# 复习提要Unit1

## OO 方法和系统分析的理念

OO面向对象方法，把信息系统看作是一起工作来完成某项任务的相互作用的对象集合。

**元素与表示法**：

对象（类）

属性，操作

一般-特殊关系

整体与部分关系

关联关系

消息

包（package）

## OOA、OOD与OOP的区别

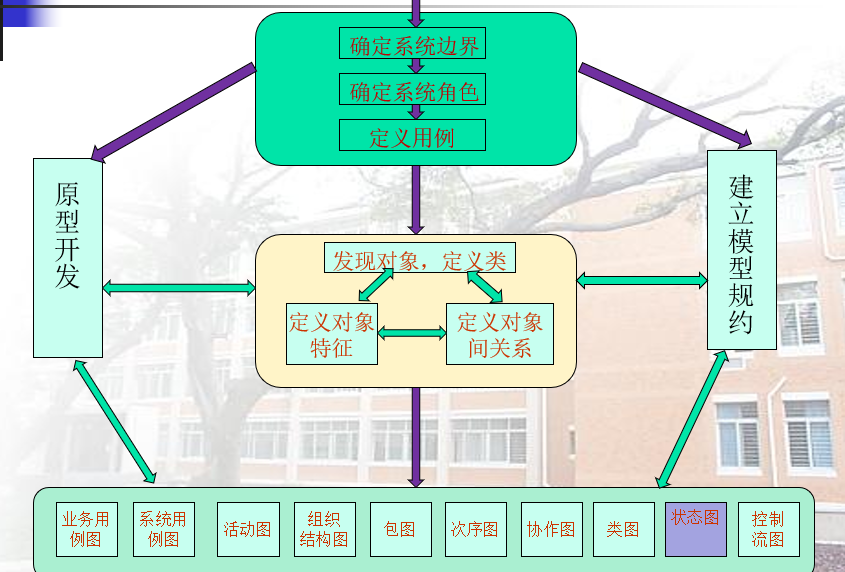
OOA是Object-Oriented Analysis(面向对象分析)

OOD是Object Oriented Design(面向对象设计)

OOP是Object Oriented Programming (面象对象程序设计)

参考：<https://blog.csdn.net/moxiaoya1314/article/details/51899048>

* OOA的主要原则
  + 抽象 abstract
  + 过程抽象和数据抽象
  + 分类
  + 封装 encapsulation
  + 继承 inheritance
  + 聚合
  + 关联
  + 消息通信 message
  + 粒度控制
  + 行为分析



## 软件开发出现的问题

用户或业务需求未得到满足

需求得不到解决

模块未集成

维护困难

缺陷的最新发现

最终用户体验质量差

负载下性能不佳

无需团队协作

构建和发布问题

## 最佳实践 DEV475\_01\_BestPractices

6种最佳实践原则：

迭代开发：瀑布模型和迭代模型

管理需求：

使用组件架构，

可视化建模，

持续验证软件质量，

管理变化

## 如何描述Complex System

## 熟悉软件生命周期

软件开发过程（英语：software development process），或软件过程（英语：software process），是软件开发的开发生命周期（software development life cycle），其各个阶段实现了软件的需求定义与分析、设计、实现、测试、交付和维护。软件过程是在开发与构建系统时应遵循的步骤，是软件开发的路线图。

## 分析设计的四大阶段

Inception，elaboration，construction，transition

初始阶段，定义项目设想

细化阶段，确定项目计划，细化特征和基础架构

构造阶段，构建项目

移交阶段，将项目交付给最终用户

## 影响软件项目成功与否的因素和起因

## 何为迭代式开发？Why use？

迭代式开发对部分系统及早引入编程和测试，并重复这一循环。这种方式通常会在还没有详细定义所有需求的情况下假设开发开始，同时使用反馈来明确和改进演化中的规格说明。迭代和进化式开发法 iterative and evolutionary development每次迭代都产生经过测试、集成并可执行的局部系统，即每个迭代周期产生一个可执行版本。

**优点**：

迭代方法减少失败可能性，提高生产率，降低缺陷率。

早期缓解高风险，在风险评估上，迭代开发在早期就可以将风险快速降低，相较而下瀑布模型的风险早期会持续比较久。

早期课件的进展。

早期反馈、用户参与和调整，产生更接近涉众真实需求的精化系统。

可控复杂性，不会陷入分析瘫痪。

一次迭代中的经验可以被系统地用于改进开发过程本身，如此反复进行下去。

## 什么是UML？Why use？

Unified Modeling Language 统一建模语言是描述、构造和文档化系统制品的可视化语言。UML仅仅是标准的图形化表示法，使用常用符号的可视化建模不能与设计和对象思想同等重要。

这可以从两个不同的角度来回答:UML定义本身以及如何使用它来产生项目工件。

* 每一个复杂的系统最好通过一个模型的一组几乎独立的视图来处理。没有一个单独的视图是足够的。
* 每个模型都可以表示为不同的保真度。
* 最好的模型是与现实相联系的。

创建可视化模型之后：

* 用户可以通过查看模型来可视化与系统的交互。
* 分析人员可以从模型中可视化对象之间的交互。
* 开发人员可以可视化需要开发的对象和每个对象需要完成的工作。
* 测试人员可以可视化对象之间的交互，并根据这些交互准备测试用例。
* 项目经理可以看到整个系统以及各个部分是如何交互的。
* 首席信息官可以查看高级模型，并查看组织中的系统如何相互交互。
* 总而言之，可视化模型为向所有相关方显示所提议的系统提供了一个强大的工具。

UML中一些概念：

•可扩展性机制(原型、标记值和约束)，

•线程和进程，

•分布和并发(例如，用于建模ActiveX/DCOM和CORBA)，

•/合作模式,

•活动图(用于业务流程建模)，

•细化(处理抽象层次之间的关系)，

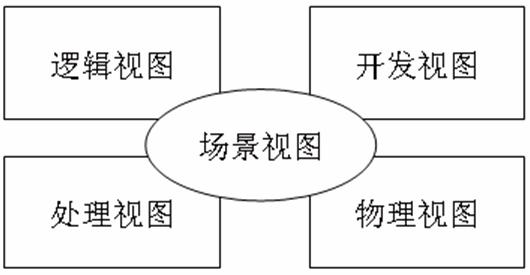
•接口和组件

•约束语言。

## 什么使UP，RUP

统一过程（RUP/UP，Rational Unified Process）是一种以用例驱动、以体系结构为核心、迭代及增量的软件过程模型，由UML方法和工具支持，广泛应用于各类面向对象项目。

## [RUP 4+1视图方法](#_分布式对软件结构的影响)



逻辑视图：当采用面向对象的设计方法时，逻辑视图即对象模型。

开发视图：描述软件在开发环境下的静态组织。

处理视图：描述系统的并发和同步方面的设计。

物理（部署）视图：描述软件如何映射到硬件，反映系统在分布方面的设计。

# 复习提要Unit2

## 自治系统的特征

## 状态集、状态转换、事件驱动

事件驱动的一般步骤：

1、确定响应事件的元素

2、为指定元素确定需要响应的事件类型

3、为指定元素的指定事件编写相应的事件处理程序

4、将事件处理程序绑定到指定元素的指定事件

对于事件驱动模式，对象包括事件类型、事件对象、事件分发器、事件处理器，每种事件都有对应的事件处理器编写于处理函数中，

事件分发器r是异步处理的事件的，处理逻辑中也可以包括发送下一个事件。

对于状态机模式，基本做法是

根据发生的事件和当前状态，来执行下一步的动作，并设置下一个状态，不同状态不同的事件处理方式，不同的转化状态。

## 消息、通讯协议、System Behavior

Message：

* 是用来请求对象执行某一处理或回答某些信息的要求。是对象的相互作用和相互联系。消息统一了数据流和控制流。程序的执行是通过对象间传递消息来实现的。它非常类似于传统的程序设计语言中子程序之间通过参数的方式调用。
* 消息的形式用消息模式来描述。
* 消息是从发送者到接受者的一条执行服务的请求。

**通讯机制；**

* 通讯语言－－协议；
* 同步－异步；
* 集中、协作；
* 主动机制。

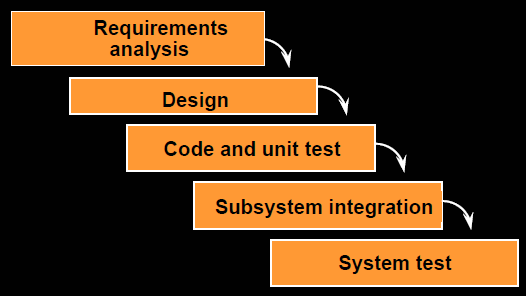
系统行为 system behavior：系统行为是一个系统的行为和反应，系统行为被用例捕捉。【用例描述系统，描述其环境以及系统和环境的关系】

## Event、Pre-conditions、Post-conditions（also see *object model、dynamic model、functional model in* Booch Notation、OOSE and OMT）

## 瀑布模型、放射性模型、迭代和进化式开发法

Waterfall development characteristics：

过程：需求分析，设计，编码和单元测试，子系统整合，系统测试



一般步骤分析需求，设计系统，开发系统，测试系统，部署系统。requirements, designed a system , developed the system , tested the system ,and deployed the system。

迭代和进化式开发法 iterative and evolutionary development开发产生可执行文件，每个迭代周期产生一个可执行版本。迭代方法有较高的成功率，生产率和低缺陷率。

在风险评估上，迭代开发在早期就可以将风险快速降低，而瀑布模型的风险早期会持续比较久。

## 需求分析 requirement overview

目的：

•与客户和其他利益相关者就系统应该做什么达成一致并保持一致。

•让系统开发人员更好地理解系统的需求。

•定界系统。

•为迭代的技术内容规划提供基础。

•提供一个评估系统开发成本和时间的基础。

•定义系统的用户界面。

与之相关的元件：[用例建模 use-case modeling](#_Use-Case_Modeling_、)，词汇表 glossary，补充性规格说明supplementary specification

与之相关的概念：

系统行为 system behavior：系统行为是一个系统的行为和反应，系统行为被用例捕捉。【用例描述系统，描述其环境以及系统和环境的关系】

## Use-Case Modeling 、 Business Model、System Model、Domain Model

用例建模 use-case modeling 中的一些概念：

[角色 Actor](#_角色（不同阶段）的类型和结构) 参与者表示与系统交互的任何东西。

用例 use case 用例是系统执行的一系列操作，这些操作会向特定的参与者产生一个可观察的值结果。

用例建模 use-case model 根据用例描述系统功能需求的模型，系统预期功能的模型(用例)及其环境(参与者)。

优点：方便沟通，识别，验证

规格说明：名称，简要描述，事件流，关系，活动图，用例图，特殊说明，前置条件，后置条件，其他图。

事件流：有一个正常的基本流程，几个可选流程：普通变化，奇怪的情况，特殊流来处理错误情况。

场景 scenario 场景是用例的一个实例。

## UC Analysis DEV475\_05\_UCAnalysis

用例分析 use case analysis

􀂊补充用例描述

􀂊对于每个用例：找到用例的行为，把行为合理分配给各个类

􀂊每个分析类：描述职责，属性和关联，限定分析机制

􀂊统一分析类

􀂊检查分析过程和结果

## UC Design DEV475\_09\_UCDesign

用例设计 use case design

􀂊描述设计对象之间的交互

􀂊用子系统简化序列图

􀂊描述持续相关行为

􀂊细化的事件流描述

􀂊统一类和子系统

## Software architecture、 Modeling architecture、 organization architecture 、role architecture

# 复习提要Unit3

## 角色（不同阶段）的类型和结构

参与者的类型，在所讨论系统中 system under discussion，SuD：

主要参与者：primary actor 具有用户目标，并通过使用SuD的的服务完成 - 发现驱动用例的用户目标

协助参与者：supporting actor 为SuD提供服务（如信息服务，存储服务，付费授权服务） - 为了确定外部接口和协议

幕后参与者： offstage actor 在用例行为中具有影响或利益，但不是主要或协助参与者 - 为了确保确定并满足所有必要的重要事物，如果不明确地对幕后参与者进行命名，有时候很容易忽略其影响或者利益

## 分析类-设计类的转化

## 子系统 subsystem的设计 DEV475\_10\_SubsystemDesign

描述子系统设计的目的和它在生命周期中的执行位置

定义中指定的行为子系统的接口包含类的协作

文档的内部结构子系统

确定依赖于元素的外部子系统

一个子系统是“cross”一个包和一个类，其实现一个或多个接口，定义其行为

设计子系统的**目标**：松散耦合，，可移植性、即插即用的兼容性，对变化绝缘，独立进化。

设计建议：不公开细节，只公开接口，只取决于其他接口

设计步骤：

1 将子系统行为分配给子系统元素：子系统职责由接口操作定义，接口操作可以通过内部类操作或内部子系统的操作的方式实现

2 定义子系统元素：

定义新的或重用已存在的设计元素，

将子系统职责分配给设计元素，

合并适用的机制（例如持久化，分布），

“接口实现”中的文档设计元素协作（1. 每个接口操作有一个或多个交互图；2. 类图包含设计元素之间的关系），

重新审查“设计识别元素”，如果有需要，调整子系统边界和依赖关系

3 描述子系统的依赖：依赖于另一个子系统，依赖于一个包

4 确认检查点。

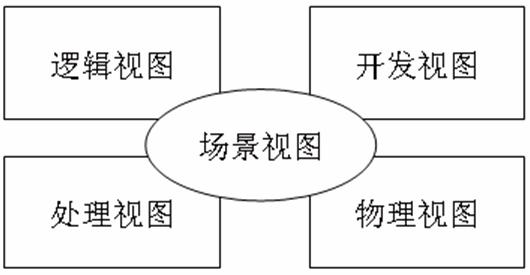
是否为子系统提供的每个接口定义了实现关联?

是否为每个子系统所使用的接口存在依赖关联?

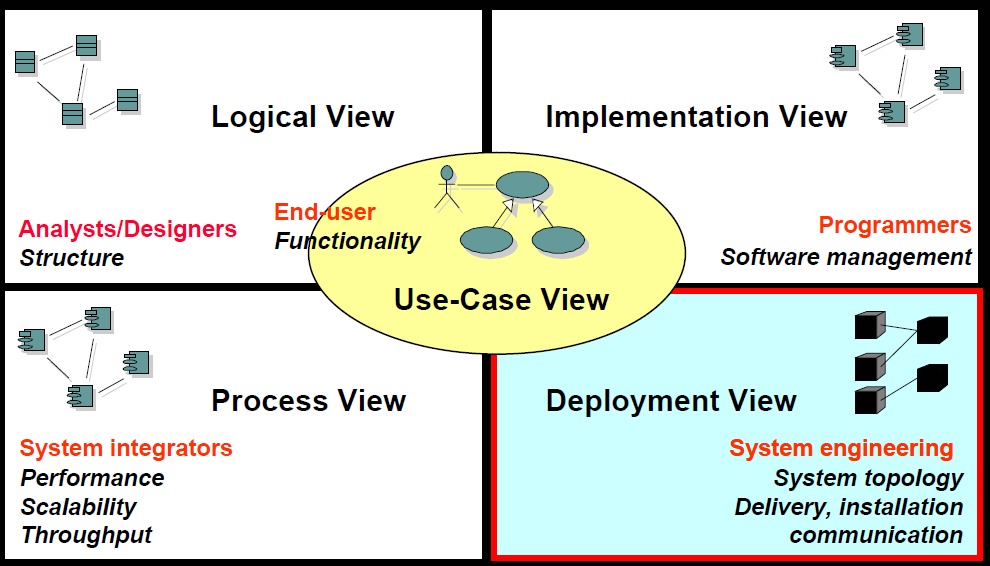
你确定子系统内的元素都没有公共可见性?

每个操作在一个接口实现的子系统交互图记录?如果没有，这个操作是由单个类实现的吗，这样就很容易看到类操作和接口操作之间有一个简单的1:1映射?

## 分布式对软件结构的影响 DEV475\_08\_Distribution



部署视图是部署模型的“架构上重要的”部分



**部署为何选择分布式**？

减少处理器负载，特殊处理要求，扩展问题，经济问题，分布式访问系统

**分布布局**：

C/S （3-tier 应用层-业务层-数据层， Fat client 应用业务层-数据层， Fat server， distributed）

P2P

**步骤**：

定义网络配置：终端用户工作站节点，“headless”处理服务器节点，特殊配置（开发，测试），专用处理器

分配流程节点：分布模式，响应时间和系统吞吐量，最小化的交叉网络流量，节点容量，通信介质的带宽，硬件和通信链路的可用性，重路由需求

## 并发对软件结构的影响

Concurrency 并发

**为什么关注并发**？

软件可能需要响应表面上随机的外部生成的事件

如果多个cpu并行执行任务可以提高性能，例子:启动系统

控制系统可以通过并发性增强

为了支持并发，系统必须提供多个控制线程。

常见的**并发机制**

􀂃多处理 multiprocessing

多个cpu并发执行

􀂃多任务 multitasking

•操作系统通过交叉执行不同的任务来模拟单个CPU上的并发性

􀂃基于应用程序解决方案 application-based solutions

•应用软件负责在适当的时候在不同的代码分支之间进行切换

## 多层软件体系结构

## 多视图描述的必要性和意义

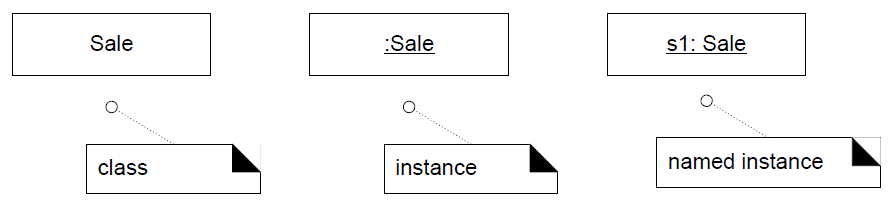
# 复习提要Unit4

## 从交互图分析系统性能[interaction diagram]

什么是交互图－动态行为

**顺序图 sequence diagram**

顺序图中栅栏中被指向的表示其拥有对应的方法，在非后产生的衍生类中表示其拥有其属性。其中在右侧添加新创建的对象。



执行者类、实例、对象、生命线[lifeline,表示交互的参与者]、消息[在垂直生命线之间带有实心箭头的实线，表示对象间的每个消息]、控制焦点[表示控制焦点的执行规格条，execution specification bar，或称为活动条或活动]

单实例类在生命线框图的右上角标识1。

其中最开始的消息称为创始消息found message，实心圆点作为起点，没有特定的发送者、发送者不明确或者消息源众多。

实例的创建中，新创建的对象被置于其创建的高度之上，《destroy》构造原型消息，用大写的X和短生命线表示对象的显式销毁。

**协作图 collaboration diagram/ communication diagram**

链 link 连接两个对象的路径，指明对象间的可能导航和可见性。对象间的每个消息都可以使用消息表达式和方向箭头表示，编号代表当前控制线程中消息的次序。

实例的创建使用create，或者使用UML的构造性《create》进行说明，可以在生命线框图中加入UML标记值{new}表示实例的创建。

消息的顺序编号：第一个消息不编号，使用合理的编号方案来表示后续消息的顺序和嵌套，例如1，1.1，2，2.1，2.2

有条件信息使用带有方括号的条件字句来表示。1[color=red]:calculate。互斥有条件路径使用条件路径字母来修改顺序编号，例如1a，1b，代表1互斥的可能有1a或1b的消息流向。

循环，使用\*来省略迭代子句，1\*[l=1…n]:num=nextInt

多态消息在超类图中不展示过多的信息，对多态的每个具体情况制作单独的图进行说明。

执行者实例、对象、链接、消息

**比较异同点**：

两者都是UML中表示交互的逻辑和过程流图，在表示方式上，顺序图具有梗丰富的符号标记，能够清晰表示消息的顺序和时间排序，有大量的表示法选项，但是消耗水平空间；通信图更加灵活，便于在有限空间内绘制草图，但是不易于查阅消息的顺序，表示法选项较少。

## 运行期和设计期间差异 DEV475\_07\_RuntimeArchitecture

**运行期架构设计步骤**：

􀂊分析并发性需求：并发需求由以下部分驱动：【系统必须分布的程度，系统事件驱动的程度，计算强度的关键算法，环境支持的并行执行的程度】，根据并发需求排名的重要性来解决冲突。

􀂊识别进程和线程：进程：【提供重量级的控制流，是独立的，可以分为单独的线程】，线程【提供轻量的控制流，在封闭的上下文中进行】

􀂊识别过程的生命周期

􀂊映射到流程的实现

􀂊分发模型元素之间的过程

## 类之间联系的种类

在概念透视图中，类图可以用于将领域模型可视化，在软件透视和设计透视中的新的建模术语位为：设计类图design class diagram，DCD。

UML中的属性可见性包括+（公有的），-（私有的），一般默认情况下是私有的。DCD中表示关联属性时，导航性箭头（navigability arrow）由源对象指向目标对象，表示实例的一个属性是指向的对象。角色名rolename 指放置在目标一端，一表示属性名称。不需要关联名称。

在领域模型中使用类图时，需要关联的名称，但是要避免使用导航箭头，因为领域模型不是软件透视图。领域建模语境下，数据类型是唯一标识不具有意义的对象，常见的数据类型都是原始类型，对于这些数据类型对象使用属性文本表示法，对其他对象使用关联线。两者在语义上一致，但是图中展示与另一个类框的关联线能够强调类与类之间的关联。

关联端点可以附加多重性值，关键字{order}代表集合元素有序，{unique}代表一组唯一元素。

注解符号：注解注释，约束使用{}括起来，方法体。

UML中方法method是操作的实现。操作是可以调用对象执行的转换或查询的规格说明，方法是操作的实现，它规定了操作关联的算法或过程

**泛化**Generalization 由子类到超类的实线和空心三角箭头表示。抽象类可以使用{abstract}标记或者用斜体名称表示。终止类和不能被子类复写的操作用{leaf}表示。

**依赖** dependency 表示客户元素了解其他提供者元素，并且表示当提供者有所改变时会对客户产生影响，从客户到提供者的虚线箭头线表示。类途中依赖线描述对象之间的全局变量、参数变量、局部变量和静态方法的依赖。客户对象对于提供者具有参数可见性，可能由操作上的耦合，构成依赖。

接口 interface的实现包括客户的接口和接口依赖，接口实现interface realization。插座线表示法（表示使用接口，和其他类协作时依赖于接口），依赖线表示法，实现并提供某种接口（使用interface关键字，由**实现**类指向接口类的虚线和空心三角箭头表示），棒棒糖表示法指明为客户实现并提供某接口，提供的接口用棒棒糖表示。

**聚合** aggregation 模糊关联关系，由聚合者指向被聚合者的空心菱形和实线表示

**组合** composition 强整体-部分聚合关系：在某一时刻部分的实例总是只属于一个组成实例，部分总是属于组成，组成要负责创建和删除其部分。由聚合者指向被聚合者的实心菱形和实线表示。

## 类图设计 DEV475\_11\_ClassDesign

步骤：

􀂊创建初始设计类

􀂊定义操作 operations

􀂊定义方法 methods

􀂊定义状态 states

􀂊定义属性 attributes

􀂊定义依赖关系 dependencies

􀂊定义关联 associations

􀂊定义泛化 generalizations

􀂊解决用例碰撞 uc collisions

􀂊处理非功能性需求 nonfunctional requirements

􀂊检查设计过程和结果

## 转出工程

# 复习提要Unit5

## 状态图的格式及作用

转换transition用有事件的箭头表示，状态state用圆角矩形表示。

通常在业务信息系统中通常只有几个复杂的状态依赖类，但是在过程控制，设备控制，协议处理和通信等领域通常有许多的状态依赖对象。

1 对复杂得到事件交互对象建模；【物理设备，事务处理以及相关的业务对象，角色转换器】

2 对操作协议和语言规范的合法序列建模。【通信协议，UI页面/窗口流或导航，UI流控制器或会话，用例系统操作，单个UI窗口的事件处理】

## 控制流图的格式及作用

控制流图(Control Flow Graph, CFG)也叫控制流程图，是一个过程或程序的抽象表现，是用在编译器中的一个抽象数据结构，由编译器在内部维护，代表了一个程序执行过程中会遍历到的所有路径。它用图的形式表示一个过程内所有基本块执行的可能流向, 也能反映一个过程的实时执行过程。

Frances E. Allen于1970年提出控制流图的概念 [1] 。此后，控制流图成为了编译器优化和静态分析的重要工具。

## 信息流图的格式及作用

信息流图是UML 行为图 ，它显示了一些高度抽象的系统实体之间的信息交换。信息流在尚未完全明确或缺少细节的情况下，显示系统信息的流转过程。

信息流 并未指定信息的性质，信息传递机制，交换序列或任何控制条件。 信息项(Information items) 可用于表示在信息流中流经系统的信息，然后才能设计其实现细节。

信息流图的局限性：

UML中的信息流图提供了相当少的表达能力。

信息项 没有提供他们传送的信息的详细信息。它们没有 特征， 泛化或关联。

## 何为业务模式、结构模式、分析模式和设计模式

结构模式/架构模式(Architectural Pattern)

Layers（分层）模式，有时也称Tiers模式

Blackboard（黑板）模式

Broker（中介）模式

Distributed Process（分散过程）模式

Microkernel（微核）模式

常划分成如下的几种：

一、 From Mud to Structure型（从混沌到结构）。帮助架构师将系统合理划分，避免形成一个对象的海洋（A sea of objects）。包括Layers（分层）模式、Blackboard（黑板）模式、Pipes/Filters（管道/过滤器）模式等。

二、分布式系统 （Distributed Systems）型。为分布式系统提供完整的架构设计，包括像Broker（代理者）模式等。

三、人机互动（Interactive Systems）型，支持包含有人机互动介面的系统的架构设计，例子包括MVC（Model-View-Controller）模式、PAC（Presentation-Abstraction-Control：表示-抽象-控制）模式等。

四、Adaptable Systems型（适应性系统），支持应用系统适应技术的变化、软件功能需求的变化。如Reflection（反射）模式、Microkernel（微核）模式等。

设计模式

一个设计模式提供一种提炼子系统或软件系统中的组件的，或者它们之间的关系的纲要设计。设计模式描述普遍存在的在相互通讯的组件中重复出现的结构，这种结构解决在一定的背景中的具有一般性的设计问题。

设计模式常常划分成不同的种类，常见的种类有：

创建型设计模式，如工厂方法（Factory Method）模式、抽象工厂（Abstract Factory）模式、原型（Prototype）模式、单例（Singleton）模式，建造（Builder）模式等

结构型设计模式，如合成（Composite）模式、装饰（Decorator）模式、代理（Proxy）模式、享元（Flyweight）模式、门面（Facade）模式、桥梁（Bridge）模式等

行为型模式，如模版方法（Template Method）模式、观察者（Observer）模式、迭代子（Iterator）模式、责任链（Chain of Responsibility）模式、备忘录（Memento）模式、命令（Command）模式、状态（State）模式、访问者（Visitor）模式等等。