Java 软件开发

Software Development In Java

(美) Sartaj Sahni　Raj Kumar 著
杜大鹏 岳丽君 李善茂 龚小平 等译　杜国梁 审校

中国水利水电出版社
www.waterpub.com.cn

Java 软件开发

[美] Sartaj Sahni　Raj Kumar　著

杜大鹏　岳丽君　李善茂　龚小平　等译

杜国梁　审校

中国水利水电出版社

内 容 提 要

虽然本书以 Java 为示例来讲述全书内容，但其内容却适合于使用各种编程语言的软件开发工作。本书是一本深入介绍软件开发的所有方面的书籍。作者讨论了诸如问题描述、模块化、编程美学、逐步完善、测试、验证和文档说明等软件工程的实施过程。除了这些专题之外，软件开发人员还需要理解性能分析和测试方法，并在数据结构和算法之间作出选择，本书也讨论了这些专题。作者使用 Java 来讲授软件开发并提供了许多示例。

本书适合作为有关软件开发、计算机科学导论和高级编程等课程的教科书。对于有经验的程序员来说，本书也是有价值的参考读物，而且是软件开发人员手头必备的书籍。

Sartaj Sahni, Raj Kumar: Software Development In Java

ISBN 0-929306-26-0

Translation rights arranged with the permission of Silicon Press

北京市版权局著作权合同登记号：01-2003-8495

图书在版编目（CIP）数据

Java 软件开发/（美）萨尼（Sahni, S.），（美）库马尔（Kumar, R.）著；杜大鹏等译. －北京：中国水利水电出版社，2004.6

书名原文：Software Development In Java

ISBN 7-5084-2173-6

Ⅰ. J… Ⅱ. ①萨…②库…③杜… Ⅲ. JAVA 语言－程序设计 Ⅳ. TP312

中国版本图书馆 CIP 数据核字（2004）第 054904 号

书　　名	Java 软件开发
作　　者	[美] Sartaj Sahni　Raj Kumar　著
译　　者	杜大鹏　岳丽君　李善茂　龚小平　等译
审　　校	杜国梁
出版　发行	中国水利水电出版社（北京市三里河路 6 号　100044） 网址：www.waterpub.com.cn E-mail：mchannel@263.net（万水） sales@waterpub.com.cn 电话：（010）63202266（总机）、68331835（营销中心）、82562819（万水）
经　　售	全国各地新华书店和相关出版物销售网点
排　　版	北京万水电子信息有限公司
印　　刷	北京北医印刷厂
规　　格	787mm×1092mm　16 开本　21.25 印张　483 千字
版　　次	2004 年 7 月第 1 版　2004 年 7 月第 1 次印刷
印　　数	0001—5000 册
定　　价	32.00 元

译者序

本书所讲述的有关软件开发的诸多问题，并不只是在 Java 语言中才存在，而是对所有编程语言的软件开发项目都同等适用。原书作者已经指出了这一点。原书作者说，本书不过是原来的《Pascal 软件开发》的 Java 版而已，这一点务必请有意选择本书的读者注意。

我们这个翻译小组虽然翻译过许多软件开发方面的书，但还是第一次接触软件开发理论，这使本书的翻译增加了难度，但同时也是对我们的挑战。善于啃硬骨头是我们这个翻译小组的长处。翻译本书既锻练了我们的“牙齿”也提高了我们的知识水平，真可谓一举两得。我们这个翻译小组由掌握不同专业知识的人员组成，此时这种组成就发挥了优越性。其中教数学课的老师成了我们的咨询人员，这使得我们对本书中大量数学内容的翻译能够做到尽量准确、符合专业要求。尽管如此，我们毕竟是第一次接触这样的内容，错误与不当之处在所难免，还望广大读者批评指正。

参加本书翻译工作的有杜大鹏、岳丽君、李善茂和龚小平等人。其中，杜大鹏翻译了前言和第 1 章～第 5 章，岳丽君翻译了第 6 章～第 8 章，李善茂翻译了第 9 章～第 11 章，龚小平翻译了第 12 章～第 14 章和附录。全书由杜国梁审校并统稿。参加本书录入、打印、校对等工作的有管英强、傅烨、魏天超、梁国珍、任建畅、马相生、刘发来、董明、迟春和杨天华等。在此对他们为本书所做出的贡献表示感谢。

译者

2004 年 3 月

前　言

本书实际上是《Pascal 软件开发》（Software Development in Pascal，作者 Sartaj Sahni，由地处佛罗里达州 Gainesville 的 Camelot 出版公司出版）一书的 Java 版本。我们的目的是提供介绍软件开发任务的各个方面的导言性的书籍。本书的内容或许是软件开发过程中遇到的困难的某种写照。本书中讲述的论题代表了我们要定义软件工程的意图。这些论题包括传统软件工程专题，如问题描述、操作界面设计、模块化、编程艺术、逐步完善、测试和编制文档等。除此之外，还包括了有关程序确认、性能分析和测试、数据结构和算法等方面。我们的看法是，后面这些论题对于软件开发来说与前面的论题是同等重要的。

这本教科书的内容可用于当前本科生所学的许多课程中。为了获得对软件开发的各个方面的较好的感受，完整地讲授本书是必要的。介绍性的软件工程课程可以以本书的第 1 章～第 9 章为基础。介绍性的数据结构和算法课程可以以本书的第 10 章～第 14 章为基础。编程的第二门课程（通常称为高级编程）可以使用本书第 1 章～第 7 章以及第 9 章～第 12 章的内容。如要全面讲述本书的内容，我们建议利用连续两个学期的时间学习。

Sartaj Sahni
Raj Kumar

致　　谢

我们要感谢 Narain Gehani 为了提高本书的质量所做的艰苦工作。如果没有他的创意和帮助，本书或许不会与读者见面。

目　　录

第 1 章　软件开发

1.1　开发活动

“软件开发”这个术语的意义包括从问题描述到获得解决问题的切实可行且文档完整的计算机程序的所有活动。给出这些活动的完整列表的一个困难是，与其说编程是一种科学，不如说是一种艺术。对于编程来说，有许多创造性的方面。更进一步说，编程科学还在不断地变化，还远没有达到成熟的程度。另一个困难是，开发好的可靠的软件是较为困难的，因为开发的某些方面会在软件系统的生命周期中始终存在。例如，要保证如操作系统、编译器以及数据库管理系统这样的大型复杂程序的完全正确无误是不可能的。结果，这些程序中的错误甚至是在其投入使用几年以后才发现。如此一来，程序维护要伴随程序寿命的全过程。另一个方面是程序的功能的提升。用户的要求随着时间在变化，因而随着需求的变化而修改或是改善程序就变得十分必要。

尽管有这些困难，我们还是列出以下在软件开发过程中通常都要完成的主要的一组活动：

- 软件描述
- 设计
- 模块化
- 选择一种或几种编程语言
- 程序开发
- 正确性验证
- 性能分析和测试
- 编写文档
- 维护
- 改进

我们通过设想一个文本处理器的开发来作为一个实例，说明这些活动。当讨论软件开发时，一定要记住，在开发过程中要涉及许多不同的个人。有以下几种人：

- 问题提出者：提出通过要开发的软件来解决某种问题的人（或几个人）。
- 设计人员：设计软件系统的人。
- 程序员：负责实现设计任务的人。
- 维护人员：软件发布投入使用之后，负责发现并改正软件中出现的错误的一组人员。我们假定，这一组人员还有在必要时提供软件增强功能的责任。
- 用户：在软件发布之后使用软件系统的群体。

在某些软件开发项目中，一个人可能充当着上述所有种类的角色。例如，提出问题，设计并实现其解决方案都由一人担当，而且还是该系统的惟一用户。进一步说，通过对该软件系统的使用，该人可能会放弃生成的程序。在这种情况下，也就没有维护与增强功能的需求。当一个软件项目涉及开发大型而复杂的软件系统时，上述每个种类中的人员都可能是不同的个体。

1.2 软件描述

在本步骤中，我们关心的主要是试图理解问题。没有对问题的充分和准确的理解，就不可能开发出令人满意的软件系统。一般来说，提出问题的人和开发最终软件的人是不同的人。这样一来，当问题第一次提出来时，那些可能成为该软件的程序员的人可能对程序所要完成的任务还没有充分理解。甚至程序员和问题的提出者是同一人时也可能出现这种情况。通常，提出问题的人可能没有通盘地考虑该问题，从而也不能确定对该程序的全部要求。

下面的说法在全世界各地的编程工作室内都可能遇到，而且为我们的命题真实性提供了证据。

当收回被评估的程序时，一个不太高兴的程序员可能会说：

“可是你没有说非得这样做才行，我认为这就是你想要做的。”

在被问到，要完成的文本编辑器程序是否应该包括一个拼写检查器时，开发指导人员可能会说：

“噢，你按照你认为的最好方式来自己决定吧。”

在文本编辑器程序的软件要求描述阶段，我们可能要回答下面一些问题：

（1）这个文本编辑器打算做什么？一般来说，此程序应该提供文本的编辑（如输入并修改）和格式化（如左右边界对齐、设置段落、选择字体、字号等）功能。目标用户群体的要求是什么？是否还要处理数学方程？如何跟踪文档的变化？

（2）编辑与格式化功能是应该分开还是集成在一起？如果像 Microsoft Word 那样，将这两种功能集成在一起，那么每当发生一种变化时，重新格式化全部文本可能是必要的。另一方面，如果两个功能是分开的，就像使用 XEmacs 软件编辑文本，然后使用 $\LaTeX$ 来格式化那样，用户在输入文本时就不知道格式化后的文本将是什么样的。

（3）以什么样的格式将生成的文件保存到磁盘上？

1.3 设计

理解要解决的问题之后，我们需要处理满足描述的较高级别的软件系统的设计问题。设计阶段主要关心的是系统的功能而不是实现的细节。我们要关心的是软件系统中那些影响其外部行为的方面，即用户界面。

在设计阶段到底发生什么通常依赖于问题的说明详细到什么程度。问题描述有可能是

如此准确，以至于设计阶段什么都不必做了。但是，在我们的文本处理器的例子中，却有许多必做的事情。假设，描述要求文本处理器带有集成的编辑器和文本格式化功能。在设计阶段，我们必须决定以下一些事情：

（1）要提供什么样的编辑功能？某些明显的功能是：在光标位置插入新文本、删除突出显示的文本、将突出显示的文本复制到缓冲区以及从缓冲区将文本粘贴在当前光标位置。

（2）是否要有联机帮助功能，从而用户可在任何时候请求如何使用文本处理器的信息？如果是这样，如何激活这种帮助功能？是否要在帮助中包括联机示例？

（3）是否要有拼写和语法检查器？如果是，用户能向拼写器字典中添加新词汇或（并）添加用于语法检查器的规则吗？

（4）要提供什么样的打印功能？是否应该支持将打印文件以多种格式（包括 PDF 和 postscript）加以保存？

人们很容易在设计阶段的问题描述中看到上面所列出的每个问题。描述和设计的真正差别在于，程序员必须设计软件系统使之符合要求的描述。因而，如果某些编辑功能是描述的一部分，那么这些功能就必须提供并按要求运行。未加明确指定的功能性可按设计人员认为最好的方式加以设计。在描述与设计之间的这种差别是重要的，因为常常会出现以下情况，程序员设计了一个并未满足描述的程序，但他们却认为所设计的程序比描述的要求还要好。这种情况是无法让人忍受的。软件描述规定了所要求的功能。如果程序在各个方面均未满足描述要求，那就是说并未解决所提出的问题。

如果设计人员感到在设计阶段所做的事情或许应该成为要求描述的一部分，或者特定的设计决定会实际上影响生成的程序，那么最好返回到问题的提出者处并与之讨论各种设计替代方案。这样就能修改并改善问题的描述。当把这一过程进行到底时，整个设计都会成为描述的一部分。

1.4　模块化

模块化是一种用于将大型且复杂的任务分解为较小且较简单任务的集合的技术。这种技术可应用于软件开发中的许多活动中，目的是将软件系统划分为许多功能独立的部分或模块，从而每个部分或模块可相对独立地进行开发。这将减少软件开发的总时间，因为解决几个较小的问题比解决大的问题要容易一些。另外，不同的模块可同时开发。如果必要，每个模块还可进一步地分解，从而形成模块的层次，如图 1.1 所示。

在顶层，文本处理程序分成编辑器、格式化程序、帮助等模块。编辑器将处理诸如复制、粘贴、剪切、用另外的词替换所有特定词的出现等编辑任务。格式化程序将处理诸如字体和字号选择、段落、列表、多栏布局等格式化任务。帮助模块将提供与使用和理解文本处理程序有关的所有任务。这些模块中的每一个模块的描述、设计和开发都可以以相对独立的方式进行，只要模块间的接口已经充分说明即可。在图 1.1 中，编辑器本身又分成以下子模块：复制功能模块、粘贴功能模块等。这些模块中的每一个都可独立地描述、设计和开发。

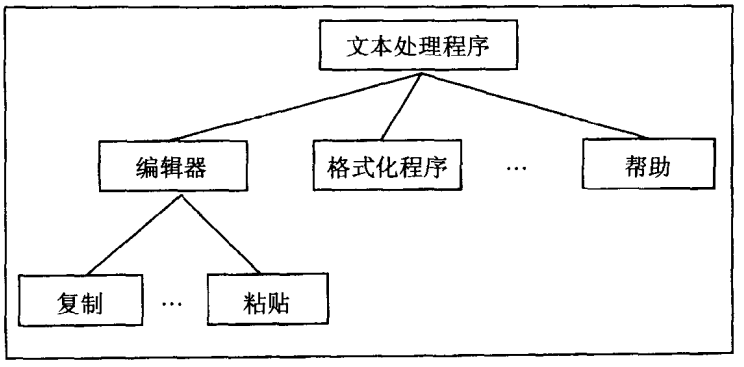

图 1.1 文本处理程序的模块层次

图 1.1 中的每个方框称为一个节点。节点 x 的子节点是与之相连的紧接在其下的节点。节点 x 是其子节点的父节点。没有子节点的节点称为“叶”(leaf)，而没有父节点的惟一节点称为“根”(root)。连接父节点与其子节点的直线称为“边”(edge)。整个层次结构称为“树”(tree)。在一棵树上，不是根节点的每个节点都有惟一的父节点。

在图 1.1 中，文本编辑程序节点是根。其子节点是编辑器、格式化程序和帮助（根还有图中没有显示的其他子节点）。帮助节点是此树上的一个叶节点。

软件系统一经描述并以模块方式加以设计时，自然开发此系统本身也是以这种方式进行的。生成的软件系统将有与模块层次结构中的节点一样多的模块。与文本处理程序或是根节点相对应的是建立文本处理程序的打开的窗口模块。当执行诸如“复制”一类的活动时就会激活复制模块。从这个示例中可以看到，模块化提高了一个问题的可智能管理性。通过将大型问题系统地划分为较小的相对独立的问题，以较小的努力解决大型问题就成为可能。独立的问题可指派给不同的人，以便平行地解决问题。在单一模块上共同承担任务较为困难。开发模块化软件的另一优越性是，在将较小的模块作为一个单元进行测试之前，独立地测试和检验较小的模块，也比测试一个大型模块来得容易。

1.5 选择编程语言

选择编写模块的一种（或几种）编程语言是由诸如以下多种因素支配的：

（1）所希望具有的可移植性程度。如果我们希望在各种机器上运行程序，必须选择所有这些机器上的共同语言。否则，程序必须经过多次重新编码。

（2）语言要适应于要编程的模块。对于那些显示菜单的模块，菜单设计语言就比使用诸如 BASIC 或 C++更好一些。对于那些执行大量计算的模块，使用带有较好的计算支持的 C++可能更好一些。当许多语言看起来对于模块是平等适用时，那就应该选择自己最为熟悉的语言。

（3）选择编程语言的另一个考虑是该语言可用的编译器或翻译器的质量。当使用不同的语言来编写一起工作的模块时，必须确保编译后的代码事实上接口良好。

（4）提出问题的人可能规定必须以某种语言来编写程序。这也正是我们将在本书中

所做的。在此郑重声明，本书中的所有程序都是以 Java 语言开发的。这是因为本书论述的是 Java 语言中的软件开发。

1.6　程序开发

除了最简单的情况之外，与模块对应的程序开发过程也要通过所有的开发阶段。我们以模块最高层的叙述开始，然后继续加以完善，使之到达最低一层，这就是 Java 代码。在这个完善过程中，我们将需要决定用于实现模块的数据结构以及算法。数据结构一般来说是以从上到下的方式来决定的，开始时在模块化层次的最高层，决定下一级模块可共享的数据结构。共享数据的所有模块必须以使用该数据的同一结构来工作。如果模块 A 和 B 使用某些不能共享的数据，那么这两个模块就没有必要了解其非共享数据是如何组织的。

1.7　检测

检测和确认是用来确立软件系统的正确性的两种活动。这两种活动可在软件开发的每个阶段来进行。检测时，每个阶段结果的正确性是建立在初始问题描述的基础之上的。而在确认时，正确性是建立在每个阶段的描述之上的。

由于在开发过程中发现错误越早改正软件系统中的错误的成本越低，检测和确认要在软件开发过程中的每一步中进行，而不是在完成之后才进行。这就意味着，问题描述要在进入软件设计阶段之前加以检测并确认其一致性、完整性和正确性。如果发现描述在某些方面的不一致、不完整或不正确，则在开始设计之前就要修改好。在制定程序计划之前，就应该检测并确认设计与给定要求描述相符合。程序计划本身也要在编写程序开始之前加以确认。每个模块和最终的软件系统都应该在发布投入使用之前加以确认。

尽管存在可帮助检测和确认的高级软件工具，在开发过程的一步中的不一致性也可能在下一步开始之后才发现。因而，在实践中，软件开发是一种几个步骤要加以重复以改正错误的反复过程。

在本节的剩余篇幅里，我们将只具体地处理程序的检测问题。这里提出的想法同样适用于开发过程的所有步骤中执行的检测和确认。检测实际上有两种方法：静态分析和测试。

在程序的静态分析中，代码并不执行。但是，要使用自动或手工方法加以分析。静态分析最常用的形式是：

- 使用编译器来检测句法错误
- 使用流程图
- 符号执行
- 部件检查、粗查和验收
- 模拟
- 正确性证明

大多数编译器，除了可检测程序句法中的错误之外，还能检测下面的错误：

- 未定义的变量。
- 已经定义但未使用的变量。这种变量表明可能存在拼写或是逻辑错误。
- 如下面的程序片断那样，在使用之前重新定义的变量。

```
x=2; y=3; z=5; x=y+z;
```

这样的变量也表明程序中可能存在错误。

- 未到达的代码段。
- 例程和函数的形式与实际参数之间的数据类型不匹配。

由于决定变量在程序执行中是否实际上使用或是程序段在程序执行中是否执行是不好判定的问题（也就是说，这些问题通过任何程序是不可解决的），编译器不能查出所有这类错误。但是，编译器可以查出大多数这类错误。

从传统上看，编译器只检查程序的句法是否允许一个变量被使用（例如，它出现在赋值语句的右边或出现在条件表达式中，或者作为方法激活时的参数等）。代码是否到达只由句法考虑来决定。例如，在下述代码片断中：

```
return(x);
x=3
```

第二条语句是不可到达的。

即使程序的语义学的句法允许使用一个变量，再考虑到代码的可到达性，这两种情况之一也可能出现。使用流程图（即规定程序中控制流程的框图）常常可查出这类问题。例如，考虑以下代码：

```
x=2;
y=3;
i f (x >y)
   z=4;
else
   x=2*y;
y=z;
```

这段代码的流程图如图 1.2 所示。通过检查本图中的路径 ABCE 和 ABDE，可以看出，只要程序采取路径 ABDE，变量 x 在使用前已经重新定义，而变量 z 在语句 E 中未加定义。

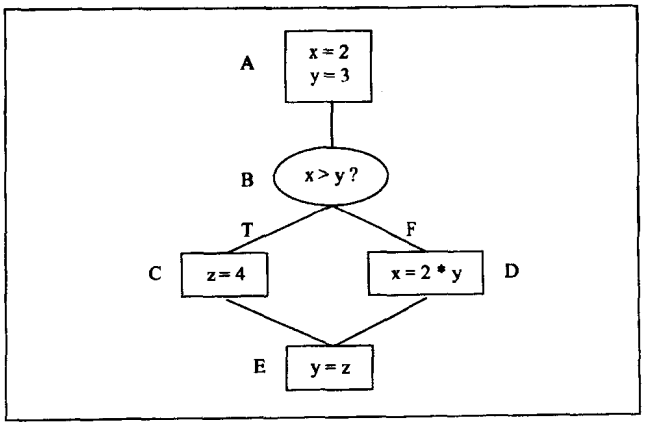

图 1.2 流程图

符号执行这段代码时[译者注1]，不如为输入变量赋值即可执行这段程序。即使图 1.7[译者注2]中的程序没有输入变量，经符号执行之后可以看出，语句 C 是不可到达的。x 和 y 在 A 中被分别赋值 2 和 3。由于 B 中的条件估值总为假，语句 C 永远也不会执行。因而，x 在使用之前总是被重新定义。而且，z 在语句 E 中总是没有定义的。

如果将 A 中的代码改变为：

```
x = 2;
y=getchar();
```

那么符号执行的结果是：

```
if 2 > y
then {x = 2; y = 4; z=4。}
else {x = 2y; y 和 z 未加定义。}
```

分析程序的传统方法是程序员坐在桌前手工检查程序。手工程序分析对于检查程序的正确性来说是一种有效的方法。由于大多数人很难找出自己的错误，最好让别人来执行这一手工检查任务。从头到尾地仔细检查是手工检测程序的两个主要原则。在这两种方法中，由软件开发人员和与软件开发无关的人组成的一组人来审核程序。这组人从软件描述到设计再到编码从头到尾地进行手工检查。

在检查过程中，描述、设计以及（或）代码都要一步一步地大声读出。可使用常见缺陷的列表来帮助查找错误和不一致性。接着进行的讨论通常会发现已有的错误和不一致性。在检查过程中，错误检查组人员在一组或多组测试数据之上手工模拟程序的执行。

在模拟过程中，建立程序的模型和程序在其中使用的环境。然后分析模型，以便确定程序是否可以达到描述的要求。

到目前为止讨论的静态分析方法都不能最终确认程序的正确性。为了解决这一问题，提出了几种严格的数学方法。我们将在后面几章中详细阐述这些方法。这些方法太麻烦，以至于不能用于任何大型程序。而且，验证过程的自动化的可能性是受计算理论的结果所支配的。这些结果表明，还没有一种算法（即一种在所有输入情况下保证终止的程序）可以确立所有程序的正确性。

静态分析再多也不能实际上确立所构建的软件系统将会在目标环境中正确地工作。这个命题即使是在有人形式上证明了程序的正确性时也是对的。这是因为，正确性的证明绝不能解决目标环境问题。通过检查和遍查的代码有时也会不能正确地工作，这是由于所用编译器或是目标计算机的操作系统的某些特性引起的，也可能是由于检查和遍查未能查出所有的错误引起的。

为了获得程序将会在目标硬件/软件环境中正确工作的信心，程序必须在该环境下使用实际数据来运行。测试是在目标环境下实际执行程序并将其性能与要求描述加以比较的过程。从测试数据中获得的结果将与从正确的程序中获得的所期望的结果加以比较。测试为我们树立程序将可正确工作的某些信心。信心的程度随着测试次数和变化的增多而增加（当

译者注1　此处原文似应为“In a symbolic execution of the code,...”故应译为“符号执行这段代码时”。

译者注2　查本章没有图 1.7。也许是图 1.2 之误。

然，这些测试要表明程序按预期来工作）。

一种提供正确性证明的不成熟的方法是对所有可能的输入来测试程序，并将结果与预期的或是“正确的”结果加以比较。如果在所有输入的情况下都是符合的，那么程序是正确的。这种耗尽式的测试程序的方法是不切实际的，其原因有以下几个：

- 要测试的输入数据的数目通常是无限的或是天文数字，因而测试不会停止。
- 正确结果常常只对某些输入是已知的。

这样一来，实际测试的次数再多也不能建立起正确性。因而，测试的实际目的是为了暴露错误（如果有的话）的存在。一组好的测试数据是那种使不正确的程序出现异常行为的数据。

有两种测试方法，即自下而上和自上而下进行测试。在自下而上的方法中，以独立地测试叶模块开始，然后沿着模块层次结构向上验证包括较高层模块的模块的正确性。在使用自下而上的方法时，必须编写正确设置测试环境（即被测试模块所需要的由其他模块创建的数据结构）的“驱动”程序。

如果对图 1.1 中的文本处理程序使用自下而上的方法，编辑器功能模块（复制、粘贴等）将会在编辑器模块测试之前加以测试；根模块（文本处理程序）要在编辑器、格式化程序和帮助模块都测试之后才会测试。

在自上而下的方法中，我们以测试根模块开始。为达到此目的，需要编写代码来模拟根模块激活较底层的模块。模拟每个模块结果的代码称为“占位程序”（stub）。当根模块测试过后，我们就将根模块与紧下一层模块一起测试。这种方法不需要驱动程序。在新加入测试的模块需要合适的测试环境之前，这种环境已经由较高层模块创建出来。

在对图 1.1 中的文本处理程序使用自上而下的方法时，根模块首先加以测试。为了执行这一测试，用于编辑器、格式化程序和帮助模块的占位程序是必需的。根模块测试之后，再引入编辑器、格式化程序和帮助，然后再次测试。这一次，用于复制和粘贴模块以及格式化程序和帮助的子模块的占位程序是必需的。

当测试自下而上进行时，叶模块测试的次数最多（因为在测试其所有上层模块时都要涉及）。当测试是自上而下进行时，较高层次的模块比较低层次的模块的测试次数要多。有时，使用占位程序不可能对模块进行有意义的测试。另外有时不可能建立模块的驱动程序（在没有耗费太多精力的情况下）。因而，常常使用自下而上和自上而下的混合方法来测试软件。

调试是与检验常常相关的另一种活动。这是一种找出并改正引起错误的原因的过程。调试常常使用计算机辅助跟踪程序来进行。在这种方法中，我们执行部分代码并检查某些变量的值。跟踪程序的一种较为困难的方法是手工引入一组称为“调试语句”的语句，这些语句可打印出想要监视的变量的值，运行修改的程序、研究输出、改正发现的错误、重新运行程序、找出新的错误，再引入更多的打印语句等。当程序能够令人满意地运行时，就可删除调试语句，然后重新运行程序，以保证删除的只是调试语句。

当使用诸如 jdb（这是 Java 开发工具箱（JDK）软件中的一个工具）一类的调试器时，调试任务可大大地简化。这种调试器允许人们一步一步地执行程序，打印任何感兴趣的变