1. redis缓存穿透的解决方案
    - 为热点数据加锁
2. O(nlogn)的排序算法
    - 快排、堆排(均不稳定)
    - 归并排序(稳定)
3. golang中sort使用了哪种排序方法，仅仅是快排吗？
    - 使用了堆排序、希尔排序、插入排序
4. golang多线程模型
    - GMP模型
    -
5. 快排的实现原理
    - 快排是一种应用了分治法的排序
    - 选择一个key值。然后依次将key后方小于key的值交换到key前方，将key前方大于key的值交换到key后方
    - 对key的左右两部分递归调用快排，直到每个部分都有序，总体即有序
6. go解决并发冲突的方案
    - channel
    - 加锁
7. 关闭一个channel后，对其读写会怎样
    - 关闭的channel仍然可以读取数据，但是当其中的数据读取完之后，继续读的话会读取到一堆默认值
    - 读取channel时最好同时获取两个返回值，第二个返回值是 `bool` 值，可以反映该channel是否已关闭
8. Tcp协议中timewait的作用
    - timewait是指在Tcp四次挥手中主动关闭连接的一方在发送完最后一次挥手之后，主动关闭连接的一方所处的状态
    - timewait状态的持续时间为2MSL，MSL是“报文最大生存时间”，可为30s,1min或2min,2MSL就是两倍的这个时间
    - timewait的作用：
        - 保证客户端发送的最后一个挥手到达服务器，如果没到达，服务端就会重发第三次挥手
        - 保证本次连接的所有报文段从网络中消失
9. Tcp为什么可靠?
    - ACK确认机制
    - 超时重传
    - 滑动窗口
    - 流量控制，深入的话要求详细讲出流量控制的机制
10. redis的基本数据类型
    - string\list\hash\set\zset

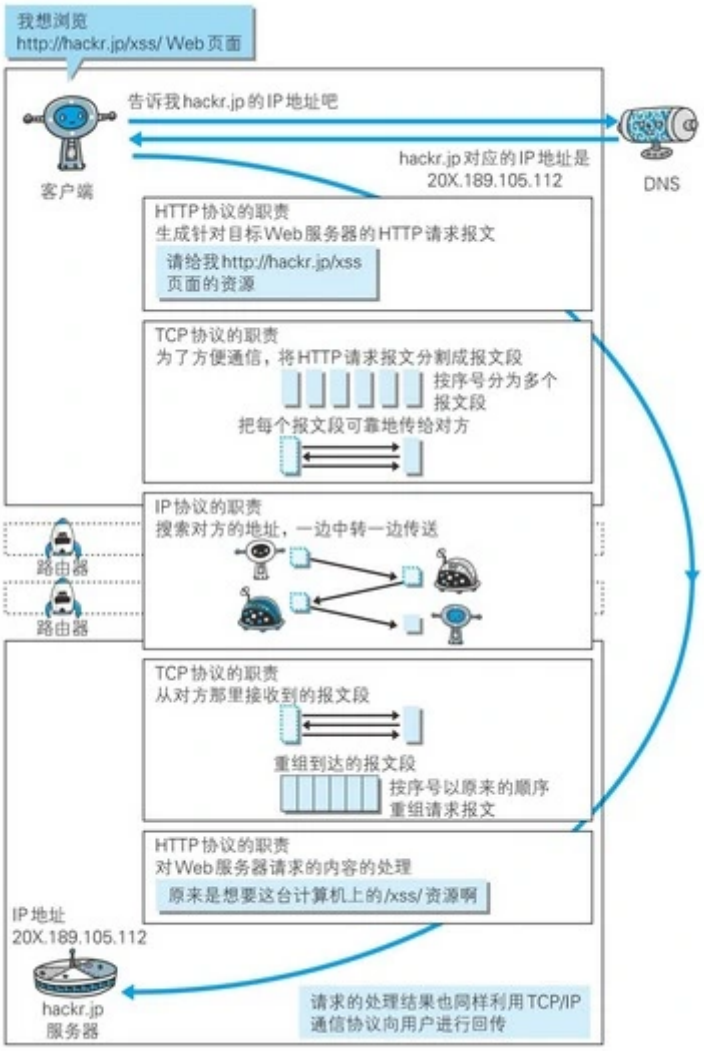
11. Redis中的sorted set，是在skiplist, dict和ziplist基础上构建起来的:
    - 当数据较少时，sorted set是由一个ziplist来实现的。
    - ziplist: 一个顺序链表
    - 当数据多的时候，sorted set是由一个叫zset的数据结构来实现的，这个zset包含一个dict + 一个skiplist。dict用来查询数据到分数(score)的对应关系，而skiplist用来根据分数查询数据（可能是范围查找）。
12. zset使用的数据结构：跳跃表(skiplist)
    - 一种有序链表
    - 链表拥有多层，最底层的节点依次指向相邻节点，上层节点指向后续不相邻节点
    - 查找原理上类似二分法，先从高层查，从而可以确认下一层的查找范围
13. redis的zset为什么使用跳跃表
    - 哈希表无序，直接排除
    - 平衡树在增删和范围查找的操作中逻辑比跳跃表复杂很多，实现难度上也是跳跃表更容易实现
    - 内存占用上，跳表更加灵活
14. hash的实现原理
    - 哈希表本质是一种(key,value)结构
    - 由此我们可以联想到，能不能把哈希表的key映射成数组的索引index呢？
    - 如果这样做的话那么查询相当于直接查询索引，查询时间复杂度为O(1)
    - 其实这也正是当key为int型时的做法 将key通过某种做法映射成index，从而转换成数组结构
15. hash实现步骤
    - 使用hash算法计算key值对应的hash值h(默认用key对应的hashcode进行计算(hashcode默认为key在内存中的地址)),得到hash值
    - 计算该(k,v)对应的索引值index ，索引值的计算公式为 index = (h % length) length为数组长度(取余法哈希)
    - 储存对应的(k,v)到数组中去，从而形成a[index] = node<k,v>,如果a[index]已经有了结点即可能发生碰撞，那么需要通过开放寻址法或拉链法(Java默认实现）解决冲突
    - 开放寻址法：从冲突地址向后寻找空闲地址
    - 拉链法：在冲突地址存储一个同义词链表
16. 队列的实现原理
17. 栈的实现原理
18. mysql的索引用什么实现的？
    - b+tree
19. 索引的原理
    - 索引的目的在于提高查询效率，与我们查阅图书所用的目录是一个道理：先定位到章，然后定位到该章下的一个小节，然后找到页数。相似的例子还有：查字典，查火车车次，飞机航班等
    - 本质都是：通过不断地缩小想要获取数据的范围来筛选出最终想要的结果，同时把随机的事件变成顺序的事件，也就是说，有了这种索引机制，我们可以总是用同一种查找方式来锁定数据。
20. b+tree的实现原理

- 多路非二叉树
- 只有叶子节点才会存数据，中间节点都作为索引
- 中间节点可以存储叶子结点的最大值或最小值，用以二分查找
- 单节点可以存储更多的元素，使得查询磁盘IO次数更少。
- 所有查询都要查找到叶子节点，查询性能稳定。
- 所有叶子节点形成有序链表，便于范围查询。

21. 高并发下保证数据唯一性方案
    - uuid生成全球唯一id,生成方式简单粗暴，本地生成，没有网络开销，效率高；缺点长度较长，没有递增趋势性，不易维护，常用于生成token令牌。
    - zookeeper通过创建顺序节点生成全局id,在高并发场景下，性能不能很好。
    - mysql自带自增生成id，oracle可以用序列生成id,但在数据库集群环境下，扩展性不好。
    - 基于雪花算法snowflake 生成全局id，本地生成，没有网络开销，效率高，但是依赖机器时钟。
    - 基于redis单线程的特点生成全局唯一id，redis性能高，支持集群分片。
22. 网络IO模型有哪些?
    - 5种网络I/O模型，阻塞、非阻塞、I/O多路复用、信号驱动IO、异步I/O。
23. 从数据从I/O设备到内核态，内核态到进程用户态分别描述这5种网络io模型的区别。
    - 内核态：数据处于与其他进程共享的空间
    - 用户态：数据处于进程私有空间
24. I/O多路复用中select/poll/epoll的区别
    - select原理
        - select在调用之前，需要手动在应用程序里将要监控的文件描述符添加到fed_set集合中。然后加载到内核进行监控。用户为了检测时间是否发生，还需要在用户程序手动维护一个数组，存储监控文件描述符。当内核事件发生，在将fed_set集合中没有发生的文件描述符清空，然后拷贝到用户区，和数组中的文件描述符进行比对。再调用selecct也是如此。每次调用，都需要了来回拷贝。
    - select/poll
        - poll相对于select只是数据结构与操作方式略微进行了优化，select的缺点poll也有
        - 缺点1：单个进程监控的文件描述符有限
        - 缺点2：采用轮询的方式扫描文件描述符，性能较差
        - 缺点3：需要频繁地进行内核态与用户态之间的数据拷贝，操作复杂
        - 缺点4：水平触发，如果下一次轮询时还未完成上一次返回的就绪文件的io操作，则下一次轮询仍然会返回该就绪文件的句柄
    - epoll
        - 没有select的缺点
        - epoll在系统中申请一个简易的文件系统，将select调用分为三个部分
            - 调用epoll_create创建一个epoll对象，包含一个红黑树和一个双向链表，并与底层建立回调机制
            - 调用epoll_ctl向epoll对象中添加文件套接字

        - 调用epoll_wait收集发生事件的文件
    - 而我们调用epoll_wait时就相当于以往调用select/poll，但是这时却不用传递socket句柄给内核，因为内核已经在epoll_ctl中拿到了要监控的句柄列表。
    - 所以，实际上在调用epoll_create后，内核就已经在内核态开始准备存储要监控的句柄了，每次调用epoll_ctl只是在往内核的数据结构里塞入新的socket句柄。
  - epoll的高效就在于当我们调用epoll_ctl往里塞入百万个句柄时，epoll_wait仍然可以飞快的返回，并有效的将发生事件的句柄给我们用户。这是由于我们在调用epoll_create时，内核除了帮我们在epoll文件系统里建了个file结点，在内核cache里建了个红黑树用于存储以后epoll_ctl传来的socket外，还会再建立一个双向链表，用于存储准备就绪的事件，当epoll_wait调用时，仅仅观察这个list链表里有没有数据即可。有数据就返回，没有数据就sleep，等到timeout时间到后即使链表没数据也返回。所以，epoll_wait非常高效
25. 常用http状态码
  - 1xx：通知
  - 2xx: 成功
  - 3xx: 重定向
  - 4xx: 客户端错误
  - 5xx: 服务端错误
  - 常用状态码：常用状态码
26. webSocket协议与http协议的异同
  - 同：都建立在tcp协议之上，通过tcp传输数据
  - 异：
    - HTTP协议为单向协议，即浏览器只能向服务器请求资源，服务器才能将数据传送给浏览器，而服务器不能主动向浏览器传递数据。分为长连接和短连接，短连接是每次http请求时都需要三次握手才能发送自己的请求，每个request对应一个response；长连接是短时间内保持连接，保持TCP不断开，指的是TCP连接
    - WebSocket解决客户端发起多个http请求到服务器资源浏览器必须要经过长时间的轮询问题。
    - 一种双向通信协议，在建立连接后，WebSocket服务器和Browser/UA都能主动的向对方发送或接收数据，就像Socket一样，不同的是WebSocket是一种建立在Web基础上的一种简单模拟Socket的协议
    - WebSocket需要通过握手连接，类似于TCP它也需要客户端和服务器端进行握手连接，连接成功后才能相互通信
    - WebSocket在建立握手连接时，数据是通过http协议传输的，“GET/chat HTTP/1.1”，这里面用到的只是http协议一些简单的字段。但是在建立连接之后，真正的数据传输阶段是不需要http协议参与的。
27. 输入url之后发生了什么？
  - 浏览器递归查找域名对应的ip，递归查找本地缓存，路由器缓存，dns服务器，域名服务器
  - 浏览器使用http协议包装数据生成请求报文，包含请求行，请求头，空行，请求数据
  - 浏览器与服务器使用tcp协议三次握手建立连接

- 浏览器将请求报文发送至服务器
- 服务器处理请求报文，包装响应数据(包含状态行，响应头，空行，响应数据)
- 服务器返回响应数据
- 短连接则关闭该次tcp连接，长连接则继续保持连接
- 浏览器解析响应数据，生成界面



28. 复习掌握常用排序算法的复杂度及实现原理排序算法原理
29. 树的前序遍历、中序遍历、后序遍历(递归解法/非递归解法)

网上常见面试题

Redis 部分

- Redis的应用场景
- Redis支持的数据类型（必考）
- zset跳表的数据结构（必考）
- Redis的数据过期策略（必考）
- Redis的LRU过期策略的具体实现
- 如何解决Redis缓存雪崩，缓存穿透问题

- Redis的持久化机制（必考）
  - RDB
  - AOF
- Redis的管道pipeline
  - 一次发送多个请求，redis返回处理好的请求

Mysql 部分

- 事务的基本要素
- 事务隔离级别
- 如何解决事务的并发问题(脏读，幻读)？
- MVCC多版本并发控制？
- binlog,redolog,undolog都是什么，起什么作用？
- InnoDB的行锁/表锁？
- myisam和innodb的区别，什么时候选择myisam？
- 为什么选择B+树作为索引结构？
- 索引B+树的叶子节点都可以存哪些东西？
- 查询在什么时候不走（预期中的）索引？
- sql如何优化?
- explain是如何解析sql的？
- order by原理