**实验九 UML，逻辑， 软件体系结构设计（一）**

**夏思晓 202100031020**

**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

**二、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。**

统一建模语言（UML）是一个通用的可视化建模语言，用于对软件进行描述、可视化处理、构造和建立软件系统制品的文档。它记录了对必须构造的系统的决定和理解，可用于对系统的理解、设计、浏览、配置、维护和信息控制。UML 适用于各种软件开发方法、软件生命周期的各个阶段、各种应用领域以及各种开发工具，UML 是一种总结了以往建模技术的经验并吸收当今优秀成果的标准建模方法。UML包括概念的语义，表示法和说明，提供了静态、动态、系统环境及组织结构的模型。它可被交互的可视化建模工具所支持，这些工具提供了代码生成器和报表生成器。UML标准并没有定义一种标准的开发过程，但它适用于迭代式的开发过程。它是为支持大部分现存的面向对象开发过程而设计的。

　　UML描述了一个系统的静态结构和动态行为。UML将系统描述为一些离散的相互作用的对象并最终为外部用户提供一定的功能的模型结构。静态结构定义了系统中的重要对象的属性和操作以及这些对象之间的相互关系。动态行为定义了对象的时间特性和对象为完成目标而相互进行通信的机制。从不同但相互联系的角度对系统建立的模型可用于不同的目的。

UML还包括可将模型分解成包的结构组件，以便于软件小组将大的系统分解成易于处理的块结构，并理解和控制各个包之间的依赖关系，在复杂的开发环境中管理模型单元。它还包括用于显示系统实现和组织运行的组件。

应用UML对系统进行建模的过程如下：

**（1）需求分析**：确定系统的目标和功能需求。通过用例图（Use Case Diagram）描述系统的主要功能、参与者（用户或其他系统）以及他们之间的交互。

**（2）静态结构建模**：通过使用类图（Class Diagram）描述系统的静态结构，包括类、接口、属性以及它们之间的关系（如继承、关联等）；使用对象图（Object Diagram）展示特定时间点的类实例以及它们之间的关系；使用包图（Package Diagram）对系统中的类和其他元素进行分组，以便更好地组织和管理模型。

**（3）动态行为建模**：通过序列图（Sequence Diagram）和协作图（Collaboration Diagram）描述对象之间的交互和消息传递顺序；通过状态图（State Diagram）展示对象在其生命周期内的状态变化以及触发状态变化的事件；通过活动图（Activity Diagram）描述系统中并发和顺序行为的流程。

**（4）系统环境建模**：通过组件图（Component Diagram）描述系统的物理结构，包括软件组件、它们之间的依赖关系以及它们如何与外部环境交互；通过部署图（Deployment Diagram）展示软件组件如何映射到硬件节点以及它们之间的通信路径。

**（5）模型验证与优化**：对建立的模型进行验证，确保其符合系统需求。根据验证结果对模型进行优化和调整。

**（6）生成文档**：利用UML工具从模型中自动生成代码框架、接口定义等。编写相关文档，如用户手册、开发者指南等，以便后续的开发和维护工作。

**（7）迭代与演化**：在系统开发过程中，随着需求的变更和系统的演化，需要不断更新和完善UML模型。通过持续的迭代和反馈，确保UML模型始终与实际系统保持一致。

**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。**

**（1）命题逻辑（Propositional Logic）：**是计算机科学中最基本的逻辑形式之一，它研究的是命题之间的逻辑关系，如“与”、“或”、“非”等。在计算机科学中，命题逻辑常用于描述和验证程序的正确性。例如，在形式化验证中，程序员可以使用命题逻辑来描述程序的行为，并通过逻辑推理来验证程序是否满足预期。

**（2）谓词逻辑（Predicate Logic）：**与命题逻辑不同，谓词逻辑可以描述更为复杂的关系和属性，包括量化关系（如“所有”和“存在”）。在人工智能领域，谓词逻辑被用于知识表示和推理。通过构建基于谓词逻辑的知识库，系统可以理解和回答更为复杂的问题。

**（3）模态逻辑（Modal Logic）**：研究带有模态词（如“可能”和“必然”）的命题。在计算机科学中，模态逻辑常用于描述系统的动态行为和属性。例如，在并发系统或分布式系统中，模态逻辑可以用于描述和验证系统的状态变迁和性质。

这些逻辑形式在计算机科学中的应用不仅限于理论层面，它们还广泛应用于实际开发中。例如，在编程语言设计和开发中，逻辑推理被用于构建类型系统、优化编译器以及实现自动化错误检测。此外，在软件工程中，逻辑推理也被用于需求分析、系统设计以及测试等方面。

1. **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。**

（见小组项目设计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. **管道/过滤器体系结构风格**

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。

①优点：简单；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。

②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。

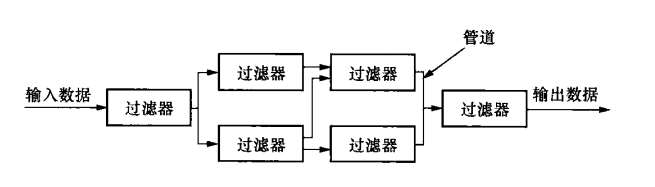


图 1 管道/过滤器体系结构风格原理图

传统的编译器就是管道/过滤器体系结构风格的一个实例。编译器由词法分析、语法分析、语义分析、中间代码生成、中间代码优化和目标代码生成几个模块组成，一个模块的输出是另一个模块的输入。源程序经过各个模块的独立处理之后，最终将产生目标程序。编译器的框架结构如图：

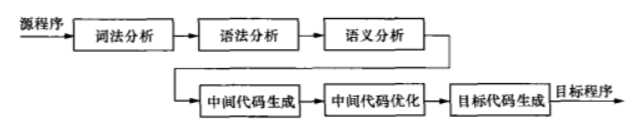


图 2 编译器框架结构

1. **面向对象体系结构风格**

在面向对象体系结构中，软件工程的模块化、信息隐藏、抽象和重用原则得到了充分的体现。在这种体系结构中，数据表示和相关原语操作都被封装在抽象数据类型中。在这种风格中，对象是构件，也成为抽象数据类型的实例。对象与对象之间，通过函数调用和过程调用来进行交互。

①优点：一个对象对外隐藏了自己的详细信息；对象将数据和操作封装在一起；继承和封装方法为对象服用提供了技术支持。

②缺点：如果一个对象要调用另一个对象，则必须知道它的标识和名称；会产生连锁反应。

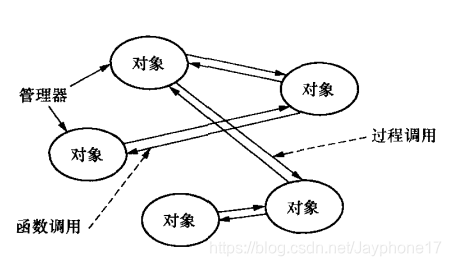


图 3 面向对象体系结构风格原理图

1. **事件驱动体系结构风格**

事件驱动就是在当前系统的基础之上，根据事件声明和发展状况来驱动整个应用程序运行。事件驱动体系结构的基本思想是：系统对外部的行为表现可以通过它对事件的处理来实现。在这种体系结构中，构件不再直接调用过程，而是声明事件。系统其他构件的过程可以在这些事件中进行注册。当触发一个事件的时候，系统会自动调用这个事件中注册的所有过程。因此，触发一个事件会引起其他构件的过程调用。

①优点：事件声明者不需要知道哪些构件会响应；提高了软件复用能力；便于系统升级。

②缺点：构件放弃了对计算的控制权，完全由系统来决定；存在数据·传输问题。

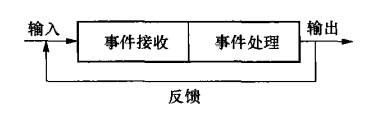


图 4 事件驱动体系结构风格原理图

1. **分层体系结构风格**

在分层风格中，系统将划分为一个层次结构。每一层都具有高度的内聚性，包含抽象程度一致的各种构件，支持信息隐藏。分层有助于将复杂系统划分为独立的模块，从而简化程序的设计和实现。通过分解，可以将系统功能划分为一些具有明确定义的层，较高层是面对特定问题，较低层具有一般性。每层都为上层提供服务，同时又利用了下层的逻辑功能。在分层体系结构中，每一层只对相邻层可见。层次之间的连接件是协议和过程调用。用以实现各层之间的交互。

①优点：设计者可以将系统分解为一个增量的步骤序列从而完成复杂的业务逻辑；每一层之多和相邻的上下两层进行交互；只要给相邻层提供相同的接口。

②缺点：并非所有系统都能够按照层次来进行划分；很难找到一种合适和正确的层次划分方法；在传输数据是，需要经过多个层次；多层结构难以调试。

分层模式一般用在一般桌面应用程序和电子商务Web应用程序。

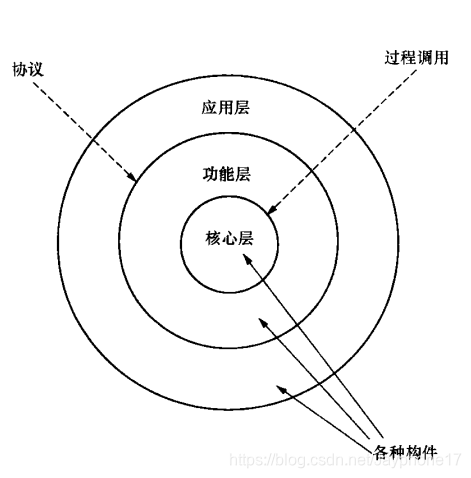
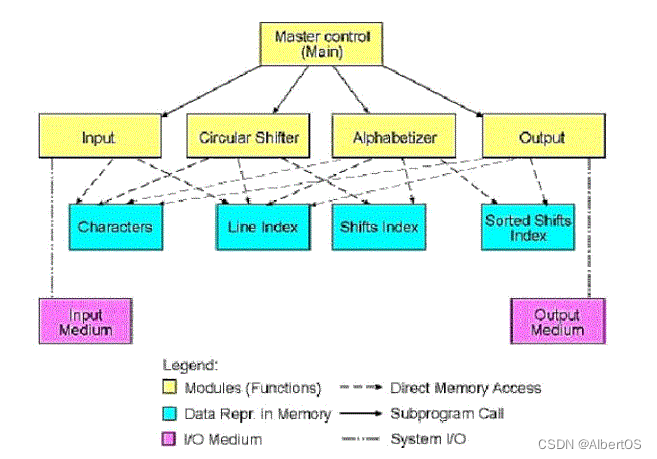


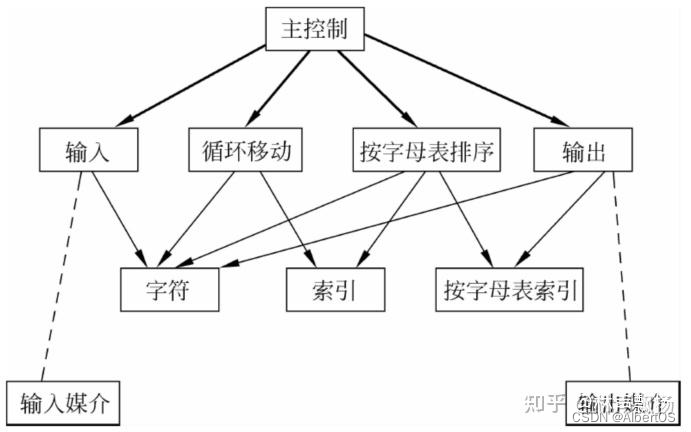
图 5 分层体系结构风格原理图

**（5）KWIC索引系统**

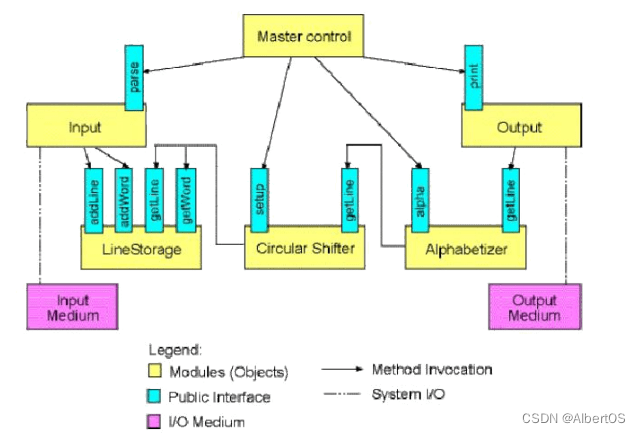
KWIC索引系统是软件体系结构研究领域的一个经典案例，它为我们提供了一个理解软件架构设计原则和实践的框架。在KWIC索引系统中可以采用使用共享数据的主程序/子程序、抽象数据类型/面向对象风格、基于事件通知的方案/响应式、基于管道的方案等解决方案。

**①使用共享数据的主程序/子程序**：将其分解为输入、移位、排列和输出四个组件，主程序依次对它们进行调用，数据通过共享存储在组件间共享，组件与数据之间读写不受约束，由主程序协调保证读写顺序。





**②抽象数据类型/面向对象风格**：与方案①相比，该方案将其分为五个模块，数据不再共享，而是相反做了接口隔离，将各个功能模块分开设计，并且利用权限控制保证了封装的特性，系统的扩展性和封装性增强，是个基本的面向对象方案。



**③基于事件通知的方案/响应式**：类似方案①，但数据更抽象，而不是暴露数据格式，数据被修改时，隐式调用计算。基于事件通知的系统中的各个功能并不是直接被调用的，而是通过组建通知或广播事件信息而触发的。其他组件可以通过注册一个与某个事件通知相关联的过程而与其发生联系。

**④基于管道的方案**：在基于管道的方案中，每个组件都会有一组输入流和一组输出流，一个组件读入，然后处理后送出到下一个组件上。这个组件一般称为filter。连接器则称为pipes。filter之间不能有数据共享，并且彼此相互独立。

管道由一系列过滤器组成，通过数据流连接；分布式，每个过滤器只要有要计算的数据就可以运行，过滤器之间的数据共享严格限于在管道上传输的数据。

参考学习资料：

（1）[13种常见软件体系结构风格定义分析、结构图、优缺点\_软件体系结构设计风格-CSDN博客](https://blog.csdn.net/Jayphone17/article/details/103651076)

（2）[10个常见软件体系结构模式-CSDN博客](https://blog.csdn.net/omnispace/article/details/80224942)

（3）[软件体系结构-KWIC索引系统\_kwic系统的四位一体图-CSDN博客](https://blog.csdn.net/Albert_weiku/article/details/131765776)

5. 完成软件需求规格说明SRS

**下周五（含）前将软件需求规格说明提交给相应的助教**

项目跟踪，建立能反映项目及小组每个人工作的进度、里程碑、工作量的跟踪图或表，将其保存到每个小组选定的协作开发平台上，每周更新。