51.TCP的TIME_WAIT和CLOSE_WAIT介绍下？如果系统中出现了大量的CLOSE_WAIT怎么排查？

TIME_WAIT : 2MSL

CLOSE_WAIT : 服务端收到FIN信号后进入该状态，该状态即为服务端处理完响应逻辑代码的阶段？

通常，CLOSE_WAIT 状态在服务器停留时间很短，如果你发现大量的 CLOSE_WAIT 状态，那么就意味着被动关闭的一方没有及时发出 FIN 包，一般有如下几种可能：

1程序问题：如果代码层面忘记了 close 相应的 socket 连接，那么自然不会发出 FIN 包，从而导致 CLOSE_WAIT 累积；

或者代码不严谨，出现死循环之类的问题，导致即便后面写了 close 也永远执行不到。

2响应太慢或者超时设置过小：如果连接双方不和谐，一方不耐烦直接 timeout，另一方却还在忙于耗时逻辑，就会导致 close 被延后。

响应太慢是首要问题，不过换个角度看，也可能是 timeout 设置过小。

52.往redis存对象，使用String好还是Hash结构好，分析优缺点？

当如下情况时redis的hash类型，底层是用ziplist编码的：

哈希对象保存的所有键值对的键和值的字符串长度都小于 64 字节；

哈希对象保存的键值对数量小于 512 个；

不满足上述情况时，redis的hash类型，底层编码格式为hashtable。

如果你的业务类型中对于缓存的读取缓存的场景更多，并且更新缓存不频繁（或者每次更新都更新json数据中的大多数key），那么选择string类型作为存储方式会比较好。

如果你的业务类型中对于缓存的更新比较频繁（特别是每次只更新少数几个键）时， 或者我们每次只想取json数据中的少数几个键值时，我们选择hash类型作为我们的存储方式会比较好。.

53.sentinel模式故障转移怎么做到外部无感知？

故障转移后客户端无法感知将无法保证正常的使用。所以，实现客户端高可用的步骤如下:

1.客户端获取sentinel节点集合

2.客户端通过sentinel get-master-addr-by-name master-name这个api来获取对应主节点信息

3.客户端验证当前获取的“主节点”是真正的主节点，这样的目的是为了防止故障转移期间主节点的变化

4.客户端保持和sentinel节点集合的联系，即订阅sentinel节点相关频道，时刻获取关于主节点的相关信息

从上面的模型可以看出，Redis sentinel客户端只有在初始化和切换主节点时需要和sentinel进行通信来获取主节点信息，

所以在设计客户端时需要将sentinel节点集合考虑成配置（相关节点信息和变化）发现服务。

54.介绍下java线程join方法

join方法是实现线程同步，可以将原本并行执行的多线程方法变成串行执行的。源码还是比较容易理解的，其实就是调用了现场wait方法实现线程同步的

join方法的作用是，举个例子，在A线程里调B线程的join方法时，要先B线程执行完成，然后才会继续执行A线程

上面调join方法是不加参数的，也可以加上参数，比如线程A.join(10)；，就是说线程A执行10s后，继续执行B线程

注意：join时间参数缺省的情况，默认是0，也就是说join()等同于join(0);0不是表示执行0s，而是表示要A线程执行完成才继续执行B线程的意思。

55.线程池用到哪些阻塞队列?

线程池	阻塞队列
FixedThreadPool	LinkedBlockingQueue
SingleThreadExecutor	LinkedBlockingQueue
CachedThreadPool	SynchronousQueue
ScheduledThreadPool	DelayedWorkQueue
SingleThreadScheduledExecutor	DelayedWorkQueue

LinkedBlockingQueue

第一种阻塞队列是 LinkedBlockingQueue，它的容量是 Integer.MAX_VALUE，是一个非常大的值，可以认为是无界队列。

FixedThreadPool 和 SingleThreadExecutor 线程池的线程数是固定的，所以没有办法增加特别多的线程来处理任务，

这时就需要 LinkedBlockingQueue 这样一个没有容量限制的阻塞队列来存放任务。

SynchronousQueue

第二种阻塞队列是 SynchronousQueue，对应的线程池是 CachedThreadPool。线程池 CachedThreadPool 的最大线程数是 Integer.MAX_VALUE，可以理解为线程数是可以无限扩展的。

CachedThreadPool 和上一种线程池 FixedThreadPool 的情况恰恰相反，FixedThreadPool 的情况是阻塞队列的容量是无限的，而这里 CachedThreadPool 是线程数可以无限扩展，

所以 CachedThreadPool 线程池并不需要一个任务队列来存储任务，因为一旦有任务被提交就直接转发给线程或者创建新线程来执行，而不需要另外保存它们。

我们自己创建使用 SynchronousQueue 的线程池时，如果不希望任务被拒绝，那么就需要注意设置最大线程数要尽可能大一些，

以免发生任务数大于最大线程数时，没办法把任务放到队列中也没有足够线程来执行任务的情况。

DelayedWorkQueue

第三种阻塞队列是DelayedWorkQueue，它对应的线程池分别是 ScheduledThreadPool 和 SingleThreadScheduledExecutor，

这两种线程池的最大特点就是可以延迟执行任务，比如说一定时间后执行任务或是每隔一定的时间执行一次任务。

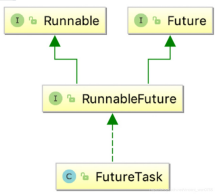
DelayedWorkQueue 的特点是内部元素并不是按照放入的时间排序，而是会按照延迟的时间长短对任务进行排序，内部采用的是“堆”的数据结构。

之所以线程池 ScheduledThreadPool 和 SingleThreadScheduledExecutor 选择 DelayedWorkQueue，是因为它们本身正是基于时间执行任务的，而延迟队列正好可以把任务按时间进行排序，方便任务的执行。

56.线程知识中Future 的作用？

简单来说就是利用线程达到异步的效果，同时还可以获取子线程的返回值。

我们可以把运算的过程放到子线程去执行，再通过 Future 去获取到计算结果。这样一来就可以把整个程序的运行效率提高，是一种异步的思想。



FutureTask实现了两个接口，Runnable和Future，所以它既可以作为Runnable被线程执行，又可以作为Future得到Callable的返回值

典型用法是，把 Callable 实例当作 FutureTask 构造函数的参数，生成 FutureTask 的对象，

然后把这个对象当作一个 Runnable 对象，放到线程池中或另起线程去执行，最后还可以通过 FutureTask 获取任务执行的结果

57.cpu如何寻址？

处理器采用多级页表来进行多次查找最终找到真正的物理地址。

由于页表是存放在内存中的，使用一级页表进行地址转换时，每次读/写数据需要访问两次内存，第一次访问一级页表获得物理地址，第二次才是真正的读/写数据；

使用两级页表时，每次读/写数据需要访问三次内存，访问两次页表（一级页表和二级页表）获得物理地址，第三次才是真正的读/写数据。

拿处理器访问两级页表举例说明，当处理器拿到一个需要访问内存的虚拟地址，MMU首先访问TLB Cache（近期页表的缓存），如果TLB Cache中含有能转换这个虚拟地址的描述符，

则直接利用此描述符进行地址转换和权限检查；否则MMU访问页表（页表是在主存中）找到描述符后再进行地址转换和权限检查，并将这个描述符填入TLB Cache中，

下次再使用这个虚拟地址时就可以直接使用TLB Cache中的地址描述符了。

58.为什么要三次握手?

为了实现可靠数据传输， TCP 协议的通信双方， 都必须维护一个序列号， 以标识发送出去的数据包中， 哪些是已经被对方收到的。

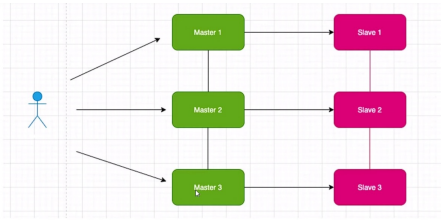
三次握手的过程即是通信双方相互告知序列号起始值， 并确认对方已经收到了序列号起始值的必经步骤。

如果只是两次握手， 至多只有连接发起方的起始序列号能被确认， 另一方选择的序列号则得不到确认。

> 同时这样可以防止已失效的连接请求又传送到服务器端，因而产生错误。

59.Redis的Cluster集群模式？

结构：



每个节点主数据不同，是数据的子集

利用多台服务器构建集群提高超大规模数据处理能力

同时提供高可用支持

分配策略：

> 特点：

- Cluster模式是Redis3.0开始推出
- 采用无中心结构，每个节点保存数据和整个集群状态，每个节点都和其他所有节点连接
- 官方要求：至少6个节点才可以保证高可用，即3主3从；扩展性强、更好做到高可用
- 各个节点会互相通信，采用gossip协议交换节点元数据信息
- 数据分散存储到各个节点上

> 故障转移：

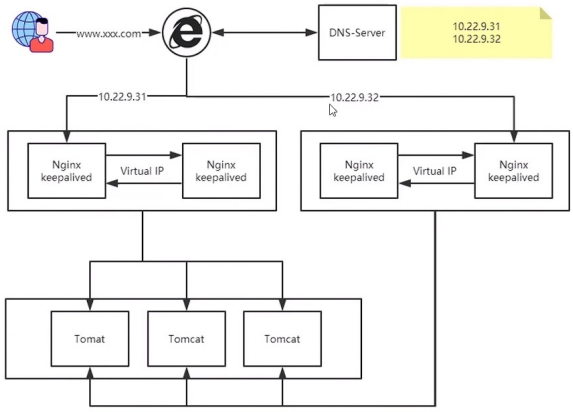
```
#设置集群时 节点连接超时时间 cluster-node-timeout 20000 ,因此20秒内持续尝试连接
21786:S 22 Jul 2021 20:14:09.732 # Error condition on socket for SYNC: Connec
21786:S 22 Jul 2021 20:14:10.765 * Connecting to MASTER 192.168.31.102:6379
21786:S 22 Jul 2021 20:14:10.766 * MASTER <-> REPLICA sync started
21786:S 22 Jul 2021 20:14:10.766 # Error condition on socket for SYNC: Connec
21786:S 22 Jul 2021 20:14:11.798 * Connecting to MASTER 192.168.31.102:6379
21786:S 22 Jul 2021 20:14:11.799 * MASTER <-> REPLICA sync started
21786:S 22 Jul 2021 20:14:11.800 # Error condition on socket for SYNC: Connec
21786:S 22 Jul 2021 20:14:12.571 * FAIL message received from 7d1b82830052fe1
#确认102主已经挂了,设置状态fail
21786:S 22 Jul 2021 20:14:12.572 # Cluster state changed: fail
21786:S 22 Jul 2021 20:14:12.627 # Start of election delayed for 504 millisec
21786:S 22 Jul 2021 20:14:12.831 * Connecting to MASTER 192.168.31.102:6379
21786:S 22 Jul 2021 20:14:12.831 * MASTER <-> REPLICA sync started
21786:S 22 Jul 2021 20:14:12.833 # Error condition on socket for SYNC: Connec
#开始故障转移
21786:S 22 Jul 2021 20:14:13.144 # Starting a failover election for epoch 7.
#111变为新的主
21786:S 22 Jul 2021 20:14:13.148 # Failover election won: I'm the new master.
21786:S 22 Jul 2021 20:14:13.149 # configEpoch set to 7 after successful fail
21786:M 22 Jul 2021 20:14:13.149 * Discarding previously cached master state.
21786:M 22 Jul 2021 20:14:13.149 # Setting secondary replication ID to c31d50
21786:M 22 Jul 2021 20:14:13.149 # Cluster state changed: ok
```

102启动恢复后,将以从的身份执行任务

```
[root@localhost redis-stable]# cat nodes.conf
d19364caee69daf2cf1d790b0eb9d5742c294154 192.168.31.103:6379@16379 master - 0
e8d7c57b764f77daf5da251a0db4b5996235c750 192.168.31.111:6379@16379 master - 0
7d1b82830052fe187f81fbcc7ce1470b0e975313 192.168.31.104:6379@16379 master - 0
2c9ee8a7ccab0c597bed8c99545709359b7ff2d5 192.168.31.110:6379@16379 myself,sla
#102被永久降级为从,并在启动时向111重新进行全量同步
6714367aa4a9d2379862ebc72b54ef3dad98acdd 192.168.31.102:6379@16379 slave e8d7
08f39d0917d24c4ebee04e47d921466dc169dda4 192.168.31.112:6379@16379 slave d193
vars currentEpoch 7 lastVoteEpoch 0
```

60.说说Web高可用架构？

DNS轮询与多VIP组合解决利率用问题



安全层面: 做好DNS劫持的预防

设备方面: 2*Nginx + N*Tomcat + 1*MySQL