精选200个Java技术面试真题，详解应聘Java程序员的常见考点
揭秘知名IT企业用人的核心准则，解读应聘IT企业的成功要诀

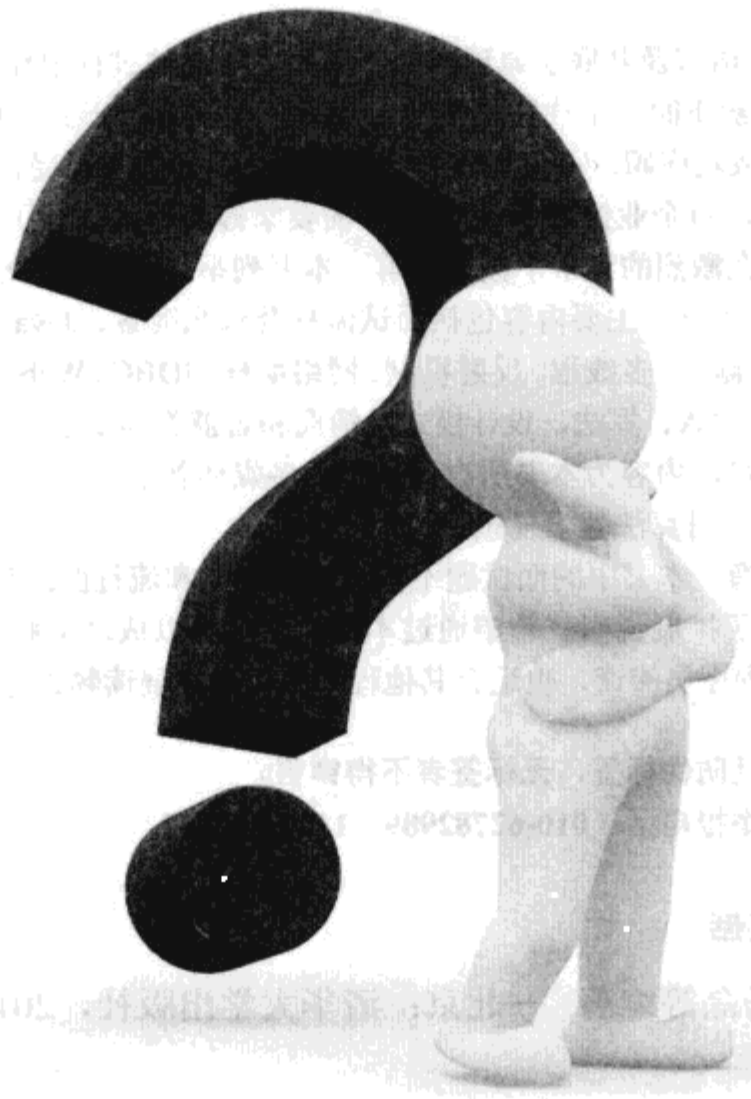
Java程序员面试宝典

另赠超值学习视频

14.5小时多媒体教学视频

杨磊 等编著

清华大学出版社



Java程序员面试宝典

杨磊　等编著

清华大学出版社
北　京

内 容 简 介

目前许多开发者对 Java 语言及其底层原理掌握不牢固，在面试过程中经常漏洞百出，无法取得好成绩。而招聘单位为了得到高素质的员工往往使出浑身解数，采用各种形式的面试考察求职者，这让面试难度大大增加。求职者要想成功应聘，不仅需要扎实的基本功，还需要经受情商和智商方面的考查。

本书通过 200 个面试题，对企业招聘 Java 程序员需要掌握的知识进行了系统、全面的总结，以帮助读者进行充分的面试准备，在激烈的竞争中拔得头筹。本书列举了各大 IT 公司的面试真题，详细分析了应聘 Java 程序员职位的常见考点，主要内容包括面试流程及求职准备、Java 语言基础、数据类型、集合框架、图形用户界面、输入与输出、多线程、反射机制、网络编程、JDBC、Web 开发基础、SSH 框架（Struts、Spring 和 Hibernate）、EJB、JPA、算法、设计模式及情商和智商类面试题。

本书附带 1 张 DVD 光盘，内容为本书所有面试题的多媒体教学视频（共 14.5 小时）及免费赠送的 55 小时 Java 教学视频和 5.5 小时算法教学视频。

授人以鱼，不如授人以渔。本书中的面试题不但以实例代码和流程图的形式对答案进行了详细解析，还对问题的相关知识点进行了扩展说明。希望通过本书，读者可以成功应聘，并提升综合素质。本书适合应聘 Java 和 J2EE 职位的程序员阅读，也适合其他程序员作为拓展读物进行阅读。

图书在版编目（CIP）数据

Java 程序员面试宝典 / 杨磊等编著. —北京：清华大学出版社，2010.9
ISBN 978-7-302-22832-5
Ⅰ. ①J…　Ⅱ. ①杨…　Ⅲ ①Java 语言－程序设计　Ⅳ. TP312

中国版本图书馆 CIP 数据核字（2010）第 097093 号

责任编辑：夏兆彦
责任校对：徐俊伟
责任印制：何　芊
出版发行：清华大学出版社　　**地　　址**：北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　　编**：100084
社　总　机：010-62770175　　**邮　购**：010-62786544
投稿与读者服务：010-62795954，jsjjc@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：清华大学印刷厂
装 订 者：三河市溧源装订厂
经　　销：全国新华书店
开　　本：185×260　**印　张**：23.25　**字　数**：581 千字
（附光盘 1 张）
版　　次：2010 年 9 月第 1 版　　**印　　次**：2010 年 9 月第1 次印刷
印　　数：1～5000
定　　价：49.80 元

产品编号：035261-01

前　言

为什么要写这本书？

作为一名 IT 技术人员，他的成长离不开周围的环境。好的公司、好的项目和好的同事都是影响一个程序员成长的重要因素。如何确定好自己的职业生涯是非常重要的。本书的主旨就是为程序员在成长的道路上提供一些参考，让他们能够正确地面对自己的求职过程，在面试的过程中能展现出自己最好的一面，提高面试成功的几率。

近几年软件开发类职位竞争异常激烈，这也带动了这个行业的蓬勃发展，软件人才越来越多，很多求职者都不是一次或两次面试就可以成功地找到自己的职业发展归宿。求职者应该把每次面试进行一个总结，找到不足之处。

本书通过 200 个有针对性的面试题，从职业规划、Java 基础知识、Java 开发高级技术、面试技巧、情商和智商等多方面系统地总结了 IT 企业招聘 Java 程序员的常见考点。本书内容大多取材于各大 IT 公司的面试真题，可以帮助读者准确把握 IT 企业面试的特点和脉络，在激烈的竞争中找到自己心仪的工作。

本书有何特色？

- 作者为书中的所有面试题都录制了多媒体教学视频，便于读者高效、直观地学习。
- 内容针对性强，对 Java 程序员求职面试中经常要碰到的技术题目和情商、智商类题目进行了精辟分析与解答，帮助读者全面了解 IT 技术面试的过程、特点和技巧，提高面试的成功率。
- 专门介绍了程序员的职业规划、合格的求职信与简历的写作技巧及程序员面试的基本过程，以便让读者在面试前做到心中有数。
- 例题丰富、典型。全书提供了 174 个 Java 程序员面试所必须掌握的技术类题目。这些题目涵盖了 Java 开发中的各类重点技术与难点技术，需要读者很好地掌握。
- 提供了 26 个情商和智商类面试题。这些题目很多都来源于跨国公司的面试题，大多是开放型问题，其答案并不唯一。本书重点剖析该类问题的解答思路，面试人员需要理解题目的考察意图，才能真正给出漂亮的解答。
- 在解答每一个题目的过程中，不但以实例代码和流程图的形式对答案进行了详细解析，还对问题的相关知识点进行了扩展说明，有助于读者开阔思路，深入学习。

本书内容及知识体系

第 1 篇　求职准备（第 1 章）

本篇主要内容包括个人心态定位、准备面试材料、投递简历、面试过程、面试实用技

巧和常见问题等。本篇力求浅显易懂，让读者一开始就做好求职的准备，了解面试的整个过程，建立自信。

第 2 篇　Java 基础知识（第 2～6 章）

本篇包含 70 个面试题，主要内容包括 Java 程序基础、语法基础、面向对象程序设计、数据类型、数据集合、图形用户界面开发等。本篇内容是 Java 语言及 Java 体系结构的基础，讲解时使用了大量实例代码和图表便于读者理解。

第 3 篇　Java 高级特性（第 7～11 章）

本篇包含 46 个面试题，主要内容包括 Java 的 I/O 体系、多线程编程、反射机制、网络编程、数据库访问等。本篇中的内容都是 Java 编程中经常会使用到的高级技术，这些技术往往也是 Java 面试的重点。

第 4 篇　Java EE 相关问题（第 12～14 章）

本篇包含 43 个面试题，主要内容包括 Web 开发基础、SSH 框架（Struts、Spring、Hibernate）、EJB 和 JPA 等。本篇中的内容是 Java Web 开发中所必须掌握的内容，也是 Java 程序员面试的另一个重点。

第 5 篇　算法和设计模式（第 15 章）

本篇包含 15 个面试题，主要内容为 Java 程序员面试过程中经常会碰到的算法和设计模式等知识。算法是程序的灵魂；设计模式是前人对一些比较经典的开发模式的总结，这些模式可以解决一些共性的问题。算法和设计模式往往也是技术面试所考察的重点内容。

第 6 篇　情商和智商经典面试题（第 16、17 章）

本篇包含 26 个面试题，主要内容为情商和智商类面试题。情商和智商测试不但可以考察求职者的综合应变能力和逻辑能力，还可以考察他们的知识广度、学习能力、判断力、解决问题的能力等。另外，还可以通过这些问题了解求职者的性格特征。所以，这类问题也是程序员面试中的热点问题。这些问题的答案并不一定是唯一的，本篇重点剖析这些问题的推理过程及解答思路，而实际面试时应该根据具体情况进行回答。

本书约定

【出现频率】：综合知名企业面试题目，对每一道收录的面试题给予星号打分，出现频率最高的以★★★★★表示，这样读者可有针对性地进行阅读。

【关键考点】：以关键字的形式抽取考核要点，方便读者定位重点。

【考题分析】：为了更清楚地分析面试题目，该部分以名词解释、示例代码、流程图、界面等多种形式进行解答。其中，示例由 Java、XML、JSP 等代码实现，在代码中对重要的变量或者复杂的语句均给出了注释和功能介绍。对于面试题目分析的重点部分，也以“注意”的形式给出了需要读者格外注意的地方。

【答案】：通常，技术类的面试题会给予正确的、标准的答案，而对于情商和智商类面试题中的开放型问题，考虑到答案可能并不唯一，所以本书仅仅剖析问题的推理过程及解答思路，引导读者正确思考该类问题。

配书光盘内容

- 本书中每一个面试题的多媒体教学视频；
- 免费赠送 55 小时 Java 入门教学视频和 5.5 小时算法教学视频。

适合阅读本书的读者

- 所有想了解 Java 程序员面试的人员；
- 计算机相关专业的应届毕业生；
- 应聘软件行业的相关就业人员；
- 技术部门的招聘主管；
- Java EE 职位的应聘者；
- Java 软件工程师职位的应聘者。

阅读本书的建议

- 无论是初、中级程序员，还是高级程序员，都建议认真阅读第 1、16、17 章；
- 没有 Java 语言基础的读者，建议从第 1 章顺次阅读并演练每一个实例；
- 有一定 Java 语言基础的读者，可以根据实际情况有重点地选择阅读；
- 对于每一个实例，先自己思考一下实现的思路，然后再阅读，学习效果更好。

本书作者及编委会成员

本书由杨磊主笔编写。其他参与编写和资料整理人员有陈世琼、陈欣、陈智敏、董加强、范礼、郭秋滟、郝红英、蒋春蕾、黎华、刘建准、刘霄、刘亚军、刘仲义、柳刚、罗永峰、马奎林、马味、欧阳昉、蒲军、齐凤莲、王海涛、魏来科、伍生全、谢平、徐学英、杨艳、岳富军、张健和张娜。在此一并表示感谢。

本书编委会成员有欧振旭、陈杰、陈冠军、项宇峰、张帆、陈刚、程彩红、毛红娟、聂庆亮、王志娟、武文娟、颜盟盟、姚志娟、尹继平、张昆、张薛。

编著者

目　录

第 1 篇　求职准备

第 2 篇　Java 基础知识

第 3 篇　Java 高级特性

第 4 篇　Java EE 相关问题

第 5 篇　算法和设计模式

第 6 篇　情商和智商经典面试题

轻松地获得面试官的青睐，从千百个竞聘者中脱颖而出

本书内容及对应的教学视频时间

- ◎ 应聘软件开发职位前必须知道的那些事
- ◎ Java程序基础（43分钟视频）
- ◎ Java语法基础（103分钟视频）
- ◎ 数据类型及类型转换（63分钟视频）
- ◎ 数组和集合的使用（54分钟视频）
- ◎ Java图形用户界面（42分钟视频）
- ◎ 输入/输出流（37分钟视频）
- ◎ 多线程编程（37分钟视频）
- ◎ Java反射机制（30分钟视频）
- ◎ Java网络编程（44分钟视频）
- ◎ Java对数据库的操作（43分钟视频）
- ◎ Web开发相关技术（60分钟视频）
- ◎ Struts、Spring和Hibernate组合（109分钟视频）
- ◎ EJB与JPA相关问题（51分钟视频）
- ◎ Java编程试题（70分钟视频）
- ◎ 情商类面试题（48分钟视频）
- ◎ 智商类面试题（43分钟视频）

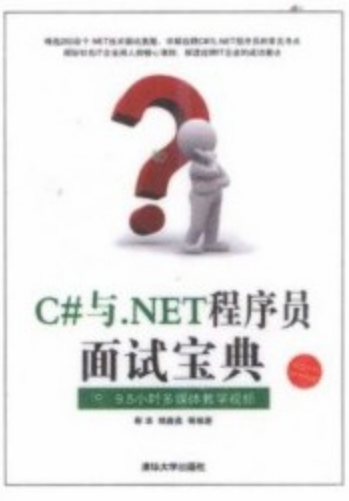
本书读者对象

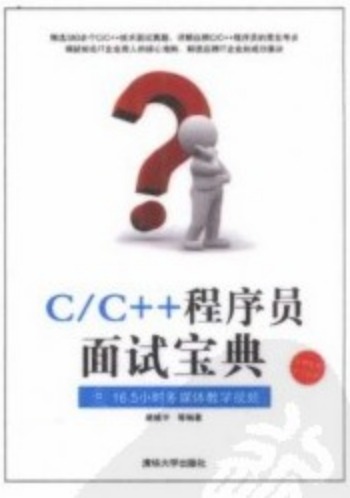
- ◎ 所有想了解Java面试的人员
- ◎ 计算机相关专业的应届毕业生
- ◎ 应聘软件行业的相关就业人员
- ◎ 技术部门的招聘主管
- ◎ Java EE职位竞聘者
- ◎ Java软件工程师职位竞聘者

超值、大容量DVD-ROM内容

- ◎ 14.5小时本书配套多媒体教学视频
- ◎ 55小时Java教学视频（免费赠送）
- ◎ 5.5小时算法教学视频（免费赠送）

程序员面试宝典





ISBN 978-7-302-22832-5
9 787302 228325 >
定价：49.80元（附视频教学DVD）

第 1 篇　求职准备

⏭ 第 1 章　应聘软件开发职位前必须知道的那些事

第 1 章　应聘软件开发职位前必须知道的那些事

人们常说：机会总是留给有准备的人。大家在求职之前，一定要明确自己的求职态度，熟悉求职的过程，做好充分的准备，把一些可预见的事情做好。这样在招聘的时候才能充满自信，处变不惊。求职过程中，有些事情是无法预料的，这些事情可能会打乱求职者的心态，影响求职的结果。如果应聘者能够尽早把准备工作做好，即便发生了一些意料之外的事情，也不会有太大的影响，把事态控制在自己所能控制的范围之内。

本章探讨作为一个计算机相关专业的应届毕业生或有志于从事软件开发的人员，应该如何摆正自己的心态，以及一些在求职过程中应该注意的问题。

1.1　认识职业规划

大部分程序员都是在职场中不断地遇到挫折、漫无目的地跳槽后，才对自己的职业发展方向产生了疑惑。合理地规划自己的职业生涯是非常有必要，也是非常有意义的事，对于任何人，完善的职业生涯规划都能让自己有一个目标和方向，然后向着这个目标前进。

1.1.1　职业规划的意义

职业规划的意义是每一个初涉职场的人必须要了解的，下面是最重要的几点。

（1）以既有的成就为基础，确立人生的方向，提供奋斗的策略，发掘自我潜能。

完善的职业生涯规划将使你正确认识自身的个性特质、现有与潜在的资源优势，帮助你重新对自己的价值进行定位并使其持续增值，并对自己的综合优势与劣势进行对比分析。还可以使你树立明确的职业发展目标与职业理想，客观评估个人目标与现实之间的差距，并可更敏锐地搜索或发现新的或有潜力的职业机会。并且，通过采用科学的方法付诸于实际行动，不断增强你的职业竞争力，即可实现自己的职业目标和理想。

（2）可以重新安排自己的职业生涯，突破生活的格线，塑造清新充实的自我。

（3）准确评价个人特点和强项，增强发展的目的性与计划性，提升成功的机会。

职业生涯的发展要有计划、有目的，不可盲目地“碰运气”。很多人在职场受挫就是由于生涯规划没有做好。好的计划是成功的开始，凡事“预则立，不预则废”就是这个道理。

（4）评估个人目标和现状的差距。

（5）职业生涯规划可以提升应对竞争的能力。

当今社会处在变革的时代，到处充满着激烈的竞争。物竞天择，适者生存。职业活动的竞争尤为突出，特别是我国加入 WTO 后，要在这场激烈的职场竞争中脱颖而出，并始终立于不败之地，必须设计好自己的职业生涯规划。做好个人的职业生涯发展规划后，在

职业发展的道路上，会有很多因素影响着个人的职业发展，其中最重要的应当是自身对职业的选择。如何降低在职业上最大的风险也是个人职业规划需要考虑的问题。如满足于现状，不思进取这样的问题是要靠更新知识结构、转换思维模式来解决的。针对不同的阶段，本章将分别讲述应届毕业生的近期职业规划，以及在职程序员（或相关职业）的职业生涯规划。

1.1.2　软件人才的职业规划

软件行业所带来的职位不仅限于程序员，编码工作也只是软件开发的一部分。一个人从事软件行业，可以只涉及其中的一个方面，也可以涵盖多个方面。如何知道哪个或哪些类型的职位更适合自己呢？下面是在面试中的一些常见的软件人才职位。

- 程序员；
- 软件工程师；
- 系统分析师；
- 数据库架构师；
- 测试工程师；
- 项目经理。

上面只是列出了一些常见的职位。公司越大，对职位的划分可能会越细。在招聘程序员时，一般会写明是.NET 工程师，还是 Java 工程师。针对不同的语言，也会有不同的职位要求。

计算机软件专业的学生中很多人有着十分短视的想法，那就是毕业后工作就是写程序，只要把程序写好就可以了，至于以后怎么样从来没有想过。如果抱着这样的态度应聘工作，那么很难找到适合自己的发展平台，或许要经历几年的徘徊才知道自己真正适合的是什么。

计算机软件技术覆盖的范围非常广，在我们国内的大学教育中，通常是以学习编程语言为主，往往一个学期要同时学习几门语言，然后才是数据结构、微机原理等课程。这也是造成很多人工作后常常为了各种语言的优劣而争论的原因。其实不同的语言、不同的平台技术的应用范围不同，各有其优点。

在编程者的职业规划中，不应以编程语言或技术平台为主轴进行设计。例如，在职业规划中先做 VB 方面的工作，再做 ASP 方面的工作，然后做.NET 方面的工作。这样是非常不科学的。编程者虽然喜爱编程，但是每个人的特点不同、长处不同，应该根据每个人自身的特点规划不同的编程应用方向。例如很多人就不喜欢汇编语言，以为晦涩难懂，但也有少数人喜欢这种语言，因为直接和计算机底层打交道，执行效率高，有助于理解计算机的工作方式。

对于喜欢轻松开发程序的人，VB 可能是比较适合的职业发展起点。可能有人会认为 VB 有着这样那样的不足之处，但就是 VB 这样的特点使其拥有数量惊人的使用群体，大量的优秀程序由 VB 开发。所以只要 VB 能从熟练到精通，也可以发展为技术高手，创造很多成就。.NET 和 Java 技术有着很多相似之处，在职业生涯的规划中有必要同时将两者融会贯通。C++是很多初学者不理解的语言。因为在很多大学中一味求新，直接学习了 C#，而对 C++的学习不注重，导致有的学生认为 C#是 C++的升级版本，在工作中掌握 C#即可，C++只是过时的语言。

在国内，新技术确实容易引起很多误会，甚至有的企业招聘时也盲目跟风，无论什么项目都只用 Java 或.NET 开发，有的应届毕业生以为自己会 Java 或.NET 即可走遍天下了。殊不知 C++（本地代码程序）程序远比 Java 或.NET 程序效率高，并且不需要安装运行环境，在某些有要求的项目中 C++才是最好的选择。只有认清这些技术的各种细节，在职业生涯规划中才不会以编程语言或技术平台为主线进行设计，而应当依据个人所适合的技术进行深入，直至成为专家。

1.2 应 聘 过 程

应聘过程是十分曲折的，任何一个环节出现问题都会导致应聘的失败。求职者应该对每一个求职环节都掌握于心，做好充分的准备，才能有的放矢，灵活应对应聘过程中出现的每一个问题。

1.2.1 掌握基本的应聘流程

既然是求职应聘，肯定得先找到一个公司，找到一个适合自己的职位了。现在应聘的渠道也比较多，对于 IT 技术职位的应聘来说，比较常见的有校园招聘、招聘网站、社会招聘会、专题招聘会等形式。

对于应届毕业生来说，校园招聘会更适合他们。因为他们一般没有什么工作经验，校园招聘也不会要求他们有工作经验，而社会招聘往往会设置一个工作经验的门槛。一些公司在每年年底的时候，会到校园去进行宣讲会，应届毕业生在这个时候去求职是再适合不过的了。

目前，招聘网站也是一个比较流行和重要的招聘信息来源。一些大型的软件公司也会在招聘网上，例如智联招聘、中华英才网等，发布他们的职位需求信息。此时，求职者投递的就是电子简历了，它相对于纸张的简历来说，亲切感要差一些。

另外，每年年初是跳槽的高峰时期，各大城市都会组织大型的招聘会，或者专题招聘会，应届毕业生和社会求职者都可以到这些招聘会上找适合自己的公司及其职位，通过面对面的交流，充分了解公司的情况和职位的情况。

通过各种渠道找到自己心仪的公司和职位以后，就可以开始去应聘了。应聘的流程大致包括：编写简历、投递简历、笔试、面试、确定薪酬和拿到 Offer，如图 1.1 所示。

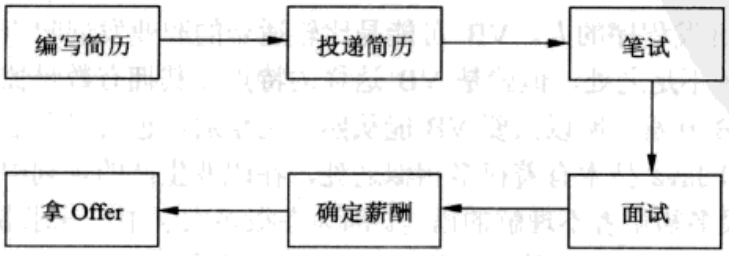


图 1.1　应聘流程

1.2.2　面试资料的准备

无论是应届毕业生还是有过工作经验的应聘者，在面试前都要准备充足的各种资料。这些资料一般包括以下几种。

1．作品

对于技术设计类的工作岗位，具体的作品是最重要的资料，也是用人企业最看重的东西。应届毕业生不要以为没有工作经验而不准备具体的作品。例如，应聘者所应聘的工作岗位是“Web 开发人员”，应试者要事先了解该岗位对 Web 开发的技术要求，然后根据要求准备自己的作品资料。即便是应届毕业生也可以准备自己的作品资料，如网站后台方面的课程设计，如果有必要还可将其修改为完整的项目。

通过应试者对自己作品的讲解（如源代码），用人企业可以直接了解应试者的技术能力，避免了各种测试所走的弯路。很多小规模的公司往往跳过了笔试步骤，直接通过应试者的作品来判断其能力是否适合工作岗位。

2．证书

由于个人简历中一般都会注明自己所获得的各种证书，很多用人企业筛选简历的依据也有证书的因素。所以应试者最好带上证书的原件，如学历证、学位证、英语等级证、程序员等级证等。由于应届毕业生没有工作经验，所以证书的作用相对比较大，而有过工作经验的反而不需要太多的证书来证明自己。

3．个人简历

虽然用人企业已经浏览过应试者的简历，但在实际中，用人企业仍然有可能会让应试者重新填写一份简历。对于很多应试者，重新填写简历时部分内容可能会与原来的有出入，所以应试者最好带上一份原始简历。

1.2.3　简历的写法及应注意的问题

简历是求职者信息的一个概况，是招聘者对求职者的情况了解的第一来源。它的重要性是不言而喻的。而事实上，大多数的简历都是不合格的。许多管理者都会抱怨简历很糟糕。那么应该如何让自己的简历做到简洁明了，在充分展示自己的同时又不显得冗余呢？其实，求职者在写简历时应该注意以下几点。

1．实事求是

招聘单位看简历，肯定是要看一个真实的求职者。求职者应该完全如实地把信息表达出来，不可以包含任何的虚假信息。这些虚假信息对求职者是没有任何帮助的。即使它们为求职赢得了一次面试机会，也会在面试时露出马脚，使得招聘者更加反感。为人真诚是一个基本的做人原则，它反映了一个人的基本素质。

2. 简历不可过长

简历过长会掩盖一些有价值的闪光点，冗长是简历最易犯的毛病。招聘单位在筛选简历的时候，往往会收到许多简历，工作人员不可能把每个人的简历都仔细读一遍。如果简历过长，工作人员可能看到的大多是没有价值的东西，而把真正有价值的东西给忽略了。所以，简历要尽可能的简短，一般来说，两页是一个比较适合的长度。

把自己的学习经历全部写上去，是一个比较普遍的错误。招聘单位一般不会关心你是哪个小学、哪个中学毕业的，教育经历从大学开始写就可以了。

另外，一些应届毕业生常常把成绩单、在校期间的获奖证书等复印件附在简历后面，这也是不必要的。只需要在简历中进行注明，如果招聘单位对这些信息感兴趣，以后提供也不迟。

3. 简历不必写自己的缺点

每个人都会有缺点，但是不必在简历中写出来。不真实的东西不可以写，缺点也可以不写。这并不是否认每个人固有的缺点，而是因为招聘单位更多的是想看到求职者的优点及这些优点可以给公司带来哪些利益。

4. 排版规整，无须过分花哨

简历在一定程度上也能反映出这个人的形象。一般说来，简历不应该过分的花哨，用平平整整的 A4 白纸就好。还有一点，错别字是千万不能出现的，用人单位肯定是无法容忍简历中还有错别字。另外，排版凌乱也会给简历扣分的。

5. 简历应该根据职位的不同而有所侧重

千万不可只准备一份简历。每一家公司的要求是不同的，职位的要求也是不一样的，除了一些必须包含的信息以外，应该针对不同的职位准备不同的简历，去掉每个职位不需要的信息，而突出需要的信息。

6. 不要注明对薪资的要求

薪资是一项敏感话题，最好不要在简历中注明自己的薪资要求，否则将会使自己冒很大的风险。如果要求过高，用人单位可能会觉得雇不起你，面试机会都没有了；如果要求过低，用人单位可能会觉得你没有什么价值。对于刚走出校门的应届毕业生来说，工作经验对他们来说更重要，就更没有必要写薪资要求了。

一份合格的求职简历应该包括以下内容。

- ❑ 姓名、联系方式等个人资料：放在简历的顶部，主要是为了方便用人单位与求职者联系。
- ❑ 专业技能：对于软件工程师的应聘，用人单位往往有特定的技术要求，把自己拥有的专业技能成列出来可以让人力资源管理者（HR）更容易关注。
- ❑ 教育背景：毕业院校、专业、培训经历等可以看出求职者的一个大概的情况。
- ❑ 工作经历和项目经验：工作经历和项目经验是一份简历中最重要的部分，用人单

位完全可以通过求职者的工作经历来判断他的一些技能、求职的心态等情况。所以，工作经历和项目经验需要写得比较详细，主要表达一下求职者曾经在哪里工作，做了什么，取得了什么成绩。

- 英文简历：如果应聘的是外企或对外语有较高要求的企业时，还需要附上一份英文简历，它在一定程度上也能反映出求职者的英文水平。

以下是一份应聘软件开发工程师的简历模板。

求 职 简 历
个人基本信息：
姓名：万世
籍贯：四川
性别：男
年龄：26
联系电话：13801301388
学历：本科
电子邮件：wanshi@163.com
工作年限：3 年
英语水平：CET-6

专业技能：
精通 J2EE 规范下的 JSP、Servlet、JDBC、JavaBean 编程；
熟练掌握 Struts+Hibernate+Spring 结构下，应用 MVC 模式的 Web 应用开发；
熟练使用 SQL 语句，有 Oracle、MySQL 等关系型数据库的开发经验；
熟练使用 IDE(Eclipse)进行 Java 软件开发；
熟练使用 Java 应用服务器(JBOSS)和 Web 服务器(Tomcat)进行 Java 程序开发；
熟悉 Unix/ Linux 平台下的 C++、Java 开发；
熟悉 HTML 标签、Javascript、DOM、CSS；
了解 EJB 3.0；
了解 Unix/Linux 环境下 C/C++应用软件开发和 Shell 编程。

教育背景：
2002/09~2006/07，电子科技大学通信工程专业，获得学士学位；
在校期间，成绩优异而突出，多次获得校级奖学金，并被评为优秀大学生。

主要课程包括：
C 语言、数据结构、网络技术基础、操作系统、数据库系统概论、C++面向对象程序设计、Java 程序设计。

在校期间的主要奖项：优秀大学生奖。

培训经历：
2006/09~2007/02　　软件工程师培训　　加拿大达内外企 IT 培训
主要课程：Unix、C/C++编程、Core Java、Java EE 的各项技术。认真完成多个项目实战（包括 Unix C 和 Java EE 项目），提高了自己的技术能力和团队协作素质。

工作经历：
2007/03~至今　　Java 软件开发工程师　　四川电信集团成都电信分公司
从事 Unix 环境下的 Java 软件开发工作，现任技术小组长。工作期间，认真完成项目中的每一项任务，连续两个季度被评为公司优秀员工，最近一次的年度评估为优秀。

项目经历：
2009/3~2009/8

由 XXXX 出版社约版编写《Java Web 从入门到精通》，为第一作者。本书由浅入深地讲解了 Java Web 开发的知识，引导 Java Web 程序员循序渐进地学习 Java Web 所涉及的各个方面知识，并且提供了丰富而生动的案例。主要内容包括：Java Web 开发环境的搭建、Servlet 与 Web 容器、JSP、JavaBean、表达式语言(EL)、标准标签库(JSTL)、在 Web 程序中使用 Log4j 来记录日志、Struts、Spring、Hibernate、Ajax 等。
2008/8~2009/2
号码百事通智能系统，该系统主要功能是通过某特定时段电话的拨打频率和内容进行智能分析，为高层决策者提供数据支持。该系统已交付使用。
运行环境：HP-UNIX、JVM1.5、JBOSS
数据库：Oracle
主要技术：Java EE(Struts、Spring、Hibernate)、AJAX
2008/1-2008/7
号码百事通注册商家管理系统，该系统为 114 的注册商家提供一个后台管理系统，包括信息查询、生成报表、投诉与建议处理等功能。
运行环境：Linux、JVM1.5、Tomcat
数据库：Oracle
主要技术：Java EE(Struts、Spring、Hibernate)、Velocity 模板
2007/4-2007/12
号码百事通客服管理系统，为 114 的客户人员提供信息查询的前端程序，包括号码信息查询、欠费查询、增值服务推荐、电话激活服务等功能。
运行环境：Linux、JVM1.5、Tomcat
数据库：Oracle
主要技术：JSP、EJB

个人评价：
谦虚、谨慎而富有挑战精神，希望您能给我一个展示自我的机会。

1.2.4　求职信的写法及应注意的问题

招聘公司第一眼看到的不是应聘者的简历，而是一封求职信，有些求职信包含在附件中，有些则直接显示在邮件中。所以求职信的内容非常关键，是招聘公司对应聘者的第一印象。求职信主要由 6 部分组成：称谓、正文、结尾、附件、署名、成文时间。

称谓写在第一行，要顶格写公司名称。正文是求职信的重点，内容主要包括以下 4 点：

（1）个人的简单介绍，尽可能一句话说明。例如，我叫万世，应聘 XXX 职位。

（2）求职原因，要直截了当地说明写此信的目的。例如，从 51job 上看到贵公司招聘高级程序员一职。

（3）个人资历及能力，求职的关键是对自己的业务能力要做出客观公允的评价，要特别突出自己的优势和闪光点。例如，在多个大型项目中担任架构师、软件开发工程师，擅长对项目整体进行可执行性评估，精通 C#编程，有较强的数据库系统规划与设计能力，同时对人机交互用户体验方面也有比较深入的研究。

（4）提出希望和要求，这属于收尾阶段的内容，要适可而止。例如，“希望贵公司能为我安排一个面试的机会”，或“盼望您的答复”、“敬候佳音”之类的语言。

结尾需另起一行，写表示敬祝的话。例如，“此致、敬礼”，或祝“工作顺利”等词语。署名和成文日期写在信的右下方，其中署名在上，成文日期在下。所署姓名的前面不要添加任何谦称的限定语，以免有阿谀之感。成文日期要年、月、日俱全，尤其不要缺少年份。

在求职信的写作过程中，需要注意以下几点：

（1）语句通畅、自然，用词不晦涩。

（2）杜绝错误：无论是错别字还是错误的英文单词，均要避免出现。

（3）言简意赅：在内容完整、重点突出的前提下，尽可能简明扼要，切不可长篇大论。

（4）具体明确：明确说明自己应聘的职位，多使用数字、实例等具体信息去表达自己。

下面列出了常见求职信的写法。

尊敬的公司领导：

　　您好！我叫万世，应聘贵公司招聘的高级程序员一职。

　　我拥有近 10 年的软件开发和需求分析工作经验，曾在外企软件公司从事 B/S 软件、商务型站点的系统分析及开发工作，熟悉煤炭、石油行业的质量安全类软件的需求。

　　我以面向对象、面向构件的思想构建框架定义，熟练掌握定义用户界面，编写详细的业务流程和业务模型文档的方法。熟练应用 Rose、Visio、PowerPoint 等软件；熟练掌握 ASP.NET、C# 编程语言，应用的数据库主要为 SQL Server，熟练掌握 XML。

　　随信附上我的简历，其中包含了我所有开发项目及工作职责。

　　如果我的工作经验及经历符合贵公司的要求，可随时参加面试。非常感谢！

　　祝工作顺利！

万世

2010.6.6

1.2.5　面试的准备

求职信投递出去后，如果接到面试电话，就要进行面试的准备了。面试者的心理反应将对面试的成败产生者定性作用。同时，面试考官的心理反应也将对面试过程产生一定的反作用。一些应聘者多方面条件都较好，但在面试时由于缺乏经验，心理素质不过关，不善于展开话题，甚至在面试中说话结结巴巴，让考官无从对其能力进行正确地评价，从而导致求职不顺利。在面试之前，应在以下几个方面作好充分的心理准备和调整。

1. 主动获取充足的信息，做到知己知彼

面试之前一定要站在企业的角度重新认识自己，才可能迅速找出自己面对所应聘岗位的不足之处。另外，应试者要主动获取有关用人企业的信息，了解企业的性质、资质、规模、业务信息及企业文化等。如果应试者对这些信息没有充分的了解，在面试中就无法准确恰当地回答面试者的问题，同时也无法确定该企业是否适合自己的发展。如果用人企业是规模较小的私营企业，应当将准备面试的重点放在和所应聘的工作岗位密切相关的方面。如果用人企业是规模较大的私营企业，则应当将准备面试的重点放在表现自己思维能力、学习能力、团队协作能力等方面。不同的企业有不同的企业文化，有些企业喜欢实干型人才，有些企业喜欢有潜力和进取心的人才，有些企业喜欢沟通能力强的人才。当应试者掌握了所应聘企业的用人风格后，即可投其所好，重新塑造出企业易接纳的形象。

2. 提前查找交通路线，以免迟到

虽然迟到看上去是不应发生的情况，但在实际的面试中，迟到现象屡见不鲜。所以当面试地点不是非常熟悉时，应查找资料，具体到在何处上下车、转换车。并且要留出充足

的时间去搭乘或转换车辆，包括一些意外情况都应考虑在内。

3．了解面试考官的心理

面试考官和应试者的心理是截然不同的。在面试时，面试考官因处于主动、支配地位，所以有心理上的优越感。这种优越感有利于主考官主动性、能动性的发挥，但把握不好也容易形成极端化倾向。同陌生人第一次见面，对方的仪表、言谈、举止等因素，可以给人留下第一印象。第一印象在面试考官心目中非常重要，很容易引起面试考官情绪上的喜欢或不喜欢。由于受这种感性情绪支配，面试考官在对应试者进一步认识的过程中，常常会不自觉地受第一印象的影响。这种影响有时虽然是不客观的、错误的，但要克服和消除却很难。所以应试者应以真诚的态度与主考官沟通信息，并且建立较好的第一印象，这样对接下来的面试过程有很大的帮助。在面试考官面前，应试者应建立什么样的第一印象呢？自卑怯懦、狂妄自大、自我封闭、计较多疑等印象都是应该杜绝的，应试者的形象应该是诚实、自信而不自负、实事求是地和面试考官交谈。

4．在面试过程中，应聘者处于一种接受提问，同时兼顾自我表现的状态

这种状态经常让应聘者出现两种极端倾向，或者因过于拘谨而表现不足，或者因表现过分而卖弄做作。显然，这两种倾向都会严重影响面试成绩。应聘者应当提前调整好自己的心理状态，对着镜子自己反复练习，以避免面试时的不恰当表现。

最后，应聘者应当注意自己的礼节。特别是应届毕业生，社会上的很多规则礼仪一定学会适应。当面试考官提出某些问题使你觉得被冒犯且与工作无关时，可以有礼貌地请问为何要提出此问题，并直接提出自己的建议。例如委婉地回答：“很抱歉，我不知道这个问题与我所应聘的工作岗位有何关系，待我进入贵公司工作后，再来讨论私人问题如何？”但千万不要说：“这是我的个人隐私。”或其他更加没有礼貌的回答。毕竟对方有可能成为你的顶头上司，倘若被录取，也恐怕日后处事有所不便。所以即使对方所提问题非常不礼貌，身为求职者，不能意气用事或表现出不礼貌的言词。你可以拒绝，但口气及态度一定要婉转温和。

注意：在面谈时，千万不要出现不礼貌的行为，因为一些小动作也会被主考官列为评判内容。

1.3　面试的方式

常见的面试有笔试、上机考试和电话面试 3 种，本节逐一介绍这 3 种面试的常见注意事项。

1.3.1　笔试

应聘软件开发职位，往往会遇到一轮笔试。尽管笔试的分数不能完全说明求职者的技术水平，但是却可以把离要求比较远的求职者淘汰出局，所以求职者不可以轻视笔试这一

轮。面对笔试，应聘者要尽量避免以下 5 个误区：

（1）因一道题不会，放弃了整个笔试。自己认为很难的题目，或许其他竞争对手也会被难住，因此没有必要因为一道题目不会就放弃整个笔试。

（2）不能完整作答的题目一字不答。如果不能回答完整，最好也写上解题思路、流程图，甚至伪代码。

（3）未能认真审题，过于匆忙答卷。通常笔试是不计时的，无论半个小时，还是两个小时完成作答对笔试成绩都没有影响，因此可以按照自己的正常速度回答问题即可。

（4）笔试没有 60 分及格线。笔试成绩无所谓及格与不及格，倘若应聘者某一个道题回答的很精彩，虽然总分并不理想，但也许会被企业录用。

（5）不同公司的笔试题目会有雷同。很多公司的笔试题目会有雷同，因此笔试结束后一定要进行总结，把未能回答出的问题搞明白，争取日后的考题都可以顺利回答。

下面列举微软公司的几个笔试题，请参考。

（1）给出一个函数来复制两个字符串 A 和 B，字符串 A 的后几个字节和字符串 B 的前几个字节重叠。

（2）如何截取键盘的响应，让所有的 a 变成 b?

（3）存储过程是什么？有什么用？什么优点？

（4）美国有多少个井盖？

除了前面介绍的误区外，面试者还要注意以下几个方面。

（1）在纸上写程序：除了笔试时在纸上写程序外，面试中也会常常被面试官要求在纸上写一段代码。最常见的是要应试者写数据结构方面的程序。大家知道编程平时都是直接在计算机上写程序，如果突然要你在纸上写程序，尤其是在面试官的眼皮子底下，你可能会比较慌乱，思路也不会太清晰。建议平时在纸上多多练习。

（2）英语的重要性：外企公司的笔试卷子基本上都是英文的，无论是出题还是答题，都要应试者回答，因此必须有很好的英文阅读能力。其实这也不需要一定要过英语六级，但却需要你有相当的单词量，然后按自己的想法进行语言的组织就行了。对于这方面，建议大家在阅读计算机文档（例如 MSDN）时，尽量选择英文版的进行阅读。

国企一般对外语不是很看重，题目都是中文的。如果你不打算去外企，也不用特别准备英语。

（3）相关智力测试：现在越来越多的笔试中加入了智力测试，很多测试题本身也相当有趣。这里建议大家不要把精力全部放在智力测试方面，而是应该以技术方面的笔试为主。可以把智力测试作为一个简单的调味品。

（4）要善于总结：如果笔试或者面试时有问题没有回答好，回来后一定要进行总结，把不会的问题弄明白，不清晰的知识点理解透彻。试问当你碰到两家公司都问了同样的问题，而你两次都不会，这将是非常郁闷的。

1.3.2　电话面试

在中国有很多专门做外包的外企，这些企业要求一些基本功底扎实，精通一种编程语言的开发人员。但实际上外包的业务并不像做项目，有些功能重复性很大，这种企业一般都会采取电话面试。还有一些正规公司是人力资源部经理先进行电话面试，了解应聘者的

薪水要求和职业规划情况。企业电话面试的程序通常分为以下 7 个步骤。

（1）确认已审阅简历并明确面试问题。

（2）确认对方与自我介绍。

（3）介绍公司与职位基本情况。

（4）确认本次及最近两次或重要岗位特别是最近工作转换的原因。

（5）关键职能与素质考察（主要涉及关键岗位职责与成就，了解求职者在团队中的相对位置和作用及个人强项与需提升的方面等）。

（6）了解求职者目前的薪酬水平与构成、期望与底线等。

（7）空出时间让应聘者提问，以表示用工双方的平等。

由于模式限制，电话面试时间通常不会很长。在面试时注意，一定要表现自信、礼貌、认真、严肃。在回答电话面试问题的时候，一定要放松。如果是外企，很有可能要求你做英文的自我介绍，或者就用英文和你对话，那么在电话面试前就要做好英文对话的准备。电话面试时一般不会涉及太多技术问题。

1.3.3　面试

对于求职者而言，面试是重点环节。有的公司的面试都会分为几轮来进行，由不同部门的人员来负责，通过各个方面来考察求职者。每一轮的面试，求职者都应该做好充分的准备。虽然你不可能预测到面试中的每一个问题，但是可以通过针对那些常见的问题，准备一些稳重而简洁的答案，以此提前来做一些准备。大多数的招聘者都会问以下一些相似的问题：

（1）谈谈你自己。

这个问题其实就是你自己的一个个人一分钟广告，概括出你这些年在这些所面试工作领域中的经历、能力和性格。抓住关键，做一次专业的个人推销。使用一些简洁的句子，证明你自己所拥有的（经历、已证实的成绩、奉献精神）是可以胜任这份工作的。

（2）为什么我们会录取你？

在这里，你需要突出那些你所拥有的，并且对这个公司有价值的特质。

（3）为什么你想来这里工作？关于我们公司你了解多少？

你可以通过这个机会展现你对这个公司有多么的了解，更重要的一点是展现你是多么的适合这个职位。

（4）你有哪些优点？

你回答时应当首先强调你适应的或已具有的技能。雇用你的决定在很大程度上取决于这些技能，你可以在后面详细介绍你与工作有关的技能。回答时，一定要简单扼要。

（5）你的不足之处是什么？

回答这个问题的秘诀是把你的不足转化成你的优势。确保你的回答能答到点上。没有人是完美的，所以，不要假装自己是圣人。

（6）对于你的上一份工作，你有哪些不满意的？你为什么离开了你的前任工作？

你必须对你所应聘的职位有深入的了解，才能把这个问题转变成一个正面的问题。最好这样去说，对于一份工作我喜欢它的许多方面，然后专注谈这份新工作是如何给你提供

机会在某一个特定的领域去奉献自己的力量，这个特定的领域就是此职位的关键。

（7）你 5 年内的打算是什么？

招聘者显然是不想听到你 5 年内的愿望是去哪里旅游，或者是在不同的行业工作。你应该谈论一些与这份工作有关的目标。这将证明你了解这个行业、了解这个公司，并且你渴望在这里能够取得成功。

沉着自信的回答任何一个问题的关键就是有备而来。永远记住，无论什么问题，如果你非常适合，并能够在这个职位上做出巨大的贡献，招聘者正在试图去发现。即使你不会技术上的问题，如果你表现出良好的求知欲望以及谦虚的作风，同样能给面试官留下很好的印象。

注意：一定要与面试官保持目光接触。有这么一句话：“眼睛是心灵的窗户”。这样能显示出你的友好、真诚与自信。很多面试者由于不良习惯或者紧张，不与对方保持目光接触，这样会让对方认为你对目前的谈话冷淡、说谎或者缺乏安全感等感觉。

1.4　小　　结

本章对应聘软件开发职位前需要知道的一些东西进行了讲解，主要包括明确自己的求职态度、应聘的渠道和流程、简历的写法与注意事项、笔试、面试和面试官常问的一些问题及其回答思路。这些知识是作者对多年以来软件开发应聘的概括和总结，这些知识或多或少能帮助读者的应聘。读者也应该熟知这些应聘知识，以增强自己的信心。

第2篇 Java基础知识

第 2 章　Java 程序基础

Java 是一门语言，同时也是一种技术规范，其涵盖了桌面应用程序、企业级程序、移动设备应用程序等各个方面。对 Java 基础概念的深入理解对于掌握 Java 更高层次的技术来说，是非常重要的。大多数的 IT 公司都会要求面试者对 Java 的基础概念有比较深入的理解，而不仅仅浮于表面。本章探讨关于 Java 开发环境的搭建和一些基础概念的问题，并且分析这些题目和知识点，帮助读者梳理这些方面的知识。

2.1　开发和运行环境

对于任何软件技术来说，开发和运行环境是一切的基础，即使写一个 Hello World 程序，也需要以开发和运行环境的搭建为前提。Java 的开发环境的搭建能够折射出 Java 技术的一些核心的概念来，对开发环境的理解程度也能够反映出求职者的一些 Java 功底。

面试题 001　JDK 和 JRE 的区别是什么？它们各自有什么作用

JDK 和 JRE 是 Java 开发和运行工具，其中 JDK 包含了 JRE，但是 JRE 是可以单独安装的，它们在 Java 开发和运行的时候起到不同的作用。本例在回答该问题的同时，全面地介绍 Java 的基础开发工具。

【出现频率】 ★★★★

【关键考点】

- JDK 对 Java 开发的作用；
- JRE 对 Java 程序运行的作用。

【考题分析】

关于 JDK 和 JRE，大家一定会记得，在安装 JDK 的时候需要分为两个步骤：安装 JDK 和安装 JRE。大家一般都会一起安装，也建议大家这样做。因为这样更能帮助大家弄清楚它们的区别。

JRE 是 Java Runtime Environment 的缩写，是 Java 程序的运行环境。既然是运行，当然要包含 JVM，也就是大家熟悉的 Java 虚拟机，还有所有 Java 类库的 class 文件，都在 lib 目录下，并且都打包成了 jar。至于在 Windows 上的虚拟机是哪个文件呢？就是<JRE 安装目录>/bin/client 中的 jvm.dll。

JDK 是 Java Development Kit 的缩写，是 Java 的开发工具包，主要包含了各种类库和工具，当然也包括了另外一个 JRE。那么为什么要包括另外一个 JRE 呢？而且<JDK 安装

目录>/JRE/bin 目录下，包含有 client 和 server 两个文件夹，它们都包含一个 jvm.dll 文件，这说明 JDK 提供了两个不同的虚拟机。另外，JDK 的 bin 目录下有各种 Java 程序需要用到的命令，与 JRE 的 bin 目录最明显的区别就是 JDK 下才有 javac，这一点很好理解，因为 JRE 只是一个运行环境而已，与开发无关。正因为如此，具备开发功能的 JDK 所包含的 JRE 下才会同时有 client 的 JVM 和 server 的 JVM，而仅仅作为运行环境的 JRE 下，只需要 client 的 jvm.dll 就够了。

注意：JDK 所提供的运行环境和工具都需要进行环境变量的配置以后，才能使用。最主要的配置就是把“<JDK 安装目录>/bin 目录”设置成为 Path 环境变量值的一部分。

另外，安装 JRE 的时候安装程序会自动把 JRE 的 java.exe 添加到了系统变量中。系统环境变量 Path 的最前面有%SystemRoot%system32;%SystemRoot%;这样的配置，那么到 Windows/system32 目录下面去看看，会发现一个 java.exe 文件。这样就无需配置环境变量，也可以运行 Java 程序了。

【答案】

JDK 是 Java 开发工具，它不仅提供了 Java 程序运行所需的 JRE，还提供了一系列的编译、运行等工具，如 javac、java、javaw 等。JRE 只是 Java 程序的运行环境，它最核心的内容就是 JVM（Java 虚拟机）及核心类库。

面试题 002　如何利用 JDK 编译和运行应用程序

JDK 是 Sun 公司提供给 Java 程序员的开发工具包，除了提供必要的运行环境以外，最主要的工具就是编译和运行，也就是 javac 和 Java 命令。如何使用这两个命令是 Java 程序员应该掌握的最基本的知识。本例在回答该问题的同时，详细地讲解 javac 和 java 命令的使用方法。

【出现频率】 ★★★

【关键考点】

- JDK 的安装与配置；
- javac 命令的使用方法；
- java 命令的使用方法。

【考题分析】

JDK 安装好以后，就可以使用 javac 和 java 命令工具了，它们均在<JDK 安装目录>的 bin 文件夹下。所以，还需要把这个目录的路径配置到 Path 环境变量中，大致步骤如下所示。

1. 打开环境变量设置对话框

右击“我的电脑”图标，在弹出的快捷菜单中，选择“属性”选项，弹出“系统属性”对话框。在该对话框中选择“高级”选项卡，单击【环境变量】按钮，此时，就打开了“环境变量”对话框。上半部分是用户环境变量，下半部分是系统环境变量。

2. 新建JAVA_HOME环境变量

在系统的环境变量中，单击【新建】按钮，弹出“新建环境变量”对话框。在其中输入变量名 JAVA_HOME；变量值 C:\Program Files\Java\jdk1.6.0_13，如图 2.1 所示。单击【确定】按钮，就完成了新建 JAVA_HOME 环境变量。

3. 编辑Path环境变量

在系统的环境变量中，选择变量名为 Path 的环境变量，单击【编辑】按钮，弹出“编辑环境变量”对话框。在原有的变量值后面加上“;%JAVA_HOME%\bin”，如图 2.2 所示。单击【确定】按钮，就完成了新建 JAVA_HOME 的环境变量。

注意：别把分号遗漏了。

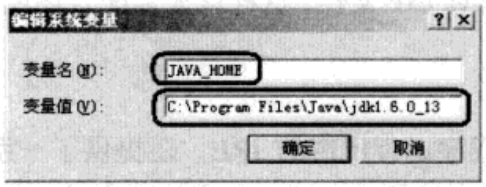


图 2.1　新建 JAVA_HOME 环境变量

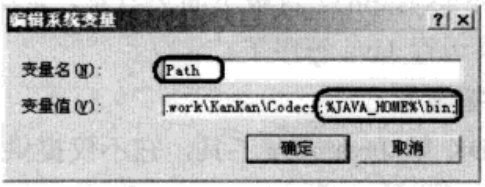


图 2.2　新建 JAVA_HOME 环境变量

通过以上的配置后，就可以在任何命令提示符窗口中使用 javac 和 java 命令了。其中，javac 是用来把源码编译成 class 文件的，而 java 是用来运行包含主方法的 class 文件的。以下是一个简单的 Hello World 程序源码：

```
//HelloWorld类
public class HelloWorld {
    //main()主方法
    public static void main(String[] args) {
        //打印一句话
        System.out.println("Hello World!");
    }
}
```

将以上源码保存为“D:\work\java\HelloWorld.java”，然后打开一个命令提示符窗口，通过 cd 命令来到“D:\work\java\”目录。输入 javac HelloWorld.java，按下 Enter 键，此时可以发现，在“D:\work\java\”目录下多了一个 HelloWorld.class 文件，这就是编译好的 class 文件。最后，输入 java HelloWorld，可以看见打印在屏幕上的“Hello World!”字样，如图 2.3 所示。

注意：主类的名字必须与文件名的前缀相同。

如果类指定了包名，则可以为 javac 命令加上“-d”选项，在编译的时候自动生成与包相对应的目录层次。以下是在当前目录下生成与包名相对应的目录层次的编译示例：

```
javac -d . HelloWorld.java
```

【答案】

利用 JDK 提供的 javac 命令来编译源文件，利用 java 命令来运行 Java 程序。为了更加

方便地使用这两个命令，需要把<JDK 的安装目录>/bin 配置到 Path 环境变量中。

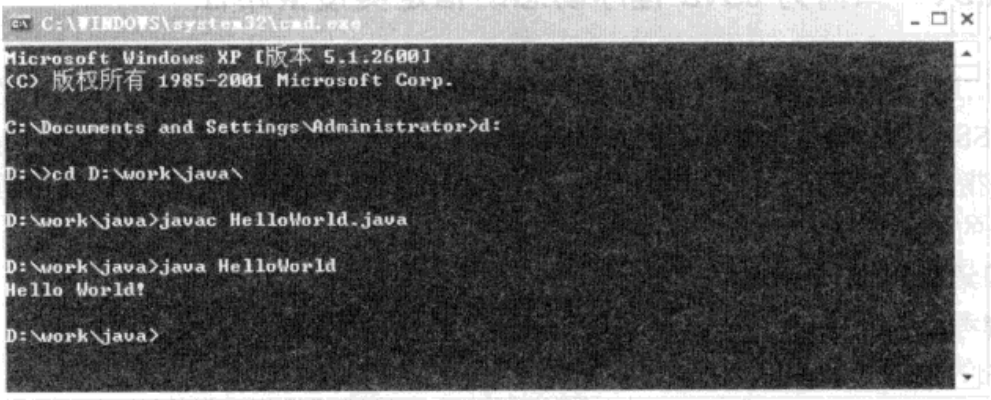


图 2.3　javac 和 java 命令使用示例

面试题 003　环境变量 CLASSPATH 的作用是什么

CLASSPATH 环境变量是用来指定 Java 程序搜索类的路径的，对于 Java 应用程序来说，该变量的意义是非常重要的。但是，许多程序员对它在 Java 程序中的作用却认识的不是很清楚。本例在回答该问题的同时，全面介绍 Java 应用程序搜索和加载类的工作机制。

【出现频率】 ★★★★

【关键考点】

- CLASSPATH 环境变量的作用；
- 类的搜索和加载原理。

【考题分析】

CLASSPATH 环境变量是在编译 Java 源码和运行程序时使用的，也就是为 Java 程序所依赖的接口、类等指定一个搜索路径。

其写法与 Path 变量类似，每个路径用分后进行分开，如果是一个 jar 文件，则直接写明该文件的绝对路径，如下所示。

```
.;c:\jar\logj4.jar;d:\work\java
```

以上的 CLASSPATH 环境变量指定了 3 个搜类路径：当前目录、logj4.jar 文件的路径和 d:\work\java 文件夹下的所有的类。

技巧：当前目录是一个常用的路径，所以 CLASSPATH 往往会包含它，也就是一个点号。

其实，Java 程序在编译和运行的时候，不仅在 CLASSPATH 中去搜索类，系统还会在 JRE 的目录下去找一个名为 rt.jar 的文件，其路径为 jre/lib/rt.jar。并且，它们是有一定搜索顺序的，先搜索的是 rt.jar，然后才是 CLASSPATH 指定的路径。

【答案】

CLASSPATH 环境变量保存的是一些目录和 jar 文件的地址，这些路径是为 Java 程序在编译和运行的时候搜索类而用的。

面试题 004　如何为 Java 程序动态的指定类搜索路径

大家知道，在默认情况下，Java 程序在编译和运行的时候，会通过 JRE 下的 rt.jar 文件和 CLASSPATH 环境变量所指定的路径进行类的搜索和加载，但是如果有时需要动态为 Java 程序指定类加载路径，又该如何办呢？本例在回答该问题的同时，详细讲解 java 和 javac 命令的使用方法。

【出现频率】 ★★★★

【关键考点】

- 类的搜索和加载原理；
- java 和 javac 命令的“-cp”和“-classpath”选项的作用。

【考题分析】

有的时候，为了动态的为 Java 程序指定类加载路径，开发者会写一些批处理文件来进行 Java 程序的编译或运行，如 Windows 下的 bat 文件和 Linux 下的 sh 文件。此时，就有一个问题，这些 Java 程序可能在不同的环境下，需要的类文件是不同的，因此，就需要使用 java 和 javac 提供给开发者的选项进行编译或运行。

动态的提供类加载路径的选项有两个即-cp 和-classpath，它们的作用是一样的，都是指定 CLASSPATH 的意思。开发者只需要在选项后边加上路径或路径变量即可。下面为一个 Hello World 程序提供一个额外的 jar 文件到 CLASSPATH 中。

```
javac -cp D:\work\java\log4j.jar HelloWorld.java
java -cp D:\work\java\log4j.jar HelloWorld
```

或

```
javac -classpath D:\work\java\log4j.jar HelloWorld.java
java -classpath D:\work\java\log4j.jar HelloWorld
```

此时，JVM 就会把选项指定的 jar 文件作为 CLASSPATH 的一个部分。当然，不仅是 jar 文件，也可以是一个路径或一个环境变量，例如：

```
set TEST=d:\test\java
javac -cp %TEST% HelloWorld.java
java -cp %TEST% HelloWorld
```

【答案】

JDK 中的 java 和 java 命令，提供了“-cp”和“-classpath”选项为 Java 程序动态地指定类搜索路径，它们的使用方法比较简单，只需要把相应的目录路径或 jar 文件路径跟在选项后边即可。

2.2　Java 语言概述

Java 作为一门计算机语言，它的编译和运行都是比较有特点的。理解 Java 语言的时候，必须深刻地理解它核心的编译和运行机制，最重要的两个概念就是字节码（class）文件和

Java 虚拟机（简称为 JVM）。

面试题 005　Java 与 C++程序在编译和运行上有什么区别

通过 Java 语法与 C++语法的相似度来看，Java 语言的设计与 C++有着千丝万缕的联系。但是它们却有着许多的不同，其中，编译和运行是它们最大的不同之处之一。本例在回答该问题的同时，全面介绍 Java 的编译和运行过程。

【出现频率】 ★★★★

【关键考点】

- C++的编译和运行过程；
- Java 的编译和运行过程。

【考题分析】

众所周知，任何一门计算机高级语言都会最终变成机器码（也就是二进制）以后，才会被计算机所识别。其中，与机器码最为接近的当属汇编了，而 Java 和 C++都会直接或间接的变成汇编以后，然后再运行。

对于像 C、C++这类高级计算机语言来说，它们的编译器（例如，Unix 下的 CC 命令，Windows 下的 CL 命令）都是把源码直接编译成计算机可以认识的机器码，如 exe、dll 之类的文件，然后直接运行即可。

Java 语言的跨平台性是它最大的亮点之一，为了达到平台无关性，它就不得不多一个中间步骤，也就是生成字节码文件。对于一个 Java 源文件来说，需要用 javac 命令把源文件编译成 class 文件，这个 class 文件是计算机无法直接识别的。但是却可以被 Java 虚拟机所认识，所以在运行一个 Java 程序的时候，肯定是需要启动一个 Java 虚拟机，然后再由虚拟机去加载这些 class 文件，如图 2.4 所示。

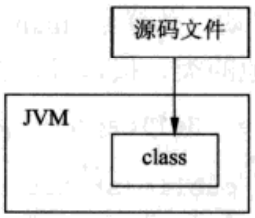


图 2.4　Java 程序编译和运行过程示意图

注意：class 文件指的是字节码文件，而不专指类编译后的文件。不管是类、接口、枚举或其他类型，都是编译成 class 文件的。

【答案】

C++源码编译以后，生成的是特定机器可以直接运行的文件，而 Java 源码经过编译后，生成的是中间的字节码文件。这些字节码文件是需要放在 JVM 中运行的，而 JVM 是有多个平台版本的。因此，Java 具有跨平台性，而 C++没有。

面试题 006　什么是 JVM 及其工作原理

JVM 是 Java 程序运行的平台，它就好像一台虚拟出来的计算机一样，负责执行 Java 编译好的字节码文件。JVM 具有非常严格的实现规范，大多数操作系统都可以安装 JVM，为 Java 语言的跨平台性起到了关键的作用。本例在回答该问题的同时，将更深入地讲解

JVM 的工作原理。

【出现频率】 ★★★★★

【关键考点】

- JVM 的作用；
- JVM 的工作原理。

【考题分析】

JVM（Java 虚拟机）是一个想象中的机器，在实际的计算机上通过软件模拟来实现。Java 虚拟机有自己想象中的硬件，如处理器、堆栈、寄存器等，还具有相应的指令系统。

说明：JVM 有很多个实现，目前用得比较多的就是 Sun 公司提供的 JRE 的 JVM，另外，IBM、BEA 等公司都有自己的 JVM 实现。

Java 语言的一个非常重要的特点就是与平台的无关性。而使用 Java 虚拟机是实现这一特点的关键。它就好像一张毯子，铺在具体操作系统平台的上面，垫在 Java 语言的下面。一般的高级语言如果要在不同的平台上运行，至少需要编译成不同的目标代码。而引入 Java 语言虚拟机后，Java 语言在不同平台上运行时不需要重新编译。Java 语言使用模式 Java 虚拟机屏蔽了与具体平台相关的信息，使得 Java 语言编译程序只需生成在 Java 虚拟机上运行的目标代码（字节码），就可以在多种平台上不加修改地运行。Java 虚拟机在执行字节码时，再把字节码解释成具体平台上的机器指令执行。

下面通过一个具体的例子来分析它的运行过程。虚拟机通过调用某个指定类的方法 main()启动，传递给 main()一个字符串数组参数，使指定的类被装载，同时链接该类所使用的其他的类、接口等，并且初始化它们，最后执行方法中的代码，示例程序如下：

```
class HelloApp
{
    public static void main(String[] args)
    {
        System.out.println("Hello World!");          //输出内容
        for (int i = 0; i < args.length; i++ )
        {
            System.out.println(args);                //输出参数
        }
    }
}
```

编译后在命令行模式下输入以下代码：

```
java HelloApp run virtual machine
```

上面代码通过调用 HelloApp 方法 main()来启动 Java 虚拟机，传递给 main()一个包含 3 个字符串 run、virtual、machine 的数组。

开始试图执行类 HelloApp 的 main()方法的时候，发现该类并没有被装载，也就是说虚拟机当前不包含该类的二进制代表，于是虚拟机使用 ClassLoader 试图寻找这样的二进制代表。如果这个进程失败，则抛出一个异常。类被装载后同时在 main()方法被调用之前，必须对类 HelloApp 与其他类型进行链接并初始化。类的初始化是对类中声明的静态初始化函数和静态域的初始化构造方法的执行。然后就开始执行 main()方法中的代码，直至方法被执行完毕。

【答案】

JVM 是一种用软件模拟出来的计算机，它用于执行 Java 程序，有一套非常严格的技术规范，是 Java 跨平台特性的依赖基础。Java 虚拟机有自己想象中的硬件，如处理器、堆栈、寄存器等，还具有相应的指令系统，它运行 Java 程序就好像一台计算机运行 C 或 C++程序一样。

面试题 007　Java 程序为什么无须 delete 语句进行内存回收

垃圾回收机制是 Java 相对于 C++的一种巨大改进，它避免了因为程序员忘记释放内存而造成内存溢出的错误。所以，在 Java 程序中，根本不需要使用 delete 语句，JVM 会自动的去检查哪些内存应该回收了，在后台就自动回收了，为程序员省了不少的事情。本例在回答该问题的同时，深入地讲解 JVM 的垃圾回收机制。

【出现频率】 ★★★★

【关键考点】

- JVM 的垃圾回收机制；
- JVM 在内存中存储对象的原理。

【考题分析】

大家知道，Java 除了那 8 种基本类型以外，其他都是对象类型（又称为引用类型）的数据。JVM 会把程序创建的对象存放在堆空间中，那什么又是堆空间呢？其实，堆（Heap）是一个运行时的数据存储区，从它可以分配大小各异的空间。一般，运行时的数据存储区有堆（Heap）和堆栈（Stack），所以要先看它们里面可以分配哪些类型的对象实体，然后才知道如何均衡使用这两种存储区。一般来说，栈中存放的是非 static 的自动变量、函数参数、表达式的临时结果和函数返回值（如果他们没有被放到寄存器中）。栈中的这些实体数据的分配和释放均是由系统自动完成的，堆中存放的实体数据都是程序中显式分配的，没有自动垃圾回收机制的系统中必须由程序代码显式地释放这些实体。

堆的管理，不同的语言实现是不同的。如 C 语言就没有把堆的分配和释放做到语言的层次，它对堆空间对象的操作是通过其库函数 malloc()和 free()来实现的；而 C++直接把对堆空间中的对象的分配和释放做到语言层次了。使用 new 和 delete 语句，Java 就做得更彻底，应用开发者只要在需要用堆分配的时候创建就行了，何时释放如何释放，都有 Java 虚拟机（JVM）来做，而不需要程序代码来显式地释放。

说明： Java 虚拟机规范并没有强制规定要实现自动垃圾回收功能，但目前大多数 JVM 都实现了自动垃圾回收机制，只是它们各自的实现算法不同。

JVM 有着各种版本的实现，它们基本上都会有垃圾回收的机制，也就是堆内存的管理的自动进行。那么，又该如何知道对象已经被回收了呢？Java 中根父类 java.lang.Object 中有个 finalize()方法，它会在垃圾回收器认为这个对象是垃圾的之后，真正回收之前被调用。因为所有的类都继承自 Object，所以它们都会有 finalize()方法。程序员可以在这个方法中写一些需要在对象被回收前做的事情，例如关闭数据库连接。

finalize()方法原型如下：

```
protected void finalize() throws Throwable
```

一般在调用这个方法之前，垃圾回收器能检测出不再被引用的对象，如果这些对象覆盖了 finalize()方法，就要调用该方法。

另外，在 java.lang.System 类中，有一个 gc()方法，它对 JVM 的垃圾回收也有一些影响。通过显式的调用它可以请求开始垃圾回收器线程，开始垃圾回收，但垃圾回收线程是否立即开始还是由 JVM 的算法决定的。

java.lang.Runtime 类的 gc 方法与 System 的作用一样，只不过 Runtime 是一个单例模式的类，需要用 getRuntime()方法来获得它的实例，然后才能调用 gc()方法，代码如下所示：

```
System.gc();
Runtime.getRuntime().gc();
```

注意：垃圾回收线程是一个优先级很低的线程。

【答案】

Java 的堆内存数据的释放功能是由垃圾回收器自动进行的，无需程序员显式的调用 delete 方法。该机制有效的避免了因为程序员忘记释放内存而造成内存溢出的错误，相对于 C++等需要显式释放内存的语言，是一种巨大的改进。

2.3　生成、部署和配置

使用 Java 作为编程语言来写程序总是会遇到各种各样的规范。写 Web 程序需要遵守 Web 程序规范，写 EJB 程序需要遵守 EJB 的规范，即使是一个普通的 Java 程序也会遵守一定的编程规范。但是，对于大多数类型的 Java 程序来说，均会包含生成、部署和配置的步骤，程序员不仅需要能够写程序，还需要对这些规范有一定的了解。

面试题 008　如何利用命名提示符把 Java 程序打包成 jar 文件

jar 文件是 Java 程序打包以后的格式文件，它一般保存的有 class 文件、配置文件等。jar 文件的主要意图就是方便程序的发布和部署，JDK 提供了一个 jar 工具命令，可以利用它来对 Java 程序进行打包。本例通过回答该问题来详细地阐述 jar 命令的使用方法。

【出现频率】 ★★★★

【关键考点】

- jar 文件的用途；
- jar 命令的使用方法。

【考题分析】

其实，jar 命令不仅仅可以用来打包 Java 程序，只要是跟 jar 有关的内容，都可以通过 jar 命令来完成，命令格式如下：

```
jar {c t x u f }[ v m e 0 M i ][-C 目录]文件名...
```

其中{c t x u}这 4 个参数必须选其一。[v f m e 0 M i]是可选参数，文件名也是必须的。

- -c：创建一个 jar 包；
- -t：显示 jar 中的内容列表；
- -x：解压 jar 包；
- -u：添加文件到 jar 包中；
- -f：指定 jar 包的文件名；
- -v：生成详细的报告，并输出至标准设备；
- -m：指定 manifest.mf 文件。manifest.mf 文件中可以对 jar 包及其中的内容作一些设置；
- -0：产生 jar 包时，不对其中的内容进行压缩处理；
- -M：不产生所有文件的清单文件（Manifest.mf），这个参数将忽略掉-m 参数的设置；
- -I：为指定的 jar 文件创建索引文件；
- -C：表示转到相应的目录下执行 jar 命令，相当于先执行 cd 命令转到指定的目录，然后不带-C 执行 jar 命令。

通过这些参数就可以完成 jar 文件的各种操作，以下是一些使用范例：

（1）创建 jar 包

```
jar cf hello.jar HelloWorld.class
```

利用 test 目录生成 hello.jar 包，如 hello.jar 存在，则覆盖。

（2）创建并显示打包过程

```
jar cvf hello.jar HelloWorld.class
```

利用 hello 目录创建 hello.jar 包，并显示创建过程。

（3）显示 jar 包

```
jar tvf hello.jar
```

查看 hello.jar 包的内容。

注意：指定的 jar 包必须真实存在，否则会抛出 FileNoutFoundException 异常而导致失败。

技巧：利用 Windows 平台的 WinRAR 程序也是可以直接打开和解压 jar 文件的。

（4）解压 jar 包

```
jar xvf hello.jar
```

解压 hello.jar 至当前目录。

（5）jar 中添加文件

```
jar uf hello.jar HelloWorld.java
```

将 HelloWorld.java 添加到 hello.jar 包中。

【答案】

利用 JDK 的 bin 目录下的 jar 命令就可以完成 Java 程序的打包，一般需要包含程序所需的 class 文件、配置文件和 manifest.mf 文件。其中 C、-V 和-f 这 3 个命令参数最为多见。以下为一个把当前目录的所有文件都打包成 jar 文件的示例：

```
jar cf test.jar .
```

面试题 009　关于 Java Web 项目的生成、部署和配置问题

Java Web 是 Java 使用最为广泛，技术发展最为成熟的一个领域，它的相关问题也是面试的热点。如何生成（Build）、部署（Deploy）和配置（Configuration）是 Web 程序最基本的知识，也是 Java Web 应用程序最基本的开发规范。本例通过回答该问题来全面地阐述 Java Web 应用程序的一些开发规范。

【出现频率】 ★★★★★

【关键考点】

- Java Web 应用程序的目录规范；
- Java Web 应用程序的基本配置方法；
- Java Web 应用程序的打包与部署。

【考题分析】

Java Web 开发指的是使用 Java 语言，并按照 Java EE 规范开发的 Web 应用程序，这些应用程序需要可以部署到任意符合该规范的 Web 容器中运行。Java EE 规范定义非常广泛，Web 只是它其中的一个方面，它定义了一个标准的 Java Web 应用程序的各种规范，包括：目录结构、Web 配置文件、打包和部署、Servlet、JSP 等。

1. 目录结构

Java Web 程序的所有文件需要包含在一个文件夹中，该文件夹的目录结构有一定的规定。必需包含一个名为 WEB-INF 的文件夹，该文件夹对于客户端来说是隐藏的，WEB-INF 文件夹还包含了存放类文件的 classes 文件夹和存放类库文件的 lib 文件夹，以及 Web 描述文件 web.xml；与 WEB-INF 文件夹同一层次的目录存放的是 JSP、HTML 等页面文件。如图 2.5 所示一个名为 javaweb 的应用程序的目录结构。

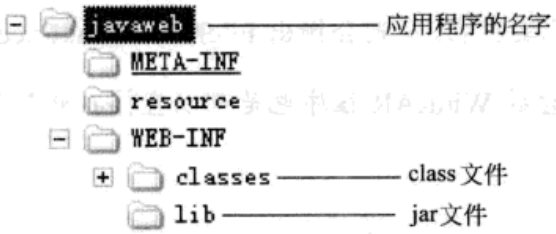


图 2.5　javaweb 应用程序的目录结构

2. Web配置文件

Web 配置文件指的是 web.xml，它是整个 Web 应用程序的配置文件，通过它定义 Servlet、过滤器、监听器和一些参数等信息。Web 容器通过该文件的配置来控制整个 Web 应用程序的行为方式。

web.xml 的书写是需要符合一定规则的。根标签为<web-app>，然后在它里面配置 Servlet（用<servlet>和<servlet-mapping>标签）、过滤器（用<filter>和<filter-mapping>标签）、

监听器（用<listener>标签）。

说明：web.xml 必须存放在 WEB-INF 目录下面。

3．Servlet和JSP

Servlet 是服务器端处理 HTTP 请求的基本组成单元，包括 JSP、过滤器等在内的许多技术都基于 Servlet 实现。Servlet 是符合一定规范的 Java 类，它存活在 Web 容器中，由容器来控制 Servlet 的生命周期。

JSP 是参考其他动态语言而设计，脚本语言就是 Java，但是它的本质也是 Servlet，它比其他动态语言更加强大。严格意义来说，熟悉 JSP 开发，并不意味着就熟悉了 Java Web 开发，只能说是使用动态语言开发 Web 程序。JSP 作为 Java Web 开发是不可或缺的一部分，在大多数的开发中，它往往是用于显示处理后的结果。

注意：如果读者知道 Applet（Java 的一种 Web 富客服端技术）技术的话，请与之区分，Applet 是在浏览器中运行的 Java 程序，这项技术使用已经非常少了。全书提到的 Java Web 开发均指的是 Java 服务器端技术的开发。

4．打包和部署

与 Java 桌面应用程序类似，Web 用程序也是可以打成 jar 包的，只不过它的后缀名为 War，而不是 jar。但是，jar 和 War 的打包原理是一样，都可以通过 JDK 提供的 jar 命令进行打包。

部署对于 Java Web 应用程序来说，是再通用不过的了。因为 Java 语言是跨平台的，Java Web 容器是符合 Java EE 规范的，所以每个 Java Web 应用程序都可以不加修改的部署到任何平台的任何 Java EE 容器中。例如，对于 Windows 平台上的 Tomcat，可以把 Java Web 应用程序的文件夹或 War 文件放在<Tomcat 安装目录>/webapps 下面来完成部署。

说明：各种 Web 容器的部署方式是不同的，请参见容器提供的相关文档。

【答案】

Java Web 应用程序的生成，需要把它需要的 class 文件编译好以后存放在 WEB-INF/classes 目录下，然后按照以上介绍的目录结构放置各类文件。如果需要添加自己的配置，需要增加或修改 WEB-INF/web.xml 文件。对于 Java Web 应用程序的部署，可以使用文件夹和 War 文件两种部署方式，其中 War 就是把文件夹按照 jar 的方法进行打包，只不过后缀为 War。

面试题 010　EJB 项目的生成和部署问题

EJB（Enterprise Java Bean）是 Java EE 的一部分，定义了一个用于开发基于组件的企业多重应用程序的标准，是分布式开发的一把利器。EJB 广泛应用于大型的分布式项目中，例如银行、电信等行业，所以它也通常出现在面试题中。本例通过回答该问题来全面地阐述 EJB 程序的一些开发规范。

【出现频率】 ★★★★

【关键考点】

- EJB 的概念和构成原理；
- EJB 程序的开发过程。

【考题分析】

EJB 作为 SUN 公司 Java EE 中的一套规范，它规定了一系列的 API 并用来实现把 EJB 概念转换成 EJB 产品。各个服务器厂家都会都会根据这个规范来开发 EJB 容器，开发人员也是根据这些规范来开发 EJB 程序，并且把它部署到符合规范的 EJB 容器中，例如，JBOSS、Weblogic、Websphere 等。

EJB 定义了 3 类的 Bean，分别是会话 Bean（Session Bean）、实体 Bean（Entity Bean）和消息驱动 Bean（MessageDriven Bean）。

- Session Bean 用于接收客户端的请求和实现业务逻辑，它可以是有状态的，也可以是无状态的。每当客户端请求时，容器就会选择一个 Session Bean 来为客户端服务；
- Entity Bean 是实体对象，用于实现 O/R 映射，负责将数据库中的表记录映射为内存中的 Entity 对象。事实上，从 EJB 3.0 开始，实体 Bean 已经被 JPA 所替代，使它开发更简单、扩展性更强；
- MessageDriven Bean 是基于 JMS 消息的，只能接收客户端发送的 JMS 消息然后处理，它适合做异步开发。

目前，EJB 3.0 使用的更加广泛一些，它相对之前的版本，大多数配置都是通过 Annotation 进行的，这使得开发更加简单了。

与 Web 应用程序有一点类似，EJB 的开发过程也是充满着规范。生成（Build）的时候，首先把 EJB 程序需要的 Class 文件进行编译，对于 JPA，需要把配置文件 persistence.xml 存放在 META-INF 目录下。部署（Deploy）期间，可以把这些文件进行 jar 打包，也是以文件夹的形式进行部署，对于 JBOSS 服务器，把它放在 deploy 文件夹下即可。

注意：不同的 Java EE 服务器对于 EJB 程序的部署方式是不同的，请参见各类 Java EE 容器所提供的相关文档。

【答案】

EJB 项目的生成过程主要有两个步骤：编译 class 文件和在特定位置中存放配置文件，例如，对于 EJB 3.0 程序的 JPA，就需要在 META-INF 文件夹下存放持久化的配置文件 persistence.xml。部署的过程也比较简单，就是把打包好的 jar 文件或不打包的文件夹，存放在 Java EE 服务器指定的路径下即可。

2.4 小　　结

本章涵盖了一些 Java 技术基础的面试题，包括开发环境、语言编译和运行原理、Java 程序的生成、部署和配置等方面。万丈高楼平地起，尽管这些东西都是非常基础的，但是它们也是最不容易掌握的知识，对这些基础知识和概念的深刻理解可以有助于开发者掌握 Java 更高层次的技术，所以它们也是软件公司经常会问到的面试题。求职者在知道这些题目答案的同时，应该加入一些自己的理解。

第 3 章　Java 语法基础

Java 语言的语法与 C/C++颇为类似，有点 C/C++的遗风，因此，有 C/C++编程基础的程序员学习 Java 是相对比较容易的。但是，一些语法相似的背后却有着本质意义上的区别，读者在了解 Java 语法的同时，需要区别它们的原理和实现与 C/C++的不同之处，这样才能更深刻的掌握 Java 的语法。本章将包含关于 Java 语法基础的一些问题，并且分析这些题目和知识点，帮助读者梳理这些方面的知识。

3.1　基础类型和语法

计算机软件就是数据与程序的集合。数据是任何软件操作的对象，任何一门计算机语言都会通过一定的方式来操作数据，把数据分为不同的类型，例如，整型、字符型、浮点型等。针对这些数据又有一系列的操作语法，形成一定的规则，让这些数据为开发者服务。Java 语言也不例外，数据类型和基础语法是 Java 语言最基本的内容，就相当于中文的拼音、英语的音标一样，是其他一切的基础。

面试题 011　变量及其作用范围

在大多数面向过程的语言中（例如 C 语言），变量可分为全局变量和局部变量。全局变量指的是可以被所有的函数在任何地址使用的变量，而局部变量则是在某一个特定的代码范围才能看见的变量。Java 是一门面向对象的编程语言，它的变量及其作用范围又有所不同，理解 Java 变量的含义及其作用范围是一项比较重要的基本技能。本例在回答该问题的同时，全面地介绍 Java 变量的含义和作用范围概念。

【出现频率】 ★★★★

【关键考点】

- 什么是变量；
- Java 变量的类别和含义；
- Java 变量的作用范围。

【考题分析】

在中学数学中，大家会学到函数，它包含了 1 个或多个变量，变量就代表一个未知数，一般用 x 表示。与之类似，计算机程序中的变量是指在程序的运行过程中，随时可以发生变化的量，是数据的临时存放场所。变量中可以存放单词、数值、日期、文本字符等数据，它们都有一个与其他变量相区别的名字，每种编程语言对变量的命名有不同的规定。此外，所有的变量都有一个作用域，用于定义变量的可见性和生存期。

在 Java 中，变量根据生成周期的不同可以分为静态变量、成员变量和局部变量 3 种。静态变量指的是在类中使用 static 修饰的变量，它的生存周期是由类来决定的，当类加载时，它们就生成并初始化。成员变量则是在类中没有使用 static 修饰的变量，它属于该类的某个实例，也就是对象，它们随着对象的加载而生成并初始化，随着对象被垃圾回收器回收而消失。局部变量则是定义在方法中的变量或方法的参数，它们随着方法的调用而创建，随着方法的执行完毕而消失。

另外，对于局部变量来说，它们还可以定义在代码块中，也就是用大括号包围起来的部分。例如，if、else、for 语句的范围中，以及直接用{}符号包围起来的代码块。尽管定义在里面的变量是不为外边所见的，但是代码块中不可以定义与外部变量名一样的变量，如下代码则会产生编译错误。

```
public static void main(String[] args) {
    int i = 0;
    {
        int i;
    }
}
```

在 main()方法中定义了 int 型的 i 变量，代码块中则不可以定义同名的 i 变量。但是，局部变量的定义却可以和成员变量、静态变量的定义同名，如下代码所示。

```
public int a;
public static int b;
public void testMethod(){
    int a;
    int b;
}
```

那么，又该如何在 testMethod()方法中使用成员变量的 a 和 b 呢？其实很简单，通过 Java 提供的 this 关键字即可，它代表的是当前对象的引用，指向了当前的对象。例如，访问成员变量 a 的代码如下所示：

```
this.a
this.b
```

说明：上例的 b 变量是一个静态成员变量，一般不推荐通过 this 来访问它，因为它是属于类范围的，所以使用类名来访问它会更合理一些。例如，类名为 TestClass，则可以用 TestClass.b。

【答案】

Java 变量可以分为：静态变量、成员变量和局部变量 3 种。静态变量指的是在类中用 static 修饰的变量，它的生存周期是由类来决定的。成员变量则是在类中没有用 static 修饰的变量，它的生存周期由对象来决定。局部变量则是定义在方法里的变量、方法的参数或代码块里定义的变量，它们的作用范围用大括号{}来界定。

面试题 012　Java 的变量分哪两种大的数据类型

Java 是一门面向对象的语言，在它的世界中总是充满了对象。其实，对象是一种数据

的表现形式，但是 Java 的所有数据都是对象形式的吗？当然不是，Java 还有 8 种基本数据类型。本例在回答该问题的同时，全面地介绍 Java 数据类型的相关概念。

【出现频率】 ★★★★★

【关键考点】

- Java 的数据类型；
- 基本数据类型和引用数据类型的区别。

【考题分析】

Java 一共有 8 种基本数据类型，分别是 byte、short、int、long、float、double、boolean 和 char。它们存储的都是数据量比较小的数据，只需要 1 个或少量几个字节就可以了。因此，这些数据类型的变量的二进制值就直接保存着它们的值。

而引用数据类型指的是除开基本数据类型以外的数据类型，它主要用来指向某个对象，有一点像 C/C++指针，但是它没有指针那么灵活。对象保存的数据一般都比较大，如果在传递参数、赋值等工作的时候，每次都把数据完全的拷贝一次就会影响到效率。因此，Java 就为所有操作对象的类型定义了一种引用类型的数据，图 3.1 为两种数据类型的对比。

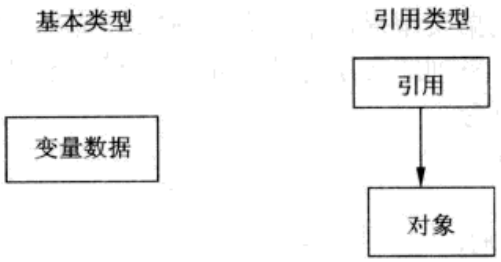


图 3.1　两种数据类型对比

注意： 引用数据类型的变量，在作用域上与基本数据类型是一样的，它们也有一定的生存周期。只不过它们存的值特别一点，是一个内存地址。

对象一般是通过 new 语句创建出来的，它保存在堆空间中，可以被多个引用类型的变量所引用。当引用 a 修改了对象的数据以后，其他的引用也是可以看到这些改变的，一个很典型的例子就是方法参数的传递代码如下所示。

```
package ch03;
class Student{                                          //学生类
    public String name;                                 //姓名
    public int age;                                     //年龄
    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
public class RefTest {
    private static void refChg(Student stu){            //修改对象数据的测试方法
        stu.name = "wangwu";
        stu.age = 10;
    }
    public static void main(String[] args) {
```

```
        Student stu1 = new Student("zhangsan",23); //创建学生对象 1
        Student stu2 = stu1;                       //把学生 1 赋值为学生 2
        refChg(stu2);
        System.out.println(stu1.name+","+stu1.age);//打印学生 1 的数据
    }
}
```

以上代码中，stu1 和 stu2 引用了同一个学生对象，stu2 数据的修改就直接影响到 stu1 的数据了，因为它们根本就指向同一块内存空间，执行结果如下：

```
wangwu,10
```

【答案】

基本数据类型和引用数据类型。它们最大的区别在于，引用数据类型存放的是数据所在的地址，而基本数据类型则是直接存放数据的值。

面试题 013　Java 包含哪些基本数据类型及其包装类

Java 一直在努力让它的一切都变成对象，但是出于效率的考虑，一直保留着基本数据类型，并且为每一种基本数据类型提供了与之对应的包装类。包装类不仅是对基本数据类型的对象包装，还提供了一系列的实用方法。本例在回答该问题的同时，全面介绍基础数据类型及其封装类的相关概念。

【出现频率】 ★★★★

【关键考点】

- Java 的基础数据类型；
- 包装类的概念及其作用。

【考题分析】

Java 一共提供了 8 种基础数据类型，分别是 byte、short、int、long、float、double、boolean 和 char。它们所对应的包装类均在 java.lang 包中，分别是 Byte、Short、Integer、Long、Float、Double、Boolean 和 Character。

当有的地方必须要使用对象或引用数据类型的时候，例如集合类（List、Set 等），基本数据类型就不能使用了，因为它们的存储原理和引用数据类型是完全不一样的。把基本数据类型的变量转换成包装类型很简单，用 new 语句创建或调用包装类的一个静态方法 valueOf()，把基本类型的变量作为参数传入即可，示例代码如下：

```
int i = 10;                          //定义 int 变量 i
Integer itg = Integer.valueOf(i);    //用 valueOf()方法把 i 转成 Integer 变量
Itg = new Integer(i);                //用构造方法创建

List list = new ArrayList();         //创建一个 List 类型的集合变量
list.add(itg);                       //把封装类型的 itg 放入 list 中
```

说明：从 Java 5.0 开始，有一种自动装箱和拆箱的功能，就可以不必显式的进行类型转换，系统会自动的按照需要进行数据类型的转换。

而包装类转换成基本类型时，则调用包装类对象的 intValue()、shortValue()、doubleValue()等方法，代码如下所示。例如：

```
Integer itg = Integer.valueOf(100);
itg.intValue();
```

【答案】

Java 一共提供了 8 种基础数据类型及其包装类，如表 3.1 所示，它们均可以实现相互的转换。

表 3.1　Java 基本数据类型及其包装类

基本数据类型	包　装　类
byte	Byte
short	Short
int	Integer
float	Float
double	Double
boolean	Boolean
char	Character

面试题 014　如何理解 Java 中的装箱和拆箱

基本数据类型的自动装箱（autoboxing）和拆箱（unboxing）是 Java 5.0 提供的新功能。虽然为打包基本数据类型提供了方便，但是在提供方便的同时却隐藏了一些细节，建议在能够区分基本数据类型与对象的差别时再使用。本例在回答该问题的同时，深入地讲述装箱和拆箱的原理。

【出现频率】 ★★★★★

【关键考点】

- Java 的基础数据类型与包装类型的关系；
- Java 装箱和拆箱的原理。

【考题分析】

在 Java 中，所有要处理的东西几乎都是对象。操作对象比操作基本数据类型更方便一些，而基本数据类型的效率更高。因此，在开发过程中，两者的转换是经常需要的。

在 Java 5.0 之后提供了自动装箱的功能，开发者可以直接使用以下语句来打包基本数据类型：

```
Integer integer = 10;
```

在进行编译时，编译器再自动根据您写下的语句，判断是否进行自动装箱动作。在上例中，integer 变量是 Integer 类的实例，同样的动作可以适用于 boolean、byte、short、char、long、float、double 等基本数据类型，分别会使用对应的包装类型 Boolean、Byte、Short、Character、Long、Float 或 Double。以下是直接使用自动装箱功能的示例。

```
public class AutoBoxDemo {
    public static void main(String[] args) {
        Integer data1 = 10;
        Integer data2 = 20;
        //转为 double 值再除以 3
```

```
        System.out.println(data1.doubleValue() / 3);
        //进行两个值的比较
        System.out.println(data1.compareTo(data2));
    }
}
```

程序看起来简洁了许多，data1 与 data2 在运行时就是 Integer 的实例，可以直接进行对象操作，执行的结果如下：

```
3.3333333333333335
-1
```

Java 5.0 中可以自动装箱，也可以自动拆箱，也就是将对象中的基本数据形态信息从对象中自动取出，例如下面这样写是可以的：

```
Integer fooInteger = 10;
int fooPrimitive = fooInteger;
```

fooInteger 变量在自动装箱为 Integer 的实例后，如果被指定给一个 int 类型的变量 fooPrimitive，则会自动变为 int 类型再指定给 fooPrimitive。另外，在运算时，也可以进行自动装箱与拆箱，代码如下：

```
Integer i = 10;
System.out.println(i + 10);
System.out.println(i++);
```

上例中会显示 20 和 10，编译器会自动进行装箱与拆箱，也就是 10 会先被装箱，然后在 i + 10 时会先拆箱，进行加法运算，i++时也是先拆箱再进行递增运算。

注意：建议新手不要使用自动装箱、拆箱的语法，最好在对对象有较深入了解之后，再来使用这个功能。

【答案】

Java 中的装箱和拆箱指的是基本数据类型和包装类型的自动相互转换，它为开发者提供了方便。开发人员也可以不使用它，而手动的进行类型转换。并且，这个自动转换的过程是在编译阶段。

面试题 015　Java 的引用和 C++的指针有什么区别

大多数的 C++程序员转学 Java 的时候，总是会说这样一句话：Java 的引用就是 C++的指针嘛。其实不完全对，它们之间既有千丝万缕的联系，也有很大的区别。本例在回答该问题的同时，深入而全面地讲述 Java 的引用与 C++的指针的区别。

【出现频率】 ★★★★★

【关键考点】

- C++指针的工作原理；
- Java 引用的工作原理。

【考题分析】

Java 的引用和 C++的指针都是指向一块内存地址的，通过引用或指针来完成对内存数据的操作。就好像风筝的线轴一样，通过线轴总能够找到风筝。但是它们在实现、原理、

作用等方面却有区别。

（1）类型：引用其值为地址的数据元素，Java 封装了的地址，可以转成字符串查看，长度可以不必关心。C++指针是一个装地址的变量，长度一般是计算机字长，可以认为是个 int。

（2）所占内存：引用声明时没有实体，不占空间。C++指针如果声明后会用到才会赋值，如果用不到不会分配内存。

（3）类型转换：引用的类型转换，也可能不成功，运行时抛异常或者编译就不能通过。C++指针只是个内存地址，指向哪里，对程序来说还都是一个地址，但可能所指的地址不是程序想要的。

（4）初始值：引用初始值为 java 关键字 null。C++指针是 int，如不初始化指针，那它的值就不是固定的了，这很危险。

（5）计算：引用是不可以计算的。C++指针是 int，它可以计算，如++或--，所以经常用指针来代替数组下标。

（6）控制：引用不可以计算，所以它只能在自己程序中，可以被控制。C++指针是内存地址，也可以计算，所以它有可能指向了一个不属于自己程序使用的内存地址，对于其他程序来说是很危险的，对自己程序来说也是不容易被控制的。

（7）内存泄露：Java 引用不会产生内存泄露。C++指针是容易产生内存泄露的，所以程序员要小心使用，及时回收。

（8）作为参数：Java 的方法参数只是传值，引用作为参数使用时，回给函数内引用的值的 COPY，所以在函数内交换两个引用参数是没意义的，因为函数只交换参数的 COPY 值，但在函数内改变一个引用参数的属性是有意义的，因为引用参数的 COPY 所引用的对象是和引用参数是同一个对象。C++指针做为参数给函数使用，实际上就是它所指的地址在被函数操作，所以函数内使用指针参数的操作都将直接作用到指针所指向的地址（变量，对象，函数等）。

总地来说，Java 中的引用和 C++中的指针本质上，都是想通过一个叫做引用或者指针的东西，找到要操作的目标，方便在程序中操作。所不同的是 Java 的办法更安全和方便一些，但没有 C++的那么灵活。

【答案】

Java 的引用与 C++的指针主要有以上介绍的 8 点区别。本质上，它们两者都是想通过一个叫做引用或者指针的东西，找到要操作的目标，方便在程序中操作。所不同的是 Java 的办法更安全、方便一些，但失去了 C++的灵活，也算是对指针的一种包装和改进。

面试题 016　请简述 Java 中的 main()方法

main()方法是 Java 程序的入口方法，它有一定的讲究，程序员应该熟知 main()方法的定义规则、参数传递等有所了解。因此，main()方法及其相关知识经常会出现在面试官的题目里。本例在回答该问题的同时，深入而全面地讲述 Java 的引用与 C++的指针的区别。

【出现频率】 ★★★★

【关键考点】

- main()方法的定义规则；
- 如何为 Java 应用程序提供参数。

【考题分析】

与 C/C++程序类似，Java 应用程序也需要一个入口函数，它就是 main()方法。main()方法是一个公开的、静态的、无返回值的、参数为一个字符串数组的方法，而且方法名必须为 main。另外，因为 Java 的方法都必须定义在类中，所以 main()方法是属于某一个类的静态方法，示例代码如下：

```
public class Main {
    //main()方法，程序的入口函数
    public static void main(String[] args) {
        //...
    }
}
```

作为 Java 程序的入口函数，它还肩负着接收外部参数的作用，也就是通过命令提示符输入的参数。main()方法的字符串数组参数 args 就是用来接收参数的，它是一个数组的形式，它的长度与实际输入参数的个数一致，以下是一段打印用户年龄的示例程序代码。

```
public class PrintAge {
    public static void main(String[] args) {
        if(args == null || args.length != 2)    //判断参数是否符合要求
            return;
        String user = args[0];                  //获取用户名参数
        String age = args[1];                   //获取年龄参数
        //打印
        System.out.println("User "+user+" is "+age+" years old.");
    }
}
```

将以上源码进行编译完成后，在用 java 命令执行的时候，就可以输入用户名和年龄的参数来进行执行，控制台就可以打印出相应的语句，如输入以下命令：

```
java PrintAge zhangsan 29
```

控制台打印出以下语句：

```
User zhangsan is 29 years old.
```

【答案】

main()方法是 Java 程序的执行入口，它是一个定义在类中的、公开的、静态的、无返回值的、参数为一个字符串数组的方法，它的参数 args 与执行参数一一对应。

面试题 017　Java 中 equal 和==的区别是什么

equal 和“= =”均表示相等的意思，但是它们在进行实际的相等判定时，却有非常大的区别，这还得从 Java 的堆栈说起。初学者对它们两者的概念的理解很容易模糊，在实际编程工作中就容易造成一些不易发现的 Bug，理解它们两者的区别是非常重要的。本例在

回答该问题的同时，深入而全面地讲述 Java 中的 equal()方法和==运算符的执行细节。

【出现频率】 ★★★★★

【关键考点】

- ==运算符的执行原理；
- equal()方法的含义。

【考题分析】

在开始讲解它们两者的区别之前，首先来看一段示例代码：

```
public class EqualTest {
    public static void main(String[] args) {
        //定义 3 个字符串对象
        String a = "1234";
        String b = "1234";
        String c = new String("1234");
        //3 种不同的布尔判定
        System.out.println(a==b);
        System.out.println(a==c);
        System.out.println(a.equals(c));
    }
}
```

以上代码中，定义了 3 个字符串变量，它们的值都是“1234”，按理来说，对它们进行布尔判定都应该返回 true。但事实上，执行结果应该是：

```
true
false
true
```

第 2 个为 false 的原因在于 a 和 c 指向的是不同的对象。==运用在基本数据类型的时候，通过比较它们实际的值来判定是否相同；而用于比较引用类型的时候，则是比较两个引用的地址是否相等，也就是是否指向同一个对象。通过 new String()来创建的字符串会单独生成一个对象，所以 a 和 c 指向的不是同一个对象。

注意：Java 的双引号表达式本身就会创建一个字符串对象。例如，1234，就创建了一个值为 1234 的 String 对象。

Equal()方法是 java.lang.Object 的方法，也就是所有的 Java 类都会有的方法。它可以被程序员覆盖重写，通过自定义的方式来判定两个对象是否相等，其中默认的方式与==相同。另外，java.lang.String 类是一个特殊的类，它不可以被继承，它的 equal()方法用来比较字符串的字符序列是否完全相等。以下是一段学生对象的相等判定的示例代码。

```
//学生类
class Student{
    private String name;                    //姓名
    private int age;                        //年龄
    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }
    //比较方法，只有姓名和年龄都相等才相等
    public boolean equals(Object obj) {
        Student stu = (Student) obj;
```

```
        return this.name.equals(stu.name) && this.age==stu.age;
    }
}
public class EqualTest {
    public static void main(String[] args) {
        //比较两个学生对象是否相等
        Student stu1 = new Student("lisi",10);
        Student stu2 = new Student("lisi",10);
        System.out.println(stu1.equals(stu2));
    }
}
```

执行结果为 true。如果把其中一个的姓名或年龄改掉，则结果变为 false。

【答案】

equal 和“==”两者均为表示相等的意思，但是它们相等的含义却有所区别。

“==”运用在基本数据类型的时候，通过比较它们实际的值来判定是否相同；而用于比较引用类型的时候，则是比较两个引用的地址是否相等，也就是是否指向同一个对象。

equal 方法是 java.lang.Object 的方法，也就是所有的 Java 类都会有的方法。它可以被程序员覆盖重写，通过自定义的方式来判定两个对象是否相等。对于字符串 java.lang.String 类来说，它的 equal 方法用来比较字符串的字符序列是否完全相等。

面试题 018　Java 提供了哪几种循环结构？它们各自的特点是什么

循环结构语句是编程中使用最为频繁的结构语句之一。在 Java 中一共提供了 3 种循环结构语句，它们适用于不同的情况，对它们的深入理解可以更好的提高开发效率和程序执行效率。本例在回答该问题的同时，全面讲述 Java 每种循环结构的用法。

【出现频率】 ★★★

【关键考点】

- for 语句；
- while 语句；
- do…while 语句。

【考题分析】

与 C/C++类似，Java 提供了 3 种循环结构，即 for、while 和 do…while 语句。它们没有谁好谁不好的说法，各自适用于不同的情况。其中，for 更适合于能确定循环次数的循环结构；while 语句则适合单条件的循环；而 do…while 语句更适合于执行某段代码之后再循环的情况。

说明：其实，它们 3 者是可以相互替代的，这跟开发者的习惯也有关系。

for 语句的语法结构如下所示：

```
for(初始化;条件判断;条件变量处理){
    //循环代码
}
```

在 for 语句的括号中，一共有 2 个分号，3 句表达式。第 1 个是初始化表达式，也就是在进入循环之前执行的；第 2 个是条件表达式，也就是每次循环的条件，一旦条件返回 false

则退出循环；第 3 个一般用于对条件变量的处理，例如 i++、i--等。以下是一段打印 1～100 的整数示例代码。

```
for(int i=1;i<=100;i++){
    System.out.println(i);
}
```

在 while 语句的括号中，则只需要提供条件表达式即可，同样是打印 1～100 的整数，也可以这样来写：

```
int i =1;
while(i<=100){
    System.out.println(i);
    i++;
}
```

对于 do…while 语句，它与 while 语句相似，只不过条件是在循环代码执行完以后才去判断，语法如下：

```
do{
    //循环代码
}while(条件);
```

注意：do…while 语句的条件括号后边一定要加上分号。

如果用 do…while 语句打印 1～100 的整数，代码如下：

```
int i = 1;
do{
    System.out.println(i++);
}while(i<100);
```

【答案】

Java 提供了 3 种循环结构，即 for、while 和 do…while 语句。它们各自适用于不同的情况，其中，for 循环适合于能确定循环次数的循环结构。while 语句则适合单条件的循环。而 do…while 语句在执行某段代码之后，再循环的时候更适合一些。

面试题 019　Java 中的三元运算符是什么

Java 只有一个三元运算符，就是“表达式一？表达式二:表达式三”，它的运用可以减少代码量。但是，三元运算的使用规则对于初学者来说是一大难点，稍有不慎就会成为产生 Bug 的地方。本例在回答该问题的同时，将更深入地讲解 Java 三元运算符的使用规则。

【出现频率】 ★★★

【关键考点】

- 三元运算符的使用方法

【考题分析】

三元运算符的使用语法如下所示：

```
表达式一？表达式二:表达式三
```

在问号“?”之前是一个布尔表达式，它只能返回 true 或 false，如果表达式一返回的是 true 则执行表达式二，否则执行表达式三，并产生相应的返回值。并且，后两个表达式的返回类型要统一。

三元运算符不是必须的，它完全可以被 if 和 else 语句所替代，它最大的作用也就是减少了代码量，让代码更简洁、更直观。

说明： 三元表达式中的“元”指的是该运算符需要多少个表达式的意思，例如，++、--属于一元运算符，+、-、*等，则为二元运算符。

以下是一段使用三元运算符打印最大值的示例代码：

```
public class Max {
    public static void main(String[] args) {
        int a = 5;                                          //定义变量 a
        int b = 8;                                          //定义变量 b
        String rst = a > b ? "a bigger than b" : "b bigger than a";
                                                            //三元运算符
        System.out.println(rst);
    }
}
```

运行结果如下：

```
b bigger than a
```

以上代码中的三元运算符可以被以下代码所替代，它的作用也就一目了然了。

```
String rst = null;
if(a>b){
    rst = "a bigger than b";
}else{
    rst = "b bigger than a";
}
```

【答案】

Java 的唯一一个三元运算符为“表达式一？表达式二:表达式三”。在问号?之前是一个布尔表达式，它只能返回 true 或 false，如果表达式一返回的是 true 则执行表达式二，否则执行表达式三，并产生相应的返回值。它的主要作用是为了使代码更简洁。

面试题 020　Java 中的注释有哪些

Java 有两种注释与 C/C++类似，就是行注释和块注释，这也是大多数语言支持的注释方式。另外，Java 还有一种被叫做文档注释的注释，它主要是用来生成 API 文档的，以及 Java 5.0 以后出现的 Annotation。对于后两种注释，一些程序员可能会对它们的概念有所模糊和混淆。本例在回答该问题的同时，深入地讲解 Java 的各种注释及其规则。

【出现频率】 ★★★

【关键考点】

- 行注释和块注释的使用；
- 文档注释的含义；

❑ Annotation 的含义及其作用。

【考题分析】

（1）行注释。

注释格式：// 注释内容

将代码的一行进行注释，也就是当这一行代码不存在，它不会进入编译层。

（2）块注释。

注释格式：/*……*/

注释若干行，通常用于方法、数据结构等的意义与用途的说明，或者算法的描述。一般位于一个文件或者一个方法的前面，起到引导的作用，也可以根据需要放在合适的位置。这种域注释不会出现在 HTML 文档中。

（3）文档注释。

注释格式：/**……*/

注意：文档注释比块注释多一个星号。

也用于注释若干行，并写入 javadoc 文档。注释文档将用来生成 HTML 格式的代码 API 报告，还可以用@开头的关键字进行注释，以达到说明方法参数、返回值、异常等各种详细信息。

（4）Annotation。

严格意义上来讲，Annotation 是不能算作注释的，它最终会进入编译层，并对程序的结果产生影响。它最普遍的作用就是用来替代 XML 提供一些配置信息，例如，JPA、Spring 等框架的配置信息就可以通过 Annotation 来提供。

【答案】

如果不算入 Annotation，Java 的注释有 3 种，即行注释、块注释和文档注释。它们往往适合于不同地方的注释，其中文档注释比较特殊，它的注释信息可以进入到 javadoc 文档中。

但是如果把 Annotation 也算作 Java 的注释的话，Java 就有 4 种注释。Annotation 与其他注释本质的区别就在于它会进入到编译层，并对程序结果产生影响。

3.2　对象和类型

面向对象（Object Oriented，简称 OO）是在当今编程界中被普遍认可的，而 Java 语言又是众多编程语言中 OO 体现得最典型的语言之一。因此，理解 Java 的面向对象特性是学习 Java 过程中非常重要的一个环节，它也是 Java 一切高级应用的基础。

面试题 021　类和对象有什么区别

类和对象是面向对象最基础的两个概念。对象代表了数据和操作的一种封装，而类又是对象的一种抽象。对象是通过类来创造的，类就好像一个模子，对象就是通过该模子来

创造的一个个实例。本例在回答该问题的同时，详细介绍 Java 的对象和类的概念以及它们的使用规则。

【出现频率】 ★★★★★

【关键考点】

- 类的概念；
- 对象的概念；
- 如何通过类来创造对象。

【考题分析】

面向对象的思想是把程序中的一切都看成是对象，也就是当作一个具体的物件来看待。对象拥有各种属性和动作，就好像一只狗它有两只耳朵、4 条腿，它可以做出跑、跳、呼吸、睡觉等动作。同时，这些对象拥有一种共性，也就是它们同属于一类，例如，所有的狗都有两只耳朵、4 条腿，都做出跑、跳、呼吸、睡觉等动作。

在 Java 中，把这种共性称为类（class），每一个实例称为对象（Object）。在类中定义属性和方法，每个对象都用 new 关键字和类名来创造。以下是一段定义一个狗（Dog）类和创建狗对象的示例代码：

```
class Dog{
    int age;                                    //年龄
    String name;                                //名字
    Dog(){                                      //无参数的构造方法
    }
    Dog(int age, String name) {                 //有参数的构造方法
        this.age = age;
        this.name = name;
    }
    void run(){                                 //跑的方法
        System.out.println("running...");
    }
}
public class ObjectTest {
    public static void main(String[] args) {
        Dog dog1 = new Dog();                   //用无参的构造器创建对象
        Dog dog2 = new Dog(6,"乐乐");           //用有参的构造器创建对象
    }
}
```

以上代码中，定义了一个 Dog 类，采用关键字 class 进行定义；它包含了两个属性（或者称为成员变量），年龄（age）和名字（name），以及一个跑（run）的方法（又称为成员方法）；在 main()方法中，采用 new 关键字进行两个狗对象的创建，分别使用无参数和有参数两种构造器（又称为构造方法）进行对象的创建；其中，dog1 的年龄为 0，名字为空，而 dog2 的年龄为 6，名字为“乐乐”。

说明： 在 Dog 类的有参数构造方法中，使用了 this 关键字，它代表了当前对象的引用，例如，当创建 dog2 对象时，this 就代表 dog2，如果还有一个 dog3 也通过该构造方法进行对象的创建时，则 this 就指向 dog3。

类的构造方法是一种比较特殊的方法，它不能被程序员显式的调用，只能在创建对象时由系统自动调用。构造方法的名字必须与类名完全相同。另外，在没有提供任何的构造

方法时，系统会为类创建一个默认的构造方法，该构造方法也是无参数的，它什么也不做，但是，一旦提供了任何一种构造方法，该默认的构造方法就不会被自动提供了。

当需要使用对象的属性和方法的时候，只需要通过对象的引用加点号“.”进行调用即可。例如，访问狗对象的名字以及调用 run()方法可以这样写代码：

```
System.out.println(dog2.name);
dog2.run();
```

类也可以有属于它的属性和方法，也就是静态（static）成员，通过 static 关键字进行定义。这些静态属性和方法属于类所有，被该类的所有对象所共享，但是它只有一份，并不会随着对象的创建而新增。例如，为 Dog 类定义一个用于统计狗对象总数的静态变量，可以这样来写：

```
class Dog{
    static int count;                    //狗对象的创建数量
    ⋮
}
```

【答案】

Java 的类通过 class 关键字进行定义，它代表了一种抽象的集合，例如，学生类、动物类等，在它的里面可以定义各种属性和方法，它们代表了每个类实例的特定的数据和动作。Java 虚拟机对类只加载一次，对它的静态成员也只加载一次。

对象，指的是某一个特定抽象的实例，它属于某一种类型，也就是对象是通过类来创建的。它必须从属于某一个类，通过 new 关键字进行创建，它代表一个特定类型的实例，对象可以被多次创建。

简而言之，类是一种抽象，而对象是类的实现。

面试题 022　Java 中如何使用继承来重用代码

继承是面向对象技术的一个基本特征，它是一种常见的代码重用的手段。Java 语言有着一套非常成熟的单继承机制，既没有像 C++那样的多继承所带来的复杂性，又不缺乏继承的灵活。本例在回答该问题的同时，详细介绍 Java 的继承机制。

【出现频率】 ★★★★

【关键考点】

- 继承的概念；
- 继承的使用；
- 方法覆盖。

【考题分析】

Java 的继承只有单继承，也就是子类只能去继承一个父类，而不能继承多个父类，下面为继承的示例代码：

```
//动物类
class Animal{
    void breath(){                       //呼吸
        System.out.println("Breath...");
    }
```

```
    void cry(){                                    //叫
        System.out.println("cry...");
    }
}
//猫类，继承自动物类
class Cat extends Animal{
    void cry(){                                    //覆盖 cry()方法
        System.out.println("苗...");
    }
}
```

以上代码中定义了两个类，一个是动物（Animal）类，另一个是猫（Cat）类。Cat 从 Animal 继承而来，通过使用 extend 关键字即可。这样一来，Cat 就拥有了 Animal 除开私有成员以外的所有成员变量和成员方法，这也就是通过继承来重用代码的形式。

示例代码中，Cat 自身没有定义 breath()方法，但是它是包含了 breath()方法的。另外，在扩展的过程中，如果需要对原有方法进行修改，只需要定义一个名字、参数、返回值一样的方法即可，父类的方法就被覆盖了，也叫做方法覆盖。

注意：方法的覆盖（或重写）和重载是两个不同的概念。重载是创建一个方法名相同，但参数列表不同的方法；而方法覆盖则是子类对父类方法的覆盖和重写。

```
public class ExtendTest {
    public static void main(String[] args) {
        Animal animal = new Animal();              //创建一个动物对象
        Cat cat = new Cat();                       //创建一个猫对象
        animal.breath();                           //调用动物的呼吸方法
        animal.cry();                              //调用动物的喊叫方法
        cat.breath();                              //调用猫的呼吸方法
        cat.cry();                                 //调用猫的喊叫方法
    }
}
```

动物和猫的呼吸（breath）方法是一样的，而喊叫（cry）方法却不同。这是因为 Cat 类把 cry()方法进行了覆盖，而 breath 方法依然使用父类原有的。

说明：Java 程序中，所有类都继承自 java.lang.Object 类，它有 9 个方法，也就是说，所有的 Java 类都包含了这 9 个方法。

【答案】

Java 采用的是单继承制，使用 extend 关键字。通过继承以后，子类就拥有了父类除开私有成员以外的所有成员，从而达到代码重用的目的。在继承过程中，可以通过方法的覆盖来实现多态，让子类拥有自己独特的方法实现方式。

面试题 023　简述 Java 中的多态

多态、封装和继承是一脉相承的，多态基于它们两者。多态让面向对象的程序变得异常的灵活和易扩展，可以说正是多态的特性让 Java 等一批面向对象的语言不断的发展壮大，多态在 Java 中的作用可见一斑。本例在回答该问题的同时，详细介绍 Java 的多态

机制。

【出现频率】 ★★★★★

【关键考点】

❑ 多态的概念

【考题分析】

“多态”一词按照字面意思来理解为“多种形式，多种状态”。Java 程序员可以把多态看作对象的一种能力，使其能调用正确的方法版本，它是建立在继承的基础之上的。先看一段示例代码：

```
class Man{
    int eyes = 2;                    //眼睛数量，为 2
    String getEyesColor(){           //返回人眼睛的颜色
        return null;
    }
}
class AsiaMan extends Man{           //亚洲人
    String getEyesColor(){           //覆盖返回人眼睛的颜色的方法，为黑色
        return "black";
    }
}
class EuroMan extends Man{           //欧洲人
    String getEyesColor(){           //覆盖返回人眼睛的颜色的方法，为蓝色
        return "blue";
    }
}
```

以上示例中，定义了一个 Man 父类，它包含一个 getEyesColor()的方法；然后，由 Man 类又继承出 AsiaMan 和 EuroMan 两个类，它们都将 getEyesColor()方法进行了覆盖重写。在这里，多态的概念就体现出来了，同样都是人，亚洲人和欧洲人的眼睛颜色是不同的，但是它们的眼睛都有颜色。由于它们都继承自 Man 类，所以可以把它们都看作是 Man 类型，只是它们的眼睛颜色不同，也就是 getEyesColor()方法的实现各不相同。

说明：本示例采用的是一个具体的类作为超类，其实也可以是抽象类和接口，在实际的开发中，接口使用得更多一些。

```
public class PolymorphismTest {
    public static void main(String[] args) {
        Man asiaMan = new AsiaMan();          //创建一个亚洲人，转换成 Man 类型
        Man euroMan = new EuroMan();          //创建一个欧洲人，转换成 Man 类型
        System.out.println("Asia men eyes' color is " + asiaMan.getEyes-
        Color());
        System.out.println("Euro men eyes' color is " + euroMan.getEyes-
        Color());
    }
}
```

以上代码中，asiaMan 和 euroMan 都是 Man 类型，但是它们的实现不同，表现出来的形态也就不同，这就是多态，代码的运行结果如下：

```
Asia men eyes' color is black
Euro men eyes' color is blue
```

【答案】

“多态”一词按照字面意思来理解为“多种形式，多种状态”。它的本质是，发送消息给某个对象，让该对象自行决定响应何种行为。通过将子类对象引用赋值给超类对象引用变量来实现动态方法调用。

面试题 024　请介绍 Java 中静态成员的特点

静态成员是一组比较特殊的成员，它不属于某个特定的类实例，而是属于一个类所有，这个类的所有实例可以共享它们。静态成员的使用尽管没有普通对象成员那么频繁，但是它们的唯一性却决定了它们在现实开发中往往会起到非常大的作用。本例在回答该问题的同时，详细介绍 Java 的静态成员的原理和运行机制。

【出现频率】 ★★★★

【关键考点】

- 静态变量的原理和机制；
- 静态方法的使用；
- 静态代码块的运行机制。

【考题分析】

类的静态成员，指的是用 static 修饰的成员，概括起来有 3 种：静态成员变量、静态方法和静态代码块。它们都具有以下一些特点。

- 在类加载的时候，就进行创建和初始化或执行代码；
- 它们对于一个类来说，都只有一份；
- 类的所有实例都可以访问到它们。

（1）静态成员变量，指的是用 static 关键字修饰的成员变量，它会在类加载以后进行创建和初始化操作，例如以下示例代码中的 a 变量，它初始化以后的值为 0。因为它的唯一性，通常用于对象的数据记录，例如，单例模式下的引用保存。

（2）静态方法，指的是用 static 关键字修饰的方法，它可以被对象访问，也可以直接通过类名来访问，例如以下示例代码中的 testMethod()方法。

（3）静态代码块，采用 static 修饰，用大括号“{...}”包围起来的代码。这些代码可以使用静态成员变量和方法，它们也是在类加载的时候被调用。

```
public class StaticTest {
    static int a;                                    //静态成员变量，int 型的 a
    static void testMethod(){                        //静态方法
        System.out.println("test method...");
    }
    static{                                          //静态代码块
        System.out.println("execute static codes...");
    }
    public static void main(String[] args) {
        System.out.println(a);                       //打印静态成员变量
        StaticTest obj = new StaticTest();
        obj.testMethod();                            //通过对象来访问静态方法
    }
}
```

在 StaticTest 类中，包含了一个 a 静态成员变量、一个 testMethod()静态方法以及一段静态代码块。当执行以上代码的时候，JVM 会加载 StaticTest 类到内存当中来，然后创建 a 变量进行并初始化，再执行 static 代码块。在 main()方法中，创建了一个 StaticTest 类的实例对象 obj，通过它来访问静态方法 testMethod，以上代码的执行结果如下：

```
execute static codes...
0
test method...
```

注意：以上代码在编译过程中会报出一个警告，也就是不建议通过对象来访问静态方法，而应该直接通过类名来访问，毕竟静态成员是属于类所有的。

【答案】

类的静态成员是通过 static 关键字修饰的成员，主要包括：静态成员变量、静态方法和静态代码块，它们具有以下一些特点。

- 在类加载的时候，就进行创建和初始化或执行代码。
- 它们对于一个类来说，都只有一份。
- 类的所有实例都可以访问到它们。

面试题 025　简述 Java 派生类中的构造方法如何为父类传递参数

整个 Java 的类体系是一个庞大的继承网络，它们的总派生类是 java.lang.Object，通过这样一层一层的继承来组成 Java 的功能体系。在它们的继承过程中，子类的创建是会以父类的构造方法的调用为前提，也就是子类的创建总是建立在父类的基础之上的。本例在回答该问题的同时，详细介绍 Java 的多态机制。

【出现频率】 ★★★★

【关键考点】

- 子类创建对象的原理；
- 如何调用父类构造方法。

【考题分析】

所有的类在没有提供其他构造方法的时候，都会由系统提供一个默认的无参数的构造方法。如果所有的类都使用这个默认的构造方法，那一切天下太平，所有的子类直接创建就好，系统会隐式地调用父类的默认的构造方法。但是，有的父类并没有提供该构造方法，而是一些需要传入参数的构造方法，那子类就必须为父类传入这些参数了。

super 关键字是用来访问父类的成员的，它可以用于调用被子类重写的父类成员方法，也可以用来调用父类的构造方法，示例代码如下：

```
class Base{                          //父类
    int a;                           //成员变量 a
    Base(int a){                     //有参的构造方法
        this.a = a;
    }
}
class Child extends Base{            //子类
```

```
    Child() {                          //无参的构造方法
        super(100);                    //为父类传入一个 100 的参数
    }
    Child(int a) {                     //有参的构造方法
        super(a);                      //为父类传入一个 a 参数
    }
}
```

以上代码中，父类 Base 只有一个需要传入一个 int 型参数的构造方法，因此子类 Child 就必须在它的构造方法中为父类提供该参数。子类 Child 有两个构造方法，一个是无参数的，另一个是有参数的，但是它们都必须调用 super()来为父类提供参数。在创建 Child 对象的时候，可以通过以下两种形式来进行：

```
new Child();                           //调用无参的构造方法
new Child(200);                        //调用有参的构造方法
```

说明：super()的使用必须放在子类构造方法的第一行，否则编译时会有语法错误。另外，如果使用父类默认的无参的构造方法，可以不必显式的使用 super()，系统会自动加上的。

【答案】

在 Java 中，使用 super 关键字加括号()的形式来为父类的构造方法提供参数，通过参数的数目和类型来决定调用哪一个构造方法。如果调用的是父类的默认的无参数构造方法，则可以不必显式地使用 super()。

面试题 026　简述接口和抽象类的区别

接口和抽象类指的都是不能具体描述一个对象的类型，包含了一些不完整的信息，需要实现类进行具体的实现。但是，它们是有区别的，适合的情况也不相同，理解它们之间的区别对于实际开发是很重要的。本例在回答该问题的同时，详细介绍 Java 的接口和抽象类。

【出现频率】 ★★★★★

【关键考点】

- 接口的概念；
- 抽象类的概念；
- 接口和抽象类的区别。

【考题分析】

在面向对象的概念中，所有的对象都是通过类来描绘的，但是并不是所有的类都是用来描绘对象的，如果一个类中没有包含足够的信息来描绘一个具体的对象，这样的类就是抽象类。具体地说，也就是包含抽象（abstract）方法的类，就是抽象类。

抽象类往往用来表示在对问题领域进行分析、设计中得出的抽象概念，一些问题是可以清楚描述的，而某些是未知的。例如，要描述“水果”，它就是一个抽象，它有质量、体积等一些共性，但又缺乏特性（水果都可以吃，但是它们的口感却不同），拿不出唯一一种能代表水果的东西（因为苹果、橘子都不能代表水果），此时就可以用抽象类来描述

它，但是抽象类是不能够实例化的。

接口则比抽象类更进了一步，它所有的东西都是抽象的，也就是所有的方法都是抽象方法，没有任何的实现。它也可以包含静态的变量，只不过这些变量的值是不能修改的。接口和抽象类都可以用于进行类型的强制转换。接口一般用来表示实现类具有某种特性，例如：可比较大小、可迭代等。以下是一段关于水果的抽象类和接口的示例代码：

```
abstract class Fruit{                          //水果类
    double weight;                             //重量
    double volume;                             //体积
    abstract String getTaste();                //味道
}
interface Packable{                            //可包装接口
    void pack();
}
interface Divisible{                           //可分瓣接口
    void divisible();
}

class Apple extends Fruit implements Packable{
    String getTaste() {                        //实现 getTaste()方法
        return "甜";
    }
    public void pack() {                       //实现包装方法
        System.out.println("包装苹果");
    }
}
class Orange extends Fruit implements Packable,Divisible{
    String getTaste() {                        //实现 getTaste()方法
        return "酸";
    }
    public void pack() {                       //实现包装方法
        System.out.println("包装橙子");
    }
    public void divisible(){                   //实现分瓣方法
        System.out.println("把橙子分瓣");
    }
}
```

抽象类，通过 abstract class 进行定义，它也是一种类，只不过它不可以实例化，例如上例中的 Fruit 类。接口通过“interface”进行定义，它不能被直接实例化，实现类通过 implements 关键字来声明实现的接口，例如，以上代码中的 Packable 和 Divisible 接口，分别代表水果可包装和可分瓣的性质。

注意：如果类包含一个或多个抽象方法时，该类必须声明为抽象类。

在上例中，苹果（Apple）类和橙子（Orange）类是 Fruit 类的实现类，它们都实现了 Packable 接口，Orange 还实现了 Divisible 接口。其实，一个实现类只能继承一个抽象类，但可以实现多个接口，这一点也是接口和抽象类最大的区别。这就好像，一个人只能有一个亲爸爸，但是可以有多个干爸爸一样。这些干爸爸都可以为这个孩子带来一些其他的特性。

在许多情况下，接口是可以代替抽象类的，如果不需要刻意表达属性上的继承的话。当然，在开发过程中，应该遵循一下继承的原始含义，的确是继承关系才使用继承，如果

不是，则使用接口更为妥当。

【答案】

抽象类是一种功能不全的类，接口只是一个抽象方法声明和静态不能被修改的数据的集合，两者都不能被实例化。从某种意义上说，接口是一种特殊形式的抽象类，在 Java 语言中，抽象类表示一种继承关系，一个类只能继承一个抽象类，而一个类却可以实现多个接口。

面试题 027　简述一下内部类的实质是什么

内部类是 Java 的类的一种形式，它的使用语法比较奇怪，让一些开发人员不能正确的理解它。那么，内部类与普通的 Java 类有什么区别呢？匿名内部类又是如何工作的呢？本例在回答该问题的同时，详细介绍匿名内部类的分类和使用方法。

【出现频率】 ★★★★

【关键考点】

- 内部类的含义；
- 内部类的分类；
- 内部类的使用方法。

【考题分析】

内部类就好像一只寄生虫一样，生存在其他类的内部。定义在类内部的类就叫做内部类，它缩小了可见性，例如下面的代码：

```
package abc;
class A{                                          //定义外部类 A
    class B{                                      //定义内部类 B
        :
    }
}
```

对于 B 类来说，它的完整类名就是 abc.A.B，命名空间进一步缩小了，并且 B 对 A 产生了一种依赖关系，只有 A 存在的时候 B 才会存在。

根据内部类的定义结构的不同，可以把内部类分为两种：成员式和局部式。成员式内部类指的是它们定义的地方与成员变量和成员方法类似，就好像类的一个成员一样。局部式内部类则是定义在方法体中，仅属于局部范围所有。

然后，成员式内部类又可以分为：静态内部类和成员内部类。局部式又可以分为：普通局部内部类和匿名内部类。

1. 静态内部类

使用 static 关键字修饰的内部类，例如下面的代码：

```
package abc;
class Outter{                                     //定义外部类 Outter
    static class Inner{                           //定义静态内部类 Inner
        :
    }
}
```

在外部类加载的时候，静态内部类也随之加载，它的完整类名是 abc.Outter.Inner，编译后的 class 文件名为“Outter$Inner.class”。由于静态内部类是静态的，所以它无法访问外部类的非静态成员。其实，静态内部类相对于外部类来说，几乎是独立的，它可以在没有外部类对象的情况下，单独创造一个内部类的对象。

说明：某种程度上来说，公开的静态内部类就相当于一个普通的类。

总结一下，静态内部类相对于外部类来说，仅仅是包含关系，缩小了命名空间，完整类名中多了一个外部类的名称。本质上是两个独立的类，JVM 也不知道它们两个有包含关系。

2. 成员内部类

没有使用 static 关键字修饰的内部类，例如下面的代码：

```
package abc;
class Outter{                                    //定义外部类 Outter
    class Inner{                                 //定义成员内部类 Inner
        ⋮
    }
}
```

此时的内部类需要等外部类创建了对象以后才会被加载到 JVM 中，它属于外部类的某个实例，因此它可以访问外部类的静态和非静态成员。创建成员内部类的时候，语法比较特殊，首先创建一个外部类的实例，然后用这个实例调用 new 语句，代码如下：

```
public static void main(String[] args) {    //主方法
    Outter o = new Outter();                 //创建外部类实例
    Outter.Inner i = o.new Inner();          //创建内部类实例
}
```

对于成员内部类的构造方法来说，系统会为它们自动加上一个外部类的参数以及一个外部类的成员变量，这是为了遵循先有外部类实例才能有内部类实例的原则，代码如下：

```
class Inner(Outter o){                           //成员内部类的本质构造方法
    this.o = o;
}
```

当内部类访问外部类的成员时，则是通过该自动添加的成员变量进行访问，就好像下面的代码：

```
o.abc();
```

3. 局部内部类

局部内部类就没有范围的概念了，它仅在定义它的方法中有效，例如下面的代码：

```
public void adc(){                               //成员方法
    class MyLocal{                               //定义局部内部类
        ⋮                                        //局部内部类体
    }
}
```

本质上来说，局部内部类它也是以独立的类，只不过它的一些使用受到了限制。例如，它不能使用 static 关键字，只能使用 final 和 abstract 关键字，仅可以访问外部类带有 final 关键字的局部变量，因为它访问的是一个字面量或镜像，该局部变量已经不存在了。但是，它可以任意访问外部类对象的成员变量。

说明： 与成员内部类类似，局部内部类的构造方法也会自动加上一个外部类类型的参数，以及为该内部类加一个外部类型的成员变量。

当局部内部类定义在静态方法中时就相当于静态内部类；当定义在普通的成员方法中的时，则相当于成员内部类。

4. 匿名内部类

如果一个局部内部类没有名字，则它就是匿名内部类，例如下面的代码：

```
public void adc(){                                    //成员方法
    new OneInterface(){                               //直接 new 一个接口
        public void interMethod(){                    //接口方法
            :
        }
    };
}
```

在以上代码中，匿名内部类的定义和使用根本没有出现 class 关键字，但事实上它还是创建了。该类实现了 OneInterface 接口，直接在方法体中就提供具体的实现，如果需要提供构造方法的参数，则直接在 OneInterface 后边的那个括号中提供即可，显得非常的灵活。其实，它的工作原理相当于局部内部类，只是没有一个具体的名字而已，外部也无法直接使用它。

注意： 匿名内部类编译后的 class 文件的命名是按照匿名内部类的排列顺序来进行的，直接在外部类后面加上“$”和序号，例如，Outter$1.class。

【答案】

内部类根据定义的情况可以分为以下 4 种。

（1）静态内部类：它相当于外部类的静态成员一样，使用 static 修饰的内部类，它隶属于外部类，使用起来相当于一个独立的外部类。

（2）成员内部类：它相当于外部类普通的成员一样，隶属于外部类的具体对象，在定义它的时候，需要先创建外部类对象，再创建它的实例。

（3）局部内部类：它定义在一个方法的方法体中，它往往仅作为方法短暂的使用，只能访问用 final 修饰的局部变量。

（4）匿名内部类：它也定义在方法体中，但是它没有一个具体的名字，具有非常大的灵活性，工作本质与局部内部类类似。

3.3　包和访问控制

在 Java 中，类的数量是非常众多的，出现重名的概率很高，包（package）是 Java 用

来解决类重名的重要手段。既然有了不同的名字空间以后，就需要设置不同的访问控制，否则，程序就会显得很混乱。Java 提供了 4 种访问控制的手段：private、protected、public 和 default。本小节将集中讨论有关包和访问控制的常见面试题。

面试题 028　包应该如何被创建及使用

包（package）是 Java 程序的关于命名空间的另外一种叫法，它主要是为了避免类重名的情况。那么，如何创建一个包呢？包应该如何使用呢？本例在回答该问题的同时，详细讲解包的含义和使用方法。

【出现频率】 ★★★★

【关键考点】

- 包的含义；
- 包的使用方法。

【考题分析】

在现实中，有很多重名的情况，例如，全国各地有很多人都叫做张三。那么如何把它们区分开来呢？一个很容易想到的方法就是用地域进行区分，例如，中国，四川省，成都市，武侯区，XXX 街的张三，这样就明确了。其实，Java 的包也是起这样一种作用的手段。

程序员在定义类的时候，就可以为该类指定一个包名，也就告诉系统该类是属于 XXX 包的，例如下面的代码：

```
package abc;                                          //定义包名
public class MyClass{                                 //在该包下定义类
    ⋮
}
```

使用 package 关键字来指明该类是属于哪个包的，如果该包已经存在则无须创建，若该包还没创建，则创建一个新包。那么，包的名字是否可以像地名一样分级呢？这是肯定的。包名可以使用“.”号来分级，例如下面的代码：

```
package a.b.c
```

以上定义的包就叫做 a.b.c，它的上一级是 a.b，如果有下一级则继续使用“.”扩展。这样的级别也与文件系统的结构是吻合的，例如，以上的包名需要对应文件夹结构：“a/b/c”，类就放在 c 文件夹中。

说明： javac 编译命令在编译有包名的类的源文件的时候，是不会为它们创建文件夹结构的，开发人员需要手动的创建这些文件夹结构，或者使用 javac –d 选项。

当需要使用其他包的类、接口、异常等东西时，需要使用 import 关键字把它们都导入进来，才能通过编译，例如下面的代码：

```
import java.util.List;                                //导入 List 类
import java.util.ArrayList;                           //导入 ArrayList 类
public class Test{
    public static void main(){                        //主方法
        List list = new ArrayList();                  //使用它们
```

```
    }
}
```

如果需要导入一个包下的多个类，则可以使用*替代，进行模糊匹配所有的类、接口、异常等，例如下面的代码：

```
import java.util.*;                          //导入 java.util 包下的所有东西
public class Test{
    public static void main(){               //主方法
        List list = new ArrayList();         //使用它们
    }
}
```

【答案】

包是 Java 程序中关于命名空间的一种手段，它可以有效的解决类重名的问题。当需要把一个类定义在某个包下的时候，需要使用 package 关键字进行定义；当需要使用其他包下的类的时候，则需要使用 import 关键字进行导入。

面试题 029　说明 private、protected、public 和 default 的区别

为了更好的组织 Java 程序的结构，Java 提供了 4 种访问控制符，让代码更加的充满结构性和安全性。那么以上的 4 种访问控制符的具体含义和使用方法是什么呢？本例在回答该问题的同时，详细讲解访问控制的含义和区别。

【出现频率】 ★★★★

【关键考点】

- 包的含义；
- 访问控制；
- private 关键字的含义；
- public 关键字的含义；
- protected 关键字的含义。

【考题分析】

该题目提到的 4 个访问控制符中，除了 default 以外，其他都是 Java 语言的关键字。default 代表的是对类成员没有进行修饰的情况，它本身也代表了一种访问控制符。对于它们 4 种访问控制符来说，它们都可以修饰类的成员（包括静态和非静态成员），这些修饰也就控制了成员能被其他地方访问的限制情况。

对于范围概念来说，Java 指的范围包括类内部、所在包下、子父类之间和外部包 4 种情况。如果一个成员需要被外部包所访问，则必须使用 public 修饰符；如果一个成员需要被定义在不同包下的子类所访问，则可以使用 public 或 protected 修饰符；如果一个成员需要被本包下的其他类所访问，则可以不用写任何的修饰符，使用 public 或 protected 也行；若一个成员想使用同类中其他成员，则使用任意一个修饰符即可；若一个成员不想被任何一个外部的类所访问，则使用 private 关键字比较恰当。下面对这几种修饰符作详细介绍。

（1）对于 public 修饰符，它具有最大的访问权限，可以访问任何一个在 CLASSPATH 下的类、接口、异常等。它往往用于对外的情况，也就是对象或类对外的一种接口的形式。

（2）对于 protected 修饰符，它主要的作用就是用来保护子类的。它的含义在于子类可以使用它修饰的成员，其他的不可以，它相当于传递给子类的一种继承的东西。

（3）对于 default 来说，有的时候也称为 friendly（友员），它是针对本包访问而设计的，任何处于本包下的类、接口、异常等，都可以互相访问，即使是父类没有用 protected 修饰的成员也可以。

（4）对于 private 来说，它的访问权限仅限于类的内部，是一种封装的体现，例如，大多数的成员变量都是修饰为 private 的，它们不希望被其他任何外部的类访问。

表 3.2 展示了 Java 访问控制符的含义和适用情况。

表 3.2　Java访问控制

	类　内　部	本　　包	子　　类	外　部　包
public	√	√	√	√
protected	√	√	√	X
default	√	√	X	X
private	√	X	X	X

注意：Java 的访问控制是停留在编译层的，也就是它不会再 class 文件中留下任何的痕迹，只在编译的时候进行访问控制的检查。其实，通过反射的手段，是可以访问任何包下任何类中的成员的，例如，访问类的私有成员也是可能的。

【答案】

它们都是访问控制符，它们的区别如下：

（1）public：可以被所有其他类访问。

（2）private：只能被自身访问和修改。

（3）protected：自身，子类及同一个包中类可以访问。

（4）default（默认）：同一个包中的类可以访问，声明时没有加修饰符，认为是 friendly。

3.4　小　　结

本章讲解了一些 Java 语法基础的面试题，涵盖了变量、类、对象、引用、操作符、接口、内部类和包等 Java 语言中的重要概念。尽管这些面试题考察的都是 Java 的基础知识，但是都有一定的深度，如果没有认真的品味过它们的原理或运行机制，是很难讲明白的，尤其是第 3.2 小节关于对象和类的概念。读者在回答这些面试题的时候，最好是结合一些实际的例子来讲解，可以把原理或道理说得更清楚一些，毕竟例子使用的就是这些底层的原理。

第 4 章　数据类型及类型转换

数据是程序操作的对象，也是程序的核心，失去数据的程序是没有任何意义的。Java 的数据类型可以分两大类：基础数据类型和引用数据类型。基础数据类型包括：byte、short、int、long、float、double、boolean 和 char 8 种。引用类型则是操作对象的一种句柄类型的数据。但是，引用类型所操作的对象也是以基础数据类型为基础的，因此，对基础数据类型的理解也就是 Java 学习的基本了。本章将包含关于 Java 数据类型基础的一些问题，并且分析这些题目和知识点，帮助读者梳理这些方面的知识。

4.1　整 型 数 据

整型是最简单最常用的数据类型，它们在计算机里直接按照二进制定义就可以了，不需要做额外的处理。在实际开发中，整型数据是用得最多的一种数据类型，如 for 循环的次数。

面试题 030　int 和 Integer 有什么区别

int 和 Integer 都用来表示整型数据，但是它们在内存中的存储方式和使用方式都有很大的差异，一定不可以将它们混为一谈。理解它们的区别，重点在于理解基础数据类型及其包装类之间的关系。本例在回答该问题的同时，全面介绍 Java 基础数据类型及其包装类的相关知识。

【出现频率】 ★★★★

【关键考点】

- int 及其他基础数据类型的存储方式及其含义；
- Integer 及其他包装类的概念；
- 基础数据类型和引用数据类型的区别。

【考题分析】

int 是 8 种基础数据类型中的一种，它的字节长度为 4，用于表示现实中的正负整型数字。由于它属于基础数据类型，它的创建不会在堆内存中开辟空间，一般是保存在栈内存中的，它可以用算术运算符进行加、减、乘、除等操作。在参数传递的时候，直接传递它的值。

说明： 所有的整型字面量默认为 int 型。

Integer 是 int 的包装类，在 java.lang 包下，它不属于基础数据类型，而是一个类。它的含义也就是表示一个整型的数字，但是，它不能被算术运算符进行加、减、乘、除等操作，只能用另外的方法进行算术运算或转换成 int 再进行运算。在参数传递的时候，传递的是它所代表的对象的一个引用。

int 和 Integer 是可以相互转换的。转换后的值也就是它们本身所代表的整型数据，示例代码如下：

```
public class IntTest {
    public static void main(String[] args) {
        int a = 10;                          //定义 int 型的 a
        Integer b = new Integer(20);         //用 new 创建一个 Integer 对象
        Integer c = Integer.valueOf(30);     //用 valueOf()方法创建一个 Integer
                                               对象
        System.out.println(++a);             //a 用于“++”运算
        System.out.println(b.intValue());    //用 intValue()方法得到 int 型数据
    }
}
```

对于 int 及其他基础数据类型，通过 new 包装类或静态的 valueOf()方法转换成包装类型，该过程叫做装箱。通过包装类的 intValue()方法把包装类型转换成基础类型，该过程叫做拆箱。int 和 Integer 分属于不同的类型，适用于不同的情况，可以在需要的时候，通过该方法进行相应的类型转换。

注意：Java 5.0 及其以后的版本，提供了一种叫做自动装箱和拆箱的功能，也就是基础数据类型及其包装类的自动转换功能，系统会根据需要进行自动的类型转换。

【答案】

int 是基础数据类型，字节长度为 4，它的创建不会在堆内存中开辟空间，一般保存在栈内存里，可以用算术运算符进行加减乘除等操作。在参数传递的时候，直接传递它的值。

Integer 是 int 的包装类，而本质是一个类，它的创建会在堆内存中开辟一块新的空间。它的含义也是表示一个整型的数字，但是，算术运算符不能操作它。在参数传递的时候，传递的是它所代表的对象的一个引用。

面试题 031　int 的取值范围

int 型数据的长度决定了它的取值范围，也决定了 int 适合哪些类型的数据表示。理解 int 的存储原理是本题目的真正意图。本例在回答该问题的同时，全面介绍 Java 的 int 型的存储原理。

【出现频率】 ★★★★

【关键考点】

- int 型数据的存储原理

【考题分析】

int 型数据的字节长度为 4，一共是 32 位。第一位用于表示正负号，其他位的数字组成它所代表的值，例如，7 的二进制表示为：

```
00000000 00000000 00000000 00000111
```

当表示负数的时候，则采用补码的原则，第一位总是为 1。例如，–7 的二进制表示为：

```
11111111 11111111 11111111 11111001
```

在计算 int 型数据的最大值的时候，也就是除第一位为 0 以外，其他位都是用 1 表示的数字就是 int 型的最大值，也就是 $2^{31}-1$。同理，负数的最大值则是第一位和最后一位为 1，其他位为 0，也就是-2^{31}。

【答案】

int 型数据采用 4 个字节进行存储，一共是 32 位。第一位用于表示正负号，因此，它的取值范围就是：$-2^{31}\sim 2^{31}-1$。

面试题 032　如何用八进制和十六进制来表示整型数据

尽管八进制和十六进制在实际开发中，使用没有十进制那么多，但是它们在有的时候却可以发挥出十进制无法达到的作用。例如，用八进制的数字来表示掩码，用十六进制的数字表示颜色。本例在回答该问题的同时，将演示如何使用 Java 的八进制和十六进制。

【出现频率】 ★★★★

【关键考点】

- 八进制数据的含义；
- 十六进制数据的含义；
- Java 如何用八进制和十六进制来表示整型数据。

【考题分析】

八进制的含义在于每位数字的进位大小为 8，它只有 0～8 这样的 9 个数字。十六进制则表示进位大小为 16，除了 0～9 的 10 个数字一位，还有 a、b、c、d、e、f 表示 10、11、12、13、14 和 15。

首先，来看以下一段 Java 代码：

```
public class JinZhiTest {
    public static void main(String[] args) {
        int a = 0123;                    //定义一个八进制的整型数字
        int b = 0x123;                   //定义一个十六进制的整型数字
        System.out.println(a);
        System.out.println(b);
    }
}
```

如果读者认为以上代码的执行结果是 123 123 的话，那么就错了。其实，对于变量 a 来说，它用八进制的方式来表示一个整型数据，仅仅需要在数字前面加多一个 0 即可。而变量 b 则是十六进制，它的写法则是在数字前面加上 0x。因此，以上代码的执行结果应该是：

```
83
291
```

说明： 默认情况下，int 型的字面量都是十进制的。

【答案】

Java 中的八进制字面量采用 0 开头，十六进制采用 0x 开头。

面试题 033　long 的取值范围

long 是比 int 的长度更长的一类整型，它往往用于存储大数据。与 int 类似，理解 long 的存储原理以后才能把它灵活运用。本例在回答该问题的同时，全面介绍 Java 的 long 型的存储原理。

【出现频率】 ★★★★

【关键考点】

- long 型数据的存储原理

【考题分析】

long 型数据的字节长度为 8，一共是 64 位。第 1 位用于表示正负号，其他位的数字组成它所代表的值，例如，7 的二进制表示为：

```
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000111
```

当表示负数的时候，则采用补码的原则，第 1 位总是为 1。例如，–7 的二进制表示为：

```
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111001
```

在计算 long 型数据的最大值的时候，也就是除开第 1 位为 0 以外，其他位都是用 1 表示的数字就是 long 型的最大值，也就是 $2^{63}-1$。同理，负数的最大值则是第一位和最后一位为 1，其他位为 0，也就是-2^{31}。

在默认情况下，所有整型字面量都是 int 型的，如果需要指定数字为 long 型，则需要在该数字的后面加上 l 或 L，以表示该变量为 long 型，例如下面的代码：

```
123L
或
123l
```

【答案】

long 型数据采用 8 个字节进行存储，一共是 64 位。第 1 位用于表示正负号，因此，它的取值范围是$-2^{63}\sim 2^{63}-1$。

4.2　实 型 数 据

实型（或称为浮点型），是用来表示带小数部分的实数数据。Java 的实型数据由整数和小数两个部分组成，根据大小和精度的不同，可以用 float 和 double 两种类型来表示实数。本小节将介绍一些关于 Java 实型的面试题。

面试题 034　float 和 double 的取值范围各是多少

为了适应实数的取值范围和精度的不同，Java 提供了 float 和 double 两种实型，各有

用途。理解实型的存储原理是使用它们的基础。本例在回答该问题的同时，详细讲解 Java 的实型的存储原理。

【出现频率】 ★★★★

【关键考点】

- float 的长度和精度；
- double 的长度和精度。

【考题分析】

float 型，内存分配 4 个字节，占 32 位，范围从 $3.4E+10^{-38}$～$3.4E+10^{38}$ 次方，例如下面的代码：

```
float x=123.456F;
float y=2e20f;
```

注意：float 型定义的数据末尾必须有"f"或"F"，为了和 double 区别。而 double 末尾可以有"d"也可以不写。

double 型，内存分配 8 个字节，范围从 $1.7E+10^{-308}$～$1.7E+10^{308}$，例如下面的代码。

```
double x=1234567.98;
double y=8980.09d;
```

在实际的使用中，double 使用得更多一些，尽管大多数时候，并不需要使用那么多位的字节长度。另外，对于字面量来说，Java 的浮点型数据默认为 double 型，因此，在需要特定使用 float 的地方需要注意一下，代码如下。

```
float i = 2.3;
```

以上代码不同通过编译，因为 2.3 默认是 double 型，不可以自动的转换为低精度的 float，需要显式的进行类型转换，或为字面量后面加上 f 或 F。例如：

```
float i = 2.3F;
float i = (float)2.3;
```

说明：在 double 向 float 转换的过程中，很有可能数据的精度就损失了，还有可能会损失部分数据。但是，float 向 double 转换，则不会有任何的损失。

【答案】

（1）float 型，范围从 $3.4E+10^{-38}$～$3.4E+10^{38}$。

（2）double 型，范围从 $1.7E+10^{-308}$～$1.7E+10^{308}$。

面试题 035　实型与整型的相互转换

因为实型比整型要多出小数部分，从整型转为实型时不会有精度丢失，但是由实型转为整型时则会丢失小数部分的数据。本例在回答该问题的同时，详细介绍关于实型与整型相互转换的知识。

【出现频率】 ★★★★

【关键考点】

- 实型数据的存储方式；
- 如何显式的把实型转换为整型；
- 何时存在隐式的类型转换。

【考题分析】

显式转换，也就是强制类型转换，例如下面的代码：

```
int i = 123;
double d = (double)i;
int k = (int)d;
```

在以上的代码中，首先将 int 型的变量 i 转换成 double 型，再赋给变量 d，然后又把 double 型的变量 d 转换成 int 型并赋给 int 型的变量 k。因为最初变量 i 的值为整数，因此在整个转换过程中不会存在数据的丢失。但是，如果变量 d 带有小数部分，那么变量 k 则将丢失小数部分的数据，例如下面的代码：

```
double d = 123.45;
int k = (int)d;
```

此时，变量 k 的值就是 123 了，小数部分的数据在转换过程中就丢失了。另外，有些转换是隐式的，代码中并没有显式的强制类型转换的代码，例如下面的代码：

```
double d2 = 12;
System.out.println(d2 + 123);
```

以上代码的两个语句都存在隐式的类型转化。第一句，字面量 12 的类型为 int，但是它却赋给了 double 型的 d2，其中就存在隐式的 int 到 double 的转换过程；第二句，double 型的 d2 变量和 int 型的字面量 123 相加，字面量 123 在相加之前也有一个隐式地转换成 double 型的过程。

注意：在算术运算过程中，类型的转换遵守一个原则，就是由低位向高位转换，以变量中类型字节长度最大的为转换目标。

【答案】

整型和实型可以进行相互装换，其中整型转为实型时，不存在精度的损失，而实型转为整型时则可能会有精度的损失。转换过程可分为显式和隐式两种方式，前者是通过强制类型转换来进行，后者是在运算、传参等情况下，由系统自动进行。

面试题 036　如何用 BigDecimal 类进行精确运算

Java 的简单浮点类型 float 和 double 在进行精确的算术运算的时候，例如，进行货币金额的计算，往往是不能得到准确的结果的。不仅是 Java，其他一些编程语言也存在这样的问题，但是每种语言都有自己的解决办法，Java 也不例外。本例在回答该问题的同时，详细讲解关于 BigDecimal 类的使用规则。

【出现频率】 ★★★★★

【关键考点】

- BigDecimal 的使用规则；
- 浮点类型和 BigDecimal 之间的相互转换。

【考题分析】

首先来看看以下的代码会出现什么样的问题。

```
public class BigDecimalTest {
    public static void main(String[] args) {
        System.out.println(0.05+0.01);
        System.out.println(1.0-0.42);
        System.out.println(4.015*100);
        System.out.println(123.3/100);
    }
}
```

运行结果如下：

```
0.060000000000000005
0.5800000000000001
401.49999999999994
1.2329999999999999
```

以上的结果肯定不会是开发者想要的，问题在哪里呢？原则上来讲，Java 的 float 和 double 只能用来进行科学计算或工程计算，在大多数的商业计算中，需要使用 java.math.BigDecimal 类。

使用 BigDecimal 类来进行计算的时候，主要分为以下步骤：

（1）用 float 或 double 变量构建 BigDecimal 对象。

（2）通过调用 BigDecimal 的加、减、乘、除等相应的方法进行算术运算。

（3）把 BigDecimal 对象转换成 float、double、int 等类型。

一般来说，可以用 BigDecimal 的构造方法或静态的 valueOf()方法把基本类型的变量或其字符串构建成 BigDecimal 对象，例如下面的代码：

```
BigDecimal v1 = new BigDecimal(Double.toString(0.05));
BigDecimal v2 = BigDecimal.valueOf(0.01);
```

对于常用的加、减、乘、除运算，BigDecimal 提供了相应的成员方法。它们都是把运算的操作对象作为参数传入，与自身进行相应的计算，并且返回计算后的值，代码如下：

```
public BigDecimal add(BigDecimal augend)              //加法
public BigDecimal subtract (BigDecimal augend)        //减法
public BigDecimal multiply(BigDecimal augend)         //乘法
public BigDecimal divide(BigDecimal augend)           //除法
```

进行相应的计算以后，可能会需要将 BigDecimal 对象转换成相应的基本数据类型的变量，就可以调用 floatValue()、doubleValue()等方法。以下是一段使用 BigDecimal 进行运算的示例代码。

```
public class BigDecimalTest {
    public static void main(String[] args) {
        System.out.println(add(0.05,0.01));
    }
    //提供精确加法计算的 add 方法
```

```
    public static double add(double v1, double v2) {
        BigDecimal b1 = new BigDecimal(Double.toString(v1));    //构建变量1
        BigDecimal b2 = new BigDecimal(Double.toString(v2));    //构建变量2
        return b1.add(b2).doubleValue();                        //返回计算好以后的值
    }
}
```

说明：读者如果有兴趣，可以参考以上的示例代码，写一个工具类，把加、减、乘、除 4 个常用的运算写成工具方法，以备开发中使用。

【答案】

使用 BigDecimal 类来进行精确的算术计算，也就是使用它所提供的一系列的 API，主要分为以下步骤。

（1）用 float 或 double 变量构建 BigDecimal 对象。通过构造方法或 valueOf()方法。

（2）通过调用 BigDecimal 的加、减、乘、除等，相应的方法进行算术运算。

（3）把 BigDecimal 对象转换成 float、double、int 等类型，通过类似 xxxValue()等方法进行转换。

4.3　布尔型数据

任何一种计算机语言都会有布尔类型，但是它们对布尔型数据的处理是不一样的。Java 对待布尔型数据比 C 语言要严格得多，这样尽管失去了一些灵活性，但是却保证了程序的安全。本节介绍关于 Java 布尔型的面试题。

面试题 037　Java 可以用非 0 来代表 true 吗

对于 C/C++程序员来说，可能会有“0 代表 false，非 0 代表 true”的习惯。这样的编码习惯可以让程序变得高效，但是也带来了一定的危险性。Java 处于安全考虑，限制了这种做法，只能用布尔型的 true 和 false 来代表真假。本例在回答该问题的同时，详细讲解关于 Java 布尔型数据的使用。

【出现频率】 ★★★★

【关键考点】

- 布尔型的语法规则；
- 布尔型数据的用法。

【考题分析】

Java 是一种强类型的语言，它对条件表达式有非常严格的规定，只能使用 boolean 型的数据进行条件判断。如果使用整型的非 0 数进行条件判断，则体现为语法错误，例如，以下代码就不能通过编译。

```
if(100){                        //int 型的值作为条件，是错误的
    ⋮
}
```

对于习惯使用这样规则的读者（例如，常年使用 C 语言的程序员）一定要注意了，if、for 这样的语句中，只能使用 false 或 true 两种值，例如下面的代码：

```
if(100 > 90){                    //条件表达式的值作为条件
    ⋮
}
```

其实，这在一定程度上也是为了保证程序的安全，让开发者明确应该如何来判断条件。如果使用非 0 的条件，有可能运行时的值并不是开发者需要的，如变量还未初始化。另外，Java 对于 boolean 型的变量，也需要在使用前进行初始化操作。例如，以下代码则不能通过编译：

```
boolean a;
if(a){                           //报出编译错误
    ⋮
}
```

【答案】

Java 不能用非 0 代表 true，也不能用 0 代表 false，只能用布尔型的 true 和 false 来进行条件判断。

面试题 038　boolean 和它的包装类的区别在哪里

Java 所有的基础数据类型都有对应的包装类，boolean 也不例外。这些包装类相对于它们的基本类型来说，除了具有相同的值以外，还多出了一种值，那就是 null。对于 boolean 来说，如果不注意这种情况的话，就很可能抛出空指针异常了。本例在回答该问题的同时，详细讲解关于 boolean 和 Boolean 的区别。

【出现频率】 ★★★★

【关键考点】

- 包装类的概念；
- 基础数据类型和引用数据类型的区别。

【考题分析】

boolean 是基础数据类型，而 Boolean 是一个类，这是它们最本质的区别。Java 之所以为每一个基础数据类型都提供一个相应包装类的目的，在于将 Java 中的所有东西都抽象成对象，可以更方便控制和使用。

从 Java 5.0 开始，提供了自动装箱和拆箱的功能，也就是系统根据需要自动的进行类型转换。对于 boolean 型数据来说，经常会用到 if、while 等条件语句，如果把 Boolean 用在这些语句时候，需要注意该对象是否为 null。因此，此时的布尔变量会存在 3 种可能的值，即 true、false 和 null。例如，以下代码不会有编译错误，但是在运行的过程中会抛出 NullPointerException。

```
public static void main(String[] args) {
    Boolean a = null;
    if(a){                              //用 Boolean 型的变量进行条件判断
        ⋮
    }
}
```

说明：对于这样的情况，读者应该养成一个对变量进行初始化的习惯。Boolean 变量一般初始化为 false。

【答案】

- boolean 是基础数据类型，Boolean 是类。
- boolean 一般存于栈空间中，Boolean 对象存方在堆空间中。
- boolean 有 true 和 false 两种值，Boolean 除了 true 和 false 以外，还有 null。
- Java5.0 以前，不能把 Boolean 用于条件语句，需要转换为 boolean 以后再使用。

4.4　字符型数据

Java 向来都和国际化概念是分不开的，它从诞生那一刻开始不得不处理各种各样的字符。Java 处理字符数据的能力非常强大，这和它在字符型数据的存储上的良好设计是有关的。本节将介绍一些关于 Java 字符型的面试题。

面试题 039　char 的取值范围

Java 的字符数据需要涵盖世界上大多数的语言字符，因此，Java 在表示字符数据的时候，不能像 C 语言那样，用简单的一个字节来编码字符。本例在回答该问题的同时，详细地讲解关于 Java 的 char 类型。

【出现频率】 ★★★

【关键考点】

- char 型的存储原理；
- char 型数据的使用。

【考题分析】

Java 中的 char 采用 Unicode 编码格式，用 2 个字节来表示一个字符，也就是说所有 char 型数据的字节长度都是 2，一共 16 位（bit），即使有些字符并不必用那么多位。那么 char 的取值范围就可以知道了，就是 0 到 2 的 16 次方减 1。

说明：尽管 char 可以表示的数值有那么多，但事实上并非每个数值都对应一个字符。

char 可以和 int、short、long 整型进行类型的相互转换。这是因为它们本质上都是以二进制整型的形式进行存储的，但是在转换的过程中需要注意长度不同所造成的数据丢失，示例如下：

```
public class CharTest {
    public static void main(String[] args) {
        char c = 'a';                    //定义 char 型的 a 变量
        short s = (short) c;             //把 a 转换成 short 型的 s 变量
        System.out.println((char)s);     //再把 short 转换成 char 进行打印
    }
}
```

以上代码中，首先用单引号“’”定义一个 char 型的变量 a，然后把它转换成 short 型的变量 s，最后再把 s 转换成 char 型并打印出来，得到的结果如下：

```
a
```

在整个转换过程中，并没有数据的丢失，因为 short 和 char 的字长是相同的，都是两个字节。

【答案】

Java 中的 char 采用 Unicode 编码格式，用两个字节来表示一个字符，一共 16 位（bit），它所能表示的最大字符数量为 2^{16}。

面试题 040　char 能否存储汉字

char 可以用来表示世界上大多数国家的大多数字符，当然也包括汉字在内。在国内的 Java 开发中，肯定会接触到中文字符的处理，理解 char 的存储原理对于中文乱码等问题的处理是很重要的。本例在回答该问题的同时，详细地讲解关于 Java 的 char 类型对中文字符的处理。

【出现频率】 ★★★

【关键考点】

- char 的存储原理；
- 汉字的编码。

【考题分析】

Java 的 char 型字符采用的是 Unicode 编码，在这些编码中就包含有简体中文和繁体中文所需要的字符。因此，char 是可以用来表示汉字的代码如下。

```
public class ChineseTest {
    public static void main(String[] args) {
        char c = '中';                    //定义一个“中”字符
        int i = c;                        //把 c 变量转换为 int 型
        System.out.println(i);            //打印出'中'所对应的整型值为多少
    }
}
```

以上代码的运行结果如下：

```
20013
```

通过以上的示例代码可以看出，字符“中”在 Java 中的编码值为 20013，其他的中文字符也就与之类似了。

说明： 对于英文字母，它们所处的范围是 0～127 之间，与 ASCII 码相同，这是因为 Unicode 兼容 ASCII 码。

【答案】

char 是可以存储汉字的，每个中文字符都有对应的 Unicode 编码。

面试题 041　如何使用转义字符

一些特殊的字符不能用一个特定的数字、字母或符号表示，例如，换行符、制表符、双引号等。为了表示这些特殊的字符，需要引入转义字符的概念，也就是用其他的一些字符来代表这些字符。那么，Java 的转义字符又是如何使用的呢？本例在回答该问题的同时，详细地讲解关于 Java 的转义字符的用法。

【出现频率】 ★★★

【关键考点】

- char 的存储原理；
- 转义字符的概念和使用。

【考题分析】

所有的计算机语言都无法回避转义字符的概念，它们都统一的用了反斜线来表示转义，例如下面的代码：

```
System.out.println("\t");
```

以上的代码打印出了一个制表符，以反斜线“\”开头，后边紧跟‘t’字母，它就代表了制表符。系统在解析这些字符的时候，一旦遇到“\”符号，就会把它后边紧跟的字符一起分析，看它们组合成了另外哪一个字符。以下为一些常见的转义字符。

- \a：响铃（BEL）007
- \b：退格（BS）008
- \f：换页（FF）012
- \n：换行（LF）010
- \r：回车（CR）013
- \t：水平制表（HT）009
- \v：垂直制表（VT）011
- \\：反斜杠 092
- \?：问号字符 063
- \'：单引号字符 039
- \"：双引号字符 034
- \0：空字符（NULL）000
- \ddd：任意字符 三位八进制
- \xhh：任意字符 二位十六进制

Java 的转义字符应用比较广泛。例如，表示 Windows 操作系统下的文件路径时候，就需要用两个斜线来表示一个斜线，例如下面的代码：

```
String path = "c:\\windows\\hello"
```

【答案】

在 Java 中，使用反斜线“\”来代表转义，它与紧跟在它后面的字符一起组成了转义字符，从而达到转义的目的。

4.5　String 型数据

字符串，也就是一段字符数据，Java 通过 java.lang.String 类来抽象所有的字符串对象。因为字符串使用频繁，所以它相对于其他普通类来说，有它的特殊性，例如，直接通过双引号就可以创建对象、字符串可以用“+”操作符、String 类不能被继承等。有时候，它又体现出基本数据类型的一些特点，因此 String 的掌握是既困难又很重要，关于它的面试题也是屡见不鲜。

面试题 042　字符串字面量是否自动生成一个 String 对象

在开发过程中，经常会需要直接写出一个字符串变量，也就是双引号包括的字符数据。那么，这样一种字面量写法的字符串，是否已经创建了一个 String 对象呢？如果没有，那么数据存放在哪里呢？如果有，并没有 new 语句啊，是怎么创建的呢？本例在回答该问题的同时，详细地讲解关于 Java 的 String 对象的概念。

【出现频率】 ★★★★★

【关键考点】

- 双引号操作符的作用；
- String 对象的概念。

【考题分析】

首先看下面的代码：

```
public class StringTest {
    public static void main(String[] args) {
        String str1 = "abc";                    //创建 abc 字符串对象
        String str2 = new String("abc");        //用 new 语句 创建 abc 字符串对象

    }
}
```

以上代码中，用两种方式创建了字符串变量 str1 和 str2。它们有什么区别呢？其实，Java 虚拟机（JVM）在执行这段代码的时候，遇到双引号操作符，它会自动创建一个 String 对象。该 String 对象所代表的值就是 abc，然后再返回该对象的一个引用。

对于 str1 字符串来说，它的创建过程同上所述。在 Java 5.0 及其以后的版本，在创建该对象之前，JVM 会在 String 对象池中去搜索该字符串对象是否已经被创建，如果已经被创建，则直接返回一个引用，否则先创建再返回引用。

而 str2 字符串变量，它的创建过程就要多一个步骤。除了类似于 str1 字符串对象创建过程以外，它还会额外的创建一个新的 String 对象，也就是 new 关键字的作用，并且返回一个引用给 str2。

尽管 str1 和 str2 的值是相同的，但是它们的引用是不同的，也就是说以下代码的返回值为 false。

```
str1 == str2;                                   //结果为 false
```

说明：若使用 equals()方法，则返回 true，因为 equals()方法比较是它们真正的值，而不仅仅是引用。

【答案】

答案是肯定的。字符串类具有一定的特殊性，JVM 在执行双引号操作符的时候，会自动的创建一个 String 对象，并返回这个对象的引用。

面试题 043　字符串对象池的作用是什么

从 Java 5.0 开始，Java 虚拟机在启动的时候会实例化 9 个对象池。这 9 个对象池分别用来存储 8 种基本类型的包装类对象和 String 对象，主要是为了效率问题。理解对象池的概念和字符串对象池的作用是很重要的，因为只有这样才能真正的灵活使用字符串数据。本例在回答该问题的同时，详细地讲解关于 Java 的 String 对象池的概念。

【出现频率】 ★★★★★

【关键考点】

- 对象池的概念和作用

【考题分析】

大家应该知道，创建 String 对象有两种办法，代码如下：

```
String str1="hello";
String str2=new String("hello");
```

这两种创建 String 对象的方法有什么区别吗？区别就在于第 1 种方法在对象池中取对象，第 2 种方法直接生成新的对象。

当在程序中直接用双引号引起来一个字符串时，JVM 就到 String 的对象池中去检查是否有一个值相同的对象。如果有，就取现成的对象；如果没有，则在对象池中创建，并返回其引用。因此，可以发现下面的代码输出 true：

```
String str1="hello";
String str2="hello";
System.out.println(str1==str2);
```

这说明 str1 和 str2 指向同一个对象，因为它们都是在对象池中取到的，而下面的代码输出为 false：

```
String str3="hello";
String str4=new String("hello");
System.out.println(str3==str4);
```

因为在任何情况下，只要 new 一个 String 对象都是创建了新的对象。其他的基本类型包装类的对象池与此类似，工作原理是一样的。对象池的存在是为了避免频繁的创建和销毁对象而影响系统性能，那么自定义的类是否也可以使用对象池呢？当然可以，如下代码所示：

```
//定义狗类
class Dog {
    private String name;              //名字
    private int age;                  //年龄
```

```
    private static HashSet<Dog> pool = new HashSet<Dog>();
                                        //用于对象的对象池
    public Dog(String name, int age) {
        this.name = name;
        this.age = age;
    }
    //使用对象池来得到对象的方法
    public static Dog newInstance(String name, int age) {
        //循环遍历对象池
        for (Dog dog : pool) {
            if (dog.name.equals(name) && dog.age == age) {
                return dog;
            }
        }
        //如果找不到值相同的 Dog 对象，则创建一个 Dog 对象
        //并把它加到对象池中然后返回该对象。
        Dog dog = new Dog(name, age);
        pool.add(dog);
        return dog;
    }
}
public class StringPoolTest {
    public static void main(String[] args) {
        //对象池中取
        Dog dog1 = Dog.newInstance("lele", 30);
        Dog dog2 = Dog.newInstance("lele", 30);
        //所以 dog1==dog2
        //重新创建，所以 dog1!=dog3
        Dog dog3 = new Dog("lele", 30);
        System.out.println(dog1 == dog2);
        System.out.println(dog1 == dog3);
    }
}
```

以上代码中，创建了一个 Dog 类，它包含 name 和 age 两个属性，并且还有一个静态的 pool 对象，是用于存储对象的池。另外，还有一个静态的 newInstance()方法，它会为创建 Dog 对象提供对象池的支持。

在 main()方法中，创建了 3 个对象，dog1、dog2 和 dog3。因为 dog1 和 dog2 是从对象池中取的对象，因此它们的引用是相同的，而 dog3 是另外创建，所以它和 dog1 的引用是不同的。代码的最终运行结果如下：

```
true
false
```

说明：以上的 Dog 类示例仅仅是为了展示对象池的机制和原理。String 的对象池并不完全与 Dog 的相同。

【答案】

对象池的存在是为了避免频繁的创建和销毁对象而影响系统性能。当 JVM 在运行用双引号引起来一个字符串的代码时，会到 String 的对象池中去检查是否有一个字符序列相同的对象。如果有，就取现成的对象，如果没有，则在对象池中创建一个对象，并返回。

面试题 044　StringBuffer 和 StringBuilder 存在的作用是什么

Java 的字符串 String 对象，有一个特性，就是不变性，它只能被创建，而不能改变其中的值。因此，一些大量使用字符串的程序可能会出现性能瓶颈，甚至内存溢出。针对这样的问题，Java 为开发者提供了相应的解决方案。本例在回答该问题的同时，详细地讲解关于 StringBuffer 和 StringBuilder 的用法。

【出现频率】 ★★★★★

【关键考点】

- String 的不变性；
- StringBuffer 和 StrngBuilder 的用法；
- StringBuffer 和 StringBuilder 的区别。

【考题分析】

先看一段关于字符串程序的代码：

```
public class StringBBTest {
    public static void main(String[] args) {
        String a = "a";                          //定义字符串变量 a
        String b = "b";                          //定义字符串变量 b
        String c = "c";                          //定义字符串变量 c
        String d = "d";                          //定义字符串变量 d
        String abcd = a + b + c + d;             //字符串变量 abcd 等于它们相加
    }
}
```

以上代码中，一共创建了 7 个 String 对象。对于 a、b、c、d 变量，它们都是通过双引号的形式来创建的 String 对象。而 abcd 变量的值则是通过它们 4 者相连得到，在相连的过程中，首先执行的 a+b 操作，产生了 ab 字符串，然后再加上 c，又产生了 abc 字符串，最后加上 d 才得到 abcd 的 String 对象。

由以上的示例可以看出，通过 String 直接相加来拼接字符串的效率是很低的，其中可能会产生多余的 String 对象，例如，ab 和 abc。如果程序中需要拼接的字符串数量成千上万的话，那么 JVM 的负荷是非常大的，严重的影响到程序的性能。其实，如果遇到有大量字符串需要拼接的话，应该使用 StringBuffer 和 StringBuilder 类，它们是对 String 的一种补充。例如，同样的功能，可以这样来写代码以提高程序性能，示例如下：

```
public class StringBBTest {
    public static void main(String[] args) {
        String a = "a";                          //定义字符串变量 a
        String b = "b";                          //定义字符串变量 b
        String c = "c";                          //定义字符串变量 c
        String d = "d";                          //定义字符串变量 d
        StringBuffer sb = new StringBuffer();    //创建 StringBuffer 对象
        sb.append(a);                            //用 append 方法追加字符串
        sb.append(b);
        sb.append(c);
        sb.append(d);
        String abcd = sb.toString();             //用 toString 方法得到 sb 的值
        System.out.println(abcd);
```

```
    }
}
```

上例是通过 StringBuffer 来拼接字符串的，它无法保证线程的安全。如果在拼接字符串的过程中可能会涉及到线程安全的问题，则应该使用 StringBuilder，它们两者的功能和 API 是类似的。

【答案】

在 Java 程序中，如果有大量拼接字符串的需要的话，应该使用 StringBuffer 和 StringBuilder 类，它们可以避免不必要的 String 对象的产生，以提高程序的性能。它们两者的作用类似，只不过 StringBuilder 线程是安全的。

面试题 045　如何输出反转过后的字符串

字符串是一串包含一定序列的字符数据。Java 的 String 类型的数据是具有不变性的，那么如何得到某一个指定字符串的反序的字符串数据呢？本小节在回答该问题的同时，详细地讲解 String 存储数据的原理。

【出现频率】 ★★★★

【关键考点】

- String 的特性；
- 存储字符数据的原理；
- StringBuffer.reverse()方法的使用。

【考题分析】

最常用的方式就是反向取出每个位置的字符，然后依次将它们输出到控制台，或者把这些字符数据反转存储在一个 char 数组中，再按照数组顺序打印在控制台上。下面是一个编程示例。

```
package ch03;
public class ReverseStringTest {
    public static void main(String[] args) {
        //原始字符串
        String s = "A quick brown fox jumps over the lazy dog.";
                                                    //原始 String
        System.out.println("原始的字符串: " + s);

        System.out.print("反转后字符串: ");
        for (int i = s.length(); i > 0; i--) {
            System.out.print(s.charAt(i - 1));      //反循环打印
        }
        //也可以转换成数组后再反转，不过有点多此一举
        char[] data = s.toCharArray();              //得到 char 数组
        System.out.println();
        System.out.print("反转后字符串: ");
        for (int i = data.length; i > 0; i--) {
            System.out.print(data[i - 1]);          //一次打印
        }
    }
}
```

运行结果如下：

```
原始的字符串：A quick brown fox jumps over the lazy dog.
反转后字符串：.god yzal eht revo spmuj xof nworb kciuq A
反转后字符串：.god yzal eht revo spmuj xof nworb kciuq A
```

以上两种方法是最容易想到的，也是最常用的两种方法，但是 Java 是一种提供许多现成功能的语言，其实 StringBuffer 可以让它更容易实现，例如下面的代码：

```
package ch03;
public class ReverseStringTest2 {
    public static void main(String[] args) {
        //原始字符串
        String s = "A quick brown fox jumps over the lazy dog.";
                                                        //原始字符串
        System.out.println("原始的字符串：" + s);
        System.out.print("反转后字符串：");
        StringBuffer buff = new StringBuffer(s);        //得到 StringBuffer
        //java.lang.StringBuffer 类的 reverse()方法可以将字符串反转
        System.out.println(buff.reverse().toString());//调用 reverse()方法
    }
}
```

运行结果如下：

```
原始的字符串：A quick brown fox jumps over the lazy dog.
反转后字符串：.god yzal eht revo spmuj xof nworb kciuq A
```

说明： 一般来说，更推荐大家使用 StringBuffer 的 reverse()方法，因为一般来说 JDK 的实现算法要高明一些。

【答案】

实现字符串的反转有一般来说有两种手段，第一是利用字符串存储字符数据的原理，取出它的 char 数组，进行重新排列并保存；第二则是生成 StringBuffer 对象，直接使用 StringBuffer 的 reverse()方法。

面试题 046　如何使用指定的字符集创造 String 对象

一般来说，创建一个字符串对象的时候，String 是按照平台默认的字符串进行 String 对象的创建，那么如果有特殊字符集设定的需要时，又该如何处理呢？本例在回答该问题的同时，详细地讲解 String 存储数据的原理和字符集的概念。

【出现频率】 ★★★★

【关键考点】

- String 存储字符数据的原理；
- 字符集的概念；
- String 的带字符集参数的构造方法的使用。

【考题分析】

一般情况下，不论是创造字符串的字面量，还是使用 String 的构造方法创造字符串对

象，都不会指定特定的字符集，JVM 会自动的帮助开发者用平台默认的字符集进行构造。例如下面的代码：

```
String a = "中文";
String b = new String("中文");
```

对于大多数中文系统来说，平台的默认字符集都是 GBK 或 GB2312。当程序员用输入法打印出这些中文的时候，它的编码其实是按照“GBK”或“GB2312”进行的，但是在编译的时候，JVM 会把这些字符编码按照 Unicode 进行重新编码，然后保存在 class 文件中。

如果一个字符数据不是来自于平台手动的收入，而是通过其他的途径传入的，例如：Web 请求参数。那么，这些字符编码就可能不是平台默认的，那么应该如何处理呢？这里就需要使用 String 带有字符编码的构造方法了，如下所示：

```
String a = "中文";
String b = new String(a.getBytes(),"GBK");
String c = new String(a.getBytes(),"UTF-8");
```

以上代码的 b 和 c 字符串就是按照指定的字符集重新创建的字符串对象，如果 a 变量的“中文”两个字符是用 GBK 编码的话，那么 c 变量就会出现编码错误，打印出来则是一串问号。

说明：带字符集的 String 的构造方法往往是解决字符乱码的一种手段。

【答案】

使用带有字符集编码的 String 的构造方法就可以用指定的字符集来重新创造字符串对象了，该方法的参数包括两个：一个是 byte 数组；另一个则是字符集编码的字符串形式，例如，UTF-8、GBK、ISO-8859-1 等。

4.6　小　　结

本章讲解了一些 Java 数据类型和类型转换的面试题，包含了整型、实型、布尔、字符、字符串等类型，以及它们之间的转换问题。数据类型是所有计算机编程语言的基础，理解这些数据类型存储数据的原理对于日常的开发效率起到至关重要的作用。本章在讲解数据类型面试题的同时，也把它们的原理说清楚了。因为，面试题是千千万万的，但是原理才是它们的核心，是所有面试题考察的根本，读者应该把更多的心思放在这些原理上面。

第 5 章　数组和集合的使用

不论是基本类型的数据，还是引用类型的数据，都可以使用集合的形式对它们进行操作。在 Java 中，操作数据最常见的形式就是数组和集合类。Java 语言的数组是比较有特点的，它在 Java 中作为一个特殊类，用不同的方式操作基本数据类型和引用数据类型。另外，java.util 包里提供了一些集合类，包括链表、集合、Map 等，它们操作数据集比数组更方便，各有特点，存储和操作数据的方式各不相同，应该在不同的情况下使用不同的集合类。本章将包含关于 Java 数组和集合类的常见面试题，并且分析这些题目和知识点，帮助读者梳理这些方面的知识。

5.1　Java 的数组

Java 语言的数组相对于其他语言来说，是比较有特色的，它在 Java 中作为一个特殊类，操作基本数据类型和引用数据类型的方式各不相同。深刻理解 Java 数组的原理，有助于读者在编程过程中更好的使用数组来操作数据集。本节将集中讨论有关 Java 数组的常见面试题。

面试题 047　如何理解数组在 Java 中作为一个类

Java 数组的使用语法与 C 和 C++的比较类似，但它的本质是一个 Java 类，只是这个类比较特殊，所以它很容易被一些程序员所误解。本例在回答该问题的同时，全面地介绍 Java 数组的特性。

【出现频率】 ★★★★

【关键考点】

- 数组的使用语法；
- Java 数组的数据存储的本质。

【考题分析】

大家知道，Java 的数据类型可以分为两种：基本类型和引用类型。那么数组属于哪种类型呢？很显然，数组肯定不会是基本类型，因为基本类型就只有那 8 种，那它就应该属于引用类型了，也就是说每一个数组实例都是一个对象，这些对象同属于一个类。

说明： Java 的基本数据类型包括 byte、char、short、int、long、float、double、boolean，8 种，其他都是引用数据类型。

首先，看一看如何使用一个 Java 数组，大致有以下几个步骤。

（1）声明数组，使用一个引用来操作数组。

（2）使用 new 关键字来为数组创建所需的内存空间，并且返回一个引用的值，还可以用{}符号来为数组赋初始值。

（3）使用数组的下标对数组的值进行操作。

以上步骤是对数组进行操作的基本过程，示例代码如下：

```
package ch05;
public class ArrayClass {
    public static void main(String[] args) {    //main()方法
        int[] arr = new int[]{1,2,3,4,5};        //定义一个 int 数组变量
        String[] arr2 = new String[3];           //定义一个 String 数组变量

        for (int i : arr) {                      //循环打印 arr 的值
            System.out.println(i);
        }
        for (String i : arr2) {                  //循环打印 arr2 的值
            System.out.println(i);
        }
    }
}
```

以上的代码创建了两个数组，一个是 int 型的数组，对数据进行了初始化；另外一个是 String 型的数组，指定数组的长度为 3。然后把这两个数组的数据进行了遍历打印，可以得到如下结果：

```
1
2
3
4
5
null
null
null
```

两种类型的数组有一个共同的特点，就是在创建的时候必须指定一个长度。其实，这是因为这些数组在创建的时候，会动态的为这些类创建指定数目的成员变量，并且为这些成员变量赋初值。但是，Java 不会让程序员像普通类那样去操作这些成员变量，而是通过[]符号，使用下标来访问这些数据。另外，Java 还为每一个数组对象提供了一个 length 属性，用于知道数组的长度。

注意：数组的下标是从 0 开始的。

那么，数组的类名是什么呢？其实，数组的类型与它存储的数据类型有关，它们的类名可以通过获取数组的 Class 类型来获取，如下代码用于获取上例两个数组的类名：

```
System.out.println("int arr type name : " + arr.getClass().getName());
System.out.println("String arr type name : " + arr2.getClass().getName());
```

打印结果如下：

```
int arr type name : [I
String arr type name : [Ljava.lang.String;
```

可以看出，数组类名总是以左中括号开头，然后是不同的标示。例如，int 型的数组则是一个 I 字符，字符串类型的数组则是 Ljava.lang.String。通过这些标示，程序员就可以知道该数组存储的是哪种数据类型。

【答案】

Java 的数组本质上是一个类，该类还保存了数据类型的信息。该类通过成员变量的形式来保存数据，并且通过[]符号，使用下标来访问这些数据。在处理基本类型数据时（如以上示例代码的 int 型数组），数组保存的是变量的值，如果程序员未提供初始值，数组会把这些变量的值初始化为 0；而处理引用类型时（如以上示例代码的 String 型数组），数组保存的是数据的引用，如果程序员未提供初始值，数组会把这些变量的值初始化为 null。

面试题 048　new Object[5]语句是否创建了 5 个对象

Java 数组的本质是一个 Java 类，它也是通过 new 语句来实例化，但是这个 new 语句却不会实例化数组中的对象，一些程序员对它的理解有偏误。本例在回答该问题的同时，深入地讲解 Java 数组在内存中的构成。

【出现频率】 ★★★★

【关键考点】

- 数组的使用语法；
- Java 数组的内存构成机制。

【考题分析】

对于基本数据类型的数组，在 new 语句创建数组实例时，会为该数组分配指定长度的内存空间，并把数组的每个元素的值初始化为 0。那么，引用类型的数组又是如何来创建和初始化这些内存空间呢？其实，引用数据类型的数组也会分配指定长度的内存空间，只不过这些内存空间是用来保存引用的，而不是具体的值，它初始化的结果为 null。

```
Object[] objArr = new Object[3];
```

以上的代码中，创建了一个存储 Object 的数组，长度为 3。这 3 个元素的值都是 null，然后把创建好的数组实例的引用赋给 objArr 变量。如果需要为这些元素分配具体的对象，则需要分别指定或用“{}”符号进行初始化，如下所示：

```
Object[] objArr = new Object[]{new Object(),null,new String("abc")};
                                                    //引用类型的数组
or
objArr[0] = new Object();                           //数组元素再赋对象引用
objArr[2] = new String("abc");                      //数组元素再赋字符对象引用
```

以上数组实例的内存构成，如图 5.1 所示。

如果把数据遍历打印出来，可以得到下面的结果：

```
java.lang.Object@35ce36
null
java.lang.Object@757aef
```

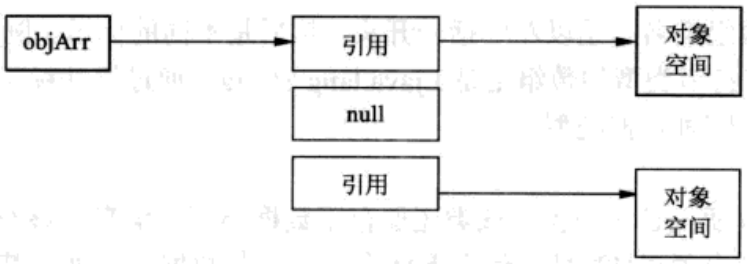


图 5.1　引用数据类型的数组内存结构图

【答案】

答案为否。题目的语句其实是创建了一个数组实例，长度为 5，每个数组元素的值均是 null，并没有创建 5 个 Object 对象。如果需要创建 5 个 Object 对象，则需要为每个数组元素分别指定。

面试题 049　如何拷贝数组的数据

程序员操作 Java 数组的时候，其实是通过一个引用来操作数组的，如果使用简单的=操作符进行赋值的话，只是复制了引用，并没有把数组的数据复制一份，一些程序员在复制数组时容易犯错。本例在回答该问题的同时，深入地讲解如何拷贝 Java 数组的数据。

【出现频率】 ★★★★

【关键考点】

- Java 处理数组的数据存储的本质；
- System.arraycopy()方法的使用。

【考题分析】

程序员在拷贝数组时，最容易犯的一种错误就是使用赋值运算符=来复制数组，然后对新数组的操作也影响到原数组的数据，那是因为它们根本指向的就是同一段内存空间，代码如下：

```
package ch05;
public class ArrayCopy {
    public static void main(String[] args) {          //main()方法
        int[] arr = new int[]{1,2,3};                  //定义一个 int 数组变量
        int[] arr2 = arr;                              //赋值引用
        arr2[2] = 7;                                   //改变一个元素的值
        for(int i : arr){                              //循环打印 arr 数组的值
            System.out.println(i);
        }
    }
}
```

以上的代码中，创建了一个长度为 3 的 int 型数组 arr，然后用赋值运算符=把 arr 赋值给 arr2。通过 arr2 把它的下标为 2 的元素的值改为 7，最后把 arr 的数组遍历打印一下，得到如下结果：

```
1
2
7
```

通过运行结果可以看出，arr 数组的数据被 arr2 给修改了，而程序员的本意却是想创建两个相对独立的数组，它们是互不影响的。其实，拷贝数组数据应该使用 System.arraycopy()方法，示例代码如下：

```
package ch05;
public class ArrayCopy {
    public static void main(String[] args) {          //main()方法
        int[] arr = new int[]{1,2,3};                  //定义一个 int 数组变量
        int[] arr2 = new int[3];                       //定义另一个 int 数组变量
        System.arraycopy(arr, 0, arr2, 0, arr.length);
                                                       //深拷贝数组
        arr2[2] = 7;
        for(int i : arr){                              //循环打印 arr 的值
            System.out.println(i);
        }
    }
}
```

打印结果如下：

```
1
2
3
```

可以看见，通过 System.arraycopy()方法，两个数组已经互不影响了。

说明：程序员也可以自己写一个数组拷贝方法，关键点在于认识数组的内存结构。

【答案】

使用 System.arraycopy()方法，或者创建一个新的数组实例，然后把数据一一装填进去，不能用=赋值运算符。

面试题 050　二维数组的长度是否固定

数组的本质是一种 Java 类，它的这种特性就导致它的多维数组比较有特点，可以无限的扩展维度，并且每个维度元素的长度也可以参差不齐。本例在回答该问题的同时，深入地介绍如何使用 Java 多维数组。

【出现频率】 ★★★

【关键考点】

- Java 处理数组的数据存储的本质；
- 多维数组。

【考题分析】

Java 的二维数组其实是这样的：先创建一个一维数组，然后该数组的元素在引用另外一个一维数组。在使用二维数组的时候，通过两个中括号[]来访问每一层维度的引用，直到访问到最终的数据。以下是一段使用二维数组的示例代码：

```
package ch05;
public class MultiDimArray {
    public static void main(String[] args) {
        //定义数组
```

```
        int[][] arr = new int[3][];
        arr[0] = new int[]{4};
        arr[1] = new int[]{4,5};
        arr[2] = new int[]{4,5,6};
        //逐个访问数组
        for(int[] a : arr){
            for(int i : a){
                System.out.print(i+"\t");
            }
            System.out.println();
        }
    }
}
```

在以上代码中，实例化二维数组时，并没有指定第二维的长度，也没有必要指定，因为它们的长度是可以各异的。上例的 arr.length 等于 3，但是 arr[i].length 却不相同。因此，遍历该二维数组时，打印出来的长度也是不同的，打印结果如下：

```
4
4    5
4    5    6
```

【答案】

长度不固定。Java 数组的长度是可以动态变化的，程序员可以任意扩展数组的维度，每一维度的元素个数都可以是不尽相同。

5.2　集 合 框 架

在 java.util 包下，有一些集合类，它们也是用于数据集的操作，功能比数组要强得多。在实际开发项目中，集合类的使用是很频繁的，它们各有特点，应用于不同的情况下。正确理解和熟练使用 Java 的集合类，有助于读者在编程过程中更好的操作数据集。本节将集中讨论有关 Java 集合类的常见面试题。

面试题 051　什么是集合

集合是用来存储其他对象的对象，它也只能用来存储其他对象。集合代表了一种底层结构，用于扩展数组的功能，它们的每一种形式往往代表了某一种数据结构。集合框架的内容比较多，结构也比较复杂，程序员往往不能把它们很好的区分开，本例在回答该问题的同时，全面地介绍 Java 集合框架。

【出现频率】 ★★★★

【关键考点】

- 集合框架的结构和内容；
- 每种集合的用途。

【考题分析】

集合，顾名思义，就是用来存储数据的对象，只不过在 Java 中，这些数据指的就是其他对象。它是一种数据结构的体现，也是对数组在功能上的扩展。java.util 包下的集合框

架主要由几个接口和实现类组成，大致组成如图 5.2 所示。

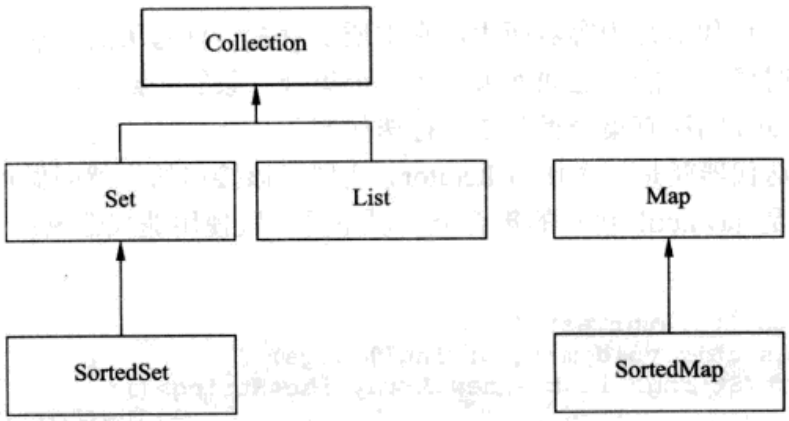


图 5.2　Java 集合框架体系

- 列表（List）：有序存放，允许重复，可以存放不同类型的对象；
- 集合（Set）：无序存放，不允许重复，可存放不同类型的对象；
- SortedSet：排好序列的 Set；
- 映射（Map）：俗称键值对，如手机中的电话本；
- SortedMap：排好序列的 Map。

注意：所有的集合实现类，都只能存放对象。如果需要存放基本数据类型的数据，则需要使用包装类。另外，JDK 5.0 以后，基本类型可以自动转换成包装类型。

【答案】

集合是用来也只能存储其他对象的对象，代表了一种底层结构，用于扩展数组的功能。集合框架由一系列的接口和实现类组成，基本包括：列表（List）、集合（Set）、映射（Map）等，它们大多具有可迭代性和可比较性。

注意：Set 也称集合，它相对于集合框架的概念跟具体一些。

面试题 052　迭代器是什么

可迭代是 Java 集合框架下的所有集合类的一种共性，也就是把集合中的所有元素遍历一遍。迭代的过程需要依赖一个迭代器对象，那么什么是迭代器呢？它应该如何使用呢？本例在回答该问题的同时，详细地讲解 Java 集合类与迭代器的关系。

【出现频率】 ★★★★

【关键考点】

- 迭代的概念；
- Iterator 接口。

【考题分析】

迭代器（Iterator）模式，又叫做游标（Cursor）模式。它的含义是，提供一种方法访问一个容器对象中各个元素，而又不需暴露该对象的内部细节。

注意：Java 的集合框架的集合类，有的时候也称为容器。

从定义可见，迭代器是为容器而生，它本质上就是一种遍历的算法。因为容器的实现千差万别，很多时候不可能知道如何去遍历一个集合对象的元素。Java 为开发者提供了使用迭代的接口，Java 的所有集合类都是进行迭代的。

简单地说，迭代器就是一个接口 Iterator。实现了该接口的类就叫做可迭代类，这些类多数时候指的就是 java.util 包下的集合类。以下是一段使用迭代器遍历 List 元素的示例代码：

```
public class IteratorTest {
    public static void main(String[] args) {
        List<String> list = new ArrayList<String>();
                                                    //创建保存字符串的 List 对象
        list.add("a");                              //放入元素 a
        list.add("b");                              //放入元素 b
        Iterator<String> it = list.iterator(); //得到 list 的迭代器
        //调用迭代器的 hasNext()方法，判断是否有下一个元素
        while(it.hasNext()){
            //将迭代的下标移动一位，并得到当前位置的元素
            System.out.println(it.next());
        }
    }
}
```

首先，创建了一个 List 的集合对象，并放入了两个字符串对象元素；然后通过 iterator()方法得到迭代器。Iterator()方法正是由 Itrable 接口所规定的，ArrayList 对该方法提供了具体的实现；在迭代器 Iterator 接口中，有以下 3 个方法。

- hasNext()：该方法用于判断集合对象是否还有下一个元素，如果已经是最后一个元素则返回 false。
- next()：把迭代器的指向移到下一个位置，同时，该方法返回下一个元素的引用。
- remove()：从迭代器指向的 Collection 中移除迭代器返回的最后一个元素，该操作使用得比较少。

说明：从 Java 5.0 开始，迭代器可以被 foreach 循环所替代，但是 foreach 循环的本质也是使用 Ierator 进行遍历的。

【答案】

迭代器，提供一种访问一个集合对象中各个元素的途径，同时又不需要暴露该对象的内部细节。Java 通过提供 Iterable 和 Iterator 两个接口来实现集合类的可迭代性。迭代器主要的用法就是，首先用 hasNext()作为循环条件；再用 next()方法得到每一个元素；最后再进行相关的操作。

面试题 053　比较器是什么

与迭代器不同，比较器指的是集合存储的元素的特性，如果元素是可以比较的，则可以进行相应的排序，否则不行。那么，什么是比较器呢？它应该如何使用呢？本例在回答

该问题的同时，详细地讲解 Java 集合类与比较器的关系。

【出现频率】 ★★★★

【关键考点】

- 可比较的概念；
- Comparable 接口；
- Comparator 接口。

【考题分析】

对于基础数据类型来说，它们一般都是具有大小顺序的，如 1 小于 2，2.5 大于 2.1 等等，就可以称它们是可比较的。那么对于一个用户自定义的类来说，又该如何比较它的实例之间大小呢？这就需要使用 Comparable 或 Comparator 接口了。

说明：对于字符串来说，String 类是实现了 Comparable 接口的，它也是可比较的。

对于 Comparable 接口来说，它往往是进行比较类需要实现的接口，它仅包含有一个 compareTo()方法，只有一个参数，返回值为 int 型数据。返回值大于 0 时，则表示本对象大于参数对象，小于 0 时，则表示本对象小于参数对象，等于 0 则表示两者相等，示例代码如下：

```
package ch05;

public class ComparableUser implements Comparable {     //可比较的用户类
    private String id;                                   //主键
    private int age;                                     //年龄
    public ComparableUser(String id, int age) {          //构造方法
        this.id = id;
        this.age = age;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public int compareTo(Object o) {                     //比较方法
        return this.age - ((ComparableUser) o).getAge();//用年龄进行比较
    }
    /**  * 测试方法    */
    public static void main(String[] args) {             //主方法
        //创建两个用户对象
        ComparableUser user1 = new ComparableUser("u1001", 25);
        ComparableUser user2 = new ComparableUser("u1002", 20);
        if(user1.compareTo(user2) > 0){                  //大于 0
            System.out.println("用户 1 大于用户 2");
        }else if(user1.compareTo(user2) < 0){            //小于 0
            System.out.println("用户 1 小于用户 2");
        }else{                                           //等于 0
```

```
            System.out.println("用户 1 等于用户 2");
        }
    }
}
```

因为 user1 的年龄大于 user2 的年龄，运行代码如下所示：

```
用户 1 大于用户 2
```

那么，Comparator 又是怎么一回事呢？Comparator 也是一个接口，它的实现者被称为比较器。它包含一个 compare()方法，有两个参数，返回值与 Comparable 的 compareTo()方法一样。不同之处是，Comparator 接口一般不会被集合元素类所实现，而是单独实现或用匿名内部类的方式实现。

说明：从 Java 5.0 开始，Comparable 和 Comparator 接口都已经支持泛型了，不再需要进行类型转换。

以上示例程序中的 ComparableUser 类也可以不用实现 Comparable 接口，用一个单独的 Comparetor 替代即可，例如下面的代码：

```
package ch05;
public class User{                                          //用户类
    private String id;                                      //主键
    private int age;                                        //年龄
    public User (String id, int age) {                      //构造方法
        this.id = id;
        this.age = age;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    /**  * 测试方法    */
    public static void main(String[] args) {                //主方法
        //创建两个用户对象
        User u1 = new User("u1001", 25);
        User u2 = new User("u1002", 20);
        Comparator comp = new UserComparator();             //生成比较器
        int rst = comp.compare(u1,u2);                      //得到比较结果
        if(rst > 0){                                        //大于 0
            System.out.println("用户 1 大于用户 2");
        }else if(rst < 0){                                  //小于 0
            System.out.println("用户 1 小于用户 2");
        }else{                                              //等于 0
            System.out.println("用户 1 等于用户 2");
        }
    }
```

```
}
class UserComparator implements Comparator{                    //比较器类
    public int compare(Object arg0, Object arg1) {             //比较方法
        User u1 = (User) arg0;                                 //类型转换
        User u2 = (User) arg1;
        return u1.getAge() - u2.getAge();                      //结果
    }
}
```

以上代码的结果是一样的。其实，比较器的运用不仅限于元素的比较，它往往还用于集合元素的排序。它可以提供强行对某个集合或数组进行整体排序的比较函数，将 Comparator 传递给 sort 方法（如 Collections.sort 或 Arrays.sort），从而允许在排序顺序上实现精确控制。

【答案】

比较器是把集合或数组的元素强行按照指定方法进行排序的对象，它是实现了 Comparator 接口类的实例。如果一个集合元素的类型是可比较的（实现了 Comparable 接口），那么它就具有了默认的排序方法，比较器则是强行改变它默认的比较方式来进行排序。或者有的集合元素不可比较（没有实现 Comparable 接口），则可用比较器来实现动态的排序。

面试题 054　Vector 与 ArrayList 的区别

Vector 与 ArrayList 都是 List 接口的实现类，它们都代表链表形式的数据结构。但是，它们的实现和适用场景却有所区别。本例在回答该问题的同时，详细地讲解 Vector 与 ArrayList 的实现和使用方法。

【出现频率】 ★★★★★

【关键考点】

- List 接口的使用方法；
- Vector 的线程安全性。

【考题分析】

SUN 公司提供的 JDK 开发包中，包含了一个 src.zip 的文件，里面是 Java 主要基础类的源码。通过打开 Vector.java 和 ArrayList.java 两个源文件，可以发现，它们二者的实现非常相似，都是用一个对象数组来存储元素的。但是，Vector 的大多数成员方法都会加上 synchronized 关键字，也就是说 Vector 线程是安全的。

由于 Vector 操作成员的方法必须保证同步，因此它的执行效率没有 ArrayList 高。所以，一般情况下，只有在必须保证同步的地方才使用 Vector，而多数情况下，使用 ArrayList 更适合一些。

它们两者的使用比较类似，一般通过 add()方法来加元素；remove()方法删除元素，size()得到集合元素的数量等。以下是一段使用 Vector 和 ArrayList 的示例代码：

```
public class ListTest {
    public static void main(String[] args) {
        Vector<String> v = new Vector<String>();    //创建 Vector 对象
        v.add("hello");                             //加入元素
```

```
        v.remove("hello");                          //删除元素
        System.out.println(v.size());               //得到 vector 的元素数量
        ArrayList<String> al = new ArrayList<String>();
                                                    //创建一个 ArrayList 对象
        al.add("hello");                            //加入元素
        al.remove("hello");                         //删除元素
        System.out.println(al.size());              //得到链表对象中的元素数量
    }
}
```

【答案】

Vector 是线程安全的，因为它操作元素的方法都是同步方法，而 ArrayList 则不是。开发过程中应该根据需要进行选择，如果需要保证线程安全的地方则需要使用 Vector，而不必要的时候则无需使用 Vector，因为 ArrayList 效率会高一些。

面试题 055　HashMap 和 HashTable 的区别

对于 Map 接口来说，它有两种比较重要的实现类 HashMap 和 HashTable，它们保存元素的时候，都是无序的。但是，它们也有一定的区别，适用于不同的情况。本例在回答该问题的同时，详细地讲解 HashMap 和 HashTable 的使用方法。

【出现频率】 ★★★★★

【关键考点】

- Map 接口是使用方法；
- HashMap 和 HashTable 的区别。

【考题分析】

HashTable 相对于 HashMap 更早提出，应用也非常广泛，HashMap 后来提出是为了代替 HashTable 的类，也就是说建议使用 HashMap，不要使用 HashTable。可能有的程序员觉得 HashTable 很好用，为什么不用呢？这里简单分析一下它们的区别。

（1）HashTable 的方法是同步的，HashMap 不能同步，所以在多线程场合要使用 HashTable，这个区别就像 Vector 和 ArrayList 一样。

（2）HashTable 不允许 null 值（key 和 value 都不可以），HashMap 允许 null 值（key 和 value 都可以）。

（3）HashTable 有一个 contains()方法，功能和 containsValue()功能一样。

（4）HashTable 使用 Enumeration 遍历，而 HashMap 使用 Iterator 进行遍历。

（5）HashTable 中 hash 数组默认大小是 11，增加的方式是：old*2+1。HashMap 中 hash 数组的默认大小是 16，而且一定是 2 的指数。

（6）哈希值的使用不同，HashTable 直接使用对象的 hashCode，而 HashMap 会重新计算 hash 值。

开发者在使用它们的时候，前 4 个区别是需要注意的，这些会直接影响到开发者的编码方式。而后两个区别，是它们底层的实现区别，开发者一般不用去关注它。其实，它们更多的是体现出 Map 接口是特性，下面是一段 Map 的使用示例代码：

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

```
        Map<String, String> map = new HashMap<String, String>();
                                                    //创建 Map 对象
        map.put("a", "123");                        //存放元素
        map.put("b", "456");
        map.put("c", "789");
        for(String key : map.keySet()){             //通过 key 的集合进行遍历
            System.out.println(key+":"+map.get(key));
                                                    //用 get()方法获取 value
        }
    }
}
```

说明：HashMap 更符合 Java 集合框架的设计思路，更推荐使用。

【答案】

HashMap 和 HashTable 的区别主要有以下几个方面。

- HashTable 的方法是同步的，HashMap 不能同步。
- HashTable 不允许 null 值（key 和 value 都不可以），HashMap 允许 null 值（key 和 value 都可以）。
- HashTable 有一个 contains()方法，功能和 containsValue()功能一样。
- HashTable 使用 Enumeration，HashMap 使用 Iterator。
- HashTable 中 hash 数组的初始化大小及其增长方式不同。
- 哈希值的使用不同，HashTable 直接使用对象的 hashCode，而 HashMap 会重新计算 hash 值。

面试题 056　集合使用泛型带来了什么好处

泛型是 Java 5.0 以后才有的特性，它的使用非常频繁，而 Java 的集合框架更是泛型使用最多的地方之一。什么是泛型？它为开发带来了什么好处？本例在回答该问题的同时，详细地讲解泛型的概念和使用方法。

【出现频率】 ★★★★★

【关键考点】

- 泛型的概念和使用方法；
- 泛型集合框架的使用方法。

【考题分析】

首先，了解一下 Java 关于泛型的概念。泛型，在 C++中被称为模板，就是一种抽象的编程方式，当开发者定义类和方法的时候，可以用一种通用的方式进行定义，而不必写出具体的类，这些未知的东西会在真正使用的时候再确定。

对于集合类来说，它们可以存放各种类型的元素。如果在存放元素之前，就能确定元素的类型，那么这样既让开发者更直观，也让代码更加简洁。下面是一段关于泛型的代码：

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

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

```
        List list1 = new ArrayList();                    //无泛型创建链表对象
        list1.add("a");
        list1.add("b");
        Iterator it = list1.iterator();                  //获取链表迭代器
        while(it.hasNext()){
            Object obj = it.next();                      //获取元素，类型为 Object
            String val = (String) obj;                   //类型转换
            System.out.println(val);
        }
        List<String> list2 = new ArrayList<String>();//String 泛型链表对象
        list2.add("a");
        list2.add("b");
        //用 foreach 进行遍历，类型直接转换为 String
        for(String str : list2){
            System.out.println(str);
        }
    }
}
```

以上代码中，list1 没有使用泛型，它存放的所有元素都是 Object 类型，在使用的时候还需要进行类型的强制转换。而泛型链表对象 list2，它在创建的时候就提供了元素的类型。因此获取元素的时候，类型也会自动转换，这也是集合使用泛型的最直接的好处。

说明：Java 的泛型是停留在编译层的，也就是说 JVM 在对待泛型数据的时候，依然把它们看成是 Object 类型。只不过在使用这些元素的时候，JVM 会自动帮助开发者进行相应的类型转换。

【答案】

集合使用泛型以后，可以达到元素类型明确的目的，避免了手动类型转换的过程，同时，也让开发者更加明确容器保存的是什么类型的数据。

面试题 057　如何把集合对象中的元素进行排序

排序是一项很常用的技能。对于 TreeSet 这类集合它们是可以自动排序的，但是对于其他一些集合类型来说，又该怎么排序呢？本例在回答该问题的同时，详细地讲解集合排序的使用方法。

【出现频率】 ★★★★★

【关键考点】

- 集合框架的体系结构；
- 集合排序的方法。

【考题分析】

在 Java 集合框架中定义的 List 的实现有 Vecto、ArrayList 和 LinkedList。这些集合提供了对对象组的访问、添加与删除支持。但是，它们并没有内置的元素排序支持。此时，开发者可以使用 java.util.Collections 类中的 sort()方法对 List 元素进行排序，可以给方法传递一个 List 对象，也可以传递一个 List 和一个 Comparator 实例。

如果列表中的元素全都是相同类型的，并且这个类实现了 Comparable 接口，可以简单的调用 Collections.sort()。如果这个类没有实现 Comparator，就可以传递一个 Comparator

实例作为 sort()的第二个参数进行排序。另外，如果不想使用默认的分类顺序进行排序，同样可以传递一个 Comparator 实例作为参数来进行排序。

说明： 如果元素是 String 对象，默认的排序顺序是按照字符编码进行的，基本上是每个字符的 ASCII/Unicode 值。如果处理的是英文，默认的排序顺序通常是足够的，因为它首先排 A-Z，然后是小写字母 a-z。如果在处理非英文字的时候，则一般需要单独处理了。

```
package ch05;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.Iterator;
import java.util.List;
class Student implements Comparable<Student> {          //学生类，可比较
    private String name;                                //姓名
    private int age;                                    //年龄
    public Student(String name, int age) {              //构造方法
        super();
        this.name = name;
        this.age = age;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public int compareTo(Student stu) {                 //比较方法
        if (this.age > stu.age)                         //大于则返回 1
            return 1;
        else if (this.age < stu.age)                    //小于则返回-1
            return -1;
        else
            return 0;                                   //等于则返回 0
    }
    @Override
    public String toString() {                          //覆盖 toString 方法
        return name + '_' + age;
    }
}
public class SortTest {                                 //测试主类
    public static void main(String[] args) {            //主方法
        List<Student> list = new ArrayList<Student>(); //创建一个 List 对象
        list.add(new Student("stu001", 21));            //添加 4 个学生对象
        list.add(new Student("stu003", 29));
        list.add(new Student("stu002", 25));
        list.add(new Student("stu004", 23));
        System.out.println("-------------原顺序---------------");
        for (Iterator<Student> iter = list.iterator(); iter.hasNext();) {
```

```
            Student temp = iter.next();
            System.out.println(temp);
        }
        System.out.println("------------默认排列--------------");
        Collections.sort(list);
        for (Iterator<Student> iter = list.iterator(); iter.hasNext();) {
            Student temp = iter.next();
            System.out.println(temp);
        }
        System.out.println("-----------降序排列-------------");
        Comparator comp = Collections.reverseOrder();
        Collections.sort(list, comp);
        for (Iterator<Student> iter = list.iterator(); iter.hasNext();) {
            Student temp = iter.next();
            System.out.println(temp);
        }
        System.out.println("-----------按照名字重新排序----------");
        Collections.sort(list, new Comparator<Student>() {
                                        //定义一个新的比较器
            public int compare(Student s1, Student s2) {
                return s1.getName().compareTo(s2.getName());
                                        //利用姓名升序排列
            }
        });
        for (Iterator<Student> iter = list.iterator(); iter.hasNext();) {
            Student temp = iter.next();
            System.out.println(temp);
        }
    }
}
```

以上代码中，定义了一个学生类，它有两个属性：姓名和年龄，并且学生类还实现了 Comparable 接口，提供了 compareTo()方法的具体实现。默认情况下，是按照学生的年龄进行排序的。

在测试方法中，使用了 4 种排序方法进行排序，分别是原顺序、默认顺序、降序排序和按照名字排序。由于 Student 类实现了 Comparable 接口，所以它有一种默认的排序，也就是按照年龄升序排列。另外，最后一个排序是按照年龄进行排序，因此提供了一个额外的比较器，它使用的是 String 的比较器。

说明： 比较器是进行排序的基础，大多数的比较和排序功能都会使用它，例如，TreeSet 本质上也要使用它。

以上示例程序的运行结果如下：

```
----------------------原顺序----------------------
stu001_21
stu003_29
stu002_25
stu004_23
-----------------默认排列-----------------------
stu001_21
stu004_23
stu002_25
stu003_29
--------------------降序排列----------------------
stu003_29
```

```
stu002_25
stu004_23
stu001_21
------------------按照名字重新排序-------------------
stu001_21
stu002_25
stu003_29
stu004_23
```

【答案】

对于没有排序功能的集合来说，都可以使用 java.util.Collections.sort()方法进行排序，它除了集合对象以外，还需要提供一个比较器。如果列表中的元素全都是相同类型的，并且这个类实现了 Comparable 接口，可以简单地调用 Collections.sort()。如果这个类没有实现 Comparator，就可以传递一个 Comparator 实例作为 sort()的第二个参数进行排序。另外，如果不想使用默认的分类顺序进行排序，同样可以传递一个 Comparator 实例作为参数来进行排序。

面试题 058　符合什么条件的数据集合可以使用 foreach 循环

许多语言都支持 foreach 循环，Java 从 5.0 开始也支持了。那么是不是所有的对象都可以使用 foreach 循环呢？使用 foreach 循环的对象应该符合哪些条件呢？本例在回答该问题的同时，详细地讲解 foreach 的使用方法和原理。

【出现频率】 ★★★★★

【关键考点】

- 迭代器的定义和使用；
- foreach 的使用方法和原理。

【考题分析】

从含义上来说，foreach 循环就是遍历一个集合中的元素，起到替代迭代器的作用。从语法上来讲，数组或者实现了 Iterable 接口的类实例，都是可以使用 foreach 循环的，例如下面的代码：

```
List list = new ArrayList();                    //定义链表集合
list.add("a");                                  //添加元素
list.add("b");
list.add("c");
for(String s : list){                           //使用 foreach 循环
    System.out.println(s);
}
String[] arr = {"a","c","b"};                   //创建数组
for (String s : arr) {                          //使用 foreach 循环
    System.out.println(s);
}
```

数组是 Java 规定的内容，开发人员无法改变它，只能遵照它的使用语法来使用。但是，对于“实现了 Iterable 接口的类实例”，开发人员则可以自定一个集合类。该自定义集合类主要需要做以下一些事情。

（1）定义一个类，包含一个整型下标成员变量和一个集合对象（如数组或链表）。

（2）将该类实现 Iterable 接口。

（3）提供一个 Iterator 接口的实现，或者它本身就实现 Iterator 接口。

（4）使用下标成员变量和集合对象来完成 Iterator 接口所需要的方法。

以下是一个自定义的可使用 foreach 循环的类：

```
package ch05;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
public class MyForeach {                                    //测试类
    public static void main(String[] args) {                //主方法
        MyList list = new MyList();                         //创建 List 集合对象
        list.getList().add("a");                            //添加元素
        list.getList().add("b");
        list.getList().add("c");
        for(String s : list){                               //使用 foreach 循环
            System.out.println(s);
        }
    }
}
class MyList implements Iterable<String>,Iterator<String>{ //自定义链表类
    private int loc = 0;                                    //当前的下标
    private List<String> list = new ArrayList<String>();
                                                            //存储数据的 ArraylList
    public boolean hasNext() {                              //是否有下一个元素
        return list.size() > loc;
    }
    public String next() {                                  //得到下一个元素
        return list.get(loc++);
    }
    public void remove() {                                  //删除当前下标的元素
        list.remove(loc);
    }
    public List<String> getList() {
        return list;
    }
    public void setList(List<String> list) {
        this.list = list;
    }
    public Iterator<String> iterator() {                    //得到迭代器
        return this;
    }
}
```

通过以上代码不难发现，Foreach 的运行原理也是比较简单的，它的主要运行步骤如下。

（1）调用指定集合对象的 Iterator()方法，得到迭代器。

（2）然后使用迭代器的 hasNext()方法判断是否有下一个元素来进行循环。

（3）每一次循环调用 next()方法，得到下一个元素。

【答案】

数组或者实现了 Iterable 接口的类实例，都是可以使用 foreach 循环的。Java 集合框架

中的集合类大多数都符合第二条，因此它们都是可以使用 foreach 循环的。

5.3 小　结

本章讲解了一些 Java 数组和集合的面试题，它们都是用于保存数据元素的。其中 Java 的数组相对于其他语言是非常有特点的，也比较难以理解，因为数组本质上就是一个类，然而处理基本类型和引用类型数据时又有所不同。java.util 包下的集合框架是几乎所有项目开发都会使用的，每一种数据结构的特点和适用场合是读者应该掌握的重点。泛型的推出更是让集合框架更便于使用，读者平时应该加强练习，理解它们的原理才是面对这类面试题的关键所在。

第 6 章　Java 图形用户界面

图形用户界面（GUI）是 Java 开发的一个重要的方面，尽管它目前的流行程度并不高。但是，一些对 Swing 情有独钟的公司依然要求开发者具有开发图形用户界面程序的能力。另外，Java 的图形用户属于 Java SE 范畴，是学习 Java 过程中的一个阶段，它的 API 设计得合理而易用，通过考察图形用户界面的知识可以看出求职者的一些 Java 开发和设计功底。本章将包含关于 Java 图形用户界面开发（Swing）的一些问题，并且分析这些题目和知识点，帮助读者梳理这些方面的知识。

6.1　图形用户界面基础

目前，Java 关于图形用户界面的开发存在两套 API，一个是 AWT，另一个是 Swing。但是 AWT 已经慢慢的退出了历史舞台，而大多数时候 Java 图形用户界面开发指的就是 Swing 开发。尽管 Swing 是一套庞大的 API 体系，但是了解它基本的设计思想以后，掌握 Swing 开发也并非难事。本节将集中讨论有关 Swing 基础的常见面试题。

面试题 059　JFrame 的作用是什么？它应该如何使用

大家知道，一个典型的图形用户界面是一个窗口的形式呈现给用户的，Swing 也不例外。其实，JFrame 就起到了 Swing 窗口的作用，其他一切的组件都在它的包含之内。本例在回答该问题的同时，详细地讲解 Swing 窗口及其 JFrame 的用法。

【出现频率】 ★★★★

【关键考点】

- Swing 的窗口定义思想；
- JFrame 的使用方法。

【考题分析】

大多数的 Swing 应用程序都是构建在基础的 Jframe 中的，它使得 Swing 程序可以在任何操作系统上创建视窗应用程序。以下为一个用 JFrame 类创建一个视窗的示例。

```
package ch06;
import javax.swing.JFrame;
public class HelloJFrame {                                    //测试类
    public static void main(String[] args) {                  //主方法
        JFrame jf = new JFrame("hello swing");                 //创建 JFrame 对象
        //设置窗口在关闭时，程序的响应
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
        jf.setSize(200, 200);                          //设置窗口大小
        jf. setVisible (true);                         //是否可见
    }
}
```

图 6.1　JFrame 窗口示例

窗口的标题通过 JFrame 构造方法的 String 类型的参数来提供。以上示例代码的窗口的标题是“hello swing”。通过 setDefaultCloseOperation()方法告诉程序应该在窗口被关闭时退出。setSize()方法则用来定义窗口的尺寸。setVisible()方法则用来告诉程序应该把窗口呈现出来。以上示例代码的窗口效果如图 6.1 所示。

注意：以上示例代码中的 setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)方法是必须要调用的，否则默认情况下，视窗关闭时是不会结束程序的。

拥有了窗口以后，就可以在窗口中添加内容了，包括：按钮（Button）、文本框（Text）、滚动条（Scroll Bar）、标签（Label）等。有了这些内容以后，窗口就可以变得互动起来。例如，以下示例代码为以上的 JFrame 示例程序添加一个标签。

```
package ch06;
import javax.swing.JFrame;
public class HelloJFrame {                             //测试类
    public static void main(String[] args) {           //主方法
        JFrame jf = new JFrame("hello swing");         //创建 JFrame 对象
        //设置窗口在关闭时，程序的响应
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setSize(200, 200);                          //设置窗口大小
        jf. setVisible (true);                         //是否可见
        //为窗口添加一个标签
        JLabel label = new JLabel();                   //创建 JLable 对象
        label.setText("Hello this is one lable");      //设置标签文本
        jf.add(label);                                 //把标签添加到 jframe 中
    }
}
```

说明：Swing 的大多数组件类的类名都是以 J 字母开头，主要是为了与 AWT 的老的组件相区别。

运行以上的示例代码，就可以看见窗口中多了行文字，如图 6.2 所示。

【答案】

JFrame 是 Swing 创建视窗的一个基础类，它像一个容器一样，可以包含其他的组件进来，是其他组件赖以生存的对象。JFrame 的使用比较简单，主要有以下几个步骤：

（1）用 new 语句创建 JFrame 对象，可以通过构造方法传入视窗标题参数。

（2）设置窗口关闭时的行为，一般是结束进程。

（3）设置视窗的外观，例如，尺寸、可见与否等。

（4）为窗口中添加其他的组件，例如，标签、按钮等。

图 6.2　JFrame 加入 Jlabel 以后的效果图

（5）必要时还可以添加一些事件，例如，WindowListener 等。

面试题 060　如何创建一个按钮

按钮是一种最常见的组件，Swing 肯定也对其提供了充分的支持。那么，如何在 Swing 程序中定义和使用一个按钮组件呢？本例在回答该问题的同时，详细地讲解 Swing 的按钮（JButton）组件的使用方法。

【出现频率】 ★★★★

【关键考点】

- Swing 的组件的含义；
- JButton 的使用方法。

【考题分析】

在 Swing 中，创建一个按钮只需要使用 JButton 类即可。按钮的文本可以通过 JButton 的构造方法提供。JButton 是一个组件，它有自己的图形绘制方式，可以在整个界面刷新的时候自动被重绘。也就是说，开发者不必显式的绘制一个按钮在界面上，只需要把它放在窗口中，自己就会被绘制出来。

说明：按钮的表面不仅可以写文字，还可以显示各种图形，详细代码如下。

```
package ch06;
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
public class HelloJButton {                                    //测试类
    public static void main(String[] args) {                   //主方法
        JFrame jf = new JFrame("hello swing");                 //创建 JFrame 对象
        //设置窗口在关闭时，程序的响应
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setSize(200, 200);                                  //设置窗口大小
        jf.setVisible(true);                                   //是否可见
        jf.setLayout(new FlowLayout());                        //设置布局格式
        //为窗口添加一个按钮
        JButton btn = new JButton("my button");                //创建一个按钮对象
        jf.add(btn);                                           //添加到窗口中
    }
}
```

以上示例代码中，首先创建了一个 JFrame 对象，然后把一个 JButton 对象加到了该窗口中。另外，在添加按钮之前，还需要设置一下 JFrame 的布局格式。以上示例采用的 FlowLayout，它是一种从左到右，从上到下连续均匀分布的一种布局管理器，效果如图 6.3 所示。

图 6.3　JButton 示例的效果图

注意：如果没有设置布局格式，JFrame 是无法直接加入 JButton 组件的，即使调用了 add()方法，界面上也是空白的。

【答案】

在 Swing 程序中，一般使用 JButton 类来定义按钮组件，它的使用思路大致如下所示：

（1）用 new 语句创建 JButton 对象，可以通过构造方法传入文本参数。

（2）把创建好的 JButton 对象加入到容器组件中，例如，JFrame。

（3）必要时还可以添加一些事件，例如，ActionListener 等。

面试题 061　如何使用文本输入组件

输入文本是任何一种图形用户界面所必须的，Swing 也对其提供了充分的支持。那么，在 Swing 程序中可以使用哪些文本区域呢？它们的定义和使用方法是什么呢？本例在回答该问题的同时，详细地讲解 Swing 的文本区域（JTextField 和 JTextArea）组件的使用方法。

【出现频率】 ★★★★

【关键考点】

- Swing 的组件的含义；
- JTextField 的使用方法；
- JTextArea 的使用方法。

【考题分析】

在图形用户界面开发领域中，根据文本输入的长度不同，一般可以分为文本框和文本域两种文本输入组件。Swing 提供了 JTextField 来对文本框提供支持，用 JTextArea 来表示一个文本框。

JTextField 代表的是只能单行输入的文本组件，与按钮类似，它也只需要放在窗口中，就可以自行完成绘制，示例代码如下：

```
package ch06;
import java.awt.FlowLayout;
import javax.swing.JFrame;
import javax.swing.JTextField;
public class HelloJTextField {                                    //测试类
    public static void main(String[] args) {                     //主方法
        JFrame jf = new JFrame("Helll Text");                     //创建 JFrame 对象
        jf.setLayout(new FlowLayout());                           //设置布局格式
        //为窗口添加一个文本框
        JTextField jtf = new JTextField(10);                      //创建一个文本框对象
        jtf.setText("初始化内容");                                  //设置初始文本内容
        jf.add(jtf);                                              //添加到窗口中
        showMe(jf);
    }
    private static void showMe(JFrame jf){
        //设置窗口在关闭时，程序的响应
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setSize(200, 200);                                     //设置窗口大小
        jf.setVisible(true);                                      //是否可见
    }
}
```

JTextField 对象在创建的时候，可以提供一个整型的参数，代表的是该文本框的绘制

长度。如果需要为该文本框赋值的话，则调用 setText()方法，传入字符串参数即可，示例程序效果如图 6.4 所示。

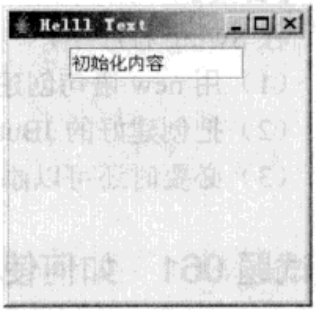


图 6.4　JTextField 示例的效果图

说明：在使用文本框的时候，一般需要在它的前面打印一个说明文字，此时一般使用 JLabel 即可。

JTextArea 也是用于文本的输入的组件，它的不同之处在于它允许多行输入，它的体积往往比 JTextField 要大一些。用户在 JtextArea 中按 Enter 键的时候，字符就出现了换行，示例代码如下：

```
package ch06;
import java.awt.FlowLayout;
import javax.swing.JFrame;
import javax.swing.JTextArea;
public class HelloJTextArea {                                    //测试类
    public static void main(String[] args) {                     //主方法
        JFrame jf = new JFrame("Helll Text");                    //创建 JFrame 对象
        jf.setLayout(new FlowLayout());                          //设置布局格式
        //为窗口添加一个文本域
        JTextArea jta = new JTextArea(5,10); //创建一个 5 行 10 列的文本域对象
        jta.setText("初始化内容");                                //设置初始文本内容
        jf.add(jta);                                             //添加到窗口中
        showMe(jf);
    }
    private static void showMe(JFrame jf){
        //设置窗口在关闭时，程序的响应
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setSize(200, 200);                                    //设置窗口大小
        jf.setVisible(true);                                     //是否可见
    }
}
```

在创建 JTextArea 时，与 JTextFiled 不同的是，需要提供两个整型参数，它们代表的是文本域的行数和列数，效果如图 6.5 所示。

图 6.5　JTextArea 示例的效果图

【答案】

Swing 提供两种文本输入组件，一个是 JTextField，它代表只能单行输入的文本框，另一个则是 JTextArea，它代表可以多行输入。它们的使用方法类似，都是通过在容器（例如，JFrame）中添加组件即可。

JTextField 组件在创建对象的时候，一般需要提供长度参数。JTextArea 组件的创建过程则提供的是长度和高度两个参数。

面试题 062　如何捕获事件

如果一个图形用户界面程序光有一些组件是远远不够的，如何让这些组件联系和互动起来才是关键。这个互动性主要就体现为事件模型，也就是如何去监听某个组件的某种行

为？如何做出响应？Swing 对事件模型提供了很好的支持。本例在回答该问题的同时，将更详细地讲解 Swing 事件模型的基本使用思路。

【出现频率】 ★★★★★

【关键考点】

- Swing 的组件模型；
- Swing 的事件模型。

【考题分析】

什么是事件？就是一个对象（对象 A）的状态改变了的时候，通知其他的对象（对象 B）发生了这么一件事，让它做出某种反应。

说明： 事件模型一般有两种模式，即推/拉模式。推模式，状态改变的对象（A）通知其他对象（B），Java 的大多事件模型都属于这一种。拉模式，其他对象（B）监听感兴趣的对象（A）（例如 WIN32 的图形用户界面编程）。

Swing 采用的事件模型有一点像订阅/发布模式，就是 B 向 A 注册（实际就是把自己的引用复制一份给 A），然后当 A 的状态改变时（例如，按钮被单击），调用 B 相关的方法，从而达到通知状态改变的目的。

在一次事件模型中，往往包含了 3 种角色，事件发生的主体（例如，按钮）、监听器（例如，监听按钮状态的监听器）和事件（例如，按钮被单击）。事件发生的主体是事件的起源地，它们的状态一般是随着用户的操作而改变；监听器是一个类，这些类往往都会继承自某个接口；事件代表了一种动作，通过它可以获取到一些用户操作的信息。在 Swing 编程的过程中，一般通过以下步骤来使用事件模型。

（1）创建组件对象，例如，JButton。

（2）创建实现了监听器接口或继承自适配器类的实现类，实现需要进行相应动作的抽象方法。

（3）调用组件对象的 addXXXListener()方法，为该组件添加监听器。

以下是一个 Swing 事件模型的示例：

```
package ch06;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JTextField;
public class HelloEvent {
    //创建一个文本框对象
    private static JTextField text = new JTextField(10);
    public static void main(String[] args) {                    //主方法
        JFrame jf = new JFrame("Helll Text");                    //创建 JFrame 对象
        jf.setLayout(new FlowLayout());                          //设置布局格式
        //为窗口添加一个文本框
        jf.add(text);                                            //添加到窗口中
        //为窗口添加一个按钮
        JButton btn = new JButton("my button");                  //创建一个按钮对象
        jf.add(btn);                                             //添加到窗口中
        btn.addActionListener(new ActionListener(){              //添加事件
```

```
            //定义事件回调方法
            public void actionPerformed(ActionEvent e) {
                HelloEvent.text.setText("按钮被点击了"); //动作
            }
        });
        showMe(jf);
    }
    private static void showMe(JFrame jf){
        //设置窗口在关闭时，程序的响应
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setSize(200, 200);                                    //设置窗口大小
        jf.setVisible(true);                                     //是否可见
    }
}
```

在以上的示例代码中，定义了一个文本框组件和一个按钮组件，当按钮被单击时则文本框中出现一段文字。监听器类是用一个匿名内部类来完成的，它实现的接口是 ActionListener，该接口可用于监听按钮被单击的事件，通过调用 JButton 组件的 addActionListener() 方法来添加该监听器。如图 6.6 所示，当按钮被单击时，文本框中打印了“按钮被点击了”。

图 6.6　事件模型示例的效果图

【答案】

在 Swing 程序中，通过注册监听器的方式来捕获事件。也就是说，组件可以根据需要来添加监听器进行某些事件的监听，其他事件则不予理睬。Swing 的事件模型编程思路有以下几项。

（1）创建组件对象，如 JButton。

（2）创建实现了监听器接口或继承自适配器类的实现类，实现需要进行相应动作的抽象方法。

（3）调用组件对象的 addXXXListener()方法，为该组件添加某监听器。

6.2　布 局 控 制

Swing 的布局方式不同于其他的 GUI 开发技术，它的布局都是通过代码来控制的。不同于网页，Swing 不是通过绝对定位来控制组件的摆放顺序，而是由布局管理器来负责控制，每一种布局管理器的所控制的组件的大小、形状和位置是各不相同。另外，布局管理器让组件的大小随着窗口的缩放而跟着变化。任何一种容器类组件，都有一个 setLayout() 的方法，它需要的参数就是一个布局管理器对象。在 Swing 中，常用的布局管理器包括：BorderLayout、FlowLayout 和 GridLayout。本节将集中讨论有关 Swing 布局的常见面试题。

面试题 063　如何使用 BorderLayout 布局

BorderLayout 是一种最常用的布局方式，它以简单易用而著称，是 Swing 容器的默认布局管理器。那么，它的含义是什么呢？它应该如何使用呢？本例在回答该问题的同时，

详细地讲解 Swing 关于 BorderLayout 布局管理器的设计思想和使用方法。

【出现频率】★★★★★

【关键考点】

- Swing 的布局管理器的含义；
- BorderLayout 布局管理器的含义；
- BorderLayout 布局管理器的使用方法。

【考题分析】

除非有其他说明，Swing 中任何一件容器类组件都默认使用 BorderLayout 布局管理器。BorderLayout 是把容器分为东、西、南、北和中间 5 个部分。除了中间部分，其他几个部分都会贴近容器的边缘，而中间是被其他四者所包围。

在编程方面，通过 add()方法把组件安放在指定方向的部位，该方法的第二个参数就是 BorderLayout 的方位，例如下面的代码：

```
frame.add(button, BorderLayout.NORTH);                    //把一个按钮放在北部
```

注意：如果通过只有一个参数的 add()方法添加组件，容器会默认地把组件放在中间位置。

表 6.1 列出了 BorderLayout 布局的方位变量及其含义。

表 6.1　BorderLayout的几种方位变量

变　量	说　明	变　量	说　明
BorderLayout.NORTH	上端	BorderLayout.WEST	左端
BorderLayout.SOUTH	下端	BorderLayout.CENTER	中部
BorderLayout.EAST	右段		

以下是一个用 BorderLayout 布局管理布局 5 个按钮组件的示例。

```
package ch06;
import java.awt.BorderLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
public class HelloBorderLayout {
    public static void main(String[] args) {                    //主方法
        JFrame jf = new JFrame("BorderLayout test");            //创建 JFrame 对象
        jf.setLayout(new BorderLayout());                       //设置布局方式
        jf.add(new JButton("east"), BorderLayout.EAST);                 //东部
        jf.add(new JButton("center"), BorderLayout.CENTER);             //中间
        jf.add(new JButton("south"), BorderLayout.SOUTH);               //南部
        jf.add(new JButton("north"), BorderLayout.NORTH);               //北部
        jf.add(new JButton("west"), BorderLayout.WEST);                 //西部

        jf.setSize(300, 200);                                   //分别代表长度和高度
        jf.setVisible(true);                                    //可见，默认为 false
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                                                                //默认为隐藏而不停止程序
    }
}
```

以上示例代码比较简单，就是在 BorderLayout 布局的 5 个部位放置 5 个按钮组件。因此，以上代码的运行效果如图 6.7 所示。

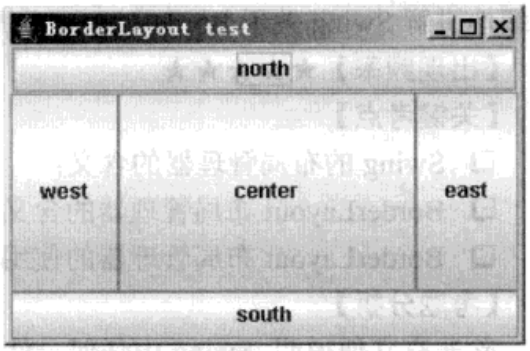


图 6.7　BorderLayout 布局管理器示例的效果图

【答案】

BorderLayout 是 Swing 容器的默认布局管理器，它的含义是采用东南西北中 5 个方位来进行布局，可以分别往这些方位上放置组件。在编程时，使用容器组件的 add()方法即可，第 1 个参数为需要添加的组件，第 2 个参数则是布局的方位。

面试题 064　如何使用 FlowLayout 布局

FlowLayout 是另外一种常用的布局方式，它的布局方式就好像用水装瓶子一样，一瓶装满了就换下一瓶。那么，它应该如何使用呢？本例在回答该问题的同时，详细地讲解 Swing 关于 FlowLayout 布局管理器的设计思想和使用方法。

【出现频率】 ★★★★

【关键考点】

- Swing 的布局管理器的含义；
- FlowLayout 布局管理器的含义；
- FlowLayout 布局管理器的使用方法。

【考题分析】

简单地说，FlowLayout 的含义就是把组件从左到右的排列，当排列到不能排以后就换到以下一排。另外，默认情况下，FlowLayout 是从容器的中间位置开始从左往右排的。

注意：如果窗口的大小发生变化，就会影响到 FlowLayout 的排列方式。例如，本来是一排，就可能变成多排了。

不同于 BorderLayout，FlowLayout 布局管理器不要求指定方位，只需要直接使用 add()方法添加组件即可，示例代码如下：

```
package ch06;
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
public class HelloFlowLayout {
    public static void main(String[] args) {                    //主方法
        JFrame jf=new JFrame("Hello FlowLayout");               //创建 JFrame 对象
        jf.setLayout(new FlowLayout());                         //设置布局管理器
        for(int i=0;i<10;i++){
            jf.add(new JButton(i+""));                          //添加 10 个按钮
        }
        jf.setSize(640,200);                                    //分别代表长度和高度
        jf.setVisible(true);                                    //可见，默认为 false
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                                                                //默认为隐藏而不停止程序
```

```
        }
}
```

以上的代码，主要就是为一个采用 FlowLayout 布局管理的 JFrame 对象添加 10 个按钮，这些按钮一次排列。因为 JFrame 的宽度比较大，所以这些按钮用一排来排列就可以了，如图 6.8 所示。

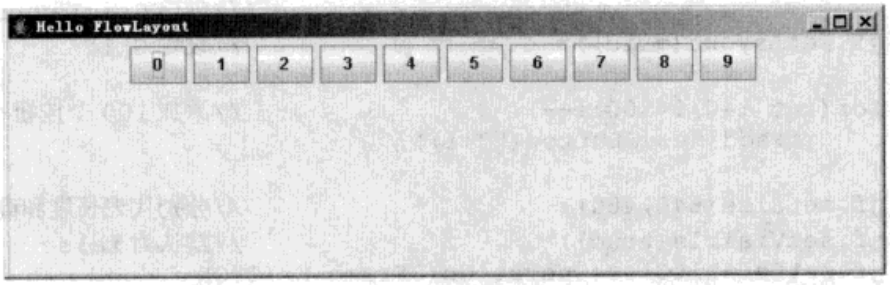


图 6.8　FlowLayout 布局管理器示例的效果图

但是，如果改变这个窗口宽度，则会出现按钮排列不相同的情况。例如，拖动鼠标改变窗框的宽度，效果如图 6.9 所示。

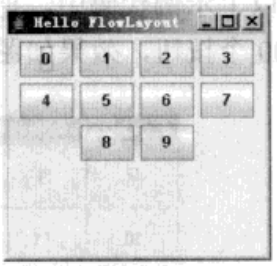


图 6.9　FlowLayout 布局管理器示例的效果图

【答案】

FlowLayout 布局器采用一种流线型的布局方式，把组件从左到右依次地放置，如果摆放不下，则换到下一行继续摆放。

当某种容器需要使用该布局管理器时，只需要调用 setLayout()方法，传入一个 FlowLayout 对象即可。在加入组件的时候，直接调用 add()方法，不需要指定其他的参数。

面试题 065　如何使用 GridLayout 布局

GridLayout 是常用的格子式的布局方式。它的布局方式与网页的表格类似，固定的 N 行 M 列。那么，它应该如何使用呢？本例在回答该问题的同时，详细地讲解 Swing 关于 GridLayout 布局管理器的设计思想和使用方法。

【出现频率】 ★★★★

【关键考点】

- Swing 的布局管理器的含义；
- GridLayout 布局管理器的含义；
- GridLayout 布局管理器的使用方法。

【考题分析】

GridLayout 是一种表格式的布局管理器，它把组件按照表格的方式来存放，从第一行到最后一行，从第一列到最后一列的摆放加入的组件。GridLayout 布局会填满整个容器（例如，一个窗口），当容器变大的时候，表格也会随之变大。GridLayout 布局管理器对象在创建的时候，需要指定它的行数和列数，这就是表格的大小，例如下面的代码：

```
package ch06;
import java.awt.GridLayout;
```

```
import javax.swing.JButton;
import javax.swing.JFrame;
//主类
public class HelloGridLayout {
    public static void main(String[] args) {              //主方法
        JFrame jf=new JFrame("GridLayout test");          //创建 JFrame 对象
        GridLayout grid = new GridLayout(10,10);
                                                  //创建 GridLayout，10 行 10 列
        jf.setLayout(grid);                       //设置 Grid

        for(int i=0;i<100;i++){                   //添加 100 个按钮
            jf.add(new JButton(""+i));
        }
        jf.setSize(640,480);                      //分别代表长度和高度
        jf.setVisible(true);                      //默认为 false
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                                                  //默认为隐藏而不停止程序
    }
}
```

以上示例代码中，把 JFrame 窗口变成一个 10×10 的表格，然后向这 100 个格子中放置 100 个按钮，效果如图 6.10 所示。

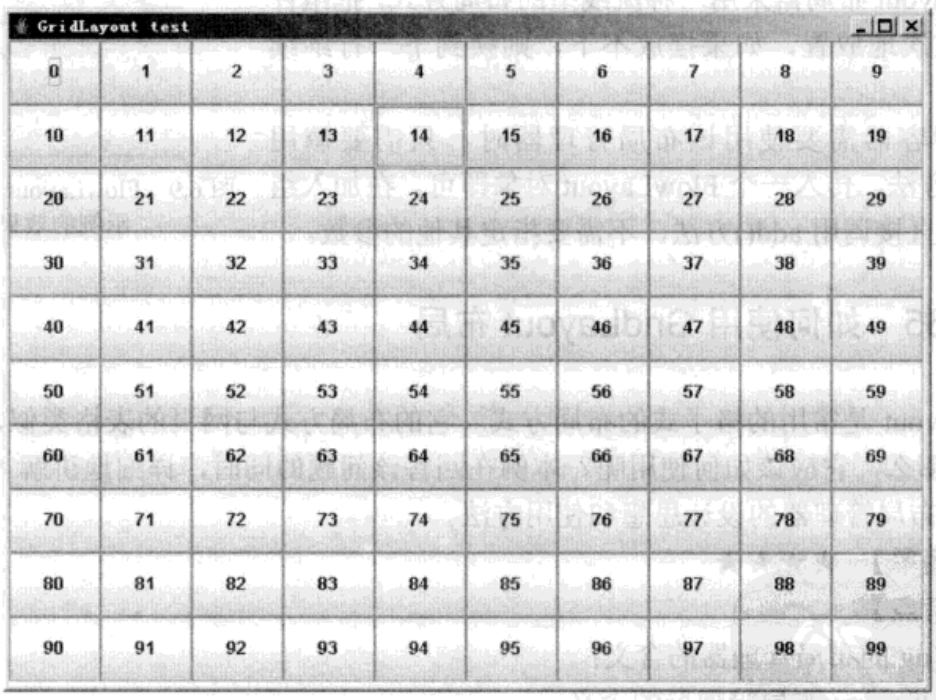


图 6.10　GridLayout 布局管理器示例的效果图

说明：当窗口变大的时候，每个格子的面积也会增大，按钮也会随着变大。

【答案】

GridLayout 布局管理器采用的是一种表格式的布局方式，它把容器等分为 N×M 个格子，然后把加入的组件依次放置在格子中。

当某种容器需要使用该布局管理器时，只需要调用 setLayout()方法，传入一个

GridLayout 对象即可，只不过 GridLayout 对象在创建的时候，需要指定它的行数和列数。在加入组件时，直接调用 add()方法，不需要指定其他的参数。

6.3　事 件 模 型

很多 GUI 开发技术都有事件模型，但各不相同。Swing 的事件模型设计是比较合理的，使用起来也比较方便，包含了一些比较通用的规则。本节将集中讨论有关 Swing 事件模型的常见面试题。

面试题 066　Swing 事件模型的通用规则是什么

Swing 的组件数量很多，每种组件都有自己特定的事件类型。如果把它们都全部记住，是不太现实的。那么，这些事件的定义和使用是否有什么通用的规则呢？本例在回答该问题的同时，详细地讲解 Swing 关于事件模型的使用思想。

【出现频率】 ★★★★★

【关键考点】

- Swing 的组件模型；
- Swing 的事件模型的设计思想；
- Swing 的事件模型的使用规则。

【考题分析】

在 Swing 的事件模型中，组件可以触发事件，每种事件的类型由不同的类来表示。当事件被触发时，它可以被一个或多个监听器所接收，并由监听器来负责处理。所以，事件发生的地方和处理的地方是分开的，这也就是 Swing 事件模型最大的特点。

说明：事件模型是 Java 的分离接口与实现的典型例子。

所谓事件监听器，就是一个实现了某种或某些监听接口的类实例。因此，开发者需要做的主要就是定义一个实现了监听接口的类，并创建具体的实例，再通过组件的 addXXXListener()方法注册监听器，这里的 XXX 指的是监听器的事件类型。通过该方法名就可以很容易知道它注册的是哪种事件的监听器，如果开发者把监听的事件类型搞错了的话，是无法通过编译的。

至于监听器的接口方法，它的方法参数一般都是事件类型对象 XXXEvent，这里的“XXX”也就代表了事件类型。另外，因为大多数的事件处理方法都会使用到组件本身的一些内容，所以可以创建一个全局的监听器类，或者使用内部类。

总结一下，所有的 Swing 组件都有 addXXXlistener()和 removeXXXlistener()方法。它们用于注册和取消注册事件监听器，它们的参数类型都是 XXXListener 接口的实现类，XXXListener 接口的方法参数类型就是 XXXEvent。这就是 Swing 事件模型的通用规则，开发者记住了这些规则以后，就可以游刃有余的使用 Swing 的事件模型了。表 6.2 列出了一些常见事件的监听器接口、事件类型、添加事件注册和取消事件注册方法。

表 6.2　常见事件的监听器、事件类型、添加和取消事件注册方法

监听器、事件	添加和取消方法	支持该事件的常见组件
ActionEvent ActionListener	addActionListener() removeActionListener()	JButton、JList、JMenu、JMenuItem
KeyEvent KeyListener	addKeyListener() removeKeyListener()	几乎所有的组件
MouseEvent MouseListener	addMouseListener() removeMouseListener()	几乎所有的组件
WindowEvent WindowListener	addWindowListener() removeWindowListener()	Window 及其子类，JDialog、JFrame
TextEvent TextListener	addTextListener() removeTextListener()	文本输入组件，JTextField、JTextArea

【答案】

尽管 Swing 的事件类型比较多，但是它们都有一个通用的使用和定义规则，主要有以下几点。

（1）组件都有 addXXXlistener()和 removeXXXlistener()方法，XXX 就代表了事件的类型和含义。

（2）XXX 事件的监听器类型叫做 XXXListener。

（3）XXX 事件的类名叫做 XXXEvent，它往往作为 XXXListener 接口方法中的参数类型。

面试题 067　监听器的适配器的作用是什么

监听器是实现了某种监听接口的类实例。这些接口中定义的方法可能不止一个，这些方法可能代表了几种不同类型的事件。如果开发者只需要实现一个方法，而其他方法根本不用去实现，这怎么办呢？适配器就是帮助开发者解决这个问题的。本例在回答该问题的同时，详细地讲解 Swing 关于监听器的适配器的使用思想。

【出现频率】 ★★★★

【关键考点】

- Swing 的事件模型的使用思想；
- 监听器的适配器的含义和使用方法。

【考题分析】

大家知道，为一个组件添加事件监听器就是加入一个监听器接口的类实例，调用 addXXXListener()方法即可。但是，如果这个接口的方法太多，而开发者只使用其中一些，这岂不是浪费，有接口侵入之嫌，代码如下：

```
package ch06;
import java.awt.FlowLayout;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import javax.swing.JFrame;
import javax.swing.JTextField;
//主类
```

```
public class HelloKeyEvent {
    public static void main(String[] args) {                    //主方法
        JFrame jf = new JFrame("hello events");                  //创建 JFrame 对象
        jf.setLayout(new FlowLayout());                          //设置布局管理器
        JTextField jtf = new JTextField(15);                     //创建文本输入框对象
        jf.add(jtf);                                        //把输入框添加到 Jframe 中
        jtf.addKeyListener(new KeyListener(){                    //添加监听器
            public void keyPressed(KeyEvent e) {                 //事件 1
                System.out.println(e.getKeyChar()+" pressed"); //实现
            }
            public void keyReleased(KeyEvent e) {                //事件 1
                //空实现
            }
            public void keyTyped(KeyEvent e) {                   //事件 1
                //空实现
            }
        });
        showMe(jf);                                              //展示
    }
    private static void showMe(JFrame jf){
        //设置窗口在关闭时，程序的响应
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setSize(200, 200);                                    //设置窗口大小
        jf.setVisible(true);                                     //是否可见
    }
}
```

以上示例代码中，为一个文本输入框 JTextField 对象添加了一个 KeyListener 的监听器，它的方法有 3 个，keyPressed、keyReleased 和 keyTyped。但是，开发者只使用了 KeyPressed，其他都只是空实现。那么，有没有一个类提供这些接口的基本实现的呢？这就是适配器，例如 KeyListener 的适配器就是 KeyAdapter。如果把以上示例代码用适配器重写的话，可以这样来完成，具体如下所示：

```
package ch06;
import java.awt.FlowLayout;
import java.awt.event.KeyEvent;
import java.awt.event.KeyAdapter;
import javax.swing.JFrame;
import javax.swing.JTextField;
public class HelloKeyEvent2 {
    public static void main(String[] args) {                    //主方法
        JFrame jf = new JFrame("hello events");                  //创建 JFrame 对象
        jf.setLayout(new FlowLayout());                          //设置布局管理器
        JTextField jtf = new JTextField(15);                     //创建文本输入框对象
        jf.add(jtf);                                        //把输入框添加到 Jframe 中
        jtf.addKeyListener(new KeyAdapter(){                     //添加监听器
            public void keyPressed(KeyEvent e) {                 //事件 1
                System.out.println(e.getKeyChar()+" pressed"); //实现
            }
        });
        showMe(jf);                                              //展示
    }
    private static void showMe(JFrame jf){
        //设置窗口在关闭时，程序的响应
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
        jf.setSize(200, 200);                                    //设置窗口大小
        jf.setVisible(true);                                     //是否可见
    }
}
```

很显然，使用适配器就可以大大减少冗余代码，让代码更加简洁。这也就是适配器最主要的用途。

说明： 适配器是一个类，如果事件处理类已经继承自其他接口的话，就不能继承自适配器类了。另外，开发者也可以自己写一个适配器，为一些事件方法提供一些自定义的初始化实现。例如，每次事件的触发就记录一次日志。

【答案】

在 Java 开发领域，有一种开发模式叫做适配器开发模式，Swing 的适配器正是采用这种模式开发的。为了避免代码的冗余，Swing 监听器的适配器就是为一些监听器接口的方法提供默认的空实现，这样开发者就可以直接继承自适配器就好，不必把每一个接口方法都实现。另外，读者也可以根据需要自定义一些适配器。

6.4　Swing 编程应用

因为 Java 图形界面开发具有很强的实践性，所以直接上机编程也是考察 Swing 开发的一种手段。本节就几类比较常见的 Swing 上机开发面试题进行讲解。

面试题 068　用 JButton 开发扫雷游戏

扫雷游戏是一款比较经典的小游戏。如果使用 Swing 的 JButton 来开发，那么应该如何组织按钮的排列呢？如何控制按钮之间的关系呢？本例在回答该问题的同时，详细地讲解扫雷游戏的设计和开发过程，加深读者对 JButton 等组件的认识。

【出现频率】 ★★★★

【关键考点】

- JButton 组件的使用方法；
- 开发实践。

【考题分析】

扫雷游戏的原理是非常简单的，就是一个平面上有很多隐藏的按钮，若干个雷随机分布在这些按钮下边，玩家通过提示的数字来标示出这些雷。如果不小心点错了就结束游戏，否则一直持续到所有的雷都被标出。

首先来考虑一下数据的问题。这些按钮肯定是一个由若干行和若干列的按钮数组来创建的，每个按钮都有一个数据，因此用一个 int 型的二维数组来存储按钮的数字信息是比较恰当的。另外，哪些按钮是雷，哪些不是，这肯定是需要进行随机抽取和确定的。因此，在游戏开始之前，就需要有一个单独的方法来完成数据的随机确定。然后就是过程，玩家主要操作的地方有以下几个方面：

（1）开始/重新开始。这里需要做的就是初始化数据。

（2）鼠标左键翻开按钮。这里就得监听按钮被按下的事件了。

（3）鼠标右键翻开按钮。这里也要监听按钮被按下的事件，但是还需要判断玩家是否翻开了雷而引起游戏结束。

最后，当游戏结束时，可能是全部找出，也可能玩家踩到地雷了，都应该做出相应的响应。如果玩家翻到雷了，则把其他雷的分布显示出来；若雷被全部标出则游戏结束，打印一句恭喜的话。

以上是扫雷游戏本身的逻辑。那么就 Swing 编程来说，它主要包含了哪些知识点呢？主要有以下几点：

（1）按钮的创建和使用。

（2）分布的问题，很显然这里需要使用 GridLayout 布局管理器。

（3）事件的捕捉。玩家主要是鼠标左键和右键的操作，还需要判断玩家单击的是左键还是右键。

（4）窗口的显示和渲染。

说明：使用按钮来实现扫雷游戏只是一种解决方案，也可以使用其他的组件，例如普通的图片或绘图也可以。

【答案】

以下代码是一个根据上述分析完成的参考示例：

```
package ch06;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class Mine extends MouseAdapter{                    //主类
    private JFrame mainFrame;                              //窗口对象
    private int[][] data;                                  //数据二维数组
    private JButton[][] buttons;                           //按钮二维数组
    private JButton startJB;                               //开始按钮
    private Label l;                                       //标签
    private int row;                                       //行数
    private int col;                                       //列数
    private int mineNumber;                                //雷的数量
    private int mineCount;                                 //当前雷被找出的数量
    private boolean isOver;                                //游戏是否结束
    public Mine() {                                        //构造方法
        row = 15;                                          //15 行，15 列
        col = 15;
        mainFrame = new JFrame("扫雷 v1.0");               //创建窗口对象
        data = new int[row][col];                          //定义数组数据
        buttons = new JButton[row][col];                   //定义按钮数组数据
        startJB = new JButton("start");                    //创建开始按钮
        l = new Label("welcome to mine!");                 //初始化标签
        mineNumber = row * col / 7;                        //雷的数量是总格子的 1/7
    }
    public void init() {                                   //初始化方法
        JPanel north = new JPanel();
        JPanel center = new JPanel();
```

```
        JPanel south = new JPanel();
        north.setLayout(new FlowLayout());          //布局
        center.setLayout(new FlowLayout());
        south.setLayout(new GridLayout(row, col, 4, 4));//雷区采用 Grid 布局
        mainFrame.setLayout(new BorderLayout());
        mainFrame.add(north, BorderLayout.NORTH);  //分别添加面板
        mainFrame.add(center, BorderLayout.CENTER);
        mainFrame.add(south, BorderLayout.SOUTH);
        north.add(l);                               //添加标签
        startJB.addActionListener(new ActionListener() {//开始按钮的监听器
            public void actionPerformed(ActionEvent e) {
                //一切初始化后从新开始
                for (int i = 0; i < row; i++) {
                    for (int j = 0; j < col; j++) {
                        buttons[i][j].setText(" ");
                        buttons[i][j].setBackground(Color.WHITE);
                        data[i][j] = 0;
                        isOver = false;
                    }
                }
                hashMine();
                mineCount = 0;
                l.setText("let's go!");
            }
        });
        center.add(startJB);
        for (int i = 0; i < row; i++) {                 //初始化按钮信息
            for (int j = 0; j < col; j++) {
                buttons[i][j] = new JButton(" ");
                buttons[i][j].setName((i + ":" + j));
                                                    //按钮的名字记录着行号和列号
                // buttons[i][j].setSize(10, 30);
                buttons[i][j].setBackground(Color.WHITE);
                buttons[i][j].addMouseListener(this);
                south.add(buttons[i][j]);
            }
        }
        hashMine();                                 //随机布局类
    }
    public void start(){
        mainFrame.setSize(800, 600);
        mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        mainFrame.setVisible(true);
    }
    public void hashMine() {                        //随机布雷的方法
        //得到随机数，确定类的位置
        for (int i = 0; i < mineNumber; i++) {
            data[(int) (Math.random() * row)][(int) (Math.random() * col)]
            = -1;
        }
        for (int i = 0; i < row; i++) {
            for (int j = 0; j < col; j++) {
                if (data[i][j] == -1)
                    continue;
                int sum = 0;                        //确定每个格子周围雷的数量
                for (int m = -1; m <= 1; m++) {
                    for (int n = -1; n <= 1; n++) {
                        if (i + m >= 0 && j + n >= 0 && i + m < row && j +
                        n < col) {
```

```
                        if (data[i + m][j + n] == -1)
                            sum++;
                    }
                }
            }
            data[i][j] = sum;
        }
    }
}
private void gameOver(boolean over) {                    //游戏结束
    if (over == true) {                                  //雷被误点以后
        for (int i = 0; i < row; i++) {                  //把所有的类都显示出来
            for (int j = 0; j < col; j++) {
                if (data[i][j] == -1) {
                    buttons[i][j].setText("M");
                    buttons[i][j].setBackground(Color.RED);
                }
            }
        }
        l.setText("-_-");
        isOver = true;
        return;
    }
    int sumPress = 0;                                    //统计被着色的按钮总数
    for (int i = 0; i < row; i++) {
        for (int j = 0; j < col; j++) {
            if (!buttons[i][j].getText().equals(" ")) {
                sumPress++;
            }
        }
    }
    if (sumPress == row * col) {
        int sum = 0;                                     //统计被找出的雷总数
        for (int i = 0; i < row; i++) {
            for (int j = 0; j < col; j++) {
                if (data[i][j] == -1 && buttons[i][j].getText().
                equals("M")) {
                    sum++;
                }
            }
        }
        if (sum >= mineNumber) {
            System.out.println(mineNumber);
            l.setText("^_^");                                //游戏胜利
        }
    }
}
public void mousePressed(MouseEvent e) {
                            // 实现 mouselistner 中的方法，监听鼠标单击事件
    try {
        if (isOver)
            return;
        if (e.getButton() == MouseEvent.BUTTON3) { //右击
            JButton jb = (JButton) e.getSource();  //获得被点的按钮对象
            if (jb.getText().equals("M")) {        //取消标志
                jb.setText(" ");
                mineCount--;
                jb.setBackground(Color.WHITE);
            } else {
```

```
                    if (mineCount < mineNumber) {          //标志还没有用完
                        jb.setText("M");
                        jb.setBackground(Color.BLUE);
                        mineCount++;
                    } else {                               //标志已经用完
                        l.setText("the mine flag is over!");
                    }
                }
            } else {
                JButton jb = (JButton) e.getSource();
                mousePress(jb);//调用 mousePress()方法处理鼠标左键被单击的事件
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        gameOver(false);//主要用于判断游戏是否已经结束，也就是雷是否被找完
    }
    private void mousePress(JButton jb) {
        String str[] = jb.getName().split(":");            //得到行号和列号
        int i = Integer.parseInt(str[0]);
        int j = Integer.parseInt(str[1]);
        if (data[i][j] == -1) {                            //如果点到雷了
            gameOver(true);
            return;
        } else {
            jb.setText(data[i][j] + "");                   //设置文本和颜色
            jb.setBackground(Color.YELLOW);
            if (data[i][j] == 0) {                         //如果四周都没有雷
                for (int m = -1; m <= 1; m++) {
                    for (int n = -1; n <= 1; n++) {
                        if (i + m >= 0 && j + n >= 0 && i + m < row && j +
                        n < col) {
                            if (buttons[i + m][j + n].getText().equals(" "))
                                mousePress(buttons[i + m][j + n]);
                                                           //撬开旁边的按钮
                        }
                    }
                }
            }
        }
    }
    public static void main(String[] args) {               //主方法
        Mine mine = new Mine();
        mine.init();
        mine.start();
    }
}
```

以上代码运行效果，如图 6.11 所示。

面试题 069　用 JTextField 和 JButton 开发计算器程序

计算器程序可以用很多技术来实现，它的原理也是非常的简单。那么如何使用 Swing 的 JTextField 和 JButton 来开发计算器程序呢？本例在回答该问题的同时，详细地讲解计算器程序的设计和开发过程，加深读者对 JTextField 和 JButton 等组件的认识。

【出现频率】 ★★★★

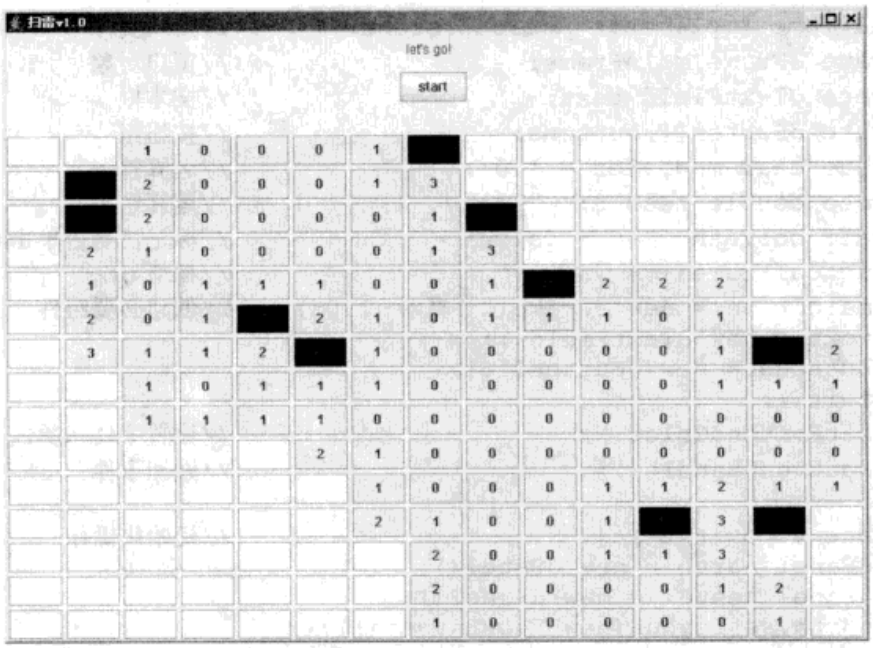


图 6.11　扫雷游戏示例效果图

【关键考点】

- JTextField 组件的使用方法；
- JButton 组件的使用方法；
- 开发实践。

【考题分析】

相信大家对计算器并不陌生，它主要由两个部分组成：数字显示和按钮组。用户通过点击数字和操作符来完成一些算术运算，并在文本框中实现结果。

关于计算器本身的逻辑原理，大家都熟悉了。那么对 Swing 编程来说，它主要有以下几点：

- 按钮的创建和使用。
- 分布的问题，很显然这里需要使用 GridLayout 布局管理器。
- 事件的捕捉。玩家主要是鼠标左键的操作，并且特定的按钮包含了特定的含义。
- 窗口的显示和渲染。

说明：为了简单起见，计算器只需要实现加、减、乘、除运算即可。

【答案】

以下代码是一个根据上述分析完成的参考示例。

```
package ch06;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class Calculator {
    public static void main(String[] args) {    //主方法
        CalFrame calculator = new CalFrame();
        calculator.show();                       //打开程序
    }
}
```

```
class CalFrame implements ActionListener {          //窗口类
    private JFrame mainFrame;                       //窗口对象
    private JTextField text;                        //文本框
    private JButton[] buttons;                      //按钮组
    private char modifier = '\0';                   //运算符
    private double result;                          //运算结果
    private boolean flag = false;                   //标记，是否单击了运算符按钮
    public CalFrame() {                             //构造方法
        mainFrame = new JFrame("计算器 v1.0");      //创建必要组件
        text = new JTextField();
        buttons = new JButton[16];
        init();
        setFontAndColor();                          //设置字体和颜色
        addEventHandle();                           //添加事件
    }
    private void init() {                           //初始化操作
        JPanel north = new JPanel();
        JPanel center = new JPanel();
        north.setLayout(new FlowLayout());
        center.setLayout(new GridLayout(4, 4, 2, 2));  // 布局
        text = new JTextField(25);
        north.add(text);
        String str = "123+456-789*0.=/";            //用字符串来表示所有按钮的字符
        for (int i = 0; i < 16; i++) {
            JButton jb = new JButton(String.valueOf(str.charAt(i)));
            buttons[i] = jb;
            center.add(jb);
        }
        mainFrame.add(north, BorderLayout.NORTH);
        mainFrame.add(center, BorderLayout.CENTER);
    }
    public void show() {                            //显示窗口
        mainFrame.pack();
        mainFrame.setVisible(true);
        mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    private void setFontAndColor() {                //设置字体和颜色
        Font f = new Font("黑体", Font.BOLD, 20);
        text.setFont(f);
        for (int i = 0; i < buttons.length; i++) {
            buttons[i].setFont(f);
            buttons[i].setForeground(Color.RED);
        }
    }
    public void addEventHandle() {
        for (int i = 0; i < buttons.length; i++) {
            buttons[i].addActionListener(this); //为每个按钮添加监听器对象
        }
    }
    //本类实现 ActionListener 接口，即可访问本对象的全部属性
    public void actionPerformed(ActionEvent e) {
        String str = e.getActionCommand();          //获得按钮上的文本)
        if ("0123456789.".indexOf(str) != -1) {     //输入的是数字
            if (flag) {                             //如果已经单击了数字按钮
                text.setText("");                   //文本框为空
                flag = false;                       //恢复 flag
            }
```

```
            text.setText(text.getText() + str);      //设置字符
        } else if ("+-*/".indexOf(str) != -1) {      //单击运算符按钮
            modifier = str.charAt(0);
            double num = Double.valueOf(text.getText());
            result = num;
            flag = true;// flag 为 true
        } else if (str.charAt(0) == '=') {           //单击运算符按钮
            if (modifier == '+') {                   //加法运算
                double num = Double.valueOf(text.getText());
                result += num;
            }
            if (modifier == '-') {                   //减法运算
                double num = Double.valueOf(text.getText());
                result -= num;
            }
            if (modifier == '*') {                   //乘法运算
                double num = Double.valueOf(text.getText());
                result *= num;
            }
            if (modifier == '/') {                   //除法运算
                double num = Double.valueOf(text.getText());
                result /= num;
            }
            text.setText(result + "");
            modifier = '\0';
            result = 0;
        }
    }
}
```

效果如图 6.12 所示。

图 6.12　计算器示例效果图

面试题 070　用 JTextArea 开发俄罗斯方块游戏

俄罗斯游戏是一款非常经典的小游戏，它在发展过程中，涌现出了许多种的版本。它的编程难度比较大，所有的方块都需要自己绘制。那么使用 Swing 的 JTextArea，应该如何组织这些方块的排列呢？如何控制方块之间的关系呢？本例在回答该问题的同时，详细地讲解俄罗斯方块游戏的设计和开发过程，加深读者对 JTextArea 等组件的认识。

【出现频率】 ★★★★

【关键考点】

- JTextArea 组件的使用方法；

- 开发实践。

【考题分析】

俄罗斯方块游戏的逻辑是比较简单的。它就类似于堆砌房子一样，各种各样的方块的形状是不同的。但是，俄罗斯方块游戏的界面被等均的分为若干行和若干列，因此方块的本质就是占用了多少个单元。

首先来考虑一下数据的问题。对于界面来说，需要一个二维的 int 型数组，它保存着那些地方应该有着色，哪些没有；然后是方块本身，尽管它们的形状不统一，但是它们都可以用一个 4×4 比例的方块所包围，因此用 16 个字节就可以把一个方块的信息保存着。

注意： 其实方块的数据也可以用 int 数组表示，但是涉及到效率问题，用位操作比用普通的算术运算要快得多。

接下来思考一下动作具体有下面几点：

（1）方块的诞生。它的诞生是需要用随机原理的，另外，它如何初始化的被放置在游戏界面的顶部？

（2）方块是需要自动的往下掉的，它在掉的过程中，还需要判断它是否与周围的环境是否发生了冲突，能不能继续往下。

（3）方块本身还可以变形，变形以后的方块具有不同的数据，判断的方式又会不一样。

（4）当用户一直按住 Down 键的时候，方块还需要持续往下掉。

然后就是过程，玩家主要操作的地方有以下几个方面：

- 左/右操作。需要监听 KeyEvent，让方块左右移动，直到碰到边界。
- 变形操作。也要监听 KeyEvent，让方块自动的变形。
- 下降操作。也要监听 KeyEvent，让方块快速的下降。

至于游戏的结束，只有一种情况，那就是诞生的方块一出世就与其他方块冲突了。因此，游戏的结束还是很好判断的。

以上是俄罗斯方块游戏本身的逻辑，那么对 Swing 编程来说，它主要有以下几点：

- JTextArea 的创建和使用。
- 分布的问题，很显然这里需要使用 GridLayout 布局管理器。
- 事件的捕捉。玩家主要是键盘按键的操作，还需要判断玩家单击的是上、下、左或右键。
- 窗口的显示和渲染。

【答案】

以下代码是一个根据上述分析完成的参考示例。

```
package ch06;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class Travil extends JFrame implements KeyListener{ //游戏类
    private JTextArea[][] grids;          //存放各自的文本域
    private int data[][];                 //对应每个格子的数据，1 代表有，0 代表无
    private int rect;                     //当前下落的方块类型，对应 all 中的数据
    private int[] allRect;                //所有的方块类型，由 16 个字节来记录
```

```
private int x,y;                          //当前方块的坐标位置，x 代表行，y 代表列
private int score;                        //得分，每消一层得 1 分
private JLabel label;                     //显示分数的标签
private boolean running;                  //游戏是否结束
public Travil(){
    grids=new JTextArea[21][12];          //最左边、最右边和最下边作为游戏的边框
    data=new int[21][12];                 //与格子对应的数据
    //所有的方块类型，由 16 个字节来记录方块占用格子的情况
    allRect=new int[]{0xcc,0x8888,0xf,0xc44,0x2e,0x88c,0xe8,0xc88,
    0xe2,0x44c,0x8e,0x8c4,0x6c,0x4c8,0xc6,0x8c8,0x4e,0x4c4,0xe4};
    label=new JLabel("socre:0");
    running=false;
    init();                               //初始化操作
}
public void init(){
    JPanel main=new JPanel();             //主面板
    JPanel right=new JPanel();            //说明信息面板
    main.setLayout(new GridLayout(21,12,1,1)); //Grid 布局
    for(int i=0;i<grids.length;i++){                //初始化每个格子
        for(int j=0;j<grids[i].length;j++){
            grids[i][j]=new JTextArea(20,20);
            grids[i][j].setBackground(Color.WHITE);
            if(j==0 || j==grids[i].length-1 || i==grids.length-1){
                grids[i][j].setBackground(Color.GREEN);
                grids[i][j].addKeyListener(this);
                data[i][j]=1;             //1 代表有需要显示的方块
            }
            grids[i][j].setEditable(false);
            main.add(grids[i][j]);
        }
    }
    right.setLayout(new GridLayout(4,4,1,1));
                                          //设置右边的说明文字和分数记录
    right.add(new JLabel(" a : left"));   //添加标签 a
    right.add(new JLabel(" d : right"));  //添加标签 b
    right.add(new JLabel(" s : down"));   //添加标签 c
    right.add(new JLabel(" w : change")); //添加标签 d
    right.add(new JLabel(""));
    right.add(new JLabel(""));
    right.add(label);
    this.setLayout(new BorderLayout());
    this.add(main,BorderLayout.CENTER);
    this.add(right,BorderLayout.EAST);
    running=true;                         //正在运行
}
public void ranRect(){                    //随即产生方块类型
    rect=allRect[(short)(Math.random()*19)];
}
public void keyPressed(KeyEvent e) {}
public void keyReleased(KeyEvent e) {}
public void keyTyped(KeyEvent e) {  //捕获按键进行左、右、下落、变形的操作
    if(e.getKeyChar()=='a'){              //左移操作
        if(running==false)
            return;
        if(y<=1)
            return;
```

```
            int tmp=0x8000;
            for(int i=x;i<x+4;i++){                    //2 次循环判断是否碰壁
                for(int j=y;j<y+4;j++){
                    if( (rect&tmp) != 0){
                        if(data[i][j-1]==1)
                            return;                    //碰壁
                    }
                    tmp>>=1;                           //位置移动 1 位
                }
            }
            clear(x,y);                                //清除图像
            y--;
            draw(x,y);                                 //重绘图像
        }
        if(e.getKeyChar()=='d'){                       //右移操作
            if(running==false)
                return;
            int tmp=0x8000;
            int num=7;
            int m=x,n=y;
            for(int i=0;i<4;i++){                      //两次循环判断是否碰壁
                for(int j=0;j<4;j++){
                    if( (tmp&rect) !=0){
                        if(n>num)
                            num=n;
                    }
                    tmp>>=1;                           //移动 1 位
                    n++;
                }
                m++;
                n-=4;
            }
            if(num>=10)
                return;
            tmp=0x8000;
            for(int i=x;i<x+4;i++){                    //判断
                for(int j=y;j<y+4;j++){
                    if( (rect&tmp) != 0){
                        if(data[i][j+1]==1)
                            return;                    //碰壁，返回
                    }
                    tmp>>=1;                           //移动一位
                }
            }
            clear(x,y);                                //清除图像
            y++;                                       //坐标加 1
            draw(x,y);                                 //绘制图像
        }
        if(e.getKeyChar()=='s'){                       //下落操作
            if(running==false)
                return;
            if(canFall(x,y)==false){                   //判断是否还可以继续下降
                saveData(x,y);                         //保存此时的数据
                return;                                //返回
            }

            clear(x,y);                                //清除图像
            x++;                                       //坐标加 1
```

```
            draw(x,y);                                    //绘制图像
        }
        if(e.getKeyChar()=='w'){                          //变形操作
            if(running==false)
                return;
            int i=0;                                      //用于记录它转动的次数
            for(;i<allRect.length;i++){ //如果是石头，则直接返回，它无法变形
                if(rect==allRect[i])
                    break;
            }
            if(i==0)
                return;
            clear(x,y);                                   //清除图像
            if(i==1 || i==2){
                rect=allRect[i==1?2:1];                   //图形类型 1
                if(y>7)
                    y=7;
            }
            if(i>=3 && i<=6){
                rect=allRect[i+1>6?3:i+1];                //图形类型 2
            }
            if(i>=7 && i<=10){
                rect=allRect[i+1>10?7:i+1];               //图形类型 2
            }
            if(i==11 || i==12){
                rect=allRect[i==11?12:11];                //图形类型 2
            }
            if(i==13 || i==14){
                rect=allRect[i==13?14:13];                //图形类型 2
            }
            if(i>=15 && i<=18){
                rect=allRect[i+1>18?15:i+1];              //图形类型 2
            }
            if(y>8)
                y=8;
            draw(x,y);                                    //绘制图像
        }
    }
    public void fall(int m,int n){                        //方块下落一格
        if(m>0)
            clear(m-1,n);
        draw(m,n);                                        //绘制图像
    }
    public void draw(int m,int n){                        //在 m，n 坐标下画出方块
        int tmp=0x8000;
        for(int j=0;j<4;j++){                             //根据方块的数据来绘制图形
            for(int k=0;k<4;k++){
                if( (tmp&rect) !=0 ){
                    //变成黑色
                    grids[m][n].setBackground(Color.BLACK);
                }
                tmp>>=1;
                n++;
            }
            m++;                                          //下一列
            n-=4;;                                        //下一行
        }
    }
```

```
public void clear(int m,int n){                    //在 m，n 坐标下清除方块
    int tmp=0x8000;
    for(int j=0;j<4;j++){                          //根据方块数据清除方块图形
        for(int k=0;k<4;k++){
            if( (tmp&rect) !=0 ){
                //变成白色
                grids[m][n].setBackground(Color.WHITE);
            }
            tmp>>=1;
            n++;
        }
        m++;                                       //下一列
        n-=4;;                                     //下一行
    }
}
public boolean canFall(int m,int n){               //判断是否还可以继续下落
    int tmp=0x8000;
    for(int i=0;i<4;i++){
        for(int j=0;j<4;j++){
            if( (tmp&rect) !=0){
                if(data[m+1][n]==1) //如果下一个地方有东西则直接返回 false
                    return false;
            }
            tmp>>=1;                               //再移动一位
            n++;
        }
        m++;                                       //下一列
        n-=4;;                                     //下一行
    }
    return true;
}
public void saveData(int m,int n){ //将 m，n 坐标下的方块保存到 data 中
    int tmp=0x8000;
    for(int j=0;j<4;j++){
        for(int k=0;k<4;k++){
            if( (tmp&rect) !=0 ){                  //有数据
                data[m][n]=1;                      //保存平面数据
            }
            tmp>>=1;
            n++;
        }
        m++;                                       //下一列
        n-=4;;                                     //下一行
    }
}
public void removeRow(int row){     //删除第 row 行，以上的依次往下降
    for(int i=row;i>=1;i--){
        for(int j=1;j<=10;j++){
            data[i][j]=data[i-1][j];               //去掉数据
        }
    }
    reflesh();                                     //刷新
    score++;                                       //分数加 1
    label.setText("socre:"+score);                 //设置文本
}
public void reflesh(){                             //根据 data 刷新每个格子
    for(int i=1;i<20;i++){
        for(int j=1;j<11;j++){
```

```
                if(data[i][j]==1)                    //有数据，为黑色
                    grids[i][j].setBackground(Color.BLACK);
                else                                 //没数据，为白色
                    grids[i][j].setBackground(Color.WHITE);
            }
        }
    }
    public void start(){                             //让方块开始自动下落
        x=0;                                         //横向从 0 开始
        y=5;                                        //纵向从 5 开始，这里可以随便设置
        for(int i=0;i<21;i++){
            try {
                Thread.sleep(1000);                  //每隔 1 秒，下降一次
                if(canFall(x,y)==false){             //如果可以掉
                    saveData(x,y);                   //保存数据
                    for(int k=x;k<x+4;k++){          //进入第一层循环
                        int sum=0;
                        for(int j=1;j<=10;j++){
                            if(data[k][j]==1)
                                sum++;               //数据加 1
                        }
                        if(sum==10)
                            removeRow(k);            //去掉一行
                    }
                    for(int j=1;j<=10;j++){
                        if(data[3][j]==1){           //如果有数据
                            running=false;           //运行结束
                            break;                   //退出循环
                        }
                    }
                    break;
                }
                x++;                                 //层加 1
                fall(x,y);                           //掉一层下来
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
    public void showMe(){                            //显示整个窗口
        this.setSize(600, 800);                      //窗口大小
        this.setVisible(true);                       //是否可见
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public void go(){                                //开始游戏
        while(true){                                 //永真循环
            if(running==false)                       //判断是否有戏结束
                break;
            ranRect();                               //绘制方格
            start();                                 //开始游戏
        }
        label.setText("game over");                  //游戏结束
    }
    public static void main(String[] args){          //主方法
        Travil t=new Travil();
        t.showMe();
        t.go();                                      //开始
```

```
        }
}
```

运行代码效果如图 6.13 所示。

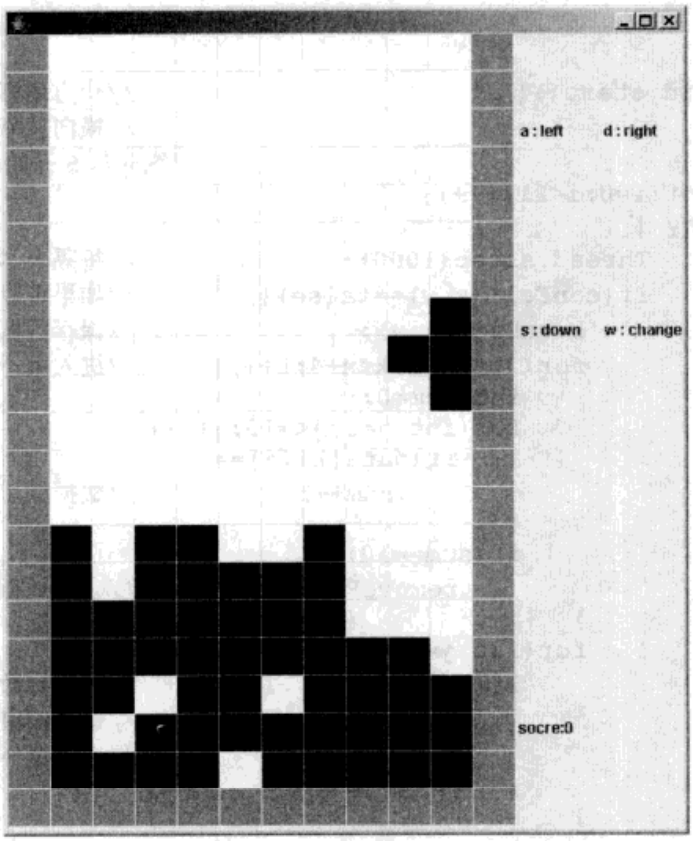


图 6.13　俄罗斯方块游戏示例效果图

6.5　小　　结

本章讲解了一些关于 Java 图形用户界面开发（主要是 Swing）的面试题。首先讲解了 Swing 的基本使用思想，包括 JFrame 的使用、定义按钮、创建文本框、捕获事件等方面。然后是 Swing 的布局种类及其使用方法，包括 BoderLayout、FlowLayout 和 GridLayout，接着讲解了 Swing 的事件模型的使用思想。最后是一些 Swing 编程应用的实践。

Java 图形用户界面开发并不是 Java 开发的强项，但是它属于 Java SE 领域，是学习 Java 过程中的一个重要阶段，通过考察这些知识可以知道求职者的基本 Java 功底和一些开发思想的领会程度。读者在面对这类面试题的时候，仅用记下具体的 API 该如何使用，把其

第3篇 Java高级特性

第7章　输入/输出流

输入与输出是任何计算机程序都需要涉及的问题，Java 程序也不例外。Java 是伴随着网络的发展而发展的，而 I/O（输入与输出）正是网络通信的基础，Java 的 I/O 设计是很合理的，使用起来也非常方便。关于 IO 的 API 是相当的多，不可能全部都记住，也不可能全都会用，关键在于理解 java.io 包下的 API 的几个核心概念。例如，流、字节流、字符流、缓存等，大多数的类或接口都是围着这些概念来进行的。本章将讲解包含关于 Java 的 I/O 的一些常见面试题，并且分析这些题目和知识点，帮助读者梳理这些方面的知识。

7.1　File 类

磁盘是数据的主要存储目的地。磁盘存储数据最基本的单位就是一个文件（File）。Java 提供了关于文件的各种 API，包括：File、FileInputStream、FileOutputString、RandomAccessFile 等，对这些接口和类的熟练使用是许多开发所必须的。

面试题 071　目录和文件操作

目录和文件是磁盘树形结构中最基本的组成单元，Java 用 File 类来统一表示它们，也是通过它来完成目录和文件的创建、删除、查询等操作。本例在回答该问题的同时，详细地讲解关于 File 类的使用方法。

【出现频率】 ★★★★

【关键考点】

- File 类的概念；
- 如何用 File 类来操作目录和文件。

【考题分析】

Java 对待目录和文件都统一使用 File 来表示，并且在创建 File 对象的时候，并不检查该目录或文件是否存在，只作为一种代表，开发者在需要时使用 isDirectory()或 isFile()方法进行判断。以下代码为一个操作目录和文件示例程序。

```
import java.io.File;
import java.io.IOException;
public class FileDirTest {
    public static void main(String[] args) {
        File file1 = new File("D:/test/a.txt");      //创建文件对象 file1
        if(!file1.exists()){                         //判断它是否已经存在
            try {
                file1.createNewFile();               //创建新文件
```

```
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        File dir = new File("D:/test");                    //创建 dir 目录对象
        if(dir.isDirectory()){                             //判断它是否为目录
            String[] files = dir.list();    //调用 list()方法获取它的所有文件
            for(String fileName : files){                  //遍历文件
                //用目录和文件名生成 File 对象
                File f = new File(dir.getPath()+File.separator+fileName);
                //进行分类打印
                if(f.isFile()){
                    System.out.println("file:"+f.getName());
                }else if(f.isDirectory()){
                    System.out.println("dir:"+f.getName());
                }
            }
        }
    }
}
```

本示例程序的功能很简单，首先检查“d:/test”文件夹是否存在文件 a.txt，若不存在则创建一个新文件。然后，将该文件夹下的所有文件和目录遍历以后，分类打印出来。其中，以下是一些 File 类经常会使用的方法。

- 构造方法：一般可以通过传递字符串形式的文件路径的方式来创建 File 对象，其中 File 并不会检查该目录或文件是否已经存在；
- isDirectory()和 isFile()方法：用于检查该 File 对象所代表的是目录还是普通文件；
- createNewFile()方法：创建新文件，采用 File 对象所存储的路径和文件名进行创建；
- list()方法：用于目录，得到目录下所有的文件名，类型为字符串数组；
- getName()方法：得到文件名，不包含它的路径；
- delete()方法：删除文件。

说明：Java 的 I/O 操作一般都需要进行异常检查，大多数时候均是 IOException。

【答案】

Java 提供了 java.io.File 类对目录和文件进行操作。主要的操作方法包括：路径字符串构造方法、isDirectory、isFile、createNewFile、list、getName、delete。

面试题 072　写一个复制文件的程序

本题目主要考察的是面试者对 FileInputStream 和 OutputStream 的使用，本质上也就是先从一个文件读出数据，然后再往另外一个文件写入数据。本例在回答该问题的同时，详细地讲解关于 File 输入/输出流的使用方法。

【出现频率】 ★★★★

【关键考点】

- InputStream 的使用方法；
- OutputStream 的使用方法。

【考题分析】

Java 的所有数据读/写，都是通过流进行的，因此要完成一个复制文件的程序，就得有两个流对象，一个是输入流，另一个是输出流。

其中，FileInputStream 类中最主要是 read()方法，通过它把数据从流中读到内存里来。read()方法的参数为一个 byte 数组，有一点像缓存，也就是把数据线读到内存中暂存，然后再进行相关操作。

而 FileOutputStream 是用于打开一个输出流的，这里就需要把 byte 数组中的数据写入到输入流中。如果每次读出的数据等于 byte 数组的大小，就可以直接把数据写出；如果数据不足 byte 数组那么大，则需要控制写出数据的长度，也就是 read()方法的返回值。

根据以上的思路，该程序具体如下所示：

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
public class FileCopy {
    public static void main(String[] args) throws IOException {
        //生成输入文件的输入流对象
        FileInputStream fin = new FileInputStream("d:/test/a.txt");
        //生成输出文件的输出流对象
        FileOutputStream fout = new FileOutputStream("d:/test/b.txt");
        byte[] buff = new byte[256];          //定义一个永固暂存数据的 byte 数组
        int len = 0;                          //每次读取数据的长度
        //循环读取到 buff 中，如果文件到底，则读取数据的长度小于 0
        while ((len = fin.read(buff)) > 0) {
            fout.write(buff, 0, len);         //把数据一次写进输出文件中
        }
        fin.close();                          //关闭输入流
        fout.close();                         //关闭输出流
    }
}
```

以上的文件路径是用硬代码写的，而实际的程序应该是从命令提示窗口中接收文件路径参数，也就是命令行参数。读者可以把以上的示例代码进行修改，用 main()方法的 args 参数来获取输入和输出文件名。

注意：程序的最后两行代码是关闭输入和输出流。其实，关闭流是一种非常重要的编程习惯。如果不关闭流，就会造成资源的浪费，从而影响程序的性能，还有可能会让文件锁住，其他程序无法操作它。

【答案】

本题目的设计思路可以有以下几个步骤。

（1）用被复制文件的路径创建一个 InputStream 对象。

（2）用复制文件的新路径创建一个 OutputStream 对象。

（3）用 read()方法循环把数据读出到一个 byte 数组中，直到读出数据的长度小于 0。

（4）用 write()方法把 byte 数组中的字节写入输出流。

（5）最后关闭输入流和输出流。

面试题 073　如何使用随机存取文件 RandomAccessFile 类

InputStream 和 OutputStream 都只能单方面的读和写文件的内容，如果有随机的读取和写入文件内容的需要那又该怎么办呢？Java 为开发者提供了一个 RandomAccessFile 类，它就是专门用来随机存取文件内容的。通过回答本例问题，可以了解到 RandomAccessFile 类的常见用法。

【出现频率】 ★★★★

【关键考点】

- RandomAccessFile 的用法

【考题分析】

RandomAccessFile 操作文件内容的时候，就好像操作一块内存区域一样。把字节用下标数字来进行定位，通过调用 RandomAccessFile 的 API 方法，把指针的指向进行移动，达到随机存取数据的目的，主要的方法如下：

```
length()。得到文件内容的字节长度。
seek()。设置到此文件开头测量到的文件指针偏移量，在该位置发生下一个读取或写入操作。
read()和 write()。读取和写入字节数据。
```

以下是将文件中所有字母 a 都换成字母的 c 的示例程序。

```
public class RanAccFileTest {
    public static void main(String[] args) throws IOException {
        //创建随机读取文件对象
        RandomAccessFile file= new RandomAccessFile("d:/test/a.txt", "rw");
        //遍历 file 的字节数据
        for (int i = 0; i < file.length(); i++) {
            byte b = (byte) file.read();          //read()方法读取一个字节
            char c = (char) b;                    //转换为 char 型
            if(c == 'a'){
                file.seek(i);                     //把指针又恢复到之前的地方
                file.write('c');                  //写入新的字符 c
            }
        }
        file.close();                             //关闭文件的打开
    }
}
```

说明： RandomAccessFile 的大多数方法都会抛出 IOException，以上示例程序只是简单进行抛出。而在实际的开发中，应该针对不同的情况，不同的异常做出不同的响应和处理。

【答案】

RandomAccessFile 的使用思路主要包括以下一些项目：

- 用 length()方法获取文件的内容长度。
- 用 seek()方法随机的到达任何需要存取数据的地方。
- 调用 read()方法获取当前位置的数据，用 write()方法写入数据。
- 完成需要以后，调用 close()关闭文件的打开。

7.2　Stream 类

流（Stream），是 I/O 中的一个重要概念。I/O 中的流就相当于现实生活中的水流一样，一打开自来水的龙头开关，水就从一头流向另外一头。Java 的 I/O 也是以流为基础的，根据方向的不同可以分为输入流和输出流；根据数据的格式不同，可以分为字节流和字符流。本节将集中讨论有关 Java 流的常见面试题。

面试题 074　字节流的处理方式

字节流是 I/O 中最原始的方式，因为计算机处理数据总是以一个 byte 为基本单位的，字节流就是每次读取的单位为 byte。字节流，是所有流的基础，也是其他高级流的前提。通过回答本例问题，可以了解到字节流作为 I/O 基础的核心概念。

【出现频率】 ★★★★

【关键考点】

- 流的概念；
- 如何使用字节流。

【考题分析】

I/O 总是分为两端的，一段作为输出，另一端则是输入端，因此就可以把流分为输入流和输出流。Java 中的基础字节输入流和输出流的类为：InputStream 和 OutputStream。通过它们再衍生出 FileInputStream 和 FileOutputStream、ObjectInputStream 和 ObjectOutputStream、BufferedInputStream 和 BufferedOutputStream 等。

字节流最大的特点，就是每次的输出和输入都是一个字节。因此，它主要应用在最原始的流的处理上，如内存缓存操作、文件复制等不需要关心流的内容是什么格式的地方。例如，面试题 072 提到的文件复制的示例程序，如果用单纯的 InputStream 和 OutputStream 详细代码如下：

```
import java.io.*;
public class FileCopy {
    public static void main(String[] args) throws IOException {
        //生成输入文件的输入流对象，字节流
        InputStream fin = new FileInputStream("d:/test/a.txt");
        //生成输出文件的输出流对象，字节流
        OutputStream fout = new FileOutputStream("d:/test/b.txt");
        byte[] buff = new byte[256];          //定义一个永固暂存数据的 byte 数组
        int len = 0;                          //每次读取数据的长度
        //循环读取读取到 buff 中，如果文件到底，则读取数据的长度小于 0
        while ((len = fin.read(buff)) > 0) {
            fout.write(buff, 0, len);         //把数据一次写进输出文件中
        }
        fin.close();                          //关闭输入流
        fout.close();                         //关闭输出流
    }
}
```

read()和 write()方法，是字节流的主要使用方法，它们的处理目标通常是一个 byte 数组，也就是把这些 byte 写入或读出。

byte 是计算机的基本单位，所以字节流可以应付几乎所有的流的处理。只不过，在处理具体数据格式的时候，效率没有具体的实现类高，如字符格式、对象格式等。但是，字节流式一切流的基础，其他的高级流类型都是基于字节流模型的。

【答案】

字节流处理的是计算机最基本的单位 byte，它可以处理任何数据格式的数据。主要的操作对象就是 byte 数组，通过 read()和 write()方法把 byte 数组中的数据写入或读出。

面试题 075　字符流的处理方式

针对文本文件，Java 提出了字符流的概念，使用字符流来写入和读出字符数据。无需再使用字节流进行包装，Java 已经为开发者提供了现成的字符流处理的接口和类。在现实的开发中，字符数据是最常用的数据格式，如何针对不同的字符需要进行类型的选择也是开发中需要具备的技能。通过回答本例问题，可以了解到字符流的相关概念和使用方法。

【出现频率】 ★★★★★

【关键考点】

- 字符流的概念；
- 字符流的使用方法。

【考题分析】

字符流是由字节流包装而来，它的输入和输出流类型包括 StringReader 和 StringWriter、BufferedReader 和 BufferedWriter。对于前者，它们的使用方法与字节流类似，主要还是 read()和 write()方法，而后者除了基本的 read()和 write()以外，有一个功能是在读取文章类型的文本文件时经常需要的，那就是 readLine()方法。

字符流对象的创建的时候，一般是需要提供一个输入或输出流的。例如，在创建 BufferedReader 或 BufferedWriter 对象的时候，需要提供一个 InputStreamReader 或 InputStreamWriter 对象。另外，对于特定字符格式的文本内容，还需要在创建 InputStreamReader 或 InputStreamWriter 对象的时候，提供字符格式类型作为构造方法的参数，以下是一段示例代码：

```
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
public class ReaderTest {
    public static void main(String[] args) throws IOException {
        InputStream in = new FileInputStream("d:/test/a.txt");
                                                    //获取输入流
        //得到 InputStreamReader 对象
        InputStreamReader isr = new InputStreamReader(in, "GBK");
        BufferedReader br = new BufferedReader(isr);    //创建 Reader 对象
        //创建 StringBuffer 对象，临时保存字符内容
        StringBuffer sb = new StringBuffer();
```

```
        String str = null;
        while ((str = br.readLine()) != null) {     //循环读取一行数据
            sb.append(str);                         //加入字符串
        }
        System.out.println("a.txt content : " + sb);        //打印内容
        br.close();                                         //关闭流
    }
}
```

以上的代码中，用 GBK 的编码方式读取文本文件的内容，也就是说，开发者不用去关心字节是如何被组装成字符的，而只需要直接使用即可。另外，readLine()方法也是一个很有用的方法，它直接就得到一行的字符数据，并返回 String 对象，开发者也就不用手动的去用 char 数组进行中转保存了。

另外，对于字符输出流来说，还可以使用 PrintWriter 类，它提供了一个非常有用的 println()方法，例如下面的代码：

```
PrintWriter pw = new PrintWriter(new FileOutputStream("d:/test/b.txt"));
pw.println("a");
pw.println("b");
pw.println("c");
pw.close();
```

以上代码就是在文件中打印了 3 行文字，分别是：a、b 和 c。

【答案】

字符流是由字节流包装而来，它的输入和输出流类型包括 StringReader 和 StringWriter、BufferedReader 和 BufferedWriter。字符流 API 的基本使用思路有以下步骤：

（1）获取输入或输出流对象。可以从 File 得到，也可以从网络或其他地方得到。

（2）根据特定的字符格式创建 InputStreamReader 或 InputStreamWriter 对象。

（3）使用 read()或 readLine()方法，得到数据。或使用 write()或 print()方法进行字符输出。

（4）最后关闭 reader 或 writer。

7.3 序　列　化

Java 对象可不可以被保存在文件中呢？答案是可以的，那就是使用 Java 的可序列化的功能。可序列化是数据持久化的一种方案，它的应用非常广泛，例如 Tomcat 保存 session 数据就会使用到它。本节将集中讨论有关 Java 序列化的常见面试题。

面试题 076　什么是序列化

序列化，目前已经得到大多数计算机语言的支持，Java 也不例外。Java 的序列化功能是作为 I/O 功能的一个子项，它的使用也是比较方便的。本例在回答该问题的同时，将更详细地讲解 Java 的序列化的工作原理。

【出现频率】 ★★★★

【关键考点】

- 序列化的概念

【考题分析】

序列化，又称为“串化”，可以形象的把它理解为把 Java 对象内存中的数据采编成一串二进制的数据，然后把这些数据存放在可以持久的数据存储设备，如磁盘。当需要还原这些数据的时候，再通过反序列化的过程，把对象又重新还原到内存中。

java.io.Serializable 接口是可以进行序列化的类的标志性接口，该接口本身没有任何需要实现的抽象方法。它仅仅是用来告诉 JVM 该类的对象的可以进行序列化的，并且它的序列化 ID 由静态的 serialVersionUID 变量提供。

serialVersionUID 变量其实是一个静态的 long 型的常量，它的作用在序列化和反序列化的过程中，起到了辨别一个类的作用。在反序列化的时候，如果两个类的类名完全相同，就通过 serialVersionUID 来判断该类是否符合要求，如果不行，则抛出异常。

Java 的 I/O 提供了一对类用作对象的序列化和反序列化，主要包括 ObjectOutputStream 和 ObjectInputStream。它们的用法与字节流相似，只不过此时处理的是对象，而不仅仅是字节数据了。

【答案】

序列化本质上就是把对象内存中的数据按照一定的规则，变成一系列的字节数据，然后再把这些字节数据写入到流中。而反序列化的过程相反，先读取字节数据，然后再重新组装成 Java 对象。

所有需要进行序列化的类，都必须实现 Serializable 接口，必要时还需要提供静态的常量 serialVersionUID。

面试题 077　如何序列化和反序列化一个 Java 对象

Java 既然对序列化和反序列化提供了支持，那肯定就会提供相应的接口或类，以方便开发者使用，其中主要就体现为：ObjectOutputStream 和 ObjectInputStream。本例在回答该问题的同时，全面地讲解 Java 的序列化相关的 I/O 的 API 的使用方法。

【出现频率】 ★★★★★

【关键考点】

- 序列化的概念和用法；
- ObjectOutputStream 和 ObjectInputStream 的使用方法。

【考题分析】

Java 的 I/O 提供了一对类用作对象的序列化和反序列化，它们是 ObjectOutputStream 和 ObjectInputStream。它们的用法与基本的字节流比较类似，只不过写入和读出方法为：readObject()和 writeObject()。以下是一个序列化和反序列化一个学生对象的示例：

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;
```

```
//学会类，实现 Serializable 接口
class Student implements Serializable {
    //序列化 ID
    private static final long serialVersionUID = 1L;
    private String name;                                    //学生姓名
    private int age;                                        //年龄
    //setters and getters
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
}
public class SerialTest {
    public static void main(String[] args) throws IOException,
    ClassNotFoundException {
        Student stu = new Student();                        //创建一个学生对象
        stu.setAge(20);                                     //设置年龄为 20
        stu.setName("zhangsan");                            //设置姓名为 zhangsan
        //创建一个对象输出流
        ObjectOutputStream oos
            = new ObjectOutputStream(new FileOutputStream("d:/test/
            obj.dat"));
        oos.writeObject(stu);                               //把对象写入输出流
        oos.close();                                        //关闭流
        //创建一个对象输入流
        ObjectInputStream ois = new ObjectInputStream(new FileInputStream
        ("d:/test/obj.dat"));
        Object obj = ois.readObject();                      //读出已经序列化的对象
        Student stuBak = (Student) obj;                     //进行类型的转换
        //打印数据在控制台，检查序列化和反序列化是否成功
        System.out.println("stu name is " + stuBak.getName());
        System.out.println("stu age is " + stuBak.getAge());
    }
}
```

注意：以上的示例中，Student 类只有两个成员变量，分别是 int 和 String 类型。如果，需要序列化的类包含其他的引用类型的成员变量，则它所包含的这些类也是需要实现 Serializable 接口的，并且这样的关系是循环嵌套一直到底的。

【答案】

对于对象的输出和输入，Java 的 I/O 体系中主要提供了 ObjectOutputStream 和 ObjectInputStream 两个类以供开发者使用，它们的基本使用思路有以下步骤。

（1）让需要序列化的类实现 java.io.Serializable 接口。

（2）提供静态的 long 型的常量 serialVersionUID。

（3）如果是序列化对象，则用一个输出流创建一个 ObjectOutputStream 对象，然后调用 writeObject()方法。

（4）如果是反序列化，首先使用一个输入流创建一个 ObjectInputStream 对象。然后调用 readObject()方法，得到一个 Object 类型的对象。最后再做类型的强制转换。

（5）最后关闭流。

7.4　小　　结

本章讲解了一些 Java 的 I/O 的面试题，包含了目录和文件的操作、字节流、字符流和对象存储（序列化和反序列化）等方面的问题。I/O 是 Java 的一个很重要的知识点，也是一个比较难以理解的知识点，它是网络编程的基础，网络数据的传输是基于 Java 基本的 I/O 知识。理解 Java 的 I/O，关键在于理解流的概念，所有的数据的输入与输出都是根据流而建立的模型。读者在多加练习的同时，应该认真思考每一种 I/O 情况使用哪种原理，这样才能把 I/O 的知识掌握透彻。

第 8 章　多线程编程

对于并发来说，一般可以有多进程和多线程两种方式。进程是占用的 CPU、内存等系统的基本单位，而线程又是进程的执行单元。多进程的并发对于数据的共享是很困难的，而多线程却相当的容易，因此大多数时候说的并发指的就是多线程。Java 的多线程是语言级的，并不依赖任何的操作系统 API，而且 Java 的多线程编程能力是很强大且好用的。本章将包含关于 Java 的多线程编程的一些常见面试题，并且分析这些题目和知识点，帮助读者梳理这些方面的知识。

8.1　多线程编程的基本概念

多线程编程得到了大多数计算机编程语言的支持，它通常也是作为操作系统的一个特性来对待。深刻的理解多线程概念是使用它们进行编程的前提。本节将集中讨论有关多线程编程的基本概念的常见面试题。

面试题 078　什么是多线程

什么是线程？什么又是多线程？多线程有什么好处？开发者要使用多线程编程，这些都是必须搞清楚的概念。通过回答本例的问题，将了解到线程及多线程的含义。

【出现频率】 ★★★★

【关键考点】

- 线程的概念；
- 多线程的概念；
- 多线程并发的含义。

【考题分析】

每个正在系统上运行的程序都是一个进程，每个进程包含一个或多个线程。进程也可能是整个程序或者是部分程序的动态执行。线程是一组指令的集合，或者是程序的特殊段，它可以在程序中独立执行，也可以把它理解为代码运行的上下文。

多线程是这样一种机制，它允许在程序中并发执行多个指令流，每个指令流都称为一个线程，彼此间互相独立。线程又称为轻量级进程，它和进程一样拥有独立的执行控制，由操作系统负责调度。

多线程是为了使得多个线程并行的工作可以完成多项任务，以提高系统的效率。线程是在同一时间需要完成多项任务的时候被实现的。使用多线程可以带来以下几点好处：

- 使用线程可以把占据长时间的程序中的任务放到后台去处理。

- 用户界面可以更加吸引人。如用户单击了一个按钮去触发某些事件的处理，可以弹出一个进度条来显示处理的进度。
- 程序的运行速度可能加快。多个人搬东西总比一个人强。
- 在一些等待的任务实现上如用户输入、文件读写和网络传输数据等，线程就比较空闲了。在这种情况下可以释放一些珍贵的资源，如内存占用等。

在当前的计算机系统中，多线程的应用程序已经是比比皆是了。多线程编程也得到了大多数语言的支持，包括 Java 在内。如果读者有 C/C++的编程经验可能就会知道，C/C++要想写出多线程的应用程序，必须依赖一些操作系统相关的 API，也就不能跨平台了。而 Java 的多线程功能是完全跨平台的，也开发者提供了方便。

【答案】

线程是进程中的一个执行单元，又称为轻量级进程，它和进程一样拥有独立的执行控制，由操作系统负责调度。

而多线程是这样一种机制：它允许在程序中并发执行多个指令流，每个指令流都称为一个线程。多线程进制下的线程彼此间互相独立，比较容易共享数据，通过并发执行的方式来提高程序的效率和性能。

面试题 079　解释进程和线程的区别

对于开发人员来说，进程和线程是两个非常重要的概念。它们的含义都是并发执行程序指令，都有提高执行任务效率的作用。但是它们有着一些比较重要的区别，会影响开发人员对程序的理解。通过回答本例的问题，将了解到进程和线程的概念及其它们之间的区别。

【出现频率】 ★★★★

【关键考点】

- 进程的概念；
- 线程的概念；
- 进程和线程的区别。

【考题分析】

进程（Process）普通的解释就是：进程是程序的一次执行。而线程（Thread）可以理解为进程中执行的一段程序片段。也就是说，它们首先是一种包含关系。

多线程共存于应用程序中，是现代操作系统中的基本特征和重要标志。在 Windows、UNIX 等多任务操作系统中，每个应用程序的执行都在操作系统内核中登记一个进程标志，操作系统根据分配的标志对应用程序的执行进行调度和系统资源分配。也就是说，进程是占用系统资源的基本单位。

线程的划分尺度小于进程，使得多线程程序的并发性高。另外，进程在执行过程中拥有独立的内存单元，而多个线程共享内存。多线程在共享数据方面更加方便，从而极大地提高了程序的运行效率。

线程在执行过程中与进程还是有区别的。每个独立的进程有一个程序运行的入口、顺序执行序列和程序的出口。但是线程不能够独立执行，必须依存在应用程序中，由应用程序提供多个线程执行控制。

当初，引入进程概念的目的是这样的：进程可以提高系统的并发性，提高 CPU 的使用率，从而提高程序的性能。在以前单道操作系统中，一次只执行一个程序，这样执行效率很低，资源得不到充分的利用。所以，后来多道操作系统出现了，由于多道操作系统一次并发执行很多程序，但管理起来非常麻烦。因此，进程的概念就被提出来了，它是一个程序动态执行表现，而且进程中包含了程序的资源，并管理程序如何去使用资源。可以说进程是程序的一种动态表现形式。

那么，又为什么要引入线程呢？进程与进程之间的通信是十分慢的，因为进程管理分配很多资源，所以转换起来很慢。就这样，线程产生了，它是比进程更小的单位，一个进程中只要有一个或多个线程，那么每个线程之间的通信相对进程而言则快了很多，因为线程只涉及到 CPU 的资源分配。一个进程中的线程是可以相互通信的，但不同进程之间的线程是不能相互通信的。

【答案】

根据以上的分析可知，进程和线程之间的区别主要包括：

- 线程的划分尺度小于进程，线程隶属于某个进程。
- 进程是程序的一种动态形式，是 CPU、内存等资源占用的基本单位，而线程是不能独立的占有这些资源的。
- 进程之间相互独立，通信比较困难，而线程之间共享一块内存区域，通信比较方便。
- 进程在执行过程中，包含比较固定的入口、执行顺序和出口，而线程的这些过程会被应用程序所控制。

8.2　Java 中的多线程编程

Java 对多线程提供了充分的支持，开发者在理解多线程概念的同时，还需要熟悉 Java 相关的多线程编程 API，才能开发出高效的多线程程序。本节将集中讨论有关 Java 多线程编程的常见面试题。

面试题 080　如何让一个类成为线程类

线程是一个方法的执行过程。既然是方法，它就必须在类中，那如何让 JVM 知道是哪个类的哪个方法呢？本例在回答该问题的同时，详细地讲解关于线程类和方法的设计思想。

【出现频率】 ★★★★★

【关键考点】

- Runnable 接口；
- Thread 类。

【考题分析】

让一个类成为线程类的方式有两种：一个是实现 java.lang.Runnable 接口，另一个是继承自 java.lang.Thread 类。其实，Thread 是实现了 Runnable 接口的，就是说，本质上线程

类是需要实现 Runnalbe 接口，只是 Thread 还提供了一些额外的方法。

在 Runnable 接口中，有一个 run()方法，它就是开发者需要实现的方法，当每个线程执行的时候，JVM 就会自动的调用这个方法。即使是继承 Thread 类，也是需要实现 run()方法的，以下是两种线程类的代码示例：

```
//实现 Runnalbe 接口的线程类
public class RunTest implements Runnable{
    public void run() {                          //run 方法
        System.out.println("thread running...");
    }
}
                                                 //继承自 Thread 类的线程类
class ThreadTest extends Thread{
    public void run(){                           //run 方法，默认为什么也不做
        System.out.println("thread running...");
    }
}
```

说明：Thread 类的 run()方法的默认实现为空实现，也就是什么也不做。

【答案】

让一个类成为线程类的方式有两种，一个是实现 java.lang.Runnable 接口，另一个是继承自 java.lang.Thread 类。

面试题 081　解释 Runnable 接口与 Thread 类的区别

Runnable 接口和 Thread 类都是用来创建线程类的，它们都需要实现 run()方法，似乎没有什么区别。其实，它们还是有差别的，尤其是要根据具体情况进行具体的选择。通过回答本例的问题，理解到它们二者的区别。

【出现频率】 ★★★★

【关键考点】

- Runnable 接口；
- Thread 类；
- 接口与类的区别。

【考题分析】

首先，大家应该知道，Java 的类是不允许多继承的，也就是只能继承自一个类。那么，如果线程类继承了 Thread 以后，它就不能再继承其他的类了。而 Runnable 接口，就不会有这样的问题，因为类是可以实现多个接口的。另外，Thread 提供了很多关于线程的方法，例如，获取线程的 Id、线程名、线程状态等方法。对于比较复杂一点的线程，可能就需要 run()方法中调用这些方法，而 Runnalbe 接口使用起来就没那么方便。

如果让一个线程类实现 Runnable 接口，那么当调用这个线程的对象开辟多个线程时，可以让这些线程调用同一个变量；若这个线程是由继承 Thread 类而来，那么就会麻烦一些，则要通过内部类来实现上述功能，利用的就是内部类可任意访问外部变量这一特性。例如，以下示例代码中的 index 变量。

```
class MyThread implements Runnable {                    //实现 Runnable 接口
    int index = 0;                                       //index 变量
    public void run() {
        ...
    }
}
class MyThread {
    int index = 0;                                       //index 变量
    private class InnerClass extends Thread {            //定义一个内部类，继承 Thread
        public void run() {
            ...
        }
    }
    Thread getThread() {                  //这个函数的作用是返回 InnerClass 的一个匿名对象
        return new InnerClass();
    }
}
```

【答案】

Runnable 接口与 Thread 类的区别主要包括以下几个方面：

- 线程类继承自 Thread 则不能继承自其他类，而 Runnable 接口可以。
- 线程类继承自 Thread 相对于 Runnable 来说，使用线程的方法更方便一些。
- 实现 Runnable 接口的线程类的多个线程，可以更方便的访问同一变量，而 Thread 类则需要内部类来进行替代。

面试题 082　如何启动一个线程

线程类需要实现 Runnable 接口或继承 Thread 类，线程执行的代码需要写在 run()方法中。那么如何启动一个线程呢？也就是并发的执行 run()方法中的代码。通过回答本例的问题，了解到如何启动一个 Java 线程。

【出现频率】 ★★★★

【关键考点】

- Thread 类的 start()方法

【考题分析】

通过上节提到的两种方式实现了一个线程类之后，线程的实例并没有被创建，因此它们也并没有被运行。其实，要启动一个线程，必须调用方法来启动它，也就是 Thread 类的 start()方法，而不是 run()方法（既不是继承 Thread 类重写的 run()方法，也不是实现 Runnable 接口的 run()方法）。run()方法中包含的是线程的主体，也就是这个线程被启动后将要运行的代码，它跟线程的启动没有任何关系。上面两种实现线程的方式在启动时会有所不同。

两种线程类的启动方式代码如下：

```
public class ThreadStartTest {
    public static void main(String[] args) {
        // 创建一个线程实例
        ThreadTest t1 = new ThreadTest();
        Thread t2 = new Thread(new RunnableTest());
        // 启动线程
        t1.start();
        t2.start();
```

```
    }
}
class ThreadTest extends Thread{
    public void run(){
        ...
    }
}
class RunnableTest implements Runnable{
    public void run(){
        ...
    }
}
```

说明：通常，可以把线程类统一为 Thread 类型，而不会用具体的线程类。也就是说，以上示例代码中，t1 的定义可以写成：Thread t1 = new ThreadTest();

【答案】

继承自 Thread 类的线程类，可以通过 new 关键创建一个线程对象以后，执行 start()方法开始一个线程。而实现了 Runnable 接口的线程类，需要用它的对象实例，作为 Thread 类构造方法的参数，创建一个 Thread 对象，然后调用 start()方法开始一个线程。

面试题 083　如何用 sychronized 来让线程同步

多线程一旦操作同一块内存的数据，就可能造成数据的混乱，也就是常说的线程安全问题。Java 针对这样的问题，采用 sychronized 关键字来帮助开发者解决这样的问题。那么 sychronized 是怎么使用的呢？本例在回答该问题的同时，全面地讲解关于 Java 多线程安全的问题。

【出现频率】 ★★★★★

【关键考点】

- 线程安全问题的造成原因；
- sychronized 的使用方法。

【考题分析】

首先，给出一段会出现线程安全问题的代码：

```
class MyThread extends Thread {
    public static int index;                          //静态变量 index
    public void run() {
        for (int i = 0; i < 100; i++) {               //循环打印 index 加 1 以后的值
            System.out.println(getName() + " : " + index++);
        }
    }
}

public class SyncTest {
    public static void main(String[] args) {
        new MyThread().start();                       //启动一个线程
        new MyThread().start();                       //启动一个线程
        new MyThread().start();                       //启动一个线程
    }
}
```

以上代码中，3 个线程都会去访问一个静态的变量 index，由于它们获取系统时间片的时刻是不确定的，因此它们对 index 的访问和修改总是穿插着的，也就会造成类似于以下的输出结果：

```
Thread-0 : 0
Thread-2 : 1
Thread-0 : 2
Thread-2 : 3
Thread-0 : 4
Thread-0 : 5
Thread-1 : 6
Thread-2 : 7
...
```

通过以上运行结果可以看出，线程之间并没有等谁执行完以后再执行，而是交织着执行的，这不符合程序的本意。有一个很好的办法，就是让它们可能会出现混乱的代码同步，也就是线程之间依次排队获取资源。

在 Java 中，使用 sychronized 关键字来保证线程的同步。sychronized 的工作原理是这样的：每个对象都可以有一个线程锁，sychronized 可以用任何一个对象的线程锁来锁住一段代码，任何想要进入该段代码的线程必须在解锁以后才能继续执行，否则就进入等待状态。其中，只有等占用该锁资源的线程执行完毕以后，该锁资源才会被释放。例如，以上程序中的 MyThread 类可以这样来写，就可以很好地解决冲突的问题：

```
class MyThread extends Thread {
    public static int index;                          //静态变量 index
    public static Object obj = new Object();          //用任意一个对象来加锁
    public void run() {
        synchronized(obj){                            //为冲突加上同步代码块
            for (int i = 0; i < 100; i++) {           //循环打印 index 加一以后的值
                System.out.println(getName() + " : " + index++);
            }
        }
    }
}
```

以上代码中，把 for 循环的代码块加上了 obj 对象的锁，当一个线程进入该段代码以后，即使其他线程进入 run()方法，也需要在获得锁的情况下才能继续运行。

另外，synchronized 还可以加在成员方法上，就叫做同步方法，此时的锁是加在 this 所引用的对象上的。加在方法上面，还可以达到把代码段进行进一步细分的作用，也免去了需要找一个对象来加锁的步骤。

说明： 不论是同步代码块，还是同步方法，它们都会损失一些效率。因此，开发者应该根据线程可能出现问题的地方加上同步，而不必大段大段的加上同步锁，这样很可能是程序的性能降低。

【答案】

sychronized 关键字代表要为某一段代码加上一个同步锁，这样的锁是绑定在某一个对象上边的。如果是同步代码块，需要为该 sychronized 关键字提供一个对象的引用；如果是同步方法，只需要加一个 sychronized 关键字的修饰。

sychronized 为某段代码加上锁以后，某个线程进入该段代码之前，首先需要检查该锁是否被占用，如果没有被占用则继续执行；如果已经被占用，则需要等到该锁被释放以后才能继续执行。其中，线程执行完该段代码就是释放锁的标志。

面试题 084　编写一个生产者与消费者模型的多线程例子程序

生产者与消费者模式，是一项非常经典的设计模式，它所涉及到的是多线程协调工作的问题。在 Java 中，一般是通过 wait()和 notify()方法进行解决。本例在回答该问题的同时，详细地讲解关于线程状态的相关概念。

【出现频率】 ★★★★

【关键考点】

- 线程状态的转变过程；
- wait()和 notify()的使用方法。

【考题分析】

如果有这样一个程序：有一个仓库类，它有一个仓库最大容积的变量以及添加货物和取走货物的方法，并且有多个线程在操作这样一个仓库对象。那么如何保证线程之间的协调呢，不至于仓库满了还被允许放置货物，或者没有货物还可以取走货物呢？这也就是传说中的生产者与消费者模式了。

大致的设计思路是这样的：每个生产者线程在添加货物之前，检查一遍仓库是否已满，如果已满则等待并通知消费者进行消费，直到消费者消费了一个货物以后，再继续添加；同样，每个消费者在取走每一件货物之前，都应该检查仓库是否为空，如果为空则等待并通知生产者进行生产，直到生产者添加了货物以后，再继续进行消费，示例代码如下：

```
//仓库类
public class Store {
    private final int MAX_SIZE;                    //仓库的最大容量
    private int count;                             //当前的货物数量
    public Store(int n){                           //初始化最大容量的构造方法
        MAX_SIZE=n;
        count=0;
    }
    //往仓库加货物的方法
    public synchronized void add(){
        while(count>=MAX_SIZE){                    //每次执行都判断仓库是否已满
            System.out.println("已经满了");
            try {
                this.wait();                       //如果满了，就进入等待池
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        count++;                                   //数量加 1
        //打印当前仓库的货物数量
        System.out.println(Thread.currentThread().toString()+" put "+count);
        //仓库中已经有东西可以取了，则通知所有的消费者线程来拿
        this.notifyAll();
    }
    //从仓库拿走货物的方法
```

```
    public synchronized void remove(){
        while(count<=0){                                    //每次执行都判断仓库是否为空
            System.out.println("空了");
            try {
                this.wait();                                //如果为空，就进入等待池
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        //打印当前仓库的货物数量
        System.out.println(Thread.currentThread().toString()+" get "+count);
        count--;                                            //数量减 1
        //仓库还没装满，通知生产者添加货物
        this.notify();
    }
    public static void main(String[] args){
        Store s=new Store(5);                               //创建容量为 5 的仓库
        //创建两个生产者和两个消费者
        Thread pro=new Producer(s);
        Thread con=new Consumer(s);
        Thread pro2=new Producer(s);
        Thread con2=new Consumer(s);
        pro.setName("producer");
        con.setName("consumer");
        pro2.setName("producer2");
        con2.setName("consumer2");
        //启动各个线程
        pro.start();
        pro2.start();
        con.start();
        con2.start();
    }
}
class Producer extends Thread{                              //生产者线程类
    private Store s;
    public Producer(Store s){
        this.s=s;
    }
    public void run(){                                      //线程方法
        while(true){                                        //永真循环
            s.add();                                        //往仓库加货物
            try {
                Thread.sleep(1000);                         //为更直观，休息 1 秒钟
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
class Consumer extends Thread{                              //消费者线程类
    private Store s;
    public Consumer(Store s){
        this.s=s;
    }
    public void run(){                                      //线程方法
        while(true){                                        //永真循环
            s.remove();                                     //从仓库取走货物
            try {
```

```
                Thread.sleep(1500);                    //为更直观，休息 1.5 秒钟
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

注意：使用了 wait()或 notify()的方法，需要加上 synchronized 关键字，否则可能会抛出 java.lang.IllegalMonitorStateException 异常。

在所有的 Java 根类 java.lang.Object 中，包含了 3 个重载的 wait()方法以及 notify()和 notifyAll()方法。它们主要的作用在于让当前的线程进入等待池或从等待池中唤醒一个或多个线程继续执行。正是通过它们，才能很好的解决生产者和消费者模型的问题。在以上示例中，由于生产者线程执行更快，因此程序会间歇性的打印出类似如下执行结果：

```
...
Thread[producer2,5,main] put 3
Thread[producer,5,main] put 4
Thread[producer,5,main] put 5
已经满了
Thread[consumer,5,main] get 5
Thread[producer2,5,main] put 5
Thread[consumer2,5,main] get 5
Thread[producer,5,main] put 5
已经满了
Thread[consumer2,5,main] get 5
...
```

同理，如果消费者线程执行更快。则会打印出若干个“空了”的结果。

【答案】

示例程序如以上的分析步骤。该程序的关键点在于，如何使用 wait()和 notify()方法协调生产者和消费者线程的交叉进行。

面试题 085　如何使用 Java 的线程池

Java 的线程池是 Java5.0 以后的新功能，它让开发者更易开发高效的多线程程序，也让多线程程序的性能大大提高。Java 提供的关于线程池的 API 是基于原有线程 API 的，只是用另外一种方式来使用 Java 的多线程编程功能。本例在回答该问题的同时，详细地讲解关于线程池的概念。

【出现频率】 ★★★★

【关键考点】

- 线程池的概念；
- Java 关于线程池的 API 使用方法。

【考题分析】

简单地说，线程池就是一个或多个线程的集合。一般而言，线程池有以下几个部分。

- 完成任务的一个或多个线程。
- 用于调度管理的管理线程。

- 要求执行的任务队列。

说明：Java 5.0 提出的线程池（ThreadPool）的概念，区别于线程组（ThreadGroup）的概念。

那么，为什么要使用线程池呢？线程池属于对象池，所有对象池都具有一个非常重要的共性，就是为了最大程度复用对象，因此，线程池最重要的特征也就是最大程度利用线程。其次，Java 线程池的编程模型相对于原有的多线程编程模式来说，还有一大改进，那就是线程代码和业务代码的分离。

线程池所对应的类为 java.util.concurrent.ThreadPoolExecutor，它在构造的时候一般需要提供池大小等的参数，常用构造方法如下：

```
ThreadPoolExecutor(int corePoolSize,
    int maximumPoolSize,
    long keepAliveTime,
    TimeUnit unit,
    BlockingQueue<Runnable> workQueue,
    RejectedExecutionHandler handler);
```

一个线程任务通过 execute(Runnable)方法被添加到线程池，任务就是一个 Runnable 类型的对象，任务的执行方法就是 Runnable 类型对象的 run()方法。

当一个任务通过 execute(Runnable)方法欲添加到线程池时，有以下几点：

- 如果此时线程池中的数量小于 corePoolSize，即使线程池中的线程都处于空闲状态，也要创建新的线程来处理被添加的任务。
- 如果此时线程池中的数量等于 corePoolSize，但是缓冲队列 workQueue 未满，那么任务被放入缓冲队列。
- 如果此时线程池中的数量大于 corePoolSize，缓冲队列 workQueue 满，并且线程池中的数量小于 maximumPoolSize，建新的线程来处理被添加的任务。
- 如果此时线程池中的数量大于 corePoolSize，缓冲队列 workQueue 满，并且线程池中的数量等于 maximumPoolSize，那么通过 handler 所指定的策略来处理此任务。

以下为一个常见的线程池应用示例：

```
import java.io.Serializable;
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.ThreadPoolExecutor;
import java.util.concurrent.TimeUnit;
public class TestThreadPool {
    private static int produceTaskSleepTime = 2000;     //间歇时间，毫秒
    public static void main(String[] args) {
        //构造一个线程池
        ThreadPoolExecutor producerPool =
            new ThreadPoolExecutor(1, 1, 0, TimeUnit.SECONDS,
                new ArrayBlockingQueue(3),
                new ThreadPoolExecutor.DiscardOldestPolicy());
        //每隔 produceTaskSleepTime 的时间向线程池派送一个任务
        int i = 1;
        while (true) {
            try {
                Thread.sleep(produceTaskSleepTime);     //休息一定的时间
                String task = "task@" + i;              //设置任务名字
```

```
                System.out.println("put " + task);
                //用 execute 方法启动一个线程
                producerPool.execute(new ThreadPoolTask(task));
                i++;
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
//线程类
class ThreadPoolTask implements Runnable, Serializable {
    private static final long serialVersionUID = 0;

    private static int consumeTaskSleepTime = 2000;     //时间间歇，毫秒
    private String threadPoolTaskData;                  //存储人物名的变量
    ThreadPoolTask(String tasks) {                      //构造方法
        this.threadPoolTaskData = tasks;
    }
    //每个任务的执行过程，现在是什么都没做，除了 print 和 sleep，:)
    public void run() {
        System.out.println("start .." + threadPoolTaskData);
        try {
            //便于观察现象，等待一段时间
            Thread.sleep(consumeTaskSleepTime);
        } catch (Exception e) {
            e.printStackTrace();
        }
        threadPoolTaskData = null;
    }
}
```

以上代码中，把所有的线程都交给一个 Executor 来管理，把对线程的各种操作集中在一起进行管理。这样，分离了业务代码和线程本身的管理代码，而且还让线程的执行效率更高且易管理。

【答案】

Java 提供了 java.util.concurrent.ThreadPoolExecutor 类来使用线程池，通过它构造的对象，可以很容易地管理线程，并把线程代码与业务代码进行分离。具体使用方法请参见以上的示例程序。

8.3　小　　结

本章讲解了一些 Java 多线程编程的面试题，涵盖了线程和进程的概念、多线程编程的思想、Java 中如何定义线程类、如何启动一个线程、如何保证线程同步以及线程池等知识点。在当今的计算机系统中，多线程应用程序比比皆是，它在提高程序性能方面起到了非常巨大的作用。但是，多线程是一种比较抽象的概念，对于入门不久的开发者来说，是一个难点，这些概念的掌握是需要基于充分的实践的。读者只有在学习 Java 关于多线程 API 使用方法的同时，还需要认真思考每一个方法的具体作用是什么，可以完成什么样子的功能，能够在脑子里建立起充分的多线程模型。只有这样才能全面的掌握多线程编程，也就能应对每一道多线程的面试题了。

第 9 章　Java 的反射机制

反射机制（Reflection），是 Java 提供的一项比较高级的功能，一般的 Java 开发者使用它的时候相对比较少。但是，Java 的反射却是大多数框架的基础，例如，Struts、Hibernate 和 Spring 等。反射机制就好像外科医生的解剖工具，突然从外太空来了一个人，如何知道它的具体构造呢？最好的办法就是把它解剖来看一看，反射机制就是用来解剖 Java 的类、接口、方法、属性等元素的。本章将包含关于 Java 的反射机制的一些常见面试题，并且分析这些题目和知识点，帮助读者梳理这些方面的知识。

9.1　反 射 基 础

反射提供了一种动态的功能，这种动态功能非常的强大。它主要体现在通过反射相关的 API，就可以知道一个陌生的 Java 类的所有信息，包括属性、方法、构造器等。而且元素完全可以在运行时动态的进行创建或调用，而不必在 JVM 运行时就进行确定。本节将集中讨论有关 Java 反射机制基础的常见面试题。

面试题 086　反射的原理是什么

严格意义上来讲，Java 并不是动态语言，它并没有 Perl 和 Python 那样灵活多变。但是，Java 的反射机制却让它显得动态起来，这是为什么呢？通过回答本例问题，将了解 Java 反射的原理是什么。

【出现频率】 ★★★★

【关键考点】

- ❑ 什么是反射机制；
- ❑ 反射机制的作用是什么。

【考题分析】

一般情况下，如果想生成这个类的对象时，运行这个程序的 JVM 会确认这个类的 Class 对象是否已经加载。如果尚未加载，JVM 就会根据类名查找.class 文件，并将其载入，一旦这个类的 Class 对象被载入内存，它就可以被用来创建这个类的所有对象。

另外，对于一个未知类型的引用来说，通常会采用强制类型转换的形式来得到开发者想要的类型引用，如果执行了一个错误的类型转换，就会抛出一个 ClassCastException 异常。

在以上两个过程中，Class 类一直都在起作用。因为 Class 类实例包含的是一个 Java 类的全部信息，包括类名、方法、属性等。换句话来说，也就是 Class 对象就是代表一个

类的类。

Java 用 Class 类来代表了所有的类，方便了开发者掌控类信息。通过 Class，开发者可以得到属性（Field）、方法（Method）、构造器（Constructor）、修饰符（Modifier）等信息。在动态的获取这些信息以后，开发者就可以用该 Class 创建实例、调用任何方法、访问任何属性等操作，这些也是反射的主要用途。

反射机制相关的 API 主要集中在 java.lang.reflect 包下面，开发者也就是利用该包下面的接口和类进行相关的反射开发的。大多数的框架，如 Struts、Hibernate 和 Spring，都会频繁地使用反射 API 来完成它们动态的功能。

【答案】

反射是为了能够动态地加载一个类，动态地调用一个方法，动态地访问一个属性等动态要求而设计的。它的出发点就在于 JVM 会为每个类创建一个 java.lang.Class 类的实例，通过该对象可以获取这个类的信息，然后通过使用 java.lang.reflect 包下的 API 以达到各种动态需求。

面试题 087　Class 类的含义和作用是什么

Class 类在反射机制中起到了非常关键的作用，它是开发者进行反射开发的入口。那应该如何获取一个类的 Class 对象呢？如何让一个类动态的加载到 JVM 中呢？通过回答本例的问题，深入地了解 Class 类的含义和用途。

【出现频率】 ★★★★★

【关键考点】

- Class 类的含义；
- 如何获取 Class 对象。

【考题分析】

在前面内容中，读者知道了 Class 是用来记录一个类的各种信息的，它伴随着类的加载而创建。那么，类会在什么时候被加载到 JVM 中呢？概括来说，一个普通的 Java 类，会在以下几种情况，被加载到 JVM 中。

（1）需要使用该类创建对象。如下代码就会到导致 Student 类被加载到 JVM 中：

```
Sutdent stu = new Student();                    //创建学生对象
```

（2）访问该类的静态成员，示例代码如下：

```
System.out.println(Student.count);              //访问静态的 count 成员变量
```

（3）使用 Class 类的静态 forName()方法，动态地加载一个指定类名的类。如果类未找到，则抛出 ClassNotFoundException 异常，示例代码如下：

```
Class.forName("com.test.Student");              //动态加载 Student 类
```

说明： JDBC 导入驱动类，就是 Class.forName()的一个常用例子。

不管通过什么样的形式，类一旦被加载进入 JVM 以后，就会为它创建一个 Class 类的实例对象。那么，开发者又该如何得到这个 Class 对象呢？概括来说，要得到一个类的 Class

对象，可以通过以下几种途径。

（1）Class 的 forName()方法的返回值就是 Class 类型，也就是动态导入类的 Class 对象的引用。forName()方法的完整定义如下：

```
public static Class<?> forName(String className)
                    throws ClassNotFoundException
```

（2）每个类都会有一个名称为 Class 的静态属性，通过它也是可以获取到 Class 对象的，示例代码如下：

```
Class<Student> clazz = Student.class;                    //访问 Student 类的 class 属性
```

（3）Object 类中有一个名为 getClass 的成员方法，它返回的是对象的运行时类的 Class 对象。因为 Object 类是所有类的父类，所以，所有的对象都可以使用该方法得到它运行时类的 Class 对象，示例代码如下：

```
Student stu = new Student();                             //创建一个 Student 对象
Class<Student> clazz = stu.getClass();                   //调用 getName 方法
```

注意：从 Java 5.0 开始，Class 类是支持泛型的，这个类型也就是它所代表的类。

获取到 Class 对象以后，就可以通过调用它的一些成员方法来获取它所代表的类的属性、方法、修饰符等信息，以及调用 newInstance()方法来创建新的实例对象，还有其他一些功能方法。

【答案】

每一个 Class 类的对象就代表了一种被加载进入 JVM 的类，它代表了该类的一种信息映射。开发者可以通过以下 3 种途径获取到 Class 对象。

- Class 类的 forName()方法的返回值。
- 访问所有类都会拥有的静态的 class 属性。
- 调用所有对象都会有的 getClass()方法。

在 Class 类中，定义许多关于类信息的方法。例如，getName()、getMethod()、getConstructor()和 newInstance()等可以用于反射开发，还有 isInstance()和 isInterface()等一些关于类的功能方法。

面试题 088　如何操作类的成员变量（Field）

Field 类，代表是的类的属性（字段），也称为成员变量。既然反射可以获取到它的对象，那又该如何使用它呢？本例在回答该问题的同时，全面地讲解 Java 反射机制关于 Field 的常用方法。

【出现频率】 ★★★★

【关键考点】

- Field 的含义；
- Field 类的常用方法。

【考题分析】

Field 对象通过 Class 类的 getDeclaredField()或 getDeclaredFields()方法获取到，处于

java.lang.reflect 包下。Field 提供有关类或接口的单个字段的信息，以及对它的动态访问权限，反射的字段可能是一个静态的字段或实例的字段。

Field 的方法主要分为两大类，即 getXXX 和 setXXX。其中 getXXX 是用于获取某个对象的该字段的值，并且有一定的类型规定，例如，getFloat()、getInt()等；而 setXXX 则是用于设置值，它们一般有两个参数，一个是对象引用，另一个则是需要设置的值。

以下为一个 Field 的使用示例，通过反射来比较两个对象的大小：

```
import java.lang.reflect.Field;
//测试类
class FieldTestClass{
    String name;                                          //属性 1，姓名
    int age;                                              //属性 2，年龄
    //构造方法
    public FieldTestClass(String name, int age) {
        super();
        this.name = name;
        this.age = age;
    }
}
public class FieldTest {
    public static void main(String[] args) {
        FieldTestClass obj1 = new FieldTestClass("hello1",100);
                                                          //创建对象 1
        FieldTestClass obj2 = new FieldTestClass("hello2",300);
                                                          //创建对象 2
        System.out.println(compare(obj1, obj2).name+" is bigger");
    }
    //用反射定义一个通用的比较方法，返回 age 相对更大的对象
    private static FieldTestClass compare(FieldTestClass obj1, Field-
    TestClass obj2){
        try {
            //以下用两种方式获取一个 Field 对象
            Field field = obj1.getClass().getDeclaredField("age");
            field = FieldTestClass.class.getDeclaredField("age");
            //获得两个对象的 age 的值
            int val1 = (Integer)field.get(obj1);
            int val2 = (Integer)field.get(obj2);
            //进行比较
            if(val1 > val2){
                return obj1;
            }else{
                return obj2;
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

说明：上例代码中，使用了 Field 的 getXXX 方法。如果读者感兴趣，还可以用 setXXX 试试。

【答案】

Field 提供有关类或接口的单个静态或实例字段的信息，它通过 Class 类的

getDeclaredField()或 getDeclaredFields()方法获取到，再置于 java.lang.reflect 包下。Field 的方法主要分为两大类，即 getXXX 和 setXXX，它们都需要提供相应的实例对象，setXXX 还需要提供需要设置的值。

面试题 089　如何操作类的方法（Method）

Method 类，代表的是类的方法（静态或非静态的）。与 Field 类似，反射可以获取到它的对象，使用它的方式则是方法动态调用。本例在回答该问题的同时，将更全面地讲解 Java 反射机制关于 Method 的常用方法。

【出现频率】 ★★★★

【关键考点】

- Method 的含义；
- Method 类的常用方法。

【考题分析】

Method 对象通过 Class 类的 getMethod()或 getMethods()方法获取到，也处于 java.lang.reflect 包下。Method 提供关于类或接口中的某个方法（以及如何访问该方法）的信息，所反映的方法可能是类方法或实例方法（包括抽象方法在内）。

Method 类中使用最多的方法是 invoke()，它的含义就是方法调用。它的第一个参数为 Class 所代表的类的一个实例对象，以后则是一个不定长的 Object 类型的参数列表，它是 Method 对象所代表的方法需要的参数列表。

说明：不定长参数是 Java 5.0 以后，才提出的新特性。

以下为一个 Method 的使用示例，通过命令行参数来实现方法的动态调用。

```
import java.lang.reflect.Method;
class MethodTestClass{
    public void m1(){                                    //测试方法 1
        System.out.println("m1 is called...");
    }
    public void m2(){                                    //测试方法 2
        System.out.println("m2 is called...");
    }
}
public class CallMethodTest {
    public static void main(String[] args) {            //main()方法
        args = new String[]{"m2"};                       //测试代码
        String methodName = args[0];                     //得到方法名参数
        if(methodName != null){
            //得到 Class 实例
            Class<MethodTestClass> clazz = MethodTestClass.class;
            try {
                //获得指定方法名的 Method 对象
                Method m = clazz.getDeclaredMethod(methodName);
                if(m != null){
                    MethodTestClass obj = clazz.newInstance();
                                                         //创建一个新对象
                    m.invoke(obj);                       //调用 obj 对象的指定方法
                }
```

```
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```

对于以上示例程序，如果通过命令行输入参数 m1，则调用 m1()方法；若输入 m2，则调用 m2()方法。

【答案】

Method 提供关于类或接口中的某个方法（以及如何访问该方法）的信息，包括了静态方法和成员方法（包括抽象方法在内）。它通过 Class 类的 getMethod()或 getMethods()方法获取到，该类定义在 java.lang.reflect 包下。Method 类的最常用的方法是 invoke()，正是通过它来完成方法被动态调用的目的。

9.2　反射应用举例

现在的 Java 应用程序中，反射机制使用得越来越多，其中的一个重要原因就是反射的动态性可以减少许多的代码冗余度，便于程序进一步扩展。本节将就几类比较常见的反射应用的面试题进行讲解。

面试题 090　如何利用反射实例化一个类

通常来说，开发者使用 new 关键字就可以实例化一个类，也就是创建一个对象。那么使用反射机制能否创建一个对象呢？答案是肯定的，这样的功能让对象的创建更加动态和灵活了。本例在回答该问题的同时，详细地讲解 Java 反射机制是如何实例化一个类的。

【出现频率】 ★★★★

【关键考点】

- Class 类的 newInstance()方法的使用；
- Constructor 类的使用。

【考题分析】

使用 new 关键字来创建对象的时候，系统将调用类的某个构造方法。构造方法往往起到初始化变量的作用，因此使用反射来创建对象的时候，也是需要调用构造方法的。对于默认的无参数的构造方法来说，反射机制可以使用 Class 类的 newInstance()方法即可。但是，如果需要调用其他的有参数的构造方法的时候，则需要使用 java.lang.reflect.Constructor 类，它代表了类的构造方法。

为了获取类的构造方法实例，需要调用 Class 类的 getConstructor()方法，得到相应的构造方法，其中，getConstructor()需要传递的参数就是对应的构造方法的参数类型的 Class 实例。得到 Constructor 对象以后，再通过它的 newInstance()方法来创建对象，该方法的参数列表也就和构造方法的参数列表一模一样了。

在以下示例代码中，展示了两种使用反射创建对象的情况。

```
package ch09;
import java.lang.reflect.Constructor;
class Student{
    private String name;                                    //姓名
    private int age;                                        //年龄
    //无参数构造方法
    public Student() {
        super();
    }
    //带参数构造方法
    public Student(String name, int age) {
        super();
        this.name = name;
        this.age = age;
    }
    //覆盖 toString()方法
    public String toString(){
        return "stu:"+name+","+age;
    }
}
public class NewInstanceTest {
    public static void main(String[] args) throws Exception {
        //使用无参数的构造方法
        Class<Student> clazz = Student.class;   //获取 Class 对象
        Student obj = clazz.newInstance();      //调用 newInstance 方法
        System.out.println(obj);                //检验一下是否创建成功
        //使用带参数的构造方法
        Constructor<Student> con = clazz.getConstructor(String.class,int.
        class);
        //用构造器创建对象
        obj = con.newInstance("zhangsan",30);
        System.out.println(obj);                //打印学生信息
    }
}
```

技巧：读者应该会发现，使用 Constructor 来实例化一个类是很麻烦的，它会涉及构造方法的查找和调用。其实，完全可以使用 Class 类的 newInstance()方法，然后再通过 setters 方法来为成员变量赋值。所以，读者最好养成一种习惯，为每一个自定义类提供一个无参数的构造方法。

【答案】

根据调用构造方法的不同，用反射机制来实例化一个类，可以有两种途径。如果使用无参数的构造方法，则直接使用 Class 类的 newInstance()方法即可；若需要使用特定的构造方法来创建对象，则需要先获取 Constructor 实例，再用 newInstance()方法创建对象。

面试题 091　如何利用反射机制来访问一个类的私有成员

根据 Java 的语法规则，类的私有成员（包括私有的成员变量和方法）只能被内部的方法所访问。但是，通过反射机制，却可以访问一个类的所有成员，包括私有成员在内。通

过回答本例问题，将了解反射机制是如何访问私有成员的。

【出现频率】 ★★★★

【关键考点】

- 反射 API 的使用；
- 如何获得私有成员的实例。

【考题分析】

如果开发者在开发反射机制程序的时候，对待私有成员像公有成员一样，那就会遇到一个异常的抛出。系统提示该属性或方法没有找到，那么这样看来，好像反射也不允许访问私有成员。而实际上，反射的确是可以访问私有成员的，如下示例：

```
package ch09;
import java.lang.reflect.Field;
class PrivateTestClass{
    private String field1;                          //私有属性
    //构造方法
    public PrivateTestClass(String field1) {
        super();
        this.field1 = field1;
    }
}
public class PrivateTest {
    public static void main(String[] args) throws Exception {
                                                    //main()方法
        PrivateTestClass obj = new PrivateTestClass("hello");
        Class clazz = obj.getClass();               //获取 Class 对象
        Field f = clazz.getDeclaredField("field1");  //获取的 Field 对象
        f.setAccessible(true);                  //把它的可访问性设置为 true
        System.out.println(f.get(obj));         //打印出来
    }
}
```

从以上代码来看，私有的成员变量 field1 是可以被成功访问的。其中比较关键一行代码就是调用 Filed 的 setAccessible()方法，让 field1 属性变得可以访问。

【答案】

在使用反射机制访问私有成员的时候，它们的可访问性是为 false 的。需要调用 setAccessible(true)方法，把原本不可访问的私有成员变为可以访问以后，才能进行成功的访问或调用。

面试题 092　如何利用反射来覆盖数据对象的 toString()方法

对于用来存储数据的对象，例如，实体对象、数据对象等，它们往往都会有一种需求，那就是遍历它们的所有属性及其相应的值。这样就可以一目了然它们的数据组成是怎样的，对于日志打印是很有用的。那么，能不能用一种通用的方式来得到对象数据的字符串组成呢？答案是肯定的。本例在回答该问题的同时，将讲解更多的 Java 反射机制关于 Field 的使用技巧。

【出现频率】 ★★★★

【关键考点】

❑ Field 的使用

【考题分析】

用于保存数据的类，它们会包含许多的属性（或称为成员变量），而且这些变量可能会发生一些变化，例如，改变名称、新增属性、去掉属性等。如果在 toString()方法中，把这些属性用硬代码（将一些可变的数据直接写在代码中，就叫做硬代码）的方式进行字符串相加，则非常不便于以后维护。

其实，使用反射机制完全可以适用于所有的这类数据对象的数据遍历的。它的基本思想就是使用 Field 类所提供的相关 API 进行字符串的拼接，示例代码如下：

```
package ch09;
import java.lang.reflect.Field;
class DataObject{
    private String name;                                    //姓名
    private String desc;                                    //描述
    private int age;                                        //年龄
    private String other;                                   //其他
    //构造方法
    public DataObject(String name, String desc, int age, String other) {
        super();
        this.name = name;
        this.desc = desc;
        this.age = age;
        this.other = other;
    }
    //覆盖 toString 方法
    public String toString() {
        StringBuffer sb = new StringBuffer();   //定义一个 StringBuffer
        //得到所有的成员变量 Fields
        Field[] fields = this.getClass().getDeclaredFields();
        //遍历一下
        for(Field f : fields){
            sb.append(f.getName());                         //得到变量的名字
            sb.append("=");                                 //添加“=”号
            try {
                sb.append(f.get(this));                     //得到变量的值
            } catch (IllegalArgumentException e) {
                e.printStackTrace();
            } catch (IllegalAccessException e) {
                e.printStackTrace();
            }
            sb.append("\n");                                //添加一个换行符
        }
        return sb.toString();                               //返回结果
    }
}
public class DataObjectTest {
    public static void main(String[] args) {    //main()方法
        //创建一个 DataObject 的对象
        Object obj = new DataObject("zhangsan","desc test",10,"other
        test");
        //打印它的 toString 方法的返回值
        System.out.print(obj);
    }
}
```

说明：以上代码中，访问 Field 的时候，并没有使用 setAccessible(true)方法，那是因为 toString()方法在类中，因此不使用。

【答案】

对于数据类型的类的 toString()方法，覆盖它的基本思路，主要有以下几点：

- 通过 getDeclaredFields()方法得到所有的 Field 对象。
- 把上一步得到的 Field 对象数组进行遍历。
- 每一次循环加上字段名和字段值。
- 返回循环叠加以后的字符串结果。

9.3　小　结

本章讲解了一些 Java 的反射机制的面试题。首先讲解了反射的基础知识，包括 Class 类、反射的原理、Field 和 Method 等，然后就几类比较常见的反射应用的面试题进行了讲解。反射机制是 Java 相对比较高级一点的功能，想要完全理解它是比较困难的。但是，如果不能完全理解它，而想要使用它来编程是举步维艰的。读者应该认真体会一下本章 9.2 节的几类实例的设计思想和用途，并加强这方面的练习，才能真正掌握和运用 Java 的反射机制。

第 10 章　Java 的网络编程

Java 从诞生的那天起，就是伴随着网络的发展而一起发展的，因此，Java 肯定会对网络编程提供充分的支持。大多数计算机语言的网络通信基本编程模式都采用 Socket（套接字）的形式，Java 也是一样，这种模式是基于目前流行于世界各地的 TPC/CP 通信协议的。因此，网络程序开发者不仅需要熟悉 Java 所提供的相关 API，还需要了解关于网络的一些基础知识。本章将包含关于 Java 的网络编程的一些常见面试题，并且分析这些题目和知识点，帮助读者梳理这些方面的知识。

10.1　网络编程基础

千里之行，始于足下。在进行实际的 Java 网络编程之前，首先需要了解 Java 网络编程的一些基础知识，因为 Java 关于网络编程 API 的设计也是以这些理论知识为基础的。本节将集中讨论有关网络基础知识的常见面试题。

面试题 093　TCP/IP 协议的理解

相信，大家一定不会对 TCP/IP 感到陌生。这个世界上，只要是互联网所能达到的地方，都离不开 TCP/IP 协议。那么，它到底是怎么样的一种协议呢？通过回答本例的问题，可以比较深刻地了解 TCP/IP 协议的含义和作用。

【出现频率】 ★★★★

【关键考点】

- TCP/IP 协议的含义；
- TCP/IP 协议的功能。

【考题分析】

TCP/IP（Transmission Control Protocol/Internet Protocol 的简写），中文译名为传输控制协议/因特网互联协议，又叫网络通讯协议。这个协议是 Internet 最基本的协议，也是 Internet 的基础。简单地说，它的名字是由网络层的 IP 协议和传输层的 TCP 协议组成的。但是确切地说，TCP/IP 协议是一组包括 TCP 协议和 IP 协议，UDP（User Datagram Protocol）协议、ICMP（Internet Control Message Protocol）协议和其他一些协议的协议组。

TCP/IP 定义了电子设备（如计算机）如何连入因特网，以及数据如何在它们之间传输的标准。它是互联网中的基本通信语言或协议，在私网中，它也被用作通信协议。当用户直接网络连接时，计算机应提供一个 TCP/IP 程序的标准实现，而且接收所发送的信息的计算机也应有一个 TCP/IP 程序的标准实现。

TCP/IP 协议并不完全符合 OSI 标准定制的七层参考模型，它采用了四层的层级结构，每一层都呼叫它的下一层所提供的网络来完成自己的需求。这四层分别为以下几个方面：

（1）应用层：应用程序间沟通的层，如万维网（WWW）、简单电子邮件传输（SMTP）、文件传输协议（FTP）、网络远程访问协议（Telnet）等。

（2）传输层：在此层中，它提供了节点间的数据传送，应用程序之间的通信服务，主要功能是数据格式化、数据确认和丢失重传等。主要协议包括 TCP 和 UDP。

（3）互连网络层：负责提供基本的数据封包传送功能，让每一块数据包都能够到达目的主机，但不检查是否被正确接收，主要体现为 IP 协议。

（4）网络接口层：接收 IP 数据报并进行传输，从网络上接收物理帧，抽取 IP 数据报转交给下一层，对实际的网络媒体的管理，定义如何使用物理网络，如以太网。

对于网络开发者来说，关心最多的应该是最高的应用层，也就是开发出能给用户直接使用的网络应用程序。开发者使用传输层所提供的接口进行开发，常见的两种通信模型为 TCP 和 UDP。

对于 Java 开发者来说，JDK 已经提供了充分的开发接口，java.net 包下的接口和类就是网络编程人员经常会光顾的地方。Java 关于网络开发的 API 的设计依然是遵循 TCP/IP 协议的基本思想的。常用的接口和类包括：ServerSocket、Socket、URL 等。

【答案】

TCP/IP 定义了电子设备（如计算机）连入因特网的标准，以及数据如何在它们之间传输的标准。它既是互联网中的基本通信语言或协议，也是局域网的通信协议。

TCP/IP 是一组包括 TCP 协议、IP 协议，UDP 协议、ICMP 协议和其他一些协议的协议组。需要进行网络通信的计算需要提供符合这些协议标准的程序以后，才能进行网络通信。

面试题 094　TCP 协议的通信特点是什么

TCP 是目前使用最多的传输控制协议之一，广泛的用于互联网中，它的一个典型例子就是大家熟知的 Web 服务器与浏览器之间的通信。它的通信原理是怎样的呢？如何使用它来编写网络应用程序呢？本例在回答该问题的同时，全面地讲解 TCP 协议。

【出现频率】 ★★★★

【关键考点】

- TCP 协议的原理

【考题分析】

TCP 是一种面向连接的、可靠的、基于字节流的运输层（Transport Layer）通信协议。在 TCP/IP 协议组中，TCP 层是位于 IP 层之上，应用层之下的中间层。不同主机的应用层之间经常需要可靠的、像管道一样的连接。但是 IP 层不提供这样的流机制，而是提供不可靠的包交换，此时 TCP 协议就可以提供这样的一种可靠的连接。

应用层向 TCP 层发送 8 位字节表示的数据流，然后 TCP 把数据流分割成适当长度的报文段，之后 TCP 把结果包传给 IP 层，由它来通过网络将包传送给接收端实体的 TCP 层。TCP 为了保证不发生丢包，采用一种精确的确认机制，一旦发送失败将提供重发的机会，可以保证数据的可靠传输。

使用 TCP 协议进行通信的应用程序是众多的。那么如何区分它们的通信通道呢？其实，TCP 提供了一种端口机制，用于区分各种网络应用程序，让它们在各自的通信端口上各行其道。TCP 端口地址都是 16 比特，可以有在 0～65535 范围内的端口号，表 10.1 为一些常见的网络应用程序及其端口号。

表 10.1　常见TCP应用程序及其端口号

属　性	端　口	说　明
Telnet	23	远程登录
FTP	21	文件传输协议
Web	80	Web 网站
SMTP	25	邮件发送

说明：以上应用程序的端口号为它们的默认端口号，并不代表它们只能用这些端口，它们的端口号可以进行修改的。

【答案】

根据以上分析的 TCP 的通信原理，可以得出 TCP 协议主要拥有如下的通信特点：

（1）面向连接的传输。

（2）端到端的通信。

（3）可靠性，确保传输数据的正确性，不出现丢失或乱序。

（4）采用字节流方式，即以字节为单位传输字节序列。

面试题 095　Java 的 TCP 编程模型是什么

Java 对 TCP 通信提供了充分的支持，它的编程模型是相对固定的，任何 TCP 应用程序都是以该模型为基础。本例在回答该问题的同时，全面地讲解 Java 关于 TCP 通信的实现和使用。

【出现频率】★★★★

【关键考点】

- TCP 通信原理；
- Java 关于 TCP 的编程模型。

【考题分析】

TCP 一般用于 C/S（客户端/服务器端）模式的应用程序，它们都会存在客户端和服务器端两个部分。对于服务器端，使用 Java 在 java.net 包下的 API，它们的基本编程步骤如下：

（1）创建一个服务器端的 Socket，指定一个端口号。Java 提供了 java.net.ServerSocket 类来实现这一步，代码可以这样写：

```
ServerSocket ss = new ServerSocket(int 型的端口号);        //创建 ServerSocket
```

（2）开始监听来自客户端的请求要求。调用 ServerSocket 的 accept()方法即可，示例代码如下：

```
ss.accept();
```

（3）获得输出流或输入流。网络也属于一种 I/O，Java 也是按照流来对待它们，开发者一旦获取这些流对象以后，就可以当成普通的文件输入/输出流一样来使用。它们都来自于连接成功以后的 Socket 对象，示例代码如下：

```
Scoket socket = ss.accept();                          //accept 获取 socket 对象
OutputStream os = socket.getOutputStream();           //获得输出流
InputStream is = socket.getInputStream();             //获得输入流
```

（4）调用输入流/输出流的 read()或 write()方法，进行数据的传输。如果是字符数据流，一般还可以用 BufferedReader 或 PrintWriter 进行数据的传输，示例代码如下：

```
OutputStream os = s.getOutputStream();                //获得输出流
PrintWriter pw = new PrintWriter(os);                 //创建 PrintWriter 对象
pw.print(内容);
pw.flush();                                           //清空缓存
//或者
InputStream is = s.getInputStream();                  //获得输入流
InputStreamReader isr = new InputStreamReader(is);
BufferedReader br = new BufferedReader(isr);          //创建 BufferReader 对象
```

注意：对于数据写出端，需要调用 flush()方法，把缓存中的数据发送到对端，否则可能造成数据不能及时送达的后果。

（5）释放资源。关闭输出流/输入流、Socket 和 Server Socket 对象，例如下面的代码：

```
pw.close();                                           //关闭输出流
s.close();                                            //关闭 socket
ss.close();                                           //关闭 server socket
```

客户端的编程步骤有点类似，只不过它不必创建 ServerSocket 对象了，它直接去连接服务器端即可，步骤如下：

（1）创建 Socket 对象，建立与服务器端的连接，示例代码：

```
Socket s = new Socket("IP 地址",端口号);        //用 IP 地址和端口创建 Socket 对象
```

（2）获得输出流或输入流。

（3）调用输入流/输出流的 read()或 write()方法，进行数据的传输。

（4）释放资源，关闭输出流/输入流、socket 对象。

以下是一个简单的 TCP 编程示例，功能：把服务器端的时间发送到客户端。

```
//服务端
public class TCPServerA {
    public static void main(String[] args) throws Exception {
        //用端口号创建一个 ServerSocket 对象
        ServerSocket ss = new ServerSocket(8888);
        Socket s = ss.accept();                          //开始监听来自客户端的请求
        OutputStream os = s.getOutputStream();           //获得输出流
        PrintWriter pw = new PrintWriter(os);            //创建 PrintWriter 对象
        pw.print("now time = " + new Date());            //向输出流中写出当前的时间
        pw.flush();                                      //清空缓存
```

```
        //关闭输出流和 Socket
        pw.close();
        s.close();
        ss.close();
    }
}
//客户端
public class TCPClientA {
    public static void main(String[] args) throws Exception {
        //用 IP 地址和端口创建 Socket 对象
        Socket s = new Socket("localhost",8888);
        //获得输入流
        InputStream is = s.getInputStream();
        //创建 BufferReader 对象
        InputStreamReader isr = new InputStreamReader(is);
        BufferedReader br = new BufferedReader(isr);
        String str = br.readLine();          //读取一行
        System.out.println(str);             //打印从服务器端来的结果
    }
}
```

注意：一般来说，任何的 TCP 程序，都是需要先运行服务器端，然后再运行客户端的。

【答案】

编写 Java 的 TCP 网络应用程序需要分为服务器端和客户端两个部分，大致有以下步骤。

服务器端：

（1）创建一个服务器端的 Socket，指定一个端口号。

（2）开始监听来自客户端的请求要求。

（3）获得输出流或输入流。

（4）调用输入流/输出流的 read()或 write()方法，进行数据的传输。

（5）释放资源，关闭输出流/输入流、Socket 和 Server Socket 对象。

客户端：

（1）创建 Socket 对象，建立与服务器端的连接。

（2）获得输出流或输入流。

（3）调用输入流/输出流的 read()或 write()方法，进行数据的传输。

（4）释放资源，关闭输出流/输入流、Socket 对象。

面试题 096　UDP 协议的通信特点是什么

UDP 协议与 TCP 协议同处于 TCP/IP 的传输层，它们都是用于数据的传输的，只不过它没有提供像 TCP 那样的可靠机制。但是，它具有更小的消耗和更快的速度，适用于一些点对点传输的，并且安全性要求不高的网络应用程序。本例在回答该问题的同时，全面地讲解 UDP 协议。

【出现频率】 ★★★★

【关键考点】

❑ UDP 协议的原理

【考题分析】

UDP 是一个简单的面向数据报的传输层协议，与 TCP 协议同处于传输层，介于应用层和 IP 协议之间。同样，UDP 网络应用程序使用 UDP 协议进行通信的时候，依然需要提供端口号，以区分各个程序之间的通信通道。

与 TCP 不同，UDP 并不提供对 IP 协议的可靠机制、流控制以及错误恢复功能等。由于 UDP 比较简单，它的头部包含了很少的字节，比 TCP 负载消耗少，速度也相对快一些。

UDP 适用于不需要 TCP 可靠机制的情形，它也服务于很多知名应用层协议，包括域名系统（DNS）、简单文件传输系统（TFTP）、动态主机配置协议（DHCP）、某些视频会议程序和某些影音串流服务等。

【答案】

根据以上分析的 UDP 的通信原理，可以得出 UDP 协议主要拥有如下的通信特点：

（1）UDP 是一个无连接协议，传输数据之前源端和终端不建立连接，当它想传送时就简单地去抓取来自应用程序的数据，并尽可能快地把它扔到网络上。

（2）不需要维护连接状态，包括收发状态等。

（3）字节开销很小。

（4）吞吐量主要受应用软件生成数据的速率、传输带宽、源端和终端主机性能等因素的限制。

面试题 097　Java 的 UDP 编程模型是什么

Java 对 UDP 通信提供了充分的支持，它的编程模型是相对固定的，任何 UDP 应用程序都是以该模型为基础。本例在回答该问题的同时，全面地讲解 Java 关于 UDP 通信的实现和使用。

【出现频率】 ★★★★

【关键考点】

❑ UDP 通信原理；

❑ Java 关于 UDP 的编程模型。

【考题分析】

UDP 一般用于安全性要求不高的点对点传输模式的应用程序，它们都会存在两个终端。两端的编程方式类似，基本编程步骤大致有以下几个方面：

（1）创建数据 Socket，指定一个端口号。Java 提供了 java.net.DatagramSocket 类来支持这一步，两端可以使用不同的端口号，示例代码：

```
DatagramSocket ds = new DatagramSocket(9999);  //创建 DatagramSocket 对象
```

（2）对于接收消息的一端来说，提供一个 byte 数组进行数据的存储；而对于发送消息一端，除此以外还需要提供对端的 IP 地址和端口号，示例代码如下：

```
//创建一个 byte 数组
byte[] buff = new byte[1024];
```

```
//创建一个数据包对象
DatagramPacket dp = new DatagramPacket(buff,1024); //长度小于等于buff的长度
//或者:
String str = "数据";
//用byte数组，创建数据包对象
DatagramPacket dp
    = new DatagramPacket(str.getBytes(),0,str.length(),
          InetAddress.getByName("localhost"),9999);
```

（3）调用 DatagramPacket 的 receive()或 send()方法进行数据的接收或发送，示例代码如下：

```
ds.receive(DatagramPacket对象);                    //接收消息
//或者
ds.send(DatagramPacket对象);                       //发送消息
```

（4）调用 DatagramPacket 的 getData()方法得到 byte 数组的数据。

```
dp.getData();
```

注意：如果缓存的 byte 数组长度与接收的数据长度不统一，则需要用某些方法进行数据的切分存储。

（5）释放资源。关闭 DatagramSocket 对象，例如下面的代码：

```
ds.close();                                        //关闭DatagramSocket
```

以下是一个简单的 UDP 编程示例，功能：发送一个简单的 abc 字符串数据。

```
//接收端
public class UDPServerA {
    public static void main(String[] args) throws Exception {
        //创建DatagramSocket，指定端口
        DatagramSocket ds = new DatagramSocket(9999);
        //创建一个byte数组
        byte[] buff = new byte[1024];
        //创建一个数据包对象
        DatagramPacket dp = new DatagramPacket(buff,1024);
        ds.receive(dp);                                    //接收消息
        String str = new String(dp.getData(),0,dp.getLength());
                                                           //得到数据
        System.out.println(str);                           //打印结果
        ds.close();//关闭DatagramSocket
    }
}
//发送端
public class UDPClientA {
    public static void main(String[] args) throws Exception {
        //创建DatagramSocket，指定端口
        DatagramSocket ds = new DatagramSocket(9998);
        String str = "abc";
        //用byte数组，创建数据包对象
        DatagramPacket dp =
            new DatagramPacket(str.getBytes(),0,str.length(),
                InetAddress.getByName("localhost"),9999);
        ds.send(dp);                    //发送数据
```

```
        ds.close();                    //关闭 DatagramSocket
    }
}
```

说明：对于接收端来说，只有收到了消息以后才能知道发信者是谁。

【答案】

编写 Java 的 UDP 网络应用程序需要分为接收端和发送端两个部分，它们大致的步骤相同，主要包括以下几方面：

（1）创建数据 Socket，指定一个端口号。

（2）对于接收消息的一端来说，提供一个 byte 数组进行数据的存储；而对于发送消息一端，除此以外还需要提供对端的 IP 地址和端口号。

（3）调用 DatagramPacket 的 receive()或 send()方法进行数据的接收或发送。

（4）调用 DatagramPacket 的 getData()方法得到 byte 数组的数据。

（5）释放资源。

10.2　Java 网络编程举例

有了以上的网络基础知识以后，就可以开始编写任何的网络应用程序了。本节将集中讨论有关 Java 网络编程实践方面的常见面试题。

面试题 098　如何创建 TCP 通信的服务器端的多线程模型

TCP 网络程序的服务器端程序往往需要接受多个客户端的请求，如果总是单线程进行处理的话，很可能在请求较多的时候，有的请求无法得到及时的响应。要解决该问题，最好的办法就是采用多线程，用每一线程来处理每一次请求。本例在回答该问题的同时，将更详细地讲解如何在 TCP 程序的服务器端使用多线程模型。

【出现频率】 ★★★★★

【关键考点】

- TCP 编程模型；
- 多线程编程；
- 网络输入/输出。

【考题分析】

在进行 TCP 网络应用程序的服务器端编程的时候，首先需要创建一个 ServerSocket 对象，在指定的端口进行监听，然后调用 accept()方法，等待客户端的连接。为了多线程处理请求，肯定需要不停地调用 accept()方法，一旦 accept()方法成功返回一个 Socket 对象，则表示有一个客户端发起了一次连接请求，因此可以把 accept()方法的返回值作为循环条件，例如：

```
Socket socket = null;                                    //定义 socket 变量
while( (socket==serverSocket.accept()) != null ){        //循环接受请求
```

```
    ...
}
```

有了如上的代码以后，此时循环体内肯定是需要开启一个线程来处理本次请求，因为主线程还需要去监听和接受其他的请求，必须尽快结束主线程的单次循环。另外，开发者需要定义一个线程类（实现 Runnalbe 接口或继承 Thread 类）。然而，TCP 连接往往是需要使用连接的输入流/输出流的，怎么给线程类的 run()方法提供呢？最好的办法就是在线程类定义一个 Socket 类型的对象引用，把 accept()方法返回的 Socket 对象通过构造方法参数的形式传递给线程类的对象。线程类的示例代码如下：

```
//线程类
class MyThread extends Thread{
    private Socket socket;                              //连接点
    public MyThread(Socket socket) {                    //构造方法
        super();
        this.socket = socket;
    }
    //线程方法
    public void run(){
        //...
    }
}
```

根据以上的思路，把面试题 095 的 TCP 编程示例用多线程的模型重写一遍，代码可以这样来完成：

```
package ch10;
import java.io.IOException;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Date;
//服务端
public class TCPServerB {
    public static void main(String[] args) throws Exception {
        //用端口号创建一个 ServerSocket 对象
        ServerSocket ss = new ServerSocket(8888);
        //开始循环监听来自客户端的请求
        Socket s = null;
        while((s=ss.accept()) != null){
            //开始一个新的线程
            new MyThread(s).start();
        }
        ss.close();
    }
}
//线程类
class MyThread extends Thread{
    private Socket socket;                              //连接点
    public MyThread(Socket socket) {                    //构造方法
        super();
        this.socket = socket;
    }
    //线程方法
    public void run(){
```

```
        try {
            OutputStream os = socket.getOutputStream();      //获得输出流
            PrintWriter pw = new PrintWriter(os);   //创建 PrintWriter 对象
            pw.print("now time = " + new Date());   //往输出流写出当前的时间
            pw.flush();                             //清空缓存
            //关闭输出流和 Socket
            pw.close();
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

采用多线程编程模型以后，不仅提高了服务器端程序处理并发的能力，而且对于代码的管理也是有帮助的，主线程只负责监听和接受请求，而业务代码则放在 run()方法中。

【答案】

使用多线程模型来编程 TCP 程序的服务器端，主要有以下步骤：

（1）创建 ServerSocket 对象，指定监听的端口号。

（2）把 accept()方法作为循环条件，循环监听客户端请求。

（3）创建线程类，定义一个 Socket 类型的成员变量，并定义一个可以为它赋值的构造方法。

（4）在 run()方法中使用 socket 变量进行任意的通信操作。

（5）在主线程的循环体内开启一个线程，并传入 accept()方法的返回值。

面试题 099　用 TCP 通信模型创建一个 Web 服务器

大家一定知道，IIS、Apache 和 Tomcat 等这些 Web 服务器软件可以用来创建 Web 站点，负责接受客户端浏览器的 HTTP 请求。那这些软件是如何实现的呢？其实基本原理很简单，它们都是采用 TCP 通信模型的。本例在回答该问题的同时，全面地讲解用 Java 的 TCP 编程 API 创建一个简易的 Web 服务器。

【出现频率】 ★★★★

【关键考点】

- TCP 编程模型；
- HTTP 协议的通信原理。

【考题分析】

HTTP 是一个应用层次的协议，是基于 TCP 协议的，它底层的通信原理依然需要遵循 TCP 的编程模型。只不过，HTTP 还定义了其他一些 Web 通信的规范。

其实，浏览器在访问一个 Web 服务器的时候，它本质上也是发了一次 TCP 连接的请求，然后服务器端程序根据请求的 URL 的不同，返回不同的结果。只不过，它们的通信存在一些特殊的规定，例如，HTTP 状态值是 200 则代表请求成功，返回的 HTML 语言的文本等。浏览器会根据返回的 HTML 代码，做出相应的解析和效果展示。

毫无疑问，Web 服务器端程序肯定也是支持多线程的。这里，有一个简单的 Web 服务器端程序示例，它始终在浏览器中打印一句：hello this is my web page.详细代码如下所示：

```
package ch10;
import java.io.IOException;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
//Web 服务端
public class WebServer {
    public static void main(String[] args) throws Exception {
        //用端口号创建一个 ServerSocket 对象，监听 Web 的端口 80
        ServerSocket ss = new ServerSocket(80);
        //开始循环监听来自客户端的请求
        Socket s = null;
        while((s=ss.accept()) != null){
            new HTTPThread(s).start();                    //开始一个新的线程
        }
        ss.close();
    }
}
//线程类，HTTP 处理线程
class HTTPThread extends Thread{
    private Socket socket;                                //连接点
    public HTTPThread(Socket socket) {                    //构造方法
        super();
        this.socket = socket;
    }
    //线程方法
    public void run(){
        try {
            OutputStream os = socket.getOutputStream();     //获得输出流
            PrintWriter pw = new PrintWriter(os);   //创建 PrintWriter 对象
            //往输出流写出当前的时间
            pw.println("<html>");
            pw.println("<body>");
            pw.println("hello this is my web page.");
            pw.println("</body>");
            pw.println("</html>");
            pw.flush();     //清空缓存
            //关闭输出流和 Socket
            pw.close();
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

说明： Web 应用的默认端口为 80，浏览器访问的时候无需提供端口号。但是，开发者完全可以进行自定义，不过在用浏览器访问的时候，需要额外提供该端口号，例如，http://localhost:8080。

把以上程序运行以后，就可以通过浏览器来发出 HTTP 请求。打开一个浏览器窗口，在地址栏输入：http://localhost，按 Enter 键，就可以得到如图 10.1 所示的结果，就如同在访问一个 Web 服务器一样。

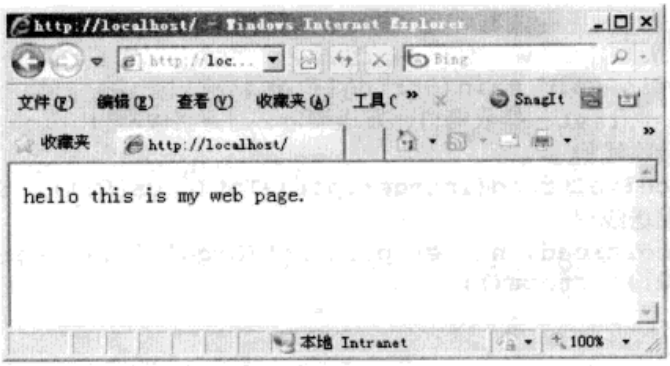


图 10.1　Web 服务器程序访问效果图

【答案】

HTTP 是一个应用层次的协议，它的服务器端的程序是基于 TCP 协议的。因此，可以用 TCP 的编程模型来模拟 Web 服务器端程序的实现，例如以上的示例。但是正式的 Web 服务器端软件，例如，IIS 和 Apache，实现得更好。

面试题 100　用 UDP 通信模型创建一个即时聊天软件

相信大家都用过聊天工具，例如著名的 QQ、MSN 等。这些聊天工具的通信原理是怎样的呢？其实，它们对安全的要求不是太高，一般都会采用 UDP 的通信模式，采用 Java 的 UDP 编程模型完全可以把它们模拟出来。本例在回答该问题的同时，全面地讲解用 Java 的 UDP 编程 API 创建一个简易的聊天软件。

【出现频率】 ★★★★

【关键考点】

- UDP 编程模型；
- 多线程编程模型。

【考题分析】

大家知道，聊天界面至少有两个部分：接收用户输入和打印聊天记录。它们既然需要同时存在于一个进程中，那肯定是多线程的。另外，UDP 的通信是点对点的，没有谁是服务器端，也没有谁是客户端的，因此 UDP 的多线程模型就不用有主线程来监听请求了，只需要在线程中分别使用 DatagramPacket 通信即可。

根据设计，可以创建一个发送消息的线程类（SendThread）和一个接收消息的线程类（ReceiveThread）。SendThread 主要做的事情就是循环监听用户的输入，然后把用户输入的数据通过调用 DatagramSocket 的 send()方法发出。而 ReceiveThread 的 run()方法，则主要是循环的调用 DatagramSocket 的 receive()方法，接收另外一端来的数据，并打印在屏幕上面来，详细代码如下：

```
package ch10;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;
```

```
//聊天类
public class Chat {
    public static void main(String[] args) throws Exception {
        //args format: 接收端口、发送端口、对方接收端口
        //启动接收线程
        new ReceiveThread(Integer.parseInt(args[0])).start();
        //启动发送线程
        new SendThread(Integer.parseInt(args[1]),Integer.parseInt
        (args[2])).start();
    }
}
//接收线程
class ReceiveThread extends Thread {
    private DatagramSocket ds;
    public ReceiveThread(int port) {                          //构造方法
        super();
        try {
            this.ds = new DatagramSocket(port);
        } catch (SocketException e) {
            e.printStackTrace();
        }
    }
    public void run() {
        try {
            //创建 byte 数组
            byte[] buff = new byte[1024];
            // 创建一个数据包对象
            DatagramPacket dp = new DatagramPacket(buff, 1024);
            //永真循环
            while (true) {
                ds.receive(dp);                               //接收消息
                String str = new String(dp.getData(), 0, dp.getLength());
                                                              //得到数据
                System.out.println("receive:"+str);          //打印结果
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally{
            ds.close();
        }
    }
}
//发送线程类
class SendThread extends Thread {
    private DatagramSocket ds;
    private int sendPort;                                     //对方端口号
    public SendThread(int port,int sendPort) {                //构造方法
        super();
        this.sendPort = sendPort;
        try {
            this.ds = new DatagramSocket(port);
        } catch (SocketException e) {
            e.printStackTrace();
        }
    }
    public void run() {
        try {
            //循环接收用户输入
            BufferedReader br = new BufferedReader(new InputStreamReader
```

```
                (System.in));
                String str = null;
                while ((str = br.readLine()) != null) {
                    //创建 DatagramPacket 对象
                    DatagramPacket dp =
                        new DatagramPacket(str.getBytes(), 0, str.length(),
                            InetAddress.getByName("localhost"), sendPort);
                    ds.send(dp);                                    //发送数据
                    System.out.println("send:"+str);                //打印结果
                }
            } catch (Exception e) {
                e.printStackTrace();
            } finally{
                ds.close();
            }
        }
    }
```

在使用以上的聊天程序时，需要在命令行中提供 3 个参数，即接收端口号、发送端口号和对方的端口号。当演示该程序的时候，需要开启两个程序，当一个程序发消息的时候，可以在另一边看见发送的消息，效果如图 10.2 和图 10.3 所示。

图 10.2　聊天客户端 A

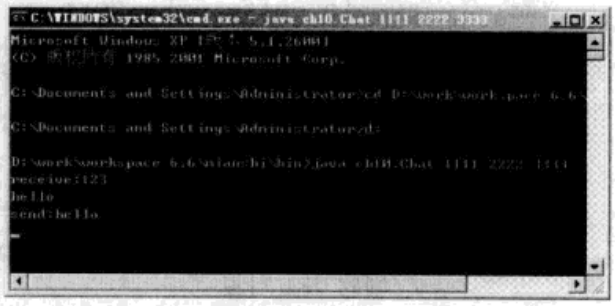


图 10.3　聊天客户端 B

说明：以上程序仅支持了两个客户端进行通信，读者如果感兴趣，可以把它改造成支持多客户端的，类似于一个聊天室，基本的实现原理是类似的。

【答案】

主要的思路包括：为接收用户输入和打印聊天记录两个界面创建两个不同的线程，它们分别使用 DatagramPacket 通信。创建一个发送消息的线程类（SendThread）和一个接收消息的线程类（ReceiveThread），分别的作用是循环监听用户的输入并把用户输入的数据通过调用 DatagramSocket 的 send()方法发出和循环的调用 DatagramSocket 的 receive()方法，接收另外一端来的数据。

面试题 101　如何使用 Java 访问 Web 站点

大家可能知道一种名叫“网络爬虫”的软件，它主要用于搜索引擎抓取网页用的。这种软件的原理就是模拟浏览器挨个的去访问 Web 站点，得到站点网页的映射。那么，Java 支持用编程的方式去访问网站吗？答案是肯定的。本例在回答该问题的同时，全面地讲解如何用 Java 的网络 API 访问 Web 站点。

【出现频率】 ★★★★

【关键考点】

- URL 的含义和使用；
- HttpURLConnection 类的使用方法；
- HTTP 协议的通信原理。

【考题分析】

统一资源定位符 URL（Uniform Resource Locator 的缩写）也被称为网页地址，是因特网上标准的资源地址，它唯一的标示了一个网络上的资源。Java 提供了 java.net.URL 类来支持它，也就是通过它来访问网络资源。

java.net.HttpURLConnection 是代表 HTTP 网络连接的类，它由 URL 类的 openConnection()方法获得。HttpURLConnection 可以模拟一个浏览器，在发出请求的时候，设置各种请求头参数，设置请求参数，传输请求数据；在收到响应结果的时候，也可以获得响应头数据和响应内容，它也是对 TCP 传输的输入流/输出流的一种包装。以下为一个使用 HttpURLConnection 访问某网站的代码示例：

```
package ch10;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.List;
import java.util.Map;

public class HttpConnTest {
    public static void main(String[] args) throws Exception {
        //创建 URL 对象
        URL url = new URL("http://www.javaeye.com");
        //用 URL 创建 HttpURLConnection 对象
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        //打开连接
        conn.connect();
        //打印请求响应的头部信息
        Map<String,List<String>> header = conn.getHeaderFields();
        for(String key : header.keySet()){
            System.out.println(key+":"+header.get(key));
        }
        //打印响应内容
        BufferedReader br =
            new BufferedReader(new InputStreamReader(conn.getInput-
            Stream(),"UTF-8"));
        String str = null;
        while((str = br.readLine()) != null){
            System.out.println(str);
        }
        conn.disconnect();        //断开连接
    }
}
```

在以上代码中，使用 HttpURLConnection 的 getHeaderFields()方法得到网站的响应头信息，采用的是 Map 类型的“键值对”格式。如果需要得到响应的内容时，则可以调用 getInputStream()方法得到输入流。

注意：如果读者使用的是代理服务器上网，则需要进行一些系统参数的设置，包括如下参数。

```
Properties systemProperties = System.getProperties();
systemProperties.setProperty("http.proxyHost",代理服务器的 IP 地址);
systemProperties.setProperty("http.proxyPort",代理服务器的端口号);
```

【答案】

使用 Java 关于网络的 API 来访问 Web 站点，主要包括以下步骤：

（1）用 URL 类创建一个资源定位的对象。

（2）调用 URL 的 openConnection()方法得到 HttpURLConnection 对象。

（3）调用 HttpURLConnection 的 open()方法打开连接。

（4）用 getHeaderFields()方法得到响应结果的头部信息。

（5）用 getInputStream()方法得到输入流对象，得到响应内容。

10.3　小　　结

本章讲解了一些 Java 的网络编程的面试题。首先讲解了一些网络编程的基础知识，包括：TCP/IP 协议、TCP 协议及其编程模型、UDP 协议及其编程模型等知识点。然后就几类比较常见的网络应用的面试题进行了讲解和解答。网络是 Java 涉及最多的领域之一，也是 Java 最成功的领域之一，它对 Java 的影响是很大的。熟悉 Java 网络编程的相关 API 和了解网络知识同样重要，尽管 Java 的网络编程模型相对固定，但是充分的实践是掌握这些知识的必要条件。读者应该认真体会一下 10.2 节的几类实例的设计思想和用途，并加强这方面的练习，才能真正熟练掌握和运用 Java 的网络编程技术。

第 11 章　Java 对数据库的操作

数据库是当今应用最多的服务器软件之一，简单地说，它的功能就是数据的存储和检索。如果需要在 Java 程序中访问数据库，是否得到支持呢？这是肯定的。Java 与连接数据库的首选技术当然是 JDBC（Java DataBase Connectivity，Java 数据库连接）了，它内嵌在 JDK 中，能够很好的屏蔽各种数据库之间的差异，使用统一的关系查询语句 SQL 进行数据的增删查改的操作。另外，JDBC 还是支持数据库连接池技术，是 J2EE 开发经常使用到的技术。本章将包含关于 Java 对数据库操作的一些常见面试题，并且分析这些题目和知识点，帮助读者梳理这些方面的知识。

11.1　SQL 基础

数据库经过几十年的发展，已经相当的成熟，尤其是关系型数据库。关系型数据库的普及程度非常的高，大多数时候说的数据库指的就是关系型数据库。SQL 语言是一种通用的操作数据库的语言，大多数程序都是使用 SQL 来完成各种数据库的操作，Java 程序也不例外。本节将集中讨论有关数据库和 SQL 的常见面试题。

面试题 102　什么是 SQL

操作数据库的方法有很多种，而使用 SQL 是最直接、最普遍的操作方法。其实，SQL 就相当于数据库提供给外界的数据操作接口，无论是手动还是编程的方式，都可以向数据库发送 SQL 命令，数据库会根据 SQL 的语法做出相应的动作响应。本例在回答该问题的同时，详细地讲解 SQL 的概念和特点。

【出现频率】★★★★

【关键考点】

- 关系数据库的概念；
- SQL 的定义和特点。

【考题分析】

SQL（发音为字母 S-Q-L）是结构化查询语句（Structured Query Language）的缩写，是一种专门用于和数据库通信的计算机语言。几乎所有的关系型数据库的管理软件（DBMS）都会对 SQL 提供支持，SQL 并不特定于某类数据库。

与其他计算机语言相比，SQL 并没有非常多的语法规定，一般不需要长篇大论，都是由很少的词构成，有的时候也可以称它为 SQL 命令。SQL 的设计目的仅仅就是为了很好的完成一项：提供一种从数据库读取数据的简单有效的方法。

SQL 语句是一种结构化的语句，它的语法结构比较固定，比较容易学会。例如，需要从数据库中查出所有用户的数据可以这样来写：

```
select *
FROM user
```

说明：SQL 语法上是大小写不敏感的，即使大小写混着写也是没问题的，也就是说 select 和 SELECT 代表了相同的含义。

以上的查询语句非常简单，仅仅 4 个字符串，即使不会 SQL 的人看了这样的语句也能猜出它大概可以做什么。所有的 SQL 语句的结构可以抽象成一句话：干什么，从哪里，怎么干。

SQL 的语法简单并不代表它的功能就弱。其实，SQL 是一种功能非常强大的语言，通过灵活的组织它的语言元素，可以进行非常复杂和高级的数据库操作。也就是说，SQL 几乎可以进行一切数据库的相关操作。概括起来，SQL 有如下优点：

（1）SQL 不是某个特定数据库提供商专有的语言。几乎所有重要的 DBMS 都支持 SQL，因此，学习 SQL 语言使开发者可以和各种数据库打交道。

（2）SQL 简单易学。它的语句全是有很强描述性的英语单词组成，大多数 SQL 语句的意图都是一目了然。

（3）SQL 功能强大。它几乎可以完成所有的数据库相关操作。

【答案】

SQL 是结构化的查询语句（Structured Query Language）的缩写，是一种专门用于和数据库通信的计算机语言，得到了大多数数据库的支持，它具有以下一些特点：

- 可以跨各种数据库使用。
- 简单易学。
- 功能强大。

面试题 103　如何使用 SQL 检索数据

查询功能是 SQL 最主要的功能之一，它使用非常频繁，掌握难度也较大。通过查询功能可以看出一个求职者的 SQL 功底。本例在回答该问题的同时，全面地讲解 SQL 的查询功能的基本使用思路。

【出现频率】★★★★★

【关键考点】

- SELECT 语句的使用

【考题分析】

首先，检索数据肯定会用到 SELECT 关键字，它的用途是从一个或多个表中检索信息。使用 SELECT 语句，必须至少给出两条信息：想选什么和从什么地方选，语法如下：

```
SELECT 列名                    /*需要选的列的名称*/
FROM 表名/视图                 /*从哪里选*/
```

例如，当需要检索某张表中的单列数据的时候，语法如下：

```
SELECT name
FROM user
```

以上就得到了用户表中所有的用户名字。如果还需要检索出其他的列的数据，则在 SELECT 语句后面加上需要的列名并用逗号隔开，例如下面的语法：

```
SELECT name, age, gender
FROM user
```

以上就得到了用户表中的用户的姓名、年龄和性别信息。另外，如果需要笼统的检索出全部列的信息呢，则使用“*”代替列名即可，例如：

```
SELECT *
FROM user
```

有的时候，开发者需要对检索出来的结果进行排序，那 SQL 是否能支持呢？答案是肯定的。排序使用“ORDER BY”关键字即可，在它的后面跟上需要排序的列名及其排序方向，如果是升序则用“ASCII”结尾，如果是倒序则用“DESC”结尾。如下语法是把检索出来的用户数据按照年龄从大到小排列：

```
SELECT *
FROM user
ORDER BY age DESC                                    /*排序规则*/
```

注意：如果默认为升序，则省略“ASCII”的关键字。

另外，查询条件也是一个重要项，SQL 提供了 WHERE 语句来支持查询结果的筛选。在 WHERE 语句中，可以使用各种条件表达式，并通过“AND”或“OR”关键字把它们串联起来。WHERE 语句的位置在 FROM 语句以下和 ORDER BY 语句以上。例如，下面的语法为选出年龄大于 20 岁的用户的 SQL 语句：

```
SELECT *
FROM user
WHERE age > 20                                       /*年龄大于 20 岁*/
ORDER BY age DESC
```

另外，如果 WHERE 后面有多个条件，则需要用“AND”或“OR”关键字把每个条件串联起来，它们分别代表“与”和“或”的关系。例如，以下语法为选出年龄大于 30 岁且性别为男性的用户的 SQL 语句：

```
SELECT *
FROM user
WHERE age > 30 AND gender='male'                     /*年龄大于 30 岁，且性别为男*/
ORDER BY age DESC
```

【答案】

SQL 提供了 SELECT 语句来检索数据，它的基本语法如下所示：

```
SELECT <列名>
FROM <表名/视图>
WHERE <筛选条件>
ORDER BY <列名> <ASC/DESC>
```

SELECT 后面跟的是需要选出的列名；FROM 后面指明了查找的对象，一般为表格；WHERE 后面罗列了筛选条件；ORDER BY 则表示排序规则。

面试题 104　如何使用 SQL 更改数据

尽管 SQL 的名称是结构化查询语言，但是它不仅仅拥有数据的查询功能，它还提供了数据的更改功能，包括插入数据、更新数据和删除数据。本例在回答该问题的同时，全面地讲解 SQL 关于增、删和改数据的语法规则。

【出现频率】★★★★

【关键考点】

- INSERT 语句；
- UPDATE 语句；
- DELETE 语句。

【考题分析】

SQL 提供了 INSERT、UPDATE 和 DELETE 3 种语句来支持对数据的更改操作，它们的含义分别是插入数据、更新数据和删除数据。它们的语法各异，但是它们有一个共同点，就是会造成数据的更改。下面对这 3 种语句作详细介绍。

1. INSERT语句

顾名思义，INSERT 是用来插入数据的，它是为数据库的表插入一行数据。插入方式有以下几种：

- 插入完整的行。
- 插入行的一部分。
- 插入某些查询的结果。

插入完整行的时候，无须提供具体的列名，只需要依次把所有列的值给齐就可以了，这些值通过 VALUES 关键字提供。例如，假定一个 user 表只有 3 列：姓名、年龄和性别，插入一个完成的用户数据语法如下所示：

```
INSERT INTO user
VALUES ('zhangsan',30,'male')                    /*依次给出需要的值*/
```

而对于只插入部分数据的时候，需要在表名后面指明仅为哪些列赋值，并用括号包含起来，在“VALUES”后面的括号中提供相应的值，示例如下：

```
INSERT INTO user (name,age)                      /*依次给出需要赋值的列名*/
VALUES ('zhangsan',30)                           /*依次给出需要的值*/
```

当需要把某些查询结果插入到另外一张表的时候，“VALUES”关键字则不再使用，取代它的是一句 SELECT 语句。假如有另外一张表（usercopy）的结构和 user 表相同，则可以通过如下的 SQL 语句把 user 的数据插入到 usercopy 表中：

```
INSERT INTO usercopy
SELECT *                                         /*SELECT 语句*/
```

```
FROM user
```

2. UPDATE语句

UPDATE 语句是对表格中的符合条件的行的某个或某些列的数据进行修改。把需要修改的列名及其新的值放在 SET 关键字后边，多个列需要修改的时候用逗号隔开。如下所示，修改所有用户的性别为男性。

```
UPDATE user
SET gender='male'                                    /*修改性别*/
```

以上的修改是对表格的所有行进行修改，如果仅需要修改某个或某些行的数据，则还需要加上 WHERE 语句，放在后面即可。例如，把姓名为 zhangsan 的用户的年龄修改为 20 岁，可以这样写：

```
UPDATE user
SET age=20
WHERE name='zhangsan'                                /*特指 zhangsan 用户*/
```

3. DELETE语句

DELETE 语句比较简单，它用于删除表格行。在 FROM 关键字后边加上表名，然后在 WHERE 语句中写上删除的条件即可。例如，删除 zhangsan 这个用户的数据，就可以这样来完成：

```
DELETE FROM user
WHERE name = 'zhangsan'                              /*特指删除 zhangsan 用户*/
```

说明：INSERT、UPDATE 和 DELETE 语句涉及到数据的更改，一般会使用数据库的事务服务，以保证数据的完整性。

【答案】

SQL 主要提供了 INSERT、UPDATE 和 DELETE 3 种语句来更改数据库表格的数据，它们的语法格式如下：

```
INSERT INTO <表名>(列名) VALUES(值列表)
UPDATE <表名> SET (键值对) WHERE (条件)
DELETE FROM <表名> WHERE (条件)
```

11.2　JDBC

JDBC 说到底就是一种用于执行 SQL 的 Java API，它是专门为 Java 程序员访问数据库而准备的。各种数据库差异巨大，但是 JDBC 却提供了一种统一的方式来访问数据库。本节将集中讨论有 JDBC 的常见面试题。

面试题 105　JDBC 的工作原理是什么

某些开发者只知道 JDBC 的一些常见 API，例如，Connection、Statement 等，但是却

对 JDBC 的基本运行进制不太了解。这是非常危险的，因为开发者完全不知道底层的驱动代码在做什么的话，是很难控制程序的数据管理层的。本例在回答该问题的同时，全面地讲解 JDBC 的工作原理。

【出现频率】★★★★

【关键考点】

- JDBC 的概念；
- JDBC 的工作原理。

【考题分析】

JDBC（Java Data Base Connectivity，Java 数据库连接）是一种用于执行 SQL 语句的 Java API，可以为多种关系数据库提供统一访问，它由一组用 Java 语言编写的类和接口组成。JDBC 为开发人员提供了一个标准的 API，据此可以构建更高级的工具和接口，使数据库开发人员能够用纯 Java 的 API 编写数据库应用程序。

JDBC 由两部分组成，第一部分是供程序员调用的 API，另一部分是需要数据库厂商实现的 SPI（Service Provider Interface，数据库厂商需要实现的接口），也就是驱动程序。对于 Java 程序员来说，是不可能知道某种数据库（如 MySQL、SQL Server、Oracle）应该如何调用的，或者需要用其他的技术或语言另外写一个接口程序，这是非常麻烦的。SUN 公司就利用 JDBC 技术很好的为程序员解决了这个问题，提供了一系列的 Java 接口给数据库厂商，让他们去实现这些接口，实现部分也就是数据库驱动程序。另一方面，JDBC 也为程序员提供了一系列的 Java API 调用接口，只要数据库厂商提供了该数据库的 JDBC 驱动程序，程序员就可以访问数据库了。如图 11.1 所示，展示了 JDBC 的构架方式。

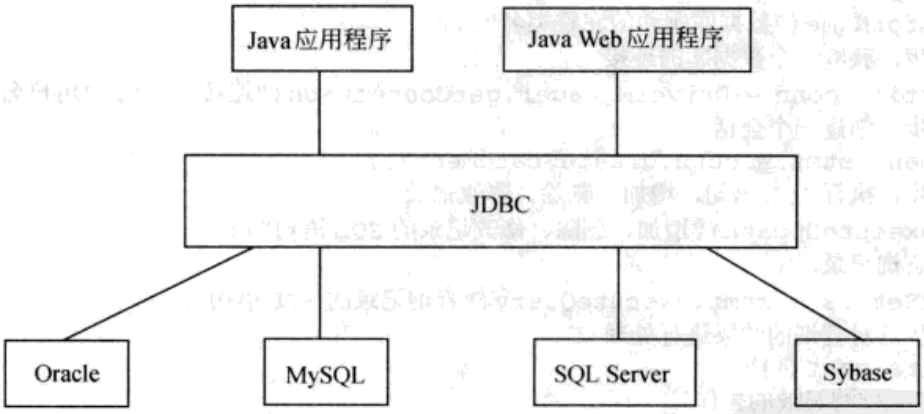


图 11.1　JDBC 的组成和构架示意图

各个厂家的数据库驱动程序是不同的。这些驱动程序一般是免费提供给开发者，读者可以到相应的数据库厂家网站上进行下载。如 MySQL 的驱动程序为一个 jar 包，在 MySQL 的官方网站 http://www.mysql.com 进行下载。笔者的 MySQL 驱动程序文件名为 mysql-connector-java-5.0.8-bin.jar。对于普通的 Java 程序，将这些 jar 包放在 Java 的类搜索路径（CLASSPATH）下即可，对于 Java Web 应用程序则放在 Web 应用目录下面的 WEB-INF/lib 文件夹中。

说明： JDK 包含了一种 JDBC-ODBC 的驱动程序，它是连接至 ODBC 的一种驱动程序，由 SUN 公司提供。如果读者已经存在某种数据库的 ODBC 连接，可以直接使用这种驱动程序。

【答案】

JDBC 采用了一种驱动模式的设计，提供了两套的接口：开发者使用的 API 和数据库厂商使用的 SPI，充分的体现了面向接口编程的好处。程序员无需关心具体数据库的连接和调用，只需要使用 JDK 中提供的标准 API 编程即可，而具体的实现由特定的数据库生产商提供，也就是 JDBC 驱动。

面试题 106　请简述 JDBC 操作数据库的编程步骤

JDBC 提供给开发者的是一系列的接口，这些接口的使用规则相对比较固定且有一些顺序的要求。因此，使用 JDBC 操作数据库的时候，步骤也是比较固定的。本例在回答该问题的同时，全面地讲解 JDBC 操作数据库的编程步骤及其某一步的含义。

【出现频率】★★★★★

【关键考点】

- JDBC 的概念；
- JDBC 的 API 使用方法。

【考题分析】

Java 所提供的关于 JDBC 操作的接口和类，主要集中在 java.sql 和 javax.sql 包下。无论是普通 Java 程序、Java Web 应用程序、EJB 程序或其他 Java 程序，使用 JDBC 操作数据库的步骤都是相对固定的，以下是一个典型的连接数据库的代码。

```
//第一步，注册驱动程序
Class.forName("数据库驱动的完整类名");
//第二步，获取一个数据库的连接
Connection conn = DriverManager.getConnection("连接 URL", "用户名", "密码");
//第三步，创建一个会话
Statement stmt = conn.createStatement();
//第四步，执行 SQL 语句，增加、删除、修改记录
stmt.executeUpdate("增加、删除、修改记录的 SQL 语句");
//或者查询记录
ResultSet rs = stmt.executeQuery("查询记录的 SQL 语句");
//第五步，对查询的结果进行处理
while(rs.next()){
    ... //对记录的操作
}
// 第六步，关闭连接
rs.close();
stmt.close();
conn.close();
```

从以上代码基本可以看出，使用 JDBC 进行数据库的相关操作大致可以分为以下 6 个步骤。

（1）注册驱动程序。就是把驱动程序类加载到 Java 虚拟机中，使得驱动管理器 DriverManager 能够找到该驱动程序，一般通过 Class.forName()进行加载。

（2）获取数据库连接。java.sql.Connection 接口代表的是一个数据库的连接，它通过驱动管理器 DriverManager 来建立连接，并返回一个 Connection 接口的实现。一般，还需要指定连接 URL、用户名和密码。其中连接 URL 会根据不同数据库而各异，它主要的目的

是告诉驱动程序使用特定的协议，连接至指定的数据库（包括 IP 地址和端口）等信息。

（3）创建会话。JDBC 的会话 Statement 主要是用于向数据库发送 SQL 命令，并返回执行后的结果，它由连接 Connection 生成。

在实际开发中，更多的是使用 java.sql.PreparedStatement 会话。它是一种预编译的会话，相当于告诉数据库即将执行的 SQL 语句，如果 SQL 中包含了未知的参数可以用占位符问号“?”进行替代，然后只需要传递参数即可，效率更高一些。另外，使用占位符可以很好的避免 SQL 注入的攻击。

（4）执行 SQL 语句。会话创建好以后，程序员就需要指定需要执行的 SQL 语句了。一般，SQL 分为查询和修改两个。查询主要是 SELECT 语句，使用 executeQuery()方法，它将返回查询后的结果集；修改包括对数据库记录的插入、修改和删除，使用 executeUpdate()方法，它执行以后返回的是影响到的记录数。

（5）处理结果集。如果是查询语句的话，会返回结果集 ResultSet，一般会使用 ResultSet.next()方法，对结果集的每一条数据进行处理。

（6）关闭连接。关闭数据库连接是一个良好的习惯，并且在关闭连接的时候，需要从小到大进行，先关结果集，再关会话，最后才是连接。

以上只是使用 JDBC 的基本步骤，其中第五步是可选的，也可以执行多条 SQL 语句以后再关闭数据库连接。图 11.2 展示了整个使用 JDBC 进行数据库操作的过程。

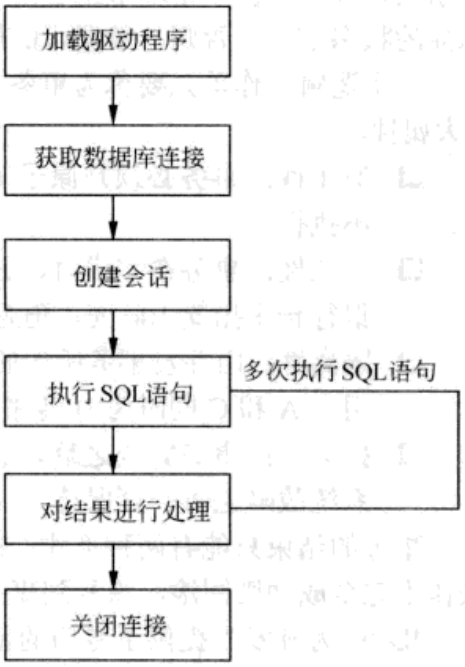


图 11.2　JDBC 进行数据库操作的过程

【答案】

根据以上的分析，JDBC 编程的步骤主要有以下几步：

（1）注册驱动程序。

（2）获取数据库连接。

（3）创建会话。

（4）执行 SQL 语句。

（5）处理结果集。

（6）关闭连接。

面试题 107　如何使用 JDBC 的事务

假设一个银行转账的程序，在某一次转账的过程中，出现一些问题，造成一边已经扣款，而另一边并没有存入相应的金额。这样的错误几乎是毁灭性的，为了避免这类错误的发生，最好的办法就是使用事务。JDBC 提供了充分的事务的支持。本例在回答该问题的同时，全面地讲解事务的含义和 JDBC 使用事务的方法。

【出现频率】★★★★

【关键考点】

- JDBC 的概念；
- JDBC 的 API 使用方法。

【考题分析】

事务，也是数据库事务，指的是作为单个逻辑工作单元执行的一系列操作。正常的情况下，操作应该顺利进行，与操作相关的所有数据库信息也成功地更新。但是，如果在这一系列过程中任何一个环节出了差错，导致操作失败了，数据库中所有信息都必须保持操作前的状态不变。否则，数据库的信息将会一片混乱而不可预测。

一个逻辑工作单元要称为事务，必须满足 ACID（原子性、一致性、隔离性和持久性）4 大属性。

- 原子性：事务必须是原子工作单元；对于数据的修改，要么全都执行，要么全都不执行。
- 一致性：事务在完成时，必须使所有的数据都保持一致状态。例如，不能出现 A 银行转账扣款 100 元，但是 B 银行的账号只增加了 10 元。
- 隔离性：由并发事务所作的修改必须与任何其他并发事务所作的修改隔离。例如，用户 A 和 C 同时为 B 转账，一定不能同时对用户 B 的账户余额进行更改。
- 持久性：事务完成之后，它对于系统的影响是永久性的。该修改即使出现致命的系统故障也将一直保持。

事务的结束只能有两种形式：提交和回滚。操作完全成功则提交，产生永久性的修改；操作不完全成功则回滚，恢复到事务开始前的状态。它们都将结束一个事务。

JDBC 为开发者提供了专门的使用事务的 API。默认情况下，JDBC 使用自动提交事务的方式，也就是说，一旦执行 SQL 语句完成，就提交事务。但是，在多数情况下，开发者需要控制事务的过程，如果发生了某个异常或者参数出现问题以后，能够回滚事务。以下步骤是一个典型的使用 JDBC 事务的过程。

（1）关闭自动提交事务。通过设置连接的自动提交事务属性为 false，示意代码如下：

```
Connection conn = DriverManager.getConnection("连接 URL", "用户名", "密码");
conn.setAutoCommit(false);                    //关闭自动提交事务
```

（2）捕获（try catch）执行代码。如果执行过程顺利，提交事务；一旦发生异常，回滚（rollback）事务，示意代码如下：

```
try {
    conn.setAutoCommit(false);                //关闭自动提交事务
    stmt = conn.createStatement();            //创建会话
    stmt.executeUpdate("sql");                //执行增删改的 SQL 命令
    conn.commit();                            //提交事务
} catch (Exception e) {
    e.printStackTrace();
    conn.rollback();                          //回滚事务
}
```

（3）关闭连接。该步骤最好是放在 finally 代码块中，这样可以确保关闭连接的操作执行了。

```
    finally{
        if(stmt != null)                    //判断会话是否为空
            stmt.close();                   //关闭会话
        if(conn != null)                    //判断连接是否为空
            conn.close();                   //关闭连接
        }
```

注意：以上介绍的只是普通的数据库事务，Java EE 还支持分布式事务等高级事务服务，有兴趣的读者可以找这方面的资源研究一下。

【答案】

JDBC 的事务主要在是在代码中控制的，关键点在于：关闭自动提交事务、调用 commit() 方法提交事务和调用 rollback()方法回滚事务。一般来说，JDBC 中使用事务服务大致有以下一些步骤。

（1）关闭自动提交事务，设置连接的自动提交事务属性为 false。

（2）捕获（try catch）执行代码。如果执行过程顺利，提交事务，一旦发生异常，回滚（rollback）事务。

（3）关闭连接。

面试题 108　如何使用 JDBC 实现数据访问对象层（DAO）

分层开发应用程序，是目前比较提倡的一种开发方式。DAO 层（数据访问对象层）是涉及数据访问的一层，它往往处于业务层之下，数据库之上，为业务层的对象提供访问数据的接口。本例在回答该问题的同时，详细地讲解如何使用 JDBC 技术实现 DAO 层的原理和步骤。

【出现频率】★★★★

【关键考点】

- DAO 层的作用；
- JDBC 关于数据的增、删、查改操作的实现原理。

【考题分析】

Java 是面向对象的语言，开发者在操作数据的时候，通常更习惯面对一个特定类型的对象，如一个用户就是一个 User 类的对象。DAO 层需要做的，就是为上层提供充分的对象支持，让上层再也看不到具体的数据，而是一个个活生生的对象。

增加、删除、查询和修改操作是 DAO 需要做的最基本的 4 项操作。查询一般需要提供遍历查询和 id 查询，对于遍历查询，DAO 需要提供 User 泛型的 List 对象，对于 id 查询则提供已经装配好数据的 User 对象。至于增加和修改操作，上层一般会提供一个 User 对象，DAO 把 User 对象中的数据使用 INSERT 语句插入到表格中。删除操作则只需要提供一个 id 即可。

以下是一段针对用户操作的 DAO 实现的示例代码：

```
package ch11;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
```

```
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
//用户类
class User{
    private long id;                                          //主键
    private String name;                                      //姓名
    private String gender;                                    //性别
    //构造方法
    public User() {
        super();
    }
    public User(long id, String name, String gender) {
        super();
        this.id = id;
        this.name = name;
        this.gender = gender;
    }
    //setters and getters...
    ...
}
//DAO类
public class JdbcDao {
    static{
        try {
            Class.forName("com.mysql.jdbc.Driver");     //加载驱动
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
    //获取连接
    private Connection getConn(){
        try {
            return DriverManager.getConnection("jdbc:mysql://localhost:
            3306/test",
                                               "root", "password");
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return null;
    }
    //释放资源
    private void release(ResultSet rs,Statement ps,Connection conn){
        if(rs!=null){
            try {
                rs.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if(ps!=null){
            try {
                ps.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if(conn!=null){
```

```
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
    //用 Id 获取用户对象
    public User getUserById(long id){
        ResultSet rs = null;
        PreparedStatement ps = null;
        Connection conn = null;
        String sql = "select * FROM user WHERE id=?";          //SQL 语句
        try {
            conn = this.getConn();
            ps = conn.prepareStatement(sql);
            ps.setLong(1, id);
            rs = ps.executeQuery();
            if(rs.next()){
                //如果存在，则直接构建并返回用户对象
                User user =
                    new User(rs.getLong("id"),rs.getString("name"),rs.
                    getString("gender"));
                return user;
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally{
            this.release(rs, ps, conn);
        }
        return null;
    }
    //查询所有用户
    public List<User> getAllUses(){
        List<User> list = new ArrayList<User>();
        ResultSet rs = null;
        PreparedStatement ps = null;
        Connection conn = null;
        String sql = "select * FROM user";                     //SQL 语句
        try {
            conn = this.getConn();
            ps = conn.prepareStatement(sql);
            rs = ps.executeQuery();
            //循环添加用户对象
            while(rs.next()){
                User user =
                    new User(rs.getLong("id"),rs.getString("name"),rs.
                    getString("gender"));
                list.add(user);
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally{
            this.release(rs, ps, conn);
        }
        return list;
    }
    //修改用户数据
    public User updateUser(User user){
        PreparedStatement ps = null;
```

```
        Connection conn = null;
        String sql = "update user set id=?,name=?,gender=?";
        try {
            conn = this.getConn();
            conn.setAutoCommit(false);
            ps = conn.prepareStatement(sql);
            ps.setLong(1, user.getId());
            ps.setString(2, user.getName());
            ps.setString(3, user.getGender());
            int rst = ps.executeUpdate();                    //执行更新
            if(rst > 0){
                return user;
            }
            conn.commit();                                   //提交事务
        } catch (Exception e) {
            e.printStackTrace();
            try {
                conn.rollback();                             //回滚
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
        } finally{
            this.release(null, ps, conn);
        }
        return null;
    }
    //根据 ID 删除用户
    public boolean deleteUser(long id){
        PreparedStatement ps = null;
        Connection conn = null;
        String sql = "delete FROM user WHERE id=?";          //SQL 语句
        try {
            conn = this.getConn();
            conn.setAutoCommit(false);
            ps = conn.prepareStatement(sql);
            ps.setLong(1, id);
            int rst = ps.executeUpdate();                    //执行更新
            if(rst > 0){
                return true;
            }
            conn.commit();                                   //提交事务
        } catch (Exception e) {
            e.printStackTrace();
            try {
                conn.rollback();                             //回滚
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
        } finally{
            this.release(null, ps, conn);
        }
        return false;
    }
    //插入用户数据
    public User insertUser(User user){
        PreparedStatement ps = null;
        Connection conn = null;
        String sql = "insert into user values(?,?,?)";
        try {
            conn = this.getConn();
```

```
            conn.setAutoCommit(false);
            ps = conn.prepareStatement(sql);
            ps.setLong(1, user.getId());
            ps.setString(2, user.getName());
            ps.setString(3, user.getGender());
            int rst = ps.executeUpdate();                    //执行更新
            if(rst > 0){
                return new User(user.getId(),user.getName(),user.
                getGender());
            }
            conn.commit();                                   //提交事务
        } catch (Exception e) {
            e.printStackTrace();
            try {
                conn.rollback();                             //回滚
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
        } finally{
            this.release(null, ps, conn);
        }
        return null;
    }
}
```

说明：以上的示例代码只是一个简单的举例说明。根据 JDBC 实现 DAO 的思路，读者可以将其进一步完善，用于更多的数据对象访问需要。

【答案】

DAO 的设计意图就是让上层对待底层数据的时候，能够再用一个对象的眼光来看待。因此，DAO 要做的事情主要就是把数据包装成对象和把对象拆分成数据。增、删、查、改是 DAO 需要做的最基本的 4 种操作，它们的实现可以参照以上的示例。除此之外，如果开发者还需要为 DAO 类添加其他方法，也是需要体现一种对象观的概念。

面试题 109　如何使用连接池技术

对于操作数据库比较多程序来说，频繁的创建数据库连接是非常耗费资源的，一旦某一时刻用户的请求太多，还有可能造成系统缓慢甚至瘫痪。为了解决这一问题，就提出了数据库连接池技术。本例在回答该问题的同时，详细地讲解 Java 应用程序如何使用连接池技术。

【出现频率】★★★★

【关键考点】

- 连接池技术的含义；
- DataSource 的使用方法。

【考题分析】

数据库连接池就好像一个池子一样，这个池子中装的是数据库的连接（Connection）。程序员需要连接数据库的时候，只需要从池子中取出一个即可。当程序员调用

Connection.close()方法的时候，这个连接就返回到池子中，而没有真正的与数据库断开连接。当连接不够用的时候，它会创建一个新的连接；同理，当连接太多以后，它会自动关闭一些不必要的连接，如图 11.3 所示。

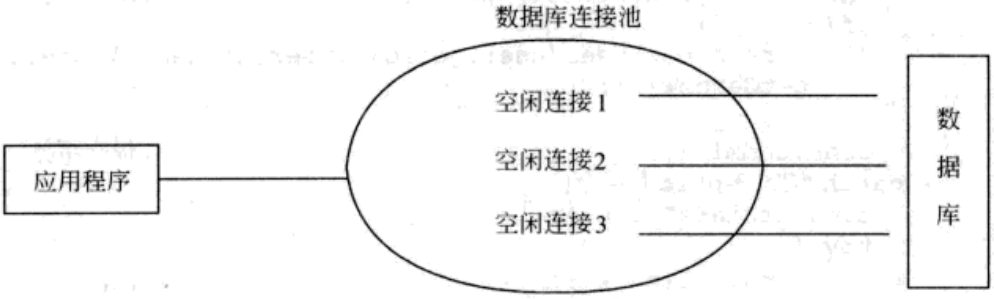


图 11.3　数据库连接池工作原理图

数据库连接池往往是作为一个单独的程序模块进行运行，由它来维护这些连接，程序员可以通过配置来控制它的一些属性，例如，最大活动连接数、最大空闲连接数、连接超时等。它与传统的 JDBC 提供连接的方式不太一样，程序员必须使用数据源（Data Source）的形式获取连接，数据源对象往往是以 JNDI（Java Naming and Directory Interface，Java 命名与目录接口）的形式提供给程序员。

数据库连接池提供商对连接池的实现是各不相同的，使用方式也会有差异，但是它们都必须实现 javax.sql.DataSource 接口，换句话说，开发者只需要面向这些接口编程就好了。对于一些 Java EE 服务器，例如 Tomcat、JBOSS 等，它们本身自带有的连接池，Web 应用程序和 EJB 程序直接使用这些连接池就好。

对于开发者来说，使用连接池和传统 JDBC 的主要差异在于获取连接的方式不同。传统的 JDBC 是通过驱动管理器（DriverManager）来获取连接，而连接池则需要用数据源（DataSource）来获取，示例代码如下：

```
//创建 JNDI 初始化上下文对象
InitialContext cxt = new InitialContext();
//通过 JNDI 上下文获取到数据源
DataSource ds = (DataSource) cxt.lookup("数据源在 JNDI 上的路径");
//通过数据源获取到对象
Connection conn = ds.getConnection();
```

【答案】

数据库连接池技术是为了避免重复创建连接而设计的，它作为一个单独的程序模块进行运行，负责维护池子中的数据库的连接（Connection）。程序员打开连接和关闭连接并不会造成真正意义上的连接创建和关闭，而只是连接池对连接对象的一种维护手段。

对于开发者来说，连接池与传统的 JDBC 提供连接的方式不太一样，程序员必须使用数据源（Data source）的形式获取连接池的连接，而数据源对象往往是以 JNDI 的形式而提供的。对于 Java Web 和 EJB 开发人员来说，需要参考一下具体的 JavaEE 服务器关于连接池的使用手册。

面试题 110　如何使用可滚动的结果集

JDBC 的结果集（ReslutSet）最开始是不支持可滚动的，从 JDBC 2.0 开始才支持可滚动的结果集。那么应该如何使用可滚动的结果集呢？本例在回答该问题的同时，将更详细地讲解可滚动结果集的使用细节。

【出现频率】★★★★

【关键考点】

- JDBC 的原理；
- 可滚动结果集的使用方法。

【考题分析】

一般说来，开发人员使用 ResultSet 类中的 next()方法可以迭代遍历结果集中的所有行。但是，有的时候会希望在结果集上前后移动，在 JDBC 1.0 的时候，并未提供 previous 方法，为了实现向后遍历，程序员不得不手动缓存结果集中的数据。

从 JDBC 2.0 以后，ResultSet 就可以滚动了，可以在结果集上前后移动并且可以跳转到结果集中的任何位置。那么，如何指定一个结果集是否为可滚动的呢？这是在创建会话对象的时候进行指定的，大致语法如下：

```
Statement stmt = conn.createStatement(sql, type, concurrency);
PreparedStatement pstmt = conn.preparedStatement(sql, type,
concurrency);
```

用变量 type 来设置是否是可滚动的结果集，可以在以下几个 ResultSet 类的静态常量中选择。

- TYPE_FORWARD_ONLY：结果集不能滚动。
- TYPE_SCROLL_INSENSITIVE：结果集可以滚动，但是对数据库变化不敏感，数据库查询生成结果集后发生了变化，结果集不发生变化。
- TYPE_SCROLL_SENSITIVE：结果集可以滚动，但是对数据库变化敏感。

注意：concurrency 变量时用于指定是否为可更新的结果集。

以下为一个更滚动结果集的示例。

```
package ch11;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class RollableTest {                                    //测试类
    public static void main(String[] args) throws Exception {  //主方法
        ResultSet rs = null;                                   //声明结果集变量
        PreparedStatement ps = null;                           //声明会话变量
        Connection conn = null;                                //声明连接变量

        try {
```

```
            Class.forName("com.mysql.jdbc.Driver");//加载驱动
            conn = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/test", "root", "password");
            String sql = "select * FROM user";          //SQL 语句
            ps = conn.prepareStatement(sql,             //创建可滚动结果集的会话
                ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_READ_ONLY);
            rs = ps.executeQuery();                     //执行查询
            rs.next();                                  //得到第一行数据
            System.out.println(rs.getInt(1));
            rs.last();                                  //来到最后一行
            System.out.println(rs.getString(1));
            System.out.println(rs.isLast());            //判断是否为最后一行
            System.out.println(rs.isAfterLast());       //判断是否还有数据
            System.out.println(rs.getRow());            //判断当前的行号
            rs.previous();                              //回一行
            System.out.println(rs.getString(1));
            rs.absolute(6);                             //直接定位到第 6 条
            System.out.println(rs.getString(1));
        } catch (Exception er) {
            er.printStackTrace();
        } finally {
            if (rs != null) {                           //关闭结果集
                try {
                    rs.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            if (ps != null) {                           //关闭会话
                try {
                    ps.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            if (conn != null) {                         //关闭连接
                try {
                    conn.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

【答案】

可滚动结果集是 JDBC 2.0 才支持的功能，它运行结果集的指针任意游动。它的使用方法比较简单，在创建会话的时候，为它指定结果集的可滚动性，包括以下 3 种。

- TYPE_FORWARD_ONLY；
- TYPE_SCROLL_INSENSITIVE；
- TYPE_SCROLL_SENSITIVE。

面试题 111　如何使用可更新的结果集

JDBC 的结果集（ReslutSet）最开始是不支持可更新的，从 JDBC 2.0 开始才支持可更新的结果集，它支持直接在查询出来的结果里边修改数据，并且同步到数据库中。那么应该如何使用可更新的结果集呢？本例在回答该问题的同时，详细地讲解可更新结果集的使用细节。

【出现频率】★★★★

【关键考点】

- JDBC 的原理；
- 可更新结果集的使用方法。

【考题分析】

与可滚动的结果集类似，可更新结果集的设置地方也是在创建会话的时候，它的语法如下：

```
Statement stmt = conn.createStatement(sql, type, concurrency);
PreparedStatement pstmt = conn.preparedStatement(sql, type,
concurrency);
```

变量 type 是用来设置是否可滚动的结果集，而设置是否可更新则使用 concurrency 变量，它包含以下两种值。

- CONCUR_READ_ONLY：结果集不能用于更新数据库；
- CONCUR_UPDATABLE：结果集可以用于更新数据库。

注意：实际上，数据库驱动程序可能无法支持对可滚动或可更新的请求。使用 DatabaseMetaData 类中的 supportsResultSetType 和 supportsResultSetConcurrency 方法，可以获知某个数据库究竟支持哪些结果集类型以及那些模式。一个复杂的查询的结果集往往是不可更新的结果集。

以下为一个更新结果集的示例：

```
package ch11;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class RollableTest {                                    //测试类
    public static void main(String[] args) throws Exception {  //主方法
        ResultSet rs = null;                                   //声明结果集变量
        PreparedStatement ps = null;                           //声明会话变量
        Connection conn = null;                                //声明连接变量

        try {
            Class.forName("com.mysql.jdbc.Driver");            //加载驱动
            conn = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/test", "root", "password");
            String sql = "select * FROM user";                 //SQL 语句
```

```
            ps = conn.prepareStatement(sql,                    //创建可滚动结果集的会话
                ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_READ_ONLY);
            rs = ps.executeQuery();                            //执行查询
            rs.next();
            // 更新一行数据
            rs.updateString(2, "AAAA");                        //修改值
            rs.updateRow();                                    //更新
        } catch (Exception er) {
            er.printStackTrace();
        } finally {
            if (rs != null) {                                  //关闭结果集
                try {
                    rs.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            if (ps != null) {                                  //关闭会话
                try {
                    ps.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            if (conn != null) {                                //关闭连接
                try {
                    conn.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

以上程序运行以后，就可以发现数据库中的 User 表的第一行已经被修改了。可更新结果集具有很高的灵活性，经常用于小表格的修改和插入操作。

【答案】

可更新结果集是 JDBC 2.0 才支持的功能，它程序员对查询出来的结果进行修改，并同步到数据库。它的使用方法比较简单，在创建会话的时候，为它指定结果集的可更新性，包括以下两种。

- CONCUR_READ_ONLY;
- CONCUR_UPDATABLE。

11.3　JDBC 操作各类数据源

大家知道，数据库有很多种，JDBC 在使用它们的时候需要使用不同的驱动程序和不同的 URL，或者使用 API 的方式也会有差异，了解它们的差别是很有必要的。本节将集中讨论有 JDBC 使用常见数据源的面试题。

面试题 112　如何使用 JDBC 操作 Oracle 数据库

Oracle 是关系数据库的老大，它在业界的使用非常普遍，尤其是一些大型的 Java EE 系统，都是使用 Oracle 数据库，因此 JDBC 使用 Oracle 是很有必要的。本例在回答该问题的同时，详细地讲解 JDBC 使用 Oracle 的使用细节。

【出现频率】★★★★

【关键考点】

- JDBC 的原理；
- Oracle 的驱动程序和连接 URL。

【考题分析】

大家知道，JDBC 是一套标准的 Java 访问数据库的 API，它的具体实现依赖于具体数据库厂商。Oracle 作为数据库第一大厂商，肯定是对 JDBC 提供了充分的支持。

首先，需要从 Oracle 的官方网站上下载 JDBC 的驱动 jar 文件，地址为：http://www.oracle.com/technology/software/tech/java/sqlj_jdbc/index.html，包含了各种 Oracle 的版本，下载以后得到 ojdbc14.jar 文件，把它放在应用程序中的 CLASSPATH 下，就可以使用了。

说明：不同类型的 Java 程序，放置驱动文件的路径不一样，请参考具体的使用手册。

通过 JDBC 获得 Oracle 数据库连接，有下面 3 种方式：

（1）OCI 方式。

（2）Thin 方式。

（3）JDBC-ODBC 桥方式。

OCI 方式依赖于本地的动态链接库，如果在本地安装了 Oracle 数据库客户端可以采用该方式；而 Thin 方式为纯 Java 的数据库连接方式；JDBC-ODBC 桥方式依赖于本地 ODBC 数据库源的配置，这种方式一般不太被采用。

最常用的方式为第二种，它无需安装 Oracle 的客户端，而且速度也不会太慢，它的连接 URL 的格式如下：

```
jdbc:oracle:thin:@<server>[:<1521>]:<database_name>
```

在“<server>”的地方，填写数据库的 IP 地址，端口号默认为 1521，最后以数据库的名称结尾，以下是一个连接 URL 的示例：

```
jdbc:oracle:thin:@localhost:1521:tiger
```

它的含义代表的是用 thin 的形式，连接本地的端口为 1521 的 Oracle 数据库，默认使用 Tiger 数据库。接下来的使用方法就是标准的 JDBC 的 API 使用，可以进行任意的数据库操作，例如增、删、查、改操作。

【答案】

JDBC 连接和使用 Oracle 数据库的时候，遵循 JDBC 标准的 API 使用方式，它与其他数据库的区别主要在于驱动文件和连接 URL 的不同。以下是一个 JDBC 连接 Oracle 数据

库的例子：

```
package ch11;                                                    //包名
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
//主类
public class Oracle {
    public static void main(String args[]) {                    //主方法
        try {
            String strurl = "jdbc:oracle:thin:@localhost:1521:tiger";
                                                                 //连接字符串
            Class.forName("oracle.jdbc.driver.OracleDriver"); //加载驱动类
            Connection conn = DriverManager.getConnection(strurl);
                                                                 //获取连接
            Statement stmt = conn.createStatement();             //创建会话
            ResultSet rs = stmt.executeQuery("select * FROM books");
                                                                 //执行查询操作
            //操作结果
            if (rs.next()) {
                System.out.println(rs.getString("name"));        //打印结果
            }
            conn.close();                                        //关闭连接
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

面试题 113　如何使用 JDBC 操作 MySQL 数据库

MySQL 是开源数据库的老大，它广泛地应用在中小型的 JavaEE 项目中，求职者应该对 JDBC 使用 MySQL 的使用方法有所了解。本例在回答该问题的同时，详细地讲解 JDBC 使用 MySQL 的使用细节。

【出现频率】★★★★

【关键考点】

- JDBC 的原理；
- MySQL 的驱动程序和连接 URL。

【考题分析】

MySQL 现在是 SUN 公司旗下的一款开源产品，它对 JDBC 的支持可见一斑，可以在 Sun 公司的下载文档和驱动文件。

与 Oracle 一样，首先需要从 MySQL 的官方网站上下载 JDBC 的驱动 jar 文件，地址为 http://www.mysql.com，现在版本是 5.1.10。下载以后得到 mysql-connector-java-5.1.10.zip 文件，把它包含的 jar 文件放在应用程序中的 CLASSPATH 下，就可以使用 MySQL 的 JDBC 驱动了。

说明： MySQL 的 JDBC 连接只有一种形式，不存在其他连接差异。

MySQL 的连接 URL 的格式如下：

```
jdbc:mysql://<hostname>[<:3306>]/<dbname>
```

在< hostname >处，填写数据库的 IP 地址，端口号默认为 3306，最后以数据库的名称结尾。以下是一个连接 URL 的示例：

```
jdbc:mysql://localhost:3306/test
```

它的含义代表的是连接本地的端口为 3306 的 MySQL 数据库，默认使用 test 数据库。接下来的使用方法就是标准的 JDBC 的 API 使用了，可以进行任意的数据库操作，例如增、删、查、改操作。

【答案】

JDBC 连接和使用 MySQL 数据库的时候，遵循 JDBC 标准的 API 使用方式，它与其他数据库的区别主要在于驱动文件和连接 URL 的不同。以下是一个 JDBC 连接 MySQL 数据库的例子：

```
package ch11;                                                    //包名
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
//主类
public class MySQL {
    public static void main(String args[]) {                    //主方法
        try {
            String strurl = " jdbc:mysql://localhost:3306/test ";
                                                                 //连接字符串
            Class.forName("com.mysql.jdbc.Driver ");             //加载驱动类
            Connection conn = DriverManager.getConnection(strurl);
                                                                 //获取连接
            Statement stmt = conn.createStatement();             //创建会话
            //执行查询操作
            ResultSet rs = stmt.executeQuery("select * FROM books");
            //操作结果
            if (rs.next()) {
                System.out.println(rs.getString("name"));        //打印结果
            }
            conn.close();                                        //关闭连接
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

面试题 114　如何使用 JDBC 操作 SQL Server 数据库

SQL Server 是另外一个常用的数据库软件，它是微软的产品，但是也对 JDBC 提供了支持。在一些推荐使用 SQL Server 的软件公司中，是需要求职者对 JDBC 使用 SQL Server 有所了解的。本例在回答该问题的同时，详细地讲解 JDBC 使用 SQL Server 的使用细节。

【出现频率】★★★★

【关键考点】

- JDBC 的原理；
- SQL Server 的驱动程序和连接 URL。

【考题分析】

与其他数据库一样，首先需要从微软官方网站上下载 JDBC 的驱动 jar 文件，地址为 http://www.microsoft.com/downloads/details.aspx?familyid=e22bc83b-32ff-4474-a44a-22b6ae2c4e17&displaylang=zh-cn，把它的 JDBC 驱动的 jar 文件放在应用程序中的 CLASSPATH 下，就可以使用 SQL Server 的 JDBC 驱动了。

说明：SQL Server 不同的版本的驱动文件是不同的，例如，SQL Server 2000 和 2005 就需要不同的驱动文件。另外，如果是 SQL Server 2000，还需要安装 SP3 补丁，才能使用 JDBC 来连接 SQL Servler。

SQL Server 的连接 URL 的格式如下所示：

```
jdbc:microsoft:sqlserver://<server_name>:<1433>;DatabaseName=<db>
```

在<server_name>处，填写数据库的 IP 地址，端口号默认为 1433，最后以数据库的名称结尾，以下是一个连接 URL 的示例：

```
jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=Northwind
```

它的含义代表的是连接本地的端口为 1433 的 SQL Server 数据库，默认使用 Northwind 数据库。接下来的使用方法就是标准的 JDBC 的 API 使用了，可以进行任意的数据库操作，例如，增、删、查、改操作。

【答案】

JDBC 连接和使用 SQL Server 数据库的时候，遵循 JDBC 标准的 API 使用方式，它与其他数据库的区别主要在于驱动文件和连接 URL 的不同。以下是一个 JDBC 连接 SQL Server 数据库的例子。

```
package ch11;                                                    //包名
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
//主类
public class SQLServer {
    public static void main(String args[]) {                    //主方法
        try {
            //连接字符串
            String strurl
                ="jdbc:microsoft:sqlserver://localhost:1433;DatabaseName
                =Northwind ";
            //加载驱动类
        Class.forName("com.microsoft.jdbc.sqlserver.SQLServer-
        Driver");
            Connection conn = DriverManager.getConnection(strurl);
                                                                 //获取连接
            Statement stmt = conn.createStatement();             //创建会话
```

```
            ResultSet rs = stmt.executeQuery("select * FROM books");
                                                          //执行查询操作
            //操作结果
            if (rs.next()) {
                System.out.println(rs.getString("name"));        //打印结果
            }
            conn.close();                                        //关闭连接
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

面试题 115　如何使用 JDBC 操作 Access

Access 是微软的 Office 系列软件中的一款，它也算是关系数据库的一种，是以文件的形式来保存数据的。那么，Java 可以用 JDBC 的 API 来操作 Access 吗？答案是肯定的。本例在回答该问题的同时，详细地讲解 JDBC 使用 Access 的使用细节。

【出现频率】★★★★

【关键考点】

- JDBC 的原理；
- JDBC-ODBC 桥驱动的原理；
- OCBC 使用 Access 的方法。

【考题分析】

Access 是一类功能很弱的，以文件的形式来提供关系数据库支持的软件，用户可以直接使用 Office 中的 Access 软件来操作这些数据。因为它太简单了，所以微软并没有提供它的 JDBC 支持，如果 Java 程序员要使用 Access 的话，可以使用 JDBC-ODBC 桥驱动来间接的使用 Access。

ODBC 是微软推出的一种屏蔽各种数据库差异的标准，所有 Windows 操作系统都会自带 ODBC。JDBC-ODBC 桥驱动是 SUN 公司自己实现的一种 JDBC 驱动，它假设 ODBC 就是一个数据源，在 Java 语言发展的早期，正是它巧妙的实现了多种数据平台的统一 API 访问的功能。

说明：JDBC-ODBC 桥驱动已经内嵌在 JDK 中，所以无需单独提供。

首先，在访问 Access 之前，需要在 ODBC 中加入 Access 数据源，也就是 Access 文件。然后才能使用 JDBC-ODBC 桥驱动来访问 Access。然后，就可以使用标准的使用 JDBC 的方式来访问数据了。

JDBC-ODBC 的连接 URL 的格式如下：

```
jdbc:odbc:<alias>
```

在<alias>的地方，填写的就是 Access 文件在 ODBC 的数据源名称，以下是一个连接 URL 的示例：

```
jdbc:odbc:test
```

它的含义代表的是使用名称为 test 的一个 ODBC 数据源。接下来的使用方法就是标准

的 JDBC 的 API 使用了，可以进行任意的数据库操作，例如增、删、查、改操作。

【答案】

JDBC 连接和使用 Access 的时候，依然遵循 JDBC 标准的 API 使用方式，它与其他数据库的区别主要在于驱动文件和连接 URL 的不同，它使用的是 JDBC-ODBC 桥驱动，在使用之前还需要进行 ODBC 的相关配置。以下是一个 JDBC 访问 Access 的例子：

```
package ch11;                                                    //包名
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
//主类
public class Access {
    public static void main(String args[]) {                    //主方法
        try {
            String strurl = "jdbc:odbc:test";                    //连接字符串
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); //加载驱动类
            Connection conn = DriverManager.getConnection(strurl);
                                                                 //获取连接
            Statement stmt = conn.createStatement();             //创建会话
            ResultSet rs = stmt.executeQuery("select * FROM books");
                                                                 //执行查询操作
            //操作结果
            if (rs.next()) {
                System.out.println(rs.getString("name"));   //打印结果
            }
            conn.close();                                        //关闭连接
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

面试题 116　如何使用 JDBC 操作 Excel

Excel 是微软的 Office 系列软件中的一款关于报表的工具，它有的时候起到了简单数据库表格的作用，是以文件的形式来保存数据的。那么，Java 可以用 JDBC 的 API 来操作 Excel 吗？答案是肯定的。本例在回答该问题的同时，详细地讲解 JDBC 使用 Excel 的使用细节。

【出现频率】★★★★

【关键考点】

- JDBC 的原理；
- JDBC-ODBC 桥驱动的原理；
- OCBC 使用 Excel 的方法。

【考题分析】

与 Access 类似，Java 程序要想使用 JDBC 来访问 Excel 文件，也可以使用 JDBC-ODBC 桥驱动。Windows 下的 ODBC 是预装了 Excel 的驱动程序的，只需要进行相关配置就可以使用了。

说明： Java 其实还可以通过其他途径来使用 Excel，只不过 JDBC-ODBC 会更符合 Java 程序员的习惯一些。

同样，在访问 Excel 之前，需要在 ODBC 中加入 Excel 数据源，也就是 Excel 文件，然后才能使用 JDBC-ODBC 桥驱动来访问 Excel。具体的数据操作代码就按照标准的 JDBC 的 API 来进行就好。

【答案】

JDBC 连接和使用 Excel 的时候，依然遵循 JDBC 标准的 API 使用方式，它与其他数据库的区别主要在于驱动文件和连接 URL 的不同，它使用的是 JDBC-ODBC 桥驱动，在使用之前还需要进行 ODBC 的相关配置，以下是一个 JDBC 访问 Excel 的例子：

```
package ch11;                                                    //包名
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
//主类
public class Excel {
    public static void main(String args[]) {                    //主方法
        try {
            String strurl = "jdbc:odbc:testExcel";               //连接字符串
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); //加载驱动类
            Connection conn = DriverManager.getConnection(strurl);
                                                                 //获取连接
            Statement stmt = conn.createStatement();             //创建会话
            ResultSet rs = stmt.executeQuery("select * FROM books");
                                                                 //执行查询操作
            //操作结果
            if (rs.next()) {
                System.out.println(rs.getString("name"));        //打印结果
            }
            conn.close();                                        //关闭连接
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

11.4　小　　结

本章讲解了一些 Java 的数据库操作的面试题。首先讲解了一些数据库和 SQL 的基础知识，包括 SQL 的基本语法、如何使用 SQL 等。然后就 JDBC 几类比较常见的面试题进行了讲解和解答，包括：JDBC 的基本原理、操作步骤和连接池技术等。JDBC 是 Java 操作数据库的基本工具，它 API 的设计无论是对于开发者，还是数据库厂商都是非常清晰和易使用的。作为一般的 Java 程序开发者，熟悉 JDBC 的相关 API 是一项基本技能，而且对数据库及其 SQL 的学习也是必不可少的。理论加上实践才是学习 Java 的数据库相关操作的真正有效办法。

第 4 篇　Java EE 相关问题

第 12 章　Web 开发相关技术

Java Web 开发是目前 Java 应用最多的一个领域，关于 Web 开发的面试题往往也是最多的。甚至有的时候，人们常说的 Java 开发，就特指 Java 的 Web 开发。在 Web 开发中，技术多而且成熟，开发者不容易掌握全面和透彻。对于比较核心一点的知识，包括 Servlet、JSP、MVC 等方面，是面试的重点。因为这些知识是高级应用的基础，通过这些知识点的考察就可以看出求职者在 Web 开发方面的功底。本章将包含关于 Java 的 Web 开发的一些常见面试题，并且分析这些题目和知识点，帮助读者梳理这些方面的知识。

12.1　Servlet 与 Web 容器

熟悉 Java 开发的人都知道，在 Java 开发中，无时无刻不充斥着各种规范，Web 开发也不例外。Java 开发的 Web 程序是必须符合 Java EE 规范的，该规范指出 Web 应用程序需要部署到 Web 容器中才能运行，Web 容器也是根据 Java EE 相关的规范进行开发而成，正因为如此，一个 Java Web 应用程序开发完成以后，可以在不同的符合该规范的 Web 容器中运行。Servlet 是 Web 容器的最基本组成单元，是最基础的 Java Web 技术。本节将集中讨论有关 Web 容器和 Servlet 的常见面试题。

面试题 117　一个 Web 应用程序应该遵守哪些规范

一个 Java Web 应用程序为了能够跨服务器运行，需要遵守一些 Java EE 规范。这些规范既需要得到服务器软件的支持，也必须得到开发者的支持才行。这些规范包括哪些内容呢？本例在回答该问题的同时，详细地讲解 Web 程序规范及其 Web 容器的相关知识。

【出现频率】★★★★

【关键考点】

- Web 应用程序规范；
- Web 容器是如何对待应用程序的。

【考题分析】

Web 容器包含着大小不一的若干个 Java Web 应用程序。这些应用程序也需要部署到 Web 容器中才能正常工作。Web 应用程序的目录结构和文件存放方式是有一定规定的，只有符合这些规定的应用程序才能正常运行在 Web 容器中，表 12.1 列出了一个标准的 Java Web 应用程序所需的目录结构和文件存放方式及其相关说明。

表 12.1　Java Web应用程序所需的目录结构

属　性	说　明
WEB-INF/	对于用户来说该文件夹是不可见的，里面存放着 class 文件、jar 文件和配置文件
WEB-INF/web.xml	web.xml 是整个 Web 应用程序的描述文件，通过它配置该应用程序的信息资源，如 Servlet、过滤器、监听器、系统参数等
WEB-INF/classes/	用于存放 class 文件，也是该 Web 应用程序的类加载路径
WEB-INF/lib/	用于存放第三方的类库 jar 文件
<应用程序的根目录>	用于存放静态页面、JSP 文件和其他资源文件

Web 容器正是根据这些规范来控制着 Web 应用程序的行为，如包含那些信息资源、会话超时间隔、信息资源的生命周期等。Web 应用程序只能访问到存放在 classes 和 lib 目录下面的 Java 类；web.xml 是整个 Web 应用程序的描述文件，里面定义着该 Web 应用程序可供访问的信息资源以及这些资源的行为，Servlet、监听器、安全验证等信息资源需要在该配置文件中进行配置才能正常使用。图 12.1 是一个名为 javaweb 的 Web 应用程序的目录结构。

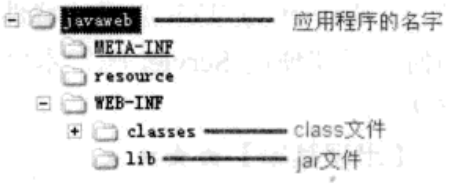


图 12.1　java web 应用程序的目录结构

当用户向 Web 容器发送一个 HTTP 请求的时候，Web 容器会根据请求的地址（URL，如 http://<域名或 IP 地址>/<应用的名字>/<资源的地址>）去定位这个资源。首先是找到应用程序的名字，然后再根据应用程序的配置文件（web.xml），去匹配是否有资源与请求的地址相吻合，如果存在则返回资源的信息。以下是一个 Servlet 的配置样例：

```
<!-- 首先定义一个名为 MyTestServlet 的 Servlet，指定好它的完整类名 -->
<servlet>
    <servlet-name>MyTestServlet</servlet-name>      <!-- Servlet 的名字 -->
    <servlet-class>MyTestServlet</servlet-class>
                                                    <!-- Servlet 的完整类名 -->
</servlet>
<!-- 再指定该 Servlet 的 URL -->
<servlet-mapping>
    <servlet-name>MyTestServlet</servlet-name>
                                                <!-- Servlet 的名字，须同上面一样 -->
    <url-pattern>/MyTestServlet</url-pattern>  <!-- URL，往往以“/”开头 -->
</servlet-mapping>
```

通过以上配置以后，Web 容器才能正确的指向到该 Servlet，调用 Servlet 接口的 service() 方法来为用户提供服务。不仅仅是 Servlet，还有监听器、过滤器、初始化参数、安全验证等东西都需要在 web.xml 中进行配置和说明。web.xml 文件就相当于一个 Web 应用程序的司令部，由它来控制该 Web 应用程序的行为。

【答案】

一个能够在 Web 容器里运行的应用程序需要遵守如下一些规范。

（1）目录结构规范：Java Web 程序的所有文件需要包含在一个文件夹中，它需要包含一个 WEB-INF 子文件夹，WEB-INF 文件夹需要包含 classes 文件夹和 lib 文件夹，以及 Web 描述文件 web.xml。

（2）类文件：第三方的 jar 文件，需要存放在“WEB-INF/lib”文件夹中。

（3）web.xml 规范：web.xml 是整个 Web 应用程序的描述文件，里面定义着该 Web 应用程序可供访问的信息资源以及这些资源的行为，包括 Servlet、过滤器、监听器、安全验证等信息资源。

（4）其他资源文件：JSP、HTML、图片和声音等资源文件需要存放在与 WEB-INF 同一级的目录中，因为 WEB-INF 文件夹对于客户端来说是不可见的。

面试题 118　什么是 Servlet

Servlet 是 Web 容器的最基本组成单元，许多高级的服务端技术都会直接或间接的使用到它。Servlet 与 Web 容器的特性和行为是一种紧密相联的关系。那么什么是 Servlet 呢？如何定义和使用 Servlet 呢？本例在回答该问题的同时，详细地讲解 Servlet 的概念和使用方法。

【出现频率】★★★★

【关键考点】

- Servlet 的概念；
- Servlet 的使用方法。

【考题分析】

HTTP 请求无非就是向 Web 服务器请求一种信息资源，如文本、图片、视频等。Servlet 在 Java Web 服务器中就充当了这种信息资源的最小表示单位，就好像它的名字一样，代表了服务器端一个资源，用户可以通过浏览器获取到这项资源。Servlet 可以进行无限的扩展，它可以使用 Java 的所有类库资源，为用户返回文本、图片、音频、视频等各类信息资源。

如果站在 Java 程序员的角度来看，Servlet 无非就是一个 Java 类，只不过这个类需要符合一些规范。它必须实现 javax.servlet.Servlet 接口的所有方法，提供一个公开的无参数的构造方法。Servlet 只能在 Web 容器中存活，由 Web 容器来控制它的创建、初始化、提供服务、销毁等。以下为一个简单的 Servlet 类示例：

```
import java.io.IOException;
import javax.servlet.Servlet;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
//Servlet 类
public class HelloServlet implements Servlet{
    //销毁方法
    public void destroy() {
        ...
    }
    //获取配置
    public ServletConfig getServletConfig() {
        return null;
    }
    //获取 Servlet 描述信息
    public String getServletInfo() {
        return null;
    }
```

```
    //初始化方法
    public void init(ServletConfig arg0) throws ServletException {
        ...
    }
    //服务方法
    public void service(ServletRequest arg0, ServletResponse arg1) throws
    ServletException, IOException {
        ...
    }
}
```

说明：以上的 Servlet 类是通过实现 javax.servlet.Servlet 接口来创建的。而实际开发中，往往会通过继承自 javax.servlet.http.HttpServlet 来创建，它可以为开发者提供一些方法的默认实现，而且还可以区别不同的请求方法（GET 或 POST）。

为了让它能够被客户端所访问，它还需要在 web.xml 文件里进行配置。定义 Servlet 时采用<Servlet>标签，配置 Servlet 的访问 URL 时采用<servlet-mapping>标签。如果需要设置 Servlet 的初始化参数和启动时机的话，则还需要在<Servlet>标签中加上<init-param>标签和<load-on-startup>标签，配置示例如下：

```
<servlet>
    <servlet-name>MyTestServlet</servlet-name> <!--Servlet 的名字 -->
    <servlet-class>HelloServlet</servlet-class> <!--Servlet 的完整类名 -->
    <init-param>                                <!--初始化参数定义 -->
        <param-name>myparam</param-name>        <!--初始化参数的名字-->
        <param-value>100</param-value>          <!--初始化参数的值-->
    </init-param>
    <load-on-startup>0</load-on-startup>    <!--Servlet 会在启动时加载-->
</servlet>
<servlet-mapping>
    <servlet-name>MyTestServlet</servlet-name>
                                        <!--Servlet 的名字，须同上面一样 -->
    <url-pattern>/*Servlet</url-pattern>    <!--URL，以 Servlet 结尾的通配-->
</servlet-mapping>
```

【答案】

Servlet 在 Java Web 服务器中就充当了信息资源的最小表示单位，代表了一个用户可以通过浏览器获取的资源。Servlet 可以进行无限的扩展，它可以使用 Java 的所有类库资源，为用户返回文本、图片、音频、视频等各类信息资源。

从编程角度来看，Servlet 是一个 Java 类，这个类需要实现 Servlet 接口，提供一个公开的无参数的构造方法。由 Web 容器来控制它的创建、初始化、提供服务、销毁等。它的各种行为方式通过在 web.xml 文件中的配置来决定。

面试题 119　Servlet 的生命周期是怎样的

Servlet 是生活在 Web 容器中的，就好像人类有生老病死一样，Servlet 在 Web 容器中也会有创建、初始化、提供服务、销毁的过程，每一个生命周期都是必不可少的，每个阶段起到了不同的作用。本例在回答该问题的同时，详细地讲解 Servlet 的生命周期及其相关知识。

【出现频率】★★★★

【关键考点】

- Servlet 的概念；
- Servlet 的生命周期。

【考题分析】

在 javax.servlet.Servlet 接口中有 3 个重要的方法，分别是 init()、destroy()和 service()。这 3 个方法分别对应 Servlet 生命周期的 3 个阶段，加上 Servlet 是一个 Java 类，所以还有加载过程，一共是 4 个阶段，分别为加载、初始化、提供服务和销毁。

1. 加载

加载阶段指的是 Servlet 类加载到 Java 虚拟机当中，并且实例化。在这个过程中，Web 容器会调用 Servlet 类的公开的无参数的构造方法，产生一个 Servlet 类的实例对象，也就是由该对象来提供服务。默认情况下，Servlet 是在第一次请求的时候被加载，但是可以通过<load-on-startup>标签设置 Servlet 在 Web 容器启动的时候加载。如果 Servlet 类没有提供无参数的构造方法或者该构造方法不是公开的，将加载失败。

2. 初始化

该阶段指的是为 Servlet 做初始化工作，就好像一个婴儿来到人间，总得给他或她取个名字、穿点衣服什么的吧。Web 容器在初始化 Servlet 的时候会调用 init()方法，所以，一般的初始化代码会放在这个方法中，如打开数据源等。前面提到的为 Servlet 配置的初始化参数也是在该方法中取得，详见下一小节。

说明： *加载和初始化阶段都可以做一些 Servlet 的初始化工作，但是初始化阶段可以获取一些 web.xml 配置的外部参数。*

3. 提供服务

提供服务指的是当有 HTTP 请求指向 Servlet 的时候，调用 service()方法的过程。该方法体包含了该 Servlet 的业务逻辑，如果是继承自 HttpServlet 的话，根据 HTTP 请求类型的不同，业务逻辑代码会包含在 doGet()或 doPost()方法中。

4. 销毁

Servlet 对象不可能长存，它总有销毁的时候，例如，重新部署 Web 应用、关闭 Web 容器等。它对应的回调方法是 destroy()，所以，一般释放资源（如关闭数据源等）的代码会放在该方法中。

Servlet 生命周期中的 4 个阶段是按照顺序进行的，每个阶段都有特定的含义，在实际的开发中，一般关注初始化和提供服务两个阶段相对多一些。如图 12.2 所示，展示了 Servlet 的整个生命周期及其工作原理。

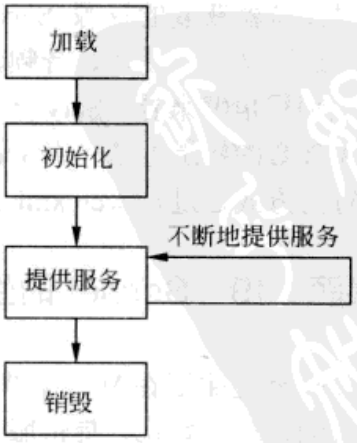


图 12.2　Servlet 的生命周期和工作原理

【答案】

Servlet 的生命周期分为 4 个阶段：加载、初始化、提供服务和销毁，这些过程都是由 Web 容器来掌控。开发者关注最多的是初始化和提供服务两个阶段，在 init()方法中，开发者可以获取配置在 web.xml 中的初始化参数；service()方法中的代码，会在 Servlet 的请求来到时被调用。

面试题 120　Servlet 接口有哪些实现类

大家知道，Servlet 接口是所有 Servlet 类都需要实现的接口。Java EE 已经为开发者提供了一些基础实现类，可以帮助开发者完成 Servlet 的开发。本例在回答该问题的同时，全面地讲解 Servlet 接口实现类的种类及其用途。

【出现频率】★★★★

【关键考点】

- Servlet 的概念；
- Servlet 的实现类及其主要用途。

【考题分析】

在 Java EE 的 SDK 中，一共提供了以下 3 个 Servlet 接口的实现类。

- javax.faces.webapp.FacesServlet。用于 JSF 的 Servlet，一般很少使用。
- javax.servlet.GenericServlet。它是一个抽象类，不能直接使用，它提供了除 server()方法以外的所有抽象方法的默认实现，可用于一般的 Servlet 开发。
- javax.servlet.http.HttpServlet。该类是开发者使用最多的一个类，它不但提供了所有抽象方法的默认实现，还提供了不同的方法以区分不同类型的 HTTP 请求，例如，doPost()、doGet()等。

GenericServlet 相对于直接实现 Servlet 接口来说，开发者仅仅需要实现 service()即可，其他的方法都有默认的实现，以下为一个继承自 GenericServlet 的示例 Servlet：

```
import java.io.IOException;
import javax.servlet.GenericServlet;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
//Servlet，继承自 GenericServlet
public class GeServlet extends GenericServlet{
    //提供 service()方法的实现
    @Override
    public void service(ServletRequest arg0, ServletResponse arg1) throws
ServletException, IOException {
        ...
    }
}
```

当 Servlet 需要针对不同的 HTTP 请求方法做出不同的响应的时候，HttpServlet 就比 GenericServlet 适合一些了。开发者只需要直接覆盖它的 doPost()或 doGet()方法即可，以下为一个继承自 HttpServlet 的示例 Servlet：

```
import java.io.IOException;
```

```
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
//Servlet，继承自 HttpServlet
public class MyHttpServlet extends HttpServlet{
    //提供 doGet()方法的实现
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
        ...
    }
    //提供 doPost()方法的实现
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse
    resp) throws ServletException, IOException {
        ...
    }

}
```

说明：不仅仅是 POST 和 GET 请求方法，其他的请求方法也得到了 HttpServlet 的支持，开发者只需要覆盖它们即可，例如，doDelete()、doHead()、doPut()等。

【答案】

在 Java EE 的 SDK 中，一共提供了以下 3 个 Servlet 接口的实现类。

- javax.faces.webapp.FacesServlet；
- javax.servlet.GenericServlet；
- javax.servlet.http.HttpServlet。

面试题 121　如何在 Servlet 中获取请求参数的值

Web 应用程序往往具有很强的互动性，用户可以通过浏览器发送请求参数到服务器端，服务器端根据这些参数做出不同的响应。那么，在 Servlet 中是如何获得这些请求参数的呢？本例在回答该问题的同时，全面地讲解 Servlet 获取参数的方法。

【出现频率】★★★★

【关键考点】

- Servlet 的概念；
- ServletRequest 的含义；
- 获取请求参数的方法。

【考题分析】

HTTP 请求的方式一般分为 GET 和 POST 两种方式，前者一般用于普通 HTTP 请求，后者则多用于表单的提交。GET 请求方式会把请求的参数追加到请求 URL 后面，首先在 URL 后面加一个问号“?”，然后使用“参数名=参数值”的形式追加参数，必须使用等号“=”隔开，如 http://127.0.0.1/SomeServlet?username=xxx&password=xxx，该 URL 包含了两个参数，即 username 和 password，它们的值均为 xxx。POST 请求方式则是把请求的参数存放在请求的正文中，不能直接通过 URL 看到。

注意：如果请求的参数或参数的值包含中文字符或空格字符的话，则浏览器会使用 UTF8 的编码方式进行编码，例如空格字符就会转换成 20%进行发送。读者如果注意到请求的 URL 包含了若干的百分号%，则证明你的 URL 包含了中文字符或空格字符，进行了重新编码。

Web 容器把每一次 HTTP 请求都看成 ServletRequest 对待，在到达 Servlet 的 service()方法之前，Web 容器会创建一个 javax.servlet.ServletRequest 接口的实现类的对象，并以 service()方法参数的形式提供给 Servlet 使用。

不管是哪种 HTTP 请求方式，Servlet 都可以通过 ServletRequest 接口的 getParameter()或 getParameterValues()方法获取到客户端提交的参数。两个方法的参数都是 HTTP 请求参数的名字，前一个方法适用于只有一个值的参数，后者用于有多值的参数（例如，复选框）。在 Servlet 的服务处理方法 service()的参数列表中，包含了请求接口实现的对象，只需要直接使用该对象即可。如果是继承自 javax.servlet.http.HttpServlet 的 Servlet，也可以在 doGet()和 doPost()方法中找到该请求对象。以下是获取请求参数示例代码，获取了用户名参数、密码参数和爱好的参数数组。

```
//获取参数名为 username 的用户名，返回字符串
String uername = request.getParameter("uername");
//获取参数名为 password 的密码，返回字符串
String password = request.getParameter("password");
//获取参数名为 hobby 的爱好数组，返回的是字符串数组
String[] hobbies = request.getParameterValues("hobby");
```

获取客户端请求参数是 Web 交互性的基础，服务器端根据不同的参数值做出不同的响应。有的参数对于服务器端的逻辑处理是必须的，例如，用户登录，用户必须输入用户名和密码才能实现正常的登录，同时，如果输入错误的话，则提示用户重新登录。

【答案】

在 Servlet 中，任何负责做出响应的方法（例如，service()、doPost()和 doGet()）都会包含一个 ServletRequest 对象参数，不管是 POST 还是 GET 的请求方式，Servlet 都可以通过 ServletRequest 接口的 getParameter()或 getParameterValues()方法获取到。前者适用于只有一个值的参数，后者多用于有多值的参数，例如，复选框（checkbox）。

面试题 122　Forward 和 Redirect 的区别

用户向服务器端发出了一次 HTTP 请求，该请求可能会经过多个信息资源处理以后才返回给用户，各个信息资源使用请求转发机制相互转发请求，但是用户是感觉不到请求转发的。根据转发方式的不同，可以区分为直接请求转发（Forward）和间接请求转发（Redirect）两种，它们有什么区别呢？本例在回答该问题的同时，全面地讲解两种请求转发方式的原理和区别。

【出现频率】★★★★

【关键考点】

- 请求转发的含义；
- Forward 转发请求的原理；

❑ Redirect 转发请求的原理。

【考题分析】

Forward 和 Redirect 代表了两种请求转发方式：直接转发和间接转发。间接转发方式本质上是两次 HTTP 请求，服务器端在响应第一次请求的时候，让浏览器再向另外一个 URL 发出请求，从而达到转发的目的。直接转发方式，客户端浏览器只发出一次请求，Servlet 把请求转发给 Servlet、HTML、JSP 或其他信息资源，由第 2 个信息资源响应该请求，在请求对象 request 中，保存的对象对于每个信息资源是共享的。

1. 间接请求转发

间接转发方式，有时也称重定向，它一般用于避免用户的非正常访问。例如，用户在没有登录的情况下访问后台管理资源，Servlet 可以将该 HTTP 请求重定向到登录页面，让用户登录以后再访问。在 Serlvet 中，通过调用 response 对象的 sendRedirect()方法，告诉浏览器重定向访问指定的 URL，示例代码如下：

```
...
    //Serlvet 中处理 get 请求的方法
    public void doGet(HttpServletRequest request, HttpServletResponse
    response) {
        response.sendRedirect("资源的 URL"); //重定向请求到另外的资源
    }
...
```

图 12.3 展示了间接转发的工作原理。

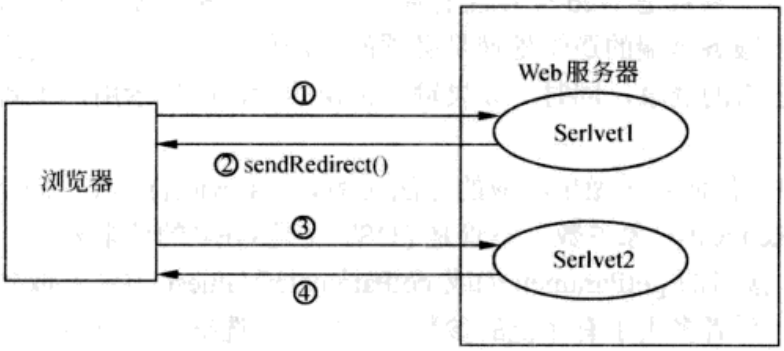


图 12.3　间接转发请求的工作原理图

图 12.3 所示的间接转发请求的过程如下：

（1）浏览器向 Servlet1 发出访问请求。

（2）Serlvet1 调用 sendRedirect()方法，将浏览器的请求重定向到 Serlvet2。

（3）浏览器向 Servlet2 发出访问请求。

（4）最终由 Servlet2 做出响应。

2. 直接请求转发

直接转发方式应用得更多一些，一般说的请求转发指的就是直接转发方式。Web 应用程序大多会有一个控制器，有控制器来控制请求应该转发给哪个信息资源。然后由这些信息资源处理请求，处理完以后还可能会转发给另外的信息资源来返回给用户，这个过程也

就是经典的 MVC 模式。

javax.servlet.RequestDispatcherh 接口是请求转发器必须实现的接口，由 Web 容器为 Servlet 提供实现该接口的对象，通过调用该接口的 forward()方法到达请求转发的目的，示例代码如下：

```
...
    //Serlvet 里处理 get 请求的方法
    public void doGet(HttpServletRequest request, HttpServletResponse
    response){
        //获取请求转发器对象，该转发器的指向通过 getRequestDispatcher()的参数设置
        RequestDispatcher requestDispatcher = request.getRequest-
        Dispatcher("资源的 URL");
        //调用 forward()方法，转发请求
        requestDispatcher.forward(request, response);
    }
...
```

图 12.4 展示了直接转发的工作原理。

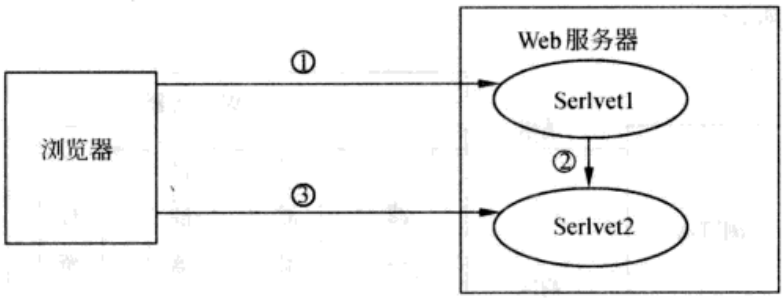


图 12.4　直接转发请求的工作原理图

图 12.4 所示的直接转发请求的过程如下：

（1）浏览器向 Servlet1 发出访问请求。

（2）Serlvet1 调用 forward()方法，在服务器端将请求转发给 Servlet2。

（3）最终由 Servlet2 做出响应，返回给浏览器。

技巧： 其实，通过浏览器就可以观察到服务器端使用的是哪种请求转发方式。当单击一个超级链接的时候，浏览器的地址栏会出现当前请求的地址，如果等服务器端响应完成以后，发现地址栏的地址变化了，则证明是间接的请求转发。相反，如果地址没有变化，则代表是直接请求转发或没有转发。

【答案】

Forward 和 Redirect 代表了两种请求转发方式：直接请求转发和间接请求转发。对应到代码里，分别是 RequestDispatcherh 类的 forward()方法和 HttpServletResponse 类的 sendRedirect()方法。

对于间接转发方式，服务器端在响应第一次请求的时候，让浏览器再向另外一个 URL 发出请求，从而达到转发的目的。它本质上是两次 HTTP 请求，对应两个 request 对象。

对于直接转发方式，客户端浏览器只发出一次请求，Servlet 把请求转发给 Servlet、HTML、JSP 或其他信息资源，由第 2 个信息资源响应该请求，两个信息资源共享同一个 request 对象。

面试题 123　过滤器的作用和工作原理是什么

过滤器是 Servlet 规范中的一部分，它在实际开发中应用也比较多，利用它可以把多个 Servlet 的相同逻辑抽象到一起来处理，著名的 Struts 2 框架就是利用过滤器来工作的。本例在回答该问题的同时，详细地讲解过滤器的工作原理。

【出现频率】★★★★

【关键考点】

- 过滤器的含义；
- 过滤器的工作原理。

【考题分析】

顾名思义，过滤器就是在目标资源与源头资源之间起到过滤作用的一个东西。就好像污水处理厂处理污水一样，污水经过污水处理设备以后，就变成了净水。对于 Web 应用程序来说，过滤器是处于 Web 容器内的，对请求信息和响应信息进行过滤的一种组件，其工作原理如图 12.5 所示。

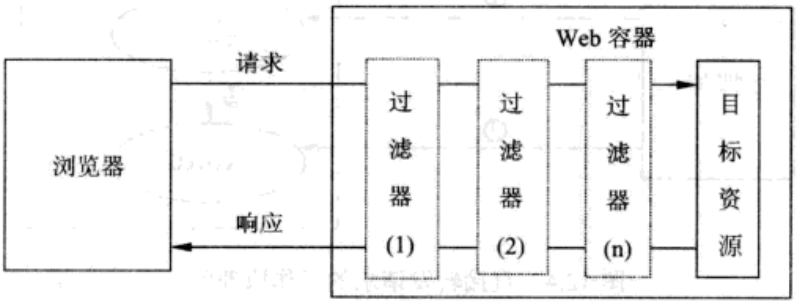


图 12.5　过滤器的工作原理图

当 Web 容器收到一个对某个信息资源请求的时候，它会判断是否有过滤器与该信息资源关联。如果有的话，会把请求一一的交给这些过滤器处理，然后再交给目标资源，同样，响应的时候就会以相反的顺序交给过滤器处理一下，再返回给用户浏览器。

说明： 过滤器是一种很重要的设计模式，不仅仅应用在 Web 开发中，其他一些开发领域也会使用过滤器模式，它可以在不侵入原有代码的基础上为它们提供一些功能。

过滤器类需要实现 javax.servlet.Filter 接口，该接口的 doFilter()方法就业务处理的核心代码区，类似于 Servlet 的 service()方法。doFilter()方法的参数列表中有一个 FilterChain 接口的实现对象，它只有一个方法：doFilter()。在调用该方法之前的代码会在达到目标资源前执行，之后的代码会在目标资源已经响应以后执行，以下是一个 Filter 代码实现的示例：

```
...
//过滤器需要实现 Filter 接口
public class MyFilter implements Filter{
    //过滤器的业务逻辑方法
    public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain) throws IOException,ServletException {
        //达到目标资源前的代码
        ...
```

```
        //该方法的调用会将请求转发给下一个过滤器或目标资源
        chain.doFilter(request, response);
        //目标资源响应以后的代码
        ...
    }

    public void init(FilterConfig filterConfig) throws ServletException {
        //初始化的代码
    }
    public void destroy() {
        //释放资源的代码
    }
}
...
```

过滤器类完成以后，还需要进行配置，Web 容器才能将过滤器与目标资源进行关联。它的配置与 Servlet 有些类似，通过 web.xml 文件中的<filter>标签进行定义，<filter-mapping>标签进行 URL 的匹配。当然，还可以像 Servlet 那样配置初始化参数。以下代码是一个过滤器的配置示例：

```
...
    <filter>
        <filter-name>MyFilter</filter-name>          <!-- Filter 的名字 -->
        <filter-class>
            MyFilter    <!-- Filter 的完整类名 -->
        </filter-class>
    </filter>
    <filter-mapping>
        <filter-name>MyFilter</filter-name>          <!-- Filter 的名字 -->
        <url-pattern>/*Servlet</url-pattern> <!-- Filter 的 URL 匹配方式 -->
    </filter-mapping>
...
```

经过以上的代码配置以后，凡是请求以“Servlet”结尾的 URL 资源的时候，该过滤器都会起到过滤作用。

【答案】

对于 Web 应用程序来说，过滤器是处于 Web 容器内的一个组件，它会过滤特定请求资源请求信息和响应信息。一个请求来到的时候，Web 容器会判断是否有过滤器与该信息资源关联，如果有，则交给过滤器一一的处理，然后再交给目标资源，响应的时候则以相反的顺序交给过滤器处理，最后再返回给用户浏览器。

过滤器对应 Filter 接口，开发者一般需要实现 doFilter()方法，并在 web.xml 文件夹中提供相应的配置。

面试题 124　监听器的作用和工作原理是什么

监听器是一种非常著名的设计模式，Web 容器中也提供了一种监听器的组件，它负责监听各种事件的发生并做出相应的响应。而开发者需要做的就是面向这些事件进行编程。本例在回答该问题的同时，详细地讲解监听器的含义和作用。

【出现频率】★★★★

【关键考点】

- 监听器的含义；

❑ 监听器的工作原理。

【考题分析】

Web 应用程序有各类的范围模型，这些模型代表了特定的范围含义。Request 代表一次请求，Session 代表一次会话，Application（Servlet 上下文）代表一个 Web 应用，它们都生存在 Web 容器中，有自己的生命周期和作用范围。Request 的作用范围最小，Session 其次，Application 最大。它们都可通过 setAttribute()和 getAttribute()方法来往相应的范围中存取数据，在它们自己范围内的所有信息资源都可以共享这些属性。表 12.2 列出了它们的代表含义、作用范围和描述。

表 12.2　Java Web应用程序中的范围模型

名　称	作 用 范 围	描　述
Request	请求	代表了一次 HTTP 请求，它的生命周期从请求开始直到请求的响应结束。中间可能会穿插多个信息资源，如过滤器、Servlet、JSP 等
Session	会话	代表了一次会话，也可认为代表了一个用户，一次会话中可以包含多次请求。如果用户长时间不发出请求，可能导致会话超时而过期，超时时长可进行设置。因为它代表一个用户，所以一般使用 Session 保存用户的数据，如购物车
Application	应用	代表了一个 Web 应用，它的生命周期从 Web 容器启动或者部署该应用开始，以 Web 容器停止或取消部署而结束。它具有最大作用那个范围，一般用于存放应用标题、其他外部资源连接器等

在这几类范围模型的生命周期中，会发生许多的事件，当这些事件发生的时候，有时需要程序做点什么事情。例如，当 Web 应用程序完成加载以后，建立一个与某外部资源的连接；当一个用户第一次访问 Web 应用程序时，程序将在线人数加一。Web 应用程序的事件监听器就可以帮助开发者达到这样的目的，为各类事件提供了丰富的接口，程序员实现这些接口即可。为了让 Web 应用程序知道这些监听器，还需要在 web.xml 配置文件中，使用<listener>标签进行如下配置：

```
...
    <listener>
        <listener-class><!-- 完整的类名 --></listener-class>
    </listener>
...
```

说明： 监听器体现了一种监听事件模型。如果读者熟悉 Java 桌面编程的话，可以知道 Java 桌面编程处处都会有事件模型，当某一个事件发生以后，就会对应一个相应的接口方法。同样，Web 应用程序的监听器是对 Web 生命周期的 3 种事件的监听。

1．Request事件监听器接口ServletRequestListener

javax.servlet.ServletRequestListener 接口是为 request 准备的，它包含两个方法，requestInitialized()是 Web 应用初始化时调用的，requestDestroyed()是 Web 应用销毁时调用的，示例代码如下：

```
...
//实现 HttpRequestListener 接口
public class MyRequestListener implements ServletRequestListener{
    public void requestDestroyed(ServletRequestEvent arg0) {
        // 请求销毁时的代码
    }
    public void requestInitialized(ServletRequestEvent arg0) {
        // 请求初始化时的代码
    }
}
...
```

2. Session事件监听器接口HttpSessionListener

javax.servlet.http.HttpSessionListener 接口是为 Session 准备的，它包含两个方法，sessionCreated ()是会话创建时调用的，sessionDestroyed ()是会话销毁时调用的，示例代码如下：

```
...
//实现 HttpSessionListener 接口
public class MySessionListener implements HttpSessionListener{
    public void sessionCreated(HttpSessionEvent arg0) {
        //会话创建时的代码
    }
    public void sessionDestroyed(HttpSessionEvent arg0) {
        //会话销毁时的代码
    }
}
...
```

3. Application事件监听器接口ServletContextListener

javax.servlet.ServletContextListener 接口是为 Application 准备的，它包含两个方法，contextInitialized()是 Web 应用初始化时调用的，contextDestroyed()是 Web 应用销毁时调用的，示例代码如下：

```
...
//实现 ServletContextListener 接口
public class MyApplicationListener implements ServletContextListener{
    public void contextDestroyed(ServletContextEvent arg0) {
        //应用销毁时的代码
    }
    public void contextInitialized(ServletContextEvent arg0) {
        //应用初始化时的代码
    }
}
...
```

【答案】

对于 Web 应用程序来说，监听器是处于 Web 容器内的一个组件，它会对 Web 容器中的 3 种范围对象进行监听：request、session 和 application。当这些范围对象在创建或销毁的时候，Web 容器会主动的调用它们的初始化或销毁的回调方法，从而达到事件响应的效果。根据范围的不同，Java EE 为开发者提供如下一些监听器接口。

- Request 事件监听器接口 ServletRequestListener；
- Session 事件监听器接口 HttpSessionListener；
- Application 事件监听器接口 ServletContextListener。

12.2　JSP 动态语言

JSP、ASP 和 PHP，是目前最流行的 3 种动态语言技术，但是 JSP 比其他两种要强大得多，因为支持它的是几乎无所不能的 Java 语言。另外，JSP 具有非常大的灵活性，它可以通过 JavaBean、表达式语言、标签库等多种技术实现扩展。本节将集中讨论有关 JSP 的常见面试题。

面试题 125　JSP 的运行机制是什么

从表面上来看，JSP 代表的是一种动态网页，它可以实现数据的动态展示，在 HTML 代码中间，穿插着 JSP 的脚本和标签。它为何能够实现页面的动态展示呢？JSP 的本质是什么？本例在回答该问题的同时，详细地讲解 JSP 的概念和工作原理。

【出现频率】★★★★

【关键考点】

- JSP 的概念；
- JSP 的工作原理。

【考题分析】

JSP（Java Server Page）是一种建立在 Serlvet 规范提供的功能之上的动态网页技术，与 ASP 和 PHP 类似，都是在网页文件中嵌入脚本代码，产生动态的内容，只不过 JSP 采用的脚本语言是 Java。

JSP 文件会在用户第一次请求时，Web 容器会将该 JSP 文件编译成为 Servlet，再由该 Servlet 处理用户的请求，所以说 JSP 在本质上是 Servlet。但是，JSP 的存在是有必要的，因为 Servlet 在处理静态内容（如 HTML 标签）时非常笨拙，不得不把静态内容以字符串的形式进行拼接，而 JSP 可以很好的实现动态和静态内容的分离，开发者可以对静态内容和动态内容分别开发。

那么 JSP 在接收客户端请求的时候，它是如何工作的呢？首先看看一个简单的 JSP 文件（hello.jsp），该 JSP 文件向浏览器输出“Hello World!”，代码如下：

```
<%@ page language="java" pageEncoding="UTF-8"%>
<html>
    <head>
        <!-- 文档标题 -->
        <title>Hello JSP</title>
    </head>
    <body>
        <%
            out.println("Hello World!");        //向客户端写 Hello World!
        %>
    </body>
```

```
</html>
```

将该文件保存在 Web 容器中，例如 Tomcat。如果存放在 Tomcat 的 Root 应用下，用浏览器访问址栏：http://localhost:8080/hello.jsp，可以看到网页显示出“Hello World!”。

那么 JSP 转换成 Servlet 的源文件和 Class 文件在哪里呢？其实，这两个文件可以在目录“<Tomcat 安装目录>\work\Catalina\localhost_\org\apache\jsp”中找到，分别是 hello_jsp.java 和 hello_jsp.class。这两个文件都是由 Web 容器生成的，整个过程如图 12.6 所示。

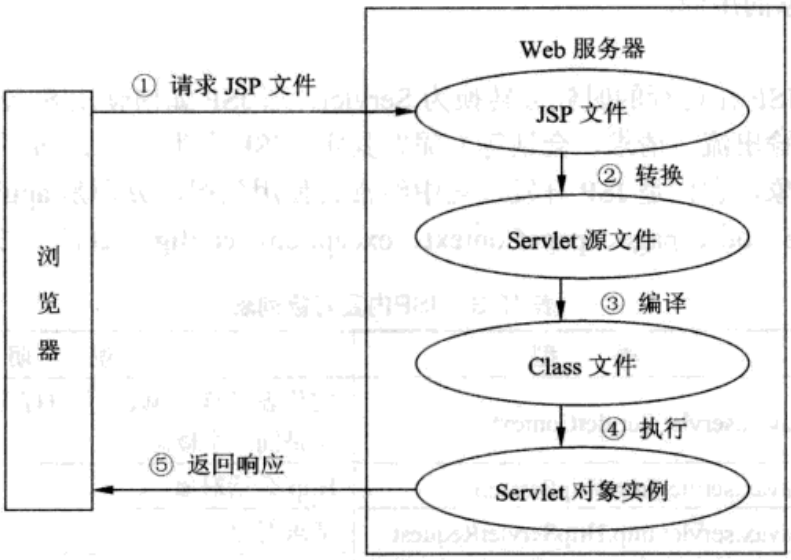


图 12.6　JSP 工作原理图

当一个对 JSP 文件的请求到来时，Web 容器会检验 JSP 的语法是否正确，将其转换成 Servlet 的源码文件。然后使用 javac 工具编译该源码文件成为 Class 文件，紧接着，创建一个该 Serlvet 对象实例，以 Servlet 的方式为请求提供服务。

说明：因为 JSP 是在第一次访问时才被转换和编译，所以第一次访问 JSP 文件会比较慢，以后的访问 Web 容器就直接调用编译好的 Servlet 对象实例了。如果 JSP 被修改过，那么整个过程会重新执行一次。

如果读者把上面示例的 JSP 生成的源文件（hello_jsp.java）打开来看一下，会发现 hello_jsp 类继承自 org.apache.jasper.runtime.HttpJspBase 类。其实，这个类是由 Tomcat 提供的，它继承自 javax.servlet.http.HttpServlet 类，也验证了 JSP 是 Servlet 的一个子类。

【答案】

当客户端发出一次对某个 JSP 的请求，Web 容器处理该请求的过程如下：

（1）Web 容器会检验 JSP 的语法是否正确。

（2）将 JSP 文件转换成 Servlet 的源码文件。

（3）编译该源码文件成为 Class 文件。

（4）创建一个该 Serlvet 类的对象实例，以 Servlet 的方式为请求提供服务。

面试题 126　JSP 的内置对象及其用途

JSP 之所以比 Servlet 更好用，除了它处理 HTML 的能力比 Servlet 更方便以外，很大

程度上还取决于 JSP 内置了许多常用的服务器端组件，开发者可以直接使用它们。JSP 包含了 9 个内置对象，各有用途，是 JSP 开发者的 9 把利器。本例在回答该问题的同时，详细地讲解 JSP 的内置对象及其作用。

【出现频率】★★★★

【关键考点】

- JSP 的内置对象；
- 内置对象的用途。

【考题分析】

大家知道，JSP 在运行的时候会转换为 Servlet。那 JSP 如何使用 Servlet 中的一些重要的对象（例如，输出流、请求、会话等）呢？其实，JSP 在生成 Servlet 的时候会为开发者准备 9 个内置对象，它们是 JSP 开发过程中经常会使用到得，分别是 application、session、request、response、out、page、pageContext、exception、config，如表 12.3 所示。

表 12.3　JSP内置对象列表

对　象	类　型	说　明
application	javax.servlet.ServletContext	它代表了整个 Web 应用程序，与 Servlet 上下文是同一个概念
session	javax.servlet.http.HttpSession	Http 会话对象
request	javax.servlet.http.HttpServletRequest	请求对象
response	javax.servlet.http.HttpServletResponse	返回对象
out	javax.servlet.jsp.JspWriter	写出流对象，用于返回数据给客户端
page	java.lang.Object	普通的页面对象
pageContext	javax.servlet.jsp.PageContext	页面上下文，代表页面的一个运行环境，通过它可以获取到其他对象，如会话、请求等
exception	javax.lang.Throwable	用于错误页面，通过该对象可获得异常的详细信息
config	javax.servlet.ServletConfig	配置对象，用于获取初始化参数等数据

说明：内置对象是放在 JSP 的服务方法中的，所以只能在脚本段和表达式元素中才可以使用。

application、session、request、response 和 config 的概念和用法与 Servlet 的用法是相同的。其中，application 与 Servlet 的 ServletContext 概念相同，代表了整个应用程序；session 和 request 代表了一种范围；config 与 Servlet 的 init()方法的参数相同，用于获取一些配置参数。下面介绍一下另外 4 个 JSP 特有的对象。

1．pageContext

pageContext 代表的是 JSP 页面上下文，也就是一个运行环境。它为开发者提供了访问其他内置对象的统一入口，也就是说，其他对象可以通过该对象间接的获取，如下所示的方法。

```
PageContext.getRequest();                    //获取请求对象
```

```
PageContext.getSession();                    //获取会话对象
PageContext.getServletContext();             //获取 Servlet 上下文对象
PageContext.getResponse();                   //获取响应对象
PageContext.getOut();                        //获取输出流对象
```

页面上下文 pageContext 也作为一个范围概念，代表了整个页面，也可以像请求、会话、Servlet 上下文一样保存和获取属性，调用方法如下：

```
PageContext.setAttribute("属性名",对象);      //保存属性
PageContext.getAttribute("属性名");           //获取属性
```

另外，由于 pageContext 是 JSP 的一个运行环境，它包含一个特殊的方法，可以按照 page、request、session 和 application 的顺序查找属性，方法如下：

```
PageContext.findAttribute("属性名");          //按顺序查找指定的属性
```

2. out

out 对象代表的是输出流对象，它是 java.io.Writer 的子类，以字符流的形式把数据写到客户端。它本身包含一个缓冲区，默认大小为 8KB，如果写的内容不足缓冲区大小或想让内容迅速的传递到客户端，需要调用 flush()方法清空缓冲区。写出内容通常调用以下两个方法：

```
out.print("内容");                            //直接写出
out.println("内容");                          //加换行符后写出
```

注意：在编写 JSP 的时候，不需要显式的调用 close()方法，因为 JSP 生成的 Servlet 代码会包含 close()方法的调用。如果调用 close()后，还有内容需要写出就可能产生错误。

3. exception

exception 对象代表了 JSP 页面运行时产生的异常，Web 容器一旦捕获到异常就会将异常赋给该对象，它只能在错误页面（在 page 指令中指明了属性 isErrorPage=true）使用。一般来说，JSP 文件会指定一个错误页面（在 page 指令中指明了属性 errorPage=error.jsp），然后该错误页面会把错误信息打印到客户端浏览器中，以便于开发人员调试程序，如下示例代码：

```
<%@ page isErrorPage="true" %>      <%--  标示为错误页面  --%>
<%
    out.println(e.getMessage());          //打印异常的描述信息
%>
```

4. page

该 page 对象只是一个普通的 Object 对象，它将会在 JSP 生成的 Servlet 代码中出现，很少使用该对象。

注意：page 与 pageContext 对象是有区别的，一般说的页面指的是 pageContext，而 page 几乎不会使用。

【答案】

JSP 包含 9 个内置对象，分别是 application、session、request、response、out、page、pageContext、exception、config，它们的用途请参见表 12.3。

面试题 127　page 和 request 作用范围的区别是什么

当客户端对某个 JSP 发出请求时候，Web 容器为这次请求创建了一个 request 对象和一个 pageContext 对象，它们有什么区别呢？其实，有的时候，page 和 request 的作用范围是相同的，但是有的时候却不同。本例在回答该问题的同时，详细地讲解 Web 信息资源作用范围的概念及其区别。

【出现频率】★★★★

【关键考点】

- 作用范围的概念；
- page 范围的概念；
- request 范围的概念。

【考题分析】

范围也就是 Web 应用的时间范围概念，它定义了什么时间内可以访问哪些对象，这些对象都捆绑到相对应的范围对象中。JSP 比 Servlet 要多一种页面范围，一共是 4 种作用域：page、request、session 和 application。这 4 种范围对象通过设置和获取属性的方式来存取对象，它们对应的 Java 类中都包含有以下两个方法：

```
setAttribute("属性名",对象);      //设置属性，保存对象
getAttribute("属性名");           //获取属性，得到对象
```

- application 范围：application 范围的对象是被存放在 application 对象中的，也就是 Servlet 上下文。该范围代表的整个应用程序的范围，在这个范围内存放的对象会保存最长的时间，只有 Web 容器关闭或应用程序重新部署，application 范围内的对象才会消失。
- session 范围：session 范围的对象是被存放在 session 对象中的。该范围代表的一次会话或一个用户，在这个范围内存放的对象往往是一些用户的数据，如用户名、密码等，会话过期之前，所有存放在请求范围内的对象将始终保持。
- request 范围：request 范围的对象是被存放在 request 对象中的。在这个范围内存放的对象可被一次请求中的所有 JSP、Servlet 共享。当请求响应完毕以后，所有存放在请求范围内的对象将全部丢失。
- page 范围：page 范围的对象是被存放在 pageContext 对象中的。在这个范围内存放的对象只能在 JSP 当前页面的范围内共享。当 JSP 运行完毕，所有存放在页面范围内的对象将全部丢失。

按常理来思考，一次请求不就是一个 JSP 或 Servlet 吗？page 和 request 的范围不是一样的吗？其实不然，一次请求有可能被转发多次，转发所到的 JSP 或 Servlet 可以共享 request 范围内的对象，而不能共享页面 page 范围内的对象。如图 12.7 所示，为请求由 3 个 JSP 页面处理，JSP1、JSP2、JSP3 共享了 request 范围内的对象，但是他们存放在各自 page 范

围内的对象是不能共享的。

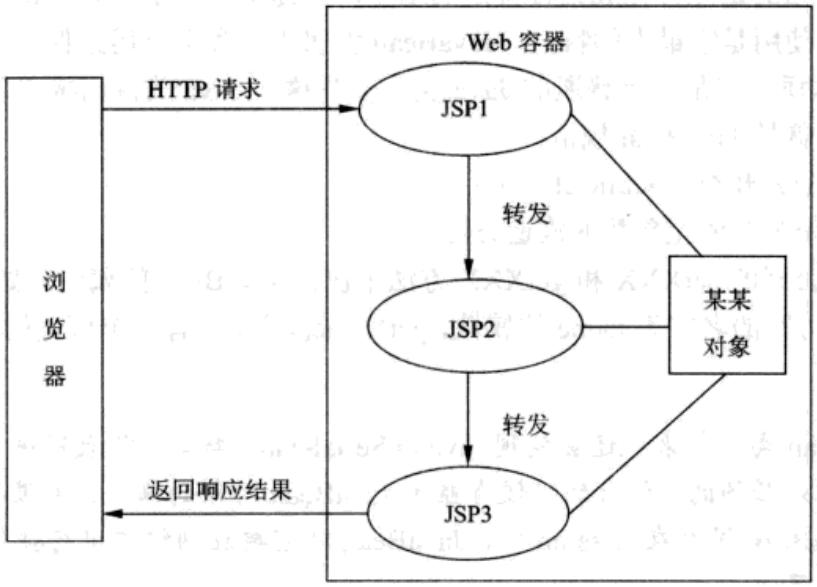


图 12.7　包含多个 JSP 页面的请求

在 JSP 中，使用动作标签<jsp:forward>来进行请求转发，Web 容器在解析到该标签的时候，则直接把请求转发到下一个 JSP 文件或其他资源，示例代码如下：

```
<jsp:forward page="jsp2.jsp"></jsp:forward>                    <! -- 转发请求到 jsp2.jsp -->
```

说明：<jsp:forward>标签的请求转发属于直接请求转发，与 RequestDispatcher 类的 forward()方法作用一样。

【答案】

Page 范围指的是当前 JSP 页面的范围，一旦该 JSP 页面处理完以后，该范围也就结束了，它对应了 JSP 中的 pageContext 内置对象。

Request 范围指的是一次请求，如果请求指向的一个单一的 JSP 文件，则此时的 page 和 request 的生命周期是一样的。但是，一次请求往往不是由单一的 JSP 来出来，因此，可以说，一次 request 的周期可以是若干个 JSP 或其他资源的周期之和。

面试题 128　JSP 如何使用 JavaBean

JavaBean 是 JSP 扩展的一种方式，它最大的特点就是可重用性。JSP+JavaBean 是 Java Web 开发的一种常见模式，将数据与展示进行分开将有利于开发者的思考和开发，也有利于程序的维护。本例在回答该问题的同时，全面地讲解 JSP+JavaBean 的开发模式。

【出现频率】★★★★

【关键考点】

- JavaBean 的规范；
- JSP 使用 JavaBean 的方式。

【考题分析】

可重用组件指的是能够完成某种特定的功能，并提供了一个或多个访问接口的聚合模块程序，可重复使用是它最大的特点，JavaBean 就属于一类可重用组件。

JavaBean 本质上就是一个普通的 Java 类，但是这个 Java 类的组成必须符合一定的规则，这个规则也就是 JavaBean 规范。

- 它是一个公开类（public class）；
- 它有一个公开的无参数的构造方法；
- 提供了公开的 setXXX 和 getXXX 方法来决定 JavaBean 的属性。如 setName()代表有一个可写的名字为 name 的属性，getName()代表了有一个可读的名字为 name 的属性。

注意：JavaBean 类一般来说还会实现 java.io.Serializable 接口，代表可序列化的意思，尽管这不是必须的，但有的时候有益于 JavaBean 对象的拷贝，如服务器集群时，如果 JavaBean 保存在 Session 中，JavaBean 就需要在网络中进行拷贝，此时就需要序列化了。

当 JSP 需要使用 JavaBean 的时候，可以有两种方式，即夹杂在 HTML 中的 JSP 脚本和 JSP 动作标签<jsp:useBean>。

1．纯JSP实现

其实，这种方式采用的是纯粹的 Java 代码来创建 JavaBean 对象和使用 JavaBean，此时的 JavaBean 跟一个普通类是没有什么区别的，示例代码如下：

```
...
    <%
        HelloBean bean = new HelloBean();              //创建 HelloBean 对象
        pageContext.setAttribute("helloBean",bean);
                                                       //把 bean 放在 page 范围中
    %>
...
    <%
        HelloBean bean2 = pageContext.setAttribute();
                                                       //获得 HelloBean 对象
        response.write(bean2.getName());               //使用 bean
    %>
...
```

2．用JSP动作标签访问JavaBean

为了简化 JavaBean 的使用，JSP 还提供了一组动作标签来帮助开发者使用 JavaBean，它们是：

（1）<jsp:useBean>，声明和创建 JavaBean。

（2）<jsp:setProperty>，为 JavaBean 的属性设置值。

（3）用<jsp:getProperty>，获得 JavaBean 的属性值。

```
...
    <!--先用<jsp:useBean>动作标签实例化 GreetBean，实例后的名字为：greetBean-->
```

```
    <jsp:useBean id="greetBean" class="GreetBean" scope="page"/>
    <!--再用<jsp:setProperty>动作标签为 greetBean 的用户名属性赋值-->
    <jsp:setProperty name="greetBean" property="username" param=
    "username" />
    <!--最后用<jsp:getProperty>动作标签返回 greetBean 的问候结果-->
    <jsp:getProperty name="greetBean" property="greeting"/>
...
```

很明显，使用动作标签来使用 JavaBean 显得更加简洁。JavaBean 对象的存储涉及到一个作用范围的问题，JavaBean 可以被保存在 application、session、request 或 page 范围中，通过<jsp:useBean>的 scope 属性指定。<jsp:useBean>标签首先会在指定的范围内去搜索是否已经有这样的一个 JavaBean，如果有则直接使用，若没有则创建一个新的。

说明：<jsp:useBean>的 scope 属性的默认值是 page，默认的把 JavaBean 保存在 pageContext 中。

【答案】

JSP 使用 JavaBean 有两种方式：JSP 的脚本中使用纯粹的 Java 代码和一些 JavaBean 的动作标签。其中，使用<jsp:useBean>等动作标签会更加简洁一些，使用得也更多。

用动作标签来使用 JavaBean 的时候，用<jsp:useBean>标签在特定范围内声明或创建一个 JavaBean，用<jsp:setProperty>标签来设置一个 JavaBean 的属性的值，用<jsp:getProperty>标签来获取一个 JavaBean 的属性的值。

12.3　表达式语言和 JSTL

表达式语言（EL）和 JSTL 都是 JSP 2.0 的技术，它们的主要作用都是为了简化 JSP 展现数据的开发，让 JSP 展现数据更加容易。EL 是一种数据表现语言，JSTL 是一个标签库，而 EL 是从 JSTL 诞生出来的，EL 让 JSTL 的表现力如虎添翼。本节将集中讨论有关表达式语言和 JSTL 的常见面试题。

面试题 129　如何使用迭代标签<c:forEach>循环显示数据

在 JSP 页面上，常常会有一些集合类的数据，需要将它们遍历出来，打印在页面上。如果使用 JSP 的脚本，会让代码非常凌乱，如果使用<c:forEach>标签，会让代码更有组织性和易维护性。本例在回答该问题的同时，详细地讲解<c:forEach>标签的使用方法。

【出现频率】★★★★

【关键考点】

- JSTL 的概念；
- <c:forEach>标签的使用方法。

【考题分析】

在 Web 应用程序中，迭代主要用于访问和显示数据集，通常是以列表或表中的一系列行的形式显示。实现迭代内容的主要 JSTL 标签是<c:forEach>。该标记支持两种不同样式

的迭代：整数范围上的迭代（类似 Java 语言的 for 循环语句）和集合上的迭代（类似 Java 语言的 Iterator、Enumeration 类和 ForEach 循环）。

1. 语法

语法 1：使用 begin 和 end 属性进行整数范围迭代。

```
<c:forEach begin="开始数字" end="结束数字" [var="循环过程中的变量"] step="每次迭代的增量">
    每次循环打印的内容
</c:forEach>
```

语法 2：使用 items 和 var 属性进行集合迭代。

```
<c:forEach items="集合变量" var="循环过程中的变量" [varStatus="status"]>
    每次循环打印的内容
</c:forEach>
```

2. 属性

使用<c:forEach>标签时，主要是使用 items 属性为标签提供数据集，使用 rar 属性指定当次变量引用名词，以及其他一些可选属性，详细参考表 12.4。

表 12.4　<c:forEach>的属性

名　称	类　型	说　明
var	String	迭代过程中的当次变量引用名称，一般用于循环过程中数据的打印
items	Object	集合对象，必须是可以迭代的对象，如 List、Set 等
begin	int	迭代的初始数字，必须为整数
end	int	迭代的结束数字，必须为整数
step	int	每次的增量，默认为 1
varStatus	String	通过它可以获取迭代中的一些数据，如迭代总数，当前迭代索引值等

3. 说明

对于整数范围内的迭代，begin 和 end 属性是必需的，它们要么是静态整数值，要么是可以得出整数值的表达式。它们分别指定迭代索引的初始值以及迭代索引的终止值。当出现 step 时，它也必须是整数值。它指定每次迭代后索引的增量，迭代索引从 begin 属性的值开始，以 step 属性的值为增量进行递增，在迭代索引超过 end 属性的值时停止迭代。如果省略了 step 属性，那么步长默认为 1。

在对集合的成员进行迭代时，会用到另一个属性：items 属性，当使用这种形式的<c:forEach>标签时，items 属性是唯一必需的属性。items 属性的值应该是一个集合，对该集合的成员进行迭代，通常使用 EL 表达式指定值。

尽管 varStatus 属性是可选的，但是有的时候它的作用非常大。例如，做一个分页程序，需要在最后一行后面显示当前页的记录数,可以通过${status.last}判断是否达到最后一行，然后使用${status.count}返回记录数即可。若没有该属性，还需要另外做大量的工作来满足

这样的需求。

4．示例

示例 1：打印 20～50 之间的偶数，代码如下：

```
<c:forEach begin="20" end="50" var="i" step="2">
    偶数：<c:out value="${i}"/><br/>
</c:forEach>
```

示例 2：使用表格展示用户列表中所有用户的数据，代码如下：

```
<table>
    <tr>
        <th>编号</th>
        <th>用户名</th>
        <th>年龄</th>
        <th>性别</th>
    </tr>
    <c:forEach items="${users}" var="user" varStatus="status">
        <tr>
            <td>${status.count}</td>
            <td>${user.name}</td>
            <td>${user.age}</td>
            <td>${user.gender}</td>
        </tr>
    </c:forEach>
</table>
```

注意：<c:forEach>标签 item 属性需要用表达式语言赋值。例如以上示例，不能写为 <c:forEach items="users”>。

【答案】

<c:forEach>标签的使用方法如下：

（1）把需要循环的内容放在<c:forEach>与</c:forEach>之间。

（2）为<c:forEach>提供 items 属性，它指定的是集合对象。

（3）为<c:forEach>提供 var 属性，它指定的是每次循环得到的集合元素。

（4）如果需要记录循环状态，则指定 varStatus 属性。

（5）在循环体中，使用表达式语言，访问 var 属性指定的变量的各种属性。

面试题 130　JSTL 提供了哪些逻辑判断标签

逻辑判断是任何计算机技术必备的项目，JSTL 也提供了逻辑判断的标签。根据判断的分支的多少，可以使用<c:if>或<c:choose>标签，那么应该如何使用它们呢？本例在回答该问题的同时，详细地讲解<c:if>和<c:choose>的使用细节。

【出现频率】★★★★

【关键考点】

- JSTL 的概念；
- <c:if>标签的使用方法；

❑ <c:choose >标签的使用方法。

【考题分析】

对于包含动态内容的 Web 页面，往往会包含大量的布尔判断，希望不同类别的用户看到不同形式的内容。例如，在博客程序中，访问者应该能够阅读各项，也许还应该能够提交反馈，但只有经过授权的用户才能公布新项，或编辑已有内容。

在同一个 JSP 页面内实现这样的功能，然后使用条件逻辑根据每条请求控制所显示的内容，这样做的目的通常能够改善实用性和软件维护。JSTL 提供了两个不同的条件化标签<c:if>和<c:choose>来实现这些功能。

1. 语法

<c:if>是较简单的一个，它简单的对单个测试表达式进行求值，接下来，仅当对表达式求出的值为 true 时，它才处理标签体的内容。如果求出的值不为 true，就忽略该标记的主体内容，语法如下：

```
<c:if test="布尔表达式" [var="变量名" scope="作用范围"]>
内容
</c:if>
```

对于需要进行互斥测试来确定应该显示什么内容的情况下，则使用<c:choose>进行操作，语法如下：

```
<c:choose>
  <c:when test="布尔表达式">
       内容
  </c:when>
  ...
  <c:otherwise>
       内容
  </c:otherwise>
</c:choose>
```

2. 属性

表 12.5　<c:if>的属性

名　称	类　型	说　明
test	boolean	通常是一个 EL 的布尔表达式，如果返回 true 则打印标签体的内容
var	String	把 test 的结果进行保存的变量名称
scope	String	被设置变量保存的作用范围

3. 说明

<c:if>可以通过其 var 和 scope 属性选择将测试结果赋给特定作用范围的变量。当测试过程非常复杂时，这种能力尤为有用：可以将结果缓存在特定的作用范围的变量中，然后在随后对<c:if>或其他 JSTL 标签的调用中检索该结果。

<c:choose>标签的每个要测试的条件都由相应的<c:when>标签来表示，至少要有一个

<c:when>标签。只会处理第一个其 test 值为 true 的<c:when>标签体内的内容。如果没有一个<c:when>测试返回 true，那么会处理<c:otherwise>标签的主体内容。尽管如此，<c:otherwise>标签却是可选的，而且<c:choose>标签至多可有一个嵌套的<c:otherwise>标签。如果所有<c:when>测试都为 false，而且又没有给出<c:otherwise>操作，则什么也不处理。

技巧：如果条件等于或大于两个，则一般使用<c:choose>标签进行条件判断。

4．示例

示例 1：如果用户有相应的权限，则打印对应的超级链接按钮，代码如下：

```
<ul>
    <c:if test="${user.permission1}">
        <li><a href="url1">链接</a></li>
    </c:if>
    <c:if test="${user.permission2}">
        <li><a href="url2">链接</a></li>
    </c:if>
    <c:if test="${user.permission3}">
        <li><a href="url3">链接</a></li>
    </c:if>
</ul>
```

示例 2：根据用户性别的标示符号来打印“男”、“女”或“未知性别”。

```
<c:choose>
    <c:when test="${gender == 'male'}">
        男
    </c:when>
    <c:when test="${gender == 'female'}">
        女
    </c:when>
    <c:otherwise>
        未知性别
    </c:otherwise>
</c:choose>
```

【答案】

JSTL 主要提供了两种标签来处理条件表达式，它们是：<c:if>或<c:choose>。

（1）标签的条件表达式一般放在它的 test 属性中，如果返回为 true 则打印标签体的内容。

（2）一般需要结合<c:when>和<c:otherwise>来使用，条件表达式写在<c:when>标签的 test 属性中，它类似于 Java 的 switch 语句。

12.4　小　　结

本章讲解了一些 Java Web 基础的面试题，主要包含了 Web 容器、Servlet 和 JSP 3 方面的内容。Web 容器和 Web 应用程序属于 Java EE 规范的内容，开发者必须在遵守这些规

范的前提下进行开发，才能保证 Web 应用程序跨服务器运行。Servlet 是 Java Web 开发的基础技术，它是其他高级技术的基础，但是它比较容易掌握，主要是掌握它作为 Web 容器中最基本的信息资源的含义。另外，JSP 也是 Web 开发的一项重要技术，掌握 JSP 更多的是需要掌握它的运行机制，读者应该根据这些原理来掌握 JSP 的其他相关知识。

Web 开发是 Java 技术的主要运用领域，而 Servlet、JSP 等这些基础知识往往是 Java 面试题的重点，读者应该在掌握这些技术理论和规范的时候，勤加练习，才能应对多变而丰富的面试题。

第 13 章　Struts、Spring 和 Hibernate 组合

在 Java Web 开发领域中，有 3 个框架是非常有名的：Struts、Spring 和 Hibernate，简称为 SSH。它们在 Web 应用程序中各司其职，完美的构架 Web 应用，既能够提高开发效率，又利于日常的维护。Struts 是一个典型的 MVC 模式的框架，它主要负责 HTTP 请求的接收和响应工作。Spring 包含了一种轻量级的容器，把程序中的功能对象组织在一起，形成了一种易扩展的构架；Hibernate 则负责数据的存取操作，它用 ORM 的思想来操作数据和对象，使开发者的思维更具有对象性。本章将包含关于 SSH 开发的一些常见面试题，并且分析这些题目和知识点，帮助读者梳理这些方面的知识。

13.1　MVC 和 Struts

MVC 是一种设计模式，它广泛地用于 Web 开发，不论是 Java EE 还是.NET，它的使用已经非常普遍了。Struts 是一个很典型的 MVC 模式的框架，是 Java Web 开发领域中的一颗明星。本节将集中讨论有关 MVC 模式和 Struts 的常见面试题。

面试题 131　什么是 MVC 设计模式

MVC 模式的主要思想是把控制器、数据模型和视图进行分离，达到高内聚和低耦合的作用。MVC 模式并不仅限于 Java Web 开发，对于任何一种 Web 服务器端的技术来说，都是可以采用的。本例在回答该问题的同时，详细地讲解 MVC 模式的结构。

【出现频率】★★★★

【关键考点】

- MVC 的含义；
- MVC 的结构。

【考题分析】

在 Java Web 开发中，存在两种普遍的开发模式，通常称为模式 1 和模式 2。模式 1 使用 JSP+JavaBean 技术将页面显示和业务逻辑分开，由 JSP 来实现页面的显示，JavaBean 对象用来保存数据和实现业务逻辑。客户端直接向 JSP 发出请求，JSP 做出相应的响应，并调用 JavaBean 对象，所有的数据都通过 JavaBean 来处理，然后再返回给 JSP，由 JSP 生成最后的返回结果，模型 1 的结构如图 13.1 所示。

在模型 1 中，JSP 往往会嵌入控制请求流程的代码和部分逻辑代码，如果把这部分代码提取出来，由一个单独的角色来承担，该角色也就是控制器，则此时就构成了模型 2。模型 2 就符合了 MVC 的设计模式，即模型-视图-控制器（Model-View-Controller）。

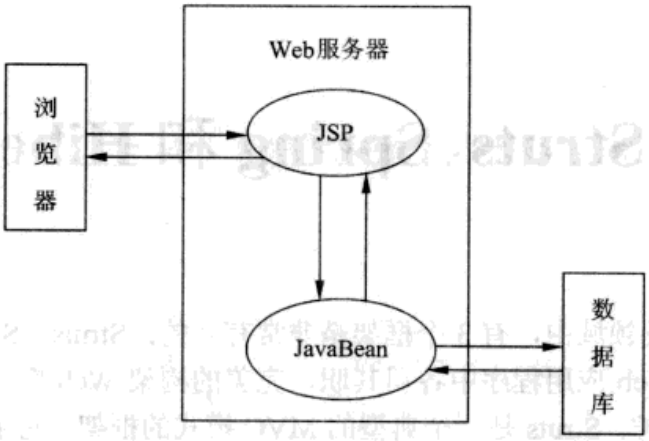


图 13.1　模型 1（JSP+JavaBean）结构图

MVC 构架的 Web 应用程序被分割成若干逻辑部件，使得程序开发编程变得更加容易。它把各种对象按照功能的不同分割成了 3 个部分，主要目的就是为了将各种对象的耦合度降到最低。被分割的 3 个部分为：模型（Model）、视图（View）和控制器（Controller）。

- 模型（Model）：代表了应用程序的数据和处理这些数据的规则。当模型发生变化时，它会通知视图，并为视图提供查询模型相关状态的能力；
- 视图（View）：用来组织模型的内容，它从模型中获取数据，然后将数据展现给用户，该角色往往由 JSP 来承担；
- 控制器（Controller）：负责从客户端接受请求，并把这些请求转换成某种行为。这些行为往往由模型来实现，这些行为完成以后，再选择一个视图来呈现给用户。

如果要求使用已有的知识来实现一个 MVC 构架的 Web 应用程序的话，可以这样来实现：由 Servlet 来充当控制器的角色，它接受请求，根据请求信息的不同将它们分发给合适的 JSP 页面来作为用户的响应，同时，Servlet 还需要实例化一个 JavaBean 对象，JSP 就可以通过使用 JavaBean 的相关标签（如<jsp:getProperty>）来得到 JavaBean 的数据，结构如图 13.2 所示。

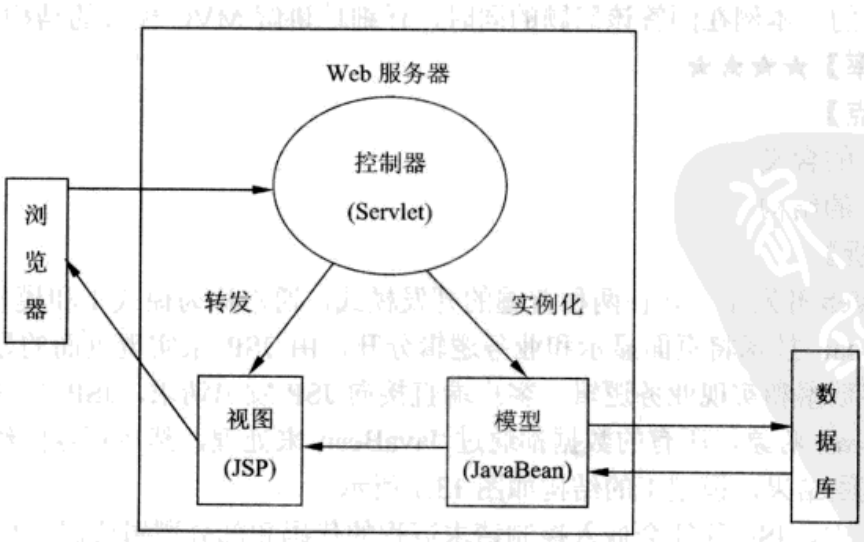


图 13.2　模型 2（MVC）结构图

采用模型 2，可以将页面的显示、业务逻辑的处理和流程的控制很清晰的分离开来。JSP 负责数据的显示，JavaBean 负责业务逻辑的处理，Servlet 负责流程的控制。MVC 模式使得 Web 应程序很容易维护和扩展，因为各个部件的功能不同，可以由不同的人进行开发和维护。例如，美工专门负责 JSP 页面，充分发挥他们的美术和设计才能，程序编写人员负责业务逻辑的实现。

注意：MVC 的 3 个部分是一个抽象的模型概念，具体情况下的实现各有不同。特殊情况下，也可以用 JSP 来做控制器，Servlet 做表现层。以上 MVC 实现是一种比较常规的实现方式。

【答案】

MVC 是一种设计模式，它要求应用程序的输入、处理和输出 3 者分离。使用 MVC 应用程序被分成 3 个核心部件：模型（Model）、视图（View）、控制器（Controller）。MVC 的名字就是由它们 3 者的首字母组成。

- 视图，是用户看到并与之交互的界面。可以是各种各样的，例如：JSP、HTML 等，甚至文本字符也可以是；
- 模型，表示数据和业务规则。它往往代表了一个应用程序的核心业务及其数据模型，由于应用于模型的代码只需写一次就可以被多个视图重用，所以把模型独立出来减少了代码的重复性；
- 控制器，接受用户的输入并调用模型和视图去完成用户的需求。所以当单击 Web 页面中的超链接和发送 HTML 表单时，控制器（例如，Servlet）本身不输出任何内容和做任何处理。它只是接收请求并决定调用哪个模型构件去处理请求，然后确定使用哪个视图来显示模型处理返回的数据。

面试题 132　如何编写一个 MVC 的 Java Web 应用程序

在不引入任何框架的情况下，使用原生态的 Java Web 技术开发 MVC 模式的 Web 程序，往往使用 Servlet、JSP 和 JavaBean 3 种技术，Servlet 充当控制器的角色，JSP 作为视图，而 JavaBean 则是模型。本例在回答该问题的同时，全面地讲解如何用 Java Web 技术开发一个 MVC 模式的 Web 程序。

【出现频率】★★★★

【关键考点】

- MVC 的含义；
- Servlet、JSP 和 JavaBean 的使用方法。

【考题分析】

其实，本面试题与其是说在考察求职者的编程能力，倒不如说是在考察设计能力。因为，Servlet、JSP 和 JavaBean 使用非常普遍，用法都比较简单，但是如何把它们组织在一起，体现 MVC 的思想呢？

首先，JSP 作为视图是无可厚非的，它就只用来展示数据；然后，那么数据从哪里来呢？肯定是从 JavaBean 里来，使用<jsp:useBean>等动作标签是一个不错的选择；最后，请求各种各样，如何来控制这些请求的流向呢？当然是 Servlet 的工作了，区分不同的请求最

简单的办法就是请求参数了。

说明：请求参数是一种比较简单的区分请求的办法，还有其他一些更高级的办法，例如，根据 URL 的目录结构来确定也是可以的。

根据以上介绍的思路，这里开发一个简单的 MVC 模式的示例程序。

（1）开发 Model。也就是开发 JavaBean，它其实很简单，就是一个普通的 Java 类，示例代码如下：

```
package javabean;                                          //定义包名
import java.io.Serializable;                               //引入序列化接口
//一个 JavaBean
public class MyBean implements Serializable {              //定义类
    // 序列化 id
    private static final long serialVersionUID = 1L;
    // properties
    private String name;                                   // 姓名
    private int age;                                       // 年龄
    //setters and getters
    public String getName() {
        return name;
    }
    public void setName(String username) {
        this.name = username;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
}
```

以上的 MyBean 类，包含两个可读写的属性：name 和 age。

（2）开发 Views。也就是开发 JSP，这里创建了两个 JSP 文件：1.jsp 和 2.jsp。代表不同请求的显示效果，其中 1.jsp 的代码如下：

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%
    String path = request.getContextPath();
    String basePath = request.getScheme() + "://" + request.getServerName()
    + ":" + request.getServerPort() + path + "/";
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <base href="<%=basePath%>">
    </head>
    <body>
        <!-- 用动作标签使用 JavaBean，范围是 request  -->
        <jsp:useBean id="myBean" scope="request" class="javabean.MyBean"
        ></jsp:useBean>

        用户姓名:
        <jsp:getProperty name="myBean" property="name"/>, <!-- 打印姓名信息
        -->
```

```
        用户年龄:
        <jsp:getProperty name="myBean" property="age"/> <!-- 打印年龄信息
        -->
    </body>
</html>
```

2.jsp 的代码如下:

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%
    String path = request.getContextPath();
    String basePath = request.getScheme() + "://" + request.getServerName()
    + ":" + request.getServerPort() + path + "/";
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <base href="<%=basePath%>">
    </head>
    <body>
        <!-- 用动作标签使用 JavaBean，范围是 request  -->
        <jsp:useBean id="myBean" scope="request" class="javabean.MyBean
        "></jsp:useBean>

        用户信息:
        <jsp:getProperty name="myBean" property="name"/>,
                                                    <!-- 打印姓名信息 -->
        <jsp:getProperty name="myBean" property="age"/> <!-- 打印年龄信息
        -->
    </body>
</html>
```

1.jsp 和 2.jsp 的区别就在于它们展示数据的方式不一样，1.jsp 是把姓名和年龄分开打印，而 2.jsp 则是一起打印。

（3）开发 Controller。也就是开发控制器 Servlet，它首先是获取一个请求参数，然后根据请求参数的值来转发相应的请求，示例代码如下:

```
package controller;                                          //定义包名

import java.io.IOException;                                  //引入需要的类和接口
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
//控制器 Servlet
public class MyController extends HttpServlet {              //定义 Servlet 类
    //构造方法
    public MyController() {
        super();
    }
    //控制器处理方法
    public void doGet(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
        String who = request.getParameter("who");            //获取参数
        if("1".equals(who)){                                 //判断参数的值
            MyBean mb = new MyBean();                        //创建 Bean 数据
            mb.setName("zhangsan");
```

```
            mb.setAge(10);
            request.setAttribute("myBean", mb);          //保存到 request 中
            //转发请求
            request.getRequestDispatcher("/1.jsp").forward(request,
            response);
        }else if ("2".equals(who)){
            MyBean mb = new MyBean();                    //创建 Bean 数据
            mb.setName("lisi");
            mb.setAge(20);
            request.setAttribute("myBean", mb);          //保存到 request 中
            request.getRequestDispatcher("/2.jsp").forward(request,
            response);
        }
    }
    //post 处理方法，直接调用 doGet()方法
    public void doPost(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
        this.doGet(request, response);
    }
}
```

注意：Servlet 的类完成开发以后，还需要在 web.xml 文件中进行相应的配置才能正常使用，否则 Web 容器是无法识别该 Servlet 资源的。

经过以上 3 步开发以后，把它们放在一个 Web 应用程序中的相应位置，就可以模拟出一个简单的 MVC 模式的程序了。当浏览器访问该 Servlet 时，提供的“who”参数的值是“1”，则展现出如图 13.3 所示的结果；若提供的是“2”，则展现出如图 13.4 所示的结果。

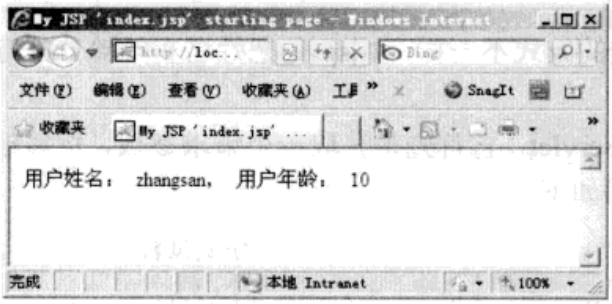


图 13.3　请求 1 的结果

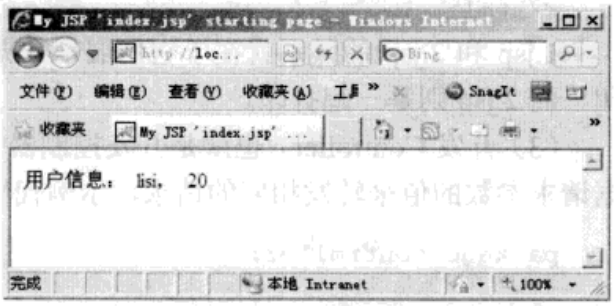


图 13.4　请求 2 的结果

【答案】

采用 Servlet、JSP 和 JavaBean 开发一个 MVC 模式的应用程序时候，主要有以下几步：

（1）开发 Model。主要是开发 JavaBean，它的作用是数据的存取。

（2）开发 Views。主要是开发 JSP，它的作用是从 JavaBean 中读取数据进行展示。

（3）开发 Controller。主要是开发 Servlet，它的作用是根据请求的参数或 URL 不同，转发相应的请求，一般还会进行数据的创建和填充。

面试题 133　Struts 框架是如何体现 MVC 模式的

Struts 框架是一款比较老资格的 MVC 开发框架了，拥有大量支持它的开发者，采用它开发的 MVC 模式的 Web 应用程序也是层出不穷。不论是开发新的项目，还是对老项目进

行维护，Struts 的使用都是一个面试重点。本例在回答该问题的同时，详细地讲解 Struts 的架构。

【出现频率】★★★★★

【关键考点】

- MVC 的含义；
- Struts 的架构；
- Struts 的基本工作原理。

【考题分析】

Struts 是 Apache 组织的一个开源项目（地址：http://struts.apache.org/），它采用 MVC 模式，能够很好地帮助 Java 开发者利用 Java EE 来开发 Web 应用程序。它让开发者能够换一种角度来思考如何使用 Servlet、JSP 等 Web 组件，开发者可以把大多数精力集中在业务逻辑的代码上来。另外，Struts 还提供了许多的 JSP 标签库，让 JSP 的开发更加容易。

说明： 这里说的 Struts 指的是 Struts 1.x 的版本，而不是目前刚出不久的 Struts 2，尽管从名字上看起来，Struts 2 是对 Struts 1 的升级，但是它们的架构和原理都有很大的不同，读者在使用时应该注意区分。

Struts 的控制器（Controller）由 ActionServlet、Action 和 struts-config.xml 组成。ActionServlet 是 Struts 的入口，所有的请求都会通过它来处理，然后由它来决定由相应的 Action 来处理请求。Action 代表了一次动作，如用户注册、购买商品等，开发者的业务逻辑代码也会在这里添加。配置文件 struts-config.xml 是对整个 Struts 的配置，包括 ActionServlet 应该将请求转发给哪个 Action，Action 处理完成以后，又该由哪个 JSP 文件作为响应等。

Struts 的模型（Model）主要由 ActionForm 来实现。它有一点类似于 JavaBean，包含了若干可读可写的属性，用于保存数据，也有数据验证的功能。一般来说，一个 Action 会配备一个 ActionForm。

Struts 的视图（View）主要由 JSP 来实现。JSP 在显示的数据可以来自 ActionForm，也可以是 Action 保存在作用范围（request、session、application）的数据。当然，使用 Struts 自带的标签库可以起到最大的简化作用。

使用 Struts 开发 Web 应用程序以后，开发者的思考方式需要做一下改变，Action、ActionForm 和 JSP 是一个整体了，每一次 HTTP 请求都需要它们三者协作来完成。JSP 代表用户可以看到的东西，ActionForm 代表的是数据，Action 代表的是业务逻辑。图 13.5 所示的是 Struts 的 MVC 各个组成部分，及其它们之间是如何一起协调工作的。

一次典型的 Struts 请求是这样完成的：首先，有浏览器客户端发送请求，请求的地址，默认为以“.do”结尾。然后，ServletAction 接收到请求以后，会根据请求的路径和参数来判断由哪个 Action 来处理该次请求。等到 Action 处理完成以后，通常是 execute 方法调用完成以后，Struts 会根据该方法返回的 ActionForward 来判断由哪个 JSP 来作为最终的响应，该过程如图 13.6 所示。

以上就是 Struts 最核心的设计思想，开发人员大多数的时候只需要完成配置文件和 Action，将大多数的精力集中在 Action 中的业务逻辑实现上面。如果业务上有改动，只需要修改 Action 即可，如果是显示上有变化，也只需要修改 JSP，两者实现了松耦合，互不

影响。

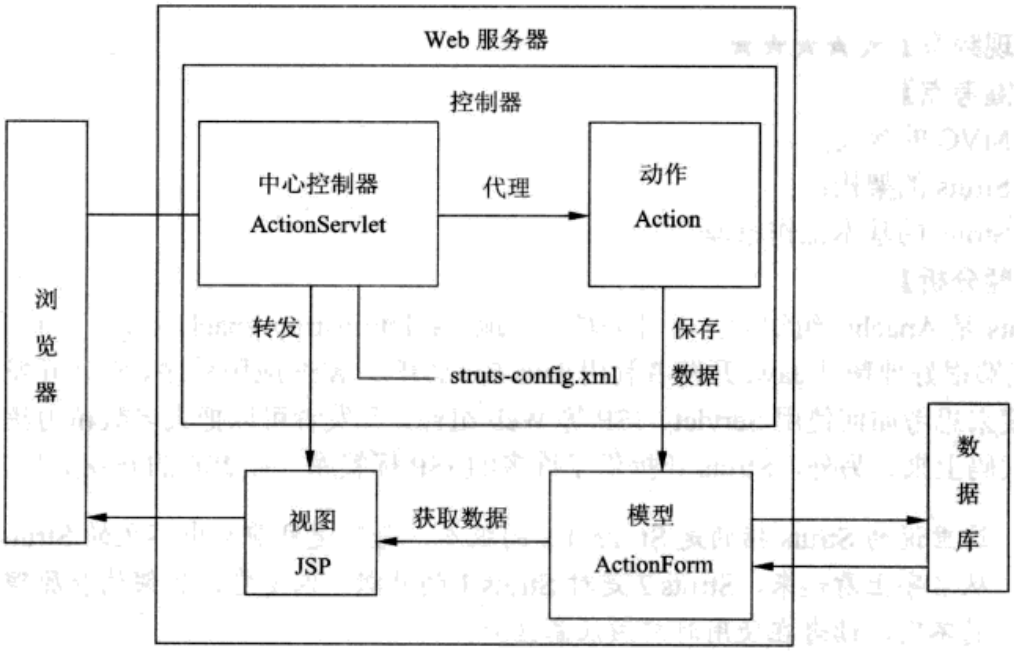


图 13.5　Struts MVC 结构图

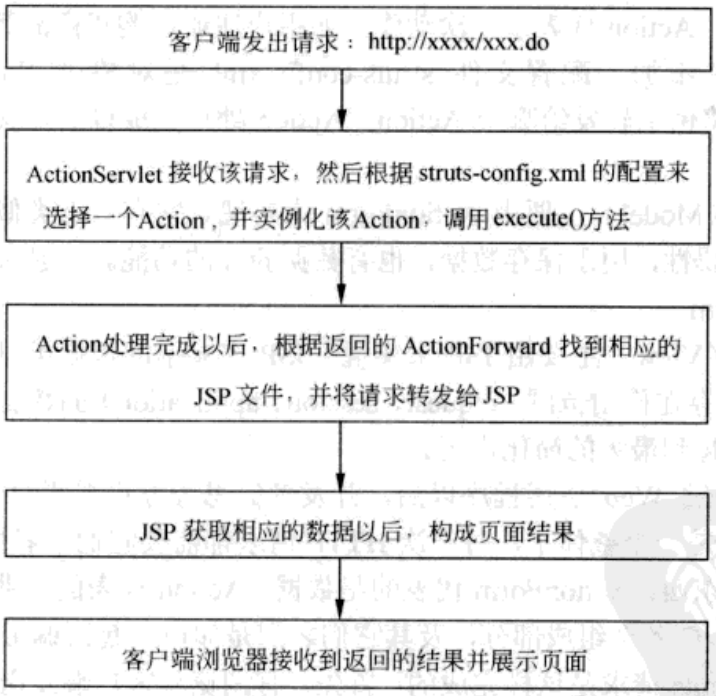


图 13.6　Struts 处理请求的过程和工作原理

【答案】

Struts 框架为开发者提供了 MVC 的 3 个逻辑组成部分，主要由 ActionServlet、Action 和 struts-config.xml 配置文件组成控制层，由 ActionForm 来承担模型层的功能，而 Struts 下的视图依然由 JSP 来完成。

它们大致是以这样的思路来处理每一次请求的：由 ActionServlet 接受一个请求，然后

根据 struts-config.xml 中的配置，来判断由于哪个 Action 来处理请求和由哪个 ActionForm 来保存数据，再通过 Action 的返回值来判断应该由哪个 JSP 来负责数据的展示，最终由 JSP 来完成结果响应。

面试题 134　开发一个 Struts 应用程序的思路是什么

在没有 Struts 框架的情况下，开发者依然能够开发出 MVC 模式的 Web 应用程序。但是有了 Struts 框架以后，开发者可以更容易的开发出 MVC 模式的 Web 程序了。那么 Struts 的基本使用思路是怎样的呢？本例在回答该问题的同时，详细地讲解 Struts 的 Action、ActionForm、struts-config.xml 等使用方法。

【出现频率】★★★★

【关键考点】

- Struts 的基本工作原理；
- Action 的定义和使用方法；
- ActionForm 的定义和使用方法；
- struts-config.xml 文件的使用方法；
- JSP 如何接收来自 Action 的数据。

【考题分析】

概括起来，使用 Struts 开发 Web 应用程序可以分为以下 4 个步骤：

（1）搭建 Struts 开发环境，主要包括必需的 jar 文件、web.xml 的配置和创建 struts-config.xml 文件。

（2）实现 Views，主要是创建 JSP 文件。

（3）实现 Model，主要是创建和配置 ActionForm 类。

（4）实现 Controller，主要是创建和配置 Action 类。

1. 搭建Struts开发环境

因为 Struts 属于 Java EE 规范以外的第三方技术，所以使用 Struts 时，必须遵守 Struts 的相关规定，才能将 Struts 整合到 Web 应用程序中。其实，Struts 在被整合到某个 Web 应用程序时，也是需要遵守 Java EE 的相关规范的，所以，只要读者对 Web 应用程序的规范比较熟悉，就很好理解 Struts 的整合细节了。

说明：一些 IDE（集成开发环境，如 eclipse、netbeans）提供了图形化界面的工具，可以很方便的为 Web 应用程序添加 Struts 环境，其实这些工具的功能本质上和以下几个步骤是类似的。

对 Web 应用程序添加 Struts 功能，可以分为以下几个步骤。

（1）复制 Struts 所需的 jar 文件到 Web 应用的/WEB-INF/lib 目录下面。首先，需要从 Struts 的网站（地址：http://struts.apache.org/）下载完成的开发包，笔者使用的版本为 1.2.9。从下载的 Zip 文件中可以找到 8 个 jar 文件，将它们全部复制到需要整合 Struts 功能的 Web 应用的/WEB-INF/lib 目录下面。

（2）在 web.xml 中配置 ActionServlet。ActionServlet 是所有 Struts 请求的入口，它本质

上就是一个 Servlet，它的默认 URL 匹配方式为“/*.do”，所有以“.do”结尾的 URL 都会由 ActionServlet 来处理，以下是一个示例配置：

```
...
    <!--  配置 ActionServlet 的 Servlet  -->
    <servlet>
        <servlet-name>action</servlet-name>
        <servlet-class>org.apache.struts.action.ActionServlet</servlet-
        class>
        <!--  指定 Struts 的配置文件地址  -->
        <init-param>
            <param-name>config</param-name>
            <param-value>/WEB-INF/struts-config.xml</param-value>
        </init-param>
        <init-param>
            <param-name>debug</param-name>
            <param-value>3</param-value>
        </init-param>
        <init-param>
            <param-name>detail</param-name>
            <param-value>3</param-value>
        </init-param>
        <load-on-startup>0</load-on-startup>
    </servlet>
    <!--  配置 ActionServlet 的 URL 匹配  -->
    <servlet-mapping>
        <servlet-name>action</servlet-name>
        <url-pattern>*.do</url-pattern>
    </servlet-mapping>
...
```

（3）创建 Struts 的配置文件 struts-config.xml。一般来说，配置文件存放在 Web 应用程序的“/WEB-INF/struts-config.xml”，通过该配置文件指定 Struts 包含哪些 Action、ActionForm 等，示例配置如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts
Configuration 1.2//EN" "http://struts.apache.org/dtds/struts-config
_1_2.dtd">
<struts-config>
    <data-sources />
    <form-beans>
        <!--  放置所有的 ActionForm 配置  -->
    </form-beans>
    <global-exceptions />
    <global-forwards />
    <action-mappings>
        <!--  放置所有的 Action 配置  -->
    </action-mappings>
    <!--  配置资源文件的路径  -->
    <message-resources parameter="xxx.xxx.xxx.ApplicationResources" />
</struts-config>
```

完成以上 3 步以后，就可以在 Web 应用程序中使用 Struts 开发应用了。

2. 实现Views

JSP 在 Struts 中的作用主要有两个：提供用户输入接口和展示数据。大多数的网页，

都包含了两个方面的内容，一个是用户可以输入数据的接口，例如，文本输入框、下拉菜单、提交按钮等。另一方面，就是来自后台的数据展现，如商品列表、数据状态等。一般来说，Struts 程序是不会让用户直接访问 JSP 文件的，而是通过 Action 进行转发。所以，JSP 表单的数据一般来自于 ActionForm，而其他的动态数据一般来自于 Action 的 request 中的属性（Attribute）。

另外，Struts 为 JSP 提供了丰富的标签库，尤其是 HTML 标签库，让 JSP 的表单和 ActionForm 称为了一个整体，提交表单的时候可以自动填充 ActionForm，响应的时候可以为表单的输入框自动添加数据。

3．实现Model

ActionForm 充当的是数据层的作用，它为开发者保存数据带来了便利，尤其对于大型 Web 项目来说，一个页面可能会包含几十个甚至上百个输入框，如果开发者对每一个请求参数都通过 request.getParameter()的方式去获取的话，工作量会非常的大，而且不利于维护。有了 ActionForm 以后，它可以帮助开发者自动填充数据。而且，它还提供了一些数据验证的功能。

自定义的 ActionForm 需要继承自 org.apache.struts.action.ActionForm 类，然后把需要添加的字段作为成员变量进行设置，并为这些成员变量加上 setter 和 getter 方法。以下为一个包含商品名称变量的 ActionForm：

```
import org.apache.struts.action.ActionForm;
//定义 ProductForm，继承自 ActionForm 类
public class ProductForm extends ActionForm {
    //商品名称
    private String name;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

ActionForm 定义完成以后，还需要在 struts-config.xml 中进行配置以后，才能供 Action 使用，使用标签为：<form-bean>，它包含 form 的名称 name 属性和类型 type 属性，示例代码如下：

```
<form-beans>
    <form-bean name="productForm"
        type="com.throne212.javaweb.ch15.struts.form.ProductForm" />
</form-beans>
```

4．实现Controller

Action 代表了一次动作，逻辑代码、数据装填代码和请求转发代码都会方在 Action 中。它是 Struts 最核心也是最重要的地方，程序员大多数时候接触的地方也会是它。

自定义的 Action 需要继承自 org.apache.struts.action.Action 类，而且需要覆盖它的 execute()方法，当有请求指向该 Action 的时候，Struts 会实例化该 Action，再调用 execute()

方法，以下是 execute()的完整定义：

```
public ActionForward execute(ActionMapping mapping, ActionForm form,
HttpServletRequest request, HttpServletResponse response) {
    ...
}
```

它包含 4 个参数，即 ActionMapping、ActionForm、HttpServletRequest、HttpServletResponse。最后的两个参数与 Servlet 的请求和响应是完全相同的。ActionMapping 指的是该 Action 映射的信息，通常会调用该对象的 findForward()来转到下一个资源或视图 JSP，该方法的参数即为下一个资源的名称，在 struts-config.xml 中通过<forward>进行配置。ActionForm 参数也就是与 Action 匹配的 ActionForm 对象，它是一个超类，往往需要进行强制类型转换，通过<action>的 name 属性指定。以下是一个包含商品浏览和搜索的 Action 的示例代码（其中，Product 类代表一件商品）：

```
package myaction;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
//定义 ProductAction，继承自 Action
public class ProductAction extends Action {
    //重载 execute()方法
    public ActionForward execute(ActionMapping mapping, ActionForm form,
    HttpServletRequest request, HttpServletResponse response) {
        Product[] ps = new Product[10];
        for(int i=0;i<ps.length;i++){
            ps[i] = new Product("测试商品"+i,100,"好东西！");
        }
        request.setAttribute("products", ps);  //设置数据到 request 范围中
        return mapping.findForward("product"); //转到商品展示页面
    }
}
```

然后，可以在 struts-config.xml 中进行如下的配置：

```
...
<action-mappings>
    <action path="/ProductAction" name="productForm"
        type="myaction.ProductAction">
        <forward name="product" path="/demo.jsp"></forward>
    </action>
</action-mappings>
...
```

注意：Action 的 path 属性需要以斜线“/”开头，它代表以本应用上下文为根目录。同样，<forward>标签的 path 属性，也是需要以斜线“/”开头的。

【答案】

在使用 Struts 框架开发一个 Web 应用程序的时候，主要需要进行以下 4 个步骤：

- 搭建 Struts 开发环境，主要包括必需的 jar 文件、web.xml 的配置和创建

struts-config.xml 文件；

- 实现 Views，主要是创建 JSP 文件，可以使用 Struts 提供的标签库；
- 实现 Model，创建和配置 ActionForm 类，它需要继承自 Struts 提供的 ActionForm 类，然后使用<form-bean>标签在 struts-config.xml 中进行配置；
- 实现 Controller，创建和配置 Action 类，它需要继承自 Struts 提供的 Action 类，并实现 execute()方法，然后使用<action>标签在 struts-config.xml 中进行配置。

面试题 135　Struts 提供了哪几类 Action

如果始终让一个 Action 类代表一个动作的话，当有成千上万个动作的时候，岂不是要写成千上万个 Action 类。其实，Struts 针对该问题提供了多种 Action 来予以解决，它们让 Struts 更加的灵活和易扩展。本例在回答该问题的同时，全面地讲解 Struts 的每种 Action 的使用方法和使用场合。

【出现频率】★★★★★

【关键考点】

- Struts 的基本工作原理；
- Action 的含义和作用；
- DispatchAction 的用法与适用场合；
- LookupDispatchAction 的用法与适用场合；
- MappingDispatchAction 的用法与适用场合。

【考题分析】

Struts 针对不同的情况，除了最原始的 Action 以外，还提出了 DispatchAction、LookupDispatchAction 和 MappingDispatchAction，后面 3 者都可以将多个 action 方法写在一个 Action 类中。例如，一个新闻管理系统，需要对新闻进行创建、编辑、删除等操作，如果将这一类动作放在一个 Action 类，对于这些动作是非常好管理的。表 13.1 列出了所有 Action 及其它们的使用方法和适用情况。

表 13.1　Struts的多种Action

名　称	适 用 情 况	使 用 简 介
Action	普通的Action	直接使用即可
DispatchAction	表单包含多个参数时适用	需要从客户端接收一个参数，该参数的值对应方法的名称
LookupDispatchAction	表单包含多个提交按钮的时适用	需要重载getKeyMethodMap()方法，返回参数与方法名之间的对应关系
MappingDispatchAction	客户端不便于传递参数时适用	需要用配置文件的<action>标签的parameter属性指定Action的方法名称

1．DispatchAction

DispatchAction 的实现类为 org.apache.struts.actions.DispatchAction，它需要从客户端接收一个参数，该参数的值与方法名相对应。Struts 会根据 struts-config.xml 中配置的<action>

标签的 parameter 属性来获取该请求参数，然后利用 Java 的反射机制调用方法名与参数值相同的方法。这些方法的参数列表和返回类型与 Action 的 execute()方法完全相同。

2．LookupDispatchAction

LookupDispatchAction 的实现类为 org.apache.struts.actions.LookupDispatchAction，它主要适合于一个表单有多个按钮的情况，例如新闻表单，既可以用来添加新闻，又可以编辑新闻。LookupDispatchAction 也需要从客户端接收一个参数，但是参数的值与方法名的对应关系需要用一个 getKeyMethodMap()方法来确定，该方法返回一个 Map，该 Map 的 Key 和 value 与资源文件 ApplicationResources.properties 的 key 和 value 对应。

3．MappingDispatchAction

MappingDispatchAction 的实现类为 org.apache.struts.actions.MappingDispatchAction，它无须从客户端接收参数，直接通过 struts-config.xml 中的<action>标签的 parameter 属性来指定方法名，客户端不需要做任何的配合，使用起来就好像一个普通的 Acton 一样。它的缺点在于可能会造成 struts-config.xml 特别的长，因为每一个方法都需要加一个<action>标签。尽管如此，它还是比原始的 Action 要好一点，至少不用写那么多的 Action 类。所以，MappingDispatchAction 更适合直接通过 URL 访问 Action 的一些情况。

【答案】

Struts 一共提供了 4 种 Action，分别是 Action、DispatchAction、LookupDispatchAction 和 MappingDispatchAction：

- 普通的 Action 类，每一个动作就对应一个 Action 类，一般不提倡使用。
- DispatchAction 类，需要从客户端提供一个特殊的参数，参数的值等于方法的名字。对于一般的表单提交性的动作，可以使用。
- LookupDispatchAction 类，需要实现 getKeyMethodMap()方法。适用于一个表单有多个提交按钮的情况。
- MappingDispatchAction 类，尽管无需从客户端获取参数来决定调用哪个方法，但是需要在 struts-config.xml 中配置多个<action>标签。适用于客户端不方便传参，且动作不是太多的情况。

13.2　Hibernate

大多数的应用程序都有数据持久化的需求，Hibernate 是一个不错的选择。Hibernate 不仅仅是数据持久化的一种解决方案，更是对象关系映射模型（ORM）的一个典型代表，拥有众多的支持者，属于 Java 程序的持久化方案的首选。本节将集中讨论有关 Hibernate 和 ORM 的常见面试题。

面试题 136　什么是对象关系映射模型（ORM）

以面向对象的思维方式去操作关系数据库中的数据，这是 Java、C#等面向对象语言的

开发者所希望的。在以前的 J2EE 开发中，EJB 的实体 Bean 能够达到这样的目的，但是真正把对象与关系之间的映射机制（简称为 ORM）建立并完善起来的是 Hibernate，它的普及程度之高，逐渐成为了事实上的 ORM 标准。因此，学习 Hibernate 是离不开对 ORM 的理解的。本例在回答该问题的同时，全面地讲解 ORM 的含义。

【出现频率】★★★★

【关键考点】

- ORM 的含义；
- ORM 的作用。

【考题分析】

任何计算机程序都是由指令和数据两部分组成，持久化数据对于大多数程序，尤其是企业级应用程序，是必不可少的。数据的持久化，也就是把数据保存起来，并且还可以供程序获取这些数据的全部或某一部分。对于 Java 程序来说，通常有 3 个持久化数据的途径，文本文件、对象的序列化和数据库。

（1）使用文本文件来保存数据指的是，把数据按照一定的格式，如键值对、以逗号相隔等形式，利用 Java 的文件系统 I/O 功能把数据进行保存和读取。这种数据持久化方式，对于数据量少和数据结构单一的情况是比较好的。如果数据量太大，由于 I/O 的速度比内存操作的速度要慢得多，会造成很大的性能问题。另外，如果数据的结构太复杂，简单的符号相隔的形式实在难以满足数据的存取功能。

（2）对象的序列化，指的是把实现了 java.io.Serializable 接口的类的实例对象保存在文件系统中，需要读取数据的时候，再通过反序列化机制把文件系统的数据又映射回内存中，还原原来的数据。这种数据持久化方式就可以满足结构复杂的数据的存取，因为对象之间的关联和组合等关系可以用来构成这些复杂的结构。但是，对象序列化本质上还是需要使用 Java 的文件系统 I/O 功能，很难解决数据量大的性能问题。而且，它也很难能完成大多数企业级程序所需要的数据检索功能。

（3）关系数据库是目前数据持久化最成熟的产品，也是 Java 程序首选的数据持久化解决方案。它能够存储海量的数据，提供高性能的数据检索功能，大多数数据库厂商也提供了 Java 的 JDBC 驱动。但是，如果使用 JDBC 来操作数据，会有以下几项不可避免的弊端。

- 数据操作的代码量巨大：使用 JDBC 写过大型应用系统的开发者都会有这样的感受，满篇都是 SQL 语句和类似的代码。每个操作数据的方法中，都会有打开连接、创建会话、定义 SQL、设置参数、执行 SQL、装填结果、关闭连接等代码，这些代码尽管类似，但是却不得不存在。
- 重复劳动的装填数据：当执行查询 SQL 语句以后，往往会使用一个 while 循环语句，把结果集 ResultSet 的数据先装填到数据对象（Data Object，简称 DO）中，再把这些数据对象放到一个链表或其他类型的集合中，例如，下面的代码：

```
...
while(rs.next){
    User user = new User();
    user.setName(rs.getString("username"));
    ...//设置 user 的每个字段
    list.add(user);
}
...
```

这些代码总是非常类似，却只能这样做。如果稍有不慎，还可能把数据装填错误，而且这些错误还不易被发现。既降低了开发效率，又不利于以后的维护。

- 面向对象的思想和关系模型之间的冲突：目前，面向对象编程已经是软件开发的主流思想了，它让开发者把事物抽象成为类和对象，对象之间的有关联、组合、聚合等关系，也有继承、多态等面向对象思想特有的关系。关系数据库是完全建立在关系模型基础上的，用开发者的角度来看，数据库中全是表格（table），表格之间可以通过外键达到一对一、一对多和多对多的关系。那么，应该怎样用关系模型来模拟面向对象思想中对象之间的关系呢？通过 SQL 和 JDBC 可以达到这样的目的，但是一旦关系复杂，SQL 语句可能会非常难写，即使写成了，也难免会有 Bug。更重要的是，它会进一步增大开发难度和维护难度，说不定程序员每天都在围着 SQL 来回的转悠。

以上介绍的这 3 点，是 Java 程序使用 SQL 和 JDBC 作为数据持久化解决方案不可回避的 3 个问题，尤其是第 3 点，它导致开发模型复杂的 Java 程序成为了高手的专利。此时，如果有一种工具，可以屏蔽对象模型与关系模型之间的差异，让开发者操作对象的时候自动完成对数据库的操作，就可以解决这些问题了。其实，这就是对象关系映射（ORM）工具，它的代表作有 Hibernate 和 ibatis。

说明：ORM 框架的底层实现是往往需要依赖 JDBC 的，它们是对 JDBC 的一种 ORM 思想的包装。

对于 Hibernate、ibatis 这样的 ORM 框架来说，其他数据持久化解决方案的缺点就是它们的优点。

【答案】

对象关系映射（Object Relational Mapping，简称 ORM）是一种为了解决面向对象与关系数据库之间的互不匹配的现象技术，起到了一座桥梁的作用。对于应用程序开发者来说，ORM 可以按照面向对象的思想来操作关系数据库中的数据。

ORM 的诞生是数据持久化技术发展的结果。由于目前关系数据库是数据持久化的最佳和最普遍的解决方案，但是程序员更愿意使用面向对象的思想对待数据，而不是关系模型，因此 ORM 的诞生就屏蔽了它们两者的差异，被大多程序员所接受。

面试题 137　Hibernate 的基本使用思想是什么

Hibernate 提供了丰富的 API，但是这些 API 如何使用呢？开发者除了应该建立起 ORM 的思想以外，还需要了解 Hibernate 是如何建立起映射关系的，开发的基本思想是什么？本例在回答该问题的同时，详细地讲解 Hibernate 的基本使用方法。

【出现频率】★★★★★

【关键考点】

- Hibernate 核心 API 的含义和使用方法；
- Hibernate 程序的基本结构和工作原理。

【考题分析】

Hibernate 是一个开放源代码的对象关系映射框架，它对 JDBC 进行了非常轻量级的对

象封装，使得 Java 程序员可以随心所欲的使用面向对象编程的思维来操纵数据库。Hibernate 可以应用在任何使用 JDBC 的场合，既可以在普通 Java 程序（或称为 Java SE 程序）中使用，也可以在 Java Web 应用程序中使用，还可以在 EJB 容器中使用，完成数据持久化的任务，极大的减轻了程序员在数据层的工作。

Hibernate 就好像 Java 程序与数据库之间的一座桥梁，它提供了丰富的 API 供程序员使用，当程序员通过这些 API 操作数据对象的时候，Hibernate 会自动完成与数据库之间的数据同步，还有跨数据库的作用。

数据库操作的基本单元是表格，而 ORM 技术操作的基本单元是实体对象，所以 Hibernate 持久化数据的操作方式又是一种全新的模式。在 Hibernate 的 API 中，有 5 个非常重要的接口：Session、SessionFactory、Configuration、Transaction 和 Query，它们同时也是 Hibernate 最核心的组成概念。

- Session 接口：Session 接口负责执行被持久化对象的增、删、查、改（CRUD）操作，有点类似于 JDBC 的 Connection 和 Statement，它代表了 Java 程序与数据库之间的一次交流。需要注意的是，Session 对象是非线程安全的，一般来说，一个线程包含一个 Session 对象；

注意：Hibernate 的 Session 不同于 Web 应用中的 HttpSession，它们两种是完全不相干的概念。

- SessionFactory 接口：是用来产生 Session 的工厂类，它负责为 Java 程序创建 Session。一般来说，一个 SessionFactory 代表了一个数据源，当需要操作多个数据库时，可以为每个数据库指定一个 SessionFactory。一般情况下，一个项目通常只需要一个 SessionFactory 就够了；
- Configuration 接口：负责 Hibernate 的配置工作，创建 SessionFactory 对象。在 Hibernate 的启动的过程中，Configuration 类的实例首先定位映射文档位置、读取配置，然后创建 SessionFactory 对象。
- Transaction 接口：负责事务相关的操作，它代表的是 Hibernate 的事务，本质上也是数据库事务。Transaction 是可选的，开发者也可以设计编写自己的底层事务处理代码。
- Query 接口：负责执行各种数据查询功能，它可以使用 Hibernate 特有的 HQL 语言和 SQL 语句两种方式。

以上这 5 个 API 接口是几乎所有 Hibernate 程序都会使用到的，也是程序员使用 Hibernate 进行开发的基础。它们之间的关系大致是这样的：首先，由 Configuration 获取配置信息并做一些初始化的工作；接着，通过 Configuration 创建 SessionFactory；再通过 SessionFactory 创建 Session；然后，程序就可以使用 Session 完成一般的增、删、查、改操作；如果需要使用复杂一点的查询功能，则可以通过 Session 获取 Query，执行查询操作；如果需要使用事务 Transaction，也是在 Session 的范围内使用，如图 13.7 所示。

【答案】

Hibernate 对 JDBC 进行了一种包装，提供了另外一套的 API 来操作数据。其中与数据直接打交道的组件是 Session，它是 Hibernate 的一级缓存，保持着应用程序与数据库之间的连接，通过它来保持内存与数据库之间的数据同步。Session 是由 SessionFactory 来创建的，它代表的含义是数据源，通常指向某个数据库。而 SessionFactory 是由 Configuration

来创建，它代表的是 Hibernate 程序的数据源及其相关属性的配置信息。如果需要使用事务，则可通过 Session 来打开、提交和回滚事务。另外，如果需要使用比较复杂的查询功能，还会使用 Query 接口，它也是通过 Session 来获得。

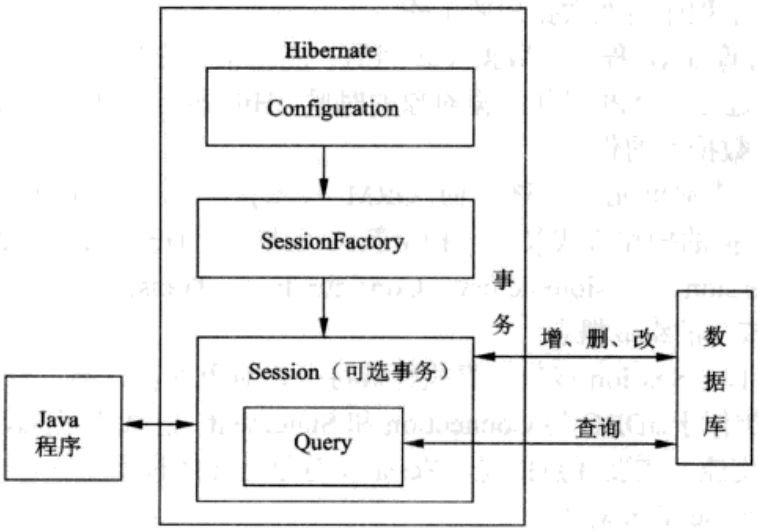


图 13.7　Hibernate 各个组件之间的工作关系

Hibernate 的 API 中，Session、SessionFactory、Configuration、Transaction 和 Query 是它的 5 个最重要的接口，它们一起工作，构成了 Hibernate 基本的使用思想。

面试题 138　Hibernate 的实体存在哪几种状态

实体，是 Hibernate 操作数据的基本单位，它对应于某个 Java 类。实体是数据在内存中的一种表现形式，而在数据库里则是表格的行。实体可能只存在于内存，而不存在数据库中；也可能内存里的数据与数据库里的数据不一致；还有可能只在数据库中，而不在内存中。Hibernate 是如何来对待实体的不同状态呢？本例在回答该问题的同时，详细地讲解 Hibernate 中的实体含义及其他的生命周期。

【出现频率】★★★★★

【关键考点】

- Hibernate 的实体的含义；
- Hibernate 的实体的声明周期。

【考题分析】

实体，作为数据的一种装载单元，它在 Hibernate 的管理过程中，会不断地发生数据和状态的变化。Hibernate 不仅仅是一个 ORM 的 API 工具，它还是一个容器，它可以帮你管理实体对象的生命周期，这个容器就是 Session 的缓存。

Hibernate 的实体对象有 3 种状态：瞬时态（Transient）、持久态（Persistent）、脱管态（Detached）。实体对象的状态会随着 Hibernate 的 API 的调用而发生变化，开发者也应该根据实体对象状态的不同，而使用不同的 API。

- 瞬时态：由 new 语句开辟内存空间的新 java 对象，如果没有变量对该对象进行引用，它将被 Java 虚拟机回收，例如下面的代码：

```
Person person=new Person("furong","女");
```

瞬时对象在内存孤立存在，它是携带信息的载体，不和数据库的数据有任何关联关系，它的最大特点就是没有主键。在 Hibernate 中，可通过 Session 的 save()或 saveOrUpdate()等方法将瞬时对象与数据库相关联，并将数据同步到数据库中，此时该瞬时对象转变成持久化对象。

- 持久态：处于该状态的对象在数据库中具有对应的记录，并拥有一个持久化标识。如果是用 Session 的 delete()方法，对应的持久对象就变成瞬时对象，因数据库中的对应数据已被删除，该对象不再关联数据库的任何记录。

当执行 Session 的 close()或 clear()、evict()之后，持久对象变成脱管对象。该对象虽然具有数据库识别值，但它已不在 Hibernate 持久层的管理之下了。

- 脱管态：当与某持久对象关联的 Session 被关闭后，该持久对象转变为脱管对象。当脱管对象被重新关联到 Session 上时，并再次转变成持久对象。脱管对象拥有数据库的识别值，可通过 update()、saveOrUpdate()等方法转变成持久对象。

脱管对象具有如下特点：本质上与瞬时对象相同，在没有任何变量引用它时，JVM 会在适当的时候将它回收，但是，它拥有主键。

图 13.8 展示了实体对象的生命周期的状态及其转换过程。

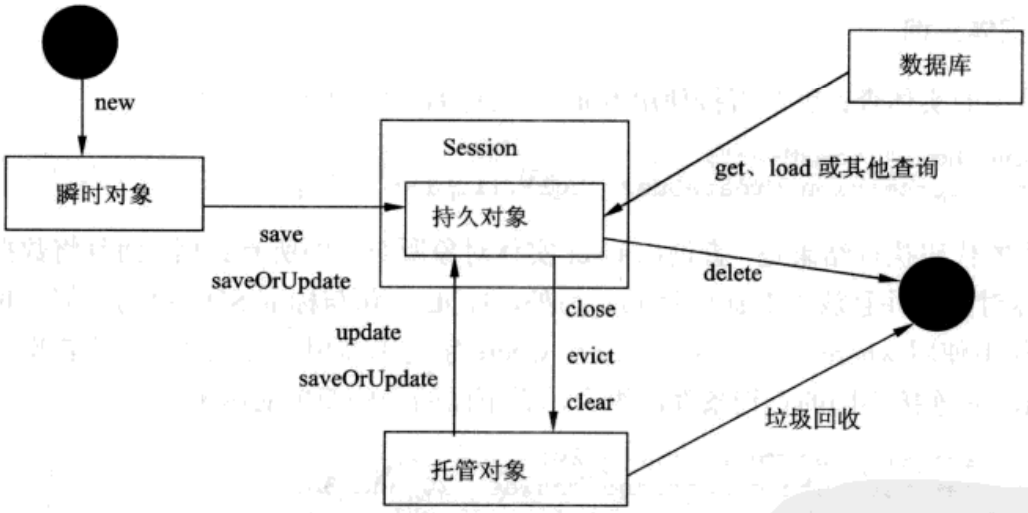


图 13.8　实体对象生命周期及其状态转换过程

【答案】

Hibernate 中的实体在它的生命周期里，一共存在下面 3 种状态。

- 瞬时状态：用 new 语句创建的实体对象就属于瞬时状态，它此时一般没有 id 值；
- 持久状态：存在于 Session 中的实体对象就属于持久状态，一般通过 save()、saveOrUpdate()等，方法转换而来；
- 托管状态：当实体从 Session 中脱离出来的时候，它的状态就是托管状态了，尽管它具有 ID 值，但是它并不在 Session 中，即使实体的数据发生变化，也不能同步到数据库中。通过调用 close()、evict()等方法转换而来。

面试题 139　HQL 查询语言的使用方法是什么

SQL 是用来操作数据库的查询语言，它可以用来查询关系数据库里的数据，被大多数的开发者所熟悉。Hibernate 则提出了一种语法与 SQL 类似，但是思想却是面向对象的 HQL 语言，用来完成 Hibernate 的查询功能。本例在回答该问题的同时，详细地讲解 Hibernate 查询功能的使用方法。

【出现频率】★★★★

【关键考点】

- HQL 的语法；
- Hibernate 的 Query 接口的使用方法。

【考题分析】

HQL（Hibernate Query Language）是 Hibernate 提供给开发者用来以面向对象的思想进行实体增、删、查、改操作的查询语言，它看起来跟 SQL 比较像，但是它提供了更加丰富的和灵活的查询特性，同时也提供了更加面向对象的封装，完整的 HQL 语句形式如下：

```
Select/update/delete ... from ... where ... group by ... having ... order
by ... asc/desc
```

1. 实体查询

最简单的实体查询就是直接使用 from 加实体名，例如下面的代码：

```
String hql="from User";
List list=session.CreateQuery(hql).list();
```

上面的代码执行结果是，查询出 User 实体对象所对应的所有数据，而且将数据封装成 User 实体对象，并且放入 List 中返回。另外，HQL 语句与标准 SQL 语句相似，也可以在 HQL 语句中使用 where 字句，并且可以在 where 字句中使用各种表达式，比较操作符以及使用 and，or 连接不同的查询条件的组合，看下面的一些简单的例子：

```
from User user where user.age=20;
from User user where user.age between 20 and 30;
from User user where user.name like '%zx%';
from User user where (user.age%2)=1;
from User user where user.age=20 and user.name like '%zx%';
```

2. 实体的更新和删除

在介绍 HQL 其他更为强大的查询功能前，先来讲解一下利用 HQL 进行实体更新和删除的技术。它与 SQL 的更新和删除有点类似，以下是更新 admin 用户年龄的示例代码：

```
Transaction trans=session.beginTransaction();
String hql="update User user set user.age=20 where user.name='admin'";
Query queryupdate=session.createQuery(hql);
int ret=queryupdate.executeUpdate();
trans.commit();
```

删除实体的操作与之类似，只不过使用的是 delete from 语句。

3．属性查询

有的时候，检索的数据并不需要获得实体对象所对应的全部数据，而只需要检索实体对象的部分属性所对应的数据。这时候就可以利用 HQL 属性查询技术，如下面程序示例：

```
List list=session.createQuery("select user.name from User user ").list();
for(Object o : list)
    System.out.println(o.toString());
}
```

上例只检索了 User 实体的 name 属性对应的数据，此时返回的包含结果集的 list 中每个条目都是 String 类型的 name 属性对应的数据。如果检索的是多个属性，则每条记录是以 Object 数组的形式返回，如下面程序：

```
List list=session.createQuery("select user.name,user.age from User user
").list();
for(int i=0;i<list.size();i++){
    Object[] obj=(Object[])list.get(i);
    System.out.println(obj[0]);
    System.out.println(obj[1]);
}
```

作为当今大多数深受面向对象思想影响的开发人员，可能会觉得上面返回 Object[]不够符合面向对象风格，那么可以利用 HQL 提供的动态构造实例的功能对这些平面数据进行封装，如下面的程序代码：

```
List list=session.createQuery("select new User(user.name,user.age) from
User user").list();
for(int i=0;i<list.size();i++){
    User user=(User)list.get(i);
    System.out.println(user.getName());
    System.out.println(user.getAge());
}
```

4．排序与分组

（1）order by 子句：与 SQL 语句相似，HQL 查询也可以通过 order by 子句对查询结果集进行排序，并且可以通过 asc 或者 desc 关键字指定排序方式，如下面的代码：

```
from User user order by user.name asc,user.age desc;
```

上面 HQL 查询语句，会以 name 属性进行升序排序，以 age 属性进行降序排序，而且与 SQL 语句一样，默认的排序方式为 asc，即升序排序。

（2）group by 子句与统计查询：在 HQL 语句中同样支持使用 group by 子句分组查询，还支持 group by 子句结合聚集函数的分组统计查询，大部分标准的 SQL 聚集函数都可以在 HQL 语句中使用，如 count()、sum()、max()、min()、avg()等，如下面的程序代码：

```
String hql="select count(user),user.age from User user group by user.age
having count(user)>10";
```

```
List list=session.createQuery(hql).list();
```

5. 参数绑定

Hibernate 中对动态参数绑定提供了丰富的支持，HQL 也像 JDBC 的 PreparedStatement 一样，可以使用占位符来实现参数动态绑定，而且提供了更丰富的特性。在 Hibernate 中，共存在 3 种参数绑定的方式具体如下。

（1）按参数名称绑定：在 HQL 语句中，定义命名参数使用“:”开头，形式如下：

```
Query query=session.createQuery("from User user where user.name=:username
and user.age =:userage ");
query.setString("username",name);
query.setInteger("userage",age);
```

上面代码中用:username 和:userage 分别定义了命名参数 username 和 userage，然后用 Query 接口的 setXXX()方法为参数设值，第 1 个参数为占位符名称，第 2 个为实际的值。

（2）按参数位置邦定：用”?”来定义参数位置，形式如下：

```
Query query=session.createQuery("from User user where user.name=? and
user.age =?");
query.setString(0,name);
query.setInteger(1,age);
```

同样使用 setXXX()方法设定参数，只不过这时的 setXXX()方法的第 1 个参数代表邦定参数在 HQL 语句中出现的位置编号（由 0 开始编号），第 2 个参数仍然是参数实际值。

注意：在实际开发中，提倡使用按名称邦定命名参数，因为这不但可以提供非常好的程序可读性，而且也提高了程序的易维护性。因为当查询参数的位置发生改变时，按名称绑定名参数的方式中，是不需要调整程序代码的。

（3）setParameter()方法：HQL 查询支持通过 setParameter()方法设置任意类型的参数，如下代码：

```
String hql="from User user where user.name=:username ";
Query query=session.createQuery(hql);
query.setParameter("username",name,Hibernate.STRING);
```

如上面代码所示，setParameter()方法包含 3 个参数，分别是命名参数名称、命名参数实际值，以及命名参数映射类型。如果类型不确定，可以省去第 3 个参数，因为 Hibernate 足够聪明，它会自动判断参数的类型

【答案】

HQL 是一种 Hibernate 的查询语言，语法类似于 SQL，但是 HQL 操作的对象是实体它的基本语法如下：

```
Select/update/delete ... from ... where ... group by ... having ... order
by ... asc/desc
```

HQL 语句一般在 Query 中使用，通过 Session 的 createQuery()方法获得。它可以用于实体查询、实体的更新和删除、属性查询、排序和分组查询等，也可以使用占位符来动态的设置参数。

面试题 140　如何使用 Hibernate 进行分页查询

分页查询对于 Web 应用程序来说，使用非常普遍。那么 Hibernate 能否为开发者提供跨数据库的分页查询功能呢？答案是肯定的，而且使用方法也比较简单。本例在回答该问题的同时，详细地讲解 Hibernate 分页查询的使用方法。

【出现频率】★★★★

【关键考点】

- Hibernate 查询功能的实现方法；
- Hibernate 分页查询 API 的使用方法。

【考题分析】

分页查询在 Web 应用程序中几乎是必不可少的。分页查询的实现方式包括两种：逻辑分页和物理分页。使用逻辑分页的话，需要控制从数据库中，读出特定起始和结尾行的数据记录是一件不易的事情，而且效率还不高；若使用物理分页，不同的数据库的 SQL 是不一样的，例如，MySQL 使用 limit 语句，而 Oracle 使用 rownum。

而 Hibernate 对分页查询的支持却非常的好，可以说程序员使用 Hibernate 来实现分页查询是傻瓜式的，简单到了极致，因为只需要通过调用两个方法告诉 Hibernate 从那行开始和最多的行数，其他的细节全部都交给 Hibernate 了。例如，以下是查询出第 2 页的所有用户数据示例，其中每页最大记录数为 10。

```
int pageNo = 2;
int pageSize = 10;
String hql="from User";
Query query = session.CreateQuery(hql);
query.setFirstResult((pageNo - 1) * pageSize);
query.setMaxResults(pageSize);
query.list();
```

说明：Hibernate 的分页查询也是可以跨数据库的，以上的示例代码在翻译成 SQL 语句的时候，SQL 语句会根据数据库的不同而各异，例如，MySQL 会使用 limit 关键字，Oracle 使用 rownum，而 SQL Server 则使用 top。

【答案】

Hiberante 在实现分页查询的时候，开发者除了提供 HQL 语句以外，还需要提供从哪行开始（startIndex）和最多的行数（maxLength）两个信息给 Hibernate 即可，并分别调用 Query 接口的 setFirstResult()和 setMaxResults()方法。

面试题 141　get()和 load()方法的区别是什么

get()和 load()方法都是 Session 接口的方法，它们都是用于获取实体对象的，都只需要提供两个参数，一个是实体类的 Class 对象或完整的类名，而另一个是 ID 值。那么，它们有什么区别呢？本例在回答该问题的同时，详细地讲解 get()和 load()方法的主要功能细节。

【出现频率】★★★★

【关键考点】

- get()和 load()方法的作用；
- get()和 load()方法的区别。

【考题分析】

根据主键去获取实体，是最常见也是最简单的查询。如果使用 JDBC 的话，可能会使用到如下这样的 SQL 去数据库得到查询结果：

```
select * from XXX where id=?;
```

然后再把所有的字段逐一的装填到实体对象中。但是，如果使用 Hibernate，一切都变得简单起来，只需要调用 get()或 load()方法即可。它们的参数列表相同，第 1 个是实体的类型或完整类名，第 2 个为主键。查询得到的实体对象，都将进入持久态。以下是查询主键为 100 的学生实体对象的示例代码：

```
Student stu1 = session.get(Student.class,(long)100);
Student stu2 = session.load(Student.class,(long)100);
//或者
Student stu1 = session.get(Student.class.getName(),(long)100);
Student stu2 = session.load(Student.class.getName(),(long)100);
```

尽管 get()和 load()方法的作用类似，但是它们的意义却有些不同。get()方法表示从数据库中去获取实体对象，如果没有的话则返回 null；而 load()的意义在于把指定主键的实体对象加载到缓存中来，如果实体不存在的话，Hibernate 将抛出 ObjectNotFoundException 异常。

技巧：如果读者在获取实体对象的时候，不太确定该主键是否一定存在于数据库中，可以所有地方都使用 get()。

【答案】

总之对于 get()和 load()的根本区别，一句话，hibernate 对于 load()方法认为该数据在数据库中一定存在，可以放心的使用代理来延迟加载，如果在使用过程中发现了问题，只能抛异常了，而 get()则可以不存在。

面试题 142　如何映射一对一关系

对于关系数据库，可以比较容易的实现一对一的关系建模。那么 Java 的实体，又应该如何建立一对一的关系呢？其实也比较简单，就是两个实体之间互相关联即可，彼此之间有一个对方的引用。那么在 Hibernate 中，如何建立起这样的映射模型呢？本例在回答该问题的同时，详细地讲解一对一关系映射的原理和过程。

【出现频率】★★★★

【关键考点】

- 一对一关系的含义；
- <one-to-one>标签的使用方法。

【考题分析】

在现实生活中，一对一的关系是比较多的，例如，夫妻，一个丈夫只能有一个妻子，一个妻子也只能有一个丈夫。在数据库中，一般用这样的形式来表示一对一的关系：一张表中的一条记录在另一张表中有且只有一条记录与其对应（主外键，但包含外键的表其外键值也是主键），表格的结构可以这样来建立：

```
CREATE TABLE husband(
 id bigint primary key ,
 name varchar(20) not null,
 age int
);
CREATE TABLE wife(
 id bigint primary key ,
 name varchar(20) not null,
 age int,
 foreign key (id) references husband (id) on delete cascade
) ;
```

然后，在丈夫实体和妻子实体中各自包含一个对方的属性，示例代码如下：

```
public class Husband {                          //丈夫实体类
    private Long id;                            //主键
    private String name;                        //姓名
    private Integer age;                        //年龄
    private Wife wife;                          //妻子
    // setters 和 getters
    //...
}
public class Wife {                             //妻子实体类
    private Long id;                            //主键
    private String name;                        //姓名
    private Integer age;                        //年龄
    private Husband Husband;                    //丈夫
    // setters 和 getters
    //...
}
```

在完成映射配置文件的时候，需要在双方都加上<one-to-one>标签。对于外键的一方，主键的生成策略则采用外键（foreign）生成策略，以下为丈夫和妻子实体的映射配置示例：

```
<!--  丈夫实体的配置  -->
<class name="Husband" table="husband">
    <id name="id" column="id">
        <generator class="increment" />
    </id>
    <property name="name"></property>          <!--  姓名  -->
    <property name="age"></property>           <!--  年龄  -->
    <!--  一对一的配置  -->
    <one-to-one name="wife" cascade="all" class="Wife"></one-to-one>
</class>
<!--  妻子实体的配置  -->
<class name="Wife" table="wife">
    <id name="id" column="id">
        <generator class="foreign"><!--  依靠外键来作为主键的输入  -->
            <param name="property">husband</param>
        </generator>
```

```
    </id>
    <property name="name"></property>          <!--  姓名  -->
    <property name="age"></property>           <!--  年龄  -->
    <one-to-one name="husband" cascade="all" class="Husband">
    </one-to-one>
</class>
```

说明：一对一关系映射模型的建立，并非一定要按照以上的方式来建立，使用一对多的方式模拟一对一可以。

至此，丈夫和妻子的一对一映射关键就建立起来了。对于程序员来说，当需要从丈夫操作妻子或者从妻子操作丈夫的时候，可以通过 getWife()和 setWife()或 getHusband()和 setHusband()来进行，而不用管底层的 SQL 是如何操作的了。

【答案】

建立一对一关系映射的时候，需要保证数据库和实体之间都是一对一的关系统一。在数据库中，一般用一张表中的一条记录在另一张表中有且只有一条记录，与其对应的形式来表示一对一的关系。而在 Java 代码中，实体类需要加上对方的引用。

在映射配置文件中，需要在双方都加上<one-to-one>标签。对于外键的一方，主键的生成策略则采用外键（foreign）生成策略。

面试题 143　如何映射一对多关系

一对多是关系模型中使用最多的一种情况，它的实现也比较容易，数据库的外键就是一对多的关系。而 Java 实体类则可以通过集合类的形式来创建一对多的关系。那么在 Hibernate 中，如何建立起这样的映射模型呢？本例在回答该问题的同时，详细地讲解一对多关系映射的原理和过程。

【出现频率】★★★★

【关键考点】

- 一对多关系的含义；
- <one-to-many>标签的使用方法；
- <many-to-one>标签的使用方法。

【考题分析】

在实际开发中，一对多的关系是关系映射中出现得最多的一种，例如，商品类别和商品，一件商品只能属于一种商品类别，而商品类别下会包含多个商品。在数据库中，一般用这样的形式来表示一对多的关系：一张表中的一条记录在另一张表中有多条记录与其对应，反过来只能找到一条记录对应（典型的主外键关系），表格的结构可以这样来建立：

```
CREATE TABLE Catalog(                    --建立目录表
 id bigint primary key ,
 name varchar(20) not null
);
CREATE TABLE product(                    --建立产品表
 id bigint primary key ,
 name varchar(20) not null,
 price double,
```

```
 catalog_id bigint,
 foreign key (catalog_id) references Catalog (id)
) ;
```

在实体类的定义中，商品 Product 实体包含一个类别 Catalog 的属性，Catalog 类包含一个集合属性，该集合属性用于存储商品实体，示例代码如下：

说明：集合一般采用 java.util.Set、java.util.List 等类型。

```
public class Catalog {                          //商品类别实体类
    private Long id;                            //主键
    private String name;                        //类别名称
    private Set<Product> products;              //商品集合
    // setters 和 getters
    ...
}
public class Product {                          //商品实体类
    private Long id;                            //主键
    private String name;                        //姓名
    private Double price;                       //年龄
    private Catalog catalog;                    //所属类别
    // setters 和 getters
    ...
}
```

在完成映射配置文件的时候，需要在商品实体上加上<many-to-one>标签，表示对一个类别的引用。对于类别实体，则使用集合类（例如，<set>）标签，表示包含多个商品实体。以下为商品和商品类别实体的映射示例：

```
<!--  类别实体的配置  -->
<class name="Catalog" table="catalog">
    <id name="id">
        <generator class="increment" />      <!--  自动增长型的主键  -->
    </id>
    <property name="name"></property>        <!--  名称  -->
    <!--  集合配置  -->
    <set name="products" cascade="all" lazy="false" table="product" >
        <key column="catalog_id"></key>      <!--  外键  -->
        <one-to-many class="Product"/>       <!--  对应类型  -->
    </set>
</class>
<!--  商品实体的配置  -->
<class name="Product" table="product">
    <id name="id">
        <generator class="increment" />      <!--  自动增长型的主键  -->
    </id>
    <property name="name"></property>        <!--  名称  -->
    <property name="price"></property>       <!--  价格  -->
    <!--  多对一的配置  -->
    <many-to-one name="catalog" cascade="save-update" class=
    "Catalog"/>
</class>
```

说明：在数据量比较大的时候，拥有集合的一方可以使用延迟加载机制来提高程序的性能。集合标签（如<set>）包含一个 lazy 属性，它表示是否延迟加载引用的对象数据，如果需要则设置为 false，否则为 true。

至此，商品类别和商品的一对多映射关键就建立起来了。对于程序员来说，当需要从商品操作类别或者从类别操作商品的时候，可以通过 getCatalog()和 setCatalog()或 getProducts()和 setProducts()来进行，那些包含复杂表连接的 SQL，Hibernate 会帮助开发者自动执行。

【答案】

建立一对多关系映射的时候，需要保证数据库和实体之间都是一对多的关系统一。在数据库中，一般用一张表中的一条记录在另一张表中有多条记录，与其对应的形式来表示一对多的关系。而在 Java 代码中，一边实体类需要加上对方的引用，而另一边需要加上集合类型的引用。

在映射配置文件中，代表“多”的一方需要加上<many-to-one>标签，代表对对方的一个引用；而代表“一”的一方则需要加上集合标签和<one-to-many>标签，代表对对方的多个引用。

面试题 144　如何映射多对多关系

多对多关系也是一种比较普遍的关系模型，在数据库中使用中间表的形式来实现。Java 实体类则可以通过双向的集合类的形式来创建多对多的关系。那么在 Hibernate 中，如何建立起这样的映射模型呢？本例在回答该问题的同时，详细地讲解多对多关系映射的原理和过程。

【出现频率】★★★★

【关键考点】

- 多对多关系的含义；
- <many-to-many>标签的使用方法。

【考题分析】

多对多关系，相当于两个一对多的关系，例如，学生和老师。一个学生可以上多个老师的课，同样，一个老师也不会只有一个学生。在数据库中，一般用这样的形式来表示多对多的关系：一张表中的一条记录在另一张表中有多条记录与其对应，反过来也可以找到多条记录（3 张表，其中有一个中间关系表，关系表为复合主键，并且包含另外两个表的主外键），表格的结构可以这样来建立：

```
CREATE TABLE student(                    --建立学生表
 id bigint primary key ,
 name varchar(20) not null
);
CREATE TABLE teacher(                    --建立老师表
 id bigint primary key ,
 name varchar(20) not null
);
CREATE TABLE student_teacher(            --建立中间表
 stu_id bigint,
```

```
 tea_id bigint,
 primary key(stu_id,tea_id),
 foreign key (stu_id) references student (id),
 foreign key (tea_id) references teacher (id)
) ;
```

在实体类的定义中，学生实体包含一个集合属性，用于保存对应的老师实体对象。同样，老师实体也包含一个存储学生实体对象的集合属性，示例代码如下：

```
public class Student {                          //学生实体类
    private Long id;                            //主键
    private String name;                        //姓名
    private Set<Teacher> teachers;              //老师集合
    // setters 和 getters
    ...
}
public class Teacher {                          //老师实体类
    private Long id;                            //主键
    private String name;                        //姓名
    private Set<Student> students;              //学生集合
    // setters 和 getters
    ...
}
```

在完成映射配置文件的时候，需要在双方都加上<set>标签，并且在<set>标签中使用<many-to-many>标签，表示多对多的关系，以下为学生和老师实体的映射示例。

注意：多对多关系的某一边，需要为<set>加上 inverse=true 的属性，由某一边负责维护外键就可以了，否则可能出错。

```
<!--  学生实体的配置  -->
<class name="Student" table="student">
    <id name="id">
        <generator class="increment" />     <!--  自动增长  -->
    </id>
    <property name="name"></property>       <!--  名称  -->
    <!--  集合配置  -->
    <set name="teachers" cascade="all" lazy="false" table="stu_
    teacher" inverse="false">
        <key column="stu_id"></key>
        <many-to-many class="Teacher" column="tea_id"/>
    </set>
</class>
<!--  老师实体的配置  -->
<class name="Teacher" table="teacher">
    <id name="id">
        <generator class="increment" />     <!--  自动增长  -->
    </id>
    <property name="name"></property>       <!--  名称  -->
    <!--  集合配置  -->
    <set name="students" cascade="all" lazy="false" table="stu_
    teacher" inverse="true">
        <key column="tea_id"></key>
        <many-to-many class="Student" column="stu_id"/>
    </set>
</class>
```

至此，学生和老师的多对多映射关键就建立起来了。对于程序员来说，当需要从学生操作老师或者从老师操作学生的时候，可以通过 getTeachers()和 setTeachers()或 getStudents()和 setStudents()来进行，它们的中间表及其复杂的外键关系，Hibernate 会自动执行 SQL 予以维护。

【答案】

建立多对多关系映射的时候，需要保证数据库和实体之间都是多对多的关系统一。在数据库中，一般用一张表中的一条记录在另一张表中有多条记录与其对应，反过来也可以找到多条记录形式来表示多对多的关系，也就是用第 3 张表来代表它们之间的关系。而在 Java 代码中，两边都需要加上集合类型的引用。

在映射配置文件中，双方都使用集合类标签和<many-to-many>标签来表示多对多的关系，需要指定中间表的名字“table”、由谁来维护它们之间的关系（inverse）、外键的名字（key）等信息。

面试题 145　继承关系的映射策略有哪些

继承关系，是面向对象思想特有的关系，而数据库却不存在继承关系。如果需要把 Java 程序里的继承关系映射到关系型数据库，就需要用主外键的关系模拟这种继承关系。一般情况下，用 Hibernate 建立继承关系的映射，可以有两种解决方案，它们各有优缺点，应该根据不同的情况，而采用不同的映射策略。本例在回答该问题的同时，全面地讲解继承映射的方案及其优劣。

【出现频率】★★★★

【关键考点】

- 继承映射的含义和原理；
- 单表实现继承映射的思想；
- 每个子类一张字表的实现思想。

【考题分析】

在 Java 程序中，继承关系的实体是很多的，例如，银行卡包括借记卡和信用卡两种，借记卡和信用卡都继承自银行卡。在数据库中，如果用一张表来存储所有的银行卡，则需要有一个单独的字段（如 type）来标识当前行的类型，也就是银行卡的类型。例如，所有的银行都有用户姓名、余额属性，信用卡有最大信用透支额度属性。在实体类的定义中，首先定义一个总的银行卡 Card 类，包含主键、用户姓名和余额属性，然后再定义信用卡类（CreditCard）和借记卡类（SavingCard），它们分别拥有最大透支额度 maxCredit 属性和最小余额 minBalance 属性，示例代码如下：

```
public class Card {                          //银行卡实体类
    protected Long id;                       //主键
    protected String name;                   //用户姓名
    protected Double balance;                //余额
    // setters 和 getters
    ...
}
public class CreditCard extends Card {       //继承自 Card 的信用卡实体类
```

```
    private Double maxCredit;                    //最大透支额度
    // setters 和 getters
    ...
}
public class SavingCard extends Card {           //继承自 Card 的借记卡实体类
    private Double minBalance;                   //最小余额
    // setters 和 getters
    ...
}
```

单表实现继承映射的时候，只需要定义一个映射配置文件即可。这里，用 Card.hbm.xml 文件进行配置，用<discriminator>标签指定用于区别类型的字段，并且为每一个子类添加一个<subclass>标签来指定它们的类名及其特有的属性。以下为银行卡、信用卡和借记卡的继承映射配置示例：

```
<!--  银行卡及其他的子类的实体配置  -->
<class name="Card" table="card">
    <id name="id" column="id">
        <generator class="increment"/>
    </id>
    <!--  指定类型区别字段名及其数据类型  -->
    <discriminator column="type" type="string"></discriminator>
    <property name="name"></property>
    <property name="balance"></property>
    <!--  为每一个子类添加一个<subclass>标签来指定它们的类名及其特有的属性-->
    <subclass name="CreditCard" discriminator-value="credit" >
        <property name="maxCredit"></property>      <!--  子类属性  -->
    </subclass>
    <subclass name="SavingCard" discriminator-value="saving" >
        <property name="minBalance"></property>     <!--  子类属性  -->
    </subclass>
</class>
```

至此，采用单表实现的继承映射就建立起来了。它们之间的继承与被继承的关系，由 Hibernate 负责维护，大多数的 SQL 语句都会在 WHERE 语句后面加上 type=xxx 之类的条件限制。

单表实现继承映射的优点在于：没有任何的表连接，查询的速度会相对较快，多态查询效率高。但是它把所有的数据都存在一张表中，有可能会造成表特别大的后果，不利于后期的维护，而且，如果子类的属性很多，可能会有很多的 NULL 值。所以，单表实现的继承映射更适合数据量小、属性个数少，或者多态查询时候比较多的情况。

说明： 多态查询指的是，查询所有继承了某一个类的所有实体。有一点抽象查询的意思，例如，现在有一张不知道是什么类型的卡，但是它肯定是一种银行卡，它肯定会有用户姓名、余额等属性，程序就可以把这张卡看做是一张银行卡来进行相关的查询。

如果是每个子类一张表，在数据库中，用一张表来存储所有银行卡共有的信息，如用户姓名、余额等。每种具体类型的卡单独用一张表来存储，并且它的主键是引用主表的主键。数据库的建表 SQL 可以这样写：

```
CREATE TABLE card (                                    --建立卡表
  id bigint(20) NOT NULL,
  name varchar(255) ,
  balance double ,
  PRIMARY KEY(id)
);
CREATE TABLE credit_card (                             --建立信用卡表
  card_id bigint(20) NOT NULL,
  maxCredit double ,
  PRIMARY KEY  (card_id),
  FOREIGN KEY (card_id) REFERENCES card (id)
)
CREATE TABLE saving_card (                             --建立储蓄卡表
  card_id bigint(20) NOT NULL,
  minBalance double ,
  PRIMARY KEY  (card_id),
  FOREIGN KEY (card_id) REFERENCES card (id)
)
```

至于映射配置文件，也只需要一个 Card.hbm.xml 文件即可，用<joined-subclass>标签定义一个子类及其他的表名，在<joined-subclass>标签里需要包含一个<key>标签，它是用来指定分表的主键名的。以下为银行卡、信用卡和借记卡的多表实现的继承映射配置示例：

```
<!--  银行卡及其他的子类的实体配置  -->
<class name="Card" table="card">
    <id name="id" column="id">
        <generator class="increment"/>                 <!--  自动增长  -->
    </id>
    <property name="name"></property>                  <!--  名称  -->
    <property name="balance"></property>               <!--  余额  -->
    <joined-subclass name="CreditCard" table="credit_card">
        <key column="card_id"></key>                   <!--  外键  -->
        <property name="maxCredit"></property>         <!--  最大信用额度-->
    </joined-subclass>
    <joined-subclass name="SavingCard" table="saving_card">
        <key column="card_id"></key>                   <!--  外键  -->
        <property name="minBalance"></property>        <!--  最小余额  -->
    </joined-subclass>
</class>
```

至此，采用多表实现的继承映射就建立起来了。它们之间的继承与被继承的关系，由 Hibernate 负责维护，大多数的 SQL 语句都会加上左连接（left join）或右连接（right join）的语句，那是把各个子类的数据联合起来。

每个子类一张表的实现方式，其优点在于数据存储得比较紧凑，当查询某个子类的数据时速度比较快。但是，由于可能会有很多的表连接，不太适合多态查询，更适合侧重于操作某一类数据（例如，大多数时候查询的是信用卡的数据）的情况。所以，多表实现的继承映射比较适合数据量大、子类属性个数多，或者经常查询某一子类的情况。

【答案】

继承关系的映射策略一般有两种形式：所有的类都在一张表中和每个子类一张表（具体的实现方式请参加以上的分析）。

单表策略的优点在于无需表连接，查询速度快，适合多态查询。缺点则是可能会造成表太大的后果，不利于维护。

每个子类一张表策略的优点在于数据存储得比较紧凑，当查询某个子类的数据时速度比较快。缺点则是可能会有很多的表连接，不太适合多态查询。

13.3　Spring

Spring 是一个轻量级的 Java EE 容器，它也是一种从实际需求出发，着眼于轻便，灵巧，易于开发，易测试和易部署的轻量级开发框架。Spring 在一定程度上影响了 Java EE 的开发模式。本节将集中讨论有关 Spring 的常见面试题。

面试题 146　依赖注入的方式有哪些

依赖注入是 Spring 的 ICO 容器中的一个核心概念，它本质上是通过容器来管理对象数据的填充和对象之间的关联关系。例如，A 需要填充某种类型的数据，A 需要关联 B 等。那么，Spring 支持哪些方式来注入这些关系呢？本例在回答该问题的同时，详细地讲解 Spring 的依赖注入的含义和方式。

【出现频率】★★★★

【关键考点】

- 依赖注入的含义；
- 依赖注入的方式及其区别。

【考题分析】

依赖注入，指的是组件之间的依赖关系由容器在运行时决定。控制权由对象本身转向容器；由容器根据配置文件去创建实例，并创建各个实例之间的依赖关系。它的设计思想在于将业务代码与程序本身的结构代码分开。容器尽量不要侵入到应用程序中去，应用程序本身可以依赖于抽象的接口，容器根据这些接口所需要的资源注入到应用程序中，也就是说应用程序不会主动向容器请求资源；容器会自动把这些对象给应用程序。

IOC 容器的核心是 Spring 提供的 Bean 工厂（BeanFactory）。在 Spring 中，Bean 工厂创建的各个实例称作 Bean。Bean 工厂负责读取 Bean 定义文件，管理对象的加载、生成，维护 Bean 对象与其他 Bean 对象的依赖关系，负责 Bean 的生命周期。对于简单的应用程序来说，使用 BeanFactory 就已经足够来管理 Bean 了，在对象的管理上就可以获得许多的便利，而很少看到 new 语句。

以下是创建一个用户服务 Bean 的配置示例：

```
<bean id="userBean" class="UserBean"></bean>
```

id 是 Bean 的唯一标示性属性，一个 Spring 容器中只能有一个该 id 为指定值的 Bean，其他 Bean 如果需要依赖注入该 Bean，也需要使用 id 属性。

注意：默认情况下，BeanFactory 生产的 Bean 实例采用的是单例模式，也就是说一个 IOC 容器产生的 A 对象只有一个。

在以上的配置示例中，并没有任何数据或其他对象的注入。Spring 的 Bean 可以包含各

类属性，这些属性影响着 Bean 的行为。例如，String 类型的配置属性、其他 Bean 的引用、列表、集合等。Bean 工厂在创建 Bean 的时候，根据配置文件的设置将这些属性都装配好。Spring 的依赖注入的两种方式：setter 方法注入和构造方法注入。

1．通过setter方法注入

Bean 需要按照 JavaBean 的可写属性的 setter 方法的规范来定义。Spring 会利用反射机制，在 Bean 构建好以后，调用 setter 方法，把属性的值设置给 Bean 实例。例如，以下是一个用户服务 Bean，它包含一个名为 str 的可写属性，具体如下：

```
public class UserBean {                          //定义一个普通的 Java 类
    private String str;                          //str 成员变量
    public void setStr(String str) {             //str 的 setter()方法
        this.str = str;
    }
    ...
}
```

然后，就可以在配置文件中为 str 属性赋值，示例如下：

```
<bean id="userBean" class="UserBean">
    <!-- 为 str 设置一个字符串值 -->
    <property name="str" value="hello user bean"></property>
</bean>
```

以上代码就完成了 UserBean 的一个字符串类型的属性 str 的装配，其中 name 属性的值应该与 JavaBean 规范规定的属性名相同，然后用 value 属性设置它的值为“hello user Bean”。除了字符串类型，整型、布尔、浮点等，基础类型的属性也是可以通过配置来完成装配，甚至链表、集合这样的 Java 容器类型也是可以通过配置的方式来装配，以下是 List 类型的属性装配配置示例：

```
<!-- 链表 List 属性配置示例 -->
<bean id="testBean" class="TestBean">
    <property name="listProperty">
        <list>                              <!-- list 类型的数据 -->
            <value>1</value>
            <value>2</value>
            <value>3</value>
        </list>
    </property>
</bean>
```

以上的属性设置，都是用 value 属性或 value 标签装配固定的值，如果属性引用的是另外的 Bean 呢？那么就得使用 ref 或 bean 属性了。例如，以下是通过 setter 为 UserBean 设置 UserDaoBean 的配置示例：

```
<!-- 配置 userBean，并配置它依赖的 UserDao -->
<bean id="userBean" class="UserBean">
    <property name="userDao" ref="userDao"></property>
</bean>
<!-- 配置 UserDao -->
<bean id="userDao" class="UserDao">
    <!-- userDao 的属性配置... -->
</bean>
```

说明：但凡使用<property>标签来装配 Bean 的属性，必须在 Bean 类中加上属性的 setter 方法。

2. 通过构造方法注入

与 setter 方法注入不同，构造方法注入是在创建 Bean 实例的时候，通过调用 Bean 的构造方法来注入属性的值。它使用的是<constructor-arg>标签，也可以通过<value>标签或<ref>标签指定属性的值或者引用的 Bean 实例，示例如下：

```
public class ConTestBean {
    private String str;                                    //str 成员变量
    private UserBean userBean;                             //UserBean 引用
    public ConTestBean(String str, UserBean userBean) {
                                                           //包含两个参数的构造方法
        this.str = str;
        this.userBean = userBean;
    }
    ...
}
```

通过构造方法注入 ConTestBean 需要的属性代码如下：

```
<!--  配置 ConTestBean，采用构造方法注入属性  -->
<bean id="conTestBean" class="com.throne212.javaweb.ch17.ConTest-
Bean">
    <!--  需要指定构造方法参数的下标位置，从 0 开始  -->
    <constructor-arg index="0">
        <value>hello construct bean</value>
    </constructor-arg>
    <constructor-arg index="1">
        <ref bean="userBean"/>  <!--  userBean 为另外一个 Bean 实例  -->
    </constructor-arg>
</bean>
```

说明：一般情况下，使用 setter 注入的时候多一些。但是实际开发中，应该根据实际的情况选择不同的注入方式，以下是两种注入方式的区别：

- Constructor：可以在构建对象的同时，把依赖关系也构建好。对象创建好后就已经准备好了所有的资源，安全性要高一些。
- Setter：创建完对象之后再通过 set()方法进行属性的设定，更加灵活一些。

【答案】

Spring 的依赖注入可以有两种方式来完成：setter 方法注入和构造方法注入。下面对这两种方法作详细介绍。

（1）setter 方法注入，使用<property>标签进行配置，为符合 JavaBean 规范的可写属性赋值，值的类型可以是 String、int 等数据类型，也可以是任意引用类型。

（2）构造方法注入，使用<constructor-arg>标签进行配置，根据下标位置来为变量赋值，类型也是可以任意的。

面试题 147　如何使用 Spring 的声明式事务

大家知道，事务一般是在代码中使用的。但是 Spring 却把事务作为了一种中间件的形式提供给开发者，开发者可以通过配置的形式来使用事务，也叫声明式事务。本例在回答该问题的同时，详细地讲解声明式事务的原理和使用方式。

【出现频率】★★★★

【关键考点】

- AOP 思想；
- Spring 声明式事务的使用方式。

【考题分析】

AOP 是一种对 OOP 有益补充的编程技术，它可以解决 OOP 和过程化方法不能够很好解决的横切（crosscut）问题，例如，事务、安全、日志等。随着软件系统变得越来越复杂，横切关注点成为一个大问题，AOP 可以很轻松的解决横切关注点这个问题。Spring 框架对 AOP 提供了很好的支持。

简单来说，AOP 就是一种功能比较复杂的拦截器。在代码真正达到目标以前，AOP 可以对其进行拦截，提供一些通用的中间件的服务，例如，加上事务服务、记录日志等。Spring 的声明式事务也就是基于 AOP 而实现的。

声明式事务，可以最少程度的影响应用程序的代码。Spring 的声明式事务为普通 Java 类封装事务控制，底层是用动态代理技术实现的。动态代理的一个重要特征是针对接口，所以 DAO 层对象要通过动态代理来让 Spring 接管事务，就必须在 DAO 层对象前面抽象出一个接口。

说明：如果没有接口，那么 Spring 会使用 CGLIB 来解决，但是不推荐。

不像 EJB 的 CMT 绑定在 JTA 上，Spring 声明式事务管理可以在任何环境下使用。只需更改配置文件，它就可以和 JDBC、JDO、Hibernate 或其他的事务机制一起工作。而且，Spring 可以使声明式事务管理应用到普通 Java 对象。与 EJB 类似，Spring 的包含了多种事务规则，以适应不同的事务需求。

声明式事务有一个显著的优点，业务对象不需要依赖事务基础设施，也就是看不到任何的 JDBC 关于事务的代码，通常也不需要引入任何 Spring API。另外，回滚规则的概念是很重要的，它们使得开发者可以指定哪些异常应该发起自动回滚。程序员在配置文件中，而不是 Java 代码中，以声明的方式指定。例如，可以配置只有当抛出了 MyApplicationException 才回滚事务。

通常通过 TransactionProxyFactoryBean 设置 Spring 事务代理，目标对象包装在事务代理中，这个目标对象可以是一个普通的 Bean 对象。当定义 TransactionProxyFactoryBean 时，必须提供一个相关的 TransactionManager 的引用和事务属性，配置示例如下：

```
<!-- 用 TransactionProxyFactoryBean 来代理目标 bean -->
<bean id="testBean"
class="org.springframework.transaction.interceptor.TransactionProxyFact
oryBean">
```

```
    <property name="transactionManager">
        <ref bean="transactionManager" />  <!--  事务管理器的引用  -->
    </property>
    <property name="target">
        <ref bean="testBeanTarget" />       <!--  目标  -->
    </property>
    <property name="transactionAttributes">     <!--  事务属性配置  -->
        <props>
            <prop key="insert*">
                PROPAGATION_REQUIRED,-MyCheckedException
            </prop>
            <prop key="update*">PROPAGATION_REQUIRED</prop>
        </props>
    </property>
</bean>
```

事务管理 TransactionManager 是 Spring 声明式事务需要使用的 Bean，Spring 框架本身为开发人员提供了 JDBC、Hibernate 等常用的事务管理器。在配置事务管理器 Bean 的时候，往往需要指定一个数据源，例如，对于 Hibernate 的事务管理器可以这样来配置具体如下：

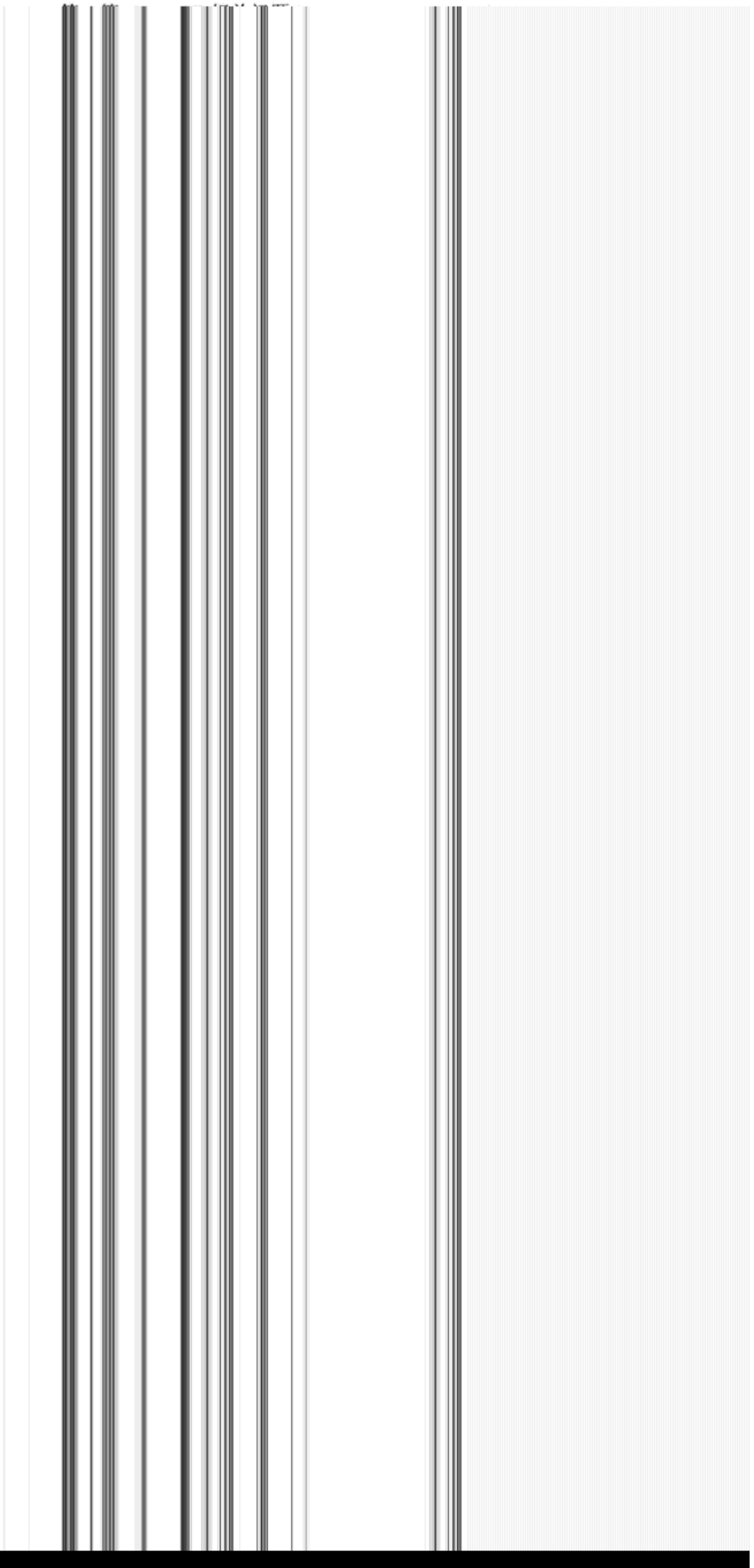
```
<!-- session factory -->
<bean id="sessionFactory"
class="org.springframework.orm.hibernate3.LocalSession
FactoryBean">
    <property name="configLocation"
        value="classpath:hibernate.cfg.xml">
                                    <!--  指定 hibernate 配置文件  -->
    </property>
</bean>
<!-- transaction manager -->
<bean id="transactionManager"
class="org.springframework.orm.hibernate3.Hibernate
TransactionManage   r">
    <property name="sessionFactory">
        <ref bean="sessionFactory" />  <!--  引用 sessionFactory  -->
    </property>
</bean>
```

事务属性（transactionAttributes）是用来规定方法使用事务的方式，包括 REQUIRED、NEVER、SUPPORTS 等，表 13.2 列出了几类常用的 Spring 支持的事务属性及其相关说明。同时，方法的匹配可以采用通配符的方式来匹配多个方法，一般来说，数据的增、删、改是需要事物的，而查询可以不用事务。

注意：上例配置中的“insert*”的映射的值包括回滚规则。添加的-MyCheckedException 指定如果方法抛出 MyCheckedException 或它的子类，事务将会自动回滚。可以用逗号分隔定义多个回滚规则。减号“-”前缀代表强制回滚，加号“+”前缀则指定提交。

表 13.2　Spring声明式事务属性

属　性	说　明
PROPAGATION_REQUIRED	要求必须使用事务，如果已经存在事务则使用已有的事务，如果没有事务则开启一个新事务



在完成 DAO 层的时候，一般需要继承 Spring 提供的 HiberanteDaoSupport 类，它需要注入一个 SessionFctory 的引用，这就是以上配置的 SessionFctory 对象。这样，DAO 层就可以比较容易的创建 Session，进行数据的相关操作。

Struts 在处理请求的时候，需要创建 Action 对象。这些 Action 往往需要引用业务层的对象，而这些业务层的对象却在 Spring 容器中。那么，Action 的创建也需要交给 Spring 来完成。开发者需要做的就是，修改 Struts 默认的控制器，而使用 Spring 提供的控制器，在 struts-config.xml 中，进行如下修改：

```
<!--  控制器配置 -->
<controller nocache="true">
    <set-property property="processorClass"
    value="org.springframework.web.struts.DelegatingRequestProcessor" />
</controller>
```

然后，在再 Spring 的配置文件中，创建和装配 Action 对象。只不过，此时 bean 的 name 需要相同于 Action 在 struts-config.xml 中配置的 URL 属性的值，例如下面的代码：

```
<!--  struts-config -->
        <action path="/login"
            type="com.throne212.javaweb.ch17.struts.action.LoginAction"
            name="userForm">
            <forward name="login" path="/login.jsp"></forward>
            <forward name="succ" path="/succ.jsp"></forward>
        </action>
<!--以下为 Spring 的 bean 配置 -->
<!--  applicationContext.xml -->
    <!-- 表现层 -->
    <bean name="/login" class="LoginAction">
        <property name="userService" ref="userService"></property>
    </bean>
```

说明：因为 name 的值是以斜线“/”开头的，因此不能用 id，而只能用 name。

Spring 与 Struts、Hibernate 整合在一起以后，接下来要做的就是在 Web 应用程序启动的时候，也启动 Spring 容器。Spring 提供了一个监听器的实现，是专门用于在 Web 应用程序中启动 IOC 容器的，并且这个 ApplicationContext 的对象是保存在 application 范围中的。在 web.xml 中可以这样来配置，具体如下：

```
<!-- Spring 需要的参数 -->
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:applicationContext.xml</param-value>
</context-param>
<!-- Spring 需要的监听器 -->
<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListene
r</listener-class>
</listener>
```

最后，把 3 个框架需要的 jar 文件，都保存在“WEB-INF/lib”文件夹中。至此，一个 SSH 组合的 Web 应用程序就搭建好了。当一次请求来到以后，首先是 Action 负责接收请求，并把请求的数据包装一下，然后转递给业务层的对象，业务层处理过程中，往往会调

用 DAO 的接口方法，以完成一些数据的增、删、查、改的操作。

注意：在存放 jar 文件的时候，还需要注意 jar 文件之间的冲突问题。有可能两个框架使用同一种第三方的 jar 文件，但是它们版本不一致就可能会产生一些问题。

【答案】

整合 SSH 的关键点在于如何把它们 3 者进行完美的衔接，每个衔接点 Spring 都提供了相应的支持类。整合它们 3 者有以下步骤：

（1）完成 Struts 的配置，包括 web.xml 和 struts-config.xml。

（2）完成 Hibernate 配置文件 hibernate.cfg.xml。

（3）在 web.xml 为 Spring 配置应用程序上下文监听器。

（4）用 Spring 提供的 controller 替换 Struts 原有的 controller。（在 struts-config.xml 中配置）

（5）用 Spring 提供的 SessionFactoryBean 来读取 hibernate.cfg.xml，完成 Spring 对 Hibernate 的整合。

（6）保存必须的 jar 文件到“WEB-INF/lib”文件夹中。

13.4 小　结

本章讲解了一些关于 Struts、Spring 和 Hibernate 框架的面试题。首先讲解了 Web 开发领域中经典的 MVC 开发模式和 Struts 框架，然后是 ORM 思想和 Hibernate 框架，最后是关于 Spring 的知识。它们 3 者都属于 Java EE 开发的事实主流，广泛的应用于各个领域的 Java 程序的开发，是面试的重点。

使用框架进行程序开发的意义在于：框架本身包含了各种思想，这些思想是前人为开发人员总结的，它们往往具有开发效率高、易维护、使程序开发更简单等特点，所以思想才是学习框架的根本。使用 Struts 的根本意义在于它可以带领开发者采用经典的 MVC 模式；Hibernate 则是 ORM 思想。Spring 则包含了 IOC 和 AOP 等思想。因此，读者在学习这些框架的时候，更多的应该学习它们的思想及其作用，并付诸于实践，才能应对多变而丰富的面试题。

第 14 章　EJB 与 JPA 相关问题

当 J2EE 发展到第 5 个版本，也就是 Java EE 5.0 的时候，EJB 的调整是重大的，新版本的 EJB 称为 EJB 3.0。开发 EJB 程序不再困难，它使用 Java 5.0 标准注释（Annotation）来替代 EJB 纷繁的 XML 注释，并且去掉了一些不必要的接口侵入，让 EJB 的 Bean 可以是普通 Java 类，得到众多开发者的支持。另外，实体 Bean 不再得到支持，取而代之的是 JPA，它是一组关于 ORM 的规范，它延续了 SUN 公司一贯的作风，定义规范接口，由 Hibernate、iBATIS 等具体的框架来充当实现，它也是得到了广发开发者的支持。EJB 是 JPA 使用最多的地方之一。本章将包含关于 EJB 3.0 和 JPA 的一些常见面试题，并且分析这些题目和知识点，帮助读者梳理这些方面的知识。

14.1　EJB 3.0

一提到 EJB，大家一定要注意，对方指的是 EJB 1.x 或 EJB 2.x，还是 EJB 3.0，它们之间的区别是非常大的。EJB 3.0 对老的 EJB 规范做出了重大的修改，也因此 EJB 才逐渐受到了欢迎，目前大多数的 EJB 程序都已经运行在 EJB 3.0 的容器了。所以，本章讲解的关于 EJB 的面试题仅限于 3.0 及其以上版本。

面试题 149　EJB 的类型有哪几种

EJB 一直以来都作为一个很神秘的事物而存在，其实如果把它的名字解析一下来看，它并不那么可怕。EJB 就是企业级 JavaBean，和普通的 JavaBean 相比，EJB 必须要存活在 Java EE 容器中，另外还有就是一些配置和使用规则而已。那么 EJB 有哪些种类呢？本例在回答该问题的同时，详细地讲解 EJB 的概念和各种 EJB 的区别。

【出现频率】 ★★★★

【关键考点】

- EJB 的概念和结构组成；
- 各种 EJB 的区别。

【考题分析】

EJB 是 SUN 公司推出的服务器端组件模型，它是 Java EE 服务器端组件开发的标准规范，最大的用处是开发和部署分布式应用程序。凭借 Java 跨平台的优势，用 EJB 技术部署的分布式系统可以不限于特定的平台。

EJB 是在一定背景下诞生的。企业级程序的开发经过若干年的发展以后，除了特定的业务以外，一些通用的功能逐渐的应用开来，例如，日志及其审核、集群和负载均衡、资

源池、消息服务等。这些通用的功能，如果每次企业程序开发的时候都重新开发的话，费时费力，而且还可能效率不高。那么，能不能把这些通用的功能给单独出来，通过第三方的形式提供呢，这样开发者就可以将大多数的经历集中在业务逻辑上。这些通用的被独立出来的组件就称为中间件。EJB 组件和容器正是基于这样的一种思想而设计。

EJB 程序的组成有两部分：组件和容器。组件是符合某种规范的一个或多个 Java 类和接口，它们的行为通过配置信息来控制，一般只负责具体的业务逻辑。容器则是组件赖以生存的温床，它也需要符合某种规范，并且它可以提供中间件的服务。EJB 最主要的作用就是简化企业级程序的开发，并符合一定的标准，可以向后兼容。

EJB 是分布式的。它的底层实现技术是 RMI（Remote Method Interface），是一种可以跨 JVM 调用方法的技术。EJB 与客户端程序以及 EJB 之间，是可以运行在不同的 JVM 甚至不同的服务器硬件中，它们可以自由的相互调用。

在 Java EE 中，Enterprise Java Beans（EJB）称为企业级 JavaBean，根据功能的不同而定义了 3 种不同的 Bean，分别是会话 Bean（Session Bean）、实体 Bean（Entity Bean）和消息驱动 Bean（MessageDriven Bean）下面对这些内容作详细介绍：

（1）会话 Bean（Session Bean）用于实现业务逻辑，它可以是有状态的，也可以是无状态的。每当客户端请求时，容器就会选择一个 Session Bean 来为客户端服务。会话 Bean 可以直接访问数据库，也可以通过 Entity Bean 实现数据访问。会话 Bean 分为无状态会话 Bean 和有状态会话 Bean 两种。

（2）实体 Bean（Entity Bean）是实体模型对象，用于实现 O/R 映射，负责将数据库中的表记录映射为内存中的 Entity 对象，事实上，创建一个 Entity Bean 对象相当于新建一条记录；删除一个 Entity Bean 会同时从数据库中删除对应记录；修改一个 Entity Bean 时，容器会自动将 Entity Bean 的状态和数据库同步。

说明：EJB 3.0 以后，实体 Bean 被 JPA 所取代。但是，在 Java EE 规范中依然保留着 EJB 2.0 关于实体 Bean 的定义。

（3）消息驱动 Bean（Message Driven Bean）是 EJB 2.0 中引入的新的企业 Bean，它基于 JMS 消息，只能接收客户端发送的 JMS 消息然后处理。MDB 实际上是一个异步的无状态 Session Bean，客户端调用 MDB 后无需等待，立刻返回，MDB 将异步处理客户请求。这适合于需要异步处理请求的场合，如订单处理，这样就能避免客户端长时间的等待一个方法调用直到返回结果。

【答案】

EJB 的全称为企业集 Java Bean（Enterprise Java Bean），根据用途的不同，可以分为 3 种 Bean：会话 Bean、实体 Bean 和消息驱动 Bean。

- 会话 Bean。用于实现业务逻辑，负责接收客户端请求；
- 实体 Bean。代表实体模型对象，用于实现对象关系映射；
- 消息驱动 Bean。用于异步编程，基于消息服务，相当于异步的无状态会话 Bean。

面试题 150　EJB 程序的开发思路和步骤是什么

EJB 程序是需要生存在符合标准规范的 EJB 容器中，那么 EJB 的开发过程必定是充满

了各种规范。包括如何编写业务接口、如何定义 Bean、如何完成描述文件、如何部署到容器等。本例在回答该问题的同时，详细地讲解 EJB 程序的基本开发思路和步骤。

【出现频率】 ★★★★

【关键考点】

- EJB 的概念和结构组成；
- EJB 程序的开发思路和需要遵守的各种规范。

【考题分析】

对于一个典型的 EJB 程序来说，它的基本开发思路和步骤可以分为以下几个方面。

1. 业务接口和业务Bean类

EJB 的业务接口分为本地接口和远程接口两种。本地接口主要针对容器内部的组件调用；而远程接口则是为其他的程序提供的外部接口。如何让容器知道什么是远程接口？什么是本地接口呢？通过标注 javax.ejb.Local 和 javax.ejb.Remote 来进行配置和区分。

注意：Java 5.0 以后推出了一种标准注释（Annotation），简称标注。它用于为类和方法提供一些注释信息，广泛的用于替代 XML 配置，示例如下：

```
/*本地接口示例*/
package ch14;
import javax.ejb.Local;
@Local                                          //本地接口标注
public interface HelloLocal {                   //Hello 本地接口
    public String sayHello(String name);        //接口方法
}
/*远程接口示例*/
package ch14;
import javax.ejb.Remote;
@Remote                                         //远程接口标注
public interface HelloRemote {                  //Hello 远程接口定义
    public String sayHello(String name);        //远程接口方法
}
```

通过以上的示例代码可以看出，EJB 的业务接口只是一个普通的 Java 接口（POJI），它暴露了客户端或内部组件可以调用的业务方法。然后，有了接口还不够，还需要提供这些接口的实现，也就是 Bean 类，一般来说是会话 Bean，例如下面的代码：

```
package ch14;
import javax.ejb.Local;
import javax.ejb.Remote;
import javax.ejb.Stateless;
@Stateless                                      //无状态会话 Bean 的标注
@Remote(HelloRemote.class)                      //指定远程接口
@Local(HelloLocal.class)                        //指定本地接口
public class HelloBean implements HelloRemote,HelloLocal{  //实现业务接口

    public String sayHello(String name) {       //实现方法
        return "hello, " + name;
    }
}
```

EJB 的业务 Bean 是一个普通的 Java 对象（POJO）即可，它一般会带有适合的标注，例如，Stateless、Stateful 等。它一般需要实现业务接口的方法，并指定它的本地接口和远程接口是谁。

2．编译业务接口和业务Bean类

EJB 程序的编写，一般会使用 javax.ejb 包下的一些类、接口和标注等，那么这些东西在哪里呢？EJB 属于 Java EE 的重要组成部分，因此在 SUN 公司发布的 Java EE SDK 中是可以找到的，这就是 javaee.jar 文件。把该 jar 文件放在 CLASSPATH 文件中，就可以完成编译了。

3．提供部署描述文件（可选）

根据 Java EE 的规范，如果还需要对 EJB 的某些行为进行配置，则还需要提供标准的配置文件 ejb-jar.xml，存放在 META-INF 目录下。在文件中，可以提供 EJB 更详细的信息，例如，描述、接口、事务类型以及其他 EJB 的服务。

说明：ejb-jar.xml 是标准的部署描述文件，某些特定的 Java EE 服务器还可能会规定一些自定义的配置文件，例如，galssfish 的 sun-ejb-jar.xml。

4．部署

这一步对于使用 IDE 的开发者就比较容易了，只需要通过 IDE 提供的工具就可以直接把 EJB 程序部署到 Java EE 服务器中。如果开发者没有 IDE，则需要把业务接口和业务 Bean 的 class 文件，以及部署描述文件进行打包成一个 jar 文件，再把这个 jar 文件存放在 Java EE 服务器指定的部署文件夹下。

Java EE 在扫描到该 jar 文件的时候，会自动的去解析部署描述文件、业务接口和业务类，为 EJB 程序创建 JNDI、连接池等资源，让 EJB 程序运行起来。

5．检查部署成功与否

一个 EJB 程序在部署的过程中，可能会遇到某些不可预见的错误而部署失败。那么如何知道该 EJB 程序是否已经成功的部署到 EJB 容器中了呢？最简单的办法就是去看 JNDI 的列表，因为 EJB 往往是需要为外部提供接口的。Java EE 服务器会把 EJB 的业务接口以 JNDI 的形式暴露给外部的程序，如果可以找到该 EJB 在 JNDI 中的路径，则证明它已经部署成功了。

6．写一个客户端程序测试

大多数调用 EJB 的客户端程序都是先通过 JNDI 来获取到业务接口，然后再使用的。根据 Java EE 服务器的不同，JNDI 的使用规则也不同。对于 glassfish，可以这样来写一个客户端的测试程序，具体如下：

```
package hello;
import java.util.Properties;
import javax.naming.Context;
```

```
import javax.naming.InitialContext;
public class Main {
    主方法
    public static void main(String[] args) throws Exception {
        Properties props = new Properties();
        props.setProperty("java.naming.factory.initial",
             "com.sun.enterprise.naming.SerialInitContextFactory");
        props.setProperty("java.naming.factory.url.pkgs"
             , "com.sun.enterprise.naming");
        props.setProperty("java.naming.factory.state",
             "com.sun.corba.ee.impl.presentation.rmi.JNDIStateFactor
             yImpl");
        props.setProperty("org.omg.CORBA.ORBInitialHost", "localhost");
                                                          //主机地址
        props.setProperty("org.omg.CORBA.ORBInitialPort", "3700");
                                                          //EJB 端口号
        Context ctx = new InitialContext(props);          //创建 jndi 上下文
        HelloBeanRemote hr =
              (HelloBeanRemote)ctx.lookup(HelloBeanRemote.class.getName());
        System.out.println(hr.getClass().getName());      //打印接口类名
        System.out.println(hr.sayHello("zbs"));           //调用测试
    }

}
```

说明：以上客户端示例代码只是针对 galssfish，而且还需要在 CLASSPATH 引入 galssfish 安装目录下的 lib/appserv-rt.jar 文件。如果是其他的 Java EE 服务器，例如 Jboss、Weblogic，请参加具体的说明文档。

【答案】

一个典型的 EJB 程序的基本开发思路和步骤如下：

（1）完成业务接口和业务 Bean 类。使用 Local、Remote、Stateless 等标注。

（2）编译。引入 javaee.jar 文件进行编译。

（3）提供部署描述文件 ejb-jar.xml。

（4）部署、打包和发布。

（5）检查部署成功与否。核对 JNDI 列表。

（6）写一个客户端程序进行检验。

面试题 151　无状态会话 Bean 的生命周期是怎样的

会话 Bean 是 EJB 使用最多的一种 Bean，它往往充当了业务逻辑与数据访问的核心，是开发者经常会打交道的 Bean。无状态会话 Bean 简单好用，使用得更是频繁。无状态会话 Bean 跟其他的 Bean 一样，也是需要运行在 EJB 容器中的，那么它的生命周期是怎样的呢？期间有哪几种事件呢？本例在回答该问题的同时，详细地讲解无状态会话 Bean 的概念和生命周期。

【出现频率】 ★★★★★

【关键考点】

- 无状态会话 Bean 的定义规范；

❑ 无状态会话 Bean 的生命周期及其相关事件。

【考题分析】

会话 Bean 一般用于建模业务逻辑，它的声明周期比较短暂，Bean 的存活时间往往取决与客户端的请求时间。会话 Bean 有可能在初始化（例如，容器启动）的时候并不存在，当客户端调用时，才实例化相应的会话 Bean。

无状态会话 Bean，一般只需要一次方法的调用就可以完成任务，无需维护客户端的状态，一个 Bean 实例还可能会被多个客户端所共享。

注意：EJB 中，所有的 Bean 都只能为单一线程服务，故一般不可以编写多线程的代码。

在编程方面，一般会使用以下 3 个标注：

（1）Stateless 标注：它用于表示某个 Bean 类是无状态回话 Bean。

（2）Remote 标注：它用于表示 Bean 的远程接口的定义或指定。

（3）Local 标注：它用于表示 Bean 的本地接口的定义或指定。

示例代码如下：

```
/*本地接口示例*/
package ch14;
import javax.ejb.Local;
import javax.ejb.Remote;
import javax.ejb.Stateless;
@Local                                          //本地接口标注
public interface HelloLocal {                   //Hello 本地接口
    public String sayHello(String name);        //接口方法
}
/*远程接口示例*/
@Remote                                         //远程接口标注
public interface HelloRemote {                  //Hello 远程接口定义
    public String sayHello(String name);        //远程接口方法
}
@Stateless                                      //无状态会话 Bean 的标注
@Remote(HelloRemote.class)                      //指定远程接口
@Local(HelloLocal.class)                        //指定本地接口
//Bean 类，实现了两种业务接口
public class HelloBean implements HelloRemote,HelloLocal{

    public String sayHello(String name) {       //实现方法
        return "hello, " + name;
    }
}
```

那么，无状态会话 Bean 部署到 EJB 容器以后，它是如何的被创建、提供服务和销毁的呢？其实，在无状态会话 Bean 的生命周期中，只存在两种状态，即不存在和准备就绪。在客户端请求来到之前，Bean 的实例是不存在；一旦有一个请求指向了这个无状态会话 Bean，EJB 容器会创建一个 Bean 的实体，或从对象池里取出一个，让它进入准备状态；然后，选择相应的方法进行调用来提供服务，如图 14.1 所示。

在生命周期里，有两个主要的方法：PostConstruct 和 PreDestroy，它们对应于 Bean 的就绪和销毁过程。PostConstruct 多用于为 Bean 的成员变量设置初始化值，而 PreDestroy 则用于回收一些资源，例如，关闭数据库连接。在定义它们两个方法的时候，需要使用

PostConstruct 和 PreDestroy 标注，告诉 EJB 容器应该调用哪个方法，把这两个方法定义在 Bean 类中就可以了，示例代码如下：

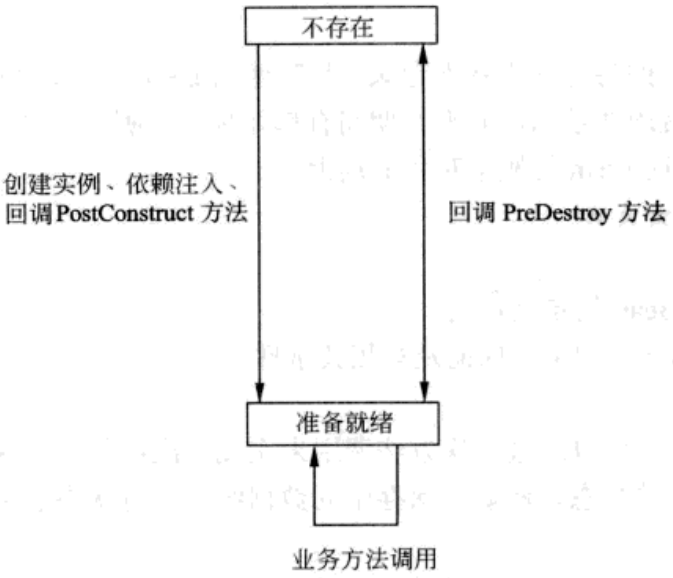


图 14.1　无状态会话 Bean 的生命周期

```
package ch14;
import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;
import javax.ejb.Stateless;
@Stateless
public class HelloBean implements HelloBeanRemote {     //无状态会话 Bean
    public String sayHello(String who){
        return "hello," + who;
    }
    @PostConstruct                                  //PostConstruct 标注
    public void doInit(){                           //PostConstruct 回调方法
     System.out.println("bean construct...");
    }
    @PreDestroy                                     //PreDestroy 标注
    public void doDestroy(){                        //PreDestroy()方法
     System.out.println("bean destroying...");
    }
}
```

说明： PostConstruct 和 PreDestroy 回调方法不必定义在 Bean 类中，也可以用单独的类来定义。

在该 Bean 从未被请求过的情况下，当一个新的请求来到的时候，示例代码的 doInit()方法被调用；当关闭 Java EE 服务器时，doDestroy()方法被调用。

【答案】

无状态会话 Bean 只需要一次方法的调用就可以完成任务，无需维护客户端的状态，多个客户端可共享同一个 Bean 实例。它的生命周期有两个状态：不存在状态和准备就绪状态。为了方便开发者控制 Bean 状态的变化，EJB 提供了两个回调方法：PostConstruct

和 PreDestroy，用于初始化操作和资源回收操作。

面试题 152 有状态会话 Bean 的生命周期是怎样的

有状态会话 Bean 的特点在于状态记录，与其他的 Bean 一样，也是需要运行在 EJB 容器中的。那么它的生命周期是怎样的呢？期间有哪几种事件呢？本例在回答该问题的同时，详细地讲解有状态会话 Bean 的概念和生命周期。

【出现频率】 ★★★★

【关键考点】

- 有状态会话 Bean 的定义规范；
- 有状态会话 Bean 的生命周期及其相关事件。

【考题分析】

有状态会话 Bean，可以通过多次方法调用来完成一次任务。在这多次调用的过程中，Bean 必须保存客户端的状态，例如，保存中间数据结果。有状态会话 Bean 不可以被多个客户端所共享。

与无状态会话 Bean 最主要的区别在于，有状态会话 Bean 会保持每一次连接的状态，直到此次连接结束。这些状态包括临时的变量、上下文引用、数据库事务等。而无状态会话 Bean 则不会保留上一次的状态。这与无状态会话 Bean 和有状态会话 Bean 的运行原理是相关的。

对于有状态会话 Bean 来说，只要有客户端发送对有状态会话 Bean 的访问，服务器都会创建一个会话 Bean 实例与该客户端对应，这个实例与这个客户端就是一一对应的。如果客户端在 Bean 实例中保存了信息，之后还可以使用。

对于无状态会话 Bean 来说，服务器端会维持一个实例池，创建好若干个实例对象供客户端调用。当从客户端发送创建会话 Bean 的请求时，并不一定会真地创建 EJB，多数情况下是从实例池中得到一个实例，用完之后重新放回实例池。如果下次再访问，再从实例池中取出一个实例使用，并不一定是上次的实例。即使两次访问使用的是同一个实例，在两次访问之间也有可能有其他的客户端访问了该实例。所以，并不能保证在多次访问之间的信息会被保存。所以，无状态会话 Bean 不会专门保存客户端的信息。

说明： 两种会话 Bean 各有好处。有状态会话 Bean 尽管可以保持会话状态，但是它因此而付出的代价也是比较高的。无状态会话 Bean 尽管不能保持会话状态，但是它的执行效高得多。因此，不是一定要保持客户端状态的时候，一般都建议使用无状态会话 Bean。

在编程方面，有状态会话 Bean 一般会使用以下 3 个标注：

（1）Stateful 标注：它用于表示某个 Bean 类是有状态回话 Bean。

（2）Remote 标注：它用于表示 Bean 的远程接口的定义或指定。

（3）Local 标注：它用于表示 Bean 的本地接口的定义或指定。

示例代码如下：

```
/*本地接口示例*/
package ch14;
```

```
import javax.ejb.Local;
import javax.ejb.Remote;
import javax.ejb.Stateless;
@Local                                              //本地接口标注
public interface HelloLocal {                       //Hello 本地接口
    public String sayHello(String name);            //接口方法
}
/*远程接口示例*/
@Remote                                             //远程接口标注
public interface HelloRemote {                      //Hello 远程接口定义
    public String sayHello(String name);            //远程接口方法
}
@Stateful                                           //有状态会话 Bean 的标注
@Remote(HelloRemote.class)                          //指定远程接口
@Local(HelloLocal.class)                            //指定本地接口
//Bean 类，实现了两种业务接口
public class HelloBean implements HelloRemote,HelloLocal{

    public String sayHello(String name) {           //实现方法
        return "hello, " + name;
    }
}
```

那么，有状态会话 Bean 部署到 EJB 容器以后，它是如何的被创建、提供服务、销毁、钝化和激活状态的呢？其实，在有状态会话 Bean 的生命周期中，一般有不存在、准备就绪和钝化状态。在客户端请求来到之前，Bean 的实例是不存在；一旦有一个请求指向了这个有状态会话 Bean，EJB 容器会创建一个新的 Bean 实例，让它进入准备状态；然后，选择相应的方法进行调用来提供服务；在会话被关闭之前，Bean 会钝化客户端的数据，进入钝化状态；当请求再次来临的时候，则把钝化的数据激活，继续为客户端提供服务，如图 14.2 所示。

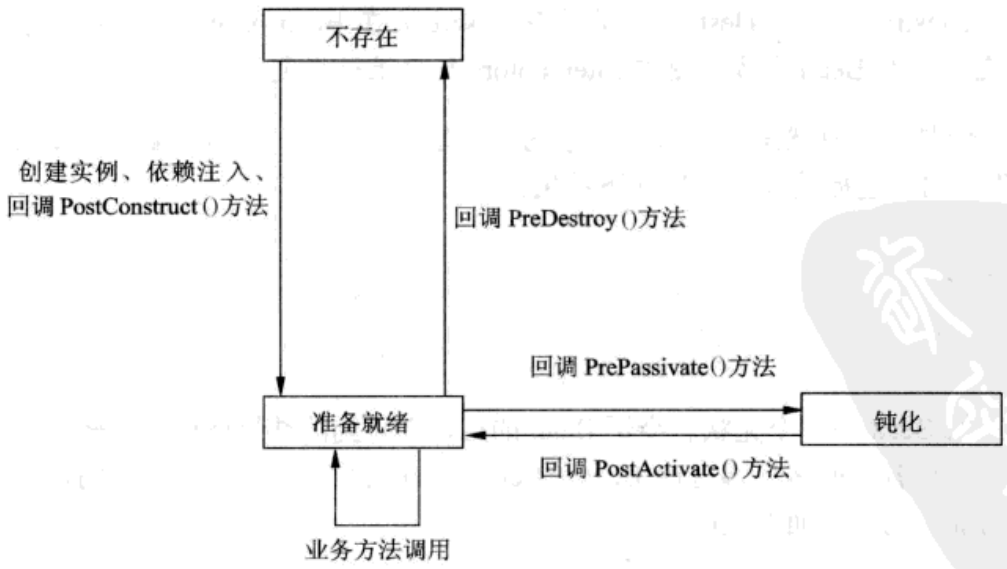


图 14.2　有状态会话 Bean 的生命周期

在生命周期中，除了 PostConstruct()和 PreDestroy()方法以外，无状态会话 Bean 还有

PrePassivate 和 PostActivate 事件方法，它们对应于 Bean 在钝化和激活的过程。PrePassivate 用于钝化之前的操作，PostActivate 则用于激活以后的操作。在定义它们两个方法的时候，需要使用 PrePassivate 和 PostActivate 标注，告诉 EJB 容器应该调用哪个方法，这两个方法定义在 Bean 类中就可以了，示例代码如下：

```
package ch14;
import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;
import javax.ejb.Stateless;
@Stateful
public class CountBean implements HelloBeanRemote {//无状态会话 Bean
    private int val;                                    //测试变量
     public int count(){                                //接口方法
         return ++val;                                  //返回结果
     }
    @PostConstruct                                      //PostConstruct 标注
    public void doInit(){                               //PostConstruct 回调方法
        System.out.println("bean construct...");
    }
    @PreDestroy                                         //PreDestroy 标注
    public void doDestroy(){                            //PreDestroy()方法
        System.out.println("bean destroying...");
    }
     @PrePassivate                                      //PrePassivate 标注
    public void prePassivate (){                        //PrePassivate()方法
       System.out.println("prePassivate()");
    }
    @PostActivate                                       //PostActivate 标注
    public void postActivate(){                         //PostActivate()方法
       System.out.println("postActivate()");
    }
}
```

说明：PostConstruct 和 PreDestroy 回调方法不必定义在 Bean 类中，也可以用单独的类来定义，在 Bean 上方，通过 Interceptors 标注进行指定。

以上的示例是一个有状态会话 Bean，它的作用是计数。当一个客户端对它发出 5 次的 count()方法调用时，它始终能打印出如下结果：

```
1
2
3
4
5
```

假若以上的 Bean 是一个无状态会话 Bean 的话，则得到的结果就可能比较混乱了。例如，如果同时开启若干的客户端对该 Bean 的 count()方法发出请求，则它们的数字结果会出现交叉的情况，例如下面的代码：

```
1
3
4
5
7
```

这就是因为无状态不能记录客户端的状态的原因。因此，有状态会话 Bean 最大的作用就在于，客户端需要发出多次调用请求才能完成一次任务的情况。

说明：如果有状态会话 Bean 中需要使用事务的话，则默认使用容器管理事务（CMT）的模式，所有的业务方法都会有事务支持。

【答案】

有状态会话 Bean 允许多次方法的调用来完成一次任务，完好的保护每次会话的客户端的状态，多个客户端可不能共享同一个 Bean 实例。它的生命周期有 3 个状态：不存在状态、准备就绪状态和钝化状态。为了方便开发者控制 Bean 状态的变化，EJB 提供了 4 个回调方法：PostConstruct()、PreDestroy()、PrePassivate()和 PostActivate()，用于初始化操作、状态恢复、资源回收等操作。

面试题 153　Servlet 如何调用 EJB

大家知道，Servlet 是生活在 Web 容器中的组件，EJB 是生活在 EJB 容器里的组件，但是它们都属于 Java EE 服务器。那么 Servlet 能不能调用 EJB 呢？调用的方式有哪些呢？本例在回答该问题的同时，详细地讲解 Servlet 调用 EJB 的方式和原理。

【出现频率】 ★★★★

【关键考点】

- Web 容器与 EJB 容器的关系；
- Servlet 调用 EJB 的方式。

【考题分析】

首先，可以把 Servlet 看成一个单独的 JVM 程序来看待，那么就有一个比较通用的使用 EJB 的方式就是使用 JNDI 获得远程接口。这跟一个普通的 Java 程序调用 EJB 没有什么区别，用一个配置属性集创建 JNDI 的初始化上下文，然后获取到 EJB 的远程接口，再调用它的某个方法，示例代码如下：

```
package ch14;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class TestServlet extends HttpServlet {      //Servlet类
    private HelloBeanRemote remote;                 //远程接口
    @Override
    public void init(ServletConfig config) throws ServletException {
        try {
            Properties props = new Properties();
```

```
            props.setProperty("java.naming.factory.initial",
                "com.sun.enterprise.naming.SerialInitContextFactory");
            props.setProperty("java.naming.factory.url.pkgs"
                , "com.sun.enterprise.naming");
            props.setProperty("java.naming.factory.state",
                "com.sun.corba.ee.impl.presentation.rmi.JNDIState
                FactoryImpl");
                props.setProperty("org.omg.CORBA.ORBInitialHost",
                "localhost");
                                                        //主机地址
            props.setProperty("org.omg.CORBA.ORBInitialPort", "3700");
                                                        //EJB 端口号
            Context ctx = new InitialContext(props);    //jndi 上下文
            //获取到以后类型转换
            remote = (HelloBeanRemote) ctx.lookup(HelloBeanRemote.class.
            getName());
        } catch (NamingException ex) {
            Logger.getLogger(TestServlet.class.getName()).log(Level.
            SEVERE, null, ex);
        }
    }
    public void doGet(HttpServletRequest request, HttpServletResponse
    response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            ...                                         //请求处理代码
            remote.sayHello(request.getParameter("who"));
        } finally {
            out.close();
        }
    }
}
```

那么，有没有一种更加简洁的方法呢？毕竟 Servlet 和 EJB 都生活在 Java EE 服务器中。答案是有。如果 Servlet 和 EJB 都生活在同一个 Java EE 的服务器中，则可以通过 Java EE 5.0 的依赖注入机制为 Servlet 注入一个 EJB 的引用，此时使用 javax.ejb.EJB 标注即可，例如下面的代码：

```
package ch14;

import java.io.IOException;
import javax.ejb.EJB;
import java.io.PrintWriter;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class TestServlet extends HttpServlet {      //Servlet 类
    @EJB                                            //引入 EJB 的标注
    private HelloBeanRemote remote;                 //远程接口
    public void doGet(HttpServletRequest request, HttpServletResponse
    response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
```

```
        try {
            ...                                          //请求处理代码
            remote.sayHello(request.getParameter("who"));
        } finally {
            out.close();
        }
    }
}
```

以上示例代码在使用 EJB 标注的时候，没有提供任何的关于 EJB 在 JNDI 上的路径和名称的信息。其实，容器是可以自动的找到恰当的 EJB 来注入的，若需要指定特定的 JNDI 名字则只需要为 EJB 标注添加一个 mappedName 的属性即可，例如下面的代码：

```
@EJB(mappedName="HelloBean")                    //引入 EJB 的标注
private HelloBeanRemote remote;                 //远程接口
```

说明：EJB 依赖注入的方式只被符合 Java EE 5.0 以上版本的 Java EE 服务器中才支持，例如，glassfish v2、glassfish v3、Jboss 4.2 等。

【答案】

Servlet 调用 EJB 有以下两种方式：

（1）使用 JNDI，编程式的找到远程接口来进行使用。该方式通用于所有的 Java 程序访问 EJB，较为笨拙一点。

（2）通过 javax.ejb.EJB 标注，为 Servlet 注入 EJB 引用。该方式只被 Java EE 5.0 以上版本的 Java EE 服务器所支持，且 Servlet 和 EJB 需要运行在同一个 Java EE 服务器中。

面试题 154　用 EJB 发布 Web 服务的基本思路是什么

Web 服务是近两年比较流行的一项技术，它是一套基于 Web、跨语言、跨平台的分布式网络通信技术。它使用 HTTP 或 SOAP 协议进行通信，数据格式为 XML。因为 Web 不限语言的，因此它可以很轻松的实现跨语言和跨平台。EJB 也是一项分布式开发技术，那么 EJB 可否把业务接口包装成一个 Web 服务呢？本例在回答该问题的同时，详细地讲解 Web 服务的原理和 EJB 发布 Web 服务的思路。

【出现频率】 ★★★★

【关键考点】

- Web 服务的含义和作用；
- Java 对 Web 服务的支持方式；
- EJB 发布 Web 服务的思路和步骤。

【考题分析】

Web 服务（Web Service）是一种用 WSDL、HTTP、SOAP 等技术，采用 XML 文档来交换数据的网络应用程序，用于两个分布式程序的通信。Web 服务不特定于某种计算机语言，也不特定于某个技术框架，是分布式开发的一种技术解决方案。

一般说来，一个 Web 服务的应用程序中会有 3 种角色：服务提供者、UDDI 注册中心和服务请求者。

（1）服务提供者。提供 Web 服务，把描述服务的 WSDL 文档通过 UDDI 注册中心或

其他形式发布，供服务请求者访问。

（2）UUDI。UUDI（Universal Description Discovery and Integration）注册中心，是一种标准的发布 Web 服务的注册中心。

（3）服务请求者。通过 SOAP 或 HTTP 协议请求 Web 服务，得到相应的结果。

它们 3 者的关系，如图 14.3 所示。

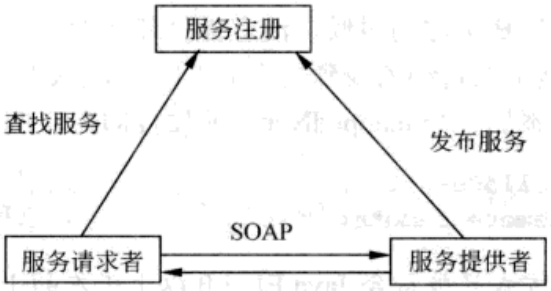


图 14.3　Web 服务的组成部分

一个典型的 Web 服务是按照如下所示的过程来运行的。

（1）服务实现者实现服务，并部署到服务器（Web 服务器或应用服务器）中，然后用 WSDL 文档对服务进行描述，并提供相应的 URL 进行下载，或将 WSDL 发布到 UDDI 注册中心去。

（2）服务请求者通过某种渠道获取到该 WSDL 文档以后，使用某种技术（如 JWS）生成本地代理，将请求绑定到 SOAP 上，发送给服务提供者，从而达到使用 Web 服务的目的。

（3）服务提供者接收到请求以后，执行接口方法，并把结果范围发送给请求者。

对于 Java 开发者来说，应该如何发布一个 Web 服务呢？使用 EJB 的无状态会话 Bean 就是一个不错的选择，Java EE 服务器提供了将 EJB 包装成 Web 服务的功能。主要需要使用 import javax.jws.WebMethod 和 import javax.jws.WebService 标注，分别标明 Web 服务方法和 Web 服务 Bean，示例代码如下：

```
/*
 * HelloWSBean.java
*/
package ws;
import javax.ejb.Stateless;
import javax.jws.WebMethod;
import javax.jws.WebService;
@Stateless                                          //无状态会话 Bean 标注
@WebService                                         //Web 服务标注
public class HelloWSBean implements HelloWSRemote {//Bean 类
    /** Creates a new instance of HelloWSBean */
    public HelloWSBean() {                          //构造方法
    }
    @WebMethod                                      //Web 服务方法
    public String sayHello(String user) {
        return "hello "+user;
    }

}
```

注意：默认情况下，被 WebService 标注修饰地类的公开方法都会成为 Web 服务方法。也就是说以上示例代码的 WebMethod 标注可以不写。

以上代码中，把一个 HelloWSBean 的无状态会话 Bean 包装成了一个 Web 服务，并把它的 sayHello()方法做出对外的 Web 服务方法。把这个 EJB 部署到 Java EE 服务器的时候，Java EE 服务器会自动的创建了一个 Web 服务，并提供了 WSDL 的下载，URL 为 http://localhost:8080/HelloWSBeanService/HelloWSBean?wsdl。那么如何使用这个服务呢？以下为一个访问该 Web 服务的客户端示例：

```
/*
 * Client.java
*/

package client;
import java.net.MalformedURLException;
import java.net.URL;
import javax.xml.namespace.QName;
import javax.xml.ws.Service;
import ws.HelloWSBean;
public class Client {                                          //客户端类
public static void main(String[] args) throws Exception {  //主方法
        //创建一个指向该 WSDL 的 URL
        URL url
              = new URL("http://localhost:8080/HelloWSBeanService/
              HelloWSBean?WSDL");
        //创建一个 QName
        QName qName =  new QName("http://ws/","HelloWSBeanService");
        Service service = Service.create(url,qName);//得到服务
        HelloWSBean hwsb = service.getPort(HelloWSBean.class);
                                                       //得到端口（接口的实现）
        String info = hwsb.sayHello("zbs");            //使用接口
        System.out.println(info);
    }
}
```

说明：以上的示例代码中的一些类和接口是需要使用 JWS 工具（wsimport）来生成的，这些工具在 JDK 中可以找到，以 WSDL 的 URL 为参数即可。一些 IDE 提供了类似的图形化操作的工具也是可以做到的。

一旦该 Web Service 发布以后，任何可以通过 HTTP 协议访问到应用服务器的客户端都是可以使用该 Web 服务的，如 Java 客户端程序可以，PHP、.NET 程序也可以。

【答案】

Java 开发 Web 服务程序的 API 为 JWS，它得到 Java EE 服务器的支持，把 EJB 包装成 Web 服务的时候，也是需要使用该 API 的 WebService 和 WebMethod 标注，分别修饰 Bean 类和接口方法，例如下面的代码：

```
@Stateless                                              //无状态会话 Bean 标注
@WebService                                             //Web 服务标注
public class HelloWSBean implements HelloWSRemote {//Bean 类
    @WebMethod                                          //Web 服务方法
    public String sayHello(String user) {
```

```
        return "hello "+user;
    }
}
```

面试题 155　JMS 分哪两种开发模式

基于消息驱动开发应用程序是一种异步开发的技术，在 Java EE 知识体系中，JMS 是专门针对消息服务而设计的。那么，什么是消息服务呢？JMS 包括哪些开发模式呢？它的开发思路是什么呢？本例在回答该问题的同时，详细地讲解消息服务的原理和 JMS 模型。

【出现频率】 ★★★★

【关键考点】

- 消息服务和消息中间件；
- JMS 的开发模型。

【考题分析】

在现实开发中，往往存在这样的需求，有些服务器端的处理是比较麻烦和漫长的，如果让客户端长期等待会显得非常不友好。那么，能不能实现客户端与服务器端的联系是异步的呢？也就是说，客户端发出某种业务请求以后，就不管服务器端是怎么处理的了，等一段时间或以监听的形式去看看处理结果如何。这就是消息服务，用消息服务来作为异步通信的桥梁。

用来处理消息的 Server 或程序就称为消息中间件，它为消息的客户端提供了消息的接收、存放、可靠的转发服务。它所带来的直接好处就是异步编程，让生成者和消费者解耦合，保证了双方甚至多方的可靠性，任何一个消费者或生产者的运行与否并不影响整体。

JMS 是 SUN 公司发布的一套保准 API，它隶属于 Java EE 规范，用来屏蔽具体的消息中间件产品之间的差异，让开发者用统一的方式来使用消息中间件服务。

说明：消息中间件往往是以一个具体的产品提供的，例如，IBM 的 MQ、微软的 MSMQ、SUN 的 Message Server 等。这些产品都有自己的一些特殊的规定，使用方式也千差万别。JMS 就是 Java 开发者使用消息中间件的一种标准形式，大多数的消息中间件产品都提供了 JMS 的 SPI 实现。

JMS 规定了两种消息模式：发布订阅模式和点对点模式。

1. 发布订阅模式

该模式的消息是以一个主题（Topic）的形式发布，可以允许有多个消费者和生成者，同一条消息可以被多个消费者所消费，而且消费者一端必须先于生成者运行，代表一种消息的订阅。

对于发布订阅模式的 JMS 程序来说，主要需要使用到得是 Topic 连接工厂（TopicConnectionFactory）、Topic 连接（TopicConnection）、Topic 会话（TopicSession）、Topic 订阅器（TopicSubscriber）和 Topic 发布器（TopicPublisher），大致的编程思路如下所示：

（1）通过 JNDI 在 Java EE 的服务器中找到适当的 TopicConnectionFactory 和 Topic。
（2）由连接工厂生产出的连接。
（3）通过连接获得会话。
（4）由主题（Topic）创造出订阅器或发布器。
（5）创建消息实例（Message）。
（6）由订阅器或发布器来订阅或发布消息。
（7）关闭连接。
以下为一个 Topic 消息订阅的示例程序：

```
package topicreceiver;
import java.util.Properties;
import javax.jms.*;
import javax.naming.*;

public class MyTopicReceiver {                          //客户端类，订阅
    public MyTopicReceiver() {                          //构造方法
    }

    public static void main(String[] args) throws Exception {//主方法
        Context ctx = getInitialContext();
        TopicConnectionFactory tcf =                    //通过 JNDI 获得连接工厂
              (TopicConnectionFactory) ctx.lookup("jms/MyTopicConnection
              Factory");
        Topic topic = (Topic) ctx.lookup("jms/MyTopic");//通过 JNDI 获得主题
        TopicConnection conn = tcf.createTopicConnection();//创造一个连接
        TopicSession session =                          //创造一个会话
              conn.createTopicSession(false,TopicSession.AUTO_
              ACKNOWLEDGE);
        TopicSubscriber subscriber =
              session.createSubscriber(topic);          //创造发布器
        conn.start();                                   //连接打开
        Message message = subscriber.receive();         //收取消息
        if(message instanceof TextMessage){
            TextMessage m = (TextMessage)message;//强转为文本类消息
            System.out.println(m.getText());            //获得消息内容
        }
        subscriber.close();                             //关闭连接
        session.close();
        conn.close();
    }
    public static Context getInitialContext(){  //获得 JNDI 初始化上下文
        Properties props = new Properties();
        props.setProperty("jndi.factory","com.sun.jndi.cosnaming.
        CNCtxFactory");
        props.setProperty("org.omg.CORBA.ORBInitialHost","localhost");
        props.setProperty("org.omg.CORBA.ORBInitialPort","3700");
        Context ctx = null;
        try {
            ctx = new InitialContext(props);            //用 prop 创建
        } catch (NamingException ex) {
            ex.printStackTrace();
        }
        return ctx;
    }
}
```

注意：以上示例代码中的连接一定要打开（start），否则无法使用。

以下为一个 Topic 消息发布的示例程序：

```
package topicsender;
import java.util.Properties;
import javax.jms.*;
import javax.naming.*;

public class MyTopicSender{                                    //客户端类，发布
    public MyTopicSender () {                                  //构造方法
    }
    public static void main(String[] args) throws Exception {//主方法
        Context ctx = getInitialContext();
        TopicConnectionFactory tcf =                           //通过 JNDI 获得连接工厂
               (TopicConnectionFactory) ctx.lookup("jms/MyTopicConnection-
                Factory");
        Topic topic = (Topic) ctx.lookup("jms/MyTopic");//通过 JNDI 获得主题
        TopicConnection conn = tcf.createTopicConnection();//创造一个连接
        TopicSession session =                                 //创造一个会话
              conn.createTopicSession(false,TopicSession.AUTO_
              ACKNOWLEDGE);
         //创造发布器
        TopicPublisher publisher  = session.createPublisher(topic);
         //创造一个文本类消息，内容为 hello
        TextMessage message = session.createTextMessage("hello!");
        message.setStringProperty("version","1");    //设置属性
        publisher.publish(message);                            //发布消息
        publisher.close();                                     //关闭连接
        session.close();
        conn.close();
    }
    public static Context getInitialContext(){        //获得 JNDI 初始化上下文
        Properties props = new Properties();
        props.setProperty("jndi.factory","com.sun.jndi.cosnaming.
        CNCtxFactory");
        props.setProperty("org.omg.CORBA.ORBInitialHost","localhost");
        props.setProperty("org.omg.CORBA.ORBInitialPort","3700");
        Context ctx = null;
        try {
            ctx = new InitialContext(props);                   //用 prop 创建
        } catch (NamingException ex) {
            ex.printStackTrace();
        }
        return ctx;
    }
}
```

以上的示例代码，首先运行的是订阅器，然后再运行发布器。这样代表着，消费者对某种消息订阅，然后生成者来满足消费者的消息。

2. 点对点模式

使用队列（Queue）的方式来存放消息，生成者可以有多个，但是同一条消息只能被一个消费者所消费。与发布订阅模式不同，点对点模式不规定消费者和生成者的运行顺序。

对于点对点模式的 JMS 程序来说，主要使用到的是 Queue 连接工厂

（QueueConnectionFactory）、Queue 连接（QueueConnection）、Queue 会话（QueueSession）、Queue 接收器（QueueReceiver）和 Queue 发送器（QueueSender）。基本的编程思路如下。

（1）通过 JNDI 在 Java EE 的服务器中找到适当的 QueueConnectionFactory 和 Queue。

（2）由连接工厂造出连接。

（3）通过连接获得会话。

（4）由会话创造出消息接收器或发送器。

（5）创建消息实例（Message）。

（6）由消息接收器或发送器来从 Queue 接收消息，或发送消息到 Queue 中。

（7）关闭连接。

以下为一个 Queue 模型的接收消息的示例程序。

```
package queuereceiver;
import java.util.Properties;
import javax.jms.*;
import javax.naming.*;

public class MyQueueReceiver {                          //客户端类，接收
    public MyQueueReceiver () {                         //构造方法
    }
    public static void main(String[] args) throws Exception {//主方法
        Context ctx = getInitialContext();
        QueueConnectionFactory tcf =                    //通过 JNDI 获得连接工厂
              (QueueConnectionFactory) ctx.lookup("jms/MyQueueConnection
              Factory");
        Queue queue= (Queue) ctx.lookup("jms/MyQueue");//通过 JNDI 获得 Queue
        QueueConnection conn = tcf.createQueueConnection(); //创造一个连接
        QueueSession session =                          //创造一个会话
              conn.createQueueSession(false, QueueSession.AUTO_
              ACKNOWLEDGE);
        QueueReceiver receiver =                        //创造接收器
              session.createReceiver(queue);
        conn.start();                                   //开始连接
        Message message = receiver.receive();           //接收消息
        if(message instanceof TextMessage){
            TextMessage tm = (TextMessage)message;      //转换消息类型
            String info = tm.getText();                 //获得消息内容
            System.out.println("received message:"+info);
        }
        receiver.close();                               //关闭连接
        session.close();
        conn.close();
    }
    public static Context getInitialContext(){          //获得 JNDI 初始化上下文
        Properties props = new Properties();
        props.setProperty("jndi.factory","com.sun.jndi.cosnaming.
        CNCtxFactory");
        props.setProperty("org.omg.CORBA.ORBInitialHost","localhost");
        props.setProperty("org.omg.CORBA.ORBInitialPort","3700");
        Context ctx = null;
        try {
            ctx = new InitialContext(props);            //用 prop 创建
        } catch (NamingException ex) {
            ex.printStackTrace();
        }
```

```
        return ctx;
    }
}
```

以下为一个 Queue 模型的发送消息的示例程序。

```
package queuesender;
import java.util.Properties;
import javax.jms.*;
import javax.naming.*;
public class MyQueueSender {                    //客户端类，接收

    public MyQueueSender () {                   //构造方法
    }
    public static void main(String[] args) throws Exception {//主方法
        Context ctx = getInitialContext();
        QueueConnectionFactory tcf =            //通过 JNDI 获得连接工厂
                (QueueConnectionFactory) ctx.lookup("jms/MyQueueConnecti-
                onFactory");
        Queue queue= (Queue) ctx.lookup("jms/MyQueue");//通过 JNDI 获得 Queue
        QueueConnection conn = tcf.createQueueConnection(); //创造一个连接
        QueueSession session =                  //创造一个会话
                conn.createQueueSession(false, QueueSession.AUTO_
                ACKNOWLEDGE);
        QueueSender sender =
                session.createSender(queue);  //创造发送器
        TextMessage message = session.createTextMessage();//创造文本消息对象
        message.setText("hello!");              //设置消息内容
        sender.send(message);                   //发送消息
        sender.close();                         //关闭连接
        session.close();
        conn.close();
    }
    public static Context getInitialContext(){//获得 JNDI 初始化上下文
        Properties props = new Properties();
        props.setProperty("jndi.factory","com.sun.jndi.cosnaming.
        CNCtxFactory");
        props.setProperty("org.omg.CORBA.ORBInitialHost","localhost");
        props.setProperty("org.omg.CORBA.ORBInitialPort","3700");
        Context ctx = null;
        try {
            ctx = new InitialContext(props); //用 prop 创建
        } catch (NamingException ex) {
            ex.printStackTrace();
        }
        return ctx;
    }
}
```

以上的示例代码，不必讲究运行顺序。如果发送器先运行，则把消息发送到服务器中的队列中保存，接收器可以随时来取。如果是接收器先运行，它会尝试去取一个消息，若没有取到则会进入阻塞状态，一直监听该 Queue，直到收到消息为止。

说明： 在接受消息的时候，也可以设置一个超时时间，如果超过这个时间仍然没有收到消息，则关闭连接，退出程序。

【答案】

根据消息的发布和接收类型不同，JMS 存在以下两种开发模式：

（1）订阅发布模式：它始终针对一个主题（Topic），就好像消费者订阅报纸一样，先发起一个订阅某种消息的请求，再发给消费者。

（2）点对点模式：它针对的是一个消息队列（Queue），生产者负责在里面发消息，消费者则负责在里面读消息。

面试题 156　如何使用消息驱动 Bean 进行异步开发

EJB 从 2.0 版本开始，提供了一种消息驱动 Bean 来支持消息程序的开发，它的主要目的是想简化消息程序的开发。那么，消息驱动 Bean 的开发思路又是什么呢？本例在回答该问题的同时，详细地讲解消息驱动 Bean 的使用思路。

【出现频率】 ★★★★

【关键考点】

- 消息服务和消息中间件；
- JMS 的开发模型；
- 消息驱动 Bean 的开发思路和步骤。

【考题分析】

消息驱动 Bean 作为 EJB 的一个 Bean 种类，它是用来作为消息消费者的。顾名思义，消息驱动 Bean 就是，一旦收到消息以后，做出某种行为的 Bean。它与普通的消息消费者程序有什么区别呢？首先，消息驱动 Bean 处于 EJB 容器中，可以享受很多 EJB 的中间件服务。另外，EJB 在对待消息驱动 Bean 的时候，会建立一个专门的对象池来处理，并发性很高。

消息驱动 Bean 不像会话 Bean，它不能被客户端直接访问，客户端只能通过发送消息的方式来使用消息驱动 Bean。但是，消息驱动 Bean 的声明周期与无状态会话 Bean 类似，它在 EJB 容器生存和活动的规律也比较像。

编写一个消息驱动 Bean，可以按照以下思路来进行。

（1）写一个类，实现 javax.jms.MessageListener 接口。

（2）为该类添加 javax.ejb.MessageDriven 标注，并用 mappedName 指明消息目的地。

（3）完成 onMessage()方法，它的参数为一个 Message 对象，它代表的就是接收到得消息实例。

（4）如果有必要，可以实现 PostConstruct()和 PreDestroy()方法。

以下是一个消息驱动 Bean 的示例：

```
/*
 * MyMDB.java
 *
*/
package jms;
import javax.annotation.Resource;
import javax.ejb.ActivationConfigProperty;
import javax.ejb.MessageDriven;
```

```
import javax.ejb.MessageDrivenContext;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.TextMessage;
@MessageDriven(mappedName = "jms/MyQueue")      //标注，指明消息目的地
public class MyMDB implements MessageListener {//实现 MessageListener 接口
    public MyMDB() {                            //构造方法
    }
    public void onMessage(Message message) {    //接口方法
        if(message instanceof TextMessage){
            TextMessage tm = (TextMessage)message;//转换消息类型
            String info;
            try {
                info = tm.getText();            //获得消息内容并打印
                System.out.println("MyMDB received message:"+info);
            } catch (JMSException ex) {
                ex.printStackTrace();
            }
        }
    }
}
```

注意：如果需要使用消息驱动上下文 MessageDrivenContext，可以通过 Resource 标注进行注入，例如下面的代码：

```
    @Resource
    private MessageDrivenContext mdc;
```

【答案】

消息驱动 Bean 是专门为消息消费而设计的，它作为 EJB 的一个种类，可以享受到 EJB 容器的一切服务，包括事务、依赖注入等。它除了接受客户端访问的方式不同于无状态会话 Bean 以外，其他的特性与无状态会话 Bean 非常类似。

消息驱动 Bean 的基本编程思路如下：

（1）写一个类，实现 javax.jms.MessageListener 接口。

（2）为该类添加 javax.ejb.MessageDriven 标注，并用 mappedName 指明消息目的地。

（3）完成 onMessage()方法。

（4）如果有必要，可以实现 PostConstruct()和 PreDestroy()方法，以及依赖注入驱动上下文对象。

14.2　JPA 规范

JPA 是一套标准的进行对象关系映射的规范，它在 EJB 3.0 中，起到了替代实体 Bean 的作用。不仅在 EJB 中，JPA 还可以应用到任何的 Java 程序中。本节将集中讨论有关 JPA 的常见面试题。

面试题 157　JPA 的使用思路是什么

JPA 是一套标准的用于对象持久化的 API，它想对于其他的一些具体的 ORM 框架，例如，Hibernafe、iBATIS。它的思想有什么不同呢？它的使用规则又有什么不同呢？本例在回答该问题的同时，详细地讲解 JPA 的定义、使用范围和使用思想。

【出现频率】 ★★★★

【关键考点】

- 对象持久化思想；
- JPA 的定义和思想；
- JPA 的 API 使用方法。

【考题分析】

JPA 是 EJB 3.0 推出的一项新的用于对象持久化的 API，它相当于官方的 ORM 的标准规范。由于 JPA 是由一套标准的接口组成，它不含具体的实现，是对具体的 ORM 框架的一种抽象，使用 JPA 编写的程序可以不依赖于具体的 ORM 解决方案。

尽管 JPA 是伴随着 EJB 3.0 而推出，但是它允许在 EJB 容器以外的地方使用，并不依赖于 EJB 容器。JPA 操作的实体不在是实体 Bean，而是 POJO（普通的 Java 对象），大大地拓宽了 JPA 的适用范围。

JPA 是实体，不再属于 EJB 的范畴，它不能被客户端直接访问，一般只能通过会话 Bean 进行间接的访问。实体的生存时间相对于会话 Bean 来说，要长得多。而且，实体是可以有一些自己的业务逻辑，例如，银行账户的存取款。

对于实体的配置，JPA 提供了一系列的标注来完成，主要包括如下内容：

（1）Entity：标明一个类为实体类。

（2）Table：用于指定该实体所对应的数据库表格。

（3）Id：标明该属性为主键。

（4）GeneratedValue：指定主键生成策略。

（5）Column：标明一个对象属性。

说明： 对于 JPA 来说，它的大多数的接口、类、枚举和标注都定义在 javax.persistence 包中。

下面为一个银行账号实体的编写示例。

```
package entity;
import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
/** * 实体类 Account* */
@Entity(name="Account1")                                    //实体标注
@Table(name="Account_table")                                //数据库表格标注
public class Account implements Serializable {             //实体类
```

```
@Id                                                    //Id
@GeneratedValue(strategy = GenerationType.AUTO)        //生成策略
private Long id;
@Column(name="name",length=20,nullable=false,unique=true)   //账号属性
private String accountName;
private double balance;
/** Creates a new instance of Account */
public Account() {
}
/**     * 获取此 Account 的 id。      * @return id     */
public Long getId() {
    return this.id;
}
/**     * 将此 Account 的 id 设置为指定的值。      * @param id, 新建 ID     */
public void setId(Long id) {
    this.id=id;
}
/**
 * 返回对象的散列代码值。该实现根据此对象
 * 中 ID 字段计算散列代码值。
 * @return 此对象的散列代码值。
 */
@Override
public int hashCode() {
    int hash = 0;
    hash += (this.getId() != null ? this.getId().hashCode() : 0);
    return hash;
}
/**
 * 确定其他对象是否等于此 Account。
 * 参数不为 null 且该参数是具有与此对象相同 ID 字段值的 Account 对象时,
 * 结果才为 <code>true</code>。
 * @param 对象, 要比较的引用对象
 * 如果此对象与参数相同, 则 @return <code>true</code>;
 * 否则为 <code>false</code>。
 */
@Override
public boolean equals(Object object) {
    if (!(object instanceof Account)) {
        return false;
    }
    Account other = (Account)object;
    if (this.getId() != other.getId() && (this.getId() == null
    || !this.getId().equals(other.getId()))) return false;
    return true;
}
/**
 * 返回对象的字符串表示法。该实现根据 ID 字段
 * 构造此表示法。
 * @return 对象的字符串表示法。
 */
@Override
public String toString() {
    return "entity.Account[id=" + getId() + "]";
}
public String getAccountName() {
```

```
        return accountName;
    }
    public void setAccountName(String accountName) {
        this.accountName = accountName;
    }
    public double getBalance() {
        return balance;
    }
    public void setBalance(double balance) {
        this.balance = balance;
    }
}
```

实体定义完成以后，就可以把实体放在持久化上下文中进行增、删、查、改等操作。JPA 的持久化上下文是一个集合对象，用来放置需要持久化的实体，由 EntityManager 所管理。EntityManager 是一个接口，用于应用程序持久化实体，它提供了一些常用的增、删、查、改实体的方法，例如下面的代码：

```
public double getBalance(Long accountId) {
    Account a = em.find(Account.class,accountId);    //根据 ID 查找实体
    return a.getBalance();
}
public void update(Account a) {
    em.merge(a);                                     //更新实体
}
public void remove(Long accountId) {
    Account a = em.find(Account.class,accountId);    //删除实体
    em.remove(a);
}
public void save(Account a) {
    em.persist(a);                                   //保存实体
}
```

如果是在EJB容器中，可以通过javax.persistence.PersistenceContext标注进行依赖注入；若在容器以外使用的话，则通过实体管理器工厂 EntityManagerFactory 来创建。

说明： EntityManager 相当于 Hibernate 的 Session，EntityManagerFactory 则相当于 Hibernate 的 SessionFactory。

【答案】

使用 JPA 的基本步骤如下。

（1）使用标注定义和配置实体及其属性。

（2）通过 EJB 容器注入，或单独的 EntityManagerFactory 创建 EntityManager。

（3）使用 EntityManager 的方法，完成对实体的增、删、查、改的操作。

（4）关闭 EntityManager。

面试题 158　无状态会话 Bean 如何获得和使用 EntityManager

尽管 JPA 是可以在任何环境下使用，但是 EJB 容器可以让它比其他任何一个环境更加如鱼得水。对于一个无状态的会话 Bean 来说，应该如何去获得一个 EntityManager 呢？最简单的办法就是依赖注入。本例在回答该问题的同时，详细地讲解 JPA 在无状态会话 Bean

中的使用方法。

【出现频率】 ★★★★

【关键考点】

- JPA 的定义和思想；
- EJB 依赖注入 EntityManager 的原理和方式是什么。

【考题分析】

大家知道，一个 EntityManager 是由 EntityManagerFactory 创建的，它对应了一个特定的数据源，这些数据源的信息是与 EntityManagerFactory 对象所对应的。那么 EJB 容器是如何来创建这个指向某个数据源的 EntityManagerFactory 的呢？

在 EJB 中，一个特定的实体持久化环境称为持久化单元，它对应着一个数据源和一个具体的 ORM 解决方案（例如，Hibernate 和 iBATIS）。这些信息可以通过“META-INF”文件夹下的 persistence.xml 文件进行配置，每个持久化单元都有一个具体的与其他单元区别的名字，以下是配置示例：

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0"
    xmlns="http://java.sun.com/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
    http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
  <persistence-unit name="MyPU" transaction-type="JTA">
                                                <!-- 持久化单元 -->
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
                                                <!-- 具体的 JPA 实现框架 -->
    <jta-data-source>jdbc/sample</jta-data-source>  <!-- 数据源 -->
    <properties>
      <property name="hibernate.hbm2ddl.auto" value="update"/>
                                                <!-- 配置属性 -->
    </properties>
  </persistence-unit>
</persistence>
```

当 EJB 模块被部署到 Java EE 服务器的时候，Java EE 服务器会根据配置信息创建持久化单元，也就是 EntityManagerFactory 实例。接下来，就可以在会话 Bean 中注入 EntityManager，注入方式是使用 PersistenceContext 标注，例如代码如下：

```
import java.util.List;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
/** * * @author  */
@Stateless(mappedName="AccountMgmtBean")
public class AccountMgmtBean implements AccountMgmtRemote {

    @PersistenceContext(unitName="MyPU")              //指明持久化单元的上下文
    private EntityManager em;                         //实体管理器
    //...
}
```

技巧：如果 persistent.xml 文件中只定义了一个持久化单元，则在使用 PersistenceContext 依赖注入的时候，无须指定 unitName。

EntityManager 是一个接口，相当于 Hibernate 中的 Session 接口，它是开发人员主要使用的接口，用于应用程序持久化实体，它提供了一些常用的增、删、查、改实体的方法。在会话 Bean 中使用 EntityManager 时，它会为每个方法执行创造一个新的实例，执行完毕以后也无须关闭，容器会负责这些冗余的事情，例如下面的代码：

```
pachage mysb;
import java.util.List;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

@Stateless(mappedName="AccountMgmtBean")                    //无状态会话 Bean
public class AccountMgmtBean implements AccountMgmtRemote {
    @PersistenceContext(unitName="MyPU")                    //指明持久化单元的上下文
    private EntityManager em;                               //实体管理器
     public double getBalance(Long accountId) {
        Account a = em.find(Account.class,accountId);       //根据 Id 查找实体
        return a.getBalance();
    }
    public void update(Account a) {
        em.merge(a);                                        //更新实体
    }
    public void remove(Long accountId) {
        Account a = em.find(Account.class,accountId);       //删除实体
        em.remove(a);
    }
    public void save(Account a) {
        em.persist(a);                                      //保存实体
    }
}
```

【答案】

在 EJB 容器中使用 JPA 可以简化开发者的开发步骤，让开发者更注意业务逻辑，而不是持久化上下文的事情。一个无状态会话 Bean 在使用 JPA 的时候，一般只关注 EntityManager，它的注入和使用思路如下所示：

（1）在“META-INF”文件下的 persistence.xml 中，配置持久化单元。

（2）使用 javax.persistence.PersistenceContext 注入特定持久化单元的 EntityManager。

（3）在业务方法中，使用 EntityManager 对象进行实体的增、删、查、改操作。

（4）EntityManager 的打开和关闭由 EJB 容器负责维护，开发者无须关心。

面试题 159　JPA 可以在 EJB 容器以外的地方使用吗？

既然 JPA 是一套标准的对象持久化操作的 API，它无疑是可以在任何地方使用的。那么对于一个普通的 Java 程序来说，应该如何使用 JPA 的那些标注来配置实体，使用 EntityManager 来操作实体；使用 EntityManagerFactory 来生成 EntityManager 呢？本例在回答该问题的同时，全面地讲解普通 Java 程序使用 JPA 的思路。

【出现频率】 ★★★★

【关键考点】

- JPA 的定义和思想；
- JPA 的 API 使用方法。

【考题分析】

在一个普通的 Java 程序中，要使用 JPA 进行对象持久化操作的话，依然需要用 persistence.xml 的完成对持久化单元的配置。需要指定的内容包括：

（1）持久化单元的名字。

（2）事务类型，一般用 JTA 或本地事务。

（3）持久化实现类，例如，Hibernate 提供的是 org.hibernate.ejb.HibernatePersistence。

（4）ORM 框架本身需要的配置。例如，Hibernate 的连接信息。

以下是一个使用本地事务的，以 Hibernate 为支持的 JPA 的持久化单元的配置示例：

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
    http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd" version=
    "1.0">
     <!-- 本地事务 -->
     <persistence-unit name="MyJPAPU" transaction-type="RESOURCE_LOCAL">
         <provider>org.hibernate.ejb.HibernatePersistence</provider>
         <!-- Hibernate -->
         <!-- Hibernate 的连接配置信息 -->
         <properties>
             <!-- 数据库驱动类 -->
             <property name = "hibernate.connection.driver_class"
                 value = "com.mysql.jdbc.Driver"/>
             <!-- 数据库连接 URL -->
             <property name = "hibernate.connection.url"
                 value = "jdbc:mysql://localhost:3306/test"/>
             <!-- 数据库账号和密码 -->
             <property name = "hibernate.connection.username" value =
             "root"/>
             <property name = "hibernate.connection.password" value =
             "123"/>
         </properties>
     </persistence-unit>
</persistence>
```

注意：配置文件 persistence.xml 必须放在 META-INF 文件夹下，才能被程序所读取。

若是 Java Web 程序中使用 JPA，一般还可以使用 JTA 的数据源，例如，Tomcat 就提供了数据库连接池和数据源的 JNDI 的支持。此时，就把事务类型指定为“jta”，然后用<jta-data-source>标签指定数据源，例如下面的代码：

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
    http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd" version=
    "1.0">
     <!-- 本地事务 -->
```

```
    <persistence-unit name="MyJPAPU" transaction-type="jta">
        <provider>org.hibernate.ejb.HibernatePersistence</provider>
        <!-- Hibernate -->
        <jta-data-source>java:env/com/mysql</jta-data-source>
        <!--  数据源 JNDI  -->
        <!--  其他配置信息  -->
        <properties>
        </properties>
    </persistence-unit>
</persistence>
```

对于实体来说，它的定义与 EJB 中的实体定义并无区别，也是使用 javax.persistence 包下的标注进行配置，以下是一个用户实体的定义和配置示例：

```
package ch14;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
@Entity
@Table(catalog="test",name="user")                    //指定表名
public class User {
    @Id                                               //定义 ID
    private Long id;
    @Column                                           //定义属性
    private String name;
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

说明：使用 javax.persistence 之前，必须引入 Java EE 规范的 jar 文件才可以，一般是 persistence.xml 或 javaee.jar。

持久化单元和实体都准备好以后，就应该创建 EntityManagerFactory 和 EntityManager 了。应该如何创建呢？EntityManagerFactory 一般通过 javax.persistence.Persistence 类的静态方法 createEntityManagerFactory()进行创建。然后，接下来的操作就和 Hibernate 的比较类似了。示例代码如下：

```
package ch14;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
public class Test {                                   //测试类
    //主方法
    public static void main(String[] args) {
        //根据持久化单元创造 EntityManagerFactory
```

```
        EntityManagerFactory emf = Persistence.createEntityManagerFactory
        ("MyJPAPU");
        EntityManager em = emf.createEntityManager(); //创建 EntityManager
        em.getTransaction().begin();                   //开始事务
        User user = new User();
        user.setId(1L);
        user.setName("zhangsan");
        em.persist(user);                              //持久化操作
        em.getTransaction().commit();                  //提交事务
        em.close();                                    //关闭 EntityManager
    }
}
```

【答案】

JPA 可以在 EJB 容器以外的地方使用，只不过使用的方式要麻烦一些。对于一个普通的 Java 程序来说，JPA 的使用思路如下：

（1）完成配置文件 persistence.xml。它里边配置了持久化单元，根据事务的不同，可以采取本地事务或 JTA 的事务。另外，还需要把框架（如 Hibernate）需要的一些配置信息加入到< properties>中。

（2）使用 javax.persistence 包下的标注完成实体的定义和配置。

（3）在代码中，使用 javax.persistence.Persistence 创建 EntityManagerFactory，再由 EntityManagerFactory 得到 EntityManage。然后，利用 EntityManager 进行实体的各种持久化操作，每次执行完毕后，则关闭 EntityManager。

14.3　小　　结

本章讲解了一些关于 EJB 和 JPA 的面试题。首先讲解了 EJB 的概念和种类，然后是两种会话 Bean 的使用思路。接着讲了一些关于 EJB 的依赖注入的问题，以及 JMS 的使用思路和消息驱动 Bean 的使用方法。最后把 JPA 单独作为一节进行讲解，主要是 JPA 的使用思路和它与 EJB 的关系。

EJB 在 3.0 以前，只有很少的人能使用它来构建大型的企业级应用程序，而且维护难度也非常高，没有在市场上流行开来。但是 EJB 3.0 打开了一个新的局面，它的设计思想比较符合现在构造大中小型的企业级应用的思想，也被许多软件公司所采纳，因此 EJB 方面的面试题也是重点。学习 EJB 的重点在于思想的理解，因为 EJB 程序说到底就是 API 的使用规则，所以读者应该认真的去练习和体会 EJB 的各种技术的思想。

第5篇　算法和设计模式

第 15 章　Java 编程试题

算法是任何程序的灵魂，一些开发人员认为 Java 提供了丰富的 API，已经可以比较容易地完成大多数的功能，因此程序员可以不用管算法了，只要熟悉 API 的用法就好。这是错误的想法，事实上，Java 程序的执行效率大多数情况下，依然是取决于开发人员的算法。尤其对于应届毕业生，招聘单位往往不会考察应届毕业生太多实践的东西，而是要求他们具有敏捷的一个算法头脑。不仅一些传统的经典算法是 Java 笔试或面试的重点，还有一些开发模式也是 Java 面试考察的重点。本章将包含关于 Java 算法和开发模式一些常见面试题，并且分析这些题目和知识点，帮助读者梳理这些方面的知识。

15.1　基础编程试题

一些简单的算法，可以通过比较少的代码表现出来。但是，通过这些代码却可以看出求职者编程的功底和习惯。本节将集中讨论有关基础的结构化编程和数据结构算法的常见面试题。

面试题 160　打印出 100 以内的素数

相信大多数读者都应该见过类似的题目，它本质上就是判断一个整数是否为素数。实现可以多种多样，但是效率却不同。怎样实现才能达到高效率呢？本例在回答该问题的同时，详细地讲解高效率的判断素数的算法。

【出现频率】 ★★★★

【关键考点】

- 结构化编程基础；
- 编程实践能力。

【考题分析】

素数又称为质数，它的定义是：只能被 1 和被自己整除的整数。其中，1 不是素数，任何时候都不用考虑 1。

素数的判断方法是比较明确的，就是拿比自己小的整数依次进行“除以”操作，若能除尽则表示不是素数，因此很容易就想到如下的做法：

```
for(int i=2;i<num;i++){          //循环遍历
    if(num % i ==0){             //取余
        return false;            //一旦除尽则直接返回 false
    }
}
```

```
return true;                              //若一直到最后都没返回，证明它是素数
```

以上的做法是可以达到目的的，也比较符合素数的定义。但是，效率就很低了，进行了很多次不必要的循环。例如，一个数字肯定不能被大于它的 1/2 的整数所整除，因此，改进一下 for 循环的条件，不必遍历那么多次，修改以后如下所示：

```
for(int i=2;i<num/2;i++){                 //循环 num 的二分之一次就够了
    ...
}
```

尽管以上的做法已经比较有效率了，但是还不够。对数字敏感一点人就会知道，其实只需要小于该数字的二次根也是可以达到目的的，因为大于该数字二次根的数字也是不能整除了，所以，再改进一下 for 循环的条件，代码如下。

```
for(int i=2;i<Math.sprt(num);i++){        //循环到 num 的二次根次就够了
    ...
}
```

注意：以上的代码中还有一个小小的问题，就是每一次循环都进行一次取二次根的操作，这是会影响效率的。所以，应该在循环之前就把二次根的值取好。

【答案】

该编程题的思路大致如下：

（1）完成一个判断某整数是否为素数的方法。

（2）循环 1～100。

（3）每循环一次就判断一次，返回 true 则打印。

以下是该题目的编程示例：

```
package ch15;
public class Prime {
    //主方法
    public static void main(String[] args) {
        //遍历 1 到 100
        for (int i = 1; i < 100; i++) {
            if (isPrime(i)) {                       //判断是否为素数
                System.out.println(i);              //打印素数
            }
        }
    }
    //判断一个整数是不是素数的方法
    private static boolean isPrime(int num) {
        if(num == 1)                                //1 不是素数，直接返回 false
            return false;
        //从 2 开始到该整数的 2 次根之间遍历
        long sprtNum = (long) Math.sqrt(num);       //得到该数字的 2 次根
        for (int i = 2; i <= sprtNum; i++) {
            if (num % i == 0) {                     //判断是否能除尽
                return false;                       //返回 false
            }
        }
        return true;                                //返回 true
    }
}
```

面试题 161　打印九九乘法口诀表

相信没有人不知道九九乘法表吧，它就像一个梯子一样，一共 9 层，是学习算术乘法的基础。大多数人一想到它的实现，首先映入脑海的就是用两个 for 循环就可以了。但是，可不可以只用一个 for 循环就搞定呢？本例在回答该问题的同时，详细地讲解高效率的打印九九乘法表的算法。

【出现频率】 ★★★★

【关键考点】

- 结构化编程基础；
- 编程实践能力。

【考题分析】

九九乘法表，由 9 行组成，代表了 10 以内的整数之间的乘积结果。因此，很容易就会想到，用两个 for 循环，一个代表列，一个代表行，例如下面的代码：

```
for(int i = 1 ;i <= 9; i++){                    //遍历行
    for(int j = 1; j <= i; j++){                //遍历列，不能大于行的 i 值
        System.out.print(j + "*" + i + "=" + i * j + " "); //打印，空格隔开
    }
    System.out.println();                       //换行
}
```

这是很自然的想法，也没什么大问题。但是，大家可以思考一下，能不能用一个 for 循环来实现呢？答案是可以的。大体的思路是这样的：循环体内定义两个变量，一个控制列（i 变量），一个控制行（j 变量），i 变量每次循环都加一，但是它在换行以后又回到 1，j 变量则从 1～9，直到退出。那么，该思路的关键就在于如何判断是否该换行了，其实也比较简单，那就是一旦 i 变量自加 1 至它等于 j 变量以后，就应该换行了，循环代码如下：

```
for (int i = 1, j = 1; j <= 9; i++) {
    System.out.print(i + "*" + j + "=" + i * j + " ");
    if (i == j) {                               //判断是否该换行
        i = 0;
        j++;                                    //j 自加 1
        System.out.println();                   //换行
    }
}
```

【答案】

该编程题的思路大致如下：

（1）循环 1～9，采用两个循环变量，一个控制行，一个控制列。

（2）每循环一次就打印一句，若控制列的循环变量到底了，则打印换行。

以下是该题目的编程示例：

```
package ch15;
public class NineNineMulitTable {
    // 主方法
    public static void main(String[] args) {
```

```
        //循环，初始化 i 和 j 为 1
        for (int i = 1, j = 1; j <= 9; i++) {
            //间隔打印它们每一项的
            System.out.print(i + "*" + j + "=" + i * j + " ");
            if (i == j) {                          //判断是否该换行
                i = 0;                             //将 i 值赋为 0
                j++;                               //j 自加 1
                System.out.println();              //换行
            }
        }
    }
}
```

面试题 162　打印 10000 以内的回文数字

回文数是一种比较经典的算法，它比较多的出现在各种编程语言中，它的实现方式也是各种各样的。本例在回答该问题的同时，详细地讲解高效率的判断回文字的算法。

【出现频率】 ★★★★

【关键考点】

- 结构化编程基础；
- 编程实践能力。

【考题分析】

“回文”是指像“妈妈爱我，我爱妈妈”这样的，正读反读都相同的单词或句子。回文数则是指一个像 16461 这样“对称”的数，即将这个数的数字按相反的顺序重新排列后，所得到的数和原来的数一样。

根据回文数的定义，大家应该可以比较容易的想到，判断回文数的依据就是把回文数倒过来，然后比较它和原来的数字是否完全相等。那么，如何把数字给倒过来呢？熟悉 Java 字符串操作的开发者可能会说，把 int 型的数字变成字符串，再用 StringBuffer 的 reverse() 方法就可以了，然后调用 String 的 equals()方法，例如下面的代码：

```
int num = 12345;
String str = String.valueOf(num);                  //得到字符串格式的数字
StringBuffer sb = new StringBuffer(str);           //创建 StringBuffer 对象
sb.Reverse();                                      //转向
String newStr = sb.toString();                     //得到新的字符串数据
return str.equals(newStr);                         //比较两个字符串的值是否相等
```

以上的代码做法是可以达到目的的，但是它的效率比较低，经过了很多不必要的步骤。其实，完全可以不必转换成 String 以后再做比较，而直接用 int 型进行比较，关键点就在于读者要开动脑筋，实现一个调换 int 型数据的方法，例如下面的代码：

```
int temp = 0;                                      //反过来的值，初始化为 0
while (num > 0) {                                  //循环 number 的每一位数值
    temp = temp * 10 + num % 10;                   //得到一位数字
    num /= 10;                                     //num 减少一位
}
int newVal = temp;                                 //得到转换以后的值
```

以上代码中，关键点在于 while()循环，它所做的就是把原来的值循环的除以 10，得到每个位置上的数字，再把每个位置的数字反过来凑成一个新的数字。这样的算法比字符串要高明得多。

说明：10 以内的正整数不是回文数，没有判定的意义，因此可以不用判定 1～9，循环可以从 10 开始。

【答案】

该编程题的思路大致如下：

（1）完成一个把数字按位调换顺序的方法。

（2）循环 10～9999。

（3）每循环一次就判断一次，返回 true 则打印。

以下是该题目的编程示例：

```
package ch15;
public class CircleNumber {
    //主方法
    public static void main(String[] args) {
        //遍历 10～100000
        for (int i = 10; i < 10000; i++) {
            if (isCircleNumber(i)) {    //判断当前数字是否是回文数字
                System.out.println(i + "是回文数");  //打印
            }
        }
    }
    //判断是否为回文数字方法
    private static boolean isCircleNumber(int num) {
        int oldValue = num;                    //保存数值
        int temp = 0;                          //反过来的值，初始化为 0
        while (num > 0) {                      //循环 number 的每一位数值
            temp = temp * 10 + num % 10;//得到一位数字
            num /= 10;                         //num 减少一位
        }
        return temp == oldValue;               //判断反值与原值是否相等
    }
}
```

面试题 163　获得任意一个时间的下一天的时间

对日期的操作在日常开发中经常会使用到的。本题目表面上考察的是日期的判断，其实它真正想考察的是求职者对 Java 处理日期的原理是什么。如何用最简便的方式来得到任意时间的下一天呢？本例在回答该问题的同时，详细地讲解 Java 对日期格式的数据的处理方式。

【出现频率】 ★★★★

【关键考点】

- Java 的日期数据的存储原理

【考题分析】

Java 提供了 java.util.Date 类来处理日期格式的数据，通过它可以得到它所代表的日期的年月日和时分秒信息。因此，可以很自然的想到，要得到任何一个时间的下一天的时间，为 Date 的 Day 数据加上 1 天即可。但是，如果是月底怎么办？如果是年底怎么办？再如果是闰年怎么办？如果要在加上 1 天之前，进行这些判断的话，这样的程序就会变得相当的复杂。

注意：java.util.Date 没有时区的概念。因此如果需要使用时区的时候，请使用 java.util.Calendar 类。

其实，java.util.Date 类的底层的实现是通过一个 long 型的整型数据来保存日期的，这个值记录的是任何一个时间距 1970 年 1 月 1 日，0 日 0 分 0 秒的毫秒数。这里可以验证一下，通过执行下面一段代码可以得到一个整型数字 40。

说明：笔者写作时的年份是 2010 年。

```
long time = System.currentTimeMillis();                    //当前的毫秒数
System.out.println(time / 1000 / 60 / 60 / 24 / 365);  //得到距今多少年
```

以上的代码是用当前日志的毫秒数换算成年的单位，不偏不倚，正好是 40 年。因此，开发者完全可以不用管给定的时间是否是月底、年底或闰月的月底等条件，直接为它的毫秒数加上 24 小时所代表的毫秒数即可，然后再用新的 long 型的毫秒数构造一个新的 Date 类型的对象，该 Date 对象就是给定时间的下一天时间。

【答案】

以下为本题目的参考实现：

```
package ch15;
import java.util.Date;
public class NextDay {
    //主方法
    public static void main(String[] args) {
        Date now = new Date();          //获得当前时间
        //打印下一天的时间
        System.out.println(getNextDate(now));
    }
    //获得下一天
    public static Date getNextDate(Date d){
        long addTime = 1;               //以 1 为乘以的基数
        addTime *= 1;                   //1 天以后，如果是 30 天以后则这里是 30
        addTime *= 24;                  //1 天 24 小时
        addTime *= 60;                  //1 小时 60 分钟
        addTime *= 60;                  //1 分钟 60 秒
        addTime *= 1000;                //1 秒钟=1000 毫秒
        //用毫秒数构造新的日期
        Date date = new Date(d.getTime() + addTime);
        return date;                    //返回结果
    }

}
```

面试题 164　50 个人围成一圈数到 3 和 3 的倍数时出圈，问剩下的人是谁？在原来的位置是多少

出圈算法是一类比较典型的算法面试题，它可以很好地考察求职者的编程功底。由于它是一种循环的逻辑，因此它比起一般的基础算法题会更难一些。本例在回答该问题的同时，详细地讲解出圈算法的实现思路。

【出现频率】 ★★★★

【关键考点】

- 结构化编程基础；
- 编程实践能力。

【考题分析】

对于出圈的问题，它有一个比较大的困难点，就是它总是重复循环的，它的头就是它的尾巴，所以，出圈问题的循环语句是比较难写的。

该题目的圈的元素个数是 50 个，每次数到 3 或 3 的倍数的时候，就把当前元素出圈，并且继续数数，直到再遇到 3 的倍数。这里，如果下标从 0 开始，一直到一圈完成以后，它就会接到圈的首部，这应该如何处理呢？其实，最好的办法就是使用取余的办法，就可以始终得到 3 个倍数，无论它的倍数是多少，也不管它的元素个数是多少。

由于每次去掉元素以后，元素的个数会少一个，因此下一个 3 的倍数其实只需要走两步，在为其下标赋值的时候，需要减一，保持每次去掉的元素都是 3 的倍数。

说明： 如果使用从 0 开始的下标开始计算，那么初始化的时候应该使用-1，这样就可以模拟元素已经减少一个了。

至于元素的保存，可以使用数组，也可以使用链表。数组的元素去掉以后，它的下一个元素是不会自动往前移动的，不太好使用，但是也可以使用。这里，最好是使用 java.util.List 链表来表示，它既有下标，又可以很方便地获得元素的当前个数，尽管效率比数组要稍微低一些，不过已经足够了。

【答案】

该编程题的思路大致如下：

（1）首先把数据填充到数组或链表中。

（2）用一个 while 循环进行出圈，直到只剩下一个元素留下。

以下是该题目的编程示例：

```
package ch15;                                                    //包名
import java.util.LinkedList;
import java.util.List;
//测试类
public class Cycle {
    public static int cycle(int total, int k) {                  //功能方法
        List<Integer> dataList = new LinkedList<Integer>();      //创建链表对象
        for (int i = 0; i < total; i++)                          //添加数据元素
            dataList.add(new Integer(i + 1));
         //定义下标，模拟已经去掉一个元素，因此从-1 开始
```

```
        int index = -1;
        //一直循环去除数据，直到只剩下一个元素
        while (dataList.size() > 1) {
            index = (index + k) % dataList.size();          //得到应该出局的下标
            dataList.remove(index--);                       //去除元素
        }
        return ((Integer) dataList.get(0)).intValue();   //返回它的值
    }
    //主方法
    public static void main(String[] args) {
        System.out.println("该数字原来的位置是："+cycle(50, 3));
    }
}
```

该题目的结果为：11。

面试题 165　将某个时间以固定格式转化成字符串

时间的表现方式有多种多样，可以只显示年、月、日，可以只显示时分秒，可以用“-”分割年月日数据，可以用冒号“：”分割时、分、秒等。那么 Java 中，应该如何来格式化日期格式呢？本例在回答该问题的同时，详细地讲解 Java 对日期格式的数据的处理方式和 SimpleDateFormat 的使用方法。

【出现频率】 ★★★★

【关键考点】

- Java 的日期数据的存储原理；
- SimpleDateFormat 的使用方法。

【考题分析】

大家知道，日期数据用 java.util.Date 类型来表示，它默认的字符串格式一般不能满足开发需求。但是，面对纷繁乱杂的各种表示日期和时间格式的时候，如果每一个都需要开发者手动的去拼凑字符串来得到的话，是非常麻烦的。那么，JDK 中是否已经有一个比较成熟的表示日期格式的工具类呢？答案就是 java.text.SimpleDateFormat 类。

SimpleDateFormat 是一个以与语言环境有关的方式来格式化和解析日期的具体类。它允许进行格式化（日期 -> 文本）、解析（文本 -> 日期）和规范化。它本身就代表了一种字符格式的时间数据，在创建 SimpleDateFormat 对象的时候，开发者需要提供一种格式，例如下面的代码：

```
new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
```

利用以上的 SimpleDateFormat 对象就可以打印出它所代表格式的时间字符串：“xxx 年-xx 月-xx 日 xx 小时:xx 分钟:xx 秒”，例如下面的代码：

```
2009-11-11 11:11:11
```

以上的结果为：2009 年 11 月 11 日，11 时 11 分 11 秒。那么，这些样式应该如何表示呢？它们的表示有什么特殊的规定吗？其实，SimpleDateFormat 定义了一些比较特殊的表示字符，用来为时间数据作为占位符，如以上代码中的“yyyy”、“HH”等，15.1 列出了一些常用的标示符及其含义。

表 15.1　SimpleDateFormat常用匹配字符含义及使用简介

字　母	日期或时间元素	表　示	示　例
y	年	Year	09,2009
M	年份中的月	Month	July,11,12
w	年份中的周	Number	27
W	月份中的周	Number	2
D	年分中的天	Number	365
d	月份中的天	Number	31
H	一天中的小时（0-23）	Number	23
m	小时中的分钟	Number	59
s	分钟中的秒	Number	59
S	秒中的毫秒	Number	888

针对本题目，首先用特定的格式字符创建一个 SimpleDateFormat 对象，然后调用 SimpleDateFormat 的 format()方法即可，方法的参数即为 Date 型对象，例如下面的代码：

```
//定义字符串的格式
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
String str = sdf.format(date);                    //进行格式化，并得到字符串
```

说明：以上示例代码的格式只是一个简单的展示，读者完全可以根据自己的需要定制不同的日期和时间样式。

【答案】

以下为本题目的参考实现：

```
package ch15;
import java.text.SimpleDateFormat;
import java.util.Date;
public class DateFormat {
    // 主方法
    public static void main(String[] args) {
        Date now = new Date();                          //得到现在的时间
        System.out.println(date2FormatStr(now));//打印现在时间的字符串格式
    }
    // 得到固定字符串格式的方法
    public static String date2FormatStr(Date date) {
        //定义字符串的格式
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        String str = sdf.format(date);                  //进行格式化，并得到字符串
        return str;                                     //返回结果
    }

}
```

面试题 166　用 Java 实现一个冒泡排序算法

排序是一种比较经典的算法考察题目，而冒泡排序是一种常用的易理解的排序算法，也应该算开发者需要具备的基本技能之一。本例在回答该问题的同时，详细地讲解冒泡排

序的原理和实现方式。

【出现频率】 ★★★★

【关键考点】

- 冒泡排序算法；
- 编程实践能力。

【考题分析】

冒泡（Bubble）排序的基本概念是：依次比较相邻的两个数，将小数放在前面，大数放在后面。即首先比较第 1 个和第 2 个数，将小数放前，大数放后。然后比较第 2 个数和第 3 个数，将小数放前，大数放后，如此继续，直至比较最后两个数，将小数放前，大数放后。重复以上过程，仍从第一对数开始比较（因为可能由于第 2 个数和第 3 个数的交换，使得第 1 个数不再小于第 2 个数），将小数放前，大数放后，一直比较到最大数前的一对相邻数，将小数放前，大数放后，第二趟结束，在倒数第 2 个数中得到一个新的最大数。如此下去，直至最终完成排序。由于在排序过程中总是小数往前放，大数往后放，相当于气泡往上升，所以称作冒泡排序。

在编程实现中，一般用二重循环实现，外循环变量设为 i，内循环变量设为 j。外循环重复 n 次，内循环依次重复 n–1，n–2，...，1 次。每次进行比较的两个元素都是与内循环 j 有关的，它们可以分别用 a[j]和 a[j+1]标识，i 的值依次为 1，2，...，n，对于每一个 i，j 的值依次为 1，2，...，n–i，例如下面的代码：

```
for (int i = 0; i < arrys.length; i++) {
for (int j = 0; j < arrys.length - i - 1; j++) {
        if (arrys[j] > arrys[j + 1]) {          //判断当前数字与后面数字的大小
            //把大数放后边
        }
    }
}
```

说明：如果是降序排列，则是把小数放在后面。

【答案】

以下是该题目的编程示例：

```
package ch15;
public class MaoPaoSort {
    // 主方法
    public static void main(String[] args) {
        int[] arr = { 3, 5, 7, 1, 8, 11, 9 };  //定义数组
        maopaoSort(arr);                        //开始排序
    }
    //排序方法
    public static void maopaoSort(int[] arrys) {
        //定义临时变量 temp
        int temp = 0;
        //用 j 为下标，遍历数组
        for (int j = 0; j < arrys.length; j++) {
            //对于每一个数组元素，从 0 到还未来排序的最大下标，总是把最大的数字放
            在后面
            for (int k = 0; k < arrys.length - j - 1; k++) {
                if (arrys[k] > arrys[k + 1]) {//判断当前数字与后面数字的大小
```

```
                    temp = arrys[k];
                    arrys[k] = arrys[k + 1];
                    arrys[k + 1] = temp;                    //用 temp 变量进行换值
                }
            }
        }
        maopaoPrint(arrys);                                 //打印
    }
    //打印方法
    public static void maopaoPrint(int[] before) {
        for (int i = 0; i < before.length; i++) {           //遍历
            System.out.print(before[i] + " ");              //打印，以空格隔开
        }
        System.out.println();                               //换行
    }
}
```

面试题 167　用 Java 实现一个插入排序算法

插入排序是另外一种排序的算法，它是在已经排好序的一个序列中插入数据，并且插入以后依然是排好序的。其实，插入排序有一点像打麻将和打扑克牌，在适合的位置插入数据。本例在回答该问题的同时，详细地讲解插入排序的原理和实现方式。

【出现频率】 ★★★★

【关键考点】

- 插入排序算法；
- 编程实践能力。

【考题分析】

有一个已经有序的数据序列，要求在这个已经排好的数据序列中插入一个数据，但要求插入后此数据序列仍然有序，这个时候就要用到一种新的排序方法：插入排序法。插入排序的基本操作就是将一个数据插入到已经排好序的有序数据中，从而得到一个新的、个数加一的有序数据，该算法比较适用于少量数据的排序。

插入排序算法把要排序的数组分成两部分：第一部分包含了这个数组除了最后一位的所有元素，而第二部分就只包含这一个元素。在第一部分排序后，再把最后这个元素插入到第一部分中的正确位置。

说明： 这里的最后一位，也可以是最前的一位。

在编程实现中，一般用二重循环实现，外循环变量设为 i，内循环变量设为 j。把数组的前面一段排好序，然后把 j 依次减少，直到找到适合数组在 i 的数据，然后插入该数据，并把之后的数字往后移。

【答案】

以下是该题目的编程示例：

```
package ch15;
public class InsertSort {
    // 主方法
    public static void main(String[] args) {
```

```
        int[] arr = { 3, 5, 4, 1, 8, 11, 9 };          //定义数组
        doInsertSort2(arr);                             //开始排序
    }
    //排序方法
    public static void doInsertSort2(int[] src) {
        int len = src.length;                           //获取数组长度
        for (int i = 1; i < len; i++) {                 //遍历数组，从 1 开始
            int j;                                      //定义变量 j
            int temp = src[i];                          //临时存储当前的数字
            for (j = i; j > 0; j--) {                   //遍历 i 之前的数字
                //如果前面的数字大于后面的，则把大的值赋到后边
                if (src[j - 1] > temp) {
                    src[j] = src[j - 1];
                } else
                    //如果当前的数，不小于前面的数，那就说明不小于前面所有的数，
                    //因为前面已经是排好了序的，所以直接通出当前一轮的比较
                    break;
            }
            src[j] = temp;                          //把空缺位置的数字赋值为原有的值
        }
        print(src);                                     //打印
    }
    //打印方法
    public static void print(int[] before) {
        for (int i = 0; i < before.length; i++) {   //遍历
            System.out.print(before[i] + " ");      //打印，以空格隔开
        }
        System.out.println();                           //换行
    }
}
```

面试题 168　用 Java 实现一个快速排序算法

快速排序是排序算法中效率最高的一种，它的理解也比较困难。快速排序是利用递归原理，把数组无限制的分成两个部分，直到所有数据都排好序为止。本例在回答该问题的同时，详细地讲解快速排序的原理和实现方式。

【出现频率】 ★★★★

【关键考点】

- 快速排序算法；
- 编程实践能力。

【考题分析】

快速排序是对冒泡排序的一种改进。它的基本思想是通过一趟排序将要排序的数据分割成独立的两部分，其中一部分的所有数据都比另外一部分的所有数据都要小，然后再按此方法对这两部分数据分别进行快速排序，整个排序过程可以递归进行，以此达到整个数据变成有序序列。

如果要排序的数组是 A[0]……A[N–1]，首先任意选取一个数据（通常选用第一个数据）作为中间数据，然后将所有比它小的数都放到它前面，所有比它大的数都放到它后面，这个过程称为一趟快速排序。一趟快速排序的算法如下所示。

（1）设置两个变量 i、j，排序开始的时候：i=1，j=N-1。

（2）以第一个数组元素作为中间数据，赋值给 pivot，即 pivot=A[0]。

（3）从 j 开始向前搜索，即由后开始向前搜索（j--），找到第一个小于 X 的值。

（4）从 i 开始向后搜索，即由前开始向后搜索（i++），找到第一个大于 X 的值，并与上一步找到的数字交换。

（5）重复第 3、4 步，直到 i>=j。

（6）然后把 j 所在的数字与 pivot 交换。

（7）最后把 j 以前的数组和 j 到最后的数组，再进行递归的快速排序。

【答案】

以下是该题目的编程示例：

```
package ch15;
public class QuickSort {
    // 主方法
    public static void main(String[] args) {
        int[] a = new int[] { 5, 9, 8, 4, 7, 3, 6, 2 };//定义数组
        print(a);                                          //打印之前的顺序
        sort(a, 0, a.length - 1);                          //排序
        print(a);                                          //打印排序后的结果
    }

    // 打印方法
    public static void print(int[] before) {
        for (int i = 0; i < before.length; i++) {          // 遍历
            System.out.print(before[i] + " ");             // 打印，以空格隔开
        }
        System.out.println();                              // 换行
    }
    // 排序方法
    static void sort(int[] a, int low, int high) {
        if (low >= high)                //low 小于或等于 high，则直接返回
            return;
        if ((high - low) == 1) {        //如果只有两个数字，则直接比较
            if (a[0] > a[1])
                swap(a, 0, 1);
            return;
        }
        int pivot = a[low];             // 取第一个数作为中间数
        // 左滑块当前的下标数，从第 2 个数字开始，从最后一个开始
        int left = low + 1;
        int right = high;               // 右滑块当前的下标数;
        while (left < right) {          //左右循环
            //从左边开始找
            while (left < right && left <= high) { //如果左小于右则一直循环
                if (a[left] > pivot)                //找到一个大的数字没有
                    break;
                left++;                             //左下标往右边走一点
            }
            // 从右边开始找
            while (left <= right && right > low) { //如果左大于右则一直循环
                if (a[right] <= pivot)              //找到一个小的数字没有
                    break;
```

```
                right--;                                    //右下标往左走一点
            }
            if (left < right)                               //如果还没找完，则交换数字
                swap(a, right, left);
        }
        swap(a, low, right);                                //交换中间数字
        sort(a, low, right);                                //排序前面数组
        sort(a, right + 1, high);                           //排序后边数组
    }
    // 掉位方法
    private static void swap(int[] array, int i, int j) {
        int temp;
        temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }
}
```

15.2　高级编程试题

开发模式，是前人对一些典型问题的经验总结，是一种智慧的结晶。对开发模式的了解程度也可以看出求职者的开发水平如何。本节将集中讨论有关常用开发模式的常见面试题。

面试题 169　怎样实现 Singleton（单例）模式编程

单例模式开发模式中最简单、最易理解的一种模式。简单地说，它指的就是始终保持一个实例的意思。但是，Java 的类是可以创建多个实例的，这该如何控制呢？本例在回答该问题的同时，详细地讲解单例开发模式的原理和实现方式。

【出现频率】 ★★★★★

【关键考点】

- 单例开发模式的原理；
- Java 实现单例开发模式的两种方式。

【考题分析】

顾名思义，单例模式就是只有一个实例。单例模式确保某一个类只有一个实例，这个类称为单例类。单例模式有 3 个要点：①是某个类只能有一个实例；②是它必须自行创建这个实例；③是它必须自行向整个系统提供这个实例。例如，一些资源管理器常常设计成单例模式。

在计算机系统中，需要管理的资源有很多，例如每台计算机可以有若干个打印机，但只能有一个打印控制器，以避免两个打印作业同时输出到打印机中。每台计算机可以有若干传真卡，但是只应该有一个软件负责管理传真卡，以避免出现两份传真作业同时传到传真卡中的情况。

这些资源管理器构件必须只有一个实例，这是其一；它们必须自行初始化，这是其二；其三，允许整个系统访问自己。因此，它们都满足单例模式的条件，是单例模式的应用。

在 Java 中，需要单例的情况也很多，如数据库连接池，它只能有一个，它需要自行创建并运行其他模块访问它。

Java 中实现单例模式一般需要注意以下几点。

- 私有的构造方法。保证外部无法创建类实例。
- 私有的静态的类型引用。因为静态就可以保证只有一个变量引用。
- 提供获取实例的方法。方法名一般为 getInstance()。

以上 3 点是开发单例模式所必须的。另外，按照实现细节的不同，单例模式通常有两种实现形式：懒汉式和恶汉式。懒汉式是在第一次获取实例的时候才创建对象；而恶汉式则是在类加载时就已经创建对象了。

说明：懒汉式相对来说，更常用一些。

1. 懒汉式

对于懒汉式来说，一般需要在 getInstance()方法中首先判断实例是否为空，也就是第一次访问时候才会进入该 if 语句，然后再返回该实例，例如下面的代码：

```
private static XXXObject obj = null;                    //静态的类型引用
public static XXXObject getInstance(){                  //获取实例方法
    if(obj==null)
        obj=new XXXObject();                            //创建新的
    return obj;
}
```

懒汉式有一个缺点，它可能会线程不安全而无法保证 100%的单例。例如，当线程 A 在以上代码的第 4 行（进入 if 语句以后，创建实例以前）暂停了，此时线程 B 进入了 getInstance()方法，它创建了一个新的实例并返回了，然后线程 A 恢复，它又创建了一个新实例。因此，为了保证懒汉式能做到 100%的单例，还需要为 getInstance()方法加上 synchronized 关键字以保证线程同步，例如下面的代码：

```
public static synchronized XXXObject getInstance(){//获取实例方法，保证同步
    if(obj==null)
        obj=new XXXObject();                            //创建新的
    return obj;
}
```

说明：synchronized 关键字也可以运用在方法体中，用同步代码块的形式来保证线程同步，不一定用同步方法。

2. 恶汉式

对于恶汉式来说，需要做就是把 new 语句写在类型引用变量定义的地方，然后 getInstance()直接返回就可以了，例如下面的代码：

```
private static XXXObject obj = new XXXObject();         //静态的类型引用
public static XXXObject getInstance (){                 //获取实例方法
    return obj;
}
```

恶汉式不存在线程安全的问题，但是它可能会造成资源浪费的情况。因为，实例会在类加载的时候，随着静态变量的初始化而创建，但是有的时候并不会使用该实例，那么它的创建就有一些浪费了，如果该实例比较庞大的话，还可能会影响程序的性能。

总之，懒汉式和恶汉式没有绝对的优劣，主要是根据开发者的具体情况来决定。只不过，在日常开发中懒汉式使用得相对多一些。

【答案】

以下是包含了懒汉式和恶汉式两种单例模式的实现示例：

```
package ooad;
public class SinglObjectParttern {
    //主方法
    public static void main(String[] args) {
        ConnectionPoolA cp=ConnectionPoolA.getConnectionPool(); //创建 1
        ConnectionPoolA cp2=ConnectionPoolA.getConnectionPool();//创建 2
        System.out.println(cp == cp2);                          //打印 true
    }
}
/*写法 1:  恶汉式  单例。优点：实现简单；缺点：在不需要的时候，白创造了对象，造成资源浪费*/
class ConnectionPoolA{
    private static ConnectionPoolA cp=new ConnectionPoolA();//创建实例
    private ConnectionPoolA(){}                             //私有构造方法
    public static ConnectionPoolA getConnectionPool(){
        return cp;
    }
}
/*写法 2:  懒汉式  单例。优点：需要对象时候才创建；缺点：线程不安全*/
class ConnectionPoolB{
    private static ConnectionPoolB cp;
    private ConnectionPoolB(){}                             //私有的构造方法
    //以此类的锁来保证多线程的安全
    public static synchronized ConnectionPoolB getConnectionPool(){
        if(cp==null)
            cp=new ConnectionPoolB();                       //创建新实例
        return cp;
    }
}
```

因为单例模式下，得到的始终是同一个对象，因此运行以上代码的结果如下所示：

```
true
```

面试题 170　怎样实现简单工厂模式编程

Java 的类是可以创建无数个实例对象的，那么如何去管理这些对象呢？如何能够知道创建了多少个这样的对象呢？能不能提供一个统一创建对象的场所呢？这就是工厂。工厂模式就是用来创建对象的设计模式。简单工厂模式是工厂模式中最简单的一种。本例在回答该问题的同时，详细地讲解简单工厂模式的原理和实现方式。

【出现频率】 ★★★★

【关键考点】

- 简单工厂模式的原理；

❑ Java 实现简单工厂模式的方式。

【考题分析】

专门定义一个类来负责创建其他类的实例，被创建的实例通常都具有共同的父类，这称为静态工厂方法模式，属于类的创建型模式。简单工厂模式就属于这一类模式。

简单工厂模式的实质是由一个工厂类根据传入的参数，动态决定应该创建哪一个产品类的实例，并且这些产品类继承自一个父类或接口，该模式中包含的角色有以下几方面：

（1）工厂（Factory）角色。它是简单工厂模式的核心，它负责实现创建所有实例的内部逻辑。工厂类可以被外界直接调用，创建所需的产品对象。

（2）抽象（Product）角色。简单工厂模式所创建的所有对象的父类，它负责描述所有实例所共有的公共接口。例如，轿车和卡车都是车，例如下面的代码：

```
/* 车类 */
abstract class Auto {                                    //汽车的抽象类
    ...
}
class Car extends Auto {                                 //轿车
    ...
}
class Truck extends Auto {                               //卡车
    ...
}
```

（3）具体产品（Concrete Product）角色。是简单工厂模式的创建目标，是一个具体的实例对象，所有创建的对象都是充当这个角色的某个具体类的实例。一般来讲它是抽象产品类的子类，实现了抽象产品类中定义的所有接口方法。

在该模式中，工厂类是整个模式的关键所在。它包含必要的判断逻辑，能够根据外界给定的信息，决定究竟应该创建哪个具体类的对象。这个信息一般是工厂方法的参数，例如下面的代码：

```
public Atuo createAtuo(int type){
    if(type == 1){
        return new Car();                                //创造轿车
    }else{
        return new Truck();                              //创建卡车
    }
}
```

用户在使用时可以直接根据工厂类去创建所需的实例，而无须了解这些对象是如何创建以及如何组织的。有利于整个软件体系结构的优化。但是，简单工厂模式的缺点也正体现在其工厂类上，如果当系统中的具体产品类不断增多时，可能会出现要求工厂类也要做相应的比较大的改动，扩展性并不很好。

【答案】

以下是以汽车为产品的简单工厂模式的实现示例：

```
/* 车类 */
abstract class Auto {                                    //汽车的抽象类
    ...
}
```

```
class Car extends Auto {                          //轿车
    ...
}
class Truck extends Auto {                        //卡车
    ...
}
/* 简单工厂模式 */
class AutoFactory {//简单工厂模式，这个就是工厂类
    public static Auto createAuto(int autoType) {
                                                  //传递不同的参数，获得不同的产品
        if (autoType == 0)
            return new Car();                     //返回轿车实例
        else if (autoType == 1)
            return new Truck();                   //返回卡车实例
        return null;
    }
}
public class Main{
    //主方法
    public static void main(String[] args){
        Auto car = AutoFactory.createAuto(0);     //制造轿车
        Auto truck = AutoFactory.createAuto(1);   //制造卡车
    }
}
```

面试题 171　怎样实现工厂方法模式编程

简单工厂模式是工厂模式中最简单的一种，它在应对产品类别比较多的时候，往往是力不从心，因此就有了工厂方法模式的出现。本例在回答该问题的同时，详细地讲解工厂方法模式的原理和实现方式。

【出现频率】 ★★★★

【关键考点】

- 工厂方法模式的原理；
- Java 实现工厂方法模式的方式。

【考题分析】

工厂方法模式是简单工厂模式的衍生，解决了许多简单工厂模式的问题。完全实现“开闭原则”，易扩展。其次更复杂的层次结构，可以应用于产品结果复杂的场合。

工厂方法模式的对简单工厂模式进行了抽象。有一个抽象的 Factory 类（可以是抽象类或接口），这个类将不再负责具体的产品生产，而是只制定一些规范（也就是声明一些抽象方法），具体的生产工作由其子类去完成。在这个模式中，工厂类和产品类往往可以依次对应。即一个抽象工厂对应一个抽象产品，一个具体工厂对应一个具体产品，这个具体的工厂就负责生产对应的产品。例如，一个生产汽车的工厂，它既可以生产轿车，又可以生产卡车，那就先定义一个抽象的工厂类，然后再衍生出轿车工厂类和卡车工厂类，示例代码如下：

```
abstract class AbstractAutoFacroty{                //抽象汽车工厂
    public abstract Auto createAuto();             //创建一个汽车的抽象方法
}
```

```
class CarFactory extends AbstractAutoFacroty {      //轿车工厂
    public Auto createAuto() {
        return new Car();                            //返回的是轿车
    }
}
class TruckFactory extends AbstractAutoFacroty {     //卡车工厂
    public Auto createAuto() {
        return new Truck();                          //返回卡车
    }
}
```

但需要创建汽车的时候，只需要调用抽象工厂 AbstractAutoFacroty 的 createAuto()方法即可，JVM 会利用多态的原理，创建特定类型的汽车，例如下面的代码：

```
AbstractAutoFacroty factory1 = new CarFactory();    //定义轿车工厂对象
AbstractAutoFacroty factory2 = new TruckFactory(); //定义卡车工厂对象
factory1.createAuto();                              //创造了轿车
factory2.createAuto();                              //创造了卡车
```

说明：在实际开发中，具体工厂对象的给定往往不是直接 new 的，而是通过其他渠道传递过来的，使用 AbstractAutoFacroty 的程序员也不需要知道他具体使用的是哪种工厂。

【答案】

以下是以汽车为产品的工厂方法模式的实现示例：

```
/* 车类 */
abstract class Auto {                               //汽车的抽象类
    ...
}
class Car extends Auto {                            //轿车
    ...
}
class Truck extends Auto {                          //卡车
    ...
}
//抽象汽车工厂
abstract class AbstractAutoFacroty{
    public abstract Auto createAuto();              //创建一个汽车的抽象方法
}
//轿车工厂
class CarFactory extends AbstractAutoFacroty {
    public Auto createAuto() {
        return new Car();                           //返回的是轿车
    }
}
//卡车工厂
class TruckFactory extends AbstractAutoFacroty {
    public Auto createAuto() {
        return new Truck();                         //返回卡车
    }
}
public class Main{
    //主方法
    public static void main(String[] args){
```

```
        AbstractAutoFacroty factory1 = new CarFactory();//定义轿车工厂对象
        AbstractAutoFacroty factory2 = new TruckFactory();
                                                    //定义卡车工厂对象
        factory1.createAuto();                      //创造了轿车
        factory2.createAuto();                      //创造了卡车
    }
}
```

面试题 172 怎样实现抽象工厂方法模式编程

如果抽象角色不止一个的时候，也就是工厂需要造出多种产品（这些产品是有联系的）的时候，简单工厂模式和工厂方法模式就不太应付得了。此时，就需要使用抽象工厂方法模式了，它就是为了专门应对多种抽象产品而设计的。本例在回答该问题的同时，详细地讲解抽象工厂方法模式的原理和实现方式。

【出现频率】 ★★★★

【关键考点】

- 抽象工厂方法模式的原理；
- Java 实现抽象工厂方法模式的方式。

【考题分析】

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。它是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，使用者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，例如，汽车和汽车轮子、卡车和卡车轮子。

说明： 可这样理解，工厂方法模式针对的是一个产品等级结构，而抽象工厂模式针对的是多个产品等级结果。

当每个抽象产品都有多于一个的具体子类的时候，工厂角色怎么知道实例化哪一个子类呢？如每个抽象产品角色都有两个具体产品。其实，抽象工厂模式提供两个具体工厂角色，分别对应于这两个具体产品角色，每一个具体工厂角色只负责某一个产品角色的实例化。每一个具体工厂类只负责创建抽象产品的某一个具体子类的实例。例如，一个抽象的汽车工厂可以生产汽车及其配套的轮子，它的子类工厂是轿车工厂和卡车工厂，轿车工厂生产轿车和轿车轮子，卡车工厂则生产卡车和卡车轮子，示例代码如下：

```
interface CreateWheelAble {                         //可以创造一个轮胎
    public abstract Wheel createWheel();
}
abstract class AbstractAutoFacroty implements CreateWheelAble {
                                                    //抽象汽车工厂
    public abstract Auto createAuto();              //创建一个汽车的抽象方法
    public abstract Wheel createWheel();            //创建一个轮胎的抽象方法
}
class CarFactory extends AbstractAutoFacroty {      //轿车工厂
    public Auto createAuto() {
        return new Car();                           //返回的是轿车
    }
    public Wheel createWheel() {
```

```
        return new CarWheel();                          //返回的是轿车的轮胎
    }
}
class TruckFactory extends AbstractAutoFacroty {    //卡车工厂
    public Auto createAuto() {
        return new Truck();                             //返回卡车
    }
    public Wheel createWheel() {
        return new TruckWheel();                        //返回卡车轮胎
    }
}
```

由于每个具体工厂角色都需要负责两个不同等级结构的产品对象的创建，因此每个工厂角色都需要提供两个工厂方法，分别用于创建两个等级结构的产品。既然每个具体工厂角色都需要实现这两个工厂方法，所以具有一般性，不妨抽象出来，移动到抽象工厂角色中加以声明，例如，以上示例代码中的 CreateWheelAble 接口中的 createWheel()方法。

说明：在实际开发中，抽象工厂往往还会实现多个接口，例如，生成汽车玻璃、刹车、油箱等。

【答案】

以下是以汽车为产品的抽象工厂方法模式的实现示例：

```
/* 车类 */
abstract class Auto {                                   //汽车的抽象类
    ...
}
class Car extends Auto {                                //轿车
    ...
}
class Truck extends Auto {                              //卡车
    ...
}
/* 零件类 */
abstract class Wheel {                                  //抽象的轮胎类
    ...
}

class CarWheel extends Wheel {                          //轿车轮胎
    ...
}

class TruckWheel extends Wheel {                        //卡车轮胎
    ...
}
/*抽象工厂方法模式（可以生成多种不同的产品，*/
/*且这些产品之间有一定的联系，*/
/*关系在于：生成轿车的工厂只能生成轿车的轮胎，不能生成卡车的轮胎）*/
//可以创造一个轮胎接口
interface CreateWheelAble {
    public abstract Wheel createWheel();
}
abstract class AbstractAutoFacroty implements CreateWheelAble {
                                                        //抽象汽车工厂
    public abstract Auto createAuto();                  //创建一个汽车的抽象方法
```

```
    public abstract Wheel createWheel();                    //创建一个轮胎的抽象方法
}
class CarFactory extends AbstractAutoFacroty {              //轿车工厂
    public Auto createAuto() {
        return new Car();                                   //返回的是轿车
    }
    public Wheel createWheel() {
        return new CarWheel();                              //返回的是轿车的轮胎
    }
}
class TruckFactory extends AbstractAutoFacroty {            //卡车工厂
    public Auto createAuto() {
        return new Truck();                                 //返回卡车
    }
    public Wheel createWheel() {
        return new TruckWheel();                            //返回卡车轮胎
    }
}
public class Main{
    //主方法
    public static void main(String[] args){
        /* 抽象工厂方法,创建多个有联系的产品 */
        AbstractAutoFacroty aaf = new CarFactory();//轿车系列产品工厂
        a[6] = aaf.createAuto();                            //生产轿车
        Wheel w = aaf.createWheel();                        //生产轿车轮子
    }
}
```

面试题 173　怎样实现观察者模式编程

观察者模式又称为监听者模式，它广泛的应用在图形化编程中。例如，当用户单击某个按钮的时候，应该做出什么响应；当商场打折的时候，应该通过 E-mail 和电话等形式通知消费者。本例在回答该问题的同时，详细地讲解观察者模式的原理和实现方式。

【出现频率】 ★★★★

【关键考点】

- 观察者模式的原理；
- Java 实现观察者模式的方式。

【考题分析】

观察者模式（Observer）将观察者和被观察的对象分离开。举个例子，当用户单击网页上的某个按钮时，网页做出一些反应行为，例如，弹出一个对话框，那么这个对话框是怎么知道用户是否单击了按钮呢？这是因为，按钮被单击时，浏览器会通知所有观察这个按钮单击事件的观察者来做出相应。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。

观察者模式有很多实现方式，从根本上说，该模式必须包含两个关键角色：观察者和被观察对象。在上述的例子中，按钮是被观察对象，弹出对话框是观察者。观察者和被观察者之间存在“观察”的逻辑关联，当被观察者发生改变的时候，观察者就会观察到这样的变化，并且做出相应的响应。

“观察”不是“直接调用”。实现观察者模式的时候要注意，观察者和被观察对象之间的互动关系不能体现成类之间的直接调用，否则就将使观察者和被观察对象之间紧密的耦合起来，从根本上违反面向对象的设计原则。无论是观察者“观察”观察对象，还是被观察者将自己的改变“通知”观察者，都不应该直接调用。

应该通过一种注册和回调的形式来实现观察者模式。观察者往往是需要实现某个接口的，被观察者保存着这些被注册的观察者的信息（用 List、Set 等集合对象保存这些观察者对象），一旦发生变化则通过多态回调观者的接口方法，从而达到低耦合的目的。

实现观察者模式有很多形式，比较直观的一种是使用“注册、通知、撤销注册”的形式，大致过程如下：

（1）观察者（Observer）将自己注册到被观察对象（Subject）中，被观察对象将观察者存放在一个容器（Container）中。

技巧：为了防止重复注册观察者，一般采用 HashSet 来保存观察者的引用。

观察者们需要实现某一个接口或继承自某个抽象类，便于被观察者统一对待且可以调用某个方法，以便做出相应，例如下面的代码：

```
interface Observer{                                          //观察者
    public void update(Product p);                           //Product 是被观察者
}
class WebObserver implements Observer{
    public void update(Product p){                           //实现回调方法
        System.out.println("更新页面价格: "+p.getName()+":"+p.getPrice());
    }
}
class MailObserver implements Observer{
    public void update(Product p){                           //实现回调方法
        System.out.println("为所有会员发送价格变化信息:"+p.getName()+":"+
        p.getPrice());
    }
}
class Product{                                               //被观察者
    private HashSet<Observer> observers;                     //保存所有的观察者
    ...
}
```

（2）被观察对象发生了某种变化（如按钮被单击、商品价格发生变化等），从容器中得到所有注册过的观察者，将变化通知观察者。在代码中，需要做的就是遍历容器中的观察者，调用它们的回调方法，具体如下所示：

```
public void notifyObserver(){          //通知监听者执行 update()方法
    for(Observer ob:observers){
        ob.update(this);               //回调方法被调用
    }
}
```

注意：以上的代码是直接回调的，其实如果为了更高的性能和效率，可以另开一个线程来回调，这样可以不影响被观察者的其他行为。

（3）观察者告诉被观察者要撤销观察，被观察者从容器中将观察者去除。

为了需要支持撤销观察（或称为取消注册），在观察者接口方法中还需要多一个方法（cancelObserve()或 unreg()），用于撤销观察。在该方法体中，需要做的就是得到被观察者的容器，然后去掉当前的观察者，示例代码如下：

```
interface Observer{                                    //观察者
    public void unreg(Product p);
}
class WebObserver implements Observer{                 //观察者 1
    public void unreg(Product p) {
        p.getObservers().remove(this);                 //撤销观察
    }
}
class MailObserver implements Observer{                //观察者 2
    public void unreg(Product p) {
        p.getObservers().remove(this);                 //撤销观察
    }
}
```

【答案】

以下是一个观察者模式实现示例。当商品价格发生变化时，通知网页观察者修改网页数据和并通知 E-mail 观察者发送 E-mail。

```
package ch15;
import java.util.*;
public class ObserverPattern {
    //主方法
    public static void main(String[] args) {
        Product p=new Product("《java 核心技术》",103.00);//创建一个商品
        Observer o1 = new WebObserver();                   //第一个观察者
        Observer o2 = new MailObserver();                  //第二个观察者
        p.addObserver(o1);                                 //注册观察者 1
        p.addObserver(o2);                                 //注册观察者 2
        System.out.println("===第一次价格改动===");
        p.setPrice(80);                                    //修改商品价格
        o1.unreg(p);                                       //观察者 1 取消观察
        System.out.println("===第二次价格改动===");
        p.setPrice(100);                                   //再次修改价格
    }
}
interface Observer{                                        //观察者接口
    public void update(Product p);                         //价格修改的接口方法
    public void unreg(Product p);                          //撤销注册
}
class WebObserver implements Observer{                     //Web 观察者
    public void update(Product p){                         //定义回调方法
        System.out.println("更新页面价格: "+p.getName()+":"+p.getPrice());
    }
    public void unreg(Product p) {
        p.getObservers().remove(this);                     //去掉本观察者
    }
}
class MailObserver implements Observer{                    //E-mail 观察者
    public void update(Product p){                         //定义回调方法
        System.out.println("为所有会员发送价格变化信息:"+p.getName()+":"+
        p.getPrice());
```

```
    }
    public void unreg(Product p) {
        p.getObservers().remove(this);          //去掉本观察者
    }
}
//产品类
class Product{                                  //被观察者
    private double price;                       //价格
    private String name;                        //商品名称
    private HashSet<Observer> observers;        //保存所有的观察者
    //构造方法
    public Product(String name,double price){
        this.price=price;
        this.name=name;
        observers=new HashSet<Observer>();
    }
    public void addObserver(Observer ob){       //添加观察者
        observers.add(ob);
    }
    //通知监听者执行 update()方法
    public void notifyObserver(){
        for(Observer ob:observers){
            ob.update(this);                    //回调所有观察者的观察方法
        }
    }
    public double getPrice() {
        return price;
    }
    public void setPrice(double price) {
        this.price = price
        //当价格被改变的时候就通知观察者;
        notifyObserver();
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public HashSet<Observer> getObservers() {
        return observers;
    }
    public void setObservers(HashSet<Observer> observers) {
        this.observers = observers;
    }
}
```

以上代码中，设定了两个观察者，商品发生了两次价格变化。第一次价格变化时，两个观察者都做出了响应；第二次价格变化时，只有观察者 2 在进行观察。因此，示例代码的运行结果如下所示：

```
===第一次价格改动===
为所有会员发送价格变化信息：《Java 核心技术》:80.0
更新页面价格：《Java 核心技术》:80.0
===第二次价格改动===
为所有会员发送价格变化信息：《Java 核心技术》:100.0
```

面试题 174　用 Java 实现一个链表类

链表是一种数据结构，它的应用非常广泛。java.util.List 就是一种链表数据结构的接口，java.util.ArrayList 和 LinkedList 是链表的实现类。链表的特征在于，它的数据是用节点（Node）连接起来的，可以通过首节点获取到任何一个位置上的节点。那么如何自定义个链表的实现类呢？本例在回答该问题的同时，详细地讲解链表数据结构的原理和 Java 实现链表的思路。

【出现频率】 ★★★★

【关键考点】

- 链表数据结构的含义；
- 编程实践能力。

【考题分析】

链表是一种物理存储单元上非连续、非顺序的存储结构，数据节点的逻辑顺序是通过链表中的指针（或引用）链接次序实现的。链表由一系列节点（Node，链表中每一个数据元素称为节点）组成，节点可以在运行时动态生成。每个节点包括两个部分：一个是存储数据元素的数据域，另一个是存储下一个节点地址的指针域。与数组相比较，链表比较方便插入和删除操作。

其中存储数据元素信息的域称作数据域（设域名为 data），存储直接后继存储位置的域称为指针域（设域名为 next）。由 N 个节点依次链接构成的链表，称为线性表的链式存储表示，由于此类链表的每个节点中只包含一个指针域，故又称单链表或线性链表。

说明： 一般说的链表都指的是单向链表，只能从一头到另一头，不能反过来。如果是可以从两个方向遍历的，则称为双向链表。

对于 Java 语言来说，为了表示一个节点，可以创建一个 Node 类来表示，它包含两个成员变量：data 和 next。用 Object 类型来表示数据域，用 Node 类型表示指针域。另外，因为每个节点肯定是需要包含数据，因此 Node 可以有一个传入数据参数的构造方法。Node 示例代码如下所示：

```
class Node {                          //内部类，节点
    Object data;                      //数据
    Node next;                        //下一个节点的引用
    public Node(Object data) {        //构造方法，为 data 赋值
        super();
        this.data = data;
        this.next = null;             //next 为空
    }
}
```

这是一个比较典型的节点类。对于链表类的设计来说，一般需要创建一个 Node 类型的 head 变量，它始终表示链表的头一个节点，尽管它有时等于 null。有了这个 head 成员变量以后，链表的一些接口方法就比较容易实现了。例如，判断链表是否为空，只需要判断 head 是否等于 null 即可；获取链表节点的个数，只需要从 head 依次往下走，直到最后

一个节点的 next 为 null。

一个链表类是需要提供给其他的代码使用的，因此它需要提供一些接口方法。常用的方法包括：清空（clear）、插入数据（insert）、节点数量（size）、得到指定位置的节点或数据（get）等。其中，insert()方法是相对比较困难的一个方法，在链表插入一个节点可能存在 3 种情况：在链表头插入、在链表中间插入和在链表末尾插入。当在末尾插入的时候，只需要把组后一个节点的 next 指向新节点即可；当从链表头插入的时候，需要把 head 指向它，并把新节点的 next 指向原来 head 指向的节点；当从链表中间插入的时候，则把原有位置的前一个节点的 next 指向新节点，新节点的 next 指向原来位置上的节点。

技巧：链表的关键点在于它对每个节点的 next 指针（或引用），不论是插入、删除或修改操作，都是去改变某个或某些节点的 next 的指向。

【答案】

以下是该题目的编程示例：

```
package ch15;

public class MyList {
    private static class Node {          //内部类，节点
        Object data;                     //数据
        Node next;                       //下一个节点的引用
        public Node(Object data) {       //构造方法，为 data 赋值
            super();
            this.data = data;
            this.next = null;
        }
    }
    Node head;                           //头节点
    public MyList() {                    //链表的构造方法
        head = null;
    }
    public void clear() {                //清除链表
        head = null;
    }
    public void travel() {               //遍历打印链表
        Node p = head;
        while (p != null) {
            System.out.println(p.data);
            p = p.next;
        }
    }

    public boolean isEmpty() {           //判断是否为空
        return head == null;
    }
    public int size() {                  //得到节点的个数
        Node p = head;
        int sum = 0;
        while (p != null) {
            sum++;
            p = p.next;
        }
        return sum;
    }
```

```
    //在指定位置插入元素，下标从 0 开始
    public void insert(Object d, int pos) {
        if (pos < 0 || pos > size()) {          //下标异常
            throw new RuntimeException("下标出错");
        }
        Node newNode = new Node(d);
        if (pos == 0) {                         //从第一个位置插入节点
            newNode.next = head;                //把新节点的
            head = newNode;
        } else if (pos >= size() - 1) {         //从最后的位置上插入
            get(size() - 1).next = newNode;//把最后一个节点的 next 指向新节点
        } else {
            newNode.next = get(pos);    //把新节点的 next 指向 pos 位置的节点
            get(pos - 1).next = newNode;    //把 pos-1 节点的 next 指向新节点
        }
    }
    //获取特定位置上的节点
    public Node get(int pos) {
        if (pos < 0 || pos > size()) {          //下标异常
            throw new RuntimeException("下标出错");
        }
        if (pos == 0)                           //如果为 0，直接返回 head
            return head;
        Node p = head;                          //从 head 开始，顺摸到指定位置
        for (int i = 0; i < pos; i++)
            p = p.next;
        return p;
    }
    //测试主方法
    public static void main(String[] args) {
        MyList list = new MyList();             //创建 List 对象
        list.insert(10, 0);                     //在 0 位置插入元素
        list.insert(20, 1);                     //在 1 位置插入元素
        list.insert(30, 0);                     //在 0 位置插入元素
        list.insert(40, 1);                     //在 1 位置插入元素
        list.travel();                          //遍历打印链表
    }
}
```

在主方法中，首先在“0”位置插入 10，然后又在“1”位置插入了 20，因此，此时的链表中就依次包含了 10 和 20。然后，又往“0”位置插入了 30，它就排在了 10 的前面，此时的链表的数据就是 30、10 和 20。最后，在“1”位置插入了 40，它在 30 和 10 之间，因此最终的链表遍历打印结果如下：

```
30
40
10
20
```

15.3　小　　结

本章讲解了一些关于 Java 算法和开始模式的面试题。首先讲解了一些基础的算法题，

这些题目不会使用太复杂的 API，而主要考察的是求职者的编程动手能力，然后讲解了 Java 常用的几类开发模式的面试题，包括：单例模式、工厂类模式、观察者模式等。算法是程序的灵魂，是程序中真正有价值的东西，读者应该在发现问题和解决问题的过程中不断改进算法，提升自己在算法和数据结构等方面的能力。对于开发模式，应该知其然而知其所以然，并且还需要灵活的运用到自己的开发实践中，这样才能真正掌握开发模式的知识，应对各种各样的面试题。

第6篇 情商和智商经典面试题

第 16 章　情商类面试题

尽管软件开发是一项技术性的工作，但是在面试过程中，少不了一些情商类的面试题。这些面试题并不是想考察求职者的技术水平，而是通过这些问题来看出求职者的心态以及分析问题和解决问题的能力。本章将包含关于 IT 面试中一些常见的情商类面试题，并且分析这些题目，帮助读者如何根据自己的情况来适当的回答这些问题。

16.1　应届毕业生问题应答

应届毕业生一般没有什么工作经验，面试者更多的是想考察应届毕业生的志向和理想，以及通过在校期间的一些活动来得知毕业生的工作性格。本节将集中讨论有关应届毕业生常遇见的面试题。

面试题 175　你有暑期打工的经历吗？是怎样找到的

一般来说，刚毕业的学生是没有什么工作经验的，面试者想通过该题目了解毕业生什么情况呢？如果有暑假打工经历，应该如何回答？若没有又该怎么回答呢？本例将分析本题目考察意图，帮助求职者根据自身的情况回答该题目。

【出现频率】 ★★★★

【关键考点】

- 描述事物的能力；
- 对职业规划的态度。

【考题分析】

招聘单位有一个普遍的招聘意向，就是希望招聘的每一个人都可以直接上手，熟悉工作流程，可以很容易的与他人相处。大量的事实证明，大多数用人单位对有工作经历的应届毕业生都一致看好，无论参与过什么样的工作。

读书时期有过工作经历的毕业生更容易与人相处，他们会更好地安排时间，更务实而且更成熟，保持更多的共同语言。这是因为，他们在打工期间会与各种各样的人打交道，为了让自己的工作做得更好，他们需要学会与他人沟通，保持较高的工作效率。企业就是要创造利润，个人要更有效率地工作，遵章守纪，尽全力完成工作。简而言之，无论你的暑假工作多么微不足道，都要将其视为一段在企业的工作经历。

对于其他一些具体的问题而言，例如，做些什么事情、如何找到该工作的等问题，面试官的提问是为了理想地反映出应届毕业生的主动性、创造性与灵活性。例如，求职者说他是如何的通过一些艰辛的过程找到这份暑期工，并且还是受了很多委屈，但是却把这次

工作得很好，就可以看出该求职者是具有恒心的一个人，吃得苦，是一个实干型的人。

如果求职者实在是没有什么暑期工作经历，也可以把在校期间的一些活动描述一下。例如，某求职者主持过某项学校集体活动，获得了如何如何的成功。因为没有暑期工作经历的毕业生，可能会把更多的经历放在学习和研究上，面试官也会看好这类人才的。

【答案】

如果有暑期工作经历的应届毕业生直接回答即可。在描述的过程中，关键在于把自己的一些不同点描述出来，例如，自己遇到的问题及坚持。

若没有暑期工作经历的应届毕业生，可以把自己在校期间参加的活动描述一下，或者强调自己的学习和学术研究性。

面试题 176　你认为你的第一份工作能干多久

应届毕业生接触社会的时间较少，他们刚出校门的时候还无法体会社会的残酷性，经常跳槽是他们的通病。应届毕业生应该如何应该这样的题目呢？回答少了会让面试官对自己看轻，回答多了则可信度不大。本例将分析本题目考察意图，帮助求职者根据自身的情况回答该题目。

【出现频率】 ★★★★

【关键考点】

- ❑ 描述事物的能力；
- ❑ 对职业规划的态度。

【考题分析】

对于一些大公司来说，应届毕业生是一个宝。因为他们没有被社会的浑浊所侵蚀，公司可以按照自己的意愿来培养他们，让他们随着公司一起成长，创造更多的价值。但是，应届毕业生也是一个不安分的群体，他们可能会因为一些比较敏感的问题而随便辞职。

面试官提问该题目主要是想了解求职者的初步想法是什么，他是否有责任心。如果求职者回答得很短，则面试官不会有什么好印象。如果求职者回答了一个很长的时间，例如 5 年或 10 年，面试官会觉着很不现实，求职者根本没有为它们的职业规划和第一份工作的计划考虑过。而事实上，很多应届毕业生也不能确定他们的第一份工作能干多久。

针对这样的题目，求职者应该搞清楚，面试官到底想要知道毕业生什么呢？其实，面试官主要是想知道一个刚毕业的学生的兴趣是否在企业所在的行业，如果求职者能够描述出他会如何的计划和安排自己第一份工作的事情，那么就可以看出他是兴趣所向，有了兴趣就会有动力，待的时间也就更长了。

【答案】

该题目的主要目的是想看求职者是否对所从事的行业有兴趣。求职者在面对这样的问题的时候，不能只回答一个简单的时长，而应该认真的说出自己的兴趣，以及如何计划自己第一份工作。这样才能让面试官充分的了解求职者，让他对自己保持信心。

面试题 177　除了本公司，你还应聘了其他哪些公司呢

广发简历是应届毕业生经常干的事情，那么这些简历的分布特点意味着什么呢？面试

官通过求职者的求职经历可以看出什么来呢？本例将分析本题目考察意图，帮助求职者根据自身的情况回答该题目。

【出现频率】 ★★★★

【关键考点】

- 描述事物的能力；
- 对职业规划的态度。

【考题分析】

每个人的求职都有一个特定的求职意向。这些意向指的就是求职者倾向于做什么技术的工作，倾向于哪个或哪些行业，倾向于国企还是外企，倾向于哪些地域等。本题目想考察的就是求职者的求职意向。

一些应届毕业生刚走出校门的时候，并没有一个很明确的求职目标，就好像屋头苍蝇一样乱撞。招聘单位在面对这样的求职者的时候，一般都是不予以考虑的。那么如何知道求职者是否已经有一个比较明确的方向呢？最好的方法就是提问本题目。

如果一个应届毕业生应聘过与公司同行业、同地域、文化类似的其他公司，证明该求职者是做过充分市场考察的，而且就是冲这个方向而来。这样的应届毕业生一般都具有明确的职业规划，比较容易得到招聘单位的青睐。

另外，通过比较不同的公司，面试者应该了解它们之间的不同之处，并且能够指出它们各自的优缺点。单一的指出其他公司不足是不恰当的，面试官更希望看到求职者对它们彼此之间的差异有深入的了解。

【答案】

本题目的考察意图在于求职者的求职意向，看该应届毕业生是否对自己的比较明确的职业规划和求职目的。当应届毕业生在面试过程中遇到该问题的时候，即使自己面试过很多的公司，也只需要要把与当前面试的这家公司的情况类似的公司讲一下。并且，能够指出它们各自的优缺点和差异性。

面试题 178　你如何看待公司没有足够的培训课程

越来越多的应届毕业生在选择自己第一份工作的时候，往往工资水平并不是决定性因素，而是公司的文化和培训机制，这是值得倡导的。但是，如果面试的时候，面试官告诉你公司并没有太多的培训，那么应该如何看待和应对呢？本例将分析本题目考察意图，帮助求职者根据自身的情况回答该题目。

【出现频率】 ★★★★

【关键考点】

- 描述事物的能力；
- 对职业规划的态度。

【考题分析】

首先，应届毕业生应该明确一点：培训和学习并不只限于专门的课程。培训的目的在于提升自己，让自己获得更多的知识。但是培训和学习其实无处不在的，例如，每写一段代码就可以学习一个新的思维或模式，每进行一次测试就熟悉了业务逻辑等。如果太看重公司提供的专门的培训，往往会让面试官反感。

有些应届毕业生会对面试官提问一些问题，包括公司培训机制如何，我进公司以后是否能先培训再上岗呢？老板不是各个都是慈善家，它们更希望的是你能够直接上手做事情，大多数的公司都是只有在万不得已的情况下才会去培训员工，目的是为他创造更多的价值，而不是真正的为你考虑。

另外，一些公司的培训是需要昂贵的费用的，如果公司把你培训完以后你就离职的话，公司会蒙受巨大损失。这个时候公司会要求你做出大额的赔偿，很多求职者面对这样的情况，往往会整得很尴尬。

其实，分析一下本题目就不难发现，公司通过这样的问题只是想了解到求职者是否有进取学习的动力，而不是总是依赖于公司。如果一个人总是有一颗上进求学的心，它的思维是动态的，能为公司创造出更多的主意来。相反，如果一个人总是吃老本，只把自己的进步寄托在公司培训上边的话，它很难被企业看重。

话说回来，公司如果发现你是一个可造之才，肯定会去尽量挽留你，提供充分的培训机制。因此，应届毕业生应该把的自己的心态做一下调整，自己不再是一个学生，而是一名生产者，是要去创造价值的。

【答案】

该题目主要考察的是应届毕业生目前的求职心态，是还停留在学校学习的状态呢？还是已经为工作做好了充分的心理准备。应届毕业生在面对这样地问题的时候，首先需要阐明自己，已经是一个工作者的态度，需要将自己的知识转换为财富，而不是一心只想到单纯的培训和学习，然后可以说自己会在工作的过程中学习，对待公司单独的培训是一种选项，有最好，没有则可以通过其他方式补充。

16.2　常规问题应答

对于非应届毕业生的求职者来说，他们已经有一定的社会经验，可能是从另外一家公司跳槽而来，他们所面对的面试题往往与应届毕业生的有所不同。本节将集中讨论有关另外的一些常见面试题。

面试题 179　简要介绍你自己

自我介绍，往往是一次面试的开端，也是给面试官留下好映像的关键点。有的时候，自我介绍的成功与否直接决定了此次面试的结果，足以看出自我介绍的重要性。那么，应该如何来进行自我介绍呢？本例将分析本题目考察意图，帮助求职者根据自身的情况回答该题目。

【出现频率】 ★★★★★

【关键考点】

- ❑ 自我认识的能力；
- ❑ 性能特征是否符合招聘单位文化。

【考题分析】

当招聘一个职位的时候，面试官往往会面对多个求职者，如何快速的认识他们，自我

介绍是一个非常好的手段，自我介绍也就慢慢的拉开了面试的序幕。这里的自我介绍肯定不能长篇大论的说，也不能漫无目的的讲，应该注意一些技巧，在让面试官了解求职者背景的同时，又对求职者产生好的映像。

俗话说：“知己知彼，百战不殆”。面试之前的准备工作是很重要的，了解这家公司是干什么的，它在行业中的地位如何，在业界有什么样的口碑，企业文化是什么样的。这些背景知识的了解可以直接引导求职者根据自己的情况进行回答，也就是双方都得看上眼才能进行下一步。

例如，如果求职者面试的是一家外企，对英语的要求比较高，求职者可以把自己关于英语的话题说一说，让面试官感觉你是有一定英文经历的；如果招聘单位是一家经常加班的公司，那么求职者就可以多说一说自己曾经如何努力的工作，如何的加班，并且成功完成任务，取得了比较好的效果。

既然称为自我介绍，那么就一定不能少了关于求职者的一些基本信息，如姓名、年龄、学历、专业等。但是，也不能说的太详细了，至于身高体重那些无关紧要的信息就可以不用说了，点到为止，让面试官大概了解一下，有一个映像即可。

把基本信息介绍完以后，就可以说说自己擅长的一些东西了。这些擅长的东西最好是能跟招聘的职位有关系，例如，求职者面试的是一个 Java 开发的职位，那么就可以说一下自己在 Java 开发领域中所获得的成就，取得了 XXX 证书，做了哪些 Java 的项目，使用哪些框架技术等。

面试官如果能从自我介绍中就了解求职者的性格特征，那是最好的了，但是短短的几分钟是很难达到这种效果的。一般来说，求职者在进行面试之前，都应该准备好一些自我评价，但是这些自我评价并不是每次面试都应该使用，还得根据招聘单位的文化背景因素来决定。通常情况下，外向、活泼、学习能力强、容易相处、能够承受较大工作压力的评价，是比较容易受到青睐的。

最后，不管是什么招聘公司，都应该准备一份英文自我评价，毕竟 IT 使用了许多英文。英文的自我评价往往不是为了看求职者的英文口语有多强，而是想知道求职者是否有说英语的自信。

【答案】

自我介绍是一次面试的开端，它说容易也容易，说难也难。自我介绍无非就是把自己的情况做以下简短的介绍，让面试官对自己有一个大致的了解。但是，自我介绍是求职者第一次把自己暴露给面试官，稍微不注意就很可能把自己的不足之处或与公司文化相悖的一些性格特征显露出来，给面试官留下不好的映像。一般来说，在进行自我介绍时需要注意以下几点：

（1）在面试之前就对招聘单位做充分的了解，在进行自我介绍的时候应该避开一些可能会带来不便的话题。

（2）基本信息必不可少，但是不能太多。

（3）应该把话题尽量往自己擅长的方面靠拢。

（4）在自我介绍的同时，最好能有一个对自己的评价或总结。

（5）准备好一份英文的自我介绍。

面试题 180　你在上一家公司的离职原因是什么

这是一道非常普遍的面试题目，几乎所有的招聘单位都会提问到这个问题。这也是一个非常敏感的问题，对于 IT 行业来说，跳槽的原因多半就是因为报酬问题，但是如果直接回答是因为上家公司的待遇太低行不行呢？本例将分析本题目考察意图，帮助求职者根据自身的情况回答该题目。

【出现频率】 ★★★★★

【关键考点】

- 描述事物的能力；
- 对职业规划的态度。

【考题分析】

对于一家招聘单位来说，是非常不想看到自己刚招聘进来的员工刚一上手就离职了。如果求职者是一个经常变换工作的人，招聘单位肯定是不看好的，公司不希望求职者上次的离职原因也成为他在的本公司的离职原因。

本来这样的问题就比较敏感，如果求职者把离职原因说得很详细的话，面试官可能会从中看到很多问题，言多必失嘛。但是，不能回避该问题，如果求职者说只是想换一个环境而已，那么面试官会觉得该求职者没有太多的诚意。

如果上次离职就因为个人的一些负面原因，如“不诚实、懒惰、缺乏责任感、不随和”等，这些也不能说得太明显，不然面试官会直接把你看扁。另外，即使不是自己的原因，也不能是自己一些主观意识上的认为说出来，如“太辛苦、管理太混乱、公司排斥我们某某员工”等，这些问题可能只是你一个人的观点。每家公司都不是全能的，每家公司都会有一些问题，那么是不是上次你觉得这样，这次你还是会这样认为呢。

如果求职者的回答能让自己增光添彩的话，是最好不过的了。如“本来我很忠诚，工作也兢兢业业，但是金融危机，公司业务缩微而破产或裁员，我无奈只能另寻能发挥我能力的舞台”，这样的回答也能为自己加分不少。

【答案】

本面试题的答案并不是在任何场合都有效，关键在于对面试的具体情况进行把握，有意思地揣摩面试官的心理想法和公司本身的背景，然后做出符合其思想意识的回答。以下为一些回答提示思路：

（1）最重要的是求职者要让招聘单位相信，在以前单位的离职原因在此家单位中不存在。

（2）避免把离职原因说的太详细、太具体。

（3）不能要掺杂主观的负面感受，如“太辛苦、管理太混乱、公司排斥我们某某员工”等。

（4）也不能回避、躲避，如“想换换环境”等。

（5）不能把自己的负面原因说得太明显，如“不诚实、懒惰、缺乏责任感、不随和”等。

（6）尽量使自己解释的理由使求职者个人形象添彩。

以下是一段参考回答：“我离职主要是因为公司倒闭。我在这家公司工作了 3 年多，

有较深的感情。从去年开始，由于金融危机的影响，公司的业务急剧萎缩，濒临破产，我没有办法，只能面对现实，重新寻找能发挥我能力的舞台。”

面试题 181　你了解本公司吗？为什么要选择本公司

在往某一家公司投简历的时候，首先需要了解的是该公司是干什么的？它在行业里的实力情况如何？是否符合自己的兴趣？如果在没有了解公司的背景的情况下就来面试，是非常不合适的。本例将分析本题目考察意图，帮助求职者根据自身的情况回答该题目。

【出现频率】 ★★★★

【关键考点】

- ❑ 描述事物的能力；
- ❑ 对职业规划的态度。

【考题分析】

古人说：“知己知彼，百战不殆”，这在面试的过程中也是用得着的。求职者需要了解招聘单位，招聘单位也需要了解求职者。求职者了解招聘单位的情况是需要一个过程的，但是，面试官在面试求职者的时候，往往会假设该求职者是在充分了解了本公司的背景以后而来的。

了解一家招聘单位最简单的方法就是通过上网查资料，在一些论坛或门户网站上得到关于它的正面和负面的信息。对于正面信息，求职者应该看该公司的业务是否符合自己的想法，例如，求职者比较熟悉电信行业，同时又想从事电信行业，刚好这家公司又是电信行业的领头，这就比较合适了。对于负面信息，求职者应该适当的看待，对比着自身的情况，看看自己能否克服这些负面的东西，如果实在受不了，求职者就可以不再考虑这家公司。另外，任何的网上的负面信息都有夸大的成分，例如，有人在论坛上说该公司看不起员工，那么是不是每个员工都被看不起呢？肯定不是的，此时就需要求职者认真的分析和比较了。

确定自己的兴趣和该公司的业务比较合适的时候，就可以去面试了。如果求职者在被问到该问题的时候，可以遵循以下几点思路进行回答：

（1）大致描述以下该公司的业务内容是什么。

（2）该公司在行业中的规模和经济情况，例如，排名多少。

（3）描述一下自己认为该公司的特别之处，相对于其他竞争对手，有哪些优势，例如：薪酬、发展潜力、独特的文化氛围。

（4）自己的兴趣与该公司的吻合之处。

（5）最后还应该说一下自己能够为公司带来哪些价值。

【答案】

首先，求职者一定要了解一下这家公司的各种情况，包括业务、行业、竞争力、发展潜力等。在被问到该问题的时候，介绍一下这家公司在行业内的实力，规模，潜力，薪酬等，所有有竞争力的吸引人的东西都可以谈一谈。然后可以谈谈公司的企业文化，跟求职者的原则或兴趣很契合，最后说一下求职者的加入能够为公司带去哪些的收益，包括创造价值、提高效率等。

面试题 182　你如何看待加班问题的

对于软件开发行业来说，尽管不能说加班是家常便饭，但也是屡见不鲜的情况。对于任何人来说，包括老板在内，都是不愿意加班的，加班更多的时候是一种无奈的情况。那么求职者应该如何看待加班问题呢？本例将分析本题目考察意图，帮助求职者根据自身的情况回答该题目。

【出现频率】 ★★★★

【关键考点】

- 描述事物的能力；
- 对职业规划的态度。

【考题分析】

IT 行业的技术工作都有一个特点，就是它往往不是按照时间长度来计量一个员工的工作情况，而是以项目为单位来进行评估。其他的工种会比较要求员工准时上下班，用时间来考察一个员工的工作情况，例如，文员、行政人员。但是对于开发人员来说，上级布置下来的东西就是一个特定的任务，不管技术人员怎么来安排时间，必须哪天以前完成。因此，这种时间的灵活性就往往会造成加班。

造成加班的原因的又很多种，例如，上级没有安排足够的时间，或自身工作效率不够，只能靠加班来完成。如果是因为上级考虑欠妥的情况而造成加班的话，这无可厚非。但是如果是自身原因而加班的话，就需要求职者考虑一下如何去提升自己了。

加班是劳动者用自身的时间来为老板工作，创造价值。因此，加班应该得到补偿。补偿的方式首选就是加班费，国家规定，周末和加班的加班费是平时的两倍，节假日则是平时的 3 倍。这一点求职者是需要明确的，这也是求职者的权利和与招聘单位谈判的重要砝码。另外，有的公司的加班是以调休的形式来补偿给员工的。

在面对这样的问题的时候，应该摆出自己的观点，然后与招聘单位谈判加班的补偿问题，以下是几点答题思路：

（1）正确的对待加班问题。加班有的时候并不是坏事，它是软件开发行业的工作特点所导致的。

（2）摆出自己加班也会勤勤恳恳的态度。即使是加班，也像平时工作那样勤恳。

（3）说明自己加班的补偿问题。加班就必须得到补偿，如果求职者愿意用调休的方式来补偿也可以，用加班费的形式来补偿也可以。

（4）最后，可以了解一下公司加班的情况如何。公司是经常加班吗，还是偶尔加班，求职者此时还可以根据自身的情况，决定是否加入该公司。

【答案】

首先，求职者一定要明确一点：加班无法避免。在被问到该问题的时候，首先是承诺自己肯定会认真的加班工作的，但是也一定要有加班的补偿，加班费或调休。然后可以反问一下面试官，公司的加班情况如何，如果实在无法忍受公司没日没夜的加班，最好还是不要选择该公司为好。

面试题 183　自己的最大优缺点是什么

人无完人，每个人都有优点和缺点，每个人都应该了解自己的优缺点。如果无法说出自己的优缺点，那么这个人是不了解自己的，连自己都不了解，那么又如何去了解和评价他人呢，又如何与他们进行交流呢？本例将分析本题目考察意图，帮助求职者根据自身的情况回答该题目。

【出现频率】 ★★★★

【关键考点】

- 描述事物的能力；
- 对职业规划的态度。

【考题分析】

“人贵有自知之明”，一个人对自己的看法能不能体现出客观性和全面性，是素质能力体现的一个重要因素。客观性要求自我评价不能具有任何的主观意识，全面性要求评价需要包括正面和负面两个方面。

在对自己的评价中，首先要充分肯定自己，认为自己是独特的，在自己擅长的领域中是有一席之地的，这样可以充分突出自己的竞争能力和竞争优势，也给面试官一个坚定、自信的良好印象。但是，在对自己评价的表述中要真实，不能虚假。

在回答自己最大的优点的时候，应该是在自己最擅长的领域，这样既能把自己的优点说出来，又能让面试官知道自己在该领域是一个擅长的领域。

人不可能没有缺点，在谈及缺点的时候要概括集中，不要出现过多的缺点描述，过多的否定自己，明明是谦虚，也会影响考官的判断。谈及缺点时，除了说明现已清醒地认识到了不足，要特别着重表明有改变缺点的信心和方法。

不论是优点，还是缺点的自我评价中，不要泛泛而谈，最好用事实说明问题。该题目没有标准的回答，以下是该题目的回答思路：

（1）充分肯定自己。自己是有优点的，而且不止一个，最大的优点就在于自己最擅长的领域。例如，一次 Java 项目开发中，我攻克了前辈员工都没有完成的一个难题，为公司带来了巨大的价值。

（2）肯定自己有缺点但缺点不多。一定不能说自己太多的缺点，这样很可能与自己的优点相矛盾，最大的缺点最好是在与公司业务无关的领域中。

（3）标明自己发扬优点和改正缺点的决心。要让面试官觉得求职者不仅有优点，而且愿意继续发扬和提高；有缺点，但是会积极的改正。

【答案】

求职者一定不能避免该问题，“人非圣贤，孰能无过”，关键在于能够正确的认识自己。在描述自己优点的时候，最好能举自己擅长的领域中的例子，让开发者既了解自己的优点，又知道了自己擅长的领域。对于缺点，不能说得太多，只需要把缺点描述清楚就可以了，同时还可以表明有改变缺点的信心和方法。

面试题 184　你希望的待遇为多少

这是几乎每次面试都会问到的题目，说白了就是指求职者价值几何。同时，这也是一

个非常敏感的问题，求职者肯定是希望越多越好，如果说少了又对不起自己，如果说多了又怕面试失败。应该如何回答呢？本例将分析本题目考察意图，帮助求职者根据自身的情况回答该题目。

【出现频率】 ★★★★

【关键考点】

- 描述事物的能力；
- 对职业规划的态度。

【考题分析】

求职者应该明确一点，如果面试官已经问到这一步的话，则说明招聘单位已经有意要聘用你了。这就相当于菜市场买菜，最后开始讨价还价的阶段到了。

在回答该问题的时候，求职者应该在心理有一个最低价，也就是最低不能低过多少钱。这个价位需要进行充分的调查，结合当地的人才市场情况来定。另外，求职者在面试这家公司的时候，也可以进行充分的打听和了解，他们到底能给多少。如果求职者感觉到招聘单位非常想招聘自己的话，也可以把价格提高一些。

如果求职者自我觉得面试得还不错，自我膨胀就很危险了。此时很多求职者会提出很高的要价，这样会让面试官难以承受的，而最终导致面试失败。因此，最好求职者根据自己的底价给出一个比底价略高的价格范围来。如果招聘单位认为你是个人才的话，肯定是用重金挽留住你的。

待遇不仅仅是工资的问题，还有社保、公积金、出差补助等项目。对于社保和公积金来说，每家公司为员工购买的比例都不一样，求职者可以向面试官打听一下，如果公司购买这些项目的比例较少的话，可以要求高一点的工资。另外，出差补助也是一项隐形的收入，如果招聘单位是一家外包公司，则它很可能会有很多的出差时间，如果出差补助比较丰厚的话，求职者也可以酌情考虑一下。

该题目没有一个标准的答案，求职者应该根据自身和市场的环境来进行回答，以下是一些回答思路：

（1）避免说一个具体的数字，最好能说一个范围，这样可以给公司和自己留一个回旋的余地。

（2）了解公司其他的福利情况，例如，社保、公积金、出差补助等。这些福利在某种程度上也是可以作为收入的。

（3）对人才市场和自身的价值做出一个基本的判断，给自己定出一个底价。

【答案】

一般来说，待遇问题是面试过程中，最后一项问题了，该问题的回答失败有可能是导致整个面试的失败。首先，求职者应该确定一个自己的底价，这也是一个原则问题。然后，从面试官那里了解到公司的其他福利问题，如果太低的话则需要提供一些工资。最后，给出一个比底价略高的价格范围，让招聘公司进行决定。

面试题 185　你认为团队工作和独自干活哪样效率更高

现在的 IT 项目，大多是由多个人协作完成，个人孤军奋战的时代已经过去。因此，该题目有一个比较明显的回答倾向，就是团队工作效率更高。但是，应该如何解释原因呢？

团队工作的高效是体现在哪些地方的呢？本例将分析本题目考察意图，帮助求职者根据自身的情况回答该题目。

【出现频率】 ★★★★

【关键考点】

- 团队合作意识；
- 如何处理好与团队成员之间的关系。

【考题分析】

有一种程序员，他觉得和其他人的交流太浪费时间，还不如自己多干一点，效率还更高。这种想法在以前技术人员比较匮乏的年代还是可以理解的，但是，现在的项目越来越大，开发人员不可避免的也就越来越多，根本不可能一个人把所有的事情都做完，团队合作不可避免。

一般说来，如果一个求职者说他单干的效率高过于团队，那么招聘单位可能就不会再往下考虑了。这是因为，有这种思想的员工往往不太容易交流，会把自己的想法强加于他人身上，做一件事情，累个半死还不一定把事情做好。

该题目的回答其实还是比较明显的，大多数公司都希望求职者回答团队合作更高效。那么它高效在什么地方呢？团队具有多个人，每个人的想法都不同，技术水平也不一样，甚至语言还有可能不统一，这就需要很大的交流成本，貌似还不如个人单干来得快，其实这是错误的想法。

交流是一个团队进行合作的基础，既然一个团队能够走在一起做项目，那么每个人都应该有一颗共同的目标和心愿，这就是他们之间的连接点，把团队成员紧紧的包围在一起。一旦他们为了一个共同的目标而努力，他们的力量不仅仅是简单的总和，而且还会迸发出多余的能量来。

由此可见，团队的力量和效率是远远高于个人的。另外，从长远的眼光来看，一个项目从开发到维护往往不是一个人能做的，如果涉及到二次开发，还有项目文档整理等问题，这些其实也是团队的作用。

【答案】

一个团队的力量和效率往往不是个人能力和效率的简单总和，它还存在其他的集体效应，可能带来更大的效果。因此，该题目应该无可厚非的回答是团队更高效。至于理由的解释，求职者应该注意以下几点：

（1）团队的成员是有共同的目标的，它们的心是齐的，力量和效果都远远超过个人。

（2）团队的交流是它们效率有多高的关键。

（3）个人的力量往往也是需要有团队才能绽放出光芒。

（4）IT 项目其实处处都离不开团队，每个人都应该团队意识。

面试题 186　如果你所处的团队中，并不是每个成员都承担着相同的工作量，你会怎样看待

如果读者参与过软件项目的开发或测试就应该知道，团队成员的工作量是肯定不一样的，这也是不可争议的事实。那么，作为其中一份子，应该保持怎样的态度比较合适呢？本例将分析本题目考察意图，帮助求职者根据自身的情况回答该题目。

【出现频率】 ★★★★

【关键考点】

- 团队合作意识；
- 如何处理好与团队成员之间的关系；
- 保持良好的心态。

【考题分析】

这个世界没有绝对的公平，每个人都应该接受这个事实，并应该尝试着使它变得更公平。一个团队中，根据角色的不同，每个人做的事情就会不同，那么他们的工作量也会有差异，甚至差异会很大，例如，有的人可能每天都没什么事做，但是有的人却天天都在加班。

在理想状态下，应该是每个人做的事情都会差不多，所用时间也相差无几，但是现实中往往会有差异。究其原因，主要来自以下几个方面：

（1）管理上出了问题。管理层在进行任务分配的时候，没有充分的考虑到每个人的实际情况，或对每个人的技术水平了解不够。

（2）每个人的做事风格不同。其实，团队中的每个人都有一个共同的目标，应该是可以很好协调工作的，但是他们可能风格的迥异而无法保证工作效率的一致。

（3）其他一些客观的原因。当项目在开发或测试的过程中，可能会受到外界的影响，而不能每个人都统一的工作和相同时间完成。

以上列举的只是一些常见的造成工作量不统一的原因。其中，管理问题是一个比较严重的问题，这也是根源问题。当一个团队组织好以后，管理者首先应该非常清楚每个成员的能力和做事风格，然后才能进行工作任务的分配，不然就会造成有些人会有抵触情绪而影响工作。

即使管理者很好的评估了每个人的能力和做事风格，事实上人员的差异还是无法避免的。这种情况就需要每个团队成员有一定自觉性，能保持与团队步伐的一致性。例如，有的人在设计上比较突出，他就应该被安排在设计的工作任务上，而且还需要把自己设计的思想和实物时时刻刻的与他们交流，保持信息的畅通，如果别人遇到什么问题能够及时的提供自己的帮助，这在一定程度上是可以抵消掉团队成员之间的差异所带来的工作量不统一的影响。

至于客观因素来说，范围就会比较大了，也难以控制，但是，应该相信人定胜天的道理。例如，有的时候，一些角色相对比较重要的成员生病了，或其他什么原因暂时不能工作了，那么他很可能会造成其他人工作的滞后，严重的影响到效率。要解决这类问题，沟通的一个非常有效的解决办法，了解每个人的角色和他们做的什么，可以很好的降低此类事情发生的风险，即使某某人突然离职，整个项目的进度基本上也不会受到什么影响，这也是一种齐心的体现。

【答案】

首先，应该明确一点：工作量的不统一是客观存在的，无法避免的。但是，团队成员可以通过自己的努力来降低这类事情发生的风险，或把后果的影响降低到最小。在回答该题目的时候，可以根据以下一些思想来回答：

（1）该情况几乎无法避免，必须承认这个事实。

（2）如果读者面试的是管理者的职位，可以从管理者的角度来谈如何了解每个人的能

力，以及如何知人善用。

（3）每个人的能力和做事风格会参差不齐，需要团队成员为了一个共同的目标，保持畅通的交流，把这些差异影响力降低到最小。

（4）如果发生什么客观原因而造成工作量不一致，则需要及时沟通，快速的填补该原因带来的缺失。

面试题 187　你怎样为工作任务区分轻重缓急

一个人在一个时段的工作，往往不会只有一个，这些工作是相互重叠的，那么如何来安排时间呢？先做哪个，或必须把哪个做到最好？本例将分析本题目考察意图，帮助求职者根据自身的情况回答该题目。

【出现频率】 ★★★★

【关键考点】

- ❑ 时间安排能力；
- ❑ 对工作任务的态度。

【考题分析】

任何工作都有轻重缓急之分。只有分清哪些是最重要的并把它做好，你的工作才会变得井井有条，才会高效。凡取得卓越成绩的员工，办事的效率都非常高。这是因为他们能够利用有限的时间，完成最多的重要的工作，取得最大的效果。任何工作都有主次之分，如果不分主次地平均使力，不论是时间上，还是精力上都是一种浪费。所以，在关键部位，在主要工作上，要用全部精力，将其做到最好，这就是把工作任务分清轻重缓急的关键点所在。

在现代的企业中，不管做什么，都要从全局的角度来进行规划，将事情分出轻重缓急，将大目标分成若干个小目标，并坚持“要事第一”的做事原则。那么，什么是重要的事情呢？什么又是紧急的事情呢？

这里可以把任务根据紧迫感和重要性分为以下 4 种类型：

（1）重要且紧急的事情。

（2）重要但不紧急的事情。

（3）不重要但很紧急的事情。

（4）不重要也不紧急的事情。

很明显，应该把重要且紧急的事情先做，而且要花主要的精力去完成，争取做到最好，因为这类事情能产生最大的效果。对于既不重要也不要紧的事情，应该放到最后来做，也就是没有什么其他事情的情况下再做。

对于重要但不紧急的事情和不重要但紧急的事情，应该先做哪个呢？重要但不紧急的事情指的是该事情尽管很重要，但还有足够的时间去做。而对于不重要但紧急的事情，就比较难以处理了，它们谁应该先做呢？首先需要搞清楚什么是不重要但紧急的事情，举一个例子：假如某天你在家里洗澡，突然有人敲门铃，没办法，只要裹着毛巾就去开门了，但是开门以后才发现敲错门了，这就是紧急但不重要的事情。所以，重要但不紧急的事情应该优先于不重要但紧急的事情。

【答案】

工作任务往往会有时间的重叠，为了达到最大的高效，员工需要把工作任务进行分类，有先后性和重要性的区别来完成。根据事情的重要性和时间紧迫性来说，可以把工作任务分为以下 4 类：

（1）重要且紧急的任务。

（2）重要但不紧急的任务。

（3）不重要但很紧急的任务。

（4）不重要也不紧急的任务。

以上的任务分类，应该遵循从上到下，用的精力从多到少的原则来完成，这样才能收到最好的效果。

面试题 188　如果你完全不同意你上司的某个要求，你怎么处理

上司是自己的直接领导，他是不是说的什么都是对的呢？肯定不是，那么如果你发现上司的某个要求或某个观点有明显的错误，应该如何对待呢？本例将分析本题目考察意图，帮助求职者根据自身的情况回答该题目。

【出现频率】 ★★★★

【关键考点】

- 与他人沟通的能力；
- 对待工作的态度。

【考题分析】

在一个集体中，面对一个话题的时候，往往是每个人都会有自己的观点，甚至有的时候观点的方向都是完全相反的。这个时候需要进行很好的协调，否则会影响到整个团队的凝聚力。

如果一个普通员工和另外一个平级的员工的观点相悖，可以坐下来，慢慢商量，看能不能有一个折中的方案，即使在争论的过程中，出现面红耳赤的现象，也是没有关系的，主要还是要把问题或矛盾解决掉。

但是，如果你发现你的观点与上司的观点截然不同，此时就需要冷静的对待了，即使你的观点是完全正确的，也需要有技巧性的说服上司，而不能采用与其他同事激烈争论的方式。

首先，应该对自己的想法或做法进行反复的思考和论证，看看是不是哪儿有纰漏，会不会是自己出错了，也就是首先要想到是自己的错误。如果自己非常确信是上司犯了某个错误，可能会导致某某严重的后果，一定要讲出来，这是为了团队集体的利益着想。但是，在与上司谈话的过程中一定要注意技巧，切不可盲目自大，应该保持一颗谦虚学习的心态和遵循循序渐进的原则，让上司明白自己的想法，认识他的错误。理智的上司也一定会谦虚的接受的。

如果你在与上司交流的过程中发现还是自己错误了，一定要承认自己的错误，并且还应该感谢上司为你纠正了错误，牢记类似的错误，下一次一定不可以再犯。

总之，如果你完全不同意你上司的某个要求的时候，保持冷静和良好的心态是非常重要的，大致应该按照以下几个步骤来进行：

（1）自我反思。考虑是否是自己错了。

（2）心平气和的和上司交谈，慢慢的让他明白自己的错误。

（3）如果在交流的过程中发现依然是自己的错误，则需要承认自己的错误。

（4）不论过程如何，都应该以解决问题为最终目的。

【答案】

上司也是团队的一个成员，他也有可能会出错，而且他的错误可能会导致比较严重的后果。如果你完全不同意上司的观点，应该进行及时的沟通，解决问题。在沟通的过程中，应该遵循以下几点原则：

（1）谦虚谨慎。

（2）循序渐进，切不可心浮气躁。

（3）自我认识明确。

（4）以最终解决问题为目的。

（5）如果不得已发生争吵，应该向上司道歉。

16.3　小　　结

本章分析了一些关于软件开发面试过程中经常会遇见的情商类面试题。首先讲解的是针对应届毕业生的一些常见面试题目，然后是比较大众一点的面试题目。情商类题目跟技术的关系不大，它主要是看一个求职者的求职心态和性格特征，这些题目都没有一个标准的答案，求职者应该根据自身的情况进行灵活的回答。但是，这些题目都有一个原则，就是求职者应该扬长避短，尽量的把自己的长处展现出来，而把自己的缺点隐藏起来（不是回避缺点），尽量的把自己往公司招聘职位的要求靠拢，让面试官觉得你非常适合这个职位。

第 17 章　智商类面试题

在一些大公司的面试题中，往往可以看到一些智商类的面试题，这些题目可以考察出求职者的逻辑思维能力、大脑反应速度等各种信息。在现在这个时代里，不论是面试什么类型的职位，一般都会或多或少的包含一些这样的题目，也是这两年的 IT 面试的常见面试题目，它们的答案可能不太统一，但是却有一定的思路，最主要还是根据求职者的人生经历、智慧等因素来决定的。本章将包含关于 IT 面试中一些常见的智商类面试题，并且分析这些题目，帮助读者了解如何回答这类题目的思路。

17.1　脑筋急转弯

脑筋急转弯，是大家平时经常会用来考其他人的一种游戏，而它的答案往往不是平常思路所能回答它，需要求职者有足够的应变能力。本节将集中讨论有关 IT 面试中常见的脑筋急转弯面试题。

面试题 189　美国有多少辆汽车

该题目是一道开放新非常强的题目，相信没有人能知道美国具体有多少辆汽车，但是求职者不能直接回答说不知道啊。那么，这样的题目应该如何回答呢？本例将分析本题目考察意图，帮助求职者打开思路，得出本题目的最佳答案。

【出现频率】 ★★★★（2005 年微软招聘面试题）

【关键考点】

- 应变能力；
- 逻辑思考能力。

【考题分析】

很明显，有类题目是没有标准答案的，如果让 100 个人来回答，可能就会有 100 个答案。其实，招聘单位是想通过这类问题来看求职者在面对从没见过的题目的时候，会有怎样的反应能力，逻辑思考能力如何。

既然它是考察一个求职者能否对一个问题进行符合逻辑的创造性思考，并迅速通过这种思考寻求到解决问题的办法。至于答案，招聘单位显然并不关心，关心的是求职者的思考方向和过程。它的指向性很明显，应聘者是否能在很短的时间中，对出其不意的问题做出反应，并能够有逻辑地回答这样的问题，同样希望能够得到出其不意的答案，从而看出求职的弱点来。

那么，求职者在回答这类问题的时候，就不应该拘泥于一个统一的答案，而是把自己

的想法说清楚即可。并且，在描述自己想法的时候，一定要注意逻辑性，尤其是前后的因果关系。只要能把你的想法正确的表达出来，就是一个不错的答案。

有不少求职者通过在网上搜集这种试题来准备答案，使用一种固定的模式来回答，显然违背了企业的本意，因此，重复的答案都不是好答案。

【答案】

本题目想考察的是求职者的应变能力和逻辑思考能力，要求求职者能够短时间里构思出一种严密的逻辑构思，以下是一个参考回答：

美国一共有多少人口？这些人中又有多少人会开车？而会开车的人中又有多少有这样的经济能力可以购买汽车？可以购买汽车的人中是不是都已经买了？这些问题解决了，那答案自然就知道了。

面试题 190　下水道的盖子为什么是圆形的

大家可能平时没有注意到，下水道的盖子大多数的时候都是圆形的，那这是为什么呢？而该题目真正的考察目的是不是真的就是求职者，是否知道井盖为圆形的直接原因呢？本例将分析本题目考察意图，帮助求职者打开思路，得出本题目的最佳答案。

【出现频率】 ★★★★（2005 年微软招聘面试题）

【关键考点】

- 应变能力；
- 逻辑思考能力。

【考题分析】

其实，即使是专家也不一定能完全回答出下水道井盖是圆形的所有原因。那么招聘公司真正想要了解的并不是求职者是否知道这些原因，而是一种创造性的思维以及思考的过程和方式。

首先，下水道的井盖不完全是圆形的，也会有方形的井盖，只不过圆形的井盖居多一些。如果只考虑圆形井盖来说，有一个比较直接的原因，就是下水道往往是圆形的，所以井盖也是圆形的。下水道设计成圆形的原因是，圆柱形可以承受比较大的压力，而且它能比较方便的留出一个人的位置来，让他可以自由的通过井壁上的梯子来上下移动。因此，按洞的形状来分析，盖也是也圆形的。

如果要谈一谈圆形井盖相对于方形井盖来说，它有什么优势的话，可以思考一下如果使用方形的井盖会有什么弊端。例如，圆形可以受力均匀，不像方形的井盖有棱边，容易磨损而造成受力不均。另外，圆形的井盖和圆形的井口，可以保证井盖不会掉进洞中，就避免了一些人故意搞恶作剧。

美观在一定程度上也可以作为一条原因。一般来说，圆形的东西比方形的东西更具有亲近性，也就是看着更舒服。这主要还是需要求职者能够把它说明白，让它具有一定的说服力。

其实，除了以上提到的一些原因，还可以用很多种因素来解释为什么井盖是圆形的，并且每种说法都可以进行一种论述，这也就是面试官真正想听到的东西。求职者应该把一些原因列举出来，并对每一条进行分析，分析的过程中，只要求职者能够言之有理，都可以算作好答案。

【答案】

本题目想考察的是求职者的应变能力和逻辑思考能力，要求求职者能够比较快速的想到原因，并能可以清楚明白的阐述每一条原因，让这些理由有说服力。面对该问题，可以从以下几点常见观点入口，进行相关分析：

（1）圆形的井盖受力均匀，不会有棱角被破坏的可能。

（2）圆形的井身对土的压力的承受分布均匀，圆形的井身配置圆形的井盖。

（3）修理工人上下方便。

（4）美观、习惯。

（5）其他。

面试题 191　分蛋糕

创造性，是每一家公司都希望员工具备的东西，它在某些时候，可以让工作效率成倍的提高。本题目考察的就是求职者是否会用创造性的思维去解决问题。本例将分析本题目考察意图，帮助求职者打开思路，得出本题目的最佳答案。

【出现频率】 ★★★★

【关键考点】

- ❑ 应变能力；
- ❑ 逻辑思考能力；
- ❑ 创造性的思考能力。

【考题分析】

分蛋糕的题目有各种版本，但是大体的意思是类似的，例如，一盒蛋糕切成 6 份，分给六个人，但蛋糕盒中还必须留一份，怎么办？

乍一看，这是不可能的事情嘛。一共就 6 块蛋糕，都分走了，哪儿再来一块放进盒子中呢？这就是有创造性和无创造性的人的区别所在。一般人的想法都是无法解决该问题的，这是因为他们都循规蹈矩的思考问题，走常规路线，结果发现根本走不通，此时就应该考虑使用一些特别的创造性思考方式了。

其实，稍微看一下题目可以知道，题目并没有要求送人蛋糕的时候只能包含蛋糕，那就意味着可以把一块蛋糕连同盒子一起送出去，这样，不就解决了有一块蛋糕留在盒子中的要求了吗。

此时，很多人会有一种恍然大悟的感觉：原来这么简单啊。但是，你怎么就没有想到呢？有人常说：成功和失败也就只有一线的距离。是的，有很多人离成功只有一步的距离，但是他们最终还是失败了。让一个人用常规的方式把事情做好是比较容易的，但是如果要求他有创造性，则比较难了，他必须具有较强的逻辑思考能力，以及长期养成的用创造性的方式来思考问题的习惯。

【答案】

本题目想考察的是求职者的应变能力、逻辑思考能力和创造性能力，要求求职者能够通过创造性的思考方式来解决常规思维无法解决的问题，并能准确的提出解决方案，以下是一个参考性的答案：

首先把蛋糕切成六份，从中拿出五份分给五个人，再把最后一份连蛋糕盒一起分给第

6 个人。

面试题 192　你怎样改造和重新设计一个 ATM 银行自动取款机

想象力，对于做技术工作的程序员来说，是一项需要具备的特性。因为，想象力可以让程序员充满了创造力，工作的扩展性和效率都会因为想象力的丰富而提高。本题目考察的就是求职者到底具有多大的想象力。本例将分析本题目考察意图，帮助求职者打开思路，得出本题目的最佳答案。

【出现频率】 ★★★★

【关键考点】

- ❑ 应变能力；
- ❑ 逻辑思考能力；
- ❑ 丰富的想象力。

【考题分析】

ATM 银行取款机是银行的一项自助服务，它的实现是基于电子技术和机械技术的。在一般人的眼中，可以想象的它就是一台可以使用的机器，当用户插入卡、输入密码以后，就可以进行查询、取钱等操作了，没有什么好多思考的。其实，任何东西都会有改进和提升的余地。

人类总是在思考中不断进步的，ATM 自动取款机也是需要进步的，如何发挥想象力来扩展它的功能，如何改进它，如何设计它，使它能够做更多的事情呢？想象力是一个关键的因素。

很明显，该题目没有标准答案，招聘公司就是想看看求职者的想象力有多大，能创造出多大的东西出来。但是，这里的想象力，绝对不是毫无根据的乱想，必须要让想法具有理论依据。因此，求职者在进行想象的时候，一定要进行适当的想象，并且可以用比较严密的逻辑思维来说明。

例如，可以想象未来的 ATM 银行取款机除了常规的查询、取款、存款业务以外，还可以缴纳水电费、为手机充值、办理各种银行业务（如购买理财产品）等，也就是说，一切都可以依托于 ATM 机器来取代现在银行柜台的工作。这种想法是具有可行性的，例如，水电公司与银行合作，开通一个公开的缴纳账户，ATM 机上通过类似于转账的方式缴纳水电费；通讯运营商和银行合作，把银行的存款业务与通讯运营商的充值系统连接起来，通过 ATM 的转账就可以立刻缴纳手机话费。

【答案】

本题目想考察的是求职者的应变能力、逻辑思考能力和想象力，要求求职者具有充分的想象力，并且这些想象都是符合实际的想象，能够有充分的理论支持。在回答这类想象力的题目的时候，需要遵循以下一些原则：

（1）充分的发挥自己的想象力，尽量不要受现实的约束。

（2）一旦有了一定的想法以后，就需要找理论，以对自己的想法进行支持。

（3）进行一些基本的论证。

（4）有逻辑性的，有条例的表达自己的想法。

17.2　逻辑推理

逻辑推理题，需要求职者具有很强的逻辑思维能力，这也是作为一名 IT 技术人员从业者应该具备的素质。面对这类题目时，应该用严密的逻辑思考方式，进行有因果关系的推理，最终得到结果，并对结果进行相应的论证。本节将集中讨论有关 IT 面试中，常见的逻辑推理面试题。

面试题 193　3 盏灯与 3 个开关

这个世界上，有的时候会出现一种貌似无法解决的问题，但是这些问题却可以通过一种罕见的方式来解决的，只不过这种罕见的方式是需要从事物的另外一个方面观察而得出的，本题目就属于这一类。本例将分析本题目考察意图，帮助求职者打开思路，得出本题目的最佳答案。

【出现频率】 ★★★★

【关键考点】

- 应变能力；
- 逻辑思考能力。

【考题分析】

题目的全文描述大致是这样的：有两间房，一间房中有 3 盏灯，另一间房有控制这 3 盏灯的开关（这两间房是分割开的，毫无联系）。现在要你分别进这两间房一次，然后判断出这 3 盏灯分别是由哪个开关控制，你能想出办法吗？（注意：每间房只能进一次，另外，白炽灯泡效果会更好）。

按照通常的理论来讲，只进入房间一次，是没有办法判断谁是 A，谁是 B，谁是 C 灯的。这就需要求职者充分的开动脑筋，想办法来解决问题了，或许题目的最后一个注释会给求职者带来点信息。

白炽灯泡和其他的灯泡有什么区别呢？亮度要低一些、更费电、通过会很热等。亮度对于本题目来会说，似乎没有什么可以利用的，因为两个房间是完全隔离的，你无法通过一个房间获取另外一个房间的灯光，所以亮度就可以不用考虑了。至于费电程度来说，不可能通过电表来统计吧，这 3 个灯泡都是一样的啊，所以费电程度也不能作为判断的依据。最后，就剩下温度了，白炽灯泡会比其他的灯泡更能发热，这个温度会随着灯泡亮着的时间增长而升高的，而且题目也没有限制灯泡亮的时间，所以，利用灯泡的温度来区别是可行的。

因为每间房都只能进入房间一次，所以在进入开关房间的时候，只能开启一盏灯。也就是说，通过灯光来进行判断，只有一盏灯可以，其余的就只能通过温度的高低来解决了。为了让灯泡的温度能够保持至人能够感受到，因此就需要比较长一点时间的点亮某一盏灯泡。所以，答案基本就出来了：在开关房中，先长时间点亮一盏灯泡，然后关掉，再点亮另外一盏灯泡，并立刻跑进灯泡房间中，3 盏灯的区别就出来了，一盏是亮着的，另外一盏温度比较高，最后一盏的温度是凉的。

【答案】

本题目想考察的是求职者的逻辑推理能力，以及一些对外界事物的观察力。该题目的关键点需要想到可以通过灯泡的温度来进行灯泡的区别，这一点可以通过题目最后一点的提示想到，以下为一个参考性的答案：

首先进入开关房间，先开开关 A，过段时间，关 A，开 B，去另一房间，亮着的灯是 B 控制，不亮的灯中热的是 A 控制，冷得是 C 控制。

面试题 194　戴帽子

在初中数学里，有一种常考的题目，就是在一个比较复杂的几何平面图形中，判断两个线段是否平行。这类题目往往有一个特点，就是无法直接证明到这些线段是否平行，却可以通过其他的线段或角度间接的得到答案。本题目的思考方式就与平行题目类似，需要借助其他的力量来解决自己的问题。本例将分析本题目考察意图，帮助求职者打开思路，得出本题目的最佳答案。

【出现频率】 ★★★★

【关键考点】

- 应变能力；
- 逻辑思考能力。

【考题分析】

该题目的原形是这样的：一天中午，3 个男青年正在公园的草地上聊天，忽然争论起来，争论的是 3 人中谁最聪明，但无人肯服输，这时来了一个老人，老人说：“不要再争了，我这里有 5 顶帽子，3 顶黑色的，两顶白色的，你们闭上眼睛，我给你们每人戴上一顶，如谁能最先猜出自己帽子的颜色，谁就最聪明。

3 人经过商量后同意了，闭上了眼睛，老人便给 3 人都戴上了顶黑帽子，并把白帽子放进了自己口袋，此时便让 3 人把眼睛睁开了，3 人面面相觑，过后不久，一个青年突然跳了起来，他向老人说出自己戴的是黑帽子，请问他是怎么知道的？（限制条件：3 人相隔至少 5 米，无法看到对方眼中的反影；3 人不能用任何事物触及自己的帽子；3 人之间不能有任何语言或行动上的交流）。

乍一看，要想猜出自己帽子的颜色，是不太可能。这是因为，不论别人帽子什么颜色，自己的帽子是什么色的可能性都有。但是，本题目是可以通过严密的逻辑分析过程而得到答案的。

假设这 3 个人为 A、B 和 C。当 A 看到其他两个人的帽子都是黑色的时候，他自己就会想自己的帽子可能是黑色，也有可能是白色的。如果自己的帽子是白色的，那么其他两个人看到的就是一黑一白了，因此他们其中之一，例如 B，就会思考了，已经出现了一顶白色帽子，另外还有一顶如果已经出现在自己头上的话，那么 C 就会很快猜出自己帽子的颜色是黑色的，而事实上却没有，所以 B 就肯定自己戴的是黑色帽子。但是，B 同样没有很快的作出决定，所以 A 就有理由推翻自己帽子是白色的假设，所有 A 就可以肯定自己的帽子是黑色的。

【答案】

本题目考察的是间接推理和归纳能力。本例的关键在于对情况选进行假设，然后进行

推理，然后得到与实际相悖的结论。采用这种方式，就得到正确的结果。由于在题目分析中已经给出答案，所以这里不再重复。

面试题 195　海盗分金

本题目是一道非常著名的智力面试题，它非常考验求职者的逻辑思考能力，也是一种矛盾体的方式，求职者必须认真思考每一个海盗的想法以及他与其他海盗之间的利益关系，并作出严密的推断，才能得出最后的答案。本例将分析本题目考察意图，帮助求职者打开思路，得出本题目的最佳答案。

【出现频率】 ★★★★（微软招聘面试题）

【关键考点】

- 应变能力；
- 逻辑思考能力。

【考题分析】

本题目的原型为 5 个海盗，分 100 个金子，他们依次提出个方案，如果有一半或以上人同意就通过，通不过则把提出方案的海盗丢到海里，再继续分金子。海盗首先希望生存，然后希望利益最大，那么第一个海盗应该怎么提分金子的方案？

若从第一个海盗开始思考，如果第一个海盗提出自己拿全部，而其他人一个都没有的话，他肯定会被扔进海里，所以他必须分一些给其他海盗，但是如何用最少的金子，获得最大得票率呢？很多求职者一想到这里的时候，就会被这些东西所搞混。其实，这需要考虑每一个海盗的想法，也就是他们获得多少个金子以后就会支持第一个海盗，因此，应该从最后那个海盗的想法开始分析。

这里假设这 5 个海盗为 A、B、C、D 和 E。作为最后一个海盗 E，他肯定是希望前面的所有海盗都死掉，他就可以拿全部的金子了，但这不可能，至少他前面的海盗 D 肯定可以在他之前拿走所有的金子。

若只剩下 D 和 E 海盗，D 肯定是把所有的金子都据为己有，因为即便他不分一个金子为 E，他的得票率也是 50%。所以，E 不会让 C 海盗死掉，因为 C 为了得到得票率，肯定会分一些金子给 E 的。

同样，如果只剩下 C、D 和 E，D 肯定不会同意 C 的提议，因为 D 是希望 C 死掉的。那么，C 就只需要给 E 一个金子，E 就可以支持 C，从而达到超过 50%的支持率。同时，D 就不会希望 C 的上一级 B 死掉，不然他一个金子就没有了。

若是 B、C、D 和 E 4 个海盗分金子的话，C 肯定不会同意 B 的提议，因为 C 是希望 B 死掉的。那么，B 为了得到 50%的支持率，他至少需要拉拢一个人就好，这个人就是 D，因为 D 不希望 B 死掉，所以 B 的方案就会是：99：0：1：0。但是，B 不能给 E，这是因为即使 B 死掉了，E 也可以得到一个金子，若要得到 E 的支持的话，就需要给 E 两个金子了，B 的利益就少了。

此时，再来考虑 A、B、C、D 和 E 海盗都在的情况。A 海盗想要活下来，就必须至少得到另外两个海盗的支持。其中，B 是不用考虑的，因为 B 是希望 A 死掉，他就可以得到

最多的金子。而 C 却比较好收买，因为上一轮分析中，C 没有得到金子，所以只需要给他一个金子即可。对于 D 和 E 来说，收买 E 会更容易一些，因为在上一轮的分析中，他没有得到金子，所以他保全 A 是最好的选择，至少能得到一个金子。因此，最后的结果就是 98：0：1：0：1。

【答案】

本题目主要考察的是求职者的逻辑推理能力。该题目的关键点是需要能分析出每一个海盗的心理，以及一层一层的心理变化过程。这类问题的回答，除了答案很重要以外，过程也是招聘单位想要知道的重点，以下为一个参考性的答案：

（1）若只剩下 D 和 E 海盗，方案为 100：0。

（2）若只剩下 C、D 和 E 海盗，方案为 99：0：1。

（3）若只剩下 B、C、D 和 E 海盗，方案为 99：0：1：0。

（4）若是 A 海盗提议，则是 98：0：1：0：1。

面试题 196　罪犯认罪

博弈是一门很深的学问，它是两个人智慧的较量，通过博弈，就可以看出一个人的价值取向等问题。罪犯认罪是一个典型的博弈的案例，那应该如何回答该题目才能比较符合招聘单位的需要呢？本例将分析本题目考察意图，帮助求职者打开思路，得出本题目的最佳答案。

【出现频率】 ★★★★

【关键考点】

- ❑ 应变能力；
- ❑ 逻辑思考能力。

【考题分析】

本题目的原形为：有甲、乙两个人，因为违法被公安机关抓住，并进行隔离审查。如果甲、乙两个人都坦白，由于证据确凿，各判 8 年徒刑；如果其中一人坦白，另一人不坦白，则坦白的人释放，不坦白的人判 14 年徒刑；如果两个人都不坦白，各判一年徒刑。请问：如果你是其中的一人，在被隔离审查的时候，你是坦白，还是不坦白。

对于罪犯来说，最好的结果就是两人都不坦白，他们只获得 1 年的判刑。但是，他们两人是被隔离审判的，没有交流的情况下，就很难做出最好的决定了。因此，比较折中一点的答案就是坦白，至少这样可以获释或者和同伙相同的判刑，不会亏于同伙。并且，事实上太多数人都会有这样的想法。

大家肯定会想，如果这两个人在作案之前就商量好，如果被抓住，打死都不认罪的话，那么他们就可以得到最好的结果。的确，这不是没有可能的，如果两个罪犯都很重义气，都是打死不认罪的话，他们确实可以得到最好的结果。但是，现实社会中谁又会去冒这个险呢？

因此，这种博弈，一般都是警察是最大的赢家。两个都不坦白，只是一种理论上的现象，如果求职者要回答该题目，最好还是回答坦白为佳。

【答案】

本题目考察得比较灵活，没有一个比较标准的答案，根据招聘单位的意图和职位的不

同就应该有不同的回答。如果是比较追求完美的公司，则他们可能更希望求职者回答不坦白的结果，也表示你对团队成员的信任。有的公司则会看你的风险分析能力如何，选择坦白可以把风险降到最低。

17.3　计 算 推 理

计算推理题，指的是通过数学计算来推出逻辑结果的一种题目，从而解决一些比较极端的问题，这也是对求职者数学能力和逻辑能力的考察。本节将集中讨论有关 IT 面试中常见的计算推理面试题。

面试题 197　倒水问题

倒水问题是一类比较典型的计算推理问题，它有很多种变化形式，但是它的解题思路是一定的。主要是要求求职者能够快速的想到如何通过容器的变化来找到小容量的水量。本例将分析本题目考察意图，帮助求职者打开思路，得出本题目的最佳答案。

【出现频率】 ★★★★

【关键考点】

- ❑ 应变能力；
- ❑ 逻辑思考能力；
- ❑ 数学运算能力。

【考题分析】

本题目的原形为：有 3 个容器，分别是 3、7、10 体积，容器没有刻度。10 体积的容器中装满某种液体，3、7 容器是空的。没有别的工具，现在请把这种液体平分成相对精确的两份。

在没有刻度的情况下，又要精确的量出体积为 5 的水来，唯一的办法就是通过这 3 个容器的相互倒换。那么，应该如何相互倒水呢？首先需要明确的是，为了实现精确，每次倒水都必须完全把水倒出，或者完全把容器装满。

如果单单是实现把水平分的话，有很多种方法，只不过有的方法的步骤要少一些，有的要多一些。例如，以下这样一个步骤就可以实现（假设把容器从小到大编号为 A、B 和 C 容器）：

（1）把 C 的水倒 3 体积到 A 中，此时的容积比例为：3：0：7。
（2）把 A 的水倒入 B 中，此时的容积比例为：0：3：7。
（3）把 C 的水倒 3 体积到 A 中，此时的容积比例为：3：3：4。
（4）把 A 的水倒入 B 中，此时的容积比例为：0：6：4。
（5）把 C 的水倒 3 体积到 A 中，此时的容积比例为：3：6：1。
（6）把 A 的水倒入 B 中，直到加满为止，此时的容积比例为：2：7：1。
（7）把 B 的水全部倒入 C 中，此时的容积比例为：2：0：8。
（8）把 A 的水倒入 B 中，此时的容积比例为：0：2：8。
（9）把 C 的水倒 3 体积到 A 中，此时的容积比例为：3：2：5。

（10）把 A 的水倒入 B 中，此时的容积比例为：0：5：5。

以上相互倒水的过程中，有一个比较关键的地方，就是找到一个两体积的水量，这样才能为平分水，也就是找到 5 体积的水奠定基础。

说明：有的时候，这类题目还可能会规定步骤的数量，例如，在不超过 10 步的情况下完成测量。

【答案】

本题目是倒水问题的典型题目，它的关键点在于如何找到一个 2 体积的水量来，以下是该题目的参考答案和步骤：

（1）3：0：7。
（2）0：3：7。
（3）3：3：4。
（4）0：6：4。
（5）3：6：1。
（6）2：7：1。
（7）2：0：8。
（8）0：2：8。
（9）3：2：5。
（10）0：5：5。

面试题 198　找出轻球

天平称球问题是一类比较典型的用天平称重来区分东西的问题了，它的难度不大，主要是要求求职者能够充分的利用一个球更轻或更重的特点来区分它们。本例将分析本题目考察意图，帮助求职者打开思路，得出本题目的最佳答案。

【出现频率】 ★★★★

【关键考点】

- ❑ 应变能力；
- ❑ 逻辑思考能力；
- ❑ 数学运算能力。

【考题分析】

本题目的原形为：有 12 个外表完全一样的球，其中有一个球重量不一样（这里假设是更轻），给你一个没有刻度的天平，你只能用 3 次，把球找出来。

该题目和倒水题目有一点类似，只不过这个题目是通过称重量来进行区分。过程还是需要有严密的计算推理过程。首先，肯定是需要把这些球进行分堆来称，这样才能把轻球的范围缩小，但是应该如何分堆呢？如果进行平分，也就是 6：6，那么，第一次过秤的时候，就可以确定轻球在其中一堆当中。然后再把轻的那一堆中的 6 个进行平分，也就是 3：3，此时又可以确定一堆比较轻，但是此时轻球是在 3 个里面，只能一边放一个，另一边放两个，如果轻球是在两个那边，那么如何确定呢？这说明，把球按照 6：6 分是行不通的，需要改一种划分方法。

其实，这里可以把球分成 3 堆，每一堆 4 个，取其中两堆过秤，如果相同则确定轻球在第 3 堆，如果其中一边轻则确定轻球在那堆里面。确定好轻球在其中 4 个以后，就可以把那 4 个轻球再次平分过秤，也就是 2：2，然后把轻的一边再过秤，就可以很明显的找出轻球是哪个了。

【答案】

本题目是称球问题的典型题目，它的关键点在于如何划分球堆。以下是该题目的参考答案和步骤：

（1）分 3 堆，4：4：4。

（2）把上步确定好的 4 个球进行 2：2 分堆过秤。

（3）把最后两个球过秤。

面试题 199　骗子购物

本题目是一道比较简单的计算推理题，但是却考倒了很多的求职者，这是因为他本质上考察的是求职者严密的逻辑推理思维，只不过具有一定的迷惑性。本例将分析本题目考察意图，帮助求职者打开思路，得出本题目的最佳答案。

【出现频率】 ★★★★

【关键考点】

- ❑ 应变能力；
- ❑ 逻辑思考能力；
- ❑ 数学运算能力。

【考题分析】

本题目的原型为：狡猾的骗子到商店用 100 元面值的钞票买了 9 元的东西，售货员找了他 91 元钱，这时，他又称自己已有零钱，给了 9 元而要回了自己原来的 100 元。那么，他骗了商店多少钱？

该题目一般会有下面 3 种答案的出现：

（1）商店没有被骗。因为售货员只是把他原有的 100 块还给了他，所以根本没有被骗。

（2）被骗了 91 块。因为售货员把零钱收了以后，应该收回找他找出的 91 块，所以商店被骗了 91 块。

（3）91 块钱和商品本身的价值。因为售货员没有收回他不应找出去的 91 块，另外，还有商品本身的价值也是没有支付的，所以商店的损失应该是 91 块钱+商品本身的价值。

这种题目的思路有一点乱，但是如果仔细理一下还是可以比较容易得到答案的。其实，只需要分析一下，只需要判断一下骗子一共给了售货员多少，又从售货员那里拿回多少，就可以得出他到底骗了商店多少钱了。

首先，骗子给了售货员 100 元，得到 91 元的找补，并拿到商品，然后又给了 9 元，并拿回了 100 元钱。那么就可以得出，骗子一共递给售货员 100+9=109 元，拿回的金额是 91+100+商品本身，也就是说，骗子骗得的金额应该是：（91+100+商品本身）-（100+9）=82+商品本身的价值。

像这类题目的本质是比较简单的，只不过他的过程比较绕，就会觉得不太容易理清思路，所以就容易出错，而且事实上出错率是很高的。其实，只需要记住一个原则就可以了，

拿就是拿进的总额减去拿出的总额就可以知道骗子得了多少了。

【答案】

本题目是一种以迷惑方式来考察求职者思路是否清晰的面试题。它本质上来说是不困难的，但是它的过程却比较容易让人迷惑，所以求职者在回答该类问题的时候，一定要注意理清思路。

该题目的答案为：82 元+商品本身的价值。

面试题 200　烧香问题

烧香问题是一类比较典型的时间计算推理问题，它有很多种变化形式，但是它的解题思路是一定的。这类问题有的时候看起来是不可解的，其实认真分析以后，不管它如何变化，总能解决问题。本例将分析本题目考察意图，帮助求职者打开思路，得出本题目的最佳答案。

【出现频率】 ★★★★

【关键考点】

- ❑ 应变能力；
- ❑ 逻辑思考能力；
- ❑ 数学运算能力。

【考题分析】

本题目的原型为：有两根不均匀分布的香，每根香烧完的时间是一个小时，你能用什么方法来确定一段 15 分钟的时间？

毫无疑问，应该用烧香的办法来确定这 15 分钟。但是，即使把香从两头一起点燃开始燃烧的话，也只能确定 30 分钟的时间，另外一根香应该如何去燃烧呢？很明显，等第一根香烧完以后再烧第二根香是不行的。

其实，如果能想到确定 30 分钟办法的时候，你已经成功了一半了，因为 15 分钟就是 30 分钟的一半。这就是说，一根香总共燃烧时间是 60 分钟，那么用另外一根香先帮它确定 30 分钟，再把这根香的两头都点燃，那么，等第二根香燃尽的时候，不就是刚好 15 分钟吗？

大致的解题步骤如下所示：

（1）点燃第一根香的两头，和点燃第二根香的一头。

（2）等到第一根香燃尽以后，再把第二根香的另一头点燃。

（3）从第一根香燃尽开始算起，直到第二根香燃尽，时间就是 15 分钟。

【答案】

第一个两头烧，与此同时另一个一头烧。等到第一个烧完以后，第二个的另外一头开始烧。从第一根香烧完开始计算，至所有香烧完，就是 15 分钟的时间。

17.4　小　　结

本章分析了一些关于软件开发面试过程中经常会遇见的智商类面试题，包括脑筋急转

弯、逻辑推理和计算推理方面的常见题目。脑筋急转弯尽管是一种答案性不明确的题目，但是它却可以比较好的看出求职者的逻辑思维能力和反映能力，这对于做 IT 开发的技术人员来说，是非常重要的。逻辑推理题则要求求职者具有非常缜密的逻辑推理能力，这类题目就可以看出求职者是否逻辑严密，可以通过自己的思考来解决问题。计算推理题相对逻辑推理来说，它增多了一个计算的过程，这样的计算过程也是一个严密逻辑考察，并且有的题目具有一定的迷惑性，还可以看出求职者的细心程度。

轻松地获得面试官的青睐，从千百个竞聘者中脱颖而出

本书内容及对应的教学视频时间

- 应聘软件开发职位前必须知道的那些事
- Java程序基础（43分钟视频）
- Java语法基础（103分钟视频）
- 数据类型及类型转换（63分钟视频）
- 数组和集合的使用（54分钟视频）
- Java图形用户界面（42分钟视频）
- 输入/输出流（37分钟视频）
- 多线程编程（37分钟视频）
- Java反射机制（30分钟视频）
- Java网络编程（44分钟视频）
- Java对数据库的操作（43分钟视频）
- Web开发相关技术（60分钟视频）
- Struts、Spring和Hibernate组合（109分钟视频）
- EJB与JPA相关问题（51分钟视频）
- Java编程试题（70分钟视频）
- 情商类面试题（48分钟视频）
- 智商类面试题（43分钟视频）

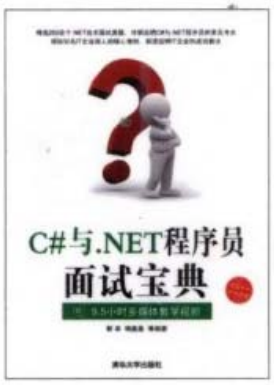
本书读者对象

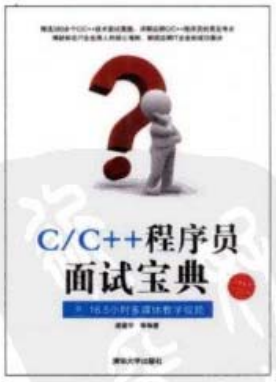
- 所有想了解Java面试的人员
- 计算机相关专业的应届毕业生
- 应聘软件行业的相关就业人员
- 技术部门的招聘主管
- Java EE职位竞聘者
- Java软件工程师职位竞聘者

超值、大容量DVD-ROM内容

- 14.5小时本书配套多媒体教学视频
- 55小时Java教学视频（免费赠送）
- 5.5小时算法教学视频（免费赠送）

程序员面试宝典





ISBN 978-7-302-22832-5
9 787302 228325 >
定价：49.80元（附视频教学DVD）