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