前言：

设计一个缓存系统，不得不考虑的问题就是：缓存穿透、缓存击穿与失效时的雪崩效应。

缓存穿透：

缓存穿透是指查询一个一定不存在的数据，由于缓存是不命中时被动写的，并且出于容错考虑，如果从存储层查不到数据则不写入缓存，这将导致这个不存在的数据每次请求都要到存储层去查询，失去了缓存的意义。在流量大时，可能DB就挂掉了，要是有人利用不存在的key频繁攻击我们的应用，这就是漏洞。

缓存穿透的解决方案：

有很多中方法可以有效地解决缓存穿透问题，最常见的则是采用布隆过滤器（https://blog.csdn.net/fouy_yun/article/details/81075432），将所有可能存在的数据哈希到一个足够大的bitmap中，一个一定不存在的数据会被这个bitmap拦截掉，从而避免了对底层存储系统的查询压力。另外还有个更加简单粗暴的方法，如果一个查询返回的数据为空（不管数据存不存在，还是系统故障），我们仍然把这个空结果进行缓存，但它的过期时间会很短，最长不超过五分钟。

缓存雪崩：

缓存雪崩是指在我们设置缓存时采用了相同的过期时间，导致缓存在某一时刻同时失效，请求全部转发到DB，DB瞬时压力过重雪崩。

缓存雪崩的解决方案：

缓存失效时的雪崩效应对底层系统的冲击非常可怕。大多数系统设计者考虑用加锁或者队列的方式保证缓存的单线程（进程）写，从而避免失效时大量的并发请求落在底层存储系统上。这里分享一个简单的方法就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，就很难引发集体失效的事件。

缓存击穿：

对于一些设置了过期时间的key，如果这些key可能会在某些时间点被超高并发地访问，是一种非常“热点”的数据。这个时候，需要考虑一个问题：缓存被“击穿”的问题，这个和缓存雪崩的区别在于这里针对某一key缓存，前者则是很多key。

缓存在某个时间点过期的时候，恰好在这个时间点对这个key有大量的并发请求过来，这些请求发现缓存过期一般都会从后端DB加载数据并回打缓存，这个时候大并发的请求可能会瞬

间把后端DB压垮。

缓存击穿的解决方案：

1.使用互斥锁：

业界比较常用的做法是使用mutex。简单的说，就是在缓存失效的时候（判断拿出来的值是不是为空），不是立即去load db，而是先使用缓存工具的某些带成功操作返回值的操作去set一个mutex key，当操作返回成功时，再进行load db的操作并回设缓存；否则，就重试整个get缓存的方法。

setnx，是set if not exists的缩写，也就是只有不存在的时候才设置，可以利用它来实现锁的效果。

```
public String get(key) {
      String value = redis.get(key);
      if (value == null) { //代表缓存值过期
          //设置3min的超时，防止del操作失败的时候，下次缓存过期一直不能load db
                  if (redis.setnx(key_mutex, 1, 3 * 60) == 1) {  //代表设置成功
               value = db.get(key);
                      redis.set(key, value, expire_secs);
                      redis.del(key_mutex);
              } else {  //这个时候代表同时候的其他线程已经load db并回设到缓存了，这时候
                      sleep(50);
                      get(key);  //重试
              }
          } else {
              return value;
          }
 }
```