****

**题目：KWIC黑板风格的实现**

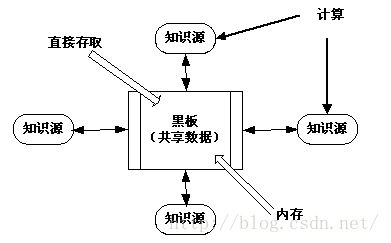
|  |  |
| --- | --- |
| 学院： | 计算机科学与技术学院 |
| 学号： | 18101212354 |
| 姓名： | 胡涛 |
|  |  |

实验报告

黑板风格的概念：黑板风格是仓库风格的一种，在仓库风格中有两种构件，一种是中央数据结构、另一种是独立构件的集合。对于系统中的数据和状态的控制方法有两种，一种是传统的方法由输入事务选择进行何种处理，并把执行结果作为当前状态存储到中央数据结构中，这时，仓库是一个传统的数 据库体系结构 ；另一种是由中央数据结构的当前状态决定进行何种处理。这就是仓库风格是一种特殊化黑板风格。

黑板系统传统上被用于在信号处理方面进行复杂解释的应用程序，以及松散耦合的构件访问共享数据的应用程序。它适用于需要解决冲突并处理可能存在的不确定性的系统。 黑板系统的得名，是因为它反映的是一种信息共享的系统如同教室里的黑板一样，有多个人读也有多个人写。

黑板风格的结构图：



KWIC系统简介：KWIC作为一个早年间在ACM的Paper提出的一个问题，被全世界各个大学的软件设计课程奉为课堂讲义或者作业的经典。KWIC索引系统接受一些行，每行有若干字，每个字由若干字符组成；每行都可以循环移位，亦即重复地把第一个字删除，然后接到行末； KWIC把所有行的各种移位情况按照字母表顺序输出。

黑板风格的实现：



在实现中，为了解决仓库和操作解耦定义了两个抽象类，一个是AbstractBlackboard，在其中定义操作的集合。另一个是AbstractAction抽象类定义了操作调用的统一接口。在系统中，以SetenceData类为中心数据结构。

1. **class** SetenceData
2. {
3. **public**:
4. std::list<std::string\*> originData;
5. std::map<**int**, std::list<std::string\*> > shiftData;
6. std::vector<**int**> sortData;
7. SetenceData()
8. {
9. originData.clear();
10. shiftData.clear();
11. sortData.clear();
12. }
13. ~SetenceData()
14. {
15. shiftData.clear();
16. sortData.clear();
17. auto listBegin = originData.begin();
18. auto listEnd = originData.end();
19. **for** (auto element = listBegin; element !=listEnd; element++)
20. **delete** \*element;
21. originData.clear();
22. }
23. };

代码1. SetenceData类

1. **class** AbstractAction
2. {
3. **protected**:
4. AbstractBlackboard \* owner;
5. **public**:
6. **void** **virtual** fire() = 0;
7. AbstractAction() {}
8. **virtual** ~AbstractAction() {}
9. **bool** setOwner(AbstractBlackboard \* abstractBlackboard)
10. {
11. **if** (abstractBlackboard == nullptr)
12. **return** **false**;
13. owner = abstractBlackboard;
14. **return** **true**;
15. }
16. };

代码2. AbstractAction抽象类

1. **class** AbstractBlackboard
2. {
3. **private**:
4. AbstractAction \* input;
5. AbstractAction \* output;
6. AbstractAction \* shift;
7. AbstractAction \* alphabetizer;
8. **public**:
9. **bool** setAbstractActionByType(**int** type, AbstractAction \* abstractAction)
10. {
11. **if** (abstractAction == nullptr)
12. **return** **false**;
13. **if** (type > AbstractActionType\_Alphabetizer or type < AbstractActionType\_Input)
14. **return** **false**;
15. **switch** (type)
16. {
17. **case** AbstractActionType\_Input :
18. {
19. **if** (input != nullptr)
20. **delete** input;
21. input = abstractAction;
22. input->setOwner(**this**);
23. }
24. **break**;
25. **case** AbstractActionType\_Output :
26. {
27. **if** (output != nullptr)
28. **delete** output;
29. output = abstractAction;
30. output->setOwner(**this**);
31. }
32. **break**;
33. **case** AbstractActionType\_Shift :
34. {
35. **if** (shift != nullptr)
36. **delete** shift;
37. shift = abstractAction;
38. shift->setOwner(**this**);
39. }
40. **break**;
41. **case** AbstractActionType\_Alphabetizer :
42. {
43. **if** (alphabetizer != nullptr)
44. **delete** alphabetizer;
45. alphabetizer = abstractAction;
46. alphabetizer->setOwner(**this**);
47. }
48. **break**;
49. **default**:
50. **break**;
51. }
52. **return** **true**;
53. }
54. AbstractAction \* getAbstractActionByType(**int** type)
55. {
56. AbstractAction \* abstractAction = nullptr;
57. **switch** (type)
58. {
59. **case** AbstractActionType\_Input :
60. abstractAction = input;
61. **break**;
62. **case** AbstractActionType\_Output :
63. abstractAction = output;
64. **break**;
65. **case** AbstractActionType\_Shift :
66. abstractAction = shift;
67. **break**;
68. **case** AbstractActionType\_Alphabetizer :
69. abstractAction = alphabetizer;
70. **break**;
71. **default**:
72. **break**;
73. }
74. **return** abstractAction;
75. }
76. **virtual** **void** \* getData() = 0;
77. **virtual** **void** call() = 0;
78. };

代码3. AbstractBlackboard抽象类

1. **class** BlackBoard : **public** AbstractBlackboard
2. {
3. **public**:
4. std::list<SetenceData\*> allData;
5. **int** state;
6. **public**:
7. BlackBoard() : AbstractBlackboard()
8. {
9. allData.clear();
10. state = BlackBoardState\_Null;
11. }
12. ~BlackBoard()
13. {
14. allData.clear();
15. }
16. **void** changeState()
17. {
18. **if** (state != BlackBoardState\_Output)
19. state++;
20. }
21. **void** \* getData()
22. {
23. **return** &allData;
24. }
25. **void** call()
26. {
27. **int** type = 0;
28. **switch**(state)
29. {
30. **case** BlackBoardState\_Null:
31. type = AbstractActionType\_Input;
32. **break**;
33. **case** BlackBoardState\_Input:
34. type = AbstractActionType\_Shift;
35. **break**;
36. **case** BlackBoardState\_Shift:
37. type = AbstractActionType\_Alphabetizer;
38. **break**;
39. **case** BlackBoardState\_Sort:
40. type = AbstractActionType\_Output;
41. **break**;
42. **default**:
43. **break**;
44. }
45. **if** (type >= 0 and type <= AbstractActionType\_Alphabetizer)
46. getAbstractActionByType(type)->fire();
47. }
48. };

代码4. BlackBoard类

运行平台：



图1. 运行平台

运行结果：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 文件名 | 10000-10-15.txt | 10000-15-20.txt | 20000-10-15.txt | 20000-15-20.txt | 30000-10-15.txt |
| 时间 | 4780048ms | 8034795ms | 9604886ms | 16008929ms | 14548607ms |

质量属性分析：在设计之初通过定义抽象类来给定功能实现的规范，对于后续的功能模块的修改和扩展提供了基础。将功能封装成单独的类有助于进行单元测试。对于可修改性而言，如果修改文件的格式，只需要实现对应的接口，就可以通过只修改原来的注册代码来实现功能的替换。