体系结构

**三要素：构件(Component)、连接件(Connector)和约束(Constraint)**

可预制和可重构

优势：容易理解、重用、控制成本、可分析性

体系结构风格

词汇表：构件连接件和约束

**数据流风格**:（批处理序列; **管道/过滤器**）

**调用/返回风格**：（主程序/子程序；面向对象风格；层次结构）

**数据抽象和面向对象组织** obj：manager抽象数据类型，封装数据的表示方法和操作 op：invocation对象通过函数和过程的调用来进行交互

**基于事件的隐式调用**：构件不直接调用一个过程，而是触发或广播一个或多个事件 **eg.观察者模式**

**分层系统**：每一层都为上一层提供了相应的服务，并且接受下一层提供的服务 **eg.通信协议**

独立构件风格：进程通讯；事件系统。？

虚拟机风格：解释器；基于规则的系统。？

**仓库风格**：数据库系统；超文本系统；黑板系统。

仓库系统：中心数据结构+独立构件

**过程控制环路**：输入、加工、输出、反馈、再输入

C/S风格

**组成=数据库服务器+客户应用程序+网络**

**优点：具有强大的数据操作和事务处理能力、（大的应用处理任务分步到低成本计算机上）节约费用**

**缺点：开发成本高、客户端程序设计复杂、软件移植困难、软件维护和升级困难**

三层C/S风格

组成部分：数据库服务器、**应用服务器**、客户应用程序和网络

三层C/S风格优点：提高系统和软件的**可维护性和可扩展性**、具有良好的**可升级性和开放性**、应用的**各层可以并行开发**、有效地**隔离开表示层与数据层**，为严格的**安全管理**奠定了坚实的基础。

三层C/S风格缺点：三层C/S结构各层间的**通信效率不高**，必须慎重考虑三层间的通信方法、通信频率及数据量。

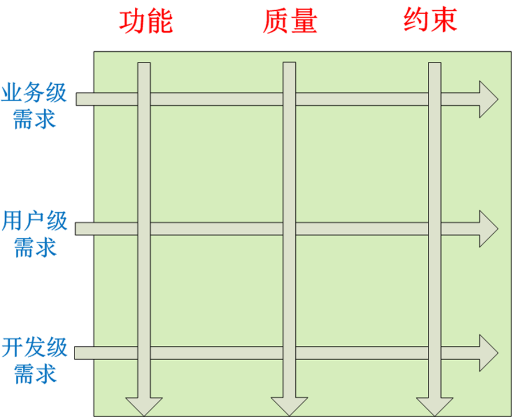
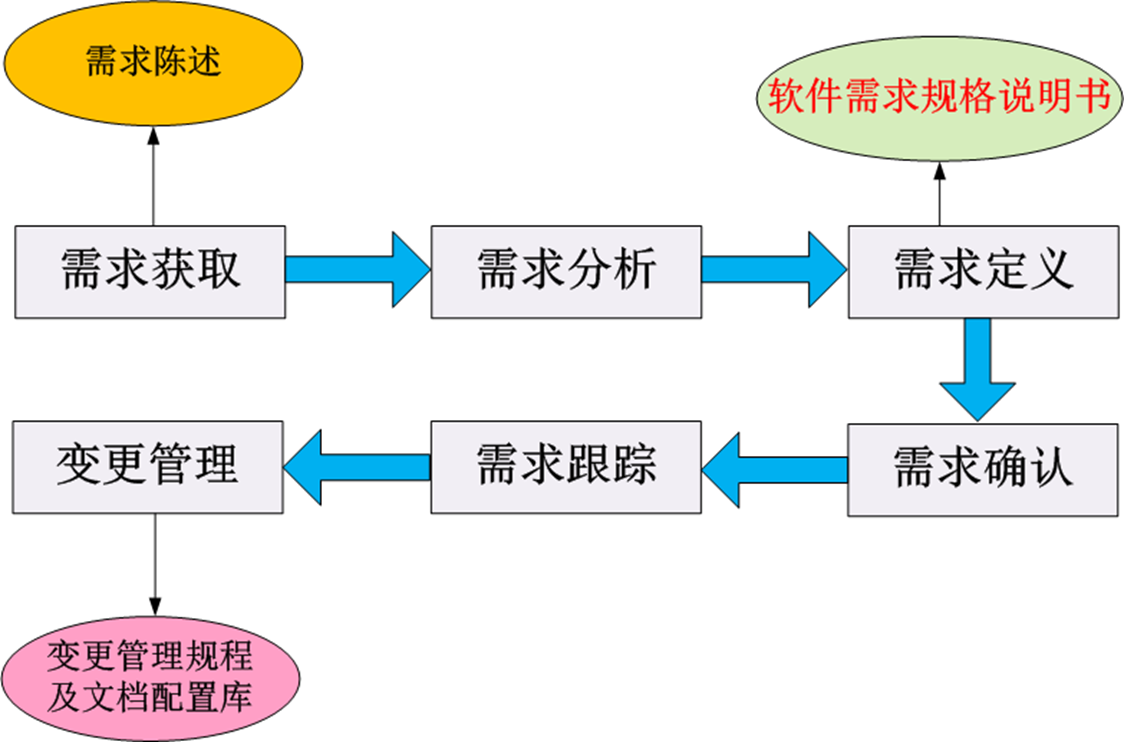
B/S

结构为：**浏览器/Web服务器**/数据库服务器。

B/S风格优点：系统安装、修改和维护全在服务器端解决，达到了**“零客户端”**的功能；提供了**异种机、异种网、异种应用服务器**的联机、联网、统一服务的最现实的开放性基础。

B/S风格缺点：缺乏对**动态页面**的支持能力，没有集成有效的数据库处理功能、系统**扩展能力差**，**安全性难以控制**、**响应速度低**于C/S体系结构、不利于**在线事务处理(**OLTP)应用。

软件需求与架构

****

三层：**业务需求、用户需求、系统需求**

三类：**功能需求、非功能需求、约束**

获取需求的方法

面谈（访谈）

问卷调查

会议（需求讨论会、重点问题讨论会、

业务专题讨论会、设计专题讨论会）

文档研究

任务示范（观察）

用例与角色扮演

原型设计（小规模试验）

研究类似公司

软件需求规格说明书(Software Requirement **Specification**, SRS)

属性：正确、清楚、无二义性、一致、必要、完备、可实现、可验证、确定优先级、阐述做什么而不是怎么做

需求跟踪技术

两种方式：正向跟踪、逆向跟踪

跟踪矩阵：

源跟踪矩阵（需求与需求来源）

功能跟踪矩阵（需求与功能）

依赖跟踪矩阵（一个需求与另一个需求）

需求基线

通过正式评审和批准的规格说明或产品，

被明确和固定下来的需求集合

**UML**

**用例图(Use Case Diagram)**

**使用用例的方法来描述系统的功能需求**

**状态图(State Diagram)**（考察单个对象的行为）

**描述一个特定对象的所有可能状态及其引起状态转移的事件。**

组成元素：

**初始状态**（Initial State）：

**终止状态**（Final State）：

**状态**（State）：

**转移**（Transition）：

**事件**（Event）：

**守护条件**（Guard Condition）：

**动作或活动**（Action）：

**活动图(Activity Diagram)**（考察跨用例或者跨线程的行为）

**表示系统中各种活动的次序，**既可用来描述用例的工作流程，也可以用来描述类中某个方法的操作行为。**一种描述工作流的方式。**

组成元素：起始活动(Start Activity)、终止活动(End Activity)、活动(Activity)、转移(Transition)或流(Flow)、判定、守护条件(Condition) [条件]、同步条(Synchronization)、泳道(Swimlane)

**顺序图（sequence diagram）**（考察单个用例内部多个对象的行为）

**一般用于确认和丰富一个使用情境的逻辑。**

纵向是时间轴，时间沿竖线向下延伸。横向轴代表了在协作中各独立对象的类元角色，类元角色的活动用生命线表示。

组成元素：

生命线：用一条纵向虚线表示

对象：对象表示为一个矩形

激活：是过程的执行，包括等待过程执行的时间。

在顺序图中激活部分替换生命线，使用长条的矩形表示。

消息：是对象之间的通信，是两个对象之间的单路通信，

是从发送者到接收者之间的控制信息流。

消息在顺序图中由有标记的箭头表示，

交互片断：一个复杂的顺序图可以划分为几个小块，每个小块称为一个交互片段。

alt:多条路径，条件为真时执行。

opt:任选，仅当条件为真时执行。

par:并行，每一片段都并发执行。

Loop:循环，片段可多次执行。

critical:临界区，只能有一个线程对它立即执行。

**类图**

**使用需要出现在系统内的不同的类来描述系统的静态结构，**类图包含类和它们之间的关系，它描述系统内所声明的类，但它没有描述系统运行时类的行为。

数据流图(Data Flow Diagram)

以图形的方式描绘数据在系统中流动和处理的过程

ER建模用于对数据进行建模（概念模型）

包图（package diagram）

包是一种把元素组织到一起的通用机制，用于描述包与包之间的关系。

组件图 (Component Diagram)

组件图显示组件以及它们之间的依赖关系，它可以用来显示程序代码如何分解成模块或组件。

部署图(Deployment Diagram)

描述系统硬件的物理拓扑结构及在此结构上执行的软件。

**面向对象设计原则：**

**1、单一职责原则(Single Responsibility Principle, SRP)**

定义：一个对象应该只包含单一的职责，并且该职责被完整地封装在一个类中。

Every object should have a single responsibility, and that responsibility should be entirely encapsulated by the class.

**2、开闭原则(Open-Closed Principle, OCP)**

定义：一个软件实体应当对扩展开放，对修改关闭。

Software entities should be open for extension, but closed for modification.

**3、里氏代换原则(Liskov Substitution Principle, LSP)**

定义：所有引用基类（父类）的地方必须能透明地使用其子类的对象。

Functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it.

**4、依赖倒转原则(Dependence Inversion Principle, DIP)**

定义：要针对接口编程，不要针对实现编程

Program to an interface, not an implementation.

**5、接口隔离原则(Interface Segregation Principle, ISP)**

定义：客户端不应该依赖那些它不需要的接口。（用多个专门的接口，而不使用单一的总接口）

Clients should not be forced to depend upon interfaces that they do not use.

**6、合成复用原则（Composite Reuse Principle, CRP）**

定义：尽量使用对象组合，而不是继承来达到复用的目的。

Favor composition of objects over inheritance as a reuse mechanism.

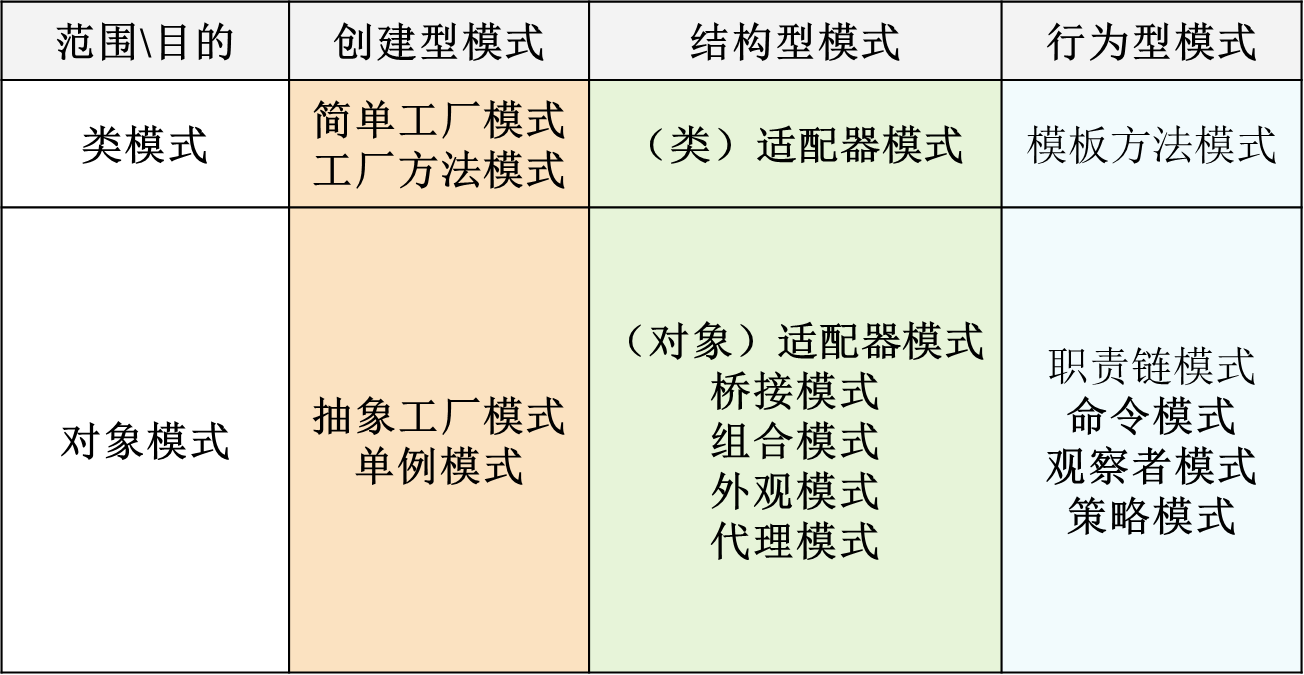
**7、迪米特法则(Law of Demeter, LoD)**

定义：一个软件实体应当尽可能少的与其他实体发生相互作用

创建型模式主要用于创建对象

结构型模式主要用于处理类或对象的组合

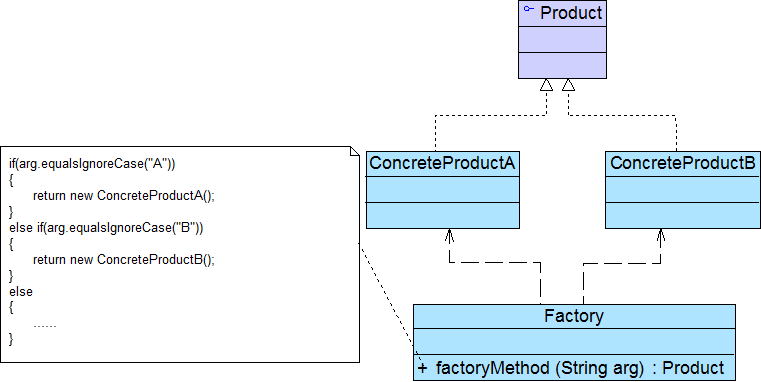
行为型模式主要用于描述类或对象如何交互和怎样分配职责



1. **创建型设计模式**

**1、简单工厂模式**

（1）类图



（2）优缺点

优点：分离类的创建和使用；客户端只需要知道具体产品类所对应的参数；可以在不修改任何客户端代码的情况下更换和增加新的具体产品类，提高系统灵活性。

缺点：工厂类的职责过重；会增加类的个数；不符合开闭原则，扩展困难；

由于使用静态工厂方法，难以形成基于继承的等级结构。

（3）适用范围

工厂类负责创建的对象比较少；客户端只知道传入工厂类的参数，对于如何创建对象不关心。

**2、工厂方法模式Factory Method Pattern**

定义一个用于创建对象的接口，但是让子类决定将哪一个类实例化。工厂方法模式让一个类的实例化延迟到其子类。

Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses。

（1）类图



（2）优缺点

优点：创建和使用分离；完全符合开闭原则

缺点：增加类的个数；增加代码理解难度

（3）适用范围

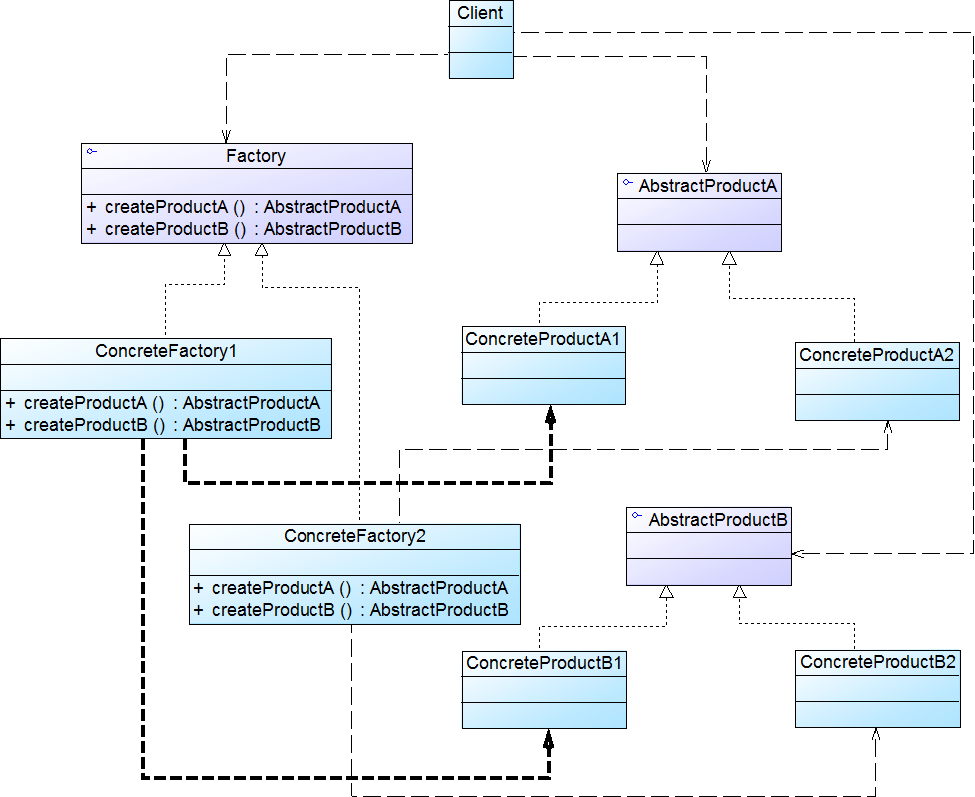
一个类不知道它所需要的对象的类；一个类通过其子类来指定创建哪个对象；将创建对象的任务委托给多个工厂子类中的某一个，客户端在使用时可以无须关心是哪一个工厂子类创建产品子类，需要时再动态指定。

**3、抽象工厂模式**

提供一个创建一系列相关或相互依赖对象的接口，而无须指定它们具体的类。抽象工厂模式又称为Kit模式，属于对象创建型模式。

**Abstract Factory Pattern**: Provide an interface for creating families of related or dependent objects without specifying their concrete classes.

（1）类图



（2）优缺点

优点：创建和使用分离；高内聚低耦合；够保证客户端始终只使用同一个产品族中的对象；

符合开闭原则

缺点：开闭原则的倾斜性（增加新的工厂和产品族容易，增加新的产品等级结构麻烦）

（3）适用范围

一个系统不应当依赖于产品类实例如何被创建、组合和表达的细节，这对于所有类型的工厂模式都是重要的。

系统中有多于一个的产品族，而每次只使用其中某一产品族。

属于同一个产品族的产品将在一起使用，这一约束必须在系统的设计中体现出来。

系统提供一个产品类的库，所有的产品以同样的接口出现，从而使客户端不依赖于具体实现。

**4、单例模式**

单例模式确保某一个类只有一个实例，而且**自行实例化**并**向整个系统**提供这个实例，这个类称为单例类，它提供全局访问的方法。（对象型创建模式）

Singleton Pattern: Ensure a class has only one instance and provide a global point of access to it.

（1）类图



（2）优缺点

优点：提供了对唯一实例的受控访问；节约系统资源，提高系统的性能；许可变数目的实例。缺点：扩展困难；违背了单一职责原则；

（3）适用范围

系统只需要一个实例对象。

客户调用类的单个实例只允许使用一个公共访问点，除了该公共访问点，不能通过其他途径访问该实例。

在一个系统中要求一个类只有一个实例时才应当使用单例模式。

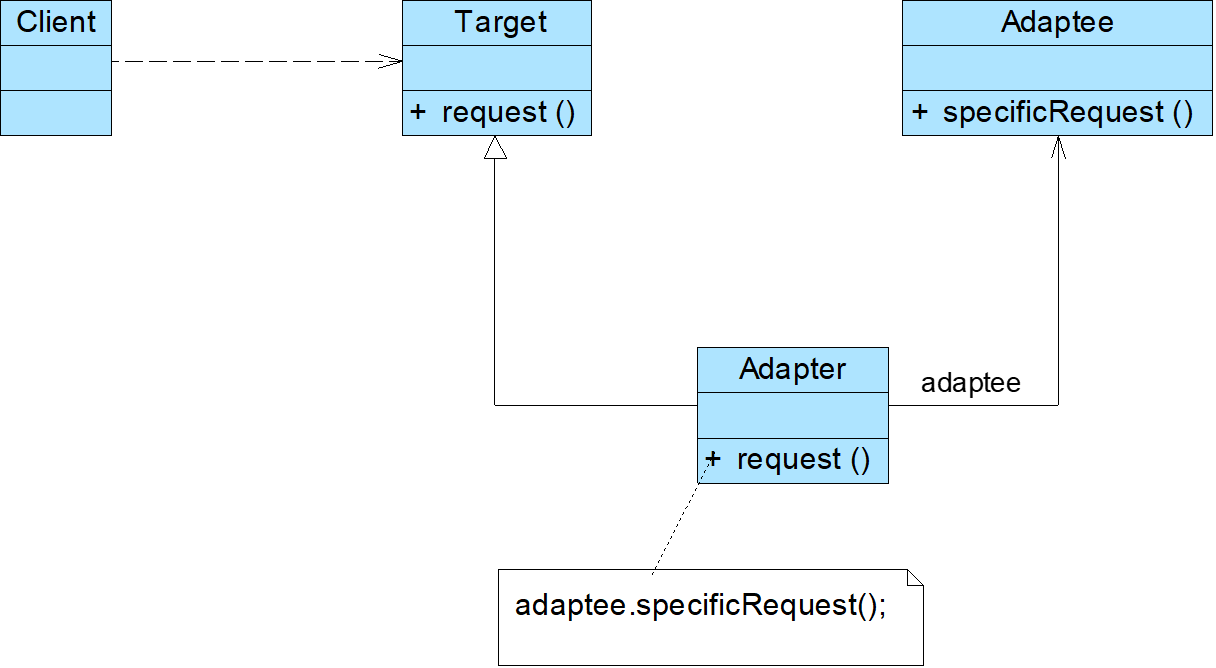
1. **结构型设计模式**

**1、适配器模式**

适配器模式(Adapter Pattern) ：将一个接口转换成客户希望的另一个接口，适配器模式使接口不兼容的那些类可以一起工作，其别名为包装器(Wrapper)。适配器模式既可以作为类结构型模式，也可以作为对象结构型模式。

Adapter Pattern: Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.

（1）类图



（2）优缺点

优点：将目标类和适配者类解耦；增加了类的透明性和复用性；符合开闭原则

类适配器模式还具有如下优点：

由于适配器类是适配者类的子类，因此可以在适配器类中置换一些适配者的方法，使得适配器的灵活性更强。

类适配器模式的缺点如下：

对于Java、C#等不支持多重继承的语言，一次最多只能适配一个适配者类，而且目标抽象类只能为抽象类，不能为具体类，其使用有一定的局限。

对象适配器模式还具有如下优点：

一个对象适配器可以把多个不同的适配者适配到同一个目标，也就是说，同一个适配器可以把适配者类和它的子类都适配到目标接口。

对象适配器模式的缺点如下：

与类适配器模式相比，要想置换适配者类的方法就不容易。如果一定要置换掉适配者类的一个或多个方法，就只好先做一个适配者类的子类，将适配者类的方法置换掉，然后再把适配者类的子类当做真正的适配者进行适配，实现过程较为复杂。

限性，不能将一个适配者类和它的子类都适配到目标接口。

1. 适用范围

系统需要使用现有的类，而这些类的接口不符合系统的需要。

想要建立一个可以重复使用的类，用于与一些彼此之间没有太大关联的一些类，包括一些可能在将来引进的类一起工作。

**1、桥接模式**

将**抽象部分**与它的**实现部分解耦**，使它们都可以独立地变化。

**Bridge Pattern**: Decouple an abstraction from its implementation so that the two can vary independently.

桥接模式将继承关系转换为关联关系，从而降低了类与类之间的耦合，减少了代码编写量。

模式优点

分离抽象接口及其实现部分

可以取代多层继承方案，极大地减少了子类的个数

提高了系统的可扩展性，在两个变化维度中任意扩展一个维度，不需要修改原有系统，符合开闭原则

会增加系统的理解与设计难度，由于关联关系建立在抽象层，要求开发者一开始就针对抽象层进行设计与编程

正确识别出系统中**两个独立变化的维度**并不是一件容易的事情

**2、组合模式**

组合多个对象形成**树形结构**以表示“整体-部分”的结构层次。（对象型）

**Composite Pattern**: Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.

（1）类图



（2）优缺点

优点：可以清楚地定义分层次的复杂对象，表示对象的全部或部分层次，使得增加新构件也更容易；客户端调用简单，客户端可以一致的使用组合结构或其中单个对象；更容易在组合体内加入对象构件，客户端不必因为加入了新的对象构件而更改原有代码。

缺点：使设计变得更加抽象；难以对容器中的构件类型进行限制。

（3）适用范围

在以下情况下可以使用组合模式：

需要表示一个对象整体或部分层次，在具有整体和部分的层次结构中，希望通过一种方式忽略整体与部分的差异，可以一致地对待它们。

对象的结构是动态的并且复杂程度不一样，但客户需要一致地处理它们。

安全组合模式、透明组合模式（父类中有访问管理成员方法）

**3、外观模式**

为**子系统中的一组接口**提供一个**一致的界面**，外观模式定义了一个**高层接口**，这个接口使得这一子系统更加容易使用。（对象型）

**Facade Pattern**: Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.

（1）类图



（2）优缺点

优点：屏蔽子系统组件，减少了客户端的对象数目并使得子系统使用起来更加容易；实现了子系统与客户之间的松耦合关系；一个子系统的修改对其他子系统没有任何影响，而且子系统的内部变化也不会影响到外观对象。

缺点：不能很好地限制客户使用子系统类；违背开闭原则

（3）适用范围

当要为一个复杂子系统提供一个简单接口时可以使用外观模式。

客户程序与多个子系统之间存在很大的依赖性。

在层次化结构中，可以使用外观模式定义系统中每一层的入口，层与层之间不直接产生联系，而通过外观类建立联系，降低层之间的耦合度。

符合单一职责原则、迪米特法则

**4、代理模式**

**给某一个对象提供一个代理，并由代理对象控制对原对象的访问。**（对象型）

**Proxy Pattern**: Provide a surrogate or placeholder for another object to control access to it.

（1）类图



（2）优缺点

优点：降低了系统的耦合度；增加和更换代理类无须修改源代码，符合开闭原则

远程代理：可以将一些消耗资源较多的对象和操作移至性能更好的计算机上，提高了系统的整体运行效率

虚拟代理：通过一个消耗资源较少的对象来代表一个消耗资源较多的对象，可以在一定程度上节省系统的运行开销

缓冲代理：为某一个操作的结果提供临时的缓存存储空间，以便在后续使用中能够共享这些结果，优化系统性能，缩短执行时间

保护代理：可以控制对一个对象的访问权限，为不同用户提供不同级别的使用权限

缺点：有些类型的代理模式可能会造成请求的**处理速度变慢**；实现代理模式需要额外的工作，有些代理模式的**实现复杂**。

（3）适用范围

当客户端对象需要访问远程主机中的对象时可以使用远程代理

当需要用一个消耗资源较少的对象来代表一个消耗资源较多的对象，从而降低系统开销、缩短运行时间时可以使用虚拟代理

当需要为某一个被频繁访问的操作结果提供一个临时存储空间，以供多个客户端共享访问这些结果时可以使用缓冲代理

当需要控制对一个对象的访问，为不同用户提供不同级别的访问权限时可以使用保护代理

当需要为一个对象的访问（引用）提供一些额外的操作时可以使用智能引用代理

1. **行为型设计模式**

**2、命令模式**

将一个请求封装为一个对象，从而可用不同的请求对客户进行**参数化**；对请求排队或者记录请求日志，以及支持可撤销的操作。（对象型）

**Command Pattern:** Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.

（1）类图



（2）优缺点

优点：**降低系统的耦合度**；新的命令可以很容易地加入到系统中；可以比较容易地设计一个命令队列和宏命令（组合命令）；可以方便地实现对请求的Undo和Redo。

缺点：增加大量具体命令类的个数，影响命令模式的使用。

（3）适用范围

系统需要将请求调用者和请求接收者解耦，使得调用者和接收者不直接交互。

系统需要在不同的时间指定请求、将请求排队和执行请求。

系统需要支持命令的撤销(Undo)操作和恢复(Redo)操作。

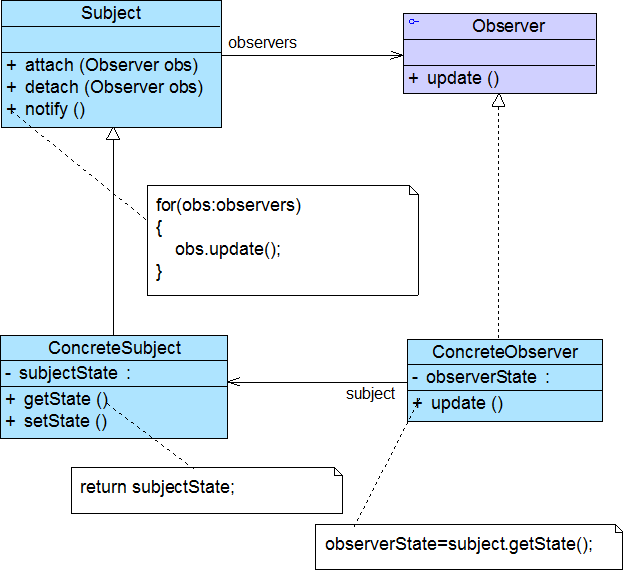
系统需要将一组操作组合在一起，即支持宏命令。

**3、观察者模式**

定义对象间的**一种一对多依赖关系**，使得每当一个对象**状态发生改变**时，其相关依赖对象皆**得到通知**并被**自动更新**。

**Observer Pattern**: Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

（1）类图



（2）优缺点

优点：表示层和数据逻辑层的分离；观察目标和观察者之间建立一个抽象的耦合；**支持广播通信；符合开闭原则**

缺点：如果一个观察目标对象有很多直接和间接的观察者的话，**时间花费多；可能存在循环依赖**的话，导致系统崩溃；仅知道发生变化**不知道如何变化**

（3）适用范围

抽象模型中存在依赖关系；一个对象的改变将导致其他不知道的对象也发生改变；一个对象必须通知其他不确定的对象；存在触发链

例：MVC模式

**4、策略模式**

定义一系列算法，将每一个算法封装起来，并让它们可以相互替换。（对象型）

**Strategy Pattern**: Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

（1）类图



（2）优缺点

优点：符合**开闭原则**；提供管理算法族的办法；替换继承关系；避免使用多重条件转移语句

缺点：客户端必须知道所有的策略类，并自行决定使用哪一个策略类；增加类的个数

（3）适用范围

一个系统需要动态地在几种算法中选择一种；避免使用多重的条件选择语句；不希望客户端知道与算法相关的数据结构，提高算法的保密性与安全性。

**4、模板方法模式**

定义一个操作中算法的骨架，而将一些步骤延迟到子类中，模板方法使得子类可以不改变一个算法的结构即可重定义该算法的某些特定步骤（类行为型:模板方法模式是一种类的行为型模式，在它的结构图中只有类之间的继承关系，没有对象关联关系。）

**Template Method Pattern**: Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure. 

**1、职责链模式**

避免请求发送者与接收者耦合在一起，让多个对象都有可能接收请求，将这些对象连接成一条链，并且沿着这条链传递请求，直到有对象处理它为止。

**Chain of Responsibility Pattern:** Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.

（1）类图



（2）优缺点

优点：降低耦合度；可简化对象的相互连接；增强给对象指派职责的灵活性；增加新的请求处理类很方便

缺点：不能保证请求一定被接收；影响系统性能；可能会存在循环调用。

（3）适用范围

在以下情况下可以使用职责链模式：

有多个对象可以处理同一个请求，具体哪个对象处理该请求由运行时刻自动确定。

在不明确指定接收者的情况下，向多个对象中的一个提交一个请求。

可动态指定一组对象处理请求。

纯的设计模式：一个请求有仅有一个对象处理者。