

js正式课第三周

回调函数

- 回调函数：把一个函数的定义（A）作为参数传给另一个函数(B),函数A称为B的回调函数 B函数运行完后才会运行A函数
- 回调函数考虑的几个方面：
 - 1.回调函数是否有参数
 - 2.回调函数执行的次数
 - 3.回调函数是否有返回值

```
var ary = [1,2,3,4,5];
Array.prototype.myMap = function(callback,context){
  //先处理第二个参数，判断是否传了
  context = context || window;
  //this指的是数组ary callback执行的次数是数组的长度
  var newAry = []; //用来存回调函数的返回值
  for(var i = 0; i < this.length; i++){
    var res = callback.call(context, this[i], i, this);
    newAry.push(res);
  }
  return newAry;
}
var res = ary.myMap(function(item, index, arr){
  console.log(this, item);
  return item * 2
}, {});
console.log(res);
```

数组中迭代的方法

- 1.forEach 遍历数组 第一个参数是一个回调函数，第二个参数是context(改变回调函数中this关键字)

```
var res = ary.forEach(function(item, index, arr){ //声明式
  //item 指数组中每一项
  //index 每一项对应的索引
  //arr 原数组
  //console.log(item);
  //console.log(this);
```

```
},{})  
console.log(res); //undefined
```

- 2.map 遍历数组 第一个参数是一个回调函数，第二个参数是context(改变回调函数中this关键字) map方法有返回值

```
var ary = [1,2,3,4,5];  
var res = ary.map(function(item,index,arr){  
    return item*2    //把回调函数执行的返回值放在一个新数组中，最后将新数组作为map方法的返回值  
},{})
```

- 3.some true/false 有一个为真则返回true,所有为假才返回false

```
var ary = [1,2,3,4,5];  
var res = ary.some(function(item,index,arr){  
    return item>2&&item<4;  
});  
console.log(res);
```

- 4.every true/false 有一个为假则会返回false,所有为真才为true

```
var ary = [1,2,3,4,5];  
var res = ary.every(function(item,index,arr){  
    return item>2;  
});  
console.log(res);
```

- 5.filter 过滤 把满足条件的留下，不满足的过滤掉 返回一个新数组

```
var ary = [1,2,3,4,5];  
var res = ary.filter(function(item,index,arr){  
    return item>2;  
});  
console.log(ary); // [1,2,3,4,5]  
console.log(res); // [3,4,5]
```

- 6.find 返回满足条件的这一项，一旦找到不会继续往下查找 找不到返回undefined
// findIndex 返回满足条件的这一项的索引 找不到返回-1

```
var ary = [1,2,13,14,15];  
var res = ary.find(function(item,index,arr){
```

```
return item>15;
});
console.log(res);// undefined
```

- 7.reduce和reduceRight 对每一项累计的结果 例如每一项累加

```
var ary = [10,2,13,14,15];
//reduce 第一个参数 回调函数 第二个参数用来设置累计项初始值
var res = ary.reduce(function(prev,cur){
//prev 累计项 cur 数组的每一项
//prev 若没有设置prev的初始值，则它的初始值是数组的第一项
return prev+cur
/*
*      prev  cur
* 1.    5    10    15
* 2.    15    2    17
* 3.    17    13    30
* 4.    30    14    44
* 5.    44    15    59
* */
},5);
console.log(res);
```

正则

- 正则是用来处理字符串的
- 正则的概念：相当于是一个模型,通过这个模型（验证的规则）匹配符合的字符串
- 正则的作用：匹配(reg.test())和捕获(reg.exec(),str.match(),str.split())
- 正则的组成部分：元字符和修饰符

特殊含义的元字符:

\ 转义

\d 匹配0~9之间的数 \D

\w 匹配字母a-zA-Z，数字0~9，下划线_ \W

\s 匹配不可见的字符 空格，制表符等 \S

\b 匹配一个单词的边界 \B

\n 换行

. 除换行以外的任意字符

[a-z] a~z之间的一个

[^a-z] 匹配不是a-z之间的任意字符

| 或

() 分组

^ 以...开头

\$ 以...结尾

量词元字符

+ 1到多次

* 0到多次

? 0次或1次 可有可无

{n} 匹配n次

{n,} 至少匹配n次

{n,m} 最少匹配n次，最多匹配m次

修饰符

g global 全文查找

i ignoreCase 忽视大小写

m multiline 多行匹配

[]

- []中的横杠表示ascii码中连续的字符
- 特殊的字符在[]中表示字符本身的意思 例如+*.?等 特殊字符不包括\d,\W...
- []中没有两位数

()

- 可以改变优先级,分组
- 括号内匹配到的内容可以提取出来
 - /(\\d+)/g ==>{0}

(?:) 匹配不捕获

正则的定义

- // 字面量的方式
- new RegExp() 构造函数的方式

正则方法

exec

- 返回值:数组 第一项：匹配的内容 第二项：匹配内容的起始索引 第三项 原字符串

exec在（ ）分组中

- 第一项：正则匹配的内容
- 从第二项开始是分组的内容
- 倒数第二项是匹配内容的起始索引
- 倒数第一项原字符串

replace 替换（字符串方法）

```
var str = "11a22b33c";  
//回调函数参数，次数(匹配的次数)，返回值(可有可无 需要替换原字符串内容时要写return )  
str=str.replace(/((\d+)/g,function($0,$1,$2){  
/*正则不加分组  
参数部分：  
    第一个参数：匹配的内容  
    第二个参数：匹配内容的起始索引  
    第三个参数：原字符串  
正则加了分组 能把分组里的内容单独提取出来  
    第一个参数：匹配的内容  
    从第二个参数开始是分组匹配的内容  
    倒数第二项 匹配内容起始索引  
    倒数第一项 原字符串*/  
    console.log(arguments);  
    //console.log($0,$1,$2);  
    return $0*2 //用return 后的值替换匹配的内容  
});  
console.log(str);
```

- 贪婪和懒惰

```
/*var reg = /\d+/; //正常情况能多拿绝不少拿  
var reg = /\d+?/; //量词后面加? 解决贪婪性 能少绝不多拿  
var str = "2018zhufeng2019";  
console.log(reg.exec(str));*///数组，第一项匹配到的内容 第二项匹配内容的起始索引 第三项 原字符串  
var reg = /\d+/g; //加修饰符g解决懒惰性 修改reg.lastIndex的值，reg.lastIndex决定从字符串哪个位置开始查找  
var str = "2018zhufeng2019";
```

match方法 一次性拿到所有匹配的内容

- console.log(str.match(reg));

- 封装一个match方法

```
RegExp.prototype.myMatch = function(str){
  //先判断下是否加g this.global ->true
  if(!this.global) return this.exec(str);
  //this 指 reg
  var res = null;
  var ary = [];
  while(res = this.exec(str)){
    ary.push(res[0]);
  }
  return ary;
}
console.log(reg.myMatch(str));
```

\d \w \s \n . \b [] a|b () ^ \$ + * ? {n}{n,}{n,m} g,i,m

reg.test()

- 验证字符串是否跟正则相匹配

reg.exec() str.match() str.split()

- 捕获（把字符串中匹配的内容拎出来）

常用正则

- /^[u0021-\u002E]\$/; //用十六进制表示没法操作的字符 不足四位在前面补零
- /^[u4e00-\u9fa5]{2,5}\$/; //匹配中文字符2位~5位
- /^(1[8-9]|[2-5]\d|6[0-5])\$/; //匹配年龄18-65 18-19 20-59 60-65
- /^18|19\$/; //18开头或者19结尾的字符串
- /^(18|19)\$/; //表示只能是18或19
- var reg = /\d+name\d+/g; //name表示普通字符，不表示变量
- var reg = new RegExp("\d+" + name + "\d+", "g");
 - 1.构造函数的方式可以表示变量，字面量不行 2.构造函数中\代表一个\

时间字符串格式化

- 方法


```
String.prototype.myFormatTime = function myFormatTime(template = '{0}年{1}月{2}
```

日 {3}时{4}分{5}秒') {

let ary = this.match(/\d+/g).map(item => (item < 10 ? 0 +item : item));

return template.replace(/{\d+}/g,...[,index] => ary[index] || 00);

};

- 需求

let str = 2018-4-30 2:23 ;

console.log(str.myFormatTime({0}/{1}/{2} {3}时{4}分{5}秒));

获取地址栏中的参数

- 排除法

获取地址栏中的参数并放入对象中

```
var str = "http://www.baidu.com?id=557131028857&name=zhufeng&age=8";
//{id:557131028857,name:"zhufeng",age:"8"}
var reg = /([^?&=]+)=([^?&=]+)/g;
var obj = {};
str.replace(reg,function($0,$1,$2){
    obj[$1] = $2;
});
console.log(obj);
```

JS盒子模型

- JS中提供了与盒子模型相关的属性
- css盒子模型 width,height,border,padding,margin

偏移量相关属性

- ele.offsetLeft 左偏移
- ele.offsetTop 右偏移
- ele.offsetParent 父集
- 参照物：最近的已经定位的父级元素,若没有找到已经定位的父级元素，则参照物是body
 - 已经定位元素指设置position:relative|absolute|fixed
- 什么是偏移量？
 - 当前元素外边框到参照物内边框的距离
- 例题:不管当前元素的参照物是谁，要求当前元素到body的偏移量

```
function offset(ele){
//先获取当前元素到参照物的偏移量
    var l = ele.offsetLeft;
    var t = ele.offsetTop;
    var p = ele.offsetParent;
    while(p&&p!==document.body){ //排除ele是body元素的情况跟p是body的情况
l+=p.offsetLeft+p.clientLeft;
t+=p.offsetTop+p.clientTop;
p = p.offsetParent
    }
    return {
        l:l,
        t:t
    }
}
console.log(offset(inner).l);
```

获得任意css属性值

- window.getComputedStyle(oDiv,null)
- ie678支持属性ele.currentStyle

client系列(跟溢出的内容无关)

- clientWidth = width+padding(左右)
- clientHeight = height+padding(上下)
- clientLeft 左边框
- clientTop 上边框

offset系列(跟溢出内容无关)

- offsetWidth = width+padding(左右)+border(左右)=clientWidth+border(左右)
- offsetHeight = height+padding(上下)+border(上下)=clientHeight+border(左右)
- 与偏移量相关
 - offsetParent 参照物
 - offsetLeft 左偏移量
 - offsetTop 上偏移量

scroll系列 (跟溢出内容有关)

- $\text{scrollWidth} \approx \text{真实宽度} + \text{左padding值}$ 约等于真实宽度
- $\text{scrollHeight} \approx \text{height (真实高度)} + \text{上padding值}$ 约等于真实的高度

为什么是约等于？

- 各个浏览器的行高不一样
- 相同浏览器是否设置overflow属性，值也不一样
- 跟滚动条相关
 - `scrollLeft` 横向卷出去的宽度
 - `scrollTop` 纵向卷出去的高度

最小值是0，最大值是真实高度

浏览器兼容处理

- 1.通过判断属性的方式
 - `window.getComputedStyle`
 - “`getComputedStyle`” in window
- 2.检测数据类型的方式
 - `typeof`
 - `instanceof` any `instanceof Array` 可以检测出对象的细分类型
 - `constructor` 指向所述的类 `ary.constructor`（若原型重写，`constructor`会有问题）
 - `Object.prototype.toString.call(null)` “[object Null]”最精准检测数据类型的方式
 - `Object.prototype.toString.call([])` “object Array” 前面的是数据类型，后面的是所属的类
- 3.判断浏览器
 - `/MSIE [6-8].0/.test(navigator.userAgent)`
 - `navigator.userAgent.indexOf("MSIE 8.0")==-1` 说明不是IE8浏览器

获取一屏的高度和整个文档的高度

一屏的高度

`document.documentElement.clientHeight || document.body.clientHeight`

文档的高度

`document.documentElement.scrollHeight || document.body.scrollHeight`

若没有溢出的内容，一屏的高度和文档的高度是一样

图片懒加载

当页面加载的时候用一张默认图片（非常小）代替所有图片，当需要加载这个图片的时候再加载真实图片

为什么使用图片懒加载优化

1. 加快首屏渲染时间

假如真个页面有100张图片，我们用一张默认图片代替，虽然也是用了100张图片，但是只从后台请求了一次数据，另外99次都是走缓存(304)

2. 图片会一次性加载完成，不会出现一层一层的加载

图片防盗用

也是优化的一种 防止其他网站用自己网站的图片

控制台network操作

Disable cache 禁用缓存

offline-网络设置 弱网环境调试 选择 fast/slow 3G

onscroll事件

- 是滚动条事件 滚动条滚动的时候出发绑定事件
 - `document.onscroll = function (){}`

onload监听事件

- 所有资源加载完成执行
 - `ele.onload = function (){}`
 - `window.onload = function (){} //只能绑定一次`

onerror事件

- 资源加载失败时执行
 - `ele.onerror = function () {}`

document.documentElement 与 document.body的应用场景

- 获取 scrollTop 方面的差异
 - 在chrome(版本 52.0.2743.116 m)下获取scrollTop只能通过 `document.body.scrollTop`
 - 页面存在DTD, 使用`document.documentElement.scrollTop`获取滚动条距离
 - 页面不存在DTD,使用`document.documentElement.scrollTop` 或 `document.body.scrollTop`都可以获取到滚动条距离

tagName 获取元素的标签名

- 语法：`ele.tagName`
 - 返回元素大写的标签名“ELE”是一个字符串

jQuery

三种each

原型上的each 给实例调用的

jQuery选择器

```
//=>JQ选择器: 基于各种选择器创建一个JQ实例(JQ对象)
//1.selector 选择器的类型(一般都是字符串, 但是支持函数或者元素对象)
//2.context 基于选择器获取元素时候指定的上下文(默认document)
//JQ对象: 一个类数组结构(JQ实例), 这个类数组集合中包含了获取到的元素
$('tbody tr td')
```

```

* JQ对象（类数组）=>JQ实例
*   0: div.tabBox
*   length: 1
*   context: document
*   selector: '.tabBox'
*
*   __proto__: jQuery.prototype
*       add
*       ...
*       __proto__: Object.prototype
*       hasOwnProperty
*       ...

```

* 把JQ对象和原生JS对象之间相互的转换

*

* [把JQ->原生JS]

* JQ对象是一个类数组集合，集合中每个索引对应的都是原生JS对象，

我们基于索引获取即可

* let \$tabBox=\$('.tabBox'); 变量名前面是以\$开始的，一般代表基于JQ选择器获取的结果

```

*   let tabBox=$tabBox[0];
*   tabBox=$tabBox.get(0); //=>GET是JQ原型上提供的方法
*   供JQ实例基于索引获取到指定的JS对象

```

* \$tabBox.eq(0): 它也是基于索引获取集合中的某一项，只不过GET获取的是JS对象，EQ会把获取的结果包裹成一个新的JQ对象(JQ实例返回)

* [把原生JS->JQ]

* let tabBox=document.querySelector('.tabBox');

* \$(tabBox) 直接使用选择器把原生JS对象包裹起来，就会把JS转换为JQ对象（因为\$()就是创建JQ的一个实例）

* 分析选择器源码，我们发现SELECTOR传递的值支持三种类型

* 1.STRING : 基于选择器获取元素

* 2.元素对象 selector.nodeType: 把JS对象转换为JQ对象

* 3.函数: 把传递的函数执行，把JQ当做实参传递给函数

* selector(jQuery)

```

// $(function ($) {
//     // ==>$: 传递进来的jQuery
//
// }) ;

```

```
jQuery(function ($) {
    //=>$:私有变量,而且特定就是JQ
    $();
});
```

```
// jQuery() => {
//     //=>函数肯定会执行,但是会在当前页面中的HTML结构都加载完成后再执行
//     //=>函数执行会形成一个闭包
// };
```

```
$(function () {
    //=>写自己的代码
});
```

```
* JQ选择器的SELECTOR可以是字符串,字符串这种格式也有两种
*   1.选择器
*   2.HTML字符串拼接的结构:把拼接好的HTML字符串转换为JQ对象,然后可以基于APPEND-TO等方法追加到页面中
*/
// $('<div id="AA"></div>').appendTo(document.body);
```

方法

```
* EACH: JQ中的EACH方法是用来进行遍历的(类似于数组的FOR-EACH)
*   [可遍历内容]
*   1.数组
*   2.对象
*   3.类数组(JQ对象)
*   ...
```

```
*   [三种EACH]
*   1.给JQUERY设置的私有属性 $.each()
*   2.给实例设置的公有属性 $([selector]).each()
*   3.内置的EACH
```

```
// $.each([12, 23, 34], (index, item) => {
//     //=>参数的顺序和内置FOR-EACH相反
//     console.log(index, item);
// });
```

```
// $.each({name: 'xxx', age: 25, 0: 100}, (key, value) => {
//     //=>原理其实就是FOR-IN循环
//     console.log(key, value);
// });
```

```

$('.tabBox li').each(function (index, item) {
    //=>非箭头函数: THIS===ITEM, 当前遍历的这一项 (原生JS对象)
    //=>$(THIS) 把当前遍历的这一项转换为JQ对象
    $(this).click(function () {
        //=>给每一个遍历的LI都绑定一个点击事件
        //THIS:当前点击的LI (原生JS对象)
        $(this).css({
            color: 'red'
        });
    });
});

```

```

$('.tabBox li').click(function () {
    //=>获取的JQ集合中有三个, 我们此处相当于给三个LI都绑定了点击事件 (
    JQ在调取CLICK的时候, 会默认的对集合进行EACH遍历, 把每一项都给
    CLICK了)
});

```

```

// jQuery.noConflict(); //=>转让JQ使用$的权利
// console.log($); //=>UNDEFINED
// jQuery();

```

```

let zzz = jQuery.noConflict(true); //=>深度转让: 把jQuery这个名字
    也让出去
console.log(jQuery); //=>UNDEFINED
console.log(zzz);

```

```

//=>常用的筛选方法:
// filter: 同级筛选
// children: 子集筛选
// find: 后代筛选

```

jQajax

```

$.ajax({
    url: 'json/product.json',
    method: 'GET',
    dataType: 'json',
    async: false,
    success: function (result) {
        console.log(result);
    }
});

```

jQ实现选项卡


```

<div class="tabBox">
  <ul class="header clearfix">
    <li class="active">新闻</li>
    <li>电影</li>
    <li>音乐</li>
  </ul>
  <div class="active">时事新闻</div>
  <div>最新电影</div>
  <div>欧美音乐</div>
</div>

```

//=>当HTML结构都加载完成执行函数

```

jQuery(function ($) {
  let $tabBox = $('.tabBox'),
      $tabList = $tabBox.find('.header>li'),
      $divList = $tabBox.children('div');
  // let $tabList = $('.tabBox>.header>li'),
  //     $divList = $('.tabBox>div');

```

//=>基于JQ内置EACH机制,给每个LI都绑定了点击事件

```

$tabList.on('click', function () {
  let index = $(this).index(); //=>获取当前点击LI的索引
  $(this).addClass('active')
    .siblings().removeClass('active')
    .parent().nextAll('div')
    .eq(index).addClass('active')
    .siblings('div').removeClass('active');
});
});

```

精简版

```

jQuery(function ($) {
  $('.tabBox>.header>li').on('click', function () {
    let index = $(this).index();
    $(this).addClass('active')
      .siblings().removeClass('active')
      .parent().nextAll()
      .eq(index).addClass('active')
      .siblings('div').removeClass('active');
  });
});

```