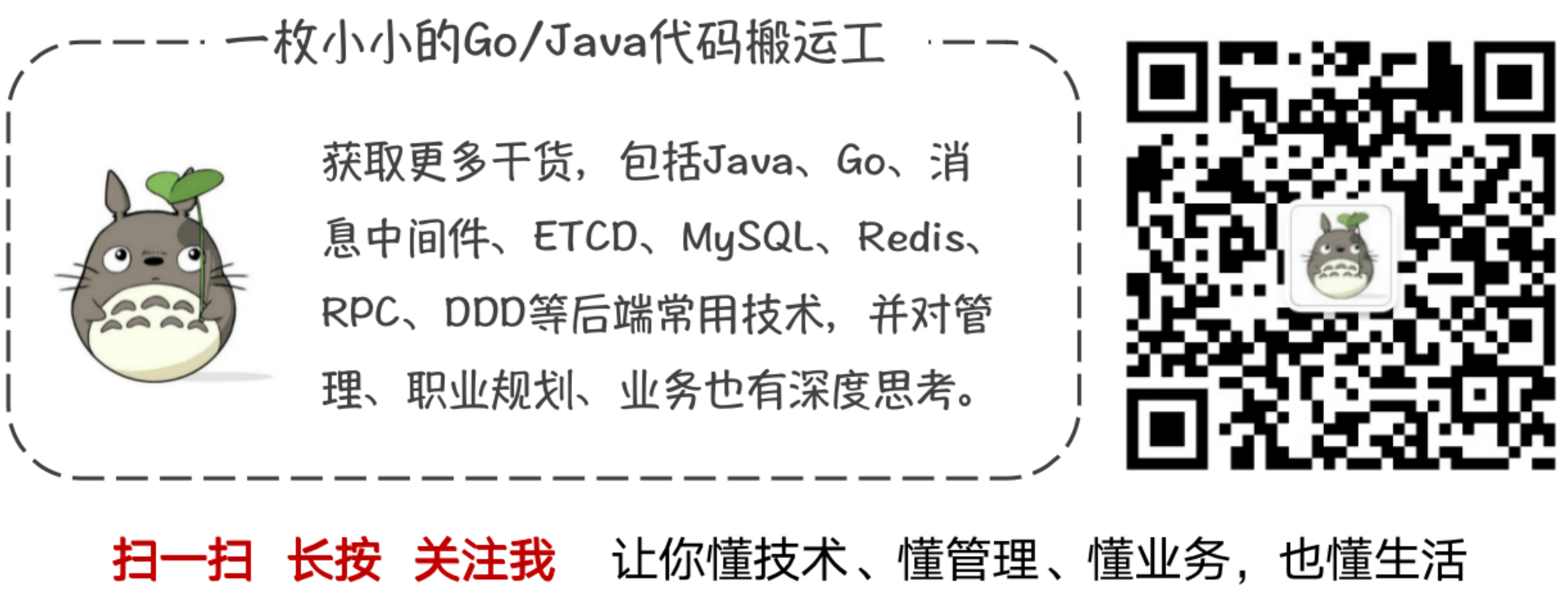
1. **缓存添加随机时间**：可在设置缓存时添加随机时间，比如 0~60s，这样就可以极大的避免大量的缓存同时失效；
2. **分布式锁**：加一个分布式锁，第一个请求将数据持久化到缓存后，其它的请求才能进入；
3. **限流和降级**：通过限流和降级策略，减少请求的流量；
4. **集群部署**：Redis 通过集群部署、主从策略，主节点宕机后，会切换到从节点，保证服务的可用性。

缓存添加随机时间示例：

```
// 缓存原本的失效时间
int exTime = 10 * 60;
// 随机数生成类
Random random = new Random();
// 缓存设置
jedis.setex(cacheKey, exTime + random.nextInt(1000) , value);
```

学习交流

可以扫下面二维码，关注「楼仔」公众号。



长按二维码，回复 **「加群」**，欢迎一起学习交流哈~~ 👏🏻👏🏻👏🏻

> 尽信书则不如无书，因个人能力有限，难免有疏漏和错误之处，如发现 bug 或者有更好的建议，欢迎批评指正，不吝感激。