NOTE FOR 设计模式

设计模式的分类

表1-1　设计模式空间

		目　的		
		创 建 型	结 构 型	行 为 型
范围	类	Factory Method(3.3)	Adapter(类)(4.1)	Interpreter(5.3) Template Method(5.10)
	对象	Abstract Factory(3.1) Builder(3.2) Prototype(3.4) Singleton(3.5)	Adapter(对象)(4.1) Bridge(4.2) Composite(4.3) Decorator(4.4) Facade(4.5) Flyweight(4.6) Proxy(4.7)	Chain of Responsibility(5.1) Command(5.2) Iterator(5.4) Mediator(5.5) Memento(5.6) Observer(5.7) State(5.8) Strategy(5.9) Visitor(5.10)

组合面向对象系统中的行为的方式:

1. 类继承；
2. 对象组合（比如委托）；
3. 参数化类型（比如模板类）；

可复用的面向对象设计原则:

)1. 针对接口编程，而不是针对实现编程。

)2. 优先使用对象组合，而不是类继承。

导致重新设计的原因，和解决其的设计模式

下面阐述了一些导致重新设计的一般原因，以及解决这些问题的设计模式：

1) 通过显式地指定一个类来创建对象　在创建对象时指定类名将使你受特定实现的约束而不是特定接口的约束。这会使未来的变化更复杂。要避免这种情况，应该间接地创建对象。

设计模式：Abstract Factory(3.1)，Factory Method(3.3)，Prototype(3.4)。

2) 对特殊操作的依赖　当你为请求指定一个特殊的操作时，完成该请求的方式就固定下来了。为避免把请求代码写死，你将可以在编译时刻或运行时刻很方便地改变响应请求的方法。

设计模式：Chain of Resposibility(5.1)，Command(5.2)。

3) 对硬件和软件平台的依赖　外部的操作系统接口和应用编程接口(API)在不同的软硬件

平台上是不同的。依赖于特定平台的软件将很难移植到其他平台上，甚至都很难跟上本地平台的更新。所以设计系统时限制其平台相关性就很重要了。

设计模式：Abstract Factory(3.1)，Bridge(4.2)。

4) *对对象表示或实现的依赖*　知道对象怎样表示、保存、定位或实现的客户在对象发生变化时可能也需要变化。对客户隐藏这些信息能阻止连锁变化。

设计模式：Abstract Factory(3.1)，Bridge(4.2)，Memento(5.6)，Proxy(4.7)

5) *算法依赖*　算法在开发和复用时常常被扩展、优化和替代。依赖于某个特定算法的对象在算法发生变化时不得不变化。因此有可能发生变化的算法应该被孤立起来。

设计模式：Builder(3.2)，Iterator(5.4)，Strategy(5.9)，Template Method(5.10)，Visitor(5.11)

6) *紧耦合*　紧耦合的类很难独立地被复用，因为它们是互相依赖的。紧耦合产生单块的系统，要改变或删掉一个类，你必须理解和改变其他许多类。这样的系统是一个很难学习、移植和维护的密集体。

松散耦合提高了一个类本身被复用的可能性，并且系统更易于学习、移植、修改和扩展。设计模式使用抽象耦合和分层技术来提高系统的松散耦合性。

设计模式：Abstract Factory(3.1)，Command(5.2)，Facade(4.5)，Mediator(5.5)，Observer(5.7)，Chain of Responsibility(5.1)。

7) *通过生成子类来扩充功能*　通常很难通过定义子类来定制对象。每一个新类都有固定的实现开销(初始化、终止处理等)。定义子类还需要对父类有深入的了解。如，重定义一个操作可能需要重定义其他操作。一个被重定义的操作可能需要调用继承下来的操作。并且子类方法会导致类爆炸，因为即使对于一个简单的扩充，你也不得不引入许多新的子类。

一般的对象组合技术和具体的委托技术，是继承之外组合对象行为的另一种灵活方法。

一般的对象组合技术和具体的委托技术，是继承之外组合对象行为的另一种灵活方法。新的功能可以通过以新的方式组合已有对象，而不是通过定义已存在类的子类的方式加到应用中去。另一方面，过多使用对象组合会使设计难于理解。许多设计模式产生的设计中，你可以定义一个子类，且将它的实例和已存在实例进行组合来引入定制的功能。

设计模式：Bridge(4.2)，Chain of Responsibility(5.1)，Composite(4.3)，Decorator(4.4)，Observer(5.7)，Strategy(5.9)。

8) *不能方便地对类进行修改*　有时你不得不改变一个难以修改的类。也许你需要源代码而又没有(对于商业类库就有这种情况)，或者可能对类的任何改变会要求修改许多已存在的其他子类。设计模式提供在这些情况下对类进行修改的方法。

设计模式：Adapter(4.1)，Decorator(4.4)，Visitor(5.11)。

这些例子反映了使用设计模式有助于增强软件的灵活性。这种灵活性所具有的重要程度取决于你将要建造的软件系统。让我们看一看设计模式在开发如下三类主要软件中所起的作用：应用程序、工具箱和框架。

设计模式所支持的设计的可变方面

表1-2　设计模式所支持的设计的可变方面

目　的	设计模式	可变的方面
创建	Abstract Factory(3.1)	产品对象家族
	Builder(3.2)	如何创建一个组合对象
	Factory Method(3.3)	被实例化的子类
	Prototype(3.4)	被实例化的类
	Singleton(3.5)	一个类的唯一实例
结构	Adapter(4.1)	对象的接口
	Bridge(4.2)	对象的实现
	Composite(4.3)	一个对象的结构和组成
	Decorator(4.4)	对象的职责，不生成子类
	Facade(4.5)	一个子系统的接口
	Flyweight(4.6)	对象的存储开销
	Proxy(4.7)	如何访问一个对象；该对象的位置
行为	Chain of Responsibility(5.1)	满足一个请求的对象
	Command(5.2)	何时、怎样满足一个请求
	Interpreter(5.3)	一个语言的文法及解释
	Iterator(5.4)	如何遍历、访问一个聚合的各元素
	Mediator(5.5)	对象间怎样交互、和谁交互
	Memento(5.6)	一个对象中哪些私有信息存放在该对象之外，以及在什么时候进行存储
	Observer(5.7)	多个对象依赖于另外一个对象，而这些对象又如何保持一致
	State(5.8)	对象的状态
	Strategy(5.9)	算法
	Template Method(5.10)	算法中的某些步骤
	Visitor(5.11)	某些可作用于一个（组）对象上的操作，但不修改这些对象的类