



File System Management

课程名： 操作系统

授课老师： 王冬青

设计者： 杜书杨

学号： 1452981

设计时间： 2017年6月19日 – 2017年6月24日

1. 项目描述及需求

基本任务：

在内存中开辟一个空间作为文件存储器，在其上实现一个简单的文件系统；

退出这个文件系统时，需要该文件系统的内容保存到磁盘上，以便下次可以将其回复到内存中来。

文件目录采用多级目录结构，目录项目中包括：文件名、物理地址、长度等信息。

文件系统提供的操作：格式化、创建子目录、删除子目录、显示目录、更改当前目录、创建文件、打开文件、关闭文件、写文件、读文件、删除文件。

具体目标：

构建一个可视化文件系统，使得用户可以在图形界面上直接进行所有的操作。布局结构均模仿windows现有的文件系统，以列表形式显示文件和各子目录。用户可通过右键菜单进行操作，可通过双击打开文件或文件夹。

2. 解决方案

开发环境：

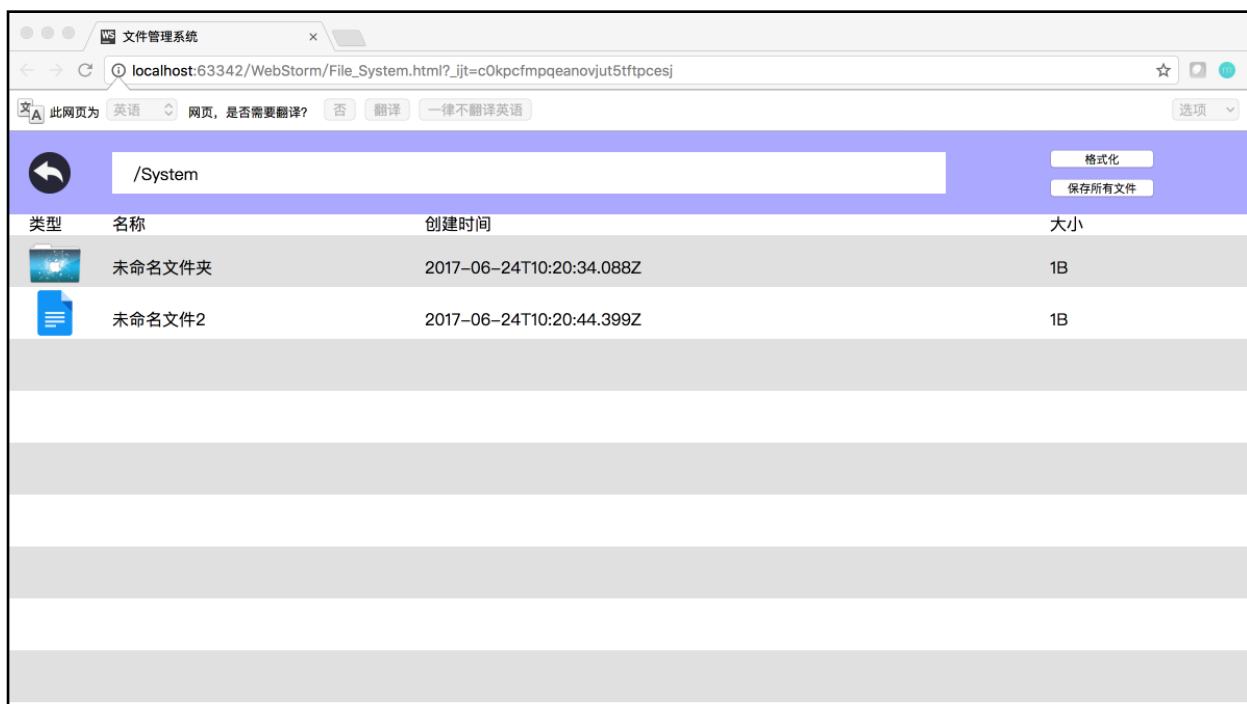
开发工具：WebStorm

开发语言：HTML5 & JavaScript

辅助工具：iWork Pages 文档，Adobe Photoshop帮助修改图片

成品：点击html文件运行

进入和读取：



基本界面如上图所示，打开后会先出现弹窗询问是否读取上次存储的数据，（如下图）如果是第一次打开或更换了新浏览器第一次打开，请点击取消，建立新的文件系统。如果

之前从来没有保存过，也请点击取消。上述三种情况如果点击确定读取上次的系统存档的话，会出现问题，无法添加和修改文件。在第一次进入并点击“保存所有文件”的按钮后，下一次用该浏览器打开就可以点击读取上次保存的数据，系统便会恢复到上次保存时的状态。



添加文件&文件夹:

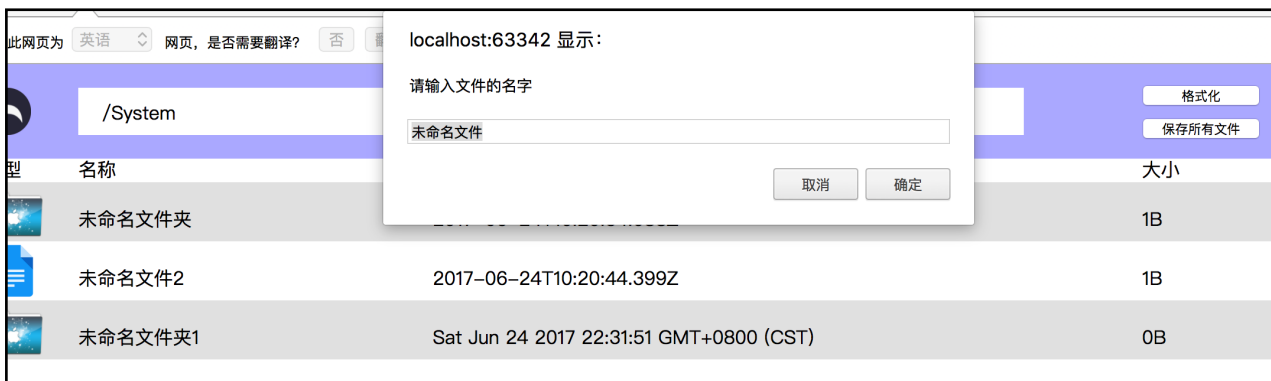


通过右键菜单添加文件夹到当前目录下。如上图所示，点击后会弹窗让用户输入文件名称，并将输入的名称与该目录下的其他文件夹名称对比，如果发生重名则不能添加，新建失败，如下图1所示；如果不重名则新建成功，文件夹会立刻显示在目录下，初始大小为0，里面没有任何元素，如下图2所示。

FILE SYSTEM MANAGEMENT





添加文件与添加文件夹相同，也是点击右键菜单中的“新建一个文件”按钮，系统在判断没有重名后会自动生成新的文件。新文件大小为1Byte，里面有1个字符“0”作为初始字符。






注：在本系统下，处于不同子目录的文件或文件夹可以重名，例如System下有叫“x”的文件夹，在x内也可以叫x的文件夹或文件。同一目录下的文件和文件夹也可以重名，例如System下可以有叫1个x的文件也可以同时有一个叫x的文件夹。重名仅针对同一子目录下的同种类型元素。

进入子目录和后退：




双击文件夹进入子目录（也可选中后右键点击打开），此时可以看到顶部路径显示栏的目录发生了变化，下方文件显示栏也发生了改变，在子目录中也可以进行各种操作，也可以新建文件夹，然后点击进入下一级子目录。点击左上角的后退键可以退回上一级目录，路径显示栏会发生变化，文件显示也发生改变。演示效果如下图。

	/System/未命名文件夹/未命名文件夹的			格式化
				保存所有文件
类型	名称	创建时间	大小	
	未命名文件a	Sat Jun 24 2017 22:59:52 GMT+0800 (CST)	5B	

	/System/未命名文件夹			格式化
				保存所有文件
类型	名称	创建时间	大小	
	未命名文件	2017-06-24T10:20:50.655Z	1B	
	未命名文件夹的	2017-06-24T10:21:01.113Z	5B	

文本的编辑和保存：

双击一个文档文件打开（也可选中后右键点击打开），系统会弹出一个窗口，在其中的文本框中可以修改文档的内容。初始的文档中只有“0”一个字符，改变内容后点击“保存并退出”，文件的大小也会改变，文本的长度即是文件的大小。文档内容发生改变后，本文件和文件各级目录文件夹的大小都会改变。如果点击“不保存退出”，文件内的文本将会保持上一次保存时的状态。演示效果如下图。

	/System/未命名文件夹			格式化
				保存所有文件
类型	名称	创建时间	大小	
	未命名文件	2017-06-24T10:20:50.655Z	11B	
	未命名文件夹的	2017-06-24T10:21:01.113Z	5B	

注：空格也会算作字符，例如下图中的“Hello World”，统计长度时会记为11个字符，因此文件大小也为11Byte。



文件及文件夹的删除：

左键选中要删除的文件后，右键弹出右键菜单，点选“删除”按钮，系统便会删除掉这个文件或文件夹，并释放其占用的内存空间以便留给其他文件，同时上层目录文件夹的大小也会因此改变。删除后，新建的文件或文件夹可以删除掉的元素的名字命名。



注：内存中只有留给文件的空间存储空间，文件夹不占空间。因此删除文件夹等同于在内存中删除掉其中的所有文件。

文件或文件夹的重命名：

左键选中要删除的文件后，右键弹出右键菜单，点选“重命名”按钮，系统会弹窗让用户输入文件或文件夹的新名称。与新建时一样，如果发生重名则会修改失败，文件已然会保留原来的名字。演示效果如下图。



FILE SYSTEM MANAGEMENT

/System/未命名文件夹				格式化	保存所有文件
类型	名称	创建时间	大小		
	未命名文件ex	2017-06-24T10:20:50.655Z	11B		
	未命名文件夹的	2017-06-24T10:21:01.113Z	5B		

格式化

点击页面右上角的按钮“格式化”，系统将删除所有的文件和文件夹以及内部的所有文本内容，保存这些数据的数组和树也会被全部初始化，内存也会全部被释放。若当前处于子目录下，则会先退回最外层目录再执行上述操作。System将回到最初空无一物的状态，用户可从头开始添加元素。

保存所有文件&读取并恢复

点击页面右上角的按钮“保存所有文件”，系统将存储数据的各个数组和树转换为JSON的string用localStorage存储在浏览器中。下一次使用同一浏览器打开时，在初始的弹窗选择确定读取，便恢复之前保存时系统内的各个文件以及里面的内容。有关这部分的代码见下图。

```
514 function save_all() // 将树、各个list和内存表转为JSON格式,存储到本地浏览器
515 {
516     var a = JSON.stringify(Total_space);
517     localStorage.setItem("storageSpace", a);
518     var b = JSON.stringify(Tree);
519     localStorage.setItem("TreeStructure", b);
520     var c = JSON.stringify(fileSize_list);
521     localStorage.setItem("fileSize", c);
522     var d = JSON.stringify(txtSize_list);
523     localStorage.setItem("txtSize", d);
524     var e = JSON.stringify(txtContent_list);
525     localStorage.setItem("txtContent", e);
526     localStorage.setItem("count", current);
527 }
528
529 function load_before() // 从浏览器localStorage中读取之前存储的数据,并显示出系统内的内容
530 {
531     Total_space = JSON.parse(localStorage.getItem("storageSpace"));
532     Tree = JSON.parse(localStorage.getItem("TreeStructure"));
533     fileSize_list = JSON.parse(localStorage.getItem("fileSize"));
534     txtSize_list = JSON.parse(localStorage.getItem("txtSize"));
535     txtContent_list = JSON.parse(localStorage.getItem("txtContent"));
536     current = parseInt(localStorage.getItem("count"));
537     current_file = Tree[0];
538     for(var i=1; i<current_file.child.length; i++) { // 显示出系统中最外层目录的内容
539         display(current_file.child[i], i);
540     }
541 }
```

部分代码展示：

由于这次代码较复杂，函数较多，仅展示部分。其余可打开js文件根据注释自行查看，注释已经将代码逻辑解释的足够详细，故不在这里占用过多版面。

```
5 // 结构体定义树的节点
6 function node(order, type, number, name, father, child, time){
7     this.order=order; // 在树中的序号
8     this.type=type; // 节点的种类是文件还是文件夹
9     this.number=number; // 文件或文件夹的编号(在相应的size数组和content数组中的序号)
10    this.name=name; // 该文件或文件夹的名称
11    this.father=father; // 该节点的父节点在树中的序号
12    this.child=child; // 该节点的孩子节点所组成的数组
13    this.time=time; // 该节点的创建时间
14 }
15
16 // 结构体定义当前选择
17 function chose(number, type, space, content){
18     this.number = number; // 当前选择的对象是当前目录的第几个子节点
19     this.type = type; // 当前选择的对象的类型是文件还是文件夹
20     this.space = space; // 当前选择的对象的文件显示栏
21     this.content = content; // 当前选择的对象所对应的的树中的节点
22 }
```

```
24 var path = document.createElement("path_display"); // 显示当前目录
25 path.innerText = "/System"; // 初始目录
26 document.getElementById("path").appendChild(path);
27 path.style.position="absolute";
28 path.style.left="20px";
29 path.style.top="10px";
30
31 var chosen = new chose(0, 0, 0, 0); // 当前选择的对象,初始为空
32
33 var fileSize_list = new Array(); // 存储各文件夹大小的数组
34 fileSize_list[0] = 0;
35
36 var txtSize_list = new Array(); // 存储各文件大小的数组
37 txtSize_list[0] = 0;
38
39 var txtContent_list = new Array(); // 存储各文件文本内容的数组
40 txtContent_list[0] = 0;
41
42 var Tree = new Array(); // 存储文件和文件夹信息的树
43 var c = new Array(); // 树的根节点的初始化
44 c[0] = 0;
45 Tree[0] = new node(0, "file", 0, "System", "none", c, 0);
46 var current_file = Tree[0]; // 当前所在目录
47 var current = 0; // 当前树中的节点数
```

```
74 // 结构体定义内存块
75 function Block(number, occupied, storage){
76     this.number = number; // 占用该内存块的文件或文件夹编号
77     this.occupied = occupied; // 该内存块目前被占用的内存
78     this.storage = storage; // 该内存块的总内存
79 }
80
81 // 初始化整个系统的内存
82 var Total_space = new Array(1024); // 该系统内存总大小为1024块,每块有64字节
83 for(var k = 0; k < Total_space.length; k++)
84 {
85     Total_space[k] = new Block(0, 0, 64);
86 }
87
88 firm(); // 提示是否要读取上次的存储记录,第一次进入应点取消
89 function firm() {
90     //利用对话框返回的值 (true 或者 false)
91     if (confirm("要读取上次创立的文件系统吗? (第一次打开或更换新浏览器后请点取消)")) {
92         load_before();
93         alert("已成功读取上次的文件系统!");
94     }
95     else {
96         alert("已新建文件系统!");
97     }
98 }
```



```
355 // 改变当前的目录,并显示改变后的目录中的内容
356 function change_file()
357 {
358     if(chosen.space !== 0) { // 改变选择对象的块的颜色为初始颜色
359         if ((chosen.number % 2) === 0)
360             chosen.space.style.background = "#FFFFFF";
361         else
362             chosen.space.style.background = "#E0E0E0";
363     }
364     for(var i=1; i<=current_file.child.length; i++) { // 删除当前显示的目录中的文件
365         var x = document.getElementById("s_"+i);
366         while(x.hasChildNodes()){
367             x.removeChild(x.firstChild);
368         }
369     }
370     for(var i=1; i<chosen.content.child.length; i++) { // 显示新目录中的文件
371         display(chosen.content.child[i], i);
372     }
373     current_file = chosen.content;
374     chosen = new chose(0,0,0,0); // 选择对象变为空
375     path_change(); // 改变当前目录的路径显示
376     if(current_file.child.length <= 10){
377         for(var i=10; i<=19; i++)
378             document.getElementById("s_"+i).style.display="none";
379     }
380 }
```

提一下内存分配部分，我做的是一个虚拟的内存分配，整个System共分为1024块，每一块大小为64Byte，如果一个文件的大小小于64字节，则可以存在一个块内，如果大于64字节，就要存放在多个块内。而由于后期修改文件内容等因素，这些块是不连续的，所以就给相应的块表上所存储的文件的序号，形成类似于FAT的链式结构（JS没有指针链表，所以就在数组里做）。

3. 工作流程

6.19-6.20 HTML写界面

6.20-6.21 JS完成右键菜单和基本的新建功能

6.22-6.23 JS完成数据结构部分，文件及其数据的存储，实现后退、删除操作

6.23-6.24 JS完成格式化、重命名、保存并恢复等功能

6.24-6.25 项目文档

5. 备注&小问题

1. 总结：经过两次的项目，我对JS也有了更多的认识，这次的代码相比于上次要精简的多，健壮性更强。即使要实现的功能多了许多，代码的总行数并没有多出太多，对比第一次电梯控制1000多行的代码，这次实现了多个功能共用一个函数，极大的减少了debug时的工作量。当然我的命名可能还是有些不规范，以后我会进一步改善。同时，这次项目还让我了解到很多浏览器的知识，以及JS自带的很多功能强大的函数，极大地提升了自己的能力，收获颇丰。

2. 备注：第一次打开HTML或更换新浏览器打开切记不可点击确定读取之前的系统，因为此时并没有存储任何数据，会导致程序出错，无法进行后续操作。应点击取消，新建文件系统，操作后点击保存，下次用同一浏览器打开时（因为localStorage是存在浏览器中），便可点击读取，系统便会恢复保存时的状态。

注：每一级最多共添加19个文件或文件夹，因为写HTML界面时，只做了19个块作为空位，初始显示前9个，大于9个会显示后面的。这个很容易改，添加成29甚至99个都没有问题，但这和项目内容主体没关系，只是个上限问题，我也是为了界面美观才做的空位并交替使用不同的颜色，不然大可不做背景颜色，无限添加。

最后说一句老师助教辛苦了！

有问题可以邮箱联系我：dushuyang@126.com