21.4.12学习记录

一、数据结构

之前有数组、json、二叉树....

现在是set数据结构：类似数组，但是里面不能有**重复值**

```
let setArr = new Set(['a', 'b', 'c', 'a']);//a,b,c
setArr.add('a');//添加
setArr2.add('a').add('b').add('c').add('d');//连续添加
setArr.delete('b');//删除
console.log(setArr.has('a'));//判断有此值否
setArr.clear();//清空所有
let newArr = [...new Set(arr)];//数组去重
let wset = new WeakSet();//初始赋值不可以，用add方法，无size
```

现在还有map数据结构：类似json，

--但是json的键值只能是字符串，而map的键值key可以是任意类型

```
let map = new Map();
map.set(key, value);
let res = map.get(json);
map.forEach((val, key) => {console.log(val, key);})
let wmap = new WeakMap();//key只能是对象
```

二、数字变化

二进制：`let a = 0b010101;//21`

八进制：`let b = 0o666;//十进制438`

十六进制：#ccc`let c = 0x999;//十进制2457`

数字：

```
console.log(Number.isNaN(NaN));//true
console.log(Number.isFinite(a));//true
console.log(Number.isFinite(NaN));//false
console.log(Number.isInteger(12.5));//false
```

安全整数： +- (2^53 -1)

```
console.log(Number.isSafeInteger(2 ** 53 - 1));//true
console.log(Number.isSafeInteger(-(2 ** 53) + 1));//true
console.log(Number.MAX_SAFE_INTEGER);//9007199254740991
```

```
console.log(Number.MIN_SAFE_INTEGER);//-9007199254740991
```

Math:

```
console.log(Math.trunc(4.5));//4 , 只保留整数部分
console.log(Number.parseInt('4a'));//4

console.log(Math.sign(5));//1
console.log(Math.sign(-5));//-1
console.log(Math.sign(0));//0
console.log(Math.sign(-0));//-0

console.log(Math.cbrt(27));//立方根3
```

三、ES9（ES2018）

1、命名捕获　(?<名字>)

我认为就是把匹配到的东西换个名字且存到了groups中可解构

```
let reg = /(?<year>\d{4})-(?<month>\d{2})-(?
<day>\d{2})/;
console.log(str.match(reg).groups);
//{year: "2021", month: "04", day: "12"}
let { year, month, day } = str.match(reg).groups;
console.log(year, month, day);//2021 04 12
```

反向引用语法（普通）

--\1 \2　　$1 $2

反向引用命名捕获　\k<名字>

我认为就是利用捕获的名字

```
let reg = /^(?<lrj>welcome)-\k<lrj>$/;
//welcome-welcome为true
let reg = /^(?<lrj>welcome)-\k<lrj>-\1$/;
//welcome-welcome-welcome
```

替换 $<>

```
let str = '2021-04-12';
let reg = /(?<year>\d{4})-(?<month>\d{2})-(?
<day>\d{2})/;
let newStr = str.replace(reg, '$<month>/$<day>/$<year>');
```

```
console.log(newStr);//04/12/2021
```

2、点 .

--之前' .' 在正则里表示匹配任意东西，但是不包括\n

--现在有dotAll模式

3、标签函数

```
function fn(args) {//标签函数使用
    console.log(args);//["welcome", raw: Array(1)]
    return 1;
}
console.log(fn`welcome`);//1
```

三、proxy（代理）

扩展（增强）对象一些功能

作用:

--vue中拦截

--预警、上报、扩展功能、统计、增强对象...

proxy是设计模式一种，代理模式

```
//new Proxy(target,handler);
// let obj = new Proxy(被代理的对象,对代理对象的操作)
    // handler:
    //     {
    //         set(){} 设置时操作
    //         get(){} 获取时操作
    //         deleteProperty(){}  删除
    //         has(){}  检查是否有此东西
    //         apply(){} 调用函数处理
    //     }
```

四、Reflect（反射）

Object.xxx 语言内部方法

通过Reflect对象身上直接拿到语言内部东西

五、JQuery

--JQuery一款优秀的js库并简化原生js操作

--1.x版兼容ie678，相对其他版本文件较大，兼容浏览器很棒

入口函数

原生和jq的入口函数获取DOM元素

```
window.onload = function () {
    var img = document.getElementsByTagName('img')[0];
    console.log(img);
}
$(document).ready(function () {
    var $img = $('img')[0];
    console.log($img);
});
```

区别：

1、原生JS和JQuery入口函数的**加载模式**不同，原生JS会等到DOM元素加载完毕，并且图片也加载完毕才会执行

JQuery会等到DOM元素加载完毕，但**不会**等到图片也加载完毕

2、原生js如果编写了多个入口函数，后面会**覆盖**前面

而JQ中后面的**不会覆盖**前面的

3、四种写法，第三种最简便

```
$(document).ready(function () {
    alert(1);
});
jQuery(document).ready(function () {
    alert(2);
});
$(function () {
    alert(3);
});
jQuery(function () {
    alert(4);
});
```

使用多个框架，JQuery冲突时，

```
//1、释放$的使用权
//释放操作必须在编写其他JQuery代码之前编写
//释放后不能再使用$符号，改用JQuery
// jQuery.noConflict();
//2、自定义访问符号
var nj = jQuery.noConflict();
nj(function () {
    alert(1);
});
```

核心函数：

```
//$():代表调用JQuery的核心函数
//1、接收一个函数
//$(function(){});
$(function () {
    //2、接收一个字符串（字符串选择器或代码片段）
    //返回一个JQuery对象，对象中保存了找到的DOM元素
    let $box1 = $(".box1");
    let $box2 = $("#box1");
    let $p = $("<p>我是段落</p>");
    //3、接收一个DOM元素
    let span = document.getElementsByTagName("span")[0];
    console.log(span);
    //被包装成一个JQuery对象返回给我们
    let $span = $(span);
    console.log($span);
});
```

JQuery对象是一个伪数组！

静态方法：

each：可遍历伪数组，默认返回值遍历谁就返回谁，不支持在回调函数中对遍历数组进行处理

```
let arr = [1, 3, 5, 7, 9];
let obj = { 0: 1, 1: 3, 2: 5, 3: 7 };
//原生的forEach方法只能遍历数组，不能遍历伪数组
arr.forEach((val, index) => {
    console.log(index, val);
})
//JQuery的each方法
$.each(arr, (index, val) => {
    console.log(index, val);
})
$.each(obj, (index, val) => {
    console.log(index, val);
})
```

map：可遍历伪数组，默认返回值是空数组，可以再回调函数中通过return对遍历数组进行处理，并生成一个新数组

```
let arr = [1, 3, 5, 7, 9];
let obj = { 0: 1, 1: 3, 2: 5, 3: 7 };
arr.map((value, index, array) => {
    console.log(index, value, array);
});
$.map(arr, (val, index) => {
    console.log(index, val);
})
$.map(obj, (val, index) => {
    console.log(index, val);
})
```

$.trim()：去除空格

```
let str = '    lrj    ';
console.log(`---${str}----`);//---    lrj    ----
let newStr = $.trim(str);
console.log(`---${newStr}----`);//---lrj----
```

$.isWindow();

```
//判断传入的对象是否为window对象
let w = window;
let res = $.isWindow(w);
console.log(res);//true
```

$.isArray();

```
//判断传入的是否为真数组
let arr = [1, 3, 5, 7, 9];
let res2 = $.isArray(arr);
console.log(res2);//true
```

$.isFunction();

```
//判断传入是否为函数
let fn = function () { };
let res3 = $.isFunction(fn);
console.log(res3);//true
console.log($.isFunction(jQuery));//true
```

暂停ready方法的执行：

```
$.holdReady(true);//暂停ready方法的执行
```

数字+null=2

数字+undefined=NaN

数字+true=2

数字+false=2

数字+[{a:1}]=2[object Object]

.....

字符串+null=01null

字符串+undefined=01undefined

字符串+true=01true

字符串+false=01false

字符串+[{a:1}]=01[object Object]