第 1 章　上升到面向对象

选择题

1. A
2. A
3. B

简答题

1. 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。

2. 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。

3. 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。

4. 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。

5. 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。

6. 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。

7. 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。

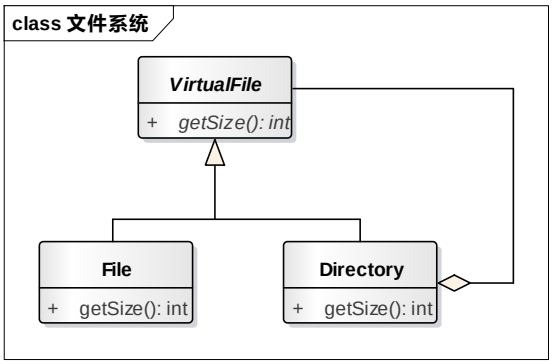
8. 什么是复用，在软件开发的哪些阶段可以进行复用？

（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。

应用题

1. 设计方案的核心是对文件（File）和目录（Directory）进行抽象，通过建立一个公共的抽象类（虚拟文件类 VirtualFile）将两个概念联系起来，并通过聚合关系建立文件和目录之间的递归关系；而文件大小的计算，直接通过相应的类的操作来实现。设计类图如下图所示。



2. 本题为论述题，没有参考答案。读者可以选择任何一种熟悉的原则，结合某个具体的业务场景进行论述。比如系统中如何分层，如何复用第三方类库或以前的项目代码等等。

第 2 章　可视化建模技术

选择题

1. A
2. C
3. B

4. D

简答题

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，以便暴露深层次的设计问题、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 的语法结构使用什么方式来定义，如何定义？

（1）UML 语法结构采用 UML 元模型来定义；

（2）主要是采用 UML 类图描述各元素的抽象语法，采用约束机制和自然语言（文本）来描述模型语义。

5. UML 的语义结构主要包含什么内容？

UML 的语义结构主要包括两类语义域

（1）结构语义定义了在建模域中关于个体的 UML 结构化模型元素的含义，这个含义可能只在某个特定的时间点是正确的，也称为静态语义。

（2）行为语义定义了在建模域中关于个体如何随着时间变化而做出不同行为的 UML 行为模型元素，也称为动态语义。

6. 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。

7. 什么是构造型，UML 中如何利用构造型进行扩展？

构造型是 UML 的一种扩展机制，其作用是基于已有的建模元素扩展新的建模元素，可用于所有的 UML 模型元素。构造型的使用非常简单，只需要通过为已有元素设定一个构造型标记，以及相应的属性即可，也可以通过图标的方式区分不同的构造型。

8. 什么是外廓，如何利用外廓图扩展 UML 模型？

（1）外廓是基于 UML 元素的子集为特定领域定义了 UML 的一个特定版本，即定义了一组对 UML 已有模型的扩展和限定机制，以用于某个特定领域。这些扩展和限定机制包括：预定义的构造型、标记值、约束和基类等。

（2）外廓图是一种用于描述 UML 扩展机制的结构图，通过外廓图可以定义外廓包，以及特定的构造型、使用的元类、构造型和元类之间的扩展关系等内容，从而完成一系列的扩展。

9. 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。

应用题

1. 不同的 UML 工具完成的图形可能会有细微的差别，本题答案主要检查核心建模元素的使用是否正确即可，不对细节要求过多。
2. 可以选择几种主流 UML 工具试用，分析其特点和主要使用场合。UML China 网站定期更新 UML 工具列表（http://www.umlchina.com/Tools/Newindex1.htm），感兴趣的可以根据该列表选择合适的工具。

第 3 章　业务建模

选择题

1. A
2. C
3. A
4. B
5. D
6. A
7. B

简答题

1. 本书讨论的 UML 分析设计过程主要包括哪几个阶段？

主要包括 6 个阶段

（1）业务建模：采用软件建模方法分析和理解带开发的业务，描述业务流程；其目标是认识业务本质，该业务本质是后续用例建模的基础。

（2）用例建模： 采用 UML 用例建模技术描述软件需求，该需求模型将为后续用例分析提供输入。

（3）用例分析： 采用 UML 用例分析技术分析软件需求，建立软件系统的分析模型。

（4）架构设计： 在系统的全局范围内，以分析模型为基础，设计系统的架构。

（5）构件设计： 根据架构设计的成果，将分析模型细化，设计系统构件的实现细节

（6）代码实现： 将系统构件映射到目标语言上。

2. 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。

3. 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。

4. 什么是业务参与者，如何识别业务参与者？

（1）业务参与者代表了与业务有关的角色，此角色由业务环境中的某个人或物扮演。

（2）识别业务参与者的关键在于明确业务边界：业务参与者是在业务边界之外的、与业务进行交互的人或组织，它接受业务所提供的服务，并关注业务所产生的结果。

（3）在实际业务建模过程中，业务参与者可以是与业务进行交互的任何个人、组织、公司或计算机，可以从以下类别中查找参与者：客户、供应商、合作伙伴、潜在客户、政府、业务中为建模部分的人或组织。

5. 什么是业务用例，如何识别业务用例？

（1）业务用例展示了业务的外部视图，它确定了业务为了向业务参与者交付期望结果，需要执行什么流程；同时还确定了，在执行业务用例时，业务与业务参与者之间需要进行哪些交互。

（2）为了能够有效地识别主要的业务用例，可以从业务参与者的角度来考虑，即业务参与者通过业务用例从业务中获取价值。

（3）还可以从业务流程内部进行封装业务用例，研究业务内部的各类活动和流程，分析活动的目标，从而确定这些活动是为外部业务参与者提供怎样的服务，这些服务即可表示为业务用例。

（4）此外，还需要注意的是不要遗漏其他方面的业务用例，如一些支撑性业务流程（包括不直接使客户受益的活动）背后的业务用例。

6. 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

7. 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。

8. 活动图中的对象节点有哪两种表示方式，有何不同？

（1）有标准的对象节点和引脚两种表示方式

（2）标准的对象节点独立存在，而引脚则依附于动作节点，表示该动作的输入参数或输出值，是一种简化的表示方式，也可以转换为标准对象节点进行建模。

9. 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

10. 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。

11. 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。

12. 业务模型和系统模型之间有何区别和联系？

（1）业务模式建模当前业务现状，而系统模型则描述系统中业务实现模式。

（2）业务模型可以为系统模型中的用例视图和逻辑视图提供输入，还可以为系统架构提供一些重要的架构机制。

应用题

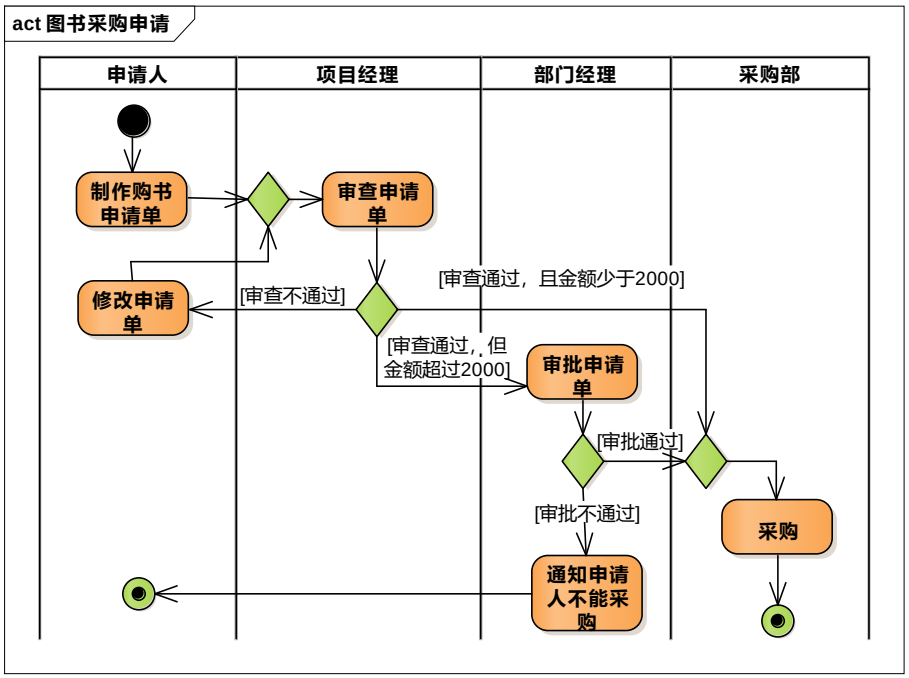
1. （1）申请人，（2）管理员，（3）公司前台

（4）需要预约，（5）需要取消预约，（6）取消接待室预约，（7）删除来访信息

（8）没有空余接待室，（9）有空余接待室，（10）预约不成功，（11）预约成功

（12）登记来访信息

2. 参考模型



第 4 章　用例建模

选择题

1. C
2. C
3. C
4. D
5. A
6. A
7. D
8. A

简答题

1. 什么是需求，有哪些类型的需求？

需求是客户可接受的、系统必须满足的条件或具备的能力。RUP 中将需求分为 FURPS+多种类型，其中：

（1）功能性（Functionality）：详细描述了系统必须有能力执行的动作，通过详细说明所期望的输入和输出条件来描述系统行为；

（2）可用性（Usability）：人为因素（审美学、易学性、易用性）和用户界面、用户文档、培训资料的一致性；

（3）可靠性（Reliability）：主要包括故障的频率和严重性、输出结果的精确性、故障平

均时间（Mean-Time-To-Failure, MTTF）、故障恢复能力和程序的可预见性；

（4）性能（Performance）：在功能性需求上施加的条件，如需求详述了交换率、速度、有效性、准确性、响应时间、恢复时间和内存使用，同时还加上了必须执行某个活动的条件；

（5）可支持性（Supportability）：综合了可扩展性、适应性和耐用性等方面的能力，以及可测试性、兼容性、可配置性和其它在系统发布以后维持系统更新需要的质量；

（6）“+”号指还包含其它类型的需求，这些需求包括：设计约束、实施需求、接口需求和物理需求等。

2. 在业务建模之后，如何寻找业务改进点？哪些业务改进点可能会作为需求？

可以从以下四个方面来寻找业务的改进点

（1）流程控制：该业务涉及到复杂的控制流程，并在多个用户或部门之间流转；手工的信息流转方式难以满足业务需求，而且容易出错；

（2）复杂业务逻辑：某些活动涉及到一些复杂的业务逻辑，手工完成有很大的难度或工作量过大；

（3）使用业务对象：某些活动主要是对业务对象的操作，手工方式难以保证操作方式的合法性，并难以记录操作的历史等信息；

（4）自动化业务：某些业务要求定期、实时进行处理，其操作过程也不涉及复杂交互，手工方式难以保证按时完成。

对于每一个业务改进点，明确是否是为了达到远景目标的需要，如果是则作为软件需求而存在，并把相应的模型转化为系统模型；如果不是则不作为需求而存在，可能作为一项潜在的需求考虑，也可能直接抛弃。

3. 什么是系统参与者，识别参与者的主要要点包括哪些？

系统参与者代表了以某种方式与系统交互的人或事。更直观的说，参与者是指在系统之外，透过系统边界与系统进行有意义交互的任何事物。识别参与者的主要要点包括：

（1）系统外：参与者不是系统的组成部分，处于系统的外部；

（2）系统边界：参与者透过边界直接与系统交互，参与者的确定代表系统边界的确定；

（3）系统角色：参与者是一个参与系统交互的角色，与使用系统的人和职务没有关系；

（4）与系统交互：参与者与系统交互的过程是系统所需要处理的，即系统职责；

（5）任何事物：参与者通常是一个使用系统的人，但有时候也可以是一个外系统或外部因素、时间等外部事物。

4. 什么是系统用例，获取用例的主要要点包括哪些？

系统用例就是支持参与者与系统交互并达成参与者使用系统的目标，它由一组用例实例构成，而用例实例是系统执行的一系列动作，这些动作将生成特定参与者可观测的结果值。识别用例的主要要点包括：

（1）可观测：用例描述的是参与者与系统的交互，而不是系统内在的活动；因此用例的定义也应该只关注系统对外所体现的行为，或者说用例它止于系统边界。

（2）结果值：每个用例都会对外界参与者产生一个有价值的结果。

（3）系统执行：用例所产生的结果值是由目标系统所生成的。

（4）由参与者执行：用例的识别和定义都是从参与者的角度出发的，以参与者的视角获取和命名用例。

5. 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。

6. 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。

7. 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。

8. 用例的补充约束主要包括哪些内容？如何描述补充约束？

用例重点在于描述功能需求，但对于系统来说，还存在很多功能之外的东西，比如非功能需求等，还有其它的一些诸如数据项的定义、业务规则、设计约束等内容。这些内容统称为补充约束。

与特定用例相关的补充约束，作为该用例文档中一部分来描述；而全局性的补充约束，单独形成一份独立的补充约束文档。在具体描述补充约束时，一般采用自然语言来描述，但针对不同类型的补充约束，可以有不同的描述方法。如针对数据需求，可以采用数据字典的方式来描述；针对业务规则，可以使用决策表、OCL 等特定的工具或语言来描述。

9. 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。

10. 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。

11. 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

12. 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。

应用题

1. 建议结合具体的实践项目讨论具体的方法，一般都会涉及到收集资料、访谈、开会等基本的方法。
2. 主要的错误包括：

（1）参与者“每月初”改为“时间”

（2）普通用户和注册用户：如果有泛化就应去掉注册用户与浏览文章关联，或去掉泛化关系

（3）<<extend>>的箭头方向画反了

（4）前置条件应去掉“且有要评论的文章”，因为用例启动前无法检测

（5）基本事件流 2 应去掉，不是该用例路径（因为登录已经作为前置条件了）

（6）缺少用例“申请开通 blog”

3.

（1）时间

（2）管理已录制的节目

（3）预约录制

（4）extend

（5）电子节目指南系统

4.

（1）注册

（2）查询商品

（3）订购

（4）库存管理系统

（5）货物配送系统

5. [综合案例：考勤系统]

参见“考勤系统参考答案和评分标准”

6. [综合案例：医院预约挂号系统]

参见“医院预约挂号系统参考答案和评分标准”

第 5 章　用例分析

选择题

1. B
2. C
3. A
4. C
5. B
6. （1）B　　（2）A　　（3）D　　（4）D
7. （1）D　　（2）A　　（3）D
8. （1）C　　（2）A

简答题

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。（具体包含的内容可以按照书上展开说明）。

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[1]中一个系统用例的表达式，它通过对象交互的方式描述了

[1] 分析和设计阶段都需要定义用例实现，其含义相同，只是内部处理细节不同。

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. 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。

6. 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。

7. 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。

8. 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。

9. 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。

10. 顺序图中的交互片段有什么用，有哪些典型的交互片段？

交互片段将顺序图中的若干消息和对象的封装为一个片段，针对这个片段可以实施不同的操作，从而来表示这个片段是选择、循环还是并行等各种非顺序执行。

交互片段支持不同类型的操作符，从而实现不同的控制结构，典型的操作符有可选（opt）、选择（alt）、循环（loop）和并行（par）等。

11. 什么是用例实现的参与类类图？

参与类类图是指与某个特定用例实现相关的类以及类之间的静态关系，表示为支持该用例实现行为所需要的分析类以及关系。

12. 什么是类的职责，如何定义职责？

职责是要求某个类的对象所要履行的行为契约，它说明了该对象能够对外提供哪些行为，在设计中将演化为类的一个或多个操作。可以从两个方面来定义类的职责：

（1）从交互图中的消息获得职责；对于每一条消息，接收该消息的对象需要提供相应的职责来响应。

（2）从非功能需求中获得其它职责。

13. 对象间的链接和类间的关联关系有何区别和联系？

链接是两个对象之间的语义联系，它允许消息从一个对象发送到另一个对象；关联关系则是类之间的一种结构化关系，是类之间的语义联系，表明类的对象之间存在着链接。

对象是由特定的类生成的，对象之间的链接也需要类之间的关系来生成。

14. 什么是多重性，如何理解类间的多重性定义？

多重性表示一个类的对象可能链接到所关联的类的多个对象上，这种“多少”即为关联角色的多重性，它表示一个整数的范围，通过多重性表达式来指名一组相关对象的可能个数。

需要从关联的另一端来理解多重性的定义，即表明另一端的一个对象可以与本方的多少个对象相链接。

15. 什么是关联类，有什么作用？

关联类是一种被附加到关联关系上的类，用来描述该关联关系自身所拥有的一些属性和行为。

当某些属于关联关系自身的特征信息无法被附加到关联两端的类时，就需要为该关联关系定义关联类。

16. 什么是聚合关系，聚合关系与关联关系、泛化关系有何不同？

聚合关系是一种特殊的关联关系，除了拥有关联关系所有的基本特征之外，两个关联的类还分别代表“整体”和“部分”，意味着整体包含部分。

与普通的关联关系相比，关联两端的类要多一层整体和部分的含义，而普通的关联关系并没有这层含义。

与泛化关系相比，关联是一种包含或拥有的关系，即整体包含或拥有部分；而泛化则是“是一种”的一般和特殊的关系，子类是一种特殊的父类。

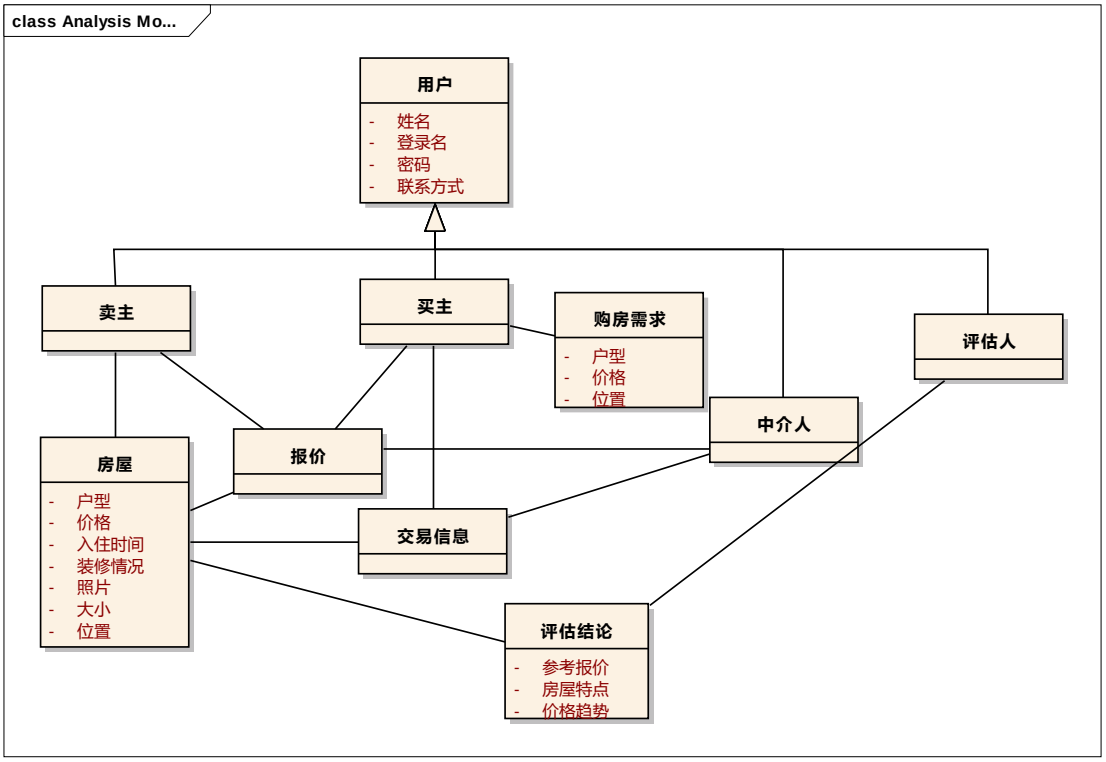
应用题

1. 参考类图如下图所示，主要考虑几个关键类以及它们之间的关系，其他辅助的类不作为考核重点：

   核心的类：房屋、评估结论、买主、卖主、购房需求、交易

   核心累的属性：房屋类、用户类的关键属性

   关系的使用：典型的关联关系（如交易和房屋、买主之间；评估结论与房屋之间），另注意泛化关系的使用。



2. [综合案例：考勤系统]

   参见“考勤系统参考答案和评分标准”

3. [综合案例：医院预约挂号系统]

   参见“医院预约挂号系统参考答案和评分标准”

第 6 章　面向对象设计原则

选择题

1. C
2. B
3. A

简答题

1. 什么是设计质量，如何评价设计质量？

设计的目标是按照需求的约定去描述软件系统，因此设计质量是设计满足需求的程度，也就是达到了 FURPS+所约定的需求指标的情况。

评价设计质量就是评价设计对需求的满足程度，包括功能性和非功能特性的评价，高质量的设计在满足功能需求的基础上，应该具有高可用性、高可靠性、高性能和高可支持性等特性。此外，还可以通过“设计的臭味”来识别糟糕的设计方案。设计的臭味主要体现在一下几个方面：（1）僵硬性：刚性，难以扩展；（2）脆弱性：易碎，难以修改；（3）牢固性：无法分解成可移植的组件；（4）粘滞性；（5）不必要的复杂性；（6）不必要的重复；（7）晦涩性。

2. 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。

3. 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 o_1，都存在一个类型 T 的对象 o_2，使得在所有针对 T 编写的程序 P 中，用 o_1 替换 o_2 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。

4. 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。

5. 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。

6. 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。

7. 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。

应用题

1. 主要要点：

方案 1：采用多重继承，实现起来比较简单，但需要编程语言支持，可能存在访问的二义性，且不易扩展。

方案 2：采用单继承加关联的方式，实现相对比较复杂，且与业务模式不一致，不易扩展和修改。

方案 3：单独抽象出权限分类的抽象接口，符合 LSP 设计原则，灵活可替代。

2. 主要要点：结合 JavaEE 一些主流的框架，如 Spring，讨论如何运用 DIP（或控制反转）隔离接口和具体实现，从而实现功能解耦和可替换性。

第 7 章　面向对象设计模式

选择题

1. B
2. A
3. C
4. D

简答题

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. 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。

3. 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。

4. 模式思维主要包括哪些方面的内容，在实践中如何有效地使用设计模式？

设计模式的根本意图是适应需求变更，而应对变更的直接手段就是封装变更，从而使变更的影响降到最小。其基本实现思路就是封装复杂性，并对外提供简单接口，通过多态包容的特性扩展新功能来应对变更。

多态包容是指宿主对象中包含抽象基类（或接口）的引用，而实际行为委托给该引用所指向的实际对象，从而使这些行为可以根据该引用所指向的实际对象不同而不同。具体的实现思路包括三个方面：（1）增加间接层；（2）针对接口编程，不针对实现编程；（3）优先使用聚合，而不是继承。

使用设计模式并不是简单的背诵和抄袭的过程。每个设计模式都有其应用背景（意图）和解决方案，只有在需要的场合选择合适的模式才能有效地发挥模式的作用，过度地滥用模式也会陷入过度设计，从而带来不必要的复杂性。初学者首先需要逐个学习设计模式的意图、适用性、解决方案等内容，并对其基本的使用有一定的了解。之后在实践中逐步地应用，通过应用来领悟设计模式的主旨和内涵，发现隐藏在设计模式背后模式思维和设计原则。

5. 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？

职责分配模式是指用于处理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。

6. 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。

应用题

1. 两种方案出发点不同，各有优缺点

第一种方案的出发点是如何消除 Button 对业务对象的依赖关系，通过增加间接层将通知机制委托给其他对象来实现 Button 行为与后续业务操作的解耦。

而第二种方案出发点则不考虑如何通知其他对象，而是只对 Button 本身的行为进行建模，相应的业务操作则单独设计子类来实现。

第一种方案中 Button 的设计逻辑相对较为复杂，但应用比较简单。而第二种方案中 Button 的行为直接抽象成一个命令，Button 的设计逻辑较为简单；而具体的行为则需要实现不同的类来支持。但该方案可扩展性更好，可以针对不同类型的行为实现不同的逻辑，如双击、右击、撤销等操作都很方便实现。

2. 以自己参与的实际项目，或者作业实践为背景讨论。一般类似于状态模式、策略模式、工厂方法、单例等模式应用比较广泛。
3. 很多 UML 工具，提供了 GoF 模式的建模模板，结合具体工具如 EA 实践具体的设计模式。
4. [综合案例：医院预约挂号系统]

参见“医院预约挂号系统参考答案和评分标准”

第 8 章　架构设计

选择题

1. D
2. C
3. A
4. B
5. A
6. A
7. D
8. B
9. D
10. C
11. A
12. A
13. C
14. B
15. C
16. （1）C　　（2）D

简答题

1. 面向对象的分析和设计有何区别和联系？

在面向对象的方法论中，设计是分析的自然延续，是对分析模型的进一步细化。与分析一样，设计活动也是针对 UML 静态和动态模型开展，作为设计成果的设计模型的表现形式与分析模型类似。

但分析和设计的出发点和关注点不同，这两个活动在具体开展过程中存在着很大的差别。这种差别体现在其目标不同，分析的目标是明确了做什么，而设计则讨论怎么做的问题。更具体来说，分析重点关注系统的业务问题本身，在不考虑实现技术的基础上有效地确定了将要构建的内容；而设计则关注系统的技术和实现细节，重点考虑采用何种技术、何种平台来实现分析模型。

2. 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。

3. 包之间的依赖关系是意味着什么，除了普通的依赖，还可以定义哪些关系？

包之间的依赖关系包含两层含义：其一是被依赖包的改变将影响到依赖包；其二则是依赖包不能够独立的复用，因为它依赖于被依赖包。

除了普通的依赖关系，可以通过构造型进一步扩展不同的依赖关系，如合并、导入和访

问等。

4. 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。

5. 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。

6. 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。

7. 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。

8. 什么设计机制，它和分析机制、实现机制有何区别和联系？

设计机制是一类架构机制，是相应分析机制在设计阶段的具体定义，为概念上的分析机制添加具体的设计细节，但不需要具体的代码细节。实现机制则是更具体的实现细节，与特定的编程语言紧密相关。

9. 什么情况下需要设计系统的运行时架构，如何设计运行时架构？

对于那些有并发访问需求，而需要建立多进程（或线程）的应用系统而言，就必须建模该系统的进程视图，以准确地描述系统的运行时架构。

设计运行时架构主要的工作包括：首先描述并发需求，之后为相应的并发业务设计独立的控制进程或线程，即对系统的进程和线程进行建模；最后将设计元素部署到相应的进程或线程中。

10. 有哪些典型的分布模式，利用什么 UML 模型描述系统部署？

分布模式包括：客户/服务器模式和对等模式，客户/服务器模式又细分为三层结构、“胖”客户结构、“胖”服务器结构和分布式客户/服务器结构等。可利用 UML 的部署图描述系统部署结构。

应用题

1. [综合案例：员工考勤系统]
   参见“考勤系统参考答案和评分标准”
2. [综合案例：医院预约挂号系统]
   参见“医院预约挂号系统参考答案和评分标准”

第 9 章　构件设计

选择题

1. C
2. C
3. C
4. B
5. A
6. C
7. C
8. C
9. D
10. C
11. （1）D　　（2）A　　（3）D　　（4）D

简答题

1. 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。

2. 什么情况下可以将用例事件流中的交互封装为独立的子系统，封装之后有什么好处？

当出现下列情况时，可以考虑把将交互图中的子流封装成独立的子系统：

- ◆ 子流在不同的用例实现中重复出现。也就是说，相同（或相似）的消息发送给相同（或相似）的对象，产生相同的最终结果。
- ◆ 子流只在一个用例实现中出现，但期望在后期的迭代或者在后续相似系统中扩展或复用。
- ◆ 子流只在一个用例实现中出现，相对比较复杂但却很容易被封装，它需要独立人或

者团队单独设计并实现，并且有明确的输入和输出。在这种情况下，复杂行为通常要求具备专门的技术或领域知识，因此适合将其封装在子系统中独立设计实现。

◆ 被封装在单独的构件中实现的子流，比如某些业务确定采用第三方构件来实现

利用子系统封装交互有效地提高了用例实现事件流的抽象级别，从而使得用例实现的结构相对比较清楚，较少混乱，尤其针对在那些非常复杂的交互而言。同时，这种方式可以在完成子系统内部设计之前可以创建用例实现，以利于并行开发。此外，封装使得用例实现变得更加通用，也更容易适应变更，因为子系统是可替换的，只要保持接口不变。

3. 什么是子系统的代理类，子系统的接口和代理类有何区别和联系？

在子系统设计时，为了便于在交互图中描述子系统，可以为每个子系统定义特定的代理类，其操作即为子系统的职责。与接口不同的是，代理类在子系统内部，对外代表特定的子系统；而接口在子系统外面，代表所有实现该接口的子系统。

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. 什么是操作和属性的可见性，有哪几种可见性？

可见性是指操作或属性可以被外界访问的程度。UML 规范定义了四种可见性：公有、私有、保护和包可见性。

7. 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。

8. 什么情况下需要进行类的状态建模，如何进行状态建模？

在类设计期间，针对那些受状态影响的对象进行状态建模，从而可以描述该对象所能够响应的事件、对这些事件的响应以及过去对当前行为的影响等方面的问题。

状态建模过程需要从几个方面展开：

- ◆ 哪些对象有重要的状态，需要进行状态建模；
- ◆ 针对需要进行状态建模的对象，如何确定该对象可能的状态；并分析状态之间的转移，完成状态机模型；
- ◆ 如何将状态模型中的状态和事件信息映射到模型的其它部分

9. 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。

10. 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。

11. 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。

12. 类间的泛化关系有什么优点和缺点？

优点：通过使用泛化关系可以实现代码的复用和对多态的支持。

缺点：

- ◆ 类间可能耦合的最强形式：子类会继承父类的所有的属性、方法和关系。
- ◆ 类层次中的封装是脆弱的：父类的改动会直接波及所有下层的所有子类。
- ◆ 在大多数语言中，继承是不能轻易改变：这种泛化关系是在编译时确定的，运行时是固定的，不能改变。

13. 面向对象的设计中，数据库设计阶段需要考虑哪些问题？

数据库设计主要包括以下几个方面的工作：

- ◆ 确定设计中需要存储的持久性类：这部分工作在前面的分析和设计中已同步开展，通过持久化构架机制对持久化类进行了说明。
- ◆ 设计适当的数据库结构以存储持久化类：数据库设计阶段的核心内容，需要根据已有的对象模型设计对应的数据模型（实体关系模型）。
- ◆ 为存储和访问持久化数据定义机制和策略，以满足系统的性能要求：遵循架构设计中的相关设计机制的所提出的策略，实现数据的存储和访问。

14. 如何将对象模型映射为数据模型？

可以利用一定的映射规则从对象模型中直接构造数据模型，从而简化数据建模过程，主要的规则包括：

映射类和属性：把每个需要持久化的实体类映射成一张表，持久化属性对应表中的字段；类的对象对应表中的记录。

映射关联关系：类之间的关联关系在数据模型中通过主外键的约束来表达，根据多重性的不同，有不同的映射规则。

映射泛化关系：数据模型没有提供泛化关系的直接实现机制，可以采用不同的设计方案来实现泛化关系

应用题

1. [综合案例：员工考勤系统]
   参见“考勤系统参考答案和评分标准”
2. [综合案例：医院预约挂号系统]
   参见“医院预约挂号系统参考答案和评分标准”

第 10 章 从模型到代码

选择题

1. D

简答题

1. 什么是正向工程，一般可以对哪些 UML 设计模型进行正向工程？

   正向工程是指按照软件开发的基本过程，将抽象层次较高的模型转换为相对具体的模型的过程。可以对一下三类 UML 设计模型进行正向工程，以生成目标代码：（1）从类图生成框架代码。（2）从交互图（主要指顺序图）生成方法中操作的调用代码。（3）从状态机图生成状态转换控制代码。
2. 由类图生成代码时，类（或接口）之间的依赖、关联、聚合、组合、泛化和实现这 6 种关系分别如何处理？

   依赖关系：没有具体的代码实现；

   关联关系：在有导航性的一段添加引用属性，提供对目标对象的访问；该引用属性的类型为目标类类型，名称为对方引用的端点名（没有定义端点名则采用一种默认的规则生成属性名称），可见性为端点的可见性。对于多重性为多的关联关系，还需要添加合适的容器类来实现。聚合和组合关系与关联相同

   泛化和实现关系：一般语言都提供了而相应的语法直接支持。
3. 什么是逆向工程，一般什么时候使用逆向工程？

   逆向工程是正向工程的逆操作，即根据已有的源代码获得其设计模型。

   逆向工程主要有两种使用场合：

   （1）在编码时，可能会存在和设计模型不一致的地方，可以通过逆向工程更新原有的

设计模型，从而需要保持设计模型的有效性。

（2）针对已有的系统，缺少或丢失了设计文档时，可以通过逆向工程重新获得系统的设计模型，以便理解程序和完善文档。

应用题

1. （1）getEPG();　close()

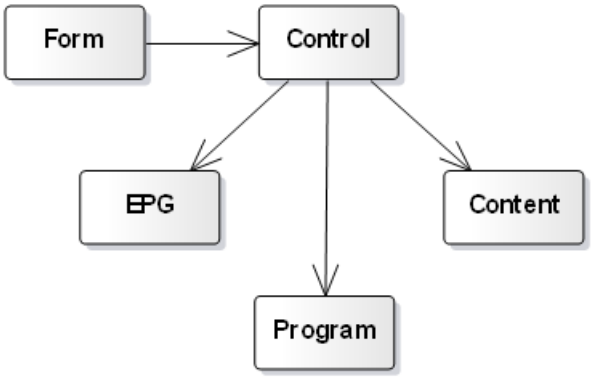
   （2）Control.initRecord();

   EPG.close();

   Program.init();

   Content.save();

   （3）5 个类之间最可能的关系如图所示：



2. 建议在 GitHub 上克隆一个开源项目源码，利用 UML 工具逆向工程建立设计模型，之后再修改设计模型后同步到代码。
3. 建议参照 OMG 相关的标准和规范，以及一些相关的学术论文，调研该领域的发展现状。