实践技巧篇：动手实现一个缓存框架设计

极客时间

扫码试看/订阅

《分布式缓存高手课》视频课程

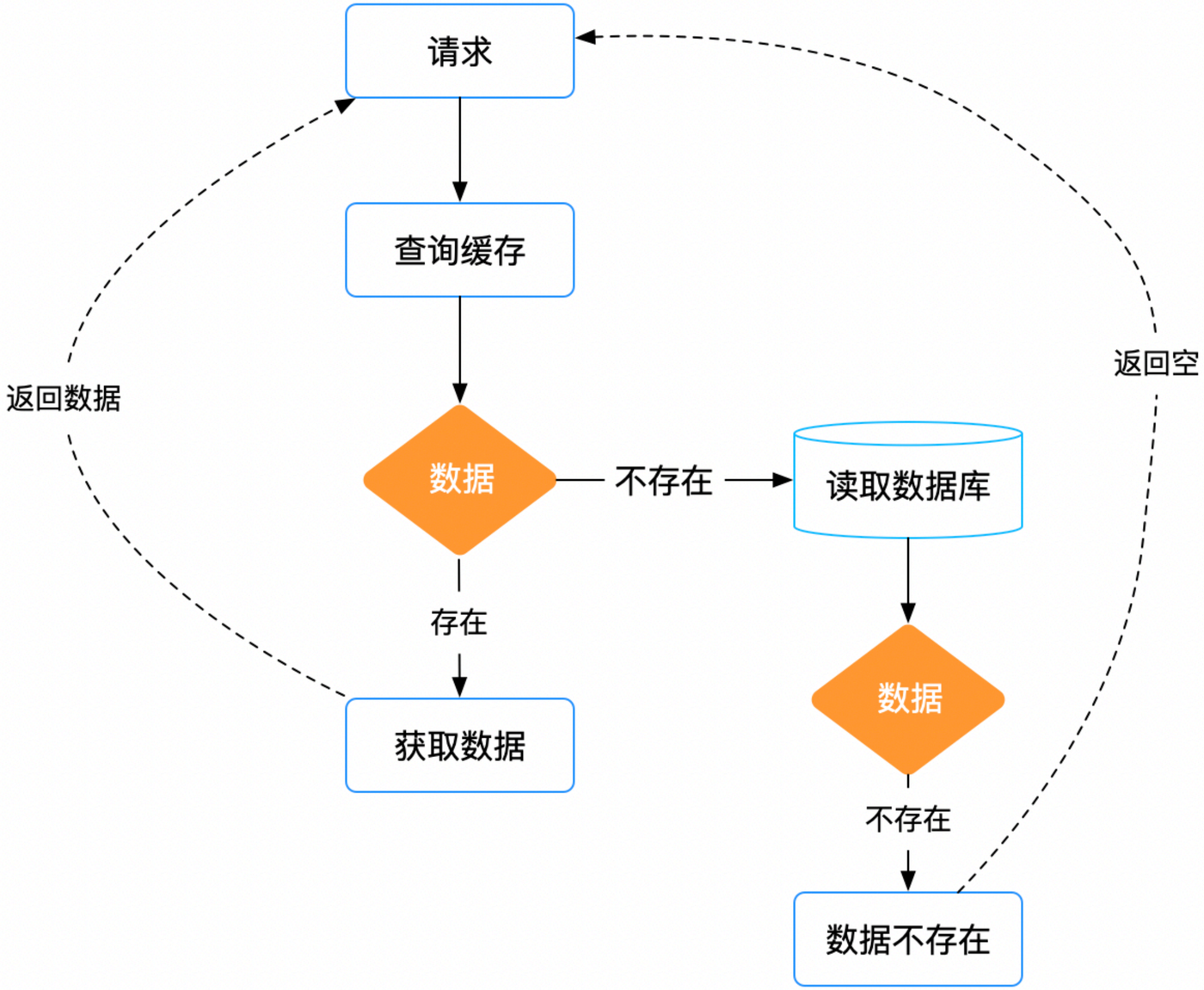
缓存穿透：发生场景和常见的破解招数

缓存穿透：发生场景和常见的破解招数

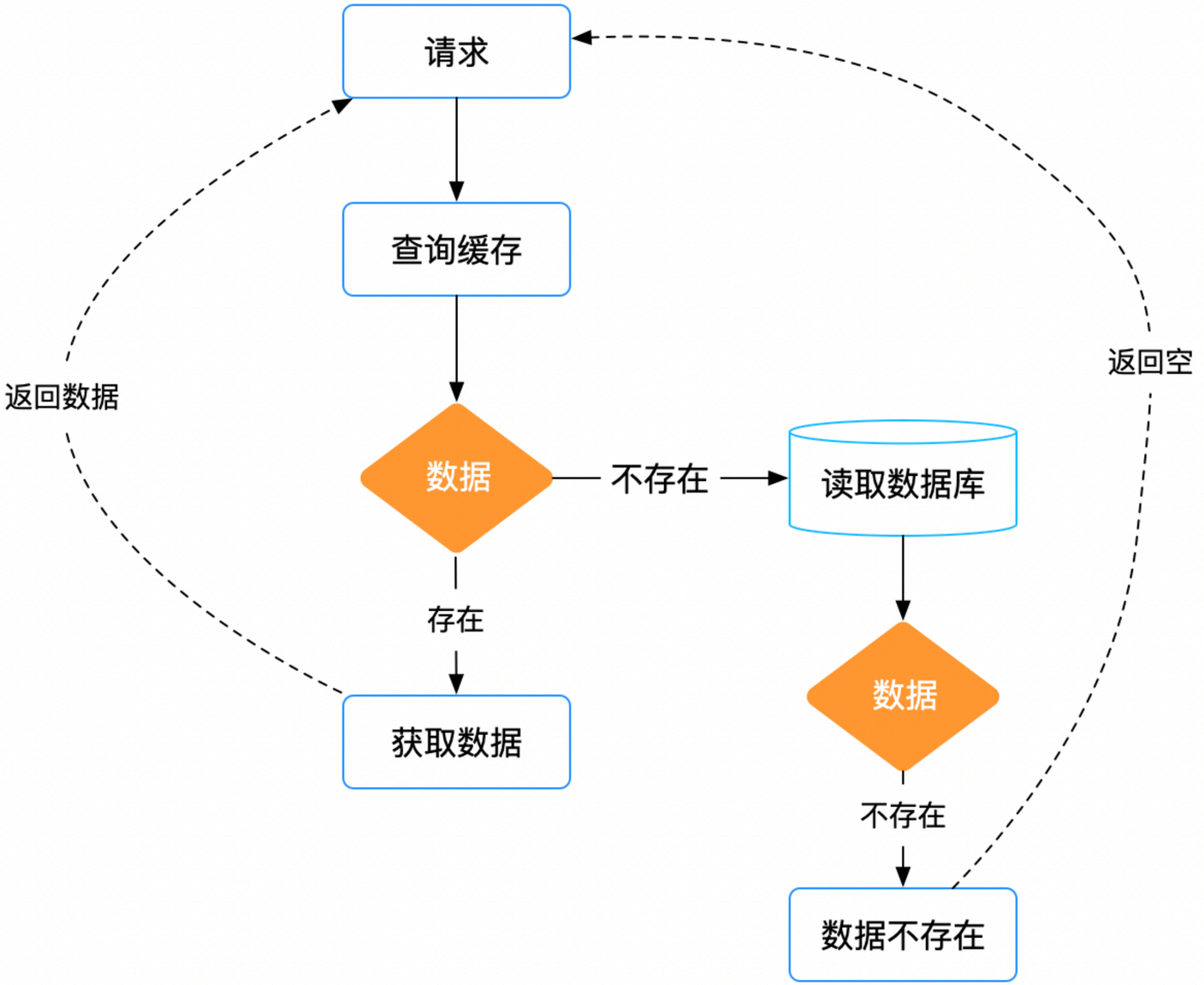
- 什么是缓存穿透?

缓存穿透：发生场景和常见的破解招数

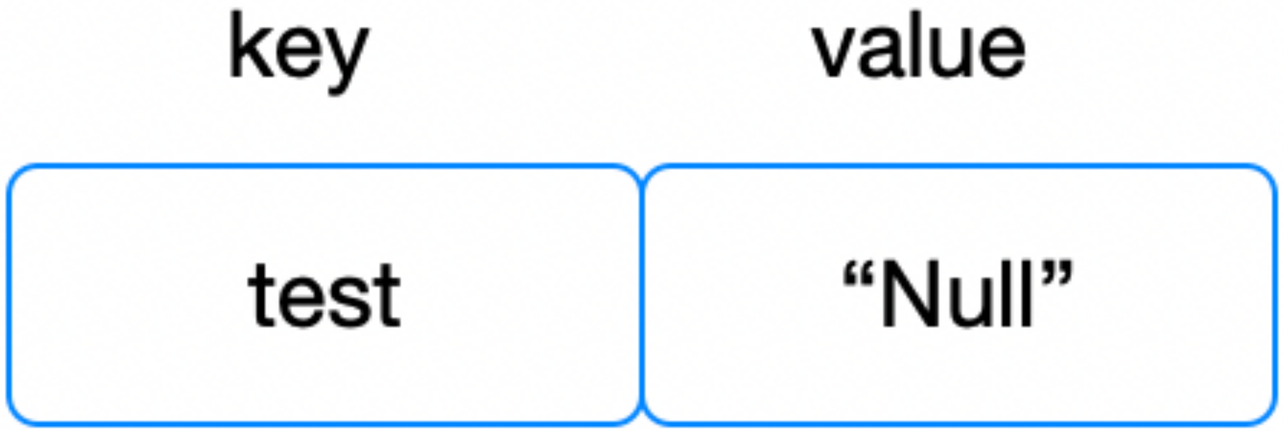
- 发生缓存穿透的场景
  - 小张要查询银行配置信息，因为银行配置信息访问量都比较大，所以缓存在 Redis 中，可是小张在查询缓存的时候输错了银行编码，缓存中没有查到，而数据库中也没有相应的银行配置信息，而小张又不断发起重新查询。

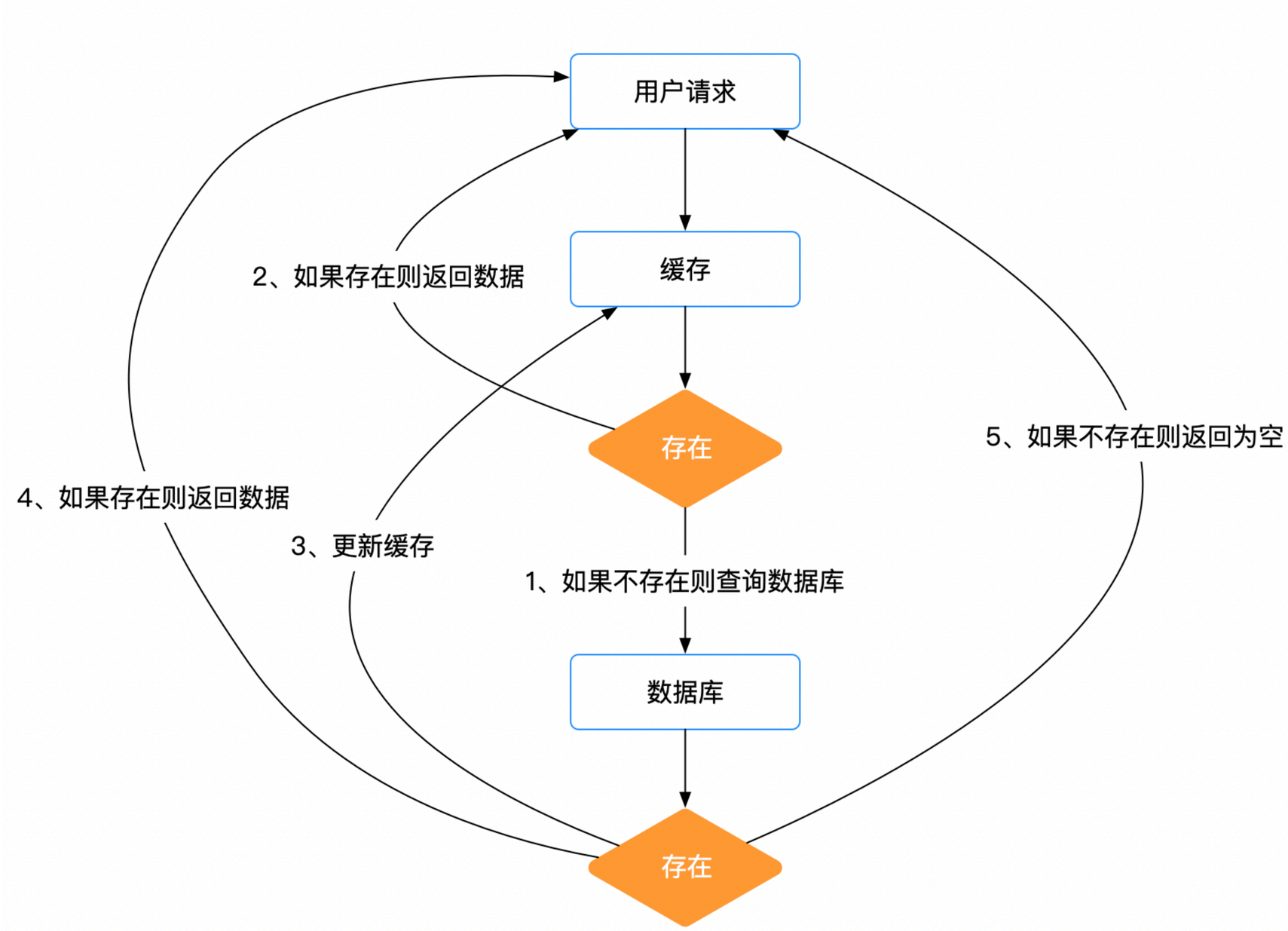


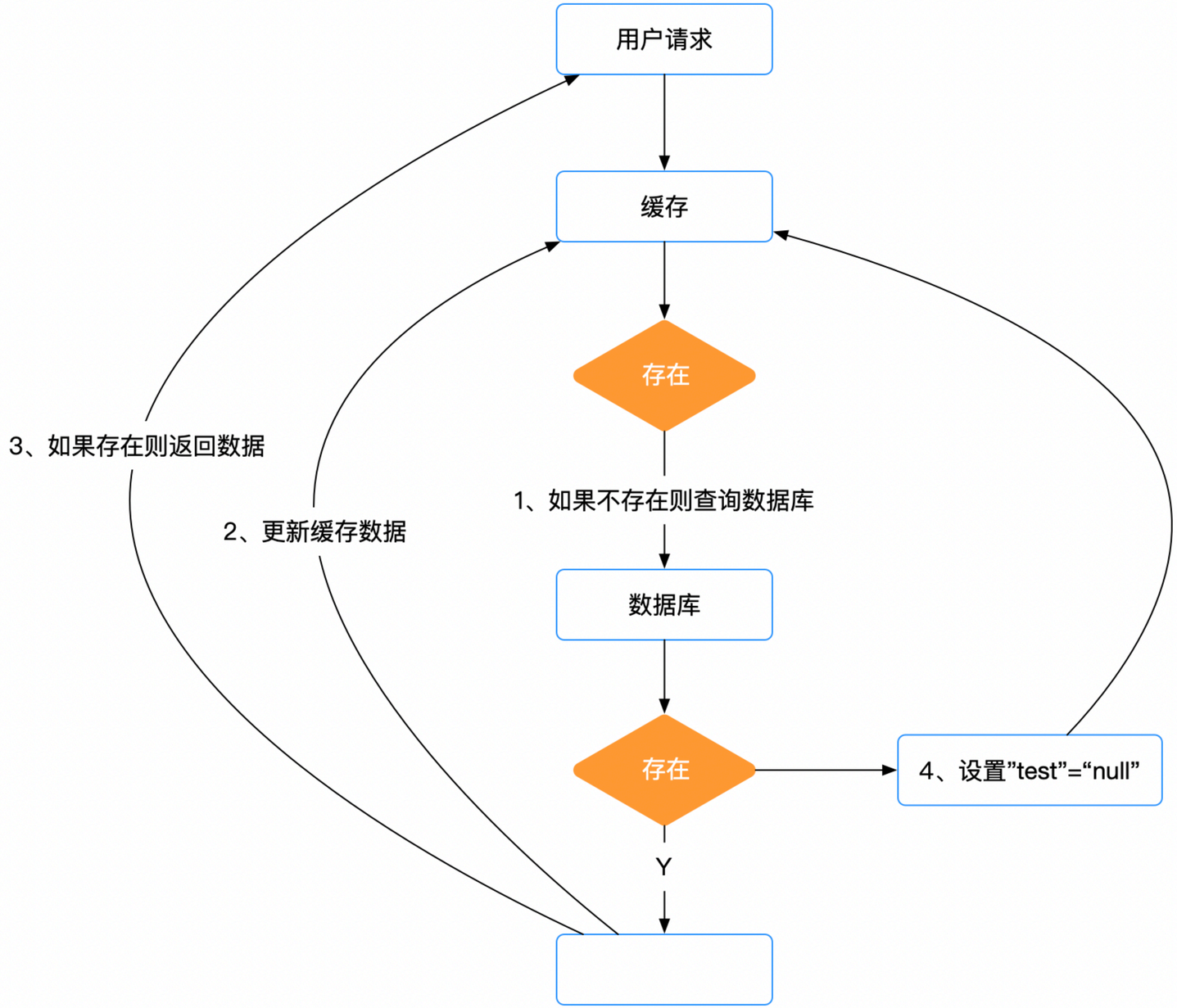
- 如何来解决呢？

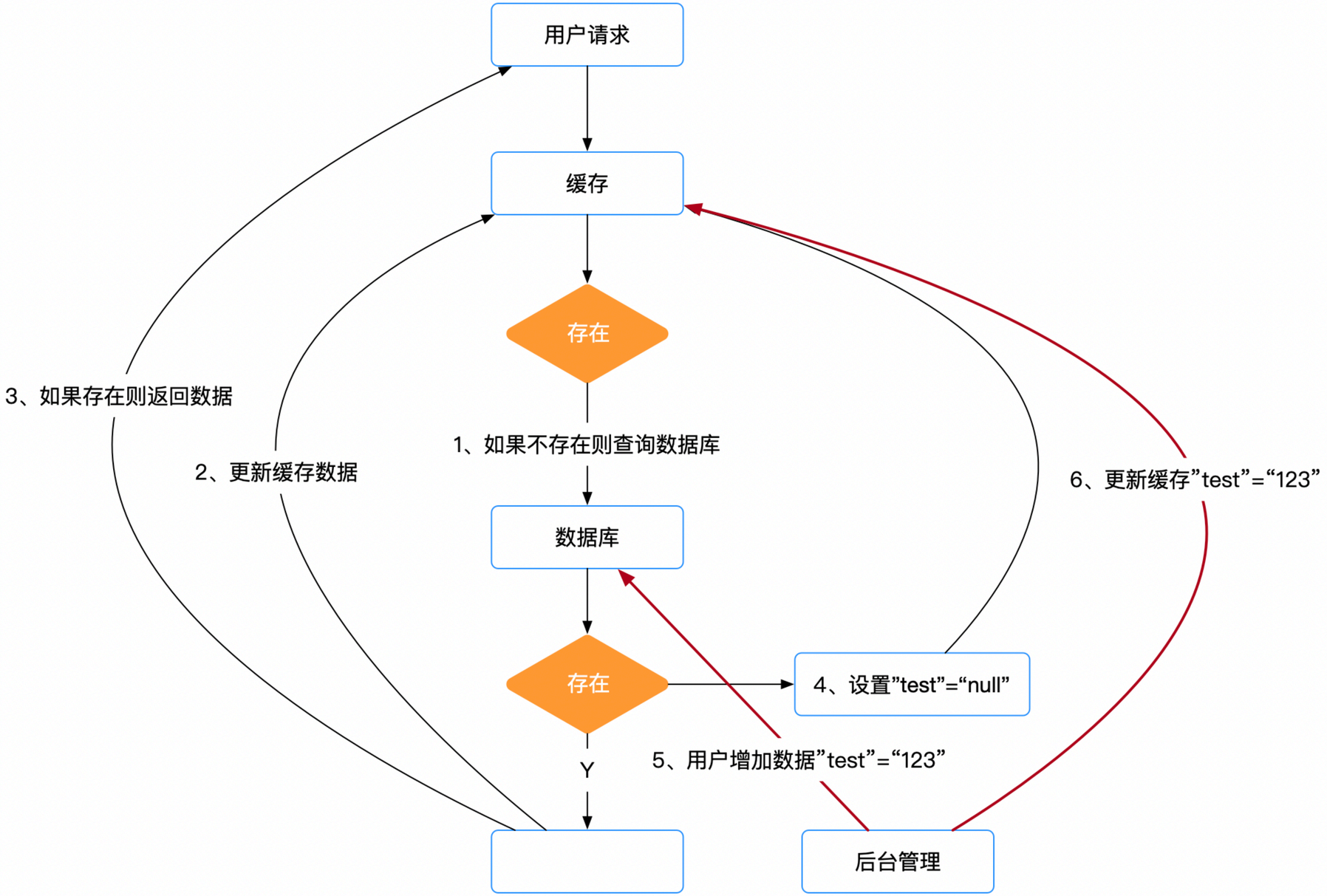


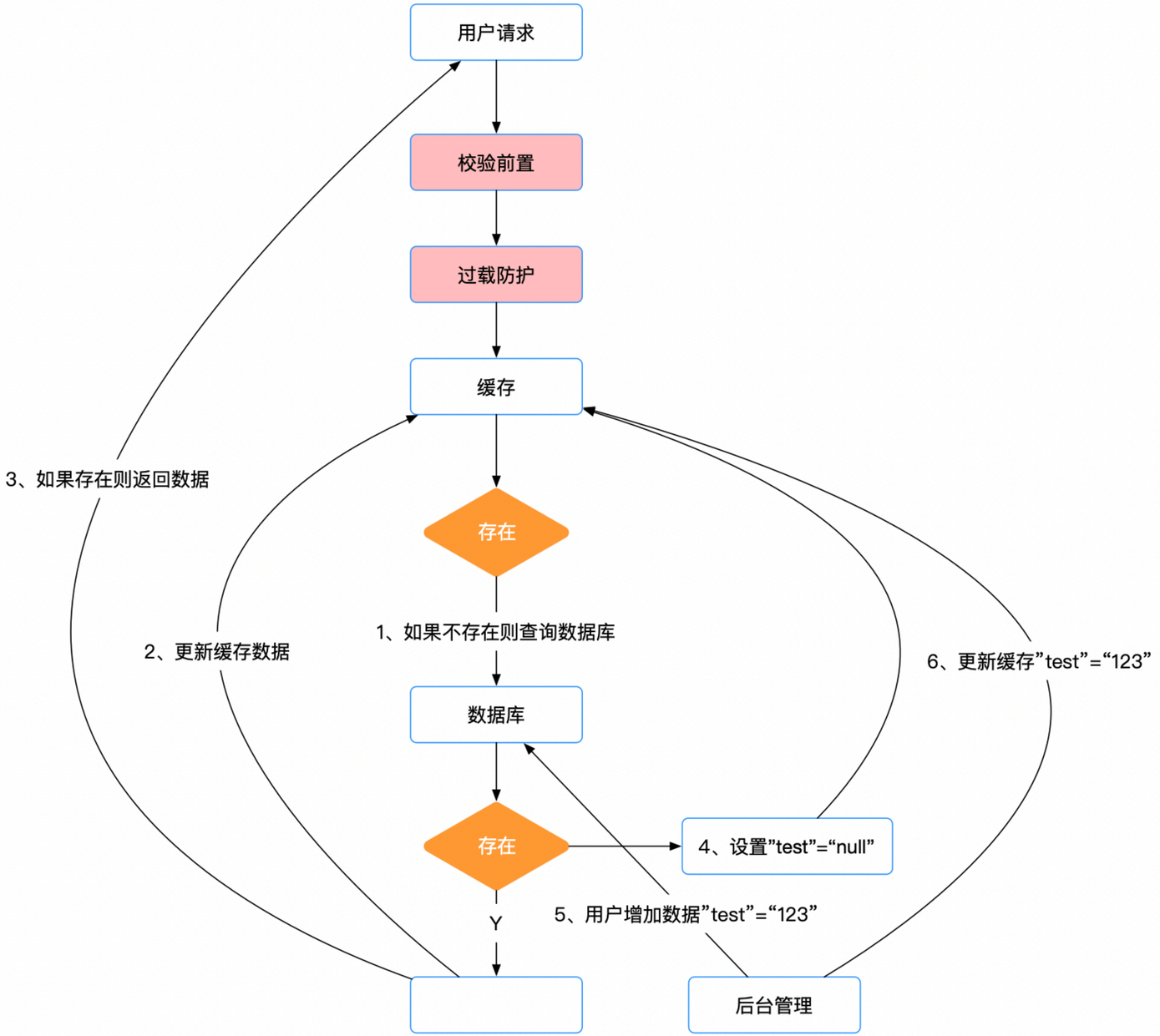
- 解决方案一




用户请求
缓存
2、如果存在则返回数据
存在
5、如果不存在则返回为空
4、如果存在则返回数据
3、更新缓存
1、如果不存在则查询数据库
数据库
存在


用户请求
缓存
存在
1、如果不存在则查询数据库
数据库
存在
4、设置”test”=“null”
Y
2、更新缓存数据
3、如果存在则返回数据

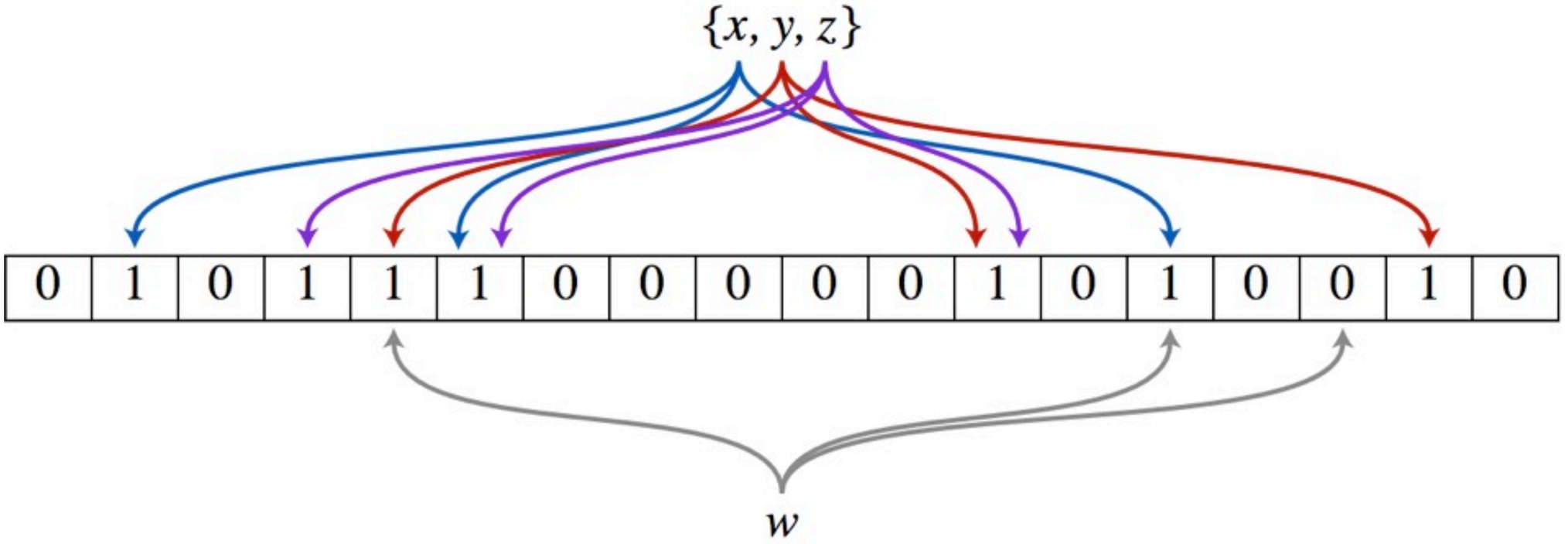

用户请求
缓存
存在
1、如果不存在则查询数据库
数据库
存在
Y
3、如果存在则返回数据
2、更新缓存数据
4、设置”test”=“null”
5、用户增加数据”test”=“123”
6、更新缓存”test”=“123”
后台管理


用户请求
校验前置
过载防护
缓存
存在
1、如果不存在则查询数据库
数据库
存在
4、设置”test”=“null”
Y
5、用户增加数据”test”=“123”
后台管理
3、如果存在则返回数据
2、更新缓存数据
6、更新缓存”test”=“123”

总结

- 事前：
  - 校验前置
  - 空值防范
- 事后：
  - 过载防护

- 解决方案二
  - 布隆过滤器



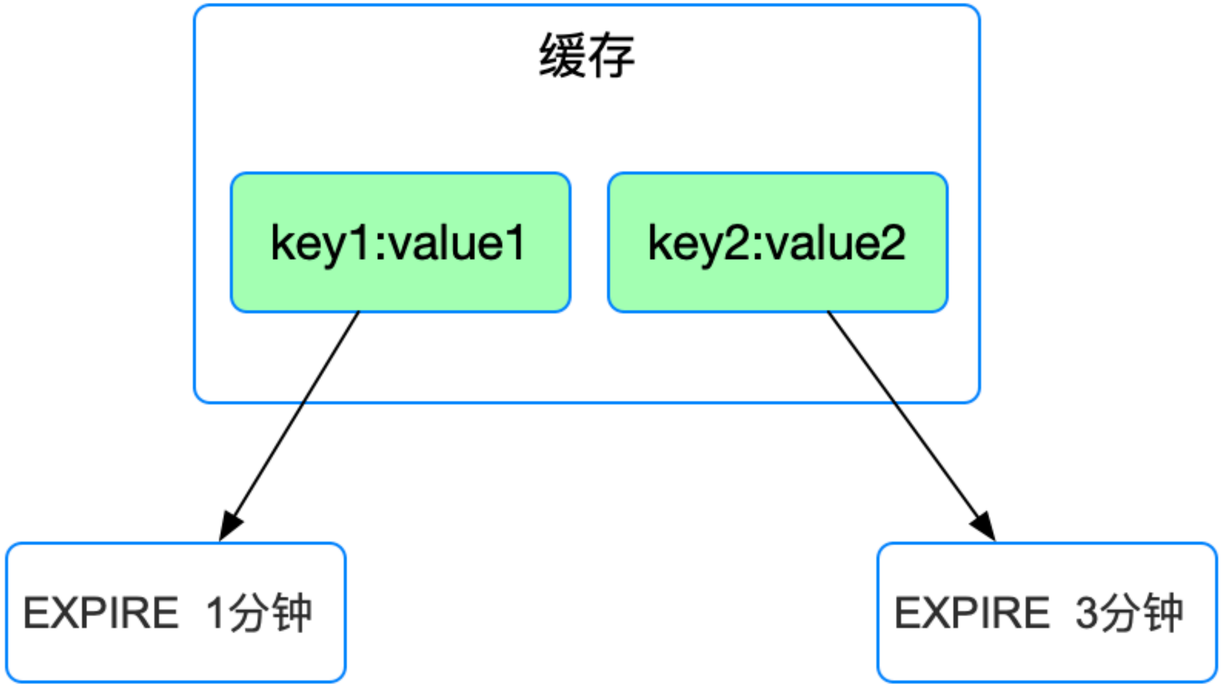
```
<dependency>
        <groupId>com.google.guava</groupId>
        <artifactId>guava</artifactId>
        <version>28.1-jre</version>
</dependency>

private static int size = 5000;
    private static BloomFilter<Integer> demoFilter = BloomFilter.create(Funnels.integerFunnel(),
size);
    public static void main(String[] args) {
        for (int i = 0; i < size; i++) {
            demoFilter.put(i);
        }
        //判断5000数字中是否包含200这个数
        if(demoFilter.mightContain(200)) {
            System.out.println("找到了");
        }
    }
```

缓存雪崩

缓存雪崩

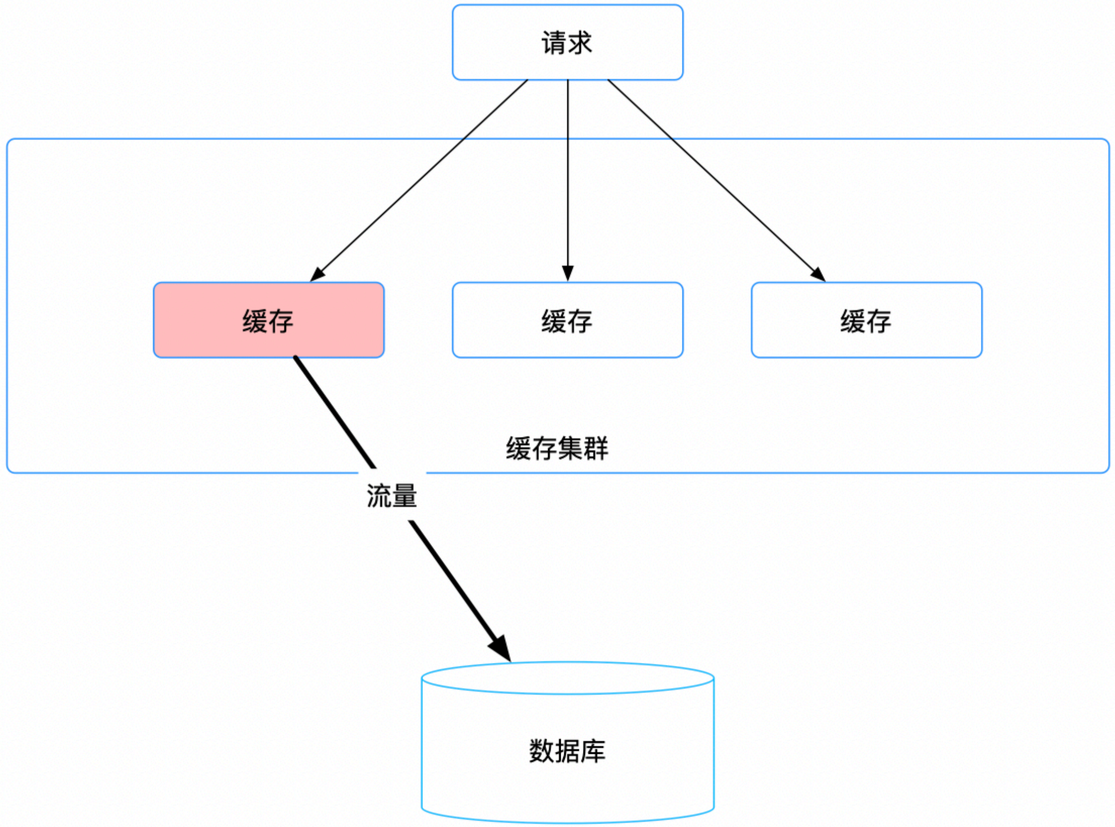
- 缓存雪崩的发生场景之一
  - 双十一抢购，一波商品时间比较集中的放入了缓存。假设缓存一个小时，那么到了凌晨一点钟的时候，这批商品的缓存就都过期了，而对这批商品的访问查询，都落到了数据库上，对于数据库而言，就会产生周期性的压力波峰。

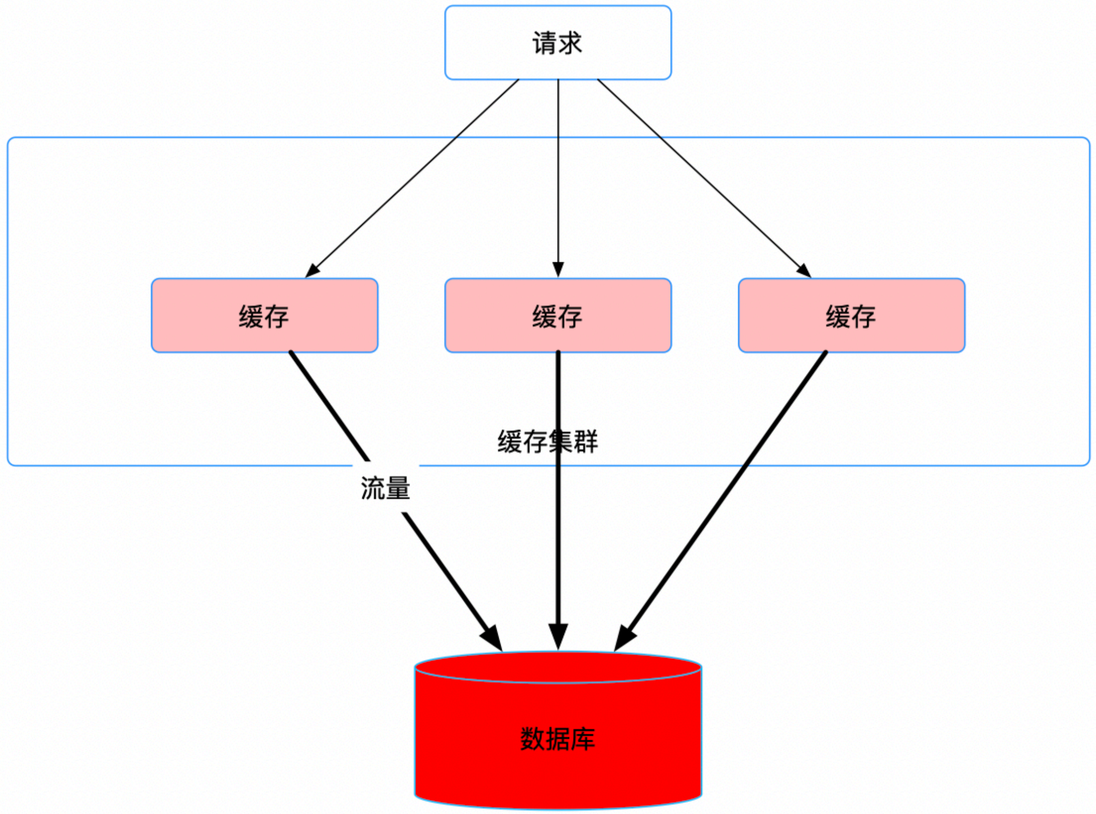
- 方案一
  - 一个是将缓存失效时间分散开，比如我们可以在原有的失效时间基础上，去增加一个随机值，比如1～5分钟随机，这样每一个缓存的过期时间的重复率就会降低，也就很难引发集体失效的事件了。



- 方案二
  - 缓存不过期

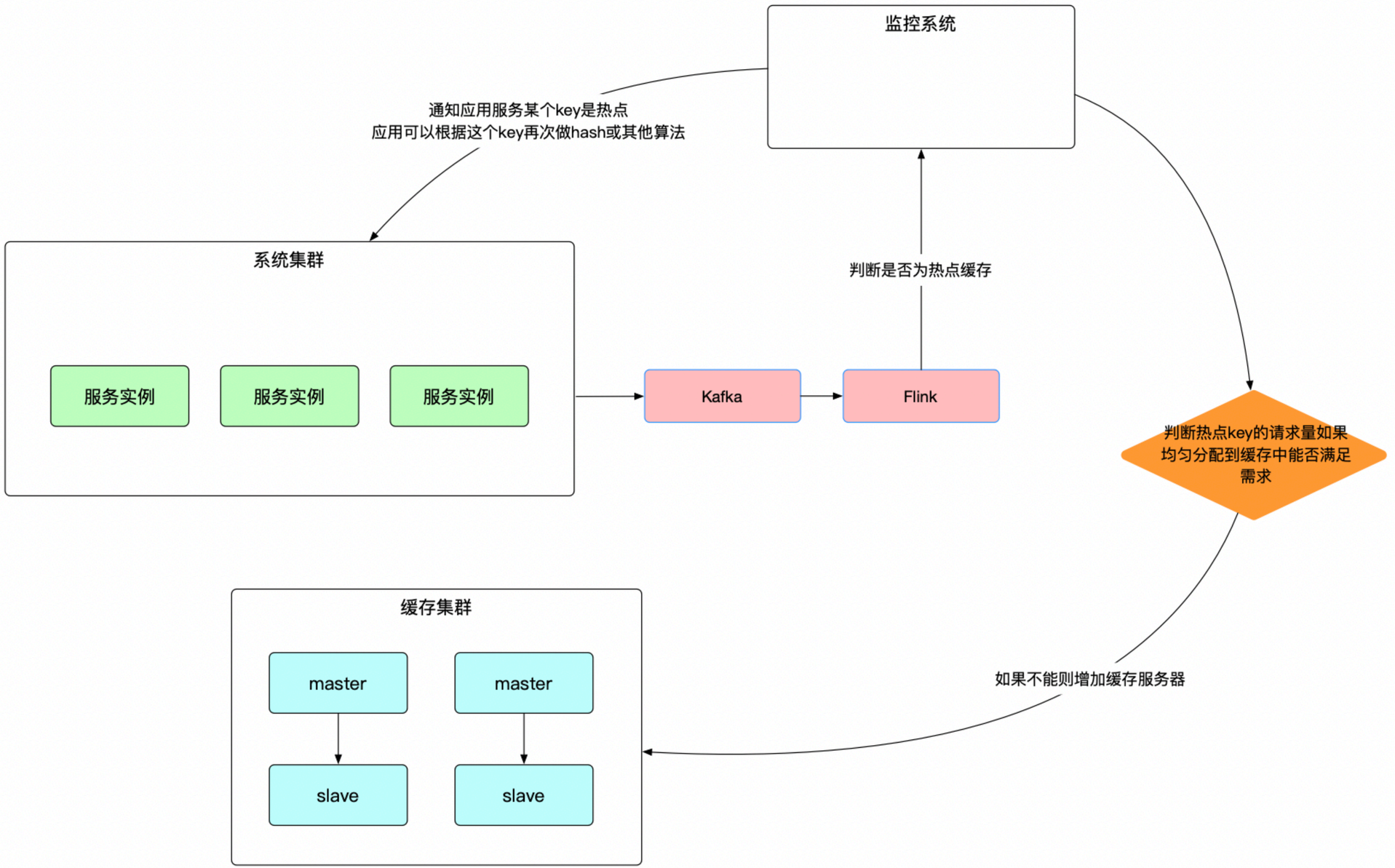
- 缓存雪崩最严重的问题是，在流量高峰的时候，某个缓存服务器的节点出现宕机或出现问题，导致缓存流量透传到数据库上面，更严重的是某个缓存服务器出问题，导致缓存集群出现问题。

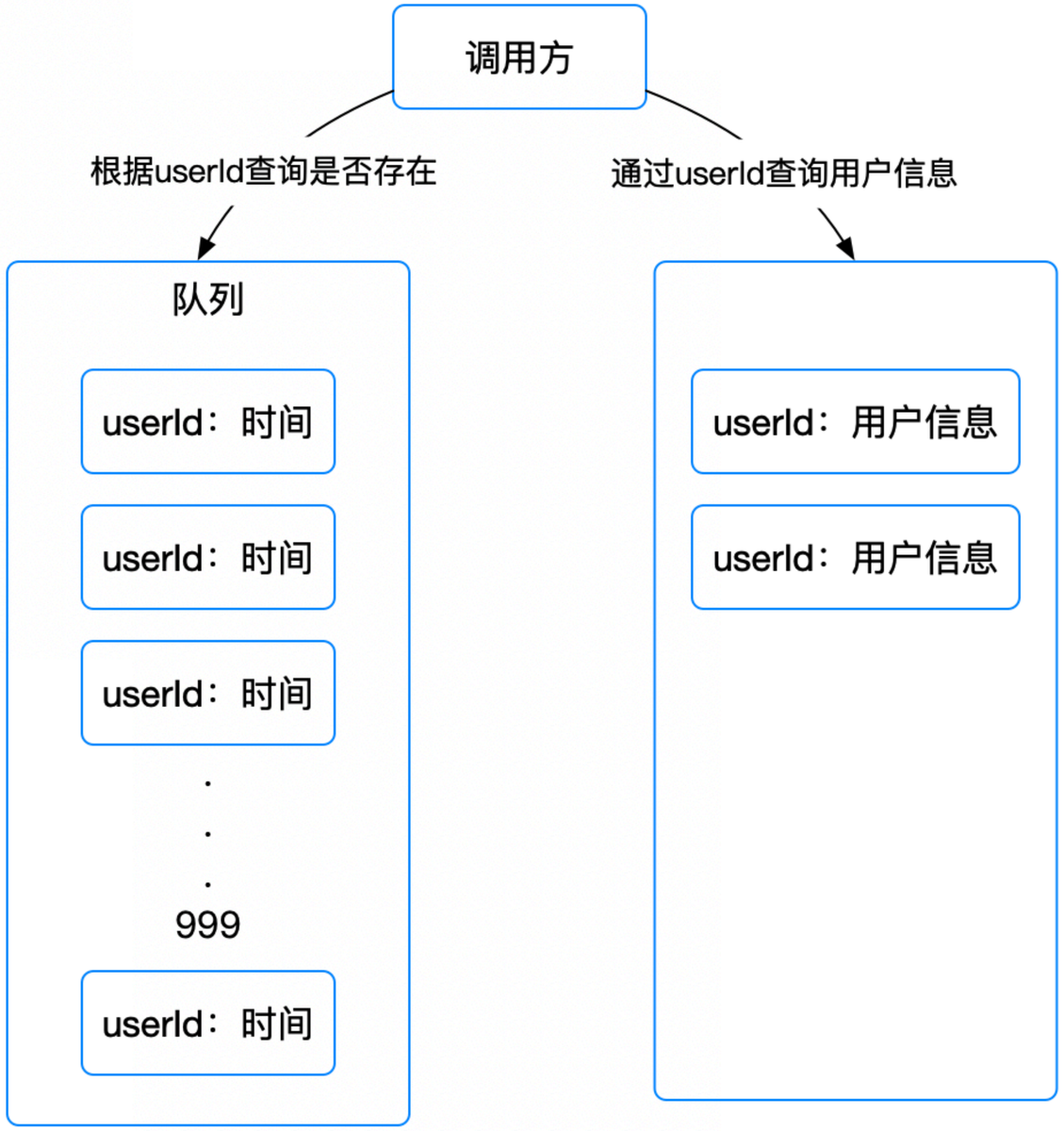




- 缓存雪崩的场景二
  - 一个 key 非常热点，在不停的扛着大并发，大并发集中对这一个点进行访问，当这个 key 在失效的瞬间，持续的大并发就穿破缓存，直接请求数据库，就像在一个屏障上凿开了一个洞。

- 如何发现热点缓存？

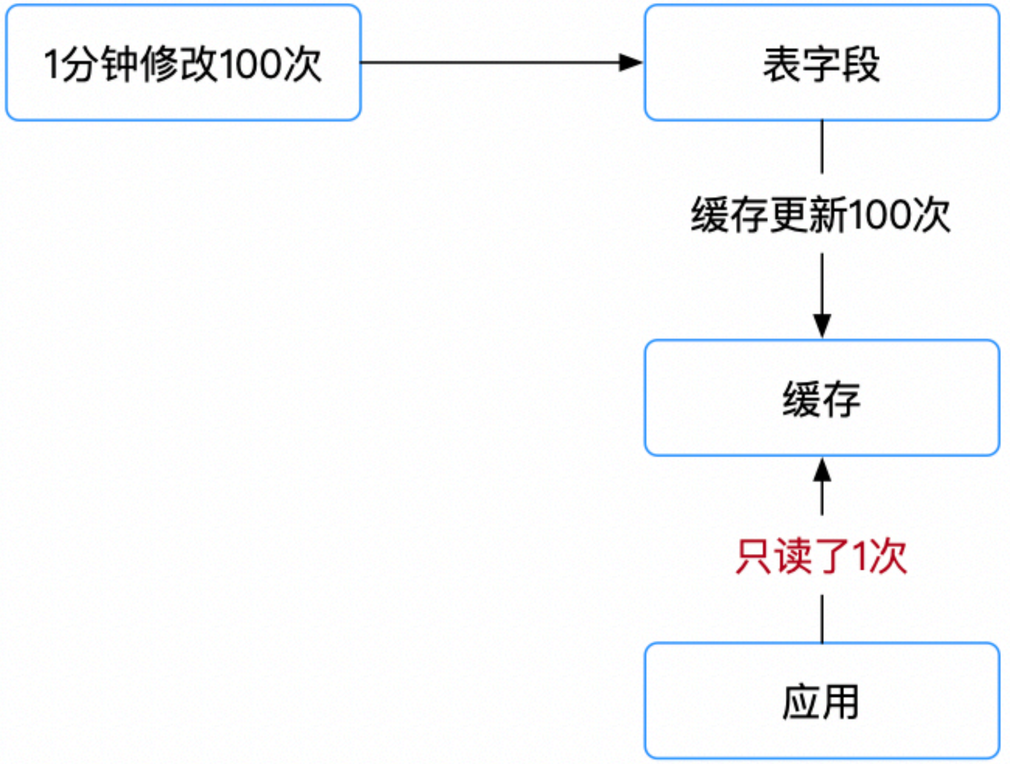

监控系统
通知应用服务某个key是热点
应用可以根据这个key再次做hash或其他算法
系统集群
服务实例
服务实例
服务实例
Kafka
Flink
判断是否为热点缓存
判断热点key的请求量如果
均匀分配到缓存中能否满足
需求
如果不能则增加缓存服务器
缓存集群
master
master
slave
slave


调用方
根据userId查询是否存在
通过userId查询用户信息
队列
userId：时间
userId：时间
userId：时间
999
userId：时间
userId：用户信息
userId：用户信息

多级缓存与数据库之间的一致性如何保证？

- 一个比较经典的数据库与缓存使用的模式
  - 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，先删除缓存，然后更新数据库。

- 一个比较经典的数据库与缓存使用的模式
  - 比较好的方法是在需要的时候删除缓存，不用每次都重新做复杂的计算



- 数据库缓存一致的四个方案

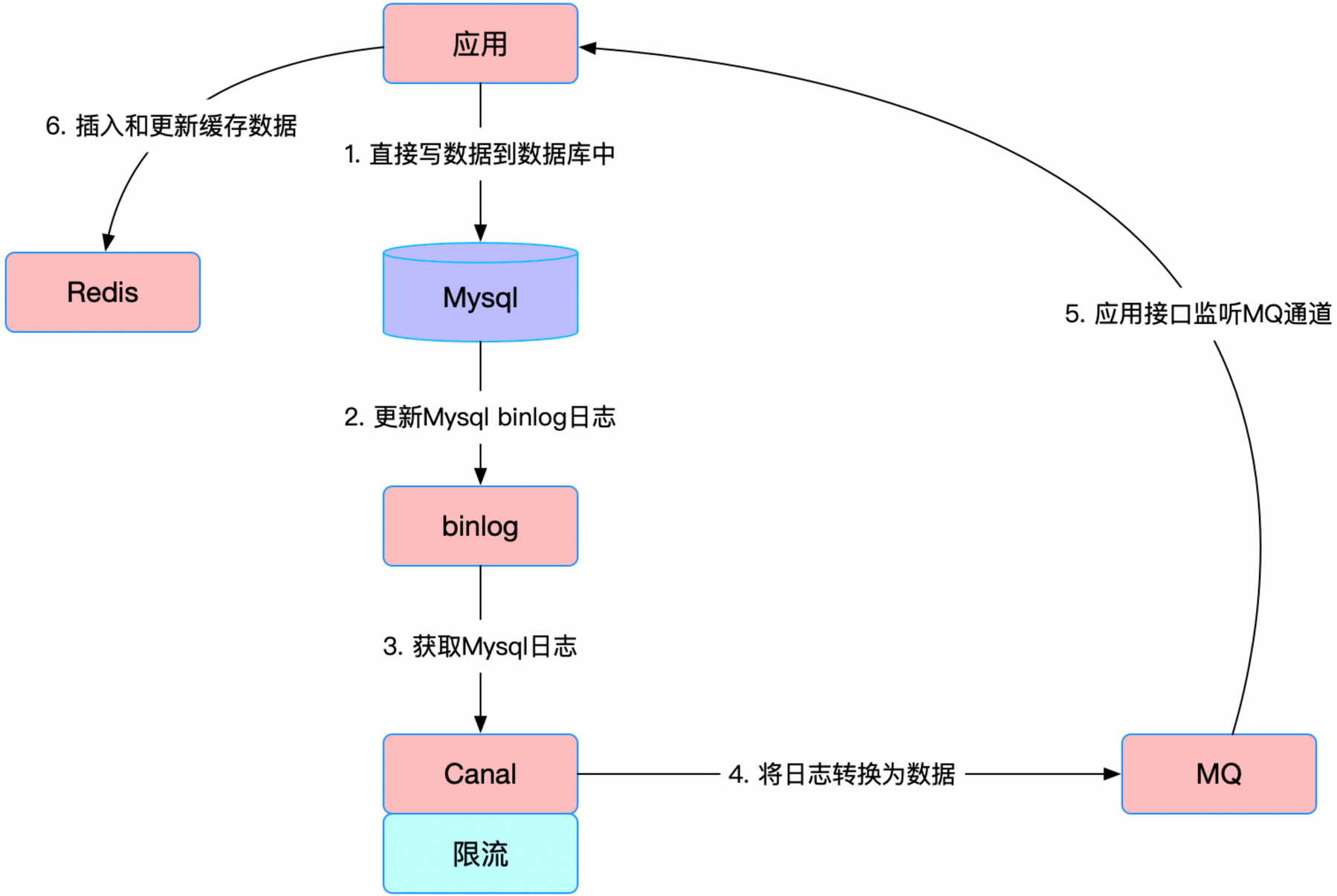
- 数据库缓存一致的四个方案
- 方案一
  - 通过 Redis 的过期时间来更新缓存，MySQL 数据库更新不会触发 Redis 更新，只有当 Redis 的 key 过期后才会重新加载。
- 这种方案的缺点：
  - 据不一致的时间会较长，也会产生一定的脏数据。
  - 完全依赖过期时间，时间太短容易缓存频繁失效，太长容易有长时间更新延迟。

- 数据库缓存一致的四个方案
- 方案二
  - 在方案一的基础上扩展，让 key 的过期时间兜底，在更新 MySQL 的同时更新 Redis。
- 这种方案的缺点：
  - 如果更新Mysql成功，更新Redis失败，就成了方案一。

- 数据库缓存一致的四个方案
- 方案三
  - 在方案二的基础上，对 Redis 更新进行优化，增加消息队列，将 Redis 的更新操作交给 MQ 由消息队列来保证可靠性，异步更新 Redis。
- 这种方案的缺点：
  - 解决不了时序的问题，如果多个业务实例对同一条数据进行更新，数据的先后顺序可能会乱。
  - 引入MQ，增加MQ的维护成本。

- 数据库缓存一致的四个方案
- 方案四
  - 将 MySQL 更新和 Redis 更新放到一个事务中操作，这样就可以达到一致性。
- 这种方案的缺点：
  - MySQL 或 Redis 任何一个环节出现问题，都会造成数据回滚或撤消。
  - 如果网络出现超时，不仅可能会造成数据回滚或撤消，还会引起并发问题。

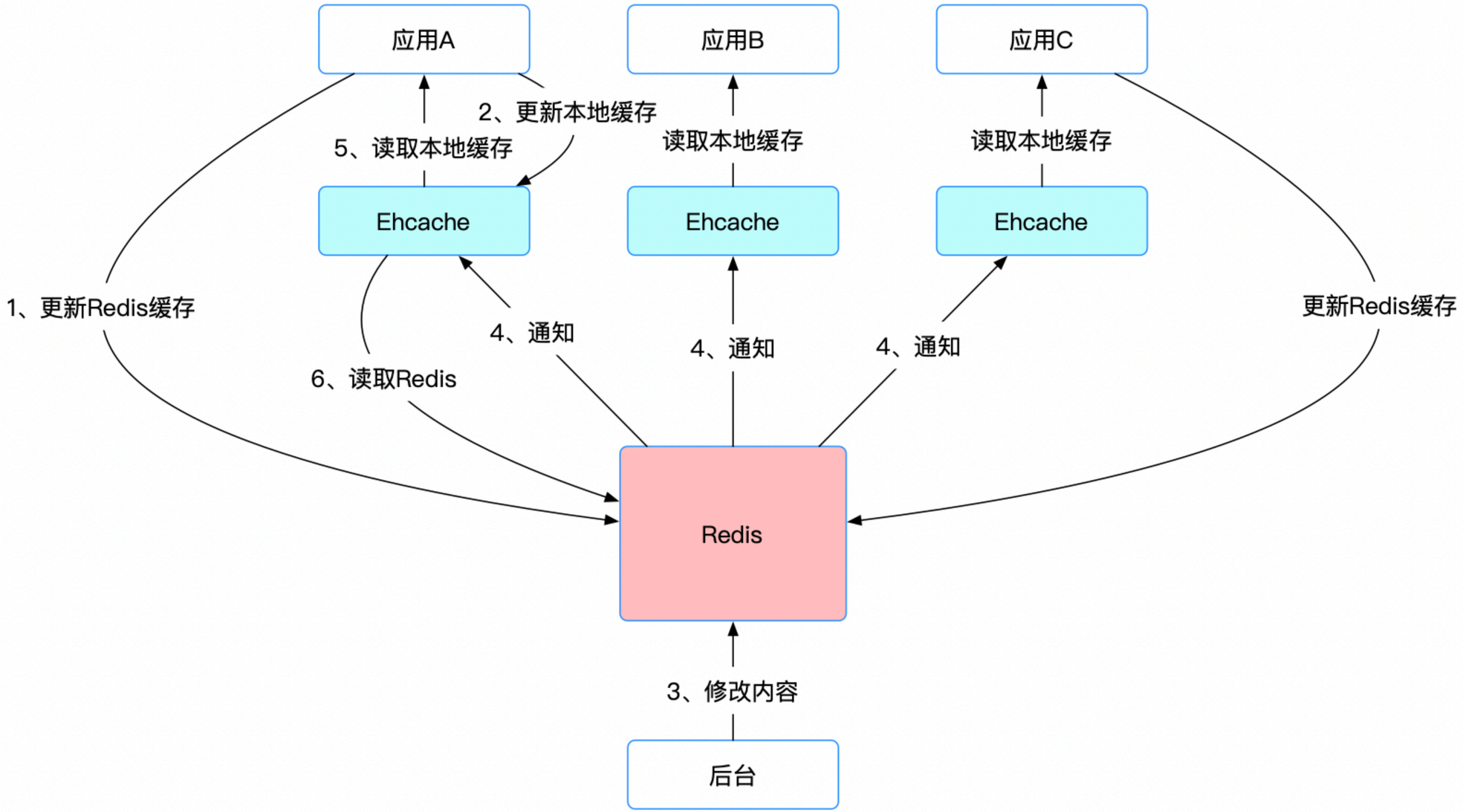
- 数据库缓存一致的四个方案
- 方案五
- 通过订阅 Binlog 来更新 Redis，把我们搭建的消费服务，作为MySQL 的一个 slave，订阅 Binlog，解析出更新内容，再更新到 Redis。
- 这种方案的缺点：
  - 要单独搭建一个同步服务，并且引入 Binlog 同步机制，成本较大。


应用
6. 插入和更新缓存数据
1. 直接写数据到数据库中
Redis
Mysql
5. 应用接口监听MQ通道
2. 更新Mysql binlog日志
binlog
3. 获取Mysql日志
Canal
4. 将日志转换为数据
MQ
限流

Ehcache 和 Redis
如何搭配来实现缓存的最终一致性

为什么要使用本地 Ehcache 缓存和 Redis 的组合?

- 本地缓存和集中式缓存数据更新的策略
  - 广播更新策略
  - 定时更新策略


应用A
应用B
应用C
2、更新本地缓存
5、读取本地缓存
读取本地缓存
读取本地缓存
Ehcache
Ehcache
Ehcache
1、更新Redis缓存
4、通知
4、通知
4、通知
更新Redis缓存
6、读取Redis
Redis
3、修改内容
后台

- 多级缓存使用注意点：
  - 本地缓存必须设置超时时间，必须定时更更新本地缓存，防止因各种原因导致的本地缓存和 Redis 缓存不一致,保证缓存的一致性。
  - 对于并发量不大的场景，Redis 缓存可以不用设置永久缓存，防止因更新失败导致的缓存不一致，以及僵尸类型的 key 占用服务器内存。
  - 项目启动时要清空本地与服务器同步的缓存区域，以保证缓存的一致性。

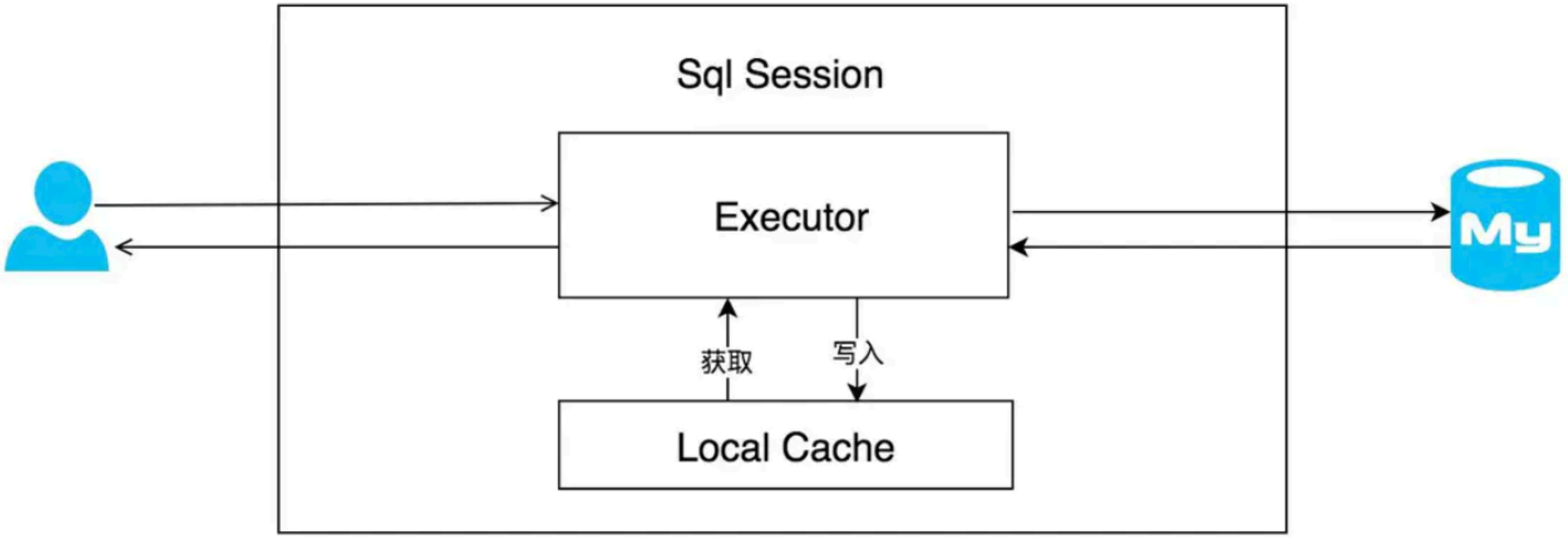
在 MyBatis 下：缓存究竟是怎么玩的？

- 先来再次明确 MyBatis 的几个重要概念：
  - SqlSession：
    - 代表和数据库的一次会话，向用户提供了操作数据库的方法。
  - MappedStatement:
    - 代表要发往数据库执行的指令，可以理解为是 SQL 的抽象表示。

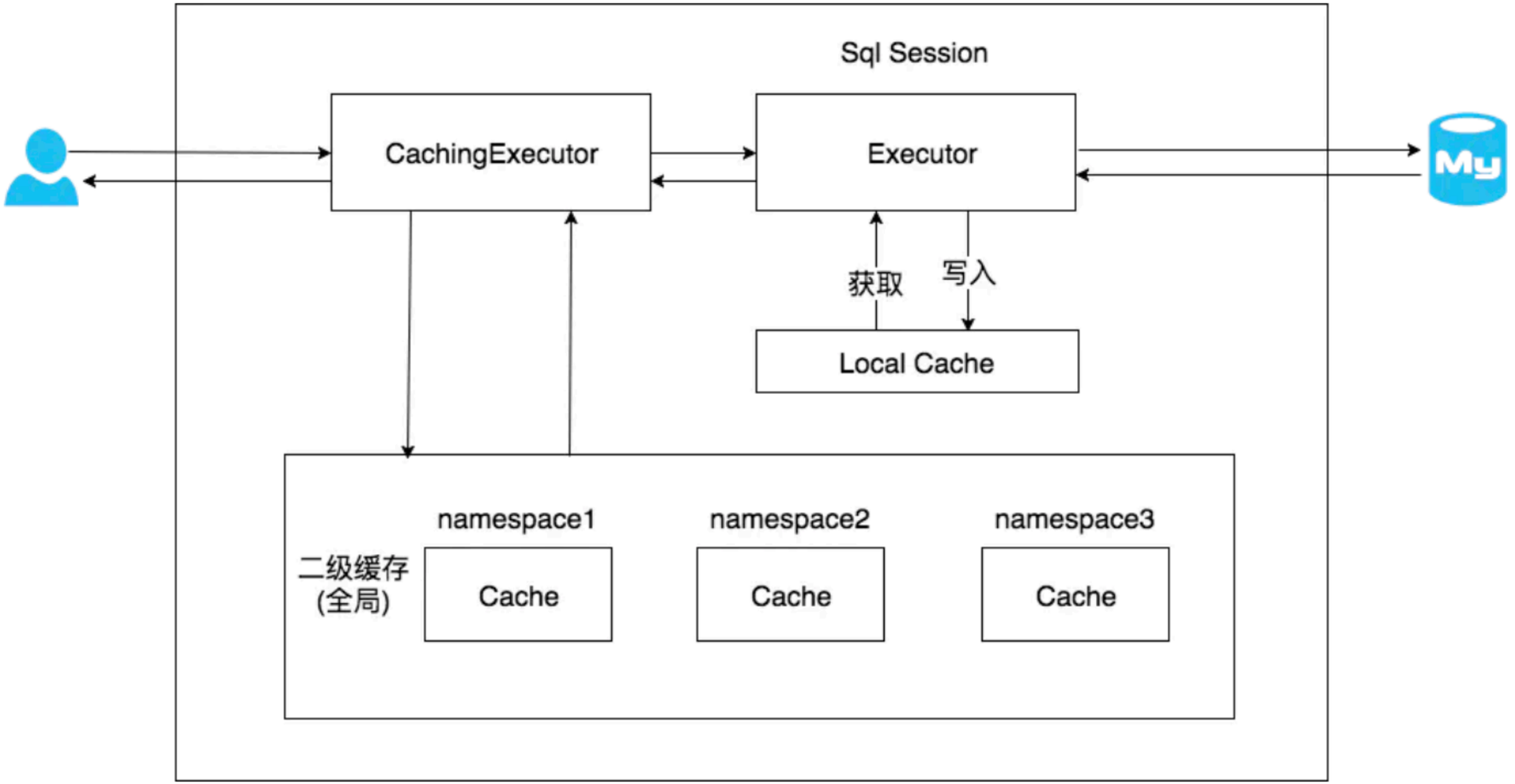
- 先来再次明确 MyBatis 的几个重要概念：
  - Executor：
    - 具体用来和数据库交互的执行器，接受 MappedStatement 作为参数。
  - 映射接口：
    - 在接口中会要执行的 SQL 用一个方法来表示，具体的 SQL 写在映射文件中。
  - 映射文件：
    - MyBatis 编写 SQL 的文件，通常来说每一张单表都会对应着一个映射文件。

与 Hiberate一样，MyBatis 也分为一级缓存和二级缓存

- 一级缓存介绍：
  - 我们在代码运行的时候，有可能会在一个数据库会话中执行多次相同的 SQL，这种反复的查询会带来一定的开销，如果查询的次数多对数据库性能也是有影响的。



- 二级缓存介绍


Sql Session
CachingExecutor
Executor
My
获取
写入
Local Cache
namespace1
namespace2
namespace3
二级缓存
(全局)
Cache
Cache
Cache

一、二级缓存的使用注意点：

- MyBatis 默认的 session 级别一级缓存，由于 Spring Boot 中默认使用了 hikariCP，所以基本没用，需要开启事务才有用。但一级缓存作用域仅限同一 sqlSession 内，无法感知到其他 sqlSession 的增删改，所以极易产生脏数据。

- 二级缓存可通过 cache-ref 让多个 mapper.xml 共享同一 namespace，从而实现缓存共享，但多表联查时配置略微繁琐。

- 生产环境建议将一级缓存设置为 statment 级别（即关闭一级缓存），如果有必要，可以开启二级缓存。

- 在分布式环境中也是不建议开启二级缓存的，因为缓存是保存到本地的，这样也会导致产生脏数据。

极客时间

扫码试看/订阅

《分布式缓存高手课》视频课程