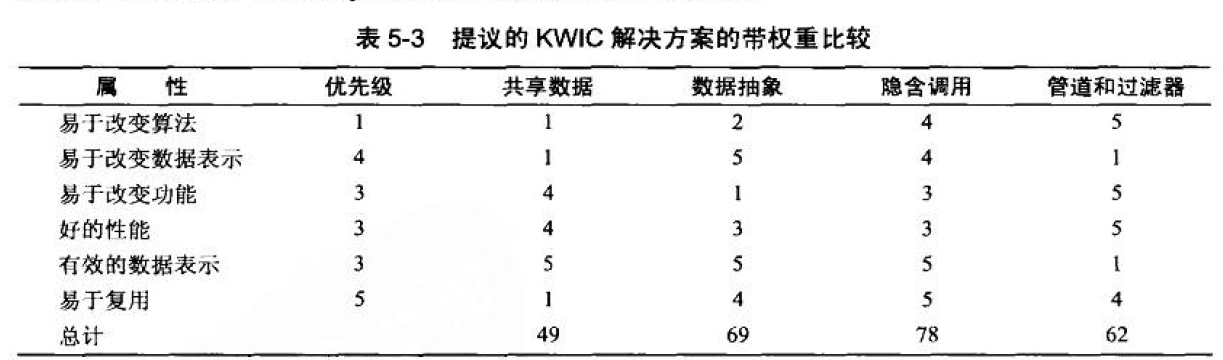
# 评价标准

我们可以参考教材第五章表5-3所示，对KWIC的解决方案和我们小组项目的设计方案进行打分。



# KWIC的解决方案

解决方案众多这里以子程序，面向对象和管道技术来讨论

1、采用主/子程序体系结构风格实现KWIC关键词索引系统

主程序/子程序风格（Main Program/Subroutine Style）将系统组织成层次结构，包括一个主程序和一系列子程序。主程序是系统的控制器，负责调度各子程序的执行。各子程序又是一个局部的控制器，调度其子程序的执行。设计词汇表:

主程序main()，子程序 shift(), sort()方法,方法的调用,返回构件和连接件类型:构件:各类子程序，如 shift(),sort()

连接件:方法的调用基本的计算模型:

子程序有 shift(), sort()方法，shift()对单词进行移位，sort()方法对单词进行排序。

风格的基本不变性:，主程序有顺序地调用子程序，单线程控制。

子程序风格的重要设计决策与约束有:主程序使用（程序调用）关系建立连接件，以层次分解的方式共同组成层次结构。

每一个上层部件可以“使用”下层部件，但下层部件不能“使用”上层部件，即不允许逆方向调用。

系统应该是单线程执行。主程序部件拥有最初的执行控制权，并在“使用”中将控制权转移给下层子程序。

子程序只能够通过上层转移来获得控制权，可以在执行中将控制权转交给下层的子子程序，并在自身执行完成之后必须将控制权还交给上层部件。

主程序/子程序风格的主要实现机制是模块实现，它将每个子程序都实现为一个模块，主程序实现为整个系统的起始模块。依照抽象规格的层次关系，实现模块也被组织为相应的层次机构，通过导入/导出关系相连接。

需要强调的是，虽然主程序/子程序风格非常类似于结构化程序的结构，但是主程序/子程序风格是基于部件与连接件建立的高层结构。它的部件不同于程序，而是更加粗粒度的模块。而且，在部件的实现模块内部，可以使用结构化方法，也可以使用面向对象方法，这并不妨碍整个系统的高层结构符合主程序/子程序风格的约定。

主程序/子程序风格的优点有:流程清晰，易于理解和强控制性。

严格的层次分解和严格的控制权转移使得主程序/子程序风格对程序的实际执行过程具备很强的控制能力，这带来了一个特点:如果一个子程序所连接的子子程序是正确的，那么就很容易保证该子程序的“正确性”。所以，主程序/子程序风格比其他常见风格更能控制程序的“正确性”

2、采用面向对象体系结构风格实现KWIC关键词索引系统

面向对象式风格,需要强调说明的是它的“对象”是部件,属于高层结构的元素，虽然名称相同，但它并不是面向对象方法中所述的“对象”实体。“面向对象式”风格的命名是因为它借鉴了面向对象方法的思想，而不是因为它使用面向对象方法实现体系结构，这也是在该风格名称中有一个“式”字的原因。

面向对象式风格的主要实现机制是模块实现,它将每个对象部件实例都实现为一个模块。存在连接的对象部件实例之间会存在模块的导入/导出关系。

每个模块内部可以是基于面向对象方法的实现，也可以是基于结构化方法的实现。设计词汇表:

(1) input, output, shift, sort, main类

(2) shift, sort, output 对象

(3)对象间的消息传递构件和连接件类型:

构件类型:对象

连接件的类型:为对象间的消息传递

基本的计算模型:

模块从文本文件 input.txt中一行一行读取单Input

Sort Shift模块用于将单词移位。模块将单词进行排序。output.txt Output模块将最终结果写到文本文件中

面向对象式风格的优点有:

内部实现的可修改性。因为面向对象式风格要求封装内部数据，隐藏内部实现，所以它可以在不影响外界的情况下，变更其内部实现。

易开发、易理解、易复用的结构组织。面向对象式风格将系统组织为一系列平等、自治的单位，每个单位负责自身的“正确性”，不同单位之间仅仅是通过方法调用相连接，这非常契合模块化思想，能够建立一个易开发、易理解、易复用的实现结构。

3、采用管道和过滤器体系结构风格实现KWIC关键词索引系统

管道-过滤器风格将系统的功能逻辑建立为部件集合。每个部件实例完成一个对数据流的独立功能处理,它接收数据流输入,进行转换和增量后进行数据流输出。连接件是管道机制,它将前一个过滤器的数据流输出传递给后一个过滤器作为数据流输入。连接件也可能会进行数据流的功能处理，进行转换或增量，但连接件进行功能处理的目的为了适配前一个过滤器的输出和后一个过滤器的输入，而不是为了直接承载软件系统的需求。

各个过滤器可以并发执行。每个过滤器都可以在数据输入不完备的情况下就开始进行处理，每次接到一部分数据流输入就处理和产生一部分输出。这样，整个的过滤器网络就形成了一条流水线。

过滤器读管道输出流，过滤器写管道输入流，过滤器通过对输入流的增量计算来完成。

风格的基本不变性:

过滤器是独立的实体，它们之间通过管道传递数据

过滤器风格可以为所有过滤器部件实例建立模块实现，每个过滤器部件-管道现为一个单独的模块，可以为所有简单的管道连接件建立一个通用的模实例被实块实现，为每个复杂的管道连接件建立一个单独的模块实现，相关联过滤器部件的实现模块要导入管道实现模块。

过滤器风格需要为所有过滤器-因为管道-过滤器风格需要并发执行，所以管道部件实例和复杂连接件实例建立进程实现，每个过滤器部件实例和复杂连接件实例被实现为一个进程。

如果操作系统能够提供管道机制，那么进程实现时的简单管道可以利用操作系统提供的机制来实现。此时，所有的实现进程都需要部署在同一个物理节点。如果有中间件平台能够提供管道机制,那么进程实现时的简单管道可以利用中间件平台提供的机制来实现。此时，各个实现进程可以被部署在不同的网络节点。

## 评分

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 属性 | 优先级 | 子程序调用 | 面向对象 | 管道过滤器 |
| 易于改变算法 | 2 | 2 | 2 | 2 |
| 易于改变数据表示 | 4 | 2 | 2 | 4 |
| 性能 | 5 | 3 | 3 | 3 |
| 是否易于复用 | 3 | 3 | 4 | 3 |
| 结构是否灵活 | 2 | 3 | 3 | 1 |
| 总计 |  | 42 | 45 | 46 |

# BBS八哥论坛的设计

## 解决方案

我们采用分层模式、浏览器-服务器端模式、模型-视图-控制器（MVC）  这三个混合的体系架构

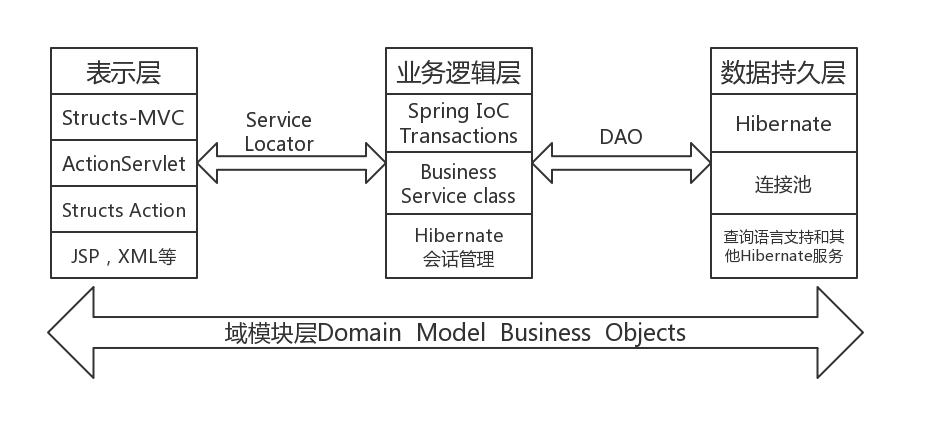
### 分层模式

本系统采用SSH框架开发，系统架构主要分为四层：表示层、业务逻辑层、数据持久层和域模块层(实体层)。

在表示层中，首先通过JSP页面实现交互界面，负责传送请求(Request)和接收响应(Response)，然后Struts根据配置文件(struts-config.xml)将ActionServlet接收到的Request委派给相应的Action处理。

在业务逻辑层中，管理服务组件的Spring IoC容器负责向Action提供业务模型(Model)组件和该组件的协作对象数据处理(DAO)组件完成业务逻辑，并提供事务处理、缓冲池等容器组件以提升系统性能和保证数据的完整性。

在持久层中，则依赖于Hibernate的对象化映射和数据库交互，处理DAO组件请求的数据，并返回处理结果。



采用上述开发模型，不仅实现了视图、控制器与模型的彻底分离，而且还实现了业务逻辑层与持久层的分离。这样无论前端如何变化，模型层只需很少的改动，并且数据库的变化也不会对前端有所影响，大大提高了系统的可复用性。

### 浏览器-服务端模式

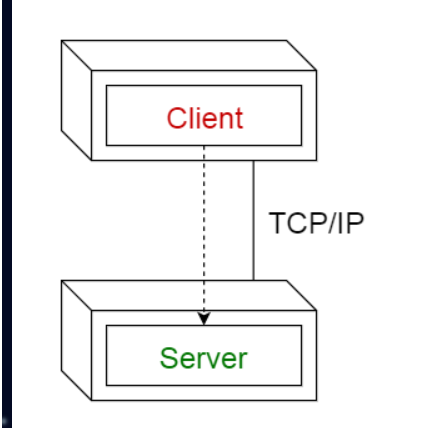
我们采用浏览器-服务端模式具有以下优点：

●分布性强，客户端零维护。只要有网络、浏览器，可以随时随地进行查询、浏览等业务处理。

●业务扩展简单方便，通过增加网页即可增加服务器功能。

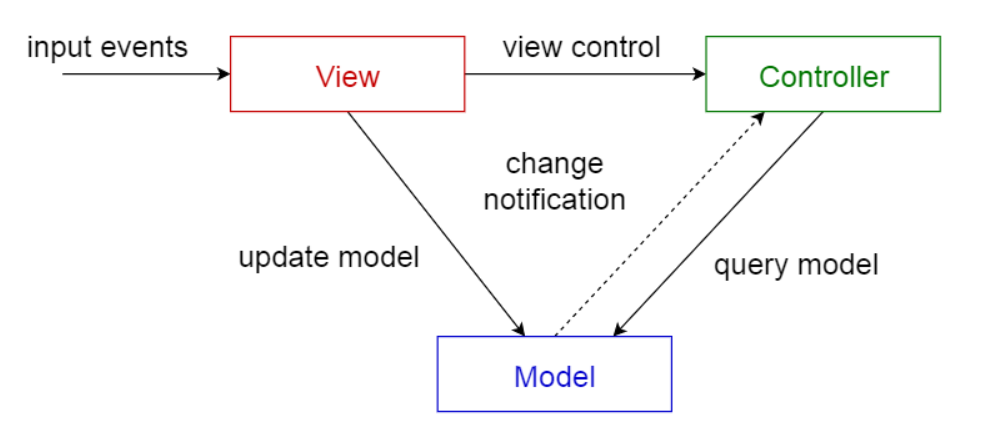
●维护简单方便，只需要改变网页，即可实现所有用户的同步更新。

●开发简单，共享性强。



### 模型-视图-控制器模式

该模式也叫 MVC 模式，划分交互程序为3个部分：模型——包含核心功能和数据，视图——显示信息给用户（多个视图可被定义），控制器——处理用户输入。它通过分割用户信息的内部陈述和呈现、接受方式来实现，解耦组件并允许高效的代码复用。



视图和数据模型的分离:使用不同的视图对相同的数据进行展示﹔分离可视和不可视的组件，能够对模型进行独立测试。因为分离了可视组件减少了外部依赖利于测试。（数据库也是一种外部组件)

视图和表现逻辑(Contro11er)的分离:Contro11er是一个表现逻辑的组件，并非一个业务逻辑组件。MVc可以作为表现模式也可以作为建构模式，意味这Contro11er也可以是业务逻辑。分离逻辑和具体展示，能够对逻辑进行独立测试。

### 评价

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 属性 | 优先级 | 分层模式 | B/S模式 | 模型-视图-控制器模式 |
| 易于改变算法 | 2 | 3 | 3 | 4 |
| 易于改变数据表示 | 4 | 3 | 4 | 5 |
| 性能 | 5 | 4 | 3 | 3 |
| 是否易于复用 | 3 | 3 | 4 | 4 |
| 结构是否灵活 | 2 | 2 | 3 | 3 |
| 总计 |  | 51 | 55 | 61 |