1. 列举一个常用的Redis客户端的并发模型

```
lock = 0;
while (timeout > 0) {
  if (setnxexpire(key, value)) {
    lock = 1;
    return lock;
  }

  timeout -= sleeptime
  sleep(sleeptime);
}
```

2. 如何实现一个Hashtable?你的设计如何考虑Hash冲突？如何优化?

3. 分布式缓存，一致性hash

1、一致性hash算法：我们的memcached客户端（这里我看的spymemcache的源码），使用了一致性hash算法ketama进行数据存储节点的选择。与常规的hash算法思路不同，只是对我们要存储数据的key进行hash计算，分配到不同节点存储。一致性hash算法是对我们要存储数据的服务器进行hash计算，进而确认每个key的存储位置。这里提到的一致性hash算法ketama的做法是：选择具体的机器节点不在只依赖需要缓存数据的key的hash本身了，而是机器节点本身也进行了hash运算。

1、一致性hash算法是分布式系统中常用算法，设计目的是为了解决因特网中的热点(hot spot)问题。解决了P2P环境最为关键问题—如何在动态网络拓扑中分布存储和路由；

2、一致性hash算法引入虚拟节点机制，解决服务节点少时数据倾斜问题(即对每一个服务节点计算多个哈希，每个计算结果位置都放置一个此服务节点，称为虚拟节点。)；

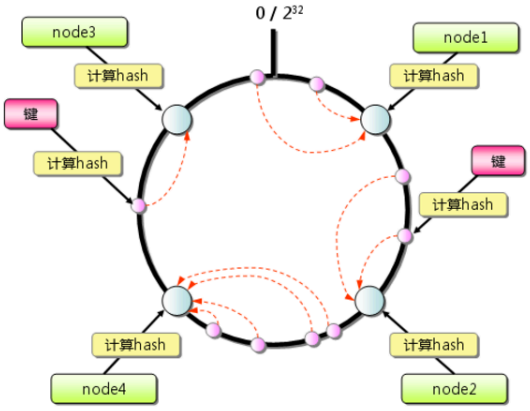
2、具体做法：如果有一个写入缓存的请求，其中Key值为K，计算器hash值Hash(K)， Hash(K) 对应于图 – 1环中的某一个点，如果该点对应没有映射到具体的某一个机器节点，那么顺时针查找，直到第一次找到有映射机器的节点，该节点就是确定的目标节点，如果超过了2^32仍然找不到节点，则命中第一个机器节点。比如 Hash(K) 的值介于A~B之间，那么命中的机器节点应该是B节点（如上图 ）。

3、数据保存流程：

1、首先求出memcached服务器（节点）的哈希值，并将其配置到0～232的圆（continuum）上。

2、然后采用同样的方法求出存储数据的键的哈希值，并映射到相同的圆上。

3、然后从数据映射到的位置开始顺时针查找，将数据保存到找到的第一个服务器上。如果超过232仍然找不到服务器，就会保存到第一台memcached服务器上。



4. LRU算法， slab分配，如何减少内存碎片

memcached预先将分配的内存分割成各种尺寸的块(chunk)，并把尺寸相同的块分成组(chunk的集合)，以此克服内存碎片化问题

5. 如何解决缓存单机热点问题

a. 原因：

1、缓存服务器自身有限流保持

缓存服务器数量 * 单机能够承受的qps > 用户最大的QPS 就会触发限流保护

针对这个原因：可以做横向扩容。加机器即可

2、用户访问过来cache服务器集中打到一台上面了。大流量并没有按预期的那样分摊到不同的cache机器上导致出现单机热点。(热点数据)

针对这个原因：只要计算cache-hash算法不出问题，那基本上可以做到缓存的随机分布均匀的

3、缓存里面的value过大，导致虽然QPS不高，但网络流量（qps * 单个value的大小）还是过大，触发了cache机器单台机器的网络流量限流；

针对这个原因：需要把大value进行精简，部分可以放在本机内存而不需要走远程获取这种方式的。

b. 解决方法：针对cache中元素key的访问监控。一旦发现cache有qps限流或网络大小限流时，能够通过监控看到到底是哪个key并发访问量过大导致，或者哪些key返回的value大小较大，再结合cache散列算法，通过一定的规则动态修改key值去平摊到各个cache机器上去。

6. memcache与redis的区别

1. Redis中，并不是所有的数据都一直存储在内存中的，这是和Memcached相比一个最大的区别。

2. Memcache仅仅支持简单的k/v类型的数据，Redis同时还提供String, list，set，hash等数据结构的存储。

3. Redis支持数据的备份，即master-slave模式的数据备份。

4. Redis支持数据的持久化，可以将内存中的数据保持在磁盘中（rdb定时快照和aof实时记录操作命令的日志备份），重启的时候可以再次加载进行使用。Redis在很多方面具备数据库的特征，或者说就是一个数据库系统，而Memcached只是简单的K/V缓存

5. Redis可以做一些聚合、排序操作。

6. memcache使用cas乐观锁做一致性：拿版本号，操作，对比版本号，如果一致就操作，不一致就放弃任何操作；

7. 大数据memcached性能更高。由于Redis只使用单核，而Memcached可以使用多核，所以平均每一个核上Redis在存储小数据时比Memcached性能更 高。而在100k以上的数据中，Memcached性能要高于Redis 。

7. redis 本身有持久化，为什么还要写进 mysql 呢?

- RDB：快照形式是直接把内存中的数据保存到一个 dump 文件中，定时保存，保存策略。
- AOF：把所有的对Redis的服务器进行修改的命令都存到一个文件里，命令的集合。

RDB会丢数据，AOP性能不行

- 有改动先插入数据库，再插缓存，比较靠谱但性能一般；
- 有改动先插缓存，批量更新到数据库，靠谱度略差，但性能好。

8. redis的数据结构和各种应用场景?

a. 更多的数据结构；

b. 可持久化；

c. 计数器；

d. 发布-订阅功能；

e. 事务功能；

f. 过期回调功能；

g. 队列功能；

h. 排序、聚合查询功能。

9. redis数据结构?

a. Redis有5个基本数据结构，string、list、hash、set和zset。

b. String：string表示的是一个可变的字节数组，我们初始化字符串的内容、可以拿到字符串的长度，可以获取string的子串，可以覆盖string的子串内容，可以追加子串。

c. List：Redis将列表数据结构命名为list而不是array，是因为列表的存储结构用的是链表而不是数组，而且链表还是双向链表。因为它是链表，所以随机定位性能较弱，首尾插入删除性能较优。如果list的列表长度很长，使用时我们一定要关注链表相关操作的时间复杂度。

d. hash：哈希等价于Java语言的HashMap或者是Python语言的dict，在实现结构上它使用二维结构，第一维是数组，第二维是链表，hash的内容key和value存放在链表中，数组里存放的是链表的头指针。通过key查找元素时，先计算key的hashcode，然后用hashcode对数组的长度进行取模定位到链表的表头，再对链表进行遍历获取到相应的value值，链表的作用就是用来将产生了「hash碰撞」的元素串起来。Java语言开发者会感到非常熟悉，因为这样的结构和HashMap是没有区别的。哈希的第一维数组的长度也是2^n。

e. set：Java程序员都知道HashSet的内部实现使用的是HashMap，只不过所有的value都指向同一个对象。

Redis的set结构也是一样，它的内部也使用hash结构，所有的value都指向同一个内部值。

f. SortedSet(zset)：是Redis提供的一个非常特别的数据结构，一方面它等价于Java的数据结构`Map<String, Double>`，可以给每一个元素value赋予一个权重`score`，另一方面它又类似于`TreeSet`，内部的元素会按照权重score进行排序，可以得到每个元素的名次，还可以通过score的范围来获取元素的列表。

10. codis和redis集群的区别?

	Codis	Redis-cluster	备注
Redis版本	2.8.13分支开发	>= 3.0	
部署	较复杂。	简单	
运维	Dashboard,运维方便。	运维人员手动通过命令操作。	
监控	可在Dashboard里监控当前redis-server节点情况，较为便捷。	不提供监控功能。	
组织架构	Proxy-Based 类中心化架构，集群管理层与存储层解耦。	P2P模型，gossip协议。 去中心化，	
伸缩性	动态伸缩。		
Server群组主从复制	不负责	负责。	
主节点失效处理	不自动选主，交由运维人员处理。	自动选主。	
数据迁移	以slot为单位，迁移期间保证可用性。	--	
升级	l 基于redis 2.8.13分支开发，后续升级不能保证； l Redis-server必须是此版本的codis，无法使用新版本redis的增强特性。	Redis官方推出，后续升级可保证。	升级需要重新加载数据，数据量较大的情况下， 分批次升级bin对运维要求比较高。
可靠性	经过线上服务验证，可靠性较高。	新推出，坑会比较多。遇到bug之后需要等官网升级。	

11.