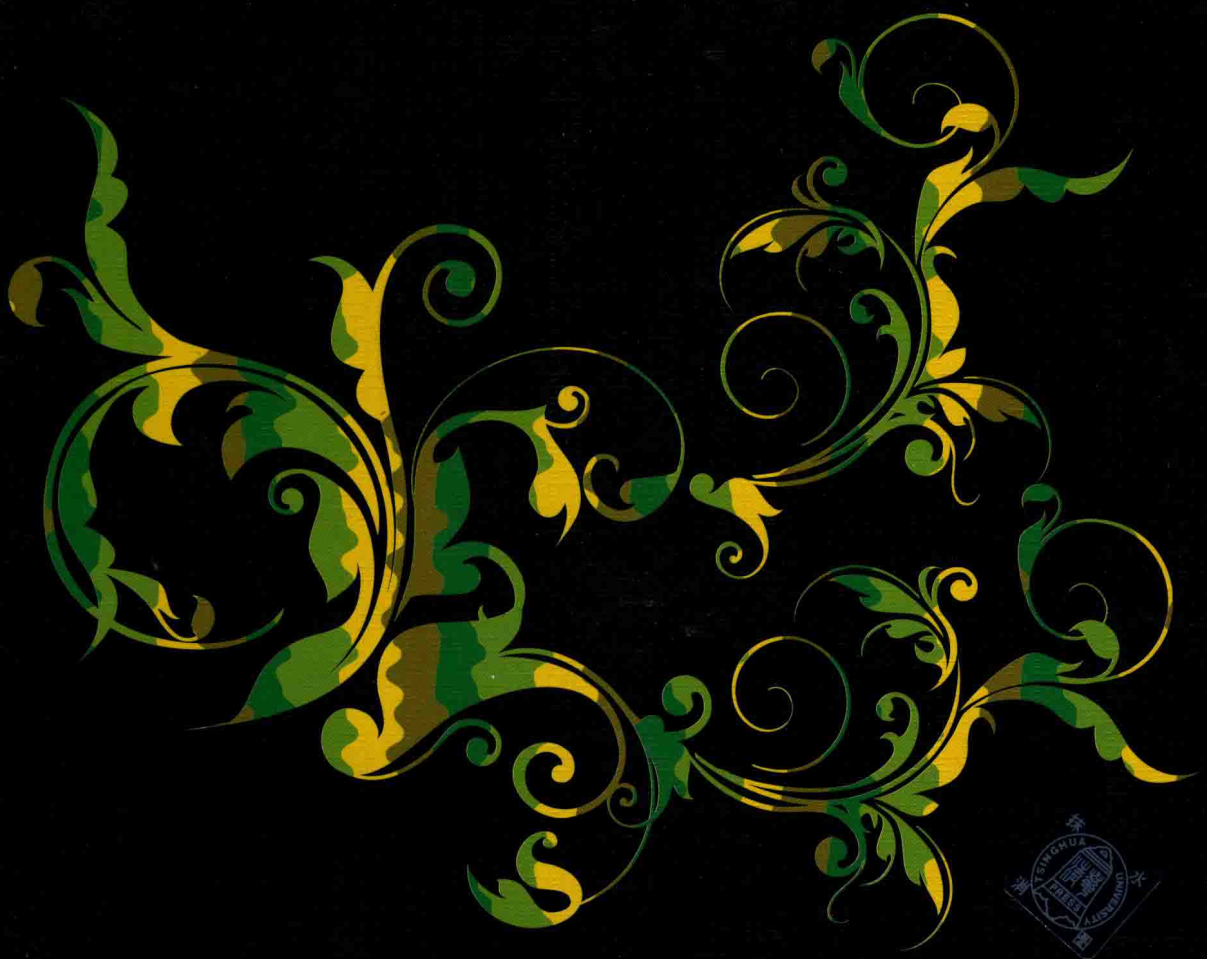
世界著名计算机教材精选

软件架构与模式

Joachim Goll 著

贾 山 等译



清华大学出版社

世界著名计算机教材精选

软件架构与模式

Joachim Goll 著

贾山 等译

清华大学出版社
北京

本书为德文版 ***Buide to Assembly Language: Aconcise Introduction*** 的简体中文翻译版，作者 **Joachim Goll**，由 **Springer** 出版社授权清华大学出版社出版发行。

北京市版权局著作权合同登记号　图字：01-2014-7914 号

图书在版编目（CIP）数据

软件架构与模式 /（德）乔希姆·戈尔（Joachim Goll）著；贾山等译. —北京：清华大学出版社，2017
书名原文：Architektur-und Entwurfsmuster der Softwaretechnik
（世界著名计算机教材精选）
ISBN 978-7-302-45099-3

Ⅰ. ①软…　Ⅱ. ①乔…　②贾…　Ⅲ. ①软件设计–教材　Ⅳ. ①TP311.5

中国版本图书馆 CIP 数据核字（2016）第 227185 号

责任编辑： 龙启铭　战晓雷
封面设计： 何凤霞
责任校对： 李建庄
责任印制： 宋　林

出版发行： 清华大学出版社
网　　址： http://www.tup.com.cn, http://www.wqbook.com
地　　址： 北京清华大学学研大厦 A 座　　**邮　　编：** 100084
社 总 机： 010-62770175　　**邮　　购：** 010-62786544
投稿与读者服务： 010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈： 010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载： http://www.tup.com.cn,010-62795954
印 刷 者： 清华大学印刷厂
装 订 者： 三河市溧源装订厂
经　　销： 全国新华书店
开　　本： 185mm×260mm　　**印　　张：** 20.75　　**字　　数：** 505 千字
版　　次： 2017 年 1 月第 1 版　　**印　　次：** 2017 年 1 月第 1 次印刷
印　　数： 1～2000
定　　价： 49.00 元

产品编号： 059060-01

译 者 序

随着中德两国交往的不断加深，各行各业都在不断地拓展多方位的合作。但是中德两国在软件行业的合作却并不多见，来自德国的计算机类翻译著作也非常少。

德国企业出于严谨的风格和安全性的考虑，基本很少有软件外包，对于应用软件的开发和使用一般也都局限在德国本土范围内（除了一些大型公司，如 SAP），所以我们对德国计算机行业的发展了解得并不多。

本书的作者 Goll 教授不仅有多年的计算机软件工作经验，同时还在德国 Esslingen 应用技术大学创建了软件专业，1994 年他还建立了 Steinbeis-Transferzentrum Systemtechnik 软件公司。因为该公司所在地是奔驰公司和博世公司的总部，所以主要从事汽车和自动化方向的软件开发。我在上学期间曾得到 Goll 教授的耐心指导。Goll 教授无论是在专业技术还是在教学业务上都是令人敬重和赞佩的。

希望这本书的引进能够使广大读者对德国的软件技术有初步的认识。本书共分为 5 章。前 3 章主要是介绍一些软件技术的理论。第 4 章介绍了常用的软件模式，具有一定的软件基础知识的读者通过学习本章的内容可以提高编写软件的质量，同时加深对软件理论知识的理解。第 5 章介绍了软件架构模式，读者在熟练掌握软件模式后，通过本章可以扩展视野，逐步了解大型软件开发所使用的架构模式。对于软件开发人员来说，通过学习本章的内容可以加快在专业方面的成长。

贾山翻译了本书的第 1 章、第 3 章和第 4 章，天津理工大学的李欣老师翻译了第 2 章和第 5 章。在本书翻译过程中，译者得到了清华大学出版社的大力支持和帮助，在此致以衷心的感谢。读者如果在阅读中有任何疑问，可以直接发送电子邮件到 zd_jiashan@126.com。

贾山

2016 年 10 月

前　言

本书的内容

软件系统的架构应该是易于扩展和标准化的，这样便于开发者对系统架构进行修改。在面向对象设计方法中，已经有很多有意义的架构和设计模式。这些模式都是建立在面向对象理论基础上的，例如依赖倒置原则。所以，本书首先介绍的是一些基本的原则，接下来讲解如何把这些面向对象的原则运用到系统架构和设计模式中。所有这些讲解都配有Java 语言的程序实例。

在讲解设计原则之后，本书将重点探讨系统架构和设计模式，通过附带的实例，读者可以从中选择适合自己系统的模式。书中的一些实例只截取了部分代码，完整的实例可以从相应的网站上查看。

本书可供计算机专业学生、工科学者、系统开发人员和大型系统的系统架构设计人员阅读。本书的目标是让读者掌握系统架构和模式的基本原理与实际应用。

书中的实例都是以 Java 语言为基础的。在讲解模式中类和类之间的静态关系或者是对象之间的动态关系时，均是借助于 UML 语言进行描述的。所以读者应该具备 Java 语言和 UML 语言的基础知识。

书中的图标表示对相应的内容做简短的总结。图标提醒读者，这是在实际开发过程中经常容易导致错误的地方。

每章附带简单的练习，书中没有提供答案，读者可以在相应的网站上查看。

本书的缘起

本书作者在此前的另一本书《软件技术的方法和架构》[Gol11][1]中的第 2 章曾经简短地介绍了设计和架构模式。在本书中，主要介绍这部分内容，不仅对这部分内容做认真的整理，而且还增加了面向对象设计的基本原理，因为这是设计模式的基础和加深理解模式所必需的。在吸收了模式后面隐藏的设计原理并重新修订以后，就形成了此书。

各章概要

下面简单地介绍各章的主要内容。

第 1 章讨论面向对象设计的基本原则，包括对于一个类的设计和多个类间合作的设计原则。一个类的设计原则有封装、抽象、信息隐藏、关注点分离、单一职责原则和接口隔

1　本书正文中提及的参考文献均以 5 字符或 6 字符代号表示，详见书末的“参考文献”。——译注

离原则。多个类的合作涉及松耦合原则、里氏代换原则、契约式设计原则、开闭原则和依赖倒置原则。随后分析了控制反转和在对象创建过程中减少相互依赖性，这两个是还没有形成“原则”的技术。

第 2 章关注软件架构概念的定义和软件架构关于非功能性的质量。软件设计中的参考架构和模式的相互比较。在分析和讨论构建一个系统的主要任务、软件架构的不同层次和结构模型以后，再研究软件架构师对一个项目的意义。

第 3 章研究架构模式、设计模式和惯用法的每一个特性，最后介绍描述设计模式和架构模式的模板。

第 4 章研究面向对象设计模式。面向对象设计模式多用于在软件开发中解决子系统中的特定问题。设计模式由类组成，通过类之间的互相协作解决特定的问题。每一个模式适用于一类问题的解决。本章将介绍结构模式、行为模式和创建型模式。在结构模式中讲解适配器模式、桥梁模式、装饰模式、外观模式、组合模式和代理模式。在行为模式中研究模板方法模式、命令模式、观察者模式、策略模式、中间者模式、状态模式、角色模式、拜访者模式和迭代器模式。创建型模式包括工厂方法模式、抽象工厂模式、单例模式和对象池模式。

第 5 章介绍架构模式。架构模式可以把系统划分为系统组件。一个架构模式可以含有多个设计模式，也可以不含有设计模式，例如分层架构模式就不包含设计模式。本章介绍分层架构模式、管道和过滤器架构模式、插件架构模式、中介模式、面向服务的设计模式和模型-视图-控制器（MVC）模式。

书写格式

本书中重要的概念加粗显示。

相应网址的重要提示

本书相应的网址为 http://pan.baidu.com/s/1o6MEsqu，包含各章练习答案和书中的实例。

感谢

作者在编写本书的过程中，从实例到文字得到了很多人的帮助。他们是 Benjamin Adolphi 先生、Sebastian Bickel 先生、Manuel Gotin 先生、Konstantin Holl 先生、Dominic Kadynski 先生、MichaKoller 先生、Paul Krohmer 先生和 Philipp Stehle 先生。Steffen Wahl 先生、Christian Tolk 先生、Fabian Wirsum 先生和 Jennifer Rauscher 女士对文字处理和资源配置做了长期大量的细致工作。作者在此表示衷心的感谢。

J. Goll/M. Dausmann

书中涉及的概念

派生类

一个派生类是从一个基类派生出来的。派生类继承了基类的结构（属性的名称和类型）和方法。

依赖性

依赖性是指两个模型元素间的特殊关系。它表明，一个独立元素的变化会影响到相关的元素。

抽象窗口工具包（AWT）

AWT 类库是 Swing 类库的前身。AWT-GUI 组件与操作系统密切相关。Swing-GUI 组件则借助于 Java 2D 的类库，通过 Java 的模拟机显示在屏幕上，所以看起来操作系统没有依赖性。

抽象类

抽象类不能构造实例。从技术上讲，一个抽象类包含一个或多个方法的定义，但是在抽象类里并没有实现这些方法。没有实现的方法只能在其派生类中实现。

抽象类也可以是不包含抽象方法的。这种抽象类只是为表达自身，不能被实例化。例如，一个“电器”或“饮料”不能被实例化。

抽象

抽象就是去掉每一个不重要的细节，而只专注于重点内容。

聚集

聚集是一种特殊的组合，它表达的是部分与整体的关系，说明部分与整体的生存期没有关联。这一点与组合（composition）正好相反。

参与者

参与者是指一个陌生的系统或者一个设备在系统环境中的角色。参与者与系统之间是相互作用的。

用例图（use case）

用例图是描述系统可以提供的服务，包括实现这些功能的步骤。用例图的功能通常由协作图（对象之间的交互）实现。

架构

架构可以理解为：

- 把系统分解为模块。

- 描述如何通过模块间的合作实现目标功能。
- 描述结构的策略，即分解（静态的）和功能（动态的）。

软件架构设计的目标就是让系统对外能实现所期望的功能。

关联

关联描述了连接的规则，用于连接两个或多个对象（对象是同级别的）。关联从原则上说是一个类间的对称结构关系。人们可以把关联限制成单向的。

属性

属性的概念存在于类、对象、关联和数据库中。面向对象中的属性概念来自面向数据范例。这意味着：

- 属性属于类或对象。
- 一个关联类描述了一个关联的特性。
- 数据库中关系（表）的一列也称为属性。属性与列的取值并不相关，而只是列的标题。属性值是列里面具体的一个取值。

基类

基类处在继承的层次结构中，在被关注类的上方。

关系

两个或多个元素之间的关系是指这些元素之间存在的联系。关系有动态的和静态的。

分类器

分类器是来自 UML 元模型的一个概念。一般认为，如果 UML 的模型元素能够生成一个实例，它就是分类器。一个分类器正常情况下有一个结构和一个功能，例如抽象类是一个分类器。接口可以特殊地当作是分类器，因为接口没有属性，也就是说没有结构。

委托

委托是一种结构，一个对象收到一条信息后，并不是自己实现全部方法，而是把消息继续传递下去。

依赖倒置

一个对象依赖于另一个对象，使用依赖倒置后，依赖的方向会倒转，即依赖和被依赖的实体关系被倒转。

依赖倒置原则

依赖倒置原则是为了避免或者减少不同等级中模块间的依赖关系。这就要求一个高一级的类不能依赖于一个低一级的类。高一级的类应该聚集了一个接口或者是一个抽象类。这个抽象类或接口应该由高一级的类决定，低一级的类应该依赖于这个抽象类或接口。

图表

图表是观察系统的特定视角。

在 UML 中的图表大部分包含许多用线连接起来的节点。此外，还有时间图，用于模

拟电子的脉冲图表。

实体

实体在所要处理的问题框架内，具有一定的意义。它可以表示物品、生物或者草案。

实体对象

实体对象就是现实世界中物品、生物或者草案的抽象化表示。

设计模式（design pattern）

在针对一个问题的最佳解决方案中，类或对象通过相互合作所实现的功能，可以成为这类问题的解决方案。

事件

事件是控制流或者控制流的组合。系统应对事件做出反应。

框架

框架可以为正在开发的系统提供类，其派生类可以继承它的功能。框架可以决定应用的结构，即粗略的结构。框架定义类和对象的划分、各自的中心职责、类和对象的合作以及它们的控制流[Gam95]。

保密原则

保密原则确保类的对象中的内部和私有结构对外是不可以使用的，只有在类的内部才可以识别内部结构和方法的具体实现部分。接口和它的实现部分是分离的。对象中的数据只能通过接口中定义的方法来使用。

泛化

泛化与细分的方向相反。在泛化时，要在继承的层次结构的上端整理综合的属性，然后向下细分，泛化的属性也会被继承。在继承的层次结构中，向上是泛化，向下是细分。

业务流程

业务流程是专业技术方面的工作进程。它体现了为达到一个业务目标所采用的各个相关措施的结合。

分组

分组是多个元素的组合。在 UML 中存在包的模型元素。包并不是系统的组成部分，而是概念上的示意图，把元素清楚地按组进行组合。

标识

即使不同的对象拥有相同的属性值时也可以区分开它们，因为每个对象都有自己的标识。

惯用法

惯用法是在特定的程序语言中的模型，是在深层次的抽象水平上作为设计模式或架构模式，即可以用一种编程语言来实现设计模式，用来解决不适用于设计模式的特定技术

问题。

实例

实例是 UML 模型元素的具体表现形式。

实例化

生成类型的实例。

交互

交互是通过两个对象间传送消息或者是多个对象相互作用而体现出的动态关系。

进程间通信

进程间通信（IPC）是把进程间的通信看做联动机制。进程间通信的概念包括操作系统进程对操作系统进程的通信、线程对线程的通信。在一台计算机上要进行进程间通信，需要借助于这台计算机的操作系统进行信息交换。如果并行的进程分布在不同的计算机，则计算机之间必须可以使用计算机对计算机的通信。

控制反转

在控制流反转时，原先掌握控制流的程序要把控制权转交给其他的模块（大部分情况是可复用的模块），这样可以从轮询转换到事件驱动。一个可复用的模块（例如框架）常常调用专有模块（例如应用程序）。

通道

通道表示点到点的连接，用直线来表示。通道遵守先进先出原则，即先进入通道的先离开。

封装

封装是面向对象程序设计中的重要概念之一。一般认为把数据和所属的功能放在一个盒子里（类）是十分有意义的，这样数据和所属功能没有被分开。

基数

基数的概念在数据库和 UML 中都存在：

- 在数据库领域，基数在系统分析阶段确定实体与实体之间数量的对应关系，或者在系统设计阶段确定表与表的数据记录数量的对应关系。基数可以依照关系类型标明 1 对 1、1 对 n 或者 n 对 m。
- 在 UML 中基数表明所关注的 UML 元素的个数。借助于基数可以表示出多倍的关系（例如，1..m）。

类

类在面向对象中体现的是一种数据类型，它可以生成对象。类含有结构和功能。结构包括属性。类里面的方法决定了其对象的功能。类的每一个对象都有自己的标识。

类图

类图展示的是类与类之间的静态关系（关联、泛化、实现或依赖）。

协作

协作由一系列对象组成，它们通过合作实现在应用场景下的功能。对象的协作可以实现一个功能。

通信图

通信图是在二维空间里展示所关注的对象、对象间的连接和沿连接实现的信息在对象间的交换。通过把对象分组和对象间的合作，尽管对象的连接是从动态角度出发的，也可以清晰地看出对象的结构组织。

模块

模块的概念体现在系统划分和模块技术中：

- 模块[1]是系统的组成部分，也就是划分的产物。
- 模块在模块技术中表示它是系统模块化的一部分。模块功能的具体实现隐藏在模块对外的接口内。与类相对比，调用模块的所有方法都必须通过接口。模块可以稳定地安装在计算机上。模块在实际使用中往往包含一个或多个类、接口或小的模块。

模块模型

模块模型在软件技术中描述的是由模块组成的系统运行时的框架。模块模型定义了箱体，单模块系统在箱体里运行（运行环境），同时还定义了模块对箱体的接口和对其他模块的接口。对此已经定义了很多模块的结构，比如模型的持久性、安全性和版本管理。

组合

组合是关联的一个特例，它描述的是整体与其局部的关系。局部的存在和整体的存在相关联，一个局部只属于一个唯一的整体。

具体类

具体类可以实例化，即具体类可以生成对象。

构造函数

构造函数是一个特殊的方法。构造函数与类名相同，在生成对象时被调用，为实例化服务。构造函数没有返回值类型，而且不能被继承。

控制对象

控制对象并不是现实世界的实体。控制对象相当于一个流程控制。

生命线

生命线存在于 UML 的时序图和通信图中。对象在通信图中作为对话者可以在 UML 中被描述为生命线。在 UML 的时序图和通信图中的生命线之间可以实现信息交流。

方法

一个方法实现一个操作。

---

1 这样的模块应该有接口，模块间通过接口可以实现松耦合。模块的内部应该被隐藏。

中间件

中间件是程序的一个层次，它可以扩展延伸到多台计算机，服务于分布式应用的进程间通信。中间件还有其他功能，如持久化服务、信息安全功能或名称管理。

多重性

多重性是对集合可能采取的基数范围的说明。

消息

对象之间可以通过消息进行沟通，接收者负责对消息进行解释。

导航

通过导航可以观察哪些对象是可以访问的（这些对象的类之间存在联系）。导航就是说明连接一端的对象是否能够到达连接另一端的对象。对于直接可以导航的就不需要绕路访问了。

并行

如果事件 A 和 B 可以互不依赖地独自工作，就称为是并行的。也就是说，“A 先发生，然后 B 发生”和“B 先发生，然后 A 发生”是等效的。

注释

注释是在 UML 中对一个或者多个模型元素的注解。

对象图

对象图显示的是在一个特定时刻对象和对象之间的连接。

操作

一个操作在 UML 中表达一个抽象的方法。它包括名称、参数、返回值类型、前置条件、后置条件和操作的详细说明。一个操作将会在一个类中通过方法实现。操作的定义包括操作的名称，对应于所属方法的参数和返回值类型。

在操作的详细说明中，一般说明这个操作可以在不同的类中实现。一个操作通过详细说明拥有一个特殊的含义（语义）。这个含义对于所有通过方法实现这个操作的类是相同的。例如，一个操作可以通过“给出名称和属性值”来详细说明。在不同的类里如何用个数互不相同的属性来实现。

操作视图

操作视图只关注对用户重要的功能，不考虑系统管理员。

包

UML 中的包用作对子包的分组或模型元素的分组，如类、接口、用例等。包可以理解为命名空间，用于访问保护的单元，使组织结构更加清晰。

面板

面板是一种无形的对象容器，可以把图形组件分成不同的组。

范式

范式是一种思维模式。

持久性

对象的持久性意味着它的寿命超过了一个会话（session），因为它们存储在非易失性存储介质（如磁盘）中。

多态性

多态性意味着多样性。例如，一个子类型的对象可以看做是它的基类。

协议

通信双方进行通信的一组规则。

边界条件

边界条件经常称为限制，是必须起作用的条件。它可以正式或非正式地加以描述，可以由相邻的系统指定。

实现关系

实现关系是两个元素间（按照 UML 分类）的静态关系。其中一个元素对契约进行描述，另一个元素在实现过程中必须遵守这个契约。例如，一个类实现一个接口，在协作图中的实现用例都是实现关系。

角色

角色的概念有多重含义：

- 角色可以是代理人[1]。
- 角色可以实现一个接口。

在关联和角色设计模式中，每一个对象可以承载一个角色。对象可以实现角色定义的接口，对外实现角色的功能。

接口

接口就是操作的清单。操作可以是一个类别（类或组件）提供的服务。一个接口可以表达一个类或组件的所有操作或者一部分操作。实现接口的类或组件必须遵守接口中对操作的定义。

自我委托

一个对象的方法调用自己的另外一个方法。

时序图

时序图按照时间顺序描述对象间或对象与角色（系统角色）间信息的交换。

---

1 在从 UML 1.x 到 2.x 的升级过程中，实例的定义有所改变。在对一个类型的代表建模时，例如建立时序图，类型的代表并不是画成实例或对象，而是角色。这种情况就是角色可以代表一类事物，它可以代表很多具体的实例或对象。在 UML 1.x 版本中，一个类的代表在时序图中表示成实例。

序列化

序列化（或称为信号编集）是把方法调用的变化写入到信息中。

编程中的签名和 UML 中的签名

签名的定义在编程和 UML 中的用法不同：

（1）在 UML 中：导出接口包括方法名称、参数类型列表和返回值类型。

（2）在 Java 中：导出接口包括方法名称和参数类型列表。

会话（Session）

会话就是网络中两个地址化的单位利用建立起的连接进行数据交换。

涉众（Stakeholder）

涉众是对系统感兴趣的自然人、法人或者一组人。这种兴趣会影响到系统的软件和硬件需求。

结构

系统的结构是其静态布局。

子类

参见派生类。

表

表是类似结构的数据的综合。在关系数据库中也称为关系。

类型

类型有属性和操作，对于属性有取值范围。

继承

在面向对象中，一个类可以继承另外一个类或者是从一个类派生出来。通过继承，类自动拥有了基类的属性和方法，就是说它继承了基类的结构和功能。

继承的层次结构

通过类之间的继承关系，形成了层次结构：派生类从属于基类，或者说基类处于派生类的上一级。

在 Java 中，所有的类都是从类 Object 派生出来的，所以这个类在 Java 中处在继承层次结构的根部。

行为

系统的行为是其功能的表述。

连接

连接是关联的一个实例。

使用关系

人们要表达一个元素使用另外一个元素，可以在 UML 中通过 «use» 表明它们的依赖

关系。«use»关系就是通常所说的使用关系。使用关系表明一个元素的含义（语义）依赖于另外一个元素的含义（语义）。

状态图

按照 Harel 或者 Hatley/Pirbhai 的理论，UML 的状态图是状态转化图的一个变形。状态图由要详细描述的类元的状态（可能是区域[1]性的）和转移组成。状态可以是延迟、进入行为、退出行为和状态行为。转移描述引起状态变化的事件、条件以及类元在转化过程中的行为。在给反应式系统（reactive system）建模时需要状态图。

状态转化图

按照 Harel 或者 Hatley/Pirbhai 的理论，在状态转化图中系统的状态在建模时以图形化表示。图形借助于状态和转移（状态的变化）描述了一个自动装置。转移详细说明了触发事件、状态变化的条件以及由状态变化引起的行动。

1 区域用于描述并行性。

目　录