1. golang调度

runtime负责管理任务调度，垃圾收集与运行环境

goroutione是Go原生支持并发的具体实现，go的代码无一例外的跑在goroutine中。Go的runtime负责对goroutine进行调度。（调度就是决定何时哪个goroutine将获得资源开始执行，哪个goroutine应该停止执行让出资源，哪个goroutine应该被唤醒开始执行。）

GPM模型

> G： 表示goroutine，存储了gorotine的执行stack信息、gorotine状态以及goroutine的任务函数等；另外G对象是可以重用的。
> - P：表示processer，P的数量决定了系统内最大可并行的G的数量(系统的物理cpu核数>=P的数量)；P的最大作用还是还是其拥有的各种G对象队列，链表，一些cache和状态。
> - M：M代表着真正的执行计算的资源，在绑定有效的P后，进入schedule循环；而schedule循环机制大致是从各中队列，P的本地对列中获取G，切换到G的执行栈中并执行G的函数，调用goexit做清理工作并回到M，如果反复。M并不保留G的状态，这是G可以跨M调度的基础。

2.Go struct 能不能比较?

> 因为是强类型语言，所以不同类型的结构不能作比较，但是同一类型的实例值是可以比较的，实例不可以比较，因为是指针类型

3. go defer 先进后出,后进先出

4. select可以用于什么，常用语gorotine的完美退出

> select 就是用来监听和channel有关的IO操作，当IO操作反生时，触发相应的动作。每个case语句里必须是一个IO操作，确切的说，应该是一个面向channel的IO操作，如果IO触发，则实现对应的case代码块。没有则触发defult。

5. 当对channel进行迭代的时候，必须对channel进行关闭，否则会造成程序死锁。close(chann)之后，是对写通道的关闭，channel里的数据还是可以读出来。关闭channel之后，写数据会抛出panic错误。

6、主协程如何等其余协程完再操作

> 1.可以通过channel+select或者context，来实现。
> 2.通过sync.WaitGroup 来等待子协程完成了之后退出。

7. slice扩容 -- 使用append函数，

8. map如何顺序读取

> map不能顺序读取，是因为他是无序的，想要有序读取，首先的解决的问题就是，把key变为有序，所以可以把key放入切片，对切片进行排序，遍历切片，通过key取值。

9、golang 实现set

```
// 可以用map的健不能重复实现
type inter interface{}
type Set struct {
    m map[inter]bool
    sync.RWMutex
}
func New() *Set {
    return &Set{
    m: map[inter]bool{},
    }
}
func (s *Set) Add(item inter) {
    s.Lock()
    defer s.Unlock()
    s.m[item] = true
}
```

10. time-wait的作用

> time-wait开始的时间为TCP四次挥手中主动关闭连接放发送完最后一次挥手，主动关闭连接方所处的状态。
> time-wait的最长持续时间是2倍的报文最大生存时间，这个时间为网络情况决定。
> 为什么这么长时间：1.为了保证客户端发送的最后一个ack报文段能够达到服务器，防止报文丢失。
> 2.在第四次挥手后，经过2msl的时间足以让本次连接产生的所有报文段都从网络中消失，这样下一次新的连接中就肯定不会出现旧连接的报文段了。（防止已经失效的报文存在新的连接中）

11、http能不能一次连接多次请求，不等后端返回

> http本质上市使用socket连接，因此发送请求，接写入TCP缓冲，是可以多次进行的，这也是http是无状态的原因

12、孤儿进程 、僵尸进程

孤儿进程：一个父进程退出，而它的一个或多个子进程还在运行，那么那些子进程将成为孤儿进程。孤儿进程将被init进程(进程号为1)所收养，并由init进程对它们完成状态收集工作。

僵尸进程：一个进程使用fork创建子进程，如果子进程退出，而父进程并没有调用wait或waitpid获取子进程的状态信息，那么子进程的进程描述符仍然保存在系统中。这种进程称之为僵死进程。

13、死锁条件，如何避免

互斥条件：一个资源每次只能被一个进程使用，即在一段时间内某资源仅为一个进程所占有。此时若有其他进程请求该资源，则请　　求进程只能等待。

请求与保持条件：进程已经保持了至少一个资源，但又提出了新的资源请求，而该资源 已被其他进程占有，此时请求进程被阻塞，但对自己已获得的资源保持不放。

不可剥夺条件:进程所获得的资源在未使用完毕之前，不能被其他进程强行夺走，即只能 由获得该资源的进程自己来释放（只能是主动释放)。

循环等待条件：若干进程间形成首尾相接循环等待资源的关系

死锁的避免方法：
1）设置加锁顺序
　　假如在多线程中，一个线程需要锁，那么他必须按照一定得顺序获得锁。

2）设置加锁时限
　　在获取锁的时候尝试加一个获取锁的时限，超过时限不需要再获取锁，放弃操作对锁的请求。

3）死锁检测
　当一个线程获取锁的时候，会在相应的数据结构中记录下来，相同下

14、golang反射

反射机制就是在运行时动态的调用对象的方法和属性。

golang的类型设计原则：
- 变量包括（type, value）两部分
  理解这一点就知道为什么nil != nil了

- type 包括 static type和concrete type. 简单来说 static type是你在编码是看见的类型(如int、string), concrete type是runtime系统看见的类型
- 类型断言能否成功，取决于变量的concrete type，而不是static type.

interface及其pair的存在是golang反射的前提。反射就是用来检测存储在接口变量内部(值value；类型concrete type) pair对的一种机制。

反射的解基本介绍：
1)：反射可以在运行时动态获取变量的各种信息，比如类型(type)、类别(kind).
2)：如果是结构体变量，还可以获取结构体本身的信息(包括结构体的字段、方法)
通过反射，可以修改变量的值，可以调用关联的方法。
使用反射，需要import ("reflect")

15、类型和类别

type 是类型。Kind 是类别。Type 和 Kind 可能相同，也可能不同。通常基础数据类型的 Type 和 Kind 相同，自定义数据类型则不同。

16 redis的基本数据结构

String(字符串)、list(列表)、set(集合)、hash(哈希)、zset(有序集合)

17、Golang中除了加Mutex锁以外还有哪些方式安全读写共享变量?

golang中的goroutine可以通过channel进行安全读写共享变量

18、无缓冲 Chan 的发送和接收是否同步?

无缓冲的channle需要发送和接收进行同步，否则会造成阻塞。 有缓冲的channel可以不进行读写同步，不过当缓冲区域满的时候，会造成写入阻塞；当缓冲区为空是，会造成读取阻塞。

19. Golang 中常用的并发模型?

1. 通过channel通知实现并发控制(简单示例)

```
func main() {
  var chan01 chan int
```

```
  chan01 = make(chan int)

  aList := [5]int{1, 2, 3, 4, 5}

  go func() {
    for index, value := range aList {
      fmt.Println(index, value)
      if index == len(aList)-1 {
        chan01 <- value
      }
    }
  }()
  <- chan01
}
```

2. 通过sync包中的WaitGroup实现并发控制

```
func main() {

  var w sync.WaitGroup

  aList := [5]int{1, 2, 3, 4, 5}
  w.Add(1)

  go func() {
    defer w.Done()
    for index, value := range aList {
      fmt.Println(index, value)
      if index == len(aList)-1 {
        fmt.Println(index)
      }
    }
  }()
  w.Wait()
}
```

######3. 通过上下文context进行控制

```
func main() {
  var w sync.WaitGroup
```

```
  ctx, cancle :=
context.WithCancel(context.Background())
  aList := [5]int{1, 2, 3, 4, 5}
  w.Add(1)
  go func(context context.Context) {
    defer w.Done()
  lable:
    for index, value := range aList {
      fmt.Println(index, value)
      if index == len(aList)-1 {
        select {
        case <-context.Done():
          break lable
        default:
          continue
        }
      }
    }
  }(ctx)

  cancle()
  w.Wait()
}
```

20 JSON 标准库对 nil slice 和 空 slice 的处理是一致的吗?

```
不一致,nil slice 是没有进行make的切片， 空slice是make([]int,
0)状态。
```

21、进程、线程和协程

进程是具有一定独立功能的程序关于某个数据集合上的一次运行活动,进程是系统进行资源分配和调度的一个独立单位。每个进程都有自己的独立内存空间，不同进程通过进程间通信来通信。由于进程比较重量，占据独立的内存，所以上下文进程间的切换开销（栈、寄存器、虚拟内存、文件句柄等）比较大，但相对比较稳定安全。

• 线程是进程的一个实体,是CPU调度和分派的基本单位,它是比进程更小的能独立运行的基本单位.线程自己基本上不拥有系统资源,只拥有一点在运行中必不可少的资源(如程序计数器,一组寄存器和栈),但是它可与同属一个进程的其他的线程共享进程所拥有的全部资源。线程间通信主要通过共享内存，上下文切换很快，资源开销较少，但相比进程不够稳定容易丢失数据。

• 协程是一种用户态的轻量级线程，协程的调度完全由用户控制。协程拥有自己的寄存器上下文和栈。协程调度切换时，将寄存器上下文和栈保存到其他地方，在切回来的时候，恢复先前保存的寄存器上下文和栈，直接操作栈则基本没有内核切换的开销，可以不加锁的访问全局变量，所以上下文的切换非常快

22 进程同步

进程同步的主要任务： 是对多个相关进程执行次序上进行协调，以使并发执行的诸进程之间能有效的共享资源和相互作用，从而是程序的执行具有可再现性。

同步机制遵循的原则：

1： 空闲让进。

2： 忙则等待。（保证对临界区的互斥访问。）

3： 有限等待。（有限代表有限的时间，避免时间过长。）

4： 让权等待 （当进程不能进入自己的临界区是，应该释放处理机，以免陷入盲等状态。）

23 进程间的通信方式有哪些?

1：共享存储器系统。

2：消息传递系统。

3：管道通信系统。

4：信号量：信号量是一个计数器，可以用来控制多个进程对共享资源的访问。

5：消息队列

6：共享内存。

7：套接字。

24 进程和线程的区别和联系

– 进程是具有一定独立功能的程序关于某个数据集合上的一次运行活动。进程是系统进行资源分配和调度的一个独立单位。
– 线程是进程的一个实体，是CPU调度和分派的基本单位，它是比进程更小的独立运行的基本单位。

进程和线程的关系：
– 进程有自己的独立的地址空间，线程没有。
– 进程是资源分配的最小单位，线程是cpu调度的最小单位。
– 进程和线程的通信方式不同。
– 进程上下文切换开销大，线程开销小。
– 进程挂掉了不会影响其他进程，线程挂掉了会影响其他线程。
– 对进程操作一般开销都比较大，对线程操作开销比较小。

25 什么是线程安全

如果多线程的程序运行结果是可预期的，而且与单线程的程序的结果一致，那么说明是“线程安全”

26 同步与异步

同步：
· 指一个进程在执行某个请求的时候，若该请求需要一段时间才能返回信息，那么，这个进程将会一直等待下去，直到收到返回信息才继续执行下去。
· 特点：
1．同步是阻塞模式；
2．同步是按顺序执行，执行完一个再执行下一个，需要等待，协调运行。

异步：
· 是指进程不需要一直等待下去，而是继续执行下边的操作， 不管其他进程的状态。当有消息返回时系统会通知进程进行处理，这样可以提高执行效率。

特点：
1．已不是非阻塞模式，无需等待。
2．异步是彼此独立，在等待某时间的过程中，继续做自己的事，不需要等待这一事件完成后再工作。线程是异步实现。

同步异步的优缺点：
同步可以避免出现死锁，读脏数据的情况发生。但同步需要等待资源访问结束，浪费时间，效率低。
异步可以提高效率，控制不好容易出现资源问题。

27 守护、僵尸、孤儿进程

- 守护进程：运行在后台的一种特殊进程，独立于控制终端并周期性的执行某些任务。
- 僵尸进程： 一个进程fork子进程，子进程退出，而父进程没有wait/waitpid子进程,那么，子进程的进程描述符仍保存在系统中，这样的进程称为僵尸进程。
- 孤儿进程：一个父进程退出，而它的一个或多个子进程还在运行，这些子进程称为孤儿进程。（孤儿进程将由init进程收养并对他们完成状态进行收集。）

28 IO多路复用使用场景

IO多路复用是指内核一旦发现进程指定的一个或多个IO条件准备读取，他就通知该进程：

- 当客户处理多个描述符时(一般是交互式输入和网络套接口)，必须使用I/O复用。
- 当一个客户同时处理多个套接字时，而这种情况是可能的，但很少出现。
- 如果一个TCP服务器纪要处理监听接口，又要处理已连接的接口，一般也要用到I/O复用。
- 如果一个服务器纪要处理TCP，又要处理udp，一般要使用I/O复用。
- 如果一个服务器要处理多个服务或者协议，一般要使用I/O复用。
- 与多进程和多线程技术相比，I/O多路复用技术的最大优势是系统开销小，系统不必创建进程/线程，也不必维护这些进程/线程， 从而大大减小了系统的开销。

29 TCP UDP的区别

TCP：是面向连接的流传输控制协议、具有高可用性， 确保传输数据的正确性，有验证重发机制，因此不会出现丢失或者乱序。

UDP：是无连接的数据报服务，不对数据报进行检查与修改，无须等待对方的应答，会出现分组丢失、重复、乱序，但具有较好的实时性，udp段结构比TCP的段结构简单，因此网络开销也小。

区别：

1. TCP面向连接，通信前建立双向连接；udp面向无连接，通信前不需要建立连接。
2. tcp保障可靠传输(按序、无差错、不丢失、不重复)；udp不保障可靠传输。
3. tcp面向字节流传输，UD面向数据报传输。

30 TCP的重发机制是怎么实现的?

1． 滑动窗口机制，确立收发的边界，能让发送方知道已经发送了多少(已确认)、尚未确认发送的字节数；让接收方知道。
2．选择重传，用于对传输出错的序列进行重传。

31 Get和Post的区别：

幂等：请求一次和请求无数次的效果是相同的。
Get方式的请求：浏览器会把http header 和data一并发送出去，服务器响应200。其请求长度受浏览器限制。一般做请求资源用。请求幂等。

Post方式请求：浏览器先发送header，服务器响应 100continue，浏览器在发送data，服务器返回200。

post比get多请求一次。安全其实并不安全。

32 cookies机制和session机制的区别：

1．cookies数据保存在客户端。session数据保存在服务端。
2．cookies可以减轻服务器的压力，不安全。
3.session相对来说比较安全，但是占用服务器资源。

33 http 和 https的区别：

https = http + ssl
1．https有ca证书，http一般没有。
2．http是超文本传输协议，信息是明文传输。https则是具有安全性的ssl加密传输协议。
3.http默认是80端口，https默认是443端口。
4．http的连接很简单，是无状态的；https协议是有ssl+http协议构建的可进行加密传输、身份认证的网络协议，比http协议安全。

33 端口相关

一共有 65535个端口可用
8080 端口 http服务备用端口
80   http默认端口
21 ftp服务器开放的端口
22 ssh端口
23 telnet
25 smtp
3306 mysql
6379 redis

34 网络分层

因特网协议栈共有五层：应用层、传输层、网络层、链路层和物理层。

七层因特网协议栈：应用层、表示层、会话层、传输层、网络层、数据链路层、物理层

IP:网络层
TCP/UDP: 传输层
HTTP、RTSP、FTP:应用层协议

35 堆

堆的满足条件：
1.完全二叉树
2. 大根堆：父节点总是大于子节点
3. 小根堆： 子节点总是大于父节点
完全二叉树：一棵深度为k的有n个节点的二叉树，对根中的节点按从上至下，从左到右的顺序进行编号，如果编号为i(1<= i <=n)的节点与满二叉树中编号为i的节点在二叉树中的位置相同，则这棵树成为完全二叉树。

```go
package main

import "fmt"

// 这个函数只能实现最大支按照堆构建
func Heapify(arr []int, n int, i int) {
  // 找出i 的子节点
  //左子节点
  childL := 2*i + 1
  childR := 2*i + 2
  maxIndex := i // 先默认最大的子节点就是i

  if childL < n && arr[childL] > arr[maxIndex] {
    maxIndex = childL
  }
  if childR < n && arr[childR] > arr[maxIndex] {
    maxIndex = childR
  }

  if maxIndex != i {
    arr[maxIndex], arr[i] = arr[i], arr[maxIndex]
    Heapify(arr, n, maxIndex)
  }
}
```

```
//这个函数可以实现任意数列，按照堆构建
func HeapifyBuild(arr []int, n int) {
  lastIndex := n - 1
  parentIndex := (lastIndex - 1) / 2
  for i := parentIndex; i >= 0; i-- {
    Heapify(arr, n, i)
  }

}

// 堆排序
func HeapSort(arr []int) {
  HeapifyBuild(arr, len(arr))

  for i := len(arr) - 1; i >= 0; i-- {
    arr[0], arr[i] = arr[i], arr[0]
    Heapify(arr, i, 0)
  }
}

func main() {
  arr := []int{12, 232, 42, 23, 434, 242,  423,453}
  HeapSort(arr)
  fmt.Println(arr)
}
```

36 二叉树相关

1．满二叉树：一棵二叉树的节点要么是叶子节点，要么它有两个字节点（如果一个二叉树的层数为K，且节点总数是(2^k)-1,则它就是满二叉树。
2．完全二叉树：若设二叉树的深度为K，除第K层外，其它各(1~k-1)的节点数都达到了最大个数，第K层所有的节点都连续集中在最左边(从左到右挂叶子节点)。
3．平衡二叉树：它或者是一棵空树，或它的左子树和右子树的深度只差(平衡因子)的绝对值不超过1，且它的左子树和右字数都是一棵平衡二叉树。

二叉树的相关算法

```
package main

import "fmt"
```

```go
type Node struct {
  Left  *Node
  Right *Node
  Val   int
}

func CreateNode(num int) *Node {
  return &Node{
    Left:  nil,
    Right: nil,
    Val:   num,
  }
}

type Tree struct {
  Root     *Node
  RootList []*Node
}

func (this *Tree) Add(num int) {
  node := CreateNode(num)
  if this.Root == nil {
    this.Root = node
    this.RootList = append(this.RootList, this.Root)
    return
  }
  // 广度遍历，找到空缺的子节点

  for i := 0; i < len(this.RootList); i++ {
    tempNode := this.RootList[i]
    //tempList := this.RootList[i:]
    if tempNode.Left == nil {
      tempNode.Left = node
      this.RootList = append(this.RootList, node)
      return
    } else if tempNode.Right == nil {
      tempNode.Right = node
      this.RootList = append(this.RootList, node)
      return
    } else {
      continue
    }

  }

}
```

```go
func (this *Tree) BreadthTraversal() []int {
  var queue []int
  if len(this.RootList) == 0 {
    return queue
  }
  for i := 0; i < len(this.RootList); i++ {
    tempNode := this.RootList[i]
    queue = append(queue, tempNode.Val)
  }
  return queue
}

func (this *Tree) PreOrder(queue []int, node *Node)
[]int {
  if node == nil {
    return queue
  }
  queue = append(queue, node.Val)
  queue = this.PreOrder(queue, node.Left)
  queue = this.PreOrder(queue, node.Right)
  return queue
}

func (this *Tree) SeqOrder(queue []int, node *Node)
[]int {
  if node == nil {
    return queue
  }
  queue = this.SeqOrder(queue, node.Left)
  queue = append(queue, node.Val)
  queue = this.SeqOrder(queue, node.Right)
  return queue
}

func (this *Tree) PostOrder(queue []int, node *Node)
[]int {
  if node == nil {
    return queue
  }
  queue = this.PostOrder(queue, node.Left)
  queue = this.PostOrder(queue, node.Right)
  queue = append(queue, node.Val)
  return queue
}
```

37 快排

```
package main

func QuickSort(arr []int, start,end int){
  i, j := start, end
  midIndex := (start + end) / 2

  for i <= j{
    for arr[i] < arr[midIndex]{
      i++
    }
    for arr[j] > arr[midIndex]{
      j--
    }
    if i < j{
      arr[i],arr[j] = arr[j],arr[i]
      i++
      j--
    }
  }
  if start < j{
    QuickSort(arr, start, j)
  }
  if end >i{
    QuickSort(arr, i, end)
  }
}
```

38 二分查找

```
package main

func BinSearch(arr []int, key int) int{
  low, hight := 0, len(arr)-1
  for low <= hight{
    mid := (low + hight) / 2
    if arr[mid] < key{
      low = mid + 1
    } else if arr[mid] > key{
      hight =  mid -1
    }else{
      return mid
    }
  }
  return -1
```

```
}
```

39 数据库怎么优化查询效率

```
1.
```

40. mysql的存储原理

存储过程是一个可编程的函数，它在数据库中创建并保存。它可以有sql语句和一些特殊的控制结构组成。当希望在不同的应用程序或平台上执行相同的函数，或者封装特定的功能是，存储过程是非常有用的。数据库中的存储过程可以看做是对编程中面向对象方法的模拟。可允许控制数据的访问方式。存储过程有以下优点：
1．存储过程能实现较快的执行速度。
2．存储过程允许标准组件是编程。
3．存储过程可以用流控制语句编写，有很强的灵活性，可以完成复杂的判断和复杂的运算。
4． 存储过程可被作为一种安全机制来充分利用。
5．存储过程能够减少网络流量。

41. 事务的特性

1．原子性： 事务中的全部操作在数据库中是不可分割的，要么完全完成，要么均不执行。
2．一致性：几个并行执行的事务，其执行结果必须与按照某一顺序串执行的结果相一致。
3．隔离性：事务的执行不受其他事务的干扰，事务执行的中间结果对其他事务必须是透明的。
4.持久性：对于任意已提交事务，系统必须保证该事务对数据库的改变不被丢失，及时数据库出故障。

42.解决数据库高并发的常见方案：

1.分库分表。
2.数据库索引。
3.redis缓存数据库。
4.读写分离。
5.负载均衡集群：讲大量的并发请求分担到多个处理节点。由于单个处理节点的故障不影响多个服务，负载均衡集群同事也实现了高可用性。

43. Nosql和关系型数据库的区别：

1. SQL数据库存在特定结构的表中；而Nosql则更加灵活和可拓展，存储方式可以使json文档、哈希表或者其他方式。

2.在sql中，必须定义好表和字段结构后才能添加数据，例如定义表的主键(primary key)、索引(index)、触发器(tigger)、存储过程(stored procedure)等，表结构可以在被定义之后更新，但是如果有比较大的数据结构变更的话就会变得比较复杂。在Nosql中，数据可以在任何时候任何地方添加，不需要先定义表。

3.sql中如果需要增加外部关联数据的话，规范化做法是在原表中增加一个外键，关联外部数据表。

4. sql中关联多个表进行查询。

5. sql中允许删除已经被使用的外部数据。 nosql中没有强耦合的观念，可以随时删除任何数据。

6. sql中如果多张表数据需要同批次更新，即使如果其中一张表更新失败的话其他表也不能更新成功，这种长江可以通过事务来控制，可以在所有命令完成后再统一提交事务。而Nosql中没有事务概念，

7. nosql的性能要优于sql。

44. mysql引擎的区别

MYSQL主要有两个引擎：MyISAM 和InnoDB
1. InnoDB支持事务，MYisam不支持。事务是一种高级的处理方式，如在一些列增删改查中只要哪个出错还可以回滚还原，MYISAM不支持。
2.Innodb支持外键，myISAM不支持。
3.Myisam是默认引擎，Innodb需要制定.
4.Innodb不支持fulltext类型的索引。
5.Innodb中不保存表的行数。查找所有数据的时候，需要扫描一整张表。myisam保存有所有行数，不过当查询条件中有where的时候，会扫描整张表。
6.对于自增长的字段,innodb中必须包含只有该字段的索引，myisam表中可以和其他一起建立联合索引；清空整个表时，innodb是一行一行删除，效率非常慢，myisam会重建整个表。
7.Innodb支持行锁。(某些情况下也会锁整张表。)

45.数据库索引

数据库索引，是数据库管理系统中的一个排序的数据结构，以协助快速查询、更新数据库表中数据。索引的实现通常使用B-TREE，B-Tree索引加速了数据访问，因为存储引擎不会再去扫描整张表得到需要的数据；相反，它从根节点开始，根节点保存了子节点的指针，存储引擎会根据织织快速寻找数据。

46. 数据库怎么优化查询效率

1．储存引擎的选择： 如果数据表需要处理事务，应该考虑使用INNODB，因为它完全符合acid(一致性、隔离性、原子性、持久性)特性，如果不需要事务处理，使用默认存储引擎myisam是比较明智的。
2．分表分库，主从。
3． 对查询进行优化，要尽量避免全表扫描，首先应考虑在where及order by涉及的列上建立索引。
4．应尽量避免在where子句中对字段进行null值判断，否则将导致引擎放起使用索引而进行全表扫描。应尽量避免在where子句只能给使用!=或<>操作符，否则将引擎放起引用索引，而进行全表扫描。
5.尽量避免在where子句中使用or来连接条件，如果一个字段有索引，一个字段没有索引，将导致引擎放起使用索引而进行全表扫描。
6.Update语句，如果只更改1、2个字段，不要update全部字段，否则频繁调用将引起明显的性能消耗。
7．对于多张大数据量的表join，要先分页在join，否则逻辑读会很高，性能很差

47. 存储过程和函数的区别

相同点： 存储过程和函数都是为了可重复的执行操作数据库的sql语句和集合。
・存储过程和函数都是一次编译，就会被缓存起来，下次使用就直接命中已经编译好的sql语句，不需要重复使用，减少网络交互，减少网络访问量。
不同点： 标识符不同，函数的标识符是function，存储过程的是proceduce
・函数中有返回值，且必须有返回值，而过程没有返回值，但是可以通过设置参数类型来实现多个参数或者返回值。
・存储函数使用select调用，存储过程需要使用call调用。
・select语句可以在存储过程中调用，但是除了select..into之外的select语句都不能在函数中使用。
・用过in out参数，过程相关函数更加灵活，可以返回多个结果

48. 如何优化数据库以及提高数据库的性能?

1．对查询语句的优化
– 用程序中，保证在实现功能的基础上，尽量减少对数据库访问的次数。(可以用redis进行缓存处理。)

– 能够分开的操作尽量分开处理，提高每次的响应速度；在数据窗口使用sql时，尽量吧使用的索引放在选择的首列；算法的结构尽量简单。

– 在查询时，不要过多的使用通配符“*”

– 在可能的情况下尽量限制结果集的行数。

– 不要在应用中使用数据库的游标，游标是非常有用的工具，但比常规的，面向集的sql语句需要更大的开销，按照特定顺序提取数据的查找。

2. 避免使用不兼容的数据类型。(比如查询的过程中，传的参数类型要与定义的数据类型一致)
3.避免在where子句中对字段进行函数或表达式操作。(若进行函数或表达式操作，将导致引擎放起索引而进行全表扫描。)
4.避免使用!= 或<>,IS NULL或者is not null、in、not in等这种操作符。
5. 尽量使用数字类型字段。
6. 合理使用exists，not exists语句。
7. 尽量避免在索引过的字符串数据中，使用非大头字母索引。
8.分利用链接条件。
9.消除对大型表行数据的顺序存取。
10.避免困难的正则表达式。
11.使用视图加速查询。
12.能够使用between的就不要使用in
13.distinct的就不用group by
14.部分利用索引。
15.能用union all就不要用union。
16.不要写一些不做任何事的查询。
17.尽量不要用select into语句。
18.必要时强制查询优化器使用某个索引。
19. 对update语句的建议：
- 尽量不要修改主键的字段。
- 当修改varchar型字段时，尽量使用相同长度内容的值替代。
- 尽量最小化对于含有update触发器表的upadte操作。
- 避免update将要赋值到其他数据库的列。
- 避免update兼有多索引的列。
- 避免update在where子句条件中的列。

49.Redis

1．默认端口： 6379
2．默认过期时间：可以说永不过期，一般情况下，当配置中开启了超出最大内存限制就写磁盘的话，那么没有设置过期时间的key可能会被写到磁盘上。加入没设置，那么redis将使用lru机制，将内存中的老数据删除，并写入新数据。
3．value最大容纳的数据长度是：512M
4．reids一个实例下有16个。

50. Redis中list底层实现有哪几种？ 有什么区别?

1．列表队形的编码可以使ziplist或者linkedlist
ziplist是一种压缩链表，它的好处是更能节省内存空间，因为它所存储的内容都是在连续的内存区域当中。当列表对象元素不大，每个元素也不大的首，就采用ziplist存储。但当数据量过大时，ziplist就不是那么好用了。因为为了保证他存储内容在内存中的连续性，插入的复杂度是O(N)，即每次插入都会重新进行realloc。

51.mysql事务相关

事务个特点:
・原子性(Atomicity)： 事务中的所有操作作为一个整体，像原子一样不可分割。(事务不可分割。)
・一致性(Consistency)： 事务的执行结果必须使用数据库从一个一致性状态到另一个一致性状态。一致性状态是指：1．系统的状态满足数据的完整性约束 2．系统的状态反应数据库本应描述的显示世界的真是状态。
・隔离性(isolation)：并发执行的事务不会相互影响，其对数据库的影响和他们穿行执行的一样。(并发执行效果跟串行效果相同)
・持久性(Durabilty)： 事务一旦提交，其对数据库的更新就是持久的。任务事务或系统故障都不会丢失数据。(数据持久化)

在事务的四个特点中，一致性是事务的根本需求。
并发控制技术：为了保证事务的个隔离性和一致性。
日志恢复技术：为了保证事务的一致性、原子性、和持久性。

事务的原子性是通过undo log实现的。
事务的持久性是通过redo log实现的。
事务的隔离性是通过(读写锁+MVCC)来实现的。
事务的一致性是通过原子性、持久性，隔离性来实现的。

undo log是逻辑日志，用来数据回滚。(用来保证事务的原子性。)

redo log记录的是新数据的备份。当系统崩溃是，虽然数据没有持久化，但是 redo log已经持久化。 系统可以根据redo log的内容将所有数据恢复到最新的状态。(用来保证数据持久性。)

sql标准为事务定义了不同的隔离级别，从低到高依次是：

- 读未提交(READ UNCOMMITTED)：对事务处理的读取没有任何限制，不推荐。
- 读已提交(READ COMMITTED)
- 可重复读(pepetable read)
- 串行化(serializable)

事务的隔离级别	脏读	不可重复读	幻读
读未提交(READ UNCOMMITED)	*	*	*
读已提交(READ COMMITTED)		*	*
可重复读(pepetable read)			*
串行化(serializable)			

隔离级别相关：

set session transaction isolation level ******(级别参数)

脏读：一个事务中更新了尚未提交的数据，在另一个事务中直接等读出来。

不可重复读：在一个事务中，查询表出现了两种不同结果。(在开启一个事务的同时，在另一个事务对数据进行了修改，导致第一个事务读数据异常。)

幻读：在一个事务中插入数据，在另一个事务中插入主键相同的数据，会出现异常。

事务默认的是：可重复读。

锁的粒度：1. 记录 2.表 3.数据库 (粒度越小，效率越高)
mysql中的锁：
1.共享锁(shared Locks)(简称S锁，属于行锁)
2.排他锁(exclusive locks)(简称x锁，属于行锁)
3.意向共享过(intention shared Locks)(简称is锁，属于表锁)
4.意向排他锁(in特人体on exclusIve locks)(简称ix锁，属于表锁)
5.自增锁(用于处理主键自增的。)

共享锁：就是读共享，写不共享。
排他锁：排他锁不能与其他锁并存，如果一个事务获取了一个行的排他锁，其他事务就不能在获取该行的锁。只有当前获取了拍他所的事务可以对数据读取和修改。

52. TCP和UDP的区别以及优缺点

Udp 是面向无连接的通讯协议；tcp是数据包括目的的端口号和元端口号信息。
优点：UDP速度快、操作简单、要求系统资源较少，由于通讯不需要连接，可以实现广播发送。

缺点：Udp传输数据钱并不与对方简历连接，对接收到的数据也不发送确认信号，发送端不知道数据是否会正确接收，也不重复发送，不可靠。

Tcp 是面向连接的通讯协议，通过三次握手建立链接，通讯完成时四次挥手。
优点：tcp在数据传递是，有确认，传输，重传，阻塞等控制机制，能保证数据的正确性，较为可靠。
缺点：TCP相对与UDP速度慢，要求的资源比较多。

53. mysql建立索引无效

1. 索引的字段值中不能包含null值
2. 索引中的值一般情况下唯一。
3. 索引的值不用用长文本。

54. jwt相关

jwt: json web token 用于前后端校验

- jwt实现的过程:
  - 用户提交用户名和密码给服务端，如果登陆成功，使用jwt创建token， 并返回用户。

"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c"

  - jwt生成的token是有三段字符串凭借组成，用.链接。
    第一段字符串(HEADER)：内部包含算法和token类型，json转换成字符串然后进行base64url进行编码。

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

    第二段字符串(payload)：开发者自定义。json转化成字符串，然后做 base64url编码

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

第三段字符串(SIGNATURE)：分两步：

```
//第一步：将第一二段的密文用.拼接
base64UrlEncode(header) + "." +
base64UrlEncode(payload)
// 第二步：对前两部分的密文进行第一段类型中“alg”对应的加密类型进行加密，然后再加盐。
// 第三步：对加密后的密文再进行base64urlencode
```

如何校验加密好的token

- 获取token
- 第一步：对获取到的token用.进行分割。
- 第二步：对第二段进行base64urlDecode解密，获取payload信息。
- 第三步：在获取payload中，获取超时时间，验证时间是否超时。(检验是否超时)
- 第四步： 重复加密的第三步，获取密文，然后和token中的密文进行比较(检验密文是否正确。)

56. RabbitMQ

MQ： message queue即消息队列。基于AMQP(advanced message queue高级消息队列协议)，协议实现的消息对垒，它是一种应用程序之间的通信方法，消息对列 在分布式系统开发中应用非常广泛。(简单讲就是进程间通信)

开发中消息对列通常有如下应用场景：

1. 任务异步处理。

   将不需要同步处理的并且耗时长的操作有消息队列通知消息接收方进行异步处理。提高了应用程序的响应时间。

2. 应用程序解耦合

   MQ相当于一个中介，生产方通过MQ与消息方交互，它将应用程序进行解耦合。

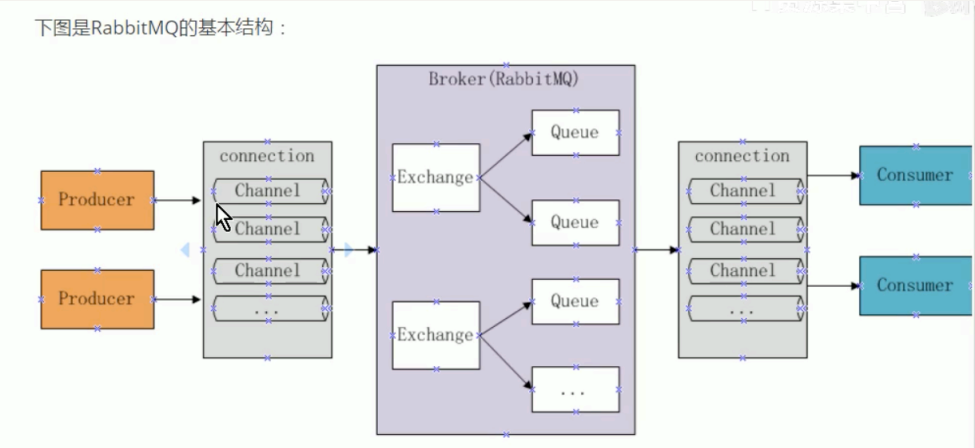
消息队列中间件是分布式系统中的重要组件 其解决的问题为：

1. 解耦 2. 异步消息 3.流量削锋(在请求服务器之前添加消息队列，防止服务器器瞬间过载。并不是用来解决高并发的问题.)

为什么使用rabbitMQ?

- 1. 使用简单，功能强大。
- 2. 基于AMQP协议。
- 3.社区活跃，文档完善。
- 4.高并发性能好
- 5.Spring Boot默认已集成RabbitMQ。

组件结构图：



组件部分说明：

- Broker：消息队列服务进程，此进程包含两个部分：Exchange 和 Queue。
- Exchange：消息队列交换机，按一定的规则将消息路由转发到某个队列，对消息进行过滤。
- Queue：消息队列，存储消息的队列，消息到达队列并转发给指定的消费方。
- Producer：消费生产者，即生产方客户端，生产方客户端将消费发送到MQ.
- Consumer： 消息消费者，即消费方客户端，接收MQ转发的消息.

消息发布接收流程：

1. 发送消息：
    - 生产者和Broker建立tcp链接。
    - 生产者和Broker建立通道。
    - 生产者通过消息通道消息发送Broker，由Exchange将消息进行转发。
    - Exchange将消息转发到指定的Queue。

2. 接收消息：
    - 消费者和Broker建立TCP连接。
    - 消费者和Broker建立通道。
    - 消费者监听指定的Queue。
    - 当有消息到达Queue时Broker默认将消息推送给消费者。
    - 消费者接收到消息。

57. RabbitMQ工作模型

· 57.1 相关参数

- 应答参数 ： 当消费者执行结束后，手动取消队列中的数据。
- 持久化参数 ： 将队列中的数据持久化，防止数据丢失。
- 分发参数 ： 消费者中竞争获取数据(默认的是轮询)

· 57.2 简单模式

就是普通的生产者消费者模式。

· 57.3 交换机模式

57.3.1 交换机之发布订阅
  1. 生产者和消息队列建立连接
  2. 生产者将数据存入到交换机中
  3. 消费者与消息队列建立连接
  4. 消费者生成自己独立的队列
  5. 队列分别和对应的交换机绑定(bind)
  6. 交换机将消息发布到每个与之绑定的队列
  7. 消费者从队列中取数据。
57.3.2 交换机之关键字
  在以上的基础上，每个绑定会关联关键字

57.3.2 交换机之通配符
  在关键字的基础上，将关键字替换成通配符：# 表示多个字符串， * 仅匹配一个字符串。

57. celery

- celery是一个简单、灵活且可靠的，处理大量消息的分布式系统。专注于实时处理的异步任务队列，同事也支持任务调度。

6 、http状态码

分类	分类描述
1**	信息，服务器收到请求，需要请求者继续执行操作
2**	成功，操作被成功接收并处理
3**	重定向，需要进一步的操作以完成请求
4**	客户端错误，请求包含语法错误或无法完成请求
5**	服务器错误，服务器在处理请求的过程中发生了错误

状态码	状态码英文名称	中文描述
100	Continue	继续。客户端应继续其请求
101	Switching Protocols	切换协议。服务器根据客户端的请求切换协议。只能切换到更高级的协议，例如，切换到HTTP的新版本协议
200	OK	请求成功。一般用于GET与POST请求
201	Created	已创建。成功请求并创建了新的资源
202	Accepted	已接受。已经接受请求，但未处理完成
203	Non-Authoritative Information	非授权信息。请求成功。但返回的meta信息不在原始的服务器，而是一个副本
204	No Content	无内容。服务器成功处理，但未返回内容。在未更新网页的情况下，可确保浏览器继续显示当前文档
205	Reset Content	重置内容。服务器处理成功，用户终端（例如：浏览器）应重置文档视图。可通过此返回码清除浏览器的表单域
206	Partial Content	部分内容。服务器成功处理了部分GET请求
300	Multiple Choices	多种选择。请求的资源可包括多个位置，相应可返回一个资源特征与地址的列表用于用户终端（例如：浏览器）选择
301	Moved Permanently	永久移动。请求的资源已被永久的移动到新URI，返回信息会包括新的URI，浏览器会自动定向到新URI。今后任何新的请求都应使用新的URI代替
302	Found	临时移动。与301类似。但资源只是临时被移动。客户端应继续使用原有URI

303	See Other	查看其它地址。与301类似。使用GET和POST请求查看
304	Not Modified	未修改。所请求的资源未修改，服务器返回此状态码时，不会返回任何资源。客户端通常会缓存访问过的资源，通过提供一个头信息指出客户端希望只返回在指定日期之后修改的资源
305	Use Proxy	使用代理。所请求的资源必须通过代理访问
306	Unused	已经被废弃的HTTP状态码
307	Temporary Redirect	临时重定向。与302类似。使用GET请求重定向
400	Bad Request	客户端请求的语法错误，服务器无法理解
401	Unauthorized	请求要求用户的身份认证
402	Payment Required	保留，将来使用
403	Forbidden	服务器理解请求客户端的请求，但是拒绝执行此请求
404	Not Found	服务器无法根据客户端的请求找到资源（网页）。通过此代码，网站设计人员可设置"您所请求的资源无法找到"的个性页面
405	Method Not Allowed	客户端请求中的方法被禁止
406	Not Acceptable	服务器无法根据客户端请求的内容特性完成请求
407	Proxy Authentication Required	请求要求代理的身份认证，与401类似，但请求者应当使用代理进行授权
408	Request Time-out	服务器等待客户端发送的请求时间过长，超时
409	Conflict	服务器完成客户端的 PUT 请求时可能返回此代码，服务器处理请求时发生了冲突
410	Gone	客户端请求的资源已经不存在。410不同于404，如果资源以前有现在被永久删除了可使用410代码，网站设计人员可通过301代码指定资源的新位置
411	Length Required	服务器无法处理客户端发送的不带Content-Length的请求信息
412	Precondition Failed	客户端请求信息的先决条件错误
413	Request Entity Too Large	由于请求的实体过大，服务器无法处理，因此拒绝请求。为防止客户端的连续请求，服务器可能会关闭连接。如果只是服务器暂时无法处理，则会包含一个Retry-After的响应信息
414	Request-URI Too Large	请求的URI过长（URI通常为网址），服务器无法处理
415	Unsupported	服务器无法处理请求附带的媒体格式

	Media Type	
416	Requested range not satisfiable	客户端请求的范围无效
417	Expectation Failed	服务器无法满足Expect的请求头信息

500	Internal Server Error	服务器内部错误，无法完成请求
501	Not Implemented	服务器不支持请求的功能，无法完成请求
502	Bad Gateway	作为网关或者代理工作的服务器尝试执行请求时，从远程服务器接收到了一个无效的响应
503	Service Unavailable	由于超载或系统维护，服务器暂时的无法处理客户端的请求。延时的长度可包含在服务器的Retry-After头信息中
504	Gateway Time-out	充当网关或代理的服务器，未及时从远端服务器获取请求
505	HTTP Version not supported	服务器不支持请求的HTTP协议的版本，无法完成处理