目录

[第一章 上升到面向对象 1](#_Toc13152285)

[1.什么是抽象，如何进行抽象？ 1](#_Toc13152286)

[2.什么是封装，通过封装如何实现信息隐藏和数据抽象？ 1](#_Toc13152287)

[3.什么是泛化，什么是多态，它们之间有什么关系？ 1](#_Toc13152288)

[4.什么是分层，分层和分解有何不同？ 1](#_Toc13152289)

[5. 与传统结构化方法相比，面向对象技术的优势主要体 现在哪些方面？ 2](#_Toc13152290)

[6. 什么是对象，什么是类，说明它们之间的区别和联系 2](#_Toc13152291)

[7. 什么是分解，结构化分解和面向对象分解有何不同？ 2](#_Toc13152292)

[8. 什么是复用，在软件开发的哪些阶段可以进行复用？ 2](#_Toc13152293)

[第二章 可视化建模技术 2](#_Toc13152294)

[1.通过建模技术，可以达到哪些目标？ 2](#_Toc13152295)

[2.在系统建模过程中，需要遵循哪些基本原则？ 3](#_Toc13152296)

[3.哪些情况下，适合使用 UML 进行系统建模？ 3](#_Toc13152297)

[4.UML 中的事物之间主要存在哪些基本关系？ 3](#_Toc13152298)

[5.什么是 UML 架构中的视图，和 UML 图有什么区别和联系？ 3](#_Toc13152299)

[第三章 业务建模 4](#_Toc13152300)

[1.什么是业务建模，软件开发过程中为什么要进行业务建模？ 4](#_Toc13152301)

[2.什么是业务用例模型，业务用例模型主要包括哪些内容？ 4](#_Toc13152302)

[\*3.活动图中的动作节点什么条件下可以执行，有哪些种类的动作节点？ 4](#_Toc13152303)

[\*4.什么是活动图中的控制节点，通过哪类控制节点可以进行并发行为建模？ 5](#_Toc13152304)

[\*5.活动图中的边可以设定哪些执行参数？ 5](#_Toc13152305)

[\*6.什么是活动分区，一般什么情况下对活动进行分区？ 5](#_Toc13152306)

[7.什么是业务对象模型，业务对象模型主要包括哪些内容？ 5](#_Toc13152307)

[第四章 5](#_Toc13152308)

[1.什么是涉众，涉众和参与者有何区别和联系？ 5](#_Toc13152309)

[2.什么是用例的前置条件和后置条件，它们有什么作用，定义时需要注意什么？ 6](#_Toc13152310)

[3.什么是用例的事件流，描述事件流是需要注意什么？有哪几种事件流，它们之间有何区别和联系？ 6](#_Toc13152311)

[4.用例模型中，可以定义哪几种用例关系，它们有何不同？ 7](#_Toc13152312)

[5.什么是扩展点，扩展点有什么作用？ 7](#_Toc13152313)

[6.如何对用例进行分级，高优先级的用例有何特征？ 7](#_Toc13152314)

[7.有哪些用例的分包策略，一般如何进行用例分包？ 7](#_Toc13152315)

[第五章 用例分析 8](#_Toc13152316)

[1.分析模型主要包括什么内容？ 8](#_Toc13152317)

[2.什么是用例实现？它和用例之间有何区别和联系？ 8](#_Toc13152318)

[3.什么是架构模式，有哪些典型的架构模式？ 8](#_Toc13152319)

[4.什么是 B-C-E 三层架构？ 8](#_Toc13152320)

[5.顺序图中主要有哪些元素，绘制顺序图的基本过程是什么？ 9](#_Toc13152321)

[6.什么是边界类，什么是控制类，如何识别这两种分析类？ 9](#_Toc13152322)

[7.什么是关键抽象，如何识别关键抽象？ 9](#_Toc13152323)

[8.什么是架构机制，什么是分析机制，有哪些典型的分析机制？ 9](#_Toc13152324)

[9.什么是实体类，如何有效地识别实体类？ 10](#_Toc13152325)

[第六章 10](#_Toc13152326)

[1.什么是面向对象的设计原则，它和设计质量有何联系？ 10](#_Toc13152327)

[2.什么是 Liskov 替换原则，该原则有什么用？ 10](#_Toc13152328)

[3.什么是开放-封闭原则，该原则与 Liskov 替换原则有何联系？ 10](#_Toc13152329)

[4.什么是单一职责原则，什么时候使用该原则？ 11](#_Toc13152330)

[5.什么是接口隔离原则，什么时候使用该原则？ 11](#_Toc13152331)

[6.什么是依赖倒置原则，在该原则中如何理解抽象层的设计？ 11](#_Toc13152332)

[第七章 11](#_Toc13152333)

[1.什么是模式，什么是设计模式，它们之间有何区别和联系？ 11](#_Toc13152334)

[2.什么是 GoF 模式，有哪些典型的 GoF 模式？ 11](#_Toc13152335)

[3.面向对象的设计原则和设计模式之间有何区别和联系？ 12](#_Toc13152336)

[4.什么是通用职责分配模式，有哪些典型的通用职责分配模式？ 12](#_Toc13152337)

[5.什么是迪米特准则？在什么时候使用该准则？ 12](#_Toc13152338)

[第八章 12](#_Toc13152339)

[1. 什么是包，有哪些包设计原则？ 12](#_Toc13152340)

[2. 什么是子系统，它和包有什么区别和联系？ 13](#_Toc13152341)

[3. 什么是接口，接口和相应的子系统之间是什么关系？ 13](#_Toc13152342)

[4. 如何进行软件架构设计，架构设计时需要考虑哪些方面的问题？ 13](#_Toc13152343)

[5. 什么是设计元素，面向对象设计中有哪些设计元素？ 13](#_Toc13152344)

[第九章 14](#_Toc13152345)

[1. 什么是关联的导航性，如何设计导航性？ 14](#_Toc13152346)

[2. 什么是类间的组合关系，和聚合关系有何区别和联系？ 14](#_Toc13152347)

[3. 什么是类间的依赖关系，哪些情况下定义为依赖关系？ 14](#_Toc13152348)

[4. 子系统设计主要包括哪些工作？ 14](#_Toc13152349)

[5. 在类设计阶段，针对三种分析类的有什么不同的设计策略？ 15](#_Toc13152350)

[6. 什么是类的操作，什么是类的方法，它们有何区别和联系？ 15](#_Toc13152351)

[7. 用例设计和用例分析有什么区别和联系？ 15](#_Toc13152352)

[选择题 16](#_Toc13152353)

[第一章 16](#_Toc13152354)

[第二章 16](#_Toc13152355)

[第三章 16](#_Toc13152356)

[第四章 17](#_Toc13152357)

[第六章 18](#_Toc13152358)

[第七章 19](#_Toc13152359)

[第十章 20](#_Toc13152360)

[名词缩写 20](#_Toc13152361)

[Model： 20](#_Toc13152362)

[OO： 20](#_Toc13152363)

[UML： 20](#_Toc13152364)

[RUP: 20](#_Toc13152365)

[UC: 20](#_Toc13152366)

[M-V-C: 21](#_Toc13152367)

[VOPC: 21](#_Toc13152368)

[LSP： 21](#_Toc13152369)

[OCP： 21](#_Toc13152370)

[SRP： 21](#_Toc13152371)

[ISP： 21](#_Toc13152372)

[DIP： 21](#_Toc13152373)

[D-设计: 21](#_Toc13152374)

[FP-设计: 21](#_Toc13152375)

[I-设计: 22](#_Toc13152376)

[GRASP: 22](#_Toc13152377)

[OMG: 22](#_Toc13152378)

[MDA: 22](#_Toc13152379)

[UML: 22](#_Toc13152380)

[MOF: 22](#_Toc13152381)

[XMI: 22](#_Toc13152382)

[CWM: 22](#_Toc13152383)

[PIM: 22](#_Toc13152384)

[PSM: 23](#_Toc13152385)

[CIV: 23](#_Toc13152386)

# 第一章 上升到面向对象

## 1.什么是抽象，如何进行抽象？

1. 抽象是揭示事物区别于其他事物的本质特征的过程；

（2）需要根据使用者的目的来进行抽象，强调使用者感兴趣的特征，而忽略那些不相关的特征。

## 2.什么是封装，通过封装如何实现信息隐藏和数据抽象？

（1）封装是指对象对其客户隐藏具体的实现；

（2）通过封装，对象的私有数据不能被外界存取，实现信息隐藏，从而保证外界以合 法的手段访问；

（3）通过封装，将数据访问过程抽象为对操作的调用，从而将数据抽象为行为。

## 3.什么是泛化，什么是多态，它们之间有什么关系？

（1）泛化是类与类之间一种关系，通过这种关系一个类可以共享另外一个或多个类的结构和行为。

（2）多态在同一外表（接口）下表现出多种行为的能力；

（3）在对象技术中，一般通过泛化关系建立类之间的抽象层次结构，再通过上层抽象 多态调用底层实现。

## 4.什么是分层，分层和分解有何不同？

（1）分层是指面向不同的目标**建立不同的抽象级别层次**，从而在不同的抽象层次对系统进行分解，进一步**简化对系统的理解**；

（2）分解一般是在系统的同一个抽象层对大的结构进行划分，而分层则是在不同的抽象层次上进行；大规模系统开发时，一般首先通过分层技术建立不同的抽象层次，之后在各个层次上进行合理的分解。

## 5. 与传统结构化方法相比，面向对象技术的优势主要体 现在哪些方面？

主要包括以下几个方面的与优势：

1. 沟通：在计算机中模拟现实世界的事和物；
2. 稳定：较小的需求变化不会导致系统结构大的改变；
3. 复用：提高质量，减少成本；
4. 改善软件结构，提高软件灵活性；增加可扩展性；支持增量式开发，支持大型软 件开发等

## 6. 什么是对象，什么是类，说明它们之间的区别和联系

1. 对象是一个实体，这个实体具有明确定义的边界和标识，并且封装了状态和行为；
2. 类就是对象的抽象描述，这些对象共享相同的属性、操作、关系和语义。
3. 类是对象的抽象，而对象是类的实例，是具体的；通过类可以构造具体的对象。

## 7. 什么是分解，结构化分解和面向对象分解有何不同？

1. 分解是指将单个大规模复杂系统划分为多个不同的小构件；分解后的构件通过抽 象和封装等技术形成相对独立的单元，这些单元可以独立地设计和开发，从而实现化繁为简、 分而治之，以应对系统的复杂性，减少软件开发成本。
2. 结构化分解中，通过函数、模块等进行功能分解，实现模块化设计。通过耦合和 内聚来判断分解的合理性，将系统分解为多个高内聚、低耦合的模块。而面向对象的分解则 是在类和对象分解的基础上，进一步考虑类之间依赖程度、复用问题和稳定性等问题，进行 合理的打包和分层，从而形成更加复杂的分解结构。

## 8. 什么是复用，在软件开发的哪些阶段可以进行复用？

1. 复用是借助于已有软件的各种有关知识建立新的软件的过程，以缩减软件开发和 维护的成本；
2. 系统开发的各个阶段都可能涉及到复用，如代码复用、设计复用、架构复用、需 求复用和领域复用

# 第二章 可视化建模技术

## 1.通过建模技术，可以达到哪些目标？

（1）可视化：模型有助于按照所需的样式可视化系统；

（2）描述：模型能够描述系统的结构和行为；

（3）构造：模型提供构造系统的模板提高质量，减少成本；

（4）文档化：模型可以文档化设计决策。

## 2.在系统建模过程中，需要遵循哪些基本原则？

（1）选择合适的模型；

（2）模型具有不同的精确程度；

（3）最好的模型是与现实相联系的；

（4）需要从多个视角创建不同的模型，单一的模型是不够的。

## 3.哪些情况下，适合使用 UML 进行系统建模？

（1）项目采用的 **OO 方法论**；

（2）提高项目**开发人员之间交流效率**，准确抓住问题本质；

（3）**系统的规模和设计都比较复杂**，需要用图形抽象地表达复杂的概念，增强设计的 灵活性、可读性和可理解性，以便暴露深层的次设计问题、降低开发风险。

（4）**需要记录已成功项目**、产品的公共设计方案，在开发新项目时可以参考、复用过 去的设计，以节省投入，提高开发效率和整体成功率。

（5）有必要采用一套通用的图形语言和符号体系描述组织的业务流程和软件需求，促进业务人员、软件开发人员之间一致、高效地交流。

## 4.UML 中的事物之间主要存在哪些基本关系？

UML 中的事物之间主要 4 类基本关系

（1）**依赖**是两个事物间的**弱语义**关系，表明两个事物之间存在着一种使用关系，其中 一个事物（独立事物）发生变化会影响另一个事物（依赖事物）的语义。

（2）**关联**是一种**强语义**联系的结构关系，表明两个事物之间存在着明确的、稳定的**语义联系**。

（3）**泛化**是一种特殊/一般关系，特殊元素（子元素）的对象可替代一般元素（父元素） 的对象。

（4）**实现**是两个事物是之间的一种契约关系，其中的一个事物（箭头指向的事物）描述了另一个事物必须实现的契约。

## 5.什么是 UML 架构中的视图，和 UML 图有什么区别和联系？

（1）视图可以理解为系统**在某个视角的模型**，每个视图面向不同的用户，提供不同的 UML 模型，以实现不同的建模目标。

（2）**UML 图是特定的 UML 模型，视图由不同的 UML 图组成。**根据视图所面向的用户和建模目标，选择不同的 UML 图进行建模。

# 第三章 业务建模

## 1.什么是业务建模，软件开发过程中为什么要进行业务建模？

（1）业务建模是一种建模方法的集合，目的是对现有业务进行分析和理解，从而建立相应的业务模型。

（2）业务建模有助于理解在**业务领域**中描述的事物是如何与**软件领域**中的事物相**联系**的，从而建立业务模型和系统模型之间的**对应关系**，以保证系统模型是能够满足业务需求的。

## 2.什么是业务用例模型，业务用例模型主要包括哪些内容？

（1）业务用例模型是说明**业务预期功能**的模型，是业务建模阶段的**核心**模型，用于确定**组织的各个角色和可交付工件**。

（2）业务用例模型由**业务用例和业务参与者**构成，主要目的是说明客户和合作伙伴是**如何开展业务**的。

## \*3.活动图中的动作节点什么条件下可以执行，有哪些种类的动作节点？

（1）当动作结点**所有**的**对象流**和**控制流**的前提条件**都**满足时，才创建动作的一次执行。

（2）根据动作执行所涉及的功能不同，可以划分为不同类别的动作，包括**基本功能**、 **行为调用**、**通信动作**和**对象处理**等不同类型的动作节点

*条件：所有对象流控制流的前提条件满足*

*种类：基本功能，行为调用，通信动作，对象处理*

## \*4.什么是活动图中的控制节点，通过哪类控制节点可以进行并发行为建模？

（1）控制节点是一种特殊的**活动节点**，用于在**动作节点**或**对象节点**之间**协调流程**，表示某一种**控制动作**。

（2）根据**分叉和汇合**这两个控制节点可以对并发执行和同步控制行为进行建模。

## \*5.活动图中的边可以设定哪些执行参数？

（1）可以为活动边设定执行条件、关联动作和权重等信息。

（2）执行条件为真时才能通过该活动边进入下一个动作关联的动作。

（3）关联动作表示在进入下一个动作节点之前需要提前执行的动作。

（4）权重规定了转移发生时输入对象的最小数目（常量或表达式），缺省为全部输入对象

## \*6.什么是活动分区，一般什么情况下对活动进行分区？

（1）活动分区用于识别具有相同特性的一组动作，这些动作被放入相同的区间。 （2）可以使用不同的分区规则进行分区，并没有严格的规范。

（3）在业务模型或需求中，往往按照组织机构的单位或系统角色进行分区，一个单位或角色负责分区中所有节点的行为。而在设计模型中，可以按照不同的类（或构件）进行分区，一个类（或构件）负责执行该分区中所有节点的行为。

## 7.什么是业务对象模型，业务对象模型主要包括哪些内容？

（1）业务对象模型从**业务人员内部的观点定义了业务用例**。该模型确定了业务人员以及他们处理和使用的对象（“业务类和对象”）之间应该具有的静态和动态关系

（2）业务对象模型主要包括：**业务工人**、**业务实体**和**业务用例实现**等内容

# 第四章

## 1.什么是涉众，涉众和参与者有何区别和联系？

用例的涉众是指**受用例所代表的业务影响的**（或者说与当前用例有利益关系的）**系统内外**部人员或组织。

**由普通的人或部门来承担的参与者一般都是涉众。**外部系统、时间等不是涉众，因为它们不是人或者组织，没有利益影响；不过**当有外系统参与者**时，那些外系统的用户往往会作为当前用例的涉众存在。

从涉众的角度来看，**用例实际上是涉众之间所达成的契约**，并以参与者为达成特定目标和系统交互的方式演绎。把用例比作一台戏，参与者和系统就是这台戏的演员，而涉众则是观众，戏的好坏由观众来评价。

## 2.什么是用例的前置条件和后置条件，它们有什么作用，定义时需要注意什么？

前置条件是指用例在**执行之前必须满足的条件**，它约束用例开始执行前系统的状态。作为用例的入口限制，前置条件**阻止参与者触发该用例直到满足**所有条件。

后置条件是指**用例执行完成之后系统的状态**。当用例存在多个事件流时，可能会对应多个不同的后置条件。利用后置条件，有助于**确保涉众理解执行用例后的结果**。

在定义前置条件和后置条件时需要注意，**只有在用例的使用者将这些条件视为附加价值的时候才使用**，而且它们均**要求是系统可以感知的**（或者说检测到的）；此外，前置条件还要求是**在用例执行前就可以感知的**。

## 3.什么是用例的事件流，描述事件流是需要注意什么？有哪几种事件流，它们之间有何区别和联系？

用例的事件流是指**参与者和系统交互的过程**。在事件流描述时并不需要将这个完整的交互过程都表示出来；**只需要描述需求部分**，即用户需要什么，系统给出什么样的结果。其次， 事件流的描述要使用户和开发人员互相理解用例的功能，需要注意以下几个方面的问题： 

·**使用业务语言**：使用用户平时所使用的语言进行描述。 

·重点描述参与者与系统交互的**信息**。 

·不使用[例如]、[等]这样的不清晰的表达。 

·**不**要过多的考虑**界面细节**。 

·不要描述系统**内部处理细节**，要描述从系统外部所看到的活动。 

·要明确描述用例的**开始和结束**：一般事件流的第一句话表明该用例在何时如何开始；最后一句话表明用例的结束，有时可以不用显示的说明用例结束。

一个用例可能会存在多个独立的事件流。其中一条**最核心的事件流称为基本事件流**，其它的事件流则为备选事件流。

**基本事件流又称为用例的主路径**，是指在**最一般**的情况下，那些用例发生的路径。它通常用来描述一个理想世界，也就是说**没有任何的错误**发生。

备选事件流代表该用例处理过程中的一些**分支或异常**情况，它一般从基本事件流的某个步骤中分离出来。

## 4.用例模型中，可以定义哪几种用例关系，它们有何不同？

用例模型中，用例的关系主要包**括包含关系、扩展关系和泛化关系**。

**包含关系**表示某个用例（基用例、主用例）中包含了其它用例（被包含用例、子用例）的行为。它提供了从两个或多个用例行为中提取公共部分的能力，把这些公共部分放到某个单独的用例中，通过包含关系来引用这些公共行为。

**扩展关系**是指某个用例（基用例、主用例）在特定情况下无法进行处理，而把这些行为委托给其它用例（扩展用例、子用例），表示该行为被扩展了。它的提出是为了将基用例的一些特殊情况分离出来，在保持基用例本身相对完整的情况下（即一般情况都能处理）来处理这些特殊行为。   
 用例之间的**泛化**表明了一种继承层次，通过这种继承层次，**特化的用例继承泛化用例的全部属性和行为**，并**参与泛化用例的各种关系**。通过用例之间的泛化关系可以**达到更高层次的需求复用**，在泛化用例中描述通用行为，而特化用例继承这些通用行为，并在适当的地方 进行特化，以处理具体的业务。

## 5.什么是扩展点，扩展点有什么作用？

扩展点是指在基用例中定义的**特定条件**，每个扩展用例都至少与一个扩展点相关联。当基用例满足了这些特定条件后，就会触发相应的扩展用例来为基用例提供附加行为。

## 6.如何对用例进行分级，高优先级的用例有何特征？

对用例分级并没有统一的标准，需要结合项目自身的业务特点以及开发团队的技术特点来综合考虑。一般来说，高级别的用例是那些**对系统架构有重要影响的用例**，这些用例体现了系统的核心价值，也将成为后续分析设计的重点。

一般根据用例的**重要性**、**复杂程度**、**风险**等各种因素进行分级。具体来说，存在以下特征的用例一般具有较高级别：

（1）对系统架构有重要影响的用例。

（2）体现系统核心业务流程的用例。

（3）存在开发风险的用例。

（4）涉及新技术或者需要创新的用例。

（5）能够尽快投入使用并带来直接经济效益的用例。

## 7.有哪些用例的分包策略，一般如何进行用例分包？

有几种用例分包的策略：

（1）**基于业务主题**的分包，按照用例所处理的业务领域不同，将面向不同业务主题的用例放在不同的包中。

（2）**按照参与者**分包，即相同参与者参与的用例放在同一个包里面，而不相关的参与者的用例放在不同的包。

（3）**基于开发团队**的分包，即结合开发团队的特点，将由同一个开发团队完成的用例放在同一个包中。

（4）**基于发布情况**的分包，即将在不同发布周期中发布的用例放在不同的包中，而将需要同时发布的用例放在一个包中。

在选择分包策略时，一般首先结合业务特点按照**业务主题**进行分包（即每个包代表一个主题），再**综合考虑开发团队和发布情况**

# 第五章 用例分析

## 1.分析模型主要包括什么内容？

分析模型是对分析所形成**目标制品**的总称；具体来说，分析模型包含两个层次的两类模型。两个层次是指**架构分析**和**用例分析**。两类模型是指**静态模型**和**动态模型**。

## 2.什么是用例实现？它和用例之间有何区别和联系？

用例实现是**分析（设计）模型中一个系统用例的表达式**，它通过**对象交互**的方式**描述了分析（设计）模型中指定的用例是如何实现的**。

通过用例实现将用例模型中的**用例**和分析（设计）模型中的**类**以及交互紧密联系起来，一个用例实现描述了一个用例**需要哪些类来实现**，**两者之间存在实现关**系：即“‘用例实现’ 实现‘用例’”。

## 3.什么是架构模式，有哪些典型的架构模式？

架构模式是那些在**开发过程中积累**下来，并经过**实践验证行之有效的**、**可复用**的**软件架构**。它表示了对软件系统的一个**基础结构组织形式**。它提供了一套**预定义子系统**，详细说明它们的职责，并且包括组织它们之间的规则和指南。

针对不同的软件类别，存在诸多架构模式，如针对系统软件的**层**、**管道**和**过滤器**、**黑板**， 针对交互式软件的**模型视图控制器**模式等等。

## **4.什么是 B-C-E 三层架构？**

B-C-E 三层架构是对 MVC 架构的另一种表述，将系统划分为三层，分别处理 3 类业务逻辑。其中 B 表示**边界层**，负责处理**系统与参与者的交互**；C 为**控制层**，处理系统的**控制逻辑**；E为**实体层**，负责管理**系统使用的信息**。

## 5.顺序图中主要有哪些元素，绘制顺序图的基本过程是什么？

顺序图是一种 UML 交互图，表示对象的交互，**由一组对象和它们之间的消息传递组成**， **强调消息的时间顺序**。主要包括**对象**、**对象生命线**、**消息**、**执行发生**等元素。

一般针对用例的每个场景，均可绘制相应的顺序图，按照 3 个步骤进行。 （1）**放置对象**：从已识别的参与用例的分析类中构造相应的对象放置到顺序图中；

（2）**描述交互**：从参与者开始，按照用例事件流（或场景）的叙述，将系统行为转化 为对象间的消息；

（3）**验证行为**：从后往前，验证对象的行为序列，确保每一个对象能够实现该行为序列。

## 6.什么是边界类，什么是控制类，如何识别这两种分析类？

边界类是从那些系统和外界进行交互的对象中归纳和抽象出来，代表了系统与外部参与 者交互的边界。 控制类封装控制系统上层的边界类和下层的实体类之间的交互行为，是整个用例行为的 协调器。

在用例分析阶段，对边界类识别的基本原则是，为每一对参与者/用例确定一个边界类。 对控制类识别的基本原则是，为每个用例确定一个控制类。

## 7.什么是关键抽象，如何识别关键抽象？

关键抽象是一个通常在需求上被揭示的概念，系统必须能够对其处理。它来源于业务，体现了业务的核心价值，即业务需要处理哪些信息；这些信息所构成的实体即可作为初步的 实体分析类。

关键抽象来自于业务领域，领域专家可以很清楚地提供业务系统的初始关键抽象候选集 合，在此基础上，再结合业务对象模型、需求和词汇表等业务文档资料补充和完善。

## 8.什么是架构机制，什么是分析机制，有哪些典型的分析机制？

架构机制是对**通用问题**的**决策**、**方针**和**实践**，它描述了针对一个经常发生的问题的一种通用解决方案。通过有效地应用架构机制，可以使项目组内部以相同的方式对待这些问题， 并复用相同的解决方案来实现复用。

分析机制是架构机制在分析阶段的表述，它以与实现无关的方式捕获解决方案的关键部 分。它们可能表示结构模式或行为模式，也可能表示这两者。它们主要用于在分析过程中向 设计人员提供复杂行为的简短表示，从而减少分析的复杂性并提高分析的一致性。

典型的分析机制包括持久性、分布、安全性等。

## 9.什么是实体类，如何有效地识别实体类？

**实体类代表了系统的核心概念**，来自于对业务中的实体对象的归纳和抽象，用于记录系统所需要维护的数据和对这些数据的处理行为。 实体类是用来表示业务信息的名词，因此识别实体类的基本思路是分析用例事件流中的名词、名词短语，找出所需的实体信息。架构分析中的关键抽象更是识别实体类最重要的来 源，而更多的实体类还需要从用例事件流、业务模型、词汇表等业务和需求的载体中获得。

# 第六章

## 1.什么是面向对象的设计原则，它和设计质量有何联系？

面向对象的设计原则是**指导**面向对象设计的**基本思想**，是评价面向对象设计的**价值观体系**，也是构造**高质量软件**的**出发点**。

## 2.什么是 Liskov 替换原则，该原则有什么用？

Liskov 替换原则是由 Liskov 提出来的针对继承层次设计时所要遵循的原则，该原则指出：若对每个类型 S 的对象 o1，都存在一个类型 T 的对象 o2，使得在所有针对 T 编写的程序 P 中，用 o1替换 o2 后，程序 P 的行为不变，则 S 是 T 的子类型。

该原则用于**指导继承层次的设计**，它要求在任何情况下，子类型与基类都是可以互换的，那么该继承的使用就是合适的，否则就可能出现问题。

## 3.什么是开放-封闭原则，该原则与 Liskov 替换原则有何联系？

开放-封闭原则是指模块应该即是开放的又是封闭的。软件模块**对于扩展是开放的**：模块的行为可以扩展，当应用的需求改变时，可以对模块进行扩展，以满足新的需求。而软件模块**对于修改是封闭的**：对模块行为扩展时，不必改动模块的源代码或二进制代码。

**LSP 是 OCP 成为可能的主要原则之一**，正是子类型的可替换性才使得使用基类型的模块在无需修改的情况下就可以扩展。通过定义抽象基类来建立软件系统的基本结构，在此结构上，只需要通过扩展相应的派生类即可应对需求变更或新的需求。

## 4.什么是单一职责原则，什么时候使用该原则？

单一职责原则是指“**对一个类而言，应该只有一类功能相关的职责**”。该职责主要用于**类的职责分配**，如果一个类承担过多的职责，那么就会有**多个引起变化的原因**，从而造成**类内部的频繁变化**。**类设计**是应遵从SRP，建立**高内聚的类**。

## 5.什么是接口隔离原则，什么时候使用该原则？

接口隔离原则是指“**使用多个专门的接口**比使用单一的总接口要好”，更具体来说，就是一个类对另外一个类的依赖性应当是**建立在最小的接口上**的。该原则**指导在接口的设计**， 一个接口应当**简单地代表一个角色**，而不是多个角色。如果系统涉及到多个角色的话，那么每一个角色都应该由一个特定的接口代表。

## 6.什么是依赖倒置原则，在该原则中如何理解抽象层的设计？

依赖倒置原则是指“**高层模块不应该依赖于低层模块，二者都应该依赖于抽象**；**抽象不应该依赖于细节，细节依赖于抽象**。”

抽象是对事物本质特征的描述，因此对**系统进行抽象的过程**就是透过现象看**本质**的过程，**通过对本质特征的描述从而建立稳定的系统结构**。在职责分配过程中，需要对**职责进行一定的抽象**，在抽象层次上去理解和分配职责。

# 第七章

## 1.什么是模式，什么是设计模式，它们之间有何区别和联系？

模式是**对以往成功应用经验的总结与复用，是对某个问题的通用解决方案，**并**可以重复**使用该方案。

设计模式是在**构件设计阶段**，通过**定义类或特定对象之间的结构和行为**，从而解决某类设计问题的通用解决方案。

模式的含义更广，可用于各类背景和领域；设计模式是模式的一种，是针对面向对象设计的微结构模式。

## 2.什么是 GoF 模式，有哪些典型的 GoF 模式？

GoF 模式是指由 Erich Gamma 等四人在《设计模式—可复用面向对象软件的基础》一书中提出的 23 种设计模式。

GoF 设计模式按照范围分为**类模式**和**对象模式**，按照目的分为**创建型模式**、**结构型模式** 和**行为型模式**。典型的 GoF 模式有**工厂方法**、**抽象工厂**、**单例**、**适配器**、**组合**、**命令**、**状 态**、**策略**等模式。

## 3.面向对象的设计原则和设计模式之间有何区别和联系？

设计原则是面向对象设计的**指导思想**，设计模式只是更好地遵循这一思想的**手段之一**。

## 4.什么是通用职责分配模式，有哪些典型的通用职责分配模式？

职责分配模式是面向对象设计中进行**类的职责分配**的**原则**和**模式**，他们结合类职责分配期间**所面临的问题**，给出了**具体分配规则**；从而可以有效地**指导**用例设计期间类的**职责分配过程**。典型的类职责分配模式包括：**创建者**、**信息专家**、**低耦合**、**控制器**、**高内聚**、**多态**、**纯虚构**、**中介**和**受保护**变化等。

## 5.什么是迪米特准则？在什么时候使用该准则？

迪米特准则是面向对象设计中另一个非常实用的**职责分配模式**。它给出了在一个方法内应该**向哪些对象发送消息**的**限制**。

该准则给出了在**一个方法内应该向哪些对象发送消息的限制**，规定在一个方法中，消息只能发往下面的对象：

* 对象本身；
* 该方法的一个参数；
* 对象本身的属性；
* 对象本身的一个属性集合中的元素；
* 该方法内部创建的对象。

# 第八章

## 1. 什么是包，有哪些包设计原则？

包是一种将**模型元素分组**的机制。它是一个容器，用来包含其它的 UML 元素；与此同 时，包还为其内部元素提供了名字空间，外界需要通过包的名字来访问其内部的元素。

包设计原则包括复用发布等价原则、共同复用原则、共同封闭原则、无环依赖原则、稳 定依赖原则和稳定抽象原则等

## 2. 什么是子系统，它和包有什么区别和联系？

子系统本质上是一种**特殊的包**，这种包是**完全封装的**，其内部元素并**不对外公开**；它实现一个或多个接口所定义的行为，外界通过接口来获取所需的服务。

子系统的接口提供了一个封装层，从而使外部模型元素看不到子系统的内部设计；这一 概念用于将它和“普通”包区分开来：“普通”包是**无语义的模型元素容器**；而子系统则表 示具有**与类相似的行为特征的包的特定用法**。

## 3. 什么是接口，接口和相应的子系统之间是什么关系？

接口定义了某一类元（类、子系统或构件等）所实现的操作集合，这些操作只有定义， 没有任何实现。在设计模型中，主要用于定义子系统的接口。子系统和接口之间构成实现关 系，子系统实现接口中的操作。

## 4. 如何进行软件架构设计，架构设计时需要考虑哪些方面的问题？

架构设计的活动在分析阶段就已经开始，分析阶段主要关注基础架构的选型和并确定核 心的分析机制。设计阶段，则需要针对分析阶段的备选架构的各个方面进行详细的定义，以 设计出符合特定系统的架构；这些具体工作包括以下几个方面。

* 确定核心元素：在架构的中高层，以分析类为出发点，确定相应的核心设计元素， 这些设计元素将作为构件设计的基本输入。
* 引入外围元素：在架构的中低层，以分析机制为出发点，确定满足分析类要求的设 计机制，并将相关的内容引入设计模型
* 优化组织结构：按照高内聚、低耦合等设计原则，整理并逐渐充实架构的层次和内 容，以建立特定系统的合理架构。
* 定义设计后的组织结构：除了考虑系统设计时的组织结构，架构设计还应该考虑设计完成后系统实现、运行以及部署等阶段的组织结构。

## 5. 什么是设计元素，面向对象设计中有哪些设计元素？

设计元素是指能够直接用于**指导实现（编码）**的**模型元素**。主要的设计元素有：**设计类**、 **子系统**、**接口**、**主动类**、**事件**和**信号**等

# 第九章

## 1. 什么是关联的导航性，如何设计导航性？

导航性是指**关联的方向**，它描述了**从源类的任何对象到目标类的一个或多个对象**的访问权限，消息**仅能在箭头的方向上传递**。

在分析阶段，没有描述导航性则默认为双向的导航。而设计阶段，则应根据需要设计单 方向的导航性。好的面向对象设计的目标是最小化类间的耦合，而使用单方向的导航性可以 降低耦合，在没有导航性的方向上就没有类间的耦合，实现时也不需要额外的支持。此外， 双方向关联难以实现，需要消耗额外的维护成本。这些因素都表明，在设计期间应尽可能采 用单方向的关联。

当类 A 与类 B 关联时，应从类 A（或类 B）对象是否需要知道类 B（或类 A）的对象入手来分析它们之间的导航性；换个角度来说，即从类 A（或类 B）对象是否向类 B（或类 A）的对象发送消息。

## 2. 什么是类间的组合关系，和聚合关系有何区别和联系？

组合关系是一种特殊的聚合关系，在整体拥有部分同时，部分不能脱离整体而存在；当 整体不存在时，部分也没有存在的意义。从实现的角度来说，聚合表示一种引用关联，即整 体保存部分的引用，部分本身可以相对独立地存在；而组合则表示一种值关联，整体直接拥 有部分的值，并负责部分的创建和删除。

## 3. 什么是类间的依赖关系，哪些情况下定义为依赖关系？

依赖是一种使用关系，表示一个类对象使用另外一个类对象的信息和服务，被使用对象 的变化可能会影响到使用对象。

定义为依赖关系的几种情况：参数引用，局部声明引用和全局引用。

## 4. 子系统设计主要包括哪些工作？

针对每一个待设计的子系统，需要完成以下三个方面的工作：

1. 将子系统行为分配给子系统元素：一个子系统对外提供的行为完全由其接口进行 描述，因此接口操作的集合代表子系统的职责。子系统设计的第一步就是针对接口所描述的 每一个操作进行设计，通过交互图将操作的职责分配给子系统内部的设计元素。
2. 描述子系统内部的设计元素：在交互图的基础上，定义每个设计元素的结构和关 系，完成子系统内部设计模型。
3. 定义子系统间的依赖关系：分析子系统与外部设计元素之间的依赖关系，明确子 系统之间的耦合，以便于子系统的复用。

## 5. 在类设计阶段，针对三种分析类的有什么不同的设计策略？

边界类的设计策略：边界类分为用户界面和系统接口，其中系统接口在架构设计时一般 定义为子系统和接口来实现，并通过子系统设计来完成其内部设计流程。而针对用户界面类， 需要研究具体的与用户交互的场景，设计满足要求的最终用户界面。界面类的设计往往依赖 项目可用的用户界面开发工具。目前大多数界面设计工具都提供了自动创建了实现用户界面 所必需的支持类的能力，这样类设计期间并不需要太多的考虑。更多地是从界面元素的布局 等人机工程学方面去考虑问题。

实体类的设计策略：由于实体类本身职责的明确性，大多数实体类都可以直接作为初始 的设计类存在。不过由于实体类往往具有持久性架构机制，因此该架构机制应用以及数据库 的一些设计原则也会影响到实体类的设计方案。此外，性能方面的要求也可能要对实体类进 行重构。

控制类的设计策略：控制类的设计首先需要明确该控制类是否有必要存在，有些控制类 只是简单地将边界类的消息转发给实体类，这种不含任何业务逻辑或处理流程的控制类就没 有存在的必要。当决定保留现有的控制类实现用例行为时，需要结合当前的用例实现和设计 质量方面的考虑，针对现有的控制类进行适当的处理，可以从以下两个方面改进控制类：（1） 提供公共控制类；（2）分解复杂的控制类

## 6. 什么是类的操作，什么是类的方法，它们有何区别和联系？

操作是**类的行为特征**，它描述了**该类对于特定请求做出应答的规范**。

方法是操作的**具体实现算法**，它描述操作如何实现的流程。

操作描述了类对外提供的接口，是类的外在行为。通过定义操作明确了参数和返回值等 接口细节；而方法则是关注操作内部实现算法的设计。

## 7. 用例设计和用例分析有什么区别和联系？

**用例设计是用例分析的延续**，通过**利用架构设计提供的素材**（设计元素和设计机制等）， 在不同的局部，**将分析的结果用设计元素加以替换和实现**。

用例设计所采用的建模方法与用例分析完全相同，主要还是交互图分析动态场景、类图 描述静态结构。但是，其出发点和关注点则完全不同：从出发点上来说，不再使用分析类的 概念分配职责，而是从设计元素、设计机制的角度，结合设计原则和模式（包括 GoF 模式 和职责分配模式等）进行职责分配；从关注点来说，用例设计关注的是**职责如何实现**而不是 目标类需要提供什么职责，即目标类提供怎样的操作才可以响应这些消息，这意味着发送到 设计类的消息对应设计类的操作，而发送到子系统的消息对应其接口的操作。

# 选择题

## 第一章

1.\_\_\_\_是一系列指导软件构造的原则

A. 一系列指导软件构造的原则

2.下列有关类的定义，正确的是

A. 对象的抽象

3.下列\_\_\_之间的关系是类和对象之间的关系。

B.老师和张老师

## 第二章

1.模型是\_\_\_\_

A.现实对象的简化

2.下列关于UML的论述，错误的是\_\_\_。

C. 可作为一种指导软件开发的通用语言

3.UML中的“统一”体现在很多方面，下列选项\_\_\_\_不是UML统一的

B.软件开发过程

4.和UML 1.x相比，UML2进行了比较大的改动，对于普通用户来说，主要的改动体现在对一些图进行了调整。下列四个选项中，\_\_\_\_是UML新增的用于描述静态结构的图

D.组合结构图

## 第三章

1.下列有关业务建模的概念和方法的论述中,错误的是()

A.业务建模是软件开发的必备环节

B.可以采用用例技术进行业务建模

C.可以通过活动图详细描述业务流程

D.业务模型可以映射到系统模型

2.下列有关业务模型的相关概念中,错误的是()

A.业务参与者在业务之外

B.业务工人在业务内部

C.业务用例为业务工人提供价值

D.业务实体在业务内部

3.下列关于活动图的论述中,错误的是()

A.可以包括多个起点

B.分区用来表示该分区内的活动是由谁负责的

C.活动可以简单,可以复杂

D.可以使用活动图描述业务用例流程

4.下列选项中,()不会出现在活动图中

A.活动

B.用例

C.对象

D.分叉

5.业务模型中的业务实体,在系统模型中最有可能成为(

A.系统用例

B.参与者

C.控制类

D.实体类

6.以某海鲜酒家为研究对象,下列选项中,()是业务工人

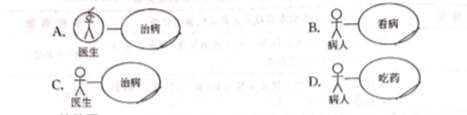
A.服务员

B.菜

C.食客

D.菜

7.以某医院为研究对象，下列业务用例图中，正确的是（B）



## 第四章

1.下列选项中,关于业务参与者和系统参与者的论述,正确的是(

A.业务参与者一定是系统参与者

B.系统参与者一定是业务工人

C.系统参与者一定要与系统交互

D.系统涉众一定是系统参与者

2.下面4个选项中,()肯定不能作为系统的参与者

A.直接使用系统的人

B.需要交互的外部系统

C.系统自身的数据库

D.时间

3.下列选项中,()不会出现在需求阶段的用例文档中

A.基本事件流

B.备选事件流

C.用例实现场景

D.前置条

4.下列有关用例文档相关内容的论述中,正确的是()

A.涉众等同于参与者

B.每个用例都应有前置条件和后置条件

C前置条件必须在用例开始执行前就能检测到

D.编写用例的事件流时应尽可能细化各种实现细节

5.在一个“订单管理子系统”中,创建新订单和更新订单都需要核查用户账号是否正确,那么,用例“创建新订单”“更新订单”与用例“核查客户账号”之间是()关系

A.包含

B.扩展

C.泛化

D.实现

6.考虑某客户服务系统,客服部人员接听完客户电话后,需要通过该系统记录客户来电的内容,则用例“记录客户来电”的前置条件最可能是()

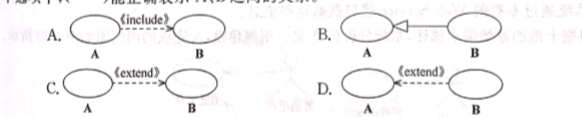
A.客服部人员已经登录

B.有客户打来电话

C.客服部人员有空

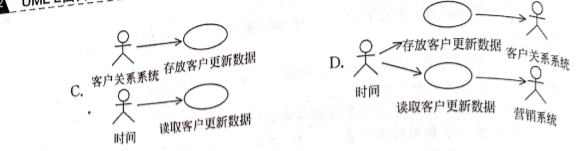
D.客服部人员接听完客户电话

7.用例之间存在3种关系,即包含、扩展和泛化。已知用例A表示一般情况,而用例B是在用例A到达一个特定点时才发生的情况(该特定点可能到达,可能不到达),则下列4个选项中,(D)能正确表示A、B之间的关系



8.某电信营销系统需要为营销人员提供各种客户数据的统计分析功能,而这些客户数据来自另一个外部系统一—客户关系系统。客户关系系统定时(如24:00点)把客户数据的更新存放在某个约定的位置,营销系统也定时(如01:00点)去约定位置读取更新数据,请同以下用例图最准确地描绘了该营销系统有关客户数据更新功能需求的是(A)





## 第六章

1.下列选项中，\_\_\_是顺序图具备而通信图不具备的功能。

C.显示交互时对象的执行发生

2.下列选项中，\_\_\_是通信图具备而顺序图不具备的功能。

B.显示交互对象间的关系

3.面向对象的设计原则是指导我们进行面向对象设计的基本思想，如果违背了这些原则，则设计模型可能会存在很严重的间题；现发现在一个已有的设计模型中，有一些使用父类正常运行的方法，在使用子类时无法运行，这种现象可能是因为我们违背了\_\_\_\_设计原则。

A.LSP

## 第七章

1.面向对象的设计原则与设计模式最本质的区别是()

A.设计原则用于构架设计,而设计模式用于构件设计

B.设计原则是基本指导思想,而设计模式则是具体技术的应用

C.设计原则与编程语言无关,而设计模式依赖于特定的编程语言

D.设计原则适用于所有的面向对象系统,而设计模式只适用于特定的应用系统

2.下列有关设计原则和设计模式的论述中，错误的是\_\_\_\_

A.设计原则是构造高质量设计的出发点

B.设计模式是遵循设计原则的手段之一

C.设计原则来自于设计模式的具体应用

D.设计模式的核心思想是多态包容

3.设计模式是设计中通用问题的解决方案；GoF的23种设计模式为我们的设计提供了许多优秀的解决方案。在某一系统的设计过程中发现这样一个问题：已有的两个设计类需要互相通信，但接口不一致，此时我们应该考虑使用（）来解决。

A.状态（State）模式

B.装饰（Decorator）模式

C.适配器（Adapter）模式

D.命令（Command）模式

4.迪米特（Demeter）准则用于指导详细设计阶段类的职责分配，根据该准则，在一个对象的方法中，其消息不应该发往（）。

A.对象本身

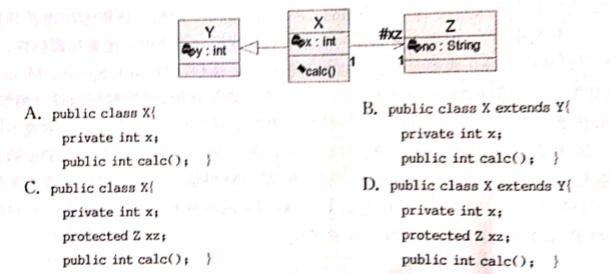
B.该方法的参数

C.该方法内创建的对象

D.直接依赖于该对象的对象

## 第十章

1.根据设计类图（如下图），产生类X的框架代码，下列选项中完全正确的是\_D\_\_



# 名词缩写

## UML：

统一建模语言

## RUP

Rational统一过程

## M-V-C:

模型-视图-控制器

## VOPC:

参与类类图

## LSP：

Liskov替换原则

## OCP：

开放-封闭原则

## SRP：

单一职责原则

## ISP：

接口隔离原则

## DIP：

依赖倒置原则

## GRASP:

职责分配模式

老师提供：

## OMG:

对象管理组织

## MDA:

模型驱动架构

## UML:

统一建模语言

## MOF:

元对象设施

## XMI:

XML元数据交换

## CWM:

公共仓库元模型

## PIM:

平台独立模型

## PSM:

待定平台模型

## CIV:

计算无关的视角