穿透：大量数据访问，在redis内不存在，下沉到数据库；穿透了redis的保护，增加了底层数据的压力

解决：

在接口访问层对用户增加校验；

利用布隆过滤器，将数据库层有的数据key存储在位数组中，以判断访问的key在底层数据库中是否存在

布隆过滤器：

它实际上是一个很长的二进制向量和一系列随机映射函数。主要用于判断一个元素是否在一个集合中。

原理：对于输入进来的一个值，通过k个哈希函数，得到k个值，将二进制向量上的k个位置设置为1

查询的时候对输入的值通过k个哈希函数，如果这k个位置有一个为0，则不存在，如果都为1则有

可能存在

误判率：产生误判的根本原因，会出现多个元素占用一个位置

特性：判断不存在就一定不存在，判断存在就不一定存在；可以添加元素但是不能删除元素，删除会导致误判率增加

插入和查询的时间复杂度都为O(K)，K指有k个哈希函数

击穿：穿透表示底层数据库没有数据切缓存中也没有数据，击穿表示底层数据库有数据但缓存中没有，通常是热点key过期导致的，大量访问同时请求这个数据，就会将查询下沉到数据库层。

解决：

延长热点key的过期时间或者设置永不过期；利用互斥锁，确保同时只有一个客户端可以查询底层数据库，一旦查询到就缓存到redis内，使用互斥锁时需要避免出现死锁或者锁过期的情况。

雪崩：

是击穿的大面积版，击穿是热点key过期，雪崩是大量key同时过期，然后大量查询打到数据库上，这种现象为缓存雪崩

解决：

在可接受的时间范围内随机设置key的过期时间，分散key的过期时间，以防止大量的key在同一时刻过期；

对于一定要在固定时间让key失效的场景，可以在固定的失效时间时在接口服务端设置随机延时，将请求的时间打散，让

一部分查询先将数据缓存起来；

延长热点key的过期时间或者设置永不过期。

缓存预热：

缓存预热如字面意思，当系统上线时，缓存内还没有数据，如果直接提供给用户使用，每个请求都会穿过缓存去访问底层数据库，如果并发大的话，很有可能在上线当天就会宕机，因此我们需要在上线前先将数据库内的热点数据缓存至Redis内再提供出去使用，这种操作就成为"缓存预热"。

缓存预热的实现方式有很多，比较通用的方式是写个批任务，在启动项目时或定时去触发将底层数据库内的热点数据加载到缓存内。

缓存更新：

缓存服务（Redis）和数据服务（底层数据库）是相互独立且异构的系统，在更新缓存或更新数据的时候无法做到原子性的同时更新两边的数据，因此在并发读写或第二步操作异常时会遇到各种数据不一致的问题。如何解决并发场景下更新操作的双写一致是缓存系统的一个重要知识点。

第二步操作异常：缓存和数据的操作顺序中，第二个动作报错。如数据库被更新， 此时失效缓存的时候出错，缓存内数据仍是旧版本；

缓存更新的设计模式有四种：

Cache aside：查询：先查缓存，缓存没有就查数据库，然后加载至缓存内；更新：先更新数据库，然后让缓存失效；或者先失效缓存然后更新数据库；

Read through：在查询操作中更新缓存，即当缓存失效时，Cache Aside 模式是由调用方负责把数据加载入缓存，而 Read Through 则用缓存服务自己来加载；

Write through：在更新数据时发生。当有数据更新的时候，如果没有命中缓存，直接更新数据库，然后返回。如果命中了缓存，则更新缓存，然后由缓存自己更新数据库；

Write behind caching：俗称write back，在更新数据的时候，只更新缓存，不更新数据库，缓存会异步地定时批量更新数据库；

Cache aside：

为了避免在并发场景下，多个请求同时更新同一个缓存导致脏数据，因此不能直接更新缓存而是另缓存失效。

先更新数据库后失效缓存：并发场景下，推荐使用延迟失效（写请求完成后给缓存设置1s过期时间），在读请求缓存数据时若redis内已有该数据（其他写请求还未结束）则不更新。当redis内没有该数据的时候（其他写请求已令该缓存失效），读请求才会更新redis内的数据。这里的读请求缓存数据可以加上失效时间，以防第二步操作异常导致的不一致情况。

先失效缓存后更新数据库：并发场景下，推荐使用延迟失效（写请求开始前给缓存设置1s过期时间），在写请求失效缓存时设置一个1s延迟时间，然后再去更新数据库的数据，此时其他读请求仍然可以读到缓存内的数据，当数据库端更新完成后，缓存内的数据已失效，之后的读请求会将数据库端最新的数据加载至缓存内保证缓存和数据库端数据一致性；在这种方案下，第二步操作异常不会引起数据不一致，例如设置了缓存1s后失效，然后在更新数据库时报错，即使缓存失效，之后的读请求仍然会把更新前的数据重新加载到缓存内。

推荐使用先失效缓存，后更新数据库，配合延迟失效来更新缓存的模式；

缓存本身就是通过牺牲强一致性来提高性能，因此使用缓存提升性能，就会有数据更新的延迟性。这就需要我们在评估需求和设计阶段根据实际场景去做权衡了。

缓存降级：

缓存降级是指当访问量剧增、服务出现问题（如响应时间慢或不响应）或非核心服务影响到核心流程的性能时，即使是有损部分其他服务，仍然需要保证主服务可用。可以将其他次要服务的数据进行缓存降级，从而提升主服务的稳定性。

降级的目的是保证核心服务可用，即使是有损的。如去年双十一的时候淘宝购物车无法修改地址只能使用默认地址，这个服务就是被降级了，这里阿里保证了订单可以正常提交和付款，但修改地址的服务可以在服务器压力降低，并发量相对减少的时候再恢复。

降级可以根据实时的监控数据进行自动降级也可以配置开关人工降级。是否需要降级，哪些服务需要降级，在什么情况下再降级，取决于大家对于系统功能的取舍。