1、何为Redis：Redis是一个开源的使用ANSI C语言编写（离操作系统更近所以性能好）、支持网络、可基于内存亦可持久化的日志型、Key-Value数据库，并提供多种语言的API，默认端口6379：

- Redis安装在磁盘；
- Redis数据存储在内存。（读取快的根本原因）

2、Redis版本：版本号第二位为奇数，为非稳定版本（2.7、2.9、3.1）；第二为偶数，为稳定版本（2.6、2.8、3.0）；当前奇数版本是下一个稳定版本的开发版本，如2.9是3.0的开发版本。

2、Redis为什么这么快?

- 绝大部分请求是纯粹的内存操作（非常快速）；
- 采用单线程,避免了不必要的上下文切换和竞争条件。单线程指的是网络请求模块使用了一个线程（所以不需考虑并发安全性），即一个线程处理所有网络请求，其他模块仍用了多个线程。
- 非阻塞IO - IO多路复用（采用epoll，epoll是线程安全的）。

2、Redis采用了定期删除和惰性删除两种淘汰策略
（https://blog.csdn.net/u011032846/article/details/80402352）

2、如果在配置文件中设置了密码，则通过auth命令解密后才能使用其他redis命令。

2、Redis配置文件中bind参数详解（生产环境为什么设置成redis所在服务器ip，而不是127.0.0.1）：https://blog.csdn.net/hel12he/article/details/46911159

2、通过config get xxx查询Redis参数配置，config set xxx设置环境参数。在客户端通过config set设置环境参数后重启redis服务设置就会失效，所以我们应该修改redis.config来设置参数。

2、使用场景（https://www.jianshu.com/p/40dbc78711c8）

- 缓存数据库：将mysql等关系型数据库中的数据放到redis中，应用可以从redis中获取数据
- 排行榜：借助redis的SortedSet进行热点数据的排序。
- 计数器：redis由于incrby命令可以实现原子性的递增，所以可以运用于高并发的秒杀活动、分布式序列号的生成、具体业务还体现在比如限制一个手机号发多少条短信、一个接口一分钟限制多少请求、一个接口一天限制调用多少次等等。
- 社交关系

2、Redis默认有16个数据库【0-15】，数据库个数可以在配置文件中修改，通过select n切换到对应的库。在一个项目中可以将不同的业务数据存到不同的库中。在微服务架构中，可以将不同的服务的数据存放到不同的库中。

3、停止Redis服务使用命令【redis -cli -h {host} -p {port} shutdown】，避免使用Kill -9。因为使用shutdown命令断开连接会生成持久化文件，相对安全；使用kill关闭，此方式不会做持久化，还会造成缓冲区非法关闭，可能会造成AOF和丢失数据；

4、Redis单线程执行命令，所有命令进行队列，按顺序执行，但有个问题使假如其中某个命令执行比较慢，则会造成其它命令的阻塞。

5、Redis针对value来说有五种数据结构：String（字符串）、Hash（哈希）、List（链表）、Set（集合）、Zset（有序集合）。针对key来说，所有的key（键）都是字符串。

6、虽然Redis对客户端的请求处理很快（就算堆积了几万条指令处理起来也是一瞬间的事），但是当我们的java客户端将多个指令分开发送的时候网络通信耗费比较严重，会造成redis性能下降。以插入多个数据为例我们可以批量操作，即【set name "ljq" 、 set age 10】改为【mset name "ljq" age "10"】，mset为批量插入指令。

7、Hash：Hash是一个Mapmap结构，指值本身又是一种键值对结构，如 value={field1 value1 ... fieldN valueN}：

- 使用场景：，假如数据库中有一张T_user表，id为主键，现在我们需要把T_user表中的记录存储到redis中，则我们可以通过hash存储（以主键id为key）：

数据库有张用户表结构如下：

id为主键

| id | name | age | city |
| --- | --- | --- | --- |
| 1 | jack | 23 | NULL |
| 2 | james | 18 | shanghai |

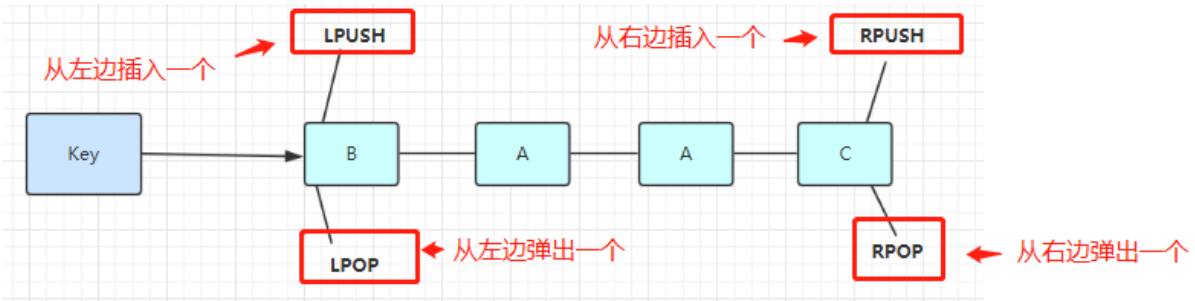
```
OK
127.0.0.1:6379> hmset 1 name "jack" age 23
OK
127.0.0.1:6379> hmset 2 name "james" age 18 city "shanghai"
OK
127.0.0.1:6379> hkeys 1
1) "name"
2) "age"
127.0.0.1:6379> hvals 2
1) "james"
2) "18"
3) "shanghai"
127.0.0.1:6379>
```

通过hash将id为1和2的表记录存到redis

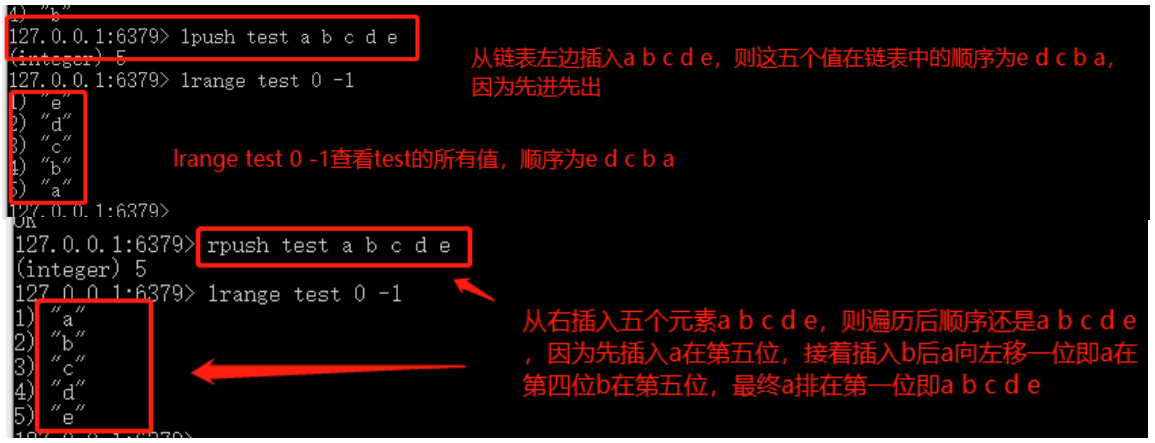
通过hkeys获取id为1的所有field值，通过hvals获取id为2的所有value值

- **Hash特别适合存储对象，相比原生和序列方式存储的优缺点如下图：**


**8、List 链表：（redis 使用双端链表实现的 List【对应java中的LinkedList】），是有序的，value可以重复，可以通过下标取出对应的value值，同时左右两边都能进行插入和删除数据，还可以从两端弹出数据：**

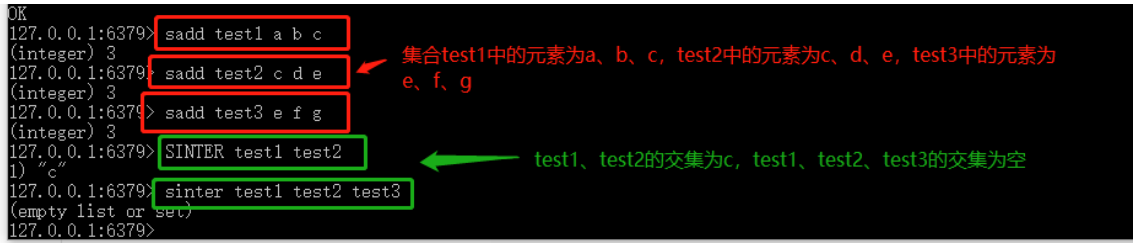


- **插入多个数据和范围查找（需要注意插入之后value在链表中的顺序，从左插入和从右插入后value在链表中的顺序是不一样的）：**

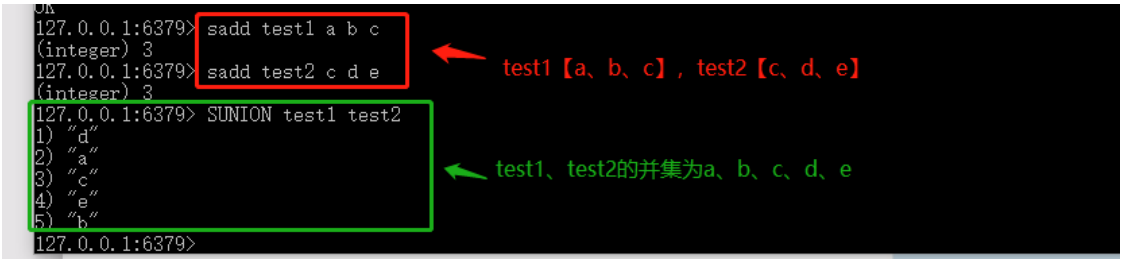



**9、Set集合：集合类型也是用来保存多个字符串的元素，但和列表不同的是集合中 1. 不允许有重复的元素，2.集合中的元素是无序的，不能通过索引下标获取元素，3.支持集合间的操作，可以取多个集合取交集、并集、差集：**

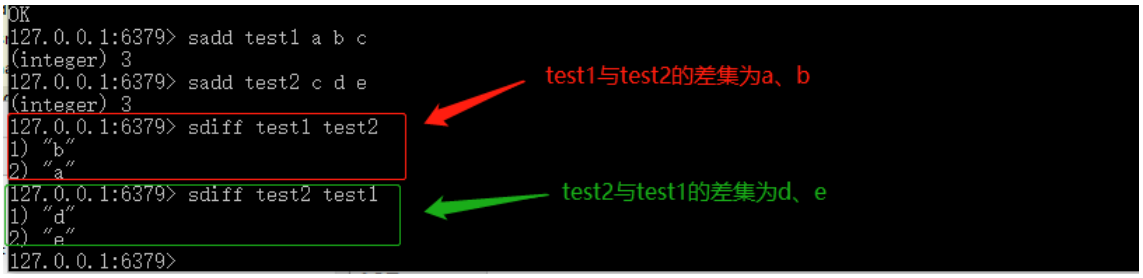
- **交集，返回多个key中共存的元素：**




- **并集，返回多个key中所有元素，重复的只显示一个：**



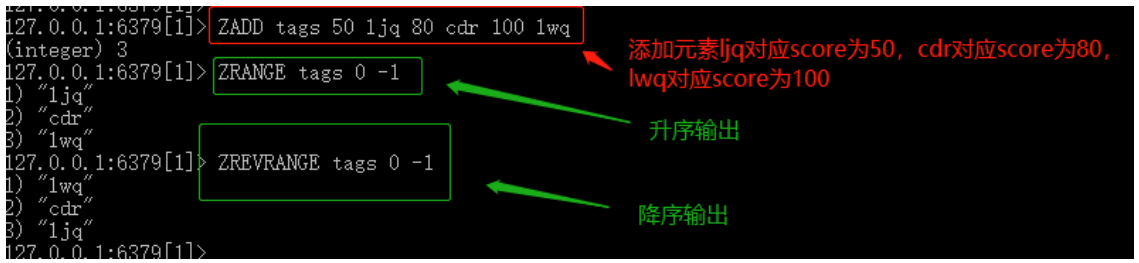

- **差集，和数学中的差集有点区别，数学中A与B、B与A的差集是一样的，但是在redis中A与B的差集只返回A中存在且B中不存在的元素，B与A的差集只返回B中存在且A中不存在的元素：**




- 应用场景：标签，社交，查询有共同兴趣爱好的人，共同好友、智能推荐

10、SortedSet有序集合：有序集合和集合有着必然的联系，保留了集合不能有重复成员的特性，区别是，有序集合中的元素是可以排序的，它给每个元素设置一个分数【score，score可以重复，且score 值可以是整数值或双精度浮点数】，作为排序的依据：

- 使用，ZADD添加数据，ZRANGE升序返回结果，ZREVRANGE降序返回结果：




- 使用场景：排行榜是有序集合经典使用场景。例如小说视频等网站需要对用户上传的小说视频做排行榜，榜单可以按照用户关注数，视频点赞数，时间的前后【因为score要求整数值或双精度浮点数，所以可以通过时间戳实现】，字数等打分，做排行。

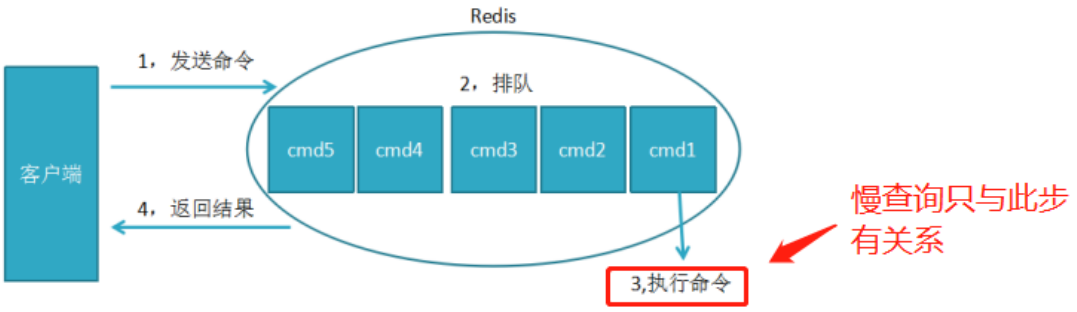
11、key的设计越短越好，节约内存。通常用冒号分割，如【user1:order】

12、JedisPool是jedis连接池，可以单例出来或者放到spring容器中。

13、LUA脚本：将多个操作写在一个.lua文件中，LUA会保证所有操作的原子性，即要么全部成功，要么全部失败。

14、Redis生命周期：发送→排队→执行→返回

14、Redis慢查询：Redis慢查询只统计生命周期中第三步执行命令所花费的时间，发送、排队、返回结果所用时间不计入慢查询统计范围：




14、慢查询阈值默认为10000微秒(microsecond)即10毫秒(millisecond)，有两种设置方式：

- 动态设置6379:> config set slowlog-log-slower-than 10000 //10毫秒 10000微秒

使用config set完后,若想将配置持久化保存到redis.conf，要执行config rewrite；

- redis.conf修改：找到slowlog-log-slower-than 10000 ，修改保存即可；
- 注意：slowlog-log-slower-than =0记录所有命令 -1命令都不记录。

14、慢查询原理：慢查询记录也是存在队列里的，slow-max-len 存放的记录最大条数，比如设置的slow-max-len = 10，当有第11条慢查询命令插入时，队列的第一条命令就会出列，第11条入列到慢查询队列中。可以config set动态设置，也可以修改redis.conf完成配置。

14、慢查询是先进先出的队列，访问日志记录出列丢失，需定期执行slow get,将结果存储到其它设备中（如mysql）。

15、./redis-server --test-memory 1024 //检测操作系统能否提供1024M内存给redis,常用于测试，想快速占满机器内存做极端条件的测试，可使用这个指令。

16、Jedis底层使用的Tcp协议，resp只是一种报文格式。

17、redis-benchmark性能测试工具。

**18、手写jedis其实是通过Socket与Redis服务端进行通讯，将报文以resp协议格式进行发送就ok，以调用jedis的set()方法为例，全过程如下，详情可看源码：**

```
public void set(final byte[] key, final byte[] value) {
  sendCommand(Command.SET, key, value);
}

protected Connection sendCommand(final ProtocolCommand cmd, final byte[]... args) {
  try {
    connect();
    Protocol.sendCommand(outputStream, cmd, args);
    pipelinedCommands++;
    return this;
  } catch (JedisConnectionException ex) {
    // Any other exceptions related to connection?
    broken = true;
    throw ex;
  }
}

public void connect() {
  if (!isConnected()) {
    try {
      socket = new Socket();
      // ->@wjw_add
      socket.setReuseAddress(true);
      socket.setKeepAlive(true); // Will monitor the TCP connection is
      // valid
      socket.setTcpNoDelay(true); // Socket buffer Whetherclosed, to
      // ensure timely delivery of data
      socket.setSoLinger( on: true,  linger: 0); // Control calls close () method,
      // the underlying socket is closed
      // immediately
      // <-@wjw_add

      socket.connect(new InetSocketAddress(host, port), connectionTimeout);
      socket.setSoTimeout(soTimeout);
      outputStream = new RedisOutputStream(socket.getOutputStream());
      inputStream = new RedisInputStream(socket.getInputStream());
    } catch (IOException ex) {
      broken = true;
      throw new JedisConnectionException(ex);
    }
  }
}

private static void sendCommand(final RedisOutputStream os, final byte[] command,
    final byte[]... args) {
  try {
    os.write(ASTERISK_BYTE);
    os.writeIntCrLf( value: args.length + 1);
    os.write(DOLLAR_BYTE);
    os.writeIntCrLf(command.length);
    os.write(command);
    os.writeCrLf();

    for (final byte[] arg : args) {
      os.write(DOLLAR_BYTE);
      os.writeIntCrLf(arg.length);
      os.write(arg);
      os.writeCrLf();
    }
  } catch (IOException e) {
    throw new JedisConnectionException(e);
  }
}
```

与服务端进行连接并且将命令按resp报文格式进行包装

连接服务端并获取Socket的输入输出流

根据resp格式将命令进行包装

19、pipeline出现的背景：redis客户端执行一条命令分4个过程【发送命令→命令排队→命令执行→返回结果】，这个过程称为Round trip time(简称RTT, 往返时间)，mget mset有效节约了RTT，但大部分命令（如hgetall，并没有mhgetall）不支持批量操作，需要消耗N次RTT 。例如当我们需要删除多个key的时候不使用pipeline时我们需要多次调用jedis的del命令，这样造成的网络通讯耗时比redis实际执行命令的耗时大很多，尤其是应用和redis在异地机房时会更恐怖，这个时候就需要pipeline来解决这个问题，pipeline会将多个del命令按resp报文格式组装，redis解析报文后分别去执行多条命令，这样就只需进行一次网络通讯（即将多次网络通讯减少到一次，而命令还是N条分别执行的），会节省不必要的RTT。

19、注意：Pipeline只是削减了不必要的RTT，它并不会减少命令本身的执行时间。

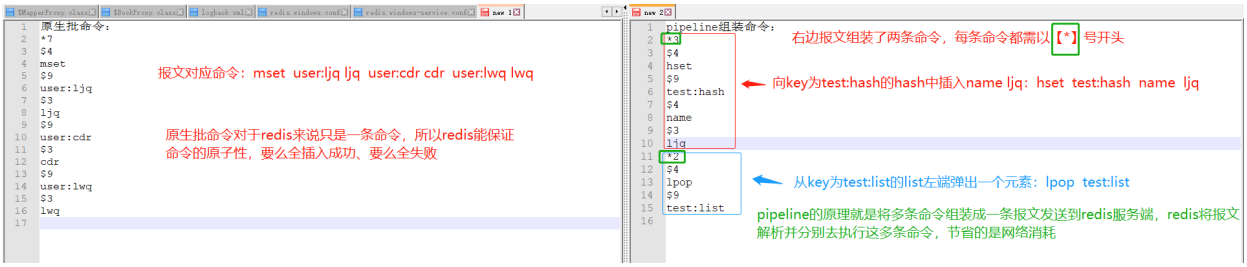
19、pipeline的目的就是为了减少网络开销，pipeline的原理就是将多个命令组装成一个resp报文发送给redis去执行，它不像lua脚本可以保证原子性，不过我们可以在组装的命令中加上事务命令去解决这个问题。

20、resp【Redis Serialization Protocol】协议（https://www.cnblogs.com/tommy-huang/p/6051577.html），【\r\n】代表换行符，即回车。

21、当我们通过一个报文组装多条命令时每条命令开头必须是【*】号，这样redis才会识别出是多条命令，否则redis会按一条命令来处理。

22、原生批命令(mset, mget)与Pipeline对比：

- 原生批命令是原子性，pipeline是非原子性。(原子性概念:一个事务是一个不可分割的最小 工作单位,要么都成功要么都失败。原子操作是指你的一个业务逻辑必须是不可拆分的. 处理一件 事情要么都成功，要么都失败，原子不可拆分)；
- 原生批命令一命令有多个key, 但pipeline是组装的多条命令（存在事务），非原子性；
- 原生批命令是redis服务端实现的，而pipeline需要服务端与客户端共同完成。




**23、使用pipeline组装的命令个数不能太多，不然数据量过大，增加客户端的等待时间，还可能造成网络阻塞，可以将大量命令的拆分多个小的pipeline命令完成。假如我们在一个pipeline中组装了十万条命令，因为redis是单线程的，当redis收到这个大报文以后解析会很慢，解析完后队列中瞬间增加了10万条命令，执行时间则会比较久，客户端等待时间就会增加。所谓物极必反，因此我们应将pipeline合理拆分。**

**24、redis事务：pipeline是多条命令的组合，为了保证它的原子性，redis提供了简单的事务。redis的简单事务，将一组需要一起执行的命令放到multi和exec两个命令之间，其中multi代表事务开始，exec代表事务结束 。watch命令：使用watch后， multi失效，事务失效。**

**24、redis事务比较鸡肋，它是若事务性的，不支持回滚。要慎用或者不用。**

**25、不会lua语言则redis白学。如果多个命令的执行需要保证强事务性那么必须使用lua脚本。**

**26、使用lua脚本的好处：**

- **减少网络开销：像pipeline一样将多个命令打包统一执行，因次如果脚本中的命令过多也会有阻塞问题；**
- **原子操作：保证脚本中所有命令的强事务性；**
- **复用性：客户端发送的脚本会永远存储在Redis中，这意味着其他客户端可以复用这一脚本来完成同样的逻辑。**

**27、jedis提供了scriptLoad(String script)方法用来加载lua脚本，它的返回值为sha。同时提供了evalsha()用来执行lua脚本。**

**28、redis的消息发布定阅很low，和专业的消息中间件（RocketMq、kafka等）相比它没有消息规程和回溯、事务消息、延时消息等功能。它的优点就是简单易用，如果业务场景**

比较简单可以考虑用它。

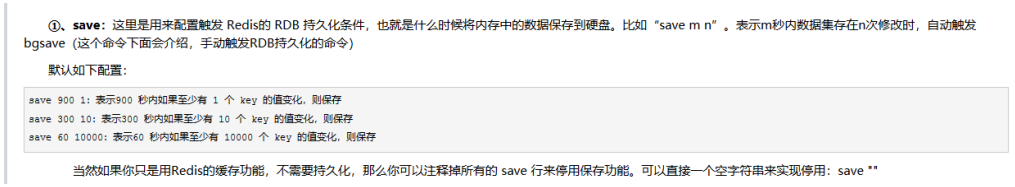
28、使用jedis的发布订阅功能，继承JedisPubSub抽象类并编写自己的业务逻辑实现（https://www.cnblogs.com/pypua/p/11044177.html）。

29、redis持久化时对linux fork()函数的妙用。一个进程调用fork()函数后，系统先给新的进程分配资源，例如存储数据和代码的空间。然后把原来的进程的所有值都复制到新的新进程中，只有少数值与原来的进程的值不同。相当于克隆了一个自己。
（https://blog.csdn.net/weixin_41805011/article/details/81217491）

29、redis持久化机制：redis是一个支持持久化的内存数据库,也就是说redis需要经常将内存中的数据同步到磁盘来保证持久化，持久化可以避免因进程退出而造成数据丢失，有两种持久化方式：1、RDB持久化，2、AOF持久化。一般线上环境需要将两种持久化机制都打开。

29、RDB持久化：把当前进程数据生成快照（.rdb）文件保存到硬盘，有手动触发和自动触发：

- 手动触发Redis进行RDB持久化的命令有两种【SAVE、BGSAVE】：
  - save命令：该命令会阻塞当前Redis服务器，执行save命令期间，Redis不能处理其他命令，直到RDB过程完成为止。显然该命令对于内存比较大的实例会造成长时间阻塞，这是致命的缺陷，为了解决此问题，Redis提供了BGSAVE命令。
  - bgsave：执行该命令时，Redis会在后台异步进行快照操作，快照同时还可以响应客户端请求。具体操作是Redis进程执行fork操作创建子进程，RDB持久化过程由子进程负责，完成后自动结束。阻塞只发生在fork阶段，一般时间很短。执行redis-cli shutdown关闭redis服务时，如果没有开启AOF持久化，会自动执行bgsave。
- 自动触发：在redis.config中进行save配置：
  - 默认配置：



①、**save**：这里是用来配置触发 Redis的 RDB 持久化条件，也就是什么时候将内存中的数据保存到硬盘。比如“save m n”。表示m秒内数据集存在n次修改时，自动触发bgsave（这个命令下面会介绍，手动触发RDB持久化的命令）

默认如下配置：

```
save 900 1：表示900 秒内如果至少有 1 个 key 的值变化，则保存
save 300 10：表示300 秒内如果至少有 10 个 key 的值变化，则保存
save 60 10000：表示60 秒内如果至少有 10000 个 key 的值变化，则保存
```

当然如果你只是用Redis的缓存功能，不需要持久化，那么你可以注释掉所有的 save 行来停用保存功能。可以直接一个空字符串来实现停用：save ""

- **原理：redis会轮询检查是否达到触发RDB持久化的条件，如果符合条件则通过bgsave指令进行RDB持久化。因此自动触发持久化时也是通过bgsave指令，即通过fork函数创建子进程。**

**29、RDB持久化的优缺点：**

- **优点：1、压缩后的二进制文件比较紧凑，适用于备份、全量复制，用于灾难恢复；2、加载RDB恢复数据远快于AOF方式。**
- **缺点：1、无法做到实时持久化，每次都要创建子进程，频繁操作成本过高；2、保存后的二进制文件，存在老版本不兼容新版本rdb文件的问题。**

**29、为什么 Redis 使用子进程而不是线程来进行后台 RDB 持久化呢？主要是出于Redis性能的考虑，我们知道Redis对客户端响应请求的工作模型是单进程和单线程的，如果在主进程内启动一个线程，这样会造成对数据的竞争条件。所以为了避免使用锁降低性能，Redis选择启动新的子进程，独立拥有一份父进程的内存拷贝，以此为基础执行RDB持久化。但是需要注意的是，fork 会消耗一定时间，并且父子进程所占据的内存是相同的，当 Redis 键值较大时，fork 的时间会很长，这段时间内 Redis 是无法响应其他命令的。除此之外，Redis 占据的内存空间会翻倍。**

**30、AOF持久化：针对RDB不适合实时持久化，redis提供了AOF持久化方式来解决。**

- **配置详解：**

```
appendonly yes        //启用aof持久化方式
# appendfsync always //每收到写命令就立即强制写入磁盘，最慢的，但是保证完全的持久化，不推荐使用
appendfsync everysec //每秒强制写入磁盘一次，性能和持久化方面做了折中，推荐
# appendfsync no      //完全依赖os，性能最好,持久化没保证（操作系统自身的同步）
no-appendfsync-on-rewrite  yes  //正在导出rdb快照的过程中,要不要停止同步aof
auto-aof-rewrite-percentage 100  //aof文件大小比起上次重写时的大小,增长率100%时,重写
auto-aof-rewrite-min-size 64mb   //aof文件,至少超过64M时,重写
```

- **AOF文件将指令以resp报文格式保存，具体如下图所示：**


**30、AOF数据恢复：AOF 文件里边包含了重建 Redis 数据所需的所有写命令（刚开始的AOF中保存了所有命令的resp报文，当重写以后就只剩所有写命令的resp报文了），所以 Redis 只要读入并重新执行一遍 AOF 文件里边保存的写命令，就可以还原 Redis 关闭之前的状态，步骤如下：**

**1. 创建一个不带网络连接的的伪客户端( fake client)，因为 Redis 的命令只能在客户端上下文中执行，而载入 AOF 文件时所使用的的命令直接来源于 AOF 文件而不是网络连接，所以服务器使用了一个没有网络连接的伪客户端来执行 AOF 文件保存的写命令，伪客户端执行命令的效果和带网络连接的客户端执行命令的效果完全一样的。**

**2. 从 AOF 文件中分析并取出一条写命令。**

**3. 使用伪客户端执行被读出的写命令。**

**4. 一直执行步骤 2 和步骤3，直到 AOF 文件中的所有写命令都被处理完毕为止。**

**30、使用子进程进行AOF重写产生的问题**

**（https://blog.csdn.net/hezhiqiang1314/article/details/69396887）。**

**30、使用 Redis 附带的 redis-check-aof 程序，可以对出错的AOF 文件进行修复。**

**30、AOF文件重写不是简单的将原有的AOF文件压缩，它是重新读取redis库，用最少的命令来记录当前库的数据全量状态。因为AOF文件中记录的是客户端执行命令的RESP报文，假如我们在redis中插入了key为test、value为1的String类型键值即【set test 1】，而后边我们又对这个key执行了一次插入操作对原有值进行覆盖【set test 2】，现在数据库中key为test的键对应的值为2。再AOF文件重写之前AOF文件中记录了两条resp报文（因为执行了两次set操作），而当AOF文件重写之后新的AOF文件中就只有一条resp报文（即【set test 2】这条命令的报文），这样不仅正确记录了数据的最终状态，而且记录的报文数量减少，则AOF文件就会变小。**

**30、总结一下AOF重写原理：通过重新读取redis库，将文件中无效的命令去除。比如：**

- **同一个key的值，只保留最后一次写入；**
- **已删除或者已过期数据相关命令会被去除，这样就避免了，aof文件过大而实际内存数据小的问题(如频繁修改数据时，命令很多，实际数据很少)。**

**31、一主多从：针对“读”较多的场景，“读”由多个从节点来分担，但节点越多，主节点同步到多节点的次数也越多，影响带宽，也加重主节点的稳定。一般采用一主一从或者一主两从，同时从节点读写权限一般配置为只读，因为通过从节点写数据会造成主从数据不一致的问题（因为通过从节点写入的数据无法同步到主节点）。配置方式，在从节点的配置文件中假如slaveof masterIp port**

**31、主从复制的缺点：**

1. **不支持高可用，若主节点出现问题，则不能提供服务，需要人工修改配置将从变主；**
2. **主从复制主节点的写能力单机，能力有限；**
3. **单机节点的存储能力也有限。**

**32、哨兵只解决了主从复制模式无法自动进行故障迁移的问题。且哨兵模式只能主节点进行写，存储能力有限（各机器存储的内容相同，浪费内存），部署比较麻烦。**

**33、JedisCluster为什么不需要如JedisPool一样的池来管理?因为JedisCLuster每次操作Redis时都会去获取新的连接。**

**34、Redis集群部署采用了槽分区算法，crc16。将16384个槽位平均到不同的机器。分配算法**
**【crc16（key）& 16383】得到的值确定是在那台机器（16383是因为从0开始，一共有16384个槽，这是操作系统规定的）。假如集群部署了四主四从，则1~5个主从分配到的槽位分别为0~4095、4096~8191、8192~12281、12282~16383，如果crc16（key）& 16383结果为4000，则被分配到第一台机器，如果为9000，则被分配到第三台机器。**

**34、Redis使用集群部署后发送请求会有路由重定向的操作。**

**34、Redis集群部署以后批操作不能保证原子性了，因为不同的key会被hash到不同的机器。所以说使用集群以后lua脚本的原子性也无法保证了。**

**34、Redis集群模式解决了存储内存大小的限制，可以在多台机器分散存储。同时也保证了高可用。**

**35、键空间通知和键事件通知的区别，当执行del mykey时：**

- **键空间频道的订阅者将接收到被执行的事件的名字，在这个例子中，就是 del 。**
- **键事件频道的订阅者将接收到被执行事件的键的名字，在这个例子中，就是 mykey 。**

**36、选择监听键空间通知还是键事件通知需根据业务需求来。当我们需要通过监听事件获取被执行的key时就选择键事件通知（例如获取过期key），当我们需要知道执行了那些指令是就选用键空间通知。**