# 面试宝典

## GO语言

### Map

go中map

### Slice

#### slice原理

#### Slice扩容原则

#### nil切片和空切片一样吗

nil切片和空切片指向的地址不一样。nil空切片引用数组指针地址为0（无指向任何实际地址）

空切片的引用数组指针地址是有的，且固定为一个值

#### 拷贝大切片一定比小切片代价大吗？

一样大，切片的底层数据类型为一个指向数组的指针，一个切片的长度，一个切片的容量，拷贝一个切片只会拷贝这三个值，所以无论切片大小代价都一样。

### String

#### 字符串转成byte数组，会发⽣内存拷⻉吗？

字符串转成切片，会产生拷贝。严格来说，只要是发生类型强转都会发生内存拷贝。

#### 翻转含有中⽂、数字、英⽂字⺟的字符串

将字符串转为rune的切片，再进行翻转

### Channel

#### ⽆缓冲 Chan 的发送和接收是否同步?

无缓冲的chan发送和接受需要同步，而有缓冲的chan无需发送和接受同步。

1. channel⽆缓冲时，发送阻塞直到数据被接收，接收阻塞直到读到数据。
2. channel有缓冲时，当缓冲满时发送阻塞，当缓冲空时接收阻塞。

### Mutex

### Struct

#### go struct能不能⽐较

1. 相同struct类型的可以⽐较
2. 不同struct类型的不可以⽐较,编译都不过，类型不匹配

### Goroutine

### Golang 中常⽤的并发模型

Golang 中常⽤的并发模型有三种:

1. 通过channel实现并发控制

⽆缓冲的通道指的是通道的⼤⼩为0，也就是说，这种类型的通道在接收前没有能⼒保存任何值，它要求发送goroutine 和接收 goroutine 同时准备好，才可以完成发送和接收操作。

1. 通过sync中的withGroup实现并发控制

Goroutine是异步执行的，为了防止结束main函数时结束goroutine，就需要等待，就会用到sync.withGroup, withGroup有三个函数：

1. Add() 增加或减少goroutine的数量
2. Done() goroutine执行完成后数量减一，相当于Add(-1)
3. Wait() 阻塞主线程，直到withGroup里的值减为0
4. 使用 Context上下文实现并发控制

### new和make的区别

new初始化一个指向类型的指针，传给new的是一个类型，不是一个值。返回值是指向这个新分配的零值的指针。

Make是初始化slice，map和chan并返回引用。

### 内存逃逸

Go语言编译器会自动决定把变量放在栈上还是堆上，编译器会做逃逸分析，当发现变量的作用于没有逃出函数范围，就可以放在栈上，反之放在堆上。

### GMP模型

G goroutine 协程

M thread 线程

P processor 处理器

1. Goroutine特点：

* 占用内存更小（几kb）
* 调度更灵活(runtime调度)

1. 被废弃的goroutine调度器

M想要执行、放回G都必须访问全局G队列，并且M有多个，即多线程访问同一资源需要加锁进行保证互斥/同步，所以全局G队列是有互斥锁进行保护的。

老调度器有几个缺点：

* 统创建、销毁、调度G都需要每个M获取锁，这就形成了**激烈的锁竞争**。
* M转移G会造成**延迟和额外的系统负载**。比如当G中包含创建新协程的时候，M创建了G’，为了继续执行G，需要把G’交给M’执行，也造成了**很差的局部性**，因为G’和G是相关的，最好放在M上执行，而不是其他M'。
* 系统调用(CPU在M之间的切换)导致频繁的线程阻塞和取消阻塞操作增加了系开销。

1. GMP模型的设计思想
2. GMP模型
3. 全局队列 存放等待运行的G
4. P的本都队列 同全局队列类似，存放的也是等待运行的G，存的数量有限，不超过256个。新建G'时，G'优先加入到P的本地队列，如果队列满了，则会把本地队列中一半的G移动到全局队列。
5. 所有的P都在程序启动时创建，并保存在数组中，最多有GOMAXPROCS(可配置)个。
6. M 线程想运行任务就得获取P，从P的本地队列获取G，P队列为空时，M也会尝试从全局队列**拿**一批G放到P的本地队列，或从其他P的本地队列**偷**一半放到自己P的本地队列。M运行G，G执行之后，M会从P获取下一个G，不断重复下去。
7. 调度器设计策略
8. 复用线程：避免频繁的创建、销毁线程，而是对线程的复用。
   1. work stealing机制 ：当本线程无可运行的G时，尝试从其他线程绑定的P偷取G，而不是销毁线程。
   2. hand off机制 ：当本线程因为G进行系统调用阻塞时，线程释放绑定的P，把P转移给其他空闲的线程执行。
9. 利用并行： GOMAXPROCS设置P的数量，最多有GOMAXPROCS个线程分布在多个CPU上同时运行。GOMAXPROCS也限制了并发的程度，比如GOMAXPROCS = 核数/2，则最多利用了一半的CPU核进行并行。
10. 抢占：在coroutine中要等待一个协程主动让出CPU才执行下一个协程，在Go中，一个goroutine最多占用CPU 10ms，防止其他goroutine被饿死。
11. 全局G队列：在新的调度器中依然有全局G队列，但功能已经被弱化了，当M执行work stealing从其他P偷不到G时，它可以从全局G队列获取G。
12. go func()的调度流程
13. 我们通过 go func()来创建一个goroutine；
14. 有两个存储G的队列，一个是局部调度器P的本地队列、一个是全局G队列。新创建的G会先保存在P的本地队列中，如果P的本地队列已经满了就会保存在全局的队列中；
15. G只能运行在M中，一个M必须持有一个P，M与P是1：1的关系。M会从P的本地队列弹出一个可执行状态的G来执行，如果P的本地队列为空，就会想其他的MP组合偷取一个可执行的G来执行；
16. 一个M调度G执行的过程是一个循环机制
17. 当M执行某一个G时候如果发生了syscall或则其余阻塞操作，M会阻塞，如果当前有一些G在执行，runtime会把这个线程M从P中摘除(detach)，然后再创建一个新的操作系统的线程(如果有空闲的线程可用就复用空闲线程)来服务于这个P。
18. 当M系统调用结束时候，这个G会尝试获取一个空闲的P执行，并放入到这个P的本地队列。如果获取不到P，那么这个线程M变成休眠状态， 加入到空闲线程中，然后这个G会被放入全局队列中。
19. 调度器的生命周期

### GC垃圾回收

Golang中的垃圾回收主要应用三色标记法，GC过程和其他用户goroutine可并发运行，但需要一定时间的**STW(stop the world)**，STW的过程中，CPU不执行用户代码，全部用于垃圾回收，这个过程的影响很大，Golang进行了多次的迭代优化来解决这个问题。

1. Go1.3版本之前使用标记-清除算法

主要有两两步，即标记清除：第一步，暂停程序业务逻辑，分类出可达不可达对象，做出标记；第二步，程序找出它所有可达的对象，并做上标记；第三步，开始清除未标记的对象；第四步， 停止暂停，让程序继续跑。然后循环重复这个过程，直到process程序生命周期结束

1. Go1.5 三色并发标记法

三色标记法即通过三个颜色（白色，灰色，黑色）表示不用阶段的对象。第一步，将新创建的对象标记为白色；第二步，每次GC开始。从根节点开始遍历所有节点，把遍历到的对象变成灰色（放入灰色集合）；第三步，遍历灰色集合，将灰色对象引用的对象从白色集合放到灰色集合，之后将灰色对象放入黑色集合；第四步，重复第三步直到灰色集合无对象；第五步，回收白色的对象。

缺点:

条件1: 一个白色对象被黑色对象引用(白色被挂在黑色下)

条件2: 灰色对象与它之间的可达关系的白色对象遭到破坏(灰色同时丢了该白色)

当满足以上两个条件时会造成对象丢失。

1. 屏障机制
   1. 强弱三色不变式
      * 强三色不变式，即不存在黑色对象直接引用白色对象
      * 弱三色不变式，即黑色对象引用的白色对象都处于灰色对象的保护下。
   2. 插入屏障 满足强三色不变式（不存在黑色对象引用白色对象的情况了， 因为白色会强制变成灰色）
   3. 删除屏障 满足弱三色不变式（保护灰色对象到白色对象的路径不会断）
2. Go1.8 混合写屏障机制

插入写屏障和删除写屏障的短板：

插入写屏障：结束时需要STW来重新扫描栈，标记栈上引用的白色对象的存活；

删除写屏障：回收精度低，GC开始时STW扫描堆栈来记录初始快照，这个过程会保护开始时刻的所有存活对象。

1. 混合写屏障原则
2. GC开始将栈上的对象全部扫描并标记为黑色(之后不再进行第二次重复扫描，无需STW)，
3. GC期间，任何在栈上创建的新对象，均为黑色。
4. 被删除的对象标记为灰色。
5. 被添加的对象标记为灰色。

Golang中的混合写屏障满足弱三色不变式，结合了删除写屏障和插入写屏障的优点，只需要在开始时并发扫描各个goroutine的栈，使其变黑并一直保持，这个过程不需要STW，而标记结束后，因为栈在扫描后始终是黑色的，也无需再进行re-scan操作了，减少了STW的时间。

## 微服务

### 对微服务的了解

### 微服务的优势

### 微服务的特点

### 微服务缺点

## 计算机网络

### 进程，线程，协程的区别

进程，进程是具有独立功能的的一段程序，是系统进行资源分配和调度的独立单位，

### 并发并行对的区别

### 什么是死锁，产生死锁的条件

### TCP

#### Tcp三次握手，为什么三次握手

#### Tcp四次握手，为什么有TIME\_WAIT机制，为什么四次握手，出现大量TIME\_WAIT怎么解决

### HTTP

### RPC

## Redis

### 什么是Redis

### Redis的数据类型

## MongoDB

## Mysql

### 数据库三大范式

1. 第一范式

数据库字段原子性

1. 第二范式

首先满足第一范式的基础之上，然后满足：

1. 一张表必须有一个主键
2. 非主键列必须完全依赖于主键，而不能只依赖于主键的一部分
3. 第三范式

首先满足第二范式，另外非逐渐必须直接依赖于主键，不能存在传递依赖。

### 事务

#### 四大特性

1. 原子性 指的是事务包含的所有操作要么全部成功，要么全部失败。
2. 一致性 指的是事务执行前和执行后必须处于一致性。
3. 隔离性 跟隔离级别有关
4. 持久性 指的是事务一旦被提交了，那么对数据库的影响是永久性的。

#### 隔离级别

1. 基本概念
2. 脏读
3. 不可重复读
4. 幻读
5. 四种隔离级别
   1. **Serializable**  串行化
   2. **Repeatable read**  可重复读
   3. **Read committed** 读已提交
   4. **Read uncommitted** 读未提交
6. 查看隔离级别

select @@transaction\_isolation;

1. 设置隔离级别

set session transaction isolation level read uncommitted;

### 索引

#### 什么是索引

#### 索引的优缺点

#### 索引的作用

#### 什么情况下建索引

#### 什么情况下不建索引

#### 索引的数据结构

1. B+索引
2. Hash索引

#### B+索引和hash索引的区别

#### 为什么B+树比B树更加适合数据库索引

#### 索引分类

1. 主键索引
2. 唯一索引
3. 组合索引
4. 全文索引
5. 普通索引
6. https://www.nowcoder.com/discuss/479357164675444736?sourceSSR=search

## Linux常用命令

1. Linux常用命令
   1. pwd
   2. cd
   3. ls
   4. cat
   5. cp
   6. mv
   7. mkdir
   8. rmdir
   9. touch
   10. locate
   11. find find -name xxx
   12. wget
   13. tar
2. Docker常用命令
3. docker ps -a
4. docker images
5. docker run –name=xxx --it -port xxx:xxx image -v xxx xxx名称 /bin/bash
6. docker run -name=xxx -di xxx
7. docker exec -it 容器名称 /bin/bash
8. docker start xxx
9. docker stop xxx
10. docker restart xxx
11. docker cp xxx xxx
12. docker rm 容器名/容器id
13. docker rmi 镜像名/镜像id
14. docker pull 镜像名
15. docker search 镜像

## 数据结构与算法

1. 数组
2. 栈和队列
   1. 栈实现队列
   2. 队列实现栈
3. 链表
   1. 移除链表元素
   2. 反转链表
   3. 两两交换链表元素
   4. 删除链表的倒数第n个元素
   5. 链表相交
4. 二叉树
   1. 递归遍历
   2. 迭代遍历
   3. 层序遍历
   4. 反转二叉树
   5. 对称二叉树
   6. 二叉树的最大深度
   7. 二叉树的最小深度
   8. 完全二叉树的节点个数
   9. 平衡二叉树
   10. 二叉树的所有路径
   11. 左叶子之和
   12. 找树的左下角的值
   13. 路径总和
   14. 最大二叉树
   15. 合并二叉树
   16. 二叉搜索树中的搜索
   17. 验证二叉搜索树
   18. 二叉搜索树的最小绝对差
   19. 二叉搜索树中的众数
   20. 二叉树的最近公共祖先
   21. 二叉搜索树的最近公共祖先
   22. 二叉搜索树的插入操作
   23. 删除二叉搜索中的节点
   24. 修剪二叉搜索树
5. 回溯
   1. 组合问题
   2. 子集问题
   3. 全排列
6. 动态规划
   1. 使用最小花费爬楼梯
   2. 斐波那契数列
   3. 不同路径
   4. 不同路径II
   5. 整数拆分
   6. 01背包问题
      1. 分割等和子集
      2. 最后一块石头的重量
      3. 目标和
      4. 一和零
   7. 完全背包问题
   8. 零钱兑换
   9. 完全平方数
   10. 单词拆分
   11. 多重背包问题
   12. 打家劫舍
   13. 打家劫舍II
7. 贪心算法
   1. 分发饼干
   2. 摆动序列
   3. 最大子序列和
   4. 买股票最佳时机
   5. 买股票最佳时机II
   6. 跳跃游戏
   7. 跳跃游戏II
   8. K次取反后的最大数组和
   9. 加油站
   10. 分发糖果