HZ BOOKS

软件工程技术丛书

测试与度量系列

面向对象的软件测试

A Practical Guide to Testing Object-Oriented Software

John D. McGregor David A. Sykes 著

杨文宏 李新辉 杨 洁 等译

机械工业出版社 China Machine Press

中信出版社 CITIC PUBLISHING HOUSE

软件工程技术丛书

测试与度量系列

面向对象的软件测试

A Practical Guide to Testing Object-Oriented Software

John D. McGregor David A. Sykes 著

杨文宏 李新辉 杨 洁 等译

机械工业出版社 China Machine Press

中信出版社 CITIC PUBLISHING HOUSE

本书详细介绍了面向对象软件测试的各个方面，将软件测试融合到软件开发的各个阶段，对于确保软件产品的质量具有重大意义。该书以一个拱廊游戏为例，对软件测试的各个方面给出了具体的例子，因而对于读者在面向对象软件测试方面的实践也具有很强的指导意义。

本书内容丰富，结构合理，适于计算机及相关专业的本科生和研究生以及软件工程技术人员使用。

本书版权登记号：图字：01-2001-5469

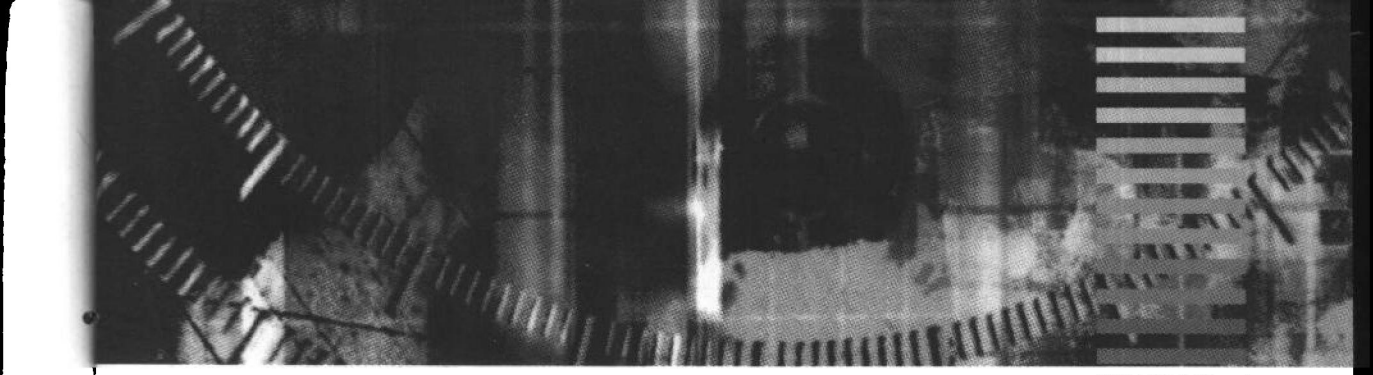
图书在版编目（CIP）数据

面向对象的软件测试/（美）麦格雷戈（McGregor, J. D.）等著；杨文宏等译. －北京：机械工业出版社，2002.8
（软件工程技术丛书）
书名原文：A Practical Guide to Testing Object Oriented Software
ISBN 7-111-10838-8

Ⅰ. 面… Ⅱ. ①麦… ②杨… Ⅲ. 软件－测试 Ⅳ. TP311.5

中国版本图书馆CIP数据核字（2002）第063188号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码100037）
责任编辑：华章
北京忠信诚印刷厂印刷·新华书店北京发行所发行
2002年8月第1版第1次印刷
787mm × 1092mm 1/16 · 18印张
印数：0 001-5 000册
定价：35.00元



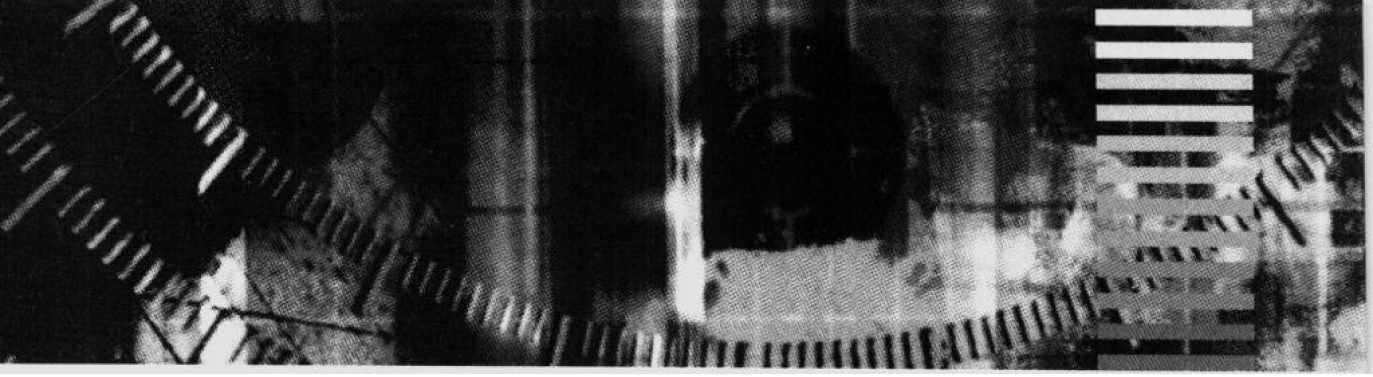
译者序

面向对象的软件测试是一项很具挑战性的工作。要做好软件测试，尤其是面向对象软件的测试，是一件很不容易的事。在面向对象的软件开发占主流的今天，一本对面向对象软件测试有实践指导意义的书是许多计算机软件从业人员盼望已久的事。因此我们很高兴能有机会向大家推荐这本面向对象软件测试方面的宝典。

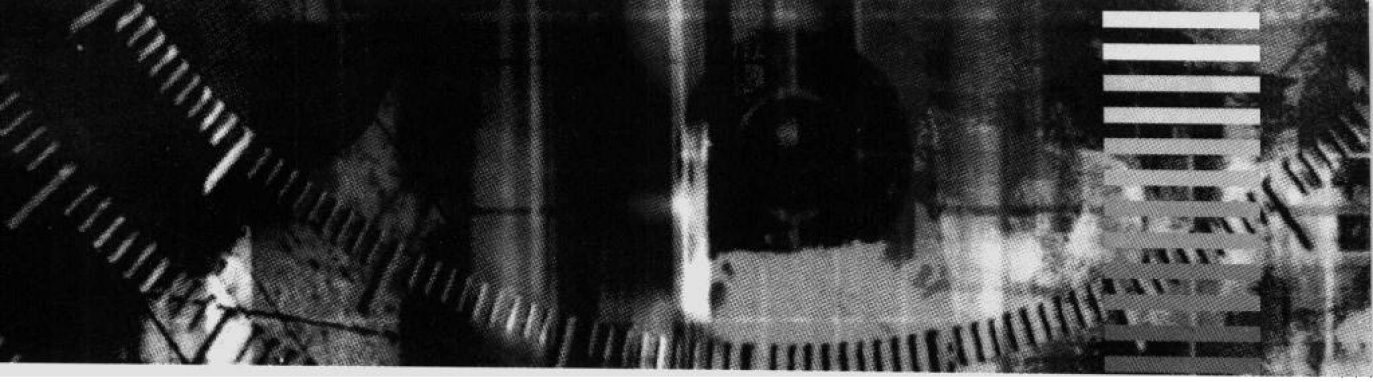
该书凝聚了作者多年从事面向对象软件开发、测试和教学的心血和经验。书中详细介绍了面向对象软件测试的各个方面，将软件测试融合到软件开发的各个阶段，对于确保软件产品的质量具有重大意义。该书以一个拱廊游戏（arcade game）为例,通过该例对于软件测试的各个方面给出了具体的例子，因而对于读者在面向对象软件测试方面的实践也具有很强的指导意义。

由于本书的作者对面向对象的软件测试已有相当的认识，对面向对象软件测试的阐述非常深刻，因而要想翻译好本书是一件非常不容易的事，加上时间匆忙，书中难免有不尽人意之处，还请读者多多见谅。但我们仍希望该书能给您的软件测试带来帮助。

参与该书的翻译人员有杨文宏、李新辉、杨洁 、戴莉萍、黄磊、付何伟、吴东方、张胜国、徐勇和何岸，张路同志参与了概述的校验。在此对以上同志一并表示感谢！



目 录



第1章

导　论

要做好软件测试不是一件容易的事，但测试的过程大家一般都比较清楚。通过单元测试、综合测试、系统测试、回归测试和接受测试后的系统一般就能够用了。

我们写这本书是因为大部分人都认为面向对象软件的测试与过程软件的测试没有什么不同。尽管许多常用的面向对象软件的测试方法和技巧与过程软件相同，或者可以从传统的测试方法和技巧中演化而来，但实践和研究表明它们之间还是存在许多不同的，面向对象的软件测试要面对某些更新的挑战。与此同时，作为增量开发过程一部分，设计良好的面向对象软件为改善传统测试过程提供了机遇。

在面向对象的编程语言中，继承和多态的特征对测试者来说是一个新的技术难点。书中描述了对大部分难点的解决方案。另外，还描述了如何在整个开发阶段对面向对象软件进行有效测试的过程和技巧。测试软件的方法非常复杂，因此，应由软件开发机构来承担测试任务。同时，我们意识到可用于测试的资源是有限的，而且有许多有效的开发软件的方法，因此，最好对本书所讲的技巧进行合理选择。

采用面向对象技术不仅给使用的编程语言带来了变化而且对软件开发的很多方面也带来了变化。对于面向对象的软件测试，我们使用了增量开发过程，重新调整并使用新的符号来分析和设计，并充分利用编程语言的新特性，这些变化提高了软件的可维护性、复用性和灵活性等等。我们之所以写这本书是因为软件开发方面的变化造成了软件测试方面的变化。以下变化为改进测试过程提供了机会：

- 我们有机会改变人们对测试工作的态度。在许多场合，管理者和开发者都将测试工作看做是一件既麻烦但又不得不做的事。那些需要开发者自己进行的测试工作会中断开发者编写代码的进程。代码检查和写单元测试驱动程序都要花费时间和金钱。强加在开发者身上的测试过程往往只是对代码进行测试。但是，如果能够让人们从一开始就认为测试工作能确保开发出正确的软件，并且确实能用来测量软件进度，使开发过程沿着正确的轨道进行，那么我们就能够开发出更好的软件了。
- 我们有机会改变人们对应该在开发过程中的哪个阶段进行测试的看法。几乎所有

的人都认为问题发现得越早，用于解决问题的费用就越少。单元测试和综合测试可以发现问题，但并不是只有开始编码了才能开始测试。系统测试一般都是在整个开发过程接近尾声的时候或是在某个安排好的重要阶段进行。有人认为系统测试主要用来测试开发者在满足需求方面做得怎么样。当然，这种看法是错误的。要确定进行多少次测试才够，什么时候进行，该有谁来做，这些都应该遵照一个慎重考虑过的，与项目的软件开发过程相配套的测试计划。我们将向大家介绍怎样尽早开始测试。如何边开发边测试，并使它们彼此相互促进以获得成功的结果。

- 我们有机会使用新技术来进行测试。正像面向对象技术给软件开发带来了好处一样，它也给软件测试带来了好处。我们将介绍如何对面向对象的分析和设计模型进行测试，以及如何使用面向对象的编程技巧来开发单元测试驱动程序从而减少测试软件组件所需的代码。

读者对象

我们写这本书是为

- **程序员**：他们有测试软件的经验，但想对面向对象软件的测试有更多的了解。
- **管理员**：他们负责软件开发并想知道该如何及在什么地方安排测试计划。
- **开发者**：他们负责测试自己编写的软件，并将测试融合到分析、设计和编码中去。

面对广大的读者群，我们力争将面向对象的软件开发和测试——包括与软件测试相关的基本概念、面向对象的编程和用于表达分析和设计结果的统一建模语言（UML）进行详细介绍，而对于那些我们认为读者为理解本书内容只需简单了解的部分只作简单概括。当需要使用代码举例时，我们选用C++和Java。但描述的方法和技巧适用于所有的面向对象的程序，而不仅仅是那些用C++和Java编写的程序。

下面我们假定一个理想的软件开发环境：

- 过程必须是可增加的，且每增加一次都进行一次反复。
- 用UML表达的模型必须是有效的。
- 软件设计必须与好的设计准则相一致，并考虑使用继承、数据隐藏、抽象、低耦合和高聚合等面向对象的特性。

但是，我们意识到大部分的组织都有自己的工作步骤和表现方式，因此，我们的着重点在基本准则和技巧上。

什么是软件测试

通俗地讲，软件测试（在本书中用“测试”表示）是发现并指出软件系统缺陷的过程。

缺陷在开发和维护的任何阶段都有可能发生，并由此产生一个或多个“漏洞”——错误、误解和冗余，有时甚至会误导开发者。测试包括寻找缺陷，但不包括跟踪漏洞及其修复。换句话说，测试不包括调试和修复[①]。

测试是很重要的，因为它能充分保证应用软件做它想要做的事。有些测试的重点延伸到确保一个应用程序仅仅做它该做的事[②]。软件的缺陷会造成时间、财产、客户甚至生命的流失，而测试在任何场合都能对防止软件出现错误做出重要贡献。

什么是软件呢？我们将软件定义为：为在计算机上完成某些任务而必需的指令代码和数据，当然也包括所有那些与指令和数据相关的表示法，特别是那些不仅仅包含程序源代码和数据文件还包含在分析和设计阶段创造的模型的表示法。软件能够而且应该在它的所有的表示法中进行测试。就像建筑师和施工人员在破土动工之前，对一个新建筑的设计蓝图进行检查以发现是否存在问题一样。因此，我们应该在编写第一行程序代码之前对软件的分析和设计模型进行测试。我们将介绍如何用“执行”的形式对这些模型进行测试。

测试能帮助确保一个产品满足需求，但测试并不是质量保证。有些人错误地将测试和质量保证等同起来。在许多组织中，QA通常负责开发测试计划和执行系统测试。QA可能会对开发过程中的测试进行监测和保留统计数据。测试是任何质量保证过程中必需的但不是所有的部分。QA从事的是那些用来防止和去除在产品中确实存在的缺陷的活动。项目的质量保证部门负责制订为了生产出更好的软件而所有项目成员都应该遵守的标准。这包括定义为理解设计意图而创建的各种文档的类型，指导项目活动的过程以及量化决议结果的方法。

大量的测试将提高计算机程序的质量。测试可以帮助识别错误，从而开发者们能发现问题并纠正它们。我们对一个系统做的测试越多，就越能确保它的正确性。然而测试通常不能保证某个系统的运转100%正确。因此，测试在确保质量方面的主要贡献在于识别那些我们在一开始就应该能够避免的错误。QA的使命首先是避免错误，要做到这一点除了测试外还需要其他的处理。

测试能在开发早期识别问题，从而帮助提高质量。幸运的是，我们能够在开发过程中尽早地做测试——甚至在编写代码之前。在本书中，我们描述了非常有用的技巧，但这些技巧需要测试者紧密配合开发者的工作，同时也要求开发者紧密配合测试者的工作。

测试面向对象的软件有何不同

在编程语言中，面向对象编程的特性很显然对测试的某些方面有影响。比如，类的继

① 我们注意到有些人既是软件的测试者同时又是那个软件的调试者，特别是在单元测试和综合测试时，这种现象更加普遍。但是我们对这两种行为进行了区分。测试是发现漏洞的过程。调试是跟踪漏洞产生的根源并进行修复的过程。由于测试有时可用来帮助查找漏洞的位置，因此，两者之间可能会有重叠的地方，然而测试和调试是两种不同的行为。

② 这对于延长系统的使用寿命或增加其价值的确相当重要。然而，如果没有阅读代码却要测试那些额外的软件功能就很困难，没有哪个测试者会为此花费太多精力。没有阅读代码，测试者就必须预见到开发者有可能犯的错误以及由此产生的更多的问题，所以他们往往设计实验来对这些错误进行检测。

承和接口等特性支持多态，使得代码可以控制对象而不管对象到底是什么。测试者必须确保代码能正常工作而不管与那个对象确切相关的类是什么。支持和加强数据隐藏的特性可能使测试复杂化，因为有时为了支持测试必须向一个类的接口添加操作。与此同时，这些特性都能更好地、重复地使用测试软件。

编程语言的改变对测试有影响，开发过程的变化以及分析和设计重点的改变也会对测试产生影响。许多面向对象的软件的测试活动都可以在传统的过程中找到对应的活动。我们仍旧使用单元测试，尽管在这里“单元”的意义已发生了改变；我们仍将做综合测试以确保各个子系统能够一致正常地工作；我们仍将做回归测试以确保对软件最后一轮的修改不会对软件以前的功能造成负面影响。

开发和测试新软件的“老”方法和“新”方法之间的不同，远比将重点放在对象上而不是放在将输入转化为输出的函数上的不同要大。最重要的不同在于，面向对象的软件被设计成一系列的对象，这些对象从根本上形成了问题的模型，并由这些对象共同作用于一个解决方案。这种解决办法基于这样一种概念：问题的解决方案可能需要经常改变，而问题本身的结构和组件却不需要经常改变。因此，如果程序是以问题为出发点构建的（而不是直接构建在需求方案上），那么它将更能适应以后的变化。一个对问题及其组件非常熟悉的程序员能够在软件中体现出它们，这样程序就更容易维护。而且，由于组件来源于问题，它们可在解决其他相似或相关问题时重复使用，从而提高了软件组件的复用性。

这种设计方法有一个很大的好处，即分析模型可以映射成设计模型，而设计模型又可以映射成代码。因此，我们可以在分析阶段开始测试，将分析阶段的测试提炼以后又可以用于设计阶段的测试，设计阶段的测试经过提炼以后又可用于对实现阶段的测试。这意味着测试过程可以与开发过程交替进行。我们发现对分析和设计模型进行测试有3个主要优点：

1. 测试用例可在过程中更早地确定，甚至在需求确定之前。尽早测试可以帮助分析者和设计者更好地理解和表达需求，并确保特定的需求是“可测试的”。
2. 漏洞可在开发过程的早期检测出来，从而节省了时间、金钱和精力。大家都知道，问题检测出来得越早，解决问题就越简单，开销也越少。
3. 可以在项目的早期检查测试用例的正确性。测试用例的正确性——特别是系统测试用例的正确性——总是很关键的。如果能在项目中尽早确定测试用例并运用到模型上，那么测试者对需求的任何误解都能被尽早纠正。换句话说，这非常有助于确保测试者和开发者对系统需求的理解保持一致。

尽管对模型进行测试有很多益处，但需要记住的是，不能把此作为测试中惟一的侧重点，代码测试仍旧同样是测试过程中十分重要的部分。

传统的项目与采用面向对象的技术开发的项目之间的另一个不同在于软件的目标不同。例如，许多公司都将能生产出可重复使用的软件、提供可扩充的设计方案或者甚至

是设计出能够描绘可复用设计的面向对象的框架作为重要的新的目标。在实现这些目标时，我们可以（且应该）通过测试来发现故障，而传统的测试方法和技术却无法做到这一点。

测试方法概述

尽管存在时间和费用的限制，我们还是应尽可能彻底地对软件进行测试。我们测试面向对象的软件的方法是以学术研究和我们在与各行各业（例如电信和金融业）的客户们一起工作时得来的经验为基础的。

根据我们的观点，测试不是事后的事，而是从开发过程中分离出来但与其紧密相关的单独的过程。我们有一句格言送给大家：尽早测试、经常测试、充分测试。我们喜欢采用下面的反复开发过程：

- 分析一些
- 设计一些
- 编码一些
- 测试你能测试的部分

上面讲的测试是指对目前所能测试的部分进行测试，这些部分包括在技术上能测试的部分和在有限的时间和资源下能测试的部分。在进行以上反复的过程中我们能做许多有意外收获的测试。有规律地进行测试能尽早发现故障并避免在以后的反复中重复工作。最后一次反复之后我们要做的是系统测试和接受测试。但是，如果能够增量地开发一个软件系统，那么你就可以在每一次增量之后进行系统测试。

我们应该对面向对象的软件进行什么样的测试呢？答案如下：

- 模型测试
- 类测试，它用来代替单元测试
- 交互测试，它用来代替综合测试
- 系统（子系统）测试
- 接受测试
- 发布/自我测试

本书覆盖了以上所有的测试。我们的测试过程将为每一个开发活动定义一个测试活动。

我们认为你不会——甚至不应该——应用我们在本书中描述的所有的东西。对一个开发过程来说，很少能有足够的资源来实施我们所描述的所有级别的和各种各样的测试。我们希望你可以找到一种可实施的、有用的且对你的项目来说负担得起的方法和技巧。

下面我们将就我们的格言“尽早测试、经常测试、充分测试”进行解释。

尽早测试

我们应该让测试者在项目的分析和设计阶段中某个适当的时候开始测试，而不是在项目快要结束的时候开始测试。对分析和设计模型进行测试不但有利于在开发过程的早期揭露问题（这时暴露的问题更容易修复且更便宜），而且有利于通过确定我们需要对什么进行测试来确定我们需要在哪些方面进行系统测试。

尽早测试和经常测试意味着对软件的描述是抽象的或者是不完全的。

经常测试

我们非常肯定地认为反复的、递增的——有时指**反复递增的**——开发过程最适合于开发非常大型的项目。当反复在分析、设计和实现阶段完成以后，产品就应该进行测试。第1次递增完成以后，有些测试应该采用回归测试的形式。

充分测试

完全测试软件系统的所有方面是不现实的，资源应该用在能提供最大回报的地方。我们喜欢以风险分析、测试用例的重复使用以及对测试用例的输入进行统计取样为基础的技术。

测试视角

优秀的测试人员——负责测试软件的人——需要一套专门的技巧。在许多方面，做一个优秀的测试人员比做一个开发人员要难，因为测试不仅需要对开发过程及其产品有很好的了解，还要求有预见可能出现的故障和错误的能力。举一个简单的例子，假设开发者开发了一个程序，使一个图像在计算机屏幕的直角之间来回跳跃。对于这样一个例子，测试者必须预见到开发者可能造成的故障和错误并开发出有效的方法来检测这些可能的故障。对于本例，测试者可能会需要测试一下当图像正好击中直角的顶点时是否就完全不动了。由此看来测试是一项艰苦的工作。

测试者必须以一种对软件的方方面面都提出疑问的态度来思考软件。我们称这种方法为测试视角，它是第2章的主题。本书中描绘的技巧和过程就是从测试的角度进行了开发和表示。

本书的组织方式

这本书共有11章。前3章主要关于测试概念和测试过程，第4章到第10章详细描述了用于各种各样的测试的技巧，第11章是总结。每章后面都有一个小结和一套练习。我们鼓励大家把所有练习都看一遍，并把自己感兴趣的或是和自己的工作有关的做一遍。大部分的练习都没有正确答案，但是其中一些有相对较好的答案。我们希望这些练习能帮助你将我们的技巧应用到你自己的项目中去。

第1章（本章）对本书内容进行了简单介绍。在这章中我们阐述了测试的总体概念、以纲要的形式介绍了测试面向对象的软件与测试其他类型的软件如何不同以及我们步骤的简明概要。

第2章介绍了测试视角。我们采用这种视角来阐述测试过程的各个方面、检查开发过程中的各种产品以及研究面向对象的基本概念。

第3章描述了测试过程以及它如何与开发过程相互关联。我们将测试过程看作是从开发过程中分离出来的，因为在某种程度上来说它们具有相同的目的。但是这两种过程是交织在一起的。

第4章描述了如何对模型进行测试。成功进行开发的一个关键是尽早测试、经常测试。我们喜欢对分析和设计模型进行测试，因为它们代表了软件本身。尽管这些测试增加了整个项目的开支，但在测试模型中做的工作可重复使用、提炼并可扩展到当代码被开发出来后对其进行测试。

第5章讨论了对具有相当简单的接口和实现的类进行测试。它的着重点在于类测试时的基本要素。在面向对象的环境下进行的类测试大致相当于传统环境下的单元测试。本章的重点在于明确类的哪些方面必须测试以及如何测试。我们描述了一些用于实现测试驱动程序的方法。

第6章对第5章提到的技巧进行了扩充，以用于那些规范和实现需要与其他类或对象进行交互的类（这与传统环境下的综合测试相对应）。对这些类进行测试有很大难度，特别是在有动态绑定的情况下。我们提出了可用于管理众多可以开发的测试用例的技巧。

第7章描述了对继承体系结构的类进行测试。我们的技巧的着重点是尽可能地重复使用测试驱动程序以及重复使用测试用例。我们提供了用于确定测试子类所需的最小测试次数的运算法则，这个法则我们已经测试过了。我们还描述了测试抽象类的技巧。

第8章讨论了测试的并行性。在那些通过线程和/或分布式来实现的系统中，已越来越多地使用了并行性。

第9章讨论了系统测试。测试一个采用面向对象的编程语言开发的系统与测试一个采用任何其他模式开发的系统在很大程度上是相同的，因为系统测试通常是基于系统的规范而不是系统的实现。但我们在测试是否足够这个方面有一些建议和方法。

第10章讨论了与组件、框架和产品线的测试相关的主题。

第11章简要复习了前面章节的要点并探讨了“我们将向哪方面发展”的问题。

本书中的一些约定

本书中的大部分代码用C++表示，但我们描述的这些技巧并不局限于C++。之所以选择C++是因为它使用非常广泛，并且在某些方面对测试者来说是最大的挑战。我们将逐渐介绍与Java有关的测试技巧。

偶尔你会发现单独出现在一个框中的一段文字。这些框是为了增加讨论的细节或辅助信息。这些文字块的类型有所不同，具体描述如下：

提示

提示是一段建议，它将使我们的测试变得更容易或更有效。有时一段提示明确指出我们测试的是C++或Java代码。有时一段提示是我们认为有用的通用技巧。

是不是有些问题会不断地重复出现呢?
回答是肯定的。我们将在带有该标志的框里回答经常问到的问题。

相关主题

有时我们会在这样一个工具条中讨论与一个概念相关的许多细节。有时我们将使用这样的工具条来讨论一些偏离主题的东西，但它是与讨论相关的而且是我们认为你应该知道的。

一个贯穿全文的例子——Brickles游戏

我们将使用一个贯穿全文的例子来解释各种测试方法和技巧。这样我们就可以将精力放在测试技巧上而不用花费时间去假设各种例子。在这一节中，我们将介绍Brickles游戏，它是一个交互式的计算机游戏。这个游戏与Breakout游戏很类似，Breakout是第一个商业视频游戏且在AppleII计算机上很流行。

这个例子最初是当我们在教授面向对象的设计概念的课程时开发出来的。我们的目的是为了找到一个应用程序，而且这个应用程序的设计要足够简单，以致于能在一个星期的训练中掌握和完成。不管是在教授的课程中还是在本书里，我们选用这个例子的目的都是为了说明设计和测试面向对象的程序的各种概念。

Brickles的基本组成

Brickles是一个拱廊游戏，它的初始配置如图1.1所示。游戏的区域是矩形的，由两堵墙、一个天花板和一块地板组成。在游戏区域里有由一组砖头组成的“砖堆”。游戏者的任务是通过控制一块水平移动的滑动板，让一个冰球通过反射击打砖堆上的砖头。当冰球碰到了一块砖头，砖头就破碎了。冰球在墙壁、天花板、砖头（当它们破碎时）和水平滑板之间反射跳跃。游戏开始时，游戏者可以有3个冰球用来击毁砖头。如果所有的砖头都被打碎了，游戏者就赢了。如果所有3个冰球都用完了，砖头还没有全被打碎，那么游戏者就输了。

Brickles的物理原理

当冰球在游戏区域移动时，它会与游戏区的各个组件相遇。它们之间的相互作用如下（同时参考图1.2、图1.3和图1.4）：

天花板和墙壁 冰球和天花板及墙壁之间的反射与物理上的反射原理一致，但忽略了摩擦力和重力——也就是说，反射角等于入射角。

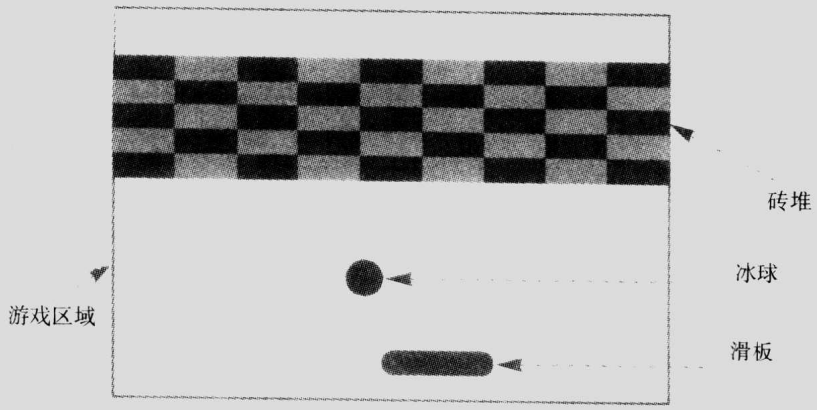


图1.1 Brickles初始状态

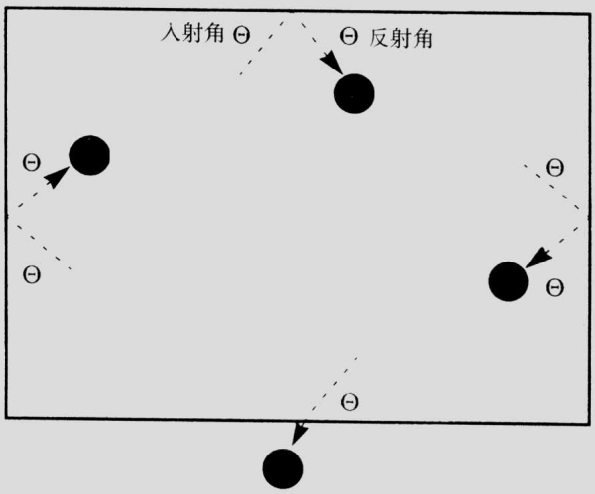


图1.2 冰球与边界的相互作用

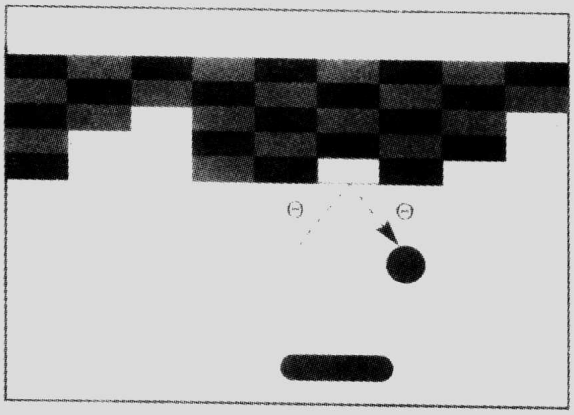


图1.3 冰球与砖块的相互作用