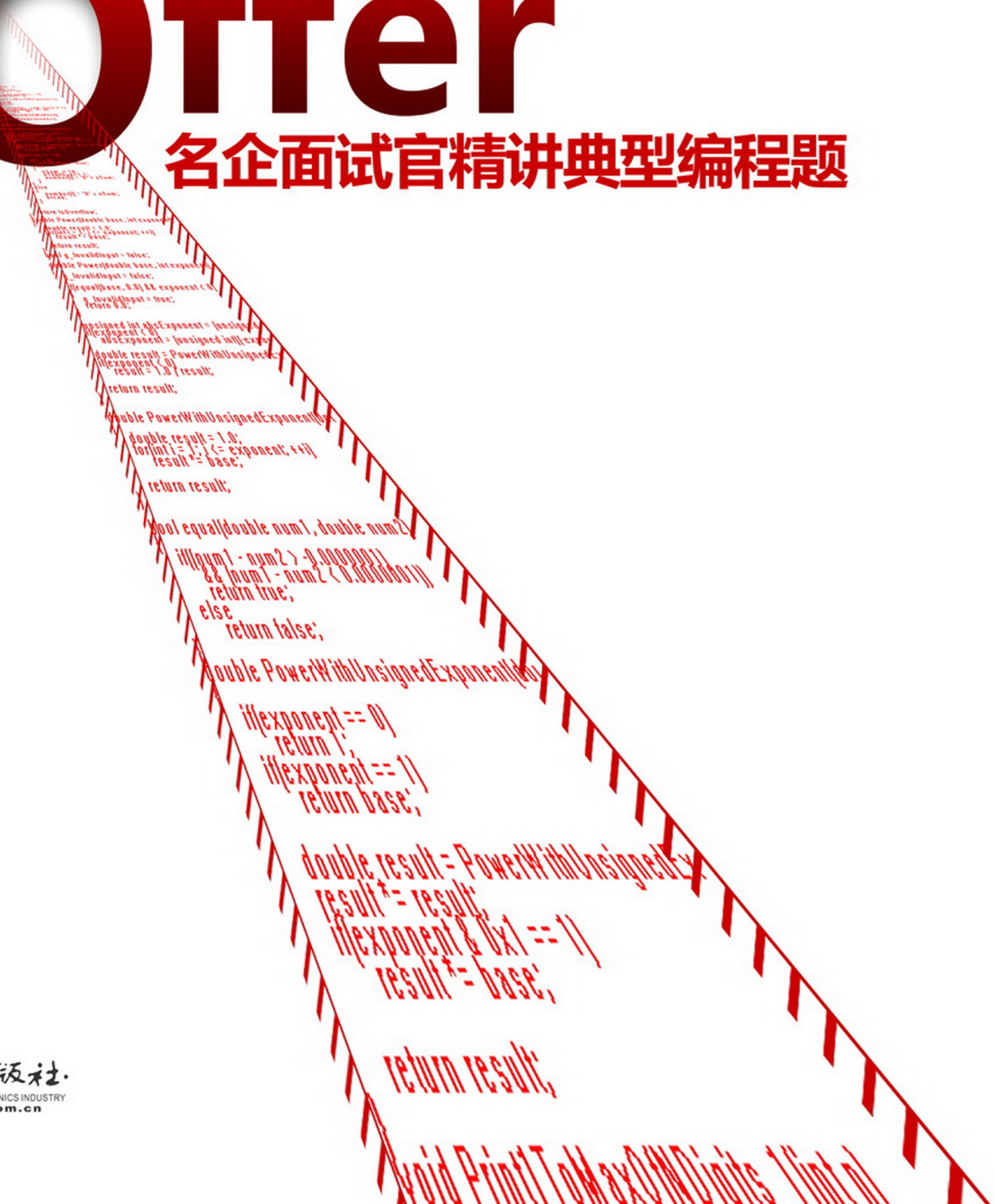
Broadview®
www.broadview.com.cn

剑指Offer

名企面试官精讲典型编程题

何海涛 著



電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

内 容 简 介

本书剖析了 50 个典型的程序员面试题，从基础知识、代码质量、解题思路、优化效率和综合能力五个方面系统整理了影响面试的 5 个要点。全书分为 7 章，主要包括面试的流程，讨论面试流程中每一环节需要注意的问题；面试需要的基础知识，从编程语言、数据结构及算法三方面总结了程序员面试的知识点；高质量的代码，讨论影响代码质量的的 3 个要素（规范性、完整性和鲁棒性），强调高质量的代码除了能够完成基本的功能之外，还能考虑到特殊情况并对非法输入进行合理的处理；解决面试题的思路，总结在编程面试中解决难题的常用思路，如果在面试过程中遇到了复杂的难题，应聘者可以利用画图、举例和分解复杂问题 3 种方法化繁为简，先形成清晰的思路再动手编程；优化时间和空间效率，介绍如何优化代码的时间效率和空间效率，读完这一章读者将学会常用的优化时间效率及空间换时间的常用算法，从而在面试中找到最优的解法；面试中的各种能力，本章总结应聘者在面试过程中如何表现学习能力和沟通能力，并通过具体的面试题讨论如何培养知识迁移能力、抽象建模能力和发散思维能力；两个面试案例，这两个案例总结了应聘者在面试过程中哪些举动是不好的行为，而哪些表现又是面试官所期待的行为。

本书适合即将走向工作岗位的大学生阅读，也适合作为正在应聘软件行业的相关就业人员和计算机爱好者的参考书。

图书在版编目（CIP）数据

剑指 Offer：名企面试官精讲典型编程题 / 何海涛著. —北京：电子工业出版社，2012.1
ISBN 978-7-121-14875-0

Ⅰ. ①剑… Ⅱ. ①何… Ⅲ. ①程序设计－工程技术人员－资格考试－习题集 Ⅳ. ①TP311.1-44

中国版本图书馆 CIP 数据核字（2011）第 215025 号

策划编辑：张春雨
责任编辑：李云静
特约编辑：赵树刚
印　　刷：北京丰源印刷厂
装　　订：三河市鹏成印业有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编 100036
开　　本：787×980　1/16　印张：17.25　字数：354 千字
印　　次：2012 年 1 月第 1 次印刷
印　　数：4000 册　　定价：45.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。
质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。
服务热线：（010）88258888。

前　言

2011 年 9 月份以来，我的面试题博客（http://zhedahht.blog.163.com/）点击率上升很快，累计点击量超过了 70 万，并且平均每天还会增加约 3000 次点击。每一年随着秋季新学期的开始，新一轮招聘高峰也即将来到。这不禁让我想起几年前自己找工作的情形。那个时候的我，也是在网络的各个角落搜索面试经验，尽可能多地收集各个公司的面试题。

当时网上的面试经验还很零散，应聘者如果想系统地收集面试题，需要付出很大的努力。于是我萌生了一个念头，在博客上系统地收集、整理有代表性的面试题，这样可以极大地方便后来人。经过一段时间的准备，我于 2007 年 2 月在网易博客上发表了第一篇关于编程面试题的博客。

在过去 4 年多的日子里，我陆续发表了 60 余篇关于面试题的博客。随着博客数目的增加，我也逐渐意识到一篇篇博客仍然是零散的。一篇博客只是单纯地分析一个面试题，但对解题思路缺乏系统性的梳理。于是在 2010 年 10 月我有了把博客整理成一本书的想法。经过一年的努力，这本书终于和读者见面了。

本书内容

全书分为 7 章，各章的主要内容如下：

第 1 章介绍面试的流程。通常整个面试过程可以分为电话面试、共享桌面远程面试和现场面试 3 个阶段，每一轮面试又可以分为行为面试、技

术面试和应聘者提问 3 个环节。本章详细讨论了面试中每一环节需要注意的问题。其中第 1.3.2 节深入讨论了技术面试中的 5 个要素，是全书的大纲，接下来的第 2～6 章逐一讨论每个要点。

第 2 章梳理应聘者接受技术面试时需要用到的基础知识。本章从编程语言、数据结构及算法三方面总结了程序员面试的知识点。

第 3 章讨论应聘者在面试时写出高质量代码的 3 个要点。通常面试官除了期待应聘者写出的代码能够完成基本的功能之外，还能应对特殊情况并对非法输入进行合理的处理。读完这一章，读者将学会如何从规范性、完整性和鲁棒性 3 个方面提高代码的质量。

第 4 章总结在编程面试中解决难题的常用思路。如果在面试过程中遇到复杂的难题，应聘者最好在写代码之前形成清晰的思路。读者在读完这一章之后将学会如何用画图、举例和分解复杂问题 3 种思路来解决问题。

第 5 章介绍如何优化代码的时间效率和空间效率。如果一个问题有多种解法，面试官总是期待应聘者能找到最优的解法。读完这一章，读者将学会优化时间效率及空间换时间的常用算法。

第 6 章总结面试中的各项能力。面试官在面试过程中会一直关注应聘者的学习能力和沟通能力。除此之外，有些面试官还喜欢考查应聘者的知识迁移能力、抽象建模能力和发散思维能力。读完这一章，读者将学会如何培养和运用这些能力。

第 7 章是两个面试的案例。在这两个案例中，我们将看到应聘者在面试过程中的哪些举动是不好的行为，而哪些表现又是面试官所期待的行为。衷心地希望应聘者能在面试时少犯甚至不犯错误，完美地表现出自己的综合素质，最终拿到心仪的 Offer。

本书特色

正如前面提到的那样，本书的原型是我过去 4 年多陆陆续续发表的几十篇博客，但这本书也不仅仅是这些博客的总和，它在博客的基础上添加了如下内容。

本书试图以面试官的视角来剖析面试题。本书前 6 章的第一节都是“面试官谈面试”，收录了分布在不同 IT 企业（或者 IT 部门）的面试官们对代码质量、应聘者如何形成及表达解题思路等方面的理解。在本书中穿插着

几十条“面试小提示”，是我作为面试官给应聘者在面试方法、技巧方面的建议。在第 7 章的案例中，“面试官心理”揭示了面试官在听到应聘者不同回答时的心理活动。应聘者如果能了解面试官的心理活动，无疑能在面试时更好地表现自己。

本书总结了解决面试难题的常用方法，而不仅仅只是解决一道道零散的题目。在仔细分析、解决了几十道典型的面试题之后，我发现其实是有一些通用的方法可以在面试的时候帮助我们解题的。举个例子，如果面试的时候遇到的题目很难，我们可以试图把一个大的复杂的问题分解成若干个小的简单的子问题，然后递归地去解决这些子问题。再比如，我们可以用数组实现一个简单的哈希表解决一系列与字符串相关的面试题。在详细分析了一道面试题之后，很多章节都会在“相关题目”中列举出同类型的面试题，并在“举一反三”中总结解决这一类型题目的方法和要点。

本书收集的面试题是都是各大公司的编程面试题，极具实战意义。包括谷歌、微软在内的知名 IT 企业在招聘的时候，都非常重视应聘者的编程能力，编程技术面试也是整个面试流程中最为重要的一个环节。本书选取的题目都是被各大公司面试官反复采用的编程题。如果读者一开始觉得书中的有些题目比较难，那也正常，没有必要感到气馁，因为像谷歌、微软这样的大企业的面试本身就不简单。读者逐步掌握了书中总结的解题方法之后，编程能力和分析复杂问题的能力将会得到很大的提升，再去大公司面试将会轻松很多。

本书附带提供了 50 道编程题的完整的源代码，其中包含了每道题的测试用例。很多面试官在应聘者写完程序之后，都会要求应聘者自己想一些测试用例来测试自己的代码，一些没有实际项目开发经验的应聘者不知道如何做单元测试。相信读者朋友在读完这本书之后就会知道如何从基本功能测试、边界值测试、性能测试等方面去设计测试用例，从而提高编写高质量代码的能力。

本书体例

在本书的正文中间或者章节的末尾，穿插了不少特殊体例。这些体例或用来给应聘者提出建议，或用来总结解题方法，希望能够引起读者的注意。

面试小提示：

本条目是从面试官的角度对应聘者的建议或者希望应聘者能够注意到的细节。

源代码：

读者将在本条目中看到一个格式为XX_YYYYY或者XX_Y_ZZZZZ的项目名称，该名称与用Visual Studio打开文件InterviewQuestions.sln之后看到的项目名称对应。本书附带的源代码请到电子工业出版社的官方网站下载。读者下载源代码并解压缩之后，请用Visual Studio 2008或者更新的版本阅读或者运行代码。

测试用例：

本条目列举应聘者在面试时可以用来测试代码是否完整、鲁棒的单元测试用例。通常本书从基本功能、边界值、无效的输入等方面测试代码的完整性和鲁棒性，针对在时间效率或者空间效率有要求的面试题还包含性能测试的测试用例。

本题考点：

本条目总结面试官采用一道面试题的考查要点。

相关题目：

本条目列举一些和详细分析的面试例题相关或者类似的面试题。

举一反三：

本条目从解决面试例题中提炼出常用的解题方法。这些解题方法能够应用到解决其他同类型的问题中去，达到举一反三的目的。

面试官心理：

在第七章的面试案例中，本条目用来模拟面试官听到应聘者的回答之后的心理活动。

关于遗漏的问题

由于时间仓促，再加上笔者的能力有限，书中难免会有一些遗漏。今后一旦发现遗漏的问题，我将第一时间在博客（http://zhedahht.blog.163.com/）上公布勘误信息。读者如果发现任何问题或者有任何建议，也请在博客上留言、评论，或者通过微博（http://weibo.com/zhedahht）和我联系。

致谢

在写博客及把博客整理成书的过程中，我得到了很多人的帮助。没有他们，也就没有这本书。因此，我想在这里对他们诚挚地说一声：谢谢！

首先我要谢谢个人博客上的读者。网友们的鼓励让我在博客上的写作从 2007 年 2 月开始坚持到了现在。也正是由于网友们的鼓励，我最终下定决心把博客整理成一本书。

在本书的写作过程中，我得到了很多同学、同事的帮忙，包括 Autodesk 的马凌洲、刘景勇、王海波，支付宝殷焰，百度的张珺、张晓禹，Intel 的尹彦，交通银行的朱麟，淘宝的尧敏，微软的陈黎明、田超，NVidia 的吴斌，SAP 的何幸杰和华为的韩伟东（在书稿写作阶段他还在盛大工作）。感谢他们和大家分享了对编程面试的理解和思考。同时还要感谢 GlaxoSmithKline Investment 的 Recruitment & HRIS Manager 蔡咏来（也是

2008 年把我招进微软的 HR）和大家分享了微软所推崇的 STAR 简历模型。还要感谢在微软期间我的两个老板徐鹏阳和 Matt Gibbs，他们都是在微软有十几年面试经验的资深面试官，对面试有着深刻的理解。感谢二位在百忙之中抽时间为本书写序，为本书增色不少。

我同样要感谢现在思科的老板 Min Lu 及 TQSG 上海团队的同事王劭、赵斌和朱波对我的理解。他们在我写作期间替我分担了大量的工作，让我能够集中更多的精力来写书。

感谢电子工业出版社的工作人员，尤其是张春雨和赵树刚的帮助。两位编辑大到全书的构架，小到文字的推敲，都给予了我极大的帮助，从而使本书的质量有了极大的提升。

本书还得到了很多朋友的支持和帮助，限于篇幅，虽然不能在此一一说出他们的名字，但我一样对他们心存感激。

最后，我要衷心地感谢我的爱人刘素云。感谢她在过去一年中对我的理解和支持，为我营造了一个温馨而又浪漫的家，让我能够心无旁骛地写书。我无以为谢，谨以此书献给她及我们尚未出生的小宝宝。

何海涛

2011 年 9 月 8 日清晨于上海三泾南宅

目　录

CONTENTS

推荐序二

I had the privilege of working with Harry at Microsoft. His background and industry experience are a great asset in learning about the process and techniques of technical interviews. Harry shares practical information about what to expect in a technical interview that goes beyond the core engineering skills. An interview is more than a skills assessment. It is the chance for you and a prospective employer to gauge whether there is a mutual fit. Harry includes reminders about the key factors that can determine a successful interview as well as success in your new job.

Harry takes you through a set of interview questions to share his insight into the key aspects of the question. By understanding these questions, you can learn how to approach any question more effectively. The basics of languages, algorithms and data structures are discussed as well as questions that explore how to write robust solutions after breaking down problems into manageable pieces. Harry also includes examples to focus on modeling and creative problem solving.

The skills that Harry teaches for problem solving can help you with your next interview and in your next job. Understanding better the key problem solving techniques that are analyzed in an interview can help you get the first job after university or make your next career move.

Matt Gibbs

Direct of Development, Asia Research & Development

Microsoft Corporation

推荐序一

海涛 2008 年在我的团队做过软件开发工程师。他是一个很细心的员工，对面试这个话题很感兴趣，经常和我及其他员工讨论，积累了很多面试方面的技巧和经验。他曾跟我提过想要写本有关面试的书，三年过后他把书写出来了！他是一个有目标、有耐心和持久力的人。

我在微软做了很多年的面试官，后面七年多作为把关面试官也面试了很多应聘者。应聘者要想做好面试，确实应把面试当作一门技巧来学习，更重要的是要提高自身的能力。我遇到很多应试者可能自身能力也不差但因为不懂得怎样回答提问，不能很好发挥。也有很多校园来的应聘者也学过数据结构和算法分析，可是到处理具体问题时不能用学过的知识来有效地解决问题。这些朋友读读海涛的这本书，会很受益，在面试中的发挥也会有很大提高。这本书也可以作为很好的教学补充资料，让学生不只学到书本知识，也学到解决问题的能力。

在向我汇报的员工中有面试发挥很好但工作平平的，也有面试一般但工作优秀的。对于追求职业发展的人来说，通过面试只是迈过一个门槛而不是目的，真正的较量是在入职后的成长。就像学钓鱼，你可能在有经验的垂钓者的指导下能钓到几条鱼，但如果没有学到垂钓的真谛，离开了指导者你可能就很难钓到很多鱼。我希望读这本书的朋友不要只学一些技巧来对付面试，而是通过学习如何解决面试中的难题来提高自己的编程和解决问题的能力，进而提高自信心，在职场中能迅速成长。

徐鹏阳（Pung Xu）

Principal Development Manager, Search Technology Center Asia

Microsoft

第 2 章

面试需要的基础知识

2.1 面试官谈基础知识

“C++的基础知识，如面向对象的特性、构造函数、析构函数、动态绑定等，能够反映出应聘者是否善于把握问题本质，有没有耐心深入一个问题。另外还有常用的设计模式、UML 图等，这些都能体现应聘者是否有软件工程方面的经验。”

——王海波（Autodesk，软件工程师）

“对基础知识的考查我特别重视 C++中对内存的使用管理。我觉得内存管理是 C++程序员特别要注意的，因为内存的使用和管理会影响程序的效率和稳定性。”

——蓝诚（Autodesk，软件工程师）

“基础知识反映了一个人的基本能力和基础素质，是以后工作中最核心的能力要求。我一般考查：(1) 数据结构和算法；(2) 编程能力；(3) 部分数学知识，如概率；(4) 问题的分析和推理能力。”

——张晓禹（百度，技术经理）

“我比较重视四块基础知识：(1) 编程基本功（特别喜欢字符串处理这一类的问题）；(2) 并发控制；(3) 算法、复杂度；(4) 语言的基本概念。”

——张珺（百度，高级软件工程师）

“我会考查编程基础、计算机系统基础知识、算法以及设计能力。这些是一个软件工程师的最基本的东西，这些方面表现出色的人，我们一般认为是有发展潜力的。”

——韩伟东（盛大，高级研究员）

“(1) 对 OS 的理解程度。这些知识对于工作中常遇到的内存管理、文件操作、程序性能、多线程、程序安全等有重要帮助。对于 OS 理解比较深入的人对于偏底层的工作上手一般比较快。(2) 对于一门编程语言的掌握程度。一个热爱编程的人应该会对某种语言有比较深入的了解。通常这样的人对于新的编程语言上手也比较快，而且理解比较深入。(3) 常用的算法和数据结构。不了解这些的程序员基本只能写写‘Hello World’。”

——陈黎明（微软，SDE II）

2.2 编程语言

程序员写代码总是基于某一种编程语言，因此技术面试的时候直接或者间接都会涉及至少一种编程语言。在面试的过程中，面试官要么直接问语言的语法，要么让应聘者用一种编程语言写代码解决一个问题，通过写出的代码来判断应聘者对他使用的语言的掌握程度。现在流行的编程语言很多，不同公司开发用的语言也不尽相同。做底层开发比如经常写驱动的人更习惯用 C，Linux 下有很多程序员用 C++开发应用程序，基于 Windows 的 C#项目已经越来越多，跨平台开发的程序员则可能更喜欢 Java，随着苹果 iPad、iPhone 的热销已经有很多程序员投向了 Objective C 的阵营，同时还有很多人喜欢用脚本语言如 Perl、Python 开发短小精致的小应用软件。因此，不同公司面试的时候对编程语言的要求也有所不同。每一种编程语言都可以写出一本大部头的书籍，本书限于篇幅不可能面面俱到。本书中所有代码都用 C/C++/C#实现，下面简要介绍一些 C++/C#常见的面试题。

2.2.1 C++

国内绝大部分高校都开设 C++的课程，因此绝大部分程序员都学过 C++，于是 C++成了各公司面试的首选编程语言。包括 Autodesk 在内的很多公司在面试的时候会有大量的 C++的语法题，其他公司虽然不直接面试 C++的语法，但面试题要求用 C++实现算法。因此总的说来，应聘者不管去什么公司求职，都应该在一定程度上掌握 C++。

通常语言面试有 3 种类型。第一种类型是面试官直接询问应聘者对 C++概念的理解。这种类型的问题，面试官特别喜欢了解应聘者对 C++关键字的理解程度。例如：在 C++中，有哪 4 个与类型转换相关的关键字？这些关键字各有什么特点，应该在什么场合下使用？

在这种类型的题目中，sizeof 是经常被问到的一个概念。比如下面的面试片段，就反复出现在各公司的技术面试中。

面试官：定义一个空的类型，里面没有任何成员变量和成员函数。对该类型求 sizeof，得到的结果是多少？

应聘者：答案是 1。

面试官：为什么不是 0？

应聘者：空类型的实例中不包含任何信息，本来求 sizeof 应该是 0，但是当我们声明该类型的实例的时候，它必须在内存中占有一定的空间，否则无法使用这些实例。至于占用多少内存，由编译器决定。Visual Studio 中每个空类型的实例占用 1 字节的空间。

面试官：如果在该类型中添加一个构造函数和析构函数，再对该类型求 sizeof，得到的结果又是多少？

应聘者：和前面一样，还是 1。调用构造函数和析构函数只需要知道函数的地址即可，而这些函数的地址只与类型相关，而与类型的实例无关，编译器也不会因为这两个函数而在实例内添加任何额外的信息。

面试官：那如果把析构函数标记为虚函数呢？

应聘者：C++的编译器一旦发现一个类型中有虚拟函数，就会为该类型生成虚函数表，并在该类型的每一个实例中添加一个指向虚函数表的指针。在 32 位的机器上，一个指针占 4 字节的空间，因此求 sizeof 得到 4；如果是 64 位的机器，一个指针占 8 字节的空间，因此求 sizeof 则得到 8。

面试 C/C++的第二种题型就是面试官拿出事先准备好的代码，让应聘者分析代码的运行结果。这种题型选择的代码通常包含比较复杂微妙的语言特性，这要求应聘者对 C++考点有着透彻的理解。即使应聘者对考点有一点点模糊，那么最终他得到的结果和实际运行的结果可能就会差距甚远。

比如面试官递给应聘者一张有如下代码的 A4 打印纸要求他分析编译运行的结果，并提供 3 个选项：A．编译错误；B．编译成功，运行时程序崩溃；C．编译运行正常，输出 10。

```
class A
{
private:
   int value;

public:
```

```
    A(int n) { value = n; }
    A(A other) { value = other.value; }

    void Print() { std::cout << value << std::endl; }
};

int _tmain(int argc, _TCHAR* argv[])
{
    A a = 10;
    A b = a;
    b.Print();

    return 0;
}
```

在上述代码中，复制构造函数 A(A other)传入的参数是 A 的一个实例。由于是传值参数，我们把形参复制到实参会调用复制构造函数。因此如果允许复制构造函数传值，就会在复制构造函数内调用复制构造函数，就会形成永无休止的递归调用从而导致栈溢出。因此 C++的标准不允许复制构造函数传值参数，在 Visual Studio 和 GCC 中，都将编译出错。要解决这个问题，我们可以把构造函数修改为 A(const A& other)，也就是把传值参数改成常量引用。

第三种题型就是要求应聘者写代码定义一个类型或者实现类型中的成员函数。让应聘者写代码的难度自然比让应聘者分析代码要高不少，因为能想明白的未必就能写得清楚。很多考查 C++语法的代码题围绕在构造函数、析构函数及运算符重载。比如面试题 1“赋值运算符函数”就是一个例子。

为了让大家能顺利地通过C++面试，更重要的是能更好地学习掌握C++这门编程语言，这里推荐几本 C++的书，大家可以根据自己的具体情况选择阅读的顺序：

- **《Effective C++》**。这本书很适合在面试之前突击 C++。这本书列举了使用 C++经常出现的问题及解决这些问题的技巧。该书中提到的问题也是面试官很喜欢问的问题。
- **《C++ Primer》**。读完这本书，就会对 C++的语法有全面的了解。
- **《Inside C++ Object Model》**。这本书有助于我们深入了解 C++对象的内部。读懂这本书后很多 C++难题，比如前面的 sizeof 的问题、虚函数的调用机制等，都会变得很容易。

- 《**The C++ Programming Language**》。如果是想全面深入掌握 C++，没有哪本书比这本书更适合的了。

面试题 1：赋值运算符函数

题目：如下为类型 CMyString 的声明，请为该类型添加赋值运算符函数。

```
class CMyString
{
public:
    CMyString(char* pData = NULL);
    CMyString(const CMyString& str);
    ~CMyString(void);

private:
    char* m_pData;
};
```

当面试官要求应聘者定义一个赋值运算符函数时，他会在检查应聘者写出的代码时关注如下几点：

- 是否把返回值的类型声明为该类型的引用，并在函数结束前返回实例自身的引用（即*this）。只有返回一个引用，才可以允许连续赋值。否则如果函数的返回值是 void，应用该赋值运算符将不能做连续赋值。假设有 3 个 CMyString 的对象：str1、str2 和 str3，在程序中语句 str1=str2=str3 将不能通过编译。
- 是否把传入的参数的类型声明为常量引用。如果传入的参数不是引用而是实例，那么从形参到实参会调用一次复制构造函数。把参数声明为引用可以避免这样的无谓消耗，能提高代码的效率。同时，我们在赋值运算符函数内不会改变传入的实例的状态，因此应该为传入的引用参数加上 const 关键字。
- 是否释放实例自身已有的内存。如果我们忘记在分配新内存之前释放自身已有的空间，程序将出现内存泄露。
- 是否判断传入的参数和当前的实例（*this）是不是同一个实例。如果是同一个，则不进行赋值操作，直接返回。如果事先不判断就进行赋值，那么在释放实例自身的内存的时候就会导致严重的问题：当*this 和传入的参数是同一个实例时，那么一旦释放了自身的内存，传入的参数的内存也同时被释放了，因此再也找不到需要赋值的内容了。

❖ 经典的解法，适用于初级程序员

当我们完整地考虑了上述 4 个方面之后，我们可以写出如下的代码：

```
CMyString& CMyString::operator =(const CMyString &str)
{
    if(this == &str)
        return *this;

    delete []m_pData;
    m_pData = NULL;

    m_pData = new char[strlen(str.m_pData) + 1];
    strcpy(m_pData, str.m_pData);

    return *this;
}
```

这是一般 C++教材上提供的参考代码。如果接受面试的是应届毕业生或者 C++初级程序员，能全面地考虑到前面四点并完整地写出代码，面试官可能会让他通过这轮面试。但如果面试的是 C++高级程序员，面试官可能会提出更高的要求。

❖ 考虑异常安全性的解法，高级程序员必备

在前面的函数中，我们在分配内存之前先用 delete 释放了实例 m_pData 的内存。如果此时内存不足导致 new char 抛出异常，m_pData 将是一个空指针，这样非常容易导致程序崩溃。也就是说一旦在赋值运算符函数内部抛出一个异常，CMyString 的实例不再保持有效的状态，这就违背了异常安全性（Exception Safety）原则。

要想在赋值运算符函数中实现异常安全性，我们有两种方法。一个简单的办法是我们先用 new 分配新内容再用 delete 释放已有的内容。这样只在分配内容成功之后再释放原来的内容，也就是当分配内存失败时我们能确保 CMyString 的实例不会被修改。我们还有一个更好的办法是先创建一个临时实例，再交换临时实例和原来的实例。下面是这种思路的参考代码：

```
CMyString& CMyString::operator =(const CMyString &str)
{
    if(this != &str)
    {
        CMyString strTemp(str);

        char* pTemp = strTemp.m_pData;
        strTemp.m_pData = m_pData;
        m_pData = pTemp;
    }
```

```
    return *this;
}
```

在这个函数中，我们先创建一个临时实例 strTemp，接着把 strTemp.m_pData 和实例自身的 m_pData 做交换。由于 strTemp 是一个局部变量，但程序运行到 if 的外面时也就出了该变量的作用域，就会自动调用 strTemp 的析构函数，把 strTemp.m_pData 所指向的内存释放掉。由于 strTemp.m_pData 指向的内存就是实例之前 m_pData 的内存，这就相当于自动调用析构函数释放实例的内存。

在新的代码中，我们在 CMyString 的构造函数里用 new 分配内存。如果由于内存不足抛出诸如 bad_alloc 等异常，我们还没有修改原来实例的状态，因此实例的状态还是有效的，这也就保证了异常安全性。

如果应聘者在面试的时候能够考虑到这个层面，面试官就会觉得他对代码的异常安全性有很深的理解，那么他自然也就能通过这轮面试了。

源代码：

本题完整的源代码详见 01_AssignmentOperator 项目。

测试用例：

- 把一个 CMyString 的实例赋值给另外一个实例。
- 把一个 CMyString 的实例赋值给它自己。
- 连续赋值。

本题考点：

- 考查对 C++的基础语法的理解，如运算符函数、常量引用等。
- 考查对内存泄露的理解。
- 对高级 C++程序员，面试官还将考查应聘者对代码异常安全性的理解。

2.2.2　C#

C#是微软在推出新的开发平台.NET 时同步推出的编程语言。由于 Windows 至今仍然是用户最多的操作系统，而.NET 又是微软近年来力推的开发平台，因此 C#无论在桌面软件还是网络应用的开发上都有着广泛的应用，所以我们也不难理解为什么现在很多基于 Windows 系统开发的公司都会要求应聘者掌握 C#。

C#可以看成是一门以 C++为基础发展起来的一种托管语言，因此它的很多关键字甚至语法都和 C++很类似。对一个学习过 C++编程的程序员而言，他用不了多长时间学习就能用 C#来开发软件。然而我们也要清醒地认识到，虽然学习 C#与 C++相同或者类似的部分很容易，但要掌握并区分两者不同的地方却不是一件很容易的事情。面试官总是喜欢深究我们模棱两可的地方以考查我们是不是真的理解了，因此我们要着重注意 C#与 C++不同的语法特点。下面的面试片段就是一个例子：

面试官：C++中可以用 struct 和 class 来定义类型。这两种类型有什么区别？

应聘者：如果没有标明成员函数或者成员变量的访问权限级别，在 struct 中默认的是 public，而在 class 中默认的是 private。

面试官：那在 C#中呢？

应聘者：C#和 C++不一样。在 C#中如果没有标明成员函数或者成员变量的访问权限级别，struct 和 class 中都是 private 的。struct 和 class 的区别是 struct 定义的是值类型，值类型的实例在栈上分配内存；而 class 定义的是引用类型，引用类型的实例在堆上分配内存。

在 C#中，每个类型中和 C++一样，都有构造函数。但和 C++不同的是，我们在 C#中可以为类型定义一个 Finalizer 和 Dispose 方法以释放资源。Finalizer 方法虽然写法与 C++的析构函数看起来一样，都是后面跟类型名字，但与 C++析构函数的调用时机是确定的不同，C#的 Finalizer 是在运行时（CLR）做垃圾回收时才会被调用，它的调用时机是由运行时决定的，因此对程序员来说是不确定的。另外，在 C#中可以为类型定义一个特殊的构造函数：静态构造函数。这个函数的特点是在类型第一次被使用之前由运行时自动调用，而且保证只调用一次。关于静态构造函数，我们有很多有意思的面试题，比如运行下面的 C#代码，输出的结果是什么？

```
class A
```

```
{
    public A(string text)
    {
        Console.WriteLine(text);
    }
}

class B
{
    static A a1 = new A("a1");
    A a2 = new A("a2");

    static B()
    {
        a1 = new A("a3");
    }

    public B()
    {
        a2 = new A("a4");
    }
}

class Program
{
    static void Main(string[] args)
    {
        B b = new B();
    }
}
```

在调用类型 B 的代码之前先执行 B 的静态构造函数。静态构造函数先初始化类型的静态变量，再执行函数体内的语句。因此先打印 a1 再打印 a3。接下来执行 B b = new B()，即调用 B 的普通构造函数。构造函数先初始化成员变量，再执行函数体内的语句，因此先后打印出 a2、a4。因此运行上面的代码，得到的结果将是打印出 4 行，分别是 a1、a3、a2、a4。

我们除了要关注 C#和 C++不同的知识点之外，还要格外关注 C#一些特有的功能，比如反射、应用程序域（AppDomain）等。这些概念还相互关联，要花很多时间学习研究才能透彻地理解它们。下面的代码就是一段关于反射和应用程序域的代码，运行它得到的结果是什么？

```
[Serializable]
internal class A : MarshalByRefObject
{
    public static int Number;

    public void SetNumber(int value)
    {
        Number = value;
    }
```

```
}

[Serializable]
internal class B
{
    public static int Number;

    public void SetNumber(int value)
    {
        Number = value;
    }
}

class Program
{
    static void Main(string[] args)
    {
        String assambly = Assembly.GetEntryAssembly().FullName;
        AppDomain domain = AppDomain.CreateDomain("NewDomain");

        A.Number = 10;
        String nameOfA = typeof(A).FullName;
        A a = domain.CreateInstanceAndUnwrap(assambly, nameOfA) as A;
        a.SetNumber(20);
        Console.WriteLine("Number in class A is {0}", A.Number);

        B.Number = 10;
        String nameOfB = typeof(B).FullName;
        B b = domain.CreateInstanceAndUnwrap(assambly, nameOfB) as B;
        b.SetNumber(20);
        Console.WriteLine("Number in class B is {0}", B.Number);
    }
}
```

上述 C#代码先创建一个名为 NewDomain 的应用程序域，并在该域中利用反射机制创建类型 A 的一个实例和类型 B 的一个实例。我们注意到类型 A 是继承自 MarshalByRefObject，而 B 不是。虽然这两个类型的结构一样，但由于基类不同而导致在跨越应用程序域的边界时表现出的行为将大不相同。

先考虑 A 的情况。由于 A 继承自 MarshalByRefObject，那么 a 实际上只是在默认的域中的一个代理实例（Proxy），它指向位于 NewDomain 域中的 A 的一个实例。当调用 a 的方法 SetNumber 时，是在 NewDomain 域中调用该方法，它将修改 NewDomain 域中静态变量 A.Number 的值并设为 20。由于静态变量在每个应用程序域中都有一份独立的拷贝，修改 NewDomain 域中的静态变量 A.Number 对默认域中的静态变量 A.Number 没有任何影响。由于 Console.WriteLine 是在默认的应用程序域中输出 A.Number，因此输出仍然是 10。

接着讨论 B。由于 B 只是从 Object 继承而来的类型，它的实例穿越应用程序域的边界时，将会完整地复制实例。因此在上述代码中，我们尽管试图在 NewDomain 域中生成 B 的实例，但会把实例 b 复制到默认的应用程序域。此时调用方法 b.SetNumber 也是在缺省的应用程序域上进行，它将修改默认的域上的 A.Number 并设为 20。再在默认的域上调用 Console.WriteLine 时，它将输出 20。

下面推荐两本 C#相关的书籍，以方便大家应对 C#面试并学习好 C#。

- **《Professional C#》**。这本书最大的特点是在附录中有几章专门写给已经有其他语言（如 VB、C++和 Java）经验的程序员，它详细讲述了 C#和其他语言的区别，看了这几章之后就不会把 C#和之前掌握的语言相混淆。
- Jeffrey Richter 的**《CLR Via C#》**。该书不仅深入地介绍了 C#语言，同时对 CLR 及.NET 做了全面的剖析。如果能够读懂这本书，那么我们就能深入理解装箱卸箱、垃圾回收、反射等概念，知其然的同时也能知其所以然，通过 C#相关的面试自然也就不难了。

面试题 2：实现 Singleton 模式

题目：设计一个类，我们只能生成该类的一个实例。

只能生成一个实例的类是实现了 Singleton（单例）模式的类型。由于设计模式在面向对象程序设计中起着举足轻重的作用，在面试过程中很多公司都喜欢问一些与设计模式相关的问题。在常用的模式中，Singleton 是唯一一个能够用短短几十行代码完整实现的模式。因此，写一个 Singleton 的类型是一个很常见的面试题。

❖ 不好的解法一：只适用于单线程环境

由于要求只能生成一个实例，因此我们必须把构造函数设为私有函数以禁止他人创建实例。我们可以定义一个静态的实例，在需要的时候创建该实例。下面定义类型 Singleton1 就是基于这个思路的实现：

```
public sealed class Singleton1
{
   private Singleton1()
   {
```

```
    }

    private static Singleton1 instance = null;
    public static Singleton1 Instance
    {
        get
        {
            if (instance == null)
                instance = new Singleton1();

            return instance;
        }
    }
}
```

上述代码在 Singleton 的静态属性 Instance 中，只有在 instance 为 null 的时候才创建一个实例以避免重复创建。同时我们把构造函数定义为私有函数，这样就能确保只创建一个实例。

❖ 不好的解法二：虽然在多线程环境中能工作但效率不高

解法一中的代码在单线程的时候工作正常，但在多线程的情况下就有问题了。设想如果两个线程同时运行到判断 instance 是否为 null 的 if 语句，并且 instance 的确没有创建时，那么两个线程都会创建一个实例，此时类型 Singleton1 就不再满足单例模式的要求了。为了保证在多线程环境下我们还是只能得到类型的一个实例，需要加上一个同步锁。把 Singleton1 稍做修改得到了如下代码：

```
public sealed class Singleton2
{
    private Singleton2()
    {
    }

    private static readonly object syncObj = new object();

    private static Singleton2 instance = null;
    public static Singleton2 Instance
    {
        get
        {
            lock (syncObj)
            {
                if (instance == null)
                    instance = new Singleton2();
            }

            return instance;
        }
```

```
    }
}
```

我们还是假设有两个线程同时想创建一个实例。由于在一个时刻只有一个线程能得到同步锁，当第一个线程加上锁时，第二个线程只能等待。当第一个线程发现实例还没有创建时，它创建出一个实例。接着第一个线程释放同步锁，此时第二个线程可以加上同步锁，并运行接下来的代码。这个时候由于实例已经被第一个线程创建出来了，第二个线程就不会重复创建实例了，这样就保证了我们在多线程环境中也只能得到一个实例。

但是类型 Singleton2 还不是很完美。我们每次通过属性 Instance 得到 Singleton2 的实例，都会试图加上一个同步锁，而加锁是一个非常耗时的操作，在没有必要的时候我们应该尽量避免。

❖ 可行的解法：加同步锁前后两次判断实例是否已存在

我们只是在实例还没有创建之前需要加锁操作，以保证只有一个线程创建出实例。而当实例已经创建之后，我们已经不需要再做加锁操作了。于是我们可以把解法二中的代码再做进一步的改进：

```
public sealed class Singleton3
{
    private Singleton3()
    {
    }

    private static object syncObj = new object();

    private static Singleton3 instance = null;
    public static Singleton3 Instance
    {
        get
        {
            if (instance == null)
            {
                lock (syncObj)
                {
                    if (instance == null)
                        instance = new Singleton3();
                }
            }

            return instance;
        }
    }
}
```

Singleton3 中只有当 instance 为 null 即没有创建时，需要加锁操作。当 instance 已经创建出来之后，则无须加锁。因为只在第一次的时候 instance 为 null，因此只在第一次试图创建实例的时候需要加锁。这样 Singleton3 的时间效率比 Singleton2 要好很多。

Singleton3 用加锁机制来确保在多线程环境下只创建一个实例，并且用两个 if 判断来提高效率。这样的代码实现起来比较复杂，容易出错，我们还有更加优秀的解法。

❖ 强烈推荐的解法一：利用静态构造函数

C#的语法中有一个函数能够确保只调用一次，那就是静态构造函数，我们可以利用 C#这个特性实现单例模式如下：

```
public sealed class Singleton4
{
    private Singleton4()
    {
    }

    private static Singleton4 instance = new Singleton4();
    public static Singleton4 Instance
    {
        get
        {
            return instance;
        }
    }
}
```

Singleton4 的实现代码非常简洁。我们在初始化静态变量 instance 的时候创建一个实例。由于 C#是在调用静态构造函数时初始化静态变量，.NET 运行时能够确保只调用一次静态构造函数，这样我们就能够保证只初始化一次 instance。

C#中调用静态构造函数的时机不是由程序员掌控的，而是当.NET 运行时发现第一次使用一个类型的时候自动调用该类型的静态构造函数。因此在 Singleton4 中，实例 instance 并不是第一次调用属性 Singleton4.Instance 的时候创建，而是在第一次用到 Singleton4 的时候就会被创建。假设我们在 Singleton4 中添加一个静态方法，调用该静态函数是不需要创建一个实例的，但如果按照 Singleton4 的方式实现单例模式，则仍然会过早地创建实例，从而降低内存的使用效率。

❖ 强烈推荐的解法二：实现按需创建实例

最后的一个实现 Singleton5 则很好地解决了 Singleton4 中的实例创建时机过早的问题：

```
public sealed class Singleton5
{
    Singleton5()
    {
    }

    public static Singleton5 Instance
    {
        get
        {
            return Nested.instance;
        }
    }

    class Nested
    {
        static Nested()
        {
        }

        internal static readonly Singleton5 instance = new Singleton5();
    }
}
```

在上述 Singleton5 的代码中，我们在内部定义了一个私有类型 Nested。当第一次用到这个嵌套类型的时候，会调用静态构造函数创建 Singleton5 的实例 instance。类型 Nested 只在属性 Singleton5.Instance 中被用到，由于其私有属性他人无法使用 Nested 类型。因此当我们第一次试图通过属性 Singleton5.Instance 得到 Singleton5 的实例时，会自动调用 Nested 的静态构造函数创建实例 instance。如果我们不调用属性 Singleton5.Instance，那么就不会触发.NET 运行时调用 Nested，也不会创建实例，这样就真正做到了按需创建。

❖ 解法比较

在前面的 5 种实现单例模式的方法中，第一种方法在多线程环境中不能正常工作，第二种模式虽然能在多线程环境中正常工作但时间效率很低，都不是面试官期待的解法。在第三种方法中我们通过两次判断一次加锁确保在多线程环境能高效率地工作。第四种方法利用 C#的静态构造函数的特性，确保只创建一个实例。第五种方法利用私有嵌套类型的特性，做到只

在真正需要的时候才会创建实例，提高空间使用效率。如果在面试中给出第四种或者第五种解法，毫无疑问会得到面试官的青睐。

源代码：

本题完整的源代码详见 02_Singleton 项目。

本题考点：

- 考查对单例（Singleton）模式的理解。
- 考查对 C#的基础语法的理解，如静态构造函数等。
- 考查对多线程编程的理解。

本题扩展：

在前面的代码中，5 种单例模式的实现把类型标记为 sealed，表示它们不能作为其他类型的基类。现在我们要求定义一个表示总统的类型 President，可以从该类型继承出 FrenchPresident 和 AmericanPresident 等类型。这些派生类型都只能产生一个实例。请问该如何设计实现这些类型？

2.3 数据结构

数据结构一直是技术面试的重点，大多数面试题都是围绕着数组、字符串、链表、树、栈及队列这几种常见的数据结构展开的，因此每一个应聘者都要熟练掌握这几种数据结构。

数组和字符串是两种最基本的数据结构，它们用连续内存分别存储数字和字符。链表和树是面试中出现频率最高的数据结构。由于操作链表和树需要操作大量的指针，应聘者在解决相关问题的时候一定要留意代码的鲁棒性，否则容易出现程序崩溃的问题。栈是一个与递归紧密相关的数据

结构，同样队列也与广度优先遍历算法紧密相关。深刻理解这两种数据结构能帮助我们解决很多算法问题。

2.3.1 数组

数组可以说是最简单的一种数据结构，它占据一块连续的内存并按照顺序存储数据。创建数组时，我们需要首先指定数组的容量大小，然后根据大小分配内存。即使我们只在数组中存储一个数字，也需要为所有的数据预先分配内存。因此数组的空间效率不是很好，经常会有空闲的区域没有得到充分利用。

由于数组中的内存是连续的，于是可以根据下标在 O(1)时间读/写任何元素，因此它的时间效率是很高的。我们可以根据数组时间效率高的优点，用数组来实现简单的哈希表：把数组的下标设为哈希表的键值（Key），而把数组中的每一个数字设为哈希表的值（Value），这样每一个下标及数组中该下标对应的数字就组成了一个键值-值的配对。有了这样的哈希表，我们就可以在 O(1)实现查找，从而可以快速高效地解决很多问题。面试题 35“第一个只出现一次的字母”就是一个很好的例子。

为了解决数组空间效率不高的问题，人们又设计实现了多种动态数组，比如 C++的 STL 中的 vector。为了避免浪费，我们先为数组开辟较小的空间，然后往数组中添加数据。当数据的数目超过数组的容量时，我们再重新分配一块更大的空间（STL 的 vector 每次扩充容量时，新的容量都是前一次的两倍），把之前的数据复制到新的数组中，再把之前的内存释放，这样就能减少内存的浪费。但我们也注意到每一次扩充数组容量时都有大量的额外操作，这对时间性能有负面影响，因此使用动态数组时要尽量减少改变数组容量大小的次数。

在 C/C++中，数组和指针是相互关联又有区别的两个概念。当我们声明一个数组时，其数组的名字也是一个指针，该指针指向数组的第一个元素。我们可以用一个指针来访问数组。但值得注意的是，C/C++没有记录数组的大小，因此用指针访问数组中的元素时，程序员要确保没有超出数组的边界。下面通过一个例子来了解数组和指针的区别。运行下面的代码，请问输出是什么？

```
int GetSize(int data[])
{
    return sizeof(data);
}
```

```
int _tmain(int argc, _TCHAR* argv[])
{
    int data1[] = {1, 2, 3, 4, 5};
    int size1 = sizeof(data1);

    int* data2 = data1;
    int size2 = sizeof(data2);

    int size3 = GetSize(data1);

    printf("%d, %d, %d", size1, size2, size3);
}
```

答案是输出“20, 4, 4”。data1 是一个数组，sizeof(data1)是求数组的大小。这个数组包含 5 个整数，每个整数占 4 字节，因此总共是 20 字节。data2 声明为指针，尽管它指向了数组 data1 的第一个数字，但它的本质仍然是一个指针。在 32 位系统上，对任意指针求 sizeof，得到的结果都是 4。在 C/C++中，当数组作为函数的参数进行传递时，数组就自动退化为同类型的指针。因此尽管函数 GetSize 的参数 data 被声明为数组，但它会退化为指针，size3 的结果仍然是 4。

面试题 3：二维数组中的查找

题目：在一个二维数组中，每一行都按照从左到右递增的顺序排序，每一列都按照从上到下递增的顺序排序。请完成一个函数，输入这样的一个二维数组和一个整数，判断数组中是否含有该整数。

例如下面的二维数组就是每行、每列都递增排序。如果在这个数组中查找数字 7，则返回 true；如果查找数字 5，由于数组不含有该数字，则返回 false。

1	2	8	9
2	4	9	12
4	7	10	13
6	8	11	15

在分析这个问题的时候，很多应聘者都会把二维数组画成矩形，然后从数组中选取一个数字，分 3 种情况来分析查找的过程。当数组中选取的数字刚好和要查找的数字相等时，就结束查找过程。如果选取的数字小于要查找的数字，那么根据数组排序的规则，要查找的数字应该在当前选取的位置的右边或者下边（如图 2.1（a）所示）。同样，如果选取的数字大于

要查找的数字，那么要查找的数字应该在当前选取的位置的上边或者左边（如图 2.1（b）所示）。

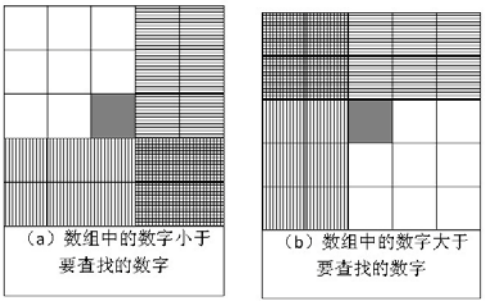


图 2.1　二维数组中的查找

注：在数组中间选择一个数（深色方格），根据它的大小判断要查找的数字可能出现的区域（阴影部分）。

在上面的分析中，由于要查找的数字相对于当前选取的位置有可能在两个区域中出现，而且这两个区域还有重叠，这问题看起来就复杂了，于是很多人就卡在这里束手无策了。

当我们需要解决一个复杂的问题时，一个很有效的办法就是从一个具体的问题入手，通过分析简单具体的例子，试图寻找普遍的规律。针对这个问题，我们不妨也从一个具体的例子入手。下面我们以在题目中给出的数组中查找数字 7 为例来一步步分析查找的过程。

前面我们之所以遇到难题，是因为我们在二维数组的中间选取一个数字来和要查找的数字做比较，这样导致下一次要查找的是两个相互重叠的区域。如果我们从数组的一个角上选取数字来和要查找的数字做比较，情况会不会变简单呢？

首先我们选取数组右上角的数字 9。由于 9 大于 7，并且 9 还是第 4 列的第一个（也是最小的）数字，因此 7 不可能出现在数字 9 所在的列。于是我们把这一列从需要考虑的区域内剔除，之后只需要分析剩下的 3 列（如图 2.2（a）所示）。在剩下的矩阵中，位于右上角的数字是 8。同样 8 大于 7，因此 8 所在的列我们也可以剔除。接下来我们只要分析剩下的两列即可（如图 2.2（b）所示）。

在由剩余的两列组成的数组中，数字 2 位于数组的右上角。2 小于 7，那么要查找的 7 可能在 2 的右边，也有可能在 2 的下边。在前面的步骤中，我们已经发现 2 右边的列都已经被剔除了，也就是说 7 不可能出现在 2 的右边，因此 7 只有可能出现在 2 的下边。于是我们把数字 2 所在的行也剔除，只分析剩下的三行两列数字（如图 2.2（c）所示）。在剩下的数字中，数字 4 位于右上角，和前面一样，我们把数字 4 所在的行也删除，最后剩下两行两列数字（如图 2.2（d）所示）。

在剩下的两行两列 4 个数字中，位于右上角的刚好就是我们要查找的数字 7，于是查找过程就可以结束了。

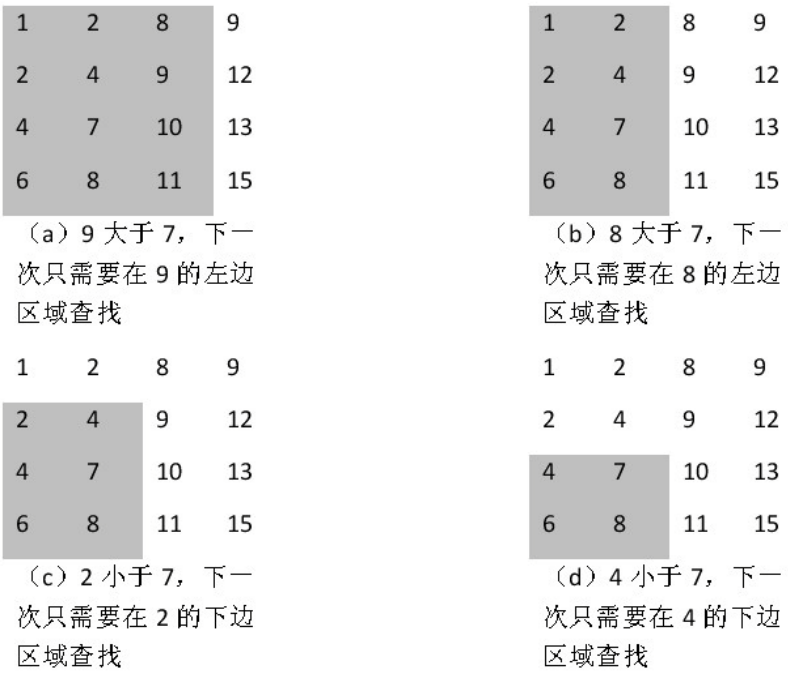


图 2.2　在二维数组中查找 7 的步骤

注：矩阵中加阴影背景的区域是下一步查找的范围。

总结上述查找的过程，我们发现如下规律：首先选取数组中右上角的数字。如果该数字等于要查找的数字，查找过程结束；如果该数字大于要查找的数字，剔除这个数字所在的列；如果该数字小于要查找的数字，剔除这个数字所在的行。也就是说如果要查找的数字不在数组的右上角，则

每一次都在数组的查找范围中剔除一行或者一列，这样每一步都可以缩小查找的范围，直到找到要查找的数字，或者查找范围为空。

把整个查找过程分析清楚之后，我们再写代码就不是一件很难的事情了。下面是上述思路对应的参考代码：

```
bool Find(int* matrix, int rows, int columns, int number)
{
    bool found = false;

    if(matrix != NULL && rows > 0 && columns > 0)
    {
        int row = 0;
        int column = columns - 1;
        while(row < rows && column >=0)
        {
            if(matrix[row * columns + column] == number)
            {
                found = true;
                break;
            }
            else if(matrix[row * columns + column] > number)
                -- column;
            else
                ++ row;
        }
    }

    return found;
}
```

在前面的分析中，我们每一次都是选取数组查找范围内的右上角数字。同样，我们也可以选取左下角的数字。感兴趣的读者不妨自己分析一下每次都选取左下角的查找过程。但我们不能选择左上角或者右下角。以左上角为例，最初数字 1 位于初始数组的左上角，由于 1 小于 7，那么 7 应该位于 1 的右边或者下边。此时我们既不能从查找范围内剔除 1 所在的行，也不能剔除 1 所在的列，这样我们就无法缩小查找的范围。

源代码：

本题完整的源代码详见 03_FindInPartiallySortedMatrix 项目。

测试用例：

- 二维数组中包含查找的数字（查找的数字是数组中的最大值和最小值，查找的数字介于数组中的最大值和最小值之间）。
- 二维数组中没有查找的数字（查找的数字大于数组中的最大值，查找的数字小于数组中的最小值，查找的数字在数组的最大值和最小值之间但数组中没有这个数字）。
- 特殊输入测试（输入空指针）。

本题考点：

- 考查应聘者对二维数组的理解及编程能力。二维数组在内存中占据连续的空间。在内存中从上到下存储各行元素，在同一行中按照从左到右的顺序存储。因此我们可以根据行号和列号计算出相对于数组首地址的偏移量，从而找到对应的元素。
- 考查应聘者分析问题的能力。当应聘者发现问题比较复杂时，能不能通过具体的例子找出其中的规律，是能否解决这个问题的关键所在。这个题目只要从一个具体的二维数组的右上角开始分析，就能找到查找的规律，从而找到解决问题的突破口。

2.3.2 字符串

字符串是由若干字符组成的序列。由于字符串在编程时使用的频率非常高，为了优化，很多语言都对字符串做了特殊的规定。下面分别讨论 C/C++和 C#中字符串的特性。

C/C++ 中每个字符串都以字符'\0'作为结尾，这样我们就能很方便地找到字符串的最后尾部。但由于这个特点，每个字符串中都有一个额外字符的开销，稍不留神就会造成字符串的越界。比如下面的代码：

```
char str[10];
strcpy(str, "0123456789");
```

我们先声明一个长度为 10 的字符数组，然后把字符串"0123456789"复制到数组中。"0123456789"这个字符串看起来只有 10 个字符，但实际上它的末尾还有一个'\0'字符，因此它的实际长度为 11 个字节。要正确地复制该字符串，至少需要一个长度为 11 个字节的数组。

为了节省内存，C/C++把常量字符串放到单独的一个内存区域。当几个指针赋值给相同的常量字符串时，它们实际上会指向相同的内存地址。但用常量内存初始化数组，情况却有所不同。下面通过一个面试题来学习这一知识点。运行下面的代码，得到的结果是什么？

```
int _tmain(int argc, _TCHAR* argv[])
{
   char str1[] = "hello world";
   char str2[] = "hello world";

   char* str3 = "hello world";
   char* str4 = "hello world";

   if(str1 == str2)
      printf("str1 and str2 are same.\n");
   else
      printf("str1 and str2 are not same.\n");

   if(str3 == str4)
      printf("str3 and str4 are same.\n");
   else
      printf("str3 and str4 are not same.\n");

   return 0;
}
```

str1 和 str2 是两个字符串数组，我们会为它们分配两个长度为 12 个字节的空间，并把"hello world"的内容分别复制到数组中去。这是两个初始地址不同的数组，因此 str1 和 str2 的值也不相同，所以输出的第一行是”str1 and str2 are not same”。

str3 和 str4 是两个指针，我们无须为它们分配内存以存储字符串的内容，而只需要把它们指向"hello world”在内存中的地址就可以了。由于"hello world”是常量字符串，它在内存中只有一个拷贝，因此 str3 和 str4 指向的是同一个地址。所以比较 str3 和 str4 的值得到的结果是相同的，输出的第二行是”str3 and str4 are same”。

在 C#中，封装字符串的类型 System.String 有一个非常特殊的性质：String 中的内容是不能改变的。一旦试图改变 String 的内容，就会产生一个新的实例。请看下面的 C#代码：

```
String str = "hello";
str.ToUpper();
str.Insert(0, " WORLD");
```

虽然我们对 str 做了 ToUpper 和 Insert 两个操作，但操作的结果都是生成一个新的 String 实例并在返回值中返回，str 本身的内容都不会发生改变，因此最终 str 的值仍然是"hello"。由此可见，如果试图改变 String 的内容，改变之后的值只可以通过返回值得到。用 String 作连续多次修改，每一次修改都会产生一个临时对象，这样开销太大会影响效率。为此 C#定义了一个新的与字符串相关的类型 StringBuilder，它能容纳修改后的结果。因此如果要连续多次修改字符串内容，用 StringBuilder 是更好的选择。

和修改 String 内容类似，如果我们试图把一个常量字符串赋值给一个 String 实例，也不是把 String 的内容改成赋值的字符串，而是生成一个新的 String 实例。请看下面的代码：

```
class Program
{
    internal static void ValueOrReference(Type type)
    {
        String result = "The type " + type.Name;

        if (type.IsValueType)
            Console.WriteLine(result + " is a value type.");
        else
            Console.WriteLine(result + " is a reference type.");
    }

    internal static void ModifyString(String text)
    {
        text = "world";
    }

    static void Main(string[] args)
    {
        String text = "hello";

        ValueOrReference(text.GetType());
        ModifyString(text);

        Console.WriteLine(text);
    }
}
```

在上面的代码中，我们先判断 String 是值类型还是引用类型。类型 String 的定义是 public sealed class String {...}。既然是 class，那么 String 自然就是引用类型。接下来在方法 ModifyString 里，对 text 赋值一个新的字符串。我们要记得 text 的内容是不能被修改的。此时会先生成一个新的内容是"world"的 String 实例，然后把 text 指向这个新的实例。由于参数 text 没有加 ref 或者 out，出了方法 ModifyString 之后，text 还是指向原来的字符串，

因此输出仍然是"hello"。要想实现出了函数之后 text 变成"world"的效果，我们必须把参数 text 标记 ref 或者 out。

面试题 4：替换空格

题目：请实现一个函数，把字符串中的每个空格替换成"%20"。例如输入“We are happy.”，则输出“We%20are%20happy.”。

在网络编程中，如果 URL 参数中含有特殊字符，如空格、'#'等，可能导致服务器端无法获得正确的参数值。我们需要将这些特殊符号转换成服务器可以识别的字符。转换的规则是在'%'后面跟上 ASCII 码的两位十六进制的表示。比如空格的 ASCII 码是 32，即十六进制的 0x20，因此空格被替换成"%20"。再比如'#'的 ASCII 码为 35，即十六进制的 0x23，它在 URL 中被替换为"%23"。

看到这个题目，我们首先应该想到的是原来一个空格字符，替换之后变成'%'、'2'和'0'这 3 个字符，因此字符串会变长。如果是在原来的字符串上做替换，那么就有可能覆盖修改在该字符串后面的内存。如果是创建新的字符串并在新的字符串上做替换，那么我们可以自己分配足够多的内存。由于有两种不同的解决方案，我们应该向面试官问清楚，让他明确告诉我们他的需求。假设面试官让我们在原来的字符串上做替换，并且保证输入的字符串后面有足够多的空余内存。

❖ 时间复杂度为O(n²)的解法，不足以拿到Offer

现在我们考虑怎么做替换操作。最直观的做法是从头到尾扫描字符串，每一次碰到空格字符的时候做替换。由于是把 1 个字符替换成 3 个字符，我们必须要把空格后面所有的字符都后移两个字节，否则就有两个字符被覆盖了。

举个例子，我们从头到尾把"We are happy."中的每一个空格替换成"%20"。为了形象起见，我们可以用一个表格来表示字符串，表格中的每个格子表示一个字符（如图 2.3（a）所示）。

图 2.3 从前往后把字符串中的空格替换成'%20'的过程

注：（a）字符串"We are happy."。（b）把字符串中的第一个空格替换成'%20'。灰色背景表示需要移动的字符。（c）把字符串中的第二个空格替换成'%20'。浅灰色背景表示需要移动一次的字符，深灰色背景表示需要移动两次的字符。

我们替换第一个空格，这个字符串变成图 2.3（b）中的内容，表格中灰色背景的格子表示需要做移动的区域。接着我们替换第二个空格，替换之后的内容如图 2.3(c)所示。同时，我们注意到用深灰色背景标注的"happy"部分被移动了两次。

假设字符串的长度是n。对每个空格字符，需要移动后面O(n)个字符，因此对含有O(n)个空格字符的字符串而言总的时间效率是$O(n^2)$。

当我们把这种思路阐述给面试官后，他不会就此满意，他将让我们寻找更快的方法。在前面的分析中，我们发现数组中很多字符都移动了很多次，能不能减少移动次数呢？答案是肯定的。我们换一种思路，把从前向后替换改成从后向前替换。

❖ 时间复杂度为 O(n)的解法，搞定 Offer 就靠它了

我们可以先遍历一次字符串，这样就能统计出字符串中空格的总数，并可以由此计算出替换之后的字符串的总长度。每替换一个空格，长度增加 2，因此替换以后字符串的长度等于原来的长度加上 2 乘以空格数目。我们还是以前面的字符串"We are happy."为例，"We are happy."这个字符串的长度是 14（包括结尾符号'\0'），里面有两个空格，因此替换之后字符串的长度是 18。

我们从字符串的后面开始复制和替换。首先准备两个指针，P1 和 P2。P1 指向原始字符串的末尾，而 P2 指向替换之后的字符串的末尾（如图 2.4（a）所示）。接下来我们向前移动指针 P1，逐个把它指向的字符复制到 P2 指向的位置，直到碰到第一个空格为止。此时字符串包含如图 2.4（b）所示，灰色背景的区域是做了字符拷贝（移动）的区域。碰到第一个空格之后，把 P1 向前移动 1 格，在 P2 之前插入字符串"%20"。由于"%20"的长度为 3，同时也要把 P2 向前移动 3 格如图 2.4（c）所示。

我们接着向前复制，直到碰到第二个空格（如图 2.4（d）所示）。和上一次一样，我们再把 P1 向前移动 1 格，并把 P2 向前移动 3 格插入"%20"（如图 2.4（e）所示）。此时 P1 和 P2 指向同一位置，表明所有空格都已经替换完毕。

从上面的分析我们可以看出，所有的字符都只复制（移动）一次，因此这个算法的时间效率是 O(n)，比第一个思路要快。

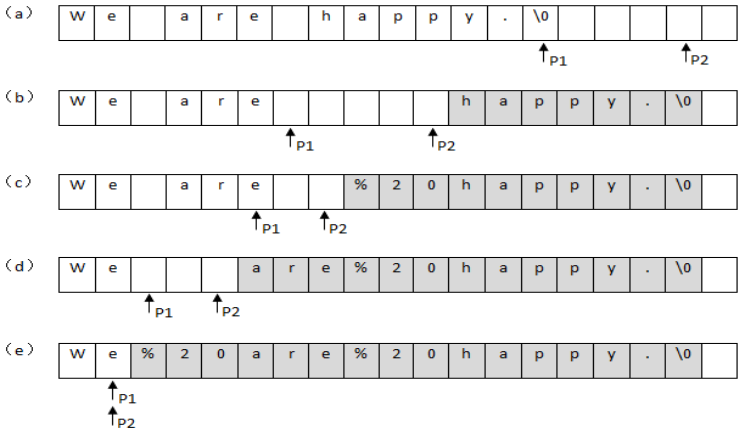


图 2.4　从后往前把字符串中的空格替换成“%20”的过程

注：图中带有阴影的区域表示被移动的字符。（a）把第一个指针指向字符串的末尾，把第二个指针指向替换之后的字符串的末尾。（b）依次复制字符串的内容，直至第一个指针碰到第一个空格。（c）把第一个空格替换成'%20'，把第一个指针向前移动 1 格，把第二个指针向前移动 3 格 （d）依次向前复制字符串中的字符，直至碰到空格。（e）替换字符串中的倒数第二个空格，把第一个指针向前移动 1 格，把第二个指针向前移动 3 格。

在面试的过程中，我们也可以和前面的分析一样画一两个示意图解释自己的思路，这样既能帮助我们理清思路，也能使我们和面试官的交流变得更加高效。在面试官肯定我们的思路之后，就可以开始写代码了。下面是参考代码：

```
/*length 为字符数组 string 的总容量*/
void ReplaceBlank(char string[], int length)
{
   if(string == NULL && length <= 0)

      return;
```

```
    /*originalLength 为字符串 string 的实际长度*/
    int originalLength = 0;
    int numberOfBlank = 0;
    int i = 0;
    while(string[i] != '\0')
    {
        ++ originalLength;

        if(string[i] == ' ')
            ++ numberOfBlank;

        ++ i;
    }

    /*newLength 为把空格替换成'%20'之后的长度*/
    int newLength = originalLength + numberOfBlank * 2;
    if(newLength > length)
        return;

    int indexOfOriginal = originalLength;
    int indexOfNew = newLength;
    while(indexOfOriginal >= 0 && indexOfNew > indexOfOriginal)
    {
        if(string[indexOfOriginal] == ' ')
        {
            string[indexOfNew --] = '0';
            string[indexOfNew --] = '2';
            string[indexOfNew --] = '%';
        }
        else
        {
            string[indexOfNew --] = string[indexOfOriginal];
        }

        -- indexOfOriginal;
    }
}
```

源代码：

本题完整的源代码详见 04_ReplaceBlank 项目。

测试用例：

- 输入的字符串中包含空格（空格位于字符串的最前面，空格位于字符串的最后面，空格位于字符串的中间，字符串中有连续多个空格）。

- 输入的字符串中没有空格。
- 特殊输入测试（字符串是个 NULL 指针、字符串是个空字符串、字符串只有一个空格字符、字符串中只有连续多个空格）。

本题考点：

- 考查对字符串的编程能力。
- 考查分析时间效率的能力。我们要能清晰地分析出两种不同方法的时间效率各是多少。
- 考查对内存覆盖是否有高度的警惕。在分析得知字符串会变长之后，我们能够意识到潜在的问题，并主动和面试官沟通以寻找问题的解决方案。
- 考查思维能力。在从前到后替换的思路被面试官否定之后，我们能迅速想到从后往前替换的方法，这是解决此题的关键。

相关题目：

有两个排序的数组 A1 和 A2，内存在 A1 的末尾有足够多的空余空间容纳 A2。请实现一个函数，把 A2 中的所有数字插入到 A1 中并且所有的数字是排序的。

和前面的例题一样，很多人首先想到的办法是在 A1 中从头到尾复制数字，但这样就会出现多次复制一个数字的情况。更好的办法是从尾到头比较 A1 和 A2 中的数字，并把较大的数字复制到 A1 的合适位置。

举一反三：

合并两个数组（包括字符串）时，如果从前往后复制每个数字（或字符）需要重复移动数字（或字符）多次，那么我们可以考虑从后往前复制，这样就能减少移动的次数，从而提高效率。

Broadview®
www.broadview.com.cn
博文视点·IT出版旗舰品牌
技术凝聚实力·专业创新出版

面试官的视角

从面试官视角剖析考题构思、现场心理、题解优劣与面试心得，尚属首例。

50 余道编程题

本书精选谷歌、微软等知名 IT 企业的 50 余道典型面试题，提供多角度的解题辅导。这些题目现今仍被大量面试官反复采用，实战参考价值颇高。

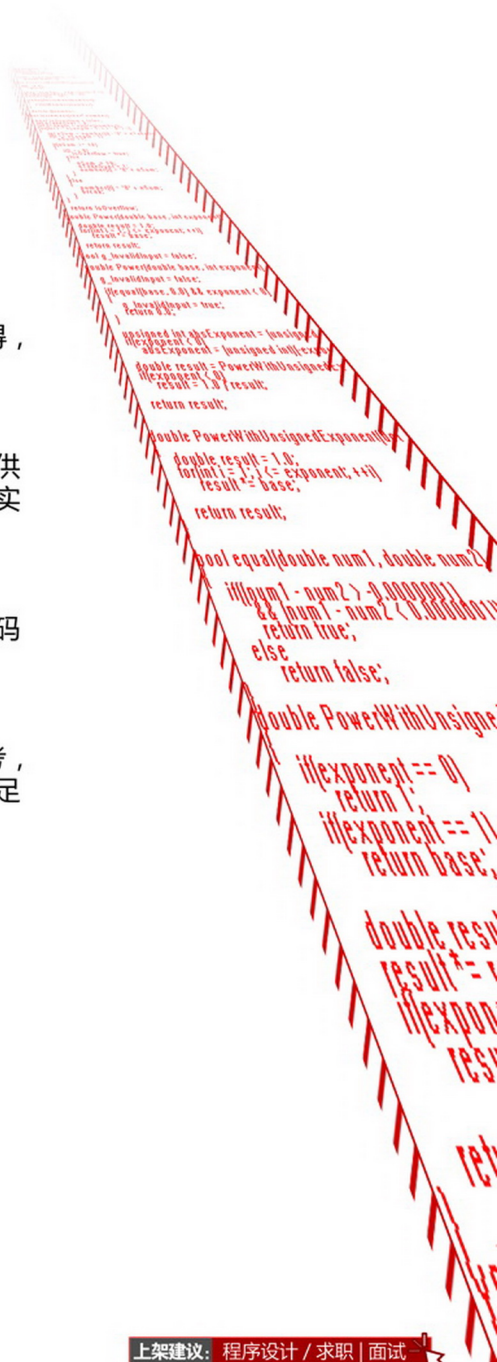
系统的解题方法

本书系统地总结了如何在面试时写出高质量代码，如何优化代码效率，以及分析、解决难题的常用方法。

超写实体验与感悟

Autodesk→微软→思科，作者一路跳槽一路“面”，既亲历被考，也做过考官，更是资深程序员，大量的一线面试与编程经验，足当确保本书品质。

本书涉及程序源代码请到
http://www.broadview.com.cn/14875 进行下载。



上架建议：程序设计 / 求职 | 面试

ISBN 978-7-121-14875-0
9 787121 148750 >
定价：45.00元

策划编辑：张春雨
责任编辑：李云静