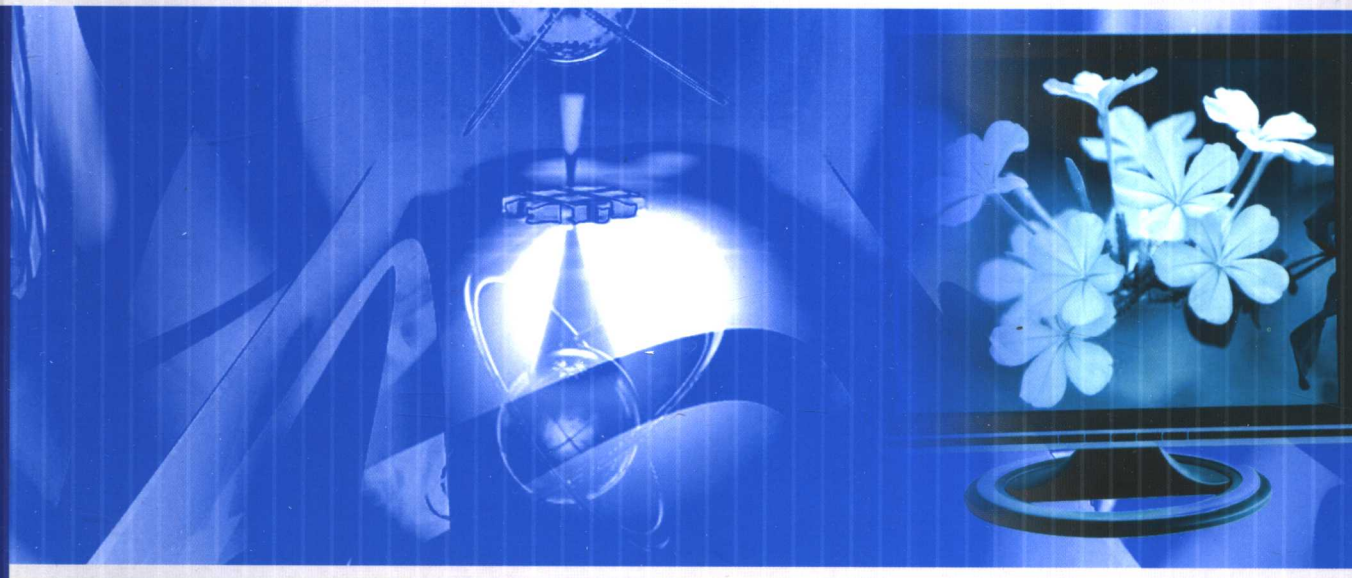
21 21世纪全国高校应用人才培养信息技术类规划教材



软件动态演化技术

李长云　何频捷　李玉龙　编著

北京大学出版社
PEKING UNIVERSITY PRESS

21 世纪全国高校应用人才培养信息技术类规划教材

软件动态演化技术

李长云　何频捷　李玉龙　编　著

北京大学出版社
PEKING UNIVERSITY PRESS

内 容 简 介

为了适应 Internet 开放环境和用户需求的不断变化，软件系统需要不断调整自身。软件动态演化技术是满足这一变化的有效手段，也是自治计算、网格计算、自适应软件和网构软件的核心技术。本书是国内外第一本对软件动态演化技术进行系统阐述的著作。作者结合多年研究和实践的经验，从开放系统发展动力到动态演化技术产生，从动态演化技术基本原理到动态演化的形态和粒度，从动态配置技术到基于反射的动态演化、动态演化的基础设施，从设计可动态演化的软件系统到动态演化技术的应用以及未来发展趋势，都进行了系统的论述。本书最后部分介绍了作者提出的软件动态演化模型 SASM 及其支持工具和平台，望起到抛砖引玉的作用。

本书内容全面、叙述清楚，注意一些最新的协议、规范及学术界、工业界研究进展，同时还非常注重实用性。本书既适用于本科高年级和研究生的教学，也可供工程技术人员自学参考之用。

图书在版编目（CIP）数据

软件动态演化技术/李长云，何频捷，李玉龙编著. —北京：北京大学出版社，2007.11
（21 世纪全国高校应用人才培养信息技术类规划教材）
ISBN 978-7-301-12989-0

I. 软… II. ①李…②何…③李… III. 软件工程－高等学校－教材 IV. TP311.5

中国版本图书馆 CIP 数据核字（2007）第 192065 号

书　　名：软件动态演化技术
著作责任者：李长云　何频捷　李玉龙　编著
责 任 编 辑：卢英华
标 准 书 号：ISBN 978-7-301-12989-0/TP・0918
出　版　者：北京大学出版社
地　　址：北京市海淀区成府路 205 号　100871
电　　话：邮购部 62752015　发行部 62750672　编辑部 62765126　出版部 62754962
网　　址：http://www.pup.cn
电 子 信 箱：xxjs@pup.pku.edu.cn
印　刷　者：北京宏伟双华印刷有限公司
发　行　者：北京大学出版社
经　销　者：新华书店
787 毫米×1092 毫米　16 开本　18.75 印张　456 千字
2007 年 11 月第 1 版　2007 年 11 月第 1 次印刷
定　　价：38.00 元

前　言

“变化”是现实世界永恒的主题，只有“变化”才能发展。Lehman 认为，现实世界的系统要么变得越来越没有价值，要么进行持续不断的变化以适应环境的变化。杨芙清院士指出，软件是对现实世界中问题空间与解空间的具体描述，是客观事物的一种反映。现实世界是不断变化的，因此，变化性是软件的基本属性。特别是在 Internet 成为主流软件运行环境之后，网络的开放性和动态性使得客户需求与计算环境更加频繁地变化，导致软件的变化性和复杂性进一步增强。有研究指出，Internet 具备如下基本特征：1）网络的开放性；2）资源的动态性；3）计算环境的多样性；4）用户需求的多变性。考察当前的主流软件技术，可以发现目前的软件系统本质都是一种封闭、静态的软件体系框架，难以适应 Internet 开放、动态环境的要求。

为适应这种发展趋势，人们纷纷从不同的视角对软件理论、技术和方法进行研究，例如从软件形态方面，提出了网构软件概念；从网络资源整合方面，提出了网格计算的概念；从计算模式方面，提出了自治计算、自适应软件的概念，以及面向服务的体系结构（SOA），反射中间件等等。纵观所有这些新的概念、思想和理论，都牵涉到一个共同的课题：开放环境下软件系统的动态演化理论和应用研究。软件演化指的是软件进行变化并达到人们所希望的形态的过程，分为静态演化和动态演化两种类型。静态演化是指软件在停机状态下的演化；动态演化是指在软件系统投入执行之后，应新环境条件和需求状况的改变而进行的演化。在传统的软件工程体系中，软件生命周期概念所强调的是从需求提出到软件提交的整个开发过程的重要性，而对于提交使用之后的软件变化过程往往只采用“软件维护”（即静态演化）加以简单概括。这样一种软件生命周期概念对于处于静态封闭环境下的软件系统的开发是合适的，但对处于 Internet 开放、动态和多变环境下软件系统的开发则有局限性。因此，Internet 环境下软件的开发对运行阶段之后的演化过程（即动态演化）的关注是必需的，甚至显得更为重要。

另一方面，在许多重要的应用领域中，系统的持续可用性是一个关键性的要求，运行期间的动态演化可减少因关机和重启而带来的损失和风险。例如对于执行关键任务的一些软件系统，通过停止、更新和重启来实现维护和演化将导致不可接受的延迟、代价和危险。容错系统发生错误时，需要切换到备用冗余部分，确保服务可用。移动计算在环境改变时，需要相应调整计算构件，适应环境变化。诸如交通控制软件、电信交换软件、Internet 服务应用以及高可用性的公共信息系统必须以 7×24 的方式运行，但是又经常需要演化以适应外部环境的变化和满足客户的更多需求，于是它们只能在运行期间进行扩展和升级。此外，越来越多的其他类型的应用软件也提出了运行时刻演化的要求，在不必对应用软件进行重新编译和加载的前提下，为最终用户提供系统定制和扩展的能力。因此，一般的应用软件如果具有运行时修改的特性，将大大提高系统的自适应性和敏捷性，从而延长软件的生命周期，增强企业的竞争力。

目前围绕软件动态演化的相关理论、方法和技术基本还处于起步阶段。结合我们这几年的研究和实践，本书试图对已有动态演化研究成果作一个较系统的整理。从开放系统发展动力到动态演化技术产生，从动态演化技术基本原理到动态演化的形态和粒度，从动态配置技术到基于反射的动态演化、动态演化的基础设施，从设计可动态演化的软件系统到动态演化技术的应用以及未来发展趋势，都进行了较系统的论述。本书最后部分介绍了作者提出的软件动态演化模型 SASM 及其支持工具和平台。

李长云负责本书的统一组织和策划。第 1、2、5、6、9 章由李长云编写，第 3、8、10 章由何频捷编写，第 4、7 章由李玉龙编写，王志兵、梁爱兰、吴岳忠、李伟参与了书稿内容和文字的斟酌、修改、校对及绘图工作。在这里要特别感谢北京航空航天大学计算机学院的马世龙教授、国防科技大学计算机学院的王怀民教授、湖南大学计算机与通信学院的李仁发教授的宝贵意见和指导。本书的研究与写作得到国家自然科学基金“开放环境下的软件动态演化研究”、湖南省教育厅优秀青年科研项目“基于高阶 π 演算的动态体系结构语言研究”、湖南省学位与研究生教改课题“应用导向型的研究生专业课程教学新模式”和湖南工业大学重点教改课题“计算机基础实验教学示范中心建设”的资助。在此我们表示衷心的感谢。

由于作者的认知和水平所限，加上国内外对软件动态演化技术进行系统阐述的资料较少，没有太多可借鉴的经验，本书体系可能是不完整的、不尽合理的，资料的来源也可能不具完全的代表性，敬请读者批评指正。

编　者

2007 年 8 月

目　录

第1章 概 述

1.1 软件演化的基本概念

“变化”是现实世界永恒的主题，只有“变化”才能发展。Lehman 认为，现实世界的系统要么变得越来越没有价值，要么进行持续不断的演化变化以适应环境的变化。软件是对现实世界中问题空间与解空间的具体描述，是客观事物的一种反映。现实世界是不断演化的，因此，演化性是软件的基本属性。特别是在 Internet 成为主流软件运行环境之后，网络的开放性和动态性使得客户需求与硬件资源更加频繁地变化，导致软件的演化性和复杂性进一步增强。

1.1.1 软件演化、软件维护与软件复用

软件演化（Software Evolution）指在软件系统的生命周期内软件维护和软件更新的行为和过程。在现代软件系统的生命周期内，演化是一项贯穿始终的活动，系统需求改变、功能实现增强、新功能加入、软件体系结构改变、软件缺陷修复、运行环境改变无不均要求软件系统具有较强的演化能力，能够快速适应改变，减少软件维护的代价。

软件演化研究致力于寻找规则、识别模式。这种规则和模式，主要用于管理那些由应用程序的编程人员或者是维护人员所作的变更。这意味着软件演化研究与导致系统不一致的变化有关。软件演化是一个模糊而范围广泛的概念，普遍的理解认为演化就是修改软件，而且意味着需要一个新的开发周期，包括新的测试和生成阶段，这样对于资源消耗来说是很危险的。尤其当调试者不是软件的开发人员的时候，就显得更加复杂了。我们认为软件演化应致力于尽可能限制那些可能被影响的变化本身的性质。

从软件演化的概念来看，软件演化和软件维护有着密切联系，但二者又有本质区别。软件维护是对现有的已交付的软件系统进行修改，使得目标系统能够完成新的功能，或是在新的环境下完成同样的功能，主要是指在软件维护期的修改活动。而软件演化则是着眼于软件的整个生命周期，从系统功能行为的角度来观察系统的变化，这种变化是软件的一种向前的发展过程，主要体现在软件功能的不断完善。在软件维护期，通过具体的维护活动可以使系统不断向前演化。因此，软件维护和软件演化可以归结为这样一种关系：前者是后者特定阶段的活动，并且前者直接是后者的组成部分。

软件演化的过程，也是通过修改软件的组成成分以适应变化的过程。在这一过程中，通常我们会尽可能复用系统已有的部分，降低演化的成本和代价。因此支持软件演化的技术通常也包含了软件复用的部分技术，如基于构件的开发、构件复用技术等等。与软件复

用一样，软件演化也可能发生在时间、平台、应用三维上。

（1）时间维：软件以以前的软件版本作为新版本的基础，适应新需求，加入新功能，不断向前演化。

（2）平台维：软件以某平台上的软件为基础，修改其和运行平台相关的部分，运行在新的平台上，适应环境变化。

（3）应用维：特定领域的软件演化后应用于相近的应用领域。

尽管软件演化和软件复用具有如上的共同特性，但它们研究问题的出发点和手段目的都是不同的。软件复用的目的是避免软件开发的重复劳动，提高软件开发的质量和效率。而软件演化着眼于整个软件的生命周期，研究如何在较低的开发代价的情况下，延长软件的生命周期，提高软件适应改变的能力。

1.1.2 软件演化的分类

软件演化可基本上分为两种：静态演化和动态演化。

（1）静态演化（Static Evolution）：是指软件在停机状态下的演化。其优点是不用考虑运行状态的迁移，同时也没有活动的进程需要处理。然而停止一个应用程序就意味着中断它提供的服务，造成软件暂时失效。

（2）动态演化（Dynamic Evolution）：是指软件在执行期间的软件演化。其优点是软件不会存在暂时的失效，有持续可用性的明显优点。但由于涉及状态迁移等问题，比静态演化从技术上更难处理。

动态演化是最复杂也是最有实际意义的演化形式。动态演化使得软件在运行过程中，可以根据应用需求和环境变化，动态地进行软件的配置、维护和更新，其表现形式包括系统元素数目的可变性、结构关系的可调节性和结构形态的动态可配置性。软件的动态演化特性对于适应未来软件发展的开放性、动态性具有重要意义。

按照变更发生的时机，软件演化可分为以下几类。

（1）设计时演化：设计时演化是指在软件编译前，通过修改软件的设计、源代码，重新编译、部署系统来适应变化。设计时演化是目前在软件开发实践中应用最广泛的演化形式。

（2）装载期演化：装载期演化是指在软件编译后、运行前进行的演化，变更发生在运行平台装载代码期间。因为系统尚未开始执行，这类演化不涉及系统状态的维护问题。

（3）运行时演化：发生在程序执行过程中的任何时刻，部分代码或者对象在执行期间被修改。这种演化是研究领域的一个热点问题。

显而易见，设计时演化是静态演化，运行时演化是一种典型的动态演化，而装载期间的演化既可以被看作是静态演化也可以看作是动态演化，取决于它怎样被平台或提供者使用。事实上，如果是用于装载类和代码，那么装载期演化就是静态演化，因为它其实是类的映射，而实际的装载代码并没有改变；另一种可能是增加一个层，允许在运行时刻动态的装载代码和卸载旧的版本，这样，通过连续的版本来更换代码，最后实现系统的演化，变更本身也可以被认为是动态的演化机制。

另外，演化可以是预设的和非预设的。

（1）预设演化是可以被开发人员所预见的演化。例如，插件技术就是一种允许程序员和维护人员在不更改应用程序核心部分的前提下扩展系统功能的机制。其优点是它可以提供依赖于动态软件演化下简单的装载和卸载机制，同时也可以提供依赖于静态演化下的API；缺点是在一些可能的变更中缺少适应性。

（2）非预设演化是指不能被开发人员所预见的那些演化。变更必须得到语言或者是执行平台（在动态演化中）的支持。其优点是包含了更多可能的演化，缺点是仍然没有一个被普遍接受的演化解决方案。

我们更加关注于非预设的动态演化，也就是说我们想要得到一个在运行期间不受演化可能性限制的演化系统。

1.2　设计时演化

设计时演化是目前在软件开发实践中应用最广泛的演化形式。设计时演化在软件编译前，通过修改软件的设计、源代码，重新编译、部署系统来适应变化。目前有多种技术用来提高软件的设计时演化能力，如基于构件的开发（CBSD）、基于软件框架（Framework）的开发、设计模式（Design Pattern）等等。

1.2.1　设计模式对设计时演化的支持

设计模式是对经过实践检验的、好的设计经验的提炼和总结，它强调了在特定环境下对反复出现的设计问题的一个软件解，侧重于解决软件设计中存在的具体问题。

设计模式解决的核心问题与设计时演化一致，要解决软件如何适应变化的问题。各种设计模式在实际上都从不同侧面封装了变化，有效地提高了软件的设计时演化能力，表 1-1 列出了部分设计模式和它们所封装的变化。

表 1-1　设计模式对设计时演化的支持

变　化	设 计 模 式
实现算法	Visitor，Strategy
用户接口操作	Command
对象接口	Adapter
对象实现	Bridge
对象之间的交互	Mediator，Facade，Proxy
对象创建过程	Abstract Factory，Factory Method，Prototype
对象结构建立过程	Builder
对象结构	Composite
遍历算法	Iterator
对象行为	State，Decorator
操作执行过程	Template
对象依赖关系	Publish-Subscribe

1.2.2 构件技术对设计时演化的支持

构件是用来构建软件系统的可复用的软件元素。构件通常表现为不同的形态，形态的差异体现在结构的组织方式和所依赖的软件开发方法。

按照构件的复用方式，构件可以分为以下两种。

- 源代码构件：这是最常见的构件形式，如结构化编程中的函数；面向对象编程中的类、类树；包（Package）等。
- 二进制构件：构件以编译后代码的形式提供，如 ActiveX 控件、EJB 构件、Java 类文件等。
- 按照应用领域，构件可分为以下三种。
- 通用基本构件：这是计算机系统的共性构成成分，如基本的数据结构、用户接口元素等，它们可以存在于各种应用系统中。
- 领域共性构件：这是应用系统所属领域的共性构成成分，它们存在于该领域的各个应用系统中。
- 应用专用构件：这是每个应用系统的特有构成成分。

基于构件的软件开发（Component-Based Software Development，CBSD）是一种利用可复用的软件构件建立应用系统的技术。这些构件由三部分组成：构件实现功能说明；构件的设计实现；构件的开发接口。

CBSD 的基本思路就是通过利用已有构件或自我组织开发的构件来组装应用程序。CBSD 基于 Software IC 的思想，强调软件开发的“组装”特性，即软件开发过程就是技术构件的构造、组织、选取、组装过程。可复用构件为有计划地进行复用提供了手段，是实现软件复用的基石，其生产和使用必须满足两个基本前提，即构件接口的标准化和构件的集成机制。目前从构件的定义和结构来说，已有比较成熟的构件规范，如 COM/DCOM、CORBA、EJB 等，这些规范为基于构件的软件开发、组装过程提供了完善的基础设施。

构件是现代软件系统的主要组成成分，是系统功能实现的载体，因此软件演化的基本单位就是构件演化，实际上构件演化也是软件演化的主要形式。支持构件设计时演化的主要技术如下。

1. 构件包装（Wrapper）

在系统开发过程中，经常出现集成的构件接口与系统设计需要的接口不兼容的情况，常见问题包括接口方法名称不一致、参数类型不一致，这就要求对原构件进行演化。对于这类接口演化问题，通常采用的做法是使用构件包装器（Component Wrapper）。构件包装器（如图 1-1）内封装了原始构件，同时提供了系统需要的接口，这样就解决了构件接口不兼容的问题。

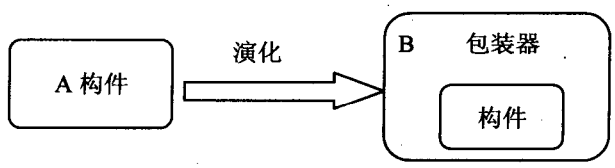


图 1-1 包装器将构件的接口 A 演化为接口 B

构件包装器本身不做核心工作，它只是进行简单的处理后，调用原始构件合适的方法，执行客户的请求，因此这是一种简单有效的构件演化的方法。

构件包装器不仅可以包装单个构件，而且可以包装多个构件，即同时封装多个构件。通过复合多个构件的功能，包装器可以提供更加强大的功能，此时包装器对于包装器的客户，实际本身也已经成为一个构件。一个封装了两个构件功能的包装器的实现如图 1-2 所示。

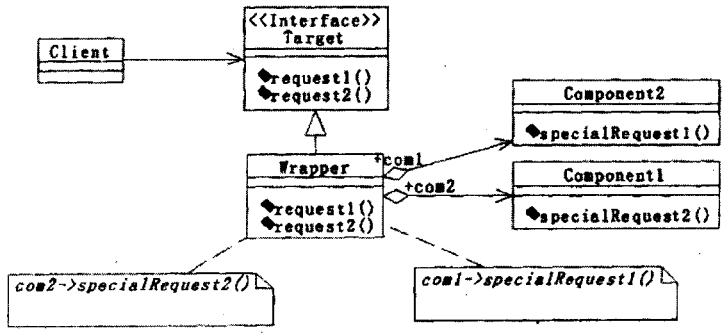


图 1-2　封装了两个构件的构件包装器的实现

2. 基于继承机制的构件演化

基于继承机制的构件演化，是一种广泛使用的构件演化技术。继承（Inheritance）和多态（Polymorphism）是对象建模技术适应变化的强有力武器。通常需要演化的构件，将自己可能需要演化的部分定义为虚函数，并提供默认实现。在新的环境中，当需要对原构件进行演化时，通过创建继承自原构件的子构件，并按照要求重新实现部分虚函数，就可以达到构件演化的目的。

如一个打印构件，负责实现系统数据的打印输出，其输出过程由三部分构成。

beforePrint()：初始化打印机。

print()：打印文档。

afterPrint()：负责打印结束后的后期处理。

对于不同的构件使用环境，打印机的初始化和后期处理可能都不相同。通过将beforePrint()和 afterPrint()定义为虚函数，针对不同的环境，开发不同的子构件，就有效实现了打印构件的演化，具体实现如图 1-3 所示。

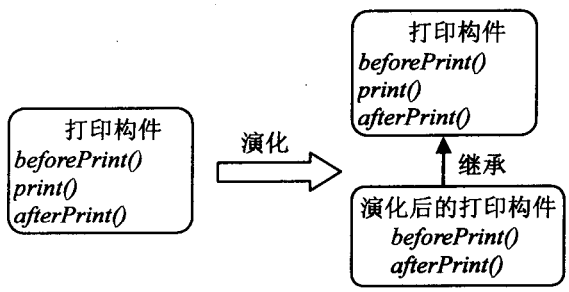


图 1-3　基于继承机制实现打印构件的演化

1.2.3 框架技术对设计时演化的支持

框架（Framework）可以看作一种大粒度的构件，其中包含若干个普通构件，定义了构件之间的协作规约，以实现一个大的系统功能。同时框架也是一个软件产品的半成品，它含可变和不变部分，通过对可变部分的定制得到不同的应用系统。对象框架的复用可以极大地降低应用开发的难度。通过复用一个好的框架可以得到一个设计简洁高效、稳定的系统。

框架是应用系统或子系统的可重用复用设计，在面向对象的系统中，框架通常由一系列的抽象类和具体类组成，系统的行为由抽象类的具体实现类的交互决定。

框架是比设计模式和构件更高层次、更大粒度的复用方式，成功的软件框架设计通常都包括了大量的设计模式（Design Pattern）和可复用构件（Reusable Component），开发者可以复用这些设计模式和构件开发自己具体的应用。

框架都是面向特定领域的，如用户接口设计或分布式通讯。表 1-2 列出了一些流行的软件框架。

表 1-2 一些面向特定领域的软件框架

应 用 框 架	简 介
JARS（Java Authentication and Authorization Service）	JARS 是一个基于 Java 的用户认证和授权安全框架，以可插拔的方式提供安全认证服务，独立于底层具体的安全认证实现
ACE（Adaptive Communication Environment）	ACE 是一个面向对象的基于 C++的分布式通讯框架，它提供了丰富的通讯模式和构件来开发跨平台的高性能分布式系统
Swing	Swing 是 Java 平台的 GUI 开发框架
MFC	MFC 是基于 Microsoft 的 WIN32 GUI 开发框架，它封装了 Win32 API，简化了 WIN32 应用程序的开发
QT	一个基于 C++的 GUI 开发框架，同时支持多个平台（Windows，Unix/Linux，Mac OS X，Embedded Linux）的应用系统开发

设计框架的主要目的是软件复用，框架是一种比构件更大粒度、更高层次的软件复用方式。从某种意义上，可以把框架看作一个大粒度的构件。和普通构件类似，使用环境的改变、软件技术的更新、系统功能的更新都要求框架处在不断的演化过程中。和普通构件不同的是，框架演化对兼容性有更高的要求，因为通常已经存在一些基于旧版本框架开发的软件系统，框架演化应该保证软件系统可以容易地移植到演化后的框架之上，以保护原有的投资。

软件框架按照对设计时演化支持方式的不同，可以分为白箱框架（White Box Framework）和黑箱框架（Black Box Framework）。

1. 白箱框架

白箱框架通常是框架开发经历的第一个阶段，这一阶段的主要特征是框架大量使用基于继承的构件演化技术。继承是框架支持设计时演化的主要手段，因为使用继承机制，需