第 11 章 面向对象编码与测试

学习目标：

理解面向对象编码和面向对象测试的定义；理解面向对象测试用例的定义；掌握常见面向对象测试策略；掌握良好的面向对象设计风格；了解面向对象设计语言的优点和技术特点；了解设计面向对象测试用例的要点。

11.1 面向对象设计语言

面向对象(OO, Objected Oriented)方法是 1979 年以后发展起来的，一种系统的软件方法学。面向对象技术与方法包括 5 个阶段，分别是，面向对象分析、面向对象设计、面向对象编码、面向对象测试和面向对象维护。

以上的五个阶段，都少不了使用面向对象设计语言。面向对象设计语言（Object-Oriented Language），它是以对象作为基本程序结构单位的一类程序设计语言，它用于描述的是以对象为核心的设计，而对象是程序运行的基本成分。面向对象设计语言中提供了类、继承等成分，有识认性、多态性、类别性和继承性四个主要特点。

11.1.1 典型的面向对象设计语言

一般认为，较典型的面向对象设计语言有：

Smalltalk 语言支持单继承、多态和动态绑定；

EIFFEL 语言支持多继承、多态和动态绑定；

C++，支持多继承、多态和部分动态绑定；

Java，支持单继承、多态和部分动态绑定。

四种语言涉及概念的含义虽然基本相同，但所用术语有别。

C#，也支持单继承，与 Java 和 C++等有很多类似之处。

11.1.2 面向对象设计语言的优点

面向对象设计语言最大的优点是符合抽象、封装、细节隐藏等基本程序设计的原则。需求分析、程序设计、编码实现等过程可以无缝衔接，面向对象设计语言必不可少。具体优点如下：

（1）一致的表达方法

面向对象的开发基于不随时间变化的并且是一致的表示方法。这种表示方法，是从问题域，到面向对象分析方法（OOA），从面向对象分析方法，到面向对象设计（OOD），最后从面向对象设计，到面向对象编程(OOP)，始终稳定不变。一致的表示方法：既有利于在软件开发过程中始终使用统一的概念，也有利于维护人员理解软件的各种配置成分。

（2）可重用性

为了能带来商业利益，必须在更广泛的范围中，运用重用机制，而不是仅仅在程序设计这个层次上。因此，在面向对象分析方法（OOA）、面向对象设计（OOD）直到面向对象编程(OOP)中都很明确，表示问题域语义，其意义是十分深远的。随着时间的推移，软件开发组织既能重用它在某个问题域内的 OOA 结果，也能重用相应的 OOD 和 OOP 结果。

（3）可维护性

以 ATM 系统为例，阐述了在程序中表达问题域语义的意义。假设系统维护没有适当的文档可供参考，那么维护人员人工浏览应用程序或使用软件工具扫描，打印或编写程序明确的问题域语义声明，如果维护人员看到“ATM”、“账户”、“现金兑换卡”等，这对维护人员维护软件有很大的帮助。

11.1.3 面向对象设计语言的技术特点

面向对象设计语言，借鉴了 20 世纪 50 年代的人工智能语言 LISP 语言，引入了动态绑定的概念，以及交互式开发环境的思想。面向对象设计语言开始于 20 世纪 60 年代的离散事件模拟语言 SIMULA67，引入了类的要领和继承，成形于 20 世纪 70 年代的 Smalltalk。

面向对象设计语言的发展有两个方向：一种是纯面向对象设计语言，如 Smalltalk、EIFFEL 等；另一种为混合型面向对象设计语言，他是在过程式语言或者其它语言中，加入类、继承等成分，如 C++、Objective-C 等。

下面介绍在选择面向对象设计语言时应该着重考察的一些技术特点。

（1）面向对象设计语言具有支持类和对象概念的定义与实现机制

几乎全部的面向对象设计语言，都提供了类的定义机制和对象的动态创建功能。类的定义是面向对象设计语言的基础，也是对象概念实现的描述。对象的动态创建可以通过类中定义的操作来创建本类的对象，也可以通过特殊的操作按定义创建对象。

（2）面向对象设计语言具有实现继承的语言机制

继承是面向对象程序设计的一个重要特色，是实现代码重用的重要手段。不同的面向对象设计语言提供了不同的继承实现机制。

（3）面向对象设计语言具有实现属性和服务的机制

面向对象设计语言支持消息连接、控制服务可见性的机制。通过消息的发送实现对象之间的相互协作，消息的格式和服务的调用方式上不同的语言有不同的实现方法。

（4）面向对象设计语言具有参数化类

参数化类是指使用一个或多个类型去参数化一个类的机制。有了这种机制，程序员就可以先定义一个参数化的类模板(即在类定义中包含以参数形式出现的一个或多个类型)， 然后把数据类型作为参数传递进来，从而把这个类模板应用在不同的应用程序中，或用在同一应用程序的不同部分。

（5）面向对象设计语言提供类型检查

面向对象设计语言的编译系统在编译时，对类型的匹配检查的严格程度，是判断语言实现能力的一个重要指标。程序设计语言按照编译时进行类型检查的严格程度分为两类：一类是弱类型，语言仅要求每个变量或属性隶属于一个对象；一类是强类型，语言要求每个变量或属性必须准确地属于某个特定的类，当今大多数新语言都是强类型的。

强类型语言有两个优点。一方面表现在有利于在编译时发现程序错误，有助于提高软件的可靠性和运行效率；另一方面，增加了优化的可能性。

通常使用强类型编译型语言开发软件产品，弱类型是用于解释的解释型语言能快速开发原型。

（6）面向对象设计语言提供类库

如何去提高软件的可重用性？大多数面向对象设计语言都提供一个实用的类库，类库中包含实现通用数据结构(如动态数组、表、队列、栈、树等)的类，我们把这些类称为包容类。在类库中，还可以找到实现各种关联的类。更完整的类库通常还提供独立于具体设备的接口

类(例如，输入 / 输出流)。此外，用于实现窗口系统的用户界面类也非常有用，它们构成一个相对独立的图形库。

（7）面向对象设计语言提供持久对象的保存

持久对象是可以长期保存的数据对象，与程序执行的生命周期无关。想要长时间保存数据有两个原因，一是实现不同程序之间的数据传输；二是恢复被中断程序的运行。

（8）面向对象设计语言提供封装与打包

可视化开发环境是目前应用最广泛的开发环境，良好的开发环境可以提高软件生产效率，保证软件产品的质量。

（9）面向对象设计语言提供可视化开发环境

不同的语言对封装的支持程度存在一些差异,允许一个类的代码去直接访问另一个类的属性以及适应继承不当等都可能损害封装。

11.1.4 选择面向对象设计语言

开发人员在选择面向对象设计语言时，还应该着重考虑以下一些实际因素。

（1）将来能否占主导地位

在若干年以后，占主导地位的将是哪种面向对象的程序设计语言?为了使自己的产品在很多年后仍然具有很强的生命力，人们可能希望采用将来占主导地位的语言编程。

根据目前占有的市场份额，以及专业书刊和学术会议上所做的分析、评价，人们往往能够对未来哪种面向对象设计语言将占据主导地位做出预测。

但是，最终决定选用哪种面向对象设计语言的实际因素，往往是诸如成本一类的经济因素，而不是技术因素。

（2）可重用性

使用面向对象方法，开发软件的基本目的和主要优点是通过重用来提高软件生产率。因此，应首先选择最完整、最准确地表达问题域语义的面向对象设计语言。

（3）类库和开发环境

可重用性不仅取决于面向对象编程语言本身，还取决于开发环境和类库。事实上，语言、开发环境和类库都聚集在一起以确定可重用性。

在考虑类库时，不仅要考虑类库是否提供，还要考虑类库中提供了哪些有价值的类。随着类库的成熟和丰富，为新应用程序编写的代码将越来越少。

为了方便可重用类的积累和现有类的重用，除了上述基本的软件工具外，开发环境还应提供易于使用的类库编辑工具和浏览工具。

（4）其它因素

选择编程语言时要考虑的其他因素有：为用户学习面向对象的分析、设计和编码技术提供的培训服务;在使用面向对象设计语言期间提供技术支持;开发人员可以使用的开发工具、开发平台和发布平台；机器性能和内存要求；易于集成现有软件等。

11.2 面向对象设计风格

程序设计风格事关软件的质量和效率。在所有的程序设计应用中，风格好坏至关重要.这一点在面向对象设计和程序设计中表现尤为突出，应引起特别注意。因为面向对象方法的许多优点都是建立在复用性好、扩充性好等良好的程序设计风格的基础上的。

一个良好的程序风格特别注重是否满足功能的需求。一般地说，根据正确的设计准则而设计的程序大多是正确、可读、可扩充且易于调试成功的。大多数用于传统程度设计语言的设计准则也适合于面向对象程序设计。然而，针对面向对象程序设计中所独有的封装、继承等特性，则需要新的设计准则。接下来主要从复用性、继承机制、扩充性、健壮性等几方面论述面向对象程序设计风格。

11.2.1 复用性

软件复用是提高软件生产力、提高软件质量和缩短软件开发周期的一种有效的手段。据分析统计，开发一个新的应用系统，40%—60%的代码是重复以前的成分，有的甚至高达 85%。

传统语言也可以具备复用性，但面向对象设计语言的语法机制大大增加了复用的可能，使复用更加方便灵活，如鱼得水。

软件复用（reuse,又称软件重用）是 80 年代以来国际上非常重视的一项新技术，被认为是最有希望成数量级地提高软件人员劳动生产率的技术，软件复用性被认为与可靠性、正确性和可维护性居于同等重要地位。

（1）代码复用的种类

存在两种代码复用类型:一种是本项目之内的代码复用，另一种是新项目对旧项目的代码复用（包括本项目复用以前项目的类部件及本项目为其它新项目复用本项目的部件做准备）。内部复用只是找出设计中的冗余代码序列（包括数据和操作），然后把它们组成各种各样的类部件，供系统本身复用。外部共享需要从长远目标考虑。设计者很可能仔细考虑一些子系统的复用，如抽象出数据类型，报表生成软件工具类库等。

（2）复用的设计准则

① 保持方法的耦合性(单纯化)：一种方法只能完成单个的功能或一组紧密相关的功能。如果某方法涉及二个或多个不相关的功能，就把它分解成更小的方法。

② 尽量设计小方法：如果方法太大，应把它分解成更小的方法。一般一个方法不要占用超过一页纸。

③ 方法的一致性：相似的方法应该有一致的名称、条件、参数次序、数据类型、返回值及出错条件等。只要有可能就应该维持平行结构。

④ 分离的策略和实现：策略方法制定决策，变换参数，并且收集全局资源。策略方法常常是高度依赖应用的，但很容易编写和理解。策略方法可喻为是建筑工程中的“水泥”。

实现方法执行具体的详细操作，而不决定是否或为什么执行。如果实现方法通到错误，则它们应该只返回状态，并不采取行动。实现方法是对具体的变量执行具体的计算，其中常常包含复杂的算法。实现方法并不访问全局资源，不制定决策,不包括缺省或协调控制流等。由于实现方法是自含式算法，它们有可能在其它上下文中也有意义且可复用，实现方法可喻为建筑工程中的“砖头”。

只有把“砖头”和“水泥”科学的结合，才能建起华丽、坚固的大厦。不要将策略和实现放入同一方法中。应把算法的核心部分放入一个可区分的，完全具体的实现方法中。这需要从策略方法中抽象出具体的参数作为调用实现方法的变量。

⑤ 统一的覆盖面：如果输入条件在各种组合中出现，则应该针对所有组合卸除方法，而不能仅仅针对当前用到的组合情况些方法。

⑥应该让方法尽可能广泛些,尽量归纳出参数类型、前提及约束。归纳出方法是如何执行的假设及方法操作的内容范畴等。来对空值、极限值及边界值以外做出有意义的响应。

例如，产生用于检索记录的条件表达式的方法，这个方法应该是对任一个数据库文件都适用，即库文件名参数化。只要传递一个实在的库文件名，该方法便可动态的返回一个可用的条件表达式。

⑦ 避免全局信息：尽量少涉及外部事物。

⑧ 利用继承机制：在面向对象程序中，使用继承机制是实现共享和提高重用程度的主要途径。

11.2.2 使用继承机制

继承是类与类之间的一种关系，继承允许新的软件模块或类可定义对现存类的扩充、专门化或合并。继承是实现利用可复用软件构件构造系统的有效机制.是面向对象软件开发的关键。某些 OOP 语言允许多重继承，即允许一个类具有一个以上的父类。

前述的准则提高了继承共享代码的机会。有时，尽管不同的类上的方法是相似的，但还不是以同一个可继承的方法来表示。下面几种技术可以用来将方法分解以继承某些代码。

（1）子过程:最简单的方法是将共同代码分解成单个的能被其它方法调用的方法。这个共存的方法可以附在祖类上。例如，控制打印机的方法可以分解成两个简单的方法，一个是接通打印机，另一个是断开打印机。

（2）分解因子:某些情况下,提高相似类的代码复用性的最好的方法是在不同类的方法之间分解出其不同因子,将余下的代码作为共享的方法。当方法之间差别不大而相似很多时，使用这种方法十分有效。有时为了组织高层次方法，必须加一个抽象类。

（3）委派：当不存在实际的父类/子类关系时，有时在一个程序内使用继承也会提高代码重用性。这时可以使用实现继承,用委派来代替。在这种情况，用委派机制来实现对所期望的代码复用，可以减少副作用。

（4）封装内部代码:程序员都很希望能重用那些同一应用中不同界面风格的代码。直接在外部代码中插入调用不如把其行为封装在一个操作或类内安全。这样方法的外部程序或软件包可以被改变或替换，程序员仅仅在一处改变其代码就可以了。

11.2.3 提高可扩充性

重用准则提高了扩充性。另外，下面的面向对象的原理也提高扩充性。

（1）封装类：如果某类的内部结构对其它类是隐藏的，则可以封装该类。只有该类上的方法才能处理它的实现。

OOP 风格与传统风格的本质区别在于选择机制。选择机制改变了系统设计者选择与相应操作匹配的数据类型的负担，而在传统设计中，这是由用户自己完成的。然而，这一微小的变化体现了 OOP 技术的全部力量。

选择机制是一个简单而有效的系统构建工具,它允许系统设计者清楚地指示他们提供的可见界面，以便用户（应用程序程序员）可以看到对象提供的所有操作，而不是数据。从用户的角度来看，对象提供了一组眼睛服务，而服务的具体实现是对用户的屏蔽。这是真正的封装。如何实现对象中的操作以及如何管理内部数据都封装在流程接口中，用户可以通过该接口访问对象。对应用程序员来说：封装意味着他只能看见对象所提供的服务，而看不见服务的具体实现，只有系统设计员才能看见对象的私有数据及其过程。

（2）封装数据结构:不能从方法得到数据结构。内部数据结构是针对方法的算法的。如果从方法中得到数据结构，则限制了以后改变算法的灵活。

（3）避免遍历多条链或方法：一种方法应该只包含对象模型中有限的知识。

（4）避免对象类型上的情况语句：使用方法来代替情况语句。情况语句可以用来测试对象的内部属性但不能用来选择基于对象类型的行为。

（5）区分公共操作和私有操作:公共操作是类外部可见的且具有明晰的接口。一旦其它类需要使用公共操作，则对它的接口的改变花费太大。因而，必须慎审定义公共操作。私有操作是内部的且用来帮助实现公共操作，私有操作可以删除或改变。

将操作分成公共和私有两类，主要的原因在于：

1）不必要使用户卷入类的细节中。

2）由于私有方法取决于内部的实现策略，所以当实现改变时，方法设计者可能改变参数的数目和类型。

3）私有方法增加了模块度。方法的内部细节只影响该类中的方法，而不影响其它方法。

同样，属性和联系也可分成公有的与私有的。另外，公有属性和联系又可以具有只读权限或只写权限二种。

11.2.4 不容忽视健壮性

在写方法时，在尽力考虑效率的同时不能忽视健壮性。健壮性(robustness，或称鲁棒性)是指在异常条件下，软件系统仍能运行的能力。当方法接收了错误的参数时，它并不失败，那我们就称方法健壮性好。健壮性讨论的是在异常条件下软件的行为，它与正确性不同，正确性讨论的是在需求规格说明中明确陈述的软件行为。内部排错与效率之间需要折衷。用户排错的健壮性是不能忽视的。

（1）防止出错。软件应该能够排除用户输入错误的能力。用户不正确的输入错误不应该引起软件的中断。任何接收用户输入的方法都必须对那些有可能引起错误的输入得以证实。

方法设计者必须考虑二种出错情形：一是分析阶段中应用（用户）错误，二是问题陈述中存在的条件转述。应尽可能地保证不出现程序错误，并且即使出现致命性错误，也应该给出良好的对话提示。

（2）优化程序代码。程序员往往花太多的努力来提高那些常用代码的效率，在进行优化之前，就测试程序的性能，我们会惊奇地发现大部分消耗的时间并不多。仔细研究应用，找出什么是重要的,比如最坏情况的次数及操作频率。如果可以使用多种方法实现某个操作，就应该从内存、速度及实现的简单程度等来估计折衷。

（3）参数合法性。对于那些用户使用的外部操作，必须严格检查它们的参数以防失败。但内部方法已假设了它们参数的合法性，公有操作尤其要检查其参数的合法性，因为外部用户很可能破坏参数上的约束。内部或私有操作则常常假设先决条件。

11.3 面向对象测试策略

许多策略可用于测试软件。其中的一个极端是，软件团队等到系统完全建成后再对整个系统执行测试，以期望发现错误。虽然这种方法很有吸引力，但效果不好，可能得到的是有许多缺陷的软件，致使所有的利益相关者感到失望。

另一个极端是，无论系统的任何一部分在何时建成，软件工程师每天都在进行测试。

简单地说，测试的目标是在一个实际的时间框架内尽可能多地查找错误，并进行可管理的工作量。对于面向对象的软件，虽然这个基本目标保持不变，但是面向对象软件的性质改变了测试策略。

11.3.1 面向对象环境中的单元测试

在考虑面向对象的软件时，单元的概念发生了变化。封装导出了类和对象的定义。这意味着每个类和类的实例包装有属性（数据）和处理这些数据的操作。封装的类常是单元测试的重点，然而，类中包含的操作（方法）是最小的可测试单元。由于类中可以包含很多不同的操作，且特殊的操作可以作为不同类的一部分存在，因此，必须改变单元测试的战术。

我们不再孤立地对单个操作进行测试（传统的单元测试观点），而是将其作为类的一部分。为便于说明，考虑一个类层次结构，在此结构内，对超类定义某操作，并且一些子类继承了操作每个子类使用等的操作，但它应用于为每个子类定义的私有属性和操作的环境内。由于操作应用的环境有细微的差别，因此有必要在每个子类的环境中测试操作这意味着在面向对象环境中，以独立的方式测试操作(传统的单元测试方法）往往是无效的。

11.3.2 面向对象环境中的集成测试

由于面向对象软件中没有明显的层次控制结构，传统的自顶向下和自底向上的集成策略意义不大。此外，由于类的组件之间存在直接或间接的交互，通常不可能将操作集成到类中(传统的增量集成方法)。

面向对象系统的集成测试有两种不同的策略：

第一种策略是基于线程的测试（thread-based testing),对响应系统的一个输入或事件所需的一组类进行集成。每个线程单独地集成和测试。应用回归测试以确保没有产生副作用。

第二种方法是基于使用的测试（use-based testing),通过测试很少使用服务类（如果有的话）的那些类（称为独立类）开始系统的构建。独立类测试完成后，利用独立类测试下一层次的类（称为依赖类）。继续依赖类的测试直到完成整个系统。

在进行面向对象系统的集成测试时，驱动模块和桩模块的使用也发生了变化。驱动模块可用于低层操作的测试以及整组类的测试。驱动模块也可用于代替用户界面，以便在界面实现之前就可以进行系统功能的测试。桩模块可用于类间需要协作但其中的一个或多个协作类还未完全实现的情况。

11.3.3 面向对象环境中的确认测试

确认测试在集成测试结束时开始，当单个组件被测试后，软件被组装成一个完整的包，并且发现并纠正了接口错误。当进行验证测试或系统级测试时，不同类型软件之间的差异已经消失，测试集中于用户的可见操作和用户可识别的系统输出。

确认可用几种方式进行定义，但是，其中一个简单（尽管粗糙）的定义是当软件可以按照客户合理的预期方式工作时，确认就算成功。在这一点上，喜欢吹毛求疵的软件开发人员可能会提出异议：“谁或者什么是合理预期的裁决者呢？”如果已经开发了软件需求规格说明文档，那么此文档就描述了所有用户可见的软件属性，并包含确认准则部分，确认准则部分就形成了确认测试方法的基础。

（1）确认测试准则

软件验证是通过一系列测试来实现的，这些测试表明软件功能满足软件要求。如果发现与规范的偏差，将创建缺陷列表。并且必须确定（利益相关者可以接受的）解决缺陷的方法。

（2）配置评审

确认过程的一个重要成分是配置评审。评审的目的是确保所有的软件配置元素已正确开发、编目，且具有改善支持活动的必要细节。有时将配置评审称为审核（audit)。

（3）α测试

对软件开发者而言，预见用户如何实际使用程序几乎是不可能的。软件使用指南（使用手册）可能会被错误理解；可能会使用户感到奇怪的数据组合；测试者看起来很明显的输出对于工作现场的用户却是难以理解的。

为客户开发定制软件时，执行一系列验收测试能使客户确认所有的需求。验收测试是由最终用户而不是软件工程师进行的，它的范围从非正式的“测试驱动”直到有计划地、系统地进行一系列测试。实际上，验收测试的执行可能超过几个星期甚至几个月，因此，可以发现长时间以来影响系统的累积错误。

若将软件开发为产品，由多个用户使用，让每个用户都进行正式的验收测试，这当然是不切实际的。多数软件开发者使用称为α测试与β测试的过程，以期查找到似乎只有最终用户才能发现的错误。

α测试是由有代表性的最终用户在开发者的场所进行。软件在自然设置下使用，开发者站在用户的后面观看，并记录错误和使用问题。α测试在受控的环境下进行。

4. β测试

β测试在一个或多个最终用户场所进行。与α测试不同，开发者通常不在场，因此，β测试是在不为开发者控制的环境下“现场”应用软件。最终用户记录测试过程中遇见的所有问题（现实存在的或想象的），并定期报告给开发者。接到β测试的问题报告之后，开发人员对软件进行修改，β测试之间的区然后准备向最终用户发布软件产品。

β测试的一种变体称为客户验收测试，有时是按照合同交付给客户时进行的。客户执行一系列的特定测试，试图在从开发者那里接收软件之前发现错误。在某些情况下（例如大公司或政府系统），验收测试可能是非常正式的，会持续很多天甚至好几个星期。

11.4 面向对象测试用例设计

目前，面向对象软件测试用例设计方法，还处于研究和发展阶段。与传统的软件测试不同的是。面向对象测试更关注于设计适当的操作序列以检查类的状态。

11.4.1 设计测试用例的要点

(1) 应该唯一标识每一个测试案例，并且与被测试的类明显地建立关联；

(2) 陈述测试对象的一组特定状态；

(3) 为每个测试建立一套测试步骤，要考虑或识别的问题包括：测试对象的一组特定状态、一组消息和操作、测试对象时可能发生的一组异常、一组外部条件和辅助信息，以帮助理解和实施测试。

11.4.2 设计类测试用例

对于面向对象软件，小型测试着重测试单个类和类的封装，即类级别的测试，测试方法有随机测试、划分测试和基于故障的测试等。

（1）类级随机测试

随机测试是针对软件在使用过程中随机产生的一系列不同的操作序列设计的测试案例，可以测试不同的类实例生存历史。

为了简要地说明这些方法，考虑一个记事本的应用。在这个应用中，类 text 有以下操作：open(打开)，new(新建)，read(读取)，write(写入)，copy(复制)，paste(粘贴)，view(查看)，save(保存)和 close(关闭)。这些操作的每一个都能应用于类 text，但是由于这个问题的本质提出了某些约束条件。例如，在其他操作执行之前，必须首先执行 open 操作，并

且在所有其他操作执行完，最后必须执行 close 操作。即使对于这些约束，还存在这些操作的许多不同的排列。例如，text 的一个的最小操作序列为：

open->new->write->save->close

另外，有其他很多行为可以出现在这个序列中

open->new->write-> [read| write| copy |paste]->save->close

这样可以随机地生成一系列不同的操作序列作为测试用例，测试类实例的不同生存历史。

（2）类级划分测试

划分测试方法与传统软件测试采用的等价划分方法类似，减少了测试类所需要的测试用例的数量。首先，用不同的划分方法(包括基于状态的划分方法、基于属性的划分方法、基于功能的划分方法)，把输入和输出分类，然后把划分出来的每个类别设计测试用例。

下面分别介绍划分类别的方法：

基于状态的划分方法是根据操作改变类状态的能力对操作进行范畴划分。仍以 text 类为例，首先将状态操作和非状态操作分开，状态操作包括 read 和 write，而非状态操作有 view，然后分别为它们设计测试用例。

测试用例 1：open->new->write->read->write->save->close

测试用例 2：open->new->write->read->write->view->save->close

（3）类级基于故障的测试

基于故障的测试与传统的错误测试推测法类似。首先，推测软件中可能有的错误，然后，设计出最可能发现这些错误的测试案例，这大程度上要依靠测试人员的经验。

11.4.3 测试类间测试用例

从面向对象的集成测试开始，设计测试用例时考虑了类间协作，通常可以从 OOA 的类关系和类行为模型中派生类间测试用例。类间测试方法包括随机测试方法、分区测试方法、基于场景的测试和行为测试。随机测试方法和分区测试方法类似于类级随机测试和类级分区测试，主要研究基于场景的测试和行为测试。

（1）基于场景的测试

基于场景的测试关注用户所做的事情，而这正是错误测试所忽略的，即不正确的规范和子系统之间的交互。当出现与不正确的规格有关的错误时，软件可能不能满足用户的需要，从而影响软件的质量；当一个子系统的行为所创建的环境使另一个子系统失败时，子系统之间的交互错误就会发生。

（2）行为测试

行为测试即从动态模型导出测试用例；用状态转换图作为表示类的动态行为模型，类的状态图可以导出测试该类的动态行为的测试用例。设计的测试用例，一方面应该覆盖所有状态，另一方面应该导出足够的测试用例，以保证该类的所有行为都被适当的测试过。

11.5 本章小结

本章介绍了面向对象的设计语言和设计风格，系统的概括了面向对象的测试策略，并阐述了面向对象测试用例设计的相关重点。

面向对象方法学把分析、设计和实现很自然地联系在一起了。虽然面向对象设计原则上不依赖于特定的实现环境，但是实现结果和实现成本却在很大程度上取决于实现环境。因此，直接支持面向对象设计范式的面向对象程序语言、开发环境及类库，对于面向对象实现来说是非常重要的。

为了把面向对象设计结果顺利地转变成面向对象程序，首先应该选择一种适当的程序设计语言。面向对象的程序设计语言非常适合用来实现面向对象设计结果。事实上，具有方便的开发环境和丰富的类库的面向对象程序设计语言，是实现面向对象设计的最佳选择。

良好的程序设计风格对于面向对象实现来说格外重要。它既包括传统的程序设计风格准则，也包括与面向对象方法的特点相适应的一些新准则。

面向对象方法学使用独特的概念和技术完成软件开发工作，因此，在测试面向对象程序的时候，除了继承传统的测试技术之外，还必须研究与面向对象程序特点相适应的新的测试技术。

面向对象测试的总目标与传统软件测试的目标相同，也是用最小的工作量发现最多的错误。但是，面向对象测试的策略和技术与传统测试有所不同，测试的焦点从过程构件(传统模块)移向了对象类。

一旦完成了面向对象程序设计，就开始对每个类进行单元测试。测试类时使用的方法主要有随机测试、划分测试和基于故障的测试。每种方法都测试类中封装的操作。应该设计测试序列以保证相关的操作受到充分测试。检查对象的状态(由对象的属性值表示)，以确定是否存在错误。

可以采用基于线程或基于使用的策略完成集成测试。基于线程的测试，集成一组相互协作以对某个输人或某个事件做出响应的类。基于使用的测试，从那些不使用服务器类的类开始，按层次构造系统。设计集成测试用例，也可以采用随机测试和划分测试方法。此外，从动态模型导出的测试用例，可以测试指定的类及其协作者。

面向对象系统的确认测试也是面向黑盒的，并且可以应用传统的黑盒方法完成测试工作。但是，基于情景的测试是面向对象系统确认测试的主要方法。

习题

1．下列关于面向对象的分析与设计的描述，正确的是（）。

A．面向对象设计描述软件要做什么

B．面向对象分析不需要考虑技术和实现层面的细节

C．面向对象分析的输入是面向对象设计的结果

D．面向对象设计的结果是简单的分析模型

2．通常对象有很多属性，但对于外部对象来说某些属性应该不能被直接访问，下面哪个不是 UML 中类成员访问限定性（）

A．Public　B．Protected C．Private D．friendly

3．一个对象和另一个对象之间，通过消息来进行通信。消息通信在面向对象的语言中即（）

A．方法实现　B．方法调用　C．方法嵌套 D．方法定义

4．面向对象方法中（）机制是子类可以自动拥有父类的全部操作。

A．约束 B．对象映射 C．信息隐藏 D．继承

5．下列关于面向对象的分析与设计的说法中，不正确的是（）。

A．面向对象分析侧重于理解问题

B．面向对象设计不需要考虑技术和实现层面的细节

C．面向对象分析描述软件要做什么

D．面向对象设计侧重于理解解决方案

6.为什么类是面向对象系统中测试的最小合理单位？

7.简述面向对象系统开发过程。

第 11 章课后题答案

1-5 B D B D B

6.为什么类是面向对象系统中测试的最小合理单位？

答：在面向对象软件中，单元的概念发生了变化，不再是传统软件单元测试中关注的算法细节和流经模块接口数据，而是测试由封装在类中的操作和类的状态行为驱动。最小的可测试单元是封装了的类，一个类包含了不同的操作，而一个操作也是有不同的类组成的，传统的单元测试已经不再能满足面向对象软件的特点了，而以类作为最小的测试单元更加合理。

7.简述面向对象系统开发过程。

答：

1.面向对象的分析（OOA）

2.面向对象的设计（OOD）（高层设计和类设计）

3.面向对象实现与测试

4.维护阶段