第一章 软件设计模式概述

1 软件设计模式的产生背景

“设计模式”这个术语最初并不是出现在软件设计中，而是被用于建筑领域的设计中。1977 年，美国著名建筑大师、加利福尼亚大学伯克利分校环境结构中心主任**克里斯托夫•亚历山大** (Christopher Alexander) 在他的著作《建筑模式语言：城镇、建筑、构造 (A Pattern Language: Towns Building Construction）中描述了一些常见的建筑设计问题，并提出了 253 种关于对城镇、邻里、住宅、花园和房间等进行设计的基本模式。

1979 年他的另一部经典著作《建筑的永恒之道》（The Timeless Way of Building）进一步强化了设计模式的思想，为后来的建筑设计指明了方向。

1987 年，**肯特•贝克**（Kent Beck）和**沃德•坎宁安**（Ward Cunningham）首先将**克里斯托夫•亚历山大**的模式思想应用在 Smalltalk 中的图形用户接口的生成中，但没有引起软件界的关注。

直到 1990 年，软件工程界才开始研讨设计模式的话题，后来召开了多次关于设计模式的研讨会。

1995 年，**艾瑞克•伽马**（ErichGamma）、**理查德•海尔姆**（Richard Helm）、**拉尔夫•约翰森**（Ralph Johnson）、**约翰•威利斯迪斯**（John Vlissides）等 4 位作者合作出版了《**设计模式**：可复用面向对象软件的基础》（Design Patterns: Elements of Reusable Object-Oriented Software）一书，在本教程中收录了 23 个设计模式，这是设计模式领域里程碑的事件，导致了软件设计模式的突破。这几位作者常被称为"四人组（Gang of Four）"，而这本书也就被称为"四人组（或 GoF）"书。。

直到今天，狭义的设计模式还是本教程中所介绍的 23 种经典设计模式。

另外，近来这一清单又增加了一些类别，最重要的是使涵盖范围扩展到更具体的问题类型。例如，Mark Grand 在 Patterns in Java: A Catalog of Reusable Design Patterns Illustrated with UML（即后述《**模式 Java 版**》一书）中增加了解决涉及诸如并发等问题

的模式，而由 Deepak Alur、John Crupi 和 Dan Malks 合著的 Core J2EE Patterns: Best Practices and Design Strategies 一书中主要关注使用 Java 2 企业技术的多层应用程序上的模式。

2 软件设计模式的概念与意义

有关软件设计模式的定义很多，有些从模式的特点来说明，有些从模式的作用来说明。本教程给出的定义是大多数学者公认的，从以下两个方面来说明。

2.1 软件设计模式的概念

软件设计模式（Software Design Pattern），又称设计模式，是一套被反复使用、多数人知晓的、经过分类编目的、代码设计经验的总结。它描述了在软件设计过程中的一些不断重复发生的问题，以及该问题的解决方案。也就是说，它是解决特定问题的一系列套路，是前辈们的代码设计经验的总结，具有一定的普遍性，可以反复使用。其目的是为了提高代码的可重用性、代码的可读性和代码的可靠性。

2.2 学习设计模式的意义

设计模式的本质是面向对象设计原则的实际运用，是对类的封装性、继承性和多态性以及类的关联关系和组合关系的充分理解。正确使用设计模式具有以下优点:

- 可以提高程序员的思维能力、编程能力和设计能力。

- 使程序设计更加标准化、代码编制更加工程化，使软件开发效率大大提高，从而缩短软件的开发周期。

- 使设计的代码可重用性高、可读性强、可靠性高、灵活性好、可维护性强。

当然，软件设计模式只是一个引导。在具体的软件开发中，必须根据设计的应用系统的特点和要求来恰当选择。对于简单的程序开发，可能写一个简单的算法要比引入某种设计模式更加容易。但对大项目的开发或者框架设计，用设计模式来组织代码显然更好。

2 .3 到底为什么要学设计模式

就 Java 语言体系来说，GOF 是 Java 基础知识和 J2EE 框架知识之间一座隐性的"桥"。

会 Java 的人越来越多，但是一直徘徊在语言层次的程序员不在少数，真正掌握 Java 中接口或抽象类的应用不是很多，大家经常以那些技术只适合大型项目为由，避开或忽略它们，实际中，Java 的接口或抽象类是真正体现 Java 思想的核心所在，这些你都将在 GoF 里领略到它们变幻无穷的魔力。

GoF 表面上好像也是一种具体的"技术"，而且新的设计模式不断在出现，设计模式自有其自己的发展轨道，而这些好像和 J2EE，.Net 等技术也无关！

实际上，**GoF 并不是一种具体"技术",它讲述的是思想**，它不仅仅展示了接口或抽象类在实际案例中的灵活应用和智慧，让你能够真正掌握接口或抽象类的应用，从而在原来的 Java 语言基础上跃进一步，更重要的是，GoF 反复向你强调一个宗旨：要**让你的程序尽可能的可重用。**

这其实在向一个极限挑战：软件需求变幻无穷，计划没有变化快，但是我们还是要寻找出不变的东西，并将它和变化的东西分离开来，这需要非常的智慧和经验。

而 GoF 的设计模式是在这方面开始探索的一块里程碑。

J2EE 等属于一种框架软件，什么是框架软件？它不同于我们以前接触的 Java API 等，那些属于 Toolkit(工具箱)，它不再被动的被使用，被调用，而是深刻的介入到一个领域中去，J2EE 等框架软件设计的目的是将一个领域中不变的东西先定义好，比如整体结构和一些主要职责(如数据库操作 事务跟踪 安全等)，剩余的就是变化的东西，针对这个领域中具体应用产生的具体不同的变化需求，而这些变化东西就是 J2EE 程序员所要做的。

由此可见，设计模式和 J2EE 在思想和动机上是一脉相承，只不过

1.设计模式更抽象，J2EE 是具体的产品代码，我们可以接触到，而设计模式在对每个应用时才会产生具体代码。

2.设计模式是比 J2EE 等框架软件更小的体系结构，J2EE 中许多具体程序都是应用设计模式来完成的，当你深入到 J2EE 的内部代码研究时，这点尤其明显，因此，如果你不具备设计模式的基础知识(GoF 的设计模式)，你很难快速的理解 J2EE。不能理解 J2EE,如何能灵

活应用?

3.J2EE 只是适合企业计算应用的框架软件，但是 GoF 的设计模式几乎可以用于任何应用！因此 GoF 的设计模式应该是 J2EE 的重要理论基础之一。

所以说，GoF 的设计模式是 Java 基础知识和 J2EE 框架知识之间一座隐性的"桥"。为什么说是隐性的?

因为很多人没有注意到这点，学完 Java 基础语言就直接去学 J2EE,有的甚至鸭子赶架，直接使用起 Weblogic 等具体 J2EE 软件，一段时间下来，发现不过如此，挺简单好用，但是你真正理解 J2EE 了吗？你在具体案例中的应用是否也是在延伸 J2EE 的思想?

如果你不能很好的延伸 J2EE 的思想，那你岂非是大炮轰蚊子，认识到 J2EE 不是适合所有场合的人至少是明智的，但我们更需要将 J2EE 用对地方，那么只有理解 J2EE 此类框架软件的精髓，那么你才能真正灵活应用 Java 解决你的问题，甚至构架出你自己企业的框架来。(我们不能总是使用别人设定好的框架，为什么不能有我们自己的框架？)

因此，首先你必须掌握 GoF 的设计模式。虽然它是隐性的，但不是可以越过的。

3 软件设计模式的基本要素

软件设计模式使人们可以更加简单方便地复用成功的设计和体系结构，它通常包含以下几个基本要素：**模式名称**、别名、动机、**问题**、**解决方案**、**效果**、结构、模式角色、合作关系、实现方法、适用性、已知应用、例程、模式扩展和相关模式等，其中最关键的元素包括以下 4 个主要部分：

1. 模式名称

每一个模式都有自己的名字，通常用一两个词来描述，可以根据模式的问题、特点、解决方案、功能和效果来命名。模式名称（PatternName）有助于我们理解和记忆该模式，也方便我们来讨论自己的设计。

2. 问题

问题（Problem）描述了该模式的应用环境，即何时使用该模式。它解释了设计问题和问题存在的前因后果，以及必须满足的一系列先决条件。

3. 解决方案

模式问题的解决方案（Solution）包括设计的组成成分、它们之间的相互关系及各自的职责和协作方式。因为模式就像一个模板，可应用于多种不同场合，所以解决方案并不描述一个特定而具体的设计或实现，而是提供设计问题的抽象描述和怎样用一个具有一般意义的元素组合（类或对象的 组合）来解决这个问题。

4. 效果

描述了模式的应用效果以及使用该模式应该权衡的问题，即模式的优缺点。主要是对时间和空间的衡量，以及该模式对系统的灵活性、扩充性、可移植性的影响，也考虑其实现问题。显式地列出这些效果（Consequence）对理解和评价这些模式有很大的帮助。

4 学习设计模式的方法

1. 带着问题去学习，看到一个设计模式的类图和时序图后，首先想一想为什么这样设计，这样设计相比我们普通的编程有什么优点，这样更容易理解一些。

2. 不要因为一开始学不会就放弃，可以先学容易的，当理解了一个模式后，会产生恍然大悟的感觉，一个设计模式的理解往往会带来对其它设计模式的更深刻理解。设计模式之间不一定非得顺序学习，可以穿插，还要反复会过来重新理解。

3. 第一次、第二次实在学不会没关系，在实际工作中写几个月或者几年的程序后再回过头来学，往往会有意想不到的收获。