

Lab1

1. 书签栏的面向对象模型

我们采用树作为存储书签栏的数据结构。我们定义了Node和Tree两个类来描述书签栏的树形结构。

Node具体描述树的结点。我们定义了以下四种属性：

属性名	数据类型	描述
name	string	目录（书签）的名称
type	enum NodeType { leaf, branch }	结点的类型，值为leaf代表叶子结点，即书签，值为branch代表分支结点，即目录
sons	Array<Node>	当前结点的儿子结点
content	string	结点的内容，即书签的url信息

Tree具体描述书签栏的树形结构，有以下三种属性：

属性名	数据类型	描述
root	Node	树的根结点
corFilePath	string	书签文件的存放路径
BookmarkMap	Map<string,number>	书签url与对应访问次数的映射关系

比如，一个书签栏的数据样例如下：

#个人收藏

##课程

[elearning](https://elearning.fudan.edu.cn/courses)

参考资料

[Markdown Guide](https://www.markdownguide.org)

函数式

[JFP](https://www.cambridge.org/core/journals/journal-of-functional-programming)

面向对象

待阅读

[Category Theory](http://www.appliedcategorytheory.org/what-is-applied-category-theory/)

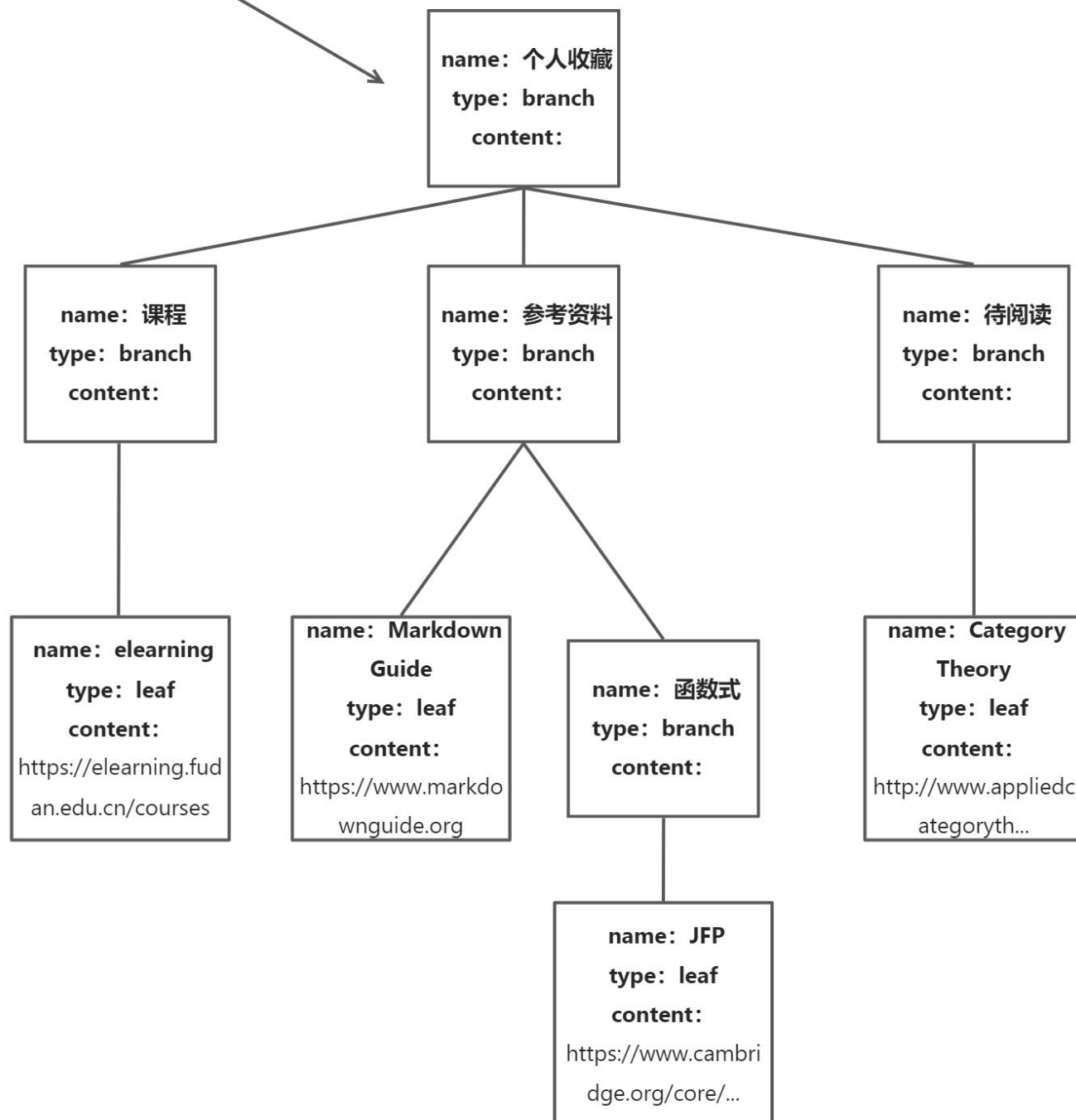
其树形结构如下：

Tree

root:

corPathFile: C:\\Users\\12828\\Desktop\\lab1\\VsCode_extention\\files\\1.bmk

BookmarkMap: {elearning : 0;Markdown Guide:0; JFP : 0; Categoty Theory : 0}



2. 设计模式使用

2.1 Adapter模式

在这次lab中，我们在两处运用到了adapter设计模式。

<1>我们在树形结构和文件操作之间设计了一个适配器。文件操作类中定义并实现了一系列处理文件的功能，这些功能是处理书签树形数据结构接口的所能利用的，所以我们通过一个适配器类将二者连接起来，代码如下：

```
1  //目标接口
2  interface TargetTree {
3      lsTree(): string;
4      writeToFile(content: string): void;
5      readFromFile(): string;
6      readArrFromFile(): Array<string>;
7  }
8  //适配器
9  class AdapterFromTreeToFile implements TargetTree {
10     private adaptee: FileOperation;
11     constructor(adaptee: FileOperation) {
12         this.adaptee = adaptee;
13     }
14     public lsTree(): string {
15         return this.adaptee.lsTreeString();
16     }
17     public writeToFile(content: string) {
18         this.adaptee.writeContent(content);
19     }
20     public readFromFile() {
21         return this.adaptee.readContent();
22     }
23     public readArrFromFile() {
24         return this.adaptee.readContentAsArray();
25     }
26 }
```

<2>另外，我们在命令语句和树的数据结构之间也使用了适配器的思想，在tree这一类中，我们定义了一些操作数据的方法，如addTitle(), addBookmark(), deleteTitle()等，这些功能是命令语句所需要的，所以我们采用了适配器模式，在利用tree类中这些功能实现了TargetCmd接口。

```

1  //目标接口
2  interface TargetCmd {
3      getFileStructure(): string;
4      getData(): string;
5      readBookmark(title: string): void;
6      lsTree(): unknown;
7      saveTree(): unknown;
8      openNewFile(filePath: string): void;
9      addTitle(title: string, folder?:string): void;
10     addBookmark(bkName: string, bkUrl: string, folder: string): void;
11     deleteTitle(title: string): void;
12     deleteBookmark(bkName: string): void;
13     showTree():void;
14 }
15 //适配器
16 class AdapterFromCmdToTree implements TargetCmd {
17     private adaptee: Tree;
18     constructor(adaptee: Tree) {
19         this.adaptee = adaptee;
20         this.adaptee.loadTreeFromFile();
21     }
22     getFileStructure(): string {
23         return this.adaptee.getFileStructure();
24     }
25     getData(): string {
26         return this.adaptee.getData();
27     }
28     readBookmark(title: string): void {
29         this.adaptee.readBookmark(title);
30     }
31     lsTree(): void {
32         this.adaptee.lsTree();
33     }
34     saveTree(): void {
35         this.adaptee.writeTreeIntoFile();
36     }
37     openNewFile(filePath: string): void {
38         this.adaptee.loadTreeFromFile(filePath);
39     }
40     showTree(): void {
41
42         this.adaptee.printTree();
43     }
44     addTitle(title: string, folder?: string): void {
45         console.log("add Tietle exc");
46         if (folder === undefined) {
47             this.adaptee.addSonBranch(title);

```

```

48     } else {
49         console.log("add Tietle exc2");
50         this.adaptee.addSonBranch(title, folder);
51     }
52 }
53 addBookmark(bkName: string, bkUrl: string, folder: string): void {
54     this.adaptee.addSonLeaf(bkName, bkUrl, folder);
55 }
56 deleteTitle(title: string): void {
57     this.adaptee.deleteNode(title);
58 }
59 deleteBookmark(bkName: string): void {
60     this.adaptee.deleteNode(bkName);
61 }
62 }
63 }

```

2.2 Command模式

对于命令行语句，我们采用Command设计模式，将发出请求(Invoker类)和执行请求(Receiver类)分隔开，二者之间通过具体的命令对象沟通，并且在这基础上实现了书签管理工具的redo和undo功能。

```

1  //调用者
2  class Invoker {
3      constructor(private command: Command) {
4          this.command = command;
5      }
6
7      public setCommand(command: Command) {
8          this.command = command;
9      }
10
11     public call() {
12         console.log("调用者执行命令command...");
13         this.command.execute();
14     }
15 }
16
17 //抽象命令
18 interface Command {
19     execute(): void;
20 }
21
22 //具体命令
23 class ConcreteCommand implements Command {
24     constructor(private receiver: Receiver) {
25         this.receiver = receiver;
26     }
27
28     public execute() {
29         this.receiver.action();
30     }
31 }
32 class AddTitleCommand implements Command {
33     constructor(private receiver: Receiver, private title: string) {
34         this.receiver = receiver;
35         this.title = title;
36     }
37     public execute() {
38         // Check logic
39         // Perform delete logic
40
41         // log logic
42         console.log("Add execute");
43         if (this.title.includes("at")) {
44             let devided = this.title.split("at");
45             let name = devided[0];
46             let folder = devided[1];
47             this.receiver.addTitle(name, folder);

```

```

48         } else {
49             this.receiver.addTitle(this.title);
50         }
51     }
52 }
53 class DeleteTitleCommand implements Command {
54     constructor(private receiver: Receiver, private title: string) {
55         this.receiver = receiver;
56     }
57     public execute() {
58         this.receiver.deleteTitle(this.title);
59     }
60 }
61 class AddBookmarkCommand implements Command {
62     constructor(private receiver: Receiver, private title: string) {
63         this.receiver = receiver;
64     }
65     public execute() {
66         this.receiver.addBookmark(this.title);
67     }
68 }
69 class DeleteBookmarkCommand implements Command {
70     constructor(private receiver: Receiver, private title: string) {
71         this.receiver = receiver;
72     }
73     public execute() {
74         this.receiver.deleteBookmark(this.title);
75     }
76 }
77 class OpenCommand implements Command {
78     constructor(private receiver: Receiver, private title: string) {
79         this.receiver = receiver;
80     }
81     public execute() {
82         this.receiver.open(this.title);
83     }
84 }
85 class BookmarkCommand implements Command {
86     constructor(private receiver: Receiver, private title: string) {
87         this.receiver = receiver;
88     }
89     public execute() {
90         this.receiver.deleteTitle(this.title);
91     }
92 }
93 class EditCommand implements Command {
94     constructor(private receiver: Receiver, private title: string) {
95         this.receiver = receiver;

```



```

96     }
97     public execute() {
98         this.receiver.deleteTitle(this.title);
99     }
100 }
101 class SaveCommand implements Command {
102     constructor(private receiver: Receiver) {
103         this.receiver = receiver;
104     }
105     public execute() {
106         this.receiver.save();
107     }
108 }
109 class ShowTreeCommand implements Command {
110     constructor(private receiver: Receiver, private title: string) {
111         this.receiver = receiver;
112     }
113     public execute() {
114         this.receiver.showTree();
115     }
116 }
117 class LsTreeCommand implements Command {
118     constructor(private receiver: Receiver) {
119         this.receiver = receiver;
120     }
121     public execute() {
122         this.receiver.lsTree();
123     }
124 }
125 class ReadBookmarkCommand implements Command {
126     constructor(private receiver: Receiver, private title: string) {
127         this.receiver = receiver;
128     }
129     public execute() {
130         this.receiver.readBookmark(this.title);
131     }
132 }
133
134
135 //接收者
136 class Receiver {
137     constructor() {
138         console.log("new Receiver constructed");
139     };
140     myCmd: TargetCmd = new AdapterFromCmdToTree(new Tree());
141     public action() {
142         console.log("接收者的action()方法被调用...");
143     }

```

```

144
145     public init() {
146         /**
147          * 初始化操作:
148          * 1.清空文件数据
149          * 2.添加个人收藏首行
150          */
151         // this.cleanData();
152         // this.fops.writeContent("# 个人收藏\n## 收藏夹1\n[elearning](http
153 s://elearning.fudan.edu.cn/courses)\n");
154     }
155
156     private getKeyIndex(strs: Array<string>, keyword: string): number {
157         // 返回strs中 有keyword的索引
158         let i: number = 0;
159         for (i = 0; i < strs.length; i++) {
160             if (strs[i].indexOf(keyword) !== -1) {
161                 return i;
162             }
163         }
164         return -1;
165     };
166
167     private getKeySIndex(strs: Array<string>, keyword: string): Array<num
168 ber> {
169         // 返回strs中 有多个keyword的索引
170         let i: number = 0;
171         let keyArr: Array<number> = [];
172         for (i = 0; i < strs.length; i++) {
173             if (strs[i].indexOf(keyword) !== -1) {
174                 keyArr.push(i);
175             }
176         }
177         return keyArr;
178     };
179
180     /**
181      * The Cmds receiver receives
182      */
183     public addTitle(title: string, folder?: string) {
184         console.log("接收者的addTitle()方法被调用...");
185         this.myCmd.addTitle(title, folder);
186     }
187
188     public deleteTitle(title: string) {
189         console.log("接收者的deleteTitle()方法被调用...");
190         this.myCmd.deleteTitle(title);
191     }

```

```

190
191 ▾ public addBookmark(args: string) {
192     console.log("addBookmark方法被调用");
193     let devide = args.split("at");
194     let folder: string = devide[1];
195     let bmkPart: string = devide[0];
196     let bmkArr: Array<string> = bmkPart.split('@');
197     let url: string = bmkArr[1];
198     let bkName: string = bmkArr[0];
199     this.myCmd.addBookmark(bkName, url, folder);
200 }
201
202 ▾ public deleteBookmark(args: string) {
203     console.log("deleteBookmark方法被调用");
204     this.myCmd.deleteBookmark(args);
205 }
206
207 ▾ public open(filePath: string) {
208     console.log("open()方法被调用...");
209     console.log("Path is", filePath);
210     this.myCmd.openNewFile(filePath);
211 }
212
213 ▾ public save() {
214     console.log("save()方法被调用...");
215     this.myCmd.saveTree();
216 }
217
218 ▾ public showTree() {
219     console.log("showTree()方法被调用...");
220     this.myCmd.showTree();
221 }
222
223 ▾ public lsTree() {
224     console.log("lsTree()方法被调用...");
225     this.myCmd.lsTree();
226 }
227
228 ▾ public readBookmark(title: string) {
229     console.log("readBookmark()方法被调用...");
230     this.myCmd.readBookmark(title);
231 }
232
233 ▾ public getData(): string {
234     return this.myCmd.getData();
235 }
236
237 ▾ public getFileStructure(): string {

```

```

238         return this.myCmd.getFileStructure();
239     }
240 }
241
242 class CommandPool {
243
244     receiver: Receiver;
245     redoStack: Array<string> = [];
246     undoStack: Array<string> = [];
247     constructor() {
248         console.log("new CmdPool constructed");
249         this.receiver = new Receiver();
250         this.redoStack = new Array<string>;
251         this.undoStack = new Array<string>;
252     };
253
254     public getData(): string {
255         return this.receiver.getData();
256     }
257     public getFileStructure(): string{
258         return this.receiver.getFileStructure();
259     }
260     public sendCommand(theCmd: string, args: string): void {
261
262         console.log("\n\n\n\n\n\n\n");
263         // 创建具体命令对象cmd并设定它的接受者
264         let cmd: Command = new ConcreteCommand(this.receiver);
265         switch (theCmd) {
266             case "addTitle":
267                 cmd = new AddTitleCommand(this.receiver, args);
268                 this.undoStack.push(theCmd + "|" + args);
269                 break;
270             case "deleteTitle":
271                 cmd = new DeleteTitleCommand(this.receiver, args);
272                 this.undoStack.push(theCmd + "|" + args);
273                 break;
274             case "addBookmark":
275                 cmd = new AddBookmarkCommand(this.receiver, args);
276                 this.undoStack.push(theCmd + "|" + args);
277                 break;
278             case "deleteBookmark":
279                 cmd = new DeleteBookmarkCommand(this.receiver, args);
280                 this.undoStack.push(theCmd + "|" + args);
281                 break;
282             case "open":
283                 cmd = new OpenCommand(this.receiver, args);
284                 break;
285             case "bookmark":

```

```

286         break;
287     case "edit":
288         break;
289     case "save":
290         cmd = new SaveCommand(this.receiver);
291         break;
292     case "undo":
293         this.undo();
294         return;
295     case "redo":
296         this.redo();
297         return;
298     case "showTree":
299         cmd = new ShowTreeCommand(this.receiver, args);
300         break;
301     case "lsTree":
302         cmd = new LsTreeCommand(this.receiver);
303         break;
304     case "readBookmark":
305         cmd = new ReadBookmarkCommand(this.receiver, args);
306         break;
307     default:
308         cmd = new ConcreteCommand(this.receiver);
309         break;
310 }
311 if (this.redoStack.length === 0) {
312     this.redoStack = [];
313 }
314
315 // 请求绑定命令
316 const ir = new Invoker(cmd);
317 console.log("客户访问调用者的call()方法...");
318 ir.call();
319 console.log(this.undoStack);
320 console.log(this.redoStack);
321 };
322
323 undo(): void {
324     if (this.undoStack.length === 0) {
325         return;
326     } else {
327         let theLastCmd = this.undoStack.pop();
328         if (theLastCmd === undefined) {
329             return;
330         }
331         let theLastCmd = theLastCmd.split("|")[0];
332         let args = theLastCmd.split("|")[1];
333         let cmd: Command;

```

```

334         switch (thelastcmd) {
335             case "addTitle":
336                 cmd = new DeleteTitleCommand(this.receiver, args);
337                 break;
338             case "deleteTitle":
339                 cmd = new AddTitleCommand(this.receiver, args);
340                 break;
341             case "addBookmark":
342                 args = args.split("@")[0];
343                 cmd = new DeleteBookmarkCommand(this.receiver, args);
344                 break;
345             case "deleteBookmark":
346                 cmd = new AddBookmarkCommand(this.receiver, args);
347                 break;
348             default:
349                 cmd = new ConcreteCommand(this.receiver);
350                 break;
351         }
352         const ir = new Invoker(cmd);
353         ir.call();
354
355         this.redoStack.push(theLastCmd);
356     }
357 }
358
359 redo(): void {
360     if (this.redoStack.length === 0) {
361         return;
362     } else {
363         let theLastCmd = this.redoStack.pop();
364         if (theLastCmd === undefined) {
365             return;
366         }
367         let thecmd = theLastCmd.split("|")[0];
368         let args = theLastCmd.split("|")[1];
369         let cmd: Command;
370         switch (thecmd) {
371             case "addTitle":
372                 cmd = new AddTitleCommand(this.receiver, args);
373                 break;
374             case "deleteTitle":
375                 cmd = new DeleteTitleCommand(this.receiver, args);
376                 break;
377             default:
378                 cmd = new ConcreteCommand(this.receiver);
379                 break;
380         }
381         const ir = new Invoker(cmd);

```

```
382         ir.call();
383     }
384 }
385 }
```