目录

[代码主要功能 - 2 -](#_Toc43747476)

[基本功能 - 2 -](#_Toc43747477)

[高级功能 - 5 -](#_Toc43747478)

[Github地址 - 9 -](#_Toc43747479)

[部署 - 9 -](#_Toc43747480)

# 代码主要功能

## 基本功能

1. **新增、删除、展现列表**

代码中实现了新增todo的功能，通过监听在todo的输入框上的keyup事件来实现，当keyup中的键盘码为13并且输入的内容不为空时，构造一条todo添加入预构建的数组中，并将其保存到本地存储中，具体代码如下：

$(".new-todo").addEventListener("keyup",function (event) {

if(event.keyCode !==13){

return ;

}

//添加操作

else{

if(currentMsg==''){

console.log("添加的todo名为空");

}

else{

items.push({name :currentMsg,completed:false});

//将输入框置空并同步

currentMsg='';

$(".new-todo").value=currentMsg;

saveMsg(currentMsg);

saveItems(items);

}

}

});

代码中实现了列表展示的功能，将数组中所有的项进行迭代，通过将completed属性与filter进行对比筛选出所有应该被展示的todo，然后以每一条筛选后的todo为基础构建一个li，并依次反序添加到预构建的无序列表中，其中每一个li中包含一个切换框，一个复制按钮，一个删除按钮。具体代码如下：

var todoList = $(".todo-list");

todoList.innerHTML='';

items.forEach(function (item,index) {

//筛选需要绘制的item

if(currentFilter=="All"

|| (currentFilter=="Active" && !item.completed)

|| (currentFilter=="Completed" && item.completed)){

//首先将对应的html元素创建出来，包括切换，标签，复制按钮，删除按钮

var itemLi = document.createElement("li");

var id ="item"+index;

itemLi.setAttribute("id",id);

if (item.completed) itemLi.classList.add("completed");

itemLi.innerHTML = [

'<div class="view" draggable="true">',

' <input class="toggle" type="checkbox">',

' <label class="todo-label">' + item.name + '</label>',

' <button class="duplicate"></button>',

' <button class="destroy"></button>',

'</div>'

].join('');

代码中实现了删除单条todo的功能，在绘制每一个todo时为对应的li添加了一个删除的按钮，并且保存了对应todo的下标，当点击删除按钮时就使用splice方法删除对应的数组元素，具体代码如下：

var deleteButton = itemLi.querySelector('.destroy');

deleteButton.addEventListener("click",function () {

items.splice(index,1);

},false);

并且该按钮通常情况下时隐藏的，在鼠标覆盖在对应的li上才显示，当鼠标覆盖在按钮上有一个加深的效果，具体代码如下：

.todo-list li .duplicate {

display: none;

position: absolute;

top: 0;

right: 60px;

bottom: 0;

width: 20px;

height: 20px;

margin: auto 0;

font-size: 10px;

background-color: darkcyan;

margin-bottom: 20px;

transition: color 0.2s ease-out;

}

.todo-list li:hover .destroy {

display: block;

}

.todo-list li .destroy:hover {

color: #af5b5e;

}

1. **全部完成**

代码中实现了全部完成或取消的功能，主要通过监听全部切换按钮的点击来实现的，保存下当前的状态来确定点击是进行完成还是取消，具体代码如下：

$(".toggle-all").addEventListener("click",function () {

items.forEach(function (item,index) {

//同样使用splice替换来进行

items.splice(index,1,{name:item.name,completed:currentState});

});

//记录当前状态

currentState=!currentState;

});

当todo状态为完成时，绘制时会为对应li加上相应的样式，具体表现为，勾选框的改变，标签加入删除线，颜色的改变等，具体代码如下：

if (item.completed) itemLi.classList.add("completed");

var itemToggle = itemLi.querySelector('.toggle');

itemToggle.checked = item.completed;

.todo-list li.completed label {

color: #d9d9d9;

text-decoration: line-through;

}

1. **删除已完成**

代码中实现了删除已完成的功能，主要通过监听删除已完成按钮的点击来实现的，并且该按钮在没有已完成的todo时不显示，具体代码如下：

items.forEach(function (item) {

if(item.completed===false){

leftItemsNumber++;

}

else{

haveCompleted=true;

}

});

$(".todo-count").innerText=leftItemsNumber+" item"+(leftItemsNumber>1?"s" : '')+" left";

//根据是否剩余completed来更改按钮的显示

if(!haveCompleted){

$(".clear-completed").style.display="none";

}

else{

$(".clear-completed").style.display="inline";

}

$(".clear-completed").addEventListener("click",function () {

items=new Proxy(items.filter(item=> !item.completed),arrayChangeHandler);

saveItems();

flush();

});

1. **保存页面状态**

代码中实现了保存页面状态的功能，保存的项包括todo列表与当前todo输入框中的内容，将存取本地存储封装为4个方法，待调用时进行调用，值得一提的是用本地存储中列表初始化内存中列表时，需要将其封装为proxy对象，代码如下：

function fetchItems(){

if(localStorage.getItem(STORAGE\_KEY)==="undefined"){

return [];

}

return JSON.parse(localStorage.getItem(STORAGE\_KEY) || '[]');

}

function saveItems(items){

localStorage.setItem(STORAGE\_KEY,JSON.stringify(items));

}

function fetchMsg(){

return JSON.parse(localStorage.getItem(MSG\_KEY) || '[]');

}

function saveMsg(msg){

localStorage.setItem(MSG\_KEY,JSON.stringify(msg));

}

function init(){

items = new Proxy(fetchItems(),arrayChangeHandler);

currentMsg=fetchMsg();

$(".new-todo").value=currentMsg;

flush();

}

## 高级功能

1. **使用proxy来代理item的变化，更加彻底地实现mvc设计模式**

在使用vue框架进行项目开发时，其双向绑定的功能十分好用，许多复杂的功能在引入双向绑定的机制后实现起来非常简单。

而在本项目中，todo列表的变化非常繁多，并且几乎每一次都涉及到对dom中列表元素的更新，很可能考虑不周漏掉更新，所以对todo列表的更新代理十分有必要。

一开始打算采用Array中的observe方法来实现，但是该方法已被弃用，大多数浏览器不再支持，所以采用了proxy对象的实现方法。在每一次更新todo列表后，handler执行将列表保存到本地存储及更新视图，具体代码如下：

var arrayChangeHandler = {

get: function(target,property) {

return target[property];

},set: function(target,property,value,receiver) {

target[property] = value;

saveItems(items);

flush();

return true;

}

};

但是该实现不能代理对todo列表中的属性进行修改的操作，所以我们将所有此类操作都使用替换操作实现，如使用

items.splice(index,1,{name:item.name,completed:currentState});

来实现单条todo的状态切换来保证该操作能被proxy捕获。

并且，该实现不能代理直接对整个列表进行赋值，所以当进行删除已完成时，不能直接将filter后的结果赋值给列表，而需要另外创建一个proxy对象，并且需要额外调用一次保存及更新的方法，具体代码如下：

items=new Proxy(items.filter(item => !item.completed),arrayChangeHandler);

saveItems();

1. **编辑单条todo**

代码中实现了编辑单条todo的功能，监听每个li中的标签的双击事件，当双击时li中增加一个输入框，默认输入框的值为原本的标签，当输入的键为esc时退出编辑，不做改变，当输入的键为回车时应用编辑的内容，值得一提的是，我们不直接对todo的内容进行更改，而是构造一个新的todo进行替换，这是为了让该操作能够被proxy进行捕获，代码如下：

var label = itemLi.querySelector('.todo-label');

//双击显示编辑框

label.addEventListener('dblclick', function() {

itemLi.classList.add("editing");

var edit = document.createElement('input');

var finished = false;

edit.setAttribute('type', 'text');

edit.setAttribute('class', 'edit');

edit.setAttribute('value', label.innerHTML);

function finish() {

if (finished) return;

finished = true;

itemLi.removeChild(edit);

itemLi.classList.remove("editing");

}

edit.addEventListener('blur', function() {

finish();

}, false);

edit.addEventListener('keyup', function(ev) {

//esc键退出编辑状态

if (ev.keyCode == 27) {

finish();

}

//Enter键应用编辑内容

else if (ev.keyCode == 13) {

label.innerHTML = this.value; items.splice(index,1,{name:this.value,completed:item.completed});

}

}, false);

itemLi.appendChild(edit);

edit.focus();

}, false);

1. **过滤todo**

代码中实现了过滤todo的功能，是通过监听三个过滤按钮的点击实现的，点击后更改内存中保存的过滤器变量，然后更改按钮的边框样式来指示当前应用的过滤器，而后重新绘制todo列表时会重新应用过滤器变量来筛选，代码如下：

$all(".filters a").forEach(function (ele) {

ele.addEventListener("click",function () {

currentFilter=ele.innerHTML;

$all(".filters a").forEach(function (element) {

element.classList.remove("selected");

});

ele.classList.add("selected");

flush();

})

});

items.forEach(function (item,index) {

//筛选需要绘制的item

if(currentFilter=="All"

|| (currentFilter=="Active" && !item.completed)

|| (currentFilter=="Completed" && item.completed))

1. **拖拽todo进行互换**

代码中实现了拖拽单个todo至另一个todo位置，然后将两个todo进行互换的功能，但该功能仅在pc浏览器能够进行使用，在移动端实现的是上下滑动交换相邻位置todo的功能，详细见高级功能6。拖拽功能的实现是通过在拖拽开始时保存下进行拖拽的todo的下标，然后再拖拽结束时通过事件目标可以获取到要交换的todo的下标(因为再绘制todo时，每个todo的id为“item”+下标)，然后就可以进行交换，具体代码如下：

itemLi.querySelector(".view").addEventListener("dragover",function (ev) {

ev.preventDefault();

});

itemLi.querySelector(".view").addEventListener("drop",function (ev) {

var temp = ev.target.parentNode.parentNode.id;

var tempIndex = parseInt(temp.substring(4));

[items[tempIndex],items[dragIndex]]=[items[dragIndex],items[tempIndex]];

dragIndex=-1;

});

itemLi.querySelector(".view").addEventListener("dragstart",function (ev) {

dragIndex=index;

});

1. **复制单个todo**

代码中实现了复制单个todo的功能，通过监听每个li中的复制按钮实现，当复制按钮被点击，会构建一个名字相同但未完成的todo将其加入列表，然后进行视图的更新，代码如下：

var duplicateButton=itemLi.querySelector('.duplicate');

duplicateButton.addEventListener(("click"),function () {

items.push({name:item.name,completed:false});

});

1. **上下滑交换相邻位置的todo**

代码中实现了上下滑交换相邻位置的todo的功能，通过移动端的touch事件实现，在touchstart发生时，保存初始点击的pageY与点击的todo的下标，在touchend发生时，取出pageY进行对比，如果相比开始位置上移，且初始点击的不是最上方的todo，那么就与上方的todo进行交换；同理，如果相比开始位置下移，且初始点击的不是最下方的todo，那么就与下方的todo进行交换，代码如下：

itemLi.querySelector(".view").addEventListener("touchend",function (ev) {

//获取停止滑动时的y坐标

let y=ev.changedTouches[0].pageY;

//如果是单击事件则不做改变，执行其他操作

if(y==beginy){

return;

}

//如果是上滑，与上边的元素互换

if(y<beginy && dragIndex<items.length-1){

[items[dragIndex+1],items[dragIndex]]=[items[dragIndex],items[dragIndex+1]];

}

//如果是下滑，与下边的元素互换

else if(dragIndex>0 && y>beginy){

[items[dragIndex-1],items[dragIndex]]=[items[dragIndex],items[dragIndex-1]];

}

});

itemLi.querySelector(".view").addEventListener("touchmove",function (ev) {

ev.preventDefault();

});

itemLi.querySelector(".view").addEventListener("touchstart",function (ev) {

//保存滑动开始的元素与y坐标

dragIndex=index;

beginy = ev.touches[0].pageY;

});

1. **背景音乐**

代码中实现了背景音乐的播放功能，使用播放控件控制播放，并且自动循环，代码如下：

<audio *src*="bg.mp3" *autoplay controls loop*></audio>

1. **删除特定todo**

代码中实现了批量删除特定todo的功能，使用监听todo的添加框的keyup事件实现，具体触发按钮为ctrl+enter，触发后会删除当前todo列表中所有名称为输入框值的项，因为该功能也是对items进行filter后直接赋值，所以该操作也需要手动调用flush方法一次，并且由于移动端没有keyup事件，所以该功能无法直接迁移到移动端，具体代码如下：

*if*(event.ctrlKey){  
 items=*new* Proxy(items.filter(item=> item.name.indexOf(currentMsg) === -1),arrayChangeHandler);  
 currentMsg='';  
 *$*(".new-todo").value=currentMsg;  
 *saveItems*(items);  
 *saveMsg*(currentMsg);  
 *flush*();  
 *return*;  
}

# Github地址

该项目的github为：<https://github.com/tuchengwu/todo-mobile>

# 部署

该项目已部署至阿里云服务器，url为：<http://123.57.129.198/todo-mobile/>