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

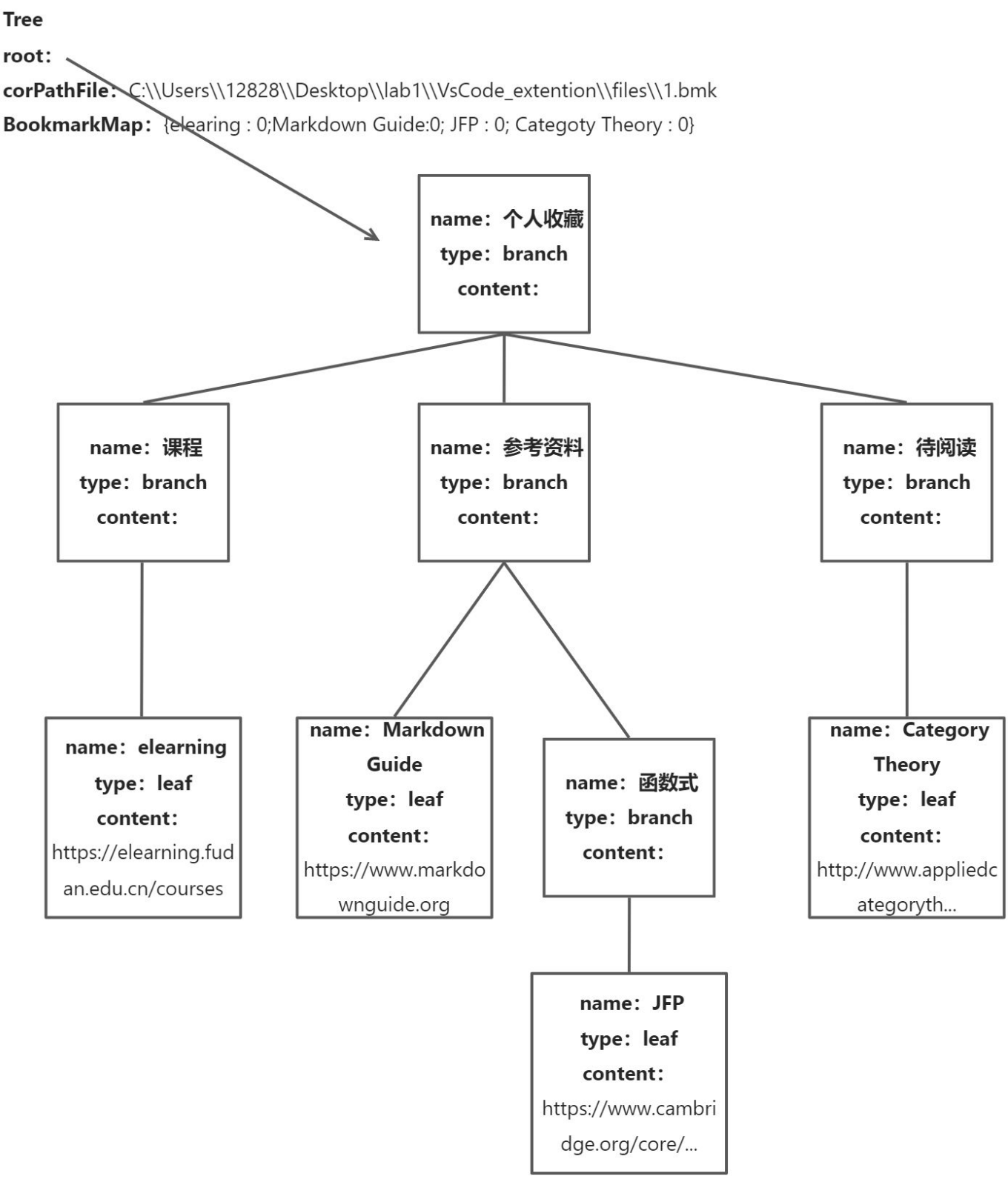
   Tree具体描述书签栏的树形结构，有以下三种属性：

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

[elearing] (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：



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：

```
//目标接口
interface TargetTree {
    lsTree(): string;
    writeToFile(content: string): void;
    readFromFile(): string;
    readArrFromFile(): Array<string>;
}
//适配器
class AdapterFromTreeToFile implements TargetTree {
    private adaptee: FileOperation;
    constructor(adaptee: FileOperation) {
        this.adaptee = adaptee;
    }
    public lsTree(): string {
        return this.adaptee.lsTreeString();
    }
    public writeToFile(content: string) {
        this.adaptee.writeContent(content);
    }
    public readFromFile() {
        return this.adaptee.readContent();
    }
    public readArrFromFile() {
        return this.adaptee.readContentAsArray();
    }
}
```

<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。

```
//目标接口
interface TargetCmd {
    getFileStructure(): string;
    getData(): string;
    readBookmark(title: string): void;
    lsTree(): unknown;
    saveTree(): unknown;
    openNewFile(filePath: string): void;
    addTitle(title: string, folder?:string): void;
    addBookmark(bkName: string, bkUrl: string, folder: string): void;
    deleteTitle(title: string): void;
    deleteBookmark(bkName: string): void;
    showTree():void;
}
//适配器
class AdapterFromCmdToTree implements TargetCmd {
    private adaptee: Tree;
    constructor(adaptee: Tree) {
        this.adaptee = adaptee;
        this.adaptee.loadTreeFromFile();
    }
    getFileStructure(): string {
        return this.adaptee.getFileStructure();
    }
    getData(): string {
        return this.adaptee.getData();
    }
    readBookmark(title: string): void {
        this.adaptee.readBookmark(title);
    }
    lsTree(): void {
        this.adaptee.lsTree();
    }
    saveTree(): void {
        this.adaptee.writeTreeIntoFile();
    }
    openNewFile(filePath: string): void {
        this.adaptee.loadTreeFromFile(filePath);
    }
    showTree(): void {

        this.adaptee.printTree();
    }
    addTitle(title: string, folder?: string): void {
        console.log("add Tietle exc");
        if (folder === undefined) {
            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(Receive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。

```
//调用者
class Invoker {
    constructor(private command: Command) {
        this.command = command;
    }

    public setCommand(command: Command) {
        this.command = command;
    }

    public call() {
        console.log("调用者执行命令command...");
        this.command.execute();
    }
}

//抽象命令
interface Command {
    execute(): void;
}

//具体命令
class ConcreteCommand implements Command {
    constructor(private receiver: Receiver) {
        this.receiver = receiver;
    }

    public execute() {
        this.receiver.action();
    }
}
class AddTitleCommand implements Command {
    constructor(private receiver: Receiver, private title: string) {
        this.receiver = receiver;
        this.title = title;
    }
    public execute() {
        // Check logic
        // Perform delete logic

        // log logic
        console.log("Add execute");
        if (this.title.includes("at")) {
            let devided = this.title.split("at");
            let name = devided[0];
            let folder = devided[1];
            this.receiver.addTitle(name, folder);
```

```
        } else {
            this.receiver.addTitle(this.title);
        }
    }
}
class DeleteTitleCommand implements Command {
    constructor(private receiver: Receiver, private title: string) {
        this.receiver = receiver;
    }
    public execute() {
        this.receiver.deleteTitle(this.title);
    }
}
class AddBookmarkCommand implements Command {
    constructor(private receiver: Receiver, private title: string) {
        this.receiver = receiver;
    }
    public execute() {
        this.receiver.addBookmark(this.title);
    }
}
class DeleteBookmarkCommand implements Command {
    constructor(private receiver: Receiver, private title: string) {
        this.receiver = receiver;
    }
    public execute() {
        this.receiver.deleteBookmark(this.title);
    }
}
class OpenCommand implements Command {
    constructor(private receiver: Receiver, private title: string) {
        this.receiver = receiver;
    }
    public execute() {
        this.receiver.open(this.title);
    }
}
class BookmarkCommand implements Command {
    constructor(private receiver: Receiver, private title: string) {
        this.receiver = receiver;
    }
    public execute() {
        this.receiver.deleteTitle(this.title);
    }
}
class EditCommand implements Command {
    constructor(private receiver: Receiver, private title: string) {
        this.receiver = receiver;
```

```
    }
    public execute() {
        this.receiver.deleteTitle(this.title);
    }
}
class SaveCommand implements Command {
    constructor(private receiver: Receiver) {
        this.receiver = receiver;
    }
    public execute() {
        this.receiver.save();
    }
}
class ShowTreeCommand implements Command {
    constructor(private receiver: Receiver, private title: string) {
        this.receiver = receiver;
    }
    public execute() {
        this.receiver.showTree();
    }
}
class LsTreeCommand implements Command {
    constructor(private receiver: Receiver) {
        this.receiver = receiver;
    }
    public execute() {
        this.receiver.lsTree();
    }
}
class ReadBookmarkCommand implements Command {
    constructor(private receiver: Receiver, private title: string) {
        this.receiver = receiver;
    }
    public execute() {
        this.receiver.readBookmark(this.title);
    }
}


//接收者
class Receiver {
    constructor() {
        console.log("new Receiver constructed");
    };
    myCmd: TargetCmd = new AdapterFromCmdToTree(new Tree());
    public action() {
        console.log("接收者的action()方法被调用...");
    }
```

```
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](https://elearning.fudan.edu.cn/courses)\n");
153     }
154
155     private getKeyIndex(strs: Array<string>, keyword: string): number {
156         // 返回strs中 有keyword的索引
157         let i: number = 0;
158         for (i = 0; i < strs.length; i++) {
159             if (strs[i].indexOf(keyword) !== -1) {
160                 return i;
161             }
162         }
163         return -1;
164     };
165
166     private getKeySIndex(strs: Array<string>, keyword: string): Array<number> {
167         // 返回strs中 有多个keyword的索引
168         let i: number = 0;
169         let keyArr: Array<number> = [];
170         for (i = 0; i < strs.length; i++) {
171             if (strs[i].indexOf(keyword) !== -1) {
172                 keyArr.push(i);
173             }
174         }
175         return keyArr;
176     };
177
178     /**
179      * The Cmds receiver receives
180      */
181     public addTitle(title: string, folder?: string) {
182         console.log("接收者的addTitle()方法被调用...");
183         this.myCmd.addTitle(title, folder);
184     }
185
186     public deleteTitle(title: string) {
187         console.log("接收者的deleteTitle()方法被调用...");
188         this.myCmd.deleteTitle(title);
189     }
```

```
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");
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":
```

```
                break;
            case "edit":
                break;
            case "save":
                cmd = new SaveCommand(this.receiver);
                break;
            case "undo":
                this.undo();
                return;
            case "redo":
                this.redo();
                return;
            case "showTree":
                cmd = new ShowTreeCommand(this.receiver, args);
                break;
            case "lsTree":
                cmd = new LsTreeCommand(this.receiver);
                break;
            case "readBookmark":
                cmd = new ReadBookmarkCommand(this.receiver,args);
                break;
            default:
                cmd = new ConcreteCommand(this.receiver);
                break;
        }
        if (this.redoStack.length === 0) {
            this.redoStack = [];
        }

        // 请求绑定命令
        const ir = new Invoker(cmd);
        console.log("客户访问调用者的call()方法...");
        ir.call();
        console.log(this.undoStack);
        console.log(this.redoStack);
    };

    undo(): void {
        if (this.undoStack.length === 0) {
            return;
        } else {
            let theLastCmd = this.undoStack.pop();
            if (theLastCmd === undefined) {
                return;
            }
            let thelastcmd = theLastCmd.split("|")[0];
            let args = theLastCmd.split("|")[1];
            let cmd:Command;
```

```
            switch (thelastcmd) {
                case "addTitle":
                    cmd = new DeleteTitleCommand(this.receiver, args);
                    break;
                case "deleteTitle":
                    cmd = new AddTitleCommand(this.receiver, args);
                    break;
                case "addBookmark":
                    args = args.split("@")[0];
                    cmd = new DeleteBookmarkCommand(this.receiver, args);
                    break;
                case "deleteBookmark":
                    cmd = new AddBookmarkCommand(this.receiver, args);
                    break;
                default:
                    cmd = new ConcreteCommand(this.receiver);
                    break;
            }
            const ir = new Invoker(cmd);
            ir.call();

            this.redoStack.push(theLastCmd);
        }
    }

    redo(): void {
        if (this.redoStack.length === 0) {
            return;
        } else {
            let theLastCmd = this.redoStack.pop();
            if (theLastCmd === undefined) {
                return;
            }
            let thecmd = theLastCmd.split("|")[0];
            let args = theLastCmd.split("|")[1];
            let cmd:Command;
            switch (thecmd) {
                case "addTitle":
                    cmd = new AddTitleCommand(this.receiver, args);
                    break;
                case "deleteTitle":
                    cmd = new DeleteTitleCommand(this.receiver, args);
                    break;
                default:
                    cmd = new ConcreteCommand(this.receiver);
                    break;
            }
            const ir = new Invoker(cmd);
```

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