

您的潜力，我们的动力

Microsoft
微软(中国)有限公司

C#面向对象设计模式纵横谈

1. 面向对象设计模式与原则

李建忠

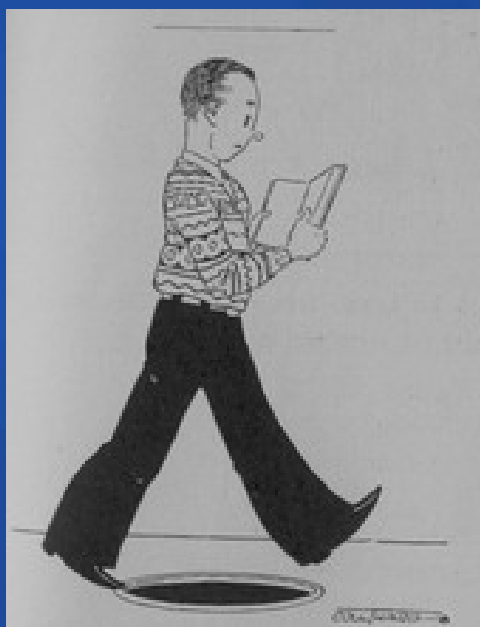
www.lijianzhong.com

上海祝成科技 高级讲师

您的潜力，我们的动力

Microsoft
微软(中国)有限公司

设计模式简介



每一个模式描述了一个在我们周围不断重复发生的问题，以及该问题的解决方案的核心。

——Christopher Alexander

设计模式描述了软件设计过程中某一类常见问题的一般性的解决方案。

面向对象设计模式描述了面向对象设计过程中、特定场景下、类与相互通信的对象之间常见的组织关系。

人是一个经验性的动物

GoF 23 种设计模式

- 历史性著作《设计模式：可复用面向对象软件的基础》一书中描述了23种经典面向对象设计模式，创立了模式在软件设计中的地位。该书四位作者被人们并称为**Gang of Four (GoF)**，“四人组”，该书描述的23种经典设计模式又被人们称为**GoF 23 种设计模式**。
- 由于《设计模式：可复用面向对象软件的基础》一书确定了设计模式的地位，人们通常所说的设计模式隐含地表示“面向对象设计模式”。但这并不意味“设计模式”就等于“面向对象设计模式”，也不意味着**GoF 23种模式**就表示了所有的“面向对象设计模式”。除了“面向对象设计模式”外，还有其他设计模式。除了**GoF 23种设计模式**外，还有更多的面向对象设计模式。
- **GoF 23 种设计模式**是学习面向对象设计模式的起点，而非终点；本培训课程的目标是让学员在建立在有效方法的基础上，掌握**GoF 23 种设计模式**。

设计模式与面向对象

- 面向对象设计模式解决的是“类与相互通信的对象之间的组织关系，包括它们的角色、职责、协作方式几个方面。
- 面向对象设计模式是“好的面向对象设计”，所谓“好的面向对象设计”是那些可以满足“应对变化，提高复用”的设计。
- 面向对象设计模式描述的是软件设计，因此它是独立于编程语言的，但是面向对象设计模式的最终实现仍然要使用面向对象编程语言来表达，本课程基于C#语言，但实际上它适用于支持.NET框架的所有.NET语言，如Visual Basic.NET、C++/CLI等。
- 面向对象设计模式不像算法技巧，可以照搬照用，它是建立在对“面向对象”纯熟、深入的理解的基础上的经验性认识。掌握面向对象设计模式的前提是首先掌握“面向对象”！

从编程语言直观了解面向对象

- 各种面向对象编程语言相互有别，但都能看到它们对面向对象三大机制的支持，即：“封装、继承、多态”
 - 封装，隐藏内部实现
 - 继承，复用现有代码
 - 多态，改写对象行为
- 使用面向对象编程语言（如**C#**），可以推动程序员以面向对象的思维来思考软件设计结构，从而强化面向对象的编程范式。
- **C#**是一门支持面向对象编程的优秀语言，包括：各种级别的封装支持；单实现继承+多接口实现；抽象方法与虚方法重写。

但OOPL并非面向对象的全部

- 通过面向对象编程语言（OOPL）认识到的面向对象，并不是面向对象的全部，甚至只是浅陋的面向对象。
- OOPL的三大机制“封装、继承、多态”可以表达面向对象的所有概念，但这三大机制本身并没有刻画出面向对象的核心精神。换言之，既可以用这三大机制做出“好的面向对象设计”，也可以用这三大机制做出“差的面向对象设计”。不是使用了面向对象的语言（例如C#），就实现了面向对象的设计与开发！因此我们不能依赖编程语言的面向对象机制，来掌握面向对象。
- OOPL没有回答面向对象的根本性问题——我们为什么要使用面向对象？我们应该怎样使用三大机制来实现“好的面向对象”？我们应该遵循什么样的面向对象原则？
- 任何一个严肃的面向对象程序员（例如C#程序员），都需要系统地学习面向对象的知识，单纯从编程语言上获得的面向对象知识，不能够胜任面向对象设计与开发。

从一个示例谈起

示例场景：

我们需要设计一个人事管理系统，其中的一个功能是对各种不同类型的员工，计算其当月的工资——不同类型的员工，拥有不同的薪金计算制度。

结构化做法

1. 获得人事系统中所有可能的员工类型
2. 根据不同的员工类型所对应的不同的薪金制度，计算其工资

```
enum EmployeeType{  
    Engineer;  
    Sales;  
    Manager;  
    ...  
}
```

```
// 计算工资程序  
If ( type==EmployeeType.Engineer) {  
    .....  
}  
else if (type== EmployeeType.Sales) {  
    .....  
}
```

面向对象设计

1. 根据不同的员工类型设计不同的类，并使这些类继承自一个 **Employee** 抽象类，其中有一个抽象方法 **GetSalary**。
2. 在各个不同的员工类中，根据自己的薪金制度，重写（**override**）**GetSalary** 方法。

```
abstract class Employee{  
    ...  
    public abstract int GetSalary();  
}
```

```
class Sales: Employee{  
    ...  
    public override int GetSalary() {  
        .....  
    }  
}
```

```
class Engineer: Employee{  
    ...  
    public override int GetSalary() {  
        .....  
    }  
}
```

```
// 显示工资程序  
Employee e=  
    emFactory.GetEmployee(id);  
  
MessageBox.Show( e.GetSalary());
```

现在需求改变了.....

示例场景：

随着客户公司业务规模的拓展，又出现了更多类型的员工，比如钟点工、计件工.....等等，这对人事管理系统提出了挑战——原有的程序必须改变。

结构化做法

几乎所有涉及到员工类型的地方（当然包括“计算工资程序”）都需要做改变.....这些代码都需要重新编译，重新部署.....

面向对象做法

只需要在新的文件里增添新的员工类，让其继承自Employee抽象类，并重写GetSalary()方法，然后在EmployeeFactory.GetEmployee方法中根据相关条件，产生新的员工类型就可以了。其他地方（显示工资程序、Engineer类、Sales类等）则不需要做任何改变。

重新认识面向对象

- 对于前面的例子，从宏观层面来看，面向对象的构建方式更能适应软件的变化，能将变化所带来的影响减为最小
- 从微观层面来看，面向对象的方式更强调各个类的“责任”，新增员工类型不会影响原来员工类型的实现代码——这更符合真实的世界，也更能控制变化所影响的范围，毕竟Engineer类不应该为新增的“钟点工”来买单.....
- 对象是什么？
 - 从概念层面讲，对象是某种拥有责任的抽象。
 - 从规格层面讲，对象是一系列可以被其他对象使用的公共接口。
 - 从语言实现层面来看，对象封装了代码和数据。
- 有了这些认识之后，怎样才能设计“好的面向对象”？
 - 遵循一定的面向对象设计原则
 - 熟悉一些典型的面向对象设计模式

从设计原则到设计模式

- 针对接口编程，而不是针对实现编程
 - 客户无需知道所使用对象的特定类型，只需要知道对象拥有客户所期望的接口。
- 优先使用对象组合，而不是类继承
 - 类继承通常为“白箱复用”，对象组合通常为“黑箱复用”。继承在某种程度上破坏了封装性，子类父类耦合度高；而对象组合则只要求被组合的对象具有良好定义的接口，耦合度低。
- 封装变化点
 - 使用封装来创建对象之间的分界层，让设计者可以在分界层的一侧进行修改，而不会对另一侧产生不良的影响，从而实现层次间的松耦合。
- 使用重构得到模式——设计模式的应用不宜先入为主，一上来就使用设计模式是对设计模式的最大误用。没有一步到位的设计模式。敏捷软件开发实践提倡的“Refactoring to Patterns”是目前普遍公认的最好的使用设计模式的方法。

几条更具体的设计原则


- 单一职责原则（SRP）：
 - 一个类应该仅有一个引起它变化的原因。
- 开放封闭原则（OCP）：
 - 类模块应该是可扩展的，但是不可修改（对扩展开放，对更改封闭）
- Liskov 替换原则（LSP）：
 - 子类必须能够替换它们的基类
- 依赖倒置原则（DIP）：
 - 高层模块不应该依赖于低层模块，二者都应该依赖于抽象。
 - 抽象不应该依赖于实现细节，实现细节应该依赖于抽象。
- 接口隔离原则（ISP）：
 - 不应该强迫客户程序依赖于它们不用的方法。

讲座总结


- 设计模式描述了软件设计过程中某一类常见问题的一般性的解决方案。面向对象设计模式描述了面向对象设计过程中、特定场景下、类与相互通信的对象之间常见的组织关系。
- 深刻理解面向对象是学好设计模式的基础，掌握一定的面向对象设计原则才能把握面向对象设计模式的精髓，从而实现灵活运用设计模式。
- 三大基本面向对象设计原则
 - 针对接口编程，而不是针对实现编程
 - 优先使用对象组合，而不是类继承
 - 封装变化点
- 使用重构得到模式。敏捷软件开发实践提倡的“Refactoring to Patterns”是目前普遍公认的最好的使用设计模式的方法。

Question & Answer

如需提出问题，请单击“提问”按钮并在随后显示的浮动面板中输入问题内容。一旦完成问题输入后，请单击“提问”按钮。

 **问题和解答 (无问题)** ▲ ×

在此会议中尚未解答任何问题。

要向演示者提问，请在此处键入问 

提问(A)

删除(D)

问题管理器(Q)

您的潜力，我们的动力

Microsoft®
微软(中国)有限公司

Microsoft®

msdn


MSDN Webcasts