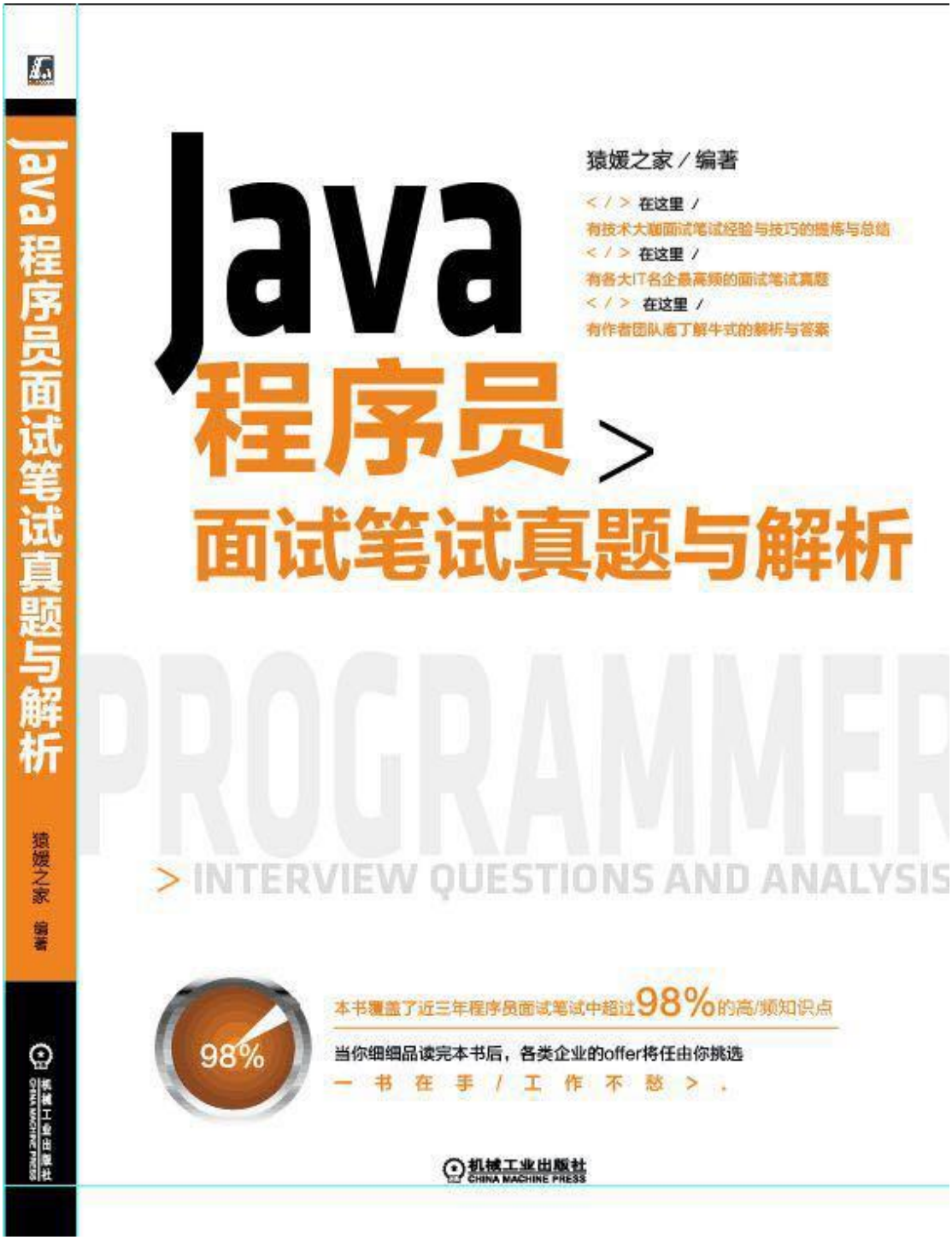


支持作者劳动成果，欢迎购买纸质图书，京东、当当、亚马逊、淘宝均有售

Java 程序员面试笔试真题与解析

猿媛之家　编著

程序员最可信赖的求职帮手

机械工业出版社

本书针对当前各大 IT 企业面试笔试中特性与侧重点，精心挑选了三年来近百家顶级 IT 企业的面试笔试真题。这些企业涉及业务包括系统软件、搜索引擎、电子商务、手机 APP、安全关键软件等，所提供的面试笔试真题非常具有代表性与参考性。同时，本书对这些题目进行了合理的划分与归类，并且对其进行了庖丁解牛式的分析与讲解，针对试题中涉及的部分重难点问题，本书都进行了适当地扩展与延伸，力求对知识点的讲解清晰而不紊乱，全面而不啰嗦，使得读者不仅能够通过本书获取到求职的知识，还能更有针对性地进行求职准备，最终收获一份满意的工作。

本书是一本计算机相关专业毕业生面试、笔试的求职用书，同时也适合期望在计算机软、硬件行业大显身手的计算机爱好者阅读。

图书在版编目（CIP）数据

Java 程序员面试笔试真题与解析 / 猿媛之家编著. —北京：机械工业出版社，2016.11

ISBN 978-7-111-55398-4

Ⅰ. ①J… Ⅱ. ①猿… Ⅲ. ①JAVA 语言－程序设计－习题集 Ⅳ. ①TP312.8-44

中国版本图书馆 CIP 数据核字（2016）第 276437 号

机械工业出版社（北京市百万庄大街 22 号 邮政编码 100037）

策划编辑：时 静 责任编辑：时 静

责任校对：张艳霞 责任印制：

印刷（ 装订）

2017 年 1 月第 1 版 • 第 1 次印刷

184mm×260mm • 22.25 印张 • 538 千字

0001－3000 册

标准书号：ISBN 978-7-111-55398-4

定价：59.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

电话服务 网络服务

服务咨询热线：（010）88361066 机 工 官 网：www.cmpbook.com

读者购书热线：（010）68326294 机 工 官 博：weibo.com/cmp1952

前　言

程序员求职始终是当前社会的一个热点，而市面上有很多关于程序员求职的书籍，例如《程序员代码面试指南》（左程云著）、《剑指 offer》（何海涛著）、《程序员面试笔试宝典》（何昊编著）、《Java 程序员面试笔试宝典》（何昊编著）、《编程之美》（《编程之美》小组著）、《编程珠玑》（Jon Bentley 著）等，它们都是针对基础知识的讲解，各有侧重点，而且在市场上反映良好。但是，我们发现，当前市面上没有一本专门针对 C/C++程序员、Java 程序员的面试笔试真题的分析与讲解，很多读者朋友们向我们反映，他们经过了精心的准备以后，感觉自己什么知识都会了，但是否真的能够在程序员面试笔试中得心应手，心里却一点底都没有。有时上网搜索一些 IT 企业的面试笔试真题，但这些题大都七零八凑，毫无系统性可言，而且绝大多数都是博主自己做的，答案简单，准确性不高，这就导致读者做完了这些真题，根本就不知道自己做得是否正确。如果下一次这个题目又被考查，可能还是不会。

针对这种情况，我们创作团队经过精心准备，从互联网上的海量面试笔试真题中，选取了当前顶级企业（包括微软、谷歌、百度、腾讯、阿里巴巴、360 和小米等）的面试笔试真题，挑选出其中最典型、考查频率最高、最具代表性的真题，做到难度适宜，兼顾各层次读者的需求，同时对真题进行知识点的分门别类，做到层次清晰、条理分明、答案简单明了，最终形成了这样一本《Java 程序员面试笔试真题与解析》。本书特点鲜明，所选真题以及写作手法具有以下特点：

第一，考查率高；本书中所选真题全是程序员面试笔试常考点，例如语言基础、操作系统、计算机网络、数据结构与算法、海量数据处理等。

第二，行业代表性强；本书中所选真题全部来自于顶级知名企业，它们是行业的风向标，代表了行业的高水准，其中绝大多数真题因为题目难易适中，而且具有非常好的区分度，经常会被众多中小企业全盘照搬，具有代表性。

第三，答案详尽；本书对每一道题目都有非常详细的解答，庖丁解牛，不只是告诉读者答案，还提供了详细的解答过程。授之以鱼的同时还授之以渔，不仅告诉答案，还告诉读者同类型题目再遇到时该如何解答。

第四，分类清晰、调理分明；本书对各个知识点都进行了分门别类，这种写法有利于读者针对个人实际情况做到有的放矢，重点把握。

由于篇幅所限，我们没法将所有的程序员面试笔试真题内容都列入其中，鉴于此，我们猿媛之家在官方网站（www.yuanyuanba.com）上提供了一个读者交流平台，读者朋友们可以在该网站上上传各类面试笔试真题，也可以查找到自己所需要的知识，同时，读者朋友们也可以向本平台提供当前最新、最热门的程序员面试笔试题、面试技巧、程序员生活等相关材料。除此以外，我们还建立了公众号：**猿媛之家**，作为对外消息发布平台，以便最大限度地满足读者需要。欢迎读者关注探讨新技术。

本书主要针对 Java 用户，我们还有专门针对 C/C++用户的图书，同期出版发行。有需要的读者可以在各大电商网站或者实体书店购买。

感谢帮助过我们的父母、亲人、同事、朋友和同学，无论我们遇到了多大的挫折与困难，他们对我们都能不离不弃，一如既往地支持与帮助我们，使我们能够开开心心地度过每一天。在此对以上所有人致以最衷心的感谢。

所有的成长和伟大，如同中药，都是一个时辰一个时辰熬出来的，所有的好书，都是逐字逐句琢磨出来的。在技术的海洋里，我们不是创造者，但我们更愿意去当好一名传播者的角色，让更多的求职者能够通过对本书的系统学习，找到一份自己满意的工作，实现自己的人生理想与抱负。

我们每个人的人生都是一场戏剧，我们每个人都要成为戏剧的主角，而不应该沦为别人戏剧的配角，所以，我建议所有的求职者在求职的道路上，无论遇到了多大的困难，遭遇了多大的挫折，都不要轻言放弃，你们的母校可能不是“985”“211”，你们的学历可能不是本科生、研究生，你们的专业可能也不是计算机相关，但这并不重要，只要你认真努力，立志成为一名程序员，百分之九十以上的企业是完全可以进去的。请记住：**在这个世界上，没有人可以让你仰视，除非你自己跪着。**

由于编者水平有限，书中不足之处在所难免，还望读者见谅。读者如果发现问题或者有此方面的困惑，都可以通过邮箱 yuancoder@foxmail.com 联系我们。

猿媛之家

程序员最可信赖的求职帮手

目　录

面试笔试经验技巧篇

面试笔试真题练习篇

面试笔试经验技巧篇

想找到一份程序员的工作，一点技术都没有显然是不行的，但是，只有技术也是不够的。面试笔试经验技巧篇主要针对程序员面试笔试中遇到的 13 个常见问题进行了深度解析，并且结合实际情景，给出了一个较为合理的参考答案以供读者学习与应用，掌握这 13 个问题的解答精髓，对于求职者大有裨益。

经验技巧 1　如何巧妙地回答面试官的问题？

所谓“来者不善，善者不来”，程序员面试中，求职者不可避免地需要回答面试官各种刁钻、犀利的问题，回答面试官的问题千万不能简单地回答“是”或者“不是”，而应该具体分析“是”或者“不是”的理由。

回答面试官的问题是一门很深入的学问。那么，面对面试官提出的各类问题，如何才能条理清晰地回答呢？如何才能让自己的回答不至于撞上枪口呢？如何才能让自己的回答结果令面试官满意呢？

谈话是一种艺术，回答问题也是一种艺术，同样的话，不同的回答方式，往往也会产生出不同的效果，甚至是截然不同的效果。在此，编者提出以下几点建议，供读者参考。首先回答问题务必谦虚谨慎。既不能让面试官觉得自己很自卑，唯唯诺诺，也不能让面试官觉得自己清高自负，而应该通过问题的回答表现出自己自信从容、不卑不亢的一面。例如，当面试官提出“你在项目中起到了什么作用”的问题时，如果求职者回答：我完成了团队中最难的工作，此时就会给面试官一种居功自傲的感觉，而如果回答：我完成了文件系统的构建工作，这个工作被认为是整个项目中最具有挑战性的一部分内容，因为它几乎无法重用以前的框架，需要重新设计。这种回答不仅不傲慢，反而有理有据，更能打动面试官。

其次，回答面试官的问题时，不要什么都说，要适当地留有悬念。人一般都有猎奇的心理，面试官自然也不例外，而且，人们往往对好奇的事情更有兴趣、更加偏爱，也更加记忆深刻。所以，在回答面试官问题时，切记说关键点而非细节，说重点而非和盘托出，通过关键点，吸引面试官的注意力，等待他们继续“刨根问底”。例如，当面试官对你的简历中一个算法问题有兴趣，希望了解时，可以如下回答：我设计的这种查找算法，对于 80%以上的情况，都可以将时间复杂度从 O(n)降低到 O(logn)，如果您有兴趣，我可以详细给您分析具体的细节。

最后，回答问题要条理清晰、简单明了，最好使用“三段式”方式。所谓“三段式”，有点类似于中学作文中的写作风格，包括“场景/任务”“行动”和“结果”三部分内容。以面试官提的问题“你在团队建设中，遇到的最大挑战是什么”为例，第一步，分析场景/任务：在我参与的一个 ERP 项目中，我们团队一共四个人，除了我以外的其他三个人中，两个人能力很给力，人也比较好相处，但有一个人却不太好相处，每次我们小组讨论问题的时候，他都不太爱说话，也很少发言，分配给他的任务也很难完成。第二步，分析行动：为了提高团队的综合实力，我决定找个时间和他好好单独谈一谈。于是我利用周末时间，约他一起吃饭，吃饭的时候，顺便讨论了一下我们的项目，我询问了一些项目中他遇到的问题，通过他的回答，我发现他并不懒，也不糊涂，只是对项目不太了解，缺乏经验，缺乏自信而已，所以越来越孤立，越来越不愿意讨论问题。为了解决这个问题，我尝试着把问题细化到他可以完成的程度，从而建立起他的自信心。第三步，分析结果：他是小组中水平最弱的人，但是，慢慢地，他的技术变得越来越厉害了，也能够按时完成安排给他的工作了，人也越来越自信了，也越来越喜欢参与我们的讨论，并发表自己的看法，我们也都愿意与他一起合作了。“三段式”回答的一个最明显的好处就是条理清晰，既有描述，也有结果，有根有据，让面试官一目了然。

回答问题的技巧，是一门大的学问。求职者完全可以在平时的生活中加以练习，提高自己与人沟通的技能，等到面试时，自然就得心应手了。

经验技巧 2　如何回答技术性的问题？

程序员面试中，面试官会经常询问一些技术性的问题，有的问题可能比较简单，都是历年的笔试面试真题，求职者在平时的复习中会经常遇到，应对自然不在话下。但有的题目可能比较难，

来源于 Google、Microsoft 等大企业的题库或是企业自己为了招聘需要设计的题库，求职者可能从来没见过或者从来都不能完整地、独立地想到解决方案，而这些题目往往又是企业比较关注的。

如何能够回答好这些技术性的问题呢？编者建议：会做的一定要拿满分，不会做的一定要拿部分分。即对于简单的题目，求职者要努力做到完全正确，毕竟这些题目，只要复习得当，完全回答正确一点问题都没有（编者认识的一个朋友据说把《编程之美》、《编程珠玑》、《程序员面试笔试宝典》上面的技术性题目与答案全都背得滚瓜烂熟了，后来找工作简直成了“offer 杀器”，完全就是一个 Bug，无解了）；对于难度比较大的题目，不要惊慌，也不要害怕，即使无法完全做出来，也要努力思考问题，哪怕是半成品也要写出来，至少要把自己的思路表达给面试官，让面试官知道你的想法，而不是完全回答不会或者放弃，因为面试官很多时候除了关注你的独立思考问题的能力以外，还会关注你技术能力的可塑性，观察求职者是否能够在别人的引导下去正确地解决问题，所以，对于你不会的问题，他们很有可能会循序渐进地启发你去思考，通过这个过程，让他们更加了解你。

一般而言，在回答技术性问题时，求职者大可不必胆战心惊，除非是没学过的新知识，否则，一般都可以采用以下六个步骤来分析解决。

（1）勇于提问

面试官提出的问题，有时候可能过于抽象，让求职者不知所措，或者无从下手，所以，对于面试中的疑惑，求职者要勇敢地提出来，多向面试官提问，把不明确或二义性的情况都问清楚。不用担心你的问题会让面试官烦恼，影响你的面试成绩，相反还对面试结果产生积极影响：一方面，提问可以让面试官知道你在思考，也可以给面试官一个心思缜密的好印象；另一方面，方便后续自己对问题的解答。

例如，面试官提出一个问题：设计一个高效的排序算法。求职者可能丈二和尚摸不到头脑，排序对象是链表还是数组？数据类型是整型、浮点型、字符型还是结构体类型？数据基本有序还是杂乱无序？数据量有多大，1000 以内还是百万以上个数？此时，求职者大可以将自己的疑问提出来，问题清楚了，解决方案也自然就出来了。

（2）高效设计

对于技术性问题，如何才能打动面试官？完成基本功能是必须的，仅此而已吗？显然不是，完成基本功能顶多只能算及格水平，要想达到优秀水平，至少还应该考虑更多的内容，以排序算法为例：时间是否高效？空间是否高效？数据量不大时也许没有问题，如果是海量数据呢？是否考虑了相关环节，例如数据的“增删改查”？是否考虑了代码的可扩展性、安全性、完整性以及鲁棒性？如果是网站设计，是否考虑了大规模数据访问的情况？是否需要考虑分布式系统架构？是否考虑了开源框架的使用？

（3）伪代码先行

有时候实际代码会比较复杂，上手就写很有可能会漏洞百出、条理混乱，所以，求职者可以首先征求面试官的同意，在编写实际代码前，写一个伪代码或者画好流程图，这样做往往会让思路更加清晰明了。

切记在写伪代码前要告诉面试官，他们很有可能对你产生误解，认为你只会纸上谈兵，实际编码能力却不行。只有征得了他们的允许，方可先写伪代码。

（4）控制节奏

如果是算法设计题，面试官都会给求职者一个时间限制用以完成设计，一般为 20min 左右。完成得太慢，会给面试官留下能力不行的印象，但完成得太快，如果不能保证百分百正确，也会给面试官留下毛手毛脚的印象，速度快当然是好事情，但只有速度，没有质量，速度快根本不会给面试加分。所以，编者建议，回答问题的节奏最好不要太慢，也不要太快，如果实在是完成得比较快，也不要急于提交给面试官，最好能够利用剩余的时间，认真仔细地检查一些边界情况、异常情况及极性情况等，看是否也能满足要求。

（5）规范编码

回答技术性问题时，多数都是纸上写代码，离开了编译器的帮助，求职者要想让面试官对自己的代码一看即懂，除了字迹要工整，不能眉飞色舞以外，最好是能够严格遵循编码规范：函数变量命名、换行缩进、语句嵌套和代码布局等，同时，代码设计应该具有完整性，保证代码能够完成基本功能、输入边界值能够得到正确地输出、对各种不合规范的非法输入能够做出合理的错误处理，否则，写出的代码即使无比高效，面试官也不一定看得懂或者看起来非常费劲，这些对面试成功都是非常不利的。

（6）精心测试

在软件界，有一句真理：任何软件都有 bug。但不能因为如此就纵容自己的代码，允许错误百出。尤其是在面试过程中，实现功能也许并不十分困难，困难的是在有限的时间内设计出的算法，各种异常是否都得到了有效的处理，各种边界值是否都在算法设计的范围内。

测试代码是让代码变得完备的高效方式之一，也是一名优秀程序员必备的素质之一。所以，在编写代码前，求职者最好能够了解一些基本的测试知识，做一些基本的单元测试、功能测试、边界测试以及异常测试。

在回答技术性问题时，注意在思考问题的时候，千万别一句话都不说，面试官面试的时间是有限的，他们希望在有限的时间内尽可能地去了解求职者，如果求职者坐在那里一句话不说，不仅会让面试官觉得求职者技术水平不行，思考问题能力以及沟通能力可能都存在问题。

其实，在面试时，求职者往往会存在一种思想误区，把技术性面试的结果看得太重要了。面试过程中的技术性问题，结果固然重要，但也并非最重要的内容，因为面试官看重的不仅仅是最终的结果，还包括求职者在解决问题的过程中体现出来的逻辑思维能力以及分析问题的能力。所以，求职者在与面试官的博弈中，要适当地提问，通过提问获取面试官的反馈信息，并抓住这些有用的信息进行辅助思考，从而博得面试官的欢心，进而提高面试的成功率。

经验技巧 3　如何回答非技术性问题？

评价一个人的能力，除了专业能力，还有一些非专业能力，如智力、沟通能力和反应能力等，所以在 IT 企业招聘过程的笔试面试环节中，并非所有的笔试内容都是 C/C++、数据结构与算法及操作系统等专业知识，也包括其他一些非技术类的知识，如智力题、推理题和作文题等。技术水平测试可以考查一个求职者的专业素养，而非技术类测试则更加强调求职者的综合素质，包括数学分析能力、反应能力、临场应变能力、思维灵活性、文字表达能力和性格特征等内容。考查的形式多种多样，但与公务员考查相似，主要包括行测（占大多数）、性格测试（大部分都有）、应用文和开放问题等内容。

每个人都有自己的答题技巧，答题方式也各不相同，以下是一些相对比较好的答题技巧（以行测　为例）：

1）合理有效的时间管理。由于题目的难易不同，所以不要对所有题目都“绝对的公平”、都“一刀切”，要有轻重缓急，最好的做法是不按顺序回答。行测中有各种题型，如数量关系、图形推理、应用题、资料分析和文字逻辑等，而不同的人擅长的题型是不一样的，因此应该首先回答自己最擅长的问题。例如，如果对数字比较敏感，那么就先答数量关系。

2）注意时间的把握。由于题量一般都比较大，可以先按照总时间/题数来计算每道题的平均答题时间，如 10s，如果看到某一道题 5s 后还没思路，则马上放弃。在做行测题目的时候，以在最短的时间内拿到最多分为目标。

3）平时多关注图表类题目，培养迅速抓住图表中各个数字要素间相互逻辑关系的能力。

4）做题要集中精力，只有集中精力、全神贯注，才能将自己的水平最大限度地发挥出来。

5）学会关键字查找，通过关键字查找，能够提高做题效率。

6）提高估算能力，有很多时候，估算能够极大地提高做题速度，同时保证正确率。

除了行测以外，一些企业非常相信个人性格对入职匹配的影响，所以都会引入相关的性格测试题用于测试求职者的性格特性，看其是否适合所投递的职位。大多数情况下，只要按照自己的真实想法选择就行了，不要弄巧成拙，因为测试是为了得出正确的结果，所以大多测试题前后都有相互验证的题目。如果求职者自作聪明，选择该职位可能要求的性格选项，则很可能导致测试前后不符，这样很容易让企业发现你是个不诚实的人，从而首先予以筛除。

经验技巧 4　如何回答快速估算类问题？

有些大企业的面试官，总喜欢使一些“阴招”“损招”，出一些快速估算类问题，对他们而言，这些问题只是手段，不是目的，能够得到一个满意的结果固然是他们所需要的，但更重要的是通过这些题目他们可以考查求职者的快速反应能力以及逻辑思维能力。由于求职者平时准备的时候可能对此类问题有所遗漏，一时很难想起解决的方案。而且，这些题目乍一看确实是毫无头绪，无从下手，完全就是坑求职者的，其实求职者只要从惊慌失措中冷静下来，稍加分析，也就那么回事。因为此类题目比较灵活，属于开放性试题，一般没有标准答案，只要弄清楚了回答要点，分析合理到位，具有说服力，能够自圆其说，就是正确答案，一点都不困难。

例如，面试官可能会问这样一个问题：“请你估算一下一家商场在促销时一天的营业额？”，求职者又不是统计局官员，如何能够得出一个准确的数据呢？求职者家又不是开商场的，如何能够得出一个准确的数据呢？即使求职者是商场的大当家，也不可能弄得清清楚楚明明白白吧？

难道此题就无解了吗？其实不然，本题只要能够分析出一个概数就行了，不一定要精确数据，而分析概数的前提就是做出各种假设。以该问题为例，可以尝试从以下思路入手：从商场规模、商铺规模入手，通过每平方米的租金，估算出商场的日租金，再根据商铺的成本构成，得到全商场日均交易额，再考虑促销时的销售额与平时销售额的倍数关系，乘以倍数，即可得到促销时一天的营业额。具体而言，包括以下估计数值：

1）以一家较大规模商场为例，商场一般按 6 层计算，每层大约长 100m，宽 100m，合计 60000m2 的面积。

2）商铺规模占商场规模的一半左右，合计 30000m2。

3）商铺租金约为 40 元/ m2，估算出年租金为 40×30000×365=4.38 亿。

4）对商户而言，租金一般占销售额 20%左右，则年销售额为 4.38 亿×5=21.9 亿。计算平均日销售额为 21.9 亿/365=600 万。

5）促销时的日销售额一般是平时的 10 倍，所以大约为 600 万*10=6000 万。

此类题目涉及面比较广，例如：估算一下北京小吃店的数量？估算一下中国在过去一年方便面的市场销售额是多少？估算一下长江的水的质量？估算一下一个行进在小雨中的人 5min 内身上淋到的雨的质量？估算一下东方明珠电视塔的质量？估算一下中国去年一年一共用掉了多少块尿布？估算一下杭州的轮胎数量？但一般都是即兴发挥，不是哪道题记住答案就可以应付得了的。遇到此类问题，一步步抽丝剥茧，才是解决之道。

经验技巧 5　如何回答算法设计问题？

程序员面试中的很多算法设计问题，都是历年来各家企业的“炒现饭”，不管求职者以前对算法知识学习得是否扎实，理解得是否深入，只要面试前买本《程序员面试笔试宝典》（编者早前编写的一本书，由机械工业出版社出版），学习上一段时间，牢记于心，应付此类题目完全没有问题，但遗憾的是，很多世界级知名企业也深知这一点，如果纯粹是出一些毫无技术含量的题

目，对于考前“突击手”而言，可能会占尽便宜，但对于那些技术好的人而言是非常不公平的。所以，为了把优秀的求职者与一般的求职者能够更好地区分开来，他们会年年推陈出新，越来越倾向于出一些有技术含量的“新”题，这些题目以及答案，不再是以前的陈谷子烂芝麻了，而是经过精心设计的好题。

在程序员面试中，算法的地位就如同是 GRE 或托福考试在出国留学中的地位一样，必须但不是最重要的，它只是众多考核方面中的一个而已，不一定就能决定求职者的生死。虽然如此，但并非说就不用去准备算法知识了，因为算法知识回答得好，必然会成为面试的加分项，对于求职成功，百利而无一害。那么如何应对此类题目呢？很显然，编者不可能将此类题目都在《程序员面试笔试宝典》中一一解答，一来由于内容众多，篇幅有限，二来也没必要，今年考过了，以后一般就不会再考了，不然还是没有区分度。编者以为，靠死记硬背肯定是行不通的，解答此类算法设计问题，需要求职者具有扎实的基本功以及良好的运用能力，编者无法左右求职者的个人基本功以及运用能力，因为这些能力需要求职者“十年磨一剑”地苦学，但编者可以提供一些比较好的答题方法和解题思路，以供求职者在面试时应对此类算法设计问题。“授之以鱼不如授之以渔”，岂不是更好？

（1）归纳法

此方法通过写出问题的一些特定的例子，分析总结其中一般的规律。具体而言就是通过列举少量的特殊情况，经过分析，最后找出一般的关系。例如，某人有一对兔子饲养在围墙中，如果它们每个月生一对兔子，且新生的兔子在第二个月后也是每个月生一对兔子，问一年后围墙中共有多少对兔子。

使用归纳法解答此题，首先想到的就是第一个月有多少对兔子，第一个月的时候，最初的一对兔子生下一对兔子，此时围墙内共有两对兔子。第二个月仍是最初的一对兔子生下一对兔子，共有 3 对兔子。到第三个月除最初的兔子新生一对兔子外，第一个月生的兔子也开始生兔子，因此共有 5 对兔子。通过举例，可以看出，从第二个月开始，每一个月兔子总数都是前两个月兔子总数之和，Un+1=Un+Un　1，一年后，围墙中的兔子总数为 377 对。

此种方法比较抽象，也不可能对所有的情况进行列举，所以，得出的结论只是一种猜测，还需要进行证明。

（2）相似法

正如编者“年年岁岁花相似，岁岁年年仍单身”一样，此方法考虑解决问题的算法是相似的。如果面试官提出的问题与求职者以前用某个算法解决过的问题相似，此时此刻就可以触类旁通，尝试改进原有算法来解决这个新问题。而通常情况下，此种方法都会比较奏效。

例如，实现字符串的逆序打印，也许求职者从来就没遇到过此问题，但将字符串逆序肯定在求职准备的过程中是见过的。将字符串逆序的算法稍加处理，即可实现字符串的逆序打印。

（3）简化法

此方法首先将问题简单化，例如改变一下数据类型、空间大小等，然后尝试着将简化后的问题解决，一旦有了一个算法或者思路可以解决这个被“阉割过”的问题，再将问题还原，尝试着用此类方法解决原有问题。

例如，在海量日志数据中提取出某日访问 xxx 网站次数最多的那个 IP。很显然，由于数据量巨大，直接进行排序不可行，但如果数据规模不大时，采用直接排序不失为一种好的解决方法。那么如何将问题规模缩小呢？于是想到了 Hash 法，Hash 往往可以缩小问题规模，然后在“阉割过”的数据里面使用常规排序算法即可找出此问题的答案。

（4）递归法

为了降低问题的复杂度，很多时候都会将问题逐层分解，最后归结为一些最简单的问题，这就是递归。此种方法，首先要能够解决最基本的情况，然后以此为基础，解决接下来的问题。

例如，在寻求全排列的时候，可能会感觉无从下手，但仔细推敲，会发现后一种排列组合往

往是在前一种排列组合的基础上进行的重新排列，只要知道了前一种排列组合的各类组合情况，只需将最后一个元素插入到前面各种组合的排列里面，就实现了目标：即先截去字符串 s[1…n] 中的最后一个字母，生成所有 s[1…n　1]的全排列，然后再将最后一个字母插入到每一个可插入的位置。

（5）分治法

任何一个可以用计算机求解的问题所需的计算时间都与其规模有关。问题的规模越小，越容易直接求解，解题所需的计算时间也越少。而分治法正是充分考虑到这一内容，将一个难以直接解决的大问题，分割成一些规模较小的相同问题，以便各个击破，分而治之。分治法一般包含以下三个步骤：

1）将问题的实例划分为几个较小的实例，最好具有相等的规模。

2）对这些较小的实例求解，而最常见的方法一般是递归。

3）如果有必要，合并这些较小问题的解，以得到原始问题的解。

分治法是程序员面试常考的算法之一，一般适用于二分查找、大整数相乘、求最大子数组和、找出伪币、金块问题、矩阵乘法、残缺棋盘、归并排序、快速排序、距离最近的点对、导线与开关等。

（6）Hash 法

很多面试笔试题目，都要求求职者给出的算法尽可能高效。什么样的算法是高效的？一般而言，时间复杂度越低的算法越高效。而要想达到时间复杂度的高效，很多时候就必须在空间上有所牺牲，用空间来换时间。而用空间换时间最有效的方式就是 Hash 法、大数组和位图法。当然，此类方法并非包治百病，有时，面试官也会对空间大小进行限制，那么此时，求职者只能再去思考其他的方法了。

其实，凡是涉及大规模数据处理的算法设计中，Hash 法就是最好的方法之一。

（7）轮询法

在设计每道面试笔试题时，往往会有一个载体，这个载体便是数据结构，例如数组、链表、二叉树或图等，当载体确定后，可用的算法自然而然地就会暴露出来。可问题是很多时候并不确定这个载体是什么。当无法确定这个载体时，一般也就很难想到合适的方法了。

编者建议，此时，求职者可以采用最原始的思考问题的方法——轮询法，在脑海中轮询各种可能的数据结构与算法，常考的数据结构与算法一共就那么几种（见表 1），即使不完全一样，也是由此衍生出来的或者相似的，总有一款适合考题的。

表 1　最常考的数据结构与算法知识点

数 据 结 构	算　　法	概　　念
链表	广度（深度）优先搜索	位操作
数组	递归	设计模式
二叉树	二分查找	内存管理（堆、栈等）
树	排序（归并排序、快速排序等）	
堆（大顶堆、小顶堆）	树的插入/删除/查找/遍历等	
栈	图论	
队列	Hash 法	
向量	分治法	
Hash 表	动态规划	

此种方法看似笨拙，其实实用，只要求职者对常见的数据结构与算法烂熟于心，一点都没有问题。

为了更好地理解这些方法，求职者可以在平时的准备过程中，应用此类方法去答题，做得多

了，自然对各种方法也就熟能生巧了，面试的时候，再遇到此类问题，也就能够收放自如了。当然，千万不要相信有着张无忌般的运气，能够在一夜之间练成乾坤大挪移这一绝世神功，称霸武林，算法设计功力的练就是平时一点一滴的付出和思维的磨练。方法与技巧也许只是给面试打了一针“鸡血”、喂一口“大补丸”，不会让自己变得从容自信，真正的功力还是需要一个长期的积累过程的。

经验技巧 6　如何回答系统设计题？

应届生在面试的时候，偶尔也会遇到一些系统设计题，而这些题目往往只是测试一下求职者的知识面，或者测试求职者对系统架构方面的了解，一般不会涉及具体的编码工作。虽然如此，对于此类问题，很多人还是感觉难以应对，也不知道从何说起。

如何应对此类题目呢？在正式介绍基础知识之前，首先罗列几个常见的系统设计相关的面试笔试题，如下所示：

1）设计一个 DNS 的 Cache 结构，要求能够满足每秒 5000 次以上的查询，满足 IP 数据的快速插入，查询的速度要快（题目还给出了一系列的数据，比如站点数总共为 5000 万、IP 地址有 1000 万等）。

2）有 N 台机器，M 个文件，文件可以以任意方式存放到任意机器上，文件可任意分割成若干块。假设这 N 台机器的宕机率小于 1/3，想在宕机时可以从其他未宕机的机器中完整导出这 M 个文件，求最好的存放与分割策略。

3）假设有三十台服务器，每台服务器上面都存有上百亿条数据（有可能重复），如何找出这三十台机器中，根据某关键字，重复出现次数最多的前 100 条？要求使用 Hadoop 来实现。

4）设计一个系统，要求写速度尽可能快，并说明设计原理。

5）设计一个高并发系统，说明架构和关键技术要点。

6）有 25T 的 log(query->queryinfo)，log 在不断地增长，设计一个方案，给出一个 query 能快速返回 queryinfo。

以上所有问题中凡是不涉及高并发的，基本可以采用 Google 的三个技术解决，即 GFS、MapReduce 和 Bigtable，这三个技术被称为“Google 三驾马车”，Google 只公开了论文而未开源代码，开源界对此非常有兴趣，仿照这三篇论文实现了一系列软件，如 Hadoop、HBase、HDFS 及 Cassandra 等。

在 Google 这些技术还未出现之前，企业界在设计大规模分布式系统时，采用的架构往往是 database+sharding+cache，现在很多公司（比如 taobao、weibo.com）仍采用这种架构。在这种架构中，仍有很多问题值得去探讨。如采用什么数据库，是 SQL 界的 MySQL 还是 NoSQL 界的 Redis/TFS，两者有何优劣？采用什么方式 sharding（数据分片），是水平分片还是垂直分片？据网上资料显示，weibo.com 和 taobao 图片存储中曾采用的架构是 Redis/MySQL/TFS+sharding+cache，该架构解释如下：前端 cache 是为了提高响应速度，后端数据库则用于数据永久存储，防止数据丢失，而 sharding 是为了在多台机器间分摊负载。最前端由大块大块的 cache 组成，要保证至少 99%（该数据在 weibo.com 架构中的是自己猜的，而 taobao 图片存储模块是真实的）的访问数据落在 cache 中，这样可以保证用户访问速度，减少后端数据库的压力。此外，为了保证前端 cache 中的数据与后端数据库中的数据一致，需要有一个中间件异步更新（为什么使用异步？理由简单：同步代价太高。异步有缺点，如何弥补?）数据，这个有些人可能比较清楚，新浪有个开源软件叫 Memcachedb（整合了 Berkeley DB 和 Memcached），正是完成此功能。另外，为了分摊负载压力和海量数据，会将用户微博信息经过分片后存放到不同节点上（称为“Sharding”）。

这种架构优点非常明显：简单，在数据量和用户量较小的时候完全可以胜任。但缺点是扩展

性和容错性太差，维护成本非常高，尤其是数据量和用户量暴增之后，系统不能通过简单地增加机器解决该问题。

鉴于此，新的架构应运而生。新的架构仍然采用 Google 公司的架构模式与设计思想，以下将分别就此内容进行分析。

GFS 是一个可扩展的分布式文件系统，用于大型的、分布式的、对大量数据进行访问的应用。它运行于廉价的普通硬件上，提供容错功能。现在开源界有 HDFS（Hadoop Distributed File System），该文件系统虽然弥补了数据库+sharding 的很多缺点，但自身仍存在一些问题，比如：由于采用 master/slave 架构，因此存在单点故障问题；元数据信息全部存放在 master 端的内存中，因而不适合存储小文件，或者说如果存储大量小文件，那么存储的总数据量不会太大。

MapReduce 是针对分布式并行计算的一套编程模型。其最大的优点是：编程接口简单，自动备份（数据默认情况下会自动备三份），自动容错和隐藏跨机器间的通信。在 Hadoop 中，MapReduce 作为分布计算框架，而 HDFS 作为底层的分布式存储系统，但 MapReduce 不是与 HDFS 耦合在一起的，完全可以使用自己的分布式文件系统替换掉 HDFS。当前 MapReduce 有很多开源实现，如 Java 实现 Hadoop MapReduce，C++实现 Sector/sphere 等，甚至有些数据库厂商将 MapReduce 集成到数据库中了。

BigTable 俗称“大表”，是用来存储结构化数据的，编者觉得，BigTable 在开源界最火爆，其开源实现最多，包括 HBase、Cassandra 和 levelDB 等，使用也非常广泛。

除了 Google 的这“三驾马车”以外，还有其他一些技术可供学习与使用：

Dynamo：亚马逊的 key-value 模式的存储平台，可用性和扩展性都很好，采用 DHT（Distributed Hash Table）对数据分片，解决单点故障问题，在 Cassandra 中，也借鉴了该技术，在 BT 和电驴这两种下载引擎中，也采用了类似算法。

虚拟节点技术：该技术常用于分布式数据分片中。具体应用场景是：有一大块数据（可能 TB 级或者 PB 级），需按照某个字段（key）分片存储到几十（或者更多）台机器上，同时想尽量负载均衡且容易扩展。传统的做法是：Hash(key) mod N，这种方法最大的缺点是不容易扩展，即增加或者减少机器均会导致数据全部重分布，代价太大。于是新技术诞生了，其中一种是上面提到的 DHT，现在已经被很多大型系统采用，还有一种是对“Hash(key) mod N”的改进：假设要将数据分布到 20 台机器上，传统做法是 Hash(key) mod 20，而改进后，N 取值要远大于 20，比如是 20000000，然后采用额外一张表记录每个节点存储的 key 的模值，比如：

node1：0~1000000

node2：1000001~2000000

……

这样，当添加一个新的节点时，只需将每个节点上部分数据移动给新节点，同时修改一下该表即可。

Thrift：Thrift 是一个跨语言的 RPC 框架，分别解释“RPC”和“跨语言”如下：RPC 是远程过程调用，其使用方式与调用一个普通函数一样，但执行体发生在远程机器上；跨语言是指不同语言之间进行通信，比如 C/S 架构中，Server 端采用 C++编写，Client 端采用 PHP 编写，怎样让两者之间通信，Thrift 是一种很好的方式。

本篇最前面的几道题均可以映射到以上几个系统的某个模块中，如：

1）关于高并发系统设计，主要有以下几个关键技术点：缓存、索引、数据分片及锁粒度尽可能小。

2）题目 2 涉及现在通用的分布式文件系统的副本存放策略。一般是将大文件切分成小的 block（如 64MB）后，以 block 为单位存放三份到不同的节点上，这三份数据的位置需根据网络拓扑结构配置，一般而言，如果不考虑跨数据中心，可以这样存放：两个副本存放在同一个机架的不同节点上，而另外一个副本存放在另一个机架上，这样从效率和可靠性上，都是最优的（这

个 Google 公布的文档中有专门的证明，有兴趣的可参阅一下）。如果考虑跨数据中心，可将两份存在一个数据中心的不同机架上，另一份放到另一个数据中心。

3）题目 4 涉及 BigTable 的模型。主要思想是将随机写转化为顺序写，进而大大提高写速度。具体是：由于磁盘物理结构的独特设计，其并发的随机写（主要是因为磁盘寻道时间长）非常慢，考虑到这一点，在 BigTable 模型中，首先会将并发写的大批数据放到一个内存表（称为“memtable”）中，当该表大到一定程度后，会顺序写到一个磁盘表（称为“SSTable”）中，这种写是顺序写，效率极高。此时可能有读者问，随机读可不可以这样优化？答案是：看情况。通常而言，如果读并发度不高，则不可以这么做，因为如果将多个读重新排列组合后再执行，系统的响应时间太慢，用户可能接受不了，而如果读并发度极高，也许可以采用类似机制。

面试笔试真题练习篇

面试笔试真题练习篇主要针对近 3 年以来近百家顶级 IT 企业的面试笔试真题而设计，这些企业涉及业务包括系统软件、搜索引擎、电子商务、手机 APP 和安全关键软件等，面试笔试真题难易适中，覆盖面广，非常具有代表性与参考性。本篇对这些真题进行了合理的划分与归类（包括 Java 语言基础知识、操作系统、计算机网络与通信、数学知识、数据库、系统设计题和海量数据处理等内容），并且对其进行了庖丁解牛式的分析与讲解，针对真题中涉及的部分重难点问题，本篇都进行了适当的扩展与延伸，力求对知识点的讲解清晰而不紊乱，全面而不啰嗦，使得读者能够通过本书不仅获取到求职的知识，同时更有针对性地进行求职准备，最终能够收获一份满意的工作。

第 1 章　Java 语言基础

1.1　Java 语言基础知识

【真题 1】 Java 语言具有哪些特点？

答案：SUN 公司对 Java 语言的描述如下：“Java is a simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, and dynamic language”。具体而言，Java 语言具有以下几个方面的优点：

1）Java 为纯面向对象的语言（《Java 编程思想》提到 Java 语言是一种“Everything is object”的语言），它能够直接反映现实生活中的对象，例如火车、动物等，因此，通过它，开发人员更容易编写程序。

2）平台无关性。Java 语言可以一次编译，到处运行。无论是在 Windows 平台还是在 Linux、MacOS 等其他平台上对 Java 程序进行编译，编译后的程序在其他平台上都可以运行。由于 Java 为解释型语言，编译器会把 Java 代码变成“中间代码”，然后在 JVM（Java Virtual Machine，Java 虚拟机）上解释执行。由于中间代码与平台无关，所以，Java 语言可以很好地跨平台执行，具有很好的可移植性。

3）Java 提供了很多内置的类库，这些类库简化了开发人员的程序设计工作，同时缩短了项目的开发时间。例如，Java 语言提供了对多线程支持，提供了对网络通信的支持，最重要的一点是提供了垃圾回收器，使开发人员从对内存的管理中解脱出来。

4）Java 语言提供了对 Web 应用开发的支持，例如 Applet、Servlet 和 JSP 可以用来开发 Web 应用程序，Socket、RMI 可以用来开发分布式应用程序的类库。

5）Java 语言具有较好的安全性和健壮性。Java 语言经常被用在网络环境中，为了增强程序的安全性，Java 语言提供了一个防止恶意代码攻击的安全机制（数组边界检测和 byte code 校验等）。Java 的强类型机制、垃圾回收器、异常处理和安全检查机制使得使用 Java 语言编写的程序有很好的健壮性。

6）Java 语言去除了 C++语言中难以理解、容易混淆的特性，例如头文件、指针、结构、单元、运算符重载、虚拟基础类、多重继承等，使得程序更加严谨、简洁。

【真题 2】 Java 整型的字节序是（　　）。

A．Little-Endian（小端）　　B．Big-Endian（大端）

C．由运行程序的 CPU 决定　　D．由编译程序的 CPU 决定

答案：B。

字节序是指多字节数据在计算机内存中存储或者网络传输时各字节的存储顺序。通常有 Little-Endian（小端）与 Big-Endian（大端）两种方式。以下将分别对这两种方式进行介绍。

（1）Little-Endian

Little-Endian（小端）是指低位字节存放在内存的低地址端，高位字节存放在内存的高地址端。例如，当按照小端模式存储时，十六进制数字表示 0x12 34 56 78 在内存中的存储方式为：

低地址 ----------->高地址

0x78 | 0x56 | 0x34 | 0x12

（2）Big-Endian

Big-Endian（大端）是指高位字节存放在内存的低地址端，低位字节存放在内存的高地址端。例如，当按照大端模式存储时，十六进制数字表示 0x12 34 56 78 在内存中的存储方式为：

低地址 ---------->高地址

0x12 | 0x34 | 0x56 | 0x78

为什么要区分大小端呢？因为在计算机系统中，所有的存储都是以字节（一个字节占用 8 bit）为单位进行存储的，但是在大部分编程语言中，除了占 1 个字节的 char 数据类型外，还有占多个字节的其他数据类型，例如，在 Java 语言中，short 类型占 2 个字节，int 类型占 4 个字节。那么如何存储这些占用多个字节的数据呢？既可以使用大端的方式存储，也可以使用小端的方式来存储。不同的编程语言，不同的处理器可能会采用不同的存储方式。

Java 是一种跨平台的语言，Java 字节序指的是在 Java 虚拟机中多字节类型数据的存放顺序，Java 字节序是 Big-Endian（大端）。所以，选项 B 正确。

既然 Java 语言的字节序是 Big-Endian（大端），那么如何获知 Java 语言的字节序呢？可以通过查看字节数组在内存中存放的方式来获知。示例代码如下：

```
import java.io.*;
public class Test
{
    public static void main(String[] args) throws IOException
    {
        byte[] arr = new byte[4];
        arr[0] = 0x78;
        arr[1] = 0x56;
        arr[2] = 0x34;
        arr[3] = 0x12;
        ByteArrayInputStream bais = new ByteArrayInputStream(arr);
        DataInputStream dis = new DataInputStream(bais);
        System.out.println(Integer.toHexString(dis.readInt()));
    }
}
```

程序的运行结果为：

```
78563412
```

从上面的运行结果可以看出，高位字节（78）存储在低位地址，显然是大端。

虽然 Java 使用的是大端，在一些情况下有可能也需要获取 CPU 是大端还是小端，Java 提供了一个类库可以用来获取 CPU 是大端还是小端：java.nio.ByteOrder.nativeOrder()。

【真题 3】 下面关于 Java 语言的描述中，正确的是（　　）。

A．可以使用 goto 跳出循环　　B．关键字 final 修饰的类无法被继承

C．String 对象的内容是无法修改的　　D．Java 类可以实现多个接口

答案：B、C、D。

对于选项 A，在 Java 语言中，goto 是保留关键字，没有 goto 语句，也没有任何使用 goto 关键字的地方。当然，在特定情况下，通过特定的手段，也是可以实现 goto 功能的。因此，选项 A 错误。

对于选项 B，被 final 修饰的类是不能被继承的。因此，选项 B 正确。

对于选项 C，因为 String 是不可变量，所以，String 的内容是不能被修改的。因此，选项 C　　正确。

对于选项 D，虽然 Java 语言不支持多重继承，但是可以通过实现多个接口的方式间接地实现多重继承。因此，选项 D 正确。

【真题 4】 以下不是 Object 类的方法的是（　　）。

A．hashCode()　　B．finalize()　　C．notify()　　D．hasNext()

答案：D。

Object 类是类层次结构的根，在 Java 语言中，所有的类从根本上而言都继承自这个类。而且，Object 类是 Java 语言中唯一没有父类的类，而其他所有的类，包括标准容器类，例如数组，都继承了 Object 类。

具体而言，Object 类的方法见表 1–1。

表 1–1　Object 类的方法

方　法　名	返 回 类 型	方 法 描 述
clone()	Object	创建并返回此对象的一个副本
equals(Object obj)	boolean	判断 obj 对象是否与此对象相等
finalize()	void	当垃圾回收器确定不存在对该对象的更多引用时，由对象的垃圾回收器调用此方法
getClass()	Class<?>	返回此 Object 的运行时类
hashCode()	int	返回该对象的散列码值
notify()	void	唤醒在此对象监视器上等待的单个线程
notifyAll()	void	唤醒在此对象监视器上等待的所有线程
toString()	String	返回该对象的字符串表示
wait()	void	在其他线程调用此对象的 notify()方法或 notifyAll()方法前，使当前线程等待
wait(long timeout)	void	在其他线程调用此对象的 notify()方法或 notifyAll()方法，或者超过指定的时间量前，使当前线程等待
wait(long timeout, int nanos)	void	在其他线程调用此对象的 notify()方法或 notifyAll()方法，或者其他某个线程中断当前线程，或者已超过某个实际时间量前，使当前线程等待

由此可见，Object 类没有 hasNext()方法。所以，选项 D 正确。

【真题 5】Math.round(12.5)的返回值等于（　　），Math.round(–12.5)的返回值等于（　　）。

答案：13，–12。

Math 类主要提供了下面 5 个与取整相关的方法：

1）static double ceil(double a)：返回大于等于 a 的最小整数。

2）static double floor(double a)：返回小于等于 a 的最大整数。

3）static double rint(double a)：四舍五入方法，返回与 a 的值最相近的整数，为 double 类型。

4）static long round(double a)：四舍五入方法，返回与 a 的值最相近的长整型数。

5）static int round(float a)：四舍五入方法，返回与 a 的值最相近的整型数。

对于本题而言，round 是一个四舍五入的方法，12.5 的小数部分为 0.5，当对其执行 Math.round()操作时，结果需要入，所以，结果为 13；–12.5 的小数部分也为 0.5，当对其执行 Math.round()操作时，结果也需要入，由于–12>–13，因此，结果四舍五入后为–12。

【真题 6】 下列关于 Java 语言基础知识的描述中，正确的是（　　）。

A．类是方法和变量的集合体　　B．抽象类或接口可以被实例化

C．数组是无序数据的集合　　D．类成员数据必须是公有的

答案：A。

对于选项 A，类可以被理解为变量和方法的集合体。因此，选项 A 正确。

对于选项 B，抽象类是不能被实例化的，只有实现了抽象类的具体类才能被实例化。接口也不能被实例化，只有实现了接口方法的类才能被实例化。因此，选项 B 错误。

对于选项 C，数组是一些相同类型数据的集合，而描述中没有提到相同类型，不是很合理。因此，选项 C 错误。

对于选项 D，类的数据类型可以是公开的，也可以是私有的，由于面向对象封装的特点，一般会把成员数据设计为私有的，然后提供公有的方法对其访问。因此，选项 D 错误。

【真题 7】 有如下代码：

```
public class Test
{
        public static void main(String[] args)
        {
            class A
            {
                public int i=3;
            }
            Object o = (Object)new A();
            A a = (A)o;
            System.out.println("i = "+ a.i);
        }
}
```


上述程序运行后的结果是（　　）。

A．i=3　　B．编译失败

C．运行结果为 ClassCastException　　D．i=0

答案：A。

类 A 是 main 方法内部的一个内部类，执行 new A()的时候初始化了一个 A 的对象，这个对象的　　属性 i 的值为 3，经过类型转换后，最终，a 是这个对象的引用，因此，输出结果为 i=3。所以，选项 A 正确。

【真题 8】 下列描述中，正确的是（　　）。

A．Java 程序经编译后会产生 Machine Code（机器码）

B．Java 程序经编译后会产生 Byte Code（字节码）

C．Java 程序经编译后会产生 DLL（动态链接库）

D．以上描述都不正确

答案：B。

Java 语言为解释性语言，运行的过程为：源代码经过 Java 编译器编译成字节码（Byte Code），然后由 JVM（Java Virtual Machine，Java 虚拟机）解释执行。而 C/C++语言为编译型语言，源代码经过编译和链接后生成可执行的二进制代码。因此，Java 语言的执行速度比 C/C++语言慢，但是 Java 语言能够跨平台执行，而 C/C++语言不能够跨平台执行。

所以，Java 程序经编译后会产生 Byte Code（字节码），选项 B 正确，而选项 A、选项 C 和选项 D 描述都不正确。

【真题 9】 Java 语言是从（　　）语言改进重新设计 。

A．BASIC　　B．C++　　C．Pascal　　D．Ada

答案：B。

对于选项 A，BASIC 语言是一种为了让用户容易控制计算机开发的语言，其特点是简单易懂，且可以用解释和编译两种方法执行。

对于选项 B，C++语言是一种静态数据类型检查的、支持多重编程范式的通用程序设计语言，它支持过程化程序设计、数据抽象、面向对象程序设计、泛型程序设计等多种程序设计风格。

对于选项 C，Pascal 语言是为提倡结构化编程而发明的语言。
对于选项 D，Ada 语言是美国军方为了整合不同语言开发的系统而发明的一种语言，其最大的特点是实时性，在 ADA95 中已加入面向对象内容。
Java 语言是一种面向对象语言，从语法结构上看，与 C++语言类似。所以，选项 B 正确。
【真题 10】 下列关于 Java 语言的描述中，正确的是（　　）。
A．Java 语言容许单独的过程与函数存在 B． Java 语言容许单独的方法存在
C．Java 语言中的方法属于类中的成员　　D．Java 语言中的方法必定隶属于某一类（对象）
答案：D。
Java 语言是纯面向对象的语言，任何变量与方法都必须定义在类中，方法与变量不能脱离类的定义而单独存在，因此，选项 A 和选项 B 错误，选项 D 正确。在 Java 语言中，方法有两种：静态方法（类的方法）与非静态方法（实例的方法），因此，选项 C 错误，因为方法有可能属于实例成员，而不属于类成员。
【真题 11】 下列关于按值传递与按引用传递的描述中，正确的是（　　）。
A．按值传递不会改变实际参数的数值　　B．按引用传递能改变实际参数的参考地址
C．按引用传递能改变实际参数的内容　　D．按引用传递不能改变实际参数的参考地址
答案：A、C、D。
按值传递指的是在方法调用时，传递的参数是实参值的副本。按引用传递指的是在方法调用时，传递的参数是实参的引用，也可以理解为实参所对应的内存空间的地址。
为了理解 Java 语言中的值传递与引用传递，首先给出下面的示例代码：

```
public class Test{
    public static void testPassParameter(StringBuffer ss1, int n)
    {
        ss1.append(" World");
        n=8;
    }
    public static void main(String[] args)
    {
        int i=1;
        StringBuffer s1=new StringBuffer("Hello");
        testPassParameter(s1,i);
        System.out.println(s1);
        System.out.println(i);
    }
}
```


程序的运行结果为：

```
Hello World
1
```

从运行结果可以看出，int 作为参数时，对形参值的修改不会影响到实参，对于 StringBuffer 类型的参数，对形参对象内容的修改影响到了实参。为了便于理解，int 类型的参数可以理解为按值传递，StringBuffer 类型的参数可以理解为引用传递。
为了便于理解，Java 课本中会经常提到在 Java 应用程序中永远不会传递对象，而只传递对象引用，因此，是按引用传递对象。从本质上来讲，引用传递还是通过值传递来实现的，Java 语言中的引用传递实际上还是值传递（传递的是地址的值）。如图 1–1 所示。

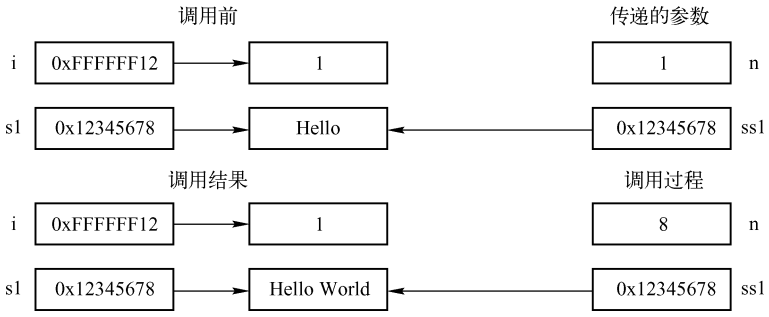


图 1–1　引用传递

首先按照传统的分析方法来理解按值传递和按引用传递：

为了便于理解，假设 1 和“Hello”存储的地址分别为 0xFFFFFF12 和 0x12345678。在调用方法 testPassParameter 时，由于 i 为基本类型，因此，参数是按值传递的，此时会创建一个 i 的副本，该副本与 i 有相同的值，把这个副本作为参数赋值给 n，作为传递的参数。而 StringBuffer 由于是一个类，因此，按引用传递，传递的是它的引用（可以理解为传递的是存储“Hello 的地址”），如图 1–1 所示，在 testPassParameter 内部修改的是 n 的值，这个值与 i 是没关系的。但是在修改 ss1 时，修改的是 ss1 这个地址指向的字符串的内容，由于形参 ss1 与实参 s1 指向的是同一块存储空间，因此，修改 ss1 后，s1 指向的字符串也被修改了。

再从另外一个角度出发来对引用传递进行详细分析：

对于变量 s1 而言，它是一个字符串对象的引用，引用的字符串的值是“Hello”，而变量 s1 的值为 0x12345678（可以理解为是“Hello”的地址，或者“Hello”的引用），那么在方法调用时，参数传递的其实就是 s1 值的一个副本（0x12345678），如图 1–1 所示，ss1 的值也为 0x12345678。如果在方法调用的过程中通过 ss1（字符串的引用或地址）来修改字符串的内容，因为 s1 与 ss1 指向同一个字符串，所以，通过 ss1 对字符串的修改对 s1 也是可见的。但是方法中对 ss1 值的修改对 s1 是没有影响的，如下例所示：

```
public class Test
{
    public static void testPassParameter(StringBuffer ss1)
    {
        ss1 = new StringBuffer("World");
    }
    public static void main(String[] args)
    {
        StringBuffer s1 = new StringBuffer("Hello");
        testPassParameter(s1);
        System.out.println(s1);
    }
}
```

程序的运行结果为：

```
Hello
```

对运行结果分析可知，在 testPassParameter 方法中，依然假设“Hello”的地址为 0xFFFFFF12（实际上是 s1 的值），在方法调用的时候，首先把 s1 的副本传递给 ss1，此时 ss1 的值也为 0xFFFFFF12，通过调用 ss1=new StringBuffer("World")语句实际上是改变了 ss1 的值（ss1 指向了另外一个字符串“World”），但是对形参 ss1 值的改变对实参 s1 没有影响，虽然 ss1 被改变“World”的引用（或者“World”的地址），s1 还是代表字符串“Hello”的引用（或可以理解为 s1 的值仍然是“Hello”的地址）。从这个角度出发来看，StringBuffer 从本质上来讲还是值传

递，它是通过值传递的方式来传递引用的。

对于本题，通过以上分析可知，值传递只是传递了一个值的副本，对形参值的改变不会影响实参的值，因此，选项 A 正确。由于参数的地址也是以值的方式传递的，因此，无法改变实参的地址，只能改变实参地址指向的对象的值，因此，选项 B 错误，选项 C 和选项 D 正确。

【真题 12】 下列关于形式参数的描述中，正确的是（　　）。

A．形式参数可被视为局部变量　　B．形式参数不可以是对象

C．形式参数为方法被调用时真正被传递的参数　　D．形式参数可被字段修饰符修饰

答案：A。

形参全称为“形式参数”，是在定义方法名和方法体的时候使用的参数，目的是用来接收调用该方法时传递的参数。它的作用范围是整个方法体，是在方法调用时的一个临时变量。实参出现在主调方法中，在方法调用的时候把实参的值赋给对应的形参，在被调用方法的内部只能使用形参，不能使用实参。

具体而言，实参和形参的主要区别如下：

1）形参的作用范围为方法内部，当方法调用结束后，形参的生命周期也就结束了，因此，在方法外不能使用形参，它只有在被调用时才分配内存单元，调用结束后会立刻释放内存空间，而实参不能在调用方法的内部使用。

2）在方法调用的时候，只能把实参的值传送给形参，而不能把形参的值反向地传送给实参。因此，在方法调用过程中，对形参值的改变不会影响实参的值。

本题中，对于选项 A，形参的作用范围只在这个方法内，因此，可以被看作 Local Variable（局部变量）。因此，选项 A 正确。

对于选项 B，形参可以是原始的数据类型（例如 int、long、char）等，也可以是对象（例如 String、List 或自定义对象类型）。因此，选项 B 不正确。

对于选项 C，在方法调用时，真正被传递的参数为实参。因此，选项 C 不正确。

对于选项 D，形参不能被字段修饰符（例如 public、protected、private 等）修饰。因此，选项 D 不正确。

【真题 13】 以下关于 Java 语言中的引用的描述中，正确的是（　　）。

A．引用实际上就是指针　　B．引用本身是 Primitive

C．一个对象只能被一个引用所指引　　D．引用就是对象本身

答案：B。

对于选项 A，在编程语言中，指针指向一块内存，它的内容是所指内存的地址；而引用是某块内存的别名。Java 语言中没有明确的指针定义，但实质上每一个 new 语句返回的都是一个指针的引用，只不过在大多时候，Java 语言并不用关心如何操作这个“指针”。虽然引用在底层是通过指针实现的，但是引用和指针不能等同，例如指针可以执行比较运算和整数加减运算，而引用却不行。所以，选项 A 错误。

对于选项 B，引用本身存储的对象的地址信息（虽然与指针不是完全相同），而这个地址信息是存储在栈中的，在声明后就会立刻在栈上给分配存储空间。在方法调用传递引用的时候，对形参引用的值本身所做的修改对实参不可见，因此，从本质上来讲，引用也是原始数据类型（Primitive）。所以，选项 B 正确。

对于选项 C，一个对象可以被多个引用同时指引，例如 String s="abc"; String s1=s;。所以，选项 C 错误。

对于选项 D，引用只是对象的一个别名，或理解为对象的地址。所以，选项 D 错误。

【真题 14】 下列关于 Java 语言基础知识的描述中，错误的有（　　）。

A．能被 java.exe 成功运行的 java class 文件必须有 main()方法

B．J2SDK 就是 Java API

C．Appletviewer.exe 可利用 jar 选项运行.jar 文件

D．能被 Appletviewer 成功运行的 java class 文件必须有 main()方法

答案：B、C、D。

对于选项 A，在 Java 语言中，main 方法是程序的入口方法，所有程序的执行都是从 main 方法开始的，如果没有 main 方法，程序是无法执行的。所以，选项 A 正确。

对于选项 B，J2SDK 是 Java 的开发环境包，它包含 JDK（开发工具包）和 JRE（运行时环境包）。而 Java API 是一些预先定义的类库，目的是提供应用程序与开发人员基于某软件或硬件的以访问一组例程的能力。所以，选项 B 错误。

对于选项 C 与选项 D，Appletviewer 是一种执行 HTML 文件上 Java 小程序的 Java 浏览器。实质上就是一个 Applet 浏览器，Applet 本身是没有 main 方法的。可以使用 jdk 工具里面的 appletviewer.exe 来运行 Applet，或者使用浏览器运行，它不能运行 jar 包文件。所以，选项 C 和选项 D 错误。

【真题 15】 Java 程序中的起始类名称必须与存放该类的文件名相同。（　　）

答案：正确。

【真题 16】 写出下面程序运行的结果：

```
public class Test {
    static boolean f(char c) {
        System.out.print(c);
        return true;
    }
    public static void main(String[] argv) {
        int i = 0;
        for (f('A'); f('B') && (i < 2); f('C')) {
            i++;
            f('D');
        }
    }
}
```

答案：ABDCBDCB。

for 循环语句的基本结构为：

```
for(表达式 1;表达式 2;表达式 3)
{
    循环体
}
```

它的执行过程如下：

1）首先执行初始化语句：表达式 1（只会被执行一次）。

2）然后执行表达式 2，如果表达式 2 的结果为 false，则结束循环，否则，执行循环体，然后执行表达式 3。

3）循环执行步骤 2），直到表达式 2 的结果为 false 时，则退出循环，或者循环体内有退出循环的语句（return 或 break）。

对于本题而言，执行步骤如下：

1）首先执行 foo('A')，输出字符'A'。

2）接着执行 foo('B') && (i < 2)，输出字符'B'，且这个表达式的结果为 true，因此，执行循环体 i++（执行后 i 的值变为 1），接着输出字符'D'，然后执行 foo('C')，输出字符'C'。

3）重复第 2）步，由于此时 i 的值为 1，所以循环条件为 true，接着会输出字符'B'、'D'、'C'。

结束这一次循环后，i 的值变为 2；然后继续执行循环条件 foo('B') && (i < 2)，首先执行 foo('B') 输出字符'B'，因为 foo('B')执行的结果为 true，所以，需要继续执行判断语句 i < 2，显然返回值为 false，此时循环结束。

因此，程序的运行结果为 ABDCBDCB。

【真题 17】 &和&&的区别是什么？

答案：&是按位与操作符，a&b 是把 a 和 b 都转换成二进制数后，然后再进行按位与的运算。而&&为逻辑与操作符，a&&b 就是当且仅当两个操作数均为 true 时，其结果才为 true，只要有一个为 false，a&&b 的结果就为 false。

此外，&&还具有短路的功能，在参与运算的两个表达式中，只有当第一个表达式的返回值为 true 时，才会去计算第二个表达式的值，如果第一个表达式的返回值为 false，则此时&&运算的结果就为 false，同时，不会去计算第二个表达式的值。例如 if(i!=0 && i++>10)，当 i 的值为 0 时，表达式 i!=0 的返回值为 false，因此，此时将不会执行第二个表达式 i++>10 的判断。

【真题 18】 下面的 Java 代码保存在 B.java 文件中是否合法？

```
class A{
    public static void main(String args[]){
        System.out.println("Hello World");
    }
}
```

答案：是合法的。

虽然文件名被命名为 B.java 是合法的，但是这段代码在 Eclipse 环境下是无法运行的。因为 Eclipse 在运行的时候会首先编译 B.java 文件，然后会在 B.class 文件中找 Java 的入口方法（main 方法），显然是找不到的，因为通过 javac B.java 命令编译后只会产生一个 A.class 文件（Java 在编译的时候，会对每个类生成一个.class 文件，.class 的文件名与类名相同）。在命令行下，可以通过 java A 命令来运行这个程序。

【真题 19】 有以下代码：

```
for (int i = 4; i > 0; i--){
    int j = 0;
    do {
        j++;
        if (j == 2) {
            break;
        }
    } while (j <= i);
    System.out.print(j);
}
```

程序的运行结果是（　　）

A．4 3 2 1　　B．1 2 3 2　　C．2 2 1 1　　D．2 2 2 2

答案：D。

do/while 循环是 while 循环的变体。在检查条件是否为真之前，该循环首先会执行一次代码块，然后检查条件是否为真，如果条件为真，就会重复这个循环。

for 循环语句的基本结构如下：

```
for（表达式 1;表达式 2;表达式 3）
{
    循环体
```

}

它的执行过程如下：

1）执行初始化语句：表达式 1（只会被执行一次）。

2）执行表达式 2，如果表达式 2 的结果为 false，则结束循环，否则，执行循环体，然后执行表达式 3。

3）循环步骤 2），直到表达式 2 的结果为 false 时退出循环，或者循环体内有退出循环的语句（return 或 break）。

对于本题而言，第一次进入 for 循环体时，i=4；然后进入 do/while 循环体，此时 j=0，然后这个循环一直执行 j++，直到 j=2 或 j>i 的时候退出循环体，显然会先满足 j=2 的条件退出循环体，此时 j 的值为 2，因此，输出 2。下一次 for 循环的时候，i=3，同理输出结果仍然为 2。下一次 for 循环的时候，i=2，同理输出结果为 2。下一次 for 循环的时候，i=1，同理也会输出 2，此时执行 for 循环的 i--操作，i 的值变为 0，不满足 i>0 的条件，因此，for 循环结束，所以，输出结果为 2 2 2 2，选项 D 正确。

【真题 20】 以下关于可变长参数的定义中，正确的是（ ）。

A．public void f(String[] aa, String... a){}　　B．public void f(String a, double b , String... a){}

C．public void f(String... a){}　　D．public void f(String... a , String[] aa){}

答案：A、C。

在 Java 语言中，可以使用省略号…来实现可变参数，可变参数通常有如下几个特点：

1）只能作为最后一个参数出现。如果参数个数不定，当其后边还有相同类型参数时，Java 语言无法区分传入的参数属于前一个可变参数还是后面的参数，所以，只能让可变参数位于最后一项。

2）只能位于变量的类型和变量名之间。

3）编译器为可变参数隐含创建一个数组，在调用的时候，可以用数组的形式来访问可变参数，如下例所示：

```
public class Test{
    public static void main(String[] args){
        print(1,2);
    }
    public static void print(int... args) {
        for (int i = 0; i < args.length; i++) {
            System.out.println(args[i]);
        }
    }
}
```


程序的运行结果为：

```
1
2
```

对于本题而言，对于选项 A 与选项 C，满足变参的要求。所以，选项 A 与选项 C 正确。

对于选项 B，有两个名字相同的参数。所以，选项 B 错误。

对于选项 D，变参不是作为最后一个参数出现的。所以，选项 D 错误。

【真题 21】 以下关于随机数的描述中，正确的是（ ）。

A．Math.random()可以生成[0,1]内的任意小数

B．Random.next(10)可以生成[0, 10]内的任意整数

C．new java.util.Random().nextInt(11)可以生成[0,10]内的任意整数

D．new java.util.Math().random()可以生成[0,1)内的任意小数

答案：C。

对于选项 A，Math 类的 random 方法的功能是生成[0, 1)的小数，不能生成 1。因此，选项 A 错误。

对于选项 B，Random 类没有 next 这个方法。因此，选项 B 错误。

对于选项 C，nextInt(n)方法的功能是生成[0, n)的整数，所以，nextInt(11)可以生成[0, 11)，即[0, 10]的整数。因此，选项 C 正确。

对于选项 D，java.util 包下没有 Math 类，Math 类属于 java.lang 包。因此，选项 D 错误。

【真题 22】 下面程序是否存在问题？如果存在，请指出问题所在，如果不存在，说明输出结果。

```
package package1;
import java.util.Date;
public class Test extends Date
{
    private void test()
    {
        System.out.println(super.getClass().getName());
    }
    public static void main(String[] args)
    {
        new Test().test();
    }
}
```

答案：不存在问题，输出结果为 package1.Test。

Java 语言提供了获取类名的方法：getClass().getName()，开发人员可以调用这个方法来获取类名。那么通过调用父类的 getClass().getName()方法来获取父类的类名是可行的吗？为了解答这个问题，首先来做一个实验。给出下面的程序：

```
class A{}
public   class Test extends A
{
    public void test()
    {
        System.out.println(super.getClass().getName());
    }
    public static void main(String[] args)
    {
        new Test().test();
    }
}
```


程序的运行结果为：

```
Test
```

为什么输出的结果不是“A”而是“Test”呢？主要原因在于 Java 语言中任何类都继承自 Object 类，getClass()方法在 Object 类中被定义为 final 与 native，而子类不能覆盖该方法。因此，this.getClass()和 super.getClass()最终都调用的是 Object 类中的 getClass()方法。而 Object 类中的 getClass()方法的释义是：返回此 Object 的运行时类。由于在上面代码

中实际运行的类是 Test 而不是 A，因此，程序输出结果为 Test。那么如何才能在子类中得到父类的名字呢？可以通过 Java 的反射机制，使用 getClass().getSuperclass().getName()。代码如下所示：

```
class A{}
public    class Test extends A
{
        public void test()
        {
                System.out.println(this.getClass().getSuperclass().getName());
        }
        public static void main(String[] args)
        {
                new Test().test();
        }
}
```

程序的运行结果为：

```
A
```

对于本题而言，调用 super.getClass().getName()也会得到当前运行类的名字，即 package1.Test。

【真题 23】 有如下代码：

```
public class Test{
        public static void hello() {System.out.println("hello");      }
        public static void main(String[] args) {    ((Test) null).hello();     }
}
```

上面程序（　　）正常运行。

A．不能　　B．能　　C．不确定

答案：B。

在 Java 语言中，给任何对象赋值为 null 都是合法的， null 可以被强制转换为任意类型的对象，转换的结果还是 null，因此，无法调用对象的方法，但是可以调用类的方法（因为类的方法是不依赖于对象而存在的）。

对于本题而言，((Test) null) 可以把 null 转换为 Test 类型的对象（转换后的值为 null），由于 hello 是一个静态方法，因此，可以直接调用，输出结果为“hello”。所以，选项 B 正确。

【真题 24】 有以下代码：

```
package com;
public class Test {
        public static void main(String[] args)    {
                Test test = new Test();
        }
}
```

下面可以获得 Class 对象的有（　　）。

A．Class　c　=　test.class;　　B．Class　c　=　new Class();

C．Class　c　=　Test.class;　　D．Class　c　=　test.getClass();

E．Class　c　=　Class.forName(“com.Test”);　　F．Class　c　=　Class.forName(“Test”);

答案：C、D、E。

在 Java 语言中，主要有如下几种方法可以用来获取 Class 对象：

1）调用对象的 getClass 方法，选项 D 就是采用这种方法。所以，选项 D 正确。
2）调用 Class.forName()方法，这个方法的参数为类的全名（包名.类名），选项 E 就是使用这种方法。所以，选项 E 正确。
3）使用.class 语法来获得 Class 对象，具体而言，就是调用类的.class 来获取 Class 对象，选项 C 就是采用这种方法。所以，选项 C 正确。

【真题 25】 编译 Java 应用程序源文件将产生相应的字节码文件，这些字节码文件的扩展名为（　　）。

A．.class　　B．.java　　C．.html　　D．.exe

答案：A。

对于选项 A 与选项 B，Java 程序源文件的后缀为.java，编译生成中间代码文件的后缀名为.class。所以，选项 A 正确，选项 B 错误。

对于选项 C，在 Web 应用程序的开发中，静态网页文件的后缀为.html。所以，选项 C 错误。

对于选项 D，一般情况下，Windows 操作系统中可执行文件的后缀为.exe。所以，选项 D 错误。

【真题 26】 如果一个 Java Applet 源程序文件只定义有一个类，该类的类名为 MyApplet，那么类 MyApplet 必须是类（　　）的子类，并且存储该源程序文件的文件名为（　　）。

答案：Applet，MyApplet.java。

Applet 指的是 Java 小应用程序，是能够嵌入到一个 HTML 页面中，并且可通过 Web 浏览器下载和执行的一种 Java 类。它不需要 main()方法，由 Web 浏览器中内嵌的 Java 虚拟机调用执行。

Applet 程序开发必须继承 Applet 类。由于只定义了一个类，因此，这个类所在的文件名必须与类名相同，文件名只能是 MyApplet.java。

【真题 27】 开发与运行 Java 程序的三个主要步骤为（　　）、（　　）和（　　）。

答案：编写源程序，编译生成字节码，解释执行。

具体而言，Java 应用程序的开发过程如下所示：首先编写代码，代码文件的后缀为.java，然后编译源代码，用 javac 命令把源代码编译成中间代码（.class 文件），最后用 java 命令运行中间代码。

【真题 28】 如果一个 Java Applet 程序文件中定义有 4 个类，则使用 Sun 公司的 JDK 编译器（　　）编译该源程序文件将产生（　　）个文件名，与类名相同而扩展名为（　　）的字节码文件。

答案：Javac，4，.class。

在 Java 语言中，不管在一个文件中定义几个类，使用 javac 命令编译后，每个类都会生成一个.class 文件（.class 文件的名字为类名）。

【真题 29】 假设 x = 5，y = 6，则 x < y 和 x >= y 的逻辑值分别为（　　）和（　　）。

答案：true，false。

本题中，由于 x=5，y=6，所以，x<y 是成立的，因此，逻辑值为 true，而 x >= y 不成立，因此，逻辑值为 false。

【真题 30】 下列代码会在（　　）出错。

```
1) public void modify(){
2)     int i, j, k;
3)     i = 100;
4)     while ( i > 0 ) {
5)         i = j * 2;
6)         System.out.println("The value of j is" + j);
7)         k = k + 1;
```

```
8)            i--;
9)        }
10) }
```

A．line 4　　B．line 6　　C．line 7　　D．line 8

答案：C。

由于使用了没有初始化的变量 j，因此会出错。

【真题 31】 下列说法正确的是（　　）。

A．java.lang.Clonable 是类　　B．java.lang.Runnable 是接口

C．Double 对象在 java.lang 包中　　D．Double a = 1.0 是正确的 Java 语句

答案：B、C、D。

【真题 32】 下列关于 JDK 版本的描述中，正确的是（　　）。

A．JDK 1.7 中，switch 操作可以支持 String 类型

B．JDK1.7 中，支持二进制字面量

C．JDK1.8 中，支持 Lambda 表达式

D．JDK1.8 中，可以在接口的方法中添加默认实现

答案：A、B、C、D。

本题考查对 JDK 新特性的理解。

对于选项 A，JDK1.7 增加了在 switch 语句中支持 String 的特性。所以，选项 A 正确。

对于选项 B，JDK1.7 支持二进制字面量，写法为：int binary = 0b1001。所以，选项 B 正确。

对于选项 C，JDK1.8 增加了对 Lambda 表达式的支持。所以，选项 C 正确。

对于选项 D，JDK1.8 通过使用关键字 default 可以给接口中的方法添加默认实现。所以，选项 D　　正确。

为了更好地说明 JDK 各个版本之间的差异，下面重点介绍 JDK1.7 与 JDK1.8 中一些其他的新特性。

JDK1.7 的部分新特性如下：

1）switch 可以接受 String 类型。随着 Java 语言的发展，在 Java7 中，switch 开始支持 String 类型。以下是一段支持 String 类型的示例代码：

```
public class Test {
    public void test(String str) {
        switch(str) {
            case "computer":
                System.out.println("computer ");
                break;
            case "book":
                System.out.println("book");
                break;
            case "iphone":
                System.out.println("iphone ");
                break;
            default:
                System.out.println("default");
        }
    }
}
```


从本质上来讲，switch 对字符串的支持，其实是对 int 类型值的匹配。它的实现原理如下：通过对 case 后面的 String 对象调用 hashCode()方法，得到一个 int 类型的 hash 值，然后用

这个 hash 值来唯一标识这个 case。当进行匹配的时候，首先调用这个字符串的 hashCode()方法，获取到一个 hash 值（int 类型），用这个 hash 值来匹配所有的 case，如果没有匹配成功，则说明不存在，如果匹配成功，则会接着调用字符串的 equals()方法进行匹配。由此可以看出，String 变量不能为 null，同时，switch 的 case 子句中使用的字符串也不能为 null。

2）可以在 catch 代码块中捕获多个异常类型，如下面代码所示：

```
try{
        //可能会抛出 Exception1 和 Exception2 异常的代码
} catch (Exception1 | Exception2    e) {
        //处理异常的代码
}
```

3）对数值字面量进行了改进 。

① 增加了二进制字面量的表示（0B001、0b111）。整数类型（例如 byte、short、int 与 long 等）也可以使用二进制数来表示。要指定一个二进制字面量，可以给二进制数字添加前缀 0b 或者 0B。相比于十六进制或者八进制，二进制字面量可以使数据之间的关系更加清晰。

② 在数字中可以添加分隔符，例如 123_456，下划线只能被用在数字中间，编译的时候这些下划线会被编译器去掉。这样做的好处是避免了一些难以通过观察代码来发现的细微错误。例如数字 10000000000 和数字 1000000000，不仔细看很难发现两个数字中谁少了一个 0 或多了一个 0，但对于 10_000_000_000 和 1_000_000_000 却不然。

4）使用泛型的时候增加了类型推断机制。

在 Java7 以前，实例化一个 HashMap 对象的写法如下：

```
Map<String, String> trades = new HashMap <String, String> ();
```

而 Java7 引进了类型推断机制，因此，可以采用更加简洁的写法，如下所示：

```
Map<String, String> trades = new HashMap <> ();
```

5）增加了 try-with-resources 语句（try-with-resources 语句是一个声明了一个或多个资源的 try 语句。这里的一个资源指的是在使用完成后，必须关闭释放的对象。try-with-resources 语句可以确保在该语句执行之后关闭每个资源），确保了每个资源都在生命周期结束后被关闭，因此，在读取文件结束后，不需要显式地调用 close 方法。示例代码如下：

```
try (InputStream fis = new FileInputStream("input.txt");){
      while(fis.read()!=-1)
            System.out.println(fis.read());
}
catch (Exception e) {
      e.printStackTrace();
}
```


同理，在使用 JDBC 访问数据库时，Statement 对象也可以采用同样的写法。

6）增加了 fork/join 框架用来增强对处理多核并行计算的支持，它的应用场景为：如果一个应用能被分解成多个子任务，并且组合多个子任务的结果就能够获得最终的答案，在这种情况下，就可以使用 fork/join 框架。

具体而言，fork 就是把一个大任务切分为若干个子任务并行地执行，join 就是合并这些子任务的执行结果，最后得到这个大任务的结果。例如，当需要计算 1+2+….+10000 的值时，可以把这样一个大的任务分割成 10 个小的子任务，每个子任务分别对其中的 1000 个数进行求和，最终汇总这 10 个子任务的结果即为最终结果。

以上讲解的都是 JDK1.7 的新特性，下面重点讲解 JDK1.8 的部分新特性。

1）增加了对 Lambda 表达式的支持。Lambda 表达式是一个匿名函数（指的是没有函数名的函数），它基于数学中的 λ 演算得名，直接对应于其中的 Lambda 抽象。Lambda 表达式可以表示闭包（注意和数学传统意义上的不同）。
Lambda 表达式允许把函数作为一个方法的参数。Lambda 表达式的基本语法如下：

```
(parameters) -> expression
```

或

```
(parameters) ->{ statements; }
```

Lambda 的使用如下例所示：

```
Arrays.asList( 1, 7, 2 ).forEach( i -> System.out.println( i ) );
```

以上这种写法中，i 的类型是由编译器推测出来的，当然，也可以显式地指定类型，如下例所示：

```
Arrays.asList( 1, 7, 2 ).forEach( ( Integer i ) -> System.out.println( i ) );
```

在 Java8 以前，Java 语言通过匿名函数的方法来代替 Lambda 表达式。
对于列表的排序，如果列表里面存放的是自定义的类，通常需要指定自定义的排序方法，传统的写法如下：

```
import java.util.Arrays;
import java.util.Comparator;
class Person
{
    public Person(String name, int age)
    {
        this.name = name;
        this.age = age;
    }
    private String name;
    private int age;
    public int getAge() {return age;}
    public String getName() {return name;}
    public String toString() {return name + ":" + age;    }
}
public class Test
{
    public static void main(String[] args)
    {
        Person[] people = { new Person("James", 25), new Person("Jack", 21) };
        // 自定义类排序方法，通过年龄进行排序
        Arrays.sort(people, new Comparator<Person>()
        {
            @Override
            public int compare(Person a, Person b)
            {
                return a.getAge() - b.getAge();
            }
        });
        for (Person p : people)
        {
```

```
                System.out.println(p);
            }
        }
    }
```

采用 Lambda 表达式后，写法如下：

```
Arrays.sort(people, (Person a, Person b) -> a.getAge()-b.getAge());
```

或

```
Arrays.sort(people, (a, b) -> a.getAge()-b.getAge());
```

显然，采用 Lambda 表达式后，代码会变得更加简洁。

Lambda 表达式是通过函数式接口（只有一个方法的普通接口）来实现的。函数式接口可以被隐式地转换为 Lambda 表达式。为了与普通的接口区分开（普通接口中可能会有多个方法），JDK1.8 新增加了一种特殊的注解@FunctionalInterface。下面给出一个函数式接口的定义：

```
@FunctionalInterface
interface fun {
    void f();
}
```

2）接口增加了方法的默认实现和静态方法。JDK1.8 通过使用关键字 default 可以给接口中的方法添加默认实现，此外，接口中还可以定义静态方法，示例代码如下：

```
interface Inter8{
    void f();
    default void g() {
        System.out.println("this is default method in interface");
    }
    static void h(){
        System.out.println("this is static method in interface");
    }
}
```

那么，为什么要引入接口中方法的默认实现呢？

其实，这样做的最重要的一个目的就是为了实现接口升级。在原有的设计中，如果想要升级接口，例如给接口中添加一个新的方法，会导致所有实现这个接口的类都需要被修改，这给 Java 语言已有的一些框架进行升级带来了很大的麻烦。如果接口能支持默认方法的实现，那么可以给这些类库的升级带来许多便利。例如，为了支持 Lambda 表达式，Collection 中引入了 foreach 方法，可以通过这个语法增加默认的实现，从而降低了对这个接口进行升级的代价，不需要对所有实现这个接口的类进行修改。

3）方法引用。方法引用指的是可以直接引用 Java 类或对象的方法。它可以被看成是一种更加简洁易懂的 Lambda 表达式，使用方法引用后，上例中的排序代码就可以使用下面更加简洁的方式来编写：

```
Arrays.sort(people, Comparator.comparing(Person::getAge));
```

方法引用共有下面 4 种形式：

① 引用构造方法：ClassName::new。

② 引用类静态方法：ClassName::methodName。

③ 引用特定类的任意对象方法：ClassName::methodName。

④ 引用某个对象的方法：instanceName::methodName。

下面给出一个使用方法引用的例子：


```
import java.util.Arrays;
import java.util.Comparator;
import java.util.function.Supplier;
class Person {
    private    String name;
    privateint age;
    public Person(){}
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
    publicstatic Person getInstance( final Supplier< Person > supplier ) {
return supplier.get();
    }
    publicvoid setAge(int age){  this.age=age;}
    publicint getAge() {return age;}
    public String getName() {return name;}
    publicstaticint compareByAge(Person a, Person b) { return b.age-a.age;}
    public String toString() {return name + ":" + age;}
}
class CompareProvider {
publicint compareByAge(Person a, Person b) {
return a.getAge()-b.getAge();
    }
}
publicclass Test2 {
    publicstaticvoid main(String[] args) {
        //引用是构造方法
        Person p1 = Person.getInstance( Person::new );
        p1.setAge(19);
        System.out.println("测试引用构造方法："+p1.getAge());
        Person[] people = { new Person("James", 25), new Person("Jack", 21) };
        //引用特定类的任意对象方法
        Arrays.sort(people, Comparator.comparing(Person::getAge));
        System.out.println("测试引用特定类的任意对象方法：");
        for (Person p : people) {
            System.out.println(p);
        }
        //引用类静态方法
        Arrays.sort(people, Person::compareByAge);
        System.out.println("测试引用类静态方法：");
        for (Person p : people) {
            System.out.println(p);
        }
        //引用某个对象的方法
        Arrays.sort(people, new CompareProvider()::compareByAge);
        System.out.println("测试引用某个对象的方法：");
        for (Person p : people) {
            System.out.println(p);
        }
    }
```

```
}
```

程序的运行结果为：

```
测试引用构造方法：19
测试引用特定类的任意对象方法：
Jack:21
James:25
测试引用类静态方法：
James:25
Jack:21
测试引用某个对象的方法：
Jack:21
James:25
```

4）注解（Annotation）。

① JDK1.5 中引入了注解机制，但是有一个限制：相同的注解在同一位置只能声明一次。JDK1.8 引入了重复注解机制后，相同的注解在同一个地方可以声明多次。

备注：注解为开发人员在代码中添加信息提供了一种形式化的方法，它使得开发人员可以在某个时刻方便地使用这些数据（通过解析注解来使用这些数据）。注解的语法比较简单，除了@符号的使用以外，它基本上与 Java 语言的语法一致，Java 语言内置了三种注解方式，它们分别是：@Override（表示当前方法是覆盖父类的方法）、@Deprecated（表示当前元素是不赞成使用的）、@SuppressWarnings（表示关闭一些不当的编译器警告信息）。需要注意的是，它们都定义在 java.lang 包中。

② JDK1.8 对注解进行了扩展。使得注解被使用的范围更广，例如可以给局部变量、泛型和方法异常等提供注解。

5）类型推测。JDK1.8 加强了类型推测机制，这种机制可以使得代码更为简洁，假如有如下类的　　定义：

```
class List<E> {
    static <Z> List<Z> nil() { ... };
    static <Z> List<Z> cons(Z head, List<Z> tail) { ... };
    E head() { ... }
}
```

在调用时，可以使用下面的代码：

```
List<Integer> l = List.nil();  //通过赋值的目标类型来推测泛型的参数
```

在 Java7 中，这种写法将会产生编译错误，正确写法如下：

```
List< Integer > l = List.< Integer >nil();
```

同理，在调用 cons 方法时的写法为：

```
List.cons(5, List.nil());  //通过方法的第一个参数来推测泛型的类型
```

而不需要显式地指定类型：List.cons(5, List.<Integer>nil());

6）参数名字。JDK1.8 通过在编译的时候增加–parameters 选项，以及增加反射 API 与 Parameter.getName()方法实现了获取方法参数名的功能。

示例代码如下：

```
import java.lang.reflect.Method;
import java.lang.reflect.Parameter;
```

```
publicclass Test
{
    publicstaticvoid main(String[] args) {
        Method method;
        try {
                method = Test.class.getMethod( "main", String[].class );
                for( final Parameter parameter: method.getParameters() ) {
                    System.out.println( "Parameter: " + parameter.getName() );
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

如果使用命令 javac Test.java 来编译并运行以上程序，程序的运行结果为 Parameter: args0。如果使用命令 javac Test.java –parameters 来编译并运行以上程序，程序的运行结果为 Parameter: args。

7）新增 Optional 类。在使用 Java 语言进行编程时，经常需要使用大量的代码来处理空指针异常，而这种操作往往会降低程序的可读性。JDK1.8 引入了 Optional 类来处理空指针的情况，从而增强了代码的可读性。下面给出一个简单的例子：

```
public class Test {
    public static void main(String[] args) {
        Optional<String> s1 = Optional.of("Hello");
        //判断是否有值
        if(s1.isPresent())
            System.out.println(s1.get());//获取值
        Optional<String> s2 = Optional.ofNullable(null);
        if(s2.isPresent())
            System.out.println(s2.get());
    }
}
```

这里只是介绍了 Optional 简单的使用示例，读者如果想要了解更多相关内容，可以查看 Java 手册来详细了解 Optional 类的使用方法。

8）新增 Stream 类。JDK1.8 新增加了 Stream 类，从而把函数式编程的风格引入到 Java 语言中，Stream 的 API 提供了非常强大的功能，使用 Stream 后，可以写出更加强大、更加简洁的代码（例如可以代替循环控制语句）。示例代码如下：

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Optional;
import java.util.stream.Collectors;
class Student
{
    private String name;
    private Integer age;
    public Student(String name,int age)
    {
        this.name=name;
        this.age=age;
```

```
        }
        public String getName() {return name;}
        public Integer getAge() {return age;}
    }
    publicclass Test {
        publicstaticvoid main(String[] args) {
            List<Student> l=new ArrayList<>();
            l.add(new Student("Wang",10));
            l.add(new Student("Li",13));
            l.add(new Student("Zhang",10));
            l.add(new Student("Zhao",15));
            System.out.println("找出年龄为 10 的第一个学生：");
            Optional<Student> s=l.stream().filter(stu -> stu.getAge().equals(10)).findFirst();
            if(s.isPresent())
                System.out.println(s.get().getName()+","+s.get().getAge());
            System.out.println("找出年龄为 10 的所有学生：");
            List<Student> searchResult=l.stream().filter(stu -> stu.getAge().equals(10)).collect (Collectors.toList());
            for(Student stu:searchResult)
                System.out.println(stu.getName()+","+stu.getAge());
            System.out.println("对学生按年龄分组：");
            Map<Integer,List<Student>> map=l.stream().collect(Collectors.groupingBy(Student::getAge));
            Iterator<Map.Entry<Integer,List<Student>>> iter = map.entrySet().iterator();
            while (iter.hasNext())
            {
                Map.Entry<Integer, List<Student>> entry = (Map.Entry<Integer, List<Student>>)
iter.next();
                int age=entry.getKey();
                System.out.print(age+":");
                List<Student> group=entry.getValue();
                for(Student stu:group)
                    System.out.print(stu.getName()+"  ");
                System.out.println();
            }
        }
    }
```

程序的运行结果为：

```
找出年龄为 10 的第一个学生：
Wang,10
找出年龄为 10 的所有学生：
Wang,10
Zhang,10
对学生按年龄分组：
10:Wang  Zhang
13:Li
15:Zhao
```

此外，Stream 类还提供了 parallel、map、reduce 等方法，这些方法用于增加对原生类并发处理的能力，有兴趣的读者可以参考 Java 官方文档学习。

9）日期新特性。在 JDK1.8 以前，处理日期相关的类主要有如下三个：

① Calendar：实现日期和时间字段之间转换，它的属性是可变的。因此，它不是线程安全的。

② DateFormat：格式化和分析日期字符串。

③ Date：用来承载日期和时间信息，它的属性是可变的。因此，它不是线程安全的。这些 API 使用起来很不方便，而且有很多缺点，以如下代码为例：

```
Date date = new Date(2015,10,1);
System.out.println(date);
```

在 Date 类传入参数中，月份为 10 月，但输出却为 Mon Nov 01 00:00:00 CST 3915。JDK1.8 对日期相关的 API 进行了改进，提供了更加强大的 API。新的 java.time 主要包含了处理日期、时间、日期/时间、时区、时刻（Instants）和时钟（Clock）等操作。下面给出一个使用示例：

```
import java.time.Clock;
import java.time.Instant;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.LocalTime;
import java.time.ZoneId;

public class Test {
public static void main( String[] args ) {
        //Clock 类通过指定一个时区，可以获取到当前的时刻、日期与时间
        Clock c = Clock.system(ZoneId.of("Asia/Shanghai")); //上海时区
        System.out.println("测试 Clock：");
        System.out.println(c.millis());
        System.out.println(c.instant());

        // Instant 使用方法
        System.out.println("测试 Instant：");
        Instant ist = Instant.now();
        System.out.println(ist.getEpochSecond());//精确到秒
        System.out.println(ist.toEpochMilli()); //精确到毫秒

        //LocalDate 以 ISO-8601 格式显示的日期类型，无时区信息
        LocalDate date = LocalDate.now();
        LocalDate dateFromClock = LocalDate.now( c );
        System.out.println("测试 LocalDate：");
        System.out.println( date );
        System.out.println( dateFromClock );

        //LocalTime 是以 ISO-8601 格式显示时间类型，无时区信息
        final LocalTime time = LocalTime.now();
        final LocalTime timeFromClock = LocalTime.now( c );
        System.out.println("测试 LocalTime：");
        System.out.println( time );
        System.out.println( timeFromClock );

        //LocalDateTime 以 ISO-8601 格式显示的日期和时间
        final LocalDateTime datetime = LocalDateTime.now();
        final LocalDateTime datetimeFromClock = LocalDateTime.now(c);
        System.out.println("测试 LocalDateTime：");
        System.out.println( datetime );
        System.out.println( datetimeFromClock );
    }
```

```
}
```

程序的运行结果为：

```
测试 Clock：
1445321400809
2015-10-20T06:10:00.809Z
测试 Instant：
1445321400
1445321400821
测试 LocalDate：
2015-10-20
2015-10-20
测试 LocalTime：
14:10:00.822
14:10:00.822
测试 LocalDateTime：
2015-10-20T14:10:00.822
2015-10-20T14:10:00.822
```

10）增加了调用 JavaScript 的引擎。JDK1.8 增加 API 使得可以通过 Java 程序来调用 JavaScript 代码，使用示例如下：

```
import javax.script.*;
publicclass Test {
    publicstaticvoid main( String[] args ) throws ScriptException {
            ScriptEngineManager manager = new ScriptEngineManager();
            ScriptEngine engine = manager.getEngineByName( "JavaScript" );
            System.out.println( engine.getClass().getName() );
            System.out.println( engine.eval( "function f() { return 'Hello'; }; f() + ' world!';" ) );
     }
}
```

程序的运行结果为：

```
jdk.nashorn.api.scripting.NashornScriptEngine
Hello world!
```

11）Base64。Base64 编码是一种常见的字符编码，可用来作为电子邮件或 Web Service 附件的传输编码。JDK1.8 把 Base64 编码添加到了标准类库中。示例代码如下：

```
import java.nio.charset.StandardCharsets;
import java.util.Base64;
public class Test {
public static void main( String[] args ) {
        String str = "Hello world";
        String encodStr = Base64.getEncoder().encodeToString( str.getBytes( StandardCharsets.UTF_8 ) );
        System.out.println( encodStr );
        String decodStr = new String(Base64.getDecoder().decode( encodStr ),StandardCharsets.UTF_8 );
        System.out.println( decodStr );
    }
}
```

程序的运行结果为：

```
SGVsbG8gd29ybGQ=
```

```
Hello world
```

12）并行数组。JDK1.8 增加了对数组并行处理的方法（parallelXxx），下面以排序为例介绍其用法。

```
import java.util.Arrays;
public class Test {
    public static void main( String[] args ) {
        int[] arr = {1,5,8,3,19,40,6};
        Arrays.parallelSort( arr );
        Arrays.stream( arr ).forEach(i -> System.out.print( i + " " ) );
        System.out.println();
    }
}
```

【真题 33】 foreach 的作用是什么？

答案：foreach 语句是 java5 的新增的特征之一，是 for 循环语句的简化版本。foreach 的使用可以大大地简化对集合、数组的遍历，给开发人员带来很大的便利。

具体而言，foreach 可以被看作是 for 的子集，能用 foreach 实现的流程控制一定能用 for 实现，但是，需要注意的是，有些 for 语句就无法用 foreach 来实现。而且，foreach 只能进行顺序遍历，无法采用索引的形式来访问数组。

foreach 的语句格式如下：

```
for(类型元素变量 x : 遍历对象 obj){
    //使用 x
}
```

foreach 并不是一个 Java 的关键字，习惯上把这种特殊的 for 语句的格式称为“foreach”语句。

下面给出一个使用 foreach 方法遍历数组的例子：

```
public class Test {
    public static void main(String[] args) {
        int[] arr={1,2,3};
        //遍历一维数组
        for(inti:arr){
            System.out.println(i);
        }
        int arr1[][]={{1,2},{3,4}};
        //遍历二维数组
        for(int[] i:arr1)        {
            for(int j:i)
                System.out.println(j);
        }
    }
}
```

【真题 34】 写代码获取年月日、小时分秒，获取从 1970 年到现在的毫秒数及格式化日期。

答案：System.currentTimeMillis 产生一个当前的毫秒，这个毫秒其实就是自 1970 年 1 月 1 日 0 时起的毫秒数，Date()就相当于 Date(System.currentTimeMillis())，因为 Date 类还有构造方法 Date(long date) 以表示自从标准基准时间（称为“历元（epoch）”，即 1970 年 1 月 1 日 00:00:00 GMT）以来的指定毫秒数。得到了这个毫秒数，也就可以计算出现在的年、月、日、周、时、时区等信息，但是这不是用户自己去计算的，因为有 Calendar，Calendar 最终输出

的结果就是年、月、日、周、时、时区等。

SimpleDateFormat 是一个以与语言环境相关的方式来格式化和分析日期的具体类，它允许进行格式化（日期—>文本）、分析（文本—>日期）和规范化。SimpleDateFormat 使得可以选择任何用户定义的日期–时间格式的模式。

通过以上分析，写出示例代码如下：

```
import java.text.SimpleDateFormat;
import java.util.Date;

public class Test {
    public static void main(String[] args) throws Exception
    {   //获取 1970 年 1 月 1 日到现在的毫秒数
        long l = System.currentTimeMillis();
        System.out.println(l);
        //获取年月日时分秒
        Date date = new Date(l);
        System.out.println(date);
        //格式化输出日期
        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        System.out.println(dateFormat.format(date));
    }
}
```

程序的运行结果为：

```
1443624645840
Wed Sep 30 22:50:45 CST 2015
2015-09-30 22:50:45
```

【真题 35】 如何编写一个 JUnit Test？

答案：JUnit Test 主要用来测试一个小单元的代码（例如，测试一个方法，测试一个类）。对每个单元的代码编写单元测试用例是一个好的编程习惯，它可以帮助开发人员在开发的前期发现代码中的 Bug。目前有两个主流的 JUnit 测试框架版本：JUnit3 和 JUnit4，其中，JUnit3 通过“test”关键字来表示测试方法，而 JUnit4 通过注解（Annotation）来识别测试方法。

下面给出一个 JUnit Test 的例子，首先，实现一个计算器的类如下：

```
public class Calculator {
public int multi(int... n) {
                int result = 0;
                for (int i : n) {
                        result *= i;
                }
                return result;
    }
}
```

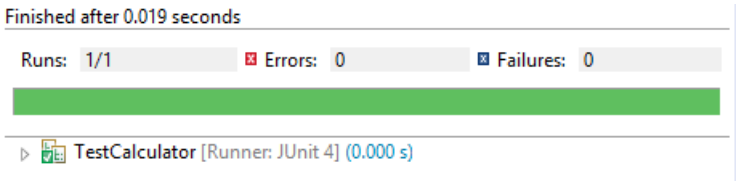
下面介绍一种通过 Eclipse 创建一个 JUnit Test 来测试上面类的方法。

创建 JUnit test 的方法如下：在 Eclipse 中选择 New->JUnit Test Case，接着输入名字：TestCalculator，编写如下测试用例：

```
import static org.junit.Assert.*;
import org.junit.Test;
public class TestCalculator {
```

```
    @Test
    public void test() {
        Calculator c = new Calculator();
         assertEquals(30, c.multi(2,3,5));
    }
}
```

运行时选择 Run As->JUnit Test，就可以得到如下的运行结果：



【真题 36】 SimpleDateFormat 类的作用是什么？

答案：SimpleDateFormat 是一个以国别敏感的方式格式化和分析数据的具体类，主要用来对日期和字符串进行转换，同时可以指定字符串的格式，因此，在项目开发中经常被用到。下面给出用“yyyy/MM/dd HH:mm:ss”形式来显示当前时间的代码：

```
import java.text.SimpleDateFormat;
import java.util.Date;
public class Test {
    public static void main(String[] args)
    {
        SimpleDateFormat sysForm = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
        Date curdate= new Date();
        System.out.println(sysForm.format(curdate));
    }
}
```


程序的运行结果为：

2015/10/13 19:13:23

需要注意的是，DateFormat 类和 SimpleDateFormat 类不都是线程安全的，在多线程环境下，调用 format()方法和 parse()方法应该使用同步代码来避免问题。

【真题 37】 访问修饰符作用范围由大到小是（ ）。

A．private-protected-default-public　　B．public-protected-default-private

C．private-default-protected-public　　D．public-default-protected-private

答案：B。

在 Java 语言中，类的权限访问修饰符有以下几种：private、default（package）、protected 和 public。以下将具体对这几个权限访问修饰符进行介绍。

1）私有权限（private）：private 可以修饰数据成员、构造方法和方法成员，不可以修饰类（此处指外部类，不考虑内部类）。被 private 修饰的成员，只能在定义它们的类中使用，在其他类中不能调用。

2）默认权限（default）：类、数据成员、构造方法和方法成员等都能够使用默认权限，即不被 private、protected 和 public 修饰。默认权限即同包权限，同包权限的元素只能在定

义它们的类中以及同包的类中被调用。
3）受保护权限（protected）：protected 可以修饰数据成员、构造方法和方法成员，不可以修饰类（此处指外部类，不考虑内部类）。被 protected 修饰的成员，能在定义它们的类中以及同包的类中被调用。如果有不同包的类想调用它们，那么这个类必须是它的子类。
4）公共权限（public）：public 可以修饰类、数据成员、构造方法和方法成员。被 public 修饰的成员，可以在任何一个类中被调用，不管同包或不同包，是权限最大的一个修饰符。
以上几种修饰符的作用范围见表 1-2（表中√表示可访问，×表示不可访问）。

表 1-2　修饰符的作用范围

范　　围	private	default	protected	public
同一类	√	√	√	√
同一包中的类	×	√	√	√
同一包中的类、不同包中的子类	×	×	√	√
所有	×	×	×	√

由表 1-2 可知，访问修饰符的作用范围由大到小依次是 public、protected、default 和 private。所以，选项 B 正确。
【真题 38】 以 public 修饰的类如下所示：public class Car{…}，则类 Car（　　）。
A．可被其他程序包中的类使用　　B．不能被其他类继承
C．不能被任意其他类使用　　D．仅能被本程序包中的类使用
答案：A。
对于选项 A 与选项 C，被 public 修饰的类的作用域最大，可以被程序中任意的类使用。因此，选项 A 正确，选项 C 错误。
对于选项 B，只有当一个类被 final 修饰时，才不能被其他类继承。因此，选项 B 错误。
对于选项 D，当作用域为 default 时（不被 public 修饰），仅能被本程序包中的类使用。因此，选项 D 错误。
【真题 39】以下关于被访问控制符 protected 修饰的成员变量的描述中，正确的是（　　）。
A．可以被三种类所引用：该类自身、与它在同一个包中的其他类、在其他包中的该类的子类
B．只能被同一个包中的类访问
C．可以被两种类访问和引用：该类本身、该类的所有子类
D．只能被该类自身所访问和修改
答案：A。
【真题 40】 以下不允许作为类及类成员访问控制符的是（　　）。
A．private　　B．protected　　C．static
D．public
答案：C。
在 Java 语言中，类的权限访问修饰符有以下几种：private、default（package）、protected 和 public，而 static 用来修饰类成员变量或属性。所以，选项 C 错误。
【真题 41】 下面不是 Java 类访问控制关键字的是（　　）。
A．private　　B．protected　　C．this　　D．public
答案：C。
在 Java 语言中，类的权限访问修饰符有以下几种：private、default（package）、protected 和 public。而关键字 this 用来指向当前实例对象，它的一个非常重要的作用就是用来区分对象

的成员变量与方法的形参（当一个方法的形参与成员变量有着相同名字的时候，就会覆盖成员变量）。为了能够对关键字 this 有一个更好的认识，首先创建一个类 People，示例如下：

```
class People
{
    String name;
    //正确的写法
    public People(String name) {        this.name=name;}
    //错误的写法
     public People(String name) {name=name;}
}
```

上例中，第一个构造方法使用 this.name 来表示左边的值为成员变量，而不是这个构造方法的形式参数。对于第二个构造方法，由于在这个方法中形参与成员变量有着相同的名字，因此，对于语句 name=name，等号左边和右边的两个 name 都代表的是形式参数。在这种情况下，只有通过 this 才能访问到成员变量。

所以，本题的答案为 C。

【真题 42】 有如下代码：

```
public class Person
{
    private String name="Person";
    int age=0;
}
public class Child extends Person
{
    public String grade;
    public static void main(String[] args)
    {
        Person p = new Child();
        System.out.println(p.name);
    }
}
```

以上代码的运行结果是（　　）。

A．Person　　B．没有输出　　C．编译出错　　D．运行出错

答案：C。

由于 name 被 private 修饰，因此，它对 Person 的实例以及 Child 类都不可见。因此，p.name 的写法是错误的，会导致编译报错。如果把 private String name="Person"中的 private 改成 protected，那么程序将会输出 Person。所以，选项 C 正确。

所以，本题的答案为 C。

【真题 43】 在 Java 程序中定义一个类，类中有一个没有访问权限修饰的方法，下面关于此方法的描述中，正确的是（　　）。

A．类的子类和同包类能访问它　　B．类外的任何方法都能访问它

C．类外的任何方法都不能访问它　　D．只有类和同包类才能访问它

答案：A。

【真题 44】 以下关于 import java.util 包的描述中，错误的是（　　）。

A．Vector 类放在/java/util/目录下　　B．Vector 类属于 java.util 包
C．Vector 类放在 java.util 文件中　　D．Vector 类是 Sun 公司的产品
答案：C。
在 Java 语言中，包是一个比较抽象的逻辑概念，它的宗旨是把.java 文件（Java 源文件）、.class 文件（编译后的文件）以及其他 resource 文件（例如.xml 文件、.avi 文件、.mp3 文件、.txt 文件等）有条理地进行组织，以供使用。它类似于 Linux 系统的文件系统，有一个根，然后从根开始有目录和文件，目录中嵌套有目录。
对于本题而言，java.util 是包名，实质上是一个目录结构，在这个包中，Java 语言提供了一些实用的方法和数据结构。例如，日期（Data）类、日历（Calendar）类用于产生和获取日期及时间，随机数（Random）类用于产生各种类型的随机数，除此以外，还提供了堆栈（Stack）、向量（Vector）、位集合（Bitset）以及散列表（Hashtable）等类。因此，选项 C 错误，因为 java.util 不是一个文件，而是一个/java/util 目录。
【真题 45】 创建一个名为 MyPackage 包的语句是（　　），该语句应该放在程序中位置为（　　）。
答案：package MyPackage;，程序第一句。
具体而言，package 主要有两个作用：第一，提供多层命令空间，解决命名冲突，通过使用 package，使得处于不同 package 中的类可以存在相同的名字。第二，对类按功能进行分类，使项目的组织更加清晰。当开发一个有非常多的类的项目时，如果不使用 package 对类进行分类，而是把所有的类都放在一个 package 下，这样的代码不仅可读性差，而且可维护性也不好，会严重影响开发效率。
package 的用法如下（源文件所在目录为当前目录）：在每个源文件的开头（必须在开头）加上"package packagename;"。
【真题 46】 下面说法正确的是（　　）。
A．如果源代码中有 package 语句，则该语句必须被放在代码的第一行（不考虑注释和空格）
B．如果源代码中有 main()方法，则该方法必须被放在代码的第一行
C．如果源代码中有 import 语句，则该语句必须被放在在代码的第一行（不考虑注释和空格）
D．如果某文件的源代码中定义了一个 public 的接口，则接口名和文件名可以不同
答案：A。
package 是 Java 语言所特有的内容，它的作用就是把若干类按包结构进行分类管理，最重要的用途是为了解决同名但作用不同的类同时存在的问题。
import 语句允许开发人员在编译时将其他类的源代码包含到源文件中，具体而言，import 语句包括 import 关键字、以点(.)分隔的包路径、类名或星号(*)。需要注意的是，每条 import 语句只可以对应一　　个包。
在 Java 语言中，package 语句必须作为 Java 源文件的第一条语句，指明该文件中定义的类所在的包。所以，如果代码中有 package 语句，则必须放在最前面，即该语句必须放在代码的第一行（不考虑注释和空格）。因此，选项 A 正确，选项 B 和选项 C 错误。
对于选项 D，Java 类或接口有如下命名规则：被 public 修饰的类或者接口必须与文件名相同。因此，选项 D 错误。
【真题 47】 下列关于 package 和 import 语句的描述中，错误的是（　　）。
A．同一个类中 package 语句可以出现 1 次或多次
B．同一个类中 import 语句可以出现 1 次或多次
C．同一个类中 import 语句必须出现在该类的第一行（不含注释）
D．同一个类中 package 语句必须出现在该类的第一行（不含注释）
答案：A、C。

【真题 48】 main 方法的声明格式包括什么？

答案：public static void main(String[] args)。

【真题 49】 下列关于 Java 语言中 main 方法的描述中，正确的是（　　）。

A．Java 程序的 main 方法必须写在类里面

B．Java 程序中可以有多个 main 方法

C．Java 程序的 main 方法中，如果只有一条语句，可以不用大括号{}括起来

D．Java 程序中类名必须与文件名一样

答案：A、B。

在 Java 语言中，main 方法是程序的入口方法，一个程序要想运行必须要有 main 方法，但是只有满足特定条件的 main 方法才能作为程序的入口方法。

本题中，对于选项 A，由于 Java 语言是纯面向对象语言，所以，所有的属性与方法都必须定义在类里面，而且，main 方法也不例外。因此，选项 A 正确。

对于选项 B，Java 程序可以定义多个 main 方法，但是只有 public static void main(String[] args) 方法才是 Java 程序的入口方法，其他 main 方法都不是，并且这个入口方法必须被定义在类名与文件名相同的被 public 修饰的类中，如下例所示（Test.java）：

```
class T{
    public static void main(String[] args) {
        System.out.println("T main");
    }
}
public class Test {
    // 程序入口方法
    public static void main(String[] args) {
        System.out.println("Test main");
    }
}
```

程序的运行结果为：

```
Test main
```

从上例可以看出，这个程序中定义了多个 main 方法，但是只有满足特定条件的 main 方法才能作为程序的入口方法。因此，选项 B 正确。

对于选项 C，在 Java 语言中，不管方法体里有几条语句，所有的方法体都必须用大括号{}括起来。因此，选项 C 错误。

对于选项 D，在 Java 语言中，一个文件内部可以有多个类的存在，但只有被 public 修饰的类的名字与文件的名字相同，其他类的名字可以根据需求随意起名字。因此，选项 D 错误。

【真题 50】 下面关于 main 方法的方法头的定义中，合法的是（　　）。

A．public　static　void　main()　　B．public static void main(String　args[])

C．public　void　main(String　arg[])　　D．public static int　main(String　[] arg)

答案：B。

【真题 51】 有如下代码：

```
class A
{
```

```
        public int f(int a)
        {
            return a+1;
        }
}
class B extends A
{
        public int f(int a, char c)
        {
            return a+2;
        }
        public static void main(String[] args)
        {
            B b=new B();
            System.out.println(b.f(0));
        }
}
```

当编译并运行上面程序时（文件名为 Test.java），输出结果是（　　）。

A．编译错误　　B．运行错误　　C．1　　D．2

答案：B。

public static void main(String[] args)为 Java 程序的入口方法，JVM 在运行程序的时候，会首先查找 main 方法。需要注意的是，只有与文件名相同的类中的 main 方法才能作为整个程序的入口方法。

对于本题而言，文件 Test.java 中没有 Test 类，因此，这个类里面的 main 方法被看作是一个普通的方法，而不是程序的入口方法。所以，在运行的时候，由于 JVM 找不到程序的入口方法，程序会运行错误。所以，选项 B 正确。

如果本题把文件名由 Test.java 改为 B.java，那么此时程序的运行结果就为 1。

【真题 52】 有如下代码：

```
class A
{
    public A()
    {
        System.out.println("A");
    }
}
class B extends A
{
    public B()
    {
        System.out.println("B");
    }
    public static void main(String[] args)
    {
        B b=new B();
    }
}
```

上述程序将（　　）。

A．不确定　　B．通过编译，输出为 AB

C．通过编译，输出为 B　　D．通过编译，运行时错误

答案：A。

本题中，当这个程序所在的 Java 文件名为 B.java 时，运行结果为 AB，否则，编译能通过，运行时会出错，因为找不到程序的入口方法 main。所以，选项 A 正确。

第 2 章　软件工程与设计模式

2.1　软件工程与 UML

【真题 372】 敏捷软件开发方法是一种（　　）。

A．数学观　　B．建模观　　C．工程观　　D．协作观

答案：D。

敏捷软件开发方法是一种应对快速变化的需求的软件开发能力。它们的具体名称、理念、过程和术语都不尽相同，相对于“非敏捷”，敏捷更强调程序员团队与业务专家之间的紧密协作、面对面的沟通（认为比书面的文档更有效）、频繁交付新的软件版本、紧凑而自我组织型的团队、能够很好地适应需求变化的代码编写和团队组织方法，也更注重作为软件开发中人的作用。所以，敏捷软件开发方法是一种创作与交流的协作观。所以，选项 D 正确。

【真题 373】 极限编程 XP 的核心思想是（　　）。

A．强调文档和以敏捷性应对变化

B．强调建模和以敏捷性应对变化

C．强调设计和以敏捷性应对变化

D．强调人和人之间的合作因素和以敏捷性应对变化

答案：D。

极限编程（Extreme Programming，XP）是一种轻量级的、灵巧的软件开发方法，同时，它也是一种非常严谨和周密的方法。它的基础和价值观是交流、朴素、反馈和勇气，即任何一个软件项目都可以从四个方面入手进行改善：加强交流；从简单做起；寻求反馈；勇于实事求是。它是敏捷开发的典型代表，其核心思想是强调人和人之间的合作因素和以敏捷性应对变化。所以，选项 D 正确。

【真题 374】 净室软件工程（Cleanroom）是软件开发的一种形式化方法，可以开发较高质量的软件，它发现和排除错误的主要机制是（　　）。

A．正确性验证　　B．黑白盒测试　　C．集成测试　　D．基本路径测试

答案：A。

净室软件工程是一种应用数学与统计学理论以经济的方式生产高质量软件的工程技术，力图通过严格的工程化的软件过程达到开发中的零缺陷或接近零缺陷。它提倡开发者不需要进行单元测试，而是进行正确性验证和统计质量控制。所以，选项 A 正确。

2.2　设计模式

【真题 383】 什么是设计模式？有哪些常见的设计模式？

答案：设计模式（Design Pattern）是一套被反复使用、多数人知晓的、经过分类编目的、代码设计经验的总结。使用设计模式的目的是为了代码重用，避免程序大量修改，同时使代码更容易被他人理解，并且保证代码可靠性。显然，设计模式不管是对自己还是对他人抑或是对系统都是有益的，设计模式使得代码编制真正地工程化，设计模式可以说是软件工程的基石。

GoF（Gang of Four）23 种经典设计模式见表 2-1。

表 2-1　GoF 经典设计模式

	创建型	结构型	行为型
类	Factory Method（工厂方法）	Adapter_Class（适配器类）	Interpreter（解释器） Template Method（模板方法）
对象	Abstract Factory（抽象工厂） Builder（生成器） Prototype（原型） Singleton（单例）	Adapter_Object（适配器对象） Bridge（桥接） Composite（组合） Decorator（装饰） Façade（外观） Flyweight（享元） Proxy（代理）	Chain of Responsibility（职责链） Command（命令） Iterator（迭代器） Mediator（中介者） Memento（备忘录） Observer（观察者） State（状态） Strategy（策略） Visitor（访问者模式）

常见的设计模式有工厂模式（Factory Pattern）、单例模式（Singleton Pattern）、适配器模式（Adapter Pattern）、享元模式（Flyweight Pattern）以及观察者模式（Observer Pattern）等。

工厂模式专门负责实例化有大量公共接口的类。工厂模式可以动态地决定将哪一个类实例化，而不必事先知道每次要实例化哪一个类。客户类和工厂类是分开的。消费者无论什么时候需要某种产品，需要做的只是向工厂提出请求即可。消费者无须修改就可以接纳新产品。当然也存在缺点，就是当产品修改时，工厂类也要做相应的修改。

工厂模式包含以下几种形态：

1）简单工厂（Simple Factory）模式。简单工厂模式的工厂类是根据提供给它的参数，返回几个可能产品中的一个类的实例，通常情况下它返回的类都有一个公共的父类和公共的方法。设计类图如图 2-1 所示。

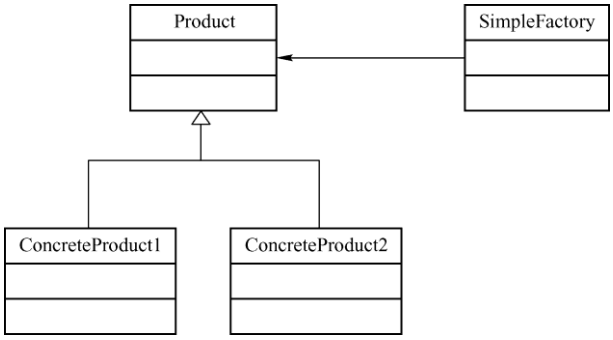


图 2-1　简单工厂模式设计类图

其中，Product 为待实例化类的基类，它可以有多个子类；SimpleFactory 类中提供了实例化 Product 的方法，这个方法可以根据传入的参数动态地创建出某一类型产品的对象。

2）工厂方法（Factory Method）模式。工厂方法模式是类的创建模式，其用意是定义一个用于创建产品对象的工厂的接口，而将实际创建工作推迟到工厂接口的子类中。它属于简单工厂模式的进一步抽象和推广。多态的使用，使得工厂方法模式保持了简单工厂模式的优点，而且克服了它的缺点。设计类图如图 2-2 所示。

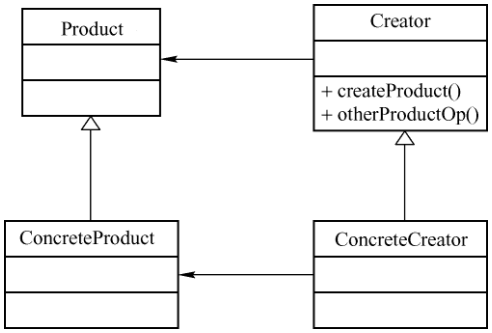


图 2-2　工厂方法模式设计类图

Product 为产品的接口或基类，所有的产品都实现这个接口或抽象类（例如 ConcreteProduct），这样就可以在运行时根据需求创建对应的产品类。Creator 实现了对产品所有的操作方法，而不实现产品对象的实例化。产品的实例化由 Creator 的子类来完成。

3）抽象工厂（Abstract Factory）模式。抽象工厂模式是所有形态的工厂模式中最为抽象和最具一般性的一种形态。抽象工厂模式是指当有多个抽象角色时使用的一种工厂模式，抽象工厂模式可以向客户端提供一个接口，使客户端在不必指定产品的具体的情况下，创建多个产品族中的产品对象。根据 LSP 原则（即 Liskov 替换原则），任何接受父类型的地方，都应当能够接受子类型。因此，实际上系统所需要的，仅仅是类型与这些抽象产品角色相同的一些实例，而不是这些抽象产品的实例。换句话说，也就是这些抽象产品的具体子类的实例。工厂类负责创建抽象产品的具体子类的实例。设计类图如图 2-3 所示。

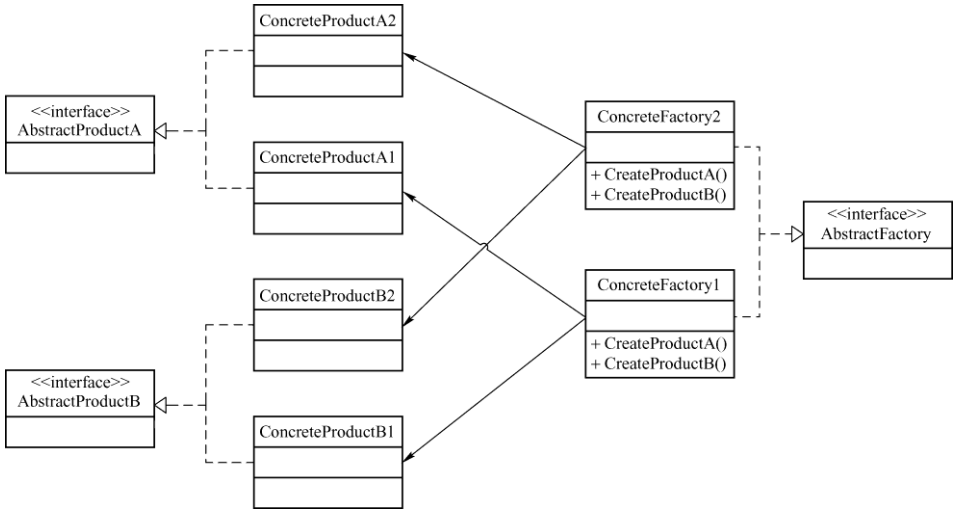


图 2-3　抽象工厂模式设计类图

AbstractProductA 和 AbstractProductB 代表一个产品家族，实现这些接口的类代表具体的产品。AbstractFactory 为创建产品的接口，能够创建这个产品家族中所有类型的产品，它的子类可以根据具体情况创建对应的产品。

【真题 384】 用 Java 语言实现一个观察者模式。

答案：观察者模式（也称为发布/订阅模式）提供了避免组件之间紧密耦合的另一种方法，它将观察者和被观察的对象分离开。在该模式中，一个对象通过添加一个方法（该方法允许另一个对象，即观察者注册自己）使本身变得可观察。当可观察的对象更改时，它会将消息发送到已注册的观察者。这些观察者收到消息后所执行的操作与可观察的对象无关，这种模式使得对象可以相互对话，而不必了解原因。Java 语言与 C#语言的事件处理机制就是采用的此种设计模式。

例如，用户界面（同一个数据可以有多种不同的显示方式）可以作为观察者，业务数据是被

观察者，当数据有变化后会通知界面，界面收到通知后，会根据自己的显示方式修改界面的显示。面向对象设计的一个原则是：系统中的每个类将重点放在某一个功能上，而不是其他方面。一个对象只做一件事情，并且将它做好。观察者模式在模块之间划定了清晰的界限，提高了应用程序的可维护性和重用性。设计类图如图 2-4 所示。

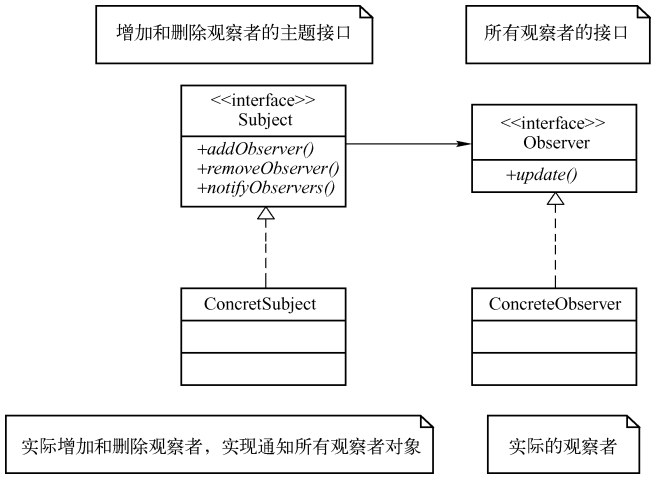


图 2-4　观察者模式设计类图

下面给出一个观察者模式的示例代码，代码的主要功能是实现天气预报，同样的温度信息可以有多种不同的展示方式：

```
import java.util.ArrayList;
interface Subject
{
    public void registerObserver(Observer o);
    public void removeObserver(Observer o);
    public void notifyObservers();
}
class Whether implements Subject
{
    private ArrayList<Observer>observers=new ArrayList<Observer>();
    private float temperature;
    @Override
    public void notifyObservers() {
        for(int i=0;i<this.observers.size();i++)
        {
            this.observers.get(i).update(temperature);
        }
    }
    @Override
    public void registerObserver(Observer o) {
        this.observers.add(o);
    }
    @Override
    public void removeObserver(Observer o) {
        this.observers.remove(o);
    }
    public void whetherChange()     {
        this.notifyObservers();
    }
    public float getTemperature(){
```

```
            return temperature;
        }
        public void setTemperature(float temperature) {
            this.temperature = temperature;
            notifyObservers();
        }
}
interface Observer
{
        //更新温度
        public void update(float temp);
}
class WhetherDisplay1 implements Observer
{
        private float temprature;
        public WhetherDisplay1(Subject whether){
            whether.registerObserver(this);
        }
        @Override
        public void update(float temp) {
            this.temprature=temp;
            display();
        }
        public void display(){
            System.out.println("display1****:"+this.temprature);
        }
}
class WhetherDisplay2 implements Observer
{
        private float temprature;
        public WhetherDisplay2(Subject whether)
        {
            whether.registerObserver(this);
        }
        @Override
        public void update(float temp) {
            this.temprature=temp;
            display();
        }

        public void display()
        {
            System.out.println("display2----:"+this.temprature);
        }
}
public class Test
{
        public static void main(String[] args)
        {
            Whether whether=new Whether();
            WhetherDisplay1 d1=new WhetherDisplay1(whether);
            WhetherDisplay2 d2=new WhetherDisplay2(whether);
            whether.setTemperature(27);
```

```
                whether.setTemperature(26);
            }
        }
```

3.2 数据库设计

【真题 418】 一个人存在于社区中，会有各种各样的身份，和不同的人相处会有不同的关系。请自行设计数据库（表结构，个数不限），保存一个人的名字、关系（包括父亲、朋友们），并用尽可能少的时间空间开销组织好每个人和其他人的关系，组织好后尝试取出一个人的关系结构。其中涉及的 SQL 语句请详细写出。涉及的数据结构、数据组织形成也请描述清楚，代码可以用伪代码或你熟悉的任何代码给出。

答案：这道题要求存储三类信息：用户信息、关系信息和用户之间的关系信息。涉及的表见表 3-2～表 3-4。

（1）用户表：存储用户基本信息

create table user_info(user_id int primary key, user_name varchar(30) ,user_age int);（这个主键可以使用数据库自增的方式来实现，不同的数据库定义的方法有所不同）

表 3-2 用户表

user_id	user_name	user_age
1	James	18
2	Ross	25
3	Jack	50

（2）用户关系定义表：主要存储用户之间所有可能的关系

create table relation_define(relation_id int primary key, relation_name varchar2(32));

表 3-3 用户关系定义表

relation_id	relation_name
1	同事
2	父子
3	朋友

（3）用户关系信息表：存储用户关系信息

create table user_relation(user_id int, rel_user_id int,relation_id int);

表 3-4 用户关系信息表

user_id	rel_user_id	relation_id
1	2	1
2	3	2
1	3	3

表 3-4 中数据表示 1（James）和 2（Ross）是同事关系，3（Jack）和 2（Ross）是父子关系，1（James）和 3（Jack）是朋友关系。

示例：查询用户 1 的社会关系。

```
select a.user_name,b.relation_name  from user_info a, relation_define b,
(select user_id,relation_id from user_relation where rel_user_id =1 union select rel_user_id as user_id,relation_id from user_relation where user_id =1) c
where a.user_id=c.user_id and b.relation_id=c.relation_id
```

运算结果见表 3-5。

表 3-5　运算结果

user_name	relation_name
Ross	同事
Jack	朋友

【真题 419】 有如下学生信息：

学生表 student(stu_id ,stu_name);

课程表 course(c_id,c_name);

成绩表 score(stu_id ,c_id,score);

1）写出向学生表中插入一条数据的 SQL 语句。

2）查询名字为 James 的学生所选的课程。

3）查询 stu_id 为 4 的学生所学课程的成绩。

答案：

1）向数据库中插入一条记录用的是 insert 语句，可以采用如下两种写法：

① insert into student(stu_id，stu_name) values(1,'james')

② insert into student values(1,'james')

如果这个表的主键为 stu_id，并且采用数据库中自增的方式生成，那么在插入的时候就不能显式地指定 stu_id 这一列，在这种情况下，添加记录的写法为：

```
insert into student(stu_name) values('james')
```

2）在数据库中查询用到的关键字为 select，由于 student 表中只存放了学生相关的信息，course 表中只存放了课程相关的信息，学生与课程是通过 score 表来建立关系的，一种思路为：首先找到名字为 Tom 的学生的 stu_id，然后在成绩表（score）中根据 stu_id 找出这个学生所选课程的 c_id，最后就可以根据 c_id 找出这个学生所选的课程。

① 可以使用下面的 select 语句来查询：

```
select c_name from course where c_id in (select c_id from score where stu_id in (select std_id from student where stu_name='Tom'))
```

② 当然也可以根据题目要求，根据三张表的关系，直接执行 select 操作，写法如下：

```
select c_name from student st, course c，score sc where st. stu_id=sc. stu_id and sc. c_id=c.c_id and st. stu_name='Tom'
```

③ 当然也可以把②的写法改为对三个表做 join 操作。

3）成绩都存在表 score 中，而课程名存储在表 course 中，因此，需要访问这两张表来找出课程与成绩，实现方法如下：

```
select c.c_name, s. score from course c，score s where s. stu_id =4 and c. c_id=s. c_id
```

【真题 420】 定义有表结构如下所示：

（1）表名：g_cardapply

字段（字段名/类型/长度）：

```
g_applyno      varchar   8；            //申请单号（关键字）
```

g_applydate bigint 8； //申请日期
g_state varchar 2； //申请状态

（2）表名：g_cardapplydetail
字段（字段名/类型/长度）：

g_applyno varchar 8； //申请单号（关键字）
g_name varchar 30； //申请人姓名
g_idcard varchar 18； //申请人身份证号
g_state varchar 2； //申请状态

其中，两个表的关联字段为申请单号。
题目：1）查询身份证号码为 612301430103082 的申请日期。
2）查询同一个身份证号码有两条以上记录的身份证号码及记录个数。
3）将身份证号码为 612301430103082 的记录在两个表中的申请状态均改为 15。
4）删除 g_cardapplydetail 表中所有姓张的记录。
答案：1）主要思路为：从 g_cardapplydetail 中找到身份证号对应的 g_applyno，然后根据 g_applyno 在表 g_cardapply 中找出申请日期即可，下面给出两种写法：
① select t1.g_applydate from g_cardapply t1, g_cardapplydetail t2 where t2.g_idcard= '612301430103082' and t1.g_applyno=t2.g_applyno
② select g_applydate from g_cardapply where g_idcard in (select g_idcard from g_cardapplydetail where g_idcard='612301430103082')
2）主要思路为：首先按身份证号码进行分组，然后统计每个身份证出现的次数，最后把出现次数大于或等于 2 的信息查询出来，SQL 语句如下：

select g_idcard, count(g_idcard) as num from g_cardapplydetail group by g_idcard having count(g_idcard)>1

3）更新记录需要使用 update 语句，可以使用两条 SQL 语句分别更新两张表：

updateg_cardapplydetailset g_state='15' where g_idcard='612301430103082'
updateg_cardapply set g_state='15' where g_applyno in (select g_applyno from g_cardappdetail where g_idcard='612301430103082')

当然也可以把这两个 SQL 语句写到一个存储过程里面，存储过程的参数为身份证号码。
为了保持数据库中数据的一致性，最好把这两条 update 语句放到一个事务中。
4）本题考查的是对 like 子句模糊查询的理解，SQL 语句如下：
delete from g_cardapplydetailwhere g_name like '张%'
【真题 421】 一个简单的论坛系统中数据库扮演着非常重要的角色，假设数据库需要存储如下的一些数据：用户名、email、主页、电话、联系地址、发帖标题、发帖内容、回复标题及回复内容。每天论坛访问量 400 万左右，更新帖子 10 万左右。请给出数据库表结构设计，并结合范式简要说明设计思路。
答案：1）在论坛系统中，最重要的对象就是用户与帖子。显然，可以给用户单独设计一张表，由于帖子对象比较特殊，每个帖子都会有回复帖，而回复帖也会有回复帖，如此递归。由于论坛中会有大量的帖子，因此，对帖子表的设计是非常重要的。为了提高查询效率，在设计的时候可以把主题帖与回复帖分开为两张表；对于回复帖的回复帖，可以考虑在回复帖的表中增加一个额外的字段（回复帖子的 id）。
2）E-R 图设计：通过以上分析可知，这个简单的论坛系统主要有 3 个实体：用户 t_user、主题帖 t_mainPost 及回复帖 t_replayPost。它们之间有如下关系：
① 一个用户可以发 0 个或多个主题帖，因此，t_user 与 t_mainPost 的关系为一对多的关

系。

② 一个用户可以有 0 个或多个回复帖，因此，t_user 与 t_replayPost 的关系为一对多的关系。

③ 一个主题帖可以有 0 个或多个回复帖，因此，t_mainPost 与 t_replayPost 的关系也是一对多的关系。

通过以上分析，下面给出数据库设计的 E-R 图，如图 3-1 所示。

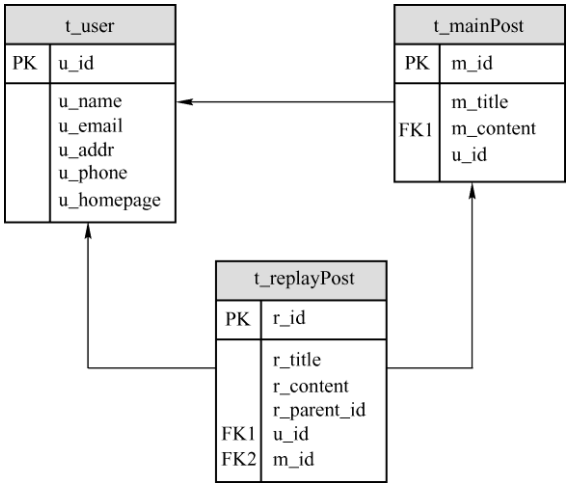


图 3-1　E-R 图

3）E-R 图转关系模型：当把 E-R 图转换为表时需要考虑如下规则：

每一个实体可以转换为一张表，实体的属性就是表的列。实体关系的转换需要遵循下面的规则：

① 一个 1：1 的关系可以有如下两种转换方式：

a）创建单独的关系表，则这个表中的主要内容为 1：1 关系的两个表的主键。

b）两个实体合并为一张表，把两个表的属性合并，创建一张表。

② 一个 1：n 的关系可以有如下两种转换方式：

a）创建单独的关系表，则这个表中的主要内容为 1：n 关系的两个表的主键。

b）通过在 n 端的表中引入一列（1 端表的主键）作为外键，一般采用这种方式来减少表的个数，从而提高查询效率。

③ 一个 m：n 关系转换为一个关系模式。只能创建单独的关系表，关系表中的主要内容为两个表的主键。

上面设计的 E-R 图只有 1：n 的关系，通过在 n 端引入 1 端实体的主键，得到数据库表结构见表 3-6～表 3-8。

t_user 表（用户信息表）：

表 3-6　t_user 表

列名	类型	键	描述
u_id	long	主键	用户 ID
u_name	varchar(20)		用户名
u_email	varchar(30)		用户邮箱
u_addr	varchar(100)		用户地址
u_phone	varchar(20)		用户号码
u_homepage	varchar(50)		用户主页 url

t_mainPost 表（主题帖表）：

表 3-7　t_mainPost 表

列名	类型	键	描述
m_id	long	主键	帖子 ID
m_title	varchar(50)		帖子主题
m_content	Varchar(1000)		帖子内容
u_id	long	外键	用户 ID

t_replayPost 表（回复帖表）：

表 3-8　t_replayPost 表

列名	类型	键	描述
r_id	long	主键	回复帖子 ID
r_title	varchar(50)		回复帖子主题
r_content	Varchar(1000)		回复帖子内容
u_id	long	外键	用户 ID
m_id	long	外键	
r_paient_id	long		回复帖的父帖 ID

4）分析：

① 显然，表中每个字段不可再分，因此，满足 1NF。

② 表中的每一行都可以唯一地用 id 区分，且不存在部分依赖，因此，满足 2NF。

③ t_replayPost 表存在传递依赖（r_id--> r_paient_id，r_paient_id -->m_id），因此，这个设计不满足 3NF。

数据库的范式主要目的是防止数据冗余、更新异常、插入异常和删除异常，范式越高，冗余越少。但是高的范式可能会带来处理速度缓慢和处理逻辑复杂的问题。因此，并不是范式越高越好，在实际设计时，需要权衡范式与效率，而不能盲目地追求高范式而忽视效率。对于本题而言，t_replayPost 被设计为不满足 3NF，虽然增加了冗余，但是能明显地提高效率。

第 6 章　数据结构与算法

数组与线性表

【真题 553】 对于顺序存储的线性数组，访问结点和增加结点、删除结点的时间复杂度分别为（　　）。

A．O(n)，O(n)　　B．O(n)，O(1)　　C．O(1)，O(n)　　D．O(n)，O(n)

答案：C。

对于线性数组，它支持随机访问，因此，访问结点的时间复杂度为 O(1)，增加结点、删除结点的时候需要移动新增结点或待删除结点后面的元素，因此，时间复杂度为 O(n)。所以，选项 C 正确。

【真题 554】 在有 n 个结点的顺序表中，算法的时间复杂度是 O(1)的操作是（　　）。

A．访问第 i 个结点（1<=i<=n）和求第 i 个结点的直接前驱（2<=i<=n）

B．在第 i 个结点后插入一个新结点（1<=i<=n）

C．删除第 i 个结点（1<=i<=n）

D．将 n 个结点从小到大排序

答案：A。

线性表也叫顺序表，在线性表中的数据元素，其关系是一一对应的，即除了第一个数据元素和最后一个数据元素之外，其他数据元素都是首尾相接的。

本题中，对于选项 A，线性表是随机存取结构，当对其执行插入和删除操作时，只要不是针对最后一个元素，此时都需要进行元素的搬家，最坏情况下的时间复杂度是 O(n)。因此，访问第 i 个结点（1<=i<=n）和求第 i 个结点的直接前驱（2<=i<=n），其时间复杂度都为 O(1)。所以，选项 A 正确。

对于选项 B 和选项 C，由于插入和删除操作都需要移动元素，此时算法的时间复杂度为 O(n)，它与题目要求的 O(1)的时间复杂度不相符。所以，选项 B 与选项 C 错误。

对于选项 D，将 n 个结点从小到大排序的时间复杂度通常介于 O(n)与 O(n^2)之间，它与题目要求的 O(1)的时间复杂度不相符。所以，选项 D 错误。

【真题 555】 以下操作中，数组比线性表速度更快的是（　　）。

A．原地逆序　　B．头部插入　　C．返回中间结点

D．返回头部结点　　E．选择随机结点

答案：A、C、E。

线性结构的基本特征为：

1）集合中必然存在唯一的一个“第一元素”。

2）集合中必然存在唯一的一个“最后元素”。

3）除最后一个元素外，均有唯一的后继。

4）除第一个元素外，均有唯一的前驱。

数组是随机存取的，线性表是逻辑上连续但物理上分开存放的，因此，查询、修改操作数组更快，但插入、删除等操作线性表更快。所以，选项 B 与选项 D 错误，选项 E 正确。

对于选项 A，当需要进行原地逆序时，数组比线性表速度更快。对于数组的逆序，具体做法如下：定义两个下标，一个下标 i 表示数组首元素，一个下标 j 表示数组尾元素，交换 i 与 j

两个位置的元素值，同时执行++i，--j 操作，一共需要经过 n/2 次交换（n 表示数组的长度）。而链表的逆序需要修改指针的指向，需要更多的操作。所以，选项 A 正确。
对于选项 C，数组可以通过 array[length/2]访问中间结点，链表需要依次查找到中间结点，所以，数组比线性表更快。所以，选项 C 正确。
所以，本题的答案为 A、C、E。
【真题 556】 二叉树是非线性数据结构，以下关于其存储结构的描述中，正确的是（　　）。
A．它不能用链式存储结构存储
B．它不能用顺序存储结构存储
C．顺序存储结构和链式存储结构都不能使用
D．顺序存储结构和链式存储结构都能存储
答案：D。
二叉树是非线性数据结构，即每个数据结点至多只有一个前驱，但可以有多个后继，可以使用顺序存储和链式存储两种结构来存储。以下将分别对这两种存储结构进行介绍。
（1）顺序存储结构
二叉树的顺序存储指的是用元素在数组中的下标表示一个结点与其孩子和父结点的关系。这种结构特别适用于近似满二叉树。这种方法的缺点是可能会有大量空间的浪费，在最坏的情况下，一个深度为 k 且只有 k 个结点的右单支树需要 2^k−1 个结点存储空间。图 6−1 和图 6−2 分别给出完全二叉树和非完全二叉树的存储示意图。

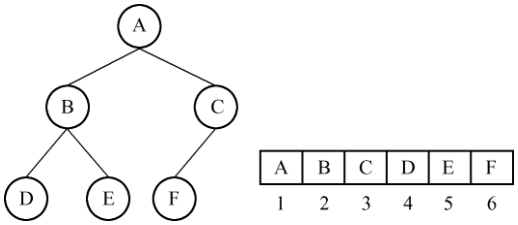


图 6−1　完全二叉树的存储方式

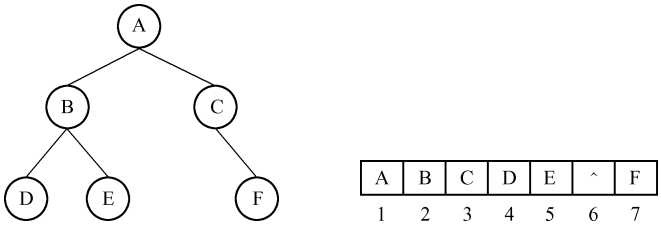


图 6−2　非完全二叉树的存储方式

（2）链式存储结构
二叉树的链式存储结构是指用链表来表示一棵二叉树。
每个结点有一个数据域，两个指针域分别指向左孩子和右孩子。其结点结构为：

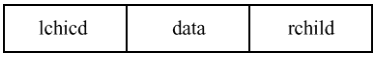
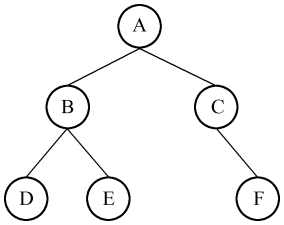


图 6−3 给出了一个二叉树的链表存储方式。



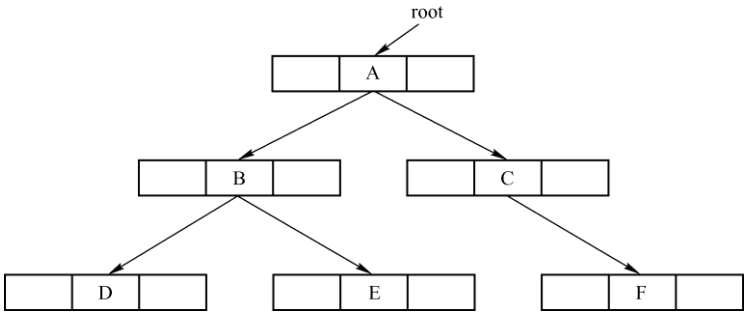


图 6-3　链表存储方式

通过上面的分析可知，选项 D 正确。

【真题 557】 下列数据结构中，同时具有较高的查找和删除性能的是（　　）。

A．有序数组　　B．有序链表　　C．AVL 树　　D．Hash 表

答案：C、D。

首先介绍常见的数据结构的操作性能：

1）有序数组：查找的时候可以采用二分查找法，因此，查找的时间复杂度为 O(logn)，其中，n 表示数组序列的长度。由于数组中的元素在内存中是顺序存放的，因此，删除数组中的一个元素后，数组中后面的元素就需要向前移动。在最坏的情况下，如果删除数组中的第一个元素，那么数组中后面的 n-1 个元素都需要向前移动，移动操作的次数为 n-1，因此，此时的时间复杂度为 O(n)。插入操作与删除操作类似，都需要数组中元素的移动，因此，其时间复杂度也为 O(n)。

2）有序链表：链表（以单链表为例）的存储特点为：每个结点的地址存储在它的前驱结点的指针域中，对链表的遍历只能从链表的首结点开始遍历，因此，此时查找的时间复杂度为 O(n)，其中，n 表示链表的长度。对于删除和插入操作，虽然删除和插入操作的时间复杂度都为 O(1)（因为不需要结点的移动操作），但是在删除前首先需要找到待删除结点的地址，这个操作的时间复杂度为 O(n)，在插入结点前首先也要找到结点应该被插入的地方，这个操作的时间复杂度也为 O(n)，因此，插入与删除的时间复杂度都为 O(n)。

3）AVL 树（平衡二叉树）：AVL 树是一棵空树或它的左右两个子树的高度差的绝对值不超过 1，并且左右两个子树都是一棵平衡二叉树。由于树的高度为 logn，其中，n 表示树中结点的个数，因此，查找的时间复杂度为 O(logn)，显然，删除与插入的时间复杂度也为 O(logn)。

4）Hash 表：Hash 表通过 Hash 值就可以定位到元素的位置，因此，查找、插入与删除的时间复杂度都为 O(1)。

5）普通数组：查找的时候只能顺序地遍历数组，在最坏的情况下需要对数组中所有的元素遍历一遍，因此，此时的时间复杂度为 O(n)，其中，n 表示数组序列的长度。插入的时候只需要把元素插入到数组的最后一个元素的后面即可，因此，时间复杂度为 O(1)，删除操作也需要移动这个元素后面的所有的元素，因此，此时的时间复杂度也为 O(n)。

6）普通二叉树：在最坏的情况下，有 n 个结点的树的高度为 n，因此，查找、插入与删除的时间复杂度都为 O(n)。

从上面的分析可以发现，平衡二叉树的查找和删除的时间复杂度都是 O(logn)，Hash 表的查找、插入的时间复杂度都是 O(1)。因此，这两个数据结构有较好的查找和删除的性能。所以，选项 C、选项 D 正确。

【真题 558】 线性表如果要频繁地执行插入和删除操作，该线性表采取的存储结构应该是（　　）。

A．散列　　B．顺序　　C．链式　　D．索引

答案：C。

线性表的顺序存储是指用一组地址连续的存储单元依次存储线性表的数据元素。
链式存储结构又叫链接存储结构，在计算机中用一组任意的存储单元存储线性表的数据元素（这组存储单元可以是连续的，也可以是不连续的）。它不要求逻辑上相邻的元素在物理位置上也相邻。因此，它没有顺序存储结构所具有的弱点，但也同时失去了顺序表可随机存取的优点。
链式存储结构有以下 5 个特点：
1）比顺序存储结构的存储密度小（每个结点都由数据域和指针域组成，所以，相同空间内假设全存满，则链式存储比顺序存储所能存储的数据少）。
2）逻辑上相邻的结点物理上不必相邻。
3）插入、删除灵活（不必移动结点，只要改变结点中的指针）。
4）查找结点时链式存储要比顺序存储慢。
5）每个结点由数据域和指针域组成。
链式结构的插入和删除操作只需要修改插入和删除结点以及其前驱结点的指针域即可，而顺序存储结构在插入和删除操作的时候需要执行大量数据的移动操作。由此可以看出，顺序表适合随机访问，不适合插入和删除操作，而链式表适合插入和删除操作，不适合随机访问操作。散列表适合查找运算，索引表在插入和删除的时候还需要修改索引表，由此链式表最适合插入和删除操作。所以，选项 C 正确。
【真题 559】 寻找一条从左上角（arr[0][0]）到右下角（arr[m-1][n-1]）的路线，使得沿途经过的数组中的整数和最小。
答案：对于这道题，可以从右下角开始倒着来分析：最后一步到达 arr[m-1][n-1]只有两条路，即通过 arr[m-2][n-1]到达或通过 arr[m-1][n-2]到达，假设从 arr[0][0]到 arr[m-2][n-1]沿途数组最小值为 f(m-2,n-1)，到 arr[m-1][n-2] 沿途数组最小值为 f(m-1,n-2)。因此，最后一步选择的路线为 min{ f(m-2,n-1), f(m-1,n-2)}。同理，选择到 arr[m-2][n-1]或 arr[m-1][n-2]的路径可以采用同样的方式来确定。
由此可以推广到一般的情况。假设到 arr[i-1][j]与 arr[i][j-1]的最短路径的和为 f(i-1,j)和 f(i,j-1)，那么到达 arr[i][j]的路径上所有数字和的最小值为 f(i,j)=min{ f(i-1,j), f(i,j-1)} + arr[i][j]。
方法 1：递归法
根据这个递归公式可知，可以采用递归的方法来实现，递归的结束条件为遍历到 arr[0][0]。在求解的过程中，还需要考虑另外一种特殊情况：遍历到 arr[i][j]（当 i=0 或 j=0）的时候，只能沿着一条固定的路径倒着往回走直到 arr[0][0]。
但是递归算法效率太低，主要因为里面有大量的重复计算过程，比如在计算 (i-1,j)与 f(j-1,i)的过程中都会去计算 f(i-1,j-1)。如果把第一次计算得到的 f(i-1,j-1)缓存起来就不需要额外的计算，而这也是典型的动态规划的思路，下面重点介绍动态规划方法。
方法 2：动态规划法
动态规划其实也是一种空间换时间的算法，通过缓存计算的中间值，从而减少重复计算的次数，从而提高算法的效率。方法 1 从 arr[m-1][n-1]开始逆向通过递归来求解，采用动态规划要求正向求解，以便利用前面计算出来的结果。
对于本题而言，显然，f(i,0)=arr[0][0]+…+arr[i][0]，f[0,j]=arr[0][0]+…+arr[0][j]。根据递推公式：f(i,j)=min{ f(i-1,j), f(i,j-1)} + arr[i][j]。从 i=1、j=1 开始顺序遍历二维数组，可以在遍历的过程中求出所有的 f(i,j)的值，同时，把求出的值保存到另外一个二维数组中以供后续使用。当然，在遍历的过程中可以确定这个最小值对应的路线，在这个算法中，除了求出最小值外顺便还打印出了最小值的路线，实现代码如下：

```
public class Test
{
```

```
    public static int getMinPath(int[][] arr) {
        if(arr == null || arr.length==0)
            return 0;
        int row = arr.length;
        int col = arr[0].length;
        //用来保存计算的中间值
        int[][] cache = new int[row][col];
        cache[0][0] = arr[0][0];
        for(int i=1; i<col; i++){
            cache[0][i] = cache[0][i-1] + arr[0][i];
        }
        for(int j=1; j<row; j++){
            cache[j][0] = cache[j-1][0] + arr[j][0];
        }
        //在遍历二维数组的过程中不断把计算结果保存到 cache 中
        for(int i=1; i<row; i++){
            for(int j=1; j<col; j++) {
            //可以确定选择的路线为 arr[i][j-1]
                if(cache[i-1][j] > cache[i][j-1]) {
                    cache[i][j] = cache[i][j-1] + arr[i][j];
                    System.out.print("["+i+","+(j-1)+"]    ");
                }
                //可以确定选择的路线为 arr[i-1][j]
                else {
                    cache[i][j] = cache[i-1][j] + arr[i][j];
                    System.out.print("["+(i-1)+","+j+"]    ");
                }
            }
        }
        System.out.println("["+(row-1)+","+(col-1)+"]");
        return cache[row-1][col-1];
    }

    public static void main(String[] args){
        int[][] arr ={{ 1, 4, 3},{ 8, 7, 5},{ 2, 1, 5} };
        System.out.print("路径：");
        System.out.println("最小值为："+getMinPath(arr));
    }
}
```

程序的运行结果为：

```
路径：[0,1]  [0,2]  [2,0]  [2,1]  [2,2]
最小值为：17
```

这个方法对二维数组进行了一次遍历，因此，其时间复杂度为 O(m*n)。此外由于这个算法同样申请了一个二维数组来保存中间结果，因此，其空间复杂度也为 O(m*n)。

【真题 560】实现对一组无序的字母进行从小到大排序（区分大小写），当两个字母相同时，小写字母放在大写字母前。要求时间复杂度为 O(n)。

答案：如果没有时间复杂度的要求，本题可以采用传统的插入排序或快速排序等方法进行排序，但是传统的排序方法在最好的情况下的时间复杂度都为 O(nlogn)，显然，不满足题目要求的 O(n)时间复杂度。对于对时间复杂度有很高要求的问题，一般可以考虑用空间换时间的方法。鉴于此，对于本题而言，可以采用如下思路：

通常，字母为 26 个，当区分大小写后，变为 26*2=52 个，所以，首先申请一个长度为 52 的 int 型数组，按照 aAbBcC…zZ（小写字母保存在下标为偶数的位置，大写字母保存在下标为奇数的位置）的顺序依次记录各个字母出现的次数，当记录完成以后，就可以遍历这个数组按照各个字母出现的次数来重组排序后的数组，实现代码如下：

```
public class Test {
    public static void main(String[] args)  {
        char[] src = { 'R', 'B', 'B', 'b', 'W', 'W', 'B', 'R', 'B', 'w' };
        sort(src);
        for (int i=0;i<src.length;i++)  {  System.out.print(src[i] + " ");  }
    }
    private static void sort(char[] src)  {
        if(src == null)  {
            System.out.println("参数不合法");
            return;
        }
        //用于保存 52 个字符出现的次数，小写字母保存在下标为偶数的位置，大写字母保存在奇//数的位置
        //采用这种保存方法，可以保证在这个数组中小写字母出现在大写字母前面，小的字符出现//在大的字符前面
        int[] charCount = new int[54];
        for (int i = 0; i<src.length; i++)  {
            //对小写字母出现的次数就行计数
            if (src[i] >'a'&&src[i] <'z') {  charCount[(src[i] - 'a') * 2]++;}
            //对大写字母出现的次数就行计数
            else if (src[i] <'Z'&&src[i] >'A')  { charCount[(src[i] - 'A') * 2 + 1]++;}
        }
        //根据各个字符出现的次数按顺序生成排序后的字符数组
        int index = 0;
        for (int i = 0; i<charCount.length; i++)  {
            //这个字符在原始字符数组中存在
            if (charCount[i] != 0){
                //小写字母
                if (i % 2 == 0)  {
                    for (int j = 0; j <charCount[i]; j++){
                        src[index++] = (char) (i / 2 + 'a');
                    }
                }
                //大写字母
                else{
                    for (int j = 0; j <charCount[i]; j++) {
                        src[index++] = (char) ((i - 1) / 2 + 'A');
                    }
                }
            }
        }
    }
}
```

程序的运行结果为：

```
b B B B B R R w W W
```

6.2 链表

【真题 566】 数组和链表有什么区别？

答案：数组是按照数据顺序存储的，其大小固定。而链表中的数据可以随机存储，大小可动态改变。

【真题 567】 链表要求元素的存储地址（　　）。

A．必须连续　　B．部分连续　　C．必须不连续　　D．连续与否均可

答案：D。

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。链表由一系列结点（链表中每一个元素称为结点）组成，结点可以在运行时动态生成。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域。由此可见，可以通过结点的指针域找到下一个结点，存储地址是否连续并不重要。所以，选项 A、选项 B 和选项 C 错误，选项 D 正确。

需要注意的是，数组与链表不同，对数组的访问是通过数组的下标来实现的，所以，对于数组而言，存储地址必须是连续的。

所以，本题的答案为 D。

【真题 568】 以链接方式存储的线性表（X1、X2、…、Xn），访问第 i 个元素的时间复杂度为（　　）。

A．O(1)　　B．O (n)　　C．O (logn)　　D．O (n^2)

答案：B。

单链表查找的时候从头结点开始一直找下一个结点，如果要查找的元素在最后，就相当于找了 n 次，所以，时间复杂度为 O(n)。所以，选项 B 正确。

6.3 字符串

【真题 572】 编写一个截取字符串的函数，输入为一个字符串和字节数，输出为按字节截取的字符串。但是要保证汉字不被截半个，例如“人 ABC”4，应该截为“人 AB”，输入“人 ABC 们 DEF”，6，应该输出为“人 ABC”而不是“人 ABC+们的半个”。

答案：在 Java 语言中，默认使用的 Unicode 编码方式，即每个字符占用两个字节，因此，可以用来存储中文。虽然 String 是由 char 所组成的，但是它采用了一种更加灵活的方式来存储，即英文占用一个字符，中文占用两个字符。采用这种存储方式的一个重要作用就是可以减少所需的存储空间，提高存储效率。根据这个特点，可以采用如下代码来完成题目的要求：

```
public class Test
{    //判断字符 c 是否是中文字符，如果是返回 true
     public static boolean isChinese(char c)
     {
          String sb = String.valueOf(c);
          return sb.getBytes().length > 1 ? true : false;
     }
     public String truncateStr(String str, int len)
     {
          if (str == null || str.equals("") || len == 0)
               return "";
          char[] chrArr = str.toCharArray();
```

```
                StringBuilder sb = new StringBuilder("");
                int count = 0; //用来记录当前截取字符的长度
                for (char cc : chrArr)
                {
                    if (count < len)
                    {
                        if (isChinese(cc))
                        {
                            //如果要求截取子串的长度只差一个字符，但是接下来的字符是中文，
                            //则截取结果子串中不保存这个中文字符
                            if (count + 1 == len)
                                return sb.toString();
                            count = count + 2;
                            sb = sb.append(cc);
                        }
                        else
                        {
                            count = count + 1;
                            sb = sb.append(cc);
                        }
                    }
                    else
                    {
                        break;
                    }
                }
                return sb.toString();
            }
            public static void main(String[] args)
            {
                Test splitStr = new Test();
                String sb = "人 ABC 们 DEF";
                System.out.println(splitStr.truncateStr(sb, 6));
            }
        }
```

程序的运行结果为：

```
人 ABC
```

【真题 573】 如何不使用 Java 类库中的方法实现对字符串的反转？

答案：如果可以使用类库中的方法，可以直接调用 StringBuffer 的 reverse 方法对字符串进行反转。由于题目明确要求不可以使用 Java 类库中的方法，此时就需要采取其他方法了。

本题的解法通常有如下几种：

方法一：数组反转法

在 C/C++语言中，字符串其实就是字符数组，可以对数组中的元素反转即可，但是在 Java 语言中，String 是一个对象，而不是字符数组，应该如何转换呢？当然，可以先把字符串转换为字符数组，然后对数组进行反转，反转后再转换为字符串。有兴趣的读者可以自己实现代码。

方法二：逆序遍历法

当把字符串转换为字符数组后，可以逆向遍历数组，把遍历到的字符串拼接起来就得到了字符串的逆序序列，实现代码如下：

```
public String reverse2(String str) {
    StringBuilder sb = new StringBuilder();
    char[] ch = str.toCharArray();
    for (int i = ch.length - 1; i >= 0; i--) {
        sb.append(ch[i]);
    }
    return sb.toString();
}
```

方法三：递归法

对于这种类型的题目，面试官更希望看到求职者使用递归的方法来解决，因为递归法能够考查到面试者的知识功底，所以，有一定的难度。

递归法的主要思路如下：递归地把字符串中第一个字符移动到字符串的最后一个位置。实现代码如下：

```
public String reverseRecursively(String str) {
    if (str.length() <= 1) {
        return str;
    }
    return reverseRecursively(str.substring(1)) + str.charAt(0);
}
```

以上这个方法虽然代码看起来简单，但是实现起来却不容易，因此，非常考验求职者的能力。当然，这个方法也有很大的缺点，由于在执行的过程中创建出大量的字符串对象，因此，效率比较低。