#### 经典软件体系结构案例KWIC研究及评分

##### 概述：

软件体系结构表示系统的框架结构，用于从较高的层次上来描述各部分之间的关系和接口，主要包括构件、构件性质和构件之间的关系。

通过使用软件体系结构，可以有效地分析用户需求，方便系统的修改，以及减小程序构造风险。

KWIC系统接受一系列有序的行，每行由有序的单词组成，每个单词由有序的字符组成。系统的目标是输出所有行的所有可能的循环位移，并按字母顺序排列。

##### 常见风格：

常见风格分类：

1. 数据流风格：批处理和管道/过滤器。
2. 调用/返回风格：主程序/子程序、层次结构和C/S。
3. 面向对象风格。
4. 独立部件风格：进程通信和事件驱动。
5. 虚拟机风格：解释器和基于规则的系统。
6. 数据共享风格：数据库系统和黑板系统。

阅读所给文档和网站后，发现主要讲述了四种KWIC系统的架构设计：

1.**程序/子程序与共享数据**(Shared Data Solution)

根据四个基本功能（输入、位移、排序和输出）进行分解，并通过共享存储进行数据通信。

优点:

数据表示高效，因为计算可以共享相同的存储。

各个计算方面被隔离在不同的模块中，具有一定的直观吸引力。

缺点:

对于数据存储格式的变化，几乎所有模块都会受到影响。

难以适应整体处理算法和系统功能增强的变化。

重用性不是特别支持。

2.**抽象数据类型**

将系统分解为类似的五个模块集，而是通过接口提供的程序来访问数据，而非直接由计算组件共享。

优点：

允许在不影响其他模块的情况下，改变单个模块中的算法和数据表示。

比第一个解决方案更好地支持重用，因为模块对它们交互的其他模块的假设更少。

缺点：

对于系统功能的增强不是特别适用，因为要向系统中添加新功能，实施者必须修改现有模块，这可能会损害它们的简单性和完整性，或者添加新模块，这可能导致性能损失。

3.**隐式调用**

使用基于共享数据的组件集成形式，但是有两个重要的区别：数据接口更抽象，计算被隐式调用。

优点：

容易支持系统的功能增强：通过将它们注册为在数据变化事件上调用，可以向系统附加额外的模块。

数据以抽象方式访问，因此可以保护计算不受数据表示变化的影响。

支持重用，因为隐式调用的模块仅依赖于某些外部触发事件的存在。

缺点：

控制隐式调用模块的处理顺序可能很困难。

由于调用是数据驱动的，这种分解的最自然实现倾向于比前面考虑的分解使用更多的空间。

**4.管道和过滤器**

使用管道解决方案，有四个过滤器：输入、位移、排序和输出。每个过滤器处理数据并将其发送到下一个过滤器。

优点：

保持了处理流程的直观性。

支持重用，因为每个过滤器可以独立工作（前提是上游过滤器以它期望的格式产生数据）。

通过在处理序列的适当位置插入过滤器，可以轻松地向系统添加新功能。

支持修改的简便性，因为过滤器在逻辑上是独立于其他过滤器的。

缺点:

几乎不可能修改设计以支持交互式系统。例如，为了删除一行，将需要一些持久的共享存储，这违反了这种方法的基本信条。

在空间使用上效率不高，因为每个过滤器必须将其所有数据复制到其输出端口。

##### KWIC的模块化

**Modularization (模块化)**

描述：系统被划分为具有明确接口的模块，每个模块小而简单，可以被彻底理解和良好编程。

**Example System 1**

**特点**：它是一个小型系统；所有模块间的接口必须在工作开始前明确指定。

模块化的**目的**：

提高系统的灵活性和可理解性；缩短开发时间，因为不同的组可以独立工作在每个模块上；提高产品灵活性，能够在不影响其他模块的情况下，对单个模块进行重大更改。

模块化的**设计决策**：

模块化包括在独立模块工作开始前必须做出的设计决策。

不同的模块化选择包括不同的决策，但所有情况都旨在描述所有“系统级”决策。

**Example System 1**是一个典型的模块化编程案例，每个模块都有特定的职责，模块间的接口被明确定义。这样设计方便系统的维护和扩展。

模块划分如下：

**Module 1: Input (输入模块)**

功能：从输入媒介读取数据行，并将它们存储在核心存储器，用于其他模块处理。

数据处理：对于字符，将每四个字符打包成一个词，使用未使用的字符表示单词的结束。

索引：维护一个索引来显示每行的起始地址。

**Module 2: Circular Shift (循环移动模块)**

功能：在输入模块完成工作后被调用，用来准备每个循环移动的索引。

输出：输出包含原始行索引和循环移动起始地址，将二者进行配对。

**Module 3: Alphabetizing (排序模块)**

功能：对模块1和2产生的数组进行接收，生成按字母顺序排列的循环移动数组。

特点：循环移动的列表按字母顺序排列。

**Module 4: Output (输出模块)**

功能：使用模块3和模块1产生的数组，生成包含所有循环移动的格式化输出列表。

**Module 5: Master Control (主控制模块)**

功能：控制其他四个模块的序列，可能还处理错误消息和空间分配等。

##### 软件结构中的模型问题

**模型问题的概念：**

页面提出，软件架构社区通过共享一组标准的示例问题，可以提高我们工作想法、展示技术以及比较结果的能力。

**当前模型问题：**

页面列出了一系列的模型问题，每个问题都有简短的描述，并且一些已经提供了解决方案或扩展问题规范。这些问题包括：

Keyword in Context (KWIC): 创建给定行的旋转的字母排序列表。

Sea Buoy: 收集并传输天气数据，可以自动进行，也可以按需进行；允许紧急服务的抢占。

Cruise Control: 维持车辆的速度。

Conference Refereeing: 为会议征集、审查和选择论文。

Mailing List Handler: 从多个来源合并地址信息，消除重复并考虑读者偏好。

Printer Spooler: 在打印机网络内管理打印作业。

Library: 自动化传统图书馆任务，如图书的借阅和归还。

Automated Teller Machine (ATM): 提供通常的银行功能，通过远程机器实现。

Calendar Scheduler: 组织会议安排。

Compiler: 将编程语言的源代码翻译成可执行形式。

Mobile Robot: 设计一个能够执行任务同时监测环境的移动机器人，例如避免障碍物。

##### 综合比较

**共享数据：**在支持算法变化、数据表示变化和重用性方面较弱，但由于直接共享数据，在性能方面表现较好。

**抽象数据类型**：允许数据表示的变化并支持重用，但不会显著影响性能。但是，由于组件之间的交互被嵌入到模块本身中，要实现一些功能，比如改变整体处理算法、添加新功能，可能需要大量更改现有系统。

**隐式调用**：特别适合添加新功能，但在数据表示和重用性方面存在一些问题，可能会引入额外的执行开销。

**管道和过滤器**：因为该架构允许在文本处理的流程中放置新的过滤器，所以能够支持处理算法变化、功能变化和重用。但是，数据表示会被嵌入到沿管道传输的数据类型中，观察交换格式，这个过程可能涉及解析和反解析数据的一些额外开销。

针对**网上书店项目**，我们小组选择解决方案4:

**管道和过滤器 (Pipes and Filters Solution)**。

原因如下：

**模块化**：管道和过滤器架构天然支持模块化，便于实现网上书店的各个功能组件，如用户注册、登录、图书浏览、购物车管理等，每个组件可以进行独立开发和维护。

**可扩展性**：随着网上书店的一系列业务的发展，新的功能可以作为新的过滤器轻松集成到现有系统中，而不需要我们对现有架构进行大规模修改。

**重用性**：过滤器可以被设计为可重用的组件，例如，用户管理模块可以用于注册、登录，也可以用于店主的客户管理。

**独立性**：每个过滤器可以独立运行，也就是说系统的某些部分可以直接进行更新和维护，且不影响其他部分。

**性能**：通过合理设计过滤器之间的数据流，可以优化系统性能，例如，通过缓存热门图书数据来减少数据库的访问压力。

**容错性**：考虑到过滤器的独立性，一个过滤器的故障不会直接影响到整个系统，提高了系统的稳定性。

**适应性**：管道和过滤器架构适合处理流式数据，对于网上书店这个项目来说，用户的浏览和购买行为可以被视为数据流，能够实时更新和响应。

综上，解决方案4能够提供所需的灵活性、可扩展性和模块化，非常适合我们网上书店这种需要处理大量用户请求和数据的系统。

**打分**：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name/grade | 共享数据 | 管道和过滤器 | 数据抽象 | 隐含调用 |
| 刘菲 | 1 | 4 | 1 | 5 |
| 王雨晨 | 2 | 5 | 2 | 3 |
| 杨玥莹 | 3 | 3 | 1 | 4 |
| 蒋安琪 | 1 | 2 | 3 | 3 |
| 任雨晴 | 2 | 5 | 4 | 2 |
| 总计 | 9 | 19 | 11 | 17 |

综上所述最后我们小组选择了**管道和过滤器方式**。