实验九 UML，逻辑， 软件体系结构设计（一）

实验目的：

1. 深入理解UML

2. 了解计算机学科中的逻辑

3. 学习对比软件体系结构设计GB和IEEE最新SAD (Software Architecture Document)的标准

4. 研究经典软件体系结构案例

3. 完成自己项目的SRS

实验内容：

**1. 阅读“The Unified Modeling Language Reference Manual”，进一步学习UML知识，理解如何应用UML对系统进行建模**

1. **模型**

"modeling"指的是使用UML来描述软件系统的过程。这包括使用UML的不同图形表示法（比如类图、用例图、活动图等）来捕捉系统的不同方面，如结构、行为和交互。通过建模，软件工程师可以更好地理解系统的需求、设计和实现，并与利益相关者进行有效的沟通。

简单理解：是对现实的简化，是对系统从语义上近似的抽象模型的作用：

精确捕获和表达项目的要求与应用领域中的知识，以使各方面利益相关者能够理解并达成一致，便于用户和各个领域的专家评审。

**建模的原则**：

简化原则（Principle of Simplicity）： 建模应该尽量简洁和简单，避免过度复杂化。简化原则要求我们使用最少的元素和关系来描述系统，以便提高可理解性和可维护性。

抽象原则（Principle of Abstraction）： 建模应该从系统中抽象出最重要的概念和特征，忽略不必要的细节。通过抽象，我们可以关注系统的核心问题，提高模型的可读性和可扩展性。

一致性原则（Principle of Consistency）： 建模应该遵循一致的规则和约定，以确保模型的内部一致性和与其他模型的一致性。一致性原则要求我们使用相同的符号、标记和命名规则，避免混淆和歧义。

精确性原则（Principle of Precision）： 建模应该准确地反映系统的特性和行为，避免模糊和不明确的描述。精确性原则要求我们使用准确的术语和概念，明确地定义对象、关系和行为。

可视化原则（Principle of Visualization）： 建模应该使用图形化表示法来可视化系统的结构和行为。可视化原则要求我们使用适当的图形符号和图表，以便更直观地传达模型的信息。

可重用性原则（Principle of Reusability）： 建模应该鼓励模块化和可重用性，以便在不同的上下文中重复使用模型的部分。可重用性原则要求我们设计模型时考虑到模块的独立性和通用性，以便提高开发效率和系统的可维护性。

迭代原则（Principle of Iteration）： 建模是一个迭代的过程，需要不断地进行反馈和改进。迭代原则要求我们在建模过程中不断地审查和修正模型，以适应需求变化和新的理解。

**(2) 面向对象的基本观点**

客观世界由对象组成。具有相同数据和相同操作的对象可以归并为一个类。从一个类可以派生出许多对象。

类可以派生出子类，子类和父类形成类的层次结构。

对象之间通过消息相互联系。面向对象=对象+类+继承+通信

一般/特殊（分类）结构：表现事务的一般与特殊的关系。术语称为泛化与特化的关系。

组装结构：表现对象类之间的部分与整体关系。

实例连接：表现对象间的静态联系。通过对象的属性来表现对象之间的关系。

消息连接：对象之间的通信联系。表现对象行为的动态联系。

**(3) 面向对象分析的基本过程**

发现对象：从对象中抽象出类的定义。

识别对象的内部特征：定义属性和服务。识别对象的外部关系：建立一般/特殊结构、整体/部分结构、实例连接、消息连接

划分主题，建立主题图

定义用例，建立交互图：发现活动者、定义用例

建立详细说明：模型的详细定义与解释，分散在其它活动之中

原型开发：可反复进行

**(4) 面向对象分析的层次**

对象模型通常由五个层次组成：

类与对象层：对象是数据及其处理的抽象。它反映了保存有关信息和与现实世界交互的能力

属性层：属性是数据元素，用来描述对象或分类结构的实例，可在图中给出并在对象的储存中指定，即给出对象定义的同时，指定属性

服务层：服务是接收到消息后必须执行的一些处理，可在图上标明它并在对象的储存中指定，即给出对象定义的同时，定义服务。 五个层次就像合并在一起的五个透明的图层一样，每一层从不同角度将对象模型更细化、更具体化

结构层：结构表示问题域的复杂性。类 - 成员结构反映了一般 - 特殊关系，整体 - 部分结构反映了整体 - 部分的关系

主题层：主题给出分析模型的总体概貌，是控制读者在同一时间所能考虑的模型规模的机制

面向对象的模型的五个层次对应着分析建模的五个主要活动。这五个活动的工作可以不按顺序进行，也没必要完成一项活动后才开始另一项活动工作。也就是说，五个主要活动可以同时（并行）处理；可以从较高抽象层转移到较低的具体层，然后再返回到较高抽象层继续处理；当系统分析员在确定类-&-对象的同时，想到该类的服务，则可以先确定服务后，再返回去继续寻找类-&-对象；没有必要遵循自顶向下，逐步求精的原则。

**(5) 面向对象分析的基本原则**

抽象原则（类 对象 一般类 属性 服务）

分类原则（把具有相同属性和服务的对象化为一类）

聚合原则（将整体化为若干部分）

关联原则（用关联原则可以明确的表示对象之间的静态联系，实例连接）

消息通信原则（动态联系）

**(6) 面向对象设计基本过程**

设计对象与类

设计系统结构：一个软件由若干子系统组成，一个子系统由若干组件构成，设计系统结构的主要任务就是设计组件与子系统，以及它们相互的静态和动态关系。

系统结构两种形式：层次结构、块状结构

设计问题论域子系统：问题论域子系统负责实现领域的业务服务。应包括与应用领域的业务直接相关的全部类与对象。对OOA模型中的某些类、对象、结构、属性、操作进行调整

设计人机交互子系统：主要任务是设计系统界面。内容包括：用户分类、描述交互场景、设计人机交互操作命令、操作顺序，设计人机交互类如窗口、对话框、菜单等

设计数据管理子系统：数据管理子系统负责数据的管理：录入、操纵、检索、存储、输出等。设计数据管理子系统的主要任务是：确定数据管理方法，设计数据库逻辑结构和物理结构，设计实现数据管理的对象类

设计任务管理子系统：一个系统运行时将有相当数量的进程被激活执行。任务管理子系统的任务就是协调和管理进程。设计软件系统内部模块运行的管理机制，把事件驱动、时钟驱动、优先级管理等任务分配给软、硬件执行

设计优化，提高系统的性能：设计的结果要尽可能的优化，尽量提高系统的性能。各性能指标之间常存在矛盾，应权衡考虑选取一个折衷方案。 准则：弱耦合、强内聚、减少通信开销、良好的可扩充性

**(7) 基于UML的面向对象分析设计过程**

发现对象，从对象中抽象出类的定义。

识别对象的内部特征，定义属性和服务。

识别对象的外部关系，建立一般/特殊结构、整体/部分结构、实例连接、消息连接。

划分主题，建立主题图。

定义用例，建立交互图。发现活动者、定义用例。

建立详细说明，模型的详细定义与解释，分散在其它活动之中。

原型开发，可反复进行

**(8) UML中的视图**

视图的分类：

用例视图 系统使用实例

逻辑视图 系统逻辑结构

组件视图 系统构成

并发视图 系统并发特性

配置视图 系统的配置

这5种视图组合构成UML完整模型。

下面详细分析这五种视图：

用例视图（Use Case View）：

用例视图描述了系统的功能需求和用户需求，以用例（Use Case）为中心。

用例视图包括用例图和用例说明文档。用例图展示了系统的功能模块和各个参与者之间的交互关系，用例说明文档则详细描述了每个用例的功能和行为。

用例视图帮助系统开发者和用户之间共同理解系统的功能和行为，以便确定系统的需求和范围。

逻辑视图（Logical View）：

逻辑视图描述了系统的结构和组织，以类图（Class Diagram）、对象图（Object Diagram）和包图（Package Diagram）为主要表示方式。

类图展示了系统中的类、接口、关联关系和继承关系，对象图展示了类的实例及其之间的关系，包图展示了系统中的包（Package）及其之间的依赖关系。

逻辑视图帮助系统开发者理解系统的静态结构和组织方式，以便进行系统的设计和实现。

组件视图（Component View）：

组件视图描述了系统的组件及其之间的依赖关系，以组件图（Component Diagram）为主要表示方式。

组件图展示了系统中的组件、接口、依赖关系和连接关系，以及组件之间的通信和交互方式。

组件视图帮助系统开发者理解系统的组织结构和模块化程度，以便进行系统的组件化设计和实现。

并发视图（Concurrency View）：

并发视图描述了系统中的并发和同步机制，以序列图（Sequence Diagram）、协作图（Collaboration Diagram）和状态图（Statechart Diagram）为主要表示方式。

序列图和协作图展示了系统中的对象之间的消息传递和交互顺序，状态图展示了系统中的对象状态和状态转换。

并发视图帮助系统开发者理解系统的并发行为和同步机制，以便进行系统的并发设计和实现。

配置视图（Deployment View）：

配置视图描述了系统的部署和配置方式，以部署图（Deployment Diagram）为主要表示方式。

部署图展示了系统中的节点（Node）及其之间的连接关系，以及节点上部署的组件和软件配置。

配置视图帮助系统开发者理解系统的部署环境和配置需求，以便进行系统的部署和维护。

**(9) UML中包含九种图**

主要分为两类：静态图和动态图。

静态图 UML中有5种静态图：用例图、类图、对象图、组件图和配置图。

动态图：

UML中有4种动态图：时序图、协作图、状态图、活动图

**(10) UML中的事物**

UML中的事物包括结构事物、行为（动作）事物、组织（分组）事物、注释事物。

结构事物 结构事物包括7种：类、接口、协作、用例、活动类、组件和节点。

动作事物 动作事物主要有两种：交互和状态机。

分组事物 分组事物是UML模型中负责分组的部分，可以把它看作一个个盒子，每个盒子里面的对象关系相对复杂，而盒子与盒子之间的关系相对简单。成组事物只有一种：包（包是一种有组织地将一系列元素分组的机制）

注释事物 属于这一类的只有注释。注释即是UML模型的解释部分。在UML图中，一般表示为折起一角的矩形。

**(11) UML通用机制**

修饰：为图中的模型元素增加了语义。

注释：以自由的文本形式出现的。

信息类型是不被UML解释的一个字符串。

规格说明 UML 中预定义的特性：

文档（Documentation）

职责（Responsibility）

永久性（Persistence）

并发性（Concurrency）

通用划分

两种通用划分(General Division)：

型-实例（值）：描述一个通用描述符与单个元素项之间的对应关系。

接口-实现

扩展机制

构造型

由建模者设计的新的模型元素。新的模型元素的设计要以UML已定义的模型元素为基础。

标记值

附加到任何模型元素上的命名的信息块。

约束

用某种形式化语言或自然语言表达的语义关系的文字说明。

**2. 浏览“LOGIC IN COMPUTER SCIENCE--Modelling and Reasoning about Systems”，了解常用逻辑及其在计算机学科中的应用**

首先，逻辑推理是逻辑学中的一个核心概念，也是计算机科学中应用最广泛的一个方面。逻辑推理允许我们根据已知的事实和规则来推导出新的结论。在计算机科学中，逻辑推理被广泛应用于编程语言的设计和开发过程。程序设计的核心就是根据给定的问题，在计算机上实现一个可执行的解决方案。在这个过程中，逻辑推理帮助程序员分析和理解问题的逻辑关系，构建正确的解决方案。通过应用逻辑推理，程序员可以有效地优化算法和提高程序的效率。

在软件工程方面，逻辑推理也发挥了重要的作用。软件工程是一门关于软件开发、维护和管理的学科，而逻辑推理可以帮助软件工程师分析和处理软件系统中的各种逻辑问题。例如，在软件测试中，逻辑推理可以帮助工程师设计有效的测试用例，以覆盖系统中所有可能的逻辑情况。

此外，命题逻辑和谓词逻辑在关系数据库中也得到了应用。数据库是计算机数据处理的核心部分，同时也是当代计算机技术的重要组成部分。而数据库管理系统是关系数据库中用于向用户提供使用数据库的语言，也被称为数据子语言。谓词逻辑是计算机中的数学基础，对于数据子语言的改进和优化也成为谓词逻辑的化简问题。命题逻辑作为逻辑系统中最基本的一种逻辑，可以将推理归纳成简单的代数演算——命题演算。而一阶谓词逻辑既能对众多非古典逻辑奠定基础，又能找出新语句的导出来源，为“归结反演原理”提供求解方法。

综上所述，逻辑在计算机科学中的应用是广泛而深入的，从编程语言的设计和开发到软件工程的维护和管理，再到关系数据库的管理，都离不开逻辑的应用。这本书为读者提供了深入了解常用逻辑及其在计算机科学中应用的宝贵资源。

**3. 分工协作，参考国标“13 - 软件(结构)设计说明(SDD)”等资料，对比参考SAD最新标准IEEE-42010.pdf，针对自己的项目设计SAD初稿。**

**4. 分工协作，学习、检索研究经典软件体系结构案例。**

On-the-Criteria-To-Be-Used-in-Decomposing-Systems-into-Modules.pdf

<http://www.cs.cmu.edu/~ModProb/index.html>

**（1）数据共享 体系结构风格**

数据共享风格也称为仓库风格。

在这种风格中，有两种不同类型的软件元素：一种是中央数据单元，也成为资源库，用于表示系统的当前状态；另一种是相互依赖的构件组，这些构件可以对中央数据单元实施操作。中央数据单元和构件之间可以进行信息交换，这是数据共享体系结构的技术实现基础。

根据所使用的控制策略不同，数据共享体系结构可以分为两种类型，一种是传统的数据库，另一种是黑板。

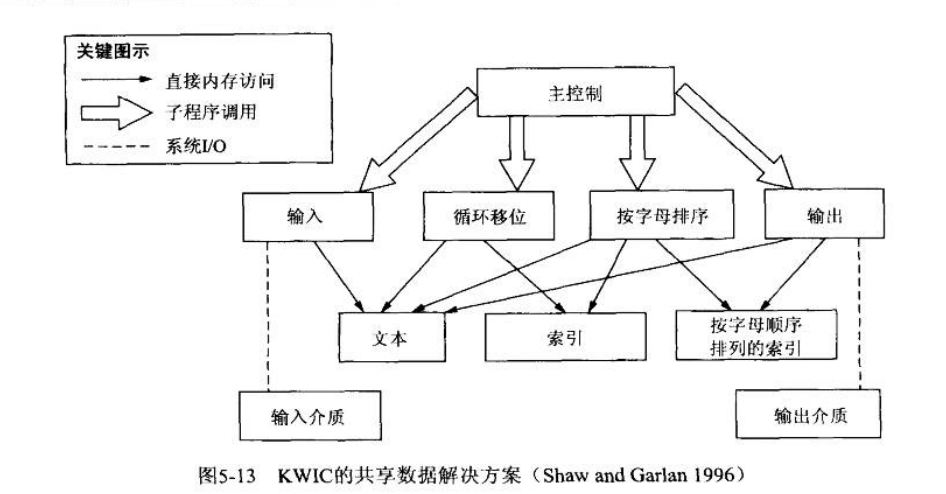
如果由输入流中的事件来驱动系统进行信息处理，把执行结构存储到中央数据单元，则这个系统就是数据库应用系统。

如果由中央数据单元的当前状态来驱动系统运行，则这个系统就是黑板应用系统。

黑板是数据共享体系结构的一个特例，用以解决状态冲突并处理可能存在的不确定性知识源。

黑板常用于信号处理，如语音和模式识别，同时在自然语言处理领域中也有广泛的应用，如机器翻译和句法分析。

原理图：



一个典型的黑板系统主要包括知识源、中央控制单元、控制单元。

优点：

便于多客户共享大量数据，而不必关心数据是何时产生的、由谁提供的及通过何种途径来提供；

便于将构件作为知识源添加到系统中来；

缺点：

对共享数据结构，不同知识源要达成一致；

需要同步机制和加锁机制来保证数据的完整性和一致性，增大了系统设计的复杂度；

**（2）数据模块解决方案**

本解决方案通过构建一个KWIC数据模块，实现了对文本数据的快速处理和展示。该模块主要包括以下几个部分：

文本预处理：对输入的文本进行清洗、分词、去除停用词等预处理操作，为后续的关键词提取和上下文分析打下基础。

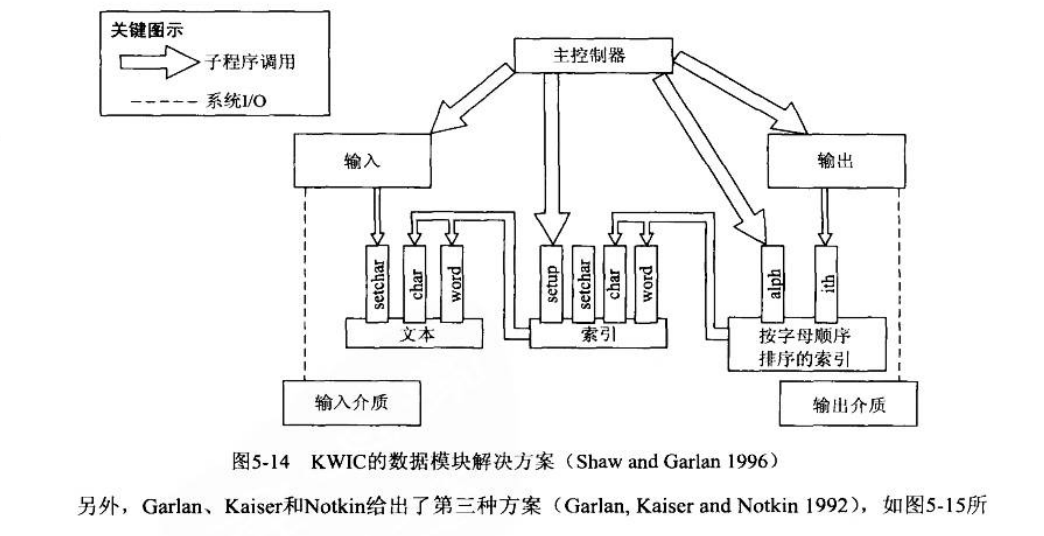
关键词提取：利用自然语言处理技术，从预处理后的文本中提取出关键词。关键词的提取可以根据具体需求进行定制，例如基于词频、TF-IDF等方法。

上下文提取：对于每个提取出来的关键词，从文本中提取其前后的上下文信息，形成一个包含关键词及其上下文的记录。上下文的长度可以根据需要进行设置，以确保能够完整地展示关键词的语境。

数据存储：将提取出来的KWIC记录存储到数据库中，以便后续的查询和分析。数据库的设计应考虑到查询效率和扩展性。

数据展示：通过前端界面或API接口，将KWIC数据展示给用户。用户可以根据关键词进行检索，并查看相应的KWIC记录，以便快速了解关键词在文本中的上下文信息。

原理图：



优点：

数据模块解决方案可以根据具体需求选择合适的数据存储方式，如数据库、文件或内存结构，从而提供灵活性和可扩展性；

将数据处理流程划分为多个模块，有利于系统的维护和扩展，每个模块可独立开发和测试；

缺点：

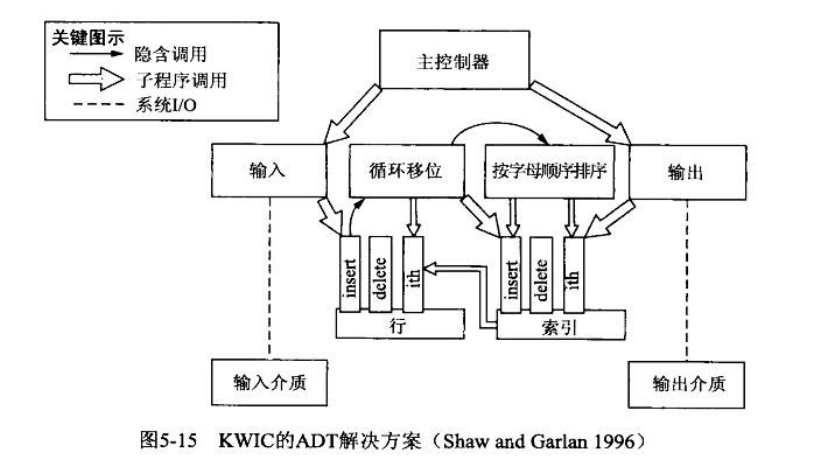
数据模块解决方案可能涉及多个模块之间的协调和通信，需要处理各种数据格式和接口，增加了系统的复杂性；

不同模块之间的数据一致性和同步可能需要额外的考虑和处理，避免数据不一致或冲突；

1. **隐含调用解决方案**

KWIC（Keyword in Context）的隐含调用解决方案是通过一系列设计和技术实现的，它允许系统中的各个组件以隐式的方式进行交互，无需显式地调用特定的函数或方法。这种解决方案通常采用事件驱动模型、观察者模式、数据流处理等技术，确保文本从输入到关键词提取、上下文构建，再到最终展示的过程能够高效、流畅地进行。通过中间件或消息队列，组件之间可以异步通信，实现分布式处理，提高系统的吞吐量和可扩展性。同时，依赖注入和配置管理等技术也被用来管理组件之间的依赖关系和系统的配置信息，使系统更加灵活和可维护。最终，模块化设计使得KWIC系统易于扩展和修改，而严格的测试和验证则保证了系统的正确性和可靠性。这种隐含调用的解决方案不仅简化了代码结构，还提高了系统的可维护性和可扩展性**。**

原理图：



优点：

代码简洁性：通过隐式调用，可以减少显式的函数调用或方法调用，使代码更加简洁和易于阅读。

灵活性：隐式调用允许系统组件之间的动态交互，使系统更加灵活，易于适应不同的需求和场景。

可扩展性：由于组件之间的依赖关系是通过事件或消息传递的，因此可以轻松添加新的组件或功能，而无需修改现有代码。

解耦性：隐式调用有助于降低组件之间的耦合度，使系统更加模块化，易于维护和测试。

缺点：

复杂性：隐式调用的实现可能比显式调用更复杂，需要深入理解事件驱动、观察者模式等底层机制。

调试困难：由于组件之间的交互是隐式的，因此在调试过程中可能难以追踪问题的来源和流程。

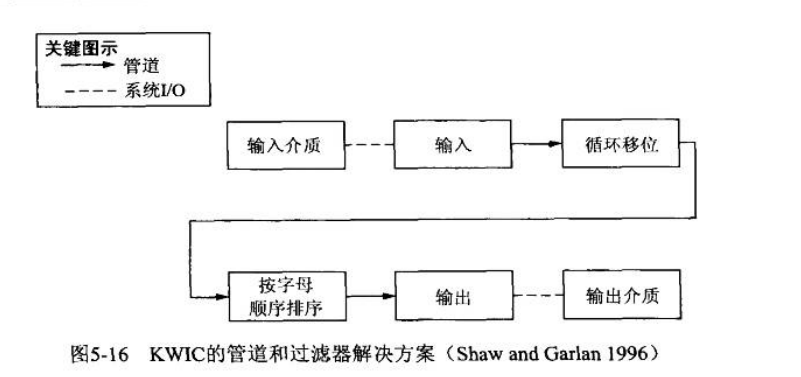
性能开销：隐式调用可能涉及额外的消息传递和事件处理，这可能会增加系统的性能开销。

依赖管理：隐式调用可能导致组件之间的依赖关系变得复杂，需要采用有效的依赖管理工具来管理这些依赖关系。

1. **管道和过滤器解决方案**

主要包括过滤器和管道两种元素。在这种结构中，构件被称为过滤器，负责对数据进行加工处理。每个过滤器都有一组输入端口和输出端口，从输入端口接收数据，经过内部加工处理之后，传送到输出端口上。数据通过相邻过滤器之间的连接件进行传输，连接件可以看作输入数据流和输出数据流之间的通路，这就是管道。

原理图：



优点：

简单性。

支持复用。

系统具有可扩展性和可进化型。

系统并发性（每个过滤器可以独立运行，不同子任务可以并行执行，提高效率）。

便于系统分析。

缺点：

系统处理工程是批处理方式。

不适合用来设计交互式应用系统。

由于没有通用的数据传输标准，因此每个过滤器都需要解析输入数据和合成数据。

难以进行错误处理。

**5. 完成软件需求规格说明SRS**