

# 测试开发面试题库

牛客网出品



# 测试工程师校招面试题库导读

## 一、学习说明

本题库均来自海量真实校招面试题目大数据进行的整理，后续也会不断更新，可永久免费在线观看，如需下载，也可直接在页面 <https://www.nowcoder.com/interview/center> 进行下载  
(下载需要用牛币兑换，一次兑换可享受永久下载权限，因为后续会更新)

需要严肃说明的是：面试题库作为帮助同学准备面试的辅助资料，但是绝对不能作为备考唯一途径，因为面试是一个考察真实水平的，不是背会了答案就可以的，需要你透彻理解的，否则追问问题答不出来反而减分，毕竟技术面试中面试官最痛恨的就是背答案这个事情了。

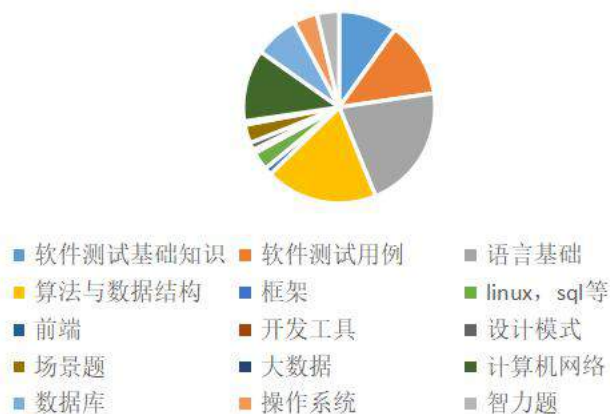
学完这个题库，把此题库都理解透彻应对各家企业面试完全没有问题。(当然，要加上好的项目以及透彻掌握)

另外，此面试题库中不包括面试中问到的项目，hr 面以及个人技术发展类。

- 项目是比较个性化的，没办法作为一个题库来给大家参考，但是如果你有一个非常有含金量的项目的话，是非常加分的，而且你的项目可能也会被问的多一些；
- hr 面的话一般来说技术面通过的话个人没有太大的和公司不符合的问题都能通过；
- 技术发展类的话这个就完全看自己啦，主要考察的会是你技术的热爱和学习能力，比如会问一些你是如何学习 xxx 技术的，或者能表达出你对技术的热爱的地方等等。此处不做赘述。

那么抛开这些，测试工程师中技术面中考察的占比如下：

测试校招面试考点分布图  
牛客网出品



**需要注意的是：**此图不绝对，因为实际面试中面试官会根据你的简历去问，比如你的项目多

可能就问的项目多一些，或者说哪里精通可能面试官就多去问你这些。而且此图是根据题库数据整理出来，并不是根据实际单场面试整理，比如基础部分不会考那么多，会从中抽着考

但是面试中必考的点且占比非常大的有测试，语言基础和**算法**。

决定你是否能拿 **sp offer**（高薪 **offer**）以及是否进名企的是项目和**算法**。

可以看出，算法除了是面试必过门槛以外，更是决定你是否能进名企或高薪 **offer** 的决定性因素。

另外关于算法部分，想要系统的学习算法思想，实现高频面试题最优解等详细讲解的话可以报名[算法名企校招冲刺班](#)或[算法高薪校招冲刺班](#)，你将能学到更先进的算法思想以及又一套系统的校招高频题目的解题套路和方法论。

多出来的服务如下：

## 算法名企校招冲刺班

- ✓ 体系化直播教学
- ✓ 全程学习委员跟班
- ✓ 金牌助教一对一答疑
- ✓ 班级群讨论



**《程序员代码面试指南》作者亲自讲解，前亚马逊，IBM，百度，Growingio技术大牛，十年算法刷题经验**

前100名报名可免费赠送签名书

如果有什么问题，也可以加 qq 咨询 1440073724，如果是早鸟的话，还可以领取优惠码哦

## 二、面试技巧

面试一般分为技术面和 hr 面，形式的话很少有群面，少部分企业可能会有一个交叉面，不过总的来说，技术面基本就是考察你的专业技术水平的，hr 面的话主要是看这个人的综合素质以及家庭情况是否符合公司要求，一般来讲，技术的话只要通过了技术面 hr 面基本上是没有问题（也有少数企业 hr 面会刷很多人）

那我们主要来说技术面，技术面的话主要是考察专业技术知识和水平，我们是可以有一定的技巧的，但是一定是基于有一定的能力水平的。

所以也慎重的告诉大家，技巧不是投机取巧，是起到辅助效果的，技术面最主要的还是要有实力，这里是基于实力水平之上的技巧。

这里告诉大家面试中的几个技巧：

### 1、简历上做一个引导：

在词汇上做好区分，比如熟悉 Java，了解 python，精通 c 语言

这样的话对自己的掌握程度有个区分，也好让面试官有个着重去问，python 本来写的也只是了解，自然就不会多问你深入的一些东西了。

### 2、在面试过程中做一个引导：

面试过程中尽量引导到自己熟知的一个领域，比如问到你来说一下 DNS 寻址，然后你简单回答（甚至这步也可以省略）之后，可以说一句，自己对这块可能不是特别熟悉，对计算机网络中的运输层比较熟悉，如果有具体的，甚至可以再加一句，比如 TCP 和 UDP

这样的话你可以把整个面试过程往你熟知的地方引导，也能更倾向于体现出你的优势而不是劣势，但是此方法仅限于掌握合适的度，比如有的知识点是必会的而你想往别处引就有点说不过去了，比如让你说几个 Java 的关键字，你一个也说不上来，那可能就真的没辙了。

### 3、在自我介绍中做一个引导：

一般面试的开头都会有一个自我介绍，在这个位置你也可以尽情的为自己的优势方面去引导。

### 4、面试过程中展示出自信：

面试过程中的态度也要掌握好，不要自卑，也不要傲娇，自信的回答出每个问题，尤其遇到不会的问题，要么做一些引导，实在不能引导也可以先打打擦边球，和面试官交流一下问题，看起来像是没听懂题意，这个过程也可以再自己思考一下，如果觉得这个过程可以免了的话也直接表明一下这个地方不太熟悉或者还没有掌握好，千万不要强行回答。

### 面试前的准备：

最重要的肯定是系统的学习了，有一个知识的框架，基础知识的牢靠程度等。

其中算法尤其重要，越来越多公司还会让你现场或者视频面试中手写代码；

另一大重要的和加分项就是项目，在面试前，一定要练习回答自己项目的三个问题：

- 这是一个怎样的项目
- 用到了什么技术，为什么用这项技术（以及每项技术很细的点以及扩展）
- 过程中遇到了什么问题，怎么解决的。

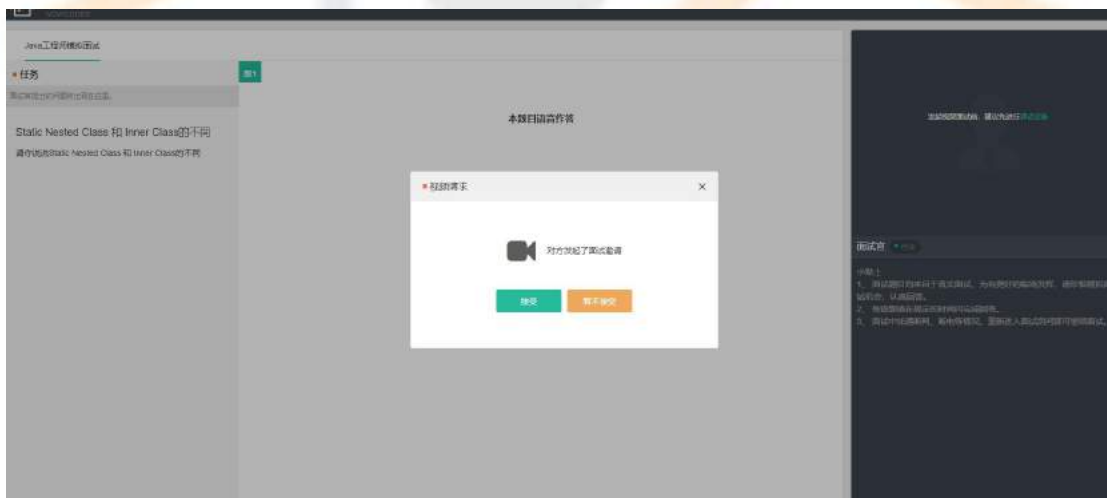
那么话说回来，这个的前提是你要有一个好的项目，牛客网 CEO 叶向宇有带大家做项目，感兴趣的可以去了解一下

- 竞争力超过 70% 求职者的项目：<https://www.nowcoder.com/courses/semester/medium>  
（专属优惠码：DjPgy3x，每期限量前 100 个）
- 竞争力超过 80% 求职者的项目：<https://www.nowcoder.com/courses/semester/senior>  
（专属优惠码：DMVSexJ，每期限量前 100 个）

知识都掌握好后，剩下的就是一个心态和模拟练习啦，因为你面试的少的话现场难免紧张，而且没在那个环境下可能永远不知道自己回答的怎么样。

因为哪怕当你都会了的情况下，你的表达和心态就显得更重要了，会了但是没有表达的很清晰就很吃亏了，牛客网这边有 AI 模拟面试，完全模拟了真实面试环境，正好大家可以真正的去练习一下，还能收获一份面试报告：

<https://www.nowcoder.com/interview/ai/index>



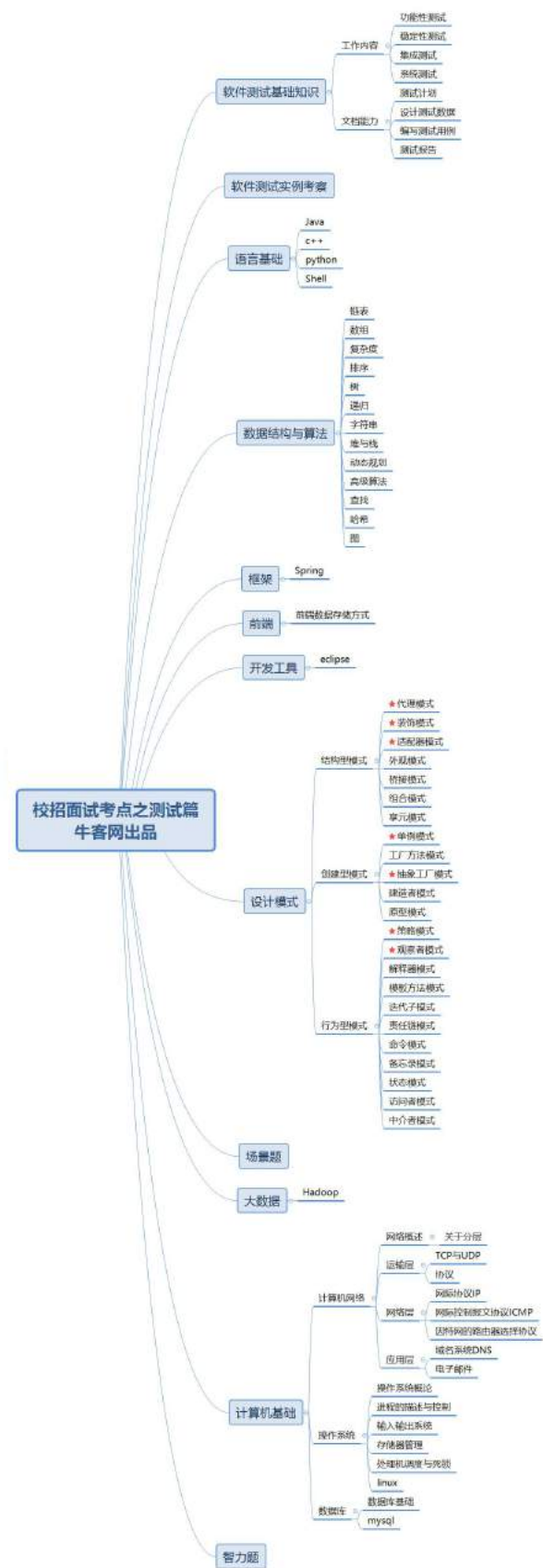
### 面试后需要做的：

面试完了的话就不用太在意结果了，有限的时间就应该做事半功倍的事情，当然，要保持电话邮箱畅通，不然别给你发 offer 你都不知道。

抛开这些，我们需要做的是及时将面试中的问题记录下来，尤其是自己回答的不够好的问题，一定要花时间去研究，并解决这些问题，下次面试再遇到相同的问题就能很好的解决，当然，即使不遇到，你这个习惯坚持住，后面也可以作为一个经历去跟面试官说，能表现出你对技术的喜爱和钻研的一个态度，同时，每次面试后你会发现自己的不足，查缺补漏的好机会，及时调整，在不断的调整和查缺补漏的过程中，你会越来越好。



### 三、面试考点导图



## 四、一对一答疑讲解戳这里

如果你对校招求职或者职业发展很困惑，欢迎与牛客网专业老师沟通，老师会帮你一对一讲解答疑哦（可以扫下方二维码或者添加微信号：niukewang985）

# 专业老师，在线答疑

• 互联网校招求职全解惑 •



互联网校招求职如何准备，如何规划...  
测试自己校招中求职竞争力，适合公司...

扫码或添加老师微信：niukewang985

## 目录

一、软件测试基础知识

二、测试实例考察

三、语言基础

1、Java

2、C++

3、python

4、shell

四、数据结构与算法

1、链表

2、数组

3、复杂度

4、排序

5、树

6、递归

7、字符串

8、堆与栈

9、动态规划

10、高级算法

11、查找

12、哈希

13、图

五、框架

六、Linux，SQL 等

七、前端

八、开发工具

九、设计模式

十、场景题

十一、大数据

1、Hadoop

十二、计算机基础

1、计算机网络

2、数据库

3、操作系统

十三、智力题

十四、职业规划

十五、技术发展

十六、hr 面

**更多名企历年笔试真题可点击直接进行练习：**

<https://www.nowcoder.com/contestRoom>





## 一、软件测试基础知识

### 1、请你分别介绍一下单元测试、集成测试、系统测试、验收测试、回归测试

**考点：**测试

**参考回答：**

1、单元测试：完成最小的软件设计单元（模块）的验证工作，目标是确保模块被正确的编码，使用过程设计描述作为指南，对重要的控制路径进行测试以发现模块内的错误，通常情况下是白盒的，对代码风格和规则、程序设计和结构、业务逻辑等进行静态测试，及早的发现和解决不易显现的错误。

2、集成测试：通过测试发现与模块接口有关的问题。目标是把通过了单元测试的模块拿来，构造一个在设计中所描述的程序结构，应当避免一次性的集成（除非软件规模很小），而采用增量集成。

自顶向下集成：模块集成的顺序是首先集成主模块，然后按照控制层次结构向下进行集成，隶属于主模块的模块按照深度优先或广度优先的方式集成到整个结构中去。

自底向上集成：从原子模块开始来进行构造和测试，因为模块是自底向上集成的，进行时要要求所有隶属于某个给顶层次的模块总是存在的，也不再使用稳定测试桩的必要。

3、系统测试：是基于系统整体需求说明书的黑盒类测试，应覆盖系统所有联合的部件。系统测试是针对整个产品系统进行的测试，目的是验证系统是否满足了需求规格的定义，找出与需求规格不相符合或与之矛盾的地方。系统测试的对象不仅仅包括需要测试的产品系统的软件，还要包含软件所依赖的硬件、外设甚至包括某些数据、某些支持软件及其接口等。因此，必须将系统中的软件与各种依赖的资源结合起来，在系统实际运行环境下来进行测试。

4、回归测试：回归测试是指在发生修改之后重新测试先前的测试用例以保证修改的正确性。理论上，软件产生新版本，都需要进行回归测试，验证以前发现和修复的错误是否在新软件版本上再次出现。根据修复好了的缺陷再重新进行测试。回归测试的目的在于验证以前出现过但已经修复好的缺陷不再重新出现。一般指对某已知修正的缺陷再次围绕它原来出现时的步骤重新测试。

5、验收测试：验收测试是指系统开发生命周期方法论的一个阶段，这时相关的用户或独立测试人员根据测试计划和结果对系统进行测试和接收。它让系统用户决定是否接收系统。它是一项确定产品是否能够满足合同或用户所规定需求的测试。验收测试包括 Alpha 测试和 Beta 测试。

Alpha 测试：是由用户在开发者的场所来进行的，在一个受控的环境中进行。

Beta 测试：由软件的最终用户在一个或多个用户场所来进行的，开发者通常不在现场，用户记录测试中遇到的问题并报告给开发者，开发者对系统进行最后的修改，并开始准备发布最终的软件。



2、请你回答一下单元测试、集成测试、系统测试、验收测试、回归测试这几步中最重要的  
是哪一步

考点：测试

参考回答：

这些测试步骤分别在软件开发的阶段对软件进行测试，我认为对软件完整功能进行测试的系统测试很重要，因为此时单元测试和集成测试已完成，能够对软件所有功能进行功能测试，能够覆盖系统所有联合的部件，是针对整个产品系统进行的测试，能够验证系统是否满足了需求规格的定义，因此我认为系统测试很重要。

3、请回答集成测试和系统测试的区别，以及它们的应用场景主要是什么？

考点：测试

参考回答：

区别：

1、计划和用例编制的先后顺序：从 V 模型来讲，在需求阶段就要制定系统测试计划和用例，HLD 的时候做集成测试计划和用例，有些公司的具体实践不一样，但是顺序肯定是先做系统测试计划用例，再做集成。

2、用例的粒度：系统测试用例相对很接近用户接受测试用例，集成测试用例比系统测试用例更详细，而且对于接口部分要重点写，毕竟要集成各个模块或者子系统。

3、执行测试的顺序：先执行集成测试，待集成测试出的问题修复之后，再做系统测试。

应用场景：

集成测试：完成单元测试后，各模块联调测试；集中在各模块的接口是否一致、各模块间的数据流和控制流是否按照设计实现其功能、以及结果的正确性验证等等；可以是整个产品的集成测试，也可以是大模块的集成测试；集成测试主要是针对程序内部结构进行测试，特别是对程序之间的接口进行测试。集成测试对测试人员的编写脚本能力要求比较高。测试方法一般选用黑盒测试和白盒测试相结合。

系统测试：针对整个产品的全面测试，既包含各模块的验证性测试（验证前两个阶段测试的正确性）和功能性（产品提交个用户的功能）测试，又包括对整个产品的健壮性、安全性、可维护性及各种性能参数的测试。系统测试测试软件《需求规格说明书》中提到的功能是否有遗漏，是否正确的实现。做系统测试要严格按照《需求规格说明书》，以它为标准。测试方法一般都使用黑盒测试法。

4、请问测试开发需要哪些知识？需要具备什么能力？

考点：测试



参考回答：

需要的知识：

软件测试基础理论知识，如黑盒测试、白盒测试等；

考编程语言基础，如 C/C++、java、python 等；

自动化测试工具，如 Selenium、Appium、Robotium 等；

计算机基础知识，如数据库、Linux、计算机网络等；

测试框架，如 JUnit 等。

需要具备的能力：

业务分析能力，分析整体业务流程、分析被测业务数据、分析被测系统架构、分析被测业务模块、分析测试所需资源、分析测试完成目标；

缺陷洞察能力，一般缺陷的发现能力、隐性问题的发现能力、发现连带问题的能力、发现问题隐患的能力、尽早发现问题的能力、发现问题根源的能力；

团队协作能力，合理进行人员分工、协助组员解决问题、配合完成测试任务、配合开发重现缺陷、督促项目整体进度、出现问题勇于承担；

专业技术能力，掌握测试基础知识、掌握计算机知识、熟练运用测试工具；

逻辑思考能力，判断逻辑的正确性、对可行性逻辑分析、站在客观角度思考；

问题解决能力，技术上的问题、工作中的问题、沟通问题；

沟通表达能力，和技术人员、产品人员、上下级的沟通；

宏观把控能力，有效控制测试时间、有效控制测试成本、有效制定测试计划、有效进行风险评估、有效控制测试方向。

## 5、请说一说黑盒与白盒的测试方法

考点：测试

参考回答：

黑盒测试：

黑盒测试也称功能测试或数据驱动测试，它是在已知产品所应具有的功能，通过测试来检测每个功能是否都能正常使用，在测试时，把程序看作一个不能打开的黑盒子，在完全不考虑程序内部结构和内部特性的情况下，测试者在程序接口进行测试，它只检查程序功能是否按照需求规格说明书的规定正常使用，程序是否能适当地接收输入数据而产生正确的输出信息，并且保持外部信息（如数据库或文件）的完整性。



“黑盒”法着眼于程序外部结构、不考虑内部逻辑结构、针对软件界面和软件功能进行测试。“黑盒”法是穷举输入测试，只有把所有可能的输入都作为测试情况使用，才能以这种方法查出程序中所有的错误。实际上测试情况有无穷多个，因此不仅要测试所有合法的输入，而且还要对那些不合法但是可能的输入进行测试。

常用的黑盒测试方法有：等价类划分法；边界值分析法；因果图法；场景法；正交实验设计法；判定表驱动分析法；错误推测法；功能图分析法。

白盒测试：

白盒测试也称为结构测试或逻辑驱动测试，是针对被测单元内部是如何进行工作的测试。它根据程序的控制结构设计测试用例，主要用于软件或程序验证。白盒测试法检查程序内部逻辑结构，对所有的逻辑路径进行测试，是一种穷举路径的测试方法，但即使每条路径都测试过了，但仍然有可能存在错误。因为：穷举路径测试无法检查出程序本身是否违反了设计规范，即程序是否是一个错误的程序；穷举路径测试不可能检查出程序因为遗漏路径而出错；穷举路径测试发现不了与数据相关的错误。

白盒测试需要遵循的原则有：1. 保证一个模块中的所有独立路径至少被测试一次；2. 所有逻辑值均需要测试真（true）和假（false）；两种情况；3. 检查程序的内部数据结构，保证其结构的有效性；4. 在上下边界及可操作范围内运行所有循环。

常用白盒测试方法：

静态测试：不用运行程序的测试，包括代码检查、静态结构分析、代码质量度量、文档测试等等，它可以由人工进行，充分发挥人的逻辑思维优势，也可以借助软件工具（Fxcop）自动进行。

动态测试：需要执行代码，通过运行程序找到问题，包括功能确认与接口测试、覆盖率分析、性能分析、内存分析等。

白盒测试中的逻辑覆盖包括语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、条件组合覆盖和路径覆盖。六种覆盖标准发现错误的能力呈由弱到强的变化：

1. 语句覆盖每条语句至少执行一次。
2. 判定覆盖每个判定的每个分支至少执行一次。
3. 条件覆盖每个判定的每个条件应取到各种可能的值。
4. 判定/条件覆盖同时满足判定覆盖条件覆盖。
5. 条件组合覆盖每个判定中各条件的每一种组合至少出现一次。
6. 路径覆盖使程序中每一条可能的路径至少执行一次。

## 6、请说一下手动测试与自动化测试的优缺点



考点：测试

参考回答：

手工测试缺点：

- 1、重复的手工回归测试，代价昂贵、容易出错。
- 2、依赖于软件测试人员的能力。

手工测试优点：

- 1、测试人员具有经验和对错误的猜测能力。
- 2、测试人员具有审美能力和心理体验。
- 3、测试人员具有是非判断和逻辑推理能力。

自动化测试的优点：

1、对程序的回归测试更方便。这可能是自动化测试最主要的任务，特别是在程序修改比较频繁时，效果是非常明显的。由于回归测试的动作和用例是完全设计好的，测试期望的结果也是完全可以预料的，将回归测试自动运行，可以极大提高测试效率，缩短回归测试时间。

2、可以运行更多更繁琐的测试。自动化的一个明显的好处是在较少的时间内运行更多的测试。

3、可以执行一些手工测试困难或不可能进行的测试。比如，对于大量用户的测试，不可能同时让足够多的测试人员同时进行测试，但是却可以通过自动化测试模拟同时有许多用户，从而达到测试的目的。

4、更好地利用资源。将繁琐的任务自动化，可以提高准确性和测试人员的积极性，将测试技术人员解脱出来投入更多精力设计更好的测试用例。有些测试不适合于自动测试，仅适合于手工测试，将可自动测试的测试自动化后，可以让测试人员专注于手工测试部分，提高手工测试的效率。

5、测试具有一致性和可重复性。由于测试是自动执行的，每次测试的结果和执行的内容的一致性是可以得到保障的，从而达到测试的可重复的效果。

6、测试的复用性。由于自动测试通常采用脚本技术，这样就有可能只需要做少量的甚至不做修改，实现在不同的测试过程中使用相同的用例。

7、增加软件信任度。由于测试是自动执行的，所以不存在执行过程中的疏忽和错误，完全取决于测试的设计质量。一旦软件通过了强有力的自动测试后，软件的信任度自然会增加。

自动化测试的缺点：

- 1、不能取代手工测试





2、手工测试比自动测试发现的缺陷更多

3、对测试质量的依赖性极大

4、测试自动化不能提高有效性

5、测试自动化可能会制约软件开发。由于自动测试比手动测试更脆弱，所以维护会受到限制，从而制约软件的开发。

6、工具本身并无想像力

7、请问你怎么看待软件测试的潜力和挑战

考点：测试

参考回答：

软件测试是正在快速发展，充满挑战的领域。尽管现在许多自动化测试软件的出现使得传统手工测试的方式被代替，但自动化测试工具的开发、安全测试、测试建模、精准测试、性能测试、可靠性测试等专项测试中仍然需要大量具有专业技能与专业素养的测试人员，并且随着云计算、物联网、大数据的发展，传统的测试技术可能不再适用，测试人员也因此面临着挑战，需要深入了解新场景并针对不同场景尝试新的测试方法，同时敏捷测试、Devops 的出现也显示了软件测试的潜力。

8、你觉得软件测试的核心竞争力是什么

考点：测试

参考回答：

测试人员的核心竞争力在于提早发现问题，并能够发现别人无法发现的问题。

1、早发现问题：问题发现的越早，解决的成本越低。如果一个需求在还未实现的时候就能发现需求的漏洞，那么这种问题的价值是最高的。

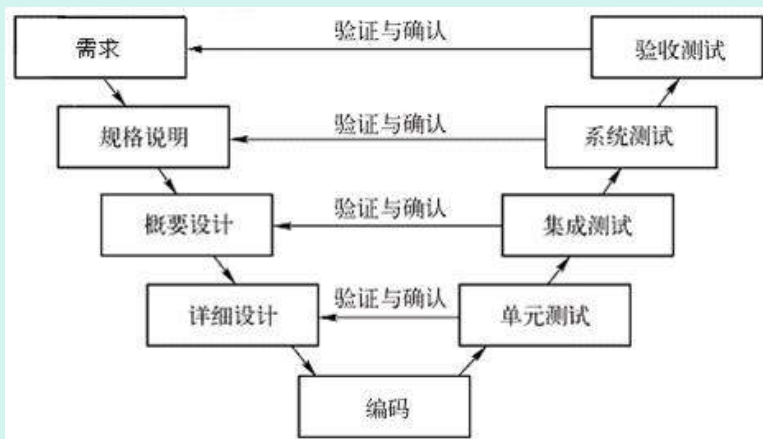
2、发现别人无法发现的问题：所有人都能发现的问题，你发现了，那就证明你是可以被替代的。别人发现不了，而你可以发现，那么你就是无法被替代。

9、你觉得测试和开发需要怎么结合才能使软件的质量得到更好的保障

考点：测试

参考回答：

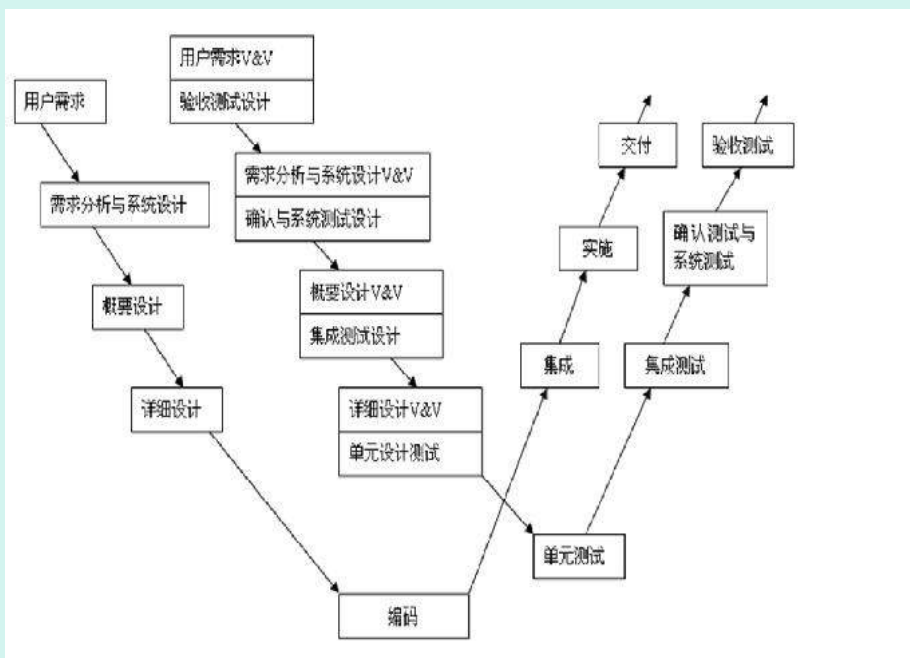
测试和开发应该按照 W 模型的方式进行结合，测试和开发同步进行，能够尽早发现软件缺陷，降低软件开发的成本。



在 V 模型中，测试过程被加在开发过程的后半部分，单元测试所检测代码的开发是否符合详细设计的要求。集成测试所检测此前测试过的各组成部分是否能完好地结合到一起。系统测试所检测已集成在一起的产品是否符合系统规格说明书的要求。而验收测试则检测产品是否符合最终用户的需求。V 模型的缺陷在于仅仅把测试过程作为在需求分析、系统设计及编码之后的一个阶段，忽视了测试对需求分析、系统设计的验证，因此需求阶段的缺陷很可能一直到后期的验收测试才被发现，此时进行弥补将耗费大量人力物力资源。

相对于 V 模型，W 模型增加了软件各开发阶段中应同步进行的验证和确认活动。W 模型由两个 V 字型模型组成，分别代表测试与开发过程，图中明确表示出了测试与开发的并行关系。

W 模型强调：测试伴随着整个软件开发周期，而且测试的对象不仅仅是程序，需求、设计等同样要测试，也就是说，测试与开发是同步进行的。W 模型有利于尽早地全面的发现问题。例如，需求分析完成后，测试人员就应该参与到对需求的验证和确认活动中，以尽早地找出缺陷所在。同时，对需求的测试也有利于及时了解项目难度和测试风险，及早制定应对措施，这将显著减少总体测试时间，加快项目进度。





W 模型中测试的活动与软件开发同步进行，测试的对象不仅仅是程序，还包括需求和设计，因此能够尽早发现软件缺陷，降低软件开发的成本。

10、你觉得单元测试可行吗

考点：测试

参考回答：

可行，单元测试可以有效地测试某个程序模块的行为，是未来重构代码的信心保证。事前可以保证质量，事后可以快速复现问题，并在修改代码后做回归自测。可行性考虑的是要用一些可行的方法做到关键的代码可测试，如通过边界条件、等价类划分、错误、因果，设计测试用例要覆盖常用的输入组合、边界条件和异常。

11、你觉得自动化测试有什么意义，都需要做些什么

考察点：自动化测试

参考回答：

自动化测试的意义在于

- 1、可以对程序的新版本自动执行回归测试
- 2、可以执行手工测试困难或者不可能实现的测试，如压力测试，并发测试，
- 3、能够更好的利用资源，节省时间和人力

执行自动化测试之前首先判断这个项目是不是和推广自动化测试，然后对项目做需求分析，指定测试计划，搭建自动化测试框架，设计测试用例，执行测试，评估

12、请你回答一下测试的相关流程是什么？

考察点：测试流程

参考回答：

测试最规范的过程如下

需求测试->概要设计测试->详细设计测试->单元测试->集成测试->系统测试->验收测试

来自 W 模型

13、请你说一下如何写测试用例

考察点：测试用例



- 1、测试人员尽早介入，彻底理解清楚需求，这个是写好测试用例的基础
- 2、如果以前有类似的需求，可以参考类似需求的测试用例，然后还需要看类似需求的 bug 情况
- 3、清楚输入、输出的各种可能性，以及各种输入的之间的关联关系，理解清楚需求的执行逻辑，通过等价类、边界值、判定表等方法找出大部分用例
- 4、找到需求相关的一些特性，补充测试用例
- 5、根据自己的经验分析遗漏的测试场景
- 6、多总结类似功能点的测试点，才能够写出质量越来越高的测试用例
- 7、书写格式一定要清晰

14、请问你觉得测试项目具体工作是什么？

考察点：测试计划，测试策略

参考回答：

搭建测试环境

撰写测试用例

执行测试用例

写测试计划，测试报告

测试，并提交 BUG 表单

跟踪 bug 修改情况

执行自动化测试，编写脚本，执行，分析，报告

进行性能测试，压力测试等其他测试，执行，分析，调优，报告

15、请问如果想进行 bug 的测评，怎么去评测 bug？

考察点：bug 的评测定级

参考回答：

Bug 的 priority ( ) 和 severity ( ) 是两个重要属性，通常人员在提交 bug 的时候，只定义 severity, 而将 priority 交给 leader 定义, 通常 bug 管理中, severity 分为四个等级 blocker、



critical、major、minor/trivial，而 priority 分为五个等级 immediate、urgent、high、normal、low。

Severity:

1、blocker: 即系统无法执行，崩溃，或严重资源不足，应用模块无法启动或异常退出，无法测试，造成系统不稳定。常见的有严重花屏、内存泄漏、用户数据丢失或破坏、系统崩溃/死机/冻结、模块无法启动或异常退出、严重的数值计算错误、功能设计与需求严重不符、其它导致无法测试的错误，如服务器 500 错误。

2、critical: 即映像系统功能或操作，主要功能存在严重缺陷，但不会映像到系统稳定性。常见的有：功能未实现，功能错误、系统刷新错误、数据通讯错误、轻微的数值计算错误、影响功能及界面的错误字或拼写错误。

3、major: 即界面、性能缺陷、兼容性，常见的有：操作界面错误，边界条件错误，提示信息错误，长时间操作无进度提示，系统未优化，兼容性问题。

4、minor/trivial: 即易用性及建议性问题。

Priority

1、immediate: 即马上解决，

2、urgent: 急需解决

3、high: 高度重视，有时间要马上解决

4、low: 在系统发布前解决，或确认可以不用解决。

## 16、请你说一说测试用例的边界

考点：测试

参考回答：

边界值分析法就是对输入或输出的边界值进行测试的一种黑盒测试方法。通常边界值分析法是作为对等价类划分法的补充，这种情况下，其测试用例来自等价类的边界。

常见的边界值

1) 对 16-bit 的整数而言 32767 和 -32768 是边界

2) 屏幕上光标在最左上、最右下位置

3) 报表的第一行和最后一行

4) 数组元素的第一个和最后一个





5)循环的第 0 次、第 1 次和倒数第 2 次、最后一次

### 17、请你说一下软件质量的六个特征

考点：测试

参考回答：

按照软件质量国家标准 GB-T8566--2001G，软件质量可以用下列特征来评价：

- a. 功能特征：与一组功能及其指定性质有关的一组属性，这里的功能是满足明确或隐含的需求的那些功能。
- b. 可靠特征：在规定的一段时间和条件下，与软件维持其性能水平的能力有关的一组属性。
- c. 易用特征：由一组规定或潜在的用户为使用软件所需作的努力和所作的评价有关的一组属性。
- d. 效率特征：与在规定条件下软件的性能水平与所使用资源量之间关系有关的一组属性。
- e. 可维护特征：与进行指定的修改所需的努力有关的一组属性。
- f. 可移植特征：与软件从一个环境转移到另一个环境的能力有关的一组属性。

### 18、请你说一下设计测试用例的方法

考点：测试

参考回答：

黑盒测试：

#### 1. 等价类划分

等价类划分是将系统的输入域划分为若干部分，然后从每个部分选取少量代表性数据进行测试。等价类可以划分为有效等价类和无效等价类，设计测试用例的时候要考虑这两种等价类。

#### 2. 边界值分析法

边界值分析法是对等价类划分的一种补充，因为大多数错误都在输入输出的边界上。边界值分析就是假定大多数错误出现在输入条件的边界上，如果边界附件取值不会导致程序出错，那么其他取值出错的可能性也就很小。



边界值分析法是通过优先选择不同等价类间的边界值覆盖有效等价类和无效等价类来更有效的进行测试，因此该方法要和等价类划分法结合使用。

### 3. 正交试验法

正交是从大量的试验点中挑选出适量的、有代表性的点。正交试验设计是研究多因素多水平的一种设计方法，他是一种基于正交表的高效率、快速、经济的试验设计方法。

### 4. 状态迁移法

状态迁移法是对一个状态在给定的条件内能够产生需要的状态变化，有没有出现不可达的状态和非法的状态，状态迁移法是设计足够的用例达到对系统状态的覆盖、状态、条件组合、状态迁移路径的覆盖。

### 5. 流程分析法

流程分析法主要针对测试场景类型属于流程测试场景的测试项下的测试子项进行设计，这是从白盒测试中路径覆盖分析法借鉴过来的一种很重要的方法。

### 6. 输入域测试法

输入域测试法是针对输入会有各种各样的输入值的一个测试，他主要考虑 极端测试、中间范围测试，特殊值测试 。

### 7. 输出域分析法

输出域分析法是对输出域进行等价类和边界值分析，确定是要覆盖的输出域样点，反推得到应该输入的输入值，从而构造出测试用例，他的目的是为了达到输出域的等价类和边界值覆盖。

### 8. 判定表分析法

判定表是分析和表达多种输入条件下系统执行不同动作的工具，他可以把复杂的逻辑关系和多种条件组合的情况表达的即具体又明确；

### 9. 因果图法

因果图是用于描述系统输入输出之间的因果关系、约束关系。因果图的绘制过程是对被测系统的外部特征的建模过程，根据输入输出间的因果图可以得到判定表，从而规划出测试用例。

### 10. 错误猜测法

错误猜测法主要是针对系统对于错误操作时对于操作的处理法的猜测法，从而设计测试用例

### 11. 异常分析法

异常分析法是针对系统有可能存在的异常操作，软硬件缺陷引起的故障进行分析，分析发生错误时系统对于错误的处理能力和恢复能力依此设计测试用例。

白盒测试：



白盒测试也称为结构测试或逻辑驱动测试，是针对被测单元内部是如何进行工作的测试。它根据程序的控制结构设计测试用例，主要用于软件或程序验证。白盒测试法检查程序内部逻辑结构，对所有的逻辑路径进行测试，是一种穷举路径的测试方法，但即使每条路径都测试过了，但仍然有可能存在错误。因为：穷举路径测试无法检查出程序本身是否违反了设计规范，即程序是否是一个错误的程序；穷举路径测试不可能检查出程序因为遗漏路径而出错；穷举路径测试发现不了与数据相关的错误。

白盒测试需要遵循的原则有：1. 保证一个模块中的所有独立路径至少被测试一次；2. 所有逻辑值均需要测试真（true）和假（false）；两种情况；3. 检查程序的内部数据结构，保证其结构的有效性；4. 在上下边界及可操作范围内运行所有循环。

常用白盒测试方法：

静态测试：不用运行程序的测试，包括代码检查、静态结构分析、代码质量度量、文档测试等等，它可以由人工进行，充分发挥人的逻辑思维优势，也可以借助软件工具（Fxcop）自动进行。

动态测试：需要执行代码，通过运行程序找到问题，包括功能确认与接口测试、覆盖率分析、性能分析、内存分析等。

白盒测试中的逻辑覆盖包括语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、条件组合覆盖和路径覆盖。六种覆盖标准发现错误的能力呈由弱到强的变化：

1. 语句覆盖每条语句至少执行一次。
2. 判定覆盖每个判定的每个分支至少执行一次。
3. 条件覆盖每个判定的每个条件应取到各种可能的值。
4. 判定/条件覆盖同时满足判定覆盖条件覆盖。
5. 条件组合覆盖每个判定中各条件的每一种组合至少出现一次。
6. 路径覆盖使程序中每一条可能的路径至少执行一次。

## 19、请你说一说测试工程师的必备技能

考点：测试

参考回答：

需要的知识：

- 软件测试基础理论知识，如黑盒测试、白盒测试等；
- 编程语言基础，如 C/C++、java、python 等；



- 自动化测试工具，如 Selenium、Appium、Robotium 等；
- 计算机基础知识，如数据库、Linux、计算机网络等；
- 测试框架，如 JUnit 等。

需要具备的能力：

- 业务分析能力，分析整体业务流程、分析被测业务数据、分析被测系统架构、分析被测业务模块、分析测试所需资源、分析测试完成目标；
- 缺陷洞察能力，一般缺陷的发现能力、隐性问题的发现能力、发现连带问题的能力、发现问题隐患的能力、尽早发现问题的能力、发现问题根源的能力；
- 团队协作能力，合理进行人员分工、协助组员解决问题、配合完成测试任务、配合开发重现缺陷、督促项目整体进度、出现问题勇于承担；
- 专业技术能力，掌握测试基础知识、掌握计算机知识、熟练运用测试工具；
- 逻辑思考能力，判断逻辑的正确性、对可行性逻辑分析、站在客观角度思考；
- 问题解决能力，技术上的问题、工作中的问题、沟通问题；
- 沟通表达能力，和技术人员、产品人员、上下级的沟通；
- 宏观把控能力，有效控制测试时间、有效控制测试成本、有效制定测试计划、有效进行风险评估、有效控制测试方向。

## 20、请你说一下 app 性能测试的指标

考点：测试

参考回答：

1、内存：内存消耗测试节点的设计目标是为了让应用不占用过多的系统资源，且及时释放内存，保障整个系统的稳定性。当然关于内存测试，在这里我们需要引入几个概念：空闲状态、中等规格、满规格。

空闲状态指打开应用后，点击 home 键让应用后台运行，此时应用处于的状态叫做空闲；中等规格和满规格指的是对应用的操作时间的间隔长短不一，中等规格时间较长，满规格时间较短。

内存测试中存在很多测试子项，清单如下：

- 空闲状态下的应用内存消耗；
- 中等规格状态下的应用内存消耗；
- 满规格状态下的应用内存消耗；



- 应用内存峰值；
- 应用内存泄露；
- 应用是否常驻内存；
- 压力测试后的内存使用。

## 2、CPU：

使用 Android 提供的 view plaincopy 在 CODE 上查看代码片派生到我的代码片

adbshell dumsys CPUinfo |grep packagename >/address/CPU.txt 来获取；

使用 top 命令 view plaincopy 在 CODE 上查看代码片派生到我的代码片

adbshell top |grep packagename>/address/CPU.txt 来获取。

## 3、流量：

网络流量测试是针对大部分应用而言的，可能还有部分应用会关注网速、弱网之类的测试。

流量测试包括以下测试项：

应用首次启动流量提示；

应用后台连续运行 2 小时的流量值；

应用高负荷运行的流量峰值。

## 4、电量：

- 测试手机安装目标 APK 前后待机功耗无明显差异；
- 常见使用场景中能够正常进入待机，待机电流在正常范围内；
- 长时间连续使用应用无异常耗电现象。

## 5、启动速度：

第一类：首次启动 --应用首次启动所花费的时间；

第二类：非首次启动 --应用非首次启动所花费的时间；

第三类：应用界面切换--应用界面内切换所花费的时间。

## 6、滑动速度、界面切换速度

## 7、与服务器交互的网络速度





## 21、请你说一说 app 测试的工具

考点：测试

参考回答：

功能测试自动化

a) 轻量接口自动化测试

jmeter,

b) APP UI 层面的自动化

android: UI Automator Viewer, Android Junit, Instrumentation, UIAutomator,

iOS: 基于 Instrument 的 iOS UI 自动化,

性能测试

a) Web 前端性能测试

网络抓包工具: Wireshark

网页文件大小

webpagetest

pagespeed insight

chrome adb

b) APP 端性能测试

Android 内存占用分析: MAT

iOS 内存问题分析: ARC 模式

Android WebView 性能分析:

iOS WebView 性能分析

c) 后台服务性能测试

负载, 压力, 耐久性

可拓展性, 基准

工具: apacheAB, Jmeter, LoadRunner,



专项测试

a) 兼容性测试

手工测试：操作系统，分辨率，rom，网络类型

云平台：testin，脚本编写，Android。

b) 流量测试

Android 自带的流量管理，

iOS 自带的 Network

tcpdump 抓包

WiFi 代理抓包：Fiddler

流量节省方法：压缩数据，json 优于 xml；WebP 优于传统的 JPG，PNG；控制访问的频次；只获取必要的数据；缓存；

c) 电量测试

基于测试设备的方法，购买电量表进行测试。

GSam Battery Monitoe Pro

iOS 基于 Instrument Energy 工具

d) 弱网络测试

手机自带的网络状况模拟工具

基于代理的弱网络的模拟：

工具：windows：Network Delay Simulator

Mac：Network Link Conditioner

## 22、请你说一说 bug 的周期，以及描述一下不同类别的 bug

考点：测试

参考回答：

1、New：（新的）



当某个“bug”被第一次发现的时候，测试人员需要与项目负责人沟通以确认发现的的确是一个 bug，如果被确认是一个 bug，就将其记录下来，并将 bug 的状态设为 New

## 2、Assigned（已指派的）

当一个 bug 被指认为 New 之后，将其反馈给开发人员，开发人员将确认这是否是一个 bug，如果是，开发组的负责人就将这个 bug 指定给某位开发人员处理，并将 bug 的状态设定为“Assigned”

## 3、Open（打开的）

一旦开发人员开始处理 bug 的时候，他（她）就将这个 bug 的状态设置为“Open”，这表示开发人员正在处理这个“bug”

## 4、Fixed（已修复的）

当开发人员进行处理（并认为已经解决）之后，他就可以将这个 bug 的状态设置为“Fixed”并将其提交给开发组的负责人，然后开发组的负责人将这个 bug 返还给测试组

## 5、Pending Reset（待在测试的）

当 bug 被返还到测试组后，我们将 bug 的状态设置为 Pending Reset”

## 6、Reset（再测试）

测试组的负责人将 bug 指定给某位测试人员进行再测试，并将 bug 的状态设置为“Reset”

## 7、Closed（已关闭的）

如果测试人员经过再次测试之后确认 bug 已经被解决之后，就将 bug 的状态设置为“Closed”

## 8、Reopen（再次打开的）

如果经过再次测试发现 bug（指 bug 本身而不是包括因修复而引发的新 bug）仍然存在的话，测试人员将 bug 再次传递给开发组，并将 bug 的状态设置为“Reopen”

## 9、Pending Reject（拒绝中）

如果测试人员传递到开发组的 bug 被开发人员认为是正常行为而不是 bug 时，这种情况下开发人员可以拒绝，并将 bug 的状态设置为“Pending Reject”

## 10、Rejected（被拒绝的）

测试组的负责人接到上述 bug 的时候，如果他（她）发现这是产品说明书中定义的正常行为或者经过与开发人员的讨论之后认为这并不能算作 bug 的时候，开发组负责人就将这个 bug 的状态设置为“Rejected”

## 11、Postponed（延期）



有些时候，对于一些特殊的 bug 的测试需要搁置一段时间，事实上有很多原因可能导致这种情况的发生，比如无效的测试数据，一些特殊的无效的功能等等，在这种情况下，bug 的状态就被设置为“Postponed”

不同类别的 bug：

Bug 类型

- 代码错误
- 界面优化
- 设计缺陷
- 配置相关
- 安装部署
- 安全相关
- 性能问题
- 标准规范
- 测试脚本
- 其他

### 23、请你说一说 PC 网络故障，以及如何排除障碍

考点：测试

参考回答：

(1)首先是排除接触故障，即确保你的网线是可以正常使用的。然后禁用网卡后再启用，排除偶然故障。打开网络和共享中心窗口，单击窗口左上侧“更改适配器设置”右击其中的“本地连接“或”无线网络连接”，单击快捷菜单中的“禁用”命令，即可禁用所选网络。接下来重启网络，只需右击后单击启用即可。

(2)使用 ipconfig 查看计算机的上网参数

1、单击“开始|所有程序|附件|命令提示符“，打开命令提示符窗口

2、输入 ipconfig，按 Enter 确认，可以看到机器的配置信息，输入 ipconfig/all, 可以看到 IP 地址和网卡物理地址等相关网络详细信息。



### (3)使用 ping 命令测试网络的连通性，定位故障范围

在命令提示符窗口中输入”ping 127. 0. 0. 1“，数据显示本机分别发送和接受了 4 个数据包，丢包率为零，可以判断本机网络协议工作正常，如显示”请求超时“，则表明本机网卡的安装或 TCP/IP 协议有问题，接下来就应该检查网卡和 TCP/IP 协议，卸载后重装即可。

### (4)ping 本机 IP

在确认 127. 0. 0. 1 地址能被 ping 通的情况下，继续使用 ping 命令测试本机的 IP 地址能否被 ping 通，如不能，说明本机的网卡驱动程序不正确，或者网卡与网线之间连接有故障，也有可能是本地的路由表面收到了破坏，此时应检查本机网卡的状态是否为已连接，网络参数是否设置正确，如果正确可是不能 ping 通，就应该重新安装网卡驱动程序。丢失率为零，可以判断网卡安装配置没有问题，工作正常。

### (5)ping 网关

网关地址能被 ping 通的话，表明本机网络连接以及正常，如果命令不成功，可能是网关设备自身存在问题，也可能是本机上网参数设置有误，检查网络参数。

## 24、请你说一说测试的常用方法

考点：测试

参考回答：

黑盒测试：

黑盒测试也称功能测试或数据驱动测试，它是在已知产品所应具有的功能，通过测试来检测每个功能是否都能正常使用，在测试时，把程序看作一个不能打开的黑盒子，在完全不考虑程序内部结构和内部特性的情况下，测试者在程序接口进行测试，它只检查程序功能是否按照需求规格说明书的规定正常使用，程序是否能适当地接收输入数据而产生正确的输出信息，并且保持外部信息（如数据库或文件）的完整性。

“黑盒”法着眼于程序外部结构、不考虑内部逻辑结构、针对软件界面和软件功能进行测试。“黑盒”法是穷举输入测试，只有把所有可能的输入都作为测试情况使用，才能以这种方法查出程序中所有的错误。实际上测试情况有无穷多个，因此不仅要测试所有合法的输入，而且还要对那些不合法但是可能的输入进行测试。

常用的黑盒测试方法有：等价类划分法；边界值分析法；因果图法；场景法；正交实验设计法；判定表驱动分析法；错误推测法；功能图分析法。

白盒测试：

白盒测试也称为结构测试或逻辑驱动测试，是针对被测单元内部是如何进行工作的测试。它根据程序的控制结构设计测试用例，主要用于软件或程序验证。白盒测试法检查程序内部逻辑结构，对所有的逻辑路径进行测试，是一种穷举路径的测试方法，但即使每条路径都测试过了，但仍然有可能存在错误。因为：穷举路径测试无法检查出程序本身是否违反了设计规范，即程序是





否是一个错误的程序；穷举路径测试不可能检查出程序因为遗漏路径而出错；穷举路径测试发现不了一些与数据相关的错误。

白盒测试需要遵循的原则有： 1. 保证一个模块中的所有独立路径至少被测试一次；2. 所有逻辑值均需要测试真（true）和假（false）；两种情况；3. 检查程序的内部数据结构，保证其结构的有效性；4. 在上下边界及可操作范围内运行所有循环。

常用白盒测试方法：

静态测试：不用运行程序的测试，包括代码检查、静态结构分析、代码质量度量、文档测试等等，它可以由人工进行，充分发挥人的逻辑思维优势，也可以借助软件工具（Fxcop）自动进行。

动态测试：需要执行代码，通过运行程序找到问题，包括功能确认与接口测试、覆盖率分析、性能分析、内存分析等。

白盒测试中的逻辑覆盖包括语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、条件组合覆盖和路径覆盖。六种覆盖标准发现错误的能力呈由弱到强的变化：

1. 语句覆盖每条语句至少执行一次。
2. 判定覆盖每个判定的每个分支至少执行一次。
3. 条件覆盖每个判定的每个条件应取到各种可能的值。
4. 判定/条件覆盖同时满足判定覆盖条件覆盖。
5. 条件组合覆盖每个判定中各条件的每一种组合至少出现一次。
6. 路径覆盖使程序中每一条可能的路径至少执行一次。

## 25、请你说一下黑盒白盒

考点：测试

参考回答：

黑盒测试：

黑盒测试也称功能测试或数据驱动测试，它是在已知产品所应具有的功能，通过测试来检测每个功能是否都能正常使用，在测试时，把程序看作一个不能打开的黑盒子，在完全不考虑程序内部结构和内部特性的情况下，测试者在程序接口进行测试，它只检查程序功能是否按照需求规格说明书的规定正常使用，程序是否能适当地接收输入数据而产生正确的输出信息，并且保持外部信息（如数据库或文件）的完整性。

“黑盒”法着眼于程序外部结构、不考虑内部逻辑结构、针对软件界面和软件功能进行测试。“黑盒”法是穷举输入测试，只有把所有可能的输入都作为测试情况使用，才能以这种方法查出



程序中所有的错误。实际上测试情况有无穷多个，因此不仅要测试所有合法的输入，而且还要对那些不合法但是可能的输入进行测试。

常用的黑盒测试方法有：等价类划分法；边界值分析法；因果图法；场景法；正交实验设计法；判定表驱动分析法；错误推测法；功能图分析法。

白盒测试：

白盒测试也称为结构测试或逻辑驱动测试，是针对被测单元内部是如何进行工作的测试。它根据程序的控制结构设计测试用例，主要用于软件或程序验证。白盒测试法检查程序内部逻辑结构，对所有的逻辑路径进行测试，是一种穷举路径的测试方法，但即使每条路径都测试过了，但仍然有可能存在错误。因为：穷举路径测试无法检查出程序本身是否违反了设计规范，即程序是否是一个错误的程序；穷举路径测试不可能检查出程序因为遗漏路径而出错；穷举路径测试发现不了与数据相关的错误。

白盒测试需要遵循的原则有： 1. 保证一个模块中的所有独立路径至少被测试一次；2. 所有逻辑值均需要测试真（true）和假（false）；两种情况；3. 检查程序的内部数据结构，保证其结构的有效性；4. 在上下边界及可操作范围内运行所有循环。

常用白盒测试方法：

静态测试：不用运行程序的测试，包括代码检查、静态结构分析、代码质量度量、文档测试等等，它可以由人工进行，充分发挥人的逻辑思维优势，也可以借助软件工具（Fxcop）自动进行。

动态测试：需要执行代码，通过运行程序找到问题，包括功能确认与接口测试、覆盖率分析、性能分析、内存分析等。

白盒测试中的逻辑覆盖包括语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、条件组合覆盖和路径覆盖。六种覆盖标准发现错误的能力呈由弱到强的变化：

1. 语句覆盖每条语句至少执行一次。
2. 判定覆盖每个判定的每个分支至少执行一次。
3. 条件覆盖每个判定的每个条件应取到各种可能的值。
4. 判定/条件覆盖同时满足判定覆盖条件覆盖。
5. 条件组合覆盖每个判定中各条件的每一种组合至少出现一次。
6. 路径覆盖使程序中每一条可能的路径至少执行一次。

## 26、请你说一说你知道的自动化测试框架

考点：测试

参考回答：



### 1、模块化测试框架

模块化测试脚本框架（TEST MODuLARITY FRAMEWORK）需要创建小而独立的可以描述的模块、片断以及待测应用程序的脚本。这些树状结构的小脚本组合起来，就能组成能用于特定的测试用例的脚本。在五种框架中，模块化框架是最容易掌握和使用的。在一个组件上方建立一个抽象层使其在余下的应用中隐藏起来，这是众所周知的编程技巧。这样应用同组件中的修改隔离开来，提供了程序设计的模块化特性。模块化测试脚本框架使用这一抽象或者封装的原理来提高自动测试组合的可维护性和可升级性。

### 2、测试库框架

测试库框架（Test Library Architecture）与模块化测试脚本框架很类似，并且具有同样的优点。不同的是测试库框架把待测应用程序分解为过程和函数而不是脚本。这个框架需要创建描述模块、片断以及待测应用程序的功能库文件。

### 3、关键字驱动或表驱动的测试框架

对于一个独立于应用的自动化框架，关键字驱动（KEYWORD DRIVEN）测试和表驱动（TABLE DRIVEN）测试是可以互换的术语。这个框架需要开发数据表和关键字。这些数据表和关键字独立于执行它们的测试自动化工具，并可以用来“驱动”待测应用程序和数据的测试脚本代码，关键字驱动测试看上去与手工测试用例很类似。在一个关键字驱动测试中，把待测应用程序的功能和每个测试的执行步骤一起写到一个表中。这个测试框架可以通过很少的代码来产生大量的测试用例。同样的代码在用数据表来产生各个测试用例的同时被复用。

### 4、数据驱动测试框架

数据驱动（DATA DRIVEN），LJ 试是一个框架。在这里测试的输入和输出数据是从数据文件中读取（数据池，ODBC 源，CSV 文件，EXCEL 文件，ADO 对象等）并且通过捕获工具生成或者手工生成的代码脚本被载入到变量中。在这个框架中，变量不仅被用来存放输入值还被用来存放输出的验证值。整个程序中，测试脚本来读取数值文件，记载测试状态和信息。这类似于表驱动测试，在表驱动测试中，它的测试用例是包含在数据文件而不是在脚本中，对于数据而言，脚本仅仅是一个“驱动器”，或者是一个传送机构。然而，数据驱动测试不同于表驱动测试，尽管导航数据并不包含在表结构中。在数据驱动测试中，数据文件中只包含测试数据。这个框架意图减少需要执行所有测试用例所需要的总的测试脚本数。数据驱动需要很少的代码来产生大量的测试用例，这与表驱动极其类似。

### 5、混合测试自动化（Hybrid Test Automation）框架

最普遍的执行框架是上面介绍的所有技术的一个结合，取其长处，弥补其不足。这个混合测试框架是由大部分框架随着时间并经过若干项目演化而来的

## 27、请你说一说 web 测试和 app 测试的不同点

考点：测试

参考回答：



系统架构方面：

web 项目，一般都是 b/s 架构，基于浏览器的

app 项目，则是 c/s 的，必须要有客户端，用户需要安装客户端。

web 测试只要更新了服务器端，客户端就会同步会更新。App 项目则需要客户端和服务端都更新。

性能方面：

web 页面主要会关注响应时间

而 app 则还需要关心流量、电量、CPU、GPU、Memory 这些。

它们服务端的性能没区别，都是一台服务器。

兼容方面：

web 是基于浏览器的，所以更倾向于浏览器和电脑硬件，电脑系统的方向的兼容

app 测试则要看分辨率，屏幕尺寸，还要看设备系统。

web 测试是基于浏览器的所以不必考虑安装卸载。

而 app 是客户端的，则必须测试安装、更新、卸载。除了常规的安装、更新、卸载还要考虑到异常场景。包括安装时的中断、弱网、安装后删除安装文件。

此外 APP 还有一些专项测试：如网络、适配性。

## 28、请问你了解什么测试方法

考察点：测试方法

参考回答：

等价类划分，边界值分析，错误推测，因果图法，逻辑覆盖法，程序插桩技术，基本路径法，符号测试，错误驱动测试

## 29、请问黑盒测试和白盒测试有哪些方法

考察点：测试方法

参考回答：

黑盒测试方法有等价类划分，边界值分析，错误推测，因果图法

白盒测试方法有逻辑覆盖法，程序插桩技术，基本路径法，符号测试，错误驱动测试

30、请问你怎么看待测试，知道哪些测试的类型，有用过哪些测试方法？

考察点：测试类型

参考回答：

测试是软件开发中不可或缺的一环，测试通过经济，高效的方法，捕捉软件中的错误，从而达到保重软件内在质量的目的。

测试分为功能测试和非功能测试，非功能测试又可以分为性能测试、压力测试、容量测试、健壮性测试、安全性测试、可靠性测试、恢复性测试、备份测试、协议测试、兼容性测试、可用性测试、配置测试、GUI 测试。

测试方法用过等价划分法、边值分析法、错误推测法、因果图法。

31、请问你怎么测试网络协议

考察点：测试设计

参考回答：

协议测试包括四种类型的测试

1、一致性测试：检测协议实现本身与协议规范的符合程度

2、互操作性测试：基于某一协议检测不同协议实现间互操作互通信的能力

3、性能测试：检测协议实现的性能指标，比如数据传输速度，连接时间，执行速度，吞吐量，并发度，

4、健壮性测试：检测协议是现在各种恶劣环境下运行的能力，比如注入干扰报文，通信故障，信道被切断

32、请你回答一下什么是 $\alpha$ 测试和 $\beta$ 测试，以及什么时候用到他们

考察点：测试类型

参考回答：

$\alpha$  测试：在受控的环境中进行，由用户在开发者的场所进行，并且在开发者对用户的指导下进行测试，开发者负责记录发现的错误和使用中遇到的问题

$\beta$  测试：在开发者不能控制的环境中的真实应用，由软件的最终用户们在一个或多个客户场所下进行，由用户记录在测试中遇到的一系列问题，并定期报给开发者

## 二、软件测试实例

### 1、给你一个字符串，你怎么判断是不是 ip 地址？ 手写这段代码，并写出测试用例

考点：编程/算法

参考回答：

IP 的格式：(1~255).(0~255).(0~255).(0~255)

方法一：基于对字符串的处理

```
public static void main(String[] args){

    Scanner scanner = new Scanner(System.in);

    String ipStr = scanner.next();

    boolean isIpLegal = isIpLegal(ipStr);

    if(isIpLegal) {

        System.out.println(ipStr + " 合法");

    }

    else{

        System.out.println(ipStr + " 非法");

    }

}

public static boolean isIpLegal(String str){

    //检查 ip 是否为空

    if(str == null){

        return false;

    }

    //检查 ip 长度，最短为： x. x. x. x(7 位)，最长为： xxx. xxx. xxx. xxx(15 位)
```





```
if(str.length() < 7 || str.length() > 15){

return false;

}

//对输入字符串的首末字符判断，如果是"."则是非法 IP

if(str.charAt(0) == '.' || str.charAt(str.length()-1) == '.'){

return false;

}

//按"."分割字符串，并判断分割出来的个数，如果不是 4 个，则是非法 IP

String[] arr = str.split("\\.");

if(arr.length != 4){

return false;

}

//对分割出来的每个字符串进行单独判断

for(int i = 0; i < arr.length; i++){

//如果每个字符串不是一位字符，且以'0'开头，则是非法的 IP，如：01.002.03.004

if(arr[i].length() > 1 && arr[i].charAt(0) == '0'){

return false;

}

//对每个字符串的每个字符进行逐一判断，如果不是数字 0-9，则是非法的 IP

for(int j = 0; j < arr[i].length(); j++){

if (arr[i].charAt(j) < '0' || arr[i].charAt(j) > '9'){

return false;

}

}

}

}
```



//对拆分的每一个字符串进行转换成数字，并判断是否在 0~255

```
for(int i = 0; i < arr.length; i++){  
  
    int temp = Integer.parseInt(arr[i]);  
  
    if(i == 0){  
  
        if (temp < 1 || temp > 255){  
  
            return false;  
  
        }  
  
    }  
  
    else{  
  
        if(temp < 0 || temp > 255){  
  
            return false;  
  
        }  
  
    }  
  
    }  
  
    return true;  
  
}
```

方法二：正则表达式

```
public static void main(String[] args) {  
  
    Scanner scanner = new Scanner(System.in);  
  
    String ipStr = scanner.next();  
  
    boolean isIpLegal = isIpLegal(ipStr);  
  
    if(isIpLegal) {  
  
        System.out.println(ipStr + " 合法");  
  
    }  
  
    else{
```



```
System.out.println(ipStr + " 非法");

}

}

public static boolean isIpLegal(String ipStr) {

    String ipRegex =
    "^[1-9]|([1-9][0-9])|(1[0-9][0-9])|(2[0-4][0-9])|(25[0-5]))\\.([0-9]|([1-9][0-9])|
    (1[0-9][0-9])|(2[0-4][0-9])|(25[0-5]))){3}$";

    Pattern pattern = Pattern.compile(ipRegex);

    Matcher matcher = pattern.matcher(ipStr);

    if (matcher.matches()) {

        return true;

    } else {

        return false;

    }

}
```

测试用例：

等价类划分：

有效可用的 IP 地址	
A 类	1. 0. 0. 0 -126. 255. 255. 254
A 私有	10. 0. 0. 0 -10. 255. 255. 254
B 类	128. 0. 0. 0
	-191. 255. 255. 254



B 私有	172. 16. 0. 0 -172. 31. 255. 254
C 类	192. 0. 0. 0 -223. 255. 255. 254
C 私有	192. 168. 0. 0-192. 168. 255. 254
windows 自动 分配	169. 254. 0. 0-169. 254. 255. 254
有效但不可用的 IP 地址	
D	224. 0. 0. 0 -239. 255. 255. 254
E	240. 0. 0. 0 -255. 255. 255. 254
全网	0. x. x. x, x. x. x. 0
广播	x. x. x. 255
回环	127. 0. 0. 0 -127. 255. 255. 254

--

输入	结果



64. 11. 22. 33	有效可用
10. 12. 13. 14	有效可用，不能直接访问公网
151. 123. 234 . 56	有效可用
172. 20. 123. 56	有效可用，不能直接访问公网
192. 127. 35. 65	有效可用
192. 168. 128 . 128	有效可用，不能直接访问公网
169. 254. 15. 200	有效可用，不能直接访问公网
224. 1. 2. 3	有效不可用，超过有效范围(D 类)
250. 11. 22. 3 3	有效不可用，超过有效范围(E 类)
0. 200. 3. 4	有效不可用，全网地址
64. 11. 22. 0	有效不可用，全网地址
10. 12. 13. 25 5	有效不可用，广播地址



0	127. 50. 60. 7	有效不可用，回环地址

## 2、请进行测试用例设计：一串数字，闰年的判别

考点：测试

参考回答：

判断闰年的标准是:能整除 4 且不能整除 100，能整除 400。设定合法的年份为 1-9999

```
public class Test2 {  
  
    public static void main(String[] args) {  
  
        Scanner in = new Scanner (System.in);  
  
        int year=in.nextInt();  
  
        if(year<=0||year>9999)  
  
        {  
  
            System.out.println("请输入正确的年份");  
  
        }  
  
        if((year%4==0&&year%100!=0)||year%400==0)  
  
        {  
  
            System.out.println("闰年");  
  
        }else  
  
        {  
  
            System.out.println("不是闰年");  
  
        }  
  
    }  
  
}
```

测试用例：



测试用例	输入	预期输出
被 4 整除，但是不被 100 整除的年份	008 <sup>2</sup>	闰年
被 4 整除，同时被 100 整除的年份，且被 400 整除的年份	000 <sup>2</sup>	闰年
被 4 整除，同时被 100 整除的年份，但是不被 400 整除的年份	900 <sup>1</sup>	不是闰年
偶数，不被 4 整除的年份	022 <sup>2</sup>	不是闰年
奇数年份	999 <sup>1</sup>	不是闰年
年份大于 9999	0000 <sup>1</sup>	请输入正确的年份
年份小于 0	0 <sup>0</sup>	请输入正确的年份

### 3、请你说一说简单用户界面登陆过程都需要做哪些分析

考点：测试

参考回答：

#### 一、功能测试

1. 输入正确的用户名和密码，点击提交按钮，验证是否能正确登录。
2. 输入错误的用户名或者密码，验证登录会失败，并且提示相应的错误信息。



3. 登录成功后能否跳转到正确的页面
4. 用户名和密码，如果太短或者太长，应该怎么处理
5. 用户名和密码，中有特殊字符（比如空格），和其他非英文的情况
6. 记住用户名的功能
7. 登陆失败后，不能记录密码的功能
8. 用户名和密码前后有空格的处理
9. 密码是否非明文显示，使用星号圆点等符号代替。
10. 牵扯到验证码的，还要考虑文字是否扭曲过度导致辨认难度大，考虑颜色（色盲使用者），刷新或换一个按钮是否好用
11. 登录页面中的注册、忘记密码，登出用另一帐号登陆等链接是否正确
12. 输入密码的时候，大写键盘开启的时候要有提示信息。
13. 什么都不输入，点击提交按钮，检查提示信息。

## 二、界面测试

1. 布局是否合理，textbox 和按钮是否整齐。
2. textbox 和按钮的长度，高度是否复合要求。
3. 界面的设计风格是否与 UI 的设计风格统一。
4. 界面中的文字简洁易懂，没有错别字。

## 三、性能测试

1. 打开登录页面，需要的时间是否在需求要求的时间内。
2. 输入正确的用户名和密码后，检查登录成功跳转到新页面的时间是否在需求要求的时间内。
3. 模拟大量用户同时登陆，检查一定压力下能否正常登陆跳转。

## 四、安全性测试

1. 登录成功后生成的 Cookie，是否是 httponly（否则容易被脚本盗取）。
2. 用户名和密码是否通过加密的方式，发送给 Web 服务器。
3. 用户名和密码的验证，应该用服务器端验证，而不能单单是在客户端用 javascript 验证。



4. 用户名和密码的输入框，应该屏蔽 SQL 注入攻击。
5. 用户名和密码的的输入框，应该禁止输入脚本（防止 XSS 攻击）。
6. 防止暴力破解，检测是否有错误登陆的次数限制。
7. 是否支持多用户在同一机器上登录。
8. 同一用户能否在多台机器上登录。

#### 五、可用性测试

1. 是否可以全用键盘操作，是否有快捷键。
2. 输入用户名，密码后按回车，是否可以登陆。
3. 输入框能否可以以 Tab 键切换。

#### 六、兼容性测试

1. 不同浏览器下能否显示正常且功能正常（IE, 6, 7, 8, 9, Firefox, Chrome, Safari, 等）。
2. 同种浏览器不同版本下能否显示正常且功能正常。
2. 不同的平台是否能正常工作，比如 Windows, Mac。
3. 移动设备上是否正常工作，比如 Iphone, Andriod。
4. 不同的分辨率下显示是否正常。

#### 七、本地化测试

1. 不同语言环境下，页面的显示是否正确。

**4、请对这个系统做出测试用例：一个系统，多个摄像头，抓拍车牌，识别车牌，上传网上，网上展示**

考点：测试

参考回答：

功能：

1. 每个摄像头都能抓拍车牌；
2. 每个摄像头抓拍到的车牌能正常交给系统处理；
3. 系统能够正确识别车牌；



4. 系统能够将识别出的车牌上传；

5. 上传至网络的车牌能够正常展示出来；

#### 一、功能测试

1. 使用正常的车牌，保持车牌静止，检查每个摄像头是否能抓拍车牌；

2. 使用类似非车牌的写有字的纸板，检查每个摄像头是否抓拍；

3. 使用正常的车牌，保持车牌较高速移动，检查每个摄像头是否能抓拍车牌；

4. 在多种情况下检查每个摄像头抓拍到的车牌能否正常交给系统处理，如临时断电、断网后能否正常将数据交给系统；

5. 使用抓拍到的正常的车牌，交由系统处理，检查系统能否识别车牌；

6. 使用非车牌的其他图片，交由系统处理，检查系统能否识别；

7. 在多种情况下检查系统能否将正常识别出的车牌进行上传，如临时断电、断网后未上传数据是否能继续上传；

8. 构造非车牌的其他内容的数据，检查系统能否将异常内容进行上传；

9. 检查上传至网络的车牌能否正常展示出来；

10. 上传非车牌的其他内容的数据，检查能否正常显示出来。

#### 二、性能测试

1. 同时向一个摄像头展示多个静止的车牌，检查摄像头能否抓拍到多个车牌；

2. 同时向一个摄像头展示多个较高速运动的车牌，检查摄像头能否抓拍到多个车牌；

3. 抓拍后，检查系统识别车牌的时间是否在需求要求的时间内；

4. 模拟大量抓拍照片同时交由系统处理，检查一定压力下系统能否正常识别车牌；

5. 模拟大量车牌同时上传，检查一定压力下能否上传成功。

#### 三、安全性测试

1. 检查是否能够通过给车牌加装饰物等方法，使摄像头无法抓拍或抓拍后系统无法正常识别车牌。

### 5、请你对吃鸡游戏进行压力测试

考点：测试



参考回答：

一. 首先明确需要测试压力的内容：

1. 游戏服务器硬件

a. 硬盘 I/o

b. 内存

c. CPU

2. 网络压力

a. 长连接

a1. 最大连接数

a2. 流量(内网、外网、进、出)

b. 长连接短周期(类似 Http 的 TCP 应用，这个比较特殊的一个需求，专门针对 LoginAgent)

b1. 每秒建立的连接数

b2. 实际处理能力

3. 数据库

a. 每秒事务数

b. 每秒锁等待数

c. 平均延时(ms)

d. CPU 暂用

4. 多线程的最优线程数

a. 数据库执行的多线程

b. 多连接处理

二. Windows Server 环境测试方式

1. 服务器性能监测

使用 Server 自带的性能监测器设置各个进程的监测参数。Window 的这个自动工具做的相当强大。大家自己摸一摸基本就会用了。每个参数都由详细的说明。

2. 案例设计注意



a. 对于数据库的性能测试上，现在由于所有的游戏服务器构架在 DB 前面都有一个实现 DB 缓冲功能的进程，以减少数据库频繁的读写操作。所以其实数据库的读是一个轻量级的数量；而数据库的写操作是一个周期性能过程。案例设计一定要能够驱动这种周期性能过程。比如我们游戏的战斗，导致游戏玩家数据的改变，或驱动所有在线玩家数据的周期性存储。

b. 选择具有代表性，并且最频繁的游戏操作。用于进行最高用户在线的各种性能指标采集。如，开枪、道具拾取、道具使用、移动、聊天

#### c. 聊天性能测试

广播聊天是最为考验游戏信息发送能力的功能。通过进行全局广播的压力测试。我们可以获取服务器进程发送信息到客户端的最高承载量。进而可以对我们的各种广播功能进行一个预估和频率限制。

#### d. 同屏玩家的移动测试

移动+广播。这两种信息，基本是网络游戏流量的 70-80%左右。同屏玩家数量，将会增加各种数据的广播需求，非常影响游戏性能。所以同屏的移动测试也是广播测试的一个必要环节。需要根据实际结果进行适当的优化。

#### e. 大量玩家同时登录测试

玩家登录时，有大量的信息需要进行分配和初始化；同时也有大量的数据需要下传客户端。服务器需要进行大量的 TCP 连接建立。所以是一个比较关键的过程。这个测试案例是一个比较特殊，但是运营是肯定会碰到的案例。

f. 由于线程池处理事务，随着事务的时耗，存在一个最优线程数的问题。过多的线程反而会降低服务器效率

### 3. 细节问题

#### a. 进行测试需要仔细思考客户端性能影响服务器最后表现的可能性。比如

a1. 模拟客户端的性能无法有效处理服务器返回信息，可能就导致服务器发送的信息缓存在服务器系统缓存，从而表现出服务器内存不断增加。表现为服务器发送能力不足，其实可能根本就是客户端的性能问题

a2. 客户端性能问题，导致发起的请求数过少，从而导致单位时间内服务器处理的请求过少。表现为服务器性能不足，其实根本就是客户端的请求能力不足。

#### b. 网络带宽导致最后表现不足

b1. 确认服务器的各个网卡，以及相互的带宽。不然可能因为相互带宽，导致服务器对于客户端请求的处理延时。表现为服务器卡机

b2. 客户端模拟多个玩家，比如 1000 个玩家。而客户端的网卡或者客户端与服务器之间的中转服务器带宽过小，导致服务器数据发送不出，内存不断增加。表现为服务器发送能力不足，其实是中间带宽问题。





c. debug i/o 导致服务器性能下降

c1. 进行性能测试，一定要取消 debug 用的同步的 i/o. 比如我们服务器的 debuginternalLog. 同步 i/o 是非常影响性能的，特别在压力测试下可能导致每秒上千上万甚至几十万次的执行。一处的文件写入操作就可以导致几十万次的处理能力变成几千次的处理能力。

c2. 客户端避免进行阻塞操作导致模拟多用户性能下降，导致服务器表现性能下降

d. 流量需要区分内网网

内、外网流量在游戏正式运行时是完全分开的。价格也是完全不同的。一个千 M 的外网是一个无法想象的运营成本，而 kmbps/s 现在已经是一个可以接受的代价。游戏进程需要进行不同网卡的配置和绑定。确定内外网流量。

## 6、请你根据微信登录界面设计测试用例

考点：测试

参考回答：

### 一、功能测试

1. 输入正确的用户名和密码，点击提交按钮，验证是否能正确登录。
2. 输入错误的用户名或者密码，验证登录会失败，并且提示相应的错误信息。
3. 登录成功后能否能否跳转到正确的页面
4. 检查能否选择不同登录方式进行登录，如使用手机号登录、使用微信号登录或扫码登录。
5. 记住用户名的功能
6. 登陆失败后，不能记录密码的功能
7. 密码是否非明文显示显示，使用星号圆点等符号代替。
8. 有验证码时，还要考虑文字是否扭曲过度导致辨认难度大，考虑颜色、刷新或换一个按钮是否好用
9. 登录页面中的注册、忘记密码，登出用另一帐号登陆等链接是否正确
10. 输入密码的时候，大写键盘开启的时候要有提示信息。
11. 什么都不输入，点击提交按钮，检查提示信息。

### 二、界面测试

1. 布局是否合理，textbox 和按钮是否整齐。



2. textbox 和按钮的长度，高度是否复合要求。

3. 界面的设计风格是否与 UI 的设计风格统一。

4. 界面中的文字简洁易懂，没有错别字。

### 三、性能测试

1. 打开登录页面，需要的时间是否在需求要求的时间内。

2. 输入正确的用户名和密码后，检查登录成功跳转到新页面的时间是否在需求要求的时间内。

3. 模拟大量用户同时登陆，检查一定压力下能否正常登陆跳转。

### 四、安全性测试

1. 登录成功后生成的 Cookie，是否是 httponly（否则容易被脚本盗取）。

2. 用户名和密码是否通过加密的方式，发送给 Web 服务器。

3. 用户名和密码的验证，应该用服务器端验证，而不能单单是在客户端用 javascript 验证。

4. 用户名和密码的输入框，应该屏蔽 SQL 注入攻击。

5. 用户名和密码的输入框，应该禁止输入脚本（防止 XSS 攻击）。

6. 防止暴力破解，检测是否有错误登陆的次数限制。

7. 是否支持多用户在同一机器上登录。

8. 同一用户能否在多台机器上登录。

### 五、兼容性测试

1. 不同移动平台或 PC 环境下能否显示正常且功能正常

2. 同种平台下不同微信版本下能否显示正常且功能正常。

3. 不同的分辨率下显示是否正常。

### 七、本地化测试

1. 不同语言环境下，页面的显示是否正确。

## 7、请你对朋友圈点赞功能进行测试

考点：测试



参考回答：

1. 是否可以正常点赞和取消；
2. 点赞的人是否在可见分组里；
3. 点赞状态是否能即时更新显示；
4. 点赞状态，共同好友是否可见；
6. 性能检测，网速快慢对其影响；
7. 点赞显示的是否正确，一行几个；
8. 点赞是否按时间进行排序，头像对应的是否正确；
9. 是否能在消息列表中显示点赞人的昵称、5. 不同手机，系统显示界面如何；

备注：

10. 可扩展性测试，点赞后是否能发表评论；
11. 是否在未登录时可查看被点赞的信息。

#### 8. 如果做一个杯子的检测，你如何测试

考点：测试

参考回答：

##### 1. 功能

- (1) 水倒水杯容量的一半
- (2) 水倒规定的安全线
- (4) 水杯容量刻度与其他水杯一致
- (5) 盖子拧紧水倒不出来

##### (6) 烫手验证

##### 2. 性能

- (1) 使用最大次数或时间
- (2) 掉地上不易损坏
- (3) 盖子拧到什么程度水倒不出来



- (4) 保温时间长
- (5) 杯子的耐热性
- (6) 杯子的耐寒性
- (7) 长时间放置水不会漏
- (8) 杯子上放置重物到什么程度杯子会被损坏

### 3. 界面

- (1) 外观完整、美观
- (2) 大小与设计一样（高、宽、容量、直径）
- (3) 拿着舒服
- (4) 材质与设计一样
- (5) 杯子上的图案掉落
- (6) 图案遇水溶解

### 4. 安全

- (1) 杯子使用的材质毒或细菌的验证
- (2) 高温材质释放毒性
- (3) 低温材质释放毒性

### 5. 易用性

- (1) 倒水方便
- (2) 喝水方便
- (3) 携带方便
- (4) 使用简单，容易操作
- (5) 防滑措施

### 6. 兼容性

- (1) 杯子能够容纳果汁、白水、酒精、汽油等。

### 7. 震动测试



(1) 杯子加包装（有填充物），六面震动，检查产品是否能应对铁路/公路/航空运输。

8. 可移植性

(1) 杯子在不同地方、温度环境下都可以正常使用。

## 9、如何对一个页面进行测试

考点：测试

参考回答：

1、UI 测试：页面布局、页面样式检查、控件长度是否够长；显示时，是否会被截断；支持的快捷键，Tab 键切换焦点顺序正确性等。

2、功能测试：页面上各类控件的测试范围，测试点。结合控件的实际作用来补充检查点：比如，密码框是否\*显示，输入是否做 trim 处理等。

3、安全测试：输入特殊字符，sql 注入，脚本注入测试。后台验证测试，对于较重要的表单，绕过 js 检验后台是否验证；数据传输是否加密处理，比如，直接请求转发，地址栏直接显示发送字符串？

4、兼容性测试

5、性能测试

## 10、如何对水壶进行测试

考点：测试

参考回答：（同快手对水杯的测试）

1. 功能

(1) 水倒水壶容量的一半

(2) 水倒规定的安全线

(4) 水壶容量刻度与其他水壶一致

(5) 盖子拧紧水倒不出来

(6) 烫手验证

2. 性能



- (1) 使用最大次数或时间
- (2) 掉地上不易损坏
- (3) 盖子拧到什么程度水倒不出来
- (4) 保温时间长
- (5) 壶的耐热性
- (6) 壶的耐寒性
- (7) 长时间放置水不会漏
- (8) 壶上放置重物到什么程度壶会被损坏

### 3. 界面

- (1) 外观完整、美观
- (2) 大小与设计一样（高、宽、容量、直径）
- (3) 拿着舒服
- (4) 材质与设计一样
- (5) 壶上的图案掉落
- (6) 图案遇水溶解

### 4. 安全

- (1) 壶使用的材质毒或细菌的验证
- (2) 高温材质释放毒性
- (3) 低温材质释放毒性

### 5. 易用性

- (1) 倒水方便
- (2) 喝水方便
- (3) 携带方便
- (4) 使用简单，容易操作
- (5) 防滑措施





6. 兼容性

(1) 壶能够容纳果汁、白水、酒精、汽油等。

7. 震动测试

(1) 壶加包装（有填充物），六面震动，检查产品是否能应对铁路/公路/航空运输。

8. 可移植性

(1) 壶在不同地方、温度环境下都可以正常使用。

## 11、如何对淘宝搜索框进行测试

考点：测试

参考回答：

### 一，功能测试

1. 输入关键字，查看：返回结果是否准确，返回的文本长度需限制

1.1 输入可查到结果的正常关键字、词、语句，检索到的内容、链接正确性；

1.2 输入不可查到结果的关键字、词、语句；

1.3 输入一些特殊的内容，如空、特殊符、标点符、极限值等，可引入等价类划分的方法等；

2. 结果显示：标题，卖家，销售量，单行/多行，是否有图片

3. 结果排序：价格 销量 评价 综合

4. 返回结果庞大时，限制第一页的现实量，需支持翻页

5. 多选项搜索：关键字 品牌 产地 价格区间 是否天猫 是否全国购

6. 是否支持模糊搜索，支持通配符的查询

7. 网速慢的情况下的搜索

8. 搜索结果为空的情况

9. 未登录情况和登录情况下的搜索（登录情况下 存储用户搜索的关键字/搜索习惯）

### 二. 性能测试：

1 压力测试：在不同发用户数压力下的表现（评价指标如响应时间等）



2 负载测试：看极限能承载多大的用户量同时正常使用

3 稳定性测试：常规压力下能保持多久持续稳定运行

4 内存测试：有无内存泄漏现象

5 大数据量测试：如模拟从庞大的海量数据中搜索结果、或搜索出海量的结果后列示出来，看表现如何等等。

三. 易用性：交互界面的设计是否便于、易于使用

1 依据不同的查询结果会有相关的人性化提示，查不到时告知？查到时统计条数并告知？有疑似输入条件错误时提示可能正确的输入项等等处理；

2 查询出的结果罗列有序，如按点击率或其他排序规则，确保每次查询出的结果位置按规则列示方便定位，显示字体、字号、色彩便于识别等等；

3 标题查询、全文检索、模糊查询、容错查询、多关键字组织查询（空格空格开）等实用的检索方式是否正常？

4 输入搜索条件的控件风格设计、位置摆放是否醒目便于使用者注意到，有否快照等快捷查看方式等人性化设计？

四. 兼容性

1 WINDOWS/LINUX/UNIX 等各类操作系统下及各版本条件下的应用

2 IE/FIREFOX/GOOGLE/360/QQ 等各类浏览器下及各版本条件下、各种显示分辨率条件下的应用

3 SQL/ORACLE/DB2/MYSQL 等各类数据库存储情况下的兼容性测试

4 简体中文、繁体中文、英文等各类语种软件平台下的兼容性测试

5 IPHONE/IPAD、安卓等各类移动应用平台下的兼容性测试

6 与各相关的监控程序的兼容性测试，如输入法、杀毒、监控、防火墙等工具同时使用

五. 安全性

1 被删除、加密、授权的数据，不允许被 SQL 注入等攻击方式查出来的，是否有安全控制设计；

2 录入一些数据库查询的保留字符，如单引号、%等等，造成查询 SQL 拼接出的语句产生漏洞，如可以查出所有数据等等，这方面要有一些黑客攻击的思想并引入一些工具和技术，如爬网等。

3 通过白盒测试技术，检查一下在程序设计上是否存在安全方面的隐患；



4 对涉及国家安全、法律禁止的内容是否进行了相关的过滤和控制；

## 12、如何对一瓶矿泉水进行测试

考点：测试

参考回答：

界面测试：查看外观是否美观

功能度：查看水瓶漏不漏；瓶中水能不能被喝到

安全性：瓶子的材质有没有毒或细菌

可靠性：从不同高度落下的损坏程度

可移植性：再不同的地方、温度等环境下是否都可以正常使用

兼容性：是否能够容纳果汁、白水、酒精、汽油等

易用性：是否烫手、是否有防滑措施、是否方便饮用

用户文档：使用手册是否对的用法、限制、使用条件等有详细描述

疲劳测试：将盛上水（案例一）放 24 小时检查泄漏时间和情况；盛上汽油（案例二）放 24 小时检查泄漏时间和情况等

压力测试：用根针并在针上面不断加重量，看压强多大时会穿透

跌落测试：测试在何种高度跌落会破坏水瓶

## 13、如何测试登陆界面

考点：测试

参考回答：

一、功能测试

1. 输入正确的用户名和密码，点击提交按钮，验证是否能正确登录。
2. 输入错误的用户名或者密码，验证登录会失败，并且提示相应的错误信息。
3. 登录成功后能否跳转到正确的页面
4. 用户名和密码，如果太短或者太长，应该怎么处理



5. 用户名和密码，中有特殊字符（比如空格），和其他非英文的情况
6. 记住用户名的功能
7. 登陆失败后，不能记录密码的功能
8. 用户名和密码前后有空格的处理
9. 密码是否明文显示，使用星号圆点等符号代替。
10. 牵扯到验证码的，还要考虑文字是否扭曲过度导致辨认难度大，考虑颜色（色盲使用者），刷新或换一个按钮是否好用
11. 登录页面中的注册、忘记密码，登出用另一帐号登陆等链接是否正确
12. 输入密码的时候，大写键盘开启的时候要有提示信息。
13. 什么都不输入，点击提交按钮，检查提示信息。

## 二、界面测试

1. 布局是否合理，textbox 和按钮是否整齐。
2. textbox 和按钮的长度，高度是否复合要求。
3. 界面的设计风格是否与 UI 的设计风格统一。
4. 界面中的文字简洁易懂，没有错别字。

## 三、性能测试

1. 打开登录页面，需要的时间是否在需求要求的时间内。
2. 输入正确的用户名和密码后，检查登录成功跳转到新页面的时间是否在需求要求的时间内。
3. 模拟大量用户同时登陆，检查一定压力下能否正常登陆跳转。

## 四、安全性测试

1. 登录成功后生成的 Cookie，是否是 httponly（否则容易被脚本盗取）。
2. 用户名和密码是否通过加密的方式，发送给 Web 服务器。
3. 用户名和密码的验证，应该用服务器端验证，而不能单单是在客户端用 javascript 验证。
4. 用户名和密码的输入框，应该屏蔽 SQL 注入攻击。
5. 用户名和密码的输入框，应该禁止输入脚本（防止 XSS 攻击）。



6. 防止暴力破解，检测是否有错误登陆的次数限制。

7. 是否支持多用户在同一机器上登录。

8. 同一用户能否在多台机器上登录。

#### 五、可用性测试

1. 是否可以全用键盘操作，是否有快捷键。

2. 输入用户名，密码后按回车，是否可以登陆。

3. 输入框能否可以以 Tab 键切换。

#### 六、兼容性测试

1. 不同浏览器下能否显示正常且功能正常（IE, 6, 7, 8, 9, Firefox, Chrome, Safari, 等）。

2. 同种浏览器不同版本下能否显示正常且功能正常。

2. 不同的平台是否能正常工作，比如 Windows, Mac。

3. 移动设备上是否正常工作，比如 Iphone, Andriod。

4. 不同的分辨率下显示是否正常。

#### 七、本地化测试

1. 不同语言环境下，页面的显示是否正确。

### 14、请你说一下 jmeter

考点：测试

参考回答：

Jmeter: Apache JMeter 是 Apache 组织开发的基于 Java 的压力测试工具。用于对软件做压力测试，它最初被设计用于 Web 应用测试，但后来扩展到其他测试领域。 它可以用于测试静态和动态资源，例如静态文件、Java 小服务程序、CGI 脚本、Java 对象、数据库、FTP 服务器，等等。JMeter 可以用于对服务器、网络或对象模拟巨大的负载，来自不同压力类别下测试它们的强度和分析整体性能。另外，JMeter 能够对应用程序做功能/回归测试，通过创建带有断言的脚本来验证你的程序返回了你期望的结果。为了最大限度的灵活性，JMeter 允许使用正则表达式创建断言。

为什么使用 Jmeter:

- 开源免费，基于 Java 编写，可集成到其他系统可拓展各个功能插件
- 支持接口测试，压力测试等多种功能，支持录制回放，入门简单

- 相较于自己编写框架或其他开源工具，有较为完善的 UI 界面，便于接口调试
- 多平台支持，可在 Linux，Windows，Mac 上运行

用例生成与导出：

Jmeter 的用例格式为 jmx 文件，实际为 xml 格式，感兴趣可以学习下自己定制生成想要的 jmx 文件。

生成原则：

每个功能模块为一个独立的 jmx 文件。增加可维护性。（尽量不要将一个 jmx 文件放入太多功能，后期维护成本会很高。）

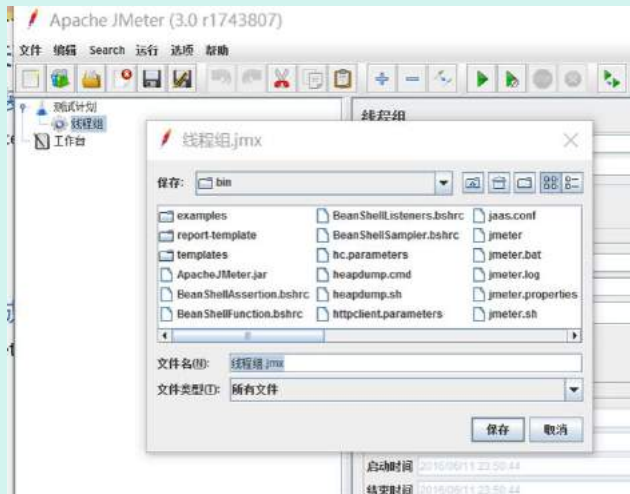
模块的私有变量保存在模块中，多模块共有的（例如服务器 ip 端口等）可以考虑存在单独的文件中读取。

接口测试不要放太多线程，毕竟不是做压力测试，意义也不大。

导出方法：

编写测试用例

文件——保存为——确定：



Jmeter 运行模式及参数

GUI 模式

打开已有的 jmx 文件（文件——打开）

点击启动按钮运行

命令行模式

依赖：

配置 jmeter 环境变量（windows 下为将\${jmeterhome}/bin 加入 Path 变量）

如果未加入环境变量，在执行的时候可以直接给出全路径或在\${jmeterhome}/bin 下执行

命令：

```
jmeter -n -t <testplan filename> -l <listener filename>
```

参数：

-h 帮助 -> 打印出有用的信息并退出

-n 非 GUI 模式 -> 在非 GUI 模式下运行 JMeter

-t 测试文件 -> 要运行的 JMeter 测试脚本文件

-l jtl 文件 -> 记录结果的文件

-r 远程执行 -> 启动远程服务

-H 代理主机 -> 设置 JMeter 使用的代理主机

-P 代理端口 -> 设置 JMeter 使用的代理主机的端口号

-j 日志文件->设置 JMeter 日志文件的名称

实例：

```
JMeter -n -t my_test.jmx -l log.jtl -H my.proxy.server -P 8000
```

执行步骤：

JMeter 默认去当前目录寻找脚本文件，并把日志记录在当前目录。比如你在 C:\tools\apache-jmeter-2.11\bin 目录下执行以上命令，JMeter 会去该目录下寻找 test.jmx 脚本并把执行结果放在该目录。如果你的脚本在其他目录，而且想要把执行结果放在另外文件夹，可以使用绝对路径告诉 JMeter。

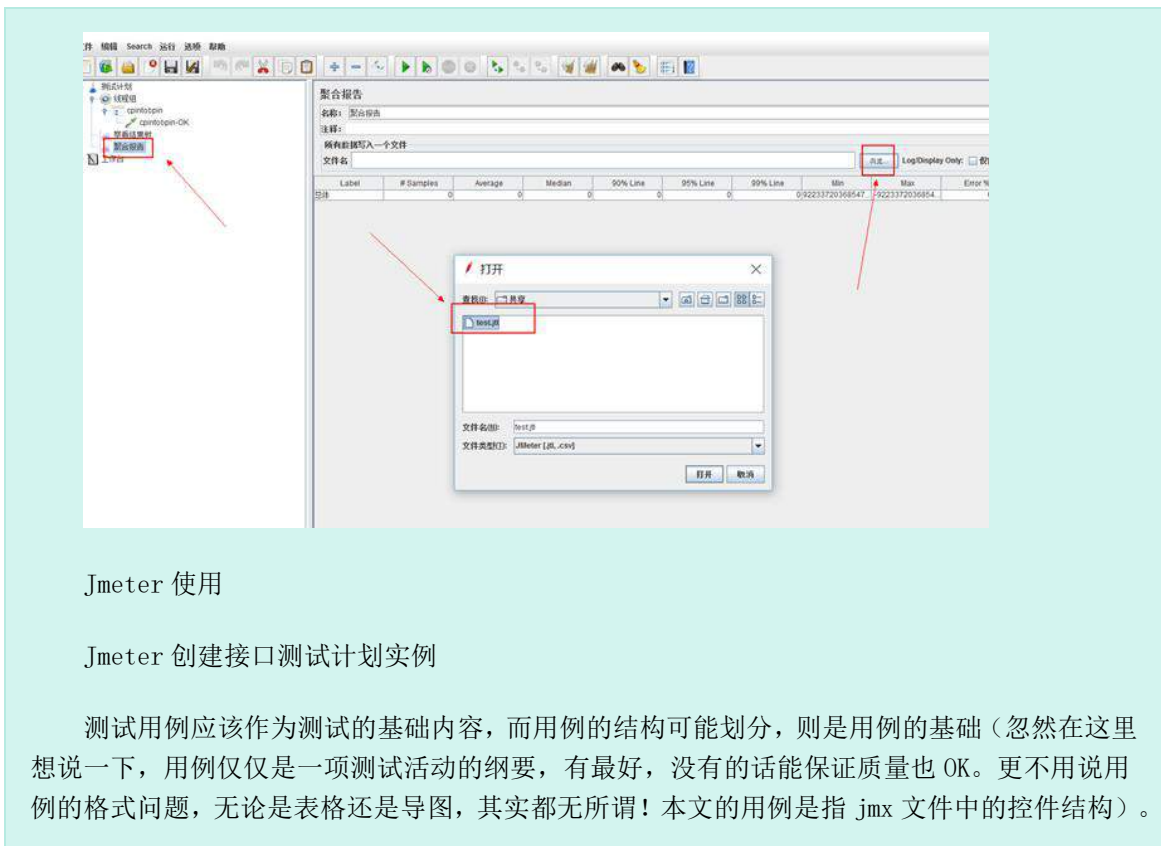
执行结果查看：

GUI 界面打开聚合报告

在 GUI 界面创建一个聚合报告

聚合报告界面点击浏览，选中生成的.jtl 文件，打开







The screenshot shows the JMeter GUI for a file named 'LoginController.jmx'. The tree structure is as follows:

- LoginController (Module Name, corresponding to the interface class) - Annotated with '模块名称, 对应接口类' and a red arrow.
  - BeanShell PreProcessor (根据系统设置文件路径)
    - 接口路径信息
    - 连接ddwdh数据库
    - 已注册账号配置
    - 未注册账号配置
    - 察看结果树
    - 聚合报告
    - 图形结果
    - jp@gc - Transactions per Second (tps值)
    - jp@gc - Response Times vs Threads
  - LoginController (Module Name, corresponding to the interface class) - Annotated with '模块名称, 对应接口类' and a red arrow.
    - TPS限制 (120)
    - 获取测试数据
      - one\_ddh\_get\_password
      - one\_ddh\_get\_ddhid - Annotated with '获取线程对应的测试数据' and a red arrow.
    - 未注册用户发送验证码
      - BeanShell PreProcessor
        - no\_register\_sendsms-OK
        - no\_register\_sendsms-registered
        - no\_register\_sendsms-60s
        - no\_register\_sendsms-errorkey
        - no\_register\_sendsms-errorpara
      - 验证码开户 (测试服未开通注册, 部分未调试)
      - 保存个人信息
      - 登录
      - 更新用户经纬度
      - 注销登录
      - 已注册用户发送验证码
      - 获得兴趣列表
      - 获得省市三级数据-成功
      - 上传头像
      - 找回密码
      - 修改密码

Annotations on the right side of the image:

- '数据准备与结果查看' (Data preparation and result viewing) with a red arrow pointing to the '察看结果树' and '聚合报告' items.
- '接口名称, 对应接口方法' (Interface name, corresponding to the interface method) with a red arrow pointing to the '未注册用户发送验证码' folder.

Below the screenshot, a list of bullet points explains the structure:

- 模块名称 (测试计划): 每个模块独立划分为一个 jmx 文件 (例如登陆模块), 最好与接口类一一对应。对应的服务器信息, 数据库信息等可存在这里。
- 数据准备: 用于测试数据的准备 (例如账号信息)。
- 结果查看: 用于放置需要查看结果的控件 (例如结果树)。
- 线程组: 所有的接口测试用例放在线程组下, 集中定义线程等信息
- 获取线程对应测试数据: 用于获取针对独立线程的测试数据, 例如在数据准备里面获得了账号信息, 在这里根据账号信息去数据库获取对应的名称, ID 等信息。
- 请求名称: 用简单控制器为文件夹, 内有不同的请求。简单控制器为一个独立的接口, 不同请求对应不同的代码路径 (例如成功请求, 失败请求等)。建议请求名称最好用英文形式, 否则后期持续集成或许会出现问题 (no zuo no die! )。

- 在每条请求内放置正则匹配（用于应对需要返回值作为下次请求的参数的情況）以及断言。

## 15、请你进行测试：前端下拉框实现，测试下拉框定位方式

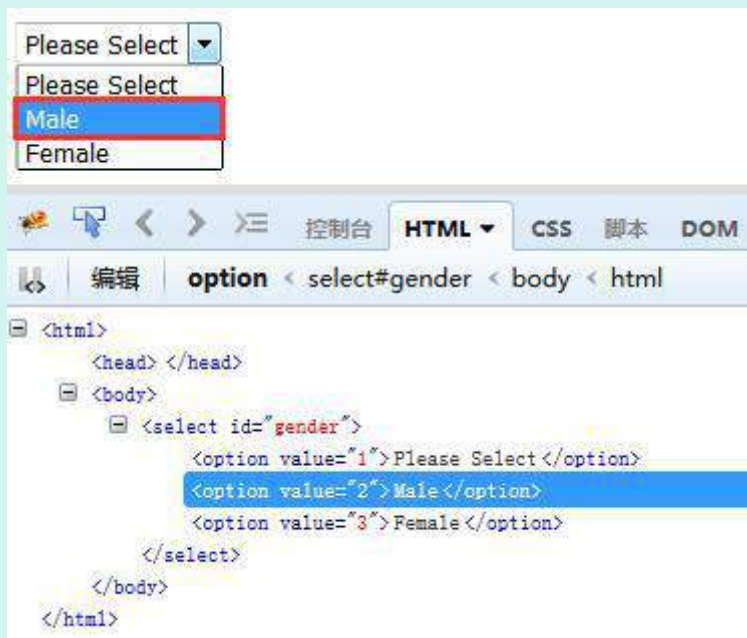
考点：测试

参考回答：

Selenium+Python 自动化测试对下拉菜单的定位

1. 通过 selenium.webdriver.support.ui 的 Select 进行定位

下拉菜单如下图：



定位代码：

```
from selenium.webdriver.support.ui import Select
```

```
# 通过 index 进行选择
```

```
Select(driver.find_element_by_id("gender")).select_by_index(1)
```

```
# 通过 value 进行选择
```

```
Select(driver.find_element_by_id("gender")).select_by_value("2")
```

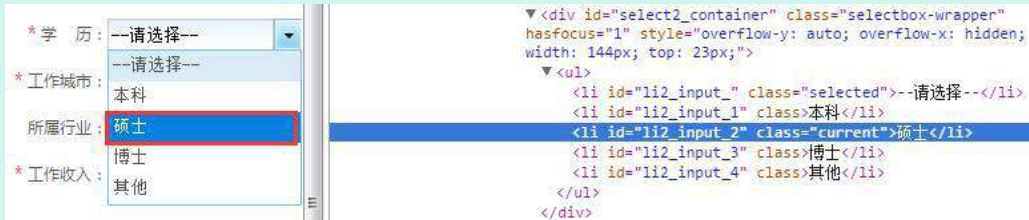
```
# 通过选项文字进行选择
```

```
Select(driver.find_element_by_id("gender")).select_by_visible_text("Male")
```

注：Select only works on `<select>` elements (Select 只对`<select>`标签的下拉菜单有效)。

## 2. 定位非`<select>`标签的下拉菜单

非`<select>`标签的下拉菜单如下图所示：



定位非`<select>`标签的下拉菜单中的选项，需要两个步骤，先定位到下拉菜单，再对其中的选项进行定位。

定位代码：

# 先定位到下拉菜单

```
drop_down = driver.find_element_by_css_selector("div#select2_container > ul")
```

# 再对下拉菜单中的选项进行选择

```
drop_down.find_element_by_id("li2_input_2").click()
```

注：也可以用此方法定位`<select>`标签的下拉菜单。

## 16、请你来聊一聊 appium 断言

考点：测试

参考回答：

appium-unittest 单元测试框架中，TestCase 类提供了一些方法来检查并报告故障，如下图：

Method	Checks that	New in
<code>assertEqual(a, b)</code>	<code>a == b</code>	
<code>assertNotEqual(a, b)</code>	<code>a != b</code>	
<code>assertTrue(x)</code>	<code>bool(x)</code> is True	
<code>assertFalse(x)</code>	<code>bool(x)</code> is False	
<code>assertIs(a, b)</code>	<code>a is b</code>	2.7
<code>assertIsNot(a, b)</code>	<code>a is not b</code>	2.7
<code>assertIsNone(x)</code>	<code>x is None</code>	2.7
<code>assertIsNotNone(x)</code>	<code>x is not None</code>	2.7
<code>assertIn(a, b)</code>	<code>a in b</code>	2.7
<code>assertNotIn(a, b)</code>	<code>a not in b</code>	2.7
<code>assertIsInstance(a, b)</code>	<code>isinstance(a, b)</code>	2.7
<code>assertNotIsInstance(a, b)</code>	<code>not isinstance(a, b)</code>	2.7

上面所提供的断言方法（`assertRaises()`，`assertRaisesRegexp()`除外）接收 `msg` 参数，如果指定，将体作为失败的错误信息。

```
try:

    num = input("Enter a number:")

    assert (num == 10), "The number is not 10!"

except AssertionError, msg:

    print msg

    print ("Sadly, num not equals to 10")
```

在上面的程序中，运行到的 `python` 的异常与断言。通过 `raw_input()` 方法要求用户输入一个数字，通过 `assert` 判断用户输入的 `num` 是否等于 10；通过 `python` 的 `AssertionError` 类型的异常来实捕获这个异常，`msg` 接收异常信息并打印，注意，`msg` 所结构的异常信息是我们自定义的（"The number is not10!"）。

- `assertEqual(first, second, msg=None)`：判断 `first` 和 `second` 的值是否相等，如果不相等则测试失败，`msg` 用于定义失败后所抛出的异常信息。

- `assertNotEqual(first, second, msg=None)`：测试 `first` 和 `second` 不相等，如果相等，则测试失败。

- `assertTure(expr, msg=None)`、`assertFalse(expr, msg=None)`：测试 `expr` 为 `Ture`（或为 `False`）

以下为 `python 2.7` 版新增的断言方法：



•`assertIs(first, second, msg=None)`、`assertIsNot(first, second, msg=None)`：测试的 `first` 和 `second` 是（或 不是）相同的对象。

•`assertIsNone(expr, msg=None)`、`assertIsNotNone(expr, msg=None)`：测试 `expr` 是（或 不是）为 `None`

•`assertIn(first, second, msg=None)`、`assertNotIn(first, second, msg=None)`：测试 `first` 是（或不是）在 `second` 中。`second` 包含是否包含 `first` 。

•`assertIsInstance(obj, cls, msg=None)`、`assertNotIsInstance(obj, cls, msg=None)`：测试 `obj` 不（或 不是）`cls` 的一个实例。（`obj` 和 `cls` 可以是一个类或元组），要检查他们的类型使用 `assertIs(type(obj), cls)`。

## 17、请你来说一下购物车的测试用例

考点：测试

参考回答：

界面测试

- 界面布局、排版是否合理；文字是否显示清晰；不同卖家的商品是否区分明显。

2. 功能测试

未登录时：

- 将商品加入购物车，页面跳转到登录页面，登录成功后购物车数量增加；
- 点击购物车菜单，页面跳转到登录页面。

登录后：

- 所有链接是否跳转正确；
- 商品是否可以成功加入购物车；
- 购物车商品总数是否有限制；
- 商品总数是否正确；
- 全选功能是否好用；
- 删除功能是否好用；
- 填写委托单功能是否好用；
- 委托单中填写的价格是否正确显示；



- 价格总计是否正确；
- 商品文字太长时是否显示完整；
- 店铺名字太长时是否显示完整；
- 创新券商品是否打标；
- 购物车中下架的商品是否有特殊标识；
- 新加入购物车商品排序（添加购物车中存在店铺的商品和购物车中不存在店铺的商品）；
- 是否支持 TAB、ENTER 等快捷键；
- 商品删除后商品总数是否减少；
- 购物车结算功能是否好用。

### 3. 兼容性测试

- 不同浏览器测试。

### 4. 易用性测试

- 删除功能是否有提示；是否有回到顶部的功能；商品过多时结算按钮是否可以浮动显示。

### 5. 性能测试

- 压力测试；并发测试。

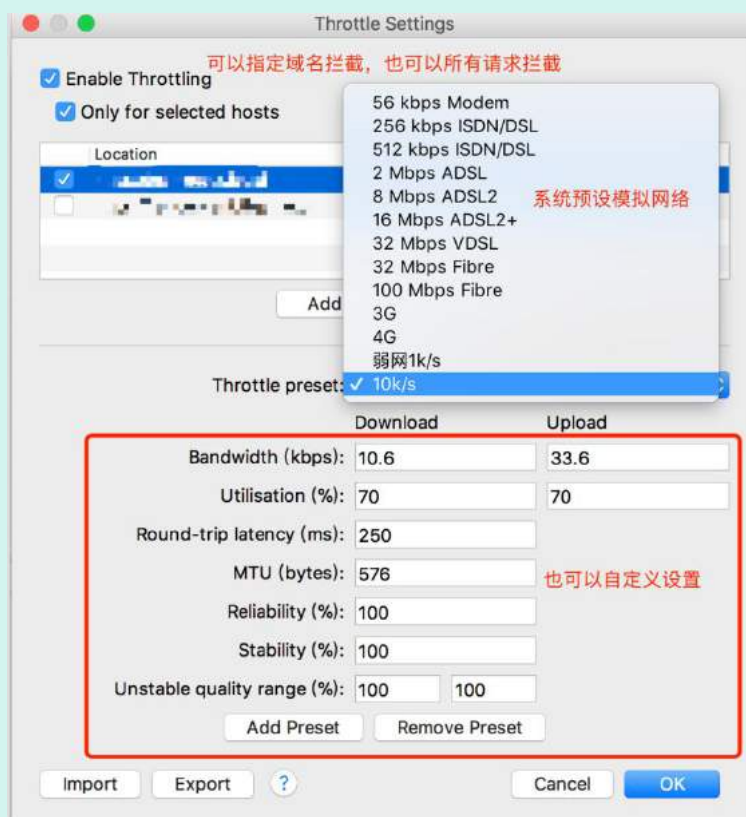
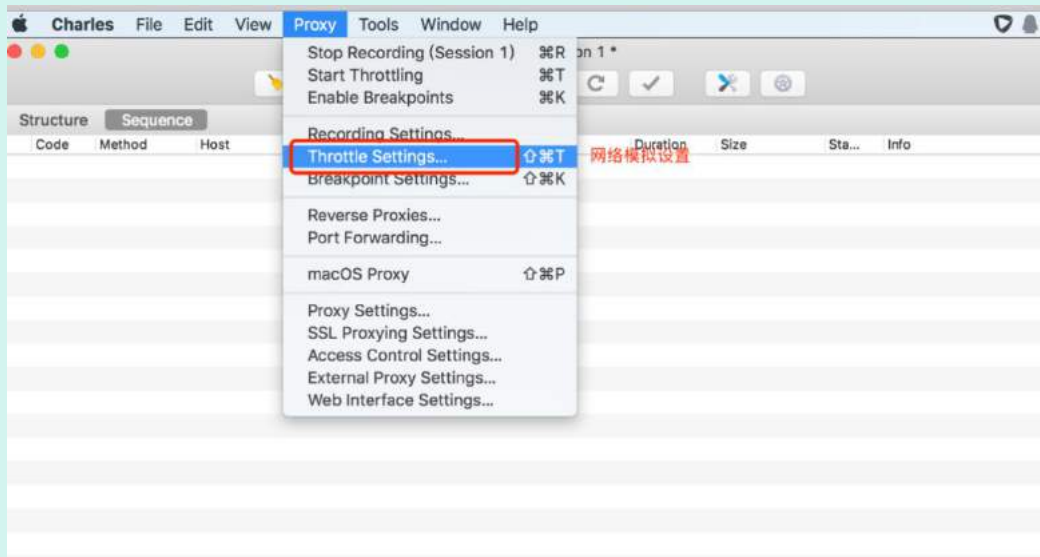
## 18、请你进行一下弱网模拟

考点：测试

参考回答：

方法一：charles 弱网模拟





配置参数解析：

bandwidth —— 带宽，即上行、下行数据传输速度

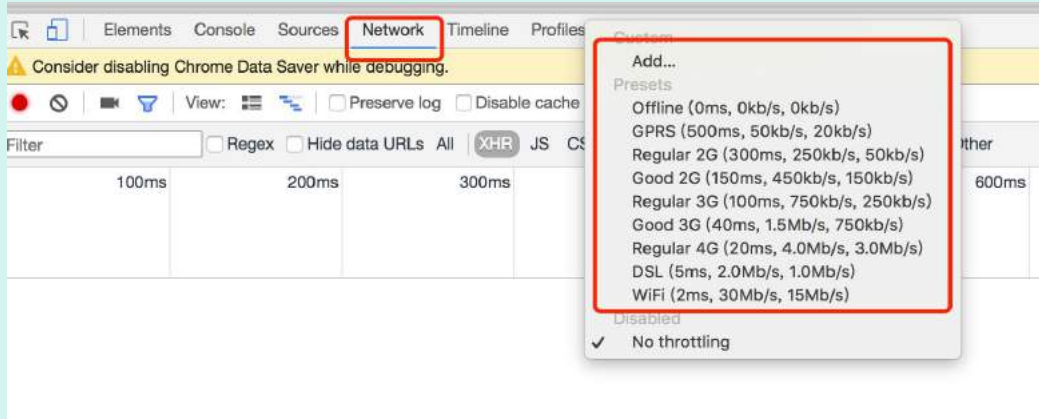
utilisation —— 带宽可用率，大部分 modern 是 100%

round-trip latency —— 第一个请求的时延，单位是 ms。

MTU —— 最大传输单元，即 TCP 包的最大 size，可以更真实模拟 TCP 层，每次传输的分包情况。

Reliability —— 指连接的可靠性。这里指的是 10kb 的可靠率。用于模拟网络不稳定。

Stability —— 连接稳定性，也会影响带宽可用性。用于模拟移动网络，移动网络连接一般不可靠。



使用 chrome 的 webview 调试工具，缺点是只适用于 web 页面的弱网模拟。

方法二：chrome 的 webview 调试工具弱网模拟

使用 chrome 的 webview 调试工具，缺点是只适用于 web 页面的弱网模拟。

具体步骤：

(1) 应用打开 webview 调试功能，具体如下：

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.KITKAT) {  
  
    WebView.setWebContentsDebuggingEnabled(true);  
  
}
```

(2) 手机链接电脑，运行 APP，进入具体 H5 页面；

(3) chrome 的 DevTools 中打开 Webview：进入 chrome://inspect/#devices，会显示已经连接设备，选中待调试 webview 的 inspect

network 页面，No throttling 下拉框，可以进行网络模拟。

方法三：iOS 手机自带 Network Link Conditioner 弱网模拟

iPhone 手机打开开发者选项，具体参考：

设置-开发者选项 > Network Link Conditioner 入口。

系统已经内置常见网络配置，也可以增加自定义配置。

具体配置参数：

in Bandwidth 下行带宽，即下行网络速度

In packet loss 下行丢包率

in delay 下行延迟，单位 ms

out bandwidth 上行带宽

out packet loss 上行丢包率

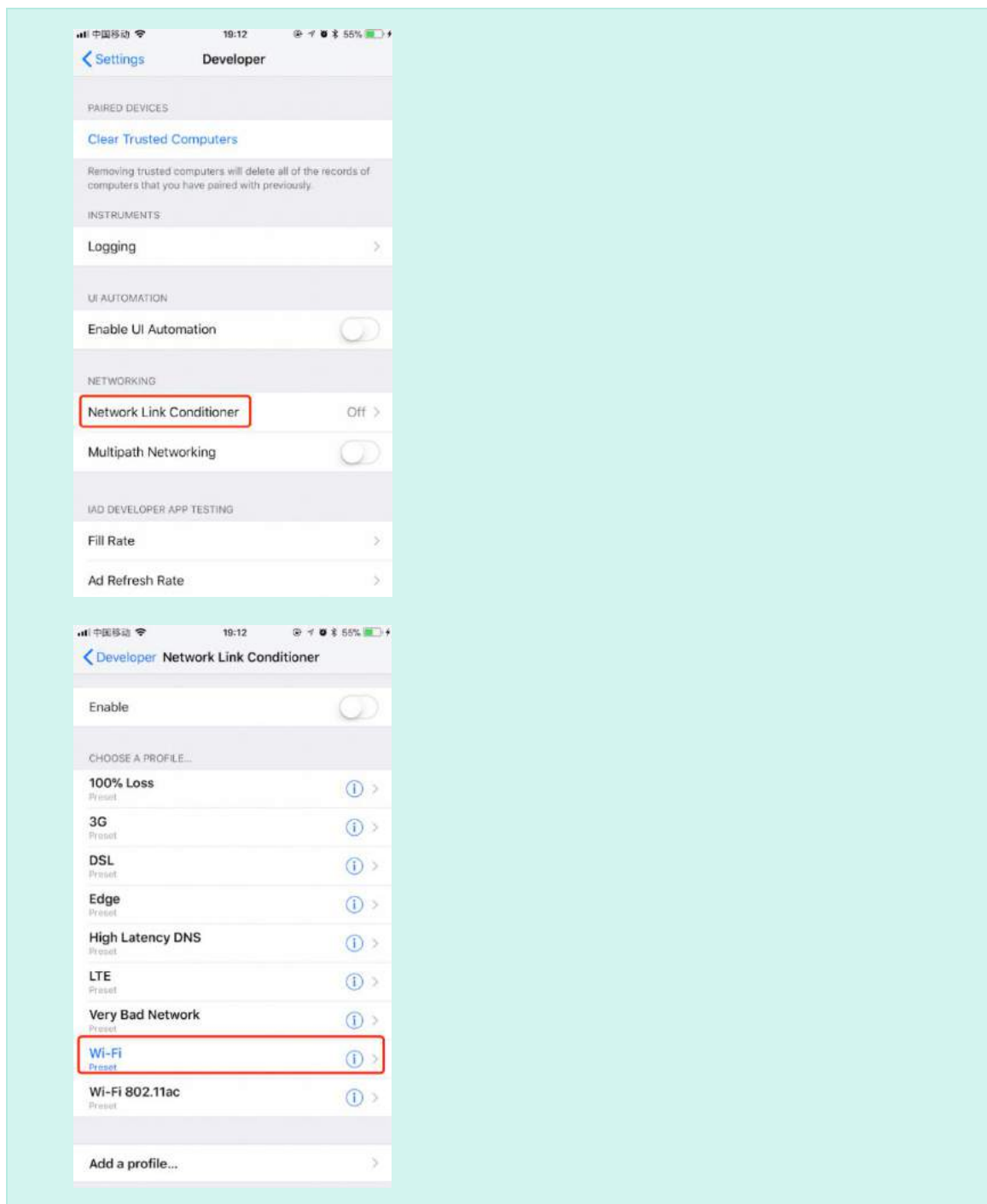
out delay 上行延迟

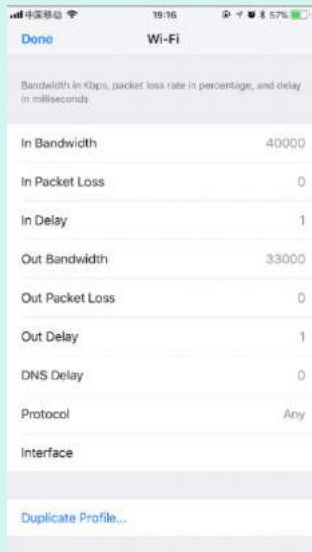
DNS delay DNS 解析延迟

protocol 支持 Any, IPV4、IPV6

interface 支持 Any, WI-Fi, cellular (蜂窝网)







Wi-Fi	
Bandwidth in Kbps, packet loss rate in percentage, and delay in milliseconds	
In Bandwidth	40000
In Packet Loss	0
In Delay	1
Out Bandwidth	33000
Out Packet Loss	0
Out Delay	1
DNS Delay	0
Protocol	Any
Interface	
Duplicate Profile...	

19、你写的测试程序是怎麼样的，你写过前端、后端程序吗？

考察点：编程能力

参考回答：

开发测试驱动程序一般分为 4 步：

1，指出需要的新特性。可以记录下来，然后为其编写一个测试。

2，编写特性的概要代码，这样程序就可以运行而没有任何语法等方面的错误，但是测试会失败。看到测试失败是很重要的，这样就能确定测试可以失败。如果测试代码中出现了错误，那么就有可能出现任何情况，测试都会成功，这样等于没测试任何东西。再强调一遍：在试图测试成功之前，先要看到它失败。

3，为特性的概要编写虚设代码，能满足测试要求就行。不用准确的实现功能，只要保证测试可以通过即可。这样一来就可以保证在开发的时候总是通过测试了，（除了第一次测试的时候）甚至在最初实现功能时亦是如此。

4，现在重写（或者重构）代码，这样它就会做自己应该做的事，从而保证测试一直成功。

在编码完成时，应该保证代码处于健康状态--不要遗留下任何测试失败。

写过前段程序。

20、请问你有没有写过测试脚本，怎么写的？

考察点：自动化测试



参考回答：

然后，撰写测试桩与驱动，白盒测试保证代码逻辑中循环和分支都能够走到，黑盒测试保证函数和首先，代码走查结合动态单步跟踪以及观察日志与文件输出，网络、CPU 状态。

功能脚本接口正确，输入输出符合设计预期。

对于异常处理，特别是变量的检查需要特别关注，变量在使用前都需要进行检查，是否为空？或者为 0？对于文件名和路径必须检查，确认文件是否存在，路径是否可达之后再行后续操作。另外，需要考虑所依赖的其他功能脚本以及二进制工具，这些功能性单元应该如何使用，调用后的返回会有哪些情况，对于正常和异常结果，脚本是否能够捕捉到并且作出正确的判断。

## 21、请问你有没有写过 web 测试，怎么写的？

考察点：web 测试

参考回答：

Web 测试主要从下面几个大方向考虑

功能测试，主要做链接测试，表单测试，cookies 测试，设计语言测试等

性能测试，考虑连接速度测试，以及负载测试，例如：Web 应用系统能允许多少个用户同时在线？如果超过了这个数量，会出现什么现象？Web 应用系统能否处理大量用户对同一个页面的请求？还有压力测试

可用性测试，比如导航测试，图形测试，内容测试，整体界面测试等

兼容性测试，市场上有很多不同的操作系统类型，最常见的有 Windows、Unix、Macintosh、Linux 等。Web 应用系统的最终用户究竟使用哪一种操作系统，取决于用户系统的配置。这样，就可能会发生兼容性问题，同一个应用可能在某些操作系统下能正常运行，但在另外的操作系统下可能会运行失败。因此，在 Web 系统发布之前，需要在各种操作系统下对 Web 系统进行兼容性测试。

安全性测试，

(1) 现在的 Web 应用系统基本采用先注册，后登陆的方式。因此，必须测试有效和无效的用户名和密码，要注意到是否大小写敏感，可以试多少次的限制，是否可以不登陆而直接浏览某个页面等。

(2) Web 应用系统是否有超时的限制，也就是说，用户登陆后在一定时间内（例如 15 分钟）没有点击任何页面，是否需要重新登陆才能正常使用。

(3) 为了保证 Web 应用系统的安全性，日志文件是至关重要的。需要测试相关信息是否写进了日志文件、是否可追踪。

(4) 当使用了安全套接字时，还要测试加密是否正确，检查信息的完整性。



(5) 服务器端的脚本常常构成安全漏洞，这些漏洞又常常被黑客利用。所以，还要测试没有经过授权，就不能在服务器端放置和编辑脚本的问题。

## 22、请问测试路由器怎么测，用命令行还是界面？

考察点：路由器测试

参考回答：

可以采用 lperf 这个命令，

Lperf 是一个网络性能测试工具，可以测量最大 tcp 和 udp 带宽，具有多种参数和特性，可以记录带宽，延迟抖动，数据包丢失，通过这些信息可以发现网络问题，检查网络质量，定位网络瓶颈。

iperf 的使用非常简单，测试的原理是在 wan 口连接一台 PC 机，在 LAN 口连接一台 PC，两边分别运行 iperf 服务端和客户端模式，用来测量 LAN->WAN 和 WAN->LAN 性能。具体命令如下：

服务端：iperf -s -w 1m

客户端：iperf -c <server ip> -w 1m -t 20 -P 10

含义是 TCP window size 为 1MByte，测试时间是 20s，线程是 10。

## 23、请你回答一下如何测试手机开机键？

考察点：测试用例情景设计

参考回答：

功能测试：

按下开机键，屏幕能否亮起

性能测试：

按下开机键，屏幕能否在规定时间内亮起

压力测试

连续多次按下开机键，观察屏幕是否能一直亮起，到多久时间失灵

健壮性测试

给定一个中了病毒的手机或者是淘汰许久的老机子，长按开机键观察屏幕能否亮起

可靠性测试

连续按下开机键有限次数，比如 1 万次，记录屏幕未亮起的次数





可用性测试

开机键按下费不费力，开机键的形状设计是否贴合手指，开机键的位置设计是否方便

24、请问你遇到过哪些印象深刻的 bug，接口测试出现 bug 的原因有哪些？

考察点：接口测试

参考回答：

面试官询问遇到过哪些印象深刻的 bug，其实它并不关心你描述的这个 bug 是否真的有价值，或有多曲折离奇？他只是：了解你平时工作中的测试能力

所以，这就要求的你平时工作中遇到 bug 时试着自己去定位，定位 bug 的过程远比你的单纯的执行测试用例有“价值”（自我技能提高的价值），在定位 bug 的过程中你需要掌握和运用更多知识。

另外，建议你平时养成总结的好习惯，发现的 bug，开发解决了，最好问问他原因以及解决的方法，这样再遇到类似问题时，自己也可以试着定位解决。遇到难解决的 bug，也可以把最终的解决过程记录下来。（这不是就有素材了）

所以，建议你平时可以主动要求去分享一些自己工作中用到或学习的技术。或者多去参加集体活动，加强自己的表达能力。From: 虫师

接口测试常见的 bug 有以下几个：

特殊值处理不当导致程序异常退出或者崩溃

类型边界溢出，导致数据独处和写入不一致

取值边界外未返回正确的错误信息

权限未处理，可以访问其他用户的信息

逻辑校验不完善，可以利用漏洞获取非正当利益

状态处理不当，导致逻辑出现错误

数组类型 item 个数为 0 或者 item 重复时程序异常退出

25、你在做项目中有做过压力测试吗，怎么做

考察点：压力测试

参考回答：

1、首先对要测试的系统进行分析，明确需要对那一部分做压力测试，比如秒杀，支付



## 2、如何对这些测试点进行施压

第一种方式可以通过写脚本产生压力机器人对服务器进行发包收报操作

第二点借助一些压力测试工具比如 Jmeter, LoadRunner

## 3、如何对这些测试点进行正确的施压

需要用压力测试工具或者其他方法录制脚本，模拟用户的操作

## 4、对测试点设计多大的压力比较合适？

需要明确压力测试限制的数量，即用户并发量

## 5、测试结束后如何通过这些数据来定位性能问题

通过测试可以得到吞吐量，平均响应时间等数据，这个数据的背后是整个后台处理逻辑综合作用的结果，这时候就可以先关注系统的 CPU，内存，然后对比吞吐量，平均响应时间达到瓶颈时这些数据的情况，然后就能确认性能问题是系统的哪一块造成的

## 26、请问你在项目中关于功能测试和接口测试是怎么做的

考察点：功能/接口测试

参考回答

功能测试：

首先制定测试计划，然后进行测试设计，将在测试计划阶段指定的测试活动分解，进而细化，为若干个可执行程序的子测试过程，然后执行测试，按照测试计划使用测试用例对待测项目进行逐一的，详细的排查分析评估，最后对测试结果进行统计和分析，

接口测试：

什么是接口（API）

API 全称 Application Programming Interface，这里面我们其实不用去关注 AP，只需要 I 上就可以。一个 API 就是一个 Interface。我们无时无刻不在使用 interfaces。我们乘坐电梯里面的按钮是一个 interface。我们开车一个踩油门它也是一个 interface。我们计算机操作系统也是有很多的接口。（这是目前个人找到比较好理解的一段解释）

接口就是一个位于复杂系统之上并且能简化你的任务，它就像一个中间人让你不需要了解详细的所有细节。那我们今天要讲的 Web API 就是这么一类东西。像谷歌搜索系统，它提供了搜索接口，简化了你的搜索任务。再像用户登录页面，我们只需要调用我们的登录接口，我们就可以达到登录系统的目的。

现在市面上有非常多种风格的 Web API，目前最流行的是也容易访问的一种风格是 REST 或者叫 RESTful 风格的 API。从现在开始，以下我提到的所有 API 都是指 RESTful 风格的 API。



### 什么是接口测试和为什么要做接口测试

接口测试是测试系统组件间接口的一种测试。接口测试主要用于检测外部系统与系统之间以及内部各个子系统之间的交互点。测试的重点是要检查数据的交换，传递和控制管理过程，以及系统间的相互逻辑依赖关系等。

现在很多系统前后端架构是分离的，从安全层面来说，只依赖前端进行限制已经完全不能满足系统的安全要求（绕过前端太容易了），需要后端同样进行控制，在这种情况下就需要从接口层面进行验证。

如今系统越来越复杂，传统的靠前端测试已经大大降低了效率，而且现在我们都推崇测试前移，希望测试能更早的介入测试，那接口测试就是一种及早介入的方式。例如传统测试，你是不是得等前后端都完成你才能进行测试，才能进行自动化代码编写。而如果是接口测试，只需要前后端定义好接口，那这时自动化就可以介入编写接口自动化测试代码，手工测试只需要后端代码完成就可以介入测试后端逻辑而不用等待前端工作完成。

### 接口测试的策略

接口测试也是属于功能测试，所以跟我们以往的功能测试流程并没有太大区别，测试流程依旧是：1. 测试接口文档（需求文档） 2. 根据接口文档编写测试用例（用例编写完全可以按照以往规则来编写，例如等价类划分，边界值等设计方法） 3. 执行测试，查看不同的参数请求，接口的返回的数据是否达到预期。

### 27、请问你有用过什么测试工具吗，用过哪些？

考察点：测试工具

参考回答：

自动化测试工具用过 selenium 和 appium

性能测试工具有用过 Jmeter

### 28、请你设计一个微信朋友圈点赞的测试用例

考察点：测试用例设计

参考回答：

功能测试：

点赞某条朋友圈，验证是否成功

接口测试：

点赞朋友圈，验证朋友能否收到提示信息

性能测试



点赞朋友圈，是否在规定时间内显示结果，是否在规定时间内在朋友手机上进行提示

兼容性测试

在不同的终端比如 ipad, 手机上点赞朋友圈，验证是否成功

29、请问如果用户点击微博的关注图标但是 app 上面没有反应，应该怎么排查这个问题

考察点：问题排查

参考回答：

首先 1. 在 Eclipse Devices 窗口，选中 app 对应的包名，然后点击 debug 图标（绿色的小虫子），然后切换到 Debug 视图。

2. 切换视图之后，可以看到 debug 下，app 的线程列表。

3. 对于 main 线程（第一个线程），选中，并将其挂起 Suspend。

4. 然后我们就可以看到，Suspend 之后，main 线程卡住的位置。

30、在做测试的过程中，假如前端和后端吵起来了都在踢皮球觉得对方该改代码，你怎么办？

参考回答：

此时就应该找技术领导拍板或 leader 们基于安全性、性能、可测试性、可维护性讨论敲定一个解决方案，做到开发环境方便开发，线上环境少配置

少依赖、少出错机会。

31、如果广东用户头条 app 刷不出东西了，你应该怎么排查问题

参考回答：

1、检查网络连接是否稳定，更换网络尝试

2、更新头条版本尝试

3、清除 app 缓存，应用数据

32、请你说一下能不能用机器学习去进行自动化测试，如何监控异常流量，如果是脉冲呢，如何和正常流量作区分

参考回答：



如何用机器学习去进行自动化测试,就是让自动化测试变得更加智能,在自动化测试过程中,当测试功能模块越来越多,没有太多的时间去全部覆盖,我们可以采用归纳学习的方式,基于自动化测试的执行结果或者手工测试执行的结果为数据输入,然后基于一定的模型(例如:以通过率和模块的重要率计算的平均值)得出测试推荐模块,或者当要执行一个功能模块时,基于历史数据和模型(bug出现的错误相关性,功能相关性等)计算出与该功能模块相关性最大模块,并推荐测试。

如何监控异常流量

1、抓包

```
tcpdump -i eth0 -w server.cap
```

对包文件使用第三方工具如: wireshark 做分析

2、iftop

```
yum install iftop
```

3、iptraf

```
yum install iptraf -y 或 yum install iptraf-ng -y
```

启动命令 ifptraf-ng

33、请问如何将大量日志的异常记录或错误揪出来

参考回答:

略

34、请问如何对登录界面进行测试

考察点: 测试用例设计

黑盒测试方法

输入正确用户名和密码, 验证是否登陆成功

输入正确的用户名和错误的密码, 验证是否登陆失败并且提示信息正确

输入未注册的用户名和任意的密码, 验证是否登陆失败并且提示信息正确

用户名和密码都为空, 验证是否登陆失败并且提示信息正确

用户名和密码两者之一为空

若启用了验证码, 输入正确的用户名密码验证码是否能登陆成功



输入正确用户名和密码，错误的验证码，能否登陆成功并且提示信息正确

用户名和密码是否大小写敏感

页面上的密码框是否加密显示

后台系统第一次创建的用户重新登录时是否提示修改密码

忘记用户名和忘记密码的功能是否可用

前端功能是否根据要求限制用户名和密码的长度

点击验证码图片是否可以更换验证码，更换后的验证码是否可用

刷新页面是否会刷新验证码

如果验证码具有时效性，分别验证时效内和时效外验证码的有效性

用户登录成功但是会话超时后是否重定向到用户登录界面

不同级别的用户登录系统后的权限是否正确

页面默认定位焦点是否定位到用户名输入框中

快捷键 tab 和回车键是否可以正常使用

非功能性需求，从安全，性能，兼容三个方面

安全：

用户密码后台存储是否加密

用户密码在网络传输过程中是否加密

密码是否具有有效期，密码有效期到期后是否提示修改密码

不登陆的时候直接在浏览框中输入登录界面后的 url 地址，是否会重新定位到登陆界面

密码输入框是否不支持复制粘贴

页面密码输入框中输入的密码是否可以在页面源码模式下被查看

用户名和密码输入框中输入 xss 跨站脚本攻击字符串验证系统的行为是否被篡改

连续多次登陆失败后系统是否会阻止用户后续的尝试

统一用户在同一终端的多种不同浏览器上登陆，验证登录功能的互斥性是否符合设计预期

同一用户先后在不同终端的浏览器上登陆用户名和密码输入框中输入典型的 sql 注入攻击字符串验证系统的返回页面



，验证登陆是否有互斥性

性能测试：

单用户登陆的响应界面是否符合预期

单用户登陆时后台请求数量是否过多

高并发场景下用户登录的响应界面是否符合预期

高并发场景下服务端的监控指标是否符合预期

高集合点并发场景下是否存在资源死锁和不合理的资源等待

长时间大量用户连续登录和登出，服务器端是否存在内存泄漏

兼容性测试：

不同浏览器下验证登陆功能的页面显示和功能正确性

相同浏览器的不同版本下验证登陆功能的页面显示和功能正确性

不同终端的不同浏览器下验证登陆功能的页面显示和功能正确性

不同分辨率下……

补充：弱网测试

网络切换和网络延迟时登陆界面是否正常

是否支持第三方登陆

是否可记住密码，记住的密码是否加密

### 35、请你说一说当前工作中涉及的测试问题(测试流程和测试性能)

考察点：测试实践

参考回答：

在测试性能中，时常会出现脚本回访卡住的问题，原因有以下几种：

1、 runtimesetting 中的 continue error 没有勾选

2、录制的脚本中存在冗余的代码部分，需要对脚本进行优化，去除冗余的部分（优化脚本）

例如：在用 FireFox 录制脚本时，脚本中会产生一个叫





”

Url=http://download.cdn.mozilla.net/pub/firefox/releases/43.0.1/update/win32/zh-CN/firefox-43.0.1.complete.mar”, “Referer=”, ENDITEM, ” 这样的代码（该代码出现的问题不止一处，在查找时一定要注意。），这是因为采用 firefox 浏览器录制时产生的压缩文件，在脚本回放时卡住的原因正是因为这个（建议：能采用 IE 录制尽量用 IE 浏览器）

解决办法：注释掉或者删除掉该段代码即可， 关联问题：在用 loadrunner 自带对比工具对比脚本后 找到需要关联的动态值。在关联后回放脚本时报错 HTTP-status code 417 (exception failed) 错误时，产生的原因如下：

1、脚本中还存在没有关联或者关联失败的动态值，利用 lr 自带对比工具仔细对比

2、脚本中的动态值被做了加密策略，仔细查看脚本中动态值的部分，看看动态值是否被做了安全策略（随机生成或者打乱动态值顺序、在动态值中加入了特殊符号），由于在 tree-response 中的动态值是未被加密的状态，在 client 向 server 发送请求时，client 的动态值发给服务器，这时服务器的动态值已经被做了参数化，所以服务器不认准 client 向服务器发送的动态值。

解决办法：去掉动态值的安全策略即可（JVM 参数）

### 36、请你说一说洗牌问题的思路并手写代码，并设计测试用例

考察点：编程，设计测试用例

参考回答：

洗牌问题：有个长度为  $2n$  的数组  $\{a_1, a_2, a_3, \dots, a_n, b_1, b_2, b_3, \dots, b_n\}$ ，希望排序后  $\{a_1, b_1, a_2, b_2, \dots, a_n, b_n\}$ ，请考虑有无时间复杂度  $O(n)$ ，空间复杂度  $O(1)$  的解法。

```
void PerfectShuffle(int *A, int n) {  
  
    if(n <= 1) {  
  
        return;  
  
    } //if  
  
    //  
  
    int size = 2*n;  
  
    int index, count;  
  
    for(int i = n; i < size; ++i) {  
  
        // 交换个数
```



```
        count = n - (i - n) - 1;

        // 待交换

        index = i;

        for(int j = 1;j <= count;++j){

            swap(A[index],A[i-j]);

            index = i - j;

        }//for

    }//for

}
```

```
};
```

可以就数组的类型，可以是 int 型的，浮点型的，还可以是大数类型，负数，进行测试。

### 37、请你测试一下游戏中英雄的技能

参考回答：

测试的设计都是通用的，首先功能测试看功能有没有实现，然后再对性能、压力、容量、健壮性、安全性、可靠性、恢复性、备份、协议、兼容性、可用性、配置、GUI 这些非功能测试去思考。具体答案这里不再赘述

### 38、请你回答一下性能测试有哪些指标，对一个登录功能做性能测试，有哪些指标，怎么测出可同时处理的最大请求数量

考察点：性能测试

参考回答

性能测试常用指标：

从外部看，主要有

- 1、吞吐量：每秒钟系统能够处理的请求数，任务数
- 2、响应时间：服务处理一个请求或一个任务的耗时
- 3、错误率：一批请求中结果出错的请求所占比例



从服务器的角度看，性能测试关注 CPU，内存，服务器负载，网络，磁盘 I O

对登录功能做性能测试

单用户登陆的响应界面是否符合预期

单用户登陆时后台请求数量是否过多

高并发场景下用户登录的响应界面是否符合预期

高并发场景下服务端的监控指标是否符合预期

高集合点并发场景下是否存在资源死锁和不合理的资源等待

长时间大量用户连续登录和登出，服务器端是否存在内存泄漏

怎么测出可同时处理的最大请求数量

可以采用性能测试工具(WeTest 服务器性能)，该工具是腾讯 wetest 团队出品，使用起来很简单方便，但测试功能相当强大，能提供 10w+以上的并发量，定位性能拐点，测出服务器模型最大并发

39、请问你有没有做过什么单元测试，怎么进行单元测试，对一个没有参数没有返回值但可能对全局变量有影响的怎么进行单元测试

考察点：单元测试

参考回答：

如何进行单元测试：

1、创建单元测试，该工具可以对任何类、接口、结构等实体中的字段、属性、构造函数、方法等进行单元测试。创建单元测试大致可以分为两类：

整体测试，整体测试是在类名称上右击鼠标，在下拉菜单中点击创建单元测试选项。这样就可以为整个类创建单元测试了，这时他会为整个类可以被测试的内容全部添加测试方法。开发人员直接在这些自动生成的测试方法中添加单元测试代码就可以了。

单独测试，如果只想单独对某个方法、属性、字段进行测试，则可以将鼠标焦点放在这个待测试的项目名称之上，然后点击鼠标右键，在右键菜单中选择创建单元测试选项。这样就可以单独为某个方法创建单元测试了。

运行单元测试

查看测试结果

编写单元测试代码



测试没有参数的函数，它可能还有别的输入，例如全局变量，成员变量，或调用子函数获得的输入（这个要使用工具才能做到），只要函数需读取的，都应该设定初始值，如果完全没有，没有输入也是一种输入，照样测试就是了。同样道理，输出也不仅仅是返回值，没有返回值还可能修改了全局变量什么的，这些也是要判断的输出。

#### 40、请问你有没有做过压力测试

考察点：压力测试

参考回答：

在软件工程中，压力测试是对系统不断施加压力的测试，是通过确定一个系统的瓶颈或者不能接收的性能点，来获得系统能提供的最大服务级别的测试。例如测试一个 Web 站点在大量的负荷下，何时系考察公司：网易

统的响应会退化或失败。网络游戏中也常用到这个词汇。

#### 41、对于有系统大量并发访问，你会如何做测试，有什么建议

考察点：测试设计

参考回答：

如何做高并发系统的测试，一般而言，整体的测试策略是：先针对部分系统进行性能测试及压力测试，得到各部分的峰值处理性能，再模拟整体流程测试，重点测试整体业务流程以及业务预期负荷，着重测试以下几点：

- 1、不同省份，不同运营商 CDN 节点性能，可采用典型压力测试方案
- 2、核心机房 BGP 网络带宽，此部分重点在于测试各运行商的 BGP 网络可靠性，实际速率，一般采用 smokeping, lxChariot 等工具
- 3、各类硬件设备性能，一般采用专业的网络设备测试工具
- 4、各类服务器并发性能，分布式处理能力，可采用压力测试方案工具
- 5、业务系统性能，采用业务系统压力测试方案
- 6、数据库处理性能，这部分需要结合业务系统进行测试，以获取核心业务场景下的数据库的 TPS/QPS，
- 7、如果有支付功能，需要进行支付渠道接口及分流测试，此部分相对而言可能是最大的瓶颈所在，此外还涉及备份方案，容灾方案，业务降级方案的测试。

## 三、语言基础

### 1、Java

#### 1、请你说一下多态

考点：java

参考回答：

什么是多态？

多态就是指程序中定义的引用变量所指向的具体类型和通过该引用变量发出的方法调用在编程时并不确定，而是在程序运行期间才确定，即一个引用变量到底会指向哪个类的实例对象，该引用变量发出的方法调用到底是哪个类中实现的方法，必须在由程序运行期间才能决定。因为在程序运行时才确定具体的类，这样，不用修改源程序代码，就可以让引用变量绑定到各种不同的类实现上，从而导致该引用调用的具体方法随之改变，即不修改程序代码就可以改变程序运行时所绑定的具体代码，让程序可以选择多个运行状态，这就是多态性。

Java 实现多态有三个必要条件：继承、重写、向上转型。

继承：在多态中必须存在有继承关系的子类 and 父类。

重写：子类对父类中某些方法进行重新定义，在调用这些方法时就会调用子类的方法。

向上转型：在多态中需要将子类的引用赋给父类对象，只有这样该引用才能够具备技能调用父类的方法和子类的方法。

Java 中有两种形式可以实现多态，继承和接口：

基于继承的实现机制主要表现在父类和继承该父类的一个或多个子类对某些方法的重写，多个子类对同一方法的重写可以表现出不同的行为。

基于接口的多态中，指向接口的引用必须是指定这实现了该接口的一个类的实例程序，在运行时，根据对象引用的实际类型来执行对应的方法。

#### 2、请问 Java 中接口与抽象类是否相同？

考点：java

参考回答：

不同，

抽象类是用来捕捉子类的通用特性的。它不能被实例化，只能被用作子类的超类。抽象类是被用来创建继承层级里子类的模板。



接口是抽象方法的集合。如果一个类实现了某个接口，那么它就继承了这个接口的抽象方法。

抽象类和接口的对比：

参数	抽象类	接口
默认的方法实现	它可以有默认的方法实现	接口完全是抽象的，不存在方法的实现
实现	子类使用 extends 关键字来继承抽象类。如果子类不是抽象类的话，它需要提供抽象类中所有声明的方法的实现。	子类使用关键字 implements 来实现接口。它需要提供接口中所有声明的方法的实现
构造器	抽象类可以有构造器	接口不能有构造器
与正常 Java 类的区别	除了你不能实例化抽象类之外，它和普通 Java 类没有任何区别	接口是完全不同的类型
访问修饰符	抽象方法可以有 public、protected 和 default 这些修饰符	接口方法默认修饰符是 public，不可以使用其它修饰符。
main 方法	抽象方法可以有 main 方法并且可以运行	接口没有 main 方法。
多继承	抽象方法可以继承一个类和实现多个接口	接口只可以继承一个或多个其它接口
速度	它比接口速度要快	接口是稍微有点慢的，因为它需要时间去寻找在类中实现的方法。

添加新方法	如果你往抽象类中添加新的方法，你可以给它提供默认的实现。因此你不需要改变你现在的代码。	如果你往接口中添加方法，那么你必须改变实现该接口的类。

什么时候使用抽象类和接口：

如果拥有一些方法并且想让它们中的一些有默认实现，使用抽象类。

如果想实现多重继承，必须使用接口。由于 Java 不支持多继承，子类不能够继承多个类，但可以实现多个接口。

如果基本功能在不断改变，那么就需要使用抽象类。如果不断改变基本功能并且使用接口，那么就需要改变所有实现了该接口的类。

### 3、请你说一下垃圾回收机制

考点：java

参考回答：

垃圾回收(Garbage Collection)是Java虚拟机(JVM)垃圾回收器提供了一种用于在空闲时间不定时回收无任何对象引用的对象占据的内存空间的一种机制。

引用：如果 Reference 类型的数据中存储的数值代表的是另外一块内存的起始地址，就称这块内存代表着一个引用。

(1) 强引用(Strong Reference)：如“Object obj = new Object ()”，这类引用是 Java 程序中最普遍的。只要强引用还存在，垃圾收集器就永远不会回收掉被引用的对象。

(2) 软引用(Soft Reference)：它用来描述一些可能还有用，但并非必须的对象。在系统内存不够用时，这类引用关联的对象将被垃圾收集器回收。JDK1.2 之后提供了 SoftReference 类来实现软引用。

(3) 弱引用(Weak Reference)：它也是用来描述非须对象的，但它的强度比软引用更弱些，被弱引用关联的对象只能生存到下一次垃圾收集发生之前。当垃圾收集器工作时，无论当前内存是否足够，都会回收掉只被弱引用关联的对象。在 JDK1.2 之后，提供了 WeakReference 类来实现弱引用。

(4) 虚引用(Phantom Reference)：最弱的一种引用关系，完全不会对其生存时间构成影响，也无法通过虚引用来取得一个对象实例。为一个对象设置虚引用关联的唯一目的是希望能在该对象被收集器回收时收到一个系统通知。JDK1.2 之后提供了 PhantomReference 类来实现虚引用。

垃圾：无任何对象引用的对象。





判断对象是否是垃圾的算法：

引用计数算法（Reference Counting Collector）、根搜索算法（Tracing Collector）：

回收：清理“垃圾”占用的内存空间而非对象本身。

Tracing 算法（Tracing Collector） 标记—清除算法：分为“标记”和“清除”两个阶段：首先标记出所需回收的对象，在标记完成后统一回收掉所有被标记的对象，它的标记过程其实就是前面的根搜索算法中判定垃圾对象的标记过程。

Compacting 算法（Compacting Collector） 标记—整理算法：标记的过程与标记—清除算法中的标记过程一样，但对标记后出的垃圾对象的处理情况有所不同，它不是直接对可回收对象进行清理，而是让所有的对象都向一端移动，然后直接清理掉端边界以外的内存。在基于 Compacting 算法的收集器的实现中，一般增加句柄和句柄表。

Copying 算法（Copying Collector）：将内存按容量分为大小相等的两块，每次只使用其中的一块（对象面），当这一块的内存用完了，就将还存活着的对象复制到另外一块内存上面（空闲面），然后再把已使用过的内存空间一次清理掉。

Adaptive 算法（Adaptive Collector）：监控当前堆的使用情况，并将选择适当算法的垃圾收集器。

发生地点：一般发生在堆内存中，因为大部分的对象都储存在堆内存中。

（堆内存为了配合垃圾回收有什么不同区域划分，各区域有什么不同？）

Java 的堆内存基于 Generation 算法（Generational Collector）划分为新生代、年老代和持久代。新生代又被进一步划分为 Eden 和 Survivor 区，最后 Survivor 由 FromSpace(Survivor0) 和 ToSpace(Survivor1) 组成。所有通过 new 创建的对象内存都在堆中分配，其大小可以通过 -Xmx 和 -Xms 来控制。分代收集基于这样一个事实：不同的对象的生命周期是不一样的。因此，可以将不同生命周期的对象分代，不同的代采取不同的回收算法进行垃圾回收（GC），以便提高回收效率。

按执行机制划分 Java 有四种类型的垃圾回收器：

- （1）串行垃圾回收器（Serial Garbage Collector）
- （2）并行垃圾回收器（Parallel Garbage Collector）
- （3）并发标记扫描垃圾回收器（CMS Garbage Collector）
- （4）G1 垃圾回收器（G1 Garbage Collector）

发生时间：程序空闲时间不定时回收。

#### 4、请你说一下 Java 中的异常处理机制



考点：java

参考回答：

在 Java 应用程序中，异常处理机制为：抛出异常，捕捉异常。

**抛出异常：**当一个方法出现错误引发异常时，方法创建异常对象并交付运行时系统，异常对象中包含了异常类型和异常出现时的程序状态等异常信息。运行时系统负责寻找处置异常的代码并执行。

**捕获异常：**在方法抛出异常之后，运行时系统将转为寻找合适的异常处理器（exception handler）。潜在的异常处理器是异常发生时依次存留在调用栈中的方法的集合。当异常处理器所能处理的异常类型与方法抛出的异常类型相符时，即为合适 的异常处理器。运行时系统从发生异常的方法开始，依次回查调用栈中的方法，直至找到含有合适异常处理器的方法并执行。当运行时系统遍历调用栈而未找到合适 的异常处理器，则运行时系统终止。同时，意味着 Java 程序的终止。

对于运行时异常、错误或可查异常，Java 技术所要求的异常处理方式有所不同：

由于运行时异常的不可查性，为了更合理、更容易地实现应用程序，Java 规定，运行时异常将由 Java 运行时系统自动抛出，允许应用程序忽略运行时异常。

对于方法运行中可能出现的 Error，当运行方法不欲捕捉时，Java 允许该方法不做任何抛出声明。因为，大多数 Error 异常属于永远不能被允许发生的状况，也属于合理的应用程序不该捕捉的异常。

对于所有的可查异常，Java 规定：一个方法必须捕捉，或者声明抛出方法之外。也就是说，当一个方法选择不捕捉可查异常时，它必须声明将抛出异常。

通常使用关键字 try、catch、finally 来捕获异常：

**try 块：**用于捕获异常。其后可接零个或多个 catch 块，如果没有 catch 块，则必须跟一个 finally 块。

**catch 块：**用于处理 try 捕获到的异常。

**finally 块：**无论是否捕获或处理异常，finally 块里的语句都会被执行。当在 try 块或 catch 块中遇到 return 语句时，finally 语句块将在方法返回之前被执行。在以下 4 种特殊情况下，finally 块不会被执行：

- 1) 在 finally 语句块中发生了异常。
- 2) 在前面的代码中用了 System.exit() 退出程序。
- 3) 程序所在的线程死亡。
- 4) 关闭 CPU。

try、catch、finally 语句块的执行顺序：



1) 当 try 没有捕获到异常时：try 语句块中的语句逐一被执行，程序将跳过 catch 语句块，执行 finally 语句块和其后的语句；

2) 当 try 捕获到异常，catch 语句块里没有处理此异常的情况：当 try 语句块里的某条语句出现异常时，而没有处理此异常的 catch 语句块时，此异常将会抛给 JVM 处理，finally 语句块里的语句还是会被执行，但 finally 语句块后的语句不会被执行；

3) 当 try 捕获到异常，catch 语句块里有处理此异常的情况：在 try 语句块中是按照顺序来执行的，当执行到某一条语句出现异常时，程序将跳到 catch 语句块，并与 catch 语句块逐一匹配，找到与之对应的处理程序，其他的 catch 语句块将不会被执行，而 try 语句块中，出现异常之后的语句也不会被执行，catch 语句块执行完后，执行 finally 语句块里的语句，最后执行 finally 语句块后的语句；

## 5、请问多线程是什么？

考点：编程

参考回答：

最开始，线程只是用于分配单个处理器的处理时间的一种工具。但假如操作系统本身支持多个处理器，那么每个线程都可分配给一个不同的处理器，真正进入“并行运算”状态。从程序设计语言的角度看，多线程操作最有价值的特性之一就是程序员不必关心到底使用了多少个处理器。程序在逻辑意义上被分割为数个线程；假如机器本身安装了多个处理器，那么程序会运行得更快，毋需作出任何特殊的调校。根据前面的论述，大家可能感觉线程处理非常简单。但必须注意一个问题：共享资源！如果有多个线程同时运行，而且它们试图访问相同的资源，就会遇到一个问题。举个例子来说，两个线程不能将信息同时发送给一台打印机。为解决这个问题，对那些可共享的资源来说（比如打印机），它们在使用期间必须进入锁定状态。所以一个线程可将资源锁定，在完成了它的任务后，再解开（释放）这个锁，使其他线程可以接着使用同样的资源。

多线程是为了同步完成多项任务，不是为了提高运行效率，而是为了提高资源使用效率来提高系统的效率。线程是在同一时间需要完成多项任务的时候实现的。

一个采用了多线程技术的应用程序可以更好地利用系统资源。其主要优势在于充分利用了 CPU 的空闲时间片，可以用尽可能少的时间来对用户的要求做出响应，使得进程的整体运行效率得到较大提高，同时增强了应用程序的灵活性。更为重要的是，由于同一进程的所有线程是共享同一内存，所以不需要特殊的数据传送机制，不需要建立共享存储区或共享文件，从而使得不同任务之间的协调操作与运行、数据的交互、资源的分配等问题更加易于解决。

## 6、请你来聊一聊集合类和内存

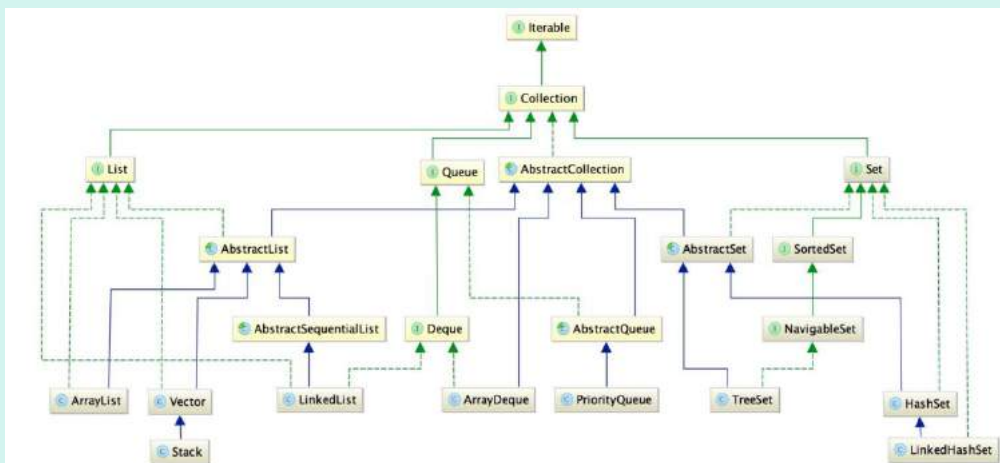
考点：java

参考回答：

## 一、集合类。

Java 中的集合包含多种数据结构，如链表、队列、哈希表等。从类的继承结构来说，可以分为两大类，一类是继承自 Collection 接口，这类集合包含 List、Set 和 Queue 等集合类。另一类是继承自 Map 接口，这主要包含了哈希表相关的集合类。

1、List、Set 和 Queue 类的继承结构图：绿色的虚线代表实现，绿色实线代表接口之间的继承，蓝色实线代表类之间的继承。



Collection 接口除了实现映射的集合类之外的所有集合类定义了一些方法

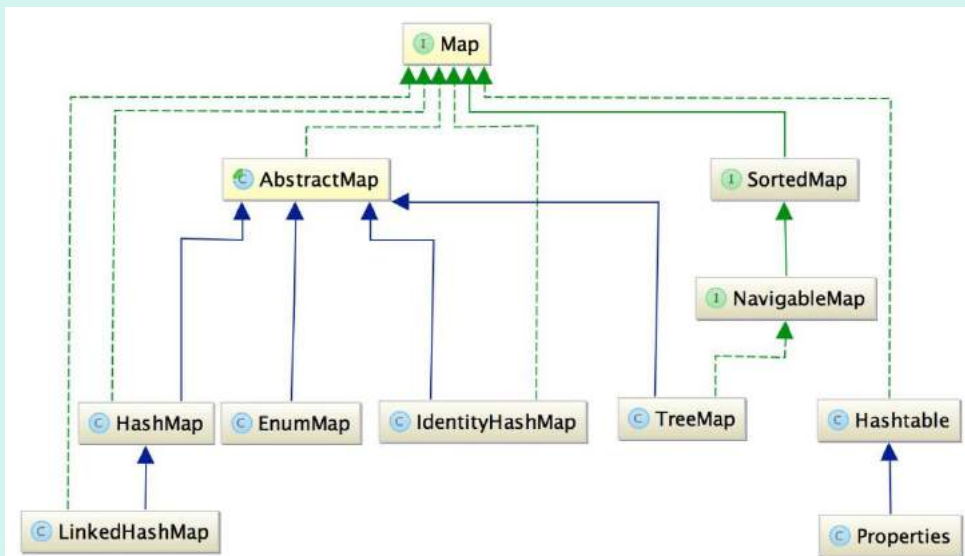
**List 集合类型：**描述了一种按位置存储数据的对象，是有序的。用的比较多 List 包括 **ArrayList** 和 **LinkedList**，这两者的区别：**ArrayList** 的底层通过数组实现，所以其随机访问的速度比较快，但是对于需要频繁的增删的情况，效率就比较低了。而对于 **LinkedList**，底层通过链表来实现，所以增删操作比较容易完成，但是对于随机访问的效率比较低。

**Queue：**一般可以直接使用 **LinkedList** 完成，**LinkedList** 继承自 **Deque**，所以 **LinkedList** 具有双端队列的功能。**PriorityQueue** 是为每个元素提供一个优先级，优先级高的元素会优先出队列。

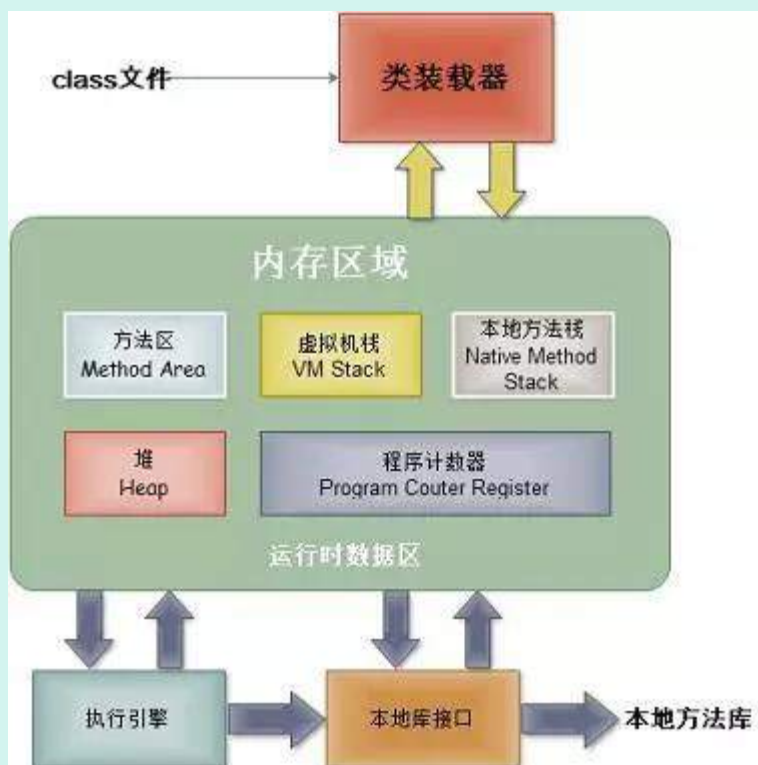
**Set：**Set 与 List 的主要区别是 Set 是不允许元素是重复的，而 List 则可以允许元素是重复的。**HashSet** 和 **LinkedHashSet** 的区别在于后者可以保证元素插入集合的元素顺序与输出顺序保持一致。而 **TresSet** 的区别在于其排序是按照 **Comparator** 来进行排序的，默认情况下按照字符的自然顺序进行升序排列。

**Iterable：**Collection 类继承自 **Iterable**，该接口的作用是提供元素遍历的功能，也就是说所有的集合类（除 Map 相关的类）都提供元素遍历的功能。**Iterable** 里面包含了 **Iterator** 的迭代器。

2、Map 类型的集合：最大的优点在于其查找效率比较高，理想情况下可以实现  $O(1)$  的时间复杂度。Map 中最常用的是 **HashMap**，**LinkedHashMap** 与 **HashMap** 的区别在于前者能够保证插入集合的元素顺序与输出顺序一致。这两者与 **TreeMap** 的区别在于 **TreeMap** 是根据键值进行排序的，其底层的实现也有本质的区别，**HashMap** 底层是一个哈希表，而 **TreeMap** 的底层数据结构是一棵树。



## 二、Java 内存区域划分



### 1. 程序计数器:

可以看做是当前线程所执行的字节码的行号指示器。在 JVM 的概念模型里，字节码解释器工作时就是通过改变这个计数器的值来选取下一条需要执行的字节码指令。

每条线程都有一个独立的程序计数器，所以程序计数器是线程私有的内存区域。



如果线程执行的是一个 Java 方法，计数器记录的是正在执行的虚拟机字节码指令的地址；如果线程执行的是一个 Native 方法，计数器的值为空。

Java 虚拟机规范中唯一一个没有规定任何 `OutOfMemoryError` 情况的区域。

## 2. Java 虚拟机栈：

描述 Java 方法执行的内存模型，每个方法执行的同时会创建一个栈帧，栈帧用于存储局部变量表、操作数栈、动态链接、方法出口等信息。每个方法从调用直至执行完成的过程，就对应着一个栈帧在虚拟机栈中入栈到出栈的过程。



Java 虚拟机栈是线程私有的，它的生命周期与线程相同。

局部变量表存放了编译时期可知的各种基本数据类型和对象引用。局部变量表所需的内存空间在编译时期完成分配，当进入一个方法时，这个方法需要在栈帧中分配多大的局部变量空间是完全确定的，在方法运行期间不会改变局部变量表的大小。

Java 虚拟机规范对这个区域规定了两种异常情况：

如果线程请求的栈深度大于虚拟机所允许的深度，将抛出 `StackOverflowError` 异常；

如果虚拟机栈可以动态扩展，如果扩展时无法申请到足够的内存，就会抛出 `OutOfMemoryError` 异常；

## 3. 本地方法栈：

本地方法栈与虚拟机栈的区别：虚拟机栈为虚拟机执行 Java 方法服务（也就是字节码），而本地方法栈为虚拟机使用到的 Native 方法服务。

Java 虚拟机规范对这个区域规定了两种异常情况：`StackOverflowError` 和 `OutOfMemoryError` 异常。

#### 4. Java 堆：

Java 堆是被所有的线程共享的一块内存区域，在虚拟机启动时创建。Java 堆的唯一目的就是存放对象实例，几乎所有的对象实例都在这里分配内存。

Java 堆是垃圾回收器管理的主要区域，从内存回收的角度看，由于现在收集器基本都采用分代收集算法，所以 Java 堆可以细分为：新生代、老生代；从内存分配的角度看，线程共享的 Java 堆可能划分出多个线程私有的分配缓冲区（TLAB）。

Java 堆可以处于物理上不连续的内存空间中，只要逻辑上是连续的即可。

Java 虚拟机规范规定，如果在堆上没有内存完成实例分配，并且堆上也无法再扩展时，将会抛出 `OutOfMemoryError` 异常。

Java 堆内存的 OOM 异常：

内存泄露：指程序中一些对象不会被 GC 所回收，它始终占用内存，即被分配的对象引用链可达但已无用。

内存溢出：程序运行过程中无法申请到足够的内存而导致的一种错误。内存溢出通常发生于 OLD 段或 Perm 段垃圾回收后，仍然无内存空间容纳新的 Java 对象的情况。

#### 5. 方法区：

被所有的线程共享的一块内存区域。它用于存储已被虚拟机加载的类信息、常量、静态变量、即时编译器编译后的代码等数据。

不需要连续的内存和可以选择固定大小或者可扩展之外，还可以选择不实现垃圾回收。

Java 虚拟机规范规定，当方法区无法满足内存分配的需求时，将抛出 `OutOfMemoryError` 异常。

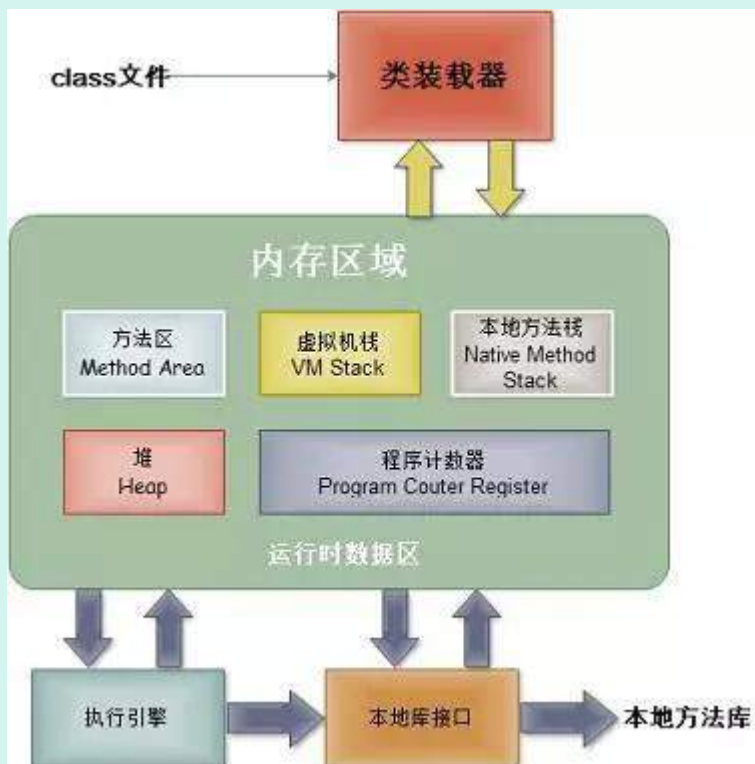
### 7、请你说一下 java jvm 的内存机制

考点：java

参考回答：

Java 内存区域划分





#### 1. 程序计数器:

可以看做是当前线程所执行的字节码的行号指示器。在 JVM 的概念模型里，字节码解释器工作时就是通过改变这个计数器的值来选取下一条需要执行的字节码指令。

每条线程都有一个独立的程序计数器，所以程序计数器是线程私有的内存区域。

如果线程执行的是一个 Java 方法，计数器记录的是正在执行的虚拟机字节码指令的地址；如果线程执行的是一个 Native 方法，计数器的值为空。

Java 虚拟机规范中唯一一个没有规定任何 OutOfMemoryError 情况的区域。

#### 2. Java 虚拟机栈:

描述 Java 方法执行的内存模型，每个方法执行的同时会创建一个栈帧，栈帧用于存储局部变量表、操作数栈、动态链接、方法出口等信息。每个方法从调用直至执行完成的过程，就对应着一个栈帧在虚拟机栈中入栈到出栈的过程。



Java 虚拟机栈是线程私有的，它的生命周期与线程相同。

局部变量表存放了编译时期可知的各种基本数据类型和对象引用。局部变量表所需的内存空间在编译时期完成分配，当进入一个方法时，这个方法需要在栈帧中分配多大的局部变量空间是完全确定的，在方法运行期间不会改变局部变量表的大小。

Java 虚拟机规范对这个区域规定了两种异常情况：

如果线程请求的栈深度大于虚拟机所允许的深度，将抛出 `StackOverflowError` 异常；

如果虚拟机栈可以动态扩展，如果扩展时无法申请到足够的内存，就会抛出 `OutOfMemoryError` 异常；

### 3. 本地方法栈：

本地方法栈与虚拟机栈的区别：虚拟机栈为虚拟机执行 Java 方法服务（也就是字节码），而本地方法栈为虚拟机使用到的 Native 方法服务。

Java 虚拟机规范对这个区域规定了两种异常情况：`StackOverflowError` 和 `OutOfMemoryError` 异常。

### 4. Java 堆：

Java 堆是被所有的线程共享的一块内存区域，在虚拟机启动时创建。Java 堆的唯一目的就是存放对象实例，几乎所有的对象实例都在这里分配内存。

Java 堆是垃圾回收器管理的主要区域，从内存回收的角度看，由于现在收集器基本都采用分代收集算法，所以 Java 堆可以细分为：新生代、老生代；从内存分配的角度看，线程共享的 Java 堆可能划分出多个线程私有的分配缓冲区（TLAB）。

Java 堆可以处于物理上不连续的内存空间中，只要逻辑上是连续的即可。



Java 虚拟机规范规定，如果在堆上没有内存完成实例分配，并且堆上也无法再扩展时，将会抛出 `OutOfMemoryError` 异常。

Java 堆内存的 OOM 异常：

内存泄露：指程序中一些对象不会被 GC 所回收，它始终占用内存，即被分配的对象引用链可达但已无用。

内存溢出：程序运行过程中无法申请到足够的内存而导致的一种错误。内存溢出通常发生于 OLD 段或 Perm 段垃圾回收后，仍然无内存空间容纳新的 Java 对象的情况。

5. 方法区：

被所有的线程共享的一块内存区域。它用于存储已被虚拟机加载的类信息、常量、静态变量、即时编译器编译后的代码等数据。

不需要连续的内存和可以选择固定大小或者可扩展之外，还可以选择不实现垃圾回收。

Java 虚拟机规范规定，当方法区无法满足内存分配的需求时，将抛出 `OutOfMemoryError` 异常。

## 8、请你说一说有哪几种垃圾回收算法

考点：java

参考回答：

Tracing 算法（Tracing Collector） 标记—清除算法：分为“标记”和“清除”两个阶段：首先标记出所需回收的对象，在标记完成后统一回收掉所有被标记的对象，它的标记过程其实就是前面的根搜索算法中判定垃圾对象的标记过程。

Compacting 算法（Compacting Collector） 标记—整理算法：标记的过程与标记—清除算法中的标记过程一样，但对标记后出的垃圾对象的处理情况有所不同，它不是直接对可回收对象进行清理，而是让所有的对象都向一端移动，然后直接清理掉端边界以外的内存。在基于 Compacting 算法的收集器的实现中，一般增加句柄和句柄表。

Copying 算法（Copying Collector）：将内存按容量分为大小相等的两块，每次只使用其中的一块（对象面），当这一块的内存用完了，就将还存活着的对象复制到另外一块内存上面（空闲面），然后再把已使用过的内存空间一次清理掉。

Adaptive 算法（Adaptive Collector）：监控当前堆的使用情况，并将选择适当算法的垃圾收集器。

## 9、请你说一说垃圾收集机制

考点：java



参考回答：

垃圾回收(Garbage Collection)是Java虚拟机(JVM)垃圾回收器提供了一种用于在空闲时间不定时回收无任何对象引用的对象占据的内存空间的一种机制。

引用：如果Reference类型的数据中存储的数值代表的是另外一块内存的起始地址，就称这块内存代表着一个引用。

(1) 强引用(Strong Reference)：如“Object obj = new Object ()”，这类引用是Java程序中最普遍的。只要强引用还存在，垃圾收集器就永远不会回收掉被引用的对象。

(2) 软引用(Soft Reference)：它用来描述一些可能还有用，但并非必须的对象。在系统内存不够用时，这类引用关联的对象将被垃圾收集器回收。JDK1.2之后提供了SoftReference类来实现软引用。

(3) 弱引用(Weak Reference)：它也是用来描述非须对象的，但它的强度比软引用更弱些，被弱引用关联的对象只能生存到下一次垃圾收集发生之前。当垃圾收集器工作时，无论当前内存是否足够，都会回收掉只被弱引用关联的对象。在JDK1.2之后，提供了WeakReference类来实现弱引用。

(4) 虚引用(Phantom Reference)：最弱的一种引用关系，完全不会对其生存时间构成影响，也无法通过虚引用来取得一个对象实例。为一个对象设置虚引用关联的唯一目的是希望能在该对象被收集器回收时收到一个系统通知。JDK1.2之后提供了PhantomReference类来实现虚引用。

垃圾：无任何对象引用的对象。

判断对象是否是垃圾的算法：

引用计数算法(Reference Counting Collector)、根搜索算法(Tracing Collector)：

回收：清理“垃圾”占用的内存空间而非对象本身。

Tracing 算法(Tracing Collector) 标记—清除算法：分为“标记”和“清除”两个阶段：首先标记出所需回收的对象，在标记完成后统一回收掉所有被标记的对象，它的标记过程其实就是前面的根搜索算法中判定垃圾对象的标记过程。

Compacting 算法(Compacting Collector) 标记—整理算法：标记的过程与标记—清除算法中的标记过程一样，但对标记后出的垃圾对象的处理情况有所不同，它不是直接对可回收对象进行清理，而是让所有的对象都向一端移动，然后直接清理掉端边界以外的内存。在基于Compacting 算法的收集器的实现中，一般增加句柄和句柄表。

Copying 算法(Copying Collector)：将内存按容量分为大小相等的两块，每次只使用其中的一块(对象面)，当这一块的内存用完了，就将还存活着的对象复制到另外一块内存上面(空闲面)，然后再把已使用过的内存空间一次清理掉。

Adaptive 算法(Adaptive Collector)：监控当前堆的使用情况，并将选择适当算法的垃圾收集器。



发生地点：一般发生在堆内存中，因为大部分的对象都储存在堆内存中。

（堆内存为了配合垃圾回收有什么不同区域划分，各区域有什么不同？）

Java 的堆内存基于 Generation 算法（Generational Collector）划分为新生代、年老代和持久代。新生代又被进一步划分为 Eden 和 Survivor 区，最后 Survivor 由 FromSpace(Survivor0) 和 ToSpace(Survivor1) 组成。所有通过 new 创建的对象内存都在堆中分配，其大小可以通过 -Xmx 和 -Xms 来控制。分代收集基于这样一个事实：不同的对象的生命周期是不一样的。因此，可以将不同生命周期的对象分代，不同的代采取不同的回收算法进行垃圾回收（GC），以便提高回收效率。

按执行机制划分 Java 有四种类型的垃圾回收器：

- （1）串行垃圾回收器（Serial Garbage Collector）
- （2）并行垃圾回收器（Parallel Garbage Collector）
- （3）并发标记扫描垃圾回收器（CMS Garbage Collector）
- （4）G1 垃圾回收器（G1 Garbage Collector）

发生时间：程序空闲时间不定时回收。

#### 10、请你回答一下 GC Root 可以是哪些

考点：java

参考回答：

- 1 、 虚拟机栈（栈帧中的本地变量表）中引用的对象。
- 2、 本地方法栈中 JNI（即一般说的 native 方法）引用的对象。
- 3、 方法区中的静态变量和常量引用的对象。

#### 11、请你说一下 OOM 可能发生在哪，怎么查看，怎么调优

考点：java

参考回答：

除了程序计数器不会抛出 OOM 外，其他各个内存区域都可能会抛出 OOM。

最常见的 OOM 情况有以下三种：



- `java.lang.OutOfMemoryError: Java heap space` -----> java 堆内存溢出，此种情况最常见，一般由于内存泄露或者堆的大小设置不当引起。对于内存泄露，需要通过内存监控软件查找程序中的泄露代码，而堆大小可以通过虚拟机参数 `-Xms, -Xmx` 等修改。

- `java.lang.OutOfMemoryError: PermGen space` -----> java 永久代溢出，即方法区溢出了，一般出现于大量 Class 或者 jsp 页面，或者采用 cglib 等反射机制的情况，因为上述情况会产生大量的 Class 信息存储于方法区。此种情况可以通过更改方法区的大小来解决，使用类似 `-XX:PermSize=64m -XX:MaxPermSize=256m` 的形式修改。另外，过多的常量尤其是字符串也会导致方法区溢出。

- `java.lang.StackOverflowError` -----> 不会抛 OOM error，但也是比较常见的 Java 内存溢出。JAVA 虚拟机栈溢出，一般是由于程序中存在死循环或者深度递归调用造成的，栈大小设置太小也会出现此种溢出。可以通过虚拟机参数 `-Xss` 来设置栈的大小。

OOM 分析--heapdump

要 dump 堆的内存镜像，可以采用如下两种方式：

- 设置 JVM 参数 `-XX:+HeapDumpOnOutOfMemoryError`，设定当发生 OOM 时自动 dump 出堆信息。不过该方法需要 JDK5 以上版本。

- 使用 JDK 自带的 `jmap` 命令。“`jmap -dump:format=b, file=heap.bin <pid>`” 其中 pid 可以通过 `jps` 获取。

dump 堆内存信息后，需要对 dump 出的文件进行分析，从而找到 OOM 的原因。常用的工具有：

- `mat`: eclipse memory analyzer，基于 eclipse RCP 的内存分析工具。

- `jhat`: JDK 自带的 java heap analyze tool，可以将堆中的对象以 html 的形式显示出来，包括对象的数量，大小等等，并支持对象查询语言 OQL，分析相关的应用后，可以通过 `http://localhost:7000` 来访问分析结果。不推荐使用，因为在实际的排查过程中，一般是先在生产环境 dump 出文件来，然后拉到自己的开发机器上分析，所以，不如采用高级的分析工具比如前面的 `mat` 来的高效。

## 12、请你说一下类加载

考点：java

参考回答：

### 1、什么是类的加载

类的加载指的是将类的 .class 文件中的二进制数据读入到内存中，将其放在运行时数据区的方法区内，然后在堆区创建一个 `java.lang.Class` 对象，用来封装类在方法区内的数据结构。类的加载的最终产品是位于堆区中的 Class 对象，Class 对象封装了类在方法区内的数据结构，并且向 Java 程序员提供了访问方法区内的数据结构的接口。类加载器并不需要等到某个类被“首





次主动使用”时再加载它，JVM 规范允许类加载器在预料某个类将要被使用时就预先加载它，如果在预先加载的过程中遇到了.class 文件缺失或存在错误，类加载器必须在程序首次主动使用该类时才报告错误（LinkageError 错误）如果这个类一直没有被程序主动使用，那么类加载器就不会报告错误。

加载.class 文件的方式

- 从本地系统中直接加载
- 通过网络下载.class 文件
- 从 zip, jar 等归档文件中加载.class 文件
- 从专有数据库中提取.class 文件
- 将 Java 源文件动态编译为.class 文件

## 2、类的生命周期

类加载的过程包括了加载、验证、准备、解析、初始化五个阶段。在这五个阶段中，加载、验证、准备和初始化这四个阶段发生的顺序是确定的，而解析阶段则不一定，它在某些情况下可以在初始化阶段之后开始，这是为了支持 Java 语言的运行时绑定（也成为动态绑定或晚期绑定）。另外注意这里的几个阶段是按顺序开始，而不是按顺序进行或完成，因为这些阶段通常都是互相交叉地混合进行的，通常在一个阶段执行的过程中调用或激活另一个阶段。

### A. 加载：查找并加载类的二进制数据

加载时类加载过程的第一个阶段，在加载阶段，虚拟机需要完成以下三件事情：

- 1、通过一个类的全限定名来获取其定义的二进制字节流。
- 2、将这个字节流所代表的静态存储结构转化为方法区的运行时数据结构。
- 3、在 Java 堆中生成一个代表这个类的 `java.lang.Class` 对象，作为对方法区中这些数据的访问入口。

相对于类加载的其他阶段而言，加载阶段（准确地说，是加载阶段获取类的二进制字节流的动作）是可控性最强的阶段，因为开发人员既可以使用系统提供的类加载器来完成加载，也可以自定义自己的类加载器来完成加载。加载阶段完成后，虚拟机外部的 二进制字节流就按照虚拟机所需的格式存储在方法区之中，而且在 Java 堆中也创建一个 `java.lang.Class` 类的对象，这样便可以通过该对象访问方法区中的这些数据。

### B. 连接

- 验证：确保被加载的类的正确性

验证是连接阶段的第一步，这一阶段的目的是为了确保 Class 文件的字节流中包含的信息符合当前虚拟机的要求，并且不会危害虚拟机自身的安全。验证阶段大致会完成 4 个阶段的检验动作：





- 文件格式验证：验证字节流是否符合 Class 文件格式的规范；例如：是否以 0xCAFEFABE 开头、主次版本号是否在当前虚拟机的处理范围之内、常量池中的常量是否有不被支持的类型。

- 元数据验证：对字节码描述的信息进行语义分析（注意：对比 javac 编译阶段的语义分析），以保证其描述的信息符合 Java 语言规范的要求；例如：这个类是否有父类，除了 java.lang.Object 之外。

- 字节码验证：通过数据流和控制流分析，确定程序语义是合法的、符合逻辑的。

- 符号引用验证：确保解析动作能正确执行。

验证阶段是非常重要的，但不是必须的，它对程序运行期没有影响，如果所引用的类经过反复验证，那么可以考虑采用 `-Xverify:none` 参数来关闭大部分的类验证措施，以缩短虚拟机类加载的时间。

- 准备：为类的静态变量分配内存，并将其初始化为默认值

准备阶段是正式为类变量分配内存并设置类变量初始值的阶段，这些内存都将在方法区中分配。

- 解析：把类中的符号引用转换为直接引用

解析阶段是虚拟机将常量池内的符号引用替换为直接引用的过程，解析动作主要针对类或接口、字段、类方法、接口方法、方法类型、方法句柄和调用点限定符 7 类符号引用进行。符号引用就是一组符号来描述目标，可以是任何字面量。直接引用就是直接指向目标的指针、相对偏移量或一个间接定位到目标的句柄。

### C. 初始化

初始化，为类的静态变量赋予正确的初始值，JVM 负责对类进行初始化，主要对类变量进行初始化。在 Java 中对类变量进行初始值设定有两种方式：

①声明类变量是指定初始值

②使用静态代码块为类变量指定初始值

#### JVM 初始化步骤

1、假如这个类还没有被加载和连接，则程序先加载并连接该类

2、假如该类的直接父类还没有被初始化，则先初始化其直接父类

3、假如类中有初始化语句，则系统依次执行这些初始化语句

类初始化时机：只有当对类的主动使用的时候才会导致类的初始化，类的主动使用包括以下六种：

- 创建类的实例，也就是 new 的方式



- 访问某个类或接口的静态变量，或者对该静态变量赋值
- 调用类的静态方法
- 反射（如 `Class.forName("com.shengsiyuan.Test")`）
- 初始化某个类的子类，则其父类也会被初始化
- Java 虚拟机启动时被标明为启动类的类（Java Test），直接使用 `java.exe` 命令来运行某个主类

#### D. 结束生命周期

在如下几种情况下，Java 虚拟机将结束生命周期

- 执行了 `System.exit()` 方法
- 程序正常执行结束
- 程序在执行过程中遇到了异常或错误而异常终止
- 由于操作系统出现错误而导致 Java 虚拟机进程终止

#### 3、类的加载

类加载有三种方式：

- 命令行启动应用时候由 JVM 初始化加载
- 通过 `Class.forName()` 方法动态加载
- 通过 `ClassLoader.loadClass()` 方法动态加载

#### 4、双亲委派模型

双亲委派模型的工作流程是：如果一个类加载器收到了类加载的请求，它首先不会自己去尝试加载这个类，而是把请求委托给父加载器去完成，依次向上，因此，所有的类加载请求最终都应该被传递到顶层的启动类加载器中，只有当父加载器在它的搜索范围中没有找到所需的类时，即无法完成该加载，子加载器才会尝试自己去加载该类。

双亲委派机制：

- 当 `AppClassLoader` 加载一个 `class` 时，它首先不会自己去尝试加载这个类，而是把类加载请求委派给父类加载器 `ExtClassLoader` 去完成。
- 当 `ExtClassLoader` 加载一个 `class` 时，它首先也不会自己去尝试加载这个类，而是把类加载请求委派给 `BootStrapClassLoader` 去完成。
- 如果 `BootStrapClassLoader` 加载失败（例如在 `$JAVA_HOME/jre/lib` 里未查找到该 `class`），会使用 `ExtClassLoader` 来尝试加载；

- 若 ExtClassLoader 也加载失败，则会使用 AppClassLoader 来加载，如果 AppClassLoader 也加载失败，则会报出异常 ClassNotFoundException。

### 13、请你说一下 AQS

考点：java 多线程

参考回答：

Java 并发包（JUC）中提供了很多并发工具，这其中，很多我们耳熟能详的并发工具，譬如 ReentrantLock、Semaphore，它们的实现都用到了一个共同的基类

—AbstractQueuedSynchronizer，简称 AQS。AQS 是一个用来构建锁和同步器的框架，使用 AQS 能简单且高效地构造出应用广泛的大量的同步器，比如我们提到的 ReentrantLock，Semaphore，其他的诸如 ReentrantReadWriteLock，SynchronousQueue，FutureTask 等等皆是基于 AQS 的。

AQS 使用一个 int 成员变量来表示同步状态，通过内置的 FIFO 队列来完成获取资源线程的排队工作。状态信息通过 procted 类型的 getState，setState，compareAndSetState 进行操作

AQS 支持两种同步方式：1. 独占式 2. 共享式。这样方便使用者实现不同类型的同步组件，独占式如 ReentrantLock，共享式如 Semaphore，CountDownLatch，组合式的如 ReentrantReadWriteLock。总之，AQS 为使用提供了底层支撑，如何组装实现，使用者可以自由发挥。同步器的设计是基于模板方法模式的，一般的使用方式是这样：

1. 使用者继承 AbstractQueuedSynchronizer 并重写指定的方法。（这些重写方法很简单，无非是对于共享资源 state 的获取和释放）

2. 将 AQS 组合在自定义同步组件的实现中，并调用其模板方法，而这些模板方法会调用使用者重写的方法。

### 14、请你说一下 volatile

考点：java 多线程

参考回答：

一旦一个共享变量（类的成员变量、类的静态成员变量）被 volatile 修饰之后，那么就具备了两层语义：1）保证了不同线程对这个变量进行操作时的可见性，即一个线程修改了某个变量的值，这新值对其他线程来说是立即可见的。2）禁止进行指令重排序。

《深入理解 Java 虚拟机》中对 volatile 的描述：“观察加入 volatile 关键字和没有加入 volatile 关键字时所生成的汇编代码发现，加入 volatile 关键字时，会多出一个 lock 前缀指令”

lock 前缀指令实际上相当于一个内存屏障（也成内存栅栏），内存屏障会提供 3 个功能：



1) 它确保指令重排序时不会把其后面的指令排到内存屏障之前的位置，也不会把前面的指令排到内存屏障的后面；即在执行到内存屏障这句指令时，在它前面的操作已经全部完成；

2) 它会强制将对缓存的修改操作立即写入主存；

3) 如果是写操作，它会导致其他 CPU 中对应的缓存行无效。

因为 volatile 关键字无法保证操作的原子性。通常来说，使用 volatile 必须具备以下 2 个条件：

1) 对变量的写操作不依赖于当前值

2) 该变量没有包含在具有其他变量的不变式中

### 15、请你说一下死锁的原因，以及如何打破，如何查看死锁进程状态

考点：java 多线程

参考回答：

1、死锁是指在一组进程中的各个进程均占有不会释放的资源，但因互相申请被其他进程所站用不会释放的资源而处于的一种永久等待状态。死锁的四个必要条件：

- 互斥条件 (Mutual exclusion)：资源不能被共享，只能由一个进程使用。
- 请求与保持条件 (Hold and wait)：已经得到资源的进程可以再次申请新的资源。
- 非剥夺条件 (No pre-emption)：已经分配的资源不能从相应的进程中被强制地剥夺。
- 循环等待条件 (Circular wait)：系统中若干进程组成环路，该环路中每个进程都在等待相邻进程正占用的资源。

java 中产生死锁可能性的最根本原因是：1) 是多个线程涉及到多个锁，这些锁存在着交叉，所以可能会导致了一个锁依赖的闭环；2) 默认的锁申请操作是阻塞的。

如，线程在获得一个锁 L1 的情况下再去申请另外一个锁 L2，也就是锁 L1 想要包含了锁 L2，在获得了锁 L1，并且没有释放锁 L1 的情况下，又去申请获得锁 L2，这个是产生死锁的最根本原因。

2、避免死锁：

- 方案一：破坏死锁的循环等待条件。
- 方法二：破坏死锁的请求与保持条件，使用 lock 的特性，为获取锁操作设置超时时间。这样不会死锁（至少不会无尽的死锁）



- 方法三：设置一个条件遍历与一个锁关联。该方法只用一把锁，没有 chopstick 类，将竞争从对筷子的争夺转换成了对状态的判断。仅当左右邻座都没有进餐时才可以进餐。提升了并发度。

### 3、linux 中查看死锁进程状态

使用 pstack 和 gdb 工具对死锁程序进行分析

pstack 进程号 查看各个线程的堆栈信息

当进程吊死的时候，多次使用，死锁的线程将一直处于等锁的状态，确定某些线程一直没有变化，一直处于等锁的状态。那么这些线程很可能是死锁了。如果怀疑哪些线程发生死锁了，可以采用 gdb 进一步 attach 线程并进行分析。

执行命令 `gdb attach 进程号`，进入 gdb 调试终端

运行：`(gdb) info thread`

## 16、请你说一下内存泄漏

考点：java 编程

参考回答：

当一个对象已经不需要再使用本该被回收时，另外一个正在使用的对象持有它的引用从而导致它不能被回收，这导致本该被回收的对象不能被回收而停留在堆内存中，这就产生了内存泄漏。内存泄漏是造成应用程序 OOM 的主要原因之一。我们知道 Android 系统为每个应用程序分配的内存是有限的，而当一个应用中产生的内存泄漏比较多时，这就难免会导致应用所需要的内存超过系统分配的内存限额，这就造成了内存溢出从而导致应用 Crash。

常见的内存泄漏：

### 1、单例造成的内存泄漏

由于单例的静态特性使得其生命周期和应用的生命周期一样长，如果一个对象已经不再需要使用了，而单例对象还持有该对象的引用，就会使得该对象不能被正常回收，从而导致了内存泄漏。

### 2、非静态内部类创建静态实例造成的内存泄漏

非静态内部类默认会持有外部类的引用，而该非静态内部类又创建了一个静态的实例，该实例的生命周期和应用的一样长，这就导致了该静态实例一直会持有该 Activity 的引用，从而导致 Activity 的内存资源不能被正常回收。

### 3、Handler 造成的内存泄漏

### 4、线程造成的内存泄漏



如果任务在 Activity 销毁之前还未完成，那么将导致 Activity 的内存资源无法被回收，从而造成内存泄漏。

#### 5、资源未关闭造成的内存泄漏

对于使用了 BroadcastReceiver, ContentObserver, File, Cursor, Stream, Bitmap 等资源，应该在 Activity 销毁时及时关闭或者注销，否则这些资源将不会被回收，从而造成内存泄漏。

#### 6、使用 ListView 时造成的内存泄漏

#### 7、集合容器中的内存泄露

#### 8、WebView 造成的泄露

避免内存泄漏：

1、在涉及使用 Context 时，对于生命周期比 Activity 长的对象应该使用 Application 的 Context。

2、对于需要在静态内部类中使用非静态外部成员变量（如：Context、View），可以在静态内部类中使用弱引用来引用外部类的变量来避免内存泄漏。

3、对于不再需要使用的对象，显示的将其赋值为 null，比如使用完 Bitmap 后先调用 recycle()，再赋为 null。

4、保持对对象生命周期的敏感，特别注意单例、静态对象、全局性集合等的生命周期。

5、对于生命周期比 Activity 长的内部类对象，并且内部类中使用了外部类的成员变量，可以这样做避免内存泄漏：

1) 将内部类改为静态内部类

2) 静态内部类中使用弱引用来引用外部类的成员变量

### 17、请你说一说 class 和 interface 的区别

考点：java

参考回答：

1、接口类似于类，但接口的成员都没有执行方式，它只是方法、属性、事件和索引的组合而已，并且也只能包含这四种成员；类除了这四种成员之外还可以有别的成员(如字段)。

2、不能实例化一个接口，接口只包括成员的签名；而类可以实例化(abstract 类除外)。

3、接口没有构造函数，类有构造函数。

4、接口不能进行运算符的重载，类可以进行运算符重载。





5、接口的成员没有任何修饰符，其成员总是公共的，而类的成员则可以有修饰符(如：虚拟或者静态)。

6、派生于接口的类必须实现接口中所有成员的执行方式，而从类派生则不然。

## 18、请你说一下内存泄漏的原因

考点：java 编程

参考回答：

当一个对象已经不需要再使用本该被回收时，另外一个正在使用的对象持有它的引用从而导致它不能被回收，这导致本该被回收的对象不能被回收而停留在堆内存中，这就产生了内存泄漏。内存泄漏是造成应用程序 OOM 的主要原因之一。我们知道 Android 系统为每个应用程序分配的内存是有限的，而当一个应用中产生的内存泄漏比较多时，这就难免会导致应用所需要的内存超过系统分配的内存限额，这就造成了内存溢出从而导致应用 Crash。

常见的内存泄漏：

### 1、单例造成的内存泄漏

由于单例的静态特性使得其生命周期和应用的生命周期一样长，如果一个对象已经不再需要使用了，而单例对象还持有该对象的引用，就会使得该对象不能被正常回收，从而导致了内存泄漏。

### 2、非静态内部类创建静态实例造成的内存泄漏

非静态内部类默认会持有外部类的引用，而该非静态内部类又创建了一个静态的实例，该实例的生命周期和应用的一样长，这就导致了该静态实例一直会持有该 Activity 的引用，从而导致 Activity 的内存资源不能被正常回收。

### 3、Handler 造成的内存泄漏

### 4、线程造成的内存泄漏

如果任务在 Activity 销毁之前还未完成，那么将导致 Activity 的内存资源无法被回收，从而造成内存泄漏。

### 5、资源未关闭造成的内存泄漏

对于使用了 BroadcastReceiver, ContentObserver, File, Cursor, Stream, Bitmap 等资源，应该在 Activity 销毁时及时关闭或者注销，否则这些资源将不会被回收，从而造成内存泄漏。

### 6、使用 ListView 时造成的内存泄漏

### 7、集合容器中的内存泄露





#### 8、WebView 造成的泄露

避免内存泄漏：

1、在涉及使用 Context 时，对于生命周期比 Activity 长的对象应该使用 Application 的 Context。

2、对于需要在静态内部类中使用非静态外部成员变量（如：Context、View），可以在静态内部类中使用弱引用来引用外部类的变量来避免内存泄漏。

3、对于不再需要使用的对象，显示的将其赋值为 null，比如使用完 Bitmap 后先调用 recycle()，再赋为 null。

4、保持对对象生命周期的敏感，特别注意单例、静态对象、全局性集合等的生命周期。

5、对于生命周期比 Activity 长的内部类对象，并且内部类中使用了外部类的成员变量，可以这样做避免内存泄漏：

1) 将内部类改为静态内部类

2) 静态内部类中使用弱引用来引用外部类的成员变量

#### 19、请你说一说强引用和弱引用

考点：java

参考回答：

强引用：

强引用是使用最普遍的引用。如果一个对象具有强引用，那垃圾回收器绝不会回收它。如下：

```
Object o=new Object(); // 强引用
```

当内存空间不足，Java 虚拟机宁愿抛出 OutOfMemoryError 错误，使程序异常终止，也不会靠随意回收具有强引用的对象来解决内存不足的问题。如果不使用时，要通过如下方式来弱化引用，如下：

```
o=null; // 帮助垃圾收集器回收此对象
```

显式地设置 o 为 null，或超出对象的生命周期范围，则 gc 认为该对象不存在引用，这时就可以回收这个对象。具体什么时候收集这要取决于 gc 的算法。

举例：

```
public void test() {  
  
    Object o=new Object();
```



```
}
```

在一个方法的内部有一个强引用，这个引用保存在栈中，而真正的引用内容（Object）保存在堆中。当这个方法运行完成后就会退出方法栈，则引用内容的引用不存在，这个 Object 会被回收。但是如果这个 o 是全局的变量时，就需要在不用这个对象时赋值为 null，因为强引用不会被垃圾回收。

弱引用：

弱引用也是用来描述非必需对象的，只具有弱引用的对象拥有更短暂的生命周期。在垃圾回收器线程扫描它所管辖的内存区域的过程中，一旦发现了只具有弱引用的对象，不管当前内存空间足够与否，都会回收它的内存。不过，由于垃圾回收器是一个优先级很低的线程，因此不一定会很快发现那些只具有弱引用的对象。提供 WeakReference 类实现弱引用。

例子：

```
String str=new String("abc");
```

```
WeakReference<String> abcWeakRef = new WeakReference<String>(str);
```

```
str=null;
```

当垃圾回收器进行扫描回收时等价于：

```
str = null;
```

```
System.gc();
```

如果这个对象是偶尔的使用，并且希望在使用时随时就能获取到，但又不想影响此对象的垃圾收集，那么你应该用 Weak Reference 来记住此对象。

```
String abc = abcWeakRef.get(); //该代码会让 str 再次变为一个强引用：
```

弱引用可以和一个引用队列（ReferenceQueue）联合使用，如果弱引用所引用的对象被垃圾回收，Java 虚拟机就会把这个弱引用加入到与之关联的引用队列中。当你想引用一个对象，但是这个对象有自己的生命周期，你不想介入这个对象的生命周期，这时候你就是用弱引用。这个引用不会在对象的垃圾回收判断中产生任何附加的影响。

## 20、请说一下你对多态的理解

考点：java

参考回答：

什么是多态？



多态就是指程序中定义的引用变量所指向的具体类型和通过该引用变量发出的方法调用在编程时并不确定，而是在程序运行期间才确定，即一个引用变量到底会指向哪个类的实例对象，该引用变量发出的方法调用到底是哪个类中实现的方法，必须在由程序运行期间才能决定。因为在程序运行时才确定具体的类，这样，不用修改源程序代码，就可以让引用变量绑定到各种不同的类实现上，从而导致该引用调用的具体方法随之改变，即不修改程序代码就可以改变程序运行时所绑定的具体代码，让程序可以选择多个运行状态，这就是多态性。

Java 实现多态有三个必要条件：继承、重写、向上转型。

1. 继承：在多态中必须存在有继承关系的子类 and 父类。
2. 重写：子类对父类中某些方法进行重新定义，在调用这些方法时就会调用子类的方法。
3. 向上转型：在多态中需要将子类的引用赋给父类对象，只有这样该引用才能够具备技能调用父类的方法和子类的方法。

Java 中有两种形式可以实现多态，继承和接口：

1. 基于继承的实现机制主要表现在父类和继承该父类的一个或多个子类对某些方法的重写，多个子类对同一方法的重写可以表现出不同的行为。
2. 基于接口的多态中，指向接口的引用必须是指定这实现了该接口的一个类的实例程序，在运行时，根据对象引用的实际类型来执行对应的方法。

## 21、手写代码：求 n 以内的最大质数

考点：编程

参考回答：

```
public class Prime {  
  
    public static void main(String[] args) {  
  
        Scanner in = new Scanner (System.in);  
  
        int N=in.nextInt();  
  
        if (N%2==0) N--;  
  
        for (int i =N; i >2; i--) {  
  
            if (isPrime(i)) {  
  
                System.out.print("N 以内最大质数为: "+i);  
  
                break;  
            }  
        }  
    }  
}
```



```
}  
  
}  
  
}  
  
}  
  
// 判断某整数是否为质数  
  
public static boolean isPrime(int m) {  
  
    if (m < 2) {  
  
        return false;  
  
    }  
  
    for (int i = 2; i * i <= m; i++) {  
  
        if (m%i == 0) {  
  
            return false;  
  
        }  
  
    }  
  
    return true;  
  
}
```

## 22、请你说一下 java 里内存泄漏和溢出的区别

考点：逻辑

参考回答：

1、内存泄漏 memory leak :是指程序在申请内存后，无法释放已申请的内存空间，一次内存泄漏似乎不会有大的影响，但内存泄漏堆积后的后果就是内存溢出。

2、内存溢出 out of memory :指程序申请内存时，没有足够的内存供申请者使用，或者说，给了你一块存储 int 类型数据的存储空间，但是你却存储 long 类型的数据，那么结果就是内存不够用，此时就会报错 OOM, 即所谓的内存溢出。

## 23、请问你用过什么语言，用这些语言写过什么程序

考察点：java/python



参考回答：

用 Java 写过一个高并发秒杀系统，用 Python 搭建过自动化测试框架

## 24、你知道 java 里面的内存管理机制吗，比如创建和回收是怎么管理的？

考察点：内存管理机制

参考回答：

java 虚拟机在执行 Java 程序的过程中会把它所管理的内存划分为若干个不同的数据区域。这些区域存储不同类型的数据，这些区域的内存分配和销毁的时间也不同，有的区域随着虚拟机进程的启动而存在，有些区域则是依赖用户线程的启动和结束而建立和销毁。根据《Java 虚拟机规范（第 2 版）》的规定，Java 虚拟机管理的内存包括五个运行时数据区域，

### 1、方法区

方法区（Method Area）是各个线程共享的内存区域，它用于存储已被虚拟机加载的类信息（包括类的名称、方法信息、成员变量信息）、常量、静态变量、以及编译器编译后的代码等数据

### 2、虚拟机栈和本地方法栈

虚拟机栈是线程私有的内存空间，每个线程都有一个线程栈，本地方法栈也是线程私有的内存空间，本地方法栈与 Java 栈所发挥的作用是非常相似的，它们之间的区别不过是 Java 栈执行 Java 方法，本地方法栈执行的是本地方法，有的虚拟机直接把本地方法栈和虚拟机栈合二为一。

### 3、堆

Java 堆是 Java 虚拟机所管理的内存中最大的一块，在虚拟机启动时创建，此内存区域的目的是存放对象实例，几乎所有的对象实例都在这里分配内存。从内存分配的角度来看，线程共享的 Java 堆中可能划分出多个线程私有的分配缓冲区（TLAB）。Java 堆可以处于物理上不连续的内存空间，只要逻辑上连续即可，在实现上，既可以实现固定大小的，也可以是扩展的。如果堆中没有足够的内存分配给实例，并且堆也无法再拓展时，将会抛出 OutOfMemoryError 异常。

Java 内存回收：

对于栈空间，当方法调用结束后，基本类型变量，引用类型变量，形参占据的空寂会被自动释放，但引用类型指向的对象在堆中，堆中的无用内存有垃圾回收线程回收，GC 线程优先级最低，只有当没有工作线程存在时 GC 线程才会执行，或者堆空间不足时会自动出发 GC 线程工作，

## 25、请你说一下 Java 里 integer 和 int 的区别，以及如何比较相等

考察点：Java

参考回答：



Integer 和 int 的区别：

- 1、integer 是 int 的包装类，int 是 Java 的一种基本数据结构
- 2、integer 变量必须实例化后才能使用，int 变量不需要
- 3、integer 实际是对象的引用，int 是直接存储数据值
- 4、integer 的默认值是 null，int 的默认值是 0

如何比较相等，首先要明白 equals 和 == 的区别

Equals 通常用来比较两个对象的内容是否相等，== 用来比较两个对象的地址是否相等，Object 类中的 equals 方法定义为判断两个对象的地址是否相等（可以理解成是否是同一个对象），地址相等则认为是对象相等。这也就意味着，我们新建的所有类如果没有复写 equals 方法，那么判断两个对象是否相等时就等同于“==”，也就是两个对象的地址是否相等。但在我们的实际开发中，通常会认为两个对象的内容相等时，则两个对象相等，equals 返回 true。对象内容不同，则返回 false。

所以可以总结为两种情况

- 1、类未复写 equals 方法，则使用 equals 方法比较两个对象时，相当于 == 比较，即两个对象的地址是否相等。地址相等，返回 true，地址不相等，返回 false。
- 2、类复写 equals 方法，比较两个对象时，则走复写之后的判断方式。通常，我们会将 equals 复写成：当两个对象内容相同时，则 equals 返回 true，内容不同时，返回 false。

## 26、请你介绍下 Java 学习情况，学习一门新的语言需要多快

参考回答：Java

参考回答：

如实回答即可，体现出一个学习能力的地方。

## 27、请你介绍一下 gc，另外如果 Java 里写一个方法，这个方法里只有一条语句，即 new 一个对象，请问方法结束以后这个对象怎么回收的？

考察点：Java 垃圾回收机制

考察公司：网易

参考回答：

GC (garbage collection) 垃圾收集，是指 JVM 用于释放那些不再使用的对象所占用的内存，常用的 JVM 都有 GC，而且大多数 gc 都使用类似的算法管理内存和执行手机操作，GC 可以通过确定对象是否被活动对象引用来确定是否收集该对象，常用的方法有引用计数和对象引用遍历

## 28、请你回答一下 protected, public, private 的区别

考察点：编程语言

参考回答：

作用域	当前类	同一 package	子孙类	其它 package
public	√	√	√	√
protected	√	√	√	×
friendly	√	√	×	×
private	√	×	×	×

## 29、请你说一下抽象类和接口的区别

考察点：编程语言

参考回答：

从语法层面来说，

- 1、抽象类可以提供成员方法的实现细节，而接口中只能存在抽象方法
- 2、抽象类中成员变量可以是多种类型，接口中成员变量必须用 public, static, final 修饰
- 3、一个类只能继承一个抽象类，但可以实现多个接口
- 4、抽象类中允许含有静态代码块和静态方法，接口不能

从设计层面而言

1. 抽象类是对整个类的属性，行为等方面进行抽象，而接口则是对行为抽象。就好比飞机和鸟，抽象类抽象出的是飞行物类。而接口则是抽闲出飞行方法。
2. 抽象类是一个模板式的设计，当在开发过程中出现需求更改的情况，只需要更改抽象类而不需要更改它的子类。接口是一种辐射性设计，当接口的内容发生改变时，需要同时对实现它的子类进行相应的修改。





3. 抽象类可以类比为模板，而接口可以类比为协议

### 30、请你说一下 List 和 ArrayList 的区别，以及 ArrayList 和 HashSet 区别

考察点：数据结构

参考回答：

List：是一个有序的集合，可以包含重复的元素。提供了按索引访问的方式。它继承 Collection。

List 有两个重要的实现类：ArrayList 和 LinkedList

ArrayList：我们可以将其看作是能够自动增长容量的数组。

利用 ArrayList 的 toArray() 返回一个数组。

Arrays.asList() 返回一个列表。

1. ArrayList 底层采用数组实现，当使用不带参数的构造方法生成 ArrayList 对象时，实际上会在底层生成一个长度为 10 的 Object 类型数组
2. 如果增加的元素个数超过了 10 个，那么 ArrayList 底层会新生成一个数组，长度为原数组的 1.5 倍+1，然后将原数组的内容复制到新数组当中，并且后续增加的内容都会放到新数组当中。当新数组无法容纳增加的元素时，重复该过程。
3. 对于 ArrayList 元素的删除操作，需要将被删除元素的后续元素向前移动，代价比较高。
4. 集合当中只能放置对象的引用，无法放置原生数据类型，我们需要使用原生数据类型的包装类才能加入到集合当中。
5. 集合当中放置的都是 Object 类型，因此取出来的也是 Object 类型，那么必须要使用强制类型转换将其转换为真正的类型（放置进去的类型）

### 31、请你回答一下 Java 的内存结构是什么，全局变量，临时变量，静态变量分别存在哪里，堆分为哪几块，比如说新生代老生代，那么新生代又分为什么

考察点：Java

参考回答：

代码区：是编译器生成的一个 exe 区段，存放函数体的二进制代码

栈区：存放函数的参数，局部变量的值等，其操作方式类似于数据结构中的栈，const 局部变量也是放在栈里

堆区：就是 malloc 和 new 之类的内存所在区段，一般由程序员分配释放，分配方式类似于链表

静态数据区：是编译器生成的一个 exe 区段，初始和未初始化的全局变量和局部变量都放在这里，

常量区：是编译器生成的一个 exe 区段，const 全局变量也放在常量区。

全局变量，临时变量，静态变量分别存在哪里



局部变量保存在栈中，全局变量和静态变量存储在静态数据区。

堆分为哪几块，比如说新生代老年代，那么新生代又分为什么？

java 垃圾收集管理器的主要区域，因此很多时候被称为“GC 堆”。

分为新生代和老年代；

新生代分为：Eden 和 Survivor。

### 32、手写代码：给出一个 int 类型 123，写一个函数，返回反转的值 321

考察点：编程

参考回答：

```
void my_itoa(int x, char *s, char radix)
{
    char zm[37]="0123456789abcdefghijklmnopqrstuvwxyz";
    int i=0;
    int sum=x;
    while(sum>0)
    {
        s[i++]=zm[sum%radix];
        sum/=radix;
    }
}

int atoi_my( char *str)
{
    int s=0;
    while(*str>='0' &&*str<='9')
    {
        s=s*10+*str-'0';
        str++;
        if(s<0)
        {
            s=2147483647;
            break;
        }
    }
    return s; //
}
```

**33、请你回答一下 c++ 和 java 的区别**

考察点：编程语言

参考回答：

- 1、C++ 创建对象后需要在使用结束后调用 delete 方法将其销毁，Java 有垃圾回收机制，用来监视 new 出来的所有对象，辨别不会再被引用的对象，然后释放内存空间
- 2、当变量作为类的成员使用时，Java 才确保给定默认值，以确保那些基本类型的成员变量得到初始化，但是 C++ 没有此功能，
- 3、c++ 引入了操作符重载机制，Java 不支持
- 4、Java 中没有 sizeof()，在 C++ 中 sizeof() 操作符能够告诉我们为数据项分配的字节数，因为 C++ 中不同的数据类型在不同的机器上可能有不同的大小，但是在 Java 中所有的数据类型在所有机器中大小都是相同的。
- 5、Java 的运行速度比 C++ 慢，因为 Java 是半解释和半编译的。
- 6、C++ 中有多继承，Java 中只有单一继承，但是 java 可以通过接口实现多继承
- 7、在 C++ 中，数组定义时，已经分配存储空间，并且可以使用，在 Java 中，数组定义时只定义了数组变量，数组是不可以使用的，只有数组 new 之后才会创建数组，并分配存储空间。
- 8、C++ 有指针，Java 没有。

**34、请你回答一下怎么判断哪些对象是可以删除的，可达是什么意思**

考察点：垃圾回收机制

参考回答：

判断方法有两种，一种是引用计数器，给对象添加一个引用计数器，每当有一个地方引用他时，计数器就加一，引用失败计数器减一，计数器为零的对象就不可能再被使用，

第二种方法是可达性分析算法。这个算法的基本思路就是通过一系列的称为“GC Roots”的对象作为起始点，从这些节点开始向下搜索，搜索走过的路径称为引用链。当一个对象到 GC Roots 没有任何引用链相连的时候，则证明此对象是不可用的。否则即是可达的。在 java 语言中，可作为 GC Roots 的对象包括下面几种：虚拟机栈中引用的对象、方法区中类静态属性引用的对象、方法区中常量引用的对象和本地方法栈中 JNI（一般说的 Native 方法）引用的对象。

**35、请你说一说接口有什么限制**

考察点：

参考回答：



- 1、变量会被隐式地指定为 `public static final` 变量，并且只能是 `public static final` 变量，用 `private` 修饰会报编译错误
- 2、方法会被隐式地指定为 `public abstract` 方法且只能是 `public abstract` 方法（用其他关键字，比如 `private`、`protected`、`static`、`final` 等修饰会报编译错误）
- 3、接口中所有的方法不能有具体的实现，也就是说，接口中的方法必须都是抽象方法

**36、请问 Java 中线程如何实现，如何实现多线程，线程安全在 Java 中是如何实现的，线程的工作区是哪里**

考察点：Java，操作系统

参考回答：

Java 中实现线程有三种方式：

1、1. 继承 Thread 类

```
public class Thread extends Object implements Runnable
```

定义 Thread 类的子类，并重写 Thread 类的 `run()` 方法，创建子类对象（即线程对象），调用线程对象的 `start()` 方法来启动该线程。

2. 实现 Runnable 接口

```
public interface Runnable
```

定义 Runnable 接口的实现类，并重写该接口的 `run()` 方法，该 `run()` 方法同样是该线程的执行体。创建该 Runnable 实现类的实例，并将此实例作为 Thread 的 `target`（即构造函数中的参数）来创建 Thread 对象（该 Thread 对象才是真正的线程对象，只是该 Threa

3. 使用 Callable 和 Future

创建 Callable 接口的实现类，并实现 `call()` 方法，该方法有返回值；创建 Callable 实现类的实例，使用 FutureTask 来包装 Callable 对象，并且也封装了 `call()` 方法的返回值；使用 FutureTask 作为 Thread 类的 `target` 创建并启动线程；调用 FutureTask 对象的 `get()` 方法返回子线程执行结束后的返回值。

如何实现多线程：

首先是继承 Thread 类并重写 `run()` 方法

```
package com.csu.multiThread;
```

```
public class MultiThreadExtendsThread extends Thread{
```



```
String name;

public MultiThreadExtendsThread(String name) {

    this.name = name;

}

public void run() {

    for(int i=0;i<5;i++) {

        System.out.println(name+"运行: "+i);

    }

}

public static void main(String[] args) {

    MultiThreadExtendsThread thread1 = new MultiThreadExtendsThread("A");

    MultiThreadExtendsThread thread2 = new MultiThreadExtendsThread("B");

    thread1.start();

    thread2.start();

}

}
```

二是实现 Runnable 接口，然后重写 run() 方法，

```
package com.csu.multiThread;

public class MultiThreadImplRunnable implements Runnable{

    String name;

    public MultiThreadImplRunnable(String name) {

        this.name = name;

    }

    @Override
```



```
public void run() {

    for(int i=0;i<5;i++) {

        System.out.println(name+"运行："+i);

    }

}

public static void main(String[] args) {

    MultiThreadExtendsThread thread1 = new MultiThreadExtendsThread("A");

    MultiThreadExtendsThread thread2 = new MultiThreadExtendsThread("B");

    new Thread(thread1).start();

    new Thread(thread2).start();

}

}
```

线程安全的实现：

最基本的：`synchronized` 关键字。这个方法是最常用的，它通过互斥的方式保证同步。我们知道 java 中有几个操作是可以保证原子性的，其中 `lock/unlock` 就是一对。虽然 java 没有提供这两个字节码的接口，但是我们可以通过 `monitorenter/monitorexit`，而 `synchronized` 会在块的前后调用两个字节码指令。同时 `synchronize` 对于同一条线程来说是可重入的；其次它也是阻塞的。我们知道 java 线程是映射到操作系统上的，而且是混用的内核态线程和用户态线程（N:M），而将线程从阻塞/唤醒，需要将线程从用户态转换到内核态，这样会消耗太亮的资源，所以 `synchronize` 是一个重量级锁

另外一种和 `synchronize` 类似的方法：`ReentrantLock`。它们两个的区别：（1）`synchronize` 是隐式的，只要块内的代码执行完，就会释放当前的锁；而后者需要显式的调用 `unlock()` 方法手动释放，所以经常搭配 `try/finally` 方法（忘记在 `finally` 中 `unlock` 是非常危险的）（2）后者可以选择等待中断——即在当前持有锁线程长期不释放锁的情况下，正在等待的线程可以选择放弃等待选择处理其他的事情。（3）后者可以选择公平锁（虽然默认是非公平的，因为公平锁的吞吐量很受影响）即先来后到，按申请的顺序获得锁。（4）可以绑定多个条件

前面提到的两种方式都是通过互斥来达到同步的目的，这其实是悲观锁的一种。下面介绍的是乐观锁，基于冲突检测的并发策略，不需要将线程挂起，因此又被成为非阻塞同步。



典型：CAS（Compare And Swap），通过 Unsafe 类提供。有三个操作数，内存位置、旧的预期值、和新的值；当且仅当内存地址 V 符合预期值 A 时，执行将值更新为新的预期值 B。存在的问题：“ABA”情况，即原值为 A，但在检测之前发生了改变，变成了 B，同时也在检测时变回了 A；即不能保证这个值没有被其他线程更改过。

接下来是无同步方案：

可重入代码（纯代码）：是一种无同步方案，在执行的任何时候去中断，转而执行其他的代码；在重新返回代码后，不会出现任何的错误。可重入性→线程安全，充分不必要条件。即可重入性的代码都是线程安全的，但反之不一定。简单判断原则：一个方法的返回结果是可预测的，只要输入了相同的数据，就都能返回相同的结果。

线程本地存储：即利用 ThreadLocal 类；每个 Thread 类中都有一个变量 ThreadLocalMap，默认是为 null 的。它将为每一个线程创立一个该变量的副本。这样线程之间就不存在数据征用的问题了。适用情况：（1）数据库的 Connection 连接 （2）WEB 中的“一个请求对应一个服务器线程”，在知乎上看到一个回答，解释的蛮清晰的。（3）Spring 中创建的默认模式是 Singleton 单例 （4）“生产者-消费者问题”

ThreadLocal 就是变量在不同线程上的副本，不同线程不共享，所以对变量改动时就不需要考虑线程间同步的问题了

ThreadLocal 在 web 应用开发中是一种很常见的技巧，当 web 端采用无状态写法时（比如 stateless session bean 和 spring 默认的 singleton），就可以考虑把一些变量放在 ThreadLocal 中

举个简单例子，以理解意思为主：你有两个方法 A 和 B 都要用到变量 userId，又不想传来传去，一个很自然的想法就是把 userId 设为成员变量，但是在无状态时，这样做就很可能有问题，因为多个 request 在同时使用同一个 instance，userId 在不同 request 下值是不一样的，就会出现逻辑错误

但由于同一个 request 下一般都是处于同一个线程，如果放在 ThreadLocal 的话，这个变量就被各个方法共享了，而又不影响其他 request，这种情况下，你可以简单把它理解为是一种没有副作用的成员变量（作者：卡斯帕尔）

线程栈 线程的每个方法被执行的时候，都会同时创建一个帧（Frame）用于存储本地变量表、操作栈、动态链接、方法出入口等信息。每一个方法的调用至完成，就意味着一个帧在 VM 栈中的入栈至出栈的过程。如果线程请求的栈深度大于虚拟机所允许的深度，将抛出 StackOverflowError 异常；如果 VM 栈可以动态扩展（VM Spec 中允许固定长度的 VM 栈），当扩展时无法申请到足够内存则抛出 OutOfMemoryError 异常。

### 37、请你说一说内存溢出和内存泄漏是怎么回事

考察点：计算机结构组成

参考回答：





内存溢出 out of memory，是指程序在申请内存时，没有足够的内存空间供其使用，出现 out of memory；比如申请了一个 integer，但给它存了 long 才能存下的数，那就是内存溢出。

内存泄露 memory leak，是指程序在申请内存后，无法释放已申请的内存空间，一次内存泄露危害可以忽略，但内存泄露堆积后果很严重，无论多少内存，迟早会被占光。

内存泄漏可以分为四类：

- 1、常发性内存泄漏，发生内存泄漏的代码会被多次执行到，每次执行都会导致内存泄漏
- 2、偶发性内存泄漏：发生内存泄漏的代码只有在某些特定环境或操作过程下才会发生，
- 3、一次性内存泄漏，发生内存泄漏的代码只会被执行一次，或者由于算法上的缺陷，导致总会有一块仅且一块内存发生泄漏。
- 4、隐式内存泄漏。程序在运行过程中不停的分配内存，但是直到结束的时候才释放内存。

从用户使用程序的角度来看，内存泄漏本身不会产生什么危害，真正有危害的是内存泄漏的堆积，这会最终消耗尽系统所有的内存。

内存溢出常见原因：

1. 内存中加载的数据量过于庞大，如一次从数据库取出过多数据；
2. 集合类中有对对象的引用，使用完后未清空，使得 JVM 不能回收；
3. 代码中存在死循环或循环产生过多重复的对象实体；
4. 使用的第三方软件中的 BUG；
5. 启动参数内存值设定的过小

解决方案：

- 1、修改 JVM 参数，直接增加内存
- 2、检查错误日志，查看内存溢出错误前是否有其他异常错误
- 3、对代码进行走查分析，找出可能发生内存溢出的位置

### 38、请你介绍一下 HashMap，HashTable，ConcurrentHashMap

考察点：数据结构

参考回答：

- 1、HashTable 与 HashMap



(1) `HashTable` 和 `HashMap` 都实现了 `Map` 接口，但是 `HashTable` 的实现是基于 `Dictionary` 抽象类。

(2) 在 `HashMap` 中，`null` 可以作为键，这样的键只有一个；可以有一个或多个键所对应的值为 `null`。当 `get()` 方法返回 `null` 值时，既可以表示 `HashMap` 中没有该键，也可以表示该键所对应的值为 `null`。因此，在 `HashMap` 中不能由 `get()` 方法来判断 `HashMap` 中是否存在某个键，而应该用 `containsKey()` 方法来判断。而在 `HashTable` 中，无论是 `key` 还是 `value` 都不能为 `null`。

(3) `HashTable` 是线程安全的，它的方法是同步了的，可以直接用在多线程环境中。`HashMap` 则不是线程安全的。在多线程环境中，需要手动实现同步机制。

2、更好的选择：`ConcurrentHashMap` `Java 5` 中新增了 `ConcurrentMap` 接口和它的实现类 `ConcurrentHashMap`。`ConcurrentHashMap` 提供了和 `HashTable` 以及 `SynchronizedMap` 中所不同的锁机制。`HashTable` 中采用的锁机制是一次锁住整个 `hash` 表，从而同一时刻只能有一个线程对其进行操作；而 `ConcurrentHashMap` 中则是一次锁住一个桶。`ConcurrentHashMap` 默认将 `hash` 表分为 16 个桶，诸如 `get`, `put`, `remove` 等常用操作只锁住当前需要用到的桶。这样，原来只能一个线程进入，现在却能同时有 16 个写线程执行，并发性能的提升是显而易见的。上面说到的 16 个线程指的是写线程，而读操作大部分时候都不需要用到锁。只有在 `size` 等操作时才需要锁住整个 `hash` 表。在迭代方面，`ConcurrentHashMap` 使用了一种不同的迭代方式。在这种迭代方式中，当 `iterator` 被创建后集合再发生改变就不再是抛出 `ConcurrentModificationException`，取而代之的是，在改变时 `new` 新的数据从而不影响原有的数据，`iterator` 完成后再将头指针替换为新的数据，这样 `iterator` 可以使用原来老的数据，而写线程也可以并发的完成改变。

### 39、请你说一下 `HashSet` 有什么特性，以及 `hashset` 判断存入的对象是否重复是如何比较的

考察点：数据结构

参考回答：

`HashSet` 是 `Set` 接口的实现类，因此，`HashSet` 中的元素也是不能重复的。`HashCode` 判断元素重复的标准时，首先计算新添加元素的 `hashCode` 值，当不重复是，则直接加入到该集合中，若发生重复，也称发生了碰撞，则进一步调用 `equals` 判断元素是否在逻辑上相同。

### 40、请你说一下 `Java` 的反射，你目前主要用他做什么，以及 `Java` 的泛型，他的主要作用是什么

考察点：`Java`

参考回答：

`JAVA` 反射机制是在运行状态中，对于任意一个类，都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意方法和属性；这种动态获取信息以及动态调用对象方法的功能称为 `java` 语言的反射机制。`Java` 反射可以用来获取一个 `class` 对象或实例化一个 `class` 表示的类的对象，还可以获取构造方法，成员变量，成员方法。



java 中泛型的引入主要是为了解决两个方面的问题：1. 集合类型元素在运行期出现类型转换异常，增加编译时类型的检查，2. 解决的时重复代码的编写，能够复用算法。下面通过例子来说明编译器的类型检查。

#### 41、请问类加载器你了解吗

考察点：类加载器

参考回答：

类加载器就是把类加载阶段中的”通过一个类的全限定名来获取描述此类的二进制字节流”这个动作放到 java 虚拟机外部来实现的代码模块。

类加载器的分类：

启动类加载器、扩展类加载器、应用类加载器（系统类加载器）、用户自定义类加载器。

启动类加载器：这个类负责将存放在 JAVA\_HOME/lib 目录或者被-Xbootclasspath 参数所指定的路径中的并且是虚拟机内存中。

扩展类加载器：负责加载 JAVA\_HOME/lib/ext 目录中或者被 java.ext.dirs 系统变量指定路径中的所有类库，开发者可以直接使用扩展类加载器。

应用程序类加载器：负责加载用户类路径上指定的类加载器，一般情况下就是程序中默认的类加载器。

#### 42、ReentrantLock 和 synchronized 有什么区别

考察点：操作系统

参考回答：

##### 1. 可重入性

字面的意思就是可以再次进入的锁，synchronized 其实也是可重锁，同一线程每进入一次，锁的计数器都会加一，在释放锁是计数器都会减一，只有当计数器为 0 时才能释放锁

##### 2. 锁的实现

ReentrantLock 是 JDK 实现的 Synchronized 是 JVM 实现前者可以直接看到源码，后者实现难以看到

##### 3. 性能的区别

在 Synchronized 优化以前，synchronized 的性能是比 ReentrantLock 差很多的，但是自从 Synchronized 引入了偏向锁，轻量级锁（自旋锁）后，两者的性能就差不多了，在两种方法都可用的情况下，官方甚至建议使用 synchronized，其实 synchronized 的优化我感觉就借鉴了

ReentrantLock 中的 CAS 技术。都是试图在用户态就把加锁问题解决，避免进入内核态的线程阻塞。

#### 4. 功能的区别

便利性：很明显 Synchronized 的使用比较方便简洁，并且由编译器去保证锁的加锁和释放，而 ReentrantLock 需要手工声明来加锁和释放锁，为了避免忘记手工释放锁造成死锁，所以最好在 finally 中声明释放锁。

锁的细粒度和灵活性：很明显 ReentrantLock 优于 Synchronized

当你需要时候一下三种功能是需要使用 ReentrantLock

ReentrantLock 可以指定公平锁还是非公平锁  
(公共锁就是先等待的线程先获得锁)

实现自旋锁，通过循环调用 CAS 操作来实现加锁，性能比较好，避免进入内核态的线程阻塞。

提供了 Condition 类，可以分组唤醒需要唤醒的线程

提供能够中断等待锁的线程的机制，lock.lockInterruptibly()

具体使用场景要根据实际的业务进行分析  
使用 Synchronized 时不需要释放锁，jvm 会帮助我们做释放锁的操作

### 43、请问 object 的 hash 该怎么设计

参考回答：

略

## 2、c++

### 1、请你说一下数组和指针的区别

考点：C/C++

参考回答：

数组：数组是用于储存多个相同类型数据的集合。

指针：指针相当于一个变量，但是它和不同变量不一样，它存放的是其它变量在内存中的地址。

区别：

- 赋值：同类型指针变量可以相互赋值，数组不行，只能一个一个元素的赋值或拷贝



- 存储方式：数组：数组在内存中是连续存放的，开辟一块连续的内存空间。数组是根据数组的下标进行访问的，多维数组在内存中是按照一维数组存储的，只是在逻辑上是多维的。指针：指针很灵活，它可以指向任意类型的数据。指针的类型说明了它所指向地址空间的内存。

- 求 sizeof：数组所占存储空间的内存：sizeof（数组名），数组的大小：sizeof（数组名）/sizeof（数据类型）。在 32 位平台下，无论指针的类型是什么，sizeof（指针名）都是 4，在 64 位平台下，无论指针的类型是什么，sizeof（指针名）都是 8。

- 初始化方式不同。

- 传参方式：数组传参时，会退化为指针，C 语言将数组的传参进行了退化。将整个数组拷贝一份传入函数时，将数组名看做常量指针，传数组首元素的地址。一级指针传参可以接受的参数类型：（1）可以是一个整形指针（2）可以是整型变量地址（3）可以是一维整型数组数组名；当函数参数部分是二级指针时，可以接受的参数类型：（1）二级指针变量（2）一级指针变量地址（3）一维指针数组的数组名

## 2、请你说一说 STL 常用的容器

考点：c++

参考回答：

（1）vector

vector 是一种动态数组，在内存中具有连续的存储空间，支持快速随机访问。由于具有连续的存储空间，所以在插入和删除操作方面，效率比较慢。vector 有多个构造函数，默认的构造函数是构造一个初始长度为 0 的内存空间，且分配的内存空间是以 2 的倍数动态增长的，即内存空间增长是按照 2<sup>0</sup>, 2<sup>1</sup>, 2<sup>2</sup>, 2<sup>3</sup>... 增长的，在 push\_back 的过程中，若发现分配的内存空间不足，则重新分配一段连续的内存空间，其大小是现在连续空间的 2 倍，再将原先空间中的元素复制到新的空间中，性能消耗比较大，尤其是当元素是非内部数据时（非内部数据往往构造及拷贝构造函数相当复杂）。

API：

```
vector<T> v; //采用模板实现类实现，默认构造函数
```

```
assign(begin(), end()); //将【begin(), end()】区间中的元素拷贝给本身
```

```
size(); //返回元素容器中元素个数
```

```
at(int idx); //返回索引 idx 所指的数据，如果 idx 越界，抛出 out_of_range 异常
```

（2）deque

deque 和 vector 类似，支持快速随机访问。二者最大的区别在于，vector 只能在末端插入数据，而 deque 支持双端插入数据。deque 的内存空间分布是小片的连续，小片间用链表相连，



实际上内部有一个 map 的指针。deque 空间的重新分配要比 vector 快，重新分配空间后，原有的元素是不需要拷贝的。

API:

```
deque<T>deqT;//默认构造形式
```

```
assign(begin,end);//将【begin，end】区间的数据拷贝赋值给自身
```

```
deque.size();//返回容器中元素的个数
```

```
push_back(elem);//在容器尾部添加一个数据
```

(3) list

list 是一个双向链表，因此它的内存空间是可以不连续的，通过指针来进行数据的访问，这使 list 的随机存储变得非常低效，因此 list 没有提供[]操作符的重载。但 list 可以很好地支持任意地方的插入和删除，只需移动相应的指针即可。

API:

```
list<T>lstT;//list 采用模板类实现对象的默认构造函数
```

```
push_back(elem);//在容器尾部加入一个元素
```

```
size();//返回元素容器中元素个数
```

```
empty();//判断容器是否为空
```

(4) map

map 是一种关联容器，该容器用唯一的关键字来映射相应的值，即具有 key-value 功能。map 内部自建一棵红黑树（一种自平衡二叉树），这棵树具有数据自动排序的功能，所以在 map 内部所有的数据都是有序的，以二叉树的形式进行组织。

API:

```
map<T1,T2>mapT;//map 默认构造函数
```

```
size();//返回元素中元素的数目
```

```
empty();//判断容器是否为空
```

```
clear();//删除所有元素
```

(5) set

set 也是一种关联性容器，它同 map 一样，底层使用红黑树实现，插入删除操作时仅仅移动指针即可，不涉及内存的移动和拷贝，所以效率比较高。set 中的元素都是唯一的，而且默认情况下会对元素进行升序排列。所以在 set 中，不能直接改变元素值，因为那样会打乱原本正确的





顺序，要改变元素值必须先删除旧元素，再插入新元素。不提供直接存取元素的任何操作函数，只能通过迭代器进行间接存取。

API:

```
set<T>sT;//set 默认构造函数
```

```
size();//返回容器中元素的数目
```

```
empty();//判断容器是否为空
```

```
insert(elem);//在容器中插入元素
```

```
clear();//清除所有元素
```

(6) queue

queue 是一个队列，实现先进先出功能，queue 不是标准的 STL 容器，却以标准的 STL 容器为基础。queue 是在 deque 的基础上封装的。之所以选择 deque 而不选择 vector 是因为 deque 在删除元素的时候释放空间，同时在重新申请空间的时候无需拷贝所有元素。

API:

```
queue<T>queT;//queue 采用模板类实现，queue 对象的默认构造形式
```

```
push(elem);//往队尾添加元素
```

(7) stack

stack 是实现先进后出的功能，和 queue 一样，也是内部封装了 deque，这也是为啥称为容器适配器的原因吧（纯属猜测）。自己不直接维护被控序列的模板类，而是它存储的容器对象来为它实现所有的功能。stack 的源代码原理和实现方式均跟 queue 相同。

API:

```
stack<T>stkT;//采用模板类实现，stack 对象的默认构造形式
```

```
push(elem);//向栈顶添加元素
```

### 3、请你说一下虚函数

考点: C++

参考回答:

拥有 Virtual 关键字的函数称之为虚函数，虚函数的作用是实现动态绑定的，也就是说程序在运行的时候动态的选择合适的成员函数。要成为虚函数必须满足两点，一就是这个函数依赖于对象调用，因为虚函数就是依赖于对象调用，因为虚函数是存在于虚函数表中，有一个虚函数指针指向这个虚表，所以要调用虚函数，必须通过虚函数指针，而虚函数指针是存在于对象中





的。二就是这个函数必须可以取地址，因为我们的虚函数表中存放的是虚函数函数入口地址，如果函数不能寻址，就不能成为虚函数。

#### 4、请你说一下动态内存分配

考点：C++

参考回答：

在程序执行的过程中动态地分配或者回收存储空间的分配内存的方法。动态内存分配不象数组等静态内存分配方法那样需要预先分配存储空间，而是由系统根据程序的需要即时分配，且分配的大小就是程序要求的大小。

#### 5、请你说一下深 copy 浅 copy

考点：C++

参考回答：

浅拷贝是增加了一个指针，指向原来已经存在的内存。而深拷贝是增加了一个指针，并新开辟了一块空间让指针指向这块新开辟的空间。浅拷贝在多个对象指向一块空间的时候，释放一个空间会导致其他对象所使用的空间也被释放了，再次释放便会出现错误。

#### 6. 请你说一下 C 中申请和释放内存的方法

考点：C/C++

参考回答：

C++:

new 运算符申请内存：

将调用相应的 `operator new(size_t)` 函数动态分配内存，在分配到的动态内存块上 初始化 相应类型的对象（构造函数）并返回其首地址。如果调用构造函数初始化对象时抛出异常，则自动调用 `operator delete(void*, void*)` 函数释放已经分配到的内存。

delete 运算符释放内存：

调用相应类型的析构函数，处理类内部可能涉及的资源释放，调用相应的 `operator delete(void *)` 函数。

C:

内存区域可以分为栈，堆，静态存储区和常量存储区。局部变量，函数形参，临时变量都是在栈上获得内存的，它们获取的方式都是由编译器自动执行的。



而 C 标准函数库提供了许多函数来实现对堆上内存管理，其中包括：malloc 函数，free 函数，calloc 函数和 realloc 函数。使用这些函数需要包含头文件 stdlib.h。

#### (1) malloc 函数

malloc 函数可以从堆上获得指定字节的内存空间，其函数声明如下：

```
void * malloc(int n);
```

其中，形参 n 为要求分配的字节数。如果函数执行成功，malloc 返回获得内存空间的首地址；如果函数执行失败，那么返回值为 NULL。由于 malloc 函数值的类型为 void 型指针，因此，可以将其值类型转换后赋给任意类型指针，这样就可以通过操作该类型指针来操作从堆上获得的内存空间。需要注意的是，malloc 函数分配得到的内存空间是未初始化的。因此，一般在使用该内存空间时，要调用另一个函数 memset 来将其初始化为全 0。memset 函数的声明如下：

```
void * memset (void * p,int c,int n) ;
```

该函数可以将指定的内存空间按字节单位置为指定的字符 c。其中，p 为要清零的内存空间的首地址，c 为要设定的值，n 为被操作的内存空间的字节长度。

#### (2) free 函数

从堆上获得的内存空间在程序结束以后，系统不会将其自动释放，需要程序员来自己管理。一个程序结束时，必须保证所有从堆上获得的内存空间已被安全释放，否则，会导致内存泄露。

```
void free (void * p);
```

由于形参为 void 指针，free 函数可以接受任意类型的指针实参。

但是，free 函数只是释放指针指向的内容，而该指针仍然指向原来指向的地方，此时，指针为野指针，如果此时操作该指针会导致不可预期的错误。安全做法是：在使用 free 函数释放指针指向的空间之后，将指针的值置为 NULL。

#### (3) calloc 函数

calloc 函数的功能与 malloc 函数的功能相似，都是从堆分配内存。其函数声明如下：

```
void *calloc(int n,int size);
```

函数返回值为 void 型指针。如果执行成功，函数从堆上获得 size X n 的字节空间，并返回该空间的首地址。如果执行失败，函数返回 NULL。该函数与 malloc 函数的一个显著不同是，\*\*calloc 函数得到的内存空间是经过初始化的，其内容全为 0。\*\*calloc 函数适合为数组申请空间，可以将 size 设置为数组元素的空间长度，将 n 设置为数组的容量。

#### (4) realloc 函数

realloc 函数的功能比 malloc 函数和 calloc 函数的功能更为丰富，可以实现内存分配和内存释放的功能，其函数声明如下：



```
void * realloc(void * p, int n);
```

其中，指针 `p` 必须为指向堆内存空间的指针，即由 `malloc` 函数、`calloc` 函数或 `realloc` 函数分配空间的指针。`realloc` 函数将指针 `p` 指向的内存块的大小改变为 `n` 字节。如果 `n` 小于或等于 `p` 之前指向的空间大小，那么，保持原有状态不变。如果 `n` 大于原来 `p` 之前指向的空间大小，那么，系统将重新为 `p` 从堆上分配一块大小为 `n` 的内存空间，同时，将原来指向空间的内容依次复制到新的内存空间上，`p` 之前指向的空间被释放。`realloc` 函数分配的空间也是未初始化的。

## 7、请你说一说 C++和 C 的区别

考点：C/C++

参考回答：

- C 是面向过程的语言，而 C++ 是面向对象的语言
- C 和 C++ 动态管理内存的方法不一样，C 是使用 `malloc/free` 函数，而 C++ 除此之外还有 `new/delete` 关键字
- C 中的 `struct` 和 C++ 的类，C++ 的类是 C 所没有的，但是 C 中的 `struct` 是可以在 C++ 中正常使用的，并且 C++ 对 `struct` 进行了进一步的扩展，使 `struct` 在 C++ 中可以和 `class` 一样当做类使用，而唯一和 `class` 不同的地方在于 `struct` 的成员默认访问修饰符是 `public`，而 `class` 默认的是 `private`；
- C++ 支持函数重载，而 C 不支持函数重载，而 C++ 支持重载的依仗就在于 C++ 的名字修饰与 C 不同，例如在 C++ 中函数 `int fun(int, int)` 经过名字修饰之后变为 `_fun_int_int`，而 C 是 `_fun`，一般是这样的，所以 C++ 才会支持不同的参数调用不同的函数；
- C++ 中有引用，而 C 没有；
- C++ 全部变量的默认链接属性是外链接，而 C 是内连接；
- C 中用 `const` 修饰的变量不可以用在定义数组时的大小，但是 C++ 用 `const` 修饰的变量可以

## 8、请你回答一下 C++ 中的多态是怎么实现的

考点：C++

参考回答：

C++ 的多态性是用虚函数和延迟绑定来实现的，在基类的函数前加上 `virtual` 关键字，在派生类中重写该函数，运行时将会根据对象的实际类型来调用相应的函数。如果对象类型是派生类，就调用派生类的函数；如果对象类型是基类，就调用基类的函数。



1. 用 `virtual` 关键字申明的函数叫做虚函数，虚函数肯定是类的成员函数。
2. 存在虚函数的类都有一个一维的虚函数表叫做虚表。类的对象有一个指向虚表开始的虚指针。虚表是和类对应的，虚表指针是和对象对应的。
3. 多态性是一个接口多种实现，是面向对象的核心。分为类的多态性和函数的多态性。
4. 多态用虚函数来实现，结合动态绑定。
5. 纯虚函数是虚函数再加上 `= 0`。
6. 抽象类是指包括至少一个纯虚函数的类。

### 9、请你说一下 C 语言的内存分配

考察点：测试实践

参考回答：

在 C 语言中，对象可以使用静态或动态的方式分配内存空间

静态分配：编译器在处理程序源代码时分配，由于是在程序执行之前进行所以效率比较高

动态分配：程序在执行时调用 `malloc` 库函数申请分配，可以灵活的处理未知数目的对象

静态与动态内存分配的主要区别如下：

静态对象是有名字的变量，可以直接对其进行操作；动态对象是没有名字的变量，需要通过指针间接地对它进行操作。

静态对象的分配与释放由编译器自动处理；动态对象的分配与释放必须由程序员显式地管理，它通过 `malloc()` 和 `free` 两个函数（C++中为 `new` 和 `delete` 运算符）来完成。

### 10、请你回答一下什么是指针，以及指针和数组的区别，指针和引用的区别

考察点：语言

参考回答：

指针就是一个存放地址的变量，当指针指向某个变量，这时这个指针里就存放了那个变量的地址

指针与数组的区别

- 1、指针和数组的分配，数组是开辟一块连续的内存空间，指针不是



2、空间分配，这里又分为两种情况。

第一，如果是全局的和静态的

```
char *p = "hello";
```

这是定义了一个指针，指向 rodata section 里面的“hello”，可以被编译器放到字符串池。在汇编里面的关键字为 `.ltorg`。意思就是在字符串池里的字符串是可以共享的，这也是编译器优化的一个措施。

```
char a[] = "hello";
```

这是定义了一个数组，分配在可写数据块，不会被放到字符串池。

第二，如果是局部的

```
char *p = "hello";
```

这是定义了一个指针，指向 rodata section 里面的“hello”，可以被编译器放到字符串池。在汇编里面的关键字为 `.ltorg`。意思就是在字符串池里的字符串是可以共享的，这也是编译器优化的一个措施。另外，在函数中可以返回它的地址，也就是说，指针是局部变量，但是它指向的内容是全局的。

```
char a[] = "hello";
```

这是定义了一个数组，分配在堆栈上，初始化由编译器进行。（短的时候直接用指令填充，长的时候就从全局字符串表拷贝），不会被放到字符串池（同样如前，可能会从字符串池中拷贝过来）。注意不应该返回它的地址。

### 3、使用方法

如果是全局指针，用于不需要修改内容，但是可能会修改指针的情况。如果是全局数组，用于不需要修改地址，但是却需要修改内容的情况。如果既需要修改指针，又需要修改内容，那么就定义一个数组，再定义一个指针指向它就可以了。

#### 指针和引用的区别

1、指针是一个变量，只不过这个变量存储的是一个地址，而引用跟原来的变量实质上是一个东西，只不过是原变量的一个别名

2、引用不可以为空，当被创建的时候，必须初始化，而指针可以是空

3、指针可以有多级，但是引用只有一级

4、指针的值在初始化后可以改变，但是引用进行初始化后就不会再改变了

5、`sizeof` 引用得到的是指向变量的大小，而指针得到的是本身的大小

6、如果返回动态分配的对象或内存，必须使用指针，否则可能引起内存泄漏

## 11、请你说一下 `const` 和指针的区别，以及运算符优先级是怎么样

考察点:C++

参考回答：



const 和指针的区别

以下三个语法的区别

(1) `const int *p1;`

(2) `int const *p2;`

(3) `int *const p3;`

由于指针\*p的赋值方式有两种

第一种是：`p = &a;`

第二种是：`*p = a;`

(1)和(2)的效果是一样的。只能通过第一种方式来修改指针指向的变量

而(3)的方式是只能在一开始的时候指定一个变量，以后不能再指向其他变量。

其实主要是看 const 后面的变量是什么，只有 const 后面的变量无法修改

运算符优先级，简单来说就是！>算术运算符>关系运算符>&&>||>赋值运算符

## 12、手写代码：写一个程序算出 100 以内 7 的倍数

考察点：编程

参考回答：

```
void beishu()
{
    int i ;
    i=7;
    while(i<=100)
    {
        if((i%7)==0)
            printf("%d",i);
        i+=7;
    }
}
```

## 13、手写代码：写一个函数，不用加法和乘法，返回他的八倍

考察点：编程

参考回答：

```
import org.junit.Test;
public class solution {
```



```
@Test
public void testFunc() {
    int i = 1;
    int res = beiShu(i);
    System.out.println("res: "+res);
}

public int beiShu(int x){
    return x<<3;
}

}
```

#### 14、请你说一下 new 和 malloc 的区别

考察点：编程语言

参考回答：

1、属性：new 是 C++关键字，需要编译器支持，malloc 是库函数，需要头文件支持

2、参数：使用 new 操作符申请内存分配时无需指定内存块的大小，编译器会根据类型信息自行计算，malloc 则需要显式的指出所需内存的尺寸

3、返回类型：new 返回的是对象类型的指针，malloc 返回 void\*，需要通过强制类型转换将 void\*指针转换成我们需要的类型

4、分配失败：new 内存分配失败时，会跑出 bad\_alloc 异常，malloc 分配内存失败返回 null

5、 自定义类型

new 会先调用 operator new 函数，申请足够的内存（通常底层使用 malloc 实现）。然后调用类型的构造函数，初始化成员变量，最后返回自定义类型指针。delete 先调用析构函数，然后调用 operator delete 函数释放内存（通常底层使用 free 实现）。

malloc/free 是库函数，只能动态的申请和释放内存，无法强制要求其做自定义类型对象构造和析构工作。

6、 重载

C++允许重载 new/delete 操作符，特别的，布局 new 的就不需要为对象分配内存，而是指定了一个地址作为内存起始区域，new 在这段内存上为对象调用构造函数完成初始化工作，并返回此地址。而 malloc 不允许重载。

7、内存区域





new 操作符从自由存储区（free store）上为对象动态分配内存空间，而 malloc 函数从堆上动态分配内存。自由存储区是 C++ 基于 new 操作符的一个抽象概念，凡是通过 new 操作符进行内存申请，该内存即为自由存储区。而堆是操作系统中的术语，是操作系统所维护的一块特殊内存，用于程序的内存动态分配，C 语言使用 malloc 从堆上分配内存，使用 free 释放已分配的对应内存。自由存储区不等于堆，如上所述，布局 new 就可以不位于堆中。

### 15、请你说一说 C++ 语言的三大特性

考察点：C++ 语言

参考回答：

封装，继承，多态

### 16、请你说一说虚函数和纯虚函数区别

考察点：语法

参考回答：

虚函数和纯虚函数区别

观点一：类里声明为虚函数的话，这个函数是实现的，哪怕是空实现，它的作用就是为了让这个函数在它的子类里面可以被重载，这样的话，这样编译器就可以使用后期绑定来达到多态了。纯虚函数只是一个接口，是个函数的声明而已，它要留到子类里去实现。

```
class A{
protected:
void foo(); //普通类函数
virtual void foo1(); //虚函数
virtual void foo2() = 0; //纯虚函数
}
```

观点二：虚函数在子类里面也可以不重载的；但纯虚必须在子类去实现，这就像 Java 的接口一样。通常我们把很多函数加上 virtual，是一个好的习惯，虽然牺牲了一些性能，但是增加了面向对象的多态性，因为你很难预料到父类里面的这个函数不在子类里面不去修改它的实现

观点三：虚函数的类用于“实作继承”，继承接口的同时也继承了父类的实现。当然我们也可以完成自己的实现。纯虚函数的类用于“介面继承”，主要用于通信协议方面。关注的是接口的统一性，实现由子类完成。一般来说，介面类中只有纯虚函数的。

观点四：带纯虚函数的类叫虚基类，这种基类不能直接生成对象，而只有被继承，并重写其虚函数后，才能使用。这样的类也叫抽象类。

虚函数是为了继承接口和默认行为

纯虚函数只是继承接口，行为必须重新定义

### 17、请你说一下 static 作用

考察点：编程语言



参考回答：

Static 作用：

1、隐藏，当同时编译多个文件时，所有未加 static 的全局变量和函数都具有全局可见性。

2、保持变量内容的持久，存储在静态数据区的变量会在程序打开是运行时就完成初始化，也是唯一一次初始化，共有两种变量存储在静态存储区，全局变量和 static 变量，PS：如果 static 局部变量在函数内定义，他的生存期为整个源程序，但其作用域和自动变量相同，只能在定义该变量的函数内使用，退出该函数后，尽管该变量还存在，但是不能使用。

3、默认初始化为 0，

### 18、请问你怎么理解多态，他有什么好处

考察点：C++

参考回答：

所谓多态，就是指程序中定义的引用变量所指向的具体类型和通过该引用变量发出的方法调用在编程时并不确定，而是在程序运行时确定，即一个引用变量到底会指向哪个类的实例对象，调用哪个类的实现方法，由程序运行期间才确定，这样不用修改程序源代码就可以让引用变量绑定到各种不同的类实现上，从而导致该引用调用的具体方法随之改变，即不修改程序代码就可以改变程序运行时所绑定的具体代码，让程序可以选择多个运行状态，这就是多态性。

### 19、手写代码：求两个数的最大公约数

考察点：编程

参考回答：

```
int division(int n,int m)
{
    if(n<m)
        division 考(m,n); //交换 m 与 n
    else if(m==0)
        return n;
    else
    {
        int temp=n;
```



```
        n=m;

        m=temp%n;

        division(n,m); //重复上述过程

    }

}
```

**20、手写代码：将字符串转 int 类型，要求不能用已有的方法**

考察点：编程

参考回答：

```
public static int stringToInt(String str) {

    int result=0;

    char[] ch=str.toCharArray();

    int len=ch.length;

    for(int i=0;i<len;i++) {

        result+=(((int)ch[i]-'0')*Math.pow(10, len-1-i));

    }

    return result;

}
```

**21、手写代码：求 x 的 n 次方**

考察点：编程

参考回答：

```
double Pow(double x, int n)

{

    double result = 1;

    while (n)

    {
```



```
        if (n & 1)           // 等价于 if (n % 2 != 0)

            result *= x;

        n >>= 1;             //右移一位相当于 n/2(类比十进制来理解)

        x *= x;

    }

    return result;

}
```

### 3、Python

#### 1、手写代码：比较两个 json 数据是否相等

考点：python

参考回答：

```
for src_list, dst_list in zip(sorted(dict1), sorted(dict2)):

    if str(dict1[src_list]) != str(dict2[dst_list]):

        print(src_list, dict1[src_list], dst_list, dict2[dst_list])
```

#### 2、请问怎么拿到 python 的输入方式？以及 python 怎么打开文件？

考察点：python

参考回答:python 读取键盘输入通过 input()内置函数,读文件则通过 open(filename, mode)

Python 打开文件 f = open('your\_file.txt', 'r')

就可以打开一个文件进行操作。第二个参数为对文件的操作方式，'w' 是写文件，已存在的同名文件会被清空，不存在则会创建一个；'r' 是读取文件，不存在会报错；'a' 是在文件尾部添加内容，不存在会创建文件，存在则直接在尾部进行添加；还有'wb' 是写二进制文件；'rb' 是读取二进制文件，比如图片之类的。

### 4、Shell

#### 1、请你说一下 shell 的基本命令，怎么看到行号?怎么查进程的 id?



考察点：shell

参考回答：

获取文本对应文本的行号, 可以用 grep, 也可以用 sed

```
grep -n "xxx" a.txt | cut -d ":" -f 1
```

```
sed -n -e '/xxx/= ' a.txt
```

shell 获取进程 ID 的方法有三种：

- 1、ps -A | grep "cmdname" | awk '{print \$1}'
- 2、pidof "cmdname"
- 3、pgrep "cmdname"

2、请你回答一下常用到的 shell 指令中与网络相关的有哪些，netstat、ping、ifconfig 这三个的区别，分别是什么功能，netstat 里面一般服务器启动后的端口状态是什么

考察点：shell

参考回答：

常用的网络相关的命令有以下几个：

1、ifconfig

这个命令用于显示网络接口，子网掩码

2、host 和 nslookup

这两个命令是 DNS 查找工具，当执行 host 时，会列出某个域名的所有 ip，nslookup 是一个类似于 host 的命令，它用于查询 DNS 相关的细节信息，以及名字解析

3、route

显示路由表

4、traceroute

这个命令显示分组途径的所有网关地址

netstat、ping、ifconfig 这三个的区别：

netstat: 显示网络状态，利用 netstat 可以让你得知整个 Linux 系统的网络情况，语法为 netstat [-acCeFghilMnNoprstuvVwx] [-A<网络类型>] [--ip]

ping: 功能是检测主机，因为执行 ping 命令会使用 icmp 传输协议，发出要求回应的信息，若远端主机的网络功能没有问题，就会回应该信息，因而得知该主机运作正常，语法为：



ping [-dfnqrRv] [-c<完成次数>] [-i<间隔秒数>] [-I<网络界面>] [-l<前置载入>] [-p<范本样式>] [-s<数据包大小>] [-t<存活数值>] [主机名称或 IP 地址]

ifconfig: 功能是显示或设置网络设备，其语法为: ifconfig [网络设备] [down up -allmulti -arp -promisc] [add<地址>] [del<地址>] [<hw<网络设备类型><硬件地址>] [io\_addr<I/O 地址>] [irq<IRQ 地址>] [media<网络媒介类型>] [mem\_start<内存地址>] [metric<数目>] [mtu<字节>] [netmask<子网掩码>] [tunnel<地址>] [-broadcast<地址>] [-pointopoint<地址>] [IP 地址]

服务器启动后一般为 listening 状态

## 四、数据结构与算法

### 1、链表

#### 1、请你说出几种基本的数据结构，

考察点：数据结构

参考回答：

常见的的基本的数据结构有链表、栈、队列、树（只列出面试常考的基本数据结构）

1、链表是一种物理存储单元上非连续、非顺序的存储结构，数据元素的逻辑顺序是通过链表中的指针链接次序实现的。链表由一系列节点组成，这些节点不必在内存中相连。每个节点由数据部分 Data 和链部分 Next，Next 指向下一个节点，这样当添加或者删除时，只需要改变相关节点的 Next 的指向，效率很高。

栈和队列是比较特殊的线性表

栈是限制插入和删除只能在一个位置上进行的表，后进先出

队列只允许在 front 端进行删除操作，在 rear 端进行插入操作，

树：树型结构是一类非常重要的非线性数据结构，考察主要以二叉树为主，

#### 2、手写代码：一个单向链表，每个节点上都有一个 value；给定 一个 value，将该 value 之前的所有 value 返回

考察点：编程

参考回答：略



### 3、手写代码：怎么判断链表有环，怎么找环节点，

考察点：编程

参考回答：

判断是否有环以及环节点

```
public class Solution {  
  
    ListNode EntryNodeOfLoop(ListNode h) {  
  
        if(h == null || h.next == null)  
  
            return null;  
  
        ListNode slow = h;  
  
        ListNode fast = h;  
  
        while(fast != null && fast.next != null ){  
  
            slow = slow.next;  
  
            fast = fast.next.next;  
  
            if(slow == fast){  
  
                ListNode p=h;  
  
                ListNode q=slow;//相当于让 q 指向了 m1  
  
                while(p != q){  
  
                    p = p.next;  
  
                    q = q.next;  
  
                }  
  
                if(p == q)  
  
                    return q;  
  
            }  
  
        }  
  
    }  
}
```





```
        return null;

    }
```

**4、手写代码：**一个单向链表，给出头结点，找出倒数第 N 个结点，要求  $O(N)$  的时间复杂度；

考点：算法

参考回答：

JAVA 版本：

```
public class Solution {

    public ListNode FindNthToTail(ListNode head,int N) {

        ListNode pre=null,p=null;

        //两个指针都指向头结点

        p=head;

        pre=head;

        //记录 N 值

        int a=N;

        //记录节点的个数

        int count=0;

        //p 指针先跑，并且记录节点数，当 p 指针跑了 N-1 个节点后，pre 指针开始跑，

        //当 p 指针跑到最后时，pre 所指指针就是倒数第 N 个节点

        while(p!=null){

            p=p.next;

            count++;

            if(N<1){

                pre=pre.next;

            }

            N--;

        }
```



```
}

//如果节点个数小于所求的倒数第 N 个节点，则返回空

if(count<a) return null;

return pre;

}

}

C/C++版本：

class Solution {

public:

ListNode* FindNthToTail(ListNode* pListHead, unsigned int N) {

if(pListHead==NULL || N==0)

return NULL;

ListNode*pTail=pListHead, *pHead=pListHead;

for(int i=1;i<N;++i)

{

if(pHead->next!=NULL)

pHead=pHead->next;

else

return NULL;

}

while(pHead->next!=NULL)

{

pHead=pHead->next;

pTail=pTail->next;

}

}
```



```
return pTail;

}

};

Python:

class Solution:

def FindNthToTail(self, head, N):

# write code here

res=[]

while head:

res.append(head)

head=head.next

if N>len(res) or N<1:

return

return res[-N]
```

### 5、请问如何判断一个单向链表存在回路？

考点：数据结构

参考回答：

方法 1：用一个指针数组 A，存储已访问过的节点。用一个指针 p，每次在链表上移动一步，然后与指针数组 A 比较，若数组中没有指针与 p 相同，说明第一次访问 p，将 p 放入数组中；若有指针与 p 相同，则存在环路，且第一次相同的节点就是环的入口点。

链表长度为 n，则需要空间  $O(n)$ ，且每次要与指针数组比较，时间复杂度为  $O(n^2)$ 。

方法 2：在节点上记录该节点是否被访问过，如果在指针移动过程中遇到已访问过的节点，说明存在环路。同样地，第一次相同的节点就是环的入口点。

方法 3：用两个指针，pSlow, pFast，一个慢一个快，慢的一次跳一步，，快的一次跳两步，如果快的能追上慢的就表示有环（pSlow == pFast ）。

### 6、请问如何判断一个链表是否有环



考点：数据结构

参考回答：

方法 1：用一个指针数组 A，存储已访问过的节点。用一个指针 p，每次在链表上移动一步，然后与指针数组 A 比较，若数组中没有指针与 p 相同，说明第一次访问 p，将 p 放入数组中；若有指针与 p 相同，则存在环路，且第一次相同的节点就是环的入口点。

链表长度为 n，则需要空间  $O(n)$ ，且每次要与指针数组比较，时间复杂度为  $O(n^2)$ 。

方法 2：在节点上记录该节点是否被访问过，如果在指针移动过程中遇到已访问过的节点，说明存在环路。同样地，第一次相同的节点就是环的入口点。

方法 3：用两个指针，pSlow, pFast，一个慢一个快，慢的一次跳一步，快的一次跳两步，如果快的能追上慢的就表示有环（pSlow == pFast）。

## 7、请问如何判断两个链表是否相交

考点：算法

参考回答：

从头遍历两个链表。创建两个栈，第一个栈存储第一个链表的节点，第二个栈存储第二个链表的节点。每遍历到一个节点时，就将该节点入栈。两个链表都入栈结束后。则通过 top 判断栈顶的节点是否相等即可判断两个单链表是否相交。因为我们知道，若两个链表相交，则从第一个相交节点开始，后面的节点都相交。若两链表相交，则循环出栈，直到遇到两个出栈的节点不相同，则这个节点的后一个节点就是第一个相交的节点。

```
node temp=NULL; //存第一个相交节点

while(!stack1.empty() && !stack2.empty()) //两栈不为空
{
    temp=stack1.top();
    stack1.pop();
    stack2.pop();

    if(stack1.top() != stack2.top())
    {
        break;
    }
}
```



```
}
```

**8、手写代码：将一个数组指定 offset 后面的元素前置，可采用链表实现**

考察点：编程

参考回答：略

**9、手写代码：循环链表插入元素**

考点：数据结构

参考回答：

```
typedef struct _tag_CircleListNode
{
    struct _tag_CircleListNode * next;
}CircleListNode;

typedef struct _tag_CircleList
{
    CircleListNode header;
    CircleListNode* slider;
    int length;
}TCircleList;

//插入元素

int CircleList_insert(CircleList* list, CircleListNode* node, int pos)
{
    int ret = 0, i=0;

    TCircleList* sList = (TCircleList*)list;
```



```
if (list == NULL || node == NULL || pos < 0)

{

return -1;

}

CircleListNode* current = (CircleListNode*)sList;

for(i=0; (i<pos) && (current->next != NULL); i++)

{

current = current->next;

}

//current->next 0 号节点的地址

node->next = current->next; //1

current->next = node; //2

//若第一次插入节点

if( sList->length == 0 )

{

sList->slider = node;

}

sList->length++;

//若头插法 current 仍然指向头部

//（原因是：跳 0 步，没有跳走） 中间第一种情况

if( current == (CircleListNode*)sList )

{

//获取最后一个元素

CircleListNode* last = CircleList_Get(sList, sList->length - 1);

last->next = current->next; //3
```



```
    }

    return ret;

}

CircleListNode* CircleList_Get(CircleList* list, int pos) // O(n)

{

    TCircleList* sList = (TCircleList*)list;

    CircleListNode* ret = NULL;

    int i = 0;

    if (list==NULL || pos<0)

    {

        return NULL;

    }

    {

        CircleListNode* current = (CircleListNode*)sList;

        for(i=0; i<pos; i++)

        {

            current = current->next;

        }

        ret = current->next;

    }

    return ret;

}
```





## 2、数组

### 1、手写代码：合并两个排序数组

考察点：编程

参考回答：

```
class Solution {  
  
public:  
  
    void merge(int *num1, int m, int *num2, int n) {  
  
        int i=m-1;  
  
        int j=n-1;  
  
        while(i>=0&&j>=0)  
  
        {  
  
            if(nums1[i]>nums2[j])  
  
            {  
  
                nums1[i+j+1]=nums1[i];  
  
                i--;  
  
            }  
  
            else{  
  
                nums1[i+j+1]=nums2[j];  
  
                j--;  
  
            }  
  
        }  
  
        while(j>=0)  
  
        {  
  
            nums1[i+j+1]=nums2[j];  
  
            j--;  
  
        }  
  
    }  
  
};
```



```
    }  
  
    }  
  
};
```

## 2、手写代码：最大子数组问题（要求时间复杂度最佳）

考点：算法

参考回答：

线性时间算法：该算法在每次元素累加和小于 0 时，从下一个元素重新开始累加。实现代码如下：

```
/*  
  
时间复杂度 O(n)  
  
和最大的子序列的第一个元素肯定是正数  
  
因为元素有正有负，因此子序列的最大和一定大于 0  
  
*/  
  
int MaxSubSum(int *arr, int len)  
{  
  
    int i;  
  
    int MaxSum = 0;  
  
    int CurSum = 0;  
  
    for(i=0; i<len; i++)  
    {  
  
        CurSum += arr[i];  
  
        if(CurSum > MaxSum)  
  
            MaxSum = CurSum;  
  
        //如果累加和出现小于 0 的情况，  
  
        //则和最大的子序列肯定不可能包含前面的元素，
```



```
//这时将累加和置 0，从下个元素重新开始累加

if (CurSum < 0)

CurSum = 0;

}

return MaxSum;

}
```

### 3、手写代码：筛选数组 arr 中重复的元素，考虑时间复杂度。

考点：编程

参考回答：

```
function duplicates(arr) {

//声明两个数组，a 数组用来存放结果，b 数组用来存放 arr 中每个元素的个数

var a = [], b = [];

//遍历 arr，如果以 arr 中元素为下标的 b 元素已存在，则该 b 元素加 1，否则设置为 1

for (var i = 0; i < arr.length; i++) {

if (!b[arr[i]]) {

b[arr[i]] = 1;

continue;

}

b[arr[i]]++;

}

//遍历 b 数组，将其中元素值大于 1 的元素下标存入 a 数组中

for (var i = 0; i < b.length; i++) {

if (b[i] > 1) {

a.push(i);

}
```



```
}  
  
}  
  
return a;  
  
}
```

时间复杂度为  $O(n)$

4、写出一个函数，输入是两个数组，输出是将两个数组中所有元素排序以后用一个数组输出。

考点：算法

参考回答：

```
class Solution {  
  
public:  
  
    int *sort(int *a, int lenA, int *b, int lenB) {  
  
        fastSort(a, 0, lenA);  
  
        fastSort(b, 0, lenB);  
  
        return merge(a, lenA, b, lenB);  
  
    }  
  
private:  
  
    //快速排序  
  
    void fastSort(int *a, int start, int end) {  
  
        if(a==NULL || end-start<=1 || start<0)  
  
            return;  
  
        int pivotPos = start;  
  
        int pivot = a[start];  
  
        int temp;  
  
        for(int i=start+1; i<end; ++i) {
```



```
        if(a[i]<pivot){

            if(++pivotPos!=i){

                temp = a[i];

                a[i] = a[pivotPos];

                a[pivotPos] = temp;

            }

        }

    }

    a[start] = a[pivotPos];

    a[pivotPos] = pivot;

    fastSort(a, start, pivotPos-1);

    fastSort(a, pivotPos+1, end);

}

//两路归并

int *merge(int *a,int lenA,int *b,int lenB){

    if(a==NULL || lenA<=0)

        return b;

    if(b==NULL || lenB<=0)

        return a;

    int *arry = new int[lenA+lenB];

    if(arry==NULL){

        cerr << "内存分配失败" << endl;

        exit(1);

    }

    int posA = 0, posB = 0 ,pos = 0;
```



```
while(posA<lenA && posB<lenB) {  
    if(a[posA]<b[posB])  
        array[pos++] = a[posA++];  
    else  
        array[pos++] = b[posB++];  
}  
while(posA<lenA)  
    array[pos++] = a[posA++];  
while(posB<lenB)  
    array[pos++] = b[posB++];  
return array;  
}  
};
```

5、请看一下这个小型的算法题，有一个长度未知的无序数组（无法获取数组长度，不许排序，数组元素为 int 范围内的任意值），如何找出第一个连续出现的重复的数（比如 4 3 4 3 5 5，找出的数是 5）

考察点：编程

参考回答：略

#### 6、手写代码：合并两个有序数组

考点：编程

参考回答：

解法一：

```
class Solution {  
public:  
    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {  
        int num[m+n]; //新建一个数组，对 nums1 和 nums2 排序，排完序赋值给 nums1
```



```
int i = 0, j = 0, k = 0;

while(i < m && j < n) {

    if(nums1[i] <= nums2[j])

        num[k++] = nums1[i++];

    else

        num[k++] = nums2[j++];

}

while(i < m) num[k++] = nums1[i++];

while(j < n) num[k++] = nums2[j++];

copy(num, num+m+n, nums1.begin());

}

};
```

解法二：

```
class Solution {

public:

    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {

        vector<int> num(m+n); //与解法一类似，不过是新建一个 vector

        int i = 0, j = 0, k = 0;

        while(i < m && j < n) {

            if(nums1[i] <= nums2[j])

                num[k++] = nums1[i++];

            else

                num[k++] = nums2[j++];

        }

        while(i < m) num[k++] = nums1[i++];
```





```
while(j < n) num[k++] = nums2[j++];

nums1.assign(num.begin(), num.end()); // nums1.swap(num) 也可以

}

};
```

解法三：直接在 nums1 里进行操作，从 nums1 的尾部开始，取 nums1 和 nums2 中的最大值放入其中。如果 n 先到达 0 就能直接得到合并好的数组；如果 m 先到达 0，只需将 n 剩下的元素复制到 nums1 中即可。

```
class Solution {

public:

void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {

int k=m+n;

while(m>0 && n>0){

if(nums1[m-1] >= nums2[n-1]){

nums1[k-1] = nums1[m-1];

--k;

--m;

}

else{

nums1[k-1] = nums2[n-1];

--k;

--n;

}

}

while(n > 0){

nums1[k-1] = nums2[n-1];

--k;
```



```
--n;  
  
}  
  
}  
  
};
```

## 7、手写代码：有三种面值的硬币 $k_1 < k_2 < k_3$ ，找 $k$ 面值的零钱，最少需要多少硬币

考点：算法

参考回答：

假设有 1 元，3 元，5 元的硬币，假设一个函数  $d(i)$  来表示需要凑出  $i$  的总价值需要的最少硬币数量。

当  $i = 0$  时， $d(0) = 0$ 。不需要凑零钱，当然也不需要任何硬币了。

当  $i = 1$  时，因为有 1 元的硬币，所以直接在第 1 步的基础上，加上 1 个 1 元硬币，得出  $d(1) = 1$ 。

当  $i = 2$  时，因为并没有 2 元的硬币，所以只能拿 1 元的硬币来凑。在第 2 步的基础上，加上 1 个 1 元硬币，得出  $d(2) = 2$ 。

当  $i = 3$  时，可以在第 3 步的基础上加上 1 个 1 元硬币，得到 3 这个结果。但其实有 3 元硬币，所以这一步的最优结果不是建立在第 3 步的结果上得来的，而是应该建立在第 1 步上，加上 1 个 3 元硬币，得到  $d(3) = 1$ 。

除了第 1 步这个看似基本的公理外，其他往后的结果都是建立在它之前得到的某一步的最优解上，加上 1 个硬币得到。得出：

$$d(i) = d(j) + 1$$

这里  $j < i$ 。通俗地讲，我们需要凑出  $i$  元，就在凑出  $j$  的结果上再加上某一个硬币就行了。

那这里我们加上的是哪个硬币呢。嗯，其实很简单，把每个硬币试一下就行了：

- 假设最后加上的是 1 元硬币，那  $d(i) = d(j) + 1 = d(i - 1) + 1$ 。
- 假设最后加上的是 3 元硬币，那  $d(i) = d(j) + 1 = d(i - 3) + 1$ 。
- 假设最后加上的是 5 元硬币，那  $d(i) = d(j) + 1 = d(i - 5) + 1$ 。

我们分别计算出  $d(i - 1) + 1$ ， $d(i - 3) + 1$ ， $d(i - 5) + 1$  的值，取其中的最小值，即为最优解，也就是  $d(i)$ 。



最后公式：

$$d(i) = \min(d(i - v_x) + 1) \quad (v_x \text{ 表示硬币的币值, } i - v_x \geq 0)。$$

代码示例：

```
public class CoinProblemBasicTest {

    private int[] d; // 储存结果

    private int[] coins = {1, 3, 5}; // 硬币种类

    private void d_func(int i, int num) {

        if (i == 0) {

            d[i] = 0;

            d_func(i + 1, num);

        }

        else {

            int min = 9999999;

            for (int coin : coins) {

                if (i >= coin && d[i - coin] + 1 < min) {

                    min = d[i - coin] + 1;

                }

            }

            d[i] = min;

            if (i < num) {

                d_func(i + 1, num);

            }

        }

    }

    public void test() throws Exception {
```



```
int sum = 11; // 需要凑 11 元

d = new int[sum + 1]; // 初始化数组

d_func(0, sum); // 计算需要凑出 0 ~ sum 元需要的硬币数量

for (int i = 0; i <= sum; i++) {

    System.out.println("凑齐 " + i + " 元需要 " + d[i] + " 个硬币");

}

}

}
```

## 8、手写代码：合并有序数组

考点：算法

参考回答：

解法一：从结尾开始归并，不会覆盖元素。从  $A[n+m-1]$  处开始往前一个元素一个元素的求，每次都要比较  $A[i]$  和  $B[j]$  的大小。需要注意的是，要考虑到：A 和 B 有一个为空时的情况

```
class Solution {

public:

    void merge(int A[], int m, int B[], int n) {

        int i , j , k ;

        for( i = m - 1, j = n - 1, k = n + m - 1; k >= 0; --k)

        {

            if( i >= 0 &&(j < 0 || A[i] >= B[j]) )

                A[k] = A[i--];

            else

                A[k] = B[j--];

        }

    }

};
```



解法二：由于合并后 A 数组的大小必定是  $m+n$ ，所以从最后面开始往前赋值，先比较 A 和 B 中最后一个元素的大小，把较大的那个插入到  $m+n-1$  的位置上，再依次向前推。如果 A 中所有的元素都比 B 小，那么前  $m$  个还是 A 原来的内容，没有改变。如果 A 中的数组比 B 大的，当 A 循环完了，B 中还有元素没加入 A，直接用个循环把 B 中所有的元素覆盖到 A 剩下的位置。

```
class Solution {  
  
    public:  
  
    void merge(int A[], int m, int B[], int n) {  
  
        int count = m + n - 1;  
  
        --m; --n;  
  
        while (m >= 0 && n >= 0) A[count--] = A[m] > B[n] ? A[m--] : B[n--];  
  
        while (n >= 0) A[count--] = B[n--];  
  
    }  
  
};
```

## 9、手写代码：一个数组找出重复的元素

考点：编程

参考回答：

```
function duplicates(arr) {  
  
    //声明两个数组，a 数组用来存放结果，b 数组用来存放 arr 中每个元素的个数  
  
    var a = [], b = [];  
  
    //遍历 arr，如果以 arr 中元素为下标的 b 元素已存在，则该 b 元素加 1，否则设置为 1  
  
    for(var i = 0; i < arr.length; i++){  
  
        if(!b[arr[i]]){  
  
            b[arr[i]] = 1;  
  
            continue;  
  
        }  
  
        b[arr[i]]++;  
  
    }  
  
}
```



```
}

//遍历 b 数组，将其中元素值大于 1 的元素下标存入 a 数组中

for(var i = 0; i < b.length; i++){

    if(b[i] > 1){

        a.push(i);

    }

}

return a;

}
```

时间复杂度为  $O(n)$

#### 10、请问如何防止数组越界

考点：编程语言

参考回答：

由于数组的元素个数默认情况下是不作为实参内容传入调用函数的，因此会带来数组访问越界的相关问题

防止数组越界：

1) 检查传入参数的合法性。

2) 可以用传递数组元素个数的方法，即：用两个实参，一个是数组名，一个是数组的长度。在处理的时候，可以判断数组的大小，保证自己不要访问超过数组大小的元素。

3) 当处理数组越界时，打印出遍历数组的索引十分有帮助，这样我们就能够跟踪代码找到为什么索引达到了一个非法的值

4) Java 中可以加入 `try{ } catch(){ }`

#### 11、请回答数组和链表的区别，以及优缺点，另外有没有什么办法能够结合两者的优点

考察点：数据结构

参考回答：

1. 数组：



数组是将元素在内存中连续存放，由于每个元素占用内存相同，可以通过下标迅速访问数组中任何元素。但是如果要在数组中增加一个元素，需要移动大量元素，在内存中空出一个元素的空间，然后将要增加的元素放在其中。同样的道理，如果想删除一个元素，同样需要移动大量元素去填掉被移动的元素。如果应用需要快速访问数据，很少插入和删除元素，就应该用数组。

## 2. 链表：

链表中的元素在内存中不是顺序存储的，而是通过存在元素中的指针联系到一起，每个结点包括两个部分：一个是存储数据元素的数据域，另一个是存储下一个结点地址的指针。如果要访问链表中一个元素，需要从第一个元素开始，一直找到需要的元素位置。但是增加和删除一个元素对于链表数据结构就非常简单了，只要修改元素中的指针就可以了。如果应用需要经常插入和删除元素你就需要用链表。

## 3. 区别：

### （1）存储位置上：

数组逻辑上相邻的元素在物理存储位置上也相邻，而链表不一定；

### （2）存储空间上：

链表存放的内存空间可以是连续的，也可以是不连续的，数组则是连续的一段内存空间。一般情况下存放相同多的数据数组占用较小的内存，而链表还需要存放其前驱和后继的空间。

### （3）长度的可变性：

链表的长度是按实际需要可以伸缩的，而数组的长度是在定义时要给定的，如果存放的数据个数超过了数组的初始大小，则会出现溢出现象。

（4）按序号查找时，数组可以随机访问，时间复杂度为  $O(1)$ ，而链表不支持随机访问，平均需要  $O(n)$ ；

（5）按值查找时，若数组无序，数组和链表时间复杂度均为  $O(1)$ ，但是当数组有序时，可以采用折半查找将时间复杂度降为  $O(\log n)$ ；

（6）插入和删除时，数组平均需要移动  $n/2$  个元素，而链表只需修改指针即可；

### （7）空间分配方面：

数组在静态存储分配情形下，存储元素数量受限制，动态存储分配情形下，虽然存储空间可以扩充，但需要移动大量元素，导致操作效率降低，而且如果内存中没有更大块连续存储空间将导致分配失败；即数组从栈中分配空间，对于程序员方便快捷，但自由度小。

链表存储的节点空间只在需要的时候申请分配，只要内存中有空间就可以分配，操作比较灵活高效；即链表从堆中分配空间，自由度大但申请管理比较麻烦。

哈希表可以结合数组和链表的优点





### 3、复杂度

1、一行里有很多 IP 地址，互相之间用 || 隔开，共有一万行。要求：取双竖线分隔的倒数第二列所有的 IP 地址并且去重并输出每个 IP 地址的出现次数。

考点：算法

参考回答：

考虑时间效率，用 trie 树统计每个 IP 出现的次数，时间复杂度是  $O(n \cdot l_e)$  ( $l_e$  表示 IP 的平均长度)。

### 2、手写代码：硬币找零问题（要求时间复杂度最佳）

考点：算法

参考回答：

给定一组硬币数，找出一组最少的硬币数，来找换零钱 N。

如何减小时间复杂度：不用全局变量来保存计算过的值，也不用递归的方法来实现，用一个一维数组，再用循环来实现。

```
public int coinChange(int[] coins, int amount) {  
  
    if (coins == null || coins.length == 0 || amount <= 0)  
  
        return 0;  
  
    int[] minNumber = new int[amount + 1];  
  
    for (int i = 1; i <= amount; i++) {  
  
        minNumber[i] = amount + 1;  
  
        for (int j = 0; j < coins.length; j++) {  
  
            if (coins[j] <= i && minNumber[i - coins[j]] != amount + 1)  
  
                minNumber[i] = Integer.min(minNumber[i], 1 + minNumber[i - coins[j]]);  
  
        }  
  
    }  
  
    if (minNumber[amount] == amount + 1)  
  
        return -1;  
}
```



```
else

return minNumber[amount];

}
```

时间复杂度为  $O(c*n)$ ,  $c$  是 coin 的数量,  $n$  是 amount 的值。

## 4、排序

### 1、请问 Java 中 collection 的 sort 方法，默认的排序方法是什么

考察点：算法

参考回答：

排序方法是归并排序

### 2、手写代码：合并两个排序数组

考察点：编程

参考回答：

```
public static int[] MergeList(int a[],int b[])

{

    int result[];

    //            定义一个新数组，长度为两个数组长度之和

    result = new int[a.length+b.length];

    //i:a 数组下标    j: b 数组下标    k: 新数组下标

    int i=0, j=0, k=0;

    //            按位循环比较两个数组，较小元素的放入新数组，下标加一（注意，较大元素对应的下标不加一），直到某一个下标等于数组长度时退出循环

    while(i<a.length && j<b.length)

        if(a[i] <= b[j]) {

            result[k++] = a[i++];

        }
```



```
        print(result);

        System.out.println();

    }else{

        result[k++] = b[j++];

    }

    /* 后面连个 while 循环是用来保证两个数组比较完之后剩下的一个数组
    里的元素能顺利传入 */

    * 此时较短数组已经全部放入新数组，较长数组还有部分剩余，最后将
    剩下的部分元素放入新数组，大功告成*/

    while(i < a.length)

        result[k++] = a[i++];

    while(j < b.length)

        result[k++] = b[j++];

    return result;

}
```

### 3、请问有哪些排序算法

考点：算法

参考回答：

冒泡排序

是最简单的排序之一了，其大体思想就是通过与相邻元素的比较和交换来把小的数交换到最前面。这个过程类似于水泡向上升一样，因此而得名。举个栗子，对 5, 3, 8, 6, 4 这个无序序列进行冒泡排序。首先从后向前冒泡，4 和 6 比较，把 4 交换到前面，序列变成 5, 3, 8, 4, 6。同理 4 和 8 交换，变成 5, 3, 4, 8, 6, 3 和 4 无需交换。5 和 3 交换，变成 3, 5, 4, 8, 6。这样一次冒泡就完了，把最小的数 3 排到最前面了。对剩下的序列依次冒泡就会得到一个有序序列。冒泡排序的时间复杂度为  $O(n^2)$ 。

选择排序

思想其实和冒泡排序有点类似，都是在一次排序后把最小的元素放到最前面。但是过程不同，冒泡排序是通过相邻的比较和交换。而选择排序是通过对整个体的选择。举个栗子，对 5, 3, 8, 6, 4 这个无序序列进行简单选择排序，首先要选择 5 以外的最小数来和 5 交换，也就是选择 3 和 5



交换，一次排序后就变成了 3, 5, 8, 6, 4. 对剩下的序列一次进行选择 and 交换，最终就会得到一个有序序列。其实选择排序可以看成冒泡排序的优化，因为其目的相同，只是选择排序只有在确定了最小数的前提下才进行交换，大大减少了交换的次数。选择排序的时间复杂度为  $O(n^2)$

### 插入排序

不是通过交换位置而是通过比较找到合适的位置插入元素来达到排序的目的。相信大家都有过打扑克牌的经历，特别是牌数较大的。在分牌时可能要整理自己的牌，牌多的时候怎么整理呢？就是拿到一张牌，找到一个合适的位置插入。这个原理其实和插入排序是一样的。举个栗子，对 5, 3, 8, 6, 4 这个无序序列进行简单插入排序，首先假设第一个数的位置时正确的，想一下在拿到第一张牌的时候，没必要整理。然后 3 要插到 5 前面，把 5 后移一位，变成 3, 5, 8, 6, 4. 想一下整理牌的时候应该也是这样吧。然后 8 不用动，6 插在 8 前面，8 后移一位，4 插在 5 前面，从 5 开始都向后移一位。注意在插入一个数的时候要保证这个数前面的数已经有序。简单插入排序的时间复杂度也是  $O(n^2)$ 。

（我用了链表，别人用数组后移更好）

### 快速排序

在实际应用当中快速排序确实也是表现最好的排序算法。快速排序虽然高端，但其实其思想是来自冒泡排序，冒泡排序是通过相邻元素的比较和交换把最小的冒泡到最顶端，而快速排序是比较和交换小数和大数，这样一来不仅把小数冒泡到上面同时也把大数沉到下面。

对 5, 3, 8, 6, 4 这个无序序列进行快速排序，思路是右指针找比基准数小的，左指针找比基准数大的，交换之。

5, 3, 8, 6, 4 用 5 作为比较的基准，最终会把 5 小的移动到 5 的左边，比 5 大的移动到 5 的右边。

5, 3, 8, 6, 4 首先设置 i, j 两个指针分别指向两端，j 指针先扫描（思考一下为什么？）4 比 5 小停止。然后 i 扫描，8 比 5 大停止。交换 i, j 位置。

5, 3, 4, 6, 8 然后 j 指针再扫描，这时 j 扫描 4 时两指针相遇。停止。然后交换 4 和基准数。

4, 3, 5, 6, 8 一次划分后达到了左边比 5 小，右边比 5 大的目的。之后对左右子序列递归排序，最终得到有序序列。

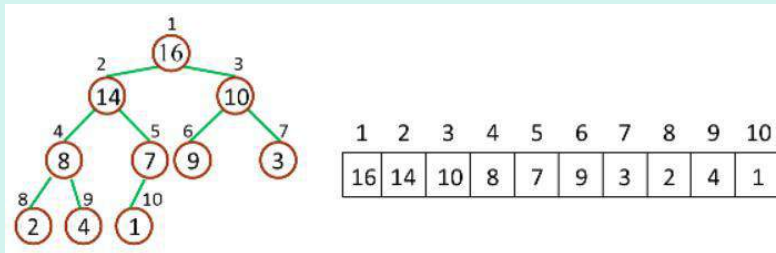
上面留下来了个问题为什么一定要 j 指针先动呢？首先这也不是绝对的，这取决于基准数的位置，因为在最后两个指针相遇的时候，要交换基准数到相遇的位置。一般选取第一个数作为基准数，那么就是在左边，所以最后相遇的数要和基准数交换，那么相遇的数一定要比基准数小。所以 j 指针先移动才能先找到比基准数小的数。快速排序是不稳定的，其时间平均时间复杂度是  $O(n \lg n)$ 。

### 堆排序

借助堆来实现的选择排序，思想同简单的选择排序，以下以大顶堆为例。注意：如果想升序排序就使用大顶堆，反之使用小顶堆。原因是堆顶元素需要交换到序列尾部。实现堆排序需要解决两个问题：

1. 如何由一个无序序列建成一个堆？
2. 如何在输出堆顶元素之后，调整剩余元素成为一个新的堆？

堆（二叉堆）可以视为一棵完全的二叉树，完全二叉树的一个“优秀”的性质是，除了最底层之外，每一层都是满的，这使得堆可以利用数组来表示（普通的一般的二叉树通常用链表作为基本容器表示），每一个结点对应数组中的一个元素。

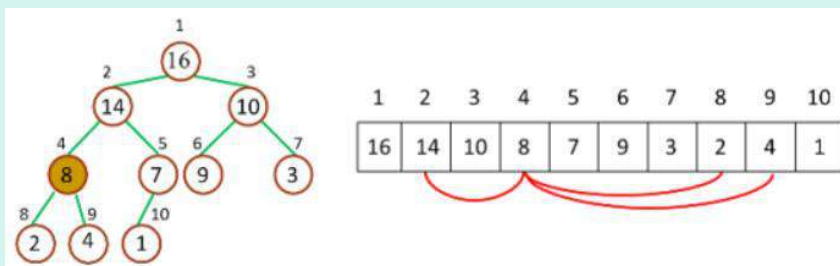


对于给定的某个结点的下标  $i$ ，可以很容易的计算出这个结点的父结点、孩子结点的下标：

$\text{Parent}(i) = \text{floor}(i/2)$ ,  $i$  的父节点下标

$\text{Left}(i) = 2i$ ,  $i$  的左子节点下标

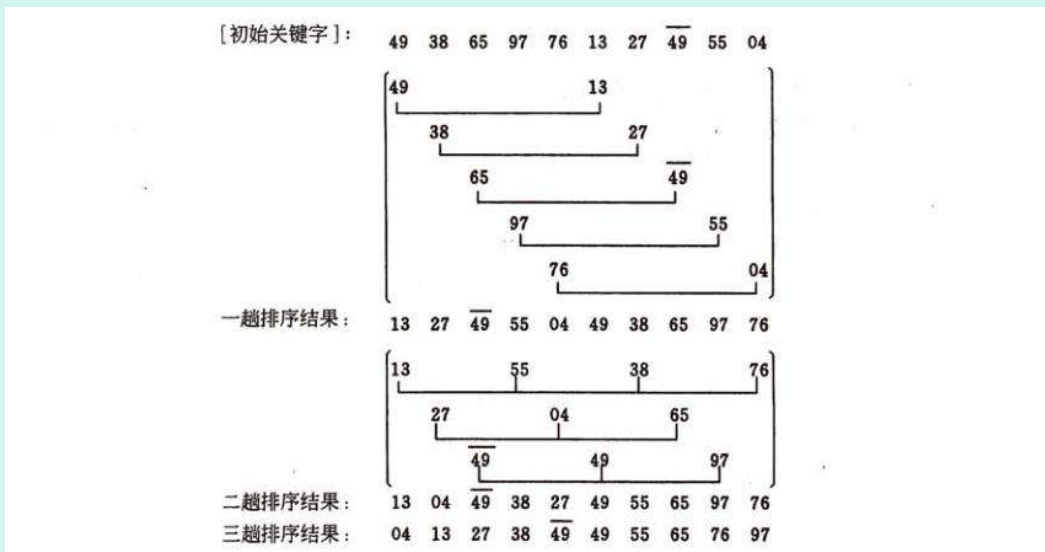
$\text{Right}(i) = 2i + 1$ ,  $i$  的右子节点下标



堆排序（Heap-Sort）是堆排序的接口算法，Heap-Sort 先调用 Build-Max-Heap 将数组改造为最大堆，然后将堆顶和堆底元素交换，之后将底部上升，最后重新调用 Max-Heapify 保持最大堆性质。

希尔排序

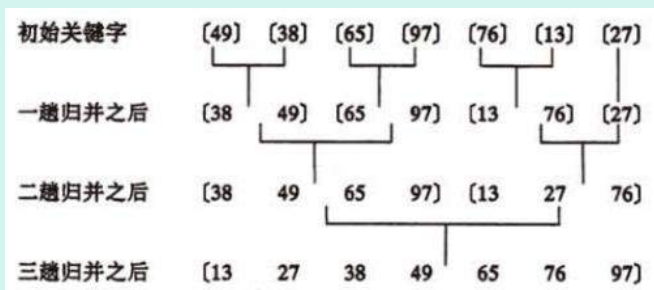
插入排序的一种高效率的实现，也叫缩小增量排序。简单的插入排序中，如果待排序列是正序时，时间复杂度是  $O(n)$ ，如果序列是基本有序的，使用直接插入排序效率就非常高。基本思想是：先将整个待排记录序列分割成为若干子序列分别进行直接插入排序，待整个序列中的记录基本有序时再对全体记录进行一次直接插入排序。



希尔排序的特点是，子序列的构成不是简单的逐段分割，而是将某个相隔某个增量的记录组成一个子序列。如上面的例子，第一趟排序时的增量为5，第二趟排序的增量为3。

### 归并排序

使用了递归分治的思想，先递归划分子问题，然后合并结果。把待排序列看成由两个有序的子序列，然后合并两个子序列，然后把子序列看成由两个有序序列。。。倒着来看，其实就是先两两合并，然后四四合并。。最终形成有序序列。空间复杂度为  $O(n)$ ，时间复杂度为  $O(n \log n)$ 。



### 计数排序

如果在面试中有面试官要求你写一个  $O(n)$  时间复杂度的排序算法，你千万不要立刻说：这不可能！虽然前面基于比较的排序的下限是  $O(n \log n)$ 。但是确实也有线性时间复杂度的排序，只不过有前提条件，就是待排序的数要满足一定的范围的整数，而且计数排序需要比较多的辅助空间。其基本思想是，用待排序的数作为计数数组的下标，统计每个数字的个数。然后依次输出即可得到有序序列。

(java 中有整型的最大最小值可以直接用)

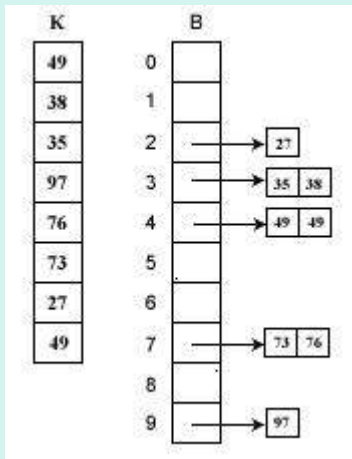
```
int MAX = Integer.MAX_VALUE;
```

```
int MIN = Integer.MIN_VALUE;
```

### 桶排序

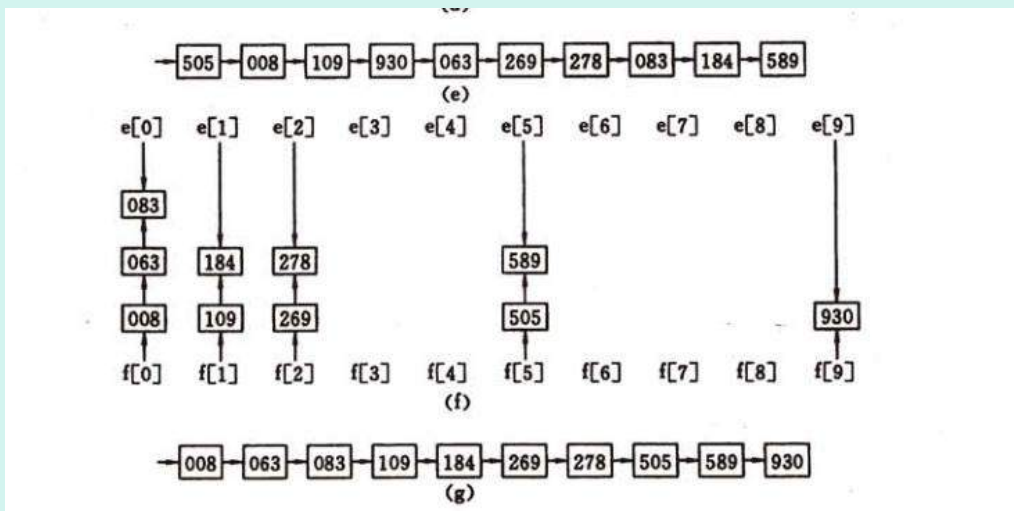


假设有一组长度为  $N$  的待排关键字序列  $K[1 \dots n]$ 。首先将这个序列划分成  $M$  个的子区间（桶）。然后基于某种映射函数，将待排序列的关键字  $k$  映射到第  $i$  个桶中（即桶数组  $B$  的下标  $i$ ），那么该关键字  $k$  就作为  $B[i]$  中的元素（每个桶  $B[i]$  都是一组大小为  $N/M$  的序列）。接着对每个桶  $B[i]$  中的所有元素进行比较排序（可以使用快排）。然后依次枚举输出  $B[0] \dots B[M]$  中的全部内容即是一个有序序列。



### 基数排序

一种借助多关键字排序思想对单逻辑关键字进行排序的方法。所谓的多关键字排序就是有多组优先级不同的关键字。比如说成绩的排序，如果两个人总分相同，则语文高的排在前面，语文成绩也相同则数学高的排在前面。。。如果对数字进行排序，那么个位、十位、百位就是不同优先级的关键字，如果要进行升序排序，那么个位、十位、百位优先级一次增加。基数排序是通过多次的收分配和收集来实现的，关键字优先级低的先进行分配和收集。



### 二分法插入排序

二分法插入排序是在插入第  $i$  个元素时，对前面的  $0 \sim i-1$  元素进行折半，先跟他们中间的那个元素比，如果小，则对前半再进行折半，否则对后半进行折半，直到  $left < right$ ，然后再把第  $i$  个元素前 1 位与目标位置之间的所有元素后移，再把第  $i$  个元素放在目标位置上。





二分插入排序是稳定的与二分查找的复杂度相同；

最好的情况是当插入的位置刚好是二分位置 所用时间为  $O(\log_2 n)$ ；

最坏的情况是当插入的位置不在二分位置 所需比较次数为  $O(n)$ ，无限逼近线性查找的复杂度。

#### 4、手写代码：冒泡排序

考点：算法

参考回答：

```
void bubblesort(int[] num) {  
  
    for(int i=0;i<num.length-1;i++) {  
  
        for(int j=num.length-1;j>i;j--) {  
  
            if(num[j]<num[j-1]) swap(num, j, j-1);  
  
        }  
  
    }  
  
    for(int i=0;i<num.length;i++) {  
  
        System.out.print(num[i]);  
  
    }  
  
}
```

#### 5、手写代码：统计排序数组中出现次数最多的元素出现的次数？

参考回答：

```
public class Main {  
  
    static void findmost(int[] array){  
  
        int lastEle=array[0];  
  
        int maxTime=0;  
  
        int presentTime=1;
```





```
int maxEle=array[0];

for(int i=1;i<array.length;i++)

{

if(array[i]==lastEle)

presentTime++;

else

{

if(presentTime>maxTime)

{

maxTime=presentTime;

maxEle=lastEle;

}

lastEle=array[i];

presentTime=1;

}

if(i==array.length-1 && presentTime>maxTime)// 考虑到比较到最大的元素（排在最后的元素），需要在循环推出前比较一次

{

maxTime=presentTime;

maxEle=lastEle;

}

}

System.out.println("出现次数最多的元素"+maxEle+" "+"出现的次数"+maxTime);

}

public static void main(String args[]) {

int[] array= {1, 1, 2, 2, 2, 3, 4, 5, 6, 6, 6, 6, 7, 8, 8, 9};
```



```
Main.findmost(array);  
  
}  
  
}
```

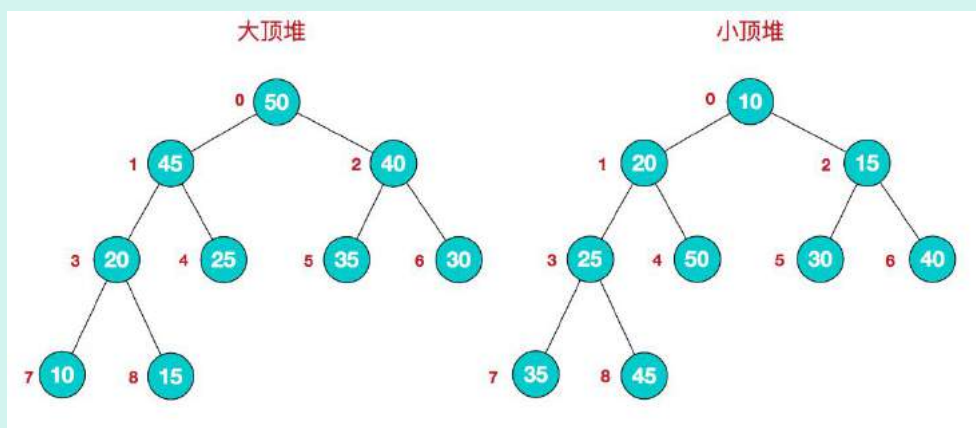
## 6、请你说一下堆排序的思想？以及怎么初始建堆？是否稳定？

考点：算法

参考回答：

堆排序是利用堆这种数据结构而设计的一种排序算法，堆排序是一种选择排序，它的最坏，最好，平均时间复杂度均为  $O(n\log n)$ ，它是不稳定的排序。

堆是具有以下性质的完全二叉树：每个结点的值都大于或等于其左右孩子结点的值，称为大顶堆；或者每个结点的值都小于或等于其左右孩子结点的值，称为小顶堆。如下图：



对堆中的结点按层进行编号，将这种逻辑结构映射到数组中就是下面这个样子

	0	1	2	3	4	5	6	7	8
arr	50	45	40	20	25	35	30	10	15

该数组从逻辑上讲就是一个堆结构，用简单的公式来描述一下堆的定义就是：

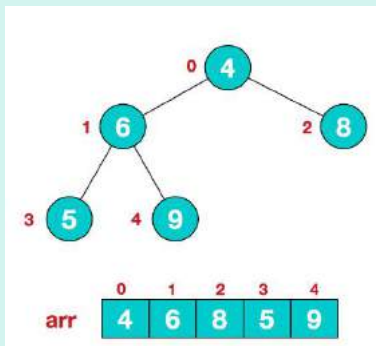
大顶堆： $arr[i] \geq arr[2i+1] \ \&\& \ arr[i] \geq arr[2i+2]$

小顶堆： $arr[i] \leq arr[2i+1] \ \&\& \ arr[i] \leq arr[2i+2]$

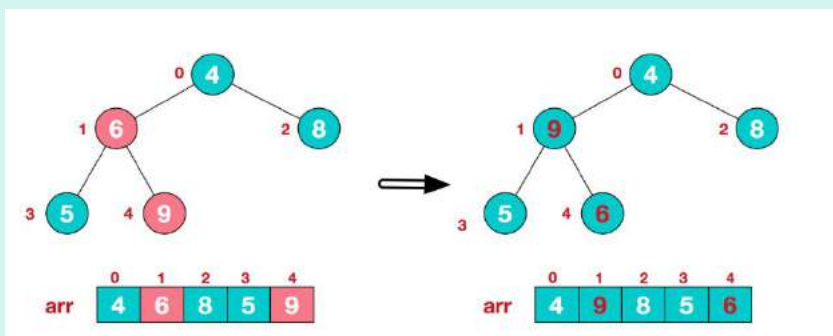
堆排序的基本思想是：将待排序序列构造成一个大顶堆，此时，整个序列的最大值就是堆顶的根节点。将其与末尾元素进行交换，此时末尾就为最大值。然后将剩余  $n-1$  个元素重新构造成一个堆，这样会得到  $n$  个元素的次小值。如此反复执行，便能得到一个有序序列了

步骤一 构造初始堆。将给定无序序列构造一个大顶堆（一般升序采用大顶堆，降序采用小顶堆）。

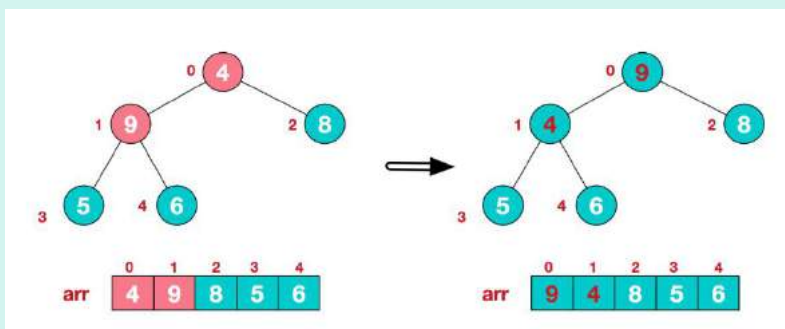
假设给定无序序列结构如下



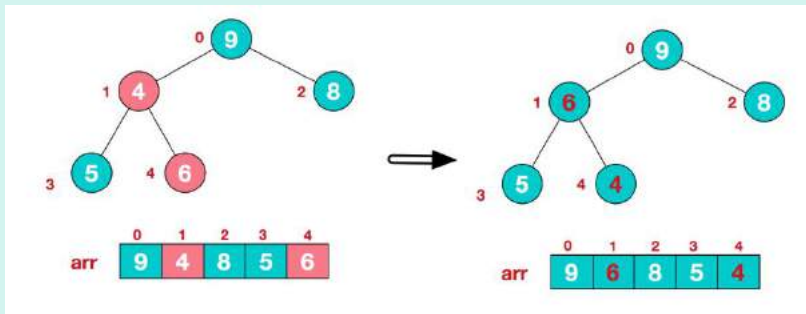
从最后一个非叶子结点开始（叶结点自然不用调整，第一个非叶子结点  $\text{arr.length}/2-1=5/2-1=1$ ，也就是下面的 6 结点），从左至右，从下至上进行调整。



找到第二个非叶节点 4，由于 [4, 9, 8] 中 9 元素最大，4 和 9 交换。



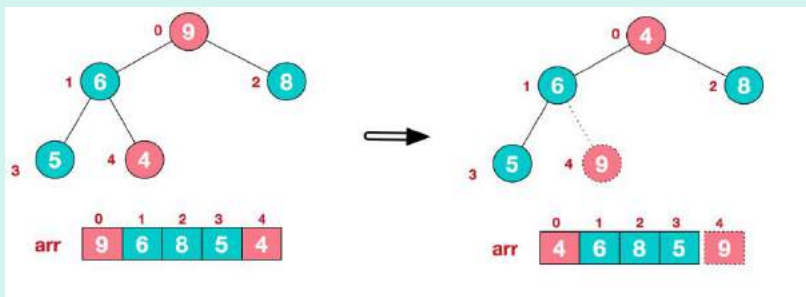
这时，交换导致了子根 [4, 5, 6] 结构混乱，继续调整，[4, 5, 6] 中 6 最大，交换 4 和 6。



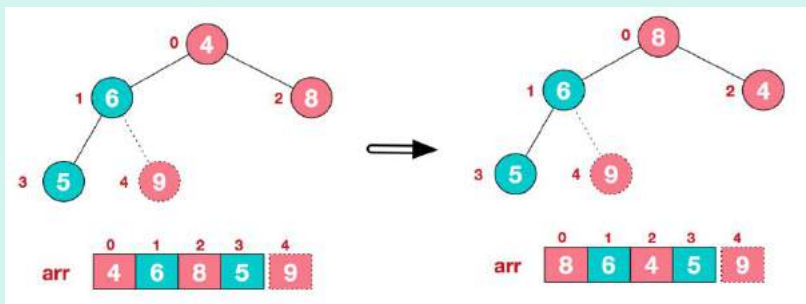
此时，我们就将一个无序序列构造成了一个最大堆。

步骤二 将堆顶元素与末尾元素进行交换，使末尾元素最大。然后继续调整堆，再将堆顶元素与末尾元素交换，得到第二大元素。如此反复进行交换、重建、交换。

a. 将堆顶元素 9 和末尾元素 4 进行交换

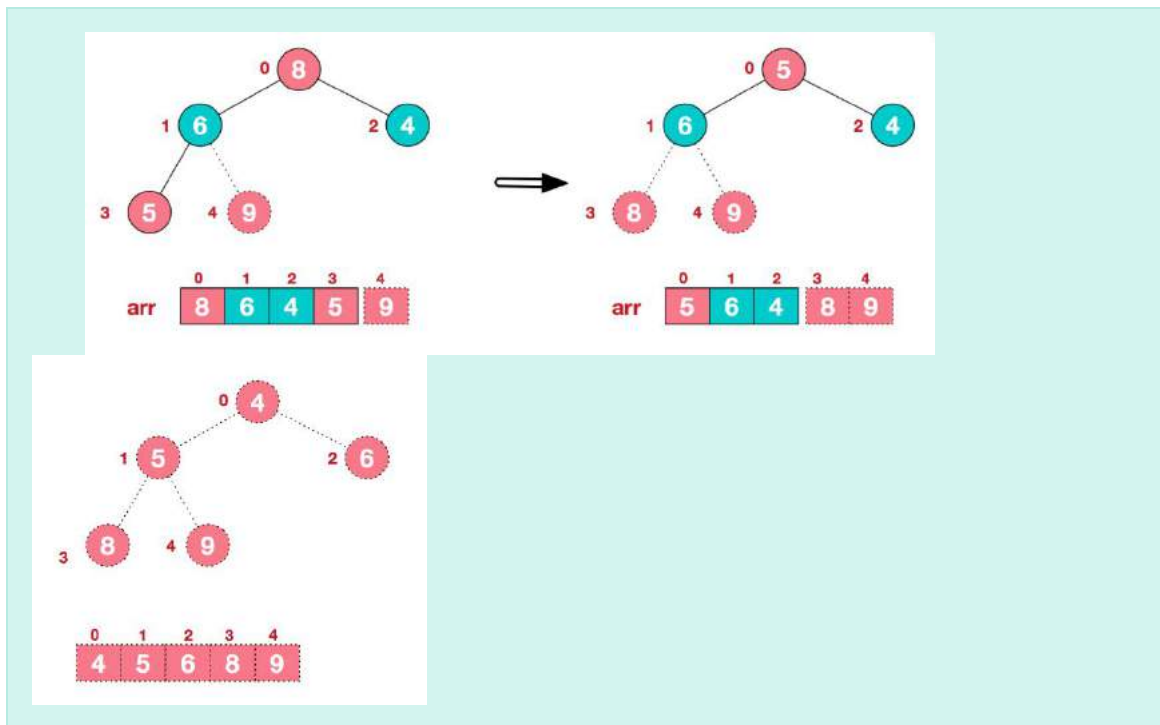


b. 重新调整结构，使其继续满足堆定义



c. 再将堆顶元素 8 与末尾元素 5 进行交换，得到第二大元素 8.

后续过程，继续进行调整，交换，如此反复进行，最终使得整个序列有序



## 7、手写代码：数组的 2-sum,3-sum,问题（leetcode 原题）

考察点：编程

参考回答：

2SUM 问题

最常见的是 2SUM 问题（1. 2SUM），就是数组中找出两个数相加和为给定的数。  
这道题有两种思路：

1. 一种思路从首尾搜索两个数的和，并逐步逼近。
2. 另外一种思路是固定一个数 A，看  $SUM-A$  是否在这个数组之中。

对于第一种思路如下：

此方法是先将数组从小到大排序

设置两个指针，一个指向数组开头，一个指向数组结尾，从两边往中间走。直到扫到满足题意的为止或者两个指针相遇为止。

此时这种搜索方法就是类似于杨氏矩阵的搜索方法，就是从 杨氏矩阵的左下角开始搜索，直到找到为止。

如果此时题目条件变为如果没有找出最接近的 2SUM 和问题，解决方法跟上面是一样的  
用这种方法 2SUM 问题的时间复杂度是  $O(n \log n)O(n \log \frac{1}{\epsilon}n)$  的，主要在于排序的时间。

第二种思路方法如下：

对数组中的每个数建立一个 map/hash\_map 然后再扫一遍这个数组，判断  $target-nums[i]$  是否存在，如果存在则说明有，不存在继续找。当然这样做的话，需要处理一个细节：判重的问题。



代码如下【注意因为题目中说一定有解所以才下面这样写，如果不一定有解，则还要再加一些判断】

```
vector<int> twoSum(vector<int>& nums, int target) {  
  
    unordered_map<int, vector<int>> mark;  
  
    for(int i=0;i<nums.size();i++)  
  
        mark[nums[i]].push_back(i);  
  
    for(int i = 0;i<nums.size();i++){  
  
        if(target-nums[i] == nums[i]){  
  
            if(mark[nums[i]].size() > 1){  
  
                vector<int> tmp{i,mark[nums[i]][1]};  
  
                return tmp;  
  
            }  
  
        }else{  
  
            if(mark.find(target-nums[i]) != mark.end()){  
  
                vector<int> tmp{i,mark[target-nums[i]][0]};  
  
                return tmp;  
  
            }  
  
        }  
  
    }  
  
}
```

### 3-SUM 问题

#### Question

Given an array nums of  $n$  integers, are there elements  $a$ ,  $b$ ,  $c$  in nums such that  $a + b + c = 0$ ? Find all unique triplets in the array which gives the sum of zero. The solution set must not contain duplicate triplets.

根据给定的容量为  $n$  的整数数组，找到所有满足  $a + b + c = 0$  的三个元素  $a$ 、 $b$ 、 $c$  组合，需去重；



### 解法

修改问题为：满足  $a + b + c = \text{target}$ ， $\text{target}$  是给定数，原题即  $\text{target} = 0$ ；

根据题目可知，与 2-SUM 问题类似，在整数数组中不放回的取出三个元素，其和等于给定数（0），不同的是，满足条件的解有多个而且需要去重；

首先想到的解法是，遍历数组，然后调用 2-SUM 问题中的方法寻找两个元素相加等于当前元素的补足，最后执行去重操作；这样的话，查询的时间复杂度是  $O(n^2)$ ，空间复杂度是  $O(n^2)$ ，去重的时间复杂度是  $O(n^2)$ ，空间复杂度是  $O(1)$ ，这绝对不能算一个好方案；

思考其他思路，既然要去重，可以先对数组执行一次排序，这样的话在遍历的时候可以跳过相同元素；在确定一个当前元素后，找另外两个元素相加作为当前元素的补足，此时的解可能是多个的，采用首尾标记的方式可以一次遍历完成查找；

```
public List<List<Integer>> threeSum(int[] nums, int target){

    int length = nums.length;

    List<List<Integer>> result = new ArrayList<>();

    Arrays.sort(nums);

    for(int i = 0; i < length - 2; i++) {

        if(nums[i] + nums[i+1] + nums[i+2] > target)break; // too large

        if(nums[i] + nums[length-1] + nums[length-2] < target)continue; // too small

        if(i > 0 && nums[i] == nums[i - 1]) continue;

        int left = i + 1;

        int right = length - 1;

        while(left < right){

            int diff = target - nums[i] - nums[left] - nums[right];

            if (diff == 0){

                result.add(new ArrayList<Integer>(Arrays.asList(nums[i],
nums[left], nums[right])));

                while(left < right && nums[left] == nums[left + 1]) left++;

                while(left < right && nums[right] == nums[right - 1]) right--;

                left++;
            }
        }
    }
}
```



```
        right--;\n\n        }else if (diff > 0){\n\n            left++;\n\n        }else {\n\n            right--;\n\n        }\n\n    }\n\n}\n\nreturn result;\n\n}
```

#### 8、手写代码：5 个扑克牌是否是顺子，大小王当成任意的

参考回答:这是剑指 offer 原题

从扑克牌中随机抽出 5 张牌，判断是不是一个顺子，即这五张牌是不是连续的。2——10 为数字本身，A 为 1，J 为 11，Q 为 12，K 为 13，而大小王为任意数字。

```
package org.marsguo.offerproject44;\n\nimport java.util.Scanner;\n\nclass SolutionMethod1{\n\n    public void sortfun(int[] arrays,int start, int end){\n\n        int numOfZero=0;\n\n        if(start>=end){\n\n            //判断数组的起始和终止是否相同，相同表示已经都全部排完，返回\n\n            return;\n\n        }\n\n        int i = start;\n\n        //i 指向数组的起始位\n\n        int j = end;\n\n        //j 指向数组的末位
```





```
int key = arrays[i];           //选取数组的第一位为关键字 key, 基
准元素

boolean flag = true;           //设置标志位, 用于判断是 i++还是
j--;这个很重要

while(i != j) {                //如果 i != j, 表示还没有比较完,
    即即关键字左右两侧还不是最小与最大

    if(flag) {

        if(key > arrays[j]) {    //从后向前遍历, 找到小于 key
            的值,

            swap(arrays, i, j);  //找到小于 key 的值后将
            arrays[i]与此值交换

            flag = false;

        }else{                  //如果没有找到的话 j--, 向前
            遍历

            j--;

        }

    }else{

        if(key < arrays[i]) {    //从前向后遍历, 找到大于 key
            的值

            swap(arrays, i, j);  //将此值与 arrays[j]进行交换

            flag = true;

        }else{                  //如果没有找到话就将 i++, 向后
            遍历

            i++;

        }

    }

}

//sprint(arrays);              //打印每次排序后的数组
```



```
sortfun(arrays, start, j-1); //递归调用，将基准元素的前半  
段数组再用此方法进行排序，直到所有都排完为止。
```

```
sortfun(arrays, i+1, end); //递归调用，将基准元素的后半段数  
组再用此方法进行排序，直到所有都排完为止。
```

```
//System.out.println("排序后的数组是：");
```

```
for(int k = 0; k < arrays.length; k++){
```

```
    if(arrays[k] == 0){
```

```
        numOfZero++;
```

```
    }
```

```
//System.out.print("numOfZero= " + numOfZero + ";"+arrays[k] + ", ");
```

```
//System.out.print(arrays[k] + ", ");
```

```
}
```

```
IsContinuousFun(arrays, numOfZero);
```

```
}
```

```
public void swap(int[] array, int i, int j){
```

```
    int temp;
```

```
    temp = array[i];
```

```
    array[i] = array[j];
```

```
    array[j] = temp;
```

```
}
```

```
public void IsContinuousFun(int[] array, int numOfZero){
```

```
    int numOfGap = 0; //判断数字中间空了多少个 0
```

```
//System.out.println("numberOfZero = " + numOfZero);
```



```
        for(int j = 1; j < array.length; j++){

            int flag = array[j] - array[j-1];    //用排序数组中后一个数字-前一个
数字

            if(array[j] == array[j-1] && array[j] != 0){    //如果数组中有重复的
非 0 数字，则不是顺子，退出

                System.out.println("有重复数字，不是顺子牌");

                return;

            }

            else if(flag != 1 && flag != 0 && array[j-1] != 0){    //判断
不是连续的数字，也不是 0，

                numOfGap += flag-1;    //非 0 数字间缺少
的数字的个数

            }

        }

        if(numOfZero != numOfGap){

            System.out.println("这不是一个顺子扑克牌");

        }

        else{

            System.out.println("这是一张顺子扑克牌");

        }

    }

}

public class IsContinuous {

    public static void main(String[] args){

        Scanner scanner = new Scanner(System.in);    //扫描键盘
输入

        System.out.println("请输入五张牌：");
```



```
String str = scanner.nextLine(); //将键盘输入转化为字符串

String[] temp = str.split(" "); //将字符串用“ ”分开转化为字符串数组

scanner.close();

int[] array = new int[temp.length]; //定义一个整型数组 array

for(int i = 0; i < temp.length; i++){ //将字符串数组强制转化为整型数组

    array[i] = Integer.parseInt(temp[i]); //这种方法非常巧妙

}

SolutionMethod1 solution1 = new SolutionMethod1();

//solution1.quickSort(array, 0, array.length-1);

solution1.sortFun(array, 0, array.length-1);

}
```

## 9、请你说一说快速排序，并手写代码

考点：算法

参考回答：

1、快速排序的基本思想：

快速排序使用分治的思想，通过一趟排序将待排序列分割成两部分，其中一部分记录的关键字均比另一部分记录的关键字小。之后分别对这两部分记录继续进行排序，以达到整个序列有序的目的。

2、快速排序的三个步骤：

(1)选择基准：在待排序列中，按照某种方式挑出一个元素，作为“基准”（pivot）

(2)分割操作：以该基准在序列中的实际位置，把序列分成两个子序列。此时，在基准左边的元素都比该基准小，在基准右边的元素都比基准大



(3)递归地对两个序列进行快速排序，直到序列为空或者只有一个元素。

### 3、选择基准的方式：

对于分治算法，当每次划分时，算法若都能分成两个等长的子序列时，那么分治算法效率会达到最大。也就是说，基准的选择是很重要的。选择基准的方式决定了两个分割后两个子序列的长度，进而对整个算法的效率产生决定性影响。最理想的方法是，选择的基准恰好能把待排序序列分成两个等长的子序列三种选择基准的方法：

#### 方法(1)：固定位置

思想：取序列的第一个或最后一个元素作为基准

#### 方法(2)：随机选取基准

引入的原因：在待排序列是部分有序时，固定选取枢轴使快排效率底下，要缓解这种情况，就引入了随机选取枢轴

思想：取待排序列中任意一个元素作为基准

#### 方法(3)：三数取中 (median-of-three)

引入的原因：虽然随机选取枢轴时，减少出现不好分割的几率，但是还是最坏情况下还是  $O(n^2)$ ，要缓解这种情况，就引入了三数取中选取枢轴

分析：最佳的划分是将待排序的序列分成等长的子序列，最佳的状态我们可以使用序列的中间的值，也就是第  $N/2$  个数。可是，这很难算出来，并且会明显减慢快速排序的速度。这样的中值的估计可以通过随机选取三个元素并用它们的中值作为枢纽元而得到。事实上，随机性并没有多大的帮助，因此一般的做法是使用左端、右端和中心位置上的三个元素的中值作为枢纽元。显然使用三数中值分割法消除了预排序输入的不好情形，并且减少快排大约 14% 的比较次数。

固定位置示例程序：

```
public class QuickSort {

    public static void main(String[] args) {

        int[] a = {1, 2, 4, 5, 7, 4, 5, 3, 9, 0};

        System.out.println(Arrays.toString(a));

        quickSort(a);

        System.out.println(Arrays.toString(a));

    }

    public static void quickSort(int[] a) {
```



```
if(a.length>0) {

    quickSort(a, 0 , a.length-1);

}

}

private static void quickSort(int[] a, int low, int high) {

    //1,找到递归算法的出口

    if( low > high) {

        return;

    }

    //2, 存

    int i = low;

    int j = high;

    //3,key

    int key = a[ low ];

    //4, 完成一趟排序

    while( i< j) {

        //4.1 , 从右往左找到第一个小于 key 的数

        while(i<j && a[j] > key){

            j--;

        }

        // 4.2 从左往右找到第一个大于 key 的数

        while( i<j && a[i] <= key) {

            i++;

        }

        //4.3 交换
```



```
    if(i<j) {  
  
        int p = a[i];  
  
        a[i] = a[j];  
  
        a[j] = p;  
  
    }  
  
}
```

// 4.4, 调整 key 的位置

```
int p = a[i];  
  
a[i] = a[low];  
  
a[low] = p;  
  
//5, 对 key 左边的数快排  
  
quickSort(a, low, i-1 );  
  
//6, 对 key 右边的数快排  
  
quickSort(a, i+1, high);  
  
}  
  
}
```

四种优化方式：

优化 1：当待排序序列的长度分割到一定大小后，使用插入排序

原因：对于很小和部分有序的数组，快排不如插排好。当待排序序列的长度分割到一定大小后，继续分割的效率比插入排序要差，此时可以使用插排而不是快排

优化 2：在一次分割结束后，可以把与 Key 相等的元素聚在一起，继续下次分割时，不用再对与 key 相等元素分割

优化 3：优化递归操作

快排函数在函数尾部有两次递归操作，我们可以对其使用尾递归优化

优点：如果待排序的序列划分极端不平衡，递归的深度将趋近于  $n$ ，而栈的大小是很有限的，每次递归调用都会耗费一定的栈空间，函数的参数越多，每次递归耗费的空间也越多。优化后，可以缩减堆栈深度，由原来的  $O(n)$  缩减为  $O(\log n)$ ，将会提高性能。



优化 4：使用并行或多线程处理子序列

**10、你最熟悉什么算法？给我说一下原理或者排序过程？它的优缺点是什么？你知道什么排序算法，介绍他们的实现方法，时间复杂度和空间复杂度，是否稳定，快排基准点怎么选择，**

考察点：算法

参考回答：

最熟悉排序算法，常见的排序算法有以下几种

1、插入排序，即逐个处理待排序的记录，每个记录与前面已排序的子序列进行比较，将它插入子序列中正确位置，时间复杂度  $O(n^2)$ ，空间复杂度为  $O(1)$ ，是稳定排序，插入排序不适合对于数据量比较大的排序应用，但是如果量级小余千，那么插入排序还是一个不错的选择，

2、冒泡排序，它重复地走访过要排序的元素，依次比较相邻两个元素，如果他们的顺序错误就把他们调换过来，直到没有元素再需要交换，排序完成。这个算法的名字由来是因为越小(或越大)的元素会经由交换慢慢“浮”到数列的顶端。他的时间复杂度和空间复杂度分别是  $O(n^2)$ ，空间复杂度是  $O(1)$  属于稳定排序，冒泡排序对于少数元素之外的数列排序是很没效率的。

3、选择排序，初始时在序列中找到最值元素，放到序列的其实位置作为已排序序列，再在剩余未排序元素中继续寻找最小(大)元素，放到已排序序列的末尾。以此类推，直到所有元素均排序完毕。他的时间复杂度是  $O(n^2)$ ，空间复杂度是  $O(1)$  属于不稳定排序

4、shell 排序，是插入排序的升级版，属于不稳定排序，希尔排序通过将比较的全部元素分为几个区域来提升插入排序的性能。这样可以让一个元素可以一次性地朝最终位置前进一大步。然后算法再取越来越小的步长进行排序，算法的最后一步就是普通的插入排序，但是到了这步，需排序的数据几乎是已排好的了（此时插入排序较快）。

假设有一个很小的数据在一个已按升序排好序的数组的末端。如果用复杂度为  $O(n^2)$  的排序（冒泡排序或直接插入排序），可能会进行  $n$  次的比较和交换才能将该数据移至正确位置。而希尔排序会用较大的步长移动数据，所以小数据只需进行少数比较和交换即可到正确位置。希尔排序的时间复杂度根据步长序列的不同而不同，空间复杂度  $O(1)$

5、归并排序，归并排序的实现分为递归实现与非递归(迭代)实现。属于稳定排序，递归实现的归并排序是算法设计中分治策略的典型应用，我们将一个大问题分割成小问题分别解决，然后用所有小问题的答案来解决整个大问题。非递归(迭代)实现的归并排序首先进行是两两归并，然后四四归并，然后是八八归并，一直下去直到归并了整个数组。他的时间复杂度是  $O(n \log n)$  空间复杂度是  $O(n)$

6、堆排序，其实现原理为首先将数组构造成一个最大/最小堆，将堆顶元素和堆尾元素互换，调出堆顶元素，重新构造堆，重复步骤直至堆中元素都被调出。堆排序的时间复杂度为  $O(n \log n)$  空间复杂度为  $O(1)$ ，属于不稳定排序。

7、快速排序，快排使用分治策略，首先从序列中挑出一个元素作为基准，然后把比基准小的元素放在一边，把比基准大的元素放在另一边，重复这个步骤，直至子序列的大小是  $O(1)$ 。快排的时间复杂度是  $O(n \log n)$  空间复杂度是  $O(\log n)$  属于不稳定算法，对于快排的基准元素选



取，可以采用三者取中法，三个随机值的中间一个。为了减少随机数生成器产生的延迟，可以选择首中尾三个元素作为随机值

排序算法的适用情况：

当  $n$  比较小且基本有序，可采用直接插入或冒泡，否则采用直接插入

当  $n$  较大，可以采用快排，归并，堆排序，快排是目前基于比较的内部排序中最好的方法，堆排序所需的辅助空间少于快速排序，并且不会出现快速排序可能出现的最坏情况。这两种排序都是不稳定的。

若要求排序稳定，则可选用归并排序。但前面介绍的从单个记录起进行两两归并的排序算法并不值得提倡，通常可以将它和直接插入排序结合在一起使用。先利用直接插入排序求得较长的有序子序列，然后再两两归并之。因为直接插入排序是稳定的，所以改进后的归并排序仍是稳定的。

## 11、请你说一下快排如何实现？

考察点：算法

参考回答：

首先选择一个轴值，小于轴值的元素被放在数组中轴值左侧，大于轴值的元素被放在数组中轴值右侧，这称为数组的一个分割(partition)。快速排序再对轴值左右子数组分别进行类似的操作

选择轴值有多种方法。最简单的方法是使用首或尾元素。但是，如果输入的数组是正序或者逆序时，会将所有元素分到轴值的一边。较好的方法是随机选取轴值

代码：

```
template <class Elem>

int partition(Elem A[],int i,int j)

{

    //这里选择尾元素作为轴值,轴值的选择可以设计为一个函数

    //如果选择的轴值不是尾元素，还需将轴值与尾元素交换

    int pivot = A[j];

    int l = i - 1;

    for(int r = i;r < j;r++)

        if(A[r] <= pivot)

            swap(A, ++l, r);
```



```
swap(A, ++l, j); //将轴值从末尾与++l 位置的元素交换

return l;

}

template <class Elem>

void qsort(Elem A[], int i, int j)

{

    if(j <= i) return;

    int p = partition<Elem>(A, i, j);

    qsort<Elem>(A, i, p - 1);

    qsort<Elem>(A, p + 1, j);

}
```

## 5、树

### 1、请你说一下 BST 的特点，并手写代码判断一棵树是不是 BST

考察点：数据结构

参考回答：

BST（二叉排序树）：

- 1、每个结点都有一个作为搜索依据的关键码，所有结点的关键码不同
- 2、左子树上所有结点的关键码小于根节点的关键码
- 3、右子树所有结点的关键码大于根节点的关键码
- 4、左子树和右子树也是 BST

判断一棵树是不是 BST

```
class Node {

    int data;
```



```
Node left;

Node right;

}

public class BSTChecker {

    private static int lastVisit = Integer.MIN_VALUE;

    public static boolean isBST(Node root) {

        if (root == null) return true;

        boolean judgeLeft = isBST(root.left); // 先判断左子树

        if (root.data >= lastVisit && judgeLeft) { // 当前节点比上次访问的数值要大

            lastVisit = root.data;

        } else {

            return false;

        }

        boolean judgeRight = isBST(root.right); // 后判断右子树

        return judgeRight;

    }

}
```

## 2、手写代码：给一个二叉树，怎么得到这棵树的镜像

考察点：编程

参考回答：



```
public class Solution {  
  
    public void Mirror(TreeNode root) {  
  
        if(root==null){  
  
            return;  
  
        }  
  
        TreeNode temp = root.left;  
  
        root.left = root.right;  
  
        root.right = temp;  
  
        Mirror(root.left);  
  
        Mirror(root.right);  
  
    }  
  
}
```

### 3、手写代码：通过前序和中序还原二叉树

考察点：编程

参考回答：

//算法1

```
#include <iostream>  
  
#include <fstream>  
  
#include <string>  
  
  
struct TreeNode  
  
{  
  
    struct TreeNode* left;  
  
    struct TreeNode* right;
```



```
char elem;

};

TreeNode* BinaryTreeFromOrderings(char* inorder, char* preorder, int length)
{
    if(length == 0)
    {
        return NULL;
    }

    TreeNode* node = new TreeNode;

    node->elem = *preorder;

    int rootIndex = 0;

    for(;rootIndex < length; rootIndex++)
    {
        if(inorder[rootIndex] == *preorder)
            break;
    }

    node->left = BinaryTreeFromOrderings(inorder, preorder +1, rootIndex);

    node->right = BinaryTreeFromOrderings(inorder + rootIndex + 1, preorder +
    rootIndex + 1, length - (rootIndex + 1));

    std::cout<<node->elem<<std::endl;

    free(node);

    return NULL;
}
```



```
int main(int argc, char** argv) {  
  
    char* pr="GDAFEMHZ";  
  
    char* in="ADEFGHMZ"; BinaryTreeFromOrderings(in, pr, 8); printf("\n"); return 0;}
```

#### 4、手写代码：求二叉树每层最大值

考点：算法

参考回答：

思路：遍历树，给树的每一个节点赋予层数的信息，定义一个数组 maxNum 数组，存储每层中最大值，其下标 i 对应于树中的第 i 层，其中根节点为第 0 层。在遍历某个树节点时，先判断该节点的层数与当前数组中已存储的数是否相等，若相等，则将该节点值与当前数组最后一位值进行比较，数组取其中的最大值。若不相等，则该节点为该层第一个节点，直接将该节点值赋值给数组对应的位。深度遍历该二叉树，最后输出这个数组即可。

```
class Solution {  
  
public:  
  
    vector<int> largestValues(TreeNode* root) {  
  
        if(root == NULL) {  
  
            return maxNum;  
  
        }  
  
        travel(root, level);  
  
        return maxNum;  
  
    }  
  
    TreeNode* travel(TreeNode* root, int level) {  
  
        if(!root) return NULL;  
  
        if(maxNum.size() < level + 1) {  
  
            maxNum.push_back(root->val);  
  
        } else {  
  
            maxNum[level] = max(root->val, maxNum[level]);  
  
        }  
  
        if(root->left) travel(root->left, level + 1);  
        if(root->right) travel(root->right, level + 1);  
    }  
};
```



```
    }

    travel(root->left, level+1);

    travel(root->right, level+1);

    return root;

}

private:

int level = 0;

vector<int> maxNum;

};
```

#### 5、手写代码：两个平衡二叉树合并是怎么做的

考察点：编程

参考回答：

首先，将两棵树分别展开为有序链表

```
public TreeNode prev = null;

    public void BSTtoLinkedList(TreeNode root) {

        if (root == null) return;

        BSTtoLinkedList(root.left);

        if (prev != null) {

            prev.right = root;

            prev.left = null;

        }

        prev = root;

        BSTtoLinkedList(root.right);

    }
```

然后将两个有序链表合并



```
public TreeNode MergeTwoLinkedList(TreeNode n1, TreeNode n2) {  
  
    TreeNode head = new TreeNode();  
  
    while (n1 != null && n2 != null) {  
  
        if (n1.val < n2.val) {  
  
            head.right = n1;  
  
            n1 = n1.right;  
  
        } else {  
  
            head.right = n2;  
  
            n2 = n2.right;  
  
        }  
  
        head = head.right;  
  
    }  
  
    if (n1 != null) {  
  
        head.right = n1;  
  
        head = head.right;  
  
    }  
  
    if (n2 != null) {  
  
        head.right = n2;  
  
        head = head.right;  
  
    }  
  
    return head.right;  
  
}
```

## 6、手写代码：求全体二叉树节点最大值

考点：算法

参考回答：





```
public int maxTreeNode(TreeNode root) {  
  
    if (root.left == null && root.right == null) {  
  
        return root.val;  
  
    } else {  
  
        if (root.left != null && root.right != null) {  
  
            return root.val > maxTreeNode(root.left) ? (root.val > maxTreeNode(root.right) ?  
root.val  
  
            : maxTreeNode(root.right))  
  
            : (maxTreeNode(root.left) > maxTreeNode(root.right) ? maxTreeNode(root.left)  
  
            : maxTreeNode(root.right));  
  
        } else if (root.left == null && root.right != null) {  
  
            return root.val > maxTreeNode(root.right) ? root.val  
  
            : maxTreeNode(root.right);  
  
        } else {  
  
            return root.val > maxTreeNode(root.left) ? root.val  
  
            : maxTreeNode(root.left);  
  
        }  
  
    }  
  
}
```

## 7、手写代码：二叉树深度优先遍历

考察点：编程

参考回答：

//深度优先搜索

//利用栈，现将右子树压栈再将左子树压栈

```
void DepthFirstSearch(BitNode *root)
```

```
{
```



```
stack<BitNode*> nodeStack;

nodeStack.push(root);

while (!nodeStack.empty())

{

    BitNode *node = nodeStack.top();

    cout << node->data << ' ';

    nodeStack.pop();

    if (node->right)

    {

        nodeStack.push(node->right);

    }

    if (node->left)

    {

        nodeStack.push(node->left);

    }

}
```

## 6、递归

### 1、手写代码：青蛙跳台阶

考点：算法

参考回答：

递归：

```
int Fib(int n) {

    if (1 == n || 2 == n)
```



```
return n;

else

return Fib(n - 1) + Fib(n - 2);

}

}
```

非递归:

```
public static int calc3(int n) {

if (1 == n || 2 == n)

return n;

int s1 = 1, s2 = 2, s3 = 1;

for (int i = 3; i <= n; i++) {

s3 = s1 + s2;

s1 = s2;

s2 = s3;

}

return s3;

}
```

## 7、字符串

### 1、手写代码：两个字符串的最长公共子序列？

考察点：编程

参考回答：

```
class LCS:

def findLCS(self, A, n, B, m):

dp=[([0]*(m+1)) for i in range(n+1)]
```



```
for i in range(1,n+1):  
  
    for j in range(1,m+1):  
  
        if(A[i-1]==B[j-1]):  
  
            dp[i][j]=dp[i-1][j-1]+1  
  
        else:  
  
            dp[i][j]=max(dp[i-1][j],dp[i][j-1])  
  
# write code here  
  
return dp[n][m]
```

## 2、手写代码：字符串逆序

考察点：编程

参考回答：

```
char* str_reverse(char* str)  
{  
  
    int n = strlen(str) / 2;  
  
    int i = 0;  
  
    char tmp = 0;  
  
    for(i = 0; i < n; i++)  
    {  
  
        tmp = str[i];  
  
        str[i] = str[strlen(str)-i-1];  
  
        str[strlen(str)-i-1] = tmp;  
  
    }  
  
    return str;
```



```
}
```

### 3、手写代码：字符串复制函数

考点：算法

参考回答：

模拟 strcpy：

```
char* my_strcpy(char* dst, const char*src)
{
    assert(dst != NULL);
    assert(src != NULL);
    char *ret = dst;
    while (*dst = *src)
    {
        dst++, src++;
    }
    return ret;
}
```

模拟 strncpy：

```
char* my_strncpy(char* dst, const char* src, int n)
{
    assert(dst!=NULL);
    assert(src!=NULL);
    char* ret=dst;
    while(n)
    {
        *dst=*src;
```



```
dst++;

src++;

n--;

}

if(*(dst-1)!='\0')

{

*dst='\0';

}

return ret;

}

模拟 strcat:

char *my_strcat(char* dst, char* src)

{

assert(dst != NULL);

assert(src != NULL);

char *ret = dst;

while (*dst)

{

dst++;

}

while (*dst = *src)

{

dst++, src++;

}

return ret;
```



```
}

模拟 strncat:

char *my_strncat(char* dst, const char *src, int n)

{

    assert(dst != NULL);

    assert(src != NULL);

    char *ret = dst;

    while (*dst)//将 dst 的指针移到 '\0' 处

    {

        dst++;

    }

    while (n)

    {

        *dst = *src;

        dst++, src++;

        n--;

    }

    if (*(dst - 1) != '\0')

    {

        *dst = '\0';

    }

    return ret;

}
```



4、手写代码：驼峰字符串问题，给定一个驼峰样式的字符串 例如

“AaABbBcBbcvQv.....” -> “bc”，两个一样的字符夹着一个不一样的字符， 处理这个字符串去掉所有的驼峰。

考点：算法

参考回答：

```
int main()
{
    char buffer[80]="AaABbBcBbcvQv";

    for(int i=0;buffer[i]!='\0';i++){

        if(buffer[i+2]=='\0'){

            printf("ok");

            printf("%c\n",buffer[i]);

            return 0;

        }

        if(buffer[i]==buffer[i+2] && buffer[i]!=buffer[i+1]){

            int j=i;

            while(buffer[j+3]!='\0'){

                buffer[j]=buffer[j+3];

                j++;

            }

            buffer[(j+2)]='\0';

            if((i-1)>0){

                i=i-1;

            }

        }

        printf("%c\n",buffer[i]);

    }
}
```





```
    return 0;

}
```

## 5、手写代码：给一个字符串找出第一个只出现一次的字符位置

考察点：编程

参考回答：

```
def first_not_repeating_char(string):

    if not string:

        return -1

    resultDict = {}

    for k, s in enumerate(string):

        resultDict[s] = [resultDict[s][0] + 1, k] if resultDict.get(s) else [1, k]

    pos = len(string)

    ret = None

    for x in resultDict :

        if resultDict[x][0] == 1 and resultDict[x][1] < pos:

            pos = resultDict[x][1]

            ret = (x, pos)

    return ret
```

## 8、堆与栈

### 1、请你说一说堆和栈的区别

考点：数据结构

参考回答：

栈区（stack）——由编译器自动分配释放，存放函数的参数值，局部变量的值等。其操作方式类似于数据结构中的栈。



堆区 (heap) — 一般由程序员分配释放，若程序员不释放，程序结束时可能由 OS 回收。它与数据结构中的堆是两回事，分配方式类似于链表。

区别：

- 申请方式： 栈：由系统自动分配。例如，声明在函数中一个局部变量 `int b`；系统自动在栈中为 `b` 开辟空间。堆：需要程序员自己申请，并指明大小，在 `c` 中用 `malloc` 函数，在 `C++` 中用 `new` 运算符。

- 申请后系统的响应：栈：只要栈的剩余空间大于所申请空间，系统将为程序提供内存，否则将报异常提示栈溢出。堆：首先应该知道操作系统有一个记录空闲内存地址的链表，当系统收到程序的申请时，会遍历该链表，寻找第一个空间大于所申请空间的堆结点，然后将该结点从空闲结点链表中删除，并将该结点的空间分配给程序，对于大多数系统，会在这块内存空间中的首地址处记录本次分配的大小，这样，代码中的 `delete` 语句才能正确的释放本内存空间。另外，由于找到的堆结点的大小不一定正好等于申请的大小，系统会自动的将多余的那部分重新放入空闲链表中。

- 申请大小的限制：栈：在 Windows 下，栈是向低地址扩展的数据结构，是一块连续的内存的区域。这句话的意思是栈顶的地址和栈的最大容量是系统预先规定好的，在 WINDOWS 下，栈的大小是 2M（也有的说是 1M，总之是一个编译时就确定的常数），如果申请的空间超过栈的剩余空间时，将提示 `overflow`。因此，能从栈获得的空间较小。堆：堆是向高地址扩展的数据结构，是不连续的内存区域。这是由于系统是用链表来存储的空闲内存地址的，自然是不连续的，而链表的遍历方向是由低地址向高地址。堆的大小受限于计算机系统中有效的虚拟内存。由此可见，堆获得的空间比较灵活，也比较大。

- 申请效率的比较：栈：由系统自动分配，速度较快。但程序员是无法控制的。堆是由 `new` 分配的内存，一般速度比较慢，而且容易产生内存碎片，不过用起来最方便。另外，在 WINDOWS 下，最好的方式是用 `VirtualAlloc` 分配内存，他不是在堆，也不是在栈是直接在进程的地址空间中保留一块内存，虽然用起来最不方便。但是速度快，也最灵活。

- 堆和栈中的存储内容：栈：在函数调用时，第一个进栈的是主函数中后的下一条指令（函数调用语句的下一条可执行语句）的地址，然后是函数的各个参数，在大多数的 C 编译器中，参数是由右往左入栈的，然后是函数中的局部变量。注意静态变量是不入栈的。当本次函数调用结束后，局部变量先出栈，然后是参数，最后栈顶指针指向最开始存的地址，也就是主函数中的下一条指令，程序由该点继续运行。堆：一般是在堆的头部用一个字节存放堆的大小。堆中的具体内容程序员安排。

- 存取效率的比较：访问堆的一个具体单元，需要两次访问内存，第一次得取得指针，第二次才是真正得数据，而栈只需访问一次。另外，堆的内容被操作系统交换到外存的概率比栈大，栈一般是不会被交换出去的。因此从操作系统层面来看，栈的效率比堆高。

## 2、请你说一说堆和栈的区别

考察点：数据结构

参考回答：

堆栈空间分配区别：

1、栈（操作系统）：由操作系统自动分配释放，存放函数的参数值，局部变量的值等。其操作方式类似于数据结构中的栈；

2、堆（操作系统）：一般由程序员分配释放，若程序员不释放，程序结束时可能由 OS 回收，分配方式倒是类似于链表。

二、堆栈缓存方式区别：

1、栈使用的是一级缓存，他们通常都是被调用时处于存储空间中，调用完毕立即释放；

2、堆是存放在二级缓存中，生命周期由虚拟机的垃圾回收算法来决定（并不是一旦成为孤儿对象就能被回收）。所以调用这些对象的速度要相对来得低一些。

三、堆栈数据结构区别：

堆（数据结构）：堆可以被看成是一棵树，如：堆排序；

栈（数据结构）：一种先进后出的数据结构。

### 3、请问有一些数，每次可以插入，或者取出第 1/4 大的数，应该用什么数据结构？

考察点：数据结构

参考回答：维护当前数据量 1/4 大小的 最小堆，插入时被挤出来的数用最大堆保存，取操作从最小堆顶部取，然后把最大堆顶部取出插入最小堆。瞎说的不知道对不对 不过看面试官当时的态度反馈 应该还算满意）by 提供面经的同学

## 9、动态规划

1、系统会给定一串数字让玩家选择，如果玩家选中一个数字，比如 M，那么玩家获得 M 分，但同时当前选中的 M，以及这串数字中所有的 M+1 和 M-1 将会全部消失。玩家可以继续选择得分，直到串为空。最终系统会根据玩家获得的积分发送奖励，积分越高，奖励越丰厚。例如系统给定的数字是[2, 3, 3, 3, 4]，如果玩家选定了 2，玩家得 2 分，并且选中的 2 和所有的 1 和 3 会消失，那么数组只剩下[4]，玩家再选择 4，数组为空，此时一共获得 6 分。如果玩家首先选中的是 3，那么玩家得 3 分，选中的 3，以及 2 和 4 都会消失，数字剩下[3, 3]，第二次和第三次玩家可以再次选择 3，这样选择一共得 9 分，这也是最优的选择方式。

考点：动态规划

参考回答：

```
public class Select {  
  
    public static void main(String[] args) {  
  
        int[] a = {2, 3, 3, 3, 4};
```



```
int max = 0;

for(int i=0;i<a.length;i++) {

    if(max<a[i])max=a[i];

}

int[] num = new int[max+1];

for(int i=0;i<a.length;i++) {

    num[a[i]]++;

}

System.out.println(maxSum(1,max,num));

}

public static int maxSum(int start, int end, int[] num){

    int[] sum = new int[end+1];

    sum[1] = num[1]*1;

    sum[2] = Max(num[1]*1,num[2]*2);

    sum[3] = Max((num[1]*1+num[3]*3),num[2]*2);

    for(int i=4;i<end+1;i++) {

        sum[i] = Max(num[i]*i+sum[i-2],num[i-1]*(i-1)+sum[i-3]);

    }

    return sum[end];

}

public static int Max(int m, int n){

    if(m>n)return m;

    else return n;

}

}
```



2、手写代码：给一个英文文本 “i have a dream i am a human you can have dream too.” 再给一个文本 “i you am ”，要求计算出第一个文本中包含第二个文本每个单词的最短文本，比如例子中最短文本就是 “i am a human you”。

考点：算法

参考回答：

```
#!/usr/bin/python

#encoding:utf-8

def lengthCa(arr):

    start = arr[0]

    end= arr[0]

    for i in arr:

        if i >start:

            start=i

        if i < end :

            end = i

    return start-end

#输入的第二个文本

text = 'ILM runs a batch processing environment capable of modeling, rendering and
compositing tens of thousands of motion picture frames per day. Thousands of machines
running Linux, IRIX, Compaq Tru64, OS X, Solaris, and Windows join together to provide
a production pipeline used by ~800 users daily. Speed of development is key, and Python
was a faster way to code (and re-code) the programs that control this production
pipeline.'

#输入的第二个文本

keywords='a of'

newtext=text.split(' ')

newkeys = keywords.split(' ')

textLen = len(newtext)
```



```
array=[]

#把 index 计算出来，用做最优路径规划使用

for i in newkeys:

    dan=[]

    for j in range(textLen):

        if i == newtext[j]:

            dan.append(j)

    array.append(dan)

print(array)

#最优规划开始

caculateArray=[]

for n in array[0]:

    temp=[]

    temp.append(n)

    caculateArray.append(n)

flag = 0

for n in array:

    if array.index(n) == 0:

        continue

    temparr=[]

    for m in caculateArray:#遍历当前最短路径

        index = caculateArray.index(m)#计算当前路径的 index 值

        tempminlen=1000000

        tempminarr=[]

        for j in n:#计算当前最短路径，添加下一个节点
```



```
if flag ==0:

    temparr.append(m)

else:

    for x in m:

        temparr.append(x)

    temparr.append(j)

    if lengthCa(temparr)<tempminlen :

        tempminlen=lengthCa(temparr)

        tempminarr=temparr

    temparr=[]

    caculateArray[index]=tempminarr

print(caculateArray)

flag+=1

tempminlen=1000000

tempminarr=[]

#找出最终所有解里的最优解，为 tempminarr

for n in caculateArray:

    if lengthCa(n)<tempminlen :

        tempminlen=lengthCa(n)

        tempminarr=n

#计算 tempminarr 的起点和重点，现在发现用 min() 和 max() 函数就可以了

start = tempminarr[0]

end = tempminarr[0]

for i in tempminarr:

    if start<i:
```



```
start=i

if end>i:

end=i

#输出起始位置和终止位置

print(start,end)

for m in range(end,start+1):

print newtext[m]
```

3、手写代码：给你一个格子，一个人在格子的左上角，他只能向右走一步，或者向下走一步，他走到右下角共有多少种方法

考点：算法

参考回答：

```
#include<stdio.h>

int n,m,dp[10005][10005];

int main()

{

while(~scanf("%d%d",&n,&m))

{

dp[0][0]=0;

for(int i=1; i<=n; i++)

dp[i][0]=1;

for(int j=1; j<=m; j++)

dp[0][j]=1;//初始化

for(int i=1; i<=n; i++)

for(int j=1; j<=m; j++)

{
```





```
dp[i][j]=dp[i-1][j]+dp[i][j-1]; //动态规划转移方程

}

printf("%d\n", dp[n][m]);

}

return 0;

}
```

#### 4、请你讲一下动态链表和静态链表的区别

考点：数据结构

参考回答：

静态链表是用类似于数组方法实现的，是顺序的存储结构，在物理地址上是连续的，而且需要预先分配地址空间大小。所以静态链表的初始长度一般是固定的，在做插入和删除操作时不需要移动元素，仅需修改指针。

动态链表是用内存申请函数(malloc/new)动态申请内存的，所以在链表的长度上没有限制。动态链表因为是动态申请内存的，所以每个节点的物理地址不连续，要通过指针来顺序访问。

#### 5、请你说一下递归和动态规划的区别？

考察点：算法

参考回答：

递归法是算法调用自身，动态规划是将一个问题分解成若干个子问题，对大问题的求解转化为对子问题的求解。动态规划有时可以通过递归实现，通常用在最优问题的求解

#### 6、手写代码：01 背包

考察点：算法

参考回答：

```
void FindMax() //动态规划

{

    int i, j;
```



```
//填表

for(i=1;i<=number;i++)

{

    for(j=1;j<=capacity;j++)

    {

        if(j<w[i])//包装不进

        {

            V[i][j]=V[i-1][j];

        }

        else//能装

        {

            if(V[i-1][j]>V[i-1][j-w[i]]+v[i])//不装价值大

            {

                V[i][j]=V[i-1][j];

            }

            else//前 i-1 个物品的最优解与第 i 个物品的价值之和更大

            {

                V[i][j]=V[i-1][j-w[i]]+v[i];

            }

        }

    }

}

}
```

## 10、高级算法

### 1、手写代码：LCS 问题

考察点：编程

参考回答：

最长公共子序列代码

```
void LCS(string X, string Y, vector & a)

{

    int m=X.length(),n=Y.length();

    for (int j = 1; j <= n; j++) {

        a[0]=0;//内层循环结束,a[0]要置零,想象求 a[1][j]所依赖的 a[0][j-1]=0,即是初始化步骤

        for (int i = 1; i <= m; i++) {

            int pre=a[0];

            int temp = a[i];

            if (X[i - 1] == Y[j - 1])

                a[i] =pre+1;

            else a[i] = max(a[i],a[i-1]);

            a[0]=temp;

        }

    }

}

//c[i][j]表示 X 的前 i 个字符和 Y 的前 j 个字符所共有的序列长度;b[i][j]用来回溯得到的公共子序列

void LCS_LENGTH(string X,string Y,vector> & b,vector> & c)

{

    int m=X.length(),n=Y.length();
```



```
for(int i=0;i<=m;i++)

    c[i][0]=0;

for(int j=0;j<=n;j++)

    c[0][j]=0;

int i, j;

for(i=1;i<=m;i++)

    for(j=1;j<=n;j++)

        if(X[i-1]==Y[j-1])

        {

            c[i][j]=c[i-1][j-1]+1;

            b[i][j]=2;

        }else if(c[i-1][j]>=c[i][j-1])

        {

            c[i][j]=c[i-1][j];

            b[i][j]=1;

        }else

        {

            c[i][j]=c[i][j-1];

            b[i][j]=0;

        }

    }

//利用回溯和表 b, 追踪 LCS

void PRINT_LCS(vector<int> & b, string X,int xlen, int ylen,string &s)

{

    }
```



```
int i=xlen, j=ylen;

if(i==0 || j==0)

    return ;

if(b[i][j]==2)//此时第 i 个元素 X[i-1][j-1]属于公共元素

{

    PRINT_LCS(b, X, i-1, j-1, s);

    s+=X[i-1];

}else if(b[i][j]==1)//上面有 c[i][j]=c[i-1][j];此时回溯 c[i-1][j]

    PRINT_LCS(b, X, i-1, j, s);

else

    PRINT_LCS(b, X, i, j-1, s);

}
```

## 2、请问你知道什么加密算法吗

考察点：加密算法

参考回答：

常见的加密算法分为三类，对称加密算法，非对称加密算法，Hash 算法

对称加密，指加密和解密使用相同密钥的加密算法，常见的有 DES、3DES、DEFS，RC4、RC5，AES

非对称加密，常见的有 RSA、DSA

Hash 算法：它是一种单向算法，用户可以通过 h a s h 算法对目标信息生成一段特定长度的唯一 Hash 值，但不能通过这个 Hash 值重新获得目标信息，常见的有 MD2、MD4、MD5

## 3、手写代码：斐波那契数列

考察点：编程

参考回答：

```
long long Fib(long long N)
```



```
{  
  
    long long first = 1;  
  
    long long second = 1;  
  
    long long ret = 0;  
  
    for (int i = 3; i <=N; ++i)  
    {  
  
        ret = first + second;  
  
        first = second;  
  
        second = ret;  
  
    }  
  
    return second;  
  
}
```

## 11、查找

### 1、手写代码：二分查找的代码

考察点：编程

参考回答：

```
int binarySearch(vector<int> x,int t)  
{  
  
    int mid,l=0;  
  
    int u = x.size()-1;  
  
    while(l<=u)  
    {  
  
        mid = l+(u-l)/2;  
  
        if(x[mid]<t)
```



```
        l=m+1;

        else if(x[m]>t)

            u=m-1;

        else

            return m;

    }

    return -1;//没有找到对应的元素，返回-1.

}
```

2、一个二维坐标系，给你  $n$  个点的坐标，画一条直线把他们分成两份（任意直线），要求数量尽量等分，复杂度不能太高。

思路参考：我下意识觉得考察我图论，想了一下感觉不是的，然后给他讲了讲我的思路：假设按照  $y$  坐标分，那么遍历  $n$  个坐标，用一个最大堆来保存，然后从顶部弹出  $n/2$  次，如果当前顶部的数和最后弹出的数不一样，就可以在中间画一条线（他说如果所有点纵坐标都一样呢？如果有点重合的呢？（我有点蒙蔽了，觉得我这思路是错的，又想了几分钟其他的方法，又不由自主往图论上想，最后还是没想起来。其实貌似再同样做法对横坐标处理一次应该可以避免他说的吧），最后问他怎么做，他说用二分思想，然后对横纵坐标求中位数、众数什么的。

## 12、哈希

1、请你说一下哈希表是做什么的？另外哈希表的实现原理也说一下

考察点：数据结构

参考回答：

Hash 表即散列表，其最突出的优点是查找和插入删除具有常数时间的复杂度

其实现原理是：把 Key 通过一个固定的算法函数即所谓的哈希函数转换成一个整型数字，然后就将该数字对数组长度进行取余，取余结果就当作数组的下标，将 value 存储在以该数字为下标的数组空间里。

而当使用哈希表进行查询的时候，就是再次使用哈希函数将 key 转换为对应的数组下标，并定位到该空间获取 value，如此一来，就可以充分利用到数组的定位性能进行数据定位。

哈希表最大的优点，就是把数据的存储和查找消耗的时间大大降低，几乎可以看成是常数时间；而代价仅仅是消耗比较多的内存。然而在当前可利用内存越来越多的情况下，用空间换时间



的做法是值得的。另外，编码比较容易也是它的特点之一。哈希表又叫做散列表，分为“开散列”和“闭散列”。

我们使用一个下标范围比较大的数组来存储元素。可以设计一个函数（哈希函数，也叫做散列函数），使得每个元素的关键字都与一个函数值（即数组下标）相对应，于是用这个数组单元来存储这个元素；也可以简单的理解为，按照关键字为每一个元素“分类”，然后将这个元素存储在相应“类”所对应的地方。

但是，不能够保证每个元素的关键字与函数值是一一对应的，因此极有可能出现对于不同的元素，却计算出了相同的函数值，这样就产生了“冲突”，换句话说，就是把不同的元素分在了相同的“类”之中。后面我们将看到一种解决“冲突”的简便做法。总的来说，“直接定址”与“解决冲突”是哈希表的两大特点。

**2、现在有 100W 个账户密码，要存起来，要求查找时速度尽可能快，你选择什么数据结构？为什么？**

考察点：数据结构

参考回答：

选择 hash\_map，因为其查找速度与数据量基本无关，是常数级别，但是对空间的要求很高，所以是已空间换时间

## 13、图

**1、请问你对图论算法了解多少？（BFS,DFS,最短路径，最小生成树，最小割最大流...）平常有用过吗？**

考察点：数据结构

参考回答：

DFS：深度优先搜索算法思路：

从顶点 V 开始，访问这个顶点，然后依次从 V 的未被访问的邻接点出发深度优先遍历图，直至图中所有和 V 有路径的相通的顶点都被访问了，如果此时还有顶点未被访问，则选择图中未被访问的那个顶点作为起点，重复上述动作。

```
void DFS_store_array(Graph_array g,int v,bool * & visit) {  
  
    cout << g.infromation[v] << " ";  
  
    visit[v] = true;  
  
    for (int i = 0; i < g.vexnum; i++) { //找出下一个位被访问的顶点
```





```
        if (g.arc[v][i] == 0 || g.arc[v][i] == INT_MAX) { //如果两个顶点不存在边

            continue;

        }

        else if (!visit[i]) { //如果没有被访问，访问该结点

            DFS_store_array(g, i, visit);

        }

    }

}
```

BFS 广度优先搜索思路就是：假设从图中的顶点  $V$  出，在访问了  $v$  之后，依次访问  $v$  的各个未被访问的邻接点，然后，分别从这些邻接点出发，依次访问他们的邻接点，并使“先被访问的顶点的邻接点”先于“后被访问的邻接点”先被访问，直至图中所有的顶点都被访问到为止，防止出现非连通图的情况，我们需要最后遍历一遍，看是否所有的点都被访问了，如果有未被访问的点，那么就把该点作为一个新的起点。

```
void BFS_list(Graph_List g, int begin) {

    int i;

    bool * visit = new bool[g.vexnum];

    for (i = 0; i < g.vexnum; i++) {

        visit[i] = false;

    }

    cout << "图的 BFS 遍历结果: " << endl;

    queue<int> q;

    for (int v = 0; v < g.vexnum; v++) {

        if (!visit[(begin - 1 + v) % g.vexnum]) //注意起点不一定是 v1

        {

            cout << g.node[(begin - 1 + v) % g.vexnum].data << " ";

            visit[(begin - 1 + v) % g.vexnum] = true;

            q.push((begin - 1 + v) % g.vexnum); //初始化我们的队列

        }

    }

}
```



```
while (!q.empty())

{

    int u = q.front();

    q.pop();

    ArcNode * next;

    next = g.node[u].firstarc;//获得依附在该顶点的第一条边的信息

    while (next) { //遍历该链表上的所有的点

        if (!visit[next->advex]) {

            cout << g.node[next->advex].data << " ";

            visit[next->advex] = true;

            q.push(next->advex);

        }

        next = next->next;

    }

}

}
```

## 五、框架

### 1、Spring

#### 1、请问 Spring 声明一个 bean 如何对其进行个性化定制

考察点框架

参考回答：略

## 2、请你说一下 Spring AOP

考察点：编程

参考回答：

AOP (Aspect Oriented Programming)，即面向切面编程，可以说是 OOP (Object Oriented Programming，面向对象编程) 的补充和完善。OOP 引入封装、继承、多态等概念来建立一种对象层次结构，用于模拟公共行为的一个集合。不过 OOP 允许开发者定义纵向的关系，但并不适合定义横向的关系，例如日志功能。日志代码往往横向地散布在所有对象层次中，而与它对应的对象的核心功能毫无关系对于其他类型的代码，如安全性、异常处理和透明的持续性也都是如此，这种散布在各处的无关的代码被称为横切 (cross cutting)，在 OOP 设计中，它导致了大量代码的重复，而不利于各个模块的重用。

AOP 技术恰恰相反，它利用一种称为“横切”的技术，剖解封装的对象内部，并将那些影响了多个类的公共行为封装到一个可重用模块，并将其命名为“Aspect”，即切面。所谓“切面”，简单说就是那些与业务无关，却为业务模块所共同调用的逻辑或责任封装起来，便于减少系统的重复代码，降低模块之间的耦合度，并有利于未来的可操作性和可维护性。

使用“横切”技术，AOP 把软件系统分为两个部分：核心关注点和横切关注点。业务处理的主要流程是核心关注点，与之关系不大的部分是横切关注点。横切关注点的一个特点是，他们经常发生在核心关注点的多处，而各处基本相似，比如权限认证、日志、事物。AOP 的作用在于分离系统中的各种关注点，将核心关注点和横切关注点分离开来。

## 3、请问 J2EE 上 request 请求先经过谁 然后再转交到 SpringMVC 上

考察点：J2EE

参考回答：DispatcherServlet

## 4、请你回答一下 Filter 主要是做什么的，filter 和 Servlet 先过哪个，FilterChain 是什么意思

考察点过滤器

参考回答：

Filter：过滤器，过滤器是一些 web 应用程序组件，可以绑定到一个 web 应用程序中，但是与其他 web 应用程序组件不同的是，过滤器是链在容器的处理过程中的，这就意味它们会在 servlet 处理器之前访问一个进入的请求，并且在外发响应信息返回到客户前访问这些响应信息。这种访问使得过滤器可以检查并修改请求和响应的内容。

filter 和 Servlet 先过哪个

先执行 filter 再执行 servlet，servlet 执行完后再执行 filter



FilterChain 是什么意思

过滤链 FilterChain

两个过滤器，EncodingFilter 负责设置编码，SecurityFilter 负责控制权限，服务器会按照 web.xml 中过滤器定义的先后循序组装成一条链，然后一次执行其中的 doFilter() 方法。执行的顺序就如下图所示，执行第一个过滤器的 chain.doFilter() 之前的代码，第二个过滤器的 chain.doFilter() 之前的代码，请求的资源，第二个过滤器的 chain.doFilter() 之后的代码，第一个过滤器的 chain.doFilter() 之后的代码，最后返回响应。

## 六、Linux，SQL 等

### 1、请问 linux 两台机器之间传文件，用的什么端口

考点：linux

参考回答：

Linux 主机之间传输文件的几种方法：

#### 1、scp 传输

scp 传输速度较慢，但使用 ssh 通道保证了传输的安全性。

命令：

将本地文件拷贝到远程：

scp 文件名 - 用户名@计算机 IP 或者计算机名称:远程路径

从远程将文件拷回本地：

scp - 用户名@计算机 IP 或者计算机名称:文件名 本地路径

#### 2、rsync 差异化传输(支持断点续传, 数据同步)

rsync 是 Linux 系统下的文件同步和数据传输工具，它采用“rsync”算法，可以将一个客户机和远程文件服务器之间的文件同步，也可以在本地系统中将数据从一个分区备份到另一个分区上。

如果 rsync 在备份过程中出现了数据传输中断，恢复后可以继续传输不一致的部分。rsync 可以执行完整备份或增量备份。

#### 3、管道传输(降低 IO 开销)

```
gzip -c sda.img | ssh root@192.168.1.110 "gunzip -c - > /image/sda.img"
```

#对 sda.img 使用 gzip 压缩, -c 参数表示输出到 stdout, 即通过管道传送



#gunzip -c - 中的“-”表示接收从管道传进的 stdin

#### 4、nc 传输(一种网络的数据流重定向)

nc 所做的就是在两台电脑之间建立 tcp 或 udp 链接,并在两个端口之间传输数据流,是一种网络的数据流重定向。

使用 dd 结合 nc 命令网络克隆磁盘分区:

主机:

```
dd if=/dev/vda | gzip -c | nc -l 50522
```

待恢复机:

```
nc 192.168.215.63 50522 | gzip -dc | dd of=/dev/sda
```

dd 命令克隆/dev/vda 磁盘,并使用 gzip 压缩,把数据流重定向到本机 50522 端口,待恢复机上使用 nc 连接主机 50522 端口,就能接收主机 50522 端口的比特数据流,然后使用 gzip 解压缩,并恢复到/dev/sda 磁盘。

dd 命令读取的是磁盘扇区,所以不论磁盘文件系统,或者分区表,磁盘 MBR 信息,dd 都能够复制,可以使用 bs, count 参数控制要克隆的大小

#### 5、建立文件服务器

通过建立文件服务器,然后通过网络挂载的方式传输,适用于经常性的拷贝。

## 2、请你说一说关于 linux 查看进程

考点: linux

参考回答:

ps 命令:

ps 命令查找与进程相关的 PID 号:

ps a 显示现行终端机下的所有程序,包括其他用户的程序。

ps -A 显示所有程序。

ps c 列出程序时,显示每个程序真正的指令名称,而不包含路径,参数或常驻服务的标示。

ps -e 此参数的效果和指定“A”参数相同。

ps e 列出程序时,显示每个程序所使用的环境变量。

ps f 用 ASCII 字符显示树状结构,表达程序间的相互关系。



ps -H 显示树状结构，表示程序间的相互关系。

ps -N 显示所有的程序，除了执行 ps 指令终端机下的程序之外。

ps s 采用程序信号的格式显示程序状况。

ps S 列出程序时，包括已中断的子程序资料。

ps -t<终端机编号> 指定终端机编号，并列出属于该终端机的程序的状况。

ps u 以用户为主的格式来显示程序状况。

ps x 显示所有程序，不以终端机来区分。

最常用的方法是 ps aux

### 3、请你说几个基本 Linux 命令

考点：linux

参考回答：

1、file

作用：file 通过探测文件内容判断文件类型，使用权限是所有用户。

格式：file [options] 文件名

[options]主要参数

-v：在标准输出后显示版本信息，并且退出。

-z：探测压缩过的文件类型。

-L：允许符合连接。

2、mkdir

作用：mkdir 命令的作用是建立名称为 dirname 的子目录，与 MS DOS 下的 md 命令类似，它的使用权限是所有用户。

格式：mkdir [options] 目录名

[options]主要参数

-m, --mode=模式：设定权限<模式>;，与 chmod 类似。

-p, --parents：需要时创建上层目录；如果目录早已存在，则不当作错误。

-v, --verbose：每次创建新目录都显示信息。



--version: 显示版本信息后离开。

### 3、grep

作用：grep 命令可以指定文件中搜索特定的内容，并将含有这些内容的行标准输出。grep 全称是 Global Regular Expression Print，表示全局正则表达式版本，它的使用权限是所有用户。

格式：grep [options]

[options]主要参数：

--c: 只输出匹配行的计数。

--i: 不区分大小写（只适用于单字符）。

--h: 查询多文件时不显示文件名。

--l: 查询多文件时只输出包含匹配字符的文件名

--n: 显示匹配行及行号。

--s: 不显示不存在或无匹配文本的错误信息。

--v: 显示不包含匹配文本的所有行。

### 4、find

作用：find 命令的作用是在目录中搜索文件，它的使用权限是所有用户。

格式：find [path][options][expression]path 指定目录路径，系统从这里开始沿着目录树向下查找文件。它是一个路径列表，相互用空格分离，如果不写 path，那么默认为当前目录。

主要参数：

[options]参数：

--depth: 使用深度级别的查找过程方式，在某层指定目录中优先查找文件内容。

--maxdepth levels: 表示至多查找到开始目录的第 level 层子目录。level 是一个非负数，如果 level 是 0 的话表示仅在当前目录中查找。

--mindepth levels: 表示至少查找到开始目录的第 level 层子目录。

--mount: 不在其它文件系统（如 Msdos、Vfat 等）的目录和文件中查找。

--version: 打印版本。



#### 4、请你说一说 Linux 命令（查看进程、top 命令、查看磁盘）

考点：linux

参考回答：

查看进程：

ps 命令：

ps 命令查找与进程相关的 PID 号：

ps a 显示现行终端机下的所有程序，包括其他用户的程序。

ps -A 显示所有程序。

ps c 列出程序时，显示每个程序真正的指令名称，而不包含路径，参数或常驻服务的标示。

ps -e 此参数的效果和指定“A”参数相同。

ps e 列出程序时，显示每个程序所使用的环境变量。

ps f 用 ASCII 字符显示树状结构，表达程序间的相互关系。

ps -H 显示树状结构，表示程序间的相互关系。

ps -N 显示所有的程序，除了执行 ps 指令终端机下的程序之外。

ps s 采用程序信号的格式显示程序状况。

ps S 列出程序时，包括已中断的子程序资料。

ps -t<终端机编号> 指定终端机编号，并列出属于该终端机的程序的状况。

ps u 以用户为主的格式来显示程序状况。

ps x 显示所有程序，不以终端机来区分。

最常用的方法是 ps aux

top 命令：

Linux top 命令用于实时显示 process 的动态。

语法

top [-] [d delay] [q] [c] [S] [s] [i] [n] [b]

参数说明：

d：改变显示的更新速度，或是在交谈式指令列( interactive command)按 s





q：没有任何延迟的显示速度，如果使用者是有 superuser 的权限，则 top 将会以最高的优先序执行

c：切换显示模式，共有两种模式，一是只显示执行档的名称，另一种是显示完整的路径与名称  
S：累积模式，会将已完成或消失的子行程（dead child process）的 CPU time 累积起来

s：安全模式，将交谈式指令取消，避免潜在的危机

i：不显示任何闲置（idle）或无用（zombie）的行程

n：更新的次数，完成后将会退出 top

b：批次档模式，搭配“n”参数一起使用，可以用来将 top 的结果输出到档案内

实例

显示进程信息# top

显示完整命令# top -c

以批处理模式显示程序信息# top -b

以累积模式显示程序信息# top -S

查看磁盘：

Linux 磁盘管理常用三个命令为 df、du 和 fdisk。

df：列出文件系统的整体磁盘使用量。df 命令参数功能：检查文件系统的磁盘空间占用情况。可以利用该命令来获取硬盘被占用了多少空间，目前还剩下多少空间等信息。语法：

df [-ahikHTm] [目录或文件名]

du：检查磁盘空间使用量。语法：du [-ahskm] 文件或目录名称

fdisk：用于磁盘分区。语法：fdisk [-l] 装置名称

选项与参数：-l：输出后面接的装置所有的分区内容。若仅有 fdisk -l 时，则系统将会把整个系统内能够搜寻到的装置的分区均列出来。

## 5、请你说几个 linux 指令

考点：linux

参考回答：

1、file



作用：file 通过探测文件内容判断文件类型，使用权限是所有用户。

格式：file [options] 文件名

[options]主要参数

-v：在标准输出后显示版本信息，并且退出。

-z：探测压缩过的文件类型。

-L：允许符合连接。

## 2、mkdir

作用：mkdir 命令的作用是建立名称为 dirname 的子目录，与 MS DOS 下的 md 命令类似，它的使用权限是所有用户。

格式：mkdir [options] 目录名

[options]主要参数

-m, --mode=模式：设定权限<模式>;，与 chmod 类似。

-p, --parents：需要时创建上层目录；如果目录早已存在，则不当作错误。

-v, --verbose：每次创建新目录都显示信息。

--version：显示版本信息后离开。

## 3、grep

作用：grep 命令可以指定文件中搜索特定的内容，并将含有这些内容的行标准输出。grep 全称是 Global Regular Expression Print，表示全局正则表达式版本，它的使用权限是所有用户。

格式：grep [options]

[options]主要参数：

-c：只输出匹配行的计数。

-i：不区分大小写（只适用于单字符）。

-h：查询多文件时不显示文件名。

-l：查询多文件时只输出包含匹配字符的文件名

-n：显示匹配行及行号。

-s：不显示不存在或无匹配文本的错误信息。



—v: 显示不包含匹配文本的所有行。

#### 4、find

作用：find 命令的作用是在目录中搜索文件，它的使用权限是所有用户。

格式：find [path][options][expression]path 指定目录路径，系统从这里开始沿着目录树向下查找文件。它是一个路径列表，相互用空格分离，如果不写 path，那么默认为当前目录。

主要参数：

[options]参数：

—depth: 使用深度级别的查找过程方式，在某层指定目录中优先查找文件内容。

—maxdepth levels: 表示至多查找到开始目录的第 level 层子目录。level 是一个非负数，如果 level 是 0 的话表示仅在当前目录中查找。

—mindepth levels: 表示至少查找到开始目录的第 level 层子目录。

—mount: 不在其它文件系统（如 Msdos、Vfat 等）的目录和文件中查找。

—version: 打印版本。

## 6、请你说一下 vector 的特性

考点：linux

参考回答：

vector 特点是：其容量在需要时可以自动分配，可以在运行时高效地添加元素，本质上是数组形式的存储方式。即在索引可以在常数时间内完成。缺点是在插入或者删除一项时，需要线性时间。但是在尾部插入或者删除，是常数时间的。

## 7、查看端口号、进程的指令是？动态查看日志的指令？怎么判断一个端口存不存在，磁盘满了怎么处理，删除一个目录下的 txt 文件，你还熟悉其他什么 linux 指令？

考察点：linux

参考回答：

查看端口号的两种指令：

netstat -tunlp|grep 端口号

lsof -i:端口号



查询进程的指令

```
ps -ef |grep 进程
```

ps:将某个进程显示出来

- A 显示所有程序。
- e 此参数的效果和指定“A”参数相同。
- f 显示 UID, PPIP, C 与 STIME 栏位。

动态查看日志

- 1、先切换到: cd usr/local/tomcat5/logs
- 2、tail -f catalina.out
- 3、这样运行时就可以实时查看运行日志了

怎么判断一个端口存不存在:

netstat -anp |grep 端口号,在输出结果中看监控状态为 LISTEN 表示已经被占用,最后一列显示被服务 mysqld 占用,查看具体端口号,只要有如图这一行就表示被占用了。

磁盘满了怎么处理

1. df -h 查看是哪个挂在目录满了,常常是根目录/占满
2. 快速定位一下应用日志大小情况,比如 tomcat 日志,应用系统自己的日志等。
3. 如果能直观地看到日志文件过大,则酌情进行删除。有时候删除日志文件之后再 df -h 查看空间依然被占满,继续排查。  
lsof file\_name 查看文件占用进程情况,如果删除的日志正在被某个进程占用,则必须重启或者 kill 掉进程。
4. 如果不能直观地排除出是某个日志多大的原因,就需要看一下指定目录下的文件和子目录大小情况,使用 du 命令。

删除一个目录下的 txt 文件

```
find . -name "*.txt" | xargs rm -rf
```

我还熟悉文本编辑指令。

## 8、请你说一下 vi 里面怎么替换字符串

考察点: vi 命令

参考回答:



vi/vim 中可以使用 :s 命令来替换字符串。该命令有很多种不同细节使用方法，可以实现复杂的功能，记录几种在此，方便以后查询。

: s/vivian/sky/ 替换当前行第一个 vivian 为 sky

: s/vivian/sky/g 替换当前行所有 vivian 为 sky

: n, \$s/vivian/sky/ 替换第 n 行开始到最后一行中每一行的第一个 vivian 为 sky

: n, \$s/vivian/sky/g 替换第 n 行开始到最后一行中每一行所有 vivian 为 sky  
n 为数字，若 n 为 .，表示从当前行开始到最后一行

: %s/vivian/sky/ (等同于 : g/vivian/s//sky/) 替换每一行的第一个 vivian 为 sky

: %s/vivian/sky/g (等同于 : g/vivian/s//sky/g) 替换每一行中所有 vivian 为 sky

## 9、请问 contrab,uptime,du,netstat 这几个指令有什么作用，如何查看磁盘分区状态

考察点: linux

参考回答:

Crontab: 被用来提交和管理用户的需要周期性执行的任务，当安装完成操作系统后，默认会安装此服务工具，并且会自动启动 crond 进程，crond 进程每分钟会定期检查是否有要执行的任务，如果有要执行的任务，则自动执行该任务。

Uptime: 查询服务器已经运行多久

Du: 查看文件和目录磁盘使用的空间情况

Netstat: 显示网络状态，利用 netstat 可以让你得知整个 Linux 系统的网络情况

使用 df 命令可以查看磁盘的适用情况以及文件系统被挂载的位置

## 10、请问如何将文本中的 T 全部替换成 t,将其中的一行复制新的一行出来

考察点: vi

参考回答:

: %s/T/t/g

## 七、前端

### 1、请你说一下前端数据存储方式 (cookies, localStorage, sessionStorage, indexedDB)

考察点: 前端存储

参考回答:

### Cookie

Cookie 最初是在客户端用于存储会话信息的，其要求服务器对任意 HTTP 请求发送 Set-Cookie HTTP 头作为响应的一部分。cookie 以 name 为名称，以 value 为值，名和值在传送时都必须是 URL 编码的。浏览器会存储这样的会话信息，在这之后，通过为每个请求添加 Cookie HTTP 头将信息发送回服务器。

### localStorage

存储方式：

以键值对(Key-Value)的方式存储，永久存储，永不失效，除非手动删除。

### sessionstorage

HTML5 的本地存储 API 中的 localStorage 与 sessionStorage 在使用方法上是相同的，区别在于 sessionStorage 在关闭页面后即被清空，而 localStorage 则会一直保存。

### IndexedDB

索引数据库 (IndexedDB) API (作为 HTML5 的一部分) 对创建具有丰富本地存储数据的数据密集型的离线 HTML5 Web 应用程序很有用。同时它还有助于本地缓存数据，使传统在线 Web 应用程序 (比如移动 Web 应用程序) 能够更快地运行和响应。

## 八、开发工具

### 1、请你说一下 eclipse 的常用操作快捷键

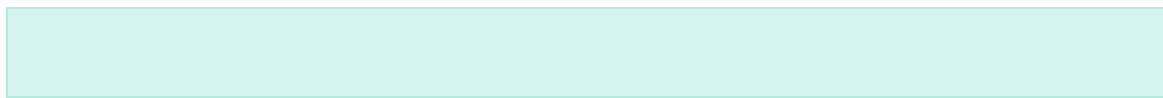
考点：常用工具

参考回答：

快捷键	描述
Ctrl+1	快速修复(最经典的快捷键,就不用多说了,可以解决很多问题,比如 import 类、try catch 包围等)



Ctrl+Shift+F	格式化当前代码
Ctrl+Shift+M	添加类的 import 导入
Ctrl+Shift+O	组织类的 import 导入（既有 Ctrl+Shift+M 的作用，又可以帮你去除没用的导入，很有用）
Ctrl+Y	重做（与撤销 Ctrl+Z 相反）
Alt+ /	内容辅助（帮你省了多少次键盘敲打，太常用了）
Ctrl+D	删除当前行或者多行
Alt+ ↓	当前行和下面一行交互位置（特别实用，可以省去先剪切，再粘贴了）
Alt+ ↑	当前行和上面一行交互位置（同上）
Ctrl+Alt+ ↓	复制当前行到下一行（复制增加）
Ctrl+Alt+ ↑	复制当前行到上一行（复制增加）
Shift+Enter	在当前行的下一行插入空行（这时鼠标可以在当前行的任一位置，不一定是最后）
Ctrl+ /	注释当前行，再按则取消注释



选择	
Alt+Shift + ↑	选择封装元素
Alt+Shift + ←	选择上一个元素
Alt+Shift + →	选择下一个元素
Shift+←	从光标处开始往左选择字符
Shift+→	从光标处开始往右选择字符
Ctrl+Shift t+←	选中光标左边的单词
Ctrl+Shift t+→	选中光标右边的单词



移动	
Ctrl l+←	光标移到左边单词的开头，相当于 vim 的 b



Ctrl+→	光标移到右边单词的末尾，相当于 vim 的 e

--

搜索	
Ctrl+K	参照选中的 Word 快速定位到下一个（如果没有选中 word，则搜索上一次使用搜索的 word）
Ctrl+Shift+K	参照选中的 Word 快速定位到上一个
Ctrl+J	正向增量查找（按下 Ctrl+J 后，你所输入的每个字母编辑器都提供快速匹配定位到某个单词，如果没有，则在状态栏中显示没有找到了，查一个单词时，特别实用，要退出这个模式，按 escape 建）
Ctrl+Shift+J	反向增量查找（和上条相同，只不过是后往前查）
Ctrl+Shift+U	列出所有包含字符串的行
Ctrl+H	打开搜索对话框
Ctrl+G	工作区中的声明
Ctrl+Shift+G	工作区中的引用

--

导航	
Ctrl+Shift+T	搜索类（包括工程和关联的第三 jar 包）
Ctrl+Shift+R	搜索工程中的文件
Ctrl+E	快速显示当前 Editor 的下拉列表（如果当前页面没有显示的用黑体表示）
F4	打开类型层次结构
F3	跳转到声明处
Alt+←	前一个编辑的页面
Alt+→	下一个编辑的页面（当然是针对上面那条来说了）
Ctrl+PageUp/PageDown	在编辑器中，切换已经打开的文件

调试	
F5	单步跳入
F6	单步跳过
F7	单步返回

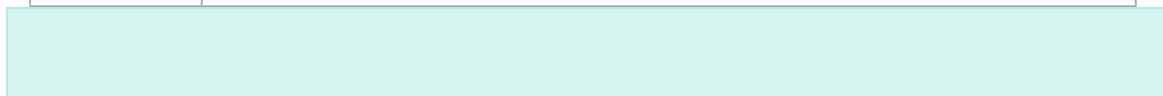
F8	继续
Ctrl+Shift+D	显示变量的值
Ctrl+Shift+B	在当前行设置或者去掉断点
Ctrl+R	运行至行(超好用，可以节省好多的断点)



重构（一般重构的快捷键都是 Alt+Shift 开头的了）	
Alt+Shift+R	重命名方法名、属性或者变量名（是我自己最爱用的一个了，尤其是变量和类的 Rename, 比手工方法能节省很多劳动力）
Alt+Shift+M	把一段函数内的代码抽取成方法（这是重构里面最常用的方法之一了，尤其是对一大堆泥团代码有用）
Alt+Shift+C	修改函数结构（比较实用，有 N 个函数调用了这个方法，修改一次搞定）
Alt+Shift+L	抽取本地变量（可以直接把一些魔法数字和字符串抽取成一个变量，尤其是多处调用的时候）
Alt+Shift+F	把 Class 中的 local 变量变为 field 变量（比较实用的功能）



Alt+Shift t+I	合并变量（可能这样说有点不妥 Inline）
Alt+Shift t+V	移动函数和变量（不怎么常用）
Alt+Shift t+Z	重构的后悔药（Undo）



其他	
Alt+E nter	显示当前选择资源的属性，windows 下的查看文件的属性就是这个快捷键，通常用来查看文件在 windows 中的实际路径
Ctrl+ ↑	文本编辑器 上滚行
Ctrl+ ↓	文本编辑器 下滚行
Ctrl+ M	最大化当前的 Edit 或 View （再按则反之）
Ctrl+ O	快速显示 OutLine（不开 Outline 窗口的同学，这个快捷键是必不可少的）
Ctrl+ T	快速显示当前类的继承结构



Ctrl+W	关闭当前 Editor（windows 下关闭打开的对话框也是这个，还有 qq、旺旺、浏览器等都是）
Ctrl+L	文本编辑器 转至行
F2	显示工具提示描述

## 九、设计模式

### 1、请你说一下常用设计模式；

考点：设计模式

参考回答：

（1）单例模式：保证一个类仅有一个实例，并提供一个访问它的全局访问点，避免一个全局使用的类频繁的创建和销毁，节省系统资源，提高程序效率。

实现方式：

将被实现的类的构造方法设计成 private 的。

添加此类引用的静态成员变量，并为其实例化。

在被实现的类中提供公共的 Create Instance 函数，返回实例化的此类，就是 2 中的静态成员变量。

单例模式只允许创建一个对象，因此节省内存，加快对象访问速度，适用场景：

需要频繁实例化然后销毁的对象。

创建对象时耗时过多或者耗资源过多，但又经常用到的对象。

有状态的工具类对象。

频繁访问数据库或文件的对象。

场景举例：

每台计算机有若干个打印机，但只能有一个 PrinterSpooler，以避免两个打印作业同时输出到打印机；

Windows 的 TaskManager（任务管理器），不能打开两个 windows task manager；

Windows 的 Recycle Bin（回收站），在整个系统运行过程中，回收站一直维护着仅有的一个实例；

网站的计数器，一般也是采用单例模式实现，否则难以同步；

（2）策略模式：策略模式是把一个类中经常改变或者将来可能改变的部分提取出来作为一个接口，然后在类中包含这个对象的实例，这样类的实例在运行时就可以随意调用实现了这个接口的类的行为。

实现方式：

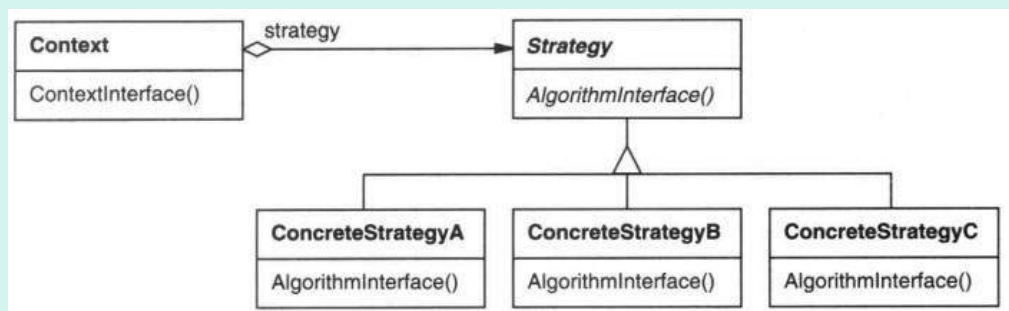
提供公共接口或抽象类，定义需要使用的策略方法。（策略抽象类）

多个实现的策略抽象类的实现类。（策略实现类）

环境类，对多个实现类的封装，提供接口类型的成员量，可以在客户端中切换。

客户端调用环境类进行不同策略的切换。

类图：



**Strategy**：策略接口，用来约束一系列具体的策略算法。**Context** 使用这个接口来调用具体的策略，实现定义的策略。

**ConcreteStrategy**：具体的策略实现，也就是具体的算法实现。

**Context**：上下文，负责与具体的策略交互，通常上下文会持有一个真正的策略实现。

适用场景：

如果在一个系统里面有许多类，它们之间的区别仅在于它们的行为，那么使用策略模式可以动态地让一个对象在许多行为中选择一种行为。

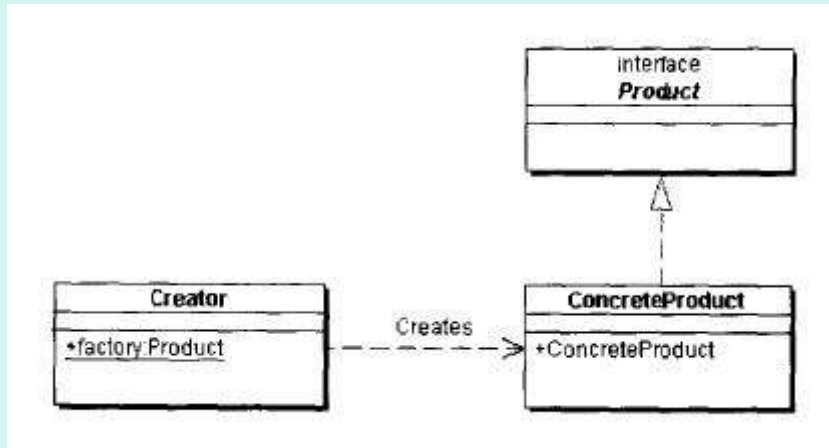
一个系统需要动态地在几种算法中选择一种。

一个类定义了多种行为，并且这些行为在这个类的操作中以多个条件语句的形式出现。将相关的条件分支移入它们各自的 **Strategy** 类中以代替这些条件语句。



(3) 简单工厂模式：定义一个用于创建对象的接口或抽象类，让子类决定实例化哪一个类，工厂方法使一个类的实例化延迟到其子类。

类图：



实现方式：

抽象产品类（Product），是所创建的所有对象的父类，负责描述所有实例所共有的公共接口

多个具体的产品类（Concrete Product），具体产品角色是创建目标，所有创建的对象都充当这个角色的某个具体类的实例。

工厂类（Creator），负责实现创建所有实例的内部逻辑

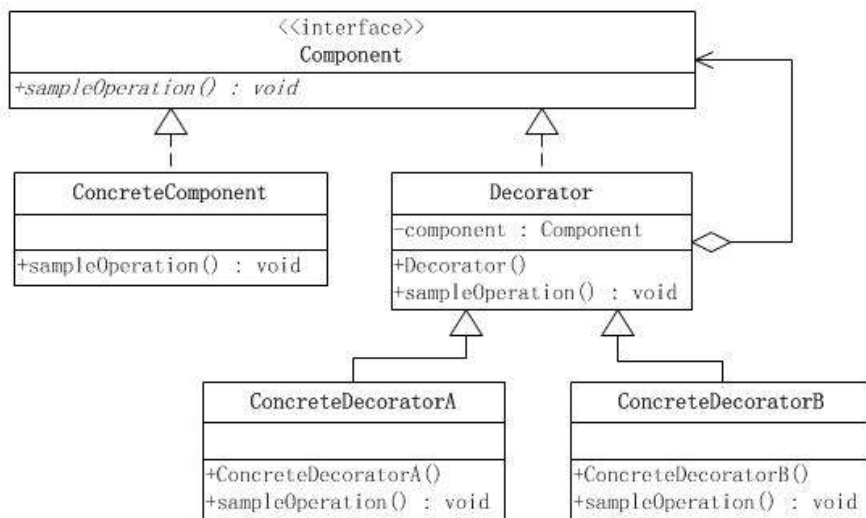
适用场景：

在任何需要生成复杂对象的地方，都可以使用工厂方法模式。

当需要系统有比较好的扩展性时，可以考虑工厂模式，不同的产品用不同的实现工厂来组装。

(4) 装饰模式：允许向一个现有的对象添加新的功能，同时又不改变其结构，以在不使用创造更多子类的情况下，将对象的功能加以扩展。

类图：



实现方式:

抽象构件 (Component) 角色: 给出一个抽象接口, 以规范准备接收附加责任的对象。

具体构件 (ConcreteComponent) 角色: 定义一个将要接收附加责任的类。

装饰 (Decorator) 角色: 持有一个构件 (Component) 对象的实例, 并定义一个与抽象构件接口一致的接口。

具体装饰 (ConcreteDecorator) 角色: 负责给构件对象 “贴上” 附加的责任。

适用场景:

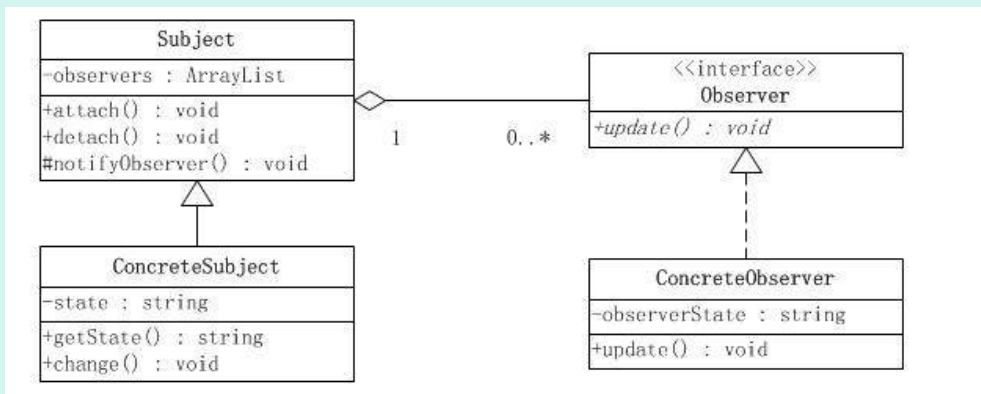
扩展一个类的功能。

动态增加功能, 动态撤销。

(5) 观察者模式: 对象间的一种一对多的依赖关系, 当一个对象的状态发生改变时, 所有依赖于它的对象都得到通知并被自动更新。

类图:





**抽象主题 (Subject) 角色：**把所有对观察者对象的引用保存在一个集合中，每个抽象主题角色都可以有任意数量的观察者。抽象主题提供一个接口，可以增加和删除观察者角色。一般用一个抽象类和接口来实现。

**抽象观察者 (Observer) 角色：**为所有具体的观察者定义一个接口，在得到主题的通知时更新自己。

**具体主题 (ConcreteSubject) 角色：**在具体主题内部状态改变时，给所有登记过的观察者发出通知。具体主题角色通常用一个子类实现。

**具体观察者 (ConcreteObserver) 角色：**该角色实现抽象观察者角色所要求的更新接口，以便使本身的状态与主题的状态相协调。通常用一个子类实现。如果需要，具体观察者角色可以保存一个指向具体主题角色的引用。

适用场景：

当一个抽象模型有两个方面，其中一个方面依赖于另一方面。将这二者封装在独立的对象中，以使它们可以各自独立地改变和复用。

当对一个对象的改变需要同时改变其它对象，而不知道具体有多少对象有待改变。

当一个对象必须通知其它对象，而它又不能假定其它对象是谁。换言之，你不希望这些对象是紧密耦合的。

## 2、请你手写一下单例模式

考点：设计模式

参考回答：

1、饿汉模式

```
public class Singleton {  
  
    private static Singleton instance = new Singleton();  
}
```



```
private Singleton () {  
  
}  
  
public static Singleton getInstance() {  
  
    return instance;  
  
}  
  
}
```

这种方式在类加载时就完成了初始化，所以类加载较慢，但获取对象的速度快。这种方式基于类加载机制避免了多线程的同步问题，但是也不能确定有其他的方式(或者其他的静态方法)导致类装载，这时候初始化 instance 显然没有达到懒加载的效果。

## 2、懒汉模式（线程不安全）

```
public class Singleton {  
  
    private static Singleton instance;  
  
    private Singleton () {  
  
    }  
  
    public static Singleton getInstance() {  
  
        if (instance == null) {  
  
            instance = new Singleton();  
  
        }  
  
        return instance;  
  
    }  
  
}
```

懒汉模式申明了一个静态对象，在用户第一次调用时初始化，虽然节约了资源，但第一次加载时需要实例化，反映稍慢一些，而且在多线程不能正常工作。

## 3、懒汉模式（线程安全）

```
public class Singleton {  
  
    private static Singleton instance;  
  
    private Singleton () {  
  
    }  
  
}
```



```
}

public static synchronized Singleton getInstance() {

    if (instance == null) {

        instance = new Singleton();

    }

    return instance;

}

}
```

这种写法能够在多线程中很好的工作，但是每次调用 `getInstance` 方法时都需要进行同步，造成不必要的同步开销，而且大部分时候我们是用不到同步的，所以不建议用这种模式。

#### 4、双重检查模式（DCL）

```
public class Singleton {

    private volatile static Singleton singleton;

    private Singleton () {

    }

    public static Singleton getInstance() {

        if (instance== null) {

            synchronized (Singleton.class) {

                if (instance== null) {

                    instance= new Singleton();

                }

            }

        }

        return singleton;

    }

}
```



这种写法在 `getSingleton` 方法中对 `singleton` 进行了两次判空，第一次是为了不必要的同步，第二次是在 `singleton` 等于 `null` 的情况下才创建实例。

### 3、请问设计模式是什么？

考察点：设计模式

参考回答：

设计模式（Design pattern）代表了最佳的实践，通常被有经验的面向对象的软件开发人员所采用。设计模式是软件开发人员在软件开发过程中面临的一般问题的解决方案。这些解决方案是众多软件开发人员经过相当长的一段时间的试验和错误总结出来的。

设计模式是一套被反复使用的、多数人知晓的、经过分类编目的、代码设计经验的总结。使用设计模式是为了重用代码、让代码更容易被他人理解、保证代码可靠性。毫无疑问，设计模式于己于他人于系统都是多赢的，设计模式使代码编制真正工程化，设计模式是软件工程的基石，如同大厦的一块块砖石一样。项目中合理地运用设计模式可以完美地解决很多问题，每种模式在现实中都有相应的原理来与之对应，每种模式都描述了一个在我们周围不断重复发生的问题，以及该问题的核心解决方案，这也是设计模式能被广泛应用的原因。

### 4、请你手写一下单例模式代码

考察点：设计模式

参考回答：

懒汉式单例模式：延迟实例化，但节省空间

```
package com.sxh.singleton;

public class Singleton {

    /*
     * volatile 关键字确保：当 uniqueInstance 变量被初始化为 Singleton 实例时，多个线程能正确的处理 uniqueInstance 变量
     * 分析：volatile 修饰的成员变量，在每次被线程访问时，都强制性的从共享内存重读该成员的值；
     * 当值发生变化是，强制线程将变化值写入共享内存，任何时候不同线程总是看到你某个成员变量的同一个值
     */

    private volatile static Singleton uniqueInstance;//利用一个静态变量来记录 Singleton 类的唯一实例
```



```
//其他有用的单件类的数据

private Singleton() {} //类外无法访问

public static Singleton getInstance() {

    /*

    * 使用”双重检查加锁“，在 getInstance 中减少使用同步

    * 首先检查是否实例已经创建了，如果尚未创建，才进行同步；只有第一次访问
    getInstance 会同步

    */

    if(uniqueInstance==null){ //确保只有一个实例

        synchronized (Singleton.class) { //多线程的情况不会出现问题，线程同步问题

            if(uniqueInstance==null){

                uniqueInstance=new Singleton(); //如果我们不需要这个实例，则永远
                不会产生

            }

        }

    }

    return uniqueInstance;

}

//其他有用的单件类的方法，单件类也可以是一般的类，具有一般的数据和方法

}

饿汉式单例模式：急切的创建实例，而不用延迟实例化

package com.sxh.singleton;

public class Singleton {
```



```
private volatile static Singleton uniqueInstance=new Singleton();
```

```
//其他有用的单件类的数据
```

```
private Singleton() {} //类外无法访问
```

```
public static Singleton getInstance() {
```

```
    return uniqueInstance;
```

```
}
```

```
//其他有用的单件类的方法，单件类也可以是一般的类，具有一般的数据和方法
```

```
}
```

IoDH 实现单例模式

```
package com.sxh.singleton;
```

```
public class Singleton {
```

```
private Singleton() {} //类外无法访问
```

```
private static class HolderClass{ //静态内部类
```

```
    private static final Singleton uniqueinstance=new Singleton();
```

```
}
```

```
public static Singleton getInstance() {
```

```
    return HolderClass.uniqueinstance;
```

```
}
```

```
//其他有用的单件类的方法，单件类也可以是一般的类，具有一般的数据和方法
```

```
}
```

## 十、场景题

### 1、上亿数量的链接，如何找出点击量排名前十的链接？

考点：大数据处理

参考回答：（这题不太会）

分两种情况：可一次读入内存，不可一次读入。

思路：trie 树+堆，数据库索引，划分子集分别统计，hash，分布式计算，近似统计，外排序。

遍历文件，对每个 url 求取  $\text{hash}(\text{url})\%1000$ ，然后根据所取得的值将 url 分别存储到 1000 个小文件（记为  $a_0, a_1, \dots, a_{999}$ ）中。

### 2、请你说几个海量数据存储常见问题以及如何解答

考点：大数据存储

参考回答：

例子 1：给你 A,B 两个文件，各存放 50 亿条 URL，每条 URL 占用 64 字节，内存限制是 4G，让你找出 A,B 文件共同的 URL。

思路：可以估计每个文件安的大小为  $5G \times 64 = 320G$ ，远远大于内存限制的 4G。所以不可能将其完全加载到内存中处理。考虑采取分而治之的方法。

- 分而治之/hash 映射：遍历文件 a，对每个 url 求取，然后根据所取得的值将 url 分别存储到 1000 个小文件（记为）中。这样每个小文件的大约为 300M。遍历文件 b，采取和 a 相同的方式将 url 分别存储到 1000 小文件中（记为）。这样处理后，所有可能相同的 url 都在对应的小文件（）中，不对应的小文件不可能有相同的 url。然后我们只要求出 1000 对小文件中相同的 url 即可。

- hash 统计：求每对小文件中相同的 url 时，可以把其中一个小文件的 url 存储到 hash\_set 中。然后遍历另一个小文件的每个 url，看其是否在刚才构建的 hash\_set 中，如果是，那么就是共同的 url，存到文件里面就可以了。

例子 2：有 10 个文件，每个文件 1G，每个文件的每一行都存放的是用户的 query，每个文件的 query 都可能重复。要你按照 query 的频度排序

思路：



- hash 映射：顺序读取 10 个文件，按照  $\text{hash}(\text{query})\%10$  的结果将 query 写入到另外 10 个文件（记为）中。这样新生成的文件每个的大小大约也 1G（假设 hash 函数是随机的）。

- hash 统计：找一台内存在 2G 左右的机器，依次对用  $\text{hash\_map}(\text{query}, \text{query\_count})$  来统计每个 query 出现的次数。注： $\text{hash\_map}(\text{query}, \text{query\_count})$  是用来统计每个 query 的出现次数，不是存储他们的值，出现一次，则  $\text{count}+1$ 。

- 堆/快速/归并排序：利用快速/堆/归并排序按照出现次数进行排序。将排序好的 query 和对应的 query\_count 输出到文件中。这样得到了 10 个排好序的文件（记为）。对这 10 个文件进行归并排序（内排序与外排序相结合）。

例子 3：有一个 1G 大小的一个文件，里面每一行是一个词，词的大小不超过 16 个字节，内存限制大小是 1M。返回频数最高的 100 个词

思路：

- 分而治之/hash 映射：顺序读文件中，对于每个词  $x$ ，取  $\text{hash}(x)\%5000$ ，然后按照该值存到 5000 个小文件（记为  $x_0, x_1, \dots, x_{4999}$ ）中。这样每个文件大概是 200k 左右。如果其中的有的文件超过了 1M 大小，还可以按照类似的方法继续往下分，直到分解得到的小文件的大小都不超过 1M。

- hash 统计：对每个小文件，采用 trie 树/hash\_map 等统计每个文件中出现的词以及相应的频率。

- 堆/归并排序：取出出现频率最大的 100 个词（可以用含 100 个结点的最小堆），并把 100 个词及相应的频率存入文件，这样又得到了 5000 个文件。最后就是把这 5000 个文件进行归并（类似于归并排序）的过程了。

例子 4：海量日志数据，提取出某日访问百度次数最多的那个 IP。

思路：

- 分而治之/hash 映射：针对数据太大，内存受限，只能是：把大文件化成（取模映射）小文件，即 16 字方针：大而化小，各个击破，缩小规模，逐个解决

- hash 统计：当大文件转化了小文件，那么我们便可以采用常规的 Hashmap(ip, value) 来进行频率统计。

- 堆/快速排序：统计完了之后，便进行排序（可采取堆排序），得到次数最多的 IP。

### 3、请顾问有两个文件，如何将这两个文件按行依次交叉存入第三个文件中

考点：文件处理

参考回答：





linux 中假设当前目录下的文件都以 wav. trn 为后缀，每个文件里的内容为一行，将它们依次添加到文件 Z 的末尾

脚本如下：

```
#!/usr/bin/bash

for filename in `find ./*.wav.trn|sort -u`
do

echo `sed -n 1p $filename` >> Z

done
```

#### 4、请你回答一下：统计文本中出现次数前十的单词，文件很大，不能一次性读入内存？

考察点：编程

参考回答：

Top K 算法详解

应用场景：

搜索引擎会通过日志文件把用户每次检索使用的所有检索串都记录下来，每个查询串的长度为 1-255 字节。

假设目前有一千万个记录（这些查询串的重复度比较高，虽然总数是 1 千万，但如果除去重复后，不超过 3 百万个。一个查询串的重复度越高，说明查询它的用户越多，也就是越热门。），请你统计最热门的 10 个查询串，要求使用的内存不能超过 1G。

必备知识：

什么是哈希表？

哈希表（Hash table，也叫散列表），是根据关键码值(Key value)而直接进行访问的数据结构。

也就是说，它通过把关键码值映射到表中一个位置来访问记录，以加快查找的速度。这个映射函数叫做散列函数，存放记录的数组叫做散列表。

哈希表的做法其实很简单，就是把 Key 通过一个固定的算法函数既所谓的哈希函数转换成一个整型数字，然后就将该数字对数组长度进行取余，取余结果就当作数组的下标，将 value 存储在以该数字为下标的数组空间里。



而当使用哈希表进行查询的时候，就是再次使用哈希函数将 key 转换为对应的数组下标，并定位到该空间获取 value，如此一来，就可以充分利用到数组的定位性能进行数据定位。

问题解析：

要统计最热门查询，首先就是要统计每个 Query 出现的次数，然后根据统计结果，找出 Top 10。所以我们可以基于这个思路分两步来设计该算法。

即，此问题的解决分为以下两个步骤：

第一步：Query 统计

(统计出每个 Query 出现的次数)

Query 统计有以下两个方法，可供选择：

### 1、直接排序法

(经常在日志文件中统计时，使用 `cat file|format key|sort | uniq -c | sort -nr | head -n 10`，就是这种方法)

首先我们最先想到的算法就是排序了，首先对这个日志里面的所有 Query 都进行排序，然后再遍历排好序的 Query，统计每个 Query 出现的次数了。

但是题目中有明确要求，那就是内存不能超过 1G，一千万条记录，每条记录是 255Byte，很显然要占据 2.375G 内存，这个条件就不满足要求了。

让我们回忆一下数据结构课程上的内容，当数据量比较大而且内存无法装下的时候，我们可以采用外排序的方法来进行排序，这里我们可以采用归并排序，因为归并排序有一个比较好的时间复杂度  $O(N\lg N)$ 。

排完序之后我们再对已经有序的 Query 文件进行遍历，统计每个 Query 出现的次数，再次写入文件中。

综合分析一下，排序的时间复杂度是  $O(N\lg N)$ ，而遍历的时间复杂度是  $O(N)$ ，因此该算法的总体时间复杂度就是  $O(N+N\lg N)=O(N\lg N)$ 。

### 2、Hash Table 法

(这种方法统计字符串出现的次数非常好)

在第 1 个方法中，我们采用了排序的办法来统计每个 Query 出现的次数，时间复杂度是  $N\lg N$ ，那么能不能有更好的方法来存储，而时间复杂度更低呢？

题目中说明了，虽然有一千万个 Query，但是由于重复度比较高，因此事实上只有 300 万的 Query，每个 Query 255Byte，因此我们可以考虑把他们放进内存中去，而现在只是需要一个合适的结构，在这里，Hash Table 绝对是我们优先的选择，因为 Hash Table 的查询速度非常的快，几乎是  $O(1)$  的时间复杂度。

那么，我们的算法就有了：



维护一个 Key 为 Query 字符串, Value 为该 Query 出现次数的 HashTable, 每次读取一个 Query, 如果该字符串不在 Table 中, 那么加入该字符串, 并且将 Value 值设为 1; 如果该字符串在 Table 中, 那么将该字符串的计数加一即可。最终我们在  $O(N)$  的时间复杂度内完成了对该海量数据的处理。

本方法相比算法 1: 在时间复杂度上提高了一个数量级, 为  $O(N)$ , 但不仅仅是时间复杂度上的优化, 该方法只需要 IO 数据文件一次, 而算法 1 的 IO 次数较多的, 因此该算法 2 比算法 1 在工程上有更好的可操作性。

第二步: 找出 Top 10

(找出出现次数最多的 10 个)

算法一: 普通排序

(我们只用找出 top10, 所以全部排序有冗余)

我想对于排序算法大家都已经不陌生了, 这里不在赘述, 我们注意的是排序算法的时间复杂度是  $N\log N$ , 在本题目中, 三百万条记录, 用 1G 内存是可以存下的。

算法二: 部分排序

题目要求是求出 Top 10, 因此我们没有必要对所有的 Query 都进行排序, 我们只需要维护一个 10 个大小的数组, 初始化放入 10 个 Query, 按照每个 Query 的统计次数由大到小排序, 然后遍历这 300 万条记录, 每读一条记录就和数组最后一个 Query 对比, 如果小于这个 Query, 那么继续遍历, 否则, 将数组中最后一条数据淘汰(还是要放在合适的位置, 保持有序), 加入当前的 Query。最后当所有的数据都遍历完毕之后, 那么这个数组中的 10 个 Query 便是我们要找的 Top10 了。

不难分析出, 这样, 算法的最坏时间复杂度是  $N*K$ , 其中 K 是指 top 多少。

算法三: 堆

在算法二中, 我们已经将时间复杂度由  $N\log N$  优化到  $N*K$ , 不得不说这是一个比较大的改进了, 可是有没有更好的办法呢?

分析一下, 在算法二中, 每次比较完成之后, 需要的操作复杂度都是 K, 因为要把元素插入到一个线性表之中, 而且采用的是顺序比较。这里我们注意一下, 该数组是有序的, 一次我们每次查找的时候可以采用二分的方法查找, 这样操作的复杂度就降到了  $\log K$ , 可是, 随之而来的问题就是数据移动, 因为移动数据次数增多了。不过, 这个算法还是比算法二有了改进。

基于以上的分析, 我们想想, 有没有一种既能快速查找, 又能快速移动元素的数据结构呢?

回答是肯定的, 那就是堆。

借助堆结构, 我们可以在  $\log$  量级的时间内查找和调整/移动。因此到这里, 我们的算法可以改进为这样, 维护一个 K(该题目中是 10)大小的小根堆, 然后遍历 300 万的 Query, 分别和根元素进行对比。

思想与上述算法二一致, 只是在算法三, 我们采用了最小堆这种数据结构代替数组, 把查找目标元素的时间复杂度有  $O(K)$  降到了  $O(\log K)$ 。



那么这样，采用堆数据结构，算法三，最终的时间复杂度就降到了  $N \log K$ ，和算法二相比，又有了比较大的改进。

总结：

至此，算法就完全结束了，经过上述第一步、先用 Hash 表统计每个 Query 出现的次数， $O(N)$ ；然后第二步、采用堆数据结构找出 Top 10， $N \cdot O(\log K)$ 。所以，我们最终的时间复杂度是： $O(N) + N \cdot O(\log K)$ 。（N 为 1000 万，N' 为 300 万）。 参考来自 CSDN 用户立喆

## 5、请你回答一下 QQ 怎么实现在线大文件的传输

考察点：计算机网络

参考回答：网上搜索了一下，比较多的一种说法是通过“md5('文件内容')”生成“唯一标识符”的方法。

## 6、请你回答一下新增一个 dota2 英雄要怎么设计，应该考虑哪些

参考回答：

略

## 7、请你说一下你用过哪些电商 app,并比较他们的好坏

参考回答：

笔者用过淘宝，京东、唯品会、网易考拉、亚马逊

首先淘宝，用的最多，的确买日用品很方便，东西齐全，但是加载首页很慢，不够流畅，此外依旧是假货横行，应当加大打假力度，

京东，只买过 U 盘，但是退货后钱款没到账，找客服两次未解决，不提了，后面就卸了，没什么印象了

唯品会：排版不行，购物车只能保持十五分钟这个笔者持反对意见

网易考拉：货少但相对淘宝来说可信度比较高，当然价格也比较高，经常缺货

亚马逊：除了运费贵英文看的费力没毛病

仅供参考



8、有一款还没有进入杭州市场的铅笔，怎么估计他的月销售量

参考回答：

略

9、请你说一下 ping 百度途径的主机是如何展示出来的

参考回答：

略

10、1-100,100 个数，取走一个，怎么快速知道我取走了哪个数。

参考回答：

略

## 十一、大数据

### 1、Hadoop

1、请你说一下分布式和集群的概念。

考点：分布式

参考回答：

分布式：是指将不同的业务分布在不同的地方，

集群：是指将几台服务器集中在一起，实现同一业务。

分布式中的每一个节点，都可以做集群，而集群并不一定就是分布式的。集群有组织性，一台服务器垮了，其它的服务器可以顶上来，而分布式的每一个节点，都完成不同的业务，一个节点垮了，哪个业务就不可访问了。

2、Hadoop 你也有了解的，那你有了解他的备份机制吧？请问怎么做到数据保持一致？

考察点：hadoop

参考回答：

Hadoop 备份机制：该方案利用 Hadoop 自身的 Failover 措施（通过配置 `dfs.name.dir`），NameNode 可以将元数据信息保存到多个目录。通常的做法，选择一个本地目录、一个远程目录（通过 NFS 进行共享），当 NameNode 发生故障时，可以启动备用机器的 NameNode，加载远程目录中的元数据信息，提供服务。

Hadoop 中有个重要的工具就是 HDFS 分布式文件系统，那么在这种分布式环境下的数据一致性是如何保证呢？

HDFS 中，存储的文件将会被分成若干的大小一致的 block（最后一个 block 的大小可能较小）分布式地存储在不同的机器上，那么就必须要有一个角色来对这些数据进行管理，也就是 NameNode 节点，而存储这些 block 的结点我们称为 DataNode，NameNode 是用来管理这些元数据的。下面讲一个例子，在客户端上传文件时，NameNode 如何保证数据的一致性。客户端上传文件时，NameNode 首先往 `edits log` 文件中记录元数据的操作日志。与此同时，NameNode 将会在磁盘做一份持久化处理（`fsimage` 文件）：他跟内存中的数据是对应的，如何保证和内存中的数据的一致性呢？在 `edits logs` 满之前对内存和 `fsimage` 的数据做同步（实际上只需要合并 `edits logs` 和 `fsimage` 上的数据即可，然后 `edits logs` 上的数据即可清除）而当 `edits logs` 满之后，文件的上传不能中断，所以将会往一个新的文件 `edits.new` 上写数据，而老的 `edits logs` 的合并操作将由 `secondNameNode` 来完成，即所谓的 `checkpoint` 操作。那么什么时候 `checkpoint` 呢？一般由两种限制，一个是 `edits logs` 的大小限制，即 `fs.checkpoint.size` 配置，一个是指定时间，即 `fs.checkpoint.period` 配置 当然根据规定，大小的限制是优先的，规定 `edits` 文件一旦超过阈值，则不管是否达到最大时间间隔，都会强制 `checkpoint`。

## 十二、计算机基础

### 1、计算机网络

#### 1、请你说一下 HTTP 的报文段是什么样的？

考点：网络

参考回答：

HTTP请求报文							
请求方法	空格	URL	空格	协议版本	回车符	换行符	请求行
头部字段名	:	值	回车符	换行符	请求头部		
		.....					
头部字段名	:	值	回车符	换行符			
回车符	换行符						
请求正文							

#### 1、请求方法

GET：请求获取 Request——URL 所标识的资源



POST：在 Request——URL 所标识的资源后附加资源

HEAD：请求获取由 Request——URL 所标识的资源的响应消息报头

PUT：请求服务器存储一个资源，由 Request——URL 作为其标识

DELETE：请求服务器删除由 Request——URL 所标识的资源

TRACE：请求服务器回送收到的请求信息（用于测试和诊断）

CONNECT：保留

OPTIONS：请求查询服务器性能

## 2、URL

URI 全名为 Uniform Resource Identifier（统一资源标识），用来唯一的标识一个资源，是一个通用的概念，URI 由两个主要的子集 URL 和 URN 组成。URL 全名为 Uniform Resource Locator（统一资源定位），通过描述资源的位置来标识资源。URN 全名为 Uniform Resource Name（统一资源命名），通过资源的名字来标识资源，与其所处的位置无关，这样即使资源的位置发生变动，其 URN 也不会变化。

## 3、协议版本

格式为 HTTP/主版本号.次版本号，常用为：HTTP/1.1 HTTP/1.0

## 4、请求头部

Host：接受请求的服务器地址，可以是 IP 或者是域名

User-Agent：发送请求的应用名称

Connection：指定与连接相关的属性，例如（Keep\_Alive，长连接）

Accept-Charset：通知服务器端可以发送的编码格式

Accept-Encoding：通知服务器端可以发送的数据压缩格式

Accept-Language：通知服务器端可以发送的语言

HTTP响应报文							
协议版本	空格	状态码	空格	状态码描述	回车符	换行符	状态行
头部字段名	:	值	回车符	换行符	响应头部		
				.....			
头部字段名	:	值	回车符	换行符			
回车符	换行符						
响应正文							

## 1、协议版本，同请求报文



2、状态码，100~199 表示请求已收到继续处理，200~299 表示成功，300~399 表示资源重定向，400~499 表示客户端请求出错，500~599 表示服务器端出错

200：响应成功

302：跳转，重定向

400：客户端有语法错误

403：服务器拒绝提供服务

404：请求资源不存在

500：服务器内部错误

3、响应头部

Server：服务器应用程序的名称和版本

Content-Type：响应正文的类型

Content-Length：响应正文的长度

Content-Charset：响应正文所使用的编码

Content-Encoding：响应正文使用的数据压缩格式

Content-Language：响应正文使用的语言

## 2、请你回答一下 HTTP 用的什么连接？

考点：网络

参考回答：

在 HTTP/1.0 中，默认使用的是短连接。也就是说，浏览器和服务器每进行一次 HTTP 操作，就建立一次连接，但任务结束就中断连接。如果客户端浏览器访问的某个 HTML 或其他类型的 Web 页中包含有其他的 Web 资源，如 JavaScript 文件、图像文件、CSS 文件等；当浏览器每遇到这样一个 Web 资源，就会建立一个 HTTP 会话。

但从 HTTP/1.1 起，默认使用长连接，用以保持连接特性。使用长连接的 HTTP 协议，会在响应头有加入这行代码：Connection:keep-alive

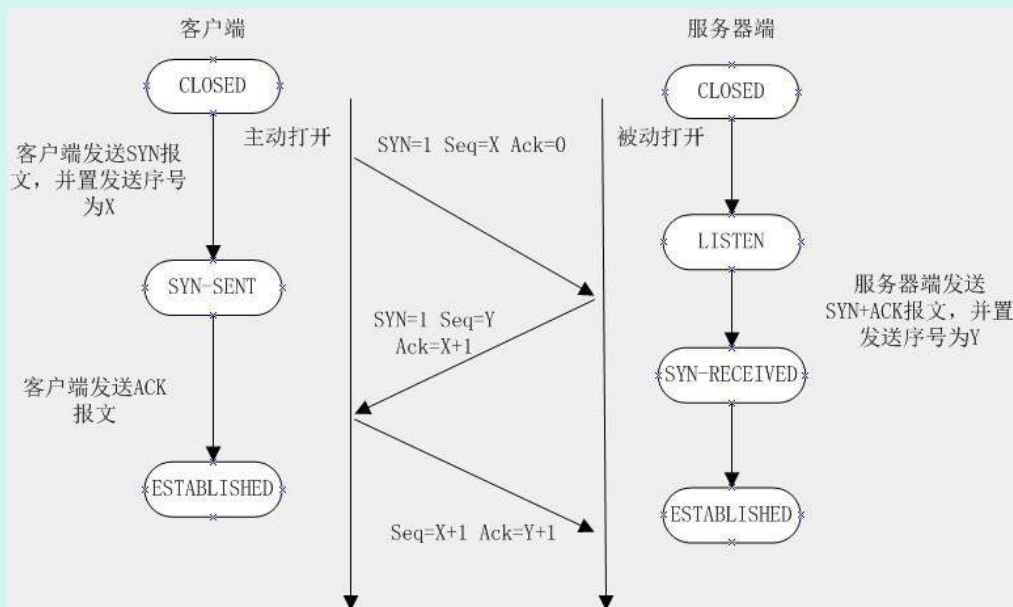
在使用长连接的情况下，当一个网页打开完成后，客户端和服务端之间用于传输 HTTP 数据的 TCP 连接不会关闭，如果客户端再次访问这个服务器上的网页，会继续使用这一条已经建立的连接。Keep-Alive 不会永久保持连接，它有一个保持时间，可以在不同的服务器软件（如 Apache）中设定这个时间。实现长连接要客户端和服务端都支持长连接。



### 3、请你说一说 TCP 的三次握手？

考点：网络

参考回答：



第一次握手：建立连接时，客户端发送 syn 包（syn=j）到服务器，并进入 SYN\_SENT 状态，等待服务器确认；SYN：同步序列编号（Synchronize Sequence Numbers）。

第二次握手：服务器收到 syn 包，必须确认客户的 SYN（ack=j+1），同时自己也发送一个 SYN 包（syn=k），即 SYN+ACK 包，此时服务器进入 SYN\_RECV 状态；

第三次握手：客户端收到服务器的 SYN+ACK 包，向服务器发送确认包 ACK(ack=k+1)，此包发送完毕，客户端和服务器进入 ESTABLISHED（TCP 连接成功）状态，完成三次握手。

### 4、请你说一下在浏览器中输入一个网址它的运行过程是怎样的？

考点：网络及协议

参考回答：

1、查询 DNS，获取域名对应的 IP。

（1）检查浏览器缓存、检查本地 hosts 文件是否有这个网址的映射，如果有，就调用这个 IP 地址映射，解析完成。

（2）如果没有，则查找本地 DNS 解析器缓存是否有这个网址的映射，如果有，返回映射，解析完成。



(3) 如果没有，则查找填写或分配的首选 DNS 服务器，称为本地 DNS 服务器。服务器接收到查询时：

如果要查询的域名包含在本地配置区域资源中，返回解析结果，查询结束，此解析具有权威性。

如果要查询的域名不由本地 DNS 服务器区域解析，但服务器缓存了此网址的映射关系，返回解析结果，查询结束，此解析不具有权威性。

(4) 如果本地 DNS 服务器也失效：

如果未采用转发模式（迭代），本地 DNS 就把请求发至 13 台根 DNS，根 DNS 服务器收到请求后，会判断这个域名（如 .com）是谁来授权管理，并返回一个负责该顶级域名服务器的 IP，本地 DNS 服务器收到顶级域名服务器 IP 信息后，继续向该顶级域名服务器 IP 发送请求，该服务器如果无法解析，则会找到负责这个域名的下一级 DNS 服务器（如 <http://baidu.com>）的 IP 给本地 DNS 服务器，循环往复直至查询到映射，将解析结果返回本地 DNS 服务器，再由本地 DNS 服务器返回解析结果，查询完成。

如果采用转发模式（递归），则此 DNS 服务器就会把请求转发至上一级 DNS 服务器，如果上一级 DNS 服务器不能解析，则继续向上请求。最终将解析结果依次返回本地 DNS 服务器，本地 DNS 服务器再返回给客户机，查询完成。

2、得到目标服务器的 IP 地址及端口号（http 80 端口，https 443 端口），会调用系统库函数 socket，请求一个 TCP 流套接字。客户端向服务器发送 HTTP 请求报文：

(1) 应用层：客户端发送 HTTP 请求报文。

(2) 传输层：（加入源端口、目的端口）建立连接。实际发送数据之前，三次握手客户端和服务器建立起一个 TCP 连接。

(3) 网络层：（加入 IP 头）路由寻址。

(4) 数据链路层：（加入 frame 头）传输数据。

(5) 物理层：物理传输 bit。

3、服务器端经过物理层→数据链路层→网络层→传输层→应用层，解析请求报文，发送 HTTP 响应报文。

4、关闭连接，TCP 四次挥手。

5、客户端解析 HTTP 响应报文，浏览器开始显示 HTML

## 5、请你说一说 http rest

考点：协议

参考回答：

REST (Representational State Transfer) 一种轻量级的 Web Service 架构。可以完全通过 HTTP 协议实现。其实现和操作比 SOAP 和 XML-RPC 更为简洁，还可以利用缓存 Cache 来提高响应速度，性能、效率和易用性上都优于 SOAP 协议。REST 架构对资源的操作包括获取、创建、修改和删除资源的操作对应 HTTP 协议提供的 GET、POST、PUT 和 DELETE 方法。REST 提供了一组架构约束，当作为一个整体来应用时，强调组件交互的可伸缩性、接口的通用性、组件的独立部署、以及用来减少交互延迟、增强安全性、封装遗留系统的中间组件。

REST 架构约束：

客户-服务器 (Client-Server)，提供服务的服务器和使用服务的客户需要被隔离对待，客户和服务器之间通过一个统一的接口来互相通讯。

无状态 (Stateless)，服务端并不会保存有关客户的任何状态，客户端自身负责用户状态的维持，并在每次发送请求时都需要提供足够的信息。

可缓存 (Cachable)，REST 系统需要能够恰当地缓存请求，以尽量减少服务端和客户端之间的信息传输，以提高性能。

分层系统 (Layered System)，服务器和客户之间的通信必须被这样标准化：允许服务器和客户之间的中间层 (Proxy: 代理, 网关等) 可以代替服务器对客户的请求进行回应，而且这些对客户来说不需要特别支持。

统一接口 (Uniform Interface)，客户和服务器之间通信的方法必须是统一化的。

## 6、请你说一说 http 请求报文

考点：协议

参考回答：

http 请求报文：

HTTP请求报文							
请求方法	空格	URL	空格	协议版本	回车符	换行符	请求行
头部字段名	:	值	回车符	换行符	请求头部		
.....							
头部字段名	:	值	回车符	换行符			
回车符	换行符						
请求正文							

1、请求方法

GET：请求获取 Request——URL 所标识的资源



POST：在 Request——URL 所标识的资源后附加资源

HEAD：请求获取由 Request——URL 所标识的资源的响应消息报头

PUT：请求服务器存储一个资源，由 Request——URL 作为其标识

DELETE：请求服务器删除由 Request——URL 所标识的资源

TRACE：请求服务器回送收到的请求信息（用于测试和诊断）

CONNECT：保留

OPTIONS：请求查询服务器性能

## 2、URL

URI 全名为 Uniform Resource Identifier（统一资源标识），用来唯一的标识一个资源，是一个通用的概念，URI 由两个主要的子集 URL 和 URN 组成。URL 全名为 Uniform Resource Locator（统一资源定位），通过描述资源的位置来标识资源。URN 全名为 Uniform Resource Name（统一资源命名），通过资源的名字来标识资源，与其所处的位置无关，这样即使资源的位置发生变动，其 URN 也不会变化。

## 3、协议版本

格式为 HTTP/主版本号.次版本号，常用为：HTTP/1.1 HTTP/1.0

## 4、请求头部

Host：接受请求的服务器地址，可以是 IP 或者是域名

User-Agent：发送请求的应用名称

Connection：指定与连接相关的属性，例如（Keep\_Alive，长连接）

Accept-Charset：通知服务器端可以发送的编码格式

Accept-Encoding：通知服务器端可以发送的数据压缩格式

Accept-Language：通知服务器端可以发送的语言

## 7、请你说一说 get 和 post 区别

考点：协议

参考回答：

GET：从指定的资源请求数据。

POST：向指定的资源提交要被处理的数据。

由于 HTTP 的规定和浏览器/服务器的限制，导致它们在实际过程中体现出一些不同。

	GET	POST
后退按钮/刷新	无害	数据会被重新提交（浏览器应该告知用户数据会被重新提交）。
缓存	能被缓存	不能缓存
编码方式	只能进行 url 编码	支持多种编码方式
是否保留在浏览历史	参数保留在浏览器历史中。	参数不会保存在浏览器历史中。
对数据长度的限制	发送数据，GET 方法向 URL 添加数据，但 URL 的长度是受限制的。	无限制。
对数据类型的限制	只允许 ASCII 字符。	没有限制。也允许二进制数据。
安全性	安全性较差，因为参数直接暴露在 url 中	因为参数不会被保存在浏览器历史或 web 服务器日志中。
可见性	数据在 URL 中对所有人都是可见的。	数据不会显示在 URL 中。
传参方式	get 参数通过 url 传递	post 放在 request body 中。

## 8、请你说一下 tcp 和 udp 的区别

考点：计算机网络

参考回答：

1、TCP 面向连接（如打电话要先拨号建立连接）；UDP 是无连接的，即发送数据之前不需要建立连接

2、TCP 提供可靠的服务。也就是说，通过 TCP 连接传送的数据，无差错，不丢失，不重复，且按序到达；UDP 尽最大努力交付，即不保证可靠交付

3、TCP 面向字节流，实际上是 TCP 把数据看成一连串无结构的字节流；UDP 是面向报文的，应用层交给 UDP 多长的报文，UDP 就照样发送，即一次发送一个报文。UDP 没有拥塞控制，因此网络出现拥塞不会使源主机的发送速率降低（对实时应用很有用，如 IP 电话，实时视频会议等）

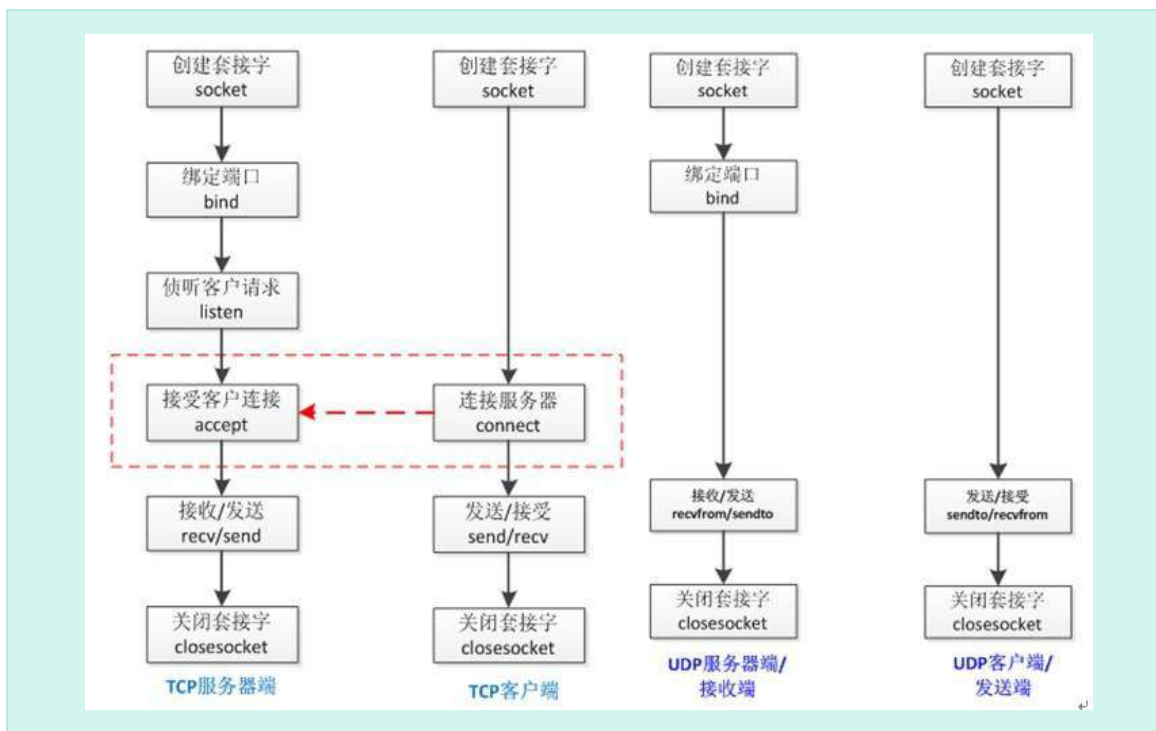
4、每一条 TCP 连接只能是点到点的；UDP 支持一对一，一对多，多对一和多对多的交互通信

5、TCP 首部开销 20 字节；UDP 的首部开销小，只有 8 个字节

6、TCP 的逻辑通信信道是全双工的可靠信道，UDP 则是不可靠信道

应用层协议	应用	传输层协议
SMTP	电子邮件	TCP
TELNET	远程终端接入	
HTTP	万维网	
FTP	文件传输	
DNS	名字转换	UDP
TFTP	文件传输	
RIP	路由选择协议	
BOOTP, DHCP	IP 地址配置	
SNMP	网络管理	
NFS	远程文件服务器	
专用协议	IP 电话	
专用协议	流式多媒体通信	

	TCP	UDP
可靠性	可靠	不可靠
连接性	面向连接	无连接
报文	面向字节流	面向报文（保留报文的边界）
效率	传输效率低	传输效率高
双工性	全双工	一对一、一对多、多对一、多对多
流量控制	有（滑动窗口）	无
拥塞控制	有（慢开始、拥塞避免、快重传、快恢复）	无



## 9、请你说一下 get 和 post 的区别

考点：计算机网络

参考回答：

GET - 从指定的资源请求数据。

POST - 向指定的资源提交要被处理的数据。

由于 HTTP 的规定和浏览器/服务器的限制，导致它们在实际应用中体现出一些不同。

	GET	POST
后退按钮/刷新	无害	数据会被重新提交（浏览器应该告知用户数据会被重新提交）。
缓存	能被缓存	不能缓存
编码方式	只能进行 url 编码	支持多种编码方式



是否保留在浏览历史	参数保留在浏览器历史中。	参数不会保存在浏览器历史中。
对数据长度的限制	发送数据，GET 方法向 URL 添加数据，但 URL 的长度是受限制的。	无限制。
对数据类型的限制	只允许 ASCII 字符。	没有限制。也允许二进制数据。
安全性	安全性较差，因为参数直接暴露在 url 中	因为参数不会被保存在浏览器历史或 web 服务器日志中。
可见性	数据在 URL 中对所有人都是可见的。	数据不会显示在 URL 中。
传参方式	get 参数通过 url 传递	post 放在 request body 中。

#### 10、请你说一下为什么 tcp 可靠，哪些方法保证可靠

考点：计算机网络

参考回答：

[1] 确认和重传机制

建立连接时三次握手同步双方的“序列号 + 确认号 + 窗口大小信息”，是确认重传、流控的基础

传输过程中，如果 Checksum 校验失败、丢包或延时，发送端重传。

[2] 数据排序

TCP 有专门的序列号 SN 字段，可提供数据 re-order

[3] 流量控制

滑动窗口和计时器的使用。TCP 窗口中会指明双方能够发送接收的最大数据量，发送方通过维持一个发送滑动窗口来确保不会发生由于发送方报文发送太快接收方无法及时处理的问题。





[4] 拥塞控制

TCP 的拥塞控制由 4 个核心算法组成：

“慢启动” (Slow Start)

“拥塞避免” (Congestion avoidance)

“快速重传” (Fast Retransmit)

“快速恢复” (Fast Recovery)

## 11、请你说一说 TCP 的流量控制

考点：计算机网络

参考回答：

滑动窗口机制：

滑动窗口协议的基本原理就是在任意时刻，发送方都维持了一个连续的允许发送的帧的序号，称为发送窗口；同时，接收方也维持了一个连续的允许接收的帧的序号，称为接收窗口。发送窗口和接收窗口的序号的上下界不一定要一样，甚至大小也可以不同。不同的滑动窗口协议窗口大小一般不同。发送方窗口内的序列号代表了那些已经被发送，但是还没有被确认的帧，或者是那些可以被发送的帧。

举例：



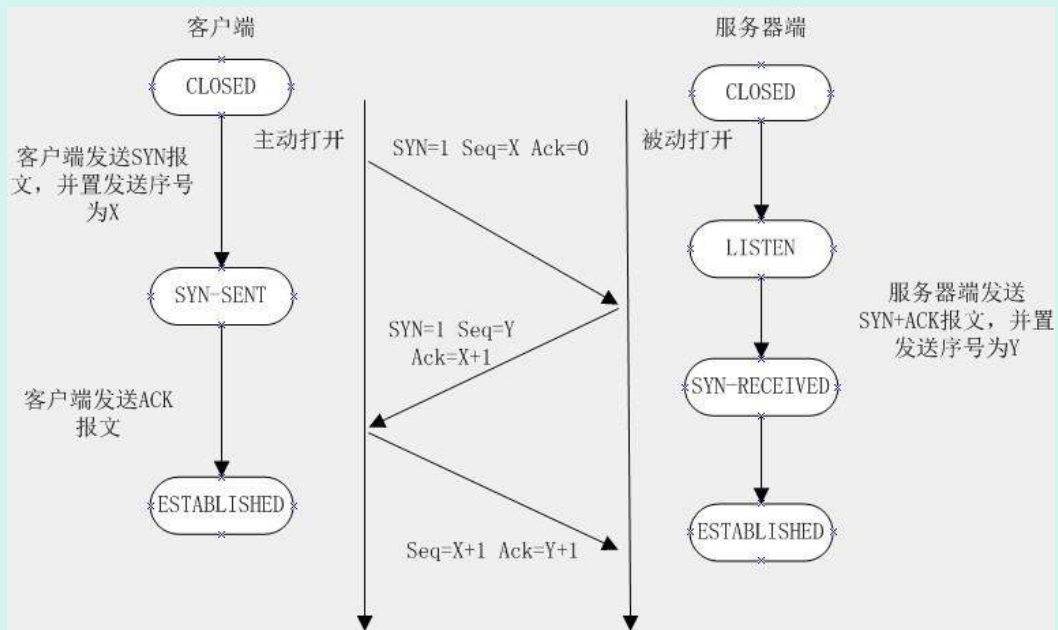
发送和接受方都会维护一个数据帧的序列，这个序列被称作窗口。发送方的窗口大小由接受方确定，目的在于控制发送速度，以免接受方的缓存不够大，而导致溢出，同时控制流量也可以避免网络拥塞。图中的 4, 5, 6 号数据帧已经被发送出去，但是未收到关联的 ACK，7, 8, 9 帧则是等待发送。可以看出发送端的窗口大小为 6，这是由接受端告知的（事实上必须考虑拥塞窗口  $cwnd$ ，这里暂且考虑  $cwnd > rwnd$ ）。此时如果发送端收到 4 号 ACK，则窗口的左边缘向右收缩，窗口的右边缘则向右扩展，此时窗口就向前“滑动了”，即数据帧 10 也可以被发送。

## 12、请你回答一下 TCP 三次握手，以及为什么不是两次

考点：计算机网络



参考回答：



第一次握手：建立连接时，客户端发送 syn 包（ $\text{syn}=j$ ）到服务器，并进入 SYN\_SENT 状态，等待服务器确认；SYN：同步序列编号（Synchronize Sequence Numbers）。

第二次握手：服务器收到 syn 包，必须确认客户的 SYN（ $\text{ack}=j+1$ ），同时自己也发送一个 SYN 包（ $\text{syn}=k$ ），即 SYN+ACK 包，此时服务器进入 SYN\_RECV 状态；

第三次握手：客户端收到服务器的 SYN+ACK 包，向服务器发送确认包 ACK（ $\text{ack}=k+1$ ），此包发送完毕，客户端和服务器进入 ESTABLISHED（TCP 连接成功）状态，完成三次握手。

为什么不是两次：

在服务端对客户端的请求进行回应(第二次握手)后，就会理所当然的认为连接已建立，而如果客户端并没有收到服务端的回应呢？此时，客户端仍认为连接未建立，服务端会对已建立的连接保存必要的资源，如果大量的这种情况，服务端会崩溃。

### 13、请你回答一下 ipv6 的位数

考点：网络

参考回答：

IPv6 的 128 位地址通常写成 8 组，每组由四个十六进制数组成。

### 14、请你说一说 osi 七层模型

考点：网络



参考回答：

#### 物理层

在 OSI 参考模型中，物理层（Physical Layer）是参考模型的最低层，也是 OSI 模型的第一层。

物理层的主要功能是：利用传输介质为数据链路层提供物理连接，实现比特流的透明传输。

物理层的作用是实现相邻计算机节点之间比特流的透明传送，尽可能屏蔽掉具体传输介质和物理设备的差异。使其上面的数据链路层不必考虑网络的具体传输介质是什么。“透明传送比特流”表示经实际电路传送后的比特流没有发生变化，对传送的比特流来说，这个电路好像是看不见的。

#### 数据链路层

数据链路层（Data Link Layer）是 OSI 模型的第二层，负责建立和管理节点间的链路。该层的主要功能是：通过各种控制协议，将有差错的物理信道变为无差错的、能可靠传输数据帧的数据链路。

在计算机网络中由于各种干扰的存在，物理链路是不可靠的。因此，这一层的主要功能是在物理层提供的比特流的基础上，通过差错控制、流量控制方法，使有差错的物理线路变为无差错的数据链路，即提供可靠的通过物理介质传输数据的方法。

该层通常又被分为介质访问控制（MAC）和逻辑链路控制（LLC）两个子层。

MAC 子层的主要任务是解决共享型网络中多用户对信道竞争的问题，完成网络介质的访问控制；

LLC 子层的主要任务是建立和维护网络连接，执行差错校验、流量控制和链路控制。

数据链路层的具体工作是接收来自物理层的位流形式的数据，并封装成帧，传送到上一层；同样，也将来自上层的数据帧，拆装为位流形式的数据转发到物理层；并且，还负责处理接收端发回的确认帧的信息，以便提供可靠的数据传输。

#### 网络层

网络层（Network Layer）是 OSI 模型的第三层，它是 OSI 参考模型中最复杂的一层，也是通信子网的最高一层。它在下两层的基础上向资源子网提供服务。其主要任务是：通过路由选择算法，为报文或分组通过通信子网选择最适当的路径。该层控制数据链路层与传输层之间的信息转发，建立、维持和终止网络的连接。具体地说，数据链路层的数据在这一层被转换为数据包，然后通过路径选择、分段组合、顺序、进/出路由等控制，将信息从一个网络设备传送到另一个网络设备。

一般地，数据链路层是解决同一网络内节点之间的通信，而网络层主要解决不同子网间的通信。例如在广域网之间通信时，必然会遇到路由（即两节点间可能有多条路径）选择问题。

在实现网络层功能时，需要解决的主要问题如下：



**寻址：**数据链路层中使用的物理地址（如 MAC 地址）仅解决网络内部的寻址问题。在不同子网之间通信时，为了识别和找到网络中的设备，每一子网中的设备都会被分配一个唯一的地址。由于各子网使用的物理技术可能不同，因此这个地址应当是逻辑地址（如 IP 地址）。

**交换：**规定不同的信息交换方式。常见的交换技术有：线路交换技术和存储转发技术，后者又包括报文交换技术和分组交换技术。

**路由算法：**当源节点和目的节点之间存在多条路径时，本层可以根据路由算法，通过网络为数据分组选择最佳路径，并将信息从最合适的路径由发送端传送到接收端。

**连接服务：**与数据链路层流量控制不同的是，前者控制的是网络相邻节点间的流量，后者控制的是从源节点到目的节点间的流量。其目的在于防止阻塞，并进行差错检测。

### 传输层

OSI 下 3 层的主要任务是数据通信，上 3 层的任务是数据处理。而传输层（Transport Layer）是 OSI 模型的第 4 层。因此该层是通信子网和资源子网的接口和桥梁，起到承上启下的作用。

该层的主要任务是：向用户提供可靠的端到端的差错和流量控制，保证报文的正确传输。传输层的作用是向高层屏蔽下层数据通信的细节，即向用户透明地传送报文。该层常见的协议：TCP/IP 中的 TCP 协议、Novell 网络中的 SPX 协议和微软的 NetBIOS/NetBEUI 协议。

传输层提供会话层和网络层之间的传输服务，这种服务从会话层获得数据，并在必要时，对数据进行分割。然后，传输层将数据传递到网络层，并确保数据能正确无误地传送到网络层。因此，传输层负责提供两节点之间数据的可靠传送，当两节点的联系确定之后，传输层则负责监督工作。综上，传输层的主要功能如下：

**传输连接管理：**提供建立、维护和拆除传输连接的功能。传输层在网络层的基础上为高层提供“面向连接”和“面向无连接”的两种服务。

**处理传输差错：**提供可靠的“面向连接”和不太可靠的“面向无连接”的数据传输服务、差错控制和流量控制。在提供“面向连接”服务时，通过这一层传输的数据将由目标设备确认，如果在指定的时间内未收到确认信息，数据将被重发。

监控服务质量。

### 会话层

会话层（Session Layer）是 OSI 模型的第 5 层，是用户应用程序和网络之间的接口，主要任务是：向两个实体的表示层提供建立和使用连接的方法。将不同实体之间的表示层的连接称为会话。因此会话层的任务就是组织和协调两个会话进程之间的通信，并对数据交换进行管理。

用户可以按照半双工、单工和全双工的方式建立会话。当建立会话时，用户必须提供他们想要连接的远程地址。而这些地址与 MAC（介质访问控制子层）地址或网络层的逻辑地址不同，它们是为用户专门设计的，更便于用户记忆。域名（DN）就是一种网络上使用的远程地址例如：www.3721.com 就是一个域名。会话层的具体功能如下：

会话管理：允许用户在两个实体设备之间建立、维持和终止会话，并支持它们之间的数据交换。例如提供单方向会话或双向同时会话，并管理会话中的发送顺序，以及会话所占用时间的长短。

会话流量控制：提供会话流量控制和交叉会话功能。

寻址：使用远程地址建立会话连接。<sup>1</sup>

出错控制：从逻辑上讲会话层主要负责数据交换的建立、保持和终止，但实际的工作却是接收来自传输层的数据，并负责纠正错误。会话控制和远程过程调用均属于这一层的功能。但应注意，此层检查的错误不是通信介质的错误，而是磁盘空间、打印机缺纸等类型的高级错误。

### 表示层

表示层（Presentation Layer）是 OSI 模型的第六层，它对来自应用层的命令和数据进行解释，对各种语法赋予相应的含义，并按照一定的格式传送给会话层。其主要功能是“处理用户信息的表示问题，如编码、数据格式转换和加密解密”等。表示层的具体功能如下：

数据格式处理：协商和建立数据交换的格式，解决各应用程序之间在数据格式表示上的差异。

数据的编码：处理字符集和数字的转换。例如由于用户程序中的数据类型（整型或实型、有符号或无符号等）、用户标识等都可以有不同的表示方式，因此，在设备之间需要具有在不同字符集或格式之间转换的功能。

压缩和解压缩：为了减少数据的传输量，这一层还负责数据的压缩与恢复。

数据的加密和解密：可以提高网络的安全性。

### 应用层

应用层（Application Layer）是 OSI 参考模型的最高层，它是计算机用户，以及各种应用程序和网络之间的接口，其功能是直接向用户提供服务，完成用户希望在网络上完成的各种工作。它在其他 6 层工作的基础上，负责完成网络中应用程序与网络操作系统之间的联系，建立与结束使用者之间的联系，并完成网络用户提出的各种网络服务及应用所需的监督、管理和服务等各种协议。此外，该层还负责协调各个应用程序间的工作。

应用层为用户提供的服务和协议有：文件服务、目录服务、文件传输服务（FTP）、远程登录服务（Telnet）、电子邮件服务（E-mail）、打印服务、安全服务、网络管理服务、数据库服务等。上述的各种网络服务由该层的不同应用协议和程序完成，不同的网络操作系统之间在功能、界面、实现技术、对硬件的支持、安全可靠性以及具有的各种应用程序接口等各个方面的差异是很大的。应用层的主要功能如下：

用户接口：应用层是用户与网络，以及应用程序与网络间的直接接口，使得用户能够与网络进行交互式联系。

实现各种服务：该层具有的各种应用程序可以完成和实现用户请求的各种服务。





### 15、请你说一说 DNS 解析过程

考点：网络

参考回答：

1、浏览器先检查自身缓存中有没有被解析过的这个域名对应的 ip 地址，如果有，解析结束。同时域名被缓存的时间也可通过 TTL 属性来设置。

2、如果浏览器缓存中没有（专业点叫还没命中），浏览器会检查操作系统缓存中有没有对应的已解析过的结果。而操作系统也有一个域名解析的过程。在 windows 中可通过 c 盘里一个叫 hosts 的文件来设置，如果你在这里指定了一个域名对应的 ip 地址，那浏览器会首先使用这个 ip 地址。

但是这种操作系统级别的域名解析规程也被很多黑客利用，通过修改你的 hosts 文件里的内容把特定的域名解析到他指定的 ip 地址上，造成所谓的域名劫持。所以在 windows7 中将 hosts 文件设置成了 readonly，防止被恶意篡改。

3. 如果至此还没有命中域名，才会真正的请求本地域名服务器（LDNS）来解析这个域名，这台服务器一般在你的城市的某个角落，距离你不会很远，并且这台服务器的性能都很好，一般都会缓存域名解析结果，大约 80%的域名解析到这里就完成了。

4. 如果 LDNS 仍然没有命中，就直接跳到 Root Server 域名服务器请求解析

5. 根域名服务器返回给 LDNS 一个所查询域的主域名服务器（gTLD Server，国际顶尖域名服务器，如 .com .cn .org 等）地址

6. 此时 LDNS 再发送请求给上一步返回的 gTLD

7. 接受请求的 gTLD 查找并返回这个域名对应的 Name Server 的地址，这个 Name Server 就是网站注册的域名服务器

8. Name Server 根据映射关系表找到目标 ip，返回给 LDNS

9. LDNS 缓存这个域名和对应的 ip

10. LDNS 把解析的结果返回给用户，用户根据 TTL 值缓存到本地系统缓存中，域名解析过程至此结束

### 16、请你说一说 http 和 https 区别

考点：网络

参考回答：

HTTP 协议传输的数据都是未加密的，也就是明文的，因此使用 HTTP 协议传输隐私信息非常不安全，为了保证这些隐私数据能加密传输，于是网景公司设计了 SSL（Secure Sockets Layer）



协议用于对 HTTP 协议传输的数据进行加密，从而就诞生了 HTTPS。简单来说，HTTPS 协议是由 SSL+HTTP 协议构建的可进行加密传输、身份认证的网络协议，要比 http 协议安全。

HTTPS 和 HTTP 的区别主要如下：

- 1、https 协议需要到 ca 申请证书，一般免费证书较少，因而需要一定费用。
- 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 和 https 使用的是完全不同的连接方式，用的端口也不一样，前者是 80，后者是 443。
- 4、http 的连接很简单，是无状态的；HTTPS 协议是由 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。

#### 17、请你说一说 get 和 post 区别

考点：网络

参考回答：

GET - 从指定的资源请求数据。

POST - 向指定的资源提交要被处理的数据。

由于 HTTP 的规定和浏览器/服务器的限制，导致它们在实际过程中体现出一些不同。

	GET	POST
后退按钮/刷新	无害	数据会被重新提交（浏览器应该告知用户数据会被重新提交）。
缓存	能被缓存	不能缓存
编码方式	只能进行 url 编码	支持多种编码方式

是否保留在浏览历史	参数保留在浏览器历史中。	参数不会保存在浏览器历史中。
对数据长度的限制	发送数据，GET 方法向 URL 添加数据，但 URL 的长度是受限制的。	无限制。
对数据类型的限制	只允许 ASCII 字符。	没有限制。也允许二进制数据。
安全性	安全性较差，因为参数直接暴露在 url 中	因为参数不会被保存在浏览器历史或 web 服务器日志中。
可见性	数据在 URL 中对所有人都是可见的。	数据不会显示在 URL 中。
传参方式	get 参数通过 url 传递	post 放在 request body 中。

## 18、请你说一下 https 中 SSL 层原理

考点：网络

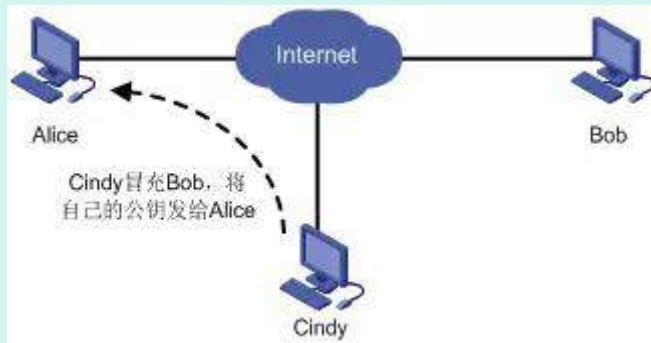
参考回答：

SSL 利用数据加密、身份验证和消息完整性验证机制，为网络上数据的传输提供安全性保证。SSL 支持各种应用层协议。由于 SSL 位于应用层和传输层之间，所以可以为任何基于 TCP 等可靠连接的应用层协议提供安全性保证。

### 1. 身份验证机制

SSL 利用数字签名来验证通信对端的身份。非对称密钥算法可以用来实现数字签名。由于通过私钥加密后的数据只能利用对应的公钥进行解密，因此根据解密是否成功，就可以判断发送者的身份，如同发送者对数据进行了“签名”。例如，Alice 使用自己的私钥对一段固定的信息加密后发给 Bob，Bob 利用 Alice 的公钥解密，如果解密结果与固定信息相同，那么就能够确认信息的发送者为 Alice，这个过程就称为数字签名。使用数字签名验证身份时，需要确保被验证者的公钥是真实的，否则，非法用户可能会冒充被验证者与验证者通信。如下图所示，Cindy 冒充 Bob，将自己的公钥发给 Alice，并利用自己的私钥计算出签名发送给 Alice，Alice 利用“Bob”的公钥（实际上为 Cindy 的公钥）成功验证该签名，则 Alice 认为 Bob 的身份验证成功，而实际上与 Alice 通信的是冒充 Bob 的 Cindy。SSL 利用 PKI 提供的机制保证公钥的真实性。



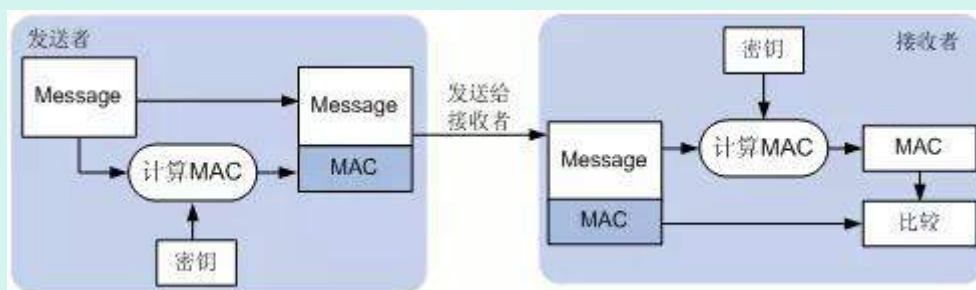


## 2. 数据传输的机密性

SSL 加密通道上的数据加解密使用对称密钥算法，目前主要支持的算法有 DES、3DES、AES 等，这些算法都可以有效地防止交互数据被破解。对称密钥算法要求解密密钥和加密密钥完全一致。因此，利用对称密钥算法加密传输数据之前，需要在通信两端部署相同的密钥。

## 3. 消息完整性验证

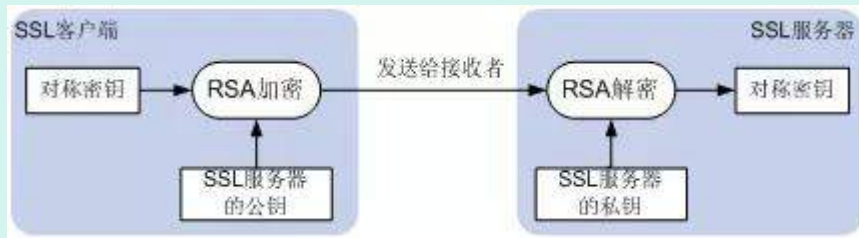
为了避免网络中传输的数据被非法篡改，SSL 利用基于 MD5 或 SHA 的 MAC 算法来保证消息的完整性。MAC 算法是在密钥参与下的数据摘要算法，能将密钥和任意长度的数据转换为固定长度的数据。利用 MAC 算法验证消息完整性的过程如下图所示。发送者在密钥的参与下，利用 MAC 算法计算出消息的 MAC 值，并将其加在消息之后发送给接收者。接收者利用同样的密钥和 MAC 算法计算出消息的 MAC 值，并与接收到的 MAC 值比较。如果二者相同，则报文没有改变；否则，报文在传输过程中被修改，接收者将丢弃该报文。



MAC 算法要求通信双方具有相同的密钥，否则 MAC 值验证将会失败。因此，利用 MAC 算法验证消息完整性之前，需要在通信两端部署相同的密钥。

## 4. 利用非对称密钥算法保证密钥本身的安全

对称密钥算法和 MAC 算法要求通信双方具有相同的密钥，否则解密或 MAC 值验证将失败。因此，要建立加密通道或验证消息完整性，必须先要在通信双方部署一致的密钥。SSL 利用非对称密钥算法加密密钥的方法实现密钥交换，保证第三方无法获取该密钥。如下图所示，SSL 客户端（如 Web 浏览器）利用 SSL 服务器（如 Web 服务器）的公钥加密密钥，将加密后的密钥发送给 SSL 服务器，只有拥有对应私钥的 SSL 服务器才能从密文中获取原始的密钥。SSL 通常采用 RSA 算法加密传输密钥。（Server 端公钥加密密钥，私钥解密密钥）



实际上，SSL 客户端发送给 SSL 服务器的密钥不能直接用来加密数据或计算 MAC 值，该密钥是用来计算对称密钥和 MAC 密钥的信息，称为 premaster secret。SSL 客户端和 SSL 服务器利用 premaster secret 计算出相同的主密钥（master secret），再利用 master secret 生成用于对称密钥算法、MAC 算法等的密钥。premaster secret 是计算对称密钥、MAC 算法密钥的关键。

#### 5. 利用 PKI 保证公钥的真实性

PKI 通过数字证书来发布用户的公钥，并提供了验证公钥真实性的机制。数字证书（简称证书）是一个包含用户的公钥及其身份信息文件，证明了用户与公钥的关联。数字证书由权威机构——CA 签发，并由 CA 保证数字证书的真实性。

SSL 客户端把密钥加密传递给 SSL 服务器之前，SSL 服务器需要将从 CA 获取的证书发送给 SSL 客户端，SSL 客户端通过 PKI 判断该证书的真实性。如果该证书确实属于 SSL 服务器，则利用该证书中的公钥加密密钥，发送给 SSL 服务器。

验证 SSL 服务器/SSL 客户端的身份之前，SSL 服务器/SSL 客户端需要将从 CA 获取的证书发送给对端，对端通过 PKI 判断该证书的真实性。如果该证书确实属于 SSL 服务器/SSL 客户端，则对端利用该证书中的公钥验证 SSL 服务器/SSL 客户端的身份。

## 19、请你说一说 TCP 断连过程，以及单向连接关闭后还能否通信

考点：网络

参考回答：

由于 TCP 连接是全双工的，因此每个方向都必须单独进行关闭。这个原则是当一方完成它的数据发送任务后就能发送一个 FIN 来终止这个方向的连接。收到一个 FIN 只意味着这一方向上没有数据流动，一个 TCP 连接在收到一个 FIN 后仍能发送数据。首先进行关闭的一方将执行主动关闭，而另一方执行被动关闭。四次挥手过程：

- （1）客户端 A 发送一个 FIN，用来关闭客户 A 到服务器 B 的数据传送。
- （2）服务器 B 收到这个 FIN，它发回一个 ACK，确认序号为收到的序号加 1。和 SYN 一样，一个 FIN 将占用一个序号。
- （3）服务器 B 关闭与客户端 A 的连接，发送一个 FIN 给客户端 A。
- （4）客户端 A 发回 ACK 报文确认，并将确认序号设置为收到序号加 1。



四次挥手原因：这是因为服务端的 LISTEN 状态下的 SOCKET 当收到 SYN 报文的建连请求后，它可以把 ACK 和 SYN（ACK 起应答作用，而 SYN 起同步作用）放在一个报文里来发送。但关闭连接时，当收到对方的 FIN 报文通知时，它仅仅代表对方没有数据发送给你了；但未必你所有的数据都全部发送给对方了，所以你可以未必会马上会关闭 SOCKET，也即你可能还需要发送一些数据给对方之后，再发送 FIN 报文给对方来表示你同意现在可以关闭连接了，所以它这里的 ACK 报文和 FIN 报文多数情况下都是分开发送的。

## 20、请你说说 TCP 和 UDP 用一个端口发送信息是否冲突

考点：网络

参考回答：

不冲突，TCP、UDP 可以绑定同一端口来进行通信，许多协议已经这样做了，例如 DNS 适用于 udp / 53 和 tcp / 53。因为数据接收时时根据五元组 {传输协议，源 IP，目的 IP，源端口，目的端口} 判断接受者的。

## 21、请你说说 HTTP 常见头

考点：网络

参考回答：

1、Accept: text/html, application/xhtml+xml, application/xml;q=0.9, image/webp, image/apng, \*/\*; q=0.8

作用：向服务器申明客户端（浏览器）可以接受的媒体类型（MIME）的资源

解释：浏览器可以接受 text/html、application/xhtml+xml、application/xml 类型，通配符\*/\* 表示任意类型的数据。并且浏览器按照该顺序进行接收。（text/html → application/xhtml+xml → application/xml）

2、Accept-encoding: gzip, deflate, br

作用：向服务器申明客户端（浏览器）接收的编码方法，通常为压缩方法

解释：浏览器支持采用经过 gzip, deflate 或 br 压缩过的资源

3、Accept-Language: en-US,en;q=0.9,zh-CN;q=0.8,zh;q=0.7

作用：向服务器申明客户端（浏览器）接收的语言

解释：浏览器能够接受 en-US, en 和 zh-CN 三种语言，其中 en-US 的权重最高（q 最高为 1，最低为 0），服务器优先返回 en-US 语言

延伸：语言与字符集的区别：zh-CN 为汉语，汉语中有许多的编码：gbk2312 等



#### 4、Cache-control: max-age=0

作用:控制浏览器的缓存,常见值为 private、no-cache、max-age、alidate,默认为 private,根据浏览器查看页面不同的方式来进行区别

解释: 浏览器在访问了该页面后,不再会访问服务器

#### 5、Cookie:

作用: 告诉服务器关于 Session 的信息, 存储让服务器辨识用户身份的信息。

#### 6、Refer: https://www.baidu.com/xxxxxxxxx

作用: 告诉服务器该页面从哪个页面链接的

解释: 该页面从 https://www.baidu.com 中的搜索结果中点击过来的

#### 7、Upgrade-insecure-requests: 1

作用: 申明浏览器支持从 http 请求自动升级为 https 请求, 并且在以后发送请求的时候都使用 https

解释: 当页面中包含大量的 http 资源的时候(图片、iframe), 如果服务器发现一旦存在上述的响应头的时候, 会在加载 http 资源的时候自动替换为 https 请求

#### 8、User-agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_13\_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.132 Safari/537.36

作用: 向服务器发送浏览器的版本、系统、应用程序的信息。

解释: Chrome 浏览器的版本信息为 63.0.3239.132, 并将自己伪装成 Safari, 使用的是 WebKit 引擎, WebKit 伪装成 KHTML, KHTML 伪装成 Gecko(伪装是为了接收那些为 Mozilla、safari、gecko 编写的界面)

延伸: 可以随便填(但不应该随便填)不过一般用于统计。

#### 9、X-Chrome-UMA-Enabled、X-Client-Data : 与 Chrome 浏览器相关的数据

Response Headers

## 22、请你说说 HTTP 状态码

考点: 网络

参考回答:

状态码, 100~199 表示请求已收到继续处理, 200~299 表示成功, 300~399 表示资源重定向, 400~499 表示客户端请求出错, 500~599 表示服务器端出错



200: 响应成功

302: 跳转，重定向

400: 客户端有语法错误

403: 服务器拒绝提供服务

404: 请求资源不存在

500: 服务器内部错误

### 23、请你说说 socket 编程和 http 协议

考点：网络

参考回答：

由于通常情况下 Socket 连接就是 TCP 连接，因此 Socket 连接一旦建立，通信双方即可开始相互发送数据内容，直到双方连接断开。但在实际网络应用中，客户端到服务器之间的通信往往需要穿越多个中间节点，例如路由器、网关、防火墙等，大部分防火墙默认会关闭长时间处于非活跃状态的连接而导致 Socket 连接断连，因此需要通过轮询告诉网络，该连接处于活跃状态。

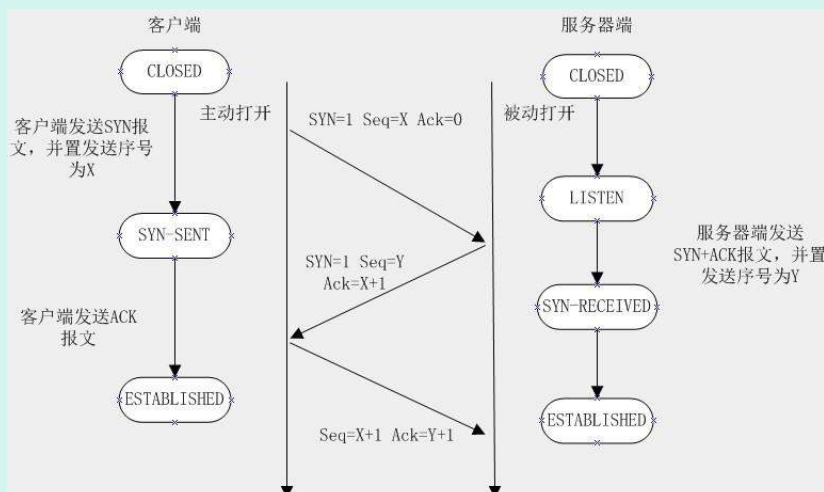
而 HTTP 连接使用的是“请求—响应”的方式，不仅在请求时需要先建立连接，而且需要客户端向服务器发出请求后，服务器端才能回复数据。

很多情况下，需要服务器端主动向客户端推送数据，保持客户端与服务器数据的实时与同步。此时若双方建立的是 Socket 连接，服务器就可以直接将数据传送给客户端；若双方建立的是 HTTP 连接，则服务器需要等到客户端发送一次请求后才能将数据传回给客户端，因此，客户端定时向服务器端发送连接请求，不仅可以保持在线，同时也是在“询问”服务器是否有新的数据，如果有就将数据传给客户端。

### 24、请你说说 tcp 三次握手四次挥手

考点：计算机网络

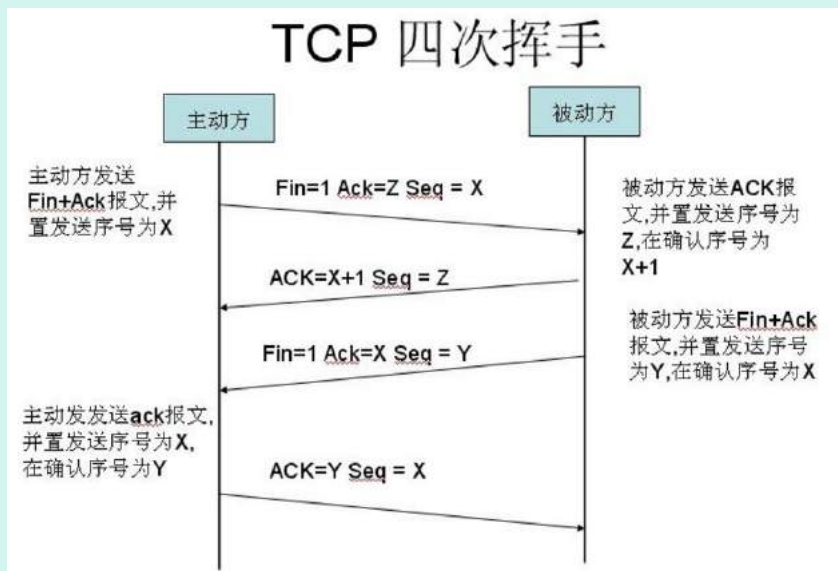
参考回答：



第一次握手：建立连接时，客户端发送 syn 包（syn=j）到服务器，并进入 SYN\_SENT 状态，等待服务器确认；SYN：同步序列编号（Synchronize Sequence Numbers）。

第二次握手：服务器收到 syn 包，必须确认客户的 SYN（ack=j+1），同时自己也发送一个 SYN 包（syn=k），即 SYN+ACK 包，此时服务器进入 SYN\_RECV 状态；

第三次握手：客户端收到服务器的 SYN+ACK 包，向服务器发送确认包 ACK(ack=k+1)，此包发送完毕，客户端和服务器进入 ESTABLISHED（TCP 连接成功）状态，完成三次握手。



由于 TCP 连接是全双工的，因此每个方向都必须单独进行关闭。这个原则是当一方完成它的数据发送任务后就能发送一个 FIN 来终止这个方向的连接。收到一个 FIN 只意味着这一方向上没有数据流动，一个 TCP 连接在收到一个 FIN 后仍能发送数据。首先进行关闭的一方将执行主动关闭，而另一方执行被动关闭。四次挥手过程：

(1) 客户端 A 发送一个 FIN，用来关闭客户 A 到服务器 B 的数据传送。



(2) 服务器 B 收到这个 FIN，它发回一个 ACK，确认序号为收到的序号加 1。和 SYN 一样，一个 FIN 将占用一个序号。

(3) 服务器 B 关闭与客户端 A 的连接，发送一个 FIN 给客户端 A。

(4) 客户端 A 发回 ACK 报文确认，并将确认序号设置为收到序号加 1。

四次挥手原因：这是因为服务端的 LISTEN 状态下的 SOCKET 当收到 SYN 报文的建连请求后，它可以把 ACK 和 SYN（ACK 起应答作用，而 SYN 起同步作用）放在一个报文里来发送。但关闭连接时，当收到对方的 FIN 报文通知时，它仅仅表示对方没有数据发送给你了；但未必你所有的数据都全部发送给对方了，所以你可以未必会马上会关闭 SOCKET，也即你可能还需要发送一些数据给对方之后，再发送 FIN 报文给对方来表示你同意现在可以关闭连接了，所以它这里的 ACK 报文和 FIN 报文多数情况下都是分开发送的。

## 25、请你说说 post 和 get 的区别

考点：网络

参考回答：

GET - 从指定的资源请求数据。

POST - 向指定的资源提交要被处理的数据。

由于 HTTP 的规定和浏览器/服务器的限制，导致它们在应用过程中体现出一些不同。

	GET	POST
后退按钮/刷新	无害	数据会被重新提交（浏览器应该告知用户数据会被重新提交）。
缓存	能被缓存	不能缓存
编码方式	只能进行 url 编码	支持多种编码方式

是否保留在浏览历史	参数保留在浏览器历史中。	参数不会保存在浏览器历史中。
对数据长度的限制	发送数据，GET 方法向 URL 添加数据，但 URL 的长度是受限制的。	无限制。
对数据类型的限制	只允许 ASCII 字符。	没有限制。也允许二进制数据。
安全性	安全性较差，因为参数直接暴露在 url 中	因为参数不会被保存在浏览器历史或 web 服务器日志中。
可见性	数据在 URL 中对所有人都是可见的。	数据不会显示在 URL 中。
传参方式	get 参数通过 url 传递	post 放在 request body 中。

## 26、请你说说 HTTP 状态码，HTTP 请求协议

考点：网络

参考回答：

状态码，100~199 表示请求已收到继续处理，200~299 表示成功，300~399 表示资源重定向，400~499 表示客户端请求出错，500~599 表示服务器端出错

200：响应成功

302：跳转，重定向

400：客户端有语法错误

403：服务器拒绝提供服务

404：请求资源不存在





500：服务器内部错误

http 请求报文：

HTTP请求报文							
请求方法	空格	URL	空格	协议版本	回车符	换行符	请求行
头部字段名	:	值	回车符	换行符	请求头部		
		.....					
头部字段名	:	值	回车符	换行符			
回车符	换行符						
请求正文							

### 1、请求方法

GET：请求获取 Request——URL 所标识的资源

POST：在 Request——URL 所标识的资源后附加资源

HEAD：请求获取由 Request——URL 所标识的资源的响应消息报头

PUT：请求服务器存储一个资源，由 Request——URL 作为其标识

DELETE：请求服务器删除由 Request——URL 所标识的资源

TRACE：请求服务器回送收到的请求信息（用于测试和诊断）

CONNECT：保留

OPTIONS：请求查询服务器性能

### 2、URL

URI 全名为 Uniform Resource Identifier（统一资源标识），用来唯一的标识一个资源，是一个通用的概念，URI 由两个主要的子集 URL 和 URN 组成。URL 全名为 Uniform Resource Locator（统一资源定位），通过描述资源的位置来标识资源。URN 全名为 Uniform Resource Name（统一资源命名），通过资源的名字来标识资源，与其所处的位置无关，这样即使资源的位置发生变动，其 URN 也不会变化。

### 3、协议版本

格式为 HTTP/主版本号.次版本号，常用为：HTTP/1.1 HTTP/1.0

### 4、请求头部

Host：接受请求的服务器地址，可以是 IP 或者是域名

User-Agent：发送请求的应用名称

Connection：指定与连接相关的属性，例如（Keep\_Alive，长连接）



Accept-Charset: 通知服务器端可以发送的编码格式

Accept-Encoding: 通知服务器端可以发送的数据压缩格式

Accept-Language: 通知服务器端可以发送的语言

## 26、请你说一说 http 缓存问题，缓存寿命，以及怎么判断文件在服务器是否更改的

考察点：计算机网络

参考回答：

### 1 缓存的类型：

缓存是一种保存资源副本并在下次请求中直接使用该副本的技术，缓存能够节约网络资源，提升页面响应速度。常见的缓存类型分为共享缓存和私有缓存

#### 1.1 私有缓存

私有缓存只能用于单独用户，常见的浏览器缓存便是私有缓存。私有缓存能够存储用户通过 http 下载过的文档，从而在用户再次访问时直接提供给用户，而不用向服务器发送请求。

#### 1.2 共享缓存

共享缓存能够被多个用户使用，常用的 web 代理中便使用的共享缓存

缓存寿命

缓存寿命的计算的依据依次是：

请求头中的 Cache-Control: max-age=N。相应的缓存寿命即为 N，从设置开始，N 秒之后过期。

Expires 属性，Expires 属性的值为过期的时间点，在这个时间点后，该缓存被认为过期

Last-Modified 信息。缓存的寿命为头里面 Date 表示的事件点减去 Last-Modified 的时间点的结果乘以 10%

判断文件是否更改可以看文件时间戳

## 27、请你回答一下搜索敏感词汇时，页面被重置的原理

考点：网络

参考回答：



根据 TCP 协议的规定，用户和服务器建立连接需要三次握手：第一次握手用户向服务器发送 SYN 数据包发出请求（SYN，x:0），第二次握手服务器向用户发送 SYN/ACK 数据包发出回应（SYN/ACK，y:x+1），第三次握手用户向服务器发送 ACK 数据包发出确认（ACK，x+1:y+1），至此一个 TCP 连接建立成功。其中 x 为用户向服务器发送的序列号，y 为服务器向用户发送的序列号。

关键字检测，针对明文或者 base64 等弱加密通讯内容，与准备好的敏感词库进行匹配，当发现敏感词时，将服务器发回的 SYN/ACK 包改成 SYN/ACK，Y:0，这代表 TCP 连接被重置，用户便主动放弃了连接，提示连接失败。让用户误认为服务器拒绝连接，而主动放弃继续与服务器连接，自动阻断记录含有敏感词的网页

## 28、请你说一说两个机器之间的通讯过程？以及计算机网络为什么有七层？

考察点：计算机网络

参考回答：

PC1 首先判断目标 ip 是否和自己在同一网段，是就进行 ARP 广播，解析出 MAC 地址。不是，则将网关的 MAC 地址作为 MAC 地址。

PC1 封装的数据包括目标、源的端口号、IP、MAC 地址。

交换机收到数据后，对比 MAC 地址表，知道从哪个口发出数据。

路由收到数据后根据路由表将数据发往下一个目标地。

最后一个路由通过 ARP 解析出 PC2 的 MAC 地址。

路由封装的数据包括目标、源的端口号、IP、MAC 地址。

建立七层模型的主要目的是为解决异种网络互连时所遇到的兼容性问题。它的最大优点是将服务、接口和协议这三个概念明确地区分开来：服务说明某一层为上一层提供一什么功能，接口说明上一层如何使用下层的服务，而协议涉及如何实现本层的服务；这样各层之间具有很强的独立性，互连网络中各实体采用什么样的协议是没有限制的，只要向上提供相同的服务并且不改变相邻层的接口就可以了。网络七层的划分也是为了使网络的不同功能模块（不同层次）分担起不同的职责，从而带来如下好处：

- 减轻问题的复杂程度，一旦网络发生故障，可迅速定位故障所处层次，便于查找和纠错；
- 在各层分别定义标准接口，使具备相同对等层的不同网络设备能实现互操作，各层之间则相对独立，一种高层协议可放在多种低层协议上运行；
- 能有效刺激网络技术革新，因为每次更新都可以在小范围内进行，不需对整个网络动大手术；

## 29、请你说一说 http 缓存问题，缓存寿命，怎么判断文件在服务器是否更改的

考察点：计算机网络

参考回答：

1 缓存的类型：



缓存是一种保存资源副本并在下次请求中直接使用该技术，缓存能够节约网络资源，提升页面响应速度。常见的缓存类型分为共享缓存和私有缓存

### 1.1 私有缓存

私有缓存只能用于单独用户，常见的浏览器缓存便是私有缓存。私有缓存能够存储用户通过 http 下载过的文档，从而在用户再次访问时直接提供给用户，而不用向服务器发送请求。

### 1.2 共享缓存

共享缓存能够被多个用户使用，常用的 web 代理中便使用的共享缓存

缓存寿命

缓存寿命的计算的依据依次是：

请求头中的 Cache-Control: max-age=N。相应的缓存寿命即为 N，从设置开始，N 秒之后过期。

Expires 属性，Expires 属性的值为过期的时间点，在这个时间点后，该缓存被认为过期

Last-Modified 信息。缓存的寿命为头里面 Date 表示的事件点减去 Last-Modified 的时间点的结果乘以 10%

判断文件是否更改可以看文件时间戳

30、请你说一说是 http 协议，http 的数据段包括什么？http 为什么是无状态的，http 和 https 的区别？ip 地址的 abcd 类是怎样分的，ABCD 分层协议为什么如此分层，什么是长连接和短链接

考察点：计算机网络

参考回答：

什么是 http 协议？

http (hyperText transport Protocol) 是超文本传输协议的缩写，它用于传送 www 方式的数据，关于 http 协议采用了请求/响应模型，客户端向服务器发送了一个请求，服务器以一个状态行作为响应

http 的数据段包括什么？

通常 http 消息包括客户机向服务器请求消息和服务器向客户机的响应消息，这两种类型的消息由一个起始行，一个或多个头域，一个指示头域结束的空行和可选的消息体组成，http 的头域包括通用头，请求头，响应头，和实体头四个部分，每个头域由一个域名，冒号，和域值三部分组成，域名是大小写无关的，域值前可以添加任何数量的空格符，头域可以被扩张成多行，在每行开始处，使用至少一个空格或制表符。



http 为什么是无状态的？

无状态是指协议对于事务处理没有记忆能力，因为 http 协议目的在于支持超文本的传输，更加广义一点就是支持资源的传输，那么在客户端浏览器向服务器发送请求，继而服务器将相应的资源发回客户这样一个过程中，无论对于客户端还是服务器，都没有必要记录这个过程，因为每一次请求和响应都是相对独立的，一般而言，一个 url 对应唯一的超文本，正因为这样 d 唯一性，所以 http 协议被设计为无状态的链接协议符合他本身的需求。

http 和 https 的区别？

http 和 https 的区别主要如下：

- 1、https 需要到 ca 申请证书，因而需要一定费用
- 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 的连接很简单，是无状态的，https 协议是由 ssl+http 协议构建的可进行加密串苏，身份验证的网络协议
- 4、http 用的端口是 80，https 用的端口是 443

ip 地址的 abcd 类是怎样分的

A 类地址的表示范围是：0.0.0.0-126.255.255.255，默认网络掩码为：255.0.0.0，A 类地址分配给规模特别大的网络使用，

B 类地址表示范围是：128.0.0.0-191.255.255.255，默认网络掩码为欸：255.255.0.0，B 类地址分配给一般的中型网络

C 类地址的表示范围是 192.0.0.0-223.255.255.255，默认网络掩码是：255.255.255.0，C 类地址分配给小型网络，如局域网

D 类地址称为广播地址，共特殊协议向选定的节点发送信息使用。

这样便于寻址和层次化的构造网络。

什么是长连接和短连接？

http1.0 中默认使用短连接，服务器和客户端没进行一次 http 操作，就建立一次连接，任务结束就终端连接，http1.1 起。默认使用长连接，用以保持连接特性，当一个网页打开完成后，服务器和客户端之间用于传输 http 数据的 tcp 连接不会关闭，客户端再次访问这个服务器时，会继续使用这一条已经建立好的连接。

31、请你说一说 tcp 数据段都包括什么？tcp 三次握手四次挥手过程、为什么挥手是四次

考察点：计算机网络



参考回答：

TCP 数据段：

源端口（Source port） 和 目的端口（Destination port）：

字段标明了在一个连接的两个端点用来跟踪同一时间内通过网络的不同会话。一般每个端口对应一个应用程序

序列号 (Sequence number)：字节号（32 位），表示一个字节的编号

初始序列号 ISNs (initial sequence numbers)：随机产生的

SYN：携带了 ISNs 和 SYN 控制位的数据段

确认号（Acknowledgement number）：期望接收的字节号（32 位）

TCP 段头长度（TCP header length）：TCP 段头长度，单位 32 位（4 字节）

保留域/字段：逐步启用，如做拥塞控制等

URG：当紧急指针使用的时候，URG 被置为 1。紧急指针是一个对于当前序列号的字节偏移量，标明紧急数据从哪里开始

当 URG=1 时，表明有紧急数据，必须首先处理收方收到这样的数据后，马上处理，处理完后恢复正常操作即使 win=0，也可以发送这样的数据

ACK：为 1 表示确认号有效，为 0 标明确认号无效

PSH：表示这是带有 PUSH 标志的数据，接收方收到这样的数据，应该立刻送到上层，而不需要缓存它

RST：被用来重置一个已经混乱的连接

SYN：用在连接建立过程中

SYN=1，ACK=0 连接请求，当 SYN=1，ACK=1 连接接受

FIN：被用来释放连接，它表示发送方已经没有数据要传输了，但是可以继续接收数据

Window size：告诉对方可以发送的数据字节数，从确认字节号开始（决定于接收方）

Checksum：提供额外的可靠性，校验的范围包括头部、数据和概念性的伪头部

选项域：选项域提供了一种增加基本头没有包含内容的方法

Tcp 三次握手：

1、第一次握手，客户端发送 syn=j 到服务器



2、服务器返回  $\text{syn}=\text{k}, \text{ack}=\text{j}+1$ ,

3、客户端再向服务器发送  $\text{ack}=\text{k}+1$ ;

三次握手结束，客户端和服务器建立连接

TCP 四次挥手：

1、客户端发送  $\text{fin}=\text{j}$  包关闭连接

2、服务器返回  $\text{ack}=\text{j}+1$

3、服务器发送  $\text{fin}=\text{k}$  包关闭连接

4、客户端返回  $\text{ack}=\text{k}+1$

四次挥手结束，连接断开

为什么连接三次，断开连接四次？

在连接中，服务器的  $\text{ack}$  和  $\text{syn}$  包是同时发送的，而在断开连接的时候，服务器向客户端发送的  $\text{ack}$  和  $\text{fin}$  包是分两次发送的，因为服务器收到客户端发送的  $\text{fin}$  包时，可能还有数据要传送，所以先发送  $\text{ack}$ ，等数据传输结束后再发送  $\text{fin}$  断开这边的连接。

**32、请你说一下 tcp 与 udp 的区别，以及 tcp 为什么可靠，tcp 滑动窗口，同传，拆包组发包是如何实现的**

考察点：计算机网络

参考回答：

TCP 和 UDP 区别在于：

1、TCP 提供面向对象的连接，通信前要建立三次握手机制的连接，UDP 提供无连接的传输，传输前不用建立连接

2、TCP 提供可靠的，有序的，不丢失的传输，UDP 提供不可靠的传输

3、TCP 提供面向字节流的传输，它能将信息分割成组，并在接收端将其充足，UDP 提供面向数据报的传输，没有分组开销

4、TCP 提供拥塞控制，流量控制机制，UDP 没有

TCP 为什么可靠

1、确认和重传机制





建立连接时三次握手连接机制是确认重传流量控制的基础，传输过程中如果校验失败，丢包或延时，发送端重传

## 2、数据排序

TCP 有专门的序列号字段，可提供数据 reorder

## 3、流量控制

TCP 窗口会指明双方能够发送接收的最大数据量

## 4、拥塞控制

TCP 滑动窗口

TCP 建立连接时，各端分配一个缓冲区用来存储接受的数据，并将缓冲区的尺寸发送给另一端，接收方发送的确认消息中包含了自己剩余的缓冲区尺寸，剩余缓冲区空间的数量叫做窗口，所谓滑动窗口，就是接收端可以根据自己的状况通告窗口大小，从而控制发送端的接收，进行流量控制。

Tcp 如何进行拆包、组装包？

拆包：

对于拆包目前常用的是以下两种方式：

1、动态缓冲区暂存方式。之所以说缓冲区是动态的是因为当需要缓冲的数据长度超出缓冲区的长度时会增大缓冲区长度。

大概过程描述如下：

A 为每一个连接动态分配一个缓冲区，同时把此缓冲区和 SOCKET 关联，常用的是通过结构体关联。

B 当接收到数据时首先把此段数据存放在缓冲区中。

C 判断缓存区中的数据长度是否够一个包头的长度，如不够，则不进行拆包操作。

D 根据包头数据解析出里面代表包体长度的变量。

E 判断缓存区中除包头外的数据长度是否够一个包体的长度，如不够，则不进行拆包操作。

F 取出整个数据包，这里的“取”的意思是不光从缓冲区中拷贝出数据包，而且要把此数据包从缓存区中删除掉。删除的办法就是把此包后面的数据移动到缓冲区的起始地址。

这种方法有两个缺点：

1) 为每个连接动态分配一个缓冲区增大了内存的使用；

2) 有三个地方需要拷贝数据，一个地方是把数据存放在缓冲区，一个地方是把完整的数据包从缓冲区取出来，一个地方是把数据包从缓冲区中删除。这种拆包的改进方法会解决和完善部分缺点。

## 2、利用底层的缓冲区来进行拆包

由于 TCP 也维护了一个缓冲区，所以我们完全可以利用 TCP 的缓冲区来缓存我们的数据，这样一来就不需要为每一个连接分配一个缓冲区了。另一方面我们知道 recv 或者 wsarecv 都有一个参数，用来表示我们要接收多长长度的数据。利用这两个条件我们就可以对第一种方法进行





优化了。

对于阻塞 SOCKET 来说，我们可以利用一个循环来接收包头长度的数据，然后解析出代表包体长度的那个变量，再用一个循环来接收包体长度的数据。

### 33、请你说一下 tcp/ip 四层网络协议

考察点：计算机网络

参考回答：

TCP/IP 四层网络协议分别是应用层，网络层，传输层，数据链路层

### 34、手写代码：从网络日志中，提取出 date 字段，并排序。

考察点：计算机网络

参考回答：

首先在 grok 中要用%{DATESTAMP:date}或者自定义模式去匹配你的时间  
然后在 filter 里，

```
filter{  
  
  date{  
  
    match =>{"date","yyyy-MM-dd HH:mm:ss:SSS"}  
  
  }  
  
}
```

### 35、从打开浏览器 输入 url 到 到达服务器上项目中某一个 Controller 上，请你来描述一下这一串过程

考察点：计算机网络

参考回答：

这个过程中发生了网络通信，即利用 t c p / i p 协议簇进行网络通信，发送端由应用层往下走，接收端由数据链路层往上走，步骤如下：

1、浏览器输入 u r l ，其中 http 是协议



- 2、应用层 DNS 解析，返回对应的 ip 地址
- 3、应用层客户端发送 http 请求，
- 4、网络层 ip 查询 m a c 地址，
- 5、传输层 t c p 传输报文
- 6、数据到达数据链路层，此时客户端发送请求结束
- 7、服务器在数据链路层收到数据包，再层层下上直到应用层，
- 8、服务器响应请求，查找客户端请求的资源并返回响应报文

### 36、请你介绍下 session

考察点：计算机网络

参考回答：

Session：在 web 开发中，服务器可以为每个用户创建一个会话对象(session 对象)，默认情况下一个浏览器独占一个 session 对象，因此在需要保存用户数据时，服务器程序可以把用户数据写到用户浏览器独占的 session 中，当用户使用浏览器访问其他程序时，其他程序可以从用户的 session 中取出该用户的数据，为用户服务，其实现原理是服务器创建 session 出来后，会把 session 的 id 号，以 cookie 的形式回写给客户机，这样只要客户机的浏览器不关，再去访问服务器时，都会带着 session 的 id 号去，服务器发现客户机浏览器带 session id 过来了，就会使用内存中与之对应的 session 服务。

Session 和 cookie 的区别：

- 1、cookie 是把用户的数据写给用户浏览器
- 2、session 是把用户的数据写到用户独占的 session 中
- 3、session 对象由服务器创建，开发人员可以调用 request 对象的 getSession 方法得到 session 对象

### 37、请问你知道跨域吗，条件是什么，在 header 里需要加什么，有几种方案

考察点：计算机网络

参考回答：

什么是跨域？



浏览器从一个域名的网页去请求另一个域名的资源时，域名、端口、协议任一不同，都是跨域

跨域的几种方案：

1: 基于 script 标签实现跨域

3: 基于 jquery 跨域

4: 通过 iframe 来跨子域

**38、请你来回答一下，比如淘宝的搜索算法，输入关键词，会给出搜索出来的商品结果，对于这样的算法，如何评价它的好坏？**

考察点：搜索算法

参考回答：

淘宝的搜索算法：

1、目标性比较强，当然，这个相对而言，从 query 来看，用户对目标商品的认知度相对较强

2、短 query/符合 query 较多，传统搜索引擎里的 xxx 的商品这种 query 较少，当然，这与淘宝搜索的处理能力也有关系，用户对 query 进行分词的情况很常见

3、属性类 query 较为常见，如雪纺、鱼嘴等等表明用户特征的 query 较为常见

4、用户对结果的判断，基本上是价格敏感+信用敏感+销量敏感，其中销量敏感和信用敏感其实是一回事，来自知乎用户李杰。

对于这样的算法，笔者认为是很符合淘宝搜索要求的，能够更加精准，

**39、商品的种类有几十万种，在这种大数据的情况下，如何评价搜索算法的好坏？**

考察点：算法

参考回答：

在大数据时代，搜索算法最重要有三点，足够快，能够将用户所潜在需要的商品全部搜索出来，性能稳定。

## 2、数据库

**1、请问什么是数据库事物**

考点：数据库



参考回答：

数据库事务是数据库管理系统执行过程中的一个逻辑单位，由一个有限的数据库操作序列构成。一个数据库事务通常包含了一个序列的对数据库的读/写操作。它的存在包含有以下两个目的：

1. 为数据库操作序列提供了一个从失败中恢复到正常状态的方法，同时提供了数据库即使在异常状态下仍能保持一致性的方法。

2. 当多个应用程序在并发访问数据库时，可以在这些应用程序之间提供一个隔离方法，以防止彼此的操作互相干扰。

当事务被提交给了 DBMS（数据库管理系统），则 DBMS（数据库管理系统）需要确保该事务中的所有操作都成功完成且其结果被永久保存在数据库中，如果事务中有的操作没有成功完成，则事务中的所有操作都需要被回滚，回到事务执行前的状态；同时，该事务对数据库或者其他事务的执行无影响，所有的事务都好像在独立的运行。

数据库事务拥有以下四个特性，被称之为 ACID 特性：

原子性（Atomicity）：事务作为一个整体被执行，包含在其中的对数据库的操作要么全部被执行，要么都不执行。

一致性（Consistency）：事务应确保数据库的状态从一个一致状态转变为另一个一致状态。一致状态的含义是数据库中的数据应满足完整性约束。

隔离性（Isolation）：多个事务并发执行时，一个事务的执行不应影响其他事务的执行。

持久性（Durability）：已被提交的事务对数据库的修改应该永久保存在数据库中。

## 2、请你说一下数据库连接泄露的含义

考点：数据库

参考回答：

数据库连接泄露指的是如果在某次使用或者某段程序中没有正确地关闭 Connection、Statement 和 ResultSet 资源，那么每次执行都会留下一些没有关闭的连接，这些连接失去了引用而不能得到重新使用，因此就造成了数据库连接的泄漏。数据库连接的资源是宝贵而且是有限的，如果在某段使用频率很高的代码中出现这种泄漏，那么数据库连接资源将被耗尽，影响系统的正常运转。

## 3、请你解释一下数据库事务的含义

公司：酷家乐

参考回答：略

#### 4、请你写一下 mysql 删除语句

考点：数据库

参考回答：

1. drop 语句。可以用来删除数据库和表。

用 drop 语句来删除数据库：drop database db;

用 drop 语句来删除表：drop table tb;

2. delete 语句。用来删除表中的字段。

delete from tb where id=1;

如果 delete 语句中没有加入 where 就会把表中的所有记录全部删除：

3. 用 truncate 来删除表中的所有字段：

truncate table tb;

#### 5、请你说一下数据库 mysql 中 CHAR 和 VARCHAR 的区别

考点：数据库

参考回答：

##### 1、char(n) 类型

char 类型是定长的类型，即当定义的是 char(10)，输入的是“abc”这三个字符时，它们占的空间一样是 10 个字节，包括 7 个空字节。当输入的字符长度超过指定的数时，char 会截取超出的字符。而且，当存储 char 值时，MySQL 是自动删除输入字符串末尾的空格。char 是适合存储很短的、一般固定长度的字符串。例如，char 非常适合存储密码的 MD5 值，因为这是一个定长的值。对于非常短的列，char 比 varchar 在存储空间上也更有效率。取数据的时候，char 类型的要用 trim() 去掉多余的空格，而 varchar 是不需要。

##### 2、varchar(n) 类型

varchar(n) 类型用于存储可变长的，长度为 n 个字节的可变长度且非 Unicode 的字符数据。n 必须是介于 1 和 8000 之间的数值，存储大小为输入数据的字节的实际长度+1/2。比如 varchar(10)，然后输入 abc 三个字符，那么实际存储大小为 3 个字节。除此之外，varchar 还需要使用 1 或 2 个额外字节记录字符串的长度，如果列的最大长度小于等于 255 字节（是定义的最长长度，不是实际长度），则使用 1 个字节表示长度，否则使用 2 个字节来表示。取数据的时候，不需要去掉多余的空格。

**6、请你写两个 sql 语句，统计 XX 人数、选出课程编号不为 XX 的学生学号**

考点：数据库

参考回答：

```
select sClass 班级, count(*) 班级学生总人数,  
  
SELECT 学号,  
  
FROM S  
  
WHERE NOT EXISTS (SELECT *  
  
FROM SC  
  
WHERE SC.课程号 = 'XX'  
  
AND S.学号 = SC.学号) ;
```

**7、请你说一下 SQL 左连接以及使用场景**

考点：数据库

参考回答：

left join(左连接) 返回包括 LEFT OUTER 子句中指定的左表的所有行，而不仅仅是联接列所匹配的行。如果左表的某行在右表中没有匹配行，则在相关联的结果集行中右表的所有选择列表列均为空值。

使用场景：可以保持左表完整加入另一表中的数据。

**8、请你写一下 SQL 查询、更新的某一行语句**

考点：数据库

参考回答：

```
SELECT 要查询的数据类型 FROM 表名 WHERE 条件  
  
UPDATE 表名 SET 列名=更新的值 WHERE 条件
```

**9、请你写一下 SQL 语句的多表查询**

考点：数据库

参考回答：

例如：按照 department\_id 查询 employees(员工表)和 departments(部门表) 的信息。

```
SELECT ... FROM ... WHERE SELECT e.last_name,e.department_id,d.department_name  
FROM employees e,departments d where e.department_id = d.department_id
```

## 10、请你说说 redis

考点：数据库

参考回答：

redis 是一个 key-value 存储系统。和 Memcached 类似，它支持存储的 value 类型相对更多，包括 string(字符串)、list(链表)、set(集合)、zset(sorted set --有序集合)和 hash(哈希类型)。这些数据类型都支持 push/pop、add/remove 及取交集并集和差集及更丰富的操作，而且这些操作都是原子性的。在此基础上，redis 支持各种不同方式的排序。与 memcached 一样，为了保证效率，数据都是缓存在内存中。区别的是 redis 会周期性的把更新的数据写入磁盘或者把修改操作写入追加的记录文件，并且在此基础上实现了 master-slave(主从)同步。

## 11、请你写一些基本的 SQL 语句

考点：数据库

参考回答：

选择：select \* from table1 where 范围

插入：insert into table1(field1,field2) values(value1,value2)

删除：delete from table1 where 范围

更新：update table1 set field1=value1 where 范围

查找：select \* from table1 where field1 like '%value1%' ---like 的语法很精妙，查资料！

排序：select \* from table1 order by field1,field2 [desc]

总数：select count as totalcount from table1

求和：select sum(field1) as sumvalue from table1

平均：select avg(field1) as avgvalue from table1



最大: `select max(field1) as maxvalue from table1`

最小: `select min(field1) as minvalue from table1`

12、某个表格中有 10 条一模一样的数据，现在要删掉其中的 9 条，请你写一下 sql 语句

考点：数据库

参考回答：

```
delete * from table_name limit 9
```

13、某个表格存着 s\_name subject score 三个字段，比如某一行是 张三 数学 76，现在要选取出所有科目成绩都大于 80 分的学生名字，请写出 sql 语句

考点：数据库

参考回答：

```
select s_name from table_name where s_name not in (select s_name from table_name where score <80)
```

14、请你说一说数据库中的聚类查询

考点：数据库

参考回答：

聚集索引中键值的逻辑顺序决定了表中相应行的物理顺序。聚集索引确定表中数据的物理顺序。聚集索引类似于电话簿，后者按姓氏排列数据。由于聚集索引规定数据在表中的物理存储顺序，因此一个表只能包含一个聚集索引。但该索引可以包含多个列（组合索引），就像电话簿按姓氏和名字进行组织一样。聚集索引对于那些经常要搜索范围值的列特别有效。使用聚集索引找到包含第一个值的行后，便可以确保包含后续索引值的行在物理相邻。例如，如果应用程序执行的一个查询经常检索某一日期范围内的记录，则使用聚集索引可以迅速找到包含开始日期的行，然后检索表中所有相邻的行，直到到达结束日期。这样有助于提高此类查询的性能。同样，如果对从表中检索的数据进行排序时经常要用到某一列，则可以将该表在该列上聚集（物理排序），避免每次查询该列时都进行排序，从而节省成本。当索引值唯一时，使用聚集索引查找特定的行也很有效率。例如，使用唯一雇员 ID 列 emp\_id 查找特定雇员的最快速的方法，是在 emp\_id 列上创建聚集索引或 PRIMARY KEY 约束。

如果不创建索引，系统会自动创建一个隐含列作为表的聚集索引。





1. 创建表的时候指定主键（注意：SQL Sever 默认主键为聚集索引，也可以指定为非聚集索引，而 MySQL 里主键就是聚集索引）

```
create table t1(  
  
id int primary key,  
  
name nvarchar(255)  
  
)
```

2. 创建表后添加聚集索引

SQL Server:

```
create clustered index clustered_index on table_name(column_name)
```

MySQL:

```
alter table table_name add primary key(column_name)
```

#### 15、请问如果 mysql 中用户密码丢了怎么办，建一个数据库表，授权命令是什么

考察点：数据库

参考回答：

对于普通用户的密码丢失，直接用 root 超级管理员登录修改密码即可

若是 root 密码丢失，可通过 mysqlld\_saft 方式找回

1、停止 mysql: service mysqld stop

2、安全模式启动: mysql\_safe-skip-grant-tables&

3、无密码回车键登录: mysql -uroot -p;

4、重置密码: use mysql update user set password=password(“)where user=' root' and host=' localhost' ;flush privileges

5、正常启动: service mysql restart

6、再使用 mysqladmin: mysqladmin password '123456'

Mysql 创建数据库



```
Create database demodb default character set utf8 collate utf8_general_ci;
```

授权

```
Grant all privileges on demodb. * [用户名称]@' %'
```

立即启动修改

```
Flush privileges
```

#### 16、写出 sql 语句：数据库统计总成绩取前十名的学生

考察点：sql

参考回答：

```
SELECT * FROM (  
  
select T.*,ROW_NUMBER()OVER(PARTITION BY 班级 order by 成绩 desc) RN  
  
FROM T  
  
)WHERE RN<=10
```

#### 17、请你说一下数据库事务、主键与外键的区别？

考察点：数据库

参考回答：

数据库的事务：事务即用户定义的一个数据库操作序列，这些操作要么全做要全不做，是一个不可分割的工作单位，它具有四个特性，ACID，原子性，一致性，隔离性，持续性

主键和外键的区别：

1. 主键是能确定一条记录的唯一标识，比如，一条记录包括身份证号，姓名，年龄。

身份证号是唯一能确定你这个人的，其他都可能重复，所以，身份证号是主键。

2. 外键用于与另一张表的关联。是能确定另一张表记录的字段，用于保持数据的一致性。

#### 18、请问对缓存技术了解吗

考察点：缓存技术

参考回答：



Redis 可以实现缓存机制，Redis 是一个 key-value 存储系统。和 Memcached 类似，它支持存储的 value 类型相对更多，包括 string(字符串)、list(链表)、set(集合)和 zset(有序集合)。这些数据类型都支持 push/pop、add/remove 及取交集并集和差集及更丰富的操作，而且这些操作都是原子性的。在此基础上，redis 支持各种不同方式的排序。与 memcached 一样，为了保证效率，数据都是缓存在内存中。区别的是 redis 会周期性的把更新的数据写入磁盘或者把修改操作写入追加的记录文件，并且在此基础上实现了 master-slave(主从)同步，当前 Redis 的应用已经非常广泛，国内像新浪、淘宝，国外像 Flickr、Github 等均在使用 Redis 的缓存服务。

#### Redis 工作方式分析

Redis 作为一个高性能的 key-value 数据库具有以下特征：

- 1、多样的数据模型
- 2、持久化
- 3、主从同步

Redis 支持丰富的数据类型，最为常用的数据类型主要由五种：String、Hash、List、Set 和 Sorted Set。Redis 通常将数据存储在内存中，或被配置为使用虚拟内存。Redis 有一个很重要的特点就是它可以实现持久化数据，通过两种方式可以实现数据持久化：使用 RDB 快照的方式，将内存中的数据不断写入磁盘；或使用类似 MySQL 的 AOF 日志方式，记录每次更新的日志。前者性能较高，但是可能会引起一定程度的数据丢失；后者相反。Redis 支持将数据同步到多台从数据库上，这种特性对提高读取性能非常有益。

#### 19、请问 count 和 sum 的区别，以及 count(\*) 和 count(列名)的区别

考察点：数据库

参考回答：

Count 和 sum 区别：求和用累加 sum()，求行的个数用累计 count

Count (\*) 包括了所有的列，在统计结果的时候不会忽略列值为 null

Count(列名) 只包括列名那一项，会忽略列值为空的计数

#### 20、请问你数据库是用的 MySQL 吗？平常数据库的语句都是怎么写的？join 作用，想删除一行怎么做

考察点：数据库

参考回答：



Mysql 和 oracle 都用过，mysql 为主，join 作用是连接两个表，假设有 2 个表——Student 表和 SC 表（选课表）：

内连接（自然连接）：当使用内连接时，如果 Student 中某些学生没有选课，则在 SC 中没有相应元组。最终查询结果舍弃了这些学生的信息

外连接：如果想以 Student 表为主体列出每个学生的基本情况及其选课情况。即使某个学生没有选课，依然在查询结果中显示（SC 表的属性上填充值）。就需要使用外连接

例子：

//内连接：查询每个学生及其选修课程的情况（没选课的学生不会列出）

```
SELECT Student.*, SC.*
```

```
FROM Student , SC
```

```
WHERE Student.Sno=SC.Sno;
```

//外连接：查询每个学生及其选修课程的情况（没选课的学生也会列出）

```
SELECT Student.*, SC.*
```

```
FROM Student LEFT JOIN SC ON(Student.Sno=SC.Sno);
```

## 21、请问如何对数据库作优化

考察点：数据库

参考回答：

1、调整数据结构的设计，对于经常访问的数据库表建立索引

2、调整 SQL 语句，ORACLE 公司推荐使用 ORACLE 语句优化器（Oracle Optimizer）和行锁管理器（row-level manager）来调整优化 SQL 语句。

3、调整服务器内存分配。内存分配是在信息系统运行过程中优化配置的，数据库管理员可以根据数据库运行状况调整数据库系统全局区（SGA 区）的数据缓冲区、日志缓冲区和共享池的大小；还可以调整程序全局区（PGA 区）的大小。

4、调整硬盘 I / O，DBA 可以将组成同一个表空间的数据文件放在不同的硬盘上，做到硬盘之间 I / O 负载均衡。

## 22、请问什么是幻读



考察点：数据库

参考回答：

脏读就是指当一个事务正在访问数据，并且对数据进行了修改，但是还没有来得及提交到数据库中，这时，另一个事务也访问这个数据，然后使用了这个数据

### 23、请你说一下 MyBatis 有什么优势，他如何做事务管理

考察点：数据库

参考回答：

MyBatis 优点：

1. 易于上手和掌握
2. sql 写在 xml 里，便于统一管理和优化。
3. 解除 sql 与程序代码的耦合。
4. 提供映射标签，支持对象与数据库的 orm 字段关系映射
5. 提供对象关系映射标签，支持对象关系组建维护
6. 提供 xml 标签，支持编写动态 sql。

Mybatis 管理事务是分为两种方式：

(1)使用 JDBC 的事务管理机制，就是利用 java.sql.Connection 对象完成对事务的提交

(2)使用 MANAGED 的事务管理机制，这种机制 mybatis 自身不会去实现事务管理，而是让程序的容器（JBoss, WebLogic）来实现对事务的管理

### 24、请你说一下事务的隔离级别，以及你一般使用的事务是哪种

考察点：数据库

参考回答：

事务的隔离性及在同一时间只允许一个事务请求同一数据，不同事物之间彼此没有任何干扰，

事务隔离级别如下：

事务隔离级别	脏读	不可重复读	幻读
读未提交 (read-uncommitted)	是	是	是
不可重复读 (read-committed)	否	是	是
可重复读 (repeatable-read)	否	否	是
串行化 (serializable)	否	否	否

### 3、操作系统

#### 1、请你说一下多进程、多线程，操作系统层面的差别和联系

考点：操作系统

参考回答：

**进程：**进程是一个具有一定独立功能的程序在一个数据集上的一次动态执行的过程，是操作系统进行资源分配和调度的一个独立单位，是应用程序运行的载体。进程是一种抽象的概念，从来没有统一的标准定义。进程一般由程序、数据集和进程控制块三部分组成。程序用于描述进程要完成的功能，是控制进程执行的指令集；数据集是程序在执行时所需要的数据和工作区；程序控制块 (Program Control Block，简称 PCB)，包含进程的描述信息和控制信息，是进程存在的唯一标志。

**线程：**在早期的操作系统中并没有线程的概念，进程是能拥有资源和独立运行的最小单位，也是程序执行的最小单位。任务调度采用的是时间片轮转的抢占式调度方式，而进程是任务调度的最小单位，每个进程有各自独立的一块内存，使得各个进程之间内存地址相互隔离。后来，随着计算机的发展，对 CPU 的要求越来越高，进程之间的切换开销较大，已经无法满足越来越复杂的程序的要求了。于是就发明了线程，线程是程序执行中一个单一的顺序控制流程，是程序执行流的最小单元，是处理器调度和分派的基本单位。一个进程可以有一个或多个线程，各个线程之间共享程序的内存空间 (也就是所在进程的内存空间)。一个标准的线程由线程 ID、当前指令指针 (PC)、寄存器和堆栈组成。而进程由内存空间 (代码、数据、进程空间、打开的文件) 和一个或多个线程组成。

**差别：**1. 线程是程序执行的最小单位，而进程是操作系统分配资源的最小单位；2. 一个进程由一个或多个线程组成，线程是一个进程中代码的不同执行路线；3. 进程之间相互独立，但同一



进程下的各个线程之间共享程序的内存空间(包括代码段、数据集、堆等)及一些进程级的资源(如打开文件和信号)，某进程内的线程在其它进程不可见；4. 调度和切换：线程上下文切换比进程上下文切换要快得多。

联系：原则上一个 CPU 只能分配给一个进程，以便运行这个进程。通常使用的计算机中只有一个 CPU，同时运行多个进程，就必须使用并发技术。通常采用时间片轮转进程调度算法，在操作系统的管理下，所有正在运行的进程轮流使用 CPU，每个进程允许占用 CPU 的时间非常短(比如 10 毫秒)，这样用户根本感觉不出来 CPU 是在轮流为多个进程服务，就好像所有的进程都在不间断地运行一样。但实际上在任何一个时间内有且仅有一个进程占有 CPU。如果一台计算机有多个 CPU，情况就不同了，如果进程数小于 CPU 数，则不同的进程可以分配给不同的 CPU 来运行，这样，多个进程就是真正同时运行的，这便是并行。但如果进程数大于 CPU 数，则仍然需要使用并发技术。在 Windows 中，进行 CPU 分配是以线程为单位的，一个进程可能由多个线程组成。操作系统将 CPU 的时间片分配给多个线程，每个线程在操作系统指定的时间片内完成(注意，这里的多个线程是分属于不同进程的)。操作系统不断的从一个线程的执行切换到另一个线程的执行，如此往复，宏观上看来，就好像是多个线程在一起执行。由于这多个线程分属于不同的进程，就好像是多个进程在同时执行，这样就实现了多任务。总线程数 $\leq$ CPU 数量时并行运行，总线程数 $>$ CPU 数量时并发运行。并行运行的效率显然高于并发运行，所以在多 CPU 的计算机中，多任务的效率比较高。但是，如果在多 CPU 计算机中只运行一个进程(线程)，就不能发挥多 CPU 的优势。

## 2、请你说一下线程通信的方法、线程的五种状态

考点：操作系统

参考回答：

线程通信的方法：

- ①同步：多个线程通过 synchronized 关键字这种方式来实现线程间的通信。
- ②while 轮询的方式
- ③wait/notify 机制
- ④管道通信就是使用 `java.io.PipedInputStream` 和 `java.io.PipedOutputStream` 进行通信

线程的五种状态：

- 1. 新建(NEW)：新创建了一个线程对象。
- 2. 可运行(RUNNABLE)：线程对象创建后，其他线程(比如main线程)调用了该对象的 `start()` 方法。该状态的线程位于可运行线程池中，等待被线程调度选中，获取 cpu 的使用权。
- 3. 运行(RUNNING)：可运行状态(runnable)的线程获得了 cpu 时间片(timeslice)，执行程序代码。





4. 阻塞(BLOCKED): 阻塞状态是指线程因为某种原因放弃了 cpu 使用权, 也即让出了 cpu timeslice, 暂时停止运行。直到线程进入可运行(runnable)状态, 才有机会再次获得 cpu timeslice 转到运行(running)状态。阻塞的情况分三种:

(一). 等待阻塞: 运行(running)的线程执行 o.wait() 方法, JVM 会把该线程放入等待队列(waiting queue)中。

(二). 同步阻塞: 运行(running)的线程在获取对象的同步锁时, 若该同步锁被别的线程占用, 则 JVM 会把该线程放入锁池(lock pool)中。

(三). 其他阻塞: 运行(running)的线程执行 Thread.sleep(long ms)或 t.join() 方法, 或者发出了 I/O 请求时, JVM 会把该线程置为阻塞状态。当 sleep() 状态超时、join() 等待线程终止或者超时、或者 I/O 处理完毕时, 线程重新转入可运行(runnable)状态。

5. 死亡(DEAD): 线程 run()、main() 方法执行结束, 或者因异常退出了 run() 方法, 则该线程结束生命周期。死亡的线程不可再次复生。

### 3、请你说一下虚拟内存

考点: 操作系统

参考回答:

虚拟内存是计算机系统内存管理的一种技术。它使得应用程序认为它拥有连续的可用的内存(一个连续完整的地址空间), 而实际上, 它通常是被分隔成多个物理内存碎片, 还有部分暂时存储在外部磁盘存储器上, 在需要进行数据交换

### 4、请你说一下线程的同步和互斥以及应用常见

考点: 操作系统

参考回答:

互斥: 指在某一时刻指允许一个进程运行其中的程序片, 具有排他性和唯一性。

对于线程 A 和线程 B 来讲, 在同一时刻, 只允许一个线程对临界资源进行操作, 即当 A 进入临界区对资源操作时, B 就必须等待; 当 A 执行完, 退出临界区后, B 才能对临界资源进行操作。

同步: 指的是在互斥的基础上, 实现进程之间的有序访问。假设现有线程 A 和线程 B, 线程 A 需要往缓冲区写数据, 线程 B 需要从缓冲区读数据, 但他们之间存在一种制约关系, 即当线程 A 写的时候, B 不能来拿数据; B 在拿数据的时候 A 不能往缓冲区写, 也就是说, 只有当 A 写完数据(或 B 取走数据), B 才能来读数据(或 A 才能往里写数据)。这种关系就是一种线程的同步关系。

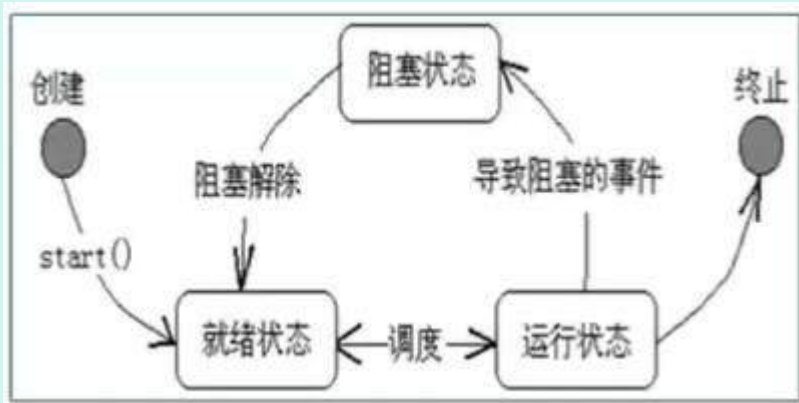
应用常见: 多线程编程中, 难免会遇到多个线程同时访问临界资源的问题, 如果不对其加以保护, 那么结果肯定是不如预期的, 因此需要线程同步与互斥。



## 5、请你说一下线程的五种状态以及转换

考点：操作系统

参考回答：



### 1、新生状态

在程序中用构造方法（new 操作符）创建一个新线程时，如 `new Thread(r)`，该线程就是创建状态，此时它已经有了相应的内存空间和其它资源，但是还没有开始执行。

### 2、就绪状态

新建线程对象后，调用该线程的 `start()` 方法就可以启动线程。当线程启动时，线程进入就绪状态（runnable）。由于还没有分配 CPU，线程将进入线程队列排队，等待 CPU 服务，这表明它已经具备了运行条件。当系统挑选一个等待执行的 Thread 对象后，它就会从等待执行状态进入执行状态。系统挑选的动作称之为“CPU 调度”。一旦获得 CPU 线程就进入运行状态并自动调用自己的 `run` 方法。

### 3、运行状态

当就绪状态的线程被调用并获得处理器资源时，线程就进入了运行状态。此时，自动调用该线程对象的 `run()` 方法。`run()` 方法定义了该线程的操作和功能。运行状态中的线程执行自己的 `run` 方法中代码。直到调用其他方法或者发生阻塞而终止。

### 4、阻塞状态

一个正在执行的线程在某些特殊情况下，如被人为挂起或需要执行耗时的输入输出操作时，`suspend()`、`wait()` 等方法，线程都将进入堵塞状态。堵塞时，线程不能进入排队队列，只有当引起堵塞的原因被消除后，线程转入就绪状态。重将让出 CPU 并暂时中止自己的执行，进入堵塞状态。在可执行状态下，如果调用 `sleep()`、新到就绪队列中排队等待，这时被 CPU 调度选中后会从原来停止的位置开始继续执行。

### 5、死亡状态



线程调用 `stop()` 方法、`destory()` 方法或 `run()` 方法执行结束后，线程即处于死亡状态。处于死亡状态的线程不具有继续运行的能力。

## 6、请你说一说消息队列、信号量的实现方式

考点：操作系统

参考回答：

消息队列是消息的链接表，存储在内核中，由消息队列 ID 来标识。每个队列都有一个 `msgid_ds` 结构与其相关联：

```
struct msgid_ds
{
    struct ipc_perm msg_perm;

    msgqnum_t msg_qnum; /* # of messages on queue */

    msglen_t msg_qbytes; /* max # of bytes on queue */

    pid_t msg_lspid; /* pid of last msgsnd() */

    pid_t msg_lrpid; /* pid of last msgrcv() */

    time_t msg_stime; /* last-msgsnd() time */

    time_t msg_rtime; /* last-msgrcv() time */

    time_t msg_ctime; /* last-change time */

    ...
};
```

此结构定义了队列的当前状态。`msgget` 用于创建一个新队列或打开一个现有队列，`msgsnd` 将消息添加到队列的尾端（每个消息包括一个长整型类型字段，一个非负的长度，实际的数据长度），`msgrcv` 用于从队列中取消息（并不一定要以先进先出次序取消息，可以按消息的类型字段取消息）。

信号量是一个计数器，用于为多个进程提供对共享对象的访问。为了正确地实现信号量，信号量的测试及加减 1 操作应当是原子操作，为此，信号量通常是在内核中实现的。

常用的信号形式是二元信号量（binary semaphore）。它控制单个资源，其初始值为 1。但是，一般而言，信号量的初值也可以是任意一个正值，表明有多少个共享单位可供共享。



内核为每个信号量集合维护着一个 `semid_ds` 结构：

```
struct semid_ds

{

    struct ipc_perm sem_perm;

    unsigned short sem_nsems; /* # of semaphores in set */

    time_t sem_otime; /* last-semop() time */

    time_t sem_ctime; /* last-change time */

    ...

};
```

每个信号量由一个无名结构体表示，至少包含下列成员：

```
struct

{

    unsigned short semval; /* semaphore value, always >= 0 */

    pid_t sempid; /* pid for last operation */

    unsigned short semncnt; /* # processes awaiting semval > curval */

    unsigned short semzcnt; /* # processes awaiting semval == 0 */

    ...

};
```

## 7、请你说一下进程和线程的区别

考点：操作系统

参考回答：

进程：是具有一定独立功能的程序、它是系统进行资源分配和调度的一个独立单位，重点在系统调度和单独的单位，也就是说进程是可以独立运行的一段程序。

线程：是进程的一个实体，是 CPU 调度和分派的基本单位，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，在运行时，只是暂用一些计数器、寄存器和栈。线程有自己的堆栈和局部变量，但线程之间没有单独的地址空间。



一个程序至少有一个进程，一个进程至少有一个线程。

## 8、请你说一下死锁的概念、原因、解决方法

考点：java 多线程

参考回答：

1、死锁是指在一组进程中的各个进程均占有不会释放的资源，但因互相申请被其他进程所站用不会释放的资源而处于的一种永久等待状态。死锁的四个必要条件：

- 互斥条件(Mutual exclusion)：资源不能被共享，只能由一个进程使用。
- 请求与保持条件(Hold and wait)：已经得到资源的进程可以再次申请新的资源。
- 非剥夺条件(No pre-emption)：已经分配的资源不能从相应的进程中被强制地剥夺。
- 循环等待条件(Circular wait)：系统中若干进程组成环路，该环路中每个进程都在等待相邻进程正占用的资源。

java 中产生死锁可能性的最根本原因是：1) 是多个线程涉及到多个锁，这些锁存在着交叉，所以可能会导致了一个锁依赖的闭环；2) 默认的锁申请操作是阻塞的。

如，线程在获得一个锁 L1 的情况下再去申请另外一个锁 L2，也就是锁 L1 想要包含了锁 L2，在获得了锁 L1，并且没有释放锁 L1 的情况下，又去申请获得锁 L2，这个是产生死锁的最根本原因。

2、避免死锁：

- 方案一：破坏死锁的循环等待条件。
- 方法二：破坏死锁的请求与保持条件，使用 lock 的特性，为获取锁操作设置超时时间。这样不会死锁（至少不会无尽的死锁）
- 方法三：设置一个条件遍历与一个锁关联。该方法只用一把锁，没有 chopstick 类，将竞争从对筷子的争夺转换成了对状态的判断。仅当左右邻座都没有进餐时才可以进餐。提升了并发度。

## 9、请你说一下多线程

考点：操作系统

参考回答：

最开始，线程只是用于分配单个处理器的处理时间的一种工具。但假如操作系统本身支持多个处理器，那么每个线程都可分配给一个不同的处理器，真正进入“并行运算”状态。从程序设



计语言的角度看，多线程操作最有价值的特性之一就是程序员不必关心到底使用了多少个处理器。程序在逻辑意义上被分割为多个线程；假如机器本身安装了多个处理器，那么程序会运行得更快，毋需作出任何特殊的调校。根据前面的论述，大家可能感觉线程处理非常简单。但必须注意一个问题：共享资源！如果有多个线程同时运行，而且它们试图访问相同的资源，就会遇到一个问题。举个例子来说，两个线程不能将信息同时发送给一台打印机。为解决这个问题，对那些可共享的资源来说（比如打印机），它们在使用期间必须进入锁定状态。所以一个线程可将资源锁定，在完成了它的任务后，再解开（释放）这个锁，使其他线程可以接着使用同样的资源。

多线程是为了同步完成多项任务，不是为了提高运行效率，而是为了提高资源使用效率来提高系统的效率。线程是在同一时间需要完成多项任务的时候实现的。

一个采用了多线程技术的应用程序可以更好地利用系统资源。其主要优势在于充分利用了 CPU 的空闲时间片，可以用尽可能少的时间来对用户的要求做出响应，使得进程的整体运行效率得到较大提高，同时增强了应用程序的灵活性。更为重要的是，由于同一进程的所有线程是共享同一内存，所以不需要特殊的数据传送机制，不需要建立共享存储区或共享文件，从而使得不同任务之间的协调操作与运行、数据的交互、资源的分配等问题更加易于解决。

#### 10、请你说一下线程之间通信的手段

考点：操作系统

参考回答：

使用全局变量

主要由于多个线程可能更改全局变量，因此全局变量最好声明为 `volatile`

使用消息实现通信

在 Windows 程序设计中，每一个线程都可以拥有自己的消息队列（UI 线程默认自带消息队列和消息循环，工作线程需要手动实现消息循环），因此可以采用消息进行线程间通信 `SendMessage`, `PostMessage`。

使用事件 `CEvent` 类实现线程间通信

Event 对象有两种状态：有信号和无信号，线程可以监视处于有信号状态的事件，以便在适当的时候执行对事件的操作。

#### 11、请你说一下死锁

考点：多线程

参考回答：

1、死锁是指在一组进程中的各个进程均占有不会释放的资源，但因互相申请被其他进程所站用不会释放的资源而处于的一种永久等待状态。死锁的四个必要条件：

互斥条件(Mutual exclusion)：资源不能被共享，只能由一个进程使用。

请求与保持条件(Hold and wait)：已经得到资源的进程可以再次申请新的资源。

非剥夺条件(No pre-emption)：已经分配的资源不能从相应的进程中被强制地剥夺。

循环等待条件(Circular wait)：系统中若干进程组成环路，该环路中每个进程都在等待相邻进程正占用的资源。

java 中产生死锁可能性的最根本原因是：1) 是多个线程涉及到多个锁，这些锁存在着交叉，所以可能会导致了一个锁依赖的闭环；2) 默认的锁申请操作是阻塞的。

如，线程在获得一个锁 L1 的情况下再去申请另外一个锁 L2，也就是锁 L1 想要包含了锁 L2，在获得了锁 L1，并且没有释放锁 L1 的情况下，又去申请获得锁 L2，这个是产生死锁的最根本原因。

2、避免死锁：

- 方案一：破坏死锁的循环等待条件。
- 方法二：破坏死锁的请求与保持条件，使用 lock 的特性，为获取锁操作设置超时时间。这样不会死锁（至少不会无尽的死锁）
- 方法三：设置一个条件遍历与一个锁关联。该方法只用一把锁，没有 chopstick 类，将竞争从对筷子的争夺转换成了对状态的判断。仅当左右邻座都没有进餐时才可以进餐。提升了并发度。

## 12、请你回答一下进程同步的方法

考点：操作系统

参考回答：

1、临界区(Critical Section)：通过对多线程的串行化来访问公共资源或一段代码，速度快，适合控制数据访问。

优点：保证在某一时刻只有一个线程能访问数据的简便办法

缺点：虽然临界区同步速度很快，但却只能用来同步本进程内的线程，而不可用来同步多个进程中的线程。

2、互斥量(Mutex)：为协调共同对一个共享资源的单独访问而设计的。

互斥量跟临界区很相似，比临界区复杂，互斥对象只有一个，只有拥有互斥对象的线程才具有访问资源的权限。





优点：使用互斥不仅仅能够在同一应用程序不同线程中实现资源的安全共享，而且可以在不同应用程序的线程之间实现对资源的安全共享。

缺点：①互斥量是可以命名的，也就是说它可以跨越进程使用，所以创建互斥量需要的资源更多，所以如果只为了在进程内部是用的话使用临界区会带来速度上的优势并能够减少资源占用量。因为互斥量是跨进程的互斥量一旦被创建，就可以通过名字打开它。

②通过互斥量可以指定资源被独占的方式使用，但如果有下面一种情况通过互斥量就无法处理，比如现在一位用户购买了一份三个并发访问许可的数据库系统，可以根据用户购买的访问许可数量来决定有多少个线程/进程能同时进行数据库操作，这时候如果利用互斥量就没有办法完成这个要求，信号量对象可以说是一种资源计数器。

3、信号量（Semaphore）：为控制一个具有有限数量用户资源而设计。它允许多个线程在同一时刻访问同一资源，但是需要限制在同一时刻访问此资源的最大线程数目。互斥量是信号量的一种特殊情况，当信号量的最大资源数=1 就是互斥量了。

优点：适用于对 Socket（套接字）程序中线程的同步。（例如，网络上的 HTTP 服务器要对同一时间内访问同一页面的用户数加以限制，只有不大于设定的最大用户数目的线程能够进行访问，而其他的访问企图则被挂起，只有在有用户退出对此页面的访问后才有可能进入。）

缺点：①信号量机制必须有公共内存，不能用于分布式操作系统，这是它最大的弱点；

②信号量机制功能强大，但使用时对信号量的操作分散，而且难以控制，读写和维护都很困难，加重了程序员的编码负担；

③核心操作 P-V 分散在各用户程序的代码中，不易控制和管理，一旦错误，后果严重，且不易发现和纠正。

4、事件（Event）：用来通知线程有一些事件已发生，从而启动后继任务的开始。

优点：事件对象通过通知操作的方式来保持线程的同步，并且可以实现不同进程中的线程同步操作。

### 13、请问进程线程的区别，进程间怎么相互通信，什么是多线程，什么是并发

考察点：操作系统，进程线程，并发

进程和线程的区别有以下几点

1、进程是资源分配的最小单位，线程是程序执行的最小单位

2、进程有自己独立地址空间，每启动一个进程，系统就会为它分配地址空间，建立数据表来维护代码段，堆栈段，数据段，而线程是共享进程中的数据的，使用相同的地址空间，但是 CPU 切换一个线程的花费远比进程要小，

3、线程之间通信方式更方便，同一进程下的线程共享全局变量等数据，而进程之间的通信方式需要以通信的方式进行，



4、多线程程序中只要有一个线程死掉了，整个进程也死掉了，而一个进程死掉了，并不会对另一个进程造成影响，因为进程有自己独立的地址空间

进程间的通信方式：

- 1、无名管道通信，数据只能单向流动，只能在具有亲缘的进程间使用
- 2、高级管道通信，将领一个程序当作一个新的进程在当前程序中启动，则他算是当前进程的子进程
- 3、有名管道通信，允许无亲缘关系进程间的通信
- 4、消息队列通信，消息队列是由消息的链表，存放在内核中并由消息队列标识符标识，消息队列克服了信息传递信息少等缺点
- 5、信号量通信，信号量用于控制多个进程对共享资源的访问
- 6、信号通信，用于通知接受进程某个事件已经发生
- 7、共享内存通信，共享内存映射一段能被其他进程所访问的内存，往往与其他通信机制配合使用，来实现进程间的同步和通信
- 8、套接字通信，他用于不同机器之间的进程通信

什么是多线程

多线程就是指一个进程中同时有多个执行路径正在执行

并发指在操作系统中，一个时间段中有几个程序都已处于已启动运行到运行完毕之间，且这几个程序都是在同一个处理机上面，但任意时刻点上只有一个程序在处理机上运行。

## 十三、智力题

1、5 只猫 五分钟捉 5 只老鼠 请问 100 分钟捉 100 只老鼠需要多少只猫？

考点：智力题

参考回答：

$$5 * X * 5M = 5Y$$

$$K * X * 100M = 100Y$$

$$K = 5$$

需要 5 只猫





2、圆桌，两个人，轮流放硬币，不能重叠，半径为1，某一方不能放下去，则为输。问先手赢 后手赢。

考点：智力题

参考回答：

先手赢。圆桌对称，先手放一个，后手都能找到对称的位置放，但是除了圆心。

3、逻辑题：3升的杯子一个，5升的杯子一个，杯子不规则形状 问怎么得到4升的水 水无限多

考点：逻辑

参考回答：

- 1、将3升的装满倒入5升的；
- 2、再一次将3升的转满，倒入5升的，把5升装满；
- 3、3升杯里剩下的就是1升水；
- 4、倒掉5升的，把1升水倒入5升杯；
- 5、第三次加满3升杯，倒入5升杯，得到4升水。

4、晚上有四个人过桥，一次只能过两个人，但是只有一只手电筒，四个人过桥时间分别是1，2，5，8，求最短过桥时间

考点：逻辑

参考回答：

假设这四人依次是甲乙丙丁：首先甲和乙过桥，甲带手电筒回来；然后丙和丁过桥，由乙带手电筒回来；最后甲再和乙一起过桥。所以最少用时间是  $2+1+8+2+2=15$ （分钟）

5、两个容积分别为5升和6升的桶，最后如何只装3升？

考点：逻辑

参考回答：



第一步：先取来 6 升水，倒进 5 升桶的水桶里，即得到 6 升桶里余下的 1 升水；

第二步：把 5L 桶清掉，把取到的 1 升水放进 5 升的水桶里保留不动，然后再取 6 升水，倒进 5 升的水桶里，6 升的桶得到的是 2 升水，把 5L 桶清掉，存放这 2 升水；

第三步：5 升水桶有 2 升水。再取 6 升水，倒进 5 升水桶里，原有 2L 升+3 升=5 升，这时 6 升-3 升=3 升，6 升里余下的就是 3 升水了。

## 6、有十张扑克牌，每次可以只出一张，也可以只出两张，要出完有多少种出法

考点：动态规划

参考回答：

还有一张牌就出完 10 张，可能的情况有两种，从 9 到 10 和从 8 到 10，已知了从 0 到 9 的出法有 N 种，如果再知道从 0 到 8 的出法有 P 种，那么从 0 到 10 级的出法就是 N+P，那么可得出：

$$F(9)=N;$$

$$F(8)=P;$$

$$F(10)=N+P;$$

$$F(10)=F(9)+F(8);$$

又有：

$$F(1)=1;$$

$$F(2)=2;$$

最后推出：F(10)=89

## 7、井盖为什么是圆的

参考回答：经典面试题

井道大都是圆形的，所以井盖就做成圆形的。

很多井道都是圆形的，所以井盖自然也就是圆形的了。那为什么井大都是圆形的呢？因为建筑学和土木工程学中，圆形通道最有利于保持土壤的压力。

圆的受力更均匀不容易碎裂和塌陷



圆形井盖受力后，会向四周扩散压力，由于扩散均匀不容易碎裂和塌陷。  
矩形的井盖由于受力不均匀，导致碎裂的几率远大于圆形。所以通过耐用性方面考虑还是圆形井盖合适。

圆形井盖从任何方向都不会掉落井下，也方便操作

矩形对角线的长度都大于矩形的长和宽。所以在对角线方向把井盖竖起来就容易掉落井下。

相对节省生成材料成本

相对于矩形或者正方形，矩形内切圆形的面积最小，生成用的材料也更少。

## 8、用 5L 和 6L 的桶，没有刻度，怎么量出 3L 的水

考察点：智力题

参考回答：

1. 先将 6L 杯装满水放，然后将 6L 杯里的水倒满 5L 杯，此时 5L 杯水满为 5L 水，6L 杯剩 1L 水；
2. 清空 5L 杯，将 6L 杯的 1L 水放到 5L 杯，此时 5L 杯有 1L 水，6L 杯是空的；
3. 将 6L 杯装满水，然后往 5L 杯倒，刚好倒满 5L 杯（原有 1L+6L 杯倒过来的 4L=5L）的时候，6L 杯里还有 2L 水。
4. 清空 5L 杯里的水，将 6L 杯的 2L 水倒到 5L 杯，此时 5L 杯里有 2L 水，6L 杯为空；
5. 将 6L 杯装满水，然后往 5L 杯里倒，刚好倒满 5L 杯（5L 杯原来有 2L+6L 杯倒过来的 3L=5L 刚好满）时，6L 杯里剩下的水就是 3L。

9、从前有座山，山脚下有 5 个海盗抢到了 100 枚金币，每一颗都一样大小和价值。他们决定通过抽签的方式，按顺序提出分配方案决定金币的归属。

首先，由 1 号提出方案，5 个人进行表决，半数人以上（包括半数）同意时，方案通过，否则他将被扔入大海喂鲨鱼，剩余海盗继续按顺序提出方案，依次类推。

假设每个海盗都是足够理性及机智，会考虑到利害及利益最大化问题，那么，1 号海盗提出怎样的分配方案才能顺利通过考验并拿到可能性内最多的金币呢？

考察点：智力题

参考回答：

海盗分金推理过程：

从后向前推，如果只剩 4 号和 5 号的话，5 号一定会投反对票让 4 号喂鲨鱼，以独吞全部金币。所以，4 号唯有支持 3 号才能保命。3 号知道这一点，就会提 (100, 0, 0) 的分配方案，对 4 号、5 号一毛不拔而将全部金币归为己有，因为他知道 4 号一无所获也会投赞成票，再加上自己一票他的方案即可通过。不过，2 号推知到 3 号的方案，就会提出 (98, 0, 1, 1) 的方案，即放弃 3 号，而给予 4 号和 5 号各一枚金币。由于该方案对于 4 号和 5 号来说比在 3 号分配时更为有



利，他们将支持他而不希望由 3 号来分配。这样，2 号将拿走 98 枚金币。不过，2 号的方案会被 1 号所洞悉，1 号并将提出 (97, 0, 1, 2, 0) 或 (97, 0, 1, 0, 2) 的方案，即放弃 2 号，而给 3 号一枚金币，同时给 4 号 (或 5 号) 2 枚金币。由于 1 号的这一方案对于 3 号和 4 号 (或 5 号) 来说，相比 2 号分配时更优，他们将投 1 号的赞成票，再加上 1 号自己的票，1 号的方案可获通过，97 枚金币可轻松落入囊中。

#### 10、烧绳子得到 15 分钟

考察点：智力题

参考回答：

烧一根不均匀的绳子，从头烧到尾总共需要 1 个小时，问如何用烧绳子的方法来确定 15 分钟？

答案：烧两根绳子，1 第一根两头一起点，第二根点一头  
2 第一根烧完后点第二根另一端，从此时起计时，至第二根烧完，即 15min（哎不知道出这种题是要干什么）

#### 11、两个盲人各买了一白一黑两双袜子，不小心弄混了，问他们自己怎么分成刚好每人一白一黑

考察点：智力题

参考回答：

额因为袜子一双都是连在一起的，所以把两双袜子扯开，互相给对方一只，即可

#### 12、一个圆桌，两个人往上放硬币，只能平铺不能重合，最后一个放的人胜利（接下来硬币无处可放了），问先放的赢还是后放的赢。

考察点：智力题

参考回答：

这道题会围棋的人一般都知道，只要先手把硬币放在圆桌正中心，随后第二个人无论把硬币放在哪，第一个人都把硬币放在对称的位置，即可，是先放的赢

## 十四、职业规划

#### 1、为什么要投测试工程师？

考察点：职业规划

参考回答：

测试工程师前景光明，目前 IT 行业对于真正有技术的测试工程师而不是测试员的需求很高，并且对测试行业很感兴趣，本身性格也比较细心谨慎，符合测试工程师的素质要求。

## 十五、技术发展

- 1、对于大数据有什么了解
- 2、有没有关注什么最新的技术
- 3、对加班的看法、
- 4、介绍实习经验
- 5、印象最深刻的一个项目，用英文将这个项目说一遍
- 6、实习过程中有没有想过开发一些自动化测试工具
- 7、你在学校的成绩，
- 8、你觉得你做的比较出彩的事情
- 9、就发表的论文讨论（
- 10、发表的论文印象最深的？
- 11、那你觉得文章效果好，好在哪里呢？
- 12、那你能给我总结一下，它的创新点在哪里？
- 13、打断一下，是有监督的学习还是无监督的？
- 14、怎么看待你的岗位？

参考回答：随着行业的发展，测试的地位越发重要，但是对测试人员的要求也越来越高，高端测试人才的需求量很大，前景光明，但是作为测试从业人员，应当不断磨练自己的专业技能，学习创新，不然极易被快速发展的测试行业淘汰。

- 15、标签的搜集是怎么搜集的？
- 16、那握手用的什么协议？上层的协议的用的那个协议？
- 17、数字认证认证成功是要通过用户名和密码么？
- 18、怎么判断是不是伪造？
- 19、具体怎么判断？信道信息能模拟么？
- 20、换了时间，信息就不一样了
- 21、相似性多大才会认为是合法的呢？
- 22、你们用的什么机器学习算法？
- 23、那你分类选的啥？你用其他算法去对比了么？
- 24、近几年有什么职业规划？
- 25、打算一辈子做测试吗？
- 26、你有遇到什么困难吗？
- 27、你的短板是什么？
- 28、项目里最成功的是什么？你自己做的吗？多大规模？你怎么实现的，从开始到实现流程是怎样的？



- 29、你为什么需要做测试？结合你自己的优势说（这里我强调了一下，我的学习成绩，学习能力强）
- 30、你有哪些与众不同的优点
- 31、让我介绍下自己，自己的优点和缺点，同学和朋友怎么看你
- 32、和别人合作过项目没（答曰做过项目负责人），是怎么协调组员合作的，有过冲突和矛盾还有困难？怎么解决的
- 33、问拿到的 offer，谈谈为什么选择头条
- 34、问你了解头条多少，下载 APP 没，怎么看待
- 35、你有什么擅长的技术
- 36、对测试开发有什么了解
- 37、问成绩、实习时间、实习时间能不能延期、开题报告了没、论文定了什么内容没？
- 38、你有没有意愿做系统测试？
- 39、你自己比较偏向于做测试开发？还是系统测试？
- 40、问我亚索的 e 在一小段时间内是不能对同一目标使用的，怎么测
- 41、玩过 dota 吗，说一下你当时玩 dota 的时候发现的 bug
- 42、游戏手感是什么？能不能用数据来描述？
- 43、FPS 啥意思？
- 44、读过什么源码吗

## 十六、Hr 面

- 1、有没有家属在腾讯？
- 2、有没有患过重大疾病、精神疾病、心里疾病？
- 3、有没有亲属在深圳？
- 4、意向工作城市？
- 5、还投过哪些公司？
- 6、为什么想来腾讯？
- 7、成绩、实习时间，实习能不能延期？
- 8、职业规划，为什么来北京
- 9、对于互联网现状的认识
- 10、参加过什么竞赛，你在里面担任了什么角色
- 11、大学这几年怎么过的
- 12、有没有测试的什么经验
- 13、如果说来实习，你住的地方怎么解决呢？大概入职时间？
- 14、你觉得自己是个什么样的人？
- 15、你对未来同事、老大、团队、公司都有什么期望？
- 16、如果老大给你一些资料让你看，然后你看不懂，老大又非常忙，你该怎么办？
- 17、你觉得在公司里什么样的事情会让你很有成就感呢？
- 18、项目主要解决的问题，从项目中学到的最好的技术，项目经验对做我们这个岗位的好处
- 19、玩过哪些游戏？

## 十七、惊喜福利

此面试题库将根据当下面试形式大数据随时更新，如果你已获得下载权限，那么你可以终身在牛币兑换中心里去兑换此面试题库的电子版，如果电子版有更新，会通过牛客站内信进行通知（前提是你已获得下载权限）。

牛币兑换中心：<https://www.nowcoder.com/coin/index>

还能兑换各种惊喜周边哦



牛客定制



热门商品



名企周边



专业书籍



虚拟商品

