1. redis（remote dictionary server远程字典服务）
2. 基本类型
   1. String字符串 key-value（一个key对应一个字符串）
      1. 共享session
      2. 分布式锁
      3. 计数器
      4. 限流
      5. 热点数据的缓存
   2. Hash哈希 key-value（一个hash对应多个key-value）
      1. 缓存用户信息
   3. List列表 key-value（一个list对应多个value）
      1. 消息队列
      2. 文章列表
   4. Set集合 key-value（一个set对应多个value）
      1. 用户标签、生成随机数抽奖、社交需求
   5. Z-set有序集合 key-value（一个Z-set对应多个value、score）
      1. 排行榜
      2. 社交需求（如用户点赞）
   6. Set和list的区别
      1. 前者不允许有重复的元素 list允许 不同点
      2. Set中的元素是无序的，不能通过索引下标获取元素 list可以通过下标获取 不同点
      3. 支持集合见得操作，可以多个集合取交、并、差集 list也可以 相同点
3. 特殊的数据结构类型
   1. Geospatial
      1. 地理位置定位
      2. 存储地理位置信息
   2. Hyperloglog
      1. 用来做基数统计算法的数据结构（如统计网站的UV）
   3. Bitmap
      1. 用一个比特位来映射元素的状态
4. Redis为什么这么快
   1. 基于内存实现
      1. Redis相对于关系型数据库，省去磁盘读写的消耗
   2. 高效的数据结构
   3. 合理的数据编码
   4. 合理的线程模型
   5. 虚拟内存机制
5. 什么是缓存击穿、缓存穿透、缓存雪崩
   1. 缓存穿透：读请求访问时，缓存和数据库都没有某个值，这样就会导致每次对这个值的查询请求都会穿透到数据库，这就是缓存穿透。（查询的时候，缓存中没有，就去数据库中查询，数据库中也没有，就不会写入进缓存，每次请求，都会到数据库查询，给数据库带来压力）

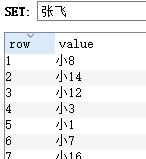
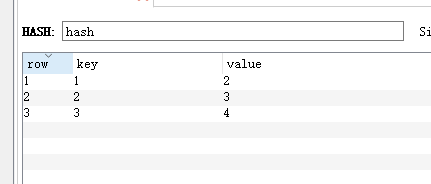
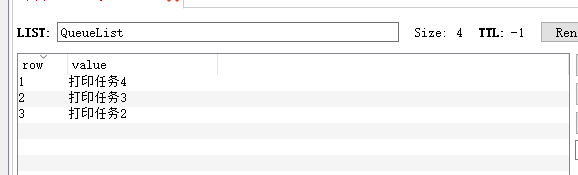
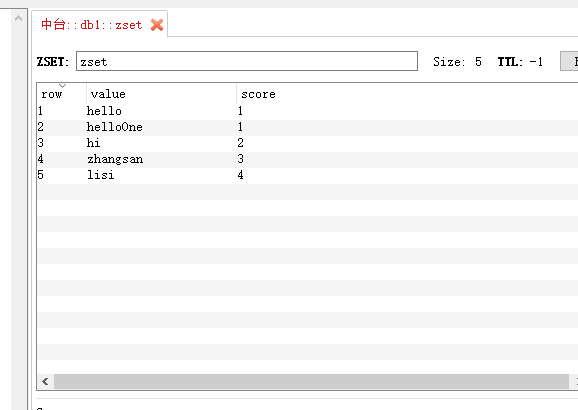
解决：1.如果是非法请求，在api的入口，对参数进行校验，过滤非法值。

1. 如果查询数据库为空，可以给缓存设置个空值，或者默认值。
2. 使用布隆过滤器快速判断数据是否存在。即一个查询请求过来时，先通过布隆过滤器判断值是够存在，存在才继续往下查。（布隆过滤器的原理？？）
   1. 缓存雪崩：缓存中数据大批量到过期时间，而查询数据量巨大，请求都直接访问数据库，引起数据库压力过大甚至down机。

解决：1.可通过均匀设置过期时间解决，即让过期时间相对离散一点，如采用一个较大的固定值+一个较小的随机值，5小时+0到1800秒这样。

1. redis故障宕机也可能引起缓存雪崩，这就需要构造redis高可用集群啦
   1. 缓存击穿：指热点key在某个时间点过期的时候，而恰好在这个时间点对这个key有大量的并发请求，从而大量的请求打到db

解决：1.缓存失效时，不是立即去加载db数据，而是先使用某些待成功返回的原子操作命令，如（redis的setnx）去操作，成功的时候，再去加载db数据库数据和设置缓存。否则就去重试获取缓存。（互斥锁方案）

1. 永不过期：没有设置过期时间，但是热点数据快要过期时，异步线程去更新和设置过期时间。
   1. 缓存击穿和缓存雪崩的区别：
      1. 前者是某个key 后者是一群key
      2. 前者是增加数据库的压力 后者是数据库压力过大甚至down机
2. 什么是热key问题，如何解决热key问题
   1. 如果某一热点key的请求到服务器主机时，由于请求量特别大，可能会导致主机资源不足，甚至宕机，从而影响正常的服务。
   2. 热点key产生的原因主要有两个
      1. 用户消费的数据远大于生产的数据，如秒杀、热点新闻等读多写少的场景
      2. 请求分片集中，超过redis服务器的性能，比如固定名称key，hash落入同一台服务器，瞬间访问量极大，超过机器瓶颈，产生热点key问题。
   3. 如何识别热点key
      1. 凭经验判断哪些是热点key
      2. 客户端统计上报
      3. 服务代理层上报
   4. 解决
      1. Redis集群扩容：增加分片副本，均衡读流量
      2. 将热点key分散到不同的服务器中
      3. 使用二级缓存，即JVM本地缓存，减少redis的读请求
3. Redis过期策略和内存淘汰策略
   1. 过期策略
      1. 定时过期： 每个设置过期时间的key都需要创建一个定时器，到过期时间就会立即对key进行清除。该策略可以立即清除过期的数据，对内存很友好，但是会占用大量的cpu资源去处理过期的数据，从而影响缓存的响应时间和吞吐量。
      2. 惰性过期：
      3. 定期过期：
4. Redis的持久化
   1. Rdb快照：把内存中当前进程的数据生成快照（.Rdb）文件保存到硬盘的过程，有手动触发和自动触发
      1. Save：阻塞当前redis，直到rdb持久化过程完成为止，若内存实例比较大，会造成长时间阻塞，线上环境不建议用它
      2. Bgsave：redis进程执行fork操作创建子进程，由子进程完成持久化，阻塞时间很短（微妙级）
      3. 以上为手动触发，自动触发指的是在配置文件.conf中的配置
   2. Aof日志
      1. Bgrewriteaof 将写命令记录在aof文件中，在重启redis服务的时候，将命令在执行一遍
   3. 混合模式
      1. 通过配置文件中的aof-use-rdb-preamble yes配置开启后，在通过bgrewriteaof执行（生成此刻之前的数据为rdb文件记录在aof文件的开头，在备份的过程中，新产生的命令记录在rdb文件的末尾）加载的时候，先加载rdb文件，在加载剩余的aof命令。
         1. 将rdb文件的内容和增量aof日志文件存在一起。
   4. 三种模式的区别以及优缺点
      1. Rdb模式 （数据完整性低，加载快）
         1. 优点：速度快 使用子进程来进行持久化，主进程不会进行任何的io操作，保证了redis的性能。
         2. 缺点：它是间隔一段时间后进行持久化，如果在这个事件段内，redis发生了故障，就会发生数据的丢失。 而且每次都要创建子进程，频繁创建成本过高，备份时占用内存。（间隔一段时间：手动中，不能一直在备份，自动中也是满足条件（一定的时间之后[多长时间内更新了多少条记录之后达到条件生成快照]）之后在备份，所以是间隔一段时间）
      2. Aof模式（数据完整性高，加载慢）
         1. 优点：因为是记录的写命令，数据能够保证是最新的，安全性较高
         2. 缺点：同时也因为是记录的命令，所以文件体积要比rdb大的多，在重启恢复的时候，时间往往会慢很多。加载慢。
      3. 混合模式
         1. 结合了rdb和aof的优点（更快的启动，减低了大量数据丢失的风险）
         2. Aof文件中添加了rdb格式的内容，使得aof文件的可读性变得很差
         3. 兼容性差，开启混合持久化，那么此混合持久化aof文件，就不能用在redis4.0之前的版本了
5. 位图（数据类型）
6. [hyperloglogs](https://www.redis.net.cn/tutorial/3513.html)
7. 内置复制
8. Lua脚本
9. Lua收回
10. Redis的16个常用使用场景
    1. 缓存：用户信息的缓存（key-value）
    2. 数据共享分布式 string类型（因为分布式的独立服务，可以再多个应用之间共享--分布式session）
    3. 分布式锁 string的set命令增加了一些参数：
       1. EX：设置键的过期时间（单位为秒）
       2. PX：设置键的过期时间（单位为毫秒）
       3. NX：只在键不存在时，才对键进行设置操作。
       4. XX：只在键已经存在时，才对键进行设置操作。
    4. 全局ID：int类型，incrby，利用原子性 incrby userid 1000分库分表的场景，一次性拿一段
    5. 计数器
    6. 限流
    7. 位统计
    8. 时间轴
    9. 消息队列
    10. 抽奖
    11. 点赞、签到、打卡
    12. 商品标签
    13. 好友关系、用户关注、推荐模型
    14. 排行榜
    15. 倒排索引
    16. 显示最新的项目列表
11. 位图：其实就是字符串，只不过该字符串的每一位都是字符0或者字符1
12. 发布订阅
    1. Unsubscribe Channels 退订给定的频道
    2. Punsubscribe Channels 退订所有给定模式的频道
    3. Subscribe Channels 订阅给定的一个或多个频道的信息
    4. Psubscribe Channels 订阅一个或多个符合给定模式的频道
    5. Publish Channels “hello world” 将信息发送到指定的频道
    6. Pubsub Channels 查看订阅与发布系统状态
13. 命令：
    1. Key
    2. String
    3. Hash
    4. List
    5. Set
    6. Z-set
    7. 连接
    8. 服务器
    9. 脚本
    10. 事务
    11. HyperLogLog
    12. 发布订阅
    13. 地理位置(geo)
14. 主从复制：例：一台redis服务器的数据，复制到其他的redis服务器。前者称为主节点，后者称为从节点，数据的复制是单向的，只能由主节点到从节点。主节点以写为主，从节点以读为主。
    1. 读写分离：就比如主从复制的模式，主节点负责写，从节点负责读。
    2. 数据冗余：主从复制实现了数据的热备份，是持久化之外的一种数据冗余方式
    3. 故障恢复：当主节点出现问题时，可以由从节点提供服务，实现快速的故障恢复，实际上是一种服务的冗余
    4. 负载均衡：在主从复制的基础上，配合读写分离，可以由主节点提供写服务，由从节点提供读服务（即写redis数据时，应用连接主节点，读redis数据时，应用连接从节点），分担服务器负载，尤其是在写少读多的场景下，通过多个从节点分担读负载，可以大大提高redis服务的并发量。
    5. 高可用（集群）基石：除了上述作用外，主从复制还是哨兵和集群能够实施的基础，因此说主从复制是redis高可用的基础。
15. 哨兵模式
    1. 监控的作用，如果监测到主服务器故障了，就会根据投票数自动将从库转换为主库。
16. 在缓存穿透中设计到的布隆过滤器：
17. redis key-value非关系型数据库 (remote dictionary server)
    1. 支持多种数据结构 string hash list set z-set
    2. 基于内存 速度快 缓解数据库的压力
    3. 应用场景
       1. 会话缓存
       2. 消息队列
       3. 排行榜
       4. 存放热点数据
18. 缓存穿透：恶意代码或缺陷代码大量访问redis与数据库没有的key，导致请求到底层都搜不到数据
    1. 如果在redis中没有找到key，就返回null
19. 缓存击穿：大量的key在同一时间失效，那么就会请求到数据库中，造成数据库崩溃
    1. 合理设置key的过期时间，尽量避免这种情况发生
20. 缓存雪崩：redis崩溃或者重启的时候，导致大量的请求直接到数据，导致数据库崩溃
    1. 采用主从赋值和集群，保证高可用，有备用的redis实例能够提供服务
21. set
    1. Sunion 返回所有给定集合的并集
    2. Sdiff 返回给定集合的差集
    3. Sinter 返回所有给定集合的交集
    4. Sunionstore 返回给定集合的并集，并将并集存储在destination集合中
    5. Sdiffstore 返回给定集合的差集，并将差集存储在destination集合中
    6. Sinterstore 返回给定集合的交集，并将交集存储在destination集合中
    7. Srandmember 随机返回集合中的一个或多个member
    8. Smembers 返回集合中的所有成员
    9. Sismember 判断member是否存在指定集合中
    10. Spop 移除并返回集合中的一个随机member
    11. Srem 移除集合中一个或多个member
    12. Smove 将member从source集合移动到destination集合中
    13. Scard 返回集合的member数量
    14. Sadd 向集合中添加一个或多个成员
    15. Sscan ？？？ 迭代集合中的元素（包括元素成员和元素分值）
        1. 返回的如果是0，证明已经循环完了/如果不是，则证明还没有循环完
        2. Cursor 和二进制相关 （循环的时候，按照游标循环）
22. hash
    1. Hset 设置hash表中的value值
    2. Hmset 同时将多个key-value对设置在hash中
    3. Hsetnx 只有在字段key不存在时，设置hash表字段的值
    4. Hget 获取存储在hash表中指定key的值
    5. Hmget 获取所有给定key的value
    6. Hgetall 获取hash表中的所有key-value
    7. Hkeys 获取所有hash表中的key
    8. Hvals 获取hash表中所有的值
    9. Hincrbyfloat 为hash表中你指定的字段的浮点数加上增量
    10. Hincrby 为hash表中指定的字段的整数值加上增量
    11. Hexists 查看指定的字段是否存在
    12. Hlen 获取hash表中的数量
    13. Hdel 删除一个或多个hash表的字段
23. list
    1. Linsert 在列表的前或者后插入元素
    2. Lset 通过索引设置列表元素的值
    3. Lindex 通过index获取列表中的元素
    4. Lrange 获取列表指定index范围内的元素
    5. Lrem 移除列表的元素
    6. Llen 查询列表的member的总数量
    7. Ltrim 对一个列表进行trim（修剪），只保留指定index区间内的元素，不在区间内的删除
    8. Lpop 移除并获取列表的第一个元素
    9. Lpush 将一个或多个值插入到列表头部
    10. Lpushx 将一个或多个值插入到【已存在】的列表头部
    11. Rpush 在列表中添加一个或多个值
    12. Rpushx 为【已存在】的列表添加值
    13. Rpop 移除并获取列表的最后一个元素
    14. RpopLpush 移除列表的最后一个元素，并将该元素添加到另一个列表并返回
    15. BrpopLpush 从列表中弹出一个值，并将该值添加到另一个列表并返回，设置超时时间
    16. BLpop 移除并获取列表的第一个元素，指定超时时间
    17. BRpop 移除并获取列表的最后一个元素，指定超时时间
24. string
    1. Setex 设置key-value 并设置过期时间（以秒为单位）
    2. Psetex 设置key-value 并设置过期时间（以毫秒为单位）
    3. Getbit 对key所存储的字符串值，获取指定偏移量offset（index）上的位（bit）
    4. Setbit 对key所存储的字符串值，设置指定偏移量offset（index）上的位（bit）
    5. Set 设置指定key的值
    6. Setnx 只有在ke【不存在】的时候 设置key的值
    7. Mset 同时设置一个或多个key-value对
    8. Msetnx 同时设置一个或多个key-value对，当且仅当所有的key都不存在
    9. Append 将value追加在key原来的值的末尾
    10. Get 获取指定key的值
    11. Mget 获取一个或多个给定key的值
    12. Getset 将给定key的值设为value，并返回key的旧值
    13. Decr 将key中存储的数字减一
    14. Decrby 将key中存储的数字减去指定的值
    15. Incr 将key中存储的数字值增一
    16. Incrby 将key所存储的值加上指定的整数值
    17. Incrbyfloar 将key所存储的值加上指定的浮点数值
    18. Getrange 返回key中字符串的子字符
    19. Strlen 返回key所存储的字符串值的长度
    20. Setrange 用value参数复写给定key所存储的字符串值，从偏移量offset（index）开始
25. z-set
    1. Zrank 返回有序集合中指定成员的index位置
    2. Zrevrank 返回有序集合中指定成员的排名（score从大到小）
    3. Zremrangebyrank 移除有序集合中给定的 index 区间的所有成员
    4. Zcount 计算有序集合中指定score区间的成员数
    5. Zlexcount 在有序集合中计算指定value区间内的成员数量
    6. Zunionstore 计算给定的一个或多个有序集合的【并集】，并存储在新的有序集合中
    7. Zinterstore 计算一个或者多个给定有序集合的【交集】并将结果集存储在新的有序集合中
    8. Zrange 通过index返回有序集合中指定区间内的成员
    9. Zrevrange 返回有序集合中指定区间内的成员，通过index，分数降序
    10. Zrevrangebyscore 返回有序集中指定score区间内的成员，分数降序
    11. Zrangebyscore 通过score返回有序集合指定区间内的成员，分数升序
    12. Zrangebylex 通过value区间返回有序集合指定区间内的成员，
    13. Zremrangebyscore 移除有序集合中给定的score区间内的所有成员
    14. Zremrangebylex 移除有序集合中给定的value区间内的所有成员
    15. Zcard 获取有序集合的成员数
    16. Zrem 移除有序集合的一个或多个成员
    17. Zincrby 有序集合中给指定的成员的score加上增量increment
    18. Zscore 返回有序集合中 成员的score
    19. Zadd 向有序集合中添加一个或多个成员，或者更新已存在成员的score
    20. Zsan？？？ 迭代有序集合中的元素（包括元素成员和元素分值）
        1. 返回的如果是0，证明已经循环完了/如果不是，则证明还没有循环完
        2. Cursor 和二进制相关 （循环的时候，按照游标循环）
    21. ZADD
        1. 添加一个或多个成员
        2. 如果这个成员已经是有序集的成员，那么更新这个成员的分数值
        3. 分数值可以是整数或者双精度浮点数
        4. 如果有序集合不存在，则创建一个空的有序集合，执行zadd操作
        5. 当key存在但不是有序类型时，返回一个错误