javascript

3月23日

for 循环

```
//var, let 报错,const
        //var 定义的是全局变量
        //let 定义是局部变量
        //const 定义的常量，值不能改变
        /* 三个表达式及循环体*/
        for(let i = 0; i < 5; i++){
            console.log(i);
        }
        //console.log("i=",i )
        //求1-100 所有偶数之和
        let sum = 0; //和
        for(let i = 1; i <= 100; i++){
            if(i % 2 == 0){//判断是否是偶数
                sum += i; //等价于 sum = sum + i
            }
        }
        console.log("1-100偶数之和是：",  sum);
        //随机100个 1-100之间的整数，求 大于等于60 的个数
        Math.random(); //随机生成[0-1)之间的小数
        //parseInt()转成整数的系统函数
        for(let i = 0; i < 10; i++){
            a = parseInt(Math.random() * (100-1)) + 1;
            console.log(a);
        }
```

随机数

```
Math.random(); //随机生成[0-1)之间的小数
parseInt()   把小数转成整数的系统函数 比如parseInt(1.2) 得到1,
parseInt(2.6) 得到2
```

随机10个 1-100之间的整数

```
for(let i = 0; i < 10; i++){
    a = parseInt(Math.random() * (100-1)) + 1;
    console.log(a);
}
```

随机100个 1-100 之间的整数，求 大于等于60 的 个数

```
console.log("**************************");
let num = 0; //大于等于60 的个数
for(let i = 0; i < 100; i++){
    a = parseInt(Math.random() * 100) + 1;
    if( a >= 60){
        num++;
    }
}
console.log("大于等于60的随机数个数是：", num)
```

数组

当有多个变量要存储时，可以考虑用数组变量存储

比如 30个 学生成绩。

数组的定义

- let arr = new Array() ; //Array是js中的关键字
- let arr2 = []; //第二种定义
- let num = [10, 15, 17, 20, 16, 30, 28]

数组中元素的访问：

数组中的每个元素访问语法：数组名[下标]

arr[0] = 98; //给数组中的元素赋值

arr[1] = 99;

打印数组

```
console.log(arr)
```

随机 10 个 100-200 之间的整数存入 数组中，并打印出来

```
let num = [];
for(let i = 0; i < 10; i++){
    num[i] = parseInt(Math.random()*101) + 100;
}
```

数组求最大值

```
let num = [10, 15, 17, 20, 16, 30, 28]
//数组的元素个数:数组.length
let max = num[0];//假设第一个元素最大
for(let i = 1; i < num.length; i++){
    if(num[i] > max){
        max = num[i];
    }
}
console.log("最大值:", max);
```

String.fromCharCode

把整数转变为对应的字符

```
document.write("<br/>"+  String.fromCharCode(20013)); //打印出 "中" 字
```

charCodeAt()

```
document.write("中".charCodeAt(0)); 把“中” 字转变为 ascii对应的整数
document.write("中国人名".charCodeAt(1)) 把国 字转变为 ascii对应的整数
```

随机字符

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script>

        //ascii 码，unicode 码
        document.write(String.fromCharCode(122));
        //String.fromCharCode 把整数转变为对应的字符
        document.write("<br/>"+  String.fromCharCode(20013));
        //charCodeAt() 把字符转变为对应的整数
        document.write("中".charCodeAt(0));
        document.write("<br/>")
        document.write("中国人名".charCodeAt(1))
        document.write("<br/>"+  String.fromCharCode(22269));
        //随机100个a-z 的字符
        //a-97, z-122, A-65, Z-90
        for(let i = 0; i < 5; i++){
            c = parseInt(Math.random()*(26)) + 97;
            //document.write(c + ", ");
```

```
        }
        //随机1000个 a-z的字符，统计m 出现的次数
        let m = 0; //字符m出现的个数
        for(let i = 0; i<1000; i++){
            c = String.fromCharCode(parseInt(Math.random()*(26)) + 97);
            if(c == "m"){
                m++;
            }
        }
        document.write("m出现的次数是：" + m)
        //随机1000个 a-z的字符，
        //统计每个字符出现的次数，并打印出来
        let times=[]; //数组用来存储每个字符出现的次数
        document.write("<h1>" + times[0] + "</h1>");
        //times[0]a的次数,times[1]b的次数.....
        //times[0] =0 , times[1] = 0, times[2]=0
        //当c=97,times[0]++
        //当c=98,times[1]++
        //当c=99,times[2]++
        for(let i = 0; i<1000; i++){
            c = parseInt(Math.random()*(26)) + 97;
            if(times[c-97]){
                times[c - 97]++;
            }
            else{
                times[c - 97] = 1;
            }
        }
        console.log(times);
        for( let i = 0; i < times.length; i++){
            document.write(String.fromCharCode(97+i) + ":" + times[i] + "次
<br/>")
        }
        document.write("<br/>***************<br/>")
        //for each 循环,遍历times数组中的每个元素，赋值给变量i
        for(let i in times){
            document.write("<br/>" + i + ":" + times[i]  + "<br/>")
        }
    </script>
</head>
<body>

</body>
</html>
```

3月30 日

模板字符串

模板字符串，是一种 方便格式化字符串的方法，可以比较便捷的格式化字符串

语法特点：两个反引号，中间可以是任意格式的html,css,代码,变量用${}包含

示例：

```
function gen(){
            let html = "";
            const bigFactory = ["淘宝", "京东", "腾讯"];
            for(let i =0; i < bigFactory.length; i++){
                html += "<li>";
                html += bigFactory[i];
                html += "</li>"
            }
            html = "";
            for(const x of bigFactory){
                html += "<li style='color:red;'>";
                html += x;
                html += "</li>"
            }
            console.log(html);//根据选择器 查找标签对象,innerHTML 任意标签里面的html
            document.querySelector("#ul").innerHTML = html;
        }
        function gen2(){
            let html = "";
            const bigFactory = ["淘宝", "京东", "腾讯"];
            bigFactory.forEach(function(x){
                html += "<li style='color:red'>";
                html += x;
                html += "</li>"
            })
            html = "";
            bigFactory.forEach(x =>{
                html += `<li style="color:blue">${x}</li>`; //反引号
            })
            document.querySelector("#ul").innerHTML = html;
        }
```

innerHTML

innerHTML 是js中 标签 的属性

可以用来设置或者获取任意标签里面的html内容

```
a = document.querySelector("#ul").innerHTML;获取html
//设置html
document.querySelector("#ul").innerHTML = "<h1>hello</h1>";
```

箭头函数

箭头函数 是一种 定义方法的 简洁语法

俗称语法糖

语法格式：(参数) => {

  函数体

}

```
//定义无参的方法，方法名是add4
let add4 = () =>{ //无参必须加括号

}
//定义一个参数的方法，方法名是add5
let add5 = x => {

}
//定义2个带默认值参数的方法
let add3 = (num1=5, num2=3) => {
    return num1 + num2;
}
```

几个函数

```
JSON.stringify(对象-数组或其他自定义类);//把对象转为json字符串
```

JSON.parse();//把json字符串解析转换为js对象(一般只能是数组)

typeof(变量) 返回变量的类型

```
 //数组的转换
function exp1(){
    let arr = [1,2,3, "ab", "xy"];
    console.log(arr, typeof(arr));
    //转换为json字符串
    let str = JSON.stringify(arr);
    console.log(str, typeof(str));
    //把json字符串转为 对象
    let arr2 = JSON.parse(str)
```

```
    console.log(arr2, typeof(arr2));
}
//set需要先转为数组，再转为json字符串
 function save1(){
     let cons = new Set(["中国", "美国", "俄罗斯", 10, 100]);
     //先把set转为数组,set.values()返回set的所有元素
     arr = Array.from(cons.values())
 }
```

浏览器本地存储

在HTML5中，新加入了一个localStorage特性，这个特性主要是用来作为本地存储来使用的，解决了cookie存储空间不足的问题(cookie中每条cookie的存储空间为4k)，localStorage中一般浏览器支持的是5M大小，这个在不同的浏览器中localStorage会有所不同。

基本用法

存储数据

可以使用 `localStorage.setItem(key, value)` 方法来存储数据，其中 `key` 是键名，`value` 是对应的值。例如：

```
localStorage.setItem('name', 'value');
```

获取数据

使用 `localStorage.getItem(key)` 方法来获取存储在LocalStorage中的数据。例如：

```
var username = localStorage.getItem('name');
console.log(username); // 输出 'value'
```

删除数据

可以使用 `localStorage.removeItem(key)` 方法来删除LocalStorage中的数据。例如：

```
localStorage.removeItem('name');
```

清空所有数据

使用 `localStorage.clear()` 方法可以清空LocalStorage中的所有数据：

```
localStorage.clear();
```

注意事项

- LocalStorage 中存储的数据是以字符串的形式存储的，如果需要存储其他类型的数据，需要先将其转换为字符串。
- 每个域名对应一个独立的LocalStorage空间，不同域名之间的LocalStorage数据无法共享。
- LocalStorage 中的数据是持久化的，除非通过代码清除或用户手动清除，否则数据会一直存在。

示例

```
htmlCopy Code<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>LocalStorage 示例</title>
</head>
<body>
    <script>
        // 存储数据
        localStorage.setItem('name', 'value');

        // 获取数据
        var username = localStorage.getItem('name');
        console.log(username); // 输出 'value'

        // 删除数据
        localStorage.removeItem('name');

        // 清空所有数据
        localStorage.clear();
    </script>
</body>
</html>
```

类和对象

简介

JavaScript 是一种基于原型的面向对象编程语言，通过使用类和对象来组织代码和数据。类用于创建对象的模板，而对象则是类的实例化。

创建类

可以使用 `class` 关键字来创建一个类。例如：

```
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }

  greet() {
    console.log(`Hello, my name is ${this.name} and I'm ${this.age} years
old.`);
  }
}
```

创建对象

可以使用 `new` 关键字来创建一个类的实例。例如：

```
const person1 = new Person('Alice', 30);
const person2 = new Person('Bob', 25);
```

类的方法

类的方法是在类中定义的函数，可以通过类的实例来调用。例如：

```
person1.greet(); // 输出 "Hello, my name is Alice and I'm 30 years old."
person2.greet(); // 输出 "Hello, my name is Bob and I'm 25 years old."
```

类的属性

类的属性是在类的构造函数中使用 `this` 关键字定义的变量。例如：

```
console.log(person1.name); // 输出 "Alice"
console.log(person2.age); // 输出 25
```

示例

- 定义学生类,在一个新文件中student.js

  ```
  class Student{ //Student是类的名字,自定义类型的名字
      //construnctor 是构造方法(自动调用的一个方法)
      constructor(sno,name,age){
          this.sno = sno; //this表示当前对象，表示自己
          this.name = name;
          this.age = age;
      }
      //定义类的普通函数
      print(){
          return `sno:${this.sno}, name:${this.name}, age:${this.age}`
      }
  }
  ```

- 使用类创建对象

```
<script>
        //定义3个学生的对象
        let stu1 = new Student(1, "草丛盖伦", 999);
        let stu2 = new Student(2, "武器大师", 888);
        let stu3 = new Student(3, "上单鳄霸", 777);
        //调用print函数，打印它的返回值
        document.write(stu1.print() + "<br/>")

        document.write(stu2.print() + "<br/>")

        document.write(stu3.print() + "<br/>")
    </script>
```

4月13日

数组的几个特殊方法

map方法

一、介绍

`map()` 是 JavaScript 中的一个高阶函数，它创建一个新数组，其结果是该数组中的每个元素都调用一个提供的函数后返回的结果。`map()` 方法特别适用于对数组的每个元素执行某种操作，并收集结果到一个新数组中。

二、基本语法

```
array.map(function(currentValue, index, arr), thisArg)
```

- `currentValue`：必须。当前元素的值。
- `index`：可选。当前元素的索引值。
- `arr`：可选。当前元素属于的数组对象。
- `thisArg`：可选。执行 callback 时用作 `this` 的对象

三、使用示例

1. **基本使用**：将数组中的每个元素都乘以2。

```
let numbers = [1, 2, 3, 4, 5];
let doubled = numbers.map(function(num) {
    return num * 2;
});
console.log(doubled); // [2, 4, 6, 8, 10]
```

2.**使用箭头函数**：箭头函数可以使代码更加简洁。

```
let numbers = [1, 2, 3, 4, 5];
let doubled = numbers.map(num => num * 2);
console.log(doubled); // [2, 4, 6, 8, 10]
```

3.**利用索引值**：有时你可能需要利用元素的索引值来进行操作。

```
let characters = ['a', 'b', 'c', 'd', 'e'];
let indexedChars = characters.map((char, index) => char + index);
console.log(indexedChars); // ['a0', 'b1', 'c2', 'd3', 'e4']
```

4.**arr**：

`arr` 是 `map()` 方法中回调函数的第三个参数，它代表了正在被操作的原始数组。虽然这个参数在大多数情况下不是必需的，因为它只是简单地引用了调用 `map()` 的数组，但在某些复杂的函数内部，如果你需要明确知道原始数组，这个参数就会很有用。

```
let originalArray = [1, 2, 3, 4, 5];  //原始数组
let modifiedArray = originalArray.map(function(currentValue, index, arr) {
    // 在这里，arr 就是 originalArray
    console.log(arr); // 会打印出整个原始数组
    // 可以做一些基于原始数组的操作
    // ...
    return currentValue * 2;
});
```

5.**thisArr**：有时你可能需要利用元素的索引值来进行操作。

`thisArg` 是 `map()` 方法的第二个参数，用于设置回调函数内部 `this` 的值。在 JavaScript 中，函数的 `this` 值取决于函数是如何被调用的。在 `map()` 方法中，如果你不特意设置 `thisArg`，那么回调函数内部的 `this` 值将是 `undefined`（在严格模式下）或者是全局对象（非严格模式下，通常是 `window`）。

通过传递 `thisArg` 参数，你可以控制回调函数中 `this` 的指向。这在某些情况下可能很有用，尤其是当你需要在回调函数中访问某个对象的方法或属性时。

```
let obj = {
    multiplier: 2,
    double: function(n) {
        return n * this.multiplier;
    }
};
```

```
let numbers = [1, 2, 3, 4, 5];
let doubledNumbers = numbers.map(function(num) {
    // 在这里，this 指向 obj，因为我们将 obj 作为 thisArg 传递给了 map()
    return this.double(num);
}, obj);

console.log(doubledNumbers); // [2, 4, 6, 8, 10]
```

在这个例子中，`thisArg` 设置为 `obj`，因此在 `map()` 的回调函数中，`this` 指向 `obj`。这使得我们可以在回调函数中使用 `obj` 的方法和属性。

然而，在现代 JavaScript 开发中，箭头函数经常被使用，而箭头函数没有自己的 `this` 值，它们会捕获定义时所在上下文的 `this` 值。因此，在使用箭头函数作为回调函数时，`thisArg` 参数通常是不必要的。

四、注意事项

1. `map()` 不会改变原始数组，而是返回一个新数组。
2. 如果数组为空，则 `map()` 返回一个空数组。

filter 方法

一、介绍

`filter()` 方法是 JavaScript 数组的一个内置方法，用于创建一个新数组，其中包含通过测试函数（由你提供）的所有元素。这个方法对于过滤出数组中满足特定条件的元素非常有用。

二、基本语法

```
array.filter(function(currentValue, index, arr), thisArg)
```

- `currentValue`：必须。当前元素的值。
- `index`：可选。当前元素的索引值。
- `arr`：可选。当前元素属于的数组对象。
- `thisArg`：可选。执行 callback 时用作 `this` 的对象。

三、使用示例

1. **基本使用**：筛选出数组中所有的偶数。

```
let numbers = [1, 2, 3, 4, 5, 6];
let evenNumbers = numbers.filter(function(num) {
    return num % 2 === 0;
});
console.log(evenNumbers); // [2, 4, 6]
```

1. **使用箭头函数**：箭头函数可以使代码更加简洁。

```
let numbers = [1, 2, 3, 4, 5, 6];
let evenNumbers = numbers.filter(num => num % 2 === 0);
console.log(evenNumbers); // [2, 4, 6]
```

1. **利用索引值**：在某些情况下，你可能需要利用元素的索引值来进行过滤。

```
 let array = ['apple', 'ba', 'ch', 'dates', 'eld'];
            let longFruits = array.filter((fruit, index) => {
                console.log(fruit, index);
                return fruit.length > index;
            });
            console.log(longFruits); // [apple', 'ba', 'dates']
```

1. **链式调用**：你可以与其他数组方法（如 `map()`）链式调用 `filter()`。

```
let numbers = [1, 2, 3, 4, 5, 6];
let doubledEvenNumbers = numbers.filter(num => num % 2 === 0).map(num => num *
2);
console.log(doubledEvenNumbers); // [4, 8, 12]
```

四、注意事项

1. `filter()` 不会改变原始数组，而是返回一个新数组。
2. 如果没有任何元素通过测试，则返回一个空数组。
3. `filter()` 方法会对数组中的每个元素调用一次提供的函数，并且如果该函数返回 `true`，则将该元素包含在新数组中。
4. 提供的函数应该返回一个布尔值来指示元素是否应该包含在新数组中。

JSON

一、JSON简介

JSON（JavaScript Object Notation，JavaScript对象表示法）是一种轻量级的数据交换格式。它基于ECMAScript的一个子集，采用完全独立于语言的文本格式来存储和表示数据。简单、清晰的层次结构使得JSON成为理想的数据交换语言。 易于人阅读和编写，同时也易于机器解析和生成。

二、JSON语法规则

JSON数据格式有两种：对象（Object）和数组（Array）。

1. 对象：由大括号包围，数据由键值对构成，键值对之间用逗号分隔。键是字符串，值可以是字符串、数字、对象、数组、布尔值或null。

示例：

```
let jon =
{
  "name": "张三",
  "age": 30,
  "isStudent": false
}
```

2. 数组：由方括号包围，数据由值构成，值之间用逗号分隔。值可以是字符串、数字、对象、数组、布尔值或null。

示例：

```
let jsonArr1 = [
  "苹果",
  "香蕉",
  "橘子"
]

let jsonArr2 = [{
  "name": "张三",
  "age": 30,
  "isStudent": false
},
                {
  "name": "张三他爹",
  "age": 60,
  "isStudent": false
}
                ]
```

三、JSON在JavaScript中的操作

1. **JSON字符串转JavaScript对象**

使用`JSON.parse()`方法可以将JSON字符串转换为JavaScript对象。

示例：

```
let jsonStr = '{"name": "张三", "age": 30, "isStudent": false}';
let obj = JSON.parse(jsonStr);
console.log(obj.name); // 输出：张三
```

2. **JavaScript对象转JSON字符串**

使用`JSON.stringify()`方法可以将JavaScript对象转换为JSON字符串。

示例：

```
let obj = {
  name: "张三",
  age: 30,
  isStudent: false
};
let jsonStr = JSON.stringify(obj);
console.log(jsonStr); // 输出：{"name":"张三","age":30,"isStudent":false}
```

四、注意事项

1. JSON中的属性名（键）必须用双引号或单引号括起来。属性值如果是字符串，也必须用双引号括起来。
2. JSON中的字符串值不能包含控制字符，不能包含引号，必须用反斜杠转义。
3. JSON中不能使用JavaScript的保留字作为键。
4. JSON中不能使用函数或undefined。

五、动态添加键值对

在JavaScript中，动态地向一个JSON对象添加键值对是非常直接的。由于JSON对象在JavaScript中实际上就是一个普通的对象，你可以使用点符号（`.`）或者方括号（`[]`）来添加新的属性。以下是一些示例：

- 使用点符号添加键值对

```
let jsonObject = {
  name: "张三",
  age: 30
};

// 动态添加一个新的键值对
jsonObject.isStudent = false;

console.log(jsonObject);
// 输出: { name: '张三', age: 30, isStudent: false }
```

- 使用方括号添加键值对

```
let jsonObject = {
  name: "张三",
  age: 30
};

// 假设我们有一个变量，它包含要添加的键名
let newKey = "isStudent";
let newValue = false;

// 使用方括号和变量来动态添加键值对
jsonObject[newKey] = newValue;
```

```
console.log(jsonObject);
// 输出: { name: '张三', age: 30, isStudent: false }
```

Axios

一、简介

Axios 是一个基于 Promise 的 HTTP 客户端，用于浏览器请求。它提供了统一的 API 来处理 HTTP 请求和响应，支持 Promise API、拦截请求和响应、转换请求和响应数据、取消请求、自动转换 JSON 数据等功能。

二、安装

在浏览器项目中，你可以通过 CDN 引入 Axios：

```
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
```

三、基本使用

1、**发起 GET 请求**

```
axios.get('https://api.example.com/data')
  .then(function (response) {
    // 处理响应数据
    console.log(response.data);
  })
  .catch(function (error) {
    // 处理错误
    console.log(error);
  });
```

2、**发起 post 请求**

```
axios.post('https://api.example.com/data', {
  key1: 'value1',
  key2: 'value2'
})
.then(function (response) {
  console.log(response.data);
})
.catch(function (error) {
  console.log(error);
});
```

3、**提交Json数据到后端接口**

```
const data = {
  key1: 'value1',
  key2: 'value2'
};

axios.post('https://api.example.com/data', data, {
  headers: {
    'Content-Type': 'application/json'
  }
})
.then(function (response) {
  console.log(response.data);
})
.catch(function (error) {
  console.log(error);
});
```

在这个例子中，我们创建了一个包含要发送数据的对象 `data`。然后，我们使用 `axios.post` 方法发起 POST 请求，并传递三个参数：请求的 URL、要发送的数据对象以及一个配置对象，该对象指定了请求头 `Content-Type` 为 `application/json`。这样，Axios 就会自动将 JavaScript 对象转换为 JSON 格式的字符串，并将其包含在请求体中。