

# 软件体系结构

## 第一次作业

**KWIC**

李今晖-1501210936

软件工程-软件开发

[jinhui.li@pku.edu.cn](mailto:jinhui.li@pku.edu.cn)

## 一、软件设计说明书

KWIC 是用于关键字检索的程序，将输入的字符串分割为单词数组，然后以单词为单位循环移位，生成所有可能的关键字。

其使用需要在命令行下调用，形式如下

下载源代码：git clone <https://github.com/JinhuiLee/arch.git>

或解压代码压缩包 arch.zip

切换至根目录后，可以编译并运行

编译：javac kwic/oo/KWIC.java

运行：java kwic/oo/KWIC.java input <noise>

其中 input 为输入文件，noise 为 Pipe and filter 下的噪声文件  
input 典型输入如下图所示

```
~/working/hw — -bash
[lijinhuideMacBook-Pro:hw jinhui] $ cat input
star Wars
The Empire Strikes Back
The Return of the Jedi
```

ms 架构下，对于该 input 的输出如下图所示

```
~/working/hw — -bash
[lijinhuideMacBook-Pro:hw jinhui] $ java kwic/ms/KWIC input
Back The Empire Strikes
Empire Strikes Back The
Return of the Jedi The
Strikes Back The Empire
The Empire Strikes Back
The Return of the Jedi
Wars star
of the Jedi The Return
star Wars
the Jedi The Return of
```

可以看到，对于每一行数据，KWIC 遍历了所有的循环移位结果，并且将所有的结果排字母序输出。

## 二、源程序及修改

源程序已托管至 Github,

地址: <https://github.com/JinhuiLee/arch>

修改之处通过 github 的 history 功能可以一目了然

以“+”开头的行号表示修改后的代码

405	// copy the indices row	412	// copy the indices row
406	tmp = new int[shift_count];	413	tmp = new int[shift_count];
407	System.arraycopy(circular_shifts[1], 0, tmp, 0, shift_count);	414	System.arraycopy(circular_shifts[1], 0, tmp, 0, shift_count);
408	circular_shifts[1] = tmp;	415	circular_shifts[1] = tmp;
409	- }	416	+ }
410	- }	417	+ }
		418	+ shift_chars = new char[1024*10];
		419	+ shift_index = new int[1024*10];
		420	+ int pos=0;
		421	+ int index_pos=0;
		422	+ for(int i = 0; i < circular_shifts[0].length; i++){
		423	+ int line_number = circular_shifts[0][i];
		424	+ int shift_start = circular_shifts[1][i];

### 1.Main/sub 的修改处

<https://github.com/JinhuiLee/arch/commit/4e8af18d3bb9e5a65270bea2590b8a8f4f954292>

### 2.OO 的修改处

<https://github.com/JinhuiLee/arch/commit/8faf0d47003512beca7d6d004ae0adf a156da52b>

### 3.Event-based 的修改处

<https://github.com/JinhuiLee/arch/commit/92ad19d65be25dfba47469c778a709 0c1c8f0429>

### 4.Pipe and filter 架构修改处

<https://github.com/JinhuiLee/arch/commit/c453401a8a5386309dc74d2e21967a b7c0091654>

每个模块做了相应的修改以达成材料中要求的功能, 详述于下:

#### 1. Main/sub 架构

- 修改了内部数据结构, 循环移位的结果保存于一维数组
- 过滤首字符为数字的情况

#### 2. Object Oriented 架构

- 增加了 Line 类, LineStorage 类的内部表示用 Line 替代 ArrayList
- 增加了交互式界面, 具有 Add,Delete,Print,Quit 功能

#### 3. Event-Based 架构

- 交互式界面, 具有 Add,Delete,Print,Quit 功能
- 增加了 Index 索引功能, 快速输出各个单词的频次

#### 4. Pipe and Filter 架构

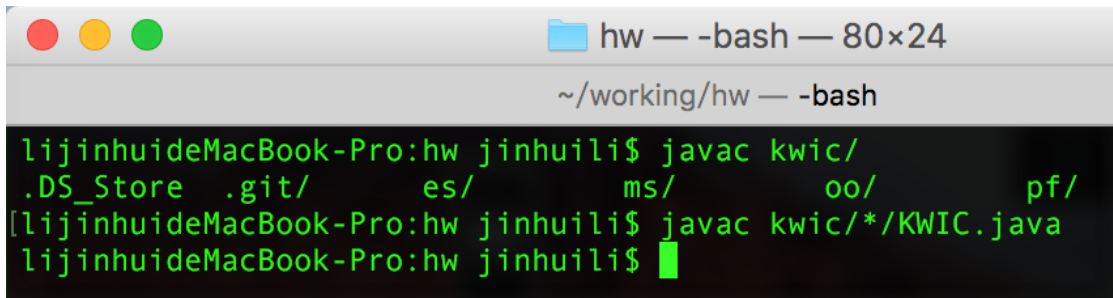
- 过滤给定噪声的功能, 以噪声为首单词的字符串会被过滤
- 首单词转换为全大写

### 三、程序使用说明

编译：

1.cd 改变目录到代码根目录

2.输入 `javac kwic/*/KWIC.java`，若无错误提示信息即完成编译

A screenshot of a macOS terminal window. The title bar shows a folder icon, the text 'hw — -bash — 80x24', and standard macOS window controls. The address bar shows '~/.working/hw — -bash'. The terminal content shows a user at 'lijinhuideMacBook-Pro:hw jinhuali\$' typing 'javac kwic/' followed by a directory listing: '.DS\_Store .git/ es/ ms/ oo/ pf/'. The next line shows the user typing 'javac kwic/\*/KWIC.java' and pressing enter, resulting in a new prompt 'lijinhuideMacBook-Pro:hw jinhuali\$'.

运行：

1.cd 改变目录到代码根目录

2.对于 es ms oo, 使用 `javac kwic/*/KWIC.java input` 运行

对于 pf, 使用 `java kwic/pf/KWIC.java input noise` 运行

其中，input 参数为输入文件，noise 为 pf 中的噪声

## 四、运行结果

### 1.Main/sub 架构输出结果

```
~/working/hw — -bash
[lijinhuideMacBook-Pro:hw jinhuili$ java kwic/ms/KWIC input
Back The Empire Strikes
Empire Strikes Back The
Return of the Jedi The
Strikes Back The Empire
The Empire Strikes Back
The Return of the Jedi
Wars star
of the Jedi The Return
star Wars
the Jedi The Return of
```

结果说明：标准的 KWIC 结果，原任务主要要求修改程序内部的数据结构，及过滤数字开头的结果。

### 2.Object Oriented 架构输出结果

```
~/working/hw — java kwic/oo/KWIC input
[lijinhuideMacBook-Pro:hw jinhuili$ java kwic/oo/KWIC input
Add, Print, Quit:a
>hello KWIC
Add, Print, Quit:p
KWIC hello
hello KWIC
Add, Print, Quit:
```

结果说明：应原任务要求,构建交互式的 KWIC 程序，可以选择 Add(增加条目), Print(打印条目), Quit(退出程序)，效果如上图所示。

### 3.Event-Based 架构输出结果

```
~/working/hw — java kwic/es/KWIC input
[lijinhuideMacBook-Pro:hw jinhuili$ java kwic/es/KWIC input
Add, Print, Index, Delete, Quit:a
>hello KWIC
Add, Print, Index, Delete, Quit:p
KWIC hello
hello KWIC
Add, Print, Index, Delete, Quit:i
KWIC: 1
hello: 1
Add, Print, Index, Delete, Quit:d
>hello KWIC
Add, Print, Index, Delete, Quit:p
Add, Print, Index, Delete, Quit:
```

结果说明：Event-Based 程序要求较为复杂，可以选择 Add(增加条目), Print(打印条目), Index(索引), Delete(删除), Quit(退出程序)，上图演示了 ADD、Print、Index、Delete 功能，均能良好工作。

#### 4. Pipe and Filter 架构输出结果

```
~/working/hw — -bash
[lijinhuideMacBook-Pro:hw jinhui]$ cat noise
star
[lijinhuideMacBook-Pro:hw jinhui]$ java kwic/pf/KWIC input noise
BACK The Empire Strikes
EMPIRE Strikes Back The
JEDI The Return of the
OF the Jedi The Return
RETURN of the Jedi The
STRIKES Back The Empire
THE Empire Strikes Back
THE Jedi The Return of
THE Return of the Jedi
WARS star
[lijinhuideMacBook-Pro:hw jinhui]$
```

结果说明：noise 表示当前的噪声是“star”，因此 star wars 这条记录被 filter 过滤，同时添加了保持首单词大写的功能。

## 五、方案分析

个人对本次作业涉及的软件体系认识如下：

1. M/S 架构较为经典，从最早的 C、Fortran 语言流传至今，特点是效率较高，本身编程模型较为简洁，但不能从容应对数据结构和功能的变化，一旦发生变化，代码需要大量修改以适应。
2. OO 架构将数据与其对应的操作绑定在一起，完成数据的封装，对外仅暴露接口，在具体的实践中发展出了很多设计模式，这些设计模式可以帮助工程师从容地应对变化。
3. Event-based 架构通过事件产生函数调用链，在相应的事件发生时，自动执行对应的回调函数，也很好地封装了变化，缺点是函数调用关系不明显。
4. Pipe-filter 架构传承于 Unix 哲学中的管道，模块间相对独立，每个模块各司其职，不同模块经管道连接，可形成独有的功能，如 `ls | grep <>` 等。管道也较为适应功能变化。在本例中，各个管道还通过 Thread 实现并发，但缺点就是数据需要经过多次拷贝，管道数量较大时代价高昂。

## 六、问题解答

### 一、M/S 架构

1. Which modules from the original KWIC system did you need to modify to implement the new data representation of circular shifts? What conclusion can you draw here? How does the KWIC system implemented by means of main/subroutine architecture with shared data withstand design changes in data representation?

答：数据表示的修改中，需要修改 `circular_shifts`, `alphabetized`, `output` 三个模块，每个模块都需要大量修改模块的变化往往导致“牵一发而动全身”的结果，修改 `circular_shifts` 的数据表示后，`alphabetized` 也需要修改，而一旦修改 `alphabetized` 之后，`output` 模块也需要相应的改动来适应变化。

结论：m/s 架构似乎不太能适应数据表示级别的变化，一旦数据表示变化，多处模块都需要修改。

2. Which modules from the original KWIC system did you need to modify to add filtering function to the system? Was it difficult to add a new processing component (filter function) accessing the shared data? Can we conclude here that the KWIC system with main/subroutine architecture with shared data is robust to design changes in system's function (assuming that new functions access the shared data)?

答：添加过滤器的修改中，可以通过添加 `filter` 函数来实现相应功能，找到合适的地方插入即可，总体不难实现，但无疑此举会让各模块耦合度更高。

结论：m/s 架构新增过滤器函数较为容易，但可能增加模块间的耦合。

3. Would it be difficult to reuse modules from the existing KWIC system in other systems? For example, suppose that in another software system we need to sort a number of lines. Suppose also that in this system lines are represented differently than in the existing KWIC system. Could we reuse the existing alphabetizing function as it is, or do we need to make some modification to the existing code to be able to reuse it?

答：将本 KWIC 系统中的函数移植到其他系统时，若数据不一致，要做多处修改。有两种办法，其一：修改复用的函数本身来适应新系统的数据表示。其二：为函数增加一层 `wrapper`，将新系统的数据格式转换为现有函数兼容的格式。两种办法都要做大量修改。

### 二、OO 架构

1. Which modules from the original KWIC system did you need to modify to implement the new data representation of lines within the `LineStorage` class? What conclusion can you draw here? How does the KWIC system implemented by means of object-oriented architecture withstand design changes in data representation within a particular module? Is this true for any other object-oriented system?

答：OO 架构中，数据表示只需要修改 `LineStorage` 类本身，外加增加一个名为 `Line` 的类即可完成。OO 架构中，类与类间仅暴露接口，数据有了良好的封装，所以 OO 对于数据表

示变化的容忍度较高，对于其他的 OO 系统，这个结论一般也成立。

2. Which modules from the original KWIC system did you need to modify to implement a new processing algorithm in the system? Can we conclude here that the KWIC system with object-oriented architecture is robust to desing changes in processing algorithm of the system?

答：OO 架构中，改为交互式 KWIC 只需要修改 Input 和 KWIC 两个模块，且只做少量修改即可完成。因此，我们可以得出结论说 OO 式的 KWIC 对处理算法的变化具有鲁棒性。

3. Would it be difficult to reuse modules from the existing KWIC system in other systems? For example, suppose that in another software system we need to sort a number of lines. Suppose that the sorting algorithm expects lines to be represented as arrays of characters. Can you describe a possible object-oriented solution to reuse the LineStorage class without need to modify the source code of the class.

答：OO 架构中，复用相对容易，因为数据与其操作都绑定在一起，只需要配套接口即可完成代码复用或移植

### 三、Event-based 架构

1. Which modules from the original KWIC system did you need to modify to implement the WordsIndex module? What can you conclude here? Do systems implemented with event based architecture allow for easy adding of new modules, assuming that the new modules apply the already existing event protocol?

答：EB 架构中，为实现 Index 功能，添加一个 WordIndex 类，并对 Circular Shifter 作少量修改添加 Delete 事件的响应，即可完成。基于事件的架构增添新的模块亦较为容易。

2. Please, describe the easiest way to implement a WordsIndex that should keep all words that appear in the original lines and the number of their occurrences in the circular shifts.

答：使用 Java 中的 Hashmap，在切分字符串得到一个个单词后，以单词本身作为 Hash 的 Key，Add 事件时累加 1，Delete 事件时递减 1，即可完成。

3. One of the main properties of event based systems is so-called "implicit" invocation of procedures. That means some of the procedures in the system are not invoked directly, but rather an event is sent into the system, and then the system itself after processing the events invokes these procedures. Now, taking into account that the processing of events and deciding which procedures to call might be quite complex in systems with a large number of modules and a complex event protocol what can you conclude about the performance of event based system? Do you think that these systems may achieve the same level of the performance as systems with direct procedure calls?

答：关于事件驱动的性能，现在实践证明，在应对 IO 密集型应用时，事件驱动的性能比传统多线程系统性能高很多，如当前最流行的 Web 平台 Node.js，它采用了主循环的事件驱动机制，应对高并发时表现十分惊艳，没有出现题目所提及的“大量模块复杂事件协议”引起的效率问题。

### 四、Pipe-filter 架构



1. In both modifications of the current system you needed to extend the functionality of the system. Which modules from the original KWIC system did you need to modify to implement the new functionality in both cases? Does this mean that the pipe and filter KWIC system is robust to design changes in the functionality of the system? Is the same true for any pipe and filter system?

答：管道-过滤器架构下，我修改功能时并不需要修改原来的模块，只需要各增加一个模块并把它们用管道连接起来即可，管道-过滤器架构能从容应对功能变化，对于其他管道-过滤器系统应当也是相同的。

2. Filters in pipe and filter systems are completely independent entities. Thus, filters:

- a. Do not share state (data) with other filters.
- b. Do not know the identity of neighbor filters in the pipeline.
- c. Process their data concurrently and incrementally, i.e., they do not wait for other filters to start or finish their jobs.

Would it be possible to implement an interactive version of the KWIC system, which will allow users to delete certain lines from the data flow but still not violate any of the above defined filter properties?

答：管道-过滤器架构下，实现交互式功能可以通过增加一个模块，在这个模块中处理交互信息，如果需要后续模块的协助则写入相应的管道，后续模块会自动处理，这样不违背以上的 a,b,c 特性。

3. Normally, filters in pipe and filter systems work as follows. They read data from the input pipe, transform it and write it to the output pipe. Usually, this involves copying of data from the input to the output pipe. If the number of filters in a pipeline of a complex pipe and filter system is large then there is a lot of in-memory data copying. How this may affect the overall system performance?

答：拷贝属于 IO，是较消耗机器资源的操作，因而多重的拷贝会严重影响系统平均性能，随着管道数量的增大，系统性能会下降得愈发明显。因而其比较适合构建较小型的系统。