第5章 单元测试

单元测试又称模块测试，是对软件设计的最小单元的功能、性能、接口和设计约束等的正确性进行检验，检查程序在语法、格式和逻辑上的错误，并验证程序是否符合规范，以发现单元内部可能存在的各种缺陷。

单元测试的对象是软件设计的最小单位——模块、函数或者类。在传统的结构化程序设计语言（如C语言）中，单元测试的对象一般是函数或者过程。在面向对象设计语言（如Java、C#）中，单元测试的对象可以是类，也可以是类的成员函数/方法。由此可见，单元测试与程序设计和编码密切关联，测试者需要根据详细设计说明书和源程序清单来了解模块的I/O条件和逻辑结构。


视频讲解

5.1 单元测试概述

单元测试是在软件测试过程中的最早期进行的测试活动。一般而言，可以把单元测试看成软件开发的一部分，也就是说开发人员每编写一段代码，便可将单元测试用于检测这段代码的某一个功能是否正确。开发人员负责编写功能代码，同时也有责任保证代码的正确性，所以单元测试是开发人员在完成一个功能模块之后，为了检测它的正确性而自行进行的测试过程。

5.1.1 单元测试的概念

单元测试是针对程序单元模块（软件设计的最小单位）来进行正确性检验的测试工作。程序单元是应用功能的最小可测试部件。在面向对象编程中，最小单元就是方法，包括基类、抽象类或者派生类（子类）中的方法。按照通俗的理解，一个单元测试判断某个特定场景条件下某个特定方法的行为，如斐波那契序列算法、冒泡排序算法。

单元测试可将被测试应用程序细分为一个个足够小的基本单元，各个单元间相互独立，互不影响。开发者能通过单元测试证明被测试单元的行为确实和开发者期望的一致。

单元测试最基本的一个功能是能够快速定位代码中的错误。从项目一开始，开发者就应该对所有的单元模块进行测试，一方面能够尽早发现问题，另一方面为项目的持续开发提供保障。

搭建起一个良好的单元测试环境后开发者就可以对所有的基本单元进行测试，这样单元测试既可以为项目提供一份API文档，也可以随时查阅方法相关参数、返回值以及运行情况。

单元测试的过程也是开发者重构和进一步优化代码的过程，并且其能够确保单元工作

正常。这个过程就是为所有函数和方法编写单元测试。在连续的单元测试环境中，只要设计出了良好的验证手段，单元测试就可以延续用于准确反映当任何变更发生时可执行程序和代码的表现，帮助开发者优化代码逻辑和代码结构。

进行单元测试时，开发者可以站在一个观察调试的角度。无论是开发先于测试，还是测试先于开发，单元测试都可以帮助项目将模块设计成易测试、易调试、易重构的状态。在这个过程中，开发者的编码能力和对业务的理解能力也将得到锻炼。

单元测试应该在项目一开始的时候进行。不可否认，项目开始就编写单元测试常常要多花费几倍的代码量，但是随着项目进行，当把基础方法都测试过以后，高层功能需要的代码量反而会大大减少。这时候单元测试也在往集成测试迁移，这是一个自然而然的过程，同时其也能为集成测试的简化提供极大的便利。

5.1.2 单元测试的内容

单元测试的主要内容如下。

1. 接口测试

对通过被测模块接口传输的数据进行测试，以检查数据能否正确输入和输出。其主要是对模块接口的以下方面进行测试。

(1) 输入的实参与形参在个数、属性、量纲和顺序上是否匹配。

(2) 被测模块调用其他模块时，传递的实参在个数、属性、量纲和顺序上与被调用模块的形参是否匹配。

(3) 调用标准函数时，传递的实参在个数、属性、量纲和顺序上是否正确。

(4) 是否存在与当前入口点无关的参数引用。

(5) 是否修改了只作输入用的只读形参。

(6) 全局变量在各个模块中的定义是否一致。

(7) 是否将某些约束条件作为形参来传递。

2. 局部数据结构测试

局部数据结构是最常见的缺陷来源，检查局部数据结构可以保证临时存储于模块内的数据在代码执行过程中是完整和正确的。检查局部数据结构应考虑如下方面。

(1) 是否存在不正确、不一致的数据类型说明。

(2) 是否存在未初始化或未赋值的变量。

(3) 变量是否存在初始化或默认值错误。

(4) 是否存在变量名拼写或书写错误。

(5) 是否存在不一致的数据类型。

(6) 是否出现上溢、下溢或地址异常。

除了检查局部数据，还应注意全局数据对模块的影响。

3. 重要执行路径测试

应对模块中的重要执行路径进行测试。对重要执行路径和循环的测试是最常用和最有效的测试技术，可用以发现因错误而导致的错误计算，错误的比较和不适当的控制流而导致的缺陷。

常见的错误计算如下。

(1) 操作符的优先次序被错误理解。

(2) 存在混合模式的计算。

(3) 存在被零除的风险。

(4) 运算精度不够。

(5) 变量的初值不正确。

(6) 表达式的符号不正确。

常见的比较和控制流错误如下。

(1) 存在不同数据类型的变量之间的比较。

(2) 存在错误的逻辑运算符或优先次序。

(3) 存在因计算机表达的局限性,导致浮点运算精度不够,致使期望值与实际值不相等的两值比较。

(4) 关系表达式中存在错误的变量和比较符。

(5) 存在不可能的循环终止条件,导致死循环。

(6) 存在迭代发散,导致不能退出。

(7) 错误修改了循环变量,导致循环次数多一次或少一次。

4. 错误处理测试

完善的设计应能预见各种出错条件,并设置适当的出错处理,以提高系统容错能力,保证逻辑正确性。错误处理测试主要测试程序处理错误的能力,检查是否存在以下问题。

(1) 输出的出错信息是否难以理解。

(2) 出错描述提供的信息是否不足,从而导致无法对发生的错误进行定位和确定出错原因。

(3) 显示错误是否与实际遇到的缺陷不符合。

(4) 对错误条件的处理是否正确,即是否存在不当的异常处理。

(5) 在程序自定义的出错处理运行之前,缺陷条件是否已经引起系统干预,即无法按照预先自定义的出错处理方式来处理。

5. 边界条件测试

程序最容易在边界上出错,应该注意对它们进行测试。

(1) 输入/输出数据的等价类边界。

(2) 选择条件和循环条件的边界。

(3) 复杂数据结构(如表)的边界。

视频讲解

5.2 单元测试的过程

单元测试环境应包括测试的运行环境和经过认可的测试工具环境。测试的运行环境一般应符合软件测试合同(或项目计划)的要求,通常是开发环境或仿真环境。

单元测试的实施步骤包括以下 4 步。

(1) 测试策划,在详细设计阶段完成单元测试计划。

(2) 测试设计,建立单元测试环境,完成测试设计和开发。

(3) 测试执行,执行单元测试用例,并详细记录测试结果。

(4) 测试总结,判定测试用例是否通过并提交测试文档。

1. 进入单元测试的条件

进入单元测试必须满足一定的条件,这些条件是测试实施的基础,其通常包含以下4方面。

(1) 满足规定的文档要求。

(2) 软件单元源程序已无错误地通过编译或汇编。

(3) 被测试软件单元已被纳入到配置管理中,并已确定所提交的版本为本阶段最终版本。

(4) 已具备了满足要求的测试环境和测试工具。

2. 测试策划

当确认上述4个条件都满足后,测试分析人员应根据测试合同(或项目计划)以及被测试软件的设计文档来对被测试软件的各模块进行分析,并确定以下内容。

(1) 确定测试充分要求根据软件单元的重要性、测试目标和约束条件,确定测试应覆盖的范围及每一范围所要求的覆盖程度,如分支覆盖率、语句覆盖率、功能覆盖率等,单元的每一个软件特性应至少被一个正常的测试用例和一个异常的测试用例覆盖。

(2) 确定测试终止的要求。指定测试过程正常终止的条件(如是否达到测试的充分性要求),并确定导致测试过程异常终止的可能情况(如软件编码错误)。

(3) 确定用于测试的资源要求。包括软件(如操作系统、编译软件、静态分析软件、测试驱动软件等)、硬件(如计算机、设备接口等)、人员数量、人员技能等。

(4) 确定需要测试的软件特征。根据软件设计文档的描述确定软件单元的功能、性能、状态、接口、数据结构、设计约束等内容和要求,并对其标识。若有需要可将其分类,并从中确定需测试的软件特性。

(5) 确定测试需要的技术和方法,如测试数据生成和验证技术等。

(6) 根据测试合同(或项目计划)的要求和被测试软件的特点确定测试准出条件。

(7) 对测试工作进行风险分析与评估,并制定应对措施。

(8) 确定由资源和被测试软件决定的软件单元测试活动的进度。

根据上述分析研究结果,按照测试规范要求编写软件单元测试计划。单元测试计划在软件详细设计阶段完成,其制订的主要依据是“软件需求说明书”“软件详细设计说明书”等。同时,还要参考并符合软件的整体测试计划。

单元测试计划的主要内容包括测试时间表、资源分配使用表、测试的基本策略和方法,如是否需要执行静态测试,是否需要测试工具,是否需要编制驱动模块和桩模块等。

应对软件单元测试计划进行评审。评审测试的范围和内容、资源、进度、各方责任等是否明确,测试方法是否合理、有效和可行,风险的分析、评估与对策是否准确可行,测试文档是否符合规范,测试活动是否独立等都属于评审范围。一般情况下,由软件的供方自行组织评审,在单元测试计划通过评审后,进入下一步工作。

单元测试计划完成后并不是立刻进入单元测试,这个时候代码可能还未完成。在代码编制时,软件详细设计文档有可能发生变化,要及时根据最新的详细设计文档来更新“单元测试计划”,并对其进行评审。

3. 测试设计

软件单元测试的设计工作由测试设计人员和测试程序员完成,一般根据单元测试计划完成以下工作。

(1) 设计测试用例。将需要测试的软件特性进行分解,针对分解后的每种情况设计测试用例。

(2) 获取测试数据。包括获取现有的测试数据和生成新的数据,并按照要求验证所有数据。

(3) 确定测试顺序。可从资源约束、风险以及测试用例失效造成的影响或后果等几个方面考虑。

(4) 获取测试资源。支持测试的软件和硬件,有的需要从现有的工具中选定,有的需要另行开发。

(5) 编写测试程序。包括开发测试支持工具、单元测试的驱动模块和桩模块。

(6) 建立和校准测试环境。

(7) 按照测试规范的要求编写软件单元测试说明。

(8) 应对软件单元测试说明进行评审,评审测试用例是否正确、可行、充分,测试环境是否正确、合理,测试文档是否符合规范。通常由软件测试方自行组织单元测试的评审,通过评审后进入下一步工作。

4. 测试执行

执行测试的工作由测试员和测试分析员完成。

测试员的主要工作是按照单元测试计划和单元测试说明的内容和要求执行测试。单元测试计划是整个单元测试的核心,在执行过程中,应认真观察并如实地记录测试过程、测试结果和发现的差错,填写测试记录。

测试分析员的工作有以下两方面。

(1) 根据每个测试用例的期望测试结果、实际测试结果和评审准则判定该测试用例是否通过。如果不通过,测试分析员应认真分析情况,并根据情况采取相应措施。

(2) 当所有的测试用例都执行完毕,测试分析员要根据测试的充分性要求和失效记录确定测试工作是否充分,是否需要增加新的测试。当测试过程正常终止时,如果发现测试工作不足,测试分析员应对软件单元进行补充测试,直到测试达到预期要求,并将附加的内容记录在单元测试报告中。当测试过程异常终止时,应记录导致终止的条件、未完成的测试和未被修改的差错等。

测试员依据需求定义、“详细设计说明书”等来完成单元测试用例的执行。并对测试中发现的错误或缺陷进行记录,生成“缺陷跟踪报告”,将该报告及时反馈给开发人员,以便开发人员及时修改。

如果需要进行静态测试,还要用到相应的标准及规范文档,编制“代码审查检查表”。

5. 测试总结

测试分析员应根据被测软件的设计文档、单元测试计划、单元测试说明、测试记录和软件问题报告单等分析和评价测试工作,一般应在单元测试报告中记录以下内容。

(1) 总结单元测试计划和单元测试说明的变化情况及其原因。

(2) 对测试异常终止情况,确定未能被测试活动充分覆盖的范围。

(3) 确定未能解决的软件测试时间以及不能解决的理由。

(4) 总结测试所反映的软件单元与软件设计文档之间的差异。

(5) 将测试结果连同所发现的出错情况同软件设计文档对照,评价单元的设计与实现,提出软件改进建议。

(6) 按照测试规范要求编写软件单元测试报告,内容包括测试结果分析、对单元的评价和建议。

(7) 根据测试记录和软件问题报告单编写测试问题报告。

应对单元测试执行活动、软件单元测试报告、测试记录和测试问题报告进行评审。评审测试执行活动的有效性、测试结果的正确性和合理性、测试目的是否达到、测试文档是否符合要求。一般情况下,评审由软件测试方自行组织。

软件单元测试完成后形成的文档一般应有软件单元测试计划、软件单元测试说明、软件单元测试报告、软件单元测试记录和软件单元测试问题报告等。可以根据需要对以上文档及文档的内容进行裁剪。

5.3 单元测试的分析


视频讲解

5.3.1 单元测试的策略

单元测试通常在编码阶段进行。在源程序代码编制完成,经过评审和验证,确认没有语法错误之后,便可以开始设计单元测试用例。

由于模块并不是一个独立程序,在考虑测试模块时,同时要考虑与其有关的外界联系,因此可以使用一些辅助模块去模拟与被测试模块相关的其他模块。辅助测试模块分为以下两种。

(1) 驱动模块(Driver):用来模拟被测模块的上级调用模块,功能要比真正的上级模块简单得多,仅仅是接受测试数据,并向被测试模块传递测试数据,启动被测模块,回收并输出测试结果。

(2) 桩模块(Stub):用来模拟被测模块在执行过程中调用的模块。它接受被测模块输出的数据并完成它被指派的任务。

图 5-1(a)表示被测软件的结构,图 5-1(b)表示用驱动模块和桩模块建立的测试模块 B 的环境。

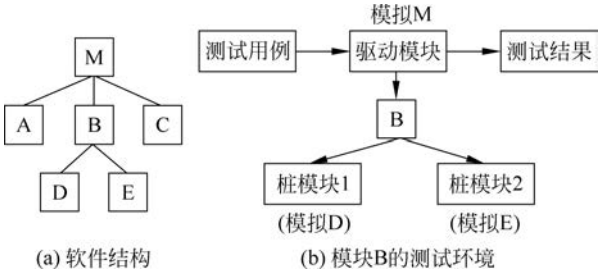

图 5-1 单元测试的环境

驱动模块和桩模块的编写会给软件开发带来额外开销,并且它们不需要和最终的软件一起提交。因此,应在保证测试质量的前提下尽量避免开发驱动模块和桩模块,以降低测试工作量。当需要模拟的单元比较简单的时候(如代码段很短、代码结构简单、不含有复杂的

循环和逻辑判断、不涉及复杂的动态内存分配和释放等),则无须专门设计驱动模块和桩模块,可以直接将测试代码与被测单元放在一起执行测试。但当被测单元较为复杂时,最好利用驱动模块或桩模块构建测试环境来运行程序。设计桩模块时,最好结合已有的测试用例来设计测试数据,使桩模块在最重要的功能和数据上实现对原始模块的正确模拟,而设计驱动模块时也应结合已有的测试用例,利用用例的测试数据来驱动被测单元,从而降低设计和编写驱动程序的工作量。

5.3.2 单元测试的用例设计

测试用例设计遵循与软件设计相同的工程原则。单元测试用例设计的具体阶段如下。

(1) 测试策略。

(2) 测试计划。

(3) 测试描述。

(4) 测试过程。

上述 4 个设计阶段适用于从单元测试到系统测试各个层面的测试。测试设计由软件设计说明驱动。单元测试用于验证模块单元实现了模块设计中定义的规格,一个完整的单元测试说明应该包含正面测试(Positive Testing)和负面的测试(Negative Testing)。正面测试验证程序应该执行的工作,负面测试则验证程序不应该执行的工作。

设计富有创造性的测试用例是测试设计的关键。本节将介绍测试说明的一般设计过程,描述一些结构化程序设计单元测试时采用的用例设计技术,同时也将增加介绍面向对象编程中对类进行单元测试采用的测试用例设计技术,这些可作为软件测试人员的参考阅读资料。

一旦模块单元设计完毕,下一个开发阶段就是设计单元测试。值得注意的是,如果在编写代码之前设计测试,测试设计就会显得更加灵活。一旦代码完成,对软件的测试可能会倾向于测试该段代码在做什么(这根本不是真正的测试),而不是测试其应该做什么。单元测试说明实际上由一系列单元测试用例组成,每个测试用例应该包含 4 个关键元素。

(1) 被测单元模块的初始状态声明,即测试用例的开始状态(仅适用于被测单元维持调用前状态的情况)。

(2) 被测单元的输入,包含由被测单元读入的任何外部数据值。

(3) 该测试用例实际测试的代码,用被测单元的功能和测试用例设计中使用的分析来说明,如单元中哪一个决策条件将被测试。

(4) 测试用例的期望输出结果。测试用例的期望输出结果总是应该在测试进行之前在测试说明中定义。

下面描述进行测试用例设计的 7 步通用步骤。

1) 首先使被测单元运行

任何单元测试说明的第一个测试用例都应该以一种可能的简单方法执行被测单元。看到被测单元第一个测试用例的运行成功可以增强人的自信心。如果其不能被正确执行,最好选择一个尽可能简单的输入对被测单元进行测试/调试。

这个阶段适合的技术有以下两种。

(1) 模块设计导出的测试。

(2) 对等区间划分。

2）正面测试

正面测试的测试用例用于验证被测单元能否执行应该完成的工作。测试设计者应该查阅相关的设计说明，使每个测试用例测试模块设计说明中的一项或多项陈述。如果涉及多个设计说明，最好使测试用例的序列对应一个模块单元的主设计说明。

正面测试适合使用的技术如下。

（1）设计说明导出的测试。

（2）对等区间划分。

（3）状态转换测试。

3）负面测试

负面测试用于验证软件是否执行其不应该完成的工作。这一步骤主要依赖于错误猜测，需要依靠测试设计者的经验来判断可能出现问题的位置。

适合负面测试的技术如下。

（1）错误猜测。

（2）边界值分析。

（3）内部边界值测试。

（4）状态转换测试。

4）设计需求中其他测试特性用例设计

如果需要，应该针对性能、余量、安全需要、保密需求等来设计测试用例。在有安全保密需求的情况下，重视安全保密分析和验证是方便的。针对安全保密问题的测试用例应该在测试说明中进行标注，同时应该加入更多的测试用例来测试所有的保密和安全风险问题。

适合这一阶段使用的技术主要是设计说明导出的测试。

5）覆盖率测试用例设计

应该增加更多的测试用例到单元测试说明中以达到特定测试的覆盖率目标。一旦覆盖测试设计好，就可以构造测试过程和执行测试。覆盖率测试一般要求语句覆盖率和判断覆盖率达到一定水准。

适合应用于这一阶段的技术如下。

（1）分支测试。

（2）条件测试。

（3）数据定义—使用测试。

（4）状态转换测试。

6）测试执行

使用上述5个步骤设计的测试说明在大多数情况下可以实现一个比较完整的单元测试。到这一步，就可以使用测试说明来构造实际用于执行测试的测试过程。该测试过程可能是特定测试工具的一个测试脚本。

测试过程的执行可以查出模块单元的错误，然后进行修复和重新测试。在测试过程中的动态分析可以产生代码覆盖率测量值，以指示覆盖目标已经达到指定的要求。因此需要在测试设计说明中增加一个完善代码覆盖率的步骤。

7）完善代码覆盖率

由于模块单元的设计文档规范不一，故在测试设计中可能引入人为的错误，测试执行

后,复杂的决策条件、循环和分支的覆盖率目标可能并没有达到,这时就需要进行分析找出原因。导致一些重要执行路径没有被覆盖的原因可能有以下几个。

(1) 不可行路径或条件。应该在测试说明中标注证明该路径或条件没有测试的原因。

(2) 不可到达或冗余代码。正确处理方法是删除这种代码。这种分析容易出错,特别是使用防卫式程序设计技术(Defensive Programming Techniques)时,如有疑义,这些防卫性程序代码就不要删除。

(3) 测试用例不足。应该重新提炼测试用例,设计更多的测试用例并将其添加到测试说明中,以覆盖没有执行过的路径。

理想情况下,完善代码覆盖率这一步骤应该在不阅读实际代码的情况下进行。然而,为达到覆盖率目标,看一下实际代码也是需要的。该步骤重要程度相对小一些。最有效的测试来自分析和说明,而不是来自试验,依赖这个步骤补充一份好的测试设计。

适合这一阶段的技术如下。

(1) 分支测试。

(2) 条件测试。

(3) 设计定义—试验测试。

(4) 状态转换测试。

注意到前面产生测试说明的 7 个步骤可以用下面的方法完成。

(1) 通常应该避免依赖先前测试用例的输出,在测试用例的执行序列早期发现的错误可能导致其他的错误,这些错误的出现会产生判定跳转,从而造成测试执行时实际测试代码量的减少。

(2) 测试用例设计过程中,包括作为试验执行这些测试用例时,常常可以在软件构建前就发现 Bug,还有可能在测试设计阶段比测试执行阶段发现更多的 Bug。

(3) 在整个单元测试设计中,主要的输入应该是被测单元的设计文档。在某些情况下,需要将试验实际代码作为测试设计过程的输入,测试设计者必须意识到不是在测试代码本身。从代码构建出来的测试说明只能证明代码执行完成的工作,而不是代码应该完成的工作。

5.4 单元测试的案例

本节利用风靡全球的“俄罗斯方块游戏排行榜”的程序作为案例来串讲本章的内容。

5.4.1 测试策划

1. 目的

俄罗斯方块游戏(Tetris)的排行榜功能经过编码后,在与其他模块进行集成之前,需要经过单元测试,测试其功能点的正确性和有效性。以便在后续的集成工作中不会引入更多的问题。

2. 背景

俄罗斯方块是一款风靡全球的电视游戏机和掌上游戏机游戏,它由俄罗斯人阿列克谢·帕基特诺夫发明,故得此名。俄罗斯方块的基本规则是移动、旋转和摆放游戏自动输出的各种方块,使之排列成完整的一行或多行并且消除得分。

排行榜功能是俄罗斯方块游戏中不可或缺的一部分，其用于将当前用户的得分与历史得分记录进行比较并重新排序。

该程序主要涉及的功能点有历史记录文件的读取、分数排名的计算与排序、新记录文件的保存、新记录的显示等。这些功能将在一局游戏结束，并获取到该局游戏的得分后启动。

3. 待测源代码

```
1   private void_gameOver (int_score)//游戏结束
2   {//Display game over
3      string s = "您的得分为：";
4      string al = "";
5      char[ ] A = {};
6      int i = 1;
7      _blockSurface.FontStyle = new Font(FontFace,BigFont); //设置基本格式
8      _blockSurface.FontFormat.Alignment = StringAlignment.Near;
9      _blockSurface.DisplayText = "GAME OVER!!";
10     string sc = Convert.ToString(_score); //得到当前玩家的分数
11      //write into file;
12     string path = "D:\test2.txt"; //文件路径
13     try{
14      FileStream fs = new FileStream
15                (path,FileMode.OpenOrCreate,FileAccess.ReadWrite);
16      StreamReader strmreader = new StreamReader(fs); //建立读文件流
17      String[ ] str = new String[5];
18      String[ ] split = new String[5];
19      while(strmreader.Peek()!= -1)
20        {
21          for(i = 0;i < 5;i++)
22          {
23            str[i] = strmreader.ReadLine();  //以行为单位进行读取,赋予数组
24                                             //str[i]
25            //按照": "将文字分开,赋予数组 split [i]
26            split[i] = str[i].split(':')[1];
27          }
28        }
29        person1 = Convert.ToInt32(split[0]); //split[0]的值赋予第一名
30        person2 = Convert.ToInt32(split[1]); //split[1]的值赋予第二名
31        person3 = Convert.ToInt32(split[2]); //split[2]的值赋予第三名
32        person4 = Convert.ToInt32(split[3]); //split[3]的值赋予第四名
33        person5 = Convert.ToInt32(split[4]); //split[4]的值赋予第五名
34        strmreader.Close(); //关闭流
35        fs.Close();
36        FileStream ffs = new
37              FileStream(path,  FileMode.OpenOrCreate,
38  FileAccess.ReadWrite));
39       StreamWriter sw = new StreamWriter(ffs)//建立写文件流
40        if(_score > person1)//如果当前分数大于第一名,排序
41        { person5 = person4; person4 = person3; person3 = person2; person2 = person1; p
42          erson1 = _score;}
43        else if(_score > person2)//如果当前分数大于第二名,排序
44        { person5 = person4; person4 = person3; person3 = person2; person2 =
45        _score;}
46        else if(_score > person3)//如果当前分数大于第三名,排序
```

```
47          { person5 = person4; person4 = person3; person3 = _score;}
48          else if(_score > person4)//如果当前分数大于第四名,排序
49          { person5 = person4; person4 = _score;}
50          else if(_score > person5)//如果当前分数大于第五名,排序
51          { person5 = _score;}
52          //在文件中的文件内容
53          string pp1 = "第一名: " + Convert.ToString(person1);
54          string pp2 = "第二名: " + Convert.ToString(person2);
55          string pp3 = "第三名: " + Convert.ToString(person3);
56          string pp4 = "第四名: " + Convert.ToString(person4);
57          string pp5 = "第五名: " + Convert.ToString(person5);
58          string
59          ppR = pp1 + "\r\n" + pp2 + "\r\n" + pp3 + "\r\n" + pp4 + "\r\n" + pp5 + "\r\n";
60          byte[] info = new UTF8Encoding(true).GetBytes(ppR);
61          sw.Write(ppR);              //将内容写入文件
62          sw.Close();
63          ffs.Close();
64        }
65      Catch(Exception ex)
66      {
67        Console.WriteLine(ex.ToString());
68      }
69      s = s + "" + sc;
70      //Draw surface to display text;
71      MessageBox.Show(s);             //在界面中显示排行榜内容
72  }
```

5.4.2 测试设计

下面将利用相关静态和动态(白盒测试、黑盒测试)方法对案例进行相应的测试,得到测试报告与错误列表,在实际项目中可进一步反馈给开发方进行 Bug 的确认与修复。

1. 代码走查

利用代码走查的方法检查该模块的代码,对代码质量进行初步评估。具体实现如表 5-1 所示。

表 5-1 代码走查情况记录

序号	项目	发现的问题
1	程序结构	① 代码结构清晰,具有良好的结构外观 ② 函数定义清晰 ③ 结构设计能够满足机能变更 ④ 整个函数组合合理 ⑤ 所有主要的数据构造描述清楚、合理 ⑥ 模块中所有的数据结构都定义为局部的 ⑦ 为外部定义了良好的函数接口
2	函数组织	① 函数都有一个标准的函数头声明 ② 函数组织:头、函数名、参数、函数体 ③ 函数都能够在最多两页纸上打印 ④ 所有变量声明每行只声明一个 ⑤ 函数名小于 64 个字符

续表

序号	项目	发现的问题
3	代码结构	① 每行代码都小于 80 个字符 ② 所有的变量名都小于 32 个字符 ③ 所有的行每行最多只有一句代码或一个表达式 ④ 复杂的表达式具备可读性 ⑤ 续行缩进 ⑥ 括号在合适位置 ⑦ 注解在代码上方,注释的位置不太好
4	函数	① 函数头清楚地描述函数及其功能 ② 代码中几乎没有相关注解 ③ 函数的名字清晰地定义了它的目标以及函数所做的事情 ④ 函数的功能清晰定义 ⑤ 函数高内聚；只做一件事情并做好 ⑥ 参数遵循一个明显的顺序 ⑦ 所有的参数都被调用 ⑧ 函数的参数个数小于 7 个 ⑨ 使用的算法说明清楚
5	数据类型与变量	① 数据类型不存在数据类型解释 ② 数据结构简单,以便降低复杂性 ③ 每一种变量都没有明确分配正确的长度、类型和存储空间 ④ 每一个变量都初始化了,但并不是每一个变量都在接近使用它的地方才初始化 ⑤ 每一个变量都在最开始的时候初始化 ⑥ 变量的命名不能完全、明确地描述该变量代表什么 ⑦ 命名不与标准库中的命名相冲突 ⑧ 程序没有使用特别的、易误解的、发音相似的命名 ⑨所有的变量都用到了
6	条件判断	① 条件检查和结果在代码中清晰 ② if/else 使用正确 ③ 普通的情况在 if 下处理而不是 else ④ 判断的次数降到最小 ⑤ 判断的次数不大于 6 次,无嵌套的 if 链 ⑥ 数字、字符、指针和 0/null/false 判断明确 ⑦ 所有的情况都考虑到 ⑧ 判断体足够短,使得一次可以看清楚 ⑨ 嵌套层次小于 3 次 ⑩ 条件比较的常量没有写到前面
7	循环	① 循环体不空 ② 循环之前做好代码初始化 ③ 循环体能够一次看清楚 ④ 代码中不存在无穷次循环 ⑤ 循环的头部进行循环控制 ⑥ 循环索引没有有意义的命名 ⑦ 循环设计得很好,它只干一件事情

续表

序号	项目	发现的问题
7	循环	⑧ 循环终止的条件清晰 ⑨ 循环体内的循环变量起到指示作用 ⑩ 循环嵌套的次数小于 3 次
8	输入输出	① 所有文件的属性描述清楚 ② 所有 OPEN/CLOSE 调用描述清楚 ③ 文件结束的条件进行检查 ④ 显示的文本无拼写和语法错误
9	注释	① 注释不清楚,主要的语句没有注释 ② 注释过于简单 ③ 看到代码不一定能明确其意义

从表 5-1 的分析中可以看出,本模块的代码基本情况如下。

(1) 代码直观。

(2) 代码和设计文档对应。

(3) 无用的代码已经被删除。

(4) 注释过于简单。

2. 基本路径测试法

基本路径测试法是在程序控制流图的基础上,通过分析控制构造的环路复杂性,导出可执行的路径集合,从而设计测试用例的方法。首先需要简化程序模块,绘制程序模块如图 5-2 所示。接着按照模块图的设计路径来覆盖策略。主要可分为以下 4 步执行。

1) 绘制程序的控制流图

基本路径测试法的第一步是绘制控制流图,根据程序模块图的逻辑关系,获得该程序块的控制流图,如图 5-3 所示。

2) 计算环路复杂度

其次是根据控制流图计算环路复杂度,环路复杂度是一种为程序逻辑复杂性提供定量测度的软件度量,该度量将用于计算程序基本的独立路径数目,为确保所有语句至少执行一次的测试数量的上界。

$$V(G)=P+1=5+1=6$$

根据以上公式确定至少要覆盖 6 条路径。

3) 导出独立路径

根据控制流图可以方便地得到以下 6 条路径。

path1:1—2—11。

path2:1—3—4—11。

path3:1—3—5—6—11。

path4:1—3—5—7—8—11。

path5:1—3—5—7—9—10—11。

path6:1—3—5—7—9—11。

4) 设计测试用例

最后设定一组初始参数,以此来设计测试用例。令:

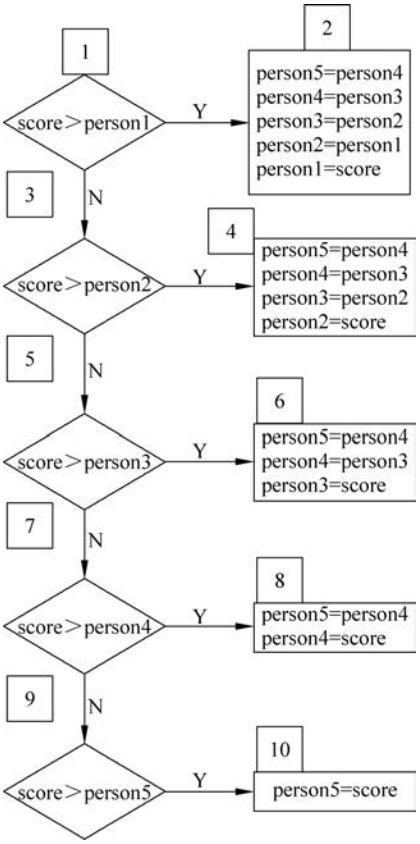

图 5-2　程序模块图

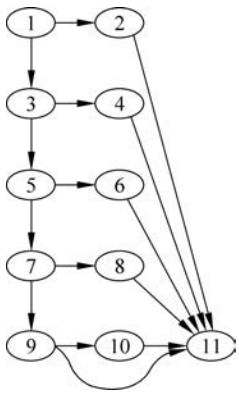

图 5-3　程序模块的控制流图

person1＝23

person2＝20

person3＝10

person4＝6

person5＝4

作为测试输入，可设计测试用例如表 5-2 所示。

表 5-2　基本路径法测试用例

编号	输入数据	输出数据					路径覆盖	判断覆盖
	score	person1	person2	person3	person4	person5		
1	24	24	23	20	10	6	1-2-11	T
2	21	23	21	20	10	6	1-3-4-11	FT
3	15	23	20	15	10	6	1-3-5-6-11	FFT
4	8	23	20	10	8	6	1-3-5-7-8-11	FFFT
5	5	23	20	10	6	5	1-3-5-7-9-10-11	FFFFT
6	0	23	20	10	6	4	1-3-5-7-9-11	FFFFF

3. 边界值分析

边界值分析法利用输入变量的最小值、略大于最小值、输入范围内任意值、略小于最大

值、最大值等来设计测试用例。

由于输入的只会是数据,且数据均大于0,因此可令:

person1 = 23

person2 = 20

person3 = 10

person4 = 6

person5 = 4

采用边界值法设计测试用例如表5-3所示。

表5-3 边界值法测试用例

序号	测试内容	输入数据(score)	期望结果
1	从大到小排序	23	person1=23 person2=23 person3=20 person4=10 person5=6
2	从大到小排序	24	person1=24 person2=23 person3=20 person4=10 person5=6
3	从大到小排序	4	person1=23 person2=20 person3=10 person4=6 person5=4
4	从大到小排序	3	person1=23 person2=20 person3=10 person4=6 person5=4
5	从大到小排序	12	person1=23 person2=20 person3=12 person4=10 person5=6

5.4.3 测试执行

将设计的测试用例整理合并为测试用例集合,必要时需要开发相应的驱动模块和桩模块。本次测试需要开发一个驱动模块,用于初始化相应的参数,并调用待测模块以达到测试效果。驱动模块代码如下。

```
1  import java.io.BufferedReader;
2  import java.io.IOException;
3  import java.io.InputStreamReader;
4   public class Main(){
5    public static void main(String[] args)throws IOException{
6      int person1 = 23,person2 = 20,person3 = 10,person4 = 6,person5 = 4;
7      int score;
8      String s;
9      BufferedReader bf = new BufferedReader(new
10 InputStreamReader(System.in));
11     s = bf.readLine();
12     score = Integer.valueOf(s);
13     _gameOver(score);
14   }
15 }
```

5.4.4 测试总结

测试结果可利用Bug记录平台进行记录,在实际项目中则可反馈给开发人员,由开发人员确认并修复。

测试结束后,形成测试报告。

5.5 本章小结

通过单元测试可以验证开发人员编写的代码是否能按照设想的方式执行，并确保其产生符合预期值的结果，这就实现了单元测试的目的。相比后面阶段的测试，单元测试的创建更简单，维护更容易，并且可以更方便地重复进行。从全程的费用来考虑，比起那些复杂且旷日持久的集成测试，对一些不稳定的软件系统来说，单元测试所需的费用是很低的。

模块单元设计完毕之后的开发阶段就是单元测试。值得注意的是，如果在编写代码之前就进行单元测试，那么测试设计就会显得更加灵活。因为一旦代码完成，对软件的测试可能就会受制于代码，倾向于测试该段代码完成什么功能，而不是测试这段代码应该做什么。因此，应该把单元测试的设计放在详细设计阶段而非代码完成阶段。

5.6 习　　题

1. 以下关于单元测试的叙述，不正确的是(　　)

A. 单元测试是指对软件中的最小可测试单元进行检查和验证

B. 单元测试是在软件开发过程中要进行的最低级别的测试活动

C. 结构化编程语言中的测试单元一般是函数或子过程

D. 单元测试不能由程序员自己完成

2. 单元测试的测试内容包括(　　)。

① 模块接口。

② 局部数据结构。

③ 模块内路径。

④ 边界条件。

⑤ 错误处理。

⑥ 系统性能。

A. ①②③④⑤⑥　　B. ①②③④⑤　　C. ①②③④　　D. ①②③

3. 单元测试主要针对的是模块的几个基本特征，故该阶段不能完成的测试是(　　)。

A. 系统功能　　B. 局部数据结构

C. 重要的执行路径　　D. 错误处理