获取localStorage的长度： window.localStorage.length

添加/编辑localStorage的内容： window.localStorage.setItem(key,value);

根据对应的索引去获取对应的localStoratge的key的值：window.localStorage.key(index索引)；

根据对应的key获取对应的 value: window.localStorage.getItem(key);

不知道各位读者有没有注意到，刚刚存储进去的是int类型，但是打印出来却是string类型，这个与localStorage本身的特点有关，localStorage只支持string类型的存储。

if(!window.localStorage){

alert("浏览器支持localstorage");

}else{

var storage=window.localStorage;

//写入a字段

storage["a"]=1;

//写入b字段

storage.b=1;

//写入c字段

storage.setItem("c",3);

console.log(typeof storage["a"]);

console.log(typeof storage["b"]);

console.log(typeof storage["c"]);

//第一种方法读取

var a=storage.a;

console.log(a);

//第二种方法读取

var b=storage["b"];

console.log(b);

//第三种方法读取

var c=storage.getItem("c");

console.log(c);

}

修改

/\*分割线\*/

storage.a=4;

console.log(storage.a);

1、将localStorage的所有内容清除

var storage=window.localStorage;

storage.a=1;

storage.setItem("c",3);

console.log(storage);

storage.clear();

console.log(storage);

2、 将localStorage中的某个键值对删除

var storage=window.localStorage;

storage.a=1;

storage.setItem("c",3);

console.log(storage);

storage.removeItem("a");

console.log(storage.a);

**localStorage的键获取**

var storage=window.localStorage;

storage.a=1;

storage.setItem("c",3);

for(var i=0;i<storage.length;i++){

var key=storage.key(i);

console.log(key);

}

使用key()方法，向其中出入索引即可获取对应的键

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

本项目中对这些方法自行封装：

**export default class** todoDb {  
 constructor(name) {  
 **this**.**name** = name;  
 **if**(**JSON**.stringify((**this**.get(**this**.**name**))==**'{}'**)){  
 **this**.set([]);  
 }  
 }  
 set(val){  
 **window**.**localStorage**.setItem(**this**.**name**,**JSON**.stringify(val));  
 }  
 get(){  
 **return JSON**.parse(**window**.**localStorage**.getItem(**this**.**name**)) || {};  
 }  
}

localStorage 不能存数组，所以关键是  
存的时候（set）先把数组对象转成字符串： JSON.stringify()；  
取的时候（get）先把字符串转成数组对象： JSON.parse()。

构造函数 constructor 里先判断如果数据为空，则默认先设置一个空的数组；不然 被 map 遍历时出错（map遍历的是数组）。

入口元件引用

constructor(props) {  
 **super**();  
 *//初始化实例，localStorage命名为todoDb* **this**.**todoDb** = **new** todoDb(**'todoDb'**);  
 **this**.**state** = {  
 **tasks**: **this**.**todoDb**.get() || [],  
 **isAllDone**:**false** }  
}

之后

addTask(newTask) {  
 **this**.**state**.**tasks**.unshift(newTask);  
 **this**.**todoDb**.set(**this**.**state**.**tasks**); //对本地存储进行操作  
 **this**.setState({  
 **tasks**: **this**.**state**.**tasks**,  
 **isAllDone**: **false** })//对前端的显示进行操作  
}

操作的对象就是state重的 tasks数组。

deleteTask(index) {  
 **this**.**state**.**tasks**.splice(index,1);

。。。。。。。

changeDone(index,isDone){  
 **this**.**state**.**tasks**[index].isDone = isDone;

。。。。。。。。

deleteDone(){  
 **let** tasks = **this**.**todoDb**.get().filter(task=> !task.**isDone**);  
 **this**.**state**.**tasks** = tasks;

。。。。。。。。

通过filter选出布尔值为true的结果

searchTasks(val) {  
 **let** searchResult = **this**.**todoDb**.get().

filter((tasks) => tasks.task.includes(val));  
 **this**.**state**.**tasks** = searchResult;

/////////////////////////////////////////

**onKeyUp=**{**this**.addEnter.bind(**this**)

addEnter(event){  
 **if**(event.keyCode === 13)*//13就是enter键* {  
 **console**.log(**'addEnter13'**);  
 **this**.addOne();  
 }  
}

/////////////////////////////////////////

**onClick=**{**this**.addClick.bind(**this**)}

addClick(){  
 **this**.addOne();  
}

两个不同的动作，不同的函数，都指向了同一个函数：

addOne

addOne(){  
 *// let task = this.refs.inputTask.value;* **let** task = ReactDOM.**findDOMNode**(**this**.**refs**.inputTask).**value**;  
 **if**(!task) **return**;  
 **let** newTask = {  
 **task**: task,  
 **isDone**: **false** }  
 **this**.**props**.addTask(newTask);  
 ReactDOM.**findDOMNode**(**this**.**refs**.inputTask).**value**=**''**;  
 ReactDOM.**findDOMNode**(**this**.**refs**.inputTask).**focus**();  
}

# JavaScript unshift() 方法

unshift() 方法可向数组的开头添加一个或更多元素，并返回新的长度。

arrayObject.unshift(newelement1,newelement2,....,newelementX)

### 返回值

arrayObject 的新长度。

请注意，unshift() 方法不创建新的创建，而是直接修改原有的数组。

spread 操作符 在子组件向孙组件传递父组件的数据时，特别好用。比如子组件 TodoMain.js 向 孙组件 MainItem.js 传递父组件 Todo.js 的数据：<MainItem {...this.props}/> 。

<**div className="todo-main"**>  
 <**ul**>  
 {**this**.**props**.**tasks**.**map**((tasks,index)=>{  
 **return** <**MainItem key=**{index} {...tasks} **index=**{index}{...**this**.**props**}/>  
 })}  
 </**ul**>  
 <**MainStatus deleteDone=**{**this**.**props**.deleteDone.bind(**this**)}{...**this**.**props**} />  
</**div**>

**filter**  
过滤，遍历数组，对每一个元素触发一个回调函数，经处理判断、保留或删除当前元素，最后返回一个新数组；  
如，在 Todo.js 的 searchTasks方法中，就是通过 filter 筛选出搜索结果的：

|  |  |
| --- | --- |
| 1 | let searchResult = this.todoDb.get().filter((tasks) => tasks.task.includes(val)); |

this.todoDb.get() 得到的是一个对象为元素的数组，具体形式如下：  
[{'task':'任务1','isDone':true},{'task':'任务2','isDone':false}]  
箭头函数无需 return 。

**some()、every()**  
遍历数组，对每一个元素触发一个回调函数，通过判断，返回一个布尔值。  
some()：只要有一个满足判断，就返回true，  
every()：只要有一项不满足判断，就返回false。

localStorage 不能存数组，所以关键是  
存的时候（set）先把数组对象转成字符串： JSON.stringify()；  
取的时候（get）先把字符串转成数组对象： JSON.parse()。

构造函数 constructor 里先判断如果数据为空，则默认先设置一个空的数组；不然 被 map 遍历时出错（map遍历的是数组）。