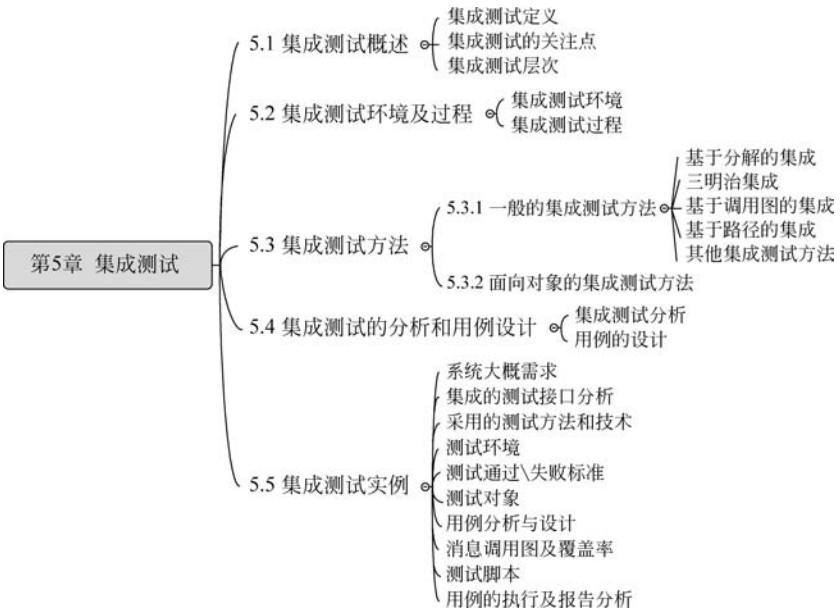
第5章 集成测试

第5章思维导图



第 1 章对集成测试做了简单描述，本章将对集成测试进行详细分析。

5.1 集成测试概述

1. 集成测试定义

集成测试(Integration Testing)也叫组装测试、联合测试，是在单元测试基础上，将所有模块按照概要设计的要求组装成为子系统或系统的测试，是对模块间接口或系统的接口以及集成后的子系统或系统的功能进行正确性检验的一项测试工作。一般来说，集成测试是由专门的测试机构组织软件测试工程师依据概要设计说明书进行的，集成测试必须遵循一定的测试过程，如制订集成测试计划等。

软件开发过程涉及从需求到需求分析、概要设计、详细设计以及编码等阶段的一个逐步细化的过程，从测试角度分析，单元测试到集成测试和系统测试的过程就是对系统的一个逆向验证的过程。在这个过程中，集成测试是介于单元测试和系统测试之间的过渡阶段，起到承上启下的作用，与软件概要设计阶段相对应，可以理解为单元测试的扩展和延伸。在进行集成测试之前，单元测试应该已经完成，并且集成测试所使用的对象应当是已经成功地通过了单元测试

的单元。如果没通过单元测试或者没做单元测试,那么集成测试会出现除接口之外的其他涉及单元本身的各种问题。另外,所有的软件项目都不能摆脱系统集成这个阶段。不管采用什么开发模式,软件单元只有经过集成才能形成一个有机的整体。由于具体的集成过程可能是显性的也可能是隐性的,因此只要有集成测试,总会出现一些常见问题,工程实践中几乎不存在软件单元集成过程中不出任何问题的情况。一般来说,集成测试需要花费的时间远远超过单元测试,直接从单元测试过渡到系统测试是极不妥当的做法,没通过单元测试或者没做单元测试就进入集成测试都是不可取的。

最简单的集成测试形式就是把两个单元模块集成或者说组合到一起,然后测试这两个模块之间的接口。当然实际项目的集成测试过程复杂得多,通常要根据具体情况采取不同的集成测试策略将多个模块组装成为子系统或系统,以验证在各单元模块通过测试的前提下各个模块能否以正确、稳定、一致的方式进行接口和交互,即验证其是否符合软件开发过程中的概要设计说明书的要求。集成测试的用例设计一般采用黑盒测试方法,但随着测试技术的发展以及软件系统复杂度的增加,尤其是在大型的跨平台的应用软件中,常常会使用白盒测试(如将不同的跨平台系统的业务路径组成更长的路径)与黑盒测试相结合的方法进行测试用例的设计。集成测试具有回归特性。

2. 集成测试的关注点

在前面的论述中了解到,集成测试主要验证通过单元测试之后的模块之间接口的正确性以及各个模块集成后系统功能的正确性和完整性。那么,在进行集成测试时应该重点关注哪几个方面呢?总结如下:

(1) 各个模块集成起来后,通过模块接口的交互的参数数量、参数数据类型、参数顺序等是否一致,是否有数据丢失,是否能够按期望的要求传递给另外一个模块。

(2) 各个模块集成起来后,是否仍然存在单元测试时所没发现的问题。

(3) 通过单元测试的子功能模块集成到一起能否实现所期望的父功能。例如,在ATM系统中,卡检验模块、密码验证模块、存款处理模块、显示打印模块集成后是否能实现正常的取款功能。

(4) 在集成过程中,随着新的被集成模块的加入,是否对其他已经集成的模块产生负面影响。

(5) 全局数据结构是否正确,数据结构的内部构成是否被不正常地修改。

(6) 随着集成的深入,系统的特性误差,尤其是功能方面的特性误差是否会累计扩大,是否会达到不可接受的程度。

(7) 在与用户界面的集成中,控件的输入内容检查、结果显示、数据类型控制等方面是否合理。

3. 集成测试层次

在第1章分析了开发和测试模型,可以看出一个软件产品要经历多个不同的开发和测试阶段。从需求→需求分析→概要设计→详细设计→编码这个过程是一个分层设计和不断细化的过程,是一个抽象的过程,编码是最底层的抽象。也就是说,这个过程经过分层的设计,由大到小逐步细化最终完成整个软件的开发。而软件测试则要从单元测试开始,然后对所有通过单元测试的模块进行集成测试,最后将系统的所有组成元素组合到一起进行系统测试,再经过验收测试到交付。那么,从集成测试本身而言又该如何理解集成的层次呢?

对于使用传统的结构化技术开发的软件系统而言,在集成时,按集成粒度不同,可以把集

成测试分为 4 个层次：

(1) 模块内集成。如果模块内部包括不同的函数或过程，则可能需要模块内集成。

(2) 子系统内集成。子系统是由不同的模块构成的，所以必须完成这些模块间的集成，集成时以模块结构图为依据。

(3) 子系统间集成。如果系统包含有多个相互独立的子系统，如某系统包含报表子系统、通信子系统、批处理子系统、业务子系统等，需要通过子系统间的集成测试，才能把各子系统组合到一起。集成时以软件结构图为依据。

(4) 不同系统之间的集成。如网上图书销售系统和银行系统间的集成实现网上购书在线支付。

对于使用面向对象技术开发的软件系统而言，在集成时，按集成粒度不同，可以把集成测试分为 4 个层次：

(1) 类内集成。对类内的不同的方法进行集成，可以依据类状态等作为集成依据。

(2) 类间集成。类实例化后，类之间有消息传递，类间集成可以以序列图和协作图为依据。

(3) 子系统间集成。分析同上。

(4) 不同系统之间的集成。分析同上。

5.2 集成测试环境及过程

1. 集成测试环境

集成测试的环境因被集成的系统不同而不同，如 MIS、在线事务处理系统、嵌入式系统等。对这些系统而言，由于其开发环境、运行环境的不同将导致集成环境的不同。同一软件系统，其集成测试环境和单元测试环境二者有相似之处，但相对于单元测试环境而言，集成测试环境的搭建比较复杂。随着各种软件构件技术(如 Microsoft 公司的 COM、SUN 公司的 J2EE、IBM 公司的 Eclipse 等)的不断发展，以及软件复用技术思想的不断成熟和完善，可以使用不同技术、基于不同平台并依据现成构件集成一个应用软件系统，这使得软件复杂性也随之增加。因此在做集成测试的过程中，可能需要利用一些专业的测试工具或测试仪来搭建集成测试环境(如测试 Java 类和服务器交互的工具 HttpUnit、测试网页链接的工具 LinkBot Pro 等)。必要时，还要开发一些专门的接口模拟工具。

在搭建集成测试环境时，可以考虑以下因素：

1) 硬件环境

在集成测试时，应尽可能考虑实际的环境。如果使用实际环境有困难且不可用，则考虑可替代的环境或在模拟环境下进行。如果测试在模拟环境下进行，还需要分析模拟环境与实际环境之间可能存在的差异。对于普通的应用软件来说，由于对软件运行速度影响最大的硬件环境主要是内存和硬盘空间的大小及 CPU 性能的优劣，因此，在搭建集成测试的硬件环境时，应该注意到测试环境和软件实际运行环境的差距。例如，很多中小型软件企业一般都是在 PC 上开发软件，甚至测试的时候也使用 PC。显而易见，在 PC 上所做的性能测试结果将会和软件在实际环境中运行的性能有很大差别。但是，集成测试的硬件环境还有可能涉及除计算机设备以外的硬件，如 ATM、条形码读码器、传送带、传感器等设备，集成测试时要么使用真实的设备，要么开发模拟器来模拟，但大多数情况下以开发模拟器为主。

2) 操作系统环境

目前,操作系统的种类很多,同一种类的操作系统也存在不同的版本,这给集成测试带来麻烦和挑战。同样一个模块、一个子系统或一个软件系统在不同的操作系统环境中运行的表现可能会有很大差别,如,在基于 Windows 和 Linux 的操作系统中往往同样的软件在这两个系统中均能运行起来,但运行的效果会不同,可能存在运行的结果不同、显示结果不一样或不显示等情况。因此在对软件进行集成测试时不但要考虑不同机型,而且要考虑到实际环境中安装的各种具体的操作系统环境,甚至,要考虑同一个操作系统的不同版本。在集成测试中充分考虑操作系统对与软件系统的后续兼容性测试(系统测试级别)是否有益。

3) 数据库环境

目前几乎所有的应用都是基于大型关系数据库的,常见的数据库系统有 Oracle、MySQL、SQL Server 和 Neo4j、TigerGraph、MongoDB 等。如 Oracle 和 SQL Server 在表的创建、数据访问、数据管理、数据备份及数据恢复等方面存在诸多不同,在开发应用系统时,用户会根据各自的实际情况来选择适合自己的数据库产品。由于基于不同的数据库系统开发的应用程序存在差别,如,很难做到在 Oracle 下开发的应用在 SQL Server 下能很正常地运行,而在集成测试过程中数据库往往起到测试数据源的载入(提供测试用例中需要的相关数据),测试的中间数据和最终结果数据保存的作用,因此,在搭建集成测试时要充分考虑所使用的数据库。可能的情况是要针对常见的几种数据库产品进行测试,形成不同的数据库系统的版本。另外,如图数据库 Neo4j 和非关系型数据库 MongoDB 集成测试时也存在类似问题。

4) 网络环境

集成测试的网络环境也是千差万别,可以是企业内部网,也可以是外网。一般来说,除了进行跨不同平台的系统集成测试外,集成测试可以把公司内部的网络环境作为集成测试的网络环境。

5) 测试工具环境

有时集成测试必须借助测试工具才能够完成,因此也需要搭建一个测试工具能够运行的环境。

6) 开发驱动器和桩

和单元测试的情况类似,集成测试一般会涉及驱动器和桩的开发,如在自顶向下的集成方法中就需要开发桩;在自底向上的集成方法中就需要开发驱动器。

7) 其他环境

除了上面提到的集成测试环境外,集成测试还可能涉及一些其他环境,如:Web 应用所需要的 Web 服务器环境、浏览器环境及测试管理工具和缺陷跟踪工具的集成环境等。这就要求测试人员根据具体要求进行搭建。

图 5-1 显示了一个系统的集成测试环境,供读者参考。

2. 集成测试过程

和单元测试一样,集成测试也包含不同的阶段,一般可以把集成测试划分为 5 个阶段:计划阶段、用例分析和设计阶段、实施阶段、执行阶段、分析评估阶段。在实际集成测试过程中可能其阶段有所不同,读者可以参考 IEEE 制定的相关标准。

1) 计划阶段

在第 4 章中简单介绍过测试计划的重要性,一个计划的好与坏直接影响着后续测试工作的进行。计划本身而言是动态的,任何计划都不可能一成不变,计划要适时适应变化,集成测

试计划也是如此。如由于需求的改变、技术的调整、人员的变化及工具的使用等原因都可能导致计划的调整。

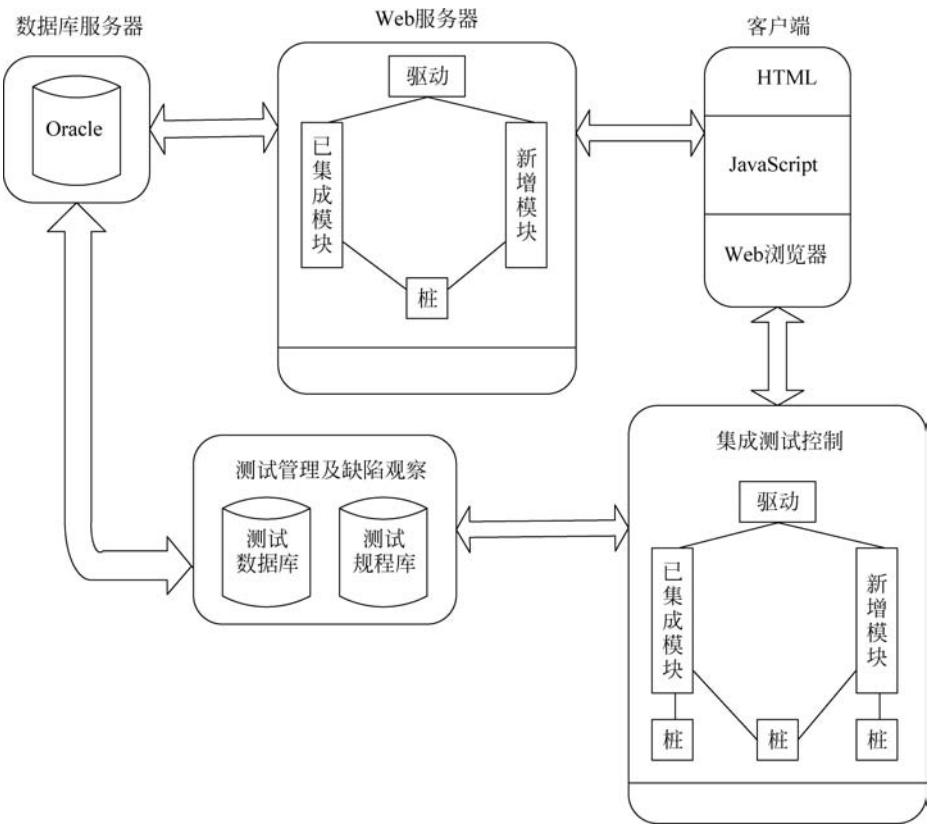


图 5-1　一个集成测试环境示意图

那么，应该在软件测试生命周期中的哪一个阶段制订集成测试计划呢？集成测试的计划一般在概要设计评审通过后进行，参考需求规格说明书、概要设计说明书、项目开发计划、单元测试计划及相关报告来制订。集成测试计划的制订涉及多方面，如：

- 确定集成测试的对象和测试范围。
- 根据集成测试的测试对象的数量及难度估算工作量，进而可以估算成本。
- 确定集成测试组织结构、角色分工和工作任务的划分。
- 标识集成测试各个阶段的开始和结束时间、任务及约束等条件。
- 根据集成测试过程中可能的风险，进行风险分析并制订分析应急计划，如时间风险、技术风险等。
- 考虑集成测试工具的选用、测试设备及环境搭建等资源。
- 考虑外部技术支援的力度和深度，以及相关培训安排。
- 确定集成测试使用的技术。
- 确定集成测试中出现缺陷的跟踪处理流程。
- 定义进入集成测试和退出集成测试的标准。

集成测试计划定稿之前可能要经过几次修改和调整才能够完成，直到通过评审并授权为止。以下是一个集成测试计划的模板，供参考。

第1章 引言

1.1 目的

本文是××系统的集成测试的大纲,主要描述如何进行集成测试活动、如何控制集成测试活动、集成测试活动的流程以及集成测试活动的工作安排等,以保证程序集成起来能正常工作,保证程序的完整运行。

1.2 范围

本测试计划主要是针对软件的集成测试:不含硬件、系统测试以及单元测试(完成单元测试是前提)。

主要的任务:

(1) 测试在把各个模块连接起来的时候,穿越模块接口的数据是否会丢失;

(2) 测试各个子功能组合起来,能否达到预期要求的父功能;

(3) 一个模块的功能是否会对另一个模块的功能产生不利的影响;

(4) 全局数据结构是否有问题;

(5) 单个模块的误差积累起来,是否会放大,从而达到不可接受的程度。

主要测试方法是黑盒测试方法。必要的集成测试是回归测试。

本文主要的读者对象是项目负责人、集成部门负责人、集成测试设计师。

1.3 术语

和业务及技术相关的术语。

1.4 测试环境

举例如表1所示。

表1 测试环境

序　　号	描　　述	配　　置
1#	浏览器	IE &Firefox
2#	输入习惯	中文
3#	操作系统环境	Windows 10
4#	测试工具	Selenium
5#	输入设备	条形码读码器

1.5 参考文件及工作产品

开始测试涉及以下文档:

- 《需求分析规约》——Requirement Analysis Specification。
- 《项目开发计划》——Project Plan。
- 《概要设计说明书》——High Level Design Specification。
- 《详细设计说明书》——Detailed Level Design Specification。
- 《单元测试报告》——Module Test Report。

执行测试前涉及的任务:

- 用例设计完成并通过评审;
- 测试脚本开发完成;
- 测试环境搭建完成;

• 测试过程及缺陷管理流程确定。

测试结束时提交的文档：测试分析与评估报告。

第 2 章　集成策略

2.1　进入标准

编码完成，单元测试完成。集成测试计划完成，时间表、工具以及人员安排到位。

2.2　集成内容

1. 函数集成

如函数间接口、函数是否调用正常。

2. 功能集成

如不同函数间实现的业务功能。

3. 数据集成

如数据传递是否正确，对于传入值的控制范围是否一致等。

4. 子系统集成

如把不同通信子系统、业务子系统及报表子系统进行集成。

2.3　集成策略

假如系统的集成测试采用自底向上的集成(Bottom-Up Integration)方式。自底向上集成方式从程序模块结构中最底层的模块开始组装和测试。因为模块是自底向上进行组装的，对于一个给定层次的模块，它的子模块(包括子模块的所有下属模块)事前已经完成组装并经过测试，所以不再需要编制桩模块。选择这种集成方法，管理方便，测试人员能较好地锁定软件故障所在位置。

集成测试中的主要步骤：

(1) 制订并审核集成测试计划。

(2) 测试用例分析、设计及评审。

(3) 测试的实施。

(4) 测试的执行。

(5) 测试的分析和评估。

如表 2 所示。

表 2　集成测试的主要步骤

活　动	输　入	输　出	职　责
制订并审核集成测试计划	概要设计说明书等	集成测试计划	制订测试计划
测试用例分析和设计及评审	集成测试计划 概要设计说明书	设计测试用例	设计测试用例并评审
测试的实施	集成测试用例 测试过程	测试脚本 测试环境	开发测试脚本、搭建测试环境
		测试驱动或桩	开发驱动或桩
测试的执行	测试脚本	测试结果	记录结果、跟踪缺陷
测试的分析和评估	集成测试计划 测试结果	测试分析和评估报告	会同开发人员评估测试结果，得出测试报告

2.4 集成顺序

1. 软件集成顺序

集成顺序：例如，自底向上，先函数、数据、功能再子系统。

2. 软件/硬件集成顺序

无。

3. 子系统集成顺序

略。

第3章 测试过程描述

3.1 软件集成测试

以下举例说明。

在××项目中，集成测试的主要过程如下：

(1) 设计集成测试用例。

自底向上集成测试的步骤：

步骤1：按照概要设计规格说明，明确哪些是被测模块。在熟悉被测模块性质的基础上对被测模块进行分层，在同一层次上的测试可以并行进行，然后排出测试活动的先后关系，制订测试进度计划。

步骤2：在步骤1的基础上，按时间线序关系将软件单元集成为模块，并测试在集成过程中出现的问题。这里可能需要测试人员开发一些驱动模块来驱动集成活动中形成的被测模块。对于比较大的模块，可以先将其中的某几个软件单元集成为子模块，然后再集成为一个较大的模块。

步骤3：将各软件模块集成为子系统(或分系统)。检测各自子系统是否能正常工作。同样，可能需要测试人员开发少量的驱动模块来驱动被测子系统。

步骤4：将各子系统集成为最终用户系统，测试各子系统能否在最终用户系统中正常工作。

(2) 集成测试：组织人员按照(1)中的集成测试用例测试系统集成度。

① 测试人员按照测试用例逐项进行测试，并且将测试结果填写在测试报告上(测试报告必须覆盖所有测试用例)；

② 测试过程中发现Bug，将Bug填写在Bugzilla(缺陷跟踪工具)上发给集成部经理(Bug状态为NEW)；

③ 对应责任人接到Bugzilla通过email发过来的Bug信息；

④ 对于明显的并且可以立刻解决的Bug，将Bug发给开发人员(Bug状态为ASSIGNED)，对于不是Bug的提交，集成部经理通知测试设计人员和测试人员，对相应文档进行修改(Bug状态为RESOLVED，决定设置为INVALID)；对于目前无法修改的，将这个Bug放到下一轮次进行修改(Bug状态为RESOLVED，决定设置为REMIND)。

(3) 问题反馈：反馈Bug给开发人员。

① 开发人员接到发过来的Bug立刻修改(Bug状态为RESOLVED，决定设置为FIXED)；

② 测试人员接到Bugzilla通过email发过来的Bug更改信息，应该逐项复测，填写新的测试报告(测试报告必须覆盖上一次中所有REOPENED的测试用例)。

(4) 回归测试：重新测试修复Bug后的系统。重复(3)，直到(4)中回归测试结果到达系统验收标准。

如果复测有问题返回第(2)步(Bug 状态为 REOPENED),否则关闭这项 Bug(Bug 状态为 CLOSED)。

本轮测试中测试用例中有 90%一次性通过测试,结束测试任务;本轮测试中发现的 Bug 有 95%经过修改并且通过再次测试(即 Bug 状态为 CLOSED),返回进行新的一轮测试。

(5) 集成测试测试总结报告:完成以上 4 步后,综合相关资料生成报告。

(6) 进入系统测试:α 测试、β 测试。

如图 1 所示。

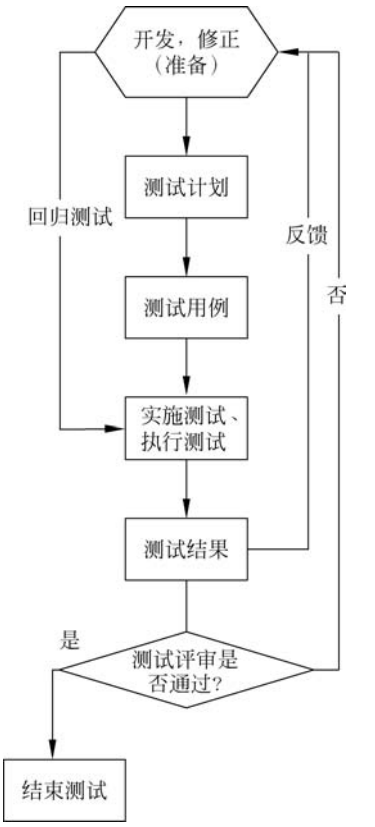


图 1 集成测试过程活动流程图

3.2 软件/硬件集成测试

主要涉及硬件和软件间集成,硬件和硬件间集成这里一般不涉及,集成关注:

(1) 功能点:根据用户文档列出所有功能点,检验其正确性。

(2) 接口:根据用户文档列出所有接口,检验其正确性。

(3) 流程处理:根据用户文档列出所有流程,检验其正确性。

(4) 外部接口:根据用户文档列出所有外部接口,检验其正确性。

3.3 子系统集成测试

完成子系统间集成。

第 4 章 集成测试验收标准

4.1 模块(指单元集成后的模块)验收标准

接口:接口提供的功能或者数据正确。

功能点:验证程序与产品描述、用户文档中的全部说明相对应,一致性。

流程处理:验证程序与产品描述、用户文档中的全部说明相对应,一致性。

外部接口:验证程序与产品描述、用户文档中的全部说明相对应,一致性。

4.2 集成测试验收标准

首先,集成测试用例中所设计的功能测试用例必须全部通过,性能及其他类型测试用例 95%以上通过。在未通过的测试用例中,不能含有“系统崩溃”和“严重错误”这两种错误,“一般错误”小于 1%。测试结果与测试用例中期望的结果一致,测试通过,否则标明测试未通过。

第 5 章 测试工具

5.1 测试工具

- 测试中心平台:Bugzilla;
- 性能测试工具:loadrunner;
- 集成测试工具:Selenium。

5.2 其他工具

电子表格软件：Excel。

图表工具软件：Microsoft Visio。

第6章 挂起、恢复和退出条件

6.1 挂起

举例：

- 进入第一轮集成测试，测试人员大体了解一下产品情况，如果发现在单元内存在三个以上错误或缺陷以及操作性的错误，退回单元测试组测试；
- 遇到有项目优先级更高的集成测试任务；
- 在复测过程中发现产品无法运行下去；
- 人员、设备不足；
- 重大突发紧急情况。

6.2 恢复

举例：

- 符合进入集成测试条件；
- 项目优先级更高的集成测试任务暂告完成；
- 复测过程中产品可以运行下去；
- 人员、设备到位；
- 突发事件处理完成。

6.3 退出

- 项目因故终止；
- 不可抗力：合同专用条款中约定等级以上的自然灾害也属不可抗力；
- 其他原因的测试工作频频被挂起或者挂起后迟迟恢复不了，并超过了客户要求的期限。

第7章 责任人和时间表

7.1 责任

测试负责人：×××。

控制并完成测试任务和测试过程，决定测试人员提交上来的Bug是否需要修改。

测试设计人员：×××，×××。

设计集成测试用例。

测试人员：×××，×××，×××。

按照测试用例进行测试活动。

开发人员：×××。

程序Bug修改，程序员间协调。

用户代表：无。

7.2 时间表

开始/结束时间表(略)。

第8章 记录和解决问题

记录：利用Bugzilla平台记录Bug，并指定相关责任人。更进一步，把Bugzilla和需求设计文档、开发文档、测试文档、测试用例等联系起来，做成一个软件研发工具套件，即可通过一个Bug方便地找到对应的文档、代码、测试用例等。

解决问题：小组会议以及开发人员协调负责人，协调测试开发之间的工作。

2）用例分析和设计阶段

一般在详细设计开始时就可以着手进行集成测试用例的分析和设计工作。可以以需求规格说明书、概要设计、集成测试计划文档作为参考依据。当然，必须在概要设计通过评审的前提下才可以进行。与制订集成测试计划一样，测试用例的分析和设计涉及多个活动环节。例如：

- 集成对象的结构分析。
- 被集成的模块及接口分析。
- 集成测试采用的策略分析。
- 采用的测试方法及测试工具分析。
- 集成测试环境分析。
- 测试用例的设计及评审。
- 测试用例集的覆盖标准分析。

通过上述活动之后，形成的工作产品是集成测试用例集，为集成测试的实施阶段做准备。

3）实施阶段

本阶段的工作是依据集成测试计划和已经设计的测试用例集完成测试执行前的准备工作，一般涉及以下活动：

- 集成测试环境的搭建。
- 测试工具的选择、工具的准备和调试。
- 测试驱动器或(和)测试桩的开发。
- 测试脚本的开发。
- 测试过程控制流的制定。

4）执行阶段

集成测试的执行阶段在执行过程中应该注意：

- 严格按照集成测试计划中制订的集成顺序执行。
- 单元测试完成并通过评审后才能执行集成测试。
- 严格按照规定的测试过程控制流管理执行过程。
- 严格记录各项测试执行结果，进行缺陷跟踪。
- 根据需要进行集成测试的回归。

5）分析评估阶段

当集成测试执行结束后，要召集相关人员，如测试经理、测试技术人员、相关编码人员、系统设计人员等根据测试计划中的集成测试退出等相关标准对测试结果进行评估，确定是否通过集成测试，产生集成测试分析和评估报告。

5.3 集成测试方法

5.3.1 一般的集成测试方法

集成测试分析为选择和确定集成测试策略提供了重要的参考依据，集成测试策略是建立在测试分析基础之上的。集成测试策略实际上就是指被测软件单元的集成方式、方法。集成测试的方法有很多种，每种方式有其自身的优点和缺点，因此要根据系统自身的实际特点来选择合适的集成测试方法，这就是策略。常见的集成测试方法有很多种，例如大爆炸集成、自顶

向下集成、自底向上集成、三明治集成、基于调用图的集成、基于路径的集成、分层集成、基于功能的集成、高频集成、基于进度的集成、基于风险的集成、基于事件的集成、基于使用的集成、客户端/服务器的集成、分布式集成等。而在实际的集成测试过程中,可以根据软件系统的体系结构和层次结构等特点同时采用不同的集成测试方法完成集成测试。下面将对主要的集成测试方法进行详细介绍。

1. 基于分解的集成

基于分解的集成测试可以分为非增量式和增量式两大类。非增量式集成测试也称作大爆炸(Big Bang)集成,就是分别对已经通过单元测试的模块按照层次结构图组装到一起进行测试,最终得到所要求的软件。增量式测试与非增量式测试相反,是一个逐步集成的过程。增量式集成(或组装)是首先对已经通过单元测试的模块逐步组装成较大的系统,在组装的过程中边组装边测试,以发现组装过程中产生的问题。增量式测试按不同的集成次序可分为两种方法,即自顶向下集成和自底向上集成。

1) 大爆炸集成

(1) 定义。

大爆炸集成属于非增值式集成(No-Incremental Integration)的方法,也称为一次性组装或整体拼装。这种集成测试的做法就是把所有通过单元测试的模块一次性集成到一起进行测试,不考虑组件之间的互相依赖性及可能存在的风险;目的是尽可能缩短测试时间,使用尽量少的测试用例来进行集成以验证系统。

(2) 方法举例。

图5-2所示是模块结构图,图中的A、B、C、D、E 5个模块均已经通过了单元测试,大爆炸集成就是将这5个模块一次性地集成在一起进行测试,找出可能出现的接口和其他类型的缺陷。

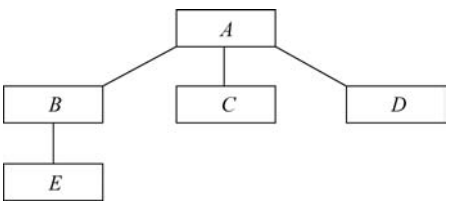


图5-2 模块结构图

(3) 优点。

- 可以同时集成所有模块,因此能够充分利用人力、物力资源,加快工作进度。
- 需要的测试用例数目少,因此测试用例设计的工作量相对比较小。
- 测试方法简单、易行。

(4) 缺点。

- 难以保证对各个模块之间的接口进行充分测试,因此很容易遗漏掉一些潜在的接口错误(如数据类型传递错误)。即使集成测试通过,也会遗漏很多错误(如接口错误等),从而增加系统测试的工作量,软件的可靠性难以得到很好的保证。
- 对全局数据结构的测试不够彻底。
- 一次集成的模块数量多,集成测试后可能会出现大量的错误,因此难以进行错误定位和修改。另外,修改了一处错误之后,很可能新增更多的新错误,新旧错误混杂,给程序的完善带来很大的麻烦。因此,往往要经过很多次集成测试才能够运行成功,集成回归。

(5) 适用范围。

- 集成时,仅仅修改或增加了少数几个模块且前期产品是稳定的。
- 功能少,模块数量不多,程序逻辑简单,并且每个组件都是已经过充分单元测试的情况。

- 基于严格的净室软件工程(开发零缺陷或接近零缺陷的软件方法)开发的产品,并且在每个开发阶段,产品质量和单元测试质量都相当高。

2) 自顶向下集成

(1) 定义。

自顶向下的集成测试就是按照系统层次结构图,以主程序模块为中心,采用自上而下地对各个模块一边组装一边进行测试。自顶向下可以分为用深度优先集成和广度优先集成两种方式,集成的过程和数据结构的深度遍历与广度遍历一致。深度优先集成是沿着系统层次结构图的纵向方向,按照一个主线路径自顶向下把所有模块逐渐集成到结构中进行测试,但是主线路径的选择是任意的,可以从左向右,也可以从右向左进行。广度优先集成是沿着系统层次结构图的横向方向,把每一层中所有直接隶属于上一层的所有模块逐渐集成起来进行测试,一直到最底层,可以从左向右,也可以从右向左进行。需要开发桩,开发的桩数量为结点数减 1。

(2) 集成方法。

过程如下:

① 以主模块为所测模块兼驱动模块,所有直属主模块的下属模块全部用桩模块对主模块进行测试。

② 采用深度遍历或广度遍历的策略,用实际源代码模块替换相应桩模块,再用桩代替它们的下属模块,与已经测试的模块或子系统集成为新的子系统,每次只替换一个桩为源代码。

③ 进行回归测试(即重新执行以前的全部测试或部分测试用例),排除集成过程中引起错误的可能。

④ 判断是否所有的模块都已经集成到系统中,若是则结束测试,否则转到②继续执行。

除了第③点所要求的回归测试之外,在软件体系结构中有增加模块、删除模块、修改模块等情况发生后的集成,需要回归测试,回归测试的测试用例是重新执行以前运行过的全部或部分测试用例,以确定集成加入新模块、删除模块或模块修改后是否引入错误或缺陷。

下面举例说明集成过程,该例子是采用自顶向下、从左向右的深度优先集成。图 5-3 的上方是模块结构图,下方是集成的过程。图 5-3 中 $S_1 \sim S_5$ 代表桩模块,在采用自顶向下、从左向右的深度优先集成过程中每次只将一个桩模块替换为源代码。

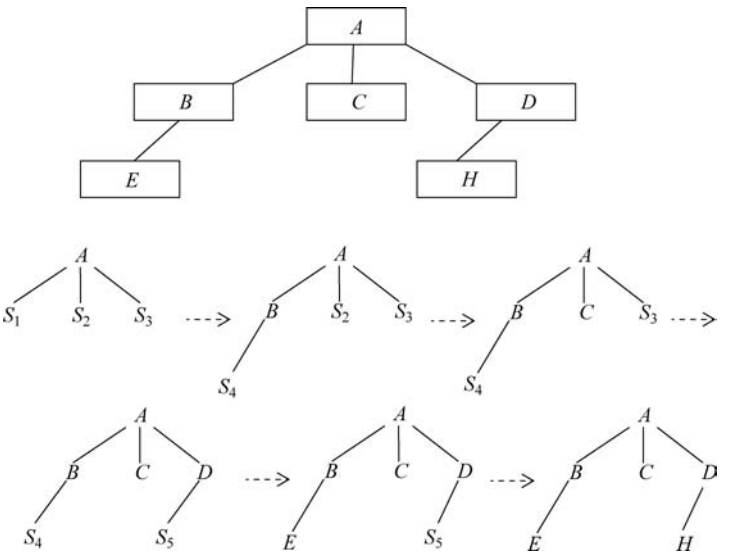


图 5-3 自顶向下的集成实例

(3) 优点。

- 在测试的过程中,可以较早地验证主要的控制和判断模块。在一个功能划分合理的程序模块结构中,控制和判断模块常常出现在较高的层次中,因而可以提前做测试,以便发现问题。
- 选择深度优先集成的方式,可以首先实现和验证一个完整的软件功能。如,采用从左向右的深度优先集成可以先对逻辑输入的分支进行集成,检查和克服潜在的错误和缺陷,验证其功能的正确性,为此后主要分支的集成提供保证。
- 不需要开发驱动器,减少开发和维护成本。
- 开发和集成测试的顺序是一致的,因此在开发的同时可以并行执行集成测试,能够灵活地适应目标环境。
- 故障隔离和错误定位容易。如果主程序模块 A 通过了测试,加入模块 B 后出现错误或缺陷,那么可以判断错误可能是出现在 B 模块或(和)A 模块或 B 模块和 A 模块的接口。

(4) 缺点。

- 测试时需要为每个模块的下层模块提供桩模块,桩数量为结点数减 1。增加桩模块的开发和维护成本。
- 底层尤其是叶子模块一般是实现功能的模块,其需求变更可能会影响到全局模块,可能需要修改整个系统的多个上层模块。因此,容易出现回归测试或多次回归。
- 在集成过程中,底层模块不断加入,整个系统变得越来越复杂,可能会导致底层模块特别是被重用的或被多个模块调用的模块测试不够充分。

(5) 适用范围。

这种集成测试方法适用于大部分采用结构化编程方法的软件产品。一般的大型复杂软件系统往往会综合使用多种不同的集成测试方法。采用自顶向下的集成测试方法可以参考以下几个方面:

- 软件的体系结构控制比较清晰。
- 软件的体系结构的高层模块接口变化少。
- 开发和集成测试并行的情况。
- 软件的体系结构的低层模块接口的最终定义较迟或经常因需求变更等原因被修改。
- 产品中的控制模块技术风险较大,需要尽可能提前验证。
- 需要尽早看到系统某些方面功能行为,如输入功能和输出功能等。
- 极限编程(Extreme Programming)中的测试优先的情况。

3) 自底向上集成

(1) 定义。

自底向上集成是从系统层次结构图(或称软件的体系结构图)的最底层模块开始进行组装的集成测试方式。对于某一个层次的特定模块,因为它的子模块(包括子模块的所有下属模块)已经组装并测试完成,所以不再开发桩模块。在集成测试过程中,如果想要从子模块得到信息可以通过直接调用子模块的源代码。集成测试的过程中需要开发相应模块的驱动器。

(2) 集成方法。自底向上的集成步骤如下:

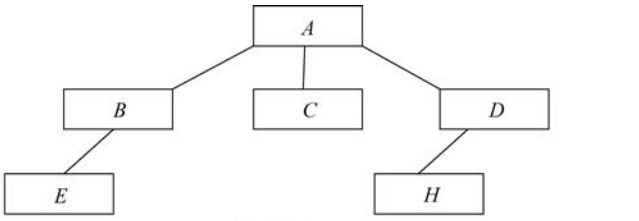
① 由驱动模块控制底层模块并进行并行测试,也可以把最底层模块组合成某一特定软件功能的组,由驱动器模块控制它并进行测试。

② 用实际模块代替驱动器，与它已经测试的直属子模块集成为子系统。

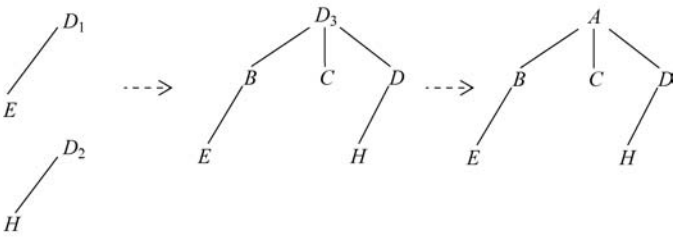
③ 按模块结构向上集成并为子系统配备新驱动模块，进行新的测试。

④ 判断是否已经集成到主模块，若是则结束测试，否则执行②。

下面举例说明集成过程。图 5-4(a)是模块结构图，图 5-4(b)是集成的过程，$D_1 \sim D_3$ 代表驱动器模块，在集成过程中，D_1 和 E 之间集成、D_2 和 H 之间集成可以并行。图 5-4(b)中开发驱动器的数为结点数-叶子数，即 6－3＝3，驱动器的数量比自顶向下集成开发的桩数量少，不过代价是驱动器模块都比较复杂。另一种集成过程如图 5-5 所示，$D_1 \sim D_5$ 为驱动器模块，这种集成过程开发的驱动器数量和自顶向下的方法一样，即结点数－1，这种集成过程增加了最后阶段的各分支和主模块的不同的驱动器间的集成。



(a) 模块结构图



(b) 集成过程

图 5-4　自底向上一种集成过程实例

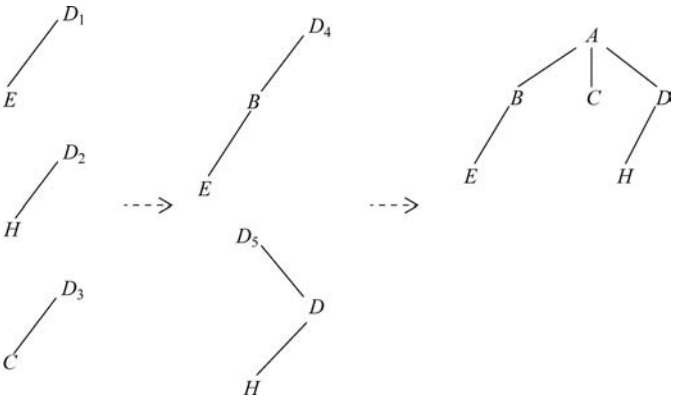


图 5-5　自底向上另一种集成过程

(3) 优点。

- 尽早地验证下层模块的行为。当任意一个叶子模块通过单元测试后，都可以随时对下层模块进行集成测试，并且驱动模块的开发还有利于规范和约束系统上层模块的设计，可在一定程度上增加系统的可测试性。

- 集成测试过程中,可以同时对系统层次结构图中不同的分支进行集成测试,具有并行性。
- 在对上层模块进行测试时,下层模块的行为就已经得到了验证,因此在向上集成的过程中,越靠近主控模块的上层模块更多的是验证其控制和逻辑。

(4) 缺点。

- 直到最后一个模块加进去之后才能看到整个系统的框架,才能发现时序问题和资源竞争问题。
- 驱动模块的开发相对复杂且工作量大。
- 主控模块的测试要到集成测试的最后才能进行,因此不能及时发现高层模块设计上的错误,如果主控模块的控制和逻辑对于软件体系结构非常关键,可能影响较大。

(5) 适用范围。

与自顶向下的集成方式类似,该方法适用于大部分结构相对比较简单、采用结构化编程方法的软件系统。采用自底向上的集成测试方法可以参考以下几个方面:

- 底层模块接口和行为比较稳定。
- 高层模块接口和行为变更比较频繁。
- 底层模块开发和单元测试工作完成较早。

以上讨论的 3 种集成测试的方法都属于基于功能分解的集成,下面讨论三明治集成测试方法。

2. 三明治集成

(1) 定义。

三明治集成是一种混合增量式集成测试方法,综合了自顶向下和自底向上两种集成方法的优点,也属于基于功能分解的集成。这种方法桩和驱动器的开发工作都比较少,不过代价是类似大爆炸集成的后果,在一定程度上增加了定位缺陷的难度。

(2) 集成方法。

分析如下:

① 对整个的模块层次结构图(软件体系结构图)而言,首先必须确定以哪一层为界来决定使用三明治集成方法,一般以模块层次结构图的中间层或接近于中间的层为界。

② 以确定为界的层及其以下的各层使用自底向上的集成方法。

③ 以确定为界的层的上面的层次使用自顶向下的集成方法,不包括确定为界的层。

④ 对系统所有模块进行整体集成测试。

三明治集成方法应该尽量减少设计驱动模块和桩模块的数量。在集成测试过程中,使用以确定为界的层各模块同相应的下层先集成的策略,而不是使用确定为界的层各模块同相应的上层先集成,是考虑到这样做可以减少桩模块的开发。

下面举例说明三明治集成方法的集成过程。图 5-6 是软件的模块结构图。图 5-7 和图 5-8 是集成步骤。

① 确定以 E 模块所在层为界。

② 以 E 模块为界及其所在层自底向上集成(见图 5-7)。

③ 以 E 模块为界的上面的层次的自顶向下的集成(见图 5-8)。

④ 对系统所有模块进行整体集成测试(见图 5-6)。

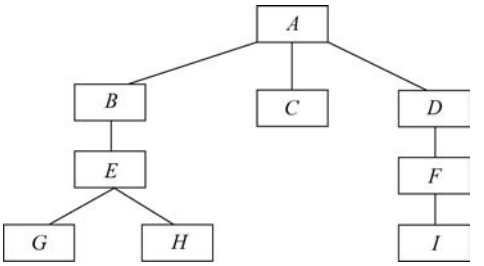


图 5-6　软件的模块结构图

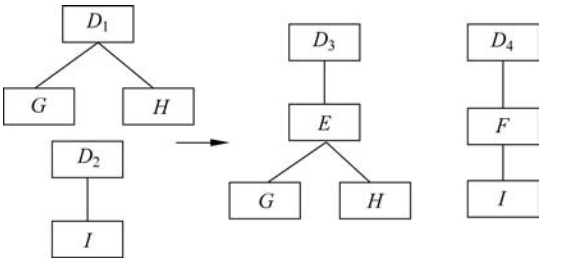


图 5-7　E 模块为界及其所在层自底向上集成

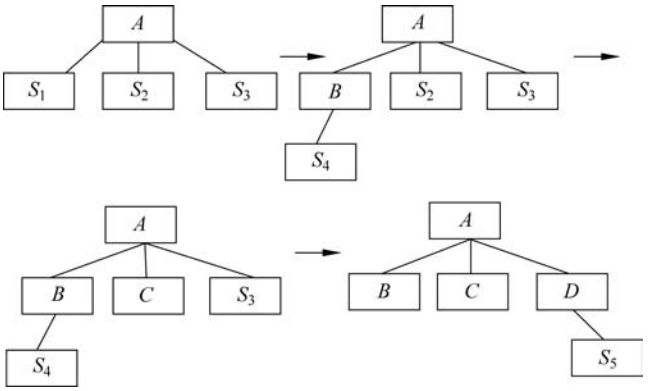


图 5-8　E 模块为界的上面的层次的自顶向下集成

(3) 优点。

① 具有自顶向下和自底向上两种集成方法的优点。

② 确定哪一层为界具有一定的技巧,可以减少桩模块和驱动模块的开发量。

③ 自顶向下和自底向上两种集成方法可以并行。

(4) 缺点。

在最后的所有模块集成阶段会增加缺陷定位的难度。

(5) 适用范围。

大多数软件系统都可以应用此集成测试方法。

3. 基于调用图的集成

基于分解的集成方法的缺点就是以系统功能分解为基础,以模块结构图为依据。如果把集成的依据改为模块调用图,则可以使集成测试向结构性测试方法发展,避免基于分解的集成方法存在的一些不足。在第 1 章介绍过图论知识。模块调用图是一种有向图,结点表示程序模块,边对应程序调用。如果模块 X 调用模块 Y,则从模块 X 到模块 Y 有一条有向边。基于调用图的集成方法有两种:成对集成和相邻集成。下面我们对这两种方式进行简单的介绍。

1) 成对集成

成对集成的基本思想就是免除桩/驱动器开发工作,使用实际代码来代替桩/驱动器。这看起来类似大爆炸集成方式,但是把这种集成限制在调用图中的一对模块或单元上。

成对集成的方法就是对应调用图的每一个边建立并执行一个集成测试对,这个测试主要关注这条边对应的接口。对整个软件系统而言,需要建立多个集成测试对且存在着一个模块分别和不同的模块建立不同的测试对,但是成对集成可以大大减少桩和驱动器开发的工作量。

图 5-9 为某系统的调用图,以此为例加以分析。该调用图共有 22 个集成测试对,其中的模块 a 分别和 x、h、i、v、l、b 建立集成测试对。图中的虚线包围列出了 3 个集成测试对。

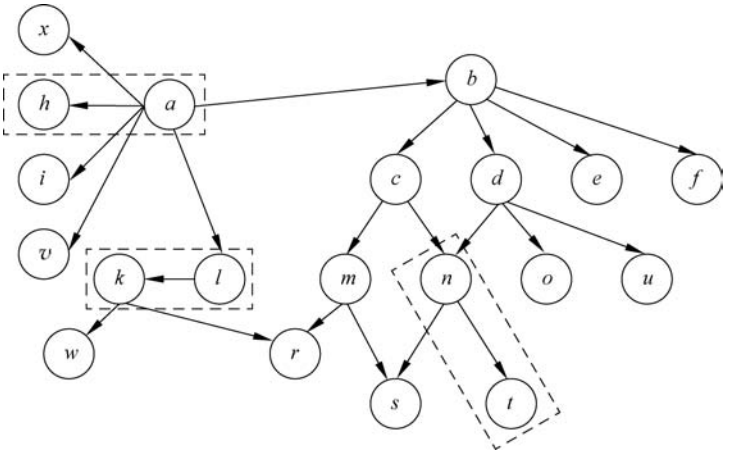


图 5-9 成对集成举例

2) 相邻集成

相邻集成中的相邻是针对模块结点而言的,模块结点的邻居就是由指定模块结点引出的结点集合。在有向图中,结点邻居包括所有直接的前驱结点和所有直接的后继结点,如图 5-10 中,模块结点 c 的直接前驱结点为 b,直接后继结点为 m;模块结点 a 没有直接前驱结点,其直接后继结点有 x、h、i、v、l。在图 5-10 中,对于模块结点 l 来说,它的邻居有 a、k 两个模块结点;对于模块结点 k 来说,它的邻居有 l、w 和 r 3 个模块结点;对于模块结点 b 来说,它的邻居有 a、c、d、e、f 5 个模块结点。图 5-10 的虚线范围给出了模块结点 l 和 d 的邻居。

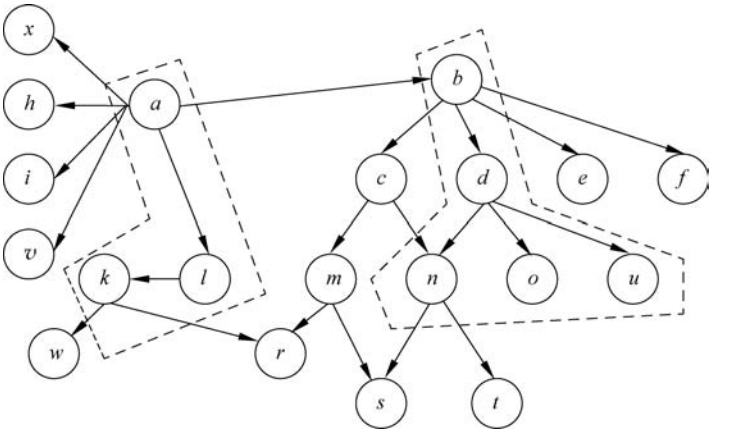


图 5-10 相邻集成举例

对于给定调用图,我们总是可以计算出邻居数量。每个内部结点(内部结点具有非零内度和非零外度)都有一个邻居,如果叶结点直接连接到根结点,则还要加上一个邻居。这样有:

内部结点=结点-(源结点+汇结点)

邻居=内部结点+源结点

经过合并,得到:

邻居=结点-汇结点

源结点和汇结点在“4.基于路径的集成”部分有详细解释。

根据上面的公式，调用图 5-10 的邻居数量为：20－12＝8。所以相邻集成和成对集成相比可大大降低集成对数。从例子中可以看出其集成对数为 22 对，而邻居数只有 8 个，并且避免了桩和驱动器的开发。

回忆一下三明治集成，可以看出相邻集成本质上是前面介绍过的三明治集成，其不同之处是相邻集成的依据是调用图，而不是分解树，它们的共同之处是都具有缺陷隔离的困难。另外，对于同时属于不同邻居的结点存在缺陷的时候，修改该结点的缺陷后，所有包括该结点的邻居都需要重新进行集成测试，也就是说要进行集成的回归。例如图 5-10 中的模块结点 n 就是这种情况。

4. 基于路径的集成

在 3.2 节中描述了程序图的概念，在程序图中语句片段作为完整语句处理，语句片段是程序图中的结点。在分析基于路径的集成方法之前，先了解与程序图的结点、路径等相关概念。

1）源结点

程序中的源结点是程序执行开始或重新开始执行处的语句片段。模块或单元中的第一个可执行语句显然是源结点。源结点还会出现在紧接转移控制到其他模块或单元的结点之后。在图 5-11 的模块 A 中的结点 1、6 是源结点；模块 B 中的 1、3 结点是源结点；模块 C 中的 1 结点是源结点。

2）汇结点

汇结点是程序执行结束处的语句片段。程序中的最后一个可执行语句显然是汇结点，转移控制到其他单元的结点也是汇结点。在图 5-11 的模块 A 中的结点 5、7 是汇结点；模块 B 中的 2、4 结点是汇结点；模块 C 中的 6 结点是汇结点。

3）模块执行路径

模块执行路径（Module Execution Path，MEP）是指模块内部以源结点开始、以汇结点结束的一系列语句，中间没有插入汇结点。依据图 5-11 中模块的情况，其所有的模块执行路径为：

MEP(A,1)＝<1,2,3,4,7>

MEP(A,2)＝<1,2,3,5>

MEP(A,3)＝<6,7>

MEP(A,4)＝<1,2,3,5,6,7>

MEP(B,1)＝<1,2>

MEP(B,2)＝<3,4>

MEP(B,3)＝<1,2,3,4>

MEP(C,1)＝<1,4,5,6>

MEP(C,2)＝<1,2,5,6>

MEP(C,3)＝<1,3,5,6>

4）消息

消息是一种程序设计语言机制，通过这种机制可以把控制从一个单元转移到另一个单元。在不同的程序设计语言中，消息可以被解释为子例程调用、过程调用、方法调用及函数引用。约定接收消息的单元总是最终将控制返回给消息源。消息可以向其他单元传递数据。

5）MM 路径

MM 路径（Method Message Path，MM-Path）是指穿插出现的模块执行路径和消息构成

的序列。如图 5-11 中的粗线所示,代表模块 A 调用模块 B,模块 B 调用模块 C,这就是一个 MM 路径。

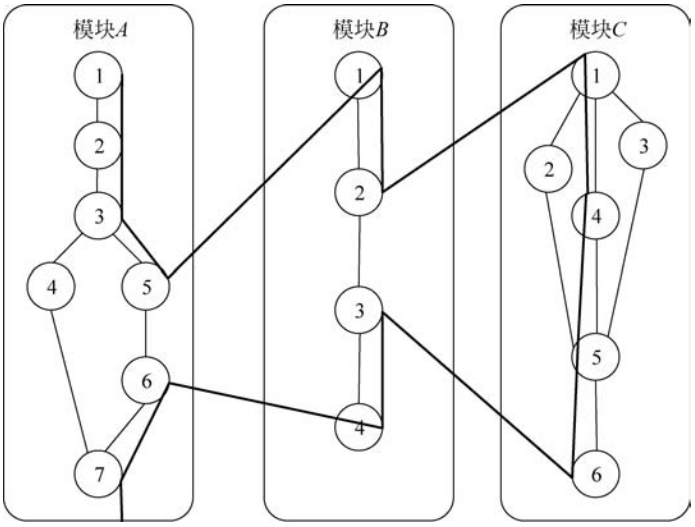


图 5-11　MM 路径举例

如果模块 A、模块 B、模块 C 之间的调用关系按照图 5-11 所示,那么我们可以将模块 A、模块 B、模块 C 之间所有的 MM 路径表示出来,以构成 MM 路径图。该 MM 路径图也包括模块内部退化的 MM 路径(没有调用关系的 MEP),如图 5-12 所示。在图 5-12 中实线表示消息,虚线表示消息的返回。图中除了反映模块 A、模块 B、模块 C 之间的调用关系的一条 MM 路径之外,还包括 MEP(A,1)、MEP(A,4)、MEP(B,3)、MEP(C,2)和 MEP(C,3)5 条退化的 MM 路径。

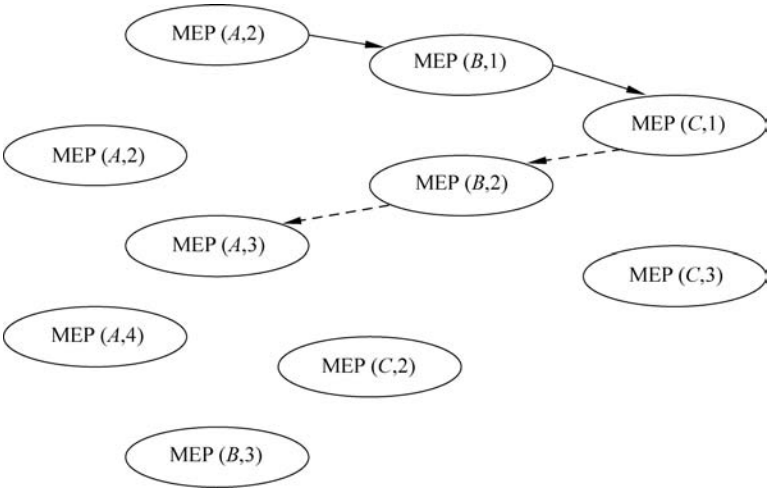


图 5-12　从图 5-11 中导出的 MM 路径图

进行路径集成测试时,选择的 MM 路径集合的最低覆盖指标是覆盖单元集合中所有从源结点到汇结点的路径、MM 路径集合覆盖所有的结点或者 MM 路径集合覆盖所有消息调用和返回的边。在上面的例子中,要设计一组测试用例使图 5-12 中的所有 MM 路径至少覆盖一次,包括退化的 MM 路径。如果存在循环,则要进行压缩,产生有向无环路图,因此可解决无

限多路径问题。

MM 路径是功能测试和结构性测试的一种结合。从测试本身来说，MM 路径是功能性的，因此可以使用所有功能性测试技术。而在测试用例的设计上使用了白盒测试中的路径思想，不过这里的路径不是模块内部的路径而是跨模块的路径，而在 MM 路径图的标识方式上也是结构性的。因此，在基于路径的集成测试过程中，很好地把功能测试和结构测试的方法结合到了一起。但是，基于路径集成的测试的关键是标识 MM 路径，再基于路径设计对应的测试用例。

5. 其他集成测试方法

1）分层集成

分层模型在通信系统中十分普遍。分层集成就是针对分层模型使用的一种集成方法。系统的层次划分可以通过逻辑的或物理的两种不同方式进行。从逻辑角度，一般通过功能把系统划分成不同功能层次的子系统，子系统内部具有较高的耦合性，子系统间的关系具有线性层次关系；从物理角度，可以根据不同单板内的系统划分为不同的硬件子系统，各硬件子系统之间根据连接具有线性层次关系。而对于那些各层次之间存在着拓扑网络关系的系统，则不适合使用该集成方法。

该方法首先划分系统的层次，确定每个层次内部的集成方法。层次内部的集成可以使用大爆炸集成、自顶向下集成、自底向上集成和三明治集成中的任何一种。一般对于顶层可能还有第二层的内部采用自顶向下的集成方法；对于中间层采用自底向上的集成方法。最后，确定层次间的集成方法，也可以使用大爆炸集成、自顶向下集成、自底向上集成和三明治集成中的任何一种方法。

该方法对具有明显线性层次关系的系统比较适合。

2）基于功能的集成

该方法是从功能实现的角度出发，按照模块的功能重要程度作为模块的集成顺序。先对最主要的功能模块进行集成测试，以此类推，最后完成整个系统的集成测试。很明显这样的方法是采用了增量的方法。

该方法首先确定功能的优先级别，分析优先级最高的功能路径，把该路径上的所有模块都集成到一起，必要时需要开发驱动模块和桩模块。在集成过程中每次增加该路径中的一个关键功能，直至该路径上的模块集成结束。再根据优先级别继续集成其他路径，直到所有模块都被集成到被测系统中。

该方法能较早地实现系统中的关键功能，但是不适用于复杂系统，因为复杂系统的功能之间的相互关联性强，不易于分析主要模块。

3）高频集成

快速迭代式开发或增量式开发存在不足之处，即一些错误或缺陷在开始的时候只是存在于功能模块或集成包内，影响不大，但随着产品开发过程中的迭代增加或增量的增加，这些错误或缺陷可能会影响到新加入的模块功能，甚至影响到系统的稳定性。这就需要在迭代或增量过程中不断地进行集成测试验证系统功能的正确性和稳定性。如果这些错误或缺陷遗留到最后再检查，其代价和风险可能很大，高频集成就是基于这种考虑而进行的测试。

高频集成具备的基本条件是该次增量结束或该次迭代开发结束，可以通过使用配置管理工具帮助获得每次增量结束或该次迭代开发结束后的程序版本。另外，高频集成可以采用必要的自动化测试工具来支持。该集成测试方法频繁地将新代码加入一个已经稳定的基线中，

避免集成错误或缺陷不被发现,同时控制可能出现的基线偏差。

高频集成可以参考以下3个步骤:

(1) 增量结束或该次迭代开发结束,从配置管理库中得到代码的增量部分,测试人员完成编写或修改对相应代码的测试包;对新增或修改过的代码进行静态测试(可能包括代码走读、检视、评审和静态分析);对代码进行重新创建并运行测试包(可能包括使用类似内存检测工具、性能检测工具进行跟踪检查);当这些组件通过测试时,将已修改过的测试包提交到集成测试部门。

(2) 集成测试人员将修改或增加的组件和配置管理库的基线上的其他组件集中形成一个新的集成体,运行测试包进行集成测试(可能需要测试工具的支持),该次集成测试结束后形成一个新的基线。

(3) 该次集成测试结束后需对测试结果进行评价。主要涉及现有的集成测试包是否按要求进行维护、测试的频率间隔是否合理、测试的必要条件是否具备,如是否增量结束等。如果该次集成测试失败,则系统将退回到原来的基线。

对于高频集成方法,其维护测试包很重要。该方法的有利因素包括:开发和集成可以并行进行;对桩代码的需要不是必需的;错误或缺陷最可能存在于新增加或修改的代码中,容易进行错误或缺陷的定位和修改等。

以上介绍了一些主要的集成测试方法,除了这些集成测试方法以外,还有基于进度的集成、基于风险的集成、基于事件的集成、基于使用的集成等,在这里不再一一详述。

5.3.2 面向对象的集成测试方法

集成测试的主要目的是检查两个或两个以上的模块或对象的接口数据是否正确。传统的集成测试,是通过各种集成策略集成各功能模块,一般可以在部分程序编译完成的情况下进行。而对于面向对象程序,相互调用的功能是散布在程序的不同类中,类通过消息相互作用申请和提供服务。类的行为与它的状态密切相关,状态不仅仅是体现在类数据成员的值,也许还包括其他类中的状态信息。由此可见,类的相互依赖极其紧密,根本无法在编译不完全的程序上对类进行测试。因此,面向对象的集成测试通常需要在整个程序编译完成后进行。此外,面向对象程序具有动态特性,程序的控制流往往无法确定,因此一般也只能对整个编译后的程序做基于黑盒子的集成测试。

在集成测试中,一个关键的问题是怎样进行集成。好的集成方法可以避免写大量的测试驱动器和桩,节约各种资源。本节不涉及讨论类内方法的集成。

面向对象的集成在不同的阶段可分成3个不同的集成类型:

- 类的集成:将有关的类集成为一个子系统或组件。
- 子系统或组件的集成:将子系统集成为一个应用层。
- 层的集成:不同的层的集成,可以是描述层、工作层、存取层或客户端/服务器(Client/Server)层的集成。

图5-13所示为一种架构的集成示意图。

在面向对象的集成测试过程中要考虑到面向对象系统是由事件操纵的特点,在集成测试时一般推荐考虑如下5个层次:

- 类的方法测试(Method Testing);
- 消息序列(Message Sequences);

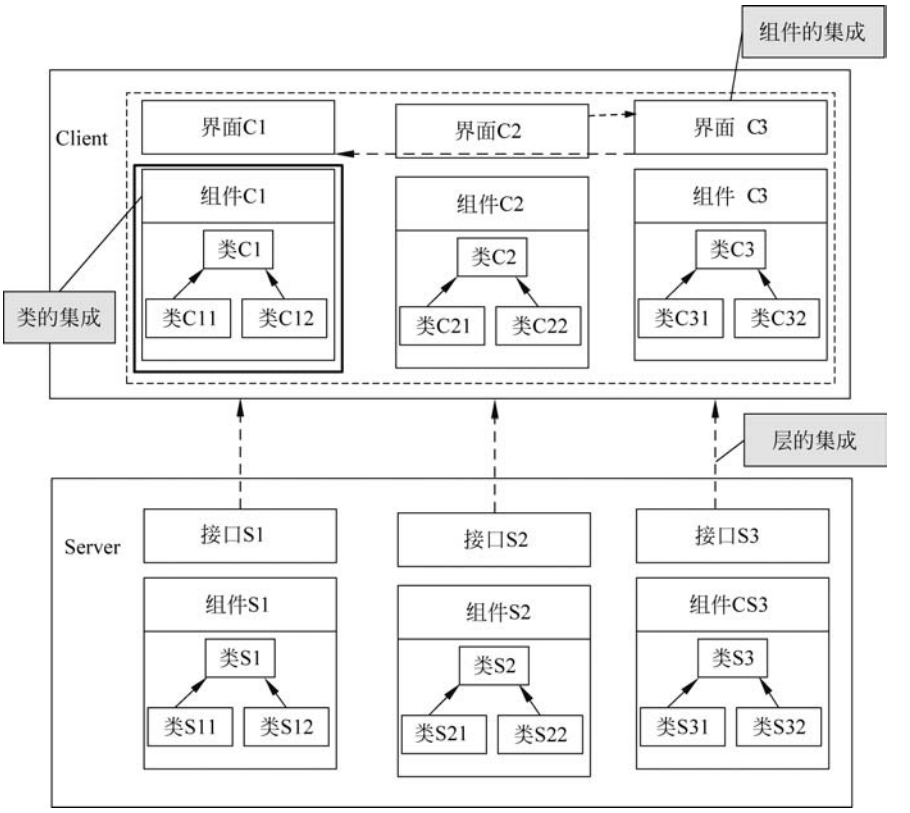


图 5-13　一种架构的集成示意

- 事件序列(Event Sequences)；
- 线程测试(Thread Testing)；
- 交互测试(Thread Interaction Testing)。

在线程测试中主要是应用了方法/消息路径(Method Message Paths，MM Paths 或 MM 路径)。所谓方法/消息路径就是一个运行的方法(Method)通过消息(Message)的传递调用另一个方法，最后到不能再产生消息的方法为止，即处于一种相对静止状态。在一个调用中可能会存在多个方法/消息路径，在线程测试中就是要覆盖这些方法/消息路径。

事件控制的测试过程中引入了原子系统功能(Atomic System Function，ASF)的概念，ASF 是由外部的事件(Input Port Event，输入端口事件)激活系统，导致系统有所反应(Output Port Event，输出端口事件)，这样的一个过程称为原子系统功能。在面向对象的集成测试中还应覆盖子系统内的原子系统功能。

图 5-14 所示为 MM-Paths/ASF 示意图。

面向对象的集成测试能够检测出相对独立的单元测试无法检测出的那些类相互作用时才会产生的错误或缺陷。基于单元测试对成员函数行为正确性的保证，集成测试只关注于系统的结构和内部的相互作用。

UML 中的协作图是基于时间先后顺序表达对象的交互，而序列图则是通过对象的交互顺序来表达同样事件或操作，这些也是集成测试的很好依据。

集成测试所要达到的覆盖指标可以是：

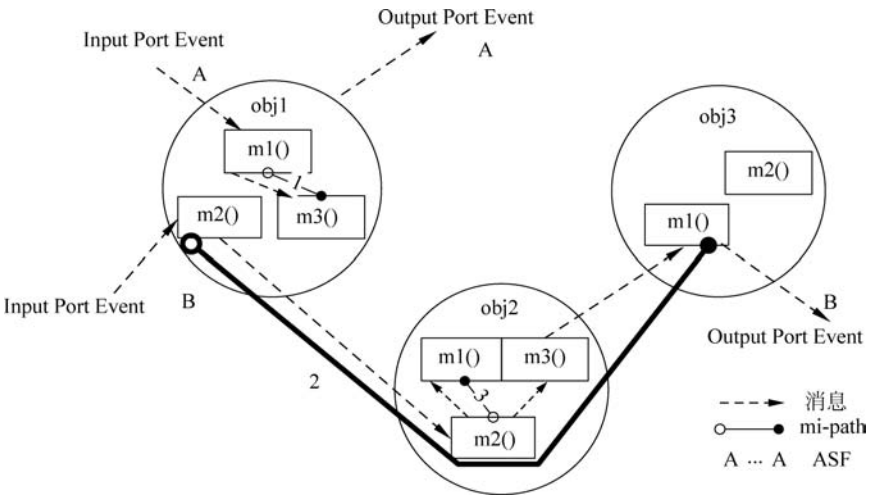


图 5-14 MM-Paths /ASF 示意图

- 所有类的所有方法至少覆盖一次;
- 协助图(合作图)或序列图中的所有消息至少覆盖一次;
- 依据类间传递的消息,达到对所有执行线程至少覆盖一次;
- 子系统内所有 ASF 至少覆盖一次;
- 所有类的所有状态至少覆盖一次;
- 对象之间的调用关系图中的边(消息)或结点(方法)至少覆盖一次。

同时也可以考虑使用现有的一些测试工具来得到集成测试接口(消息)的覆盖率。

在进行具体的分析和用例设计时,可参考下列步骤:

(1) 先选定检测的类,参考面向对象设计分析结果,以及协助图、顺序图、状态图等,仔细分析出类的状态和相应的行为、类或成员函数间传递的消息、输入或输出的界定等。

(2) 确定覆盖指标。

(3) 利用类图(E-R 图)确定待测类的所有关联。

(4) 根据程序中类的对象构造测试用例,确认使用什么输入激发类的状态、使用类的服务和期望产生什么行为等。

值得注意的是,设计测试用例时,不但要设计类功能满足的输入,还应该有意识地设计一些异常输入的例子、类是否有不合法的行为产生,如发送与类状态不相适应的消息、要求有不相适应的服务等。

所设计的测试用例必须进行评审。

5.4 集成测试的分析和用例设计

1. 集成测试分析

集成测试的分析和用例设计是集成测试过程中最为核心的阶段。集成测试的分析对用例的设计和整个集成测试过程具有重要的指导作用,在集成测试的分析过程中,主要涉及进行体系结构分析、类或模块及接口分析、集成测试方法选择策略分析,有时也会涉及可测试性分析、测试风险分析等。

1）体系结构分析

本章前面已经提及，体系结构是进行集成测试的重要依据。

对于使用传统结构化方法开发的系统，软件的体系结构是概要设计的重要组成部分，软件的体系结构就是模块的层次结构，它是集成测试的依据。在进行集成测试分析时，不但要考虑软件的体系结构，还要考虑整个系统的体系结构，这个结构可能包含软件、网络、硬件等系统要素。体系结构分析主要关注：

（1）分析软件体系结构与需求分析规约及需求的一致性。

（2）分析软件体系结构的合理性。如，体系结构的层次、调用深度和关系等，必要时建议开发人员调整体系结构。

（3）在软件体系结构中一般没有人机交互界面对应的模块，这里必须加以考虑。

（4）分析整个系统的体系结构，除软件体系结构外，还涉及哪些系统要素及其组织方式。

对于使用面向对象开发的系统，软件的体系结构可以用组件图来表示，整个系统的体系结构可以通过配置图来体现，也会涉及硬件、网络等系统要素，在集成测试分析时也必须考虑。图 5-15 为图书管理系统的图书信息管理子系统的组件图，各含义如下：

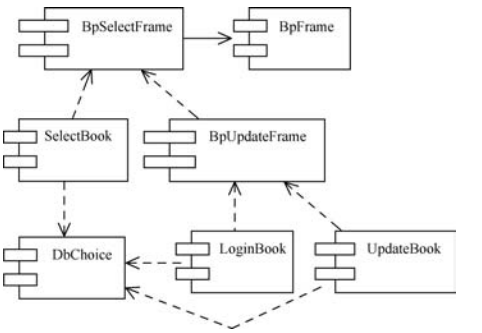


图 5-15　图书信息管理子系统组件图

- BpFrame 组件：属于用户界面包，定义系统检索与修改界面的框架。
- BpSelectFrame 组件：属于用户界面包，继承 BpFrame 类，定义检索界面框架。
- BpUpdateFrame 组件：属于用户界面包，继承 BpSelectFrame 类，定义系统修改界面框架。
- SelectBook 组件：属于用户界面包，继承 BpSelectFrame 类，与 DbChoice 类相关联，显示图书信息检索界面。
- LoginBook 组件：属于业务模型包，继承 BpUpdateFrame 类，与 DbChoice 类相关联，实现图书信息录入功能。
- UpdateBook 组件：属于业务模型包，继承 BpUpdateFrame 类，与 DbChoice 类相关联，实现图书信息修改功能。
- DbChoice 组件：属于组件包，定义了用于数据库操作的实例变量和实例方法。

从图 5-15 我们可以看出该子系统主要由几类组件组成：界面组件、业务组件、访问数据库组件，有时可以包含通信组件、数传递组件等。对系统各个组件之间的依赖关系进行分析，然后据此确定集成测试的粒度，即集成模块的大小，在此例中可以以一个组件作为一个模块，其中的界面组件负责与用户的交互，业务组件负责处理业务逻辑，访问数据库组件提供与数据库的接口。其中，业务组件是通过 3 个主要的类（SelectBook、LoginBook、UpdateBook）来实现，这 3 个类与 DbChoice 具有依赖关系。

在集成测试过程中，如果单元测试是以类为单元且类内方法均测试并达到了类内集成，在集成测试时组件可以采用以类为单元，直接进行类间集成测试。相反，如果单元测试以方法为单元且类内方法没达到类内集成，组件应该分别以方法和类为单元进行测试，然后再进行类内集成和类间集成测试。

对于 App 项目或微服务架构的项目，存在前后端或前后端分析的情况，这种架构的集成

测试在第9章和第10中有详细分析。

2) 类或模块及接口分析

模块分析及接口分析是在体系结构分析工作基础之上的细化,它在集成测试的过程中,也是一个非常重要的环节,直接影响集成测试的工作量、进度以及质量,因此也需要认真对待。对于模块分析,要依据单元测试的结果分析单元划分的合理性、单元优先级及关键模块,另外,对在单元测试时开发的驱动器和桩进行分析,以便在集成测试时修改后复用(可从配置管理库中得到单元测试的驱动器和桩代码)。

我们已经知道集成测试的主要内容是对接口的测试,这就要求对接口进行周密细致的分析,对接口进行分类,分析并找出通过接口传递的数据类型、数据的个数等。接口的划分以概要设计为基础,一般需要考虑:

(1) 分析确定系统的边界、子系统的边界和模块的边界。

(2) 分析确定模块间的接口,包括一个模块和其他多个模块的接口。

(3) 分析确定子系统内模块间的接口。

(4) 分析确定子系统间的接口。

(5) 分析确定系统与操作系统的接口。

(6) 分析确定系统与硬件的接口。

(7) 分析确定系统与第三方软件的接口。

(8) 人机交互接口。

在各种软件的开发过程中,我们会接触到各种各样的接口,可以把这些接口大致划分为系统内接口(系统内部各模块交互的接口)和系统外接口(外部系统,如人、硬件和软件等与系统交互的接口)两类,其中前者是集成测试的重点,并且可以把它进一步划分为以下几类接口:

- 函数或方法接口:通过分析函数或方法的调用和被调用关系来确定。
- 消息接口:这类接口主要应用在面向对象系统和嵌入式系统中。消息接口的特点是软件模块间并不直接发生关系,而是按照接口协议进行通信。
- 类接口:面向对象系统中最基本的接口。该接口往往都要通过继承、参数类、对于不相同类方法调用等策略来实现。
- 其他接口:其他类型接口包括全局变量、配置表、注册信息、中断等。这类接口具有一定的隐蔽性,往往测试人员会忽略这部分接口。这类接口经常是测试不充分的。我们在对这类接口进行测试时可以借助专门的自动化工具。

接口数据分析就是对通过接口进行传递的数据进行分析,针对不同类型的接口,要采取不同的分析方法。在分析的过程中可以直接设计出相应的测试用例。

(1) 函数接口分析。我们要关注其参数个数、参数属性(参数的数据类型、是输入还是输出)、参数前后顺序、参数的等价类情况、参数的边界值情况,必要的时候还要对各种组合情况加以分析。

(2) 消息接口分析。主要分析消息的类型、消息域、域的顺序、域的属性、域的取值范围、可能的异常值等。必要的时候也要对其组合情况加以分析。

(3) 类接口和交互方式分析。在面向对象应用程序中,很多类都要同其他类进行交互。因此,在这类应用程序中类交互的测试就成为集成测试的重点,对类接口和交互方式进行详细分析就成为集成测试的重中之重。类接口和交互方式大致可以分为如下几类:

- 公共操作将一个或多个类命名为正式参数的类型。

- 公共操作将一个或多个类的命名作为返回值的类型。
- 类的方法创建了另一个类的实例，将其作为在它的实现中不可缺少的一部分。
- 类的方法引用某个类的全局实例(好的设计人员会尽量减少全局变量的使用)。

(4) 对于其他类接口的分析，重点分析其读写属性、并发性、等价类和边界值等。

总之，接口分析涉及的内容很多，测试人员在工作中还要根据项目自身的特点，在参照上述指导性原则的基础上多和开发人员交流，尽量能够对应用程序进行全面的接口分析，以便更好地进行测试用例的设计，因为接口分析的好坏在很大程度上影响着集成测试工作质量的高低。

集成测试的分析还会涉及集成测试方法的选择策略及其他方面的分析。系统整体结构的分析，尤其是软件体系结构的分析为集成测试方法的选择打下基础，如，软件的体系结构比较清晰和稳定、高层模块接口变化的可能性小、底层模块定义比较晚且变化的可能性大，则可以选择自顶向下的集成方法。当然，也可以采用不同的集成方法的综合。集成测试本身会存在风险，尤其对于比较大的系统而言，风险分析很有必要，以便根据风险分析的曝光度确定集成测试的过程和环节。集成测试常见的分析包括技术风险、人员风险、测试环境风险(硬件、软件、工具等)、管理风险、市场风险、时间风险、资金风险等。

2. 用例的设计

集成测试的分析结果是设计用例的重要基础，也就是说集成测试的用例设计要综合考虑使用的测试方法、系统接口特点、覆盖要求甚至测试时间等多方面。对于测试方法而言，无论是哪一个级别的测试，都离不开基本的测试用例设计方法(白盒测试用例设计方法和黑盒测试用例设计方法)，在集成测试时一般也需要灵活交叉地使用这些方法，以达到集成测试的目的，如满足相应的测试覆盖率要求，即达到集成测试要求的功能覆盖率和接口覆盖率。集成测试的用例设计可以考虑从以下几个方面入手。在以下几个方面的分析中以 ATM 系统为例。假设在 ATM 系统中有以下模块：与 ATM 机硬件接口模块、检查卡有效性模块、密码验证模块、选择交易类型模块、存款处理模块、取款处理模块、查询处理模块、显示模块、打印收据模块等。

1) 为系统运行设计用例

集成测试关注的主要内容就是各个模块的接口是否能正常使用，接口的正确与否直接关系到后续集成测试能否顺利进行。因此，首先要设计一组测试用例保证系统能运行起来，也就是验证实现基本功能的测试用例。认识到这一点，就可以根据测试目标来设计相应的测试用例。

可考虑使用的主要测试技术有：

(1) 等价类划分。

(2) 边界值分析。

(3) 基于决策表的测试。

利用等价类的思想对 ATM 系统的“存款”“取款”“查询”进行等价类划分，在“取款”功能里对“100 元”“500 元”“1000 元”“2000 元”不同取款额进行等价类划分来设计满足基本功的测试用例。再如，在插入卡后，检查密码的时候可以利用边界值分析法的思想设计密码位数不足或密码位数过长的输入密码情况的测试用例等，目的是实现基本的 ATM 机“存款”“取款”“查询”业务。

总之，应该将基本功能、基本业务逻辑和测试技术结合起来设计一组能使系统运行起来的测试用例。

2) 为正向测试设计用例

在软件各个模块的接口设计和模块功能设计完全正确无误并且满足需求的情况下，正向

测试用例设计的主要目标是在基本功能实现的基础上,能否实现所有预期的功能。如在ATM系统中,取款1000元是成功的,那么取款10 000元能否成功;收据的打印卡号的后续部分是隐藏的;存款成功后,能否进行多次续存等。基于此测试目标,可以直接根据概要设计文档导出相关的测试用例。可以通过以下方面来考虑并设计测试用例:

(1) 输入域测试。如输入不同的ATM取款金额、不同的存款金额等。

(2) 输出域测试。如不同取款金额的输出、收据的输出、显示屏的输出等。

(3) 等价类划分。如把所有正常的ATM交易的屏幕的每个输出看成一个等价类,达到所有等价类的覆盖等。

(4) 状态转换测试。如,读卡状态→输入密码状态→选择交易类型状态→正在取款状态→打印收据状态→交易结束。

(5) 规约导出法。如,根据概要设计规约中的功能描述导出测试用例。

3) 为逆向测试设计用例

集成测试中的逆向测试就是设计测试用例来实现需求规格没有描述的功能或各种异常的情况来检查可能出现的接口遗漏,或者判断接口定义是否有错误以及可能出现的接口异常错误,包括接口数据本身的错误、接口数据顺序错误等。在接口数据量庞大的情况下,如果要对所有异常的情况,以及异常情况的组合进行测试是很难的,因此,在这样的情况下就可以基于一定的约束条件(如根据风险等级的大小、排除不可能的组合情况)进行测试。

基于面向对象应用程序和GUI程序进行测试有时还需要考虑可能出现的状态异常,包括是否遗漏或出现了不正确的状态转换、是否遗漏了有用的消息、是否会出现不可预测的行为、是否有非法的状态转换,如从一个页面可以非法进入某些只有登录以后或经过身份验证才可以访问的页面等。

可从以下方面考虑并设计测试用例:

(1) 错误猜测法。如,在做ATM交易时,当ATM断电时,猜测ATM硬件接口模块有问题。

(2) 基于风险的测试。如,取款交易涉及资金风险,如果取550元,是否会出现钱没取出,账户已经扣款的情况。

(3) 基于故障的测试。如,ATM卡槽故障,是否能读卡。

(4) 边界值分析。如,取款超出每天的最大限额为20 000元的情况。

(5) 特殊值测试。如,取款额为0或负值的情况。

(6) 状态转换测试。如,从正在取款状态强制到交易结束状态。

4) 为满足特殊需求设计用例

在早期的软件测试过程中,安全性测试、性能测试、可靠性测试等非功能性测试主要在系统测试阶段才开始进行。由于计算机系统越来越复杂、庞大,在现在的软件测试过程中,需要不断地对这些满足特殊需求的测试过程加以细化。在大部分软件产品的开发过程中,模块设计文档就已经明确地指出了接口要达到的安全性指标、性能指标等,如,在ATM系统中密码验证模块需要有对密码的加密功能,而密码在通信传输时又需要对其动态加密,这就涉及了模块级和集成级的安全性测试。此时我们应该在对模块进行单元测试和集成测试阶段就开展满足特殊需求的测试,为整个系统是否能够满足这些特殊需求把关。

规约(可能涉及需求规约、概要设计规约、详细设计规约)导出法是合理的用例设计方法。

5) 为满足覆盖指标设计用例

与单元测试所关注的覆盖重点不同,在集成测试阶段关注的主要覆盖是功能覆盖和接口

覆盖(而不是单元测试所关注的路径覆盖、条件覆盖等),通过对集成后的模块进行分析,来判断是否所有的功能以及接口(如对消息的测试,既应该覆盖到正常消息也应该覆盖到异常消息)被覆盖。接口的覆盖可以通过模块的调用图帮助分析,也可以借助工具协助。如果现有的测试用例执行后,没有实现对所有功能和接口的覆盖,则需要补充设计测试用例。

可考虑从以下方面设计用例:

(1) 功能覆盖分析。如,ATM 系统中取款功能的所有子功能覆盖,子功能包括正常金额取款功能、非正常金额取款功能、非正常金额的异常取款功能等。

(2) 接口覆盖分析。如,ATM 系统中的各模块之间的接口、软件模块和 ATM 硬件的接口、人与 ATM 的接口等。

6) 测试用例补充

在软件开发的过程中,难免会因为需求变更等原因,会有功能增加、特性修改等情况发生,因此我们不可能在测试工作一开始就 100%完成所有的集成测试用例的设计,这就需要在集成测试阶段能够及时跟踪项目变化,按照需求增加和修改来补充集成测试用例,保证进行充分的集成测试。如,ATM 系统对每次交易输入的密码错误次数从 3 次改为 5 次,则必须设计相应的测试用例作为补充。

以上从 6 个方面分析了集成测试用例的设计,在实际的集成测试过程中,通常会考虑软件开发成本、测试成本、进度和质量等方面的平衡,所以,集成测试也要考虑重点突出,如,要保证对所有重点的接口以及重要的功能进行充分的测试,然后在时间允许的前提下做其他功能和接口的测试。另外,在集成测试的过程中要吸取教训和积累经验,如,用例设计要充分考虑到可回归性以及是否便于自动化测试的执行。

从以上 6 个方面设计的集成测试用例难免会存在交叉,也就是说存在不同的测试用例覆盖相同的功能和接口,出现这种问题可以通过对测试用例的评审来解决。

5.5 集成测试实例

这里仍然以单元测试中酒店服务与管理系统为例来说明集成测试的应用。在单元测试中以 Package FlashRemoting 为例来进行单元测试。在集成测试中将实现对 FlashRemoting 模块和 EncodePassword 模块集成以及 FlashRemoting 模块和 EncodePassword 模块与界面模块的集成。

1. 系统大概需求

见单元测试实例部分。

2. 集成测试接口分析

FlashRemoting 模块接口如表 5-1 所示。

表 5-1 FlashRemoting 模块接口

标 识 符	名 称	调用层次数	调用函数次数
HLD_001_INT_001	boolean LoginAdmin (String userName, String userPassword,String Private_Key)	1	1
HLD_001_INT_002	boolean LoginUser (String userName, String userPassword)	1	1

续表

标 识 符	名 称	调用层次数	调用函数次数
HLD_001_INT_003	String newUser(String firstName,String lastName,String passWord,String eMail,String salutation)	1	4
HLD_001_INT_004	String newAdmin (String firstName, String lastName,String passWord,String eMail,String salutation,String Private_Key)	1	4
HLD_001_INT_005	boolean RepassWord(String User_ID)	1	1
HLD_001_INT_006	boolean emailFormat(String email)	0	0
HLD_001_INT_007	int CheckSet(int Set)	0	0

FlashRemoting 模块和 EncodePassword 模块集成接口如表 5-2 所示。

表 5-2 FlashRemoting 模块和 EncodePassword 模块集成接口

标 识 符	名 称	调用层次数	调用函数次数
HLD_003_INT_001	byte[] encode(byte[] input,byte[] key)	1	1
HLD_003_INT_002	byte[] decode(byte[] input,byte[] key)	1	1

FlashRemoting 模块、EncodePassword 模块和界面模块集成接口如表 5-3 所示。

表 5-3 FlashRemoting 模块、EncodePassword 模块和界面模块集成接口

接 口 序 号	接 口 描 述
GUN_INT_001	主页面会员登录界面设置接口
GUN_INT_002	主页面注册新会员界面设置接口
GUN_INT_003	注册新会员称谓选择设置接口
GUN_INT_004	注册新会员重复密码匹配检查结果显示接口
GUN_INT_005	会员详细信息界面设置接口
GUN_INT_006	登录成功之后欢迎界面设置接口
GUN_INT_007	显示最后一次登录时间接口

通过分析 FlashRemoting 模块和 EncodePassword 模块的每个对外的接口函数,其中的许多函数都只有调用一层的函数,甚至不调用。因此只测 newAdmin 和 decode 两个接口。对于 GUI 接口,鉴于其测试与系统测试有重复性,这里只进行最基本的接口功能验证。

被测接口及标识如表 5-4 所示。

表 5-4 被测接口及标识

标 识 符	名 称
HLD_001_INT_004	String newAdmin (String firstName, String lastName, String passWord, String eMail,String salutation,String Private_Key)
HLD_003_INT_002	byte[] decode(byte[]input,byte[] key)
GUN_INT_001	主页面会员登录界面设置接口
GUN_INT_002	主页面注册新会员界面设置接口
GUN_INT_003	注册新会员称谓选择设置接口
GUN_INT_004	注册新会员重复密码匹配检查结果显示接口
GUN_INT_005	会员详细信息界面设置接口
GUN_INT_006	登录成功之后欢迎界面设置接口
GUN_INT_007	显示最后一次登录时间接口

3. 采用的测试方法和技术

(1) 对 FlashRemoting 模块内集成,例子仅对 newAdmin 进行测试。

(2) 在完成 FlashRemoting 模块集成测试后,对 FlashRemoting 模块和 EncodePassword 模块进行集成测试。对于 newAdmin 和 decode 这两个接口的测试,可以从接口的输入域和输出域覆盖上进行测试用例的设计,同时考虑接口功能的完整性;虽然仅仅是两个模块的集成,但实际采用的是自顶向下的集成方法。

(3) 对于界面接口,利用场景法设计测试用例,根据用例进行手动验证其功能,同时,关注其是否调用正确的接口函数,并且是否能够正确地传递输入参数,正确地获取返回值和输出参数。

(4) 参考等价类划分方法。

(5) 参考边界值分析方法。

(6) 参考使用错误猜测方法。

(7) 覆盖分析。

4. 测试环境

(1) 硬件需求。

应用服务器、数据库服务器及两台标准开发 PC 等。

(2) 软件需求。

相关操作系统、数据库系统等。

(3) 测试工具。

Github、Selenium 等。

5. 测试通过/失败标准

测试通过的标准表述如下:

- 所有的接口用例都被执行过并通过;
- 所有发现的真实缺陷都被修正并回归测试;
- newAdmin 接口和 decode 接口的输入域被 100%覆盖;
- newAdmin 接口和 decode 接口的函数调用路径被 100%覆盖。

测试失败的标准表述如下:

- 缺陷密度大于 2 个/KLOC;
- 发现有重大结构设计问题,其修改会导致 20%以上的函数接口、功能、模块数量的变化,进一步测试相关接口已经无意义;
- 发现关键功能未被设计,该功能的设计会导致 20%以上的函数接口、功能、模块数量的变化,进一步测试相关接口已经无意义。

6. 测试对象

(1) FlashRemoting 模块 tag_02 版本。

(2) EncodePassword 模块 tag_03 版本。

(3) 界面模块 tag_05 版本。

7. 用例分析与设计

1) HDL_001_INT_004 测试分析与设计

(1) 设计标识符: IT_TD_001。

(2) 被测特性。

- 输入特定参数为空时,注册失败;
- 输入管理员邮箱不合法时,注册失败;
- 输入参数合法时,注册成功。

(3) 测试方法。

分析 newAdmin 的函数调用关系图和流程图,从 newAdmin 函数测试知道,用桩对 EmailFormat、CheckSet 和 SendEmail 3 个函数进行了替代。因此,对于 newAdmin 的集成来说,关键是验证 newAdmin 与 EmailFormat、CheckSet 和 SendEmail 的接口是否与预计的效果相同。

(4) 测试项标识如表 5-5 所示。

表 5-5 测试项标识 1

测试项标识符	测试项描述	优 先 级
IT_TD_001_001	输入特定参数为空的情况	低
IT_TD_001_002	输入管理员邮箱不合法的情况	高
IT_TD_001_003	输入参数合法的情况	高

(5) 测试通过/失败标准。

所有的用例都必须被执行,且没有发现缺陷。

(6) 对应的测试用例如表 5-6～表 5-8 所示。

表 5-6 测试用例 1

测试项编号	IT_TD_001_001	
优先级	低	
测试项描述	测试特定参数为空的错误情况	
前置条件	管理员单击“注册”按钮,进入注册界面	
用 例 序 号	输 入	期 望 结 果
001	firstName="Claude" lastName="Strife" Password="" eMail="zacks_mail163.com" saluation="先生" Pivate_Key="genocide"	返回 false 反馈密码不能为空的错误信息
002	firstName="" lastName="Strife" Password="992125gjl" eMail="zacks_mail163.com" saluation="先生" Pivate_Key="genocide"	返回 false 反馈 firstName 不能为空的错误信息
003	firstName="Claude" lastName="Strife" Password="992125gjl" eMail="zacks_mail163.com" saluation="先生" Pivate_Key=""	返回 false 反馈密钥不能为空的错误信息

表 5-7　测试用例 2

测试项编号	IT_TD_001_002	
优先级	高	
测试项描述	测试邮箱参数不合法的错误情况	
前置条件	管理员单击“注册”按钮，进入注册界面	
用 例 序 号	输　　入	期 望 结 果
001	firstName＝"Claude" lastName＝"Strife" Password＝"992125gjl" eMail＝"zacks_mail163.com" saluation＝"先生" Pivate_Key＝"genocide"	返回 false 反馈邮箱错误的错误信息

表 5-8　测试用例 3

测试项编号	IT_TD_001_003	
优先级	高	
测试项描述	测试参数合法的情况	
前置条件	管理员单击“注册”按钮，进入注册界面	
用 例 序 号	输　　入	期 望 结 果
001	firstName＝"Claude" lastName＝"Strife" Password＝"992125gjl" eMail＝"zacks_mail@163.com" saluation＝"先生" Pivate_Key＝"genocide"	返回 User_ID

2）HDL_003_INT_002 测试分析与设计

（1）设计标识符：IT_TD_002。

（2）被测特性。

- 输入管理员密钥正确的情况，返回解密后的数据库中的密码；
- 输入管理员密钥错误的情况下，返回解密后错误的编码。

（3）测试方法。

分析 decode 的函数调用关系图和流程图，从 loginAdmin 函数测试知道，用桩对 decode 进行了替代。因此，对于 decode 的集成来说，关键是验证 loginAdmin 与 decode 的接口是否与预计的效果相同。

（4）测试项标识如表 5-9 所示。

表 5-9　测试项标识 2

测试项标识符	测试项描述	优　先　级
IT_TD_002_001	输入密钥错误的情况	高
IT_TD_002_002	输入密钥为空的情况	高

（5）测试通过/失败标准：所有的用例都必须被执行，且没有发现缺陷。

（6）对应的测试用例如表 5-10 和表 5-11 所示。

表 5-10　测试用例 4

测试项编号	IT_TD_002_001	
优先级	高	
测试项描述	测试管理员登录密钥为空情况	
前置条件	网站管理员单击“登录”按钮,进入登录界面	
用例序号	输　入	期望结果
001	userName="HA00000000001" userPassword="992125gjl" Private_Key=""	返回 false

表 5-11　测试用例 5

测试项编号	IT_TD_002_002	
优先级	高	
测试项描述	测试管理员登录密钥不为空的情况	
前置条件	网站管理员单击“登录”按钮,进入登录界面	
用例序号	输　入	期望结果
001	userName="HA00000000001" userPassword="992125gjl" Private_Key="genocidc"	返回经解密算法解密后的字节流

3) GUI_INT_001 测试分析与设计

(1) 设计标识符：IT_TD_003。

(2) 被测特性：主页面会员登录界面设置接口。

(3) 测试方法：检测输入的合法会员账号、密码是否能够正确地传递给 loginUser 接口函数。检测输入的验证码为空时,是否能正确提示出错信息。检测接口输出的正确性分为两个方面：一方面,验证能否正确获取 loginUser 的返回值；另一方面,能够正确获取到错误信息并显示给用户。

(4) 测试项标识如表 5-12 所示。

表 5-12　测试项标识 3

测试项标识符	测试项描述	优　先　级
IT_TD_003_001	验证接口的输入正确性	高
IT_TD_003_002	验证返回值处理的正确性	高
IT_TD_003_003	验证错误信息捕获的正确性	高

(5) 测试通过/失败标准：所有的用例都必须被执行,且没有发现缺陷。

(6) 对应测试用例：略。

4) GUI_INT_002 测试分析与设计

(1) 设计标识符：IT_TD_004。

(2) 被测特性：主页面注册新会员界面设置接口。

(3) 测试方法：检测输入的合法注册信息是否能够正确地传递给 newUser 接口函数。检测输入的必填项为空时,是否能正确提示出错信息。检测接口输出的正确性分为两个方面：

一方面，验证能否正确获取 newUser 的返回值；另一方面，能够正确获取到错误信息并显示给用户。

(4) 测试项标识如表 5-13 所示。

表 5-13　测试项标识 4

测试项标识符	测试项描述	优　先　级
IT_TD_004_001	验证接口的输入正确性	高
IT_TD_004_002	验证返回值处理的正确性	高
IT_TD_004_003	验证错误信息捕获的正确性	高

(5) 测试通过/失败标准：所有的用例都必须被执行，且没有发现错误。

(6) 对应测试用例：略。

5) GUI_INT_003 测试分析与设计

(1) 设计标识符：IT_TD_005。

(2) 被测特性：注册新会员称谓选择设置接口。

(3) 测试方法：检测称谓下拉框中选择的数据是否能够正确地传递给 newUser 接口函数。检测是否允许不选择称谓；检测是否会显示给用户正确的提示信息。

(4) 测试项标识如表 5-14 所示。

表 5-14　测试项标识 5

测试项标识符	测试项描述	优　先　级
IT_TD_005_001	验证接口的输入正确性	高
IT_TD_005_002	验证提示信息显示的正确性	高

(5) 测试通过/失败标准：所有的用例都必须被执行，且没有发现错误。

(6) 对应测试用例：略。

6) GUI_INT_004 测试分析与设计

(1) 设计标识符：IT_TD_006。

(2) 被测特性：注册新会员重复密码匹配检查结果显示接口。

(3) 测试方法：检测重复输入的密码与第一次输入的密码不匹配时的错误信息是否能够正确显示给用户。

检测两次输入密码匹配时，是否能将密码数据正确地传递给 newUser 接口函数。

(4) 测试项标识如表 5-15 所示。

表 5-15　测试项标识 6

测试项标识符	测试项描述	优　先　级
IT_TD_006_001	验证接口的输入正确性	高
IT_TD_006_002	验证错误信息捕获的正确性	高

(5) 测试通过/失败标准：所有的用例都必须被执行，且没有发现错误。

(6) 对应测试用例：略。

7) GUI_INT_005 测试分析与设计

(1) 设计标识符：IT_TD_007。

(2) 被测特性：会员详细信息界面设置接口。

(3) 测试方法：检测输入的合法注册信息是否能够正确地传递给 newUser 接口函数。检测输入的必填项为空时，是否能正确提示出错信息。检测接口输出的正确性分为两个方面：一方面，验证能否正确获取 newUser 的返回值；另一方面，能够正确获取到错误信息并显示给用户。

(4) 测试项标识如表 5-16 所示。

表 5-16　测试项标识 7

测试项标识符	测试项描述	优　先　级
IT_TD_007_001	验证接口的输入正确性	高
IT_TD_007_002	验证返回值处理的正确性	高
IT_TD_007_003	验证错误信息捕获的正确性	高

(5) 测试通过/失败标准：所有的用例都必须被执行，且没有发现错误。

(6) 对应测试用例：略。

8) GUI_INT_006 测试分析与设计

(1) 设计标识符：IT_TD_008。

(2) 被测特性：登录成功之后欢迎界面设置接口。

(3) 测试方法：检测登录成功时是否能正确调用欢迎界面。

(4) 测试项标识如表 5-17 所示。

表 5-17　测试项标识 8

测试项标识符	测试项描述	优　先　级
IT_TD_008_001	验证接口调用的正确性	高

(5) 测试通过/失败标准：所有的用例都必须被执行，且没有发现错误。

(6) 对应测试用例：略。

9) GUI_INT_007 测试分析与设计

(1) 设计标识符：IT_TD_009。

(2) 被测特性：显示最后一次登录时间接口。

(3) 测试方法：检测接口输出的正确性；能够正确获取到返回信息并显示给用户。

(4) 测试项标识如表 5-18 所示。

表 5-18　测试项标识 9

测试项标识符	测试项描述	优　先　级
IT_TD_009_001	验证信息捕获并显示的正确性	高

(5) 测试通过/失败标准：所有的用例都必须执行，且没有发现错误。

(6) 对应测试用例：略。

8. 消息调用图及覆盖率

图 5-16 是 FlashRemoting 模块和 EncodePassword 模块之间的消息调用，Package FlashRemoting 中包含有 FindPassWord.java、HotelLogin.java 和 MemberManage.java 3 个类，EncodePassword 模块含有一个 Crypt 类，测试这两模块之间的接口需要调用 JMail 类方法，用桩代替。

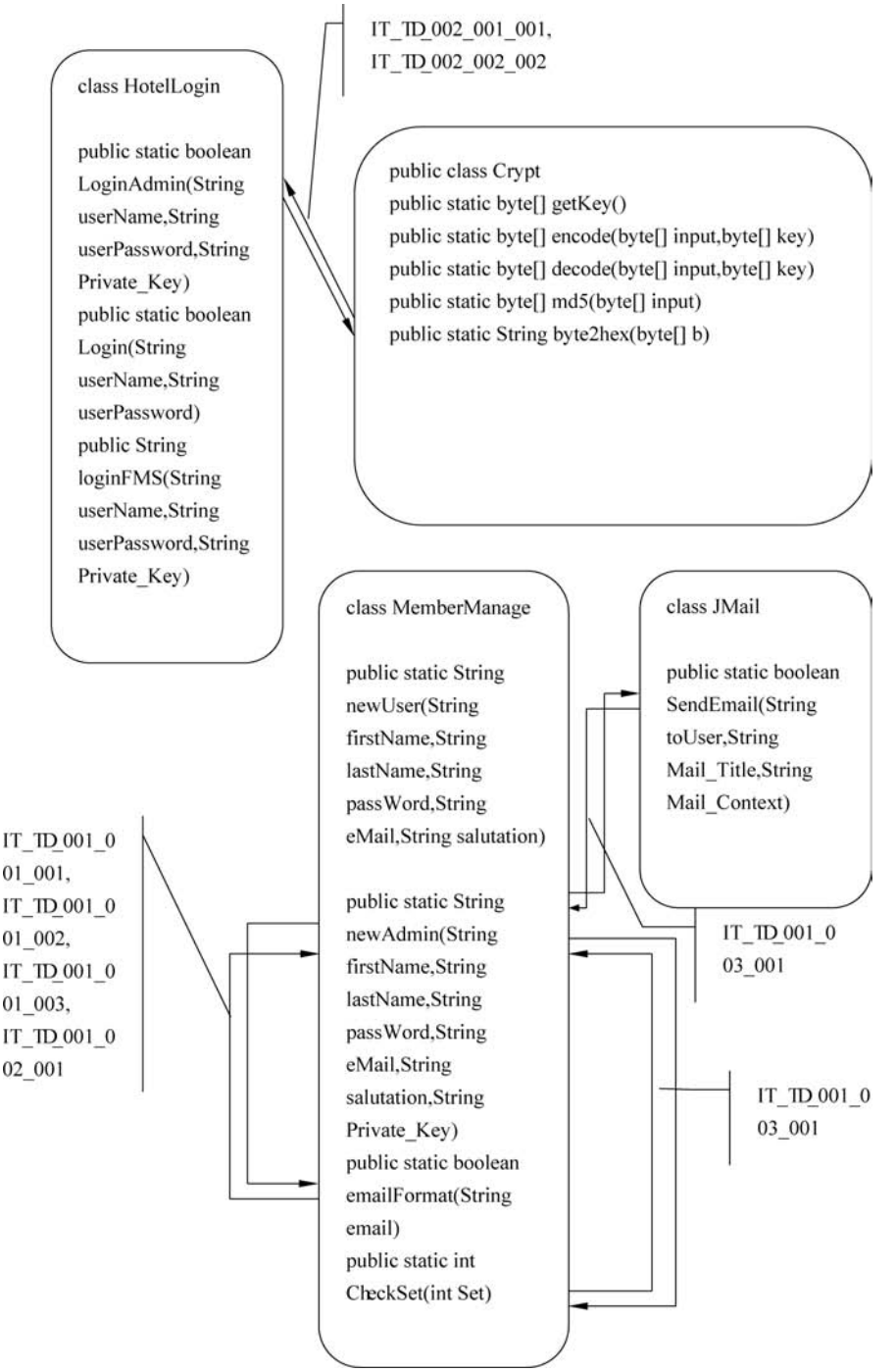


图 5-16　模块之间消息调用图

覆盖率分析表如表5-19所示。

表5-19 覆盖率分析表

标 识 符	名 称	覆盖率/%
HLD_001_INT_004	String newAdmin (String firstName, String lastName, String passWord,String eMail,String salutation,String Private_Key)	100
HLD_003_INT_002	byte[] decode(byte[] input,byte[] key)	100
GUN_INT_001	主页面会员登录界面设置接口	100
GUN_INT_002	主页面注册新会员界面设置接口	100
GUN_INT_003	注册新会员称谓选择设置接口	100
GUN_INT_004	注册新会员重复密码匹配检查结果显示接口	100
GUN_INT_005	会员详细信息界面设置接口	100
GUN_INT_006	登录成功之后欢迎界面设置接口	100
GUN_INT_007	显示最后一次登录时间接口	100

9. 测试脚本

以下列出在集成测试过程中用到的函数或方法桩代码。

MyAuthenticator_stub.java:

```
package JMailPacket;

import Javax.mail.PasswordAuthentication;

public class MyAuthenticator_stub {
  private String strUser;
  private String strPwd;

  public MyAuthenticator_stub()
  {
  this.strUser = "zacks_mail@163.com";
  this.strPwd = "992125gjl";
  }

  protected PasswordAuthentication getPasswordAuthentication()
  {
  return new PasswordAuthentication(strUser, strPwd);
  }
}
```

EmailFormat_stub.java:

```
package FlashRemoting;

public class EmailFormat_stub {
    public static boolean emailFormat(String email)
    {
        return true;
    }
}
```

Crypt_stub.java：

```
package EncodePassword;

import Java.security.*;
import Javax.crypto.*;

public class Crypt_stub
{
public static boolean STUB_SUC = true;

private static String Algorithm = "DES"; //定义加密算法,可用 DES,DESede,Blowfish

static
{
    Security.addProvider(new com.sun.crypto.provider.SunJCE());
}

//生成密钥,注意此步骤时间比较长

public static byte[] getKey() throws Exception
{
    KeyGenerator keygen = KeyGenerator.getInstance(Algorithm);
    SecretKey deskey = keygen.generateKey();
    return deskey.getEncoded();
}

//加密
public static byte[] encode(byte[] input,byte[] key) throws Exception
{
     String skey = new String(key);
         byte[] result = "false".getBytes();
         if(FlashRemoting.NewUser_UnitTest.CASE_NUM.compareTo("IT_TD_001_001_001") == 0)
         {
            if(skey.compareTo("genocide") != 0)
            {
                STUB_SUC = false;
                return result;
            }
            result = "992125gjl".getBytes();
            return result;
         }
         if((FlashRemoting.NewUser_UnitTest.CASE_NUM.compareTo("IT_TD_001_001_003") == 0)
         || (FlashRemoting.NewUser_UnitTest.CASE_NUM.compareTo("IT_TD_001_001_002") == 0)
         || (FlashRemoting.NewUser_UnitTest.CASE_NUM.compareTo("IT_TD_001_002_001") == 0))
         {
            if(skey.compareTo("genocidc") != 0)
            {
                STUB_SUC = false;
            }
            return result;
         }
         if(FlashRemoting.NewUser_UnitTest.CASE_NUM.compareTo("IT_TD_001_003_001") == 0))
         {
            if(skey.compareTo("genocide") != 0)
            {
```

```
                STUB_SUC = false;
            }
            return result;
        }

        STUB_SUC = false;
        return result;
    }

    //解密
    public static byte[] decode(byte[] input,byte[] key) throws Exception
    {
        String skey = new String(key);
        byte[] result = "false".getBytes();
        if(FlashRemoting.Login_UnitTest.CASE_NUM.compareTo("IT_TD_002_001_001") == 0)

        {
            if(skey. compareTo("genocide") != 0)
            {
                STUB_SUC = false;
                return result;
            }
            result = "992125gjl".getBytes();
            return result;
        }
        if(FlashRemoting.Login_UnitTest.CASE_NUM.compareTo("IT_TD_002_002_001") == 0)

        {
            if(skey. compareTo("genocidc") != 0)
            {
                STUB_SUC = false;
            }
            return result;
        }

        STUB_SUC = false;
        return result;
    }

    //md5()信息摘要, 不可逆
    public static byte[] md5(byte[] input) throws Exception
    {
        Java.security.MessageDigest alg = Java.security.MessageDigest.getInstance("MD5");
    //或 "SHA - 1"
        alg.update(input);
        byte[] digest = alg.digest();
        return digest;
    }

      //字节码转换成十六进制字符串
    public static String byte2hex(byte[] b)
    {
        String hs = "";
        String stmp = "";
        for (int n = 0;n < b.length;n++)
```

```
    {
        stmp = (Java.lang.Integer.toHexString(b[n] & 0XFF));
        if (stmp.length() == 1)
            hs = hs + "0" + stmp;
        else
            hs = hs + stmp;
        if (n < b.length - 1)
            hs = hs + ":";
    }
    return hs.toUpperCase();
}
}
```

JMail_stub.java：

```
package JMailPacket;

public class JMail_stub {
public static boolean SendEmail(String toUser,String Mail_Title,String Mail_Context)
{
    return true;
}
}
```

10. 用例执行及报告分析

略。

练　　习

1. 分析集成测试的重点及集成测试回归的目的。

2. 简要说明集成测试环境的创建。

3. 分析结构化开发和面向对象开发的集成测试依据。

4. 简述集成测试的覆盖指标要求。

5. 分别用自顶向下集成、自底向上集成、大爆炸集成、三明治集成方法完成对图 5-17 的集成测试。

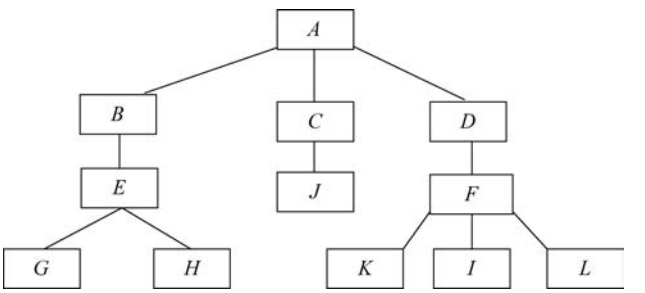


图 5-17　习题 5

6. 以一个具体项目为例分析服务端类间集成、前端和服务端分离的集成测试。