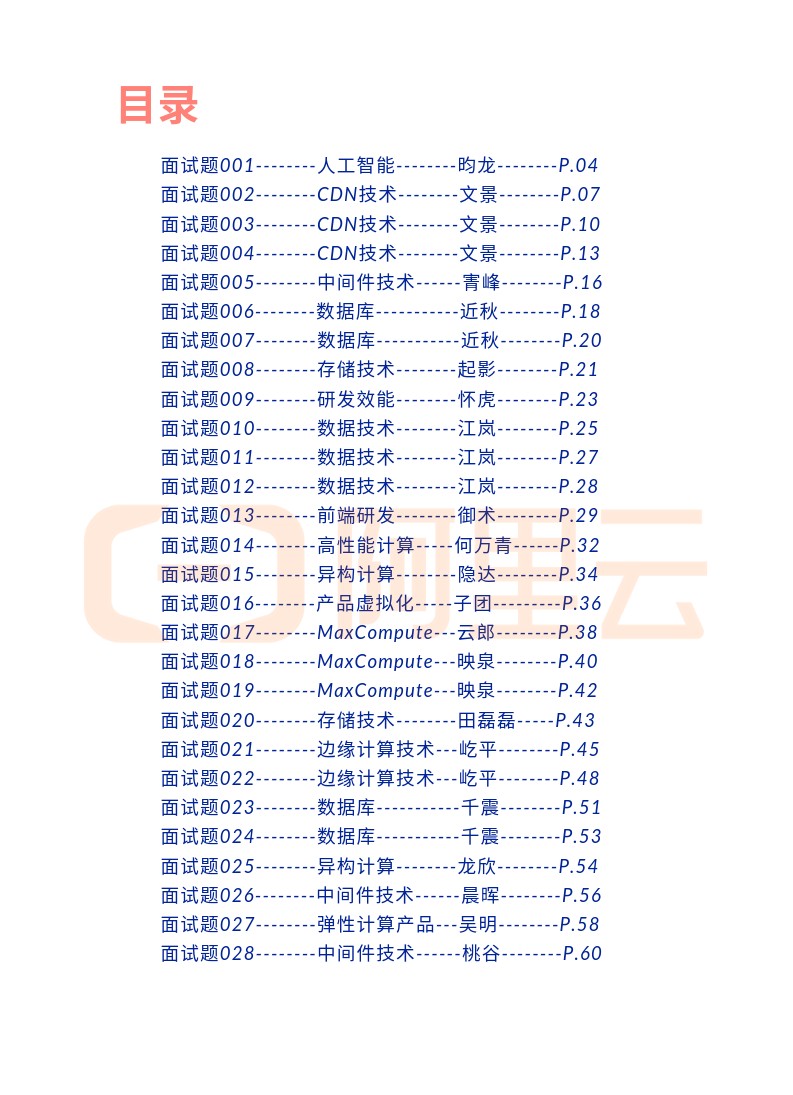
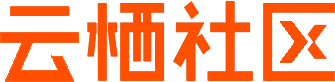
**3**



### ／ 面试题 001 如何实现一个高效的单向链表逆序输出？

#### 参考答案

**——阿里巴巴出题专家：昀龙／阿里云弹性人工智能负责人**

下面是其中一种写法，也可以有不同的写法，比如递归等。供参考。

typedefstructnode

{

intdata; structnode\*next;

node(intd):data(d),next(NULL){}

}node; voidreverse(node\*head)

{

if(NULL==head||NULL==head->next)

{

return;

}

node\*prev=NULL; node\*pcur=head->next; node\*next; while(pcur!=NULL)

{

if(pcur->next==NULL)

{

pcur->next=prev; break;

}

next=pcur->next; pcur->next=prev; prev=pcur; pcur=next;

}

head->next=pcur; node\*tmp=head->next; while(tmp!=NULL)

{

cout<<tmp->data<<"\t"; tmp=tmp->next;

}

}

### ／ 面试题 002 已知sqrt (2)约等于 1.414，要求不用数学库，求sqrt (2)精确到小数点后 10 位。

**——阿里巴巴出题专家：文景／阿里云 CDN 资深技术专家**

#### 考察点

1. 基础算法的灵活应用能力（二分法学过数据结构的同学都知道， 但不一定往这个方向考虑；如果学过数值计算的同学，应该还要能想到牛顿迭代法并解释清楚）
2. 退出条件设计

#### 参考答案

1. 已知sqrt(2)约等于 1.414，那么就可以在(1.4, 1.5)区间做二分查找，如：
   1. high=>1.5
   2. low=>1.4
   3. mid => (high+low)/2=1.45

d) 1.45\*1.45>2 ? high=>1.45 : low => 1.45

e) 循环到 c)

1. 退出条件
   1. 前后两次的差值的绝对值<=0.0000000001, 则可退出

##### 代码示例：

**const double EPSINON = 0.0000000001; double sqrt2( )**

**{**

**double low = 1.4, high = 1.5; double mid = (low + high) / 2;**

**while (high – low > EPSINON)**

**{**

**{**

**}**

**else**

**}**

**if (mid\*mid < 2) high = mid;**

**{**

**low = mid;**

**}**

**mid = (high + low) / 2;**

**return mid;**

**}**

### ／ 面试题 003 给定一个二叉搜索树(BST)，找到树中第K 小的节点。

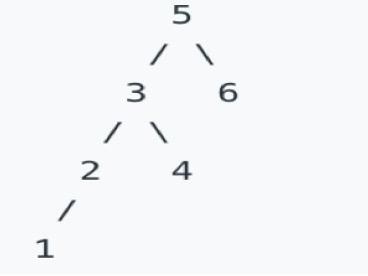
**——阿里巴巴出题专家：文景／阿里云 CDN 资深技术专家**

#### 考察点

1. 基础数据结构的理解和编码能力
2. 递归使用

##### 示例：

如下图，输入 K=3， 输出节点值 3



说明：保证输入的K 满足 1<=K<=(节点数目）

#### 参考答案

树相关的题目，第一眼就想到递归求解，左右子树分别遍历。联想到二叉搜索树的性质，root 大于左子树，小于右子树，如果左子树的节点数目等于 K-1，那么 root 就是结果，否则如果左子树节点数目小于 K-1，那么结果必然在右子树，否则就在左子树。因此在搜索的时候同时返回节点数目，跟 K 做对比，就能得出结果了。

◼ **代码示例：**

/\*\*

* Definition for a binary tree node.
* public class TreeNode {
* int val;
* TreeNode left;
* TreeNode right;
* TreeNode(int x) { val = x; }

\* }

\*/class Solution {

private class ResultType {

// 是否找到

boolean found;

// 节点数目

int val;

ResultType(boolean found, int val) { this.found = found;

this.val = val;

}

}

public int kthSmallest(TreeNode root, int k) { return kthSmallestHelper(root, k).val;

}

private ResultType kthSmallestHelper(TreeNode root, int k) {

if (root == null) {

return new ResultType(false, 0);

}

ResultType left = kthSmallestHelper(root.left, k);

// 左子树找到，直接返回

if (left.found) {

return new ResultType(true, left.val);

}

// 左子树的节点数目 = K-1，结果为 root 的值

if (k - left.val == 1) {

return new ResultType(true, root.val);

}

// 右子树寻找

ResultType right = kthSmallestHelper(root.right, k - left.val - 1); if (right.found) {

return new ResultType(true, right.val);

}

// 没找到，返回节点总数

return new ResultType(false, left.val + 1 + right.val);

}

}

复杂度分析：

时间复杂度： O(N），节点最多遍历一遍

空间复杂度：O(1），（如果算上递归深度可以当做O(logN)）

### ／ 面试题 004 LRU 缓存机制。

**——阿里巴巴出题专家：文景／阿里云 CDN 资深技术专家**设计和实现一个 LRU（最近最少使用）缓存数据结构，使它应该支持一下操作：get 和put。

get(key) - 如果key 存在于缓存中，则获取 key 的value（总是正

数），否则返回 -1。

put(key,value) - 如果 key 不存在，请设置或插入 value。当缓存达到其容量时，它应该在插入新项目之前使最近最少使用的项目作废。

#### 参考答案

◼ **代码示例：**

##### 《 Python 版本 》

class LRUCache(object):

def init (self, capacity): """

:type capacity: int """

self.cache = {} self.keys = []

self.capacity = capacity

def visit\_key(self, key): if key in self.keys:

self.keys.remove(key) self.keys.append(key)

def elim\_key(self): key = self.keys[0]

self.keys = self.keys[1:] del self.cache[key]

def get(self, key): """

:type key: int

:rtype: int """

if not key in self.cache: return -1

self.visit\_key(key) return self.cache[key]

def put(self, key, value): """

:type key: int

:type value: int

:rtype: void """

if not key in self.cache:

if len(self.keys) == self.capacity: self.elim\_key()

self.cache[key] = value self.visit\_key(key)

def main(): s =

[["put","put","get","put","get","put","get","get","get"],[[1,1],[2,2],[1],[3,3],[2],[ 4,4],[1],[3],[4]]]

obj = LRUCache(2) l=[]

for i,c in enumerate(s[0]): if(c == "get"):

l.append(obj.get(s[1][i][0])) else:

obj.put(s[1][i][0], s[1][i][1]) print(l)

if name == " main ": main()

**《 C++版本 》**class LRUCache{ public:

LRUCache(int capacity) { cap = capacity;

}

int get(int key) {

auto it = m.find(key);

if (it == m.end()) return -1; l.splice(l.begin(), l, it->second); return it->second->second;

}

void set(int key, int value) { auto it = m.find(key);

if (it != m.end()) l.erase(it->second); l.push\_front(make\_pair(key, value)); m[key] = l.begin();

if (m.size() > cap) {

int k = l.rbegin()->first; l.pop\_back();

m.erase(k);

}

### ／ 面试题 005 关于 epoll 和 select 的区别，哪些说法是正确的？（多选）

**——阿里巴巴出题专家：寈峰／阿里技术专家**

1. epoll 和 select 都是 I/O 多路复用的技术，都可以实现同时监听多个I/O 事件的状态。
2. epoll 相比 select 效率更高，主要是基于其操作系统支持的 I/O

事件通知机制，而 select 是基于轮询机制。

1. epoll 支持水平触发和边沿触发两种模式。
2. select 能并行支持 I/O 比较小，且无法修改。

#### 参考答案 A . B . C

### ／ 面试题 006 从 innodb 的索引结构分析，为什么索引的 key 长度不能太长？

**——阿里巴巴出题专家：近秋／阿里云数据库产品技术部技术专家**

#### 参考答案

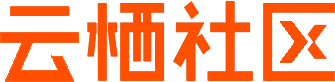
key 太长会导致一个页当中能够存放的 key 的数目变少，间接导致索引树的页数目变多，索引层次增加，从而影响整体查询变更的效率。

### ／ 面试题 007 MySQL 的数据如何恢复到任意时间点？

#### 参考答案

**——阿里巴巴出题专家：近秋／阿里云数据库产品技术部技术专家**

恢复到任意时间点以定时的做全量备份，以及备份增量的 binlog 日志为前提。恢复到任意时间点首先将全量备份恢复之后，再此基础上回放增加的 binlog 直至指定的时



### ／ 面试题 008 NFS 和 SMB 是最常见的两种 NAS

**（Network Attached Storage）协议，当把一个文件系统同时通过 NFS 和 SMB 协议共享给多个主机访问时， 以下哪些说法是错误的：（多选）**

**——阿里巴巴出题专家：起影／阿里云文件存储高级技术专家**

1. 不可能有这样的操作，即把一个文件系统同时通过 NFS 和 SMB

协议共享给多个主机访问。

1. 主机a 的用户通过 NFS 协议创建的文件或者目录，另一个主机b

的用户不能通过SMB 协议将其删除。

1. 在同一个目录下，主机 a 通过 NFS 协议看到文件 file.txt，主机b 通过SMB 协议也看到文件file.txt，那么它们是同一个文件。
2. 主机a 通过NFS 协议，以及主机b 通过 SMB 协议，都可以通过主机端的数据缓存，提升文件访问性能。

#### 参考答案 A . B . C

### ／ 面试题 009 输入ping IP 后敲回车，发包前会发生什么？

#### 参考答案

**——阿里巴巴出题专家：怀虎／阿里云云效平台负责人**

首先根据目的IP和路由表决定走哪个网卡，再根据网卡的子网掩码地址判断目的IP是否在子网内。如果不在则会通过arp缓存查询IP的网卡地址，不存在的话会通过广播询问目的IP的mac地址，得到后就开始发包了，同时mac地址也会被arp缓存起来。

### ／ 面试题 010 请解释下为什么鹿晗发布恋情的时候， 微博系统会崩溃，如何解决？

**——阿里巴巴出题专家：江岚／阿里巴巴数据技术高级技术专家**

#### 参考答案

##### 《 参考思路 》

1. 获取微博通过 pull 方式还是push 方式
2. 发布微博的频率要远小于阅读微博
3. 流量明星的发微博，和普通博主要区分对待，比如在 sharding

的时候，也要考虑这个因素

### ／ 面试题 011 现有一批邮件需要发送给订阅顾客，且有一个集群（集群的节点数不定，会动态扩容缩容）来 负责具体的邮件发送任务，如何让系统尽快地完成发送？ 请详述技术方案！

**——阿里巴巴出题专家：江岚／阿里巴巴数据技术高级技术专家**

#### 参考答案

1. 借助消息中间件，通过发布者订阅者模式来进行任务分配
2. master-slave 部署，由master 来分配任务
3. 不借助任何中间件，且所有节点均等。通过数据库的 update returning，从而实现节点之间任务的互斥

### ／ 面试题012——有一批气象观测站，现需要获取这些站点的观测数据，并存储到 Hive 中。但是气象局只提供了 api 查询，每次只能查询单个观测点。那么如果能够方便快速地获取到所有的观测点的数据？

**——阿里巴巴出题专家：江岚／阿里巴巴数据技术高级技术专家**

#### 参考答案

1. 通过shell 或python 等调用api，结果先暂存本地，最后将本地文件上传到 Hive 中。
2. 通过 datax 的 httpReader 和 hdfsWriter 插件，从而获取所需的数据。
3. 比较理想的回答，是在计算引擎的 UDF 中调用查询 api，执行UDF 的查询结果存储到对应的表中。一方面，不需要同步任务的导出导入；另一方面，计算引擎的分布式框架天生提供了分布式、容错、并发等特性。

### ／ 面试题 013 如何实现两金额数据相加（最多小数点两位）？

**——阿里巴巴出题专家：御术／蚂蚁金服数据可视化高级技术专家**

#### 参考答案

其实问题并不难，就是考察候选人对 JavaScript 数据运算上的认知以及考虑问题的缜密程度，有很多坑，可以用在笔试题，如果用在面试，回答过程中还可以随机加入有很多计算机基础的延伸。

回到这个问题，由于直接浮点相 yu 加会失精，所以要转整数；

（可以插入问遇到过吗？是否可以举个例子？）。

转整数是第一个坑，虽然只有两位可以通过乘以100转整数，但由于乘以一百和除以一百都会出现浮点数的运算，所以也会失精，还是要通过字符串来转；（可以插入问字符串转整数有几种方式？） 字符串转整是第二个坑，因为最后要对齐计算，如果没考虑周全先toFixed(2)，对于只有一位小数点数据进入计算就会错误；转整数后的计算是个加分点，很多同学往往就是直接算了，如果可以考虑大数计算的场景，恭喜同学进入隐藏关卡，这就会涉及如何有效循环、遍历、算法复杂度的问题。

### ／ 面试题 014 关于并行计算的一些基础开放问题。

**——阿里巴巴出题专家：何万青／阿里云高性能计算资深技术专家**

* 如何定义并计算，请分别阐述分布式内存到共享内存模式行编程的区别和实现（例子代码）？
* 请使用 MPI 和OpenMP 分别实现 N 个处理器对 M 个变量的求和？
* 请说明SIMD 指令在循环中使用的权限？向量化优化有哪些手段？
* 请用Amdahl 定律说明什么是并行效率以及并行算法的扩展

性？并说明扩展性的性能指标和限制因素，最后请说明在共享内存计算机中，共享内存的限制？OpenMP 是怎样实现共享内存编程环境的？MPI 阻塞和非阻塞读写的区别？

#### 参考答案

（简要答案，但必须触及，可以展开）

* 同时执行多个/算法/逻辑操作/内存访问/IO，相互独立同时运 行，分三个层次：进程级，多个节点分布式内存通过MPI通信并行；线程级，共享内存的多路机器，通过OpenMP实现多线程并行；指令集：通过SIMD指令实现单指令多数据。。。。举例吧啦吧啦。
* MPI代码，，，OpenMP代码，分别写出来 M个元素，N个处理器的累加，后者注意private 参数。

* SIMD在循环中的应用，限制在于 SIMD指令处理的每一个数组的长度，cache line利用，内部循环间的依赖和条件调用等。
* 向量化，主要看SSE和AVX指令占比率，通过编译器优化。。。。在loop代码中使用，
* 性能和计算规模随处理器增加的变化曲线，实测HPL和峰值HPL 比率，能用用Amdahl定律表达 Tpar(N) = (an + (1-a)n/N )t + C (n,N), 能够讲明白串行部分对整个并行的天花板效应，扩展性能够解释清楚算法的扩展性=并行效率随处理器数目的变化关 系，画出来。
* 共享内存计算机OpenMP对变量的限制描述，EREW，CREW，

ERCW，CRCW等区别，NUMA概念，如何保持coherent等。

* 写出OpenMP和MPI的核心函数，回答问题即可。

### ／ 面试题015 请计算XILINX公司VU9P芯片的算力相当于多少TOPS，给出计算过程与公式。

**——阿里巴巴出题专家：隐达／阿里云异构计算资深专家**

#### 参考答案

基于不同的算法，这个值在十几到几百之间。但是，如果只是单纯比算力，FPGA和ASIC、GPU相比并无太大优势，甚至大多时候有较大劣势。FPGA的优势在于高度的灵活性和算法的针对性。

### ／ 面试题016 一颗现代处理器，每秒大概可以执行多少条简单的MOV指令，有哪些主要的影响因素？

**——阿里巴巴出题专家：子团／创新产品虚拟化&稳定性资深技术专家**

#### 参考答案

及格：

每执行一条mov指令需要消耗1个时钟周期，所以每秒执行的mov 指令和CPU主频相关。

加分：

在CPU微架构上，要考虑数据预取，乱序执行，多发射，内存stall (前端stall和后端stall)等诸多因素，因此除了cpu主频外，还和流水线上的效率(IPC)强相关，比较复杂的一个问题。

### ／ 面试题 017 请分析 MaxCompute 产品与分布式技术的关系、当前大数据计算平台类产品的市场现状和发展趋势。

**——阿里巴巴出题专家：云郎／阿里 MaxCompute 高级产品专家**

#### 参考答案

开放性问题，无标准答案。

### ／ 面试题 018 对大数据平台中的元数据管理是怎么理解的，元数据收集管理体系是怎么样的，会对大数据应用有什么样的影响。

**——阿里巴巴出题专家：映泉／阿里巴巴高级技术专家**

#### 参考答案

开放性问题，无标准答案。

### ／ 面试题 019 你理解常见如阿里，和友商大数据平台的技术体系差异以及发展趋势和技术瓶颈，在存储和计算两个方面进行概述。

**——阿里巴巴出题专家：映泉／阿里巴巴高级技术专家**

#### 参考答案

开放性问题，无标准答案。

### ／ 面试题 020 在云计算大数据处理场景中，每天运行着成千上万的任务，每个任务都要进行 IO 读写。存储系统为了更好的服务，经常会保证高优先级的任务优先执行。当多个作业或用户访问存储系统时，如何保证优先级和公平性。

**——阿里巴巴出题专家：田磊磊／阿里云文件存储高级技术专家**

#### 参考答案

开放性问题，无标准答案。

### ／ 面试题 021 最大频率栈。

**——阿里巴巴出题专家：屹平／阿里云视频云边缘计算高级技术专家**实现 FreqStack，模拟类似栈的数据结构的操作的一个类。FreqStack 有两个函数： push(int x)，将整数 x 推入栈中。pop()，它移除并返回栈中出现最频繁的元素。如果最频繁的元素不只一个，则移除并返回最接近栈顶的元素。

◼ **示例：**

push [5,7,5,7,4,5]

pop() -> 返回 5，因为 5 是出现频率最高的。 栈变成

[5,7,5,7,4]。

pop() -> 返回 7，因为 5 和 7 都是频率最高的，但 7 最接近栈顶。 栈变成 [5,7,5,4]。

pop() -> 返回 5 。 栈变成 [5,7,4]。pop() -> 返回 4 。 栈变成 [5,7]。

#### 参考答案

令 freq 作为x 的出现次数的映射 Map。

此外maxfreq，即栈中任意元素的当前最大频率，因为我们必须弹出频率最高的元素。

当前主要的问题就变成了：在具有相同的（最大）频率的元素中， 怎么判断那个元素是最新的？我们可以使用栈来查询这一信息：靠近栈顶的元素总是相对更新一些。

为此，我们令 group 作为从频率到具有该频率的元素的映射。到目前，我们已经实现了 FreqStack 的所有必要的组件。

算法：

实际上，作为实现层面上的一点细节，如果 x 的频率为 f，那么我们将获取在所有 group[i] (i <= f) 中的 x,而不仅仅是栈顶的那个。这是因为每个 group[i] 都会存储与第 i 个 x 副本相关的信息。

最后，我们仅仅需要如上所述维持 freq，group，以及 maxfreq。

◼ **代码示例：**

class FreqStack { Map<Integer, Integer> freq;

Map<Integer, Stack<Integer>> group; int maxfreq;

public FreqStack() {

freq = new HashMap(); group = new HashMap(); maxfreq = 0;

}

public void push(int x) {

int f = freq.getOrDefault(x, 0) + 1;

freq.put(x, f);

if (f > maxfreq) maxfreq = f;

group.computeIfAbsent(f, z-> new Stack()).push(x);

}

public int pop() {

int x = group.get(maxfreq).pop(); freq.put(x, freq.get(x) - 1);

if (group.get(maxfreq).size() == 0) maxfreq--;

return x;

}

}

复杂度分析：

时间复杂度：对于 push 和 pop 操作， O(1)。

空间复杂度： O(N)，其中 N 为 FreqStack 中元素的数目。

### ／ 面试题 022 给定一个链表，删除链表的倒数第 N 个节点，并且返回链表的头结点。

**——阿里巴巴出题专家：屹平／阿里云视频云边缘计算高级技术专家**

◼ **示例：**

给定一个链表: 1->2->3->4->5, 和 n = 2.

当删除了倒数第二个节点后，链表变为 1->2->3->5.

说明：

给定的 n 保证是有效的。要求：

只允许对链表进行一次遍历。

#### 参考答案

我们可以使用两个指针而不是一个指针。第一个指针从列表的开头向前移动 n+1n+1 步，而第二个指针将从列表的开头出发。现在， 这两个指针被 nn 个结点分开。我们通过同时移动两个指针向前来保持这个恒定的间隔，直到第一个指针到达最后一个结点。此时第二个指针将指向从最后一个结点数起的第 nn 个结点。我们重新链接第二个指针所引用的结点的 next 指针指向该结点的下下个结点。

◼ **代码示例：**

public ListNode removeNthFromEnd(ListNode head, int n)

{

ListNode dummy = new ListNode(0); dummy.next = head;

ListNode first = dummy; ListNode second = dummy;

// Advances first pointer so that the gap between first and second is n nodes apart

for (int i = 1; i <= n + 1; i++) { first = first.next;

}

// Move first to the end, maintaining the gap while (first != null) {

first = first.next; second = second.next;

}

second.next = second.next.next; return dummy.next;

}

复杂度分析：

\*时间复杂度：O(L)，该算法对含有 L 个结点的列表进行了一次遍历。因此时间复杂度为 O(L)。

\*空间复杂度：O(1)，我们只用了常量级的额外空间。

### ／ 面试题 023 如果让你设计一个通用的、支持各种数据库秒级备份和恢复的系统，你会如何设计？

**——阿里巴巴出题专家：千震／阿里云数据库高级技术专家**

#### 参考答案

开放性问题，无标准答案。

### ／ 面试题 024 如果让你来设计一个支持数据库、NOSQL 和大数据之间数据实时流动的数据流及处理的系统，你会考虑哪些问题？如何设计？

**——阿里巴巴出题专家：千震／阿里云数据库高级技术专家**

#### 参考答案

开放性问题，无标准答案。

### ／ 面试题 025 给定一个整数数组和一个整数，返回两个数组的索引，这两个索引指向的数字的加和等于指定的整数。需要最优的算法，分析算法的空间和时间复杂度。

#### 参考答案

**——阿里巴巴出题专家：龙欣／异构计算资深技术专家**

开放性问题，无标准答案。

### ／ 面试题 026 假如给你一个新产品，你将从哪些方面来保障它的质量？

**——阿里巴巴出题专家：晨晖 ／阿里云中间件技术部测试开发专家**

#### 参考答案

可以从代码开发、测试保障、线上质量三个方面来保障。

在代码开发阶段，有单元测试、代码Review、静态代码扫描等；测试保障阶段，有功能测试、性能测试、高可用测试、稳定性测试、兼容性测试等；在线上质量方面，有灰度发布、紧急回滚、故障演练、线上监控和巡检等。

### ／ 面试题 027 如何用socket 编程实现 ftp 协议？

**——阿里巴巴出题专家：吴明／阿里云弹性计算资深技术专家**

#### 参考答案

<https://www.jianshu.com/p/e9a2e0755496>

### ／ 面试题 028 请评估一下程序的执行结果？

**——阿里巴巴出题专家：桃谷／阿里云中间件技术专家**

**public class** SynchronousQueueQuiz {

**public static void** main(String[] args) **throws** Exception { BlockingQueue<Integer> queue = **new** SynchronousQueue<>();

System.***out***.print(queue.offer(1) + **" "**); System.***out***.print(queue.offer(2) + **" "**); System.***out***.print(queue.offer(3) + **" "**); System.***out***.print(queue.take() + **" "**); System.***out***.println(queue.size());

}

}

1. true true true 1 3
2. true true true (阻塞)
3. false false false null 0
4. false false false (阻塞)

#### 参考答案