

**窦孝晨Web3班Web前端培训笔记**

**新开普电子股份有限公司**

**2019年4月22日**

Web前端笔记

# WEB资源网站

## Web技术社区论坛：

Web技术文档：https://developer.mozilla.org/zh-CN/docs/Web

Github官网：<https://github.com/>

Web前端网站（hellogirl）：<http://www.jqhtml.com/>

大前端：http://www.daqianduan.com

伯乐在线：<http://hao.jobbole.com/>

[百度静态资源公共库](https://www.baidu.com/link?url=ddGVtL9zWXLA975ORBoxQRY4WJertS4IItAewHD7eEu92bIG_QqcXtcm9uLVGzgR&wd=&eqid=9772ead200002f650000000359e064fe" \t "https://www.baidu.com/_blank)：<http://cdn.code.baidu.com/>

CSDN社区: https://www.csdn.net/

segmentfault社区：https://segmentfault.com/

博客园：https://www.cnblogs.com/

## 脚手架-工具：

npm中文网：<https://www.npmjs.com.cn/>

npm帮助文档：https://docs.npmjs.com/

Webpack中文网（Web前端脚手架）：<https://www.webpackjs.com/>

Babel中文网（JavaScript转换器）：https://www.babeljs.cn/

ESlint官网(代码标准规范)：<https://eslint.org/>

Eslint-react规则：<https://www.npmjs.com/package/eslint-plugin-react>

DvaJS中文网（React的脚手架）：https://dvajs.com/

## JS框架官网

Node.js中文网：<http://nodejs.cn/>

jQuery插件库：http://www.jq22.com/

jQuery中文API：http://www.jq22.com/

React中文网：<http://www.css88.com/react/docs/react-api.html>

Redux中文文档：http://www.redux.org.cn/

ant-design官网：http://1x.ant.design/

ant-design（ReactPC端框架）<https://ant.design/index-cn>

ant-design-mobile（React移动端框架）<https://mobile.ant.design/index-cn>

Amaze UI React（基于react的UI框架）http://amazeui.org/react/

支付宝蚂蚁金服开放平台：<https://open.alipay.com/platform/home.htm>

react-bootstrap（react的bootstrap框架）：http://react-bootstrap.cn/

AntV （react蚂蚁数据图表）：<https://antv.alipay.com/zh-cn/index.html>

VUE 中文网：<https://cn.vuejs.org/>

Element-UI官网(Vue 2.0 PC端)：http://element-cn.eleme.io/#/zh-CN

Muse-UI（vue的PC端UI框架）：<http://www.muse-ui.org/#/index>

VUX（vue的移动端UI框架）https://vux.li/#/

Mint UI（vue的移动端UI组件库）http://mint-ui.github.io/#!/zh-cn

sChart（vue数据图表）<http://blog.gdfengshuo.com/example/sChart/>

Antd-Angular：http://ng-zorro.gitee.io/docs/introduce/zh

Echarts图表：<http://echarts.baidu.com/index.html>

SUI（淘宝的PC端UI框架）http://sui.taobao.org/sui/docs/

SUI-mobile（淘宝的移动端UI框架）：<http://m.sui.taobao.org/>

SUI3（淘宝的全栈框架）http://sui3.taobao.org/css/

微信小程序开发者文档api：

<http://mp.weixin.qq.com/debug/wxadoc/dev/api/>

微信企业号开发者文档：[http://qydev.weixin.qq.com/wiki/index.php](http://qydev.weixin.qq.com/wiki/index.php?title=%E9%A6%96%E9%A1%B5)

微信公众平台开发者文档：https://mp.weixin.qq.com/wiki/home/

微信第三方接口开发平台：<https://open.weixin.qq.com/>

## UI框架官网

Bootstrap中文网：<http://www.bootcss.com/>

Amaze-UI：http://amazeui.org/

MUI官网：<http://dev.dcloud.net.cn/mui/>

H-UI：<http://www.h-ui.net/index.shtml>

layUI：<http://www.layui.com/>

Layer（layUI弹出层）：http://layer.layui.com/

Swiper中文网：<http://www.swiper.com.cn/>

## 其他公共开放平台

WordPress（快速搭建网站）：<https://cn.wordpress.org/>

百度地图AIP：<http://api.map.baidu.com/lbsapi/creatmap/>

MongoDb中文网：<http://www.mongodb.org.cn/>

Sonatype官网（Nexus服务器）：<https://help.sonatype.com/docs>

# 干货

WEB安全色：http://card.qdsay.com/color/safe

WEB颜色对照表：<http://www.114la.com/other/rgb.htm>

图片压缩网址：<https://tinypng.com/>

贴图库网站：<http://www.tietuku.com/>

七牛云：https://www.qiniu.com/

二维码生成链接：“http://qr.liantu.com/api.php?text=”+字符串

七牛云uptoken生成网址：

http://jsfiddle.net/gh/get/extjs/4.2/icattlecoder/jsfiddle/tree/master/uptoken

一览芳华（个人网站）：<http://www.51283.net/>

Js操作本地文件夹：<http://www.jb51.net/article/48538.htm>

福利(手机UC浏览器或手机微博在线观看)：http://m.guanren15.com

表情包在线生成器https://fabiaoqing.com/

种子链接：

<https://github.com/Chion82/hello-old-driver/blob/master/archives/magnet_output-2017-08-31-000003(迅雷下载)>

# 本机ip地址、localhost与127.0.0.1

|  |  |  |  |
| --- | --- | --- | --- |
|  | localhost | 127.0.0.1 | 本机IP (192.168.xxx.xxx) |
| 是否联网 | 不联网 | 联网 | 联网 |
| 传输途径 | 不使用网卡，只是内部端口使用，不受防火墙和网卡限制 | 网卡传输，受防火墙和网卡限制 | 网卡传输 ，受防火墙和网卡限制 |
| 应用环境 | 只限于本机内部端口访问 | 只限于本机内部端口访问 | 本机或外部(局域网最常用)访问 |

# 项目配置解决方案

## PHPstudy配置前端页面域名步骤

1.在PHPstudy安装目录中的WWW文件夹下新建一个项目文件夹。

2.将react编译后的文件夹（dist）放到项目文件夹里。

3.打开PHPstudy——>点击MySQL管理器——>站点域名管理——>网站目录，然后选择目录，找到并选中步骤1的文件夹，点击确定之后在网站端口里配置端口号，然后点击新增。4.点击保存并生成配置文件，最后重启PHPstudy。

# 阿里鄙视题

1.写一个有效的算法完成矩阵搜索，这个矩阵有如下特点：

1) 矩阵中的每行数字都是经过排序的，从左到右依次变大。

2) 每行的第一个数字都比上一行的最后一个数字大

例如：

[

[2, 4, 8, 9],

[10, 13, 15, 21],

[23, 31, 33, 51]

]

实现一个函数，搜索这个数组

输入：4，返回：true

输入：3，返回：false

要求：这道题要求性能优化，如果只是简单的写，还不能达到最低标准。数组一般用for of 提示：用二分法搜索， 40分钟内完成。

1. 有一个数组：  
   const imgs = ['url1', 'url2', 'url3', ...];  
   请实现效果：  
   按照图片数组顺序队列加载图片（注：加载完一张再加载下一张）

请用SE6的promise或者ES7的async await

# Utils工具函数

## 1.ajax的封装

Ajax.js:

export var APIhost=”后台接口域名”

export function ajax(options) {

options = options || {};

options.type = (options.type || "GET").toUpperCase();

options.dataType = options.dataType || "jsonp";

options.jsonpCallback= 'callback';

var params = formatParams(options.data);

var xhr;

//创建 - 第一步

if (window.XMLHttpRequest) {

xhr = new XMLHttpRequest();

}

//连接 和 发送 - 第二步

if (options.type == "GET") {

xhr.open("GET", `${options.url}?${Getparams}`, true);

xhr.send(null);

} else if (options.type == "POST") {

xhr.open("POST", options.url, true);

//设置表单提交时的内容类型

xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

xhr.send(params);

}

//接收 - 第三步

xhr.onreadystatechange = function () {

if (xhr.readyState == 4) {

var status = xhr.status;

if (status >= 200 && status < 300 || status == 304) {

options.success && options.success(xhr.responseText, xhr.responseXML);

} else {

options.error && options.error(status);

}

}

}

}

//格式化参数

export function formatParams(data) {

var arr = [];

for (var name in data) {

arr.push(encodeURIComponent(name) + "=" + encodeURIComponent(data[name]));

}

arr.push(("v=" + Math.random()).replace("."));

return arr.join("&");

}

用法：

import \* as reajax from "../utils/ajax";

Let sbdata={

...//提交的data

}

reajax.ajax({

url: reajax.APIhost+"/admin/member/card/getAllUserList", //请求地址

type: "POST", //请求方式

data: sbdata, //请求参数

dataType: 'jsonp',

success: function (resStr) { // 此处放成功后执行的代码

var res=JSON.parse(resStr);

console.log(res);

},

error: function (res) { // 此处放失败后执行的代码

console.log(res);

}

});

## 2.格式化二维数组数组，生成JSON格式的TreeData

### 源数据格式：

const data=[

//s1线

[1,'a',11,'a1',111,'a11'],

[1,'a',11,'a1',112,'a12'],

[1,'a',12,'a2',121,'a21'],

[1,'a',12,'a2',122,'a22'],

[1,'a',12,'a2',123,'a23'],

//s2线

[2,'b',21,'b1',211,'b11'],

[2,'b',21,'b1',212,'b12'],

[2,'b',21,'b1',213,'b13'],

[2,'b',22,'b2',221,'b21'],

[2,'b',22,'b2',222,'b22'],

];

console.log(data);

### 格式化函数：

export function FormatTreeData(data=[]){

var lines = {};

for(var i in data){

var linename = data[i][1];

var line = lines[linename];

if(line===undefined){

lines[linename]={'key':data[i][0],'title':data[i][1]};

}

var stations = lines[linename].stations;

if(stations===undefined){

lines[linename].stations={};

}

if(lines[linename].stations[data[i][3]]===undefined){

lines[linename].stations[data[i][3]]={'key':data[i][2],'title':data[i][3]};

}

var devices = lines[linename].stations[data[i][3]].devices;

if(devices===undefined){

lines[linename].stations[data[i][3]].devices=[];

}

lines[linename].stations[data[i][3]].devices.push({'key':data[i][4],'title':data[i][5]});

}

// console.log(lines);

var treeData = []

for(var line in lines){

var lineData = {};

lineData.title=lines[line].title;

lineData.key=lines[line].key;

lineData.children=[];

for(var station in lines[line].stations){

var stationData = {};

stationData.title=lines[line].stations[station].title;

stationData.key=lines[line].stations[station].key;

stationData.children=lines[line].stations[station].devices;

lineData.children.push(stationData);

}

treeData.push(lineData);

}

console.log(treeData);

return treeData;

}

调用：

import FormatTreeDatafrom “../utils/FormatTreeData”

FormatTreeData(data);

# 什么是html

HTML（Hyper Text Mark-up Language ）即[超文本标记语言](https://www.baidu.com/s?wd=%E8%B6%85%E6%96%87%E6%9C%AC%E6%A0%87%E8%AE%B0%E8%AF%AD%E8%A8%80&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Yzn17BPWF-mHfLnyu9nAub0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En1D1P1cd" \t "https://zhidao.baidu.com/question/_blank)，是 WWW 的描述语言，由 Tim Berners-lee提出，它定义了浏览器页面的骨架。

主要特点如下：

简易性：超级文本标记语言[版本升级](http://baike.baidu.com/view/421718.htm" \t "http://baike.baidu.com/_blank)采用[超集](http://baike.baidu.com/view/760124.htm" \t "http://baike.baidu.com/_blank)方式，从而更加灵活方便。

[可扩展性](http://baike.baidu.com/view/476360.htm" \t "http://baike.baidu.com/_blank)：超级文本标记语言的广泛应用带来了加强功能，增加[标识符](http://baike.baidu.com/view/390932.htm" \t "http://baike.baidu.com/_blank)等要求，超级文本标记语言采取子类元素的方式，为系统扩展带来保证。

平台无关性：虽然[个人计算机](http://baike.baidu.com/view/920814.htm" \t "http://baike.baidu.com/_blank)大行其道，但使用[MAC](http://baike.baidu.com/view/32702.htm" \t "http://baike.baidu.com/_blank)等其他机器的大有人在，超级文本标记语言可以使用在广泛的平台上，这也是[万维网](http://baike.baidu.com/view/7833.htm" \t "http://baike.baidu.com/_blank)（[WWW](http://baike.baidu.com/view/1453.htm" \t "http://baike.baidu.com/_blank)）盛行的另一个原因。

通用性：另外，[HTML](http://baike.baidu.com/view/692.htm" \t "http://baike.baidu.com/_blank)是网络的通用语言,一种简单、通用的全置标记语言。它允许网页制作人建立文本与图片相结合的复杂页面，这些页面可以被网上任何其他人浏览到，无论使用的是什么类型的电脑或浏览器。

什么是css：

层叠样式表，是一种用来表现[HTML](http://baike.baidu.com/view/692.htm" \t "http://baike.baidu.com/_blank)（[标准通用标记语言](http://baike.baidu.com/view/5286041.htm" \t "http://baike.baidu.com/_blank)的一个应用）或[XML](http://baike.baidu.com/view/63.htm" \t "http://baike.baidu.com/_blank)（标准通用标记语言的一个子集）等文件样式的计算机语言。CSS不仅可以静态地修饰网页，还可以配合各种脚本语言动态地对网页各元素进行格式化。

其主要特点如下：

简易性：超级文本标记语言[版本升级](http://baike.baidu.com/view/421718.htm" \t "http://baike.baidu.com/_blank)采用[超集](http://baike.baidu.com/view/760124.htm" \t "http://baike.baidu.com/_blank)方式，从而更加灵活方便。

[可扩展性](http://baike.baidu.com/view/476360.htm" \t "http://baike.baidu.com/_blank)：超级文本标记语言的广泛应用带来了加强功能，增加[标识符](http://baike.baidu.com/view/390932.htm" \t "http://baike.baidu.com/_blank)等要求，超级文本标记语言采取子类元素的方式，为系统扩展带来保证。

平台无关性：虽然[个人计算机](http://baike.baidu.com/view/920814.htm" \t "http://baike.baidu.com/_blank)大行其道，但使用[MAC](http://baike.baidu.com/view/32702.htm" \t "http://baike.baidu.com/_blank)等其他机器的大有人在，超级文本标记语言可以使用在广泛的平台上，这也是[万维网](http://baike.baidu.com/view/7833.htm" \t "http://baike.baidu.com/_blank)（[WWW](http://baike.baidu.com/view/1453.htm" \t "http://baike.baidu.com/_blank)）盛行的另一个原因。

通用性：另外，[HTML](http://baike.baidu.com/view/692.htm" \t "http://baike.baidu.com/_blank)是网络的通用语言,一种简单、通用的全置标记语言。它允许网页制作人建立文本与图片相结合的复杂页面，这些页面可以被网上任何其他人浏览到，无论使用的是什么类型的电脑或浏览器。

什么是javascript：

avaScript一种直译式脚本语言，是一种动态类型、弱类型、基于原型的语言，内置支持类型。它的解释器被称为JavaScript引擎，为-浏览器的一部分，广泛用于客户端的脚本语言，最早是在HTML（标准通用标记语言下的一个应用）网页上使用，用来给HTML网页增加动态功能。

特点：

(1)脚本语言。JavaScript是一种解释型的脚本语言,C、C++等语言先编译后执行,而JavaScript是在程序的运行过程中逐行进行解释。

(2)基于对象。JavaScript是一种基于对象的脚本语言,它不仅可以创建对象,也能使用现有的对象。

(3)简单。JavaScript语言中采用的是弱类型的变量类型,对使用的数据类型未做出严格的要求,是基于Java基本语句和控制的脚本语言,其设计简单紧凑。

(4)动态性。JavaScript是一种采用事件驱动的脚本语言,它不需要经过Web服务器就可以对用户的输入做出响应。在访问一个网页时,鼠标在网页中进行鼠标点击或上下移、窗口移动等操作JavaScript都可直接对这些事件给出相应的响应。

(5)跨平台性。JavaScript脚本语言不依赖于操作系统,仅需要浏览器的支持。因此一个JavaScript脚本在编写后可以带到任意机器上使用,前提上机器上的浏览器支持JavaScript脚本语言,目前JavaScript已被大多数的浏览器所支持。

日常用途：

嵌入动态文本于HTML页面。

对浏览器事件做出响应。

读写HTML元素。

在数据被提交到服务器之前验证数据。检测用户的浏览器信息。

与用户进行数据交互。

控制cookies，本地存储，包括创建和修改等。

基于Node.js技术进行服务器端编程。

要是通过html标记对网页中的文本，图片，声音等内容进行描述

2：HTML之所以称为超文本标记语言，不仅是因为他通过标记描述网页内容，同时也由于文本中包含了所谓的“超级链接”，通过超链接可以实现网页的跳转。从而构成了丰富多彩的Web页面。

<title>标记用于。一个HTML文档只能含有一对<title></title>标记，<title></title>之间的内容定义HTML页面的标题，即给网页取一个名字，必须位于<head>标记之内将显示在浏览器窗口的标题栏中。其基本语法格式如下：

<title>网页名称</title>

<body>标记用于定义HTML文档所要显示的内容，也称为主体标记。浏览器中显示的所有文本、图像、音频和视频等信息都必须位于<body>标记内，<body>标记中的信息才是最终展示给用户看的。

一个HTML文档只能含有一对<body>标记，且<body>标记必须在<html>标记内，位于<head>头部标记之后.

双标记也称体标记，是指由开始和结束两个标记符组成的标记。其基本语法格式如下：

<标记名></标记名>

该语法中“<标记名>”表示该标记的作用开始，一般称为“开始标记（start tag）”，“</标记名>” 表示该标记的作用结束，一般称为“结束标记（end tag）”。和开始标记相比，结束标记只是在前面加了一个关闭符“/”。

单标记也称空标记，是指用一个标记符号即可完整地描述某个功能的标记。其基本语法格式如下

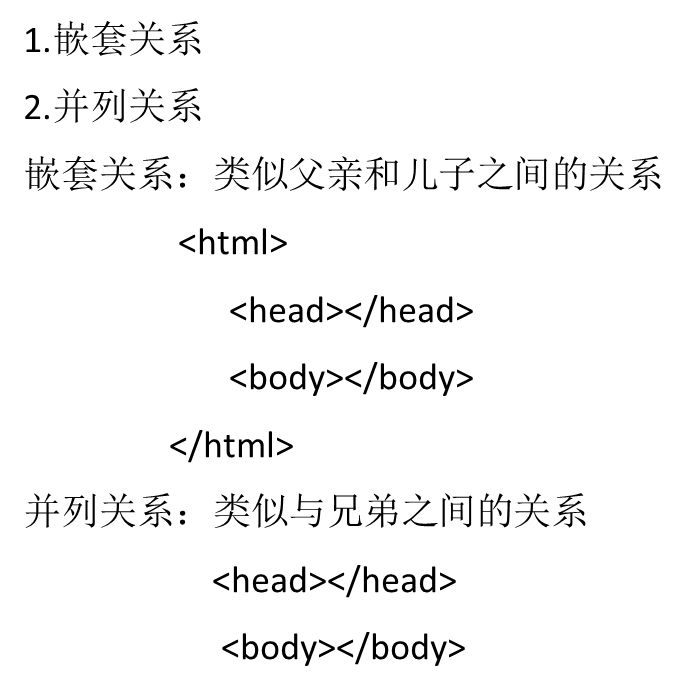
<标记名/>

在HTML中还有一种特殊的标记——注释标记。如果需要在HTML文档中添加一些便于阅读和理解但又不需要显示在页面中的注释文字，就需要使用注释标记。其基本语法格式如下：

<!--注释语句-->

注释内容不会显示在浏览器窗口中，但是作为HTML文档内容的一部分，也会被下载到用户的计算机上，查看源代码时就可以看到。

在html中，如果希望某段文本强制换行显示，就需要使用换行标记<br />，如果在控制台或弹框内显示换行，那么就是”\n”.



为此HTML提供了文本样式标记<font>，

|  |  |  |
| --- | --- | --- |
| 特殊字符 | 描述 | 字符的代码 |
|  | 空格符 | &nbsp; |
| < | 小于号 | &lt; |
| > | 大于号 | &gt; |
| & | 和号 | &amp; |
| ￥ | 人民币 | &yen; |
| © | 版权 | &copy; |
| ® | 注册商标 | &reg; |
| ° | 摄氏度 | &deg; |
| ± | 正负号 | &plusmn; |
| × | 乘号 | &times; |
| ÷ | 除号 | &divide; |
| ² | 平方2（上标2） | &sup2; |
| ³ | 立方3（上标3） | &sup3; |

有序列表即为有排列顺序的列表，其各个列表项按照一定的顺序排列定义，有序列表的基本语法格式如下：

<ol >

<li >项一</li>

<li >项二</li>

</ol>

一：主流开发：B/S（browser/server）browser浏览器

server服务器。

当你输入一个网站时，浏览器向服务器发送一个请求：http而服务器将反馈信息反馈给浏览器，浏览器在屏幕上显示。

C/S(Client/Server )Client 客户端

Server服务器

Hybird 用一个客户端来封装Brower的页面。混合开发

二：应届生选择什么技术就业：

Web前端/Unity3D>>JAVA/PHP/NET>>云计算、大数据、UL/单片机/嵌入式、UI、网络营销

三：三门技术组成了前端开发

1：HTML Hyper TextMarkUp language(超文本标记语言)描述网页的语义。

2：CSS Cascading Sheet Style（层叠样式列表）描述网页的样子

3：JavaScript（JS）

四：网页制作

Spading= 0；

Margin=0；‘

清除页面中原有的样式

Css中能被继承的属性：

1.Text-

2.Font-

3.Line-

4.Color-

这四种在嵌套从属关系中可以继承，即通用。除了这4个属性，其他的都不能嵌套包含中通用。

# 页面标签

<!DOCTYPE html>

声明必须是 HTML 文档的第一行，位于 <html> 标签之前。

<!DOCTYPE> 声明不是 HTML标签；它是指示 web 浏览器关于页面使用哪个 HTML 版本进行编写的指令。定义了页面版本里的编写指令，标记文本语言。

Doctype有三种指令：严格，过渡，框架。

<html> </html> 标签

限定了文档的开始点和结束点，在它们之间是文档的头部和主体。声明了html文档，有三个主要组成部分：head，title和body。

<meta>

提供有关页面的元信息（meta-information），比如针对搜索引擎和更新频度的描述和关键词，指定网页的编码格式，设置文档类型以及设置网页编写所用的语言。

# 块级标签

1．<h1>………</h1>h: headline描述页面标题的语义。网页标题一共有六级。分别是<h1>………</h1>~<h6>…………</h6>

2．<p></p> paragraph 描述页面段落的语义，自带上下行间距。

3．&nbsp;空格，空白折叠现象，即多个空格产生一个空格。

5．<span></span>里面专门用来写入 行元素，即“文本“的。这里的”“文本包括：文字，图片，表单元素，超链接

</div>块元素，换行 solid:实线

<input></input>标签：点击输入标签，根据其type属性不同，它很特殊，可以设置宽高和背景颜色，可分为button（按钮），checkbox（单选框），text（文字输入框）等样式，并且可以设置宽高和背景颜色。

浏览器（搜狐）在input标签内默认值padding值为1，border为2px；所以实际宽、高比设置值大了（2+1）\*2=6像素。

<label></label>标签为input元素标注，它的for属性可将label标签绑定到input等表单元素中

for属性：值为element Id，Id值。

<textarea></textarea>:文本输入框标签，用来设置一个文本输入框。

<select>有多个选项可供选择。

<option></option>

</select>下拉文本框（选择文本，如北京，上海，郑州等）

6．列表

<ul>unorderlist无序列表

<li>列表项目内容</li>

</ul>

<dl>自定义列表

<ol>有序列表

7.文本标签：用来控制网页中文本的字体、字号和颜色。其基本语法格式如下：

<font>文本内容</font>

List-tstyle：列表元素前的黑点，由于网页默认list元素前的黑点，所以通常用list-style：none来清除超链接列表元素前面的黑点。

text-decoration: none;清除超链接列表元素的下划线。

Link：网页重定向，即打开新的网页。

Ctrl+/：文本注释标签。

<hr>横线标签。

<fieldset>将表单内相关元素分组

<legend></legend>

</fieldset>

<marquee></marquee>

Marquee是个块级标签，标记在可浏览区域中滚动文本。它的公共属性可在style标签里设置。

重要的全局属性:

1. 滚动方向direction

（包括4个值：up、 down、 left和 right）

　　语法：<marquee direction="滚动方向">...</marquee>

1. 滚动方式behavior

（scroll:循环滚动，默认效果； slide:只滚动一次就停止； alternate:来回交替进行滚动）

　　语法：<marquee behavior="滚动方式">...</marquee>

1. 滚动速度scrollamount

（滚动速度是设置每次滚动时移动的长度，以像素为单位）

　　语法：<marquee scrollamount="5">...</marquee>

1. 滚动延迟scrolldelay

（设置滚动的时间间隔，单位是毫秒）

　　语法：<marquee scrolldelay="100">...</marquee>

1. 滚动循环loop

（默认值是-1，滚动会不断的循环下去）

　　语法：<marquee loop="2">...</marquee>

6.滚动范围width、height

7.滚动背景颜色bgcolor

8.空白空间hspace、vspace

五：网页制作与前端工程师的区别

HTML+CCS。网络三剑客Photoshop Fireworks DreamWheaver

Subline Webstorm

Visual Studio Core

六：table表格的制作

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

# Form表单元素

<form action=”” methd=”” name=“”>

</form>

有了表单元素，html文本才能上传到服务器。

常见表单元素：

input，a，form，select（下拉表），button，datalist。

<input>输入框，根据其type属性不同，可以变化多种形态（包括HTML5新属性）

button：按钮，

textarea：文本框

checkbox：选中按钮

tel：电话号码框

Email：邮件

Search：搜索框，与datalist 配合使用

Submit：提交按钮

file：上传按钮，点击打开计算机本地文件

range：控制条（控制时间，音量，进度等），max：最大长度，min：最小长度

Number：数字框

Color：调色板，看颜色

datetime-local：本地时间

time：时间日历

placeholder

autocomplete显示完全提示

autofocus：指定自动获取焦点

Sp额、llcheck 可对带有文本内容输入框和textarea控件设置。设完之后，类似于word文档里的错误拼写检查。

List与datalist：可为用户设置选值列表，与ooption配合使用。

Step：步伐，步骤，表示range控件每次走的步数

valueAsNumber属性

完成控件值在文本与数之间相互转换。

required 必填项

1. disabled  --  完全不可编辑，并且是不能复制
2. readonly-- 不可编辑，但是可以复制。

3.Readonly只针对input(text / password)textarea，而对于所有的表单元素都有效，包括select, radio, checkbox, button等。

4表单元素在使用了disabled后，当我们将表单以POST或GET的方式提交的话，这个元素的值不会被传递出去，而readonly会将该值传递去

<textarea>文本输入框

<select>下拉列表

<option></option>

<option></option>

<option></option>

<option></option>

</select>

select的mutiple属性：规定了有多个选项可以同时选择。

<datalist>元素规定预定义选项列表

文字格式化标签

文字标题：

<h1></h1>---<h6></h6>

段落：

<p>这是一段话</p>

<hr>段落间隔符，用于新闻日期和内容之间的间隔。

<font>:文本编辑器，用法：

<font size=”10” color=”erd”>

一段优美的文字

</font>

文字加粗标签：

<strong>**该减肥了**</strong>或<b >**该减肥了**</b>

文字斜体标签：

<em>*文字斜体*</em>或<i>*文字斜体*</i>

删除线标签：

<del>~~删除线~~</del>或<s>~~删除线~~</s>

文字下划线标签：

<ins>文字下划线</ins>或<u>文字下划线</u>

<span> span意思是跨度，测量；行内元素。

<textarea></textarea>文本输入框标签。

# 表格定义标签：

<table>属性有：

<tr>…</tr>行

<Td>…</td>列，单元格

<caption><h1>人员统计表</h1></caption>表格标题标签

Border=““：边框 soild实线 color颜色

Wideth=““：表格宽度

Heght=““：表格高度

Cellspacing=“0“：单元格间距（一盘单元格间距为0，即Cellspacing=“0“）

Cellpadding=’’’’：文字距单元格边框距离

Bgcolor=’’’’:背景颜色

Align=”center/right/left”居中/居左/居右。如果把它放到单元格标签<td>中，那么是单元格里的文字居中；如果将它放到表格标签<table>中，那么表格居中。

<th></th>:设置表格单元格标题，将哪一行设成标题，就放在哪一行的<tr>..</tr>里面，代替<td></td>。注意：表格类型<table>的标签与标题类型<h>的标签不能互相嵌套。

表格结构：

<table>

<thead></thead>

<tbody></thead>

<tfoot></thead>

</table>

表格单元格的合并：

横向合并（列合并）<td colspan=”3”>(有几就是几，写到第几个单元格就是从第几个单元格开始,然后把后面的单元格<td>删除)

纵向合并（行合并）<td rowspan=”3”>(有几就是几，写到第几个单元格就是从第几个单元格开始, 然后把后面的单元格<td>删除)

<Label for>关联标签

在我们的浏览器向服务器请求有两种方法：

1：get 特点是用户输入的信息会在地址栏上显示。

2：Post 特点是不会在地址栏中传送你输入的信息。

八HTML标签分为两大阵营

1：容器级标签：

<h1></h1> <p></p><ul></ul><ol></ol><dl></dl><dd><dt><form>

2:文本级标签 <span></span><img src=””><a><b>加粗</b>

<i>倾斜</i>

<Img sr=””>img:图片标签，行内元素。

Src属性：链接地址，可以是网页地址或本地地址。

Onload属性：onload表示加载好，换言之，没有加载好不会执行；

Complete属性：判断img是否加载完成，返回布尔值。True：加载完成，false：未加载完成。

第七天 CCS标准文档流与浮动

复习块级元素与行内元素的互相转换

哪些是块级元素？pul，ol，di，dt，dd，div，h1~h6系列。

哪些是行内元素？span、a、img、u、l、em、del、b。

他们的区别在于能否在一行内显示。

标准文档流与脱离文档流

从上而下地执行顺序称为标准文档流（织毛衣）

无规则地执行顺序称为非标准文档流。

一定要掌握的两个概念：

display：Block ；display ：inline。Display：none；

Display：inline-block：既可以像行内元素那样左右排列，又可以像块级元素那样设置宽高。

Display是显示模式，用来改变元素行为，块级性质。

Inline是“行内”。

一旦给一个标签加入display：inline；

那么这个元素就变成了行内元素。此时它和一个span无异。

此时这个div不能设置宽和高，并且能和行内元素并排。

同理：

display：inline；

“block”是“块”的意思。

让标签变成块级元素，此时一个行内元素变成了一个会计元素。

这个span能设置高和宽，这个span就占一行，别人无法与他并排，不能设置宽度，将撑满父亲。

标准流里面限制很多，标签的性质异常恶心，比如我们要并排，并且要设置高。

# Float：浮动。

由于标准文本流中，两个块级元素各自霸占一行，不能并排成一行，因此浮动的目的就是将两个块级元素并排成行。

float：right/left

可以带着自己的边框，颜色，边距而并排成行，已经脱离了文档流而不用display。

浮动是我们用CSS布局最长使用的技术。必须掌握。

并且两个元素并排了，并且可设置高和宽（这在刚才的比爱哦准文本中是不可想象的，标准文档里面不可设置宽和高）。

浮动元素不区分行内和块状元素。

饭是浮动的元素，脱离了标准流，都可以设置宽和高。

不需要加上display：none/display：block。

浮动元素的性质：

盒子塌陷现象：档浮动的元素宽度超过了父亲元素的宽度，那么它就会往下找容纳自己的宽度，如果找到，那么紧贴着找到的元素。如果没找到距离可以存放自己，那么继续往下走，这种现象会导致布局混乱。

浮动有一个“字围”效果。

Div挡住了<p>,但是<p>中的文字不会被挡住，形成“字围”效果。

关于浮动我们要记住一点，我们在初期一定要遵循把握一个原则。

永远不要一个东西单独浮动，要浮动一起浮动，否则都不浮动；即浮动是一起的。“字围”效果是个特别案列。

浮动的元素不能撑起盒子的高度，也就是说浮动只影响宽度，不影响高度。

# 清除浮动的4个方法：

清除浮动方法1：给浮动的元素的祖先元素添加高度。

如果一个元素要浮动，那么它的祖先元素一定要有高度。有高度的盒子，才能锁住浮动。

清除浮动的方法2:clear:both；

网址制作中，高度height很少出现，为什么，因为很所时候都是被内容撑起了高！那也就是说，在盒子中写上高度几乎不可见。

能不能写height，也把浮动清除了呢，也让浮动之间，互不影响。

Clear:both：就是清除，Both指的是左右，意思就是说清除别人对我的影响。这个方法有个致命的缺点，就是因为没有高，所以margin也失效了。

清除浮动方法3:隔墙法

清除浮动元素的方法4：内墙法

王炸：不让溢出：overflow：hidden。

Overflow是“溢出”的意思，hidden就是隐藏的意思。

所有溢出边框的内容，都要隐藏掉。

总结：清除浮动总结

（1）加高法

浮动的元素，只能被有高度的元素的盒子锁住，也就是说盒子内部有浮动，这个盒子有高度，那么妥妥的，浮动不会受到影响，但是，工作上，我们绝不会给盒子加高度，这是因为麻烦，并且不能自适应页面的快速变化。

1. clear:both

最简单的清除浮动的方法，就是给盒子增加clear:both。表示自己的内部元素，不受其他盒子的影响。

1. 隔墙法 在两部门盒子之间，建一堵墙，隔开两部分的浮动，让浮动的元素，不去追前面浮动的元素。
2. 内墙法 在第一个盒子后面加上一堵墙，这堵墙清除了

浮动，然后呢，还可以加上高度隔开与后面的距离。

1. Overflow:hidden 这个属性的本意，就是将溢出盒子的内容，隐藏掉，但是，我们发现这个东西能够用于浮动的清除。我们知道，一个没有高度（不设置高度）的父亲元素，不能被浮动的儿子撑起高度，但是如果这个父亲加上了overflow:hidden；那么这个父亲就能够被浮动的儿子撑起高度了，这个现象，不能解释，这是浏览器的小偏方。并且它可以让margin生效。
2. overflow：auto；清除浮动/溢出并产生拖动框
3. <ul>
4. <li>
5. <span class="title">哈哈哈哈</span>
6. <span class="date"> 2015-10-01</span>
7. </li>
8. <li>
9. <span class="title">哈哈哈哈</span>
10. <span class="date"> 2015-10-01</span>
11. </li>
12. <li>
13. <span class="title">哈哈哈哈</span>
14. <span class="date"> 2015-10-01</span>
15. </li>
16. <ol>
17. </ul>
18. IE6浏览器不支持12px以下的盒子（块元素），即任何小于12px的盒子，IE6中看起来都。

解决方案：很简单，就是将盒子的字号，设置成小于盒子的高度。比如0px；

Height：10px；

-font-size：0px；

现在我们介绍以下浏览器的一个机制hack就是“黑客“的意思，就是浏览器的后门，针对浏览器做兼容。IE6就留了一个后门，加上想要兼容的IE6浏览器的CSS属性，加上下划线”-“。

例如：-background color

第二，IE6不支持overflow:hidden清除浮动

解决办法：以毒攻毒。追加一条

-Zoom：1

完整写法：overflow:hidden

-Zoom:能够触发浏览器的layout机制，不要深究了。因为IE6有。我们不需要了解这个机制，有兴趣的同学回去搜一下。

# HTML与CSS语法扩展

第一 复习昨天的知识

浮动的目的是让所有的HTML标签脱离文档流。

标准文档流指的是浏览器从上至下按照顺序执行标签脱离文档流是指让浏览器同时读取块级元素和行内元素。

块级元素：占据一行。

行内元素是只占据自己的位置，可与其他元素并排。行内元素不许有宽和高，其宽高是文字的宽高display：online块内元素转成行内元素。Display：block行内元素转为块级元素。

## 外边框，轮廓与阴影

Border三要素：宽度，虚实，颜色。

宽度：上不封顶，无负值。

虚实：solid：实线；dashed：虚线。

Border：1px solid green

border：transparent。设置透明边框。

border-radius：向div元素添加圆角边框

Border-top/bottom/left/right:

Border-top-left:

Border-top-right

Border-radius：边框圆角：左上角，右上角，右下角，左下角。

border-radius：a px：边框圆角半径为a px；

百分数：圆角半径占宽/高的百分比；

圆点：宽高相等，并且border-radius：50%；

box-shadow:向块盒子设置阴影区域。不算作盒子本身的宽高。

box-shadow:横向偏移量 纵向偏移量 模糊范围 四周扩展范围 阴影颜色。

从左到右依次为：横向阴影位移（右正左负） 纵向阴影位移（下正上负）阴影模糊值 值越大模糊范围越广（正值延伸，负值内延伸） 阴影边框 阴影颜色；

扩展范围：必须使得纵横偏移量为0，值越大扩展范围越广。

文本阴影：text-shadow：横向偏移量（必选值）、纵向偏移量（必选值）、模糊范围（可选值）。

box-sizing: content-box|border-box|inherit;

border-box：为元素设定的宽度和高度决定了元素的边框盒。

就是说，为元素指定的任何内边距和边框都将在已设定的宽度和高度内进行绘制。通过从已设定的宽度和高度分别减去边框和内边距才能得到内容的宽度和高度。

outline-style（轮廓）：

outline（轮廓）是绘制于元素周围的一条线，位于边框边缘的外围，用于设置元素的整个轮廓的样式,可起到突出元素的作用。如果设置了none，则元素轮廓不会展现。

请始终在 outline-color 属性之前声明 outline-style 属性。元素只有获得轮廓以后才能改变其轮廓的颜色。轮廓线不会占据空间，也不一定是矩形。

None 默认。定义无轮廓。

Dotted 定义点状的轮廓。

Dashed 定义虚线轮廓。

Solid 定义实线轮廓。

Double 定义双线轮廓。双线的宽度等同于 outline-width 的值。

Groove 定义 3D 凹槽轮廓。此效果取决于 outline-color 值。

Ridge 定义 3D 凸槽轮廓。此效果取决于 outline-color 值。

Inset 定义 3D 凹边轮廓。此效果取决于 outline-color 值。

Outset 定义 3D 凸边轮廓。此效果取决于 outline-color 值。

Inherit 规定应该从父元素继承轮廓样式的设置。

## Overflow特殊用法：

overflow: hidden; /\*自动隐藏文字\*/

text-overflow: ellipsis;/\*文字隐藏后添加省略号\*/

white-space: nowrap;/\*强制不换行，不允许出现半汉字截断\*/

蓝色字体部分，二者配合，可以形成文本超出部分用省略号代替。

注意：这种方式只能在一行文本内适用，两行或以上的文本就不适用了。

（2）浮动是让标签脱离文档流

Div：分块盒子元素，其属性有：

<div Class =”div1”></div>部分等级

<ul>

<li></li>列表元素（文本文件）

</ul>无顺序列表

<ol>

<li></li>

</ol>

## Padding和margin的用法

Padding：衬料，衬垫。在CSS中它的意思是内边框到内元素的增加距离。系统body默认padding值=8px。

Padding：100px；上下左右都增加100px。即width增加了200px，height增加了200px

Padding right：100px；右内边框距离子元素增加100px。

Padding left：100px； 左内边框距离子元素增加100px。

在web制作工程里边，通常情况下，div的大小是不能改变的，所以加了padding之后相当于div的内边距和子元素之间距离增加了padding值的距离，所以div盒子要减高减宽。

一个盒子的真实宽高=width+padding+border

Margin：外边距，加边于。在css中它的意思是自己的位置外边框到兄弟元素的增加距离。系统默认margin值= px。Margin：边缘，距离，范围。

margin：100px；上下左右都增加100px。

Class：取名，命名属性。

margin top：距上个盒子

Margin bottom：距下个盒子。

Margin left：距左边的盒子。

Margin right：距右边的盒子。

Margin对定位了的元素仍然生效，但是参照点是元素定位之后的位置。

对于两个标准流上下排列div盒子的margin值来说，谁的值大谁说了算，即取较大值。这种现象叫做margin塌陷现象。由于标准文档流的盒子不能并排出现，所以没有横向塌陷的现象

注意：脱离了标准文档流的浮动的两个元素的margin值相加。

Div：分块，既可以包含嵌套，又可以同级。

## 居中

1. 盒子居中：margin：0 auto

Margin的值可取auto表示自动。（横向自动）

简写形式是margin：0 auto

注意：使用margin：0 auto的盒子<div>居中，必须有width，否则霸一行了

具有标准流的盒子才能使用margin：0 auto居中,也即是说当一个盒子浮动起来或者定位了，脱离了标准文档流，都不能使用margin：0 auto，但可以使用margin-left或者margin-right。

margin：0 auto 是居中盒子，不是居中文本。

(C) vertical-align: middle：使图片与文字上下居中

文本字体的居中使用text-align：“center“

普及一下text-align用法：

text-align：“left“（没卵用，因为文本默认是居左）

text-align：“center“（居中）

text-align：“rihgt“（居右）

在flex布局里：

justify-content: center;/\*里边的元素水平居中\*/

align-items: center; /\*里边的元素垂直居中\*/

text-indent：首行缩进，单位是em（中文字符）。

text-indent：2em代表着段落里面移动2个中文字符。

(4)善用于父亲的padding，而不是儿子的margin，这是因为儿子的margin属性本质上是描述同级兄弟之间的距离，而不是儿子与父亲（祖先）之间或者是上下包含嵌套之间的距离，所以嵌套上下级关系的距离属性我们首选padding，而兄弟同级之间的距离我们首选margin。

IE6的margin兼容问题：

IE6的bug，当连续出现浮动的元素，携带和浮动方向相同的margin时，队首的元素会出现双倍margin。

解决方案：

1. 使浮动的方向和margin的方向相反
2. 使用hack-margin-left：10px；

行高（line-height）

顾名思义，行高就是文本高度。他可以被继承，也就是说给父亲盒子设置行高之后，文本占据的高也会变成行高，但行高不能撑开父亲盒子的高度，如果行高大于父亲盒子的高度，那么文本就会跑到父盒子外面去。

Css中，所有的行都有行高，盒子模型的padding，绝对不是直接应用在文字上的，而是应用在行上的。

为了保证字在行内垂直居中，我们的设计工程师有一个约定，行高，字号一般都是偶数，这样，他们的商就是偶数，能被2整除。

单行文本垂直居中：

Line-height：”100px(这里的行高等于盒子高度)

文本在行内垂直居中

公式：行高=盒子高

需要注意的是这个小技巧，行高=盒子高只适用于单行文本，不适用于多行文本。

多行文本垂直居中

加padding减高（详见第一天笔记）

文字水平居中：text-algin:canter;

## font家族

使用font属性，能够修饰文字。文字的字号，行高，字体能够一齐设置

font-size：“字体大小”，单位是px，em，pt，rem， vw。

font-family：“字体样式“：“SimSun”，

font-style：斜体：一般都是none。

font-weight：加粗：normal普通，bold 加粗。

line-height行高：200，2，px，em，可以继承。

页面中不是所有的字体都能用，因为字体要看用户的电脑里有没有安装。

中文：我们在工作中只能用微软雅黑，宋体，黑体。如果页面中，需要其他字体，让UR工程师切图。

英文：Microfoft-yahei，Time new Roman

比如说，为了防止电脑用户中没有安装微软雅黑这种字体，那么要用英语逗号“,“来隔开备选字体。就是说如果用户电脑里没有安装某种字体，就要用英文逗号隔开。

如果将英文字体放到最前面，这样所有的中文就不能匹配英文字体。就能自动变成后面的中文字体。

所有的中文字体都有对应的英文名称。

Eg：微软雅黑：microfoft-yahei

Font’标记可以合并

Font：12px（字号）/30px（行高）{或“%300”，这里的%300是针对于字号的百分比，一般行高都大于字号。}“times new roman”，“Microsoft-yahei”，“宋体”

Font：12px/24px“宋体”等价于 font：12px/200%“宋体”。

## 超链接的创建与美化

超链接

在HTML中创建超链接非常简单，只需用<a></a>标记环绕需要被链接的对象即可，其基本语法格式如下：

a标签：

<a href="跳转目标" target="目标窗口的弹出方式">任何元素</a>

在上面的语法中，<a>标记是一个行内标记，用于定义超链接，href和target为其常用属性，下面对它们进行具体地解释。

href：用于指定链接目标的url地址，当为<a>标记应用href属性时，它就具有了超链接的功能。

**target：用于指定链接页面的打开方式，**如果在一个 <a> 标签内包含一个 target 属性，浏览器将会载入和显示用这个标签的 href 属性命名的、名称与这个目标吻合的框架或者窗口中的文档。如果这个指定名称或 id 的框架或者窗口不存在，浏览器将打开一个新的窗口，给这个窗口一个指定的标记，然后将新的文档载入那个窗口。从此以后，超链接文档就可以指向这个新的窗口。

**\_self为默认值。**

**\_blank为在新窗口中打开方式。**

**\_parent:在父框架集中打开被链接文档。**

**\_top:在整个窗口中打开被链接文档。**

**\_*framename:*在指定的框架中打开被链接文档。**

<base /> 可以设置整体链接的打开状态

Link：网页重定向，打开新的网页。其用法如下：

<Link rel=”icon” href=”123.function”>

Rel:引用，规定页面的引用。·

cursor: pointer;当鼠标悬停在链接上方时，鼠标箭头变成手指形态。

Href：取地址。

**注意:**

暂时没有确定链接目标时，通常将<a>标记的href属性值定义为“#”(即href="#")，表示该链接暂时为一个空链接。

不仅可以创建文本超链接，在网页中各种网页元素，如图像、表格、音频、视频等都可以添加超链接。

<li><a href="">数码产品</a></li>href的英文全称是hypertext

Lis-tstyle：列表元素前的黑点，由于网页默认list元素前的黑点，所以通常用list-style：none来清除超链接列表元素前面的黑点。

text-decoration: none;清除超链接列表元素的下划线

超链接的状态：

变成会计元素之后再

爱恨准则：”Love hate“先爱，后恨

1 ul li a：link：{ } 默认状态，即鼠标还没有点击 这个链接之前的状态。

2 ul li a：visited { }访问后的状态，即鼠标点击这个链接之后的状态。

3 ul li a：hover{ }悬停状态，即鼠标悬停在文本链接上的状态.

4ul li a：active:{ } 激活状态，即鼠标摁下去，还没松开，表示鼠标点击这个链接，但是不松手，一刹那的状态，英语是“激活的“。

伪类

也就是说，同一个标签，根据用户的某种状态，有着不同的样式。这就是伪类注意：“：”与后面的代码不能有间隔，否则不起作用。

伪类用冒号表示：

：link表示拥护没有点击过这个超链接，是英语“链接“的意思。

注意：这四种状态，必须按照顺序来写；如果不按照顺序来写，效果失效。这个顺序被称为“爱恨准则“。

爱恨准则：”Love hate“先爱，后恨

a:link{

}

a：visited{

}

A：hover{

}

A：active{

}

# Background家族

## 1：background-color背景颜色

背景颜色属性在CSS中表示方法有三种：单词，rgb表示法，十六进制表示法。单词表示法：

都是简单的颜色。能用英语中单词表示的颜色有限

rgd三原色表示法：

回顾一下三原色：红、绿、蓝，顺序记一下。

Red green blue都是由16进制的颜色值组成

background-color：rgb（0-255（红），0-255（绿），0-255（蓝）其中，255：纯红，255：纯绿，255：纯蓝）

用逗号隔开，rgb’取值都是0-255

十六进制法：

Background-color:#ff0000

十六进制：0-9.A,B,C,D,E,F

类似于rgd’表示法，都是将红绿蓝三种原色用0-f的十六进制数表示，设置数即可表示颜色的值。

我们的十六进制取色#6位数字或字母 可以简写#xx

Eg：#ff0000简写为#f00

#112233简写为#123.六位每相邻两位都相同的可以简写否则不能简写。

注意：在大公司里，WEB前端工程师禁止在公司配色。

## 2：Background-image:背景图片：

语法：background-image:url(相对路径)。

电脑默认是重复的，所以要想平铺或者是不重复，可以用：

Background-repeat：no-repeat（不重复）

Background-repeat：repeat-x/y（横向/纵向平铺）

背景图片有平铺的功能，就是说盒子可以被铺满，但背景图片不能撑大盒子，即使背景图片大小大于盒子大小。

其他值还可能有：

top left、top center、top right、center left、center center、center right、bottom left  
bottom center

Padding的区域也有背景图片。

background: url(“images/bynewsbg.jpg”) ;

background-image: ;

background-repeat:;

background-color: ;

background-position:;

背景图片定位属性：

1：url（“图片地址”）属性：图片地址

2：position是“位置，方位“的意思。Background-position是指背景颜色/图片在浏览器页面里面的属性，为不是自定义盒子的属性。

位置定位可以为负数。Background-position：center right，bottom，left，top。距离右，左，下，上 多少多少像素

Background-position ：100px 右100 ，200px；下100.

关于广告位：PV是多少

3：background综合属性：

Background -attachment：规定元素是否随着页面滚动而移动。

Fixed：不滚动。Scroll：默认值，滚动

inherit：规定应该从父元素继承 background-attachment 属性的设置

Background属性和border一样，是一个综合属性。

Background：red url（）no-repeat可以省略任意部分：

Background：red；

Background：颜色，url（），repeat-x，position，fixed；

清除溢出：overflow：hidden；

预告片：CSS3里面的属性

Background-orign，background-clip，background-size

做一个著名的CSS精灵（雪碧）涂鸦

北京798艺术区

## 本地图片/外部文件的引入：

通过路径url或者src来引入，引入的时候要分html文件与本地图片/外部文件的文件夹的位置关系：

1. 绝对路径：
2. 相对路径：

①该文件的父级文件夹的兄弟文件夹：../文件夹名字或者文件名。

以此类推，向上找父级文件（夹）

②该文件的同级文件或文件夹：同级文件夹名字或者文件名。

③该文件的子级文件夹内的文件：统计文件夹/同级文件夹的子文件名。

以此类推，向下找子级文件（夹）。

单行文本

（2）多行文本

注意：超链接<a></a>，所有的a标签继承text-，font的属性。因为a标签有一个伪类权。

# 定位的几种用法

position：定位，相距，单位是px或百分比(%)。

Background-position是指背景颜色/图片在浏览器页面里面的属性，为不是自定义盒子的属性。

定位可以为负数。

Position：static；浏览器默认定位（静态定位）。

定位有三种：相对定位，绝对定位，固定定位。

1. .相对定位：position：relative是个变量，如果直接设置，那么就并没有脱离标准文档流。浮动了的元素也可以设置相对定位，从而使被定位的元素压盖在父级盒子上。相对定位就是盒子相对元素自己于原来位置的改变。
2. position：relative ：必须先声明：我要相对定位了。

Left：以自己原始位置左端为准，距离自己原始位置左端向右的移动距离。可正可负，加上“-“，那么是反方向移动。

Top：以自己原始位置上端为准，距离自己原来位置上端向下的移动距离。可正可负，加上“-“，那么是反方向移动。

Right：以自己原始位置右端为准，距离自己原始位置右端向左的移动距离。可正可负，加上“-“，那么是反方向移动。

Bottom：以自己原始位置下端为准，距离自己原始位置下端向上的移动距离。可正可负，加上“-“，那么是反方向移动。

实质：浏览器给我们渲染了一个肉眼可以看到的图像，但是盒子还是在标准文本流中。

1. 绝对定位：position：absolute，他的本身意义是脱离文档流。

绝对定位是脱离标准文档流的，所以所有的标准文档流的的性质，绝对定位之后都不遵循了。

绝对定位之后，表亲就不区分所谓的行内元素，块级元素了，就不用再设置display：block了，就可以设置宽和高了。

绝对定位的参考点：有两种参考点：

1）没有祖先盒子设置定位：

如果用top描术，那么参考点就是**浏览器首屏页面尺寸（body页面）**的左上角，而不是浏览器的左上角。同理，如果是bottom，那么参考点是页面（body）的左下角，而不是浏览器的左下角。也就是说，定位是会随着 鼠标滚动而滚动，跟首屏页面的位置坐标保持不变。

以Bottom描述：那么就是浏览器首屏尺寸（body页面）的左下角为参考点，距离浏览器首屏尺寸（body页面）的左下角的移动距离。可正可负，加上“-“，那么是反方向移动。跟滚动没有半毛钱关系。

1. 已经有祖先盒子定位：

会自动寻找已经绝对或相对定位了的祖先个盒子的左上角。所以如果想让一个元素在一个盒子内定位，那么就要让父盒子先相对/绝对定位（一般是相对定位）

然后再设置该元素绝对定位。

3：固定定位：position：fixed；left：50%：以页面垂直中线为界，距离左半屏幕，也就是在右半屏幕。

4：静态定位：position：static；

Background -position：attuchment；定位在某一个位置不动

写作规范：用更少的代码完成更多的功能，大模块布局用浮动（float），小模块布局用定位（position）

注意：1：overflow：hidden对已经定位了的子元素仍然起作用。

2：margin对定位之后的元素仍然起作用，但是参照物是定位后的元素位置。

三角遮盖/精灵图/雪碧图的遮盖截取：可以用定位或浮动与overflow：hidden相结合，通过使父级盒子塌陷（height与line-height不等）然后再对父级盒子设置overflow：hidden，达到遮盖截取的目的。

# Z-index

## Z-index的应用

* Z-index值表示着谁压盖着谁，数值大的压盖着数值小的。
* 只有定位了的元素，才能使用Z-index,也就是说，不管是相对定位，绝对定位，固定定位，都可以使用z-index值，而浮动的元素不能用。（工作中不能使用）。
* Z-index值没有单位，就是一个正整数，默认的z-index值是0；
* 如果大家都没有z-index，或者是z-index的值是一样的，那么谁写在Html后面，谁在上面压盖着别的元素。定位了的元素，永远能够压住没有定位的元素。
* 从父现象，如果父亲定位的比较之中，某一个父亲怂了，那么它的儿子也怂了。

# JS基本算法与表达式

一：网页布局的流程与拆分

1. Web前段工程师从UI工程师手中接过网页设计图。

2:把设计图上的文字等转成我们的html+css布局代码。

1. 实现页面效果，添加动态内容，与服务器交互。

二：拿到设计图纸之后

1. 找到设计版心，测量版心距离；
2. 多用padding，少用margin；

Web开发，web三层

Html 超文本链接语言 描述页面骨架

Css 层叠样式表

JavaScript JS描述页面的交互

3.1 JS用来描述页面交互效果，提升用户提体验

简单介绍几个JS页面效果，他能干什么：

1：轮插图

2：选项卡

3：地图  
4：表单验证

3.2 JavaScript的history

布兰登-艾奇（Brendan Eich，1961~今天）1995年在网景公司，发明了JavaScript，叫做liveScript，但是由于当时java语言特别火，所以帮了打款，就改名为liveScript，如同“北大”与“北大青鸟”的关系。北大青鸟就是“北大”的搭牌。

同时还有其他网页，比如VBScript和Jscript等，但是后来都被JavaScript打败，所以现在所有的浏览器，只运行一种脚本，就是JS。

3.3学习JS的三大部分。

1：EcmaScript 语法

2：BOM 浏览器对象（Browser object Model）

3：DOM 文档对象（Document object Model）

EcmaScript是由ECMA国际组织前身为欧洲计算制造协会，英文名称是“European Computer MaunFactuers Asspcaition”。

JavaScript是由公司研发的，公司开发而成的一定是有些问题，不便于其他公司拓展和使用，所以这个组织，牵头是javascript的标准，取名为Ecmascript。

简单地说，Ecmascript不是一门语言，而是一个标准，符合这个标准的比较常见的有javascript，ActionScript（Flash编程），就是说你学完了JS，你也可以去其他软件的语言进行编程，

2016年6月，发布了最新版本的Ecmascript语言功能更强大，但是，浏览器厂商不能那么快的追随标准。

工匠精神 寿司之神 我为故宫修文物

3.4今天的Ecmascript承担更多责任

2003年之前，Ecmascript被称为“牛皮癣”，用来制作页面上的广告，弹窗，漂浮的个广告，什么东西惹人烦，什么东西就是Ecmascript的开发的。所以浏览器就推出了屏蔽广告的功能。

2004年Javascript的命运开始改变了，那年，谷歌推出了Ajax技术，Ajax技能就是Javascript的一个应用。并且，那时候人们逐渐开始提升客户体验了。

2007年乔帮主发布了第一代iphone。从这一年开始，全球进入了移动互联网时代，javascript开始在移动设备上大量使用。这一年，乔帮主狙击了Adobe的flash。

2011年Ryan Dahl推出了Node.Js使得Javascript可以开始开发后台服务器了。

IT行业本来就是国务院东部所有行业的薪资老大。

今天Javascript工程师是IT里面的薪资老大。能够和IOS，Android踩才下去。现在公司采用的一律都是WebAPP，就是网页技术开发手机APP，什么意思呢，手机有三大系统：IOS Windows Android，安卓工程师，windows，IOS工程师各10人，共30人。比如说“携程”，女生就必须做三个版本。现在公司采用了Web前段端。好处不用招聘那么多工程师。网页更新，

3.5 JavaScript非常好学（最起码跟着Roky Ren是这样的）

JavaScript在“对初学者友好语言排行榜”中排名第一。

JavaScript是有界面效果的，比如你学C语言的，那么对不起，白纸黑字。但是你做JavaScript，它有绚烂效果，效果是可见的，你的劳动可以被看到。

JavaScript是弱变量类型的语言，变量只需要var关键字，在C#中，顶一个数

Int a=10；

String s=“大家好”;

Double a=“1.23”;

我们的课程：

语言核心：变量，循环，运算符，函数，选择语句;

DOM让盒子移动，让盒子轮播。

BOM 让浏览器被我们的代码所控制。

学习JavaScript的学习方法和HTML，CSS截然不同。

要多去品尝JS，多去思考编程的本质，HTML，CSS好比富士康的工人，很精密，很小心。JS的重复性工作几乎是0，机会本都是需要创造性的思维来学习。而不像HTML，CSS，Padding机械重复。

永远不要背诵程序，每一个程序必须要自己写出来，今后有一个隐性的作业，老师的代码重打三遍。

我们的JS课程最大的好处就是让小白，纯0基础的同学体验到什么是变成，如何编程，其思维是什么，所以我们的JS课程，都是基础知识。很多深入的东西，后面去讲。

JavaScript是前台语言，而不是后台语言。

四 开始写第一个JavaScript程序

4.1程序的书写位置

在<style></style>的下方。

## 1.Alert(“ ”)：弹出警示框。

Alert（”哇塞！！“）;

如果要弹出两个或以上警示框，那么就要写两遍（或以上）也即是说，一个alert只能弹出一个警示框。

学习程序，是有规矩的。就是程序有相同的部分，这部分相同的我们称为语法。世界上的所有语言，都必须喊出一声Hello World向迈凯思-利威尔致敬。

我们的语法都是一样一行的执行，不能越行执行。

语法规则：

1： JS对换行，缩进，空格不敏感，也就是说，在编程时可以随意换行，缩进，空格。但是由于工程师书写，观察方便，便于纠错，工程师要按照良好的规格来书写。

2：每一句必须加上分号，虽然不必要。如果不加分号，那么压缩之后就不能运行了。所以以后写程序，必须加分号，而且所有的分号都是英文的。

还有同学们一定要注意：

**<meta charset="utf-8">**

**<style type="text/css"></style>**

**<script type="text/javascript"></script>**

不要写错。

define：定义

注意：字符串必须加引号，而数字（int型数据）并不需要加引号。

Console.log：控制台输出，相当于printf表示在审查元素时显示。浏览器页面F12调出来Console.log的输出结果。控制台是工程师/程序员调试的地方。我们经常用这个方法调试输出。

Console.log(“s”)

Console.log(s)

一：编程语言的分类以及所学知识的问题

1：脚本语言（Script语言）

Vbscript（微软出的针对ASP编程）ActionScript（Flash，已经于2012年淘汰，Adobe公司于12年宣布停止发布销售Flash）TypeScript。宿主环境是指他的代码执行在神经么环境中，那么我们的javaScript执行在浏览器环境下。**解释性语言**

2：编译语言C/C++

这些语言编译后直接可以被机器运行，并且开辟进程。C/C++编写程序是跨平台的。

3：预编译语言。

代表作：java/c#.第一次先便宜成机器系统可以识别的语言，第二次再编译成机器能够运行的语言。

简单回顾一下js函数的书写位置：

<script type=”text/javascript”>

Alert:(“”)；

console.log()；

Document.write()；

</script>

二：变量

2.1：变量的整体感知

X,y等，计算机程序设计中，也是用这样的量，就是用字幕表示数字、字符串等来表示其他东西的量成为变量。

Js中，“=”是赋值语句，而不是判断语句，“==”才是判断语句。

Var ：就是英语“varliant”变量的意思。用于给变量赋值。具体用法：

Var+变量名=变量值。即将变量值赋给变量名。这个变量的值可以是数字，也可以是字符串。

例如：var a=“10”，var name=“jeff”

注意：变量名不能与浏览器关键字重名，那么浏览器不认。

Var关键字：所谓的关键字，就是有特殊功能的语法。关键字后面一定要用空格隔开。符号表示赋值，会将等号右边的值，赋给等号左边的变量名。

2.2：变量的命名规范

Var变量名称只能由英文字母ab，数字12，下划线 \_\_，美元符号$等以及他们的组合构成，且不能含有数字开头，不能是系统关键字 并且他们之间不能加空格。

例如：var a=你好；

var \_a=“23”；

var $a123=hello；

var a\_sun=“哎”；

var a$0=“65”；

他们都是合法的。

var a-1=“23”；

var 2@=“23”；

var 5year=“23”；

var a￥=“23”；

var ！@%&=“23”；

Var a b=“nio”；

他们都是不合法的。

大写字母是可以使用的，并且js对字母大小写是敏感的。比如，A与a是不一样的两个变量。

2.3 变量的定义和赋值;

变量只有定义之后才能使用，没有定义过的变量浏览器不认识，不能用。

Var a=100；

Consle.log (a)

在js当中，永远都是用var来定义变量，这点和外面那些java，C/C++，C#等妖艳贱货不一样。

Document.write()在屏幕上书写

Console.log(“s”)输出文本字符串s

Console.log(s)输出变量s的值

**梗**：字符串加双引号，变量名不加双引号，也即是：加双引号的都是文本字符串，不加引号的都是变量名。

也就是说，一个变量如果想输出值，就不应该加双引号。

变量里面能够存储数字，字符串，变量会自动根据存储内容的类型不同，来决定自己的类型。也就是说变量有类型之分。

如果一个变量中存放了数字，那么这个变量就是数值类型的变量。

Var a=100；

Console.log(a)

Typeof显示输出数据类型：typeof（变量名）。

用法：console.log(typeof(变量名))

JS中，只要是个数，那么就是数值类型，无论是正数，负数，还是小数无论大小，都是数值类型，统统用number表示。

”浮“就是浮点数，就是”小数点“，专业术语称为浮点数。

String：字符串类型，用双引号引起来的

键盘上的”+“在js中有两个意思。

1. 连字符 。例如：”我“+”爱“+”你“就等同于”我爱你“，用来拼接用户输入内容和浏览器网页显示的内容。
2. 运算符+

同样是加号”+“有时表示的是连字符，有时候表示的是加号。什么时候是连字符？什么时候是加号？就是只有当加号两边都是数字的时候他是运算符，否则就是连字符。两边既有数字又有字符串的时候他就按照顺序执行，从第一个字符串开始，往后的数字都会隐式转化成字符串，第一个字符串之前的数字还是数字，只是用“+”相连接。字符串还是字符串，连起来之后总体来讲是个字符串，括号除外。但在工作中先写字符串。

总结：无论前面的var怎样定义，后面console.log()输出中，只要加引号，就输出字符串，只要不加引号，就输出字。

运算符“-”：

当字符串与数字用“-”相连时，会隐式将字符串转化为数字，然后再相减。

赋值语句,赋值表达式：

Var a=1；

b=2；

a=b；

a=a++等同于a=a,a++

a=++a等同于a++,a=a.

Console.log(a+b)

现在有两个变量，var a=3；var b=5.要求不使用第三方变量，交换两个变量a和b的值。

Var a=3；

Var b=5;

a=a+b;

b=a-b;

a=a-b;

Console.log(a b);

在计算机程序运算中，+，-，\*，/，% （取余）这些运算符跟数学一样，都是乘法（\*）、除法（/）、取余（%）优先级大于+、-。但是（）表示优先级。()里面的运算优先于（）外面的运算。没有中括号[ ]和{}，只有（）参与运算。

五：prompt（”“）用户对话框函数，相当于C语言中的csanf（）

Var receive=prompt（”请输入你的报价，我们汇给你折扣后的价格“）

Console.log(receive)打印出来receive的内容

用户输入到用户对话框的东西可以被receive接收到,从而在开发者页面显示出来。

receive= parseInt（receive）;

ParseInt：将用户书写的结果中的字符串中的数字转化为整数值（默认为取整数位，舍弃小数位）。强制转换类型

Console.log（receive\*0.8）;

布尔值与运算关系符，逻辑运算符。

布尔值：

学习了两种变量，数字类型的值和字符串类型。

实际上，还有很多数据类型。

数值类型的值，有无穷多个，因为世界上所有的数字，都是数值型。

布尔类型（boolean）：

布尔值只有两个：True，false；真和假

字符串：string 数值：number 布尔值：boolean，都是typeof的属性。

用法：例如：console.log(1<2);那么网页自动显示布尔值.

## 关系运算符

> 大于号,

< 小于号,

>=大于或等于 ,

<=小于或等于,

==判断是否等于，如果两边的类型不一样而数值一样，那么将进行一个隐式的转换，再比较数值。

===判断类型和数值是否相等

！==判断是否不等，用法和==相反

！===：判断数据和类型是否不等，用法与===相反

逻辑运算符：

&&：逻辑与（且）：必须同时满足，才能返回布尔值true，否则为false。

||：逻辑或：只要有一个满足，就能返回布尔值true，否则为false。如果有多个||存在，那么浏览器从前往后判断，只要第一个条件成立，后面的||条件就会自动放弃，不管了。

！逻辑非：必须不满足，用于求反。

其他运算符：+=：

## 短路表达式：

||：如果逻辑或运算符左边的值布尔转换后为true，那么返回左边的值，

如果逻辑或运算符左边的值布尔转换后为false，那么返回右边的值或执行右边的代码（不管右边的值是真还是假），如果两个操作数都是是null（NaN/undefined），返回null（NaN/undefined）。

&&：如果逻辑与运算符左边的值布尔转换后为true，那么返回右边的值或执行右边的代码（不管右边的值是真还是假）。如果逻辑与运算符左边的值布尔转换后为false，那么返回左边的值，但是当逻辑与的左边为 null/NaN/undefined ，结果就会得到null/NaN/undefined。

## 三目运算表达式：

三目运算：c=a>b?”1”:”0”;(a>b吗？如果a>b，那么为1，否则为0)

问号冒号表达式（？：）if...else...的另一种简单表达式。

语法：（条件）？值1：值2

判断条件是否成立，如果成立，那么就取值1，否则就取值2。

编程就是程序语言+算法。

## 条件语句（if...else if...else...）

If.。。Else。。。语句：

如果。。。没那么。。。否则。。。。

Else if。。。。语句：

除了上面if（真语句）之外的条件成立。。。那么。。。

If的执行顺序：if（）{}

**注意**：结果有多个else if（）{}并排出现，这个结果语句只能在当前的if（）{}中结束，不能跳过当前条件执行。也就是说，**其中一个条件语句if（）{}的结果语句只针对变量的初始值而言，并不继承上一个if（）{}的结果内容，他们之间是平行的。**

**用法如下：**

**If（真语句）{**

**执行语句；**

**}**

**Else if（真语句）{**

**执行语句；**

**}**

**Else{**

**执行语句；**

**}**

**最后一个else中，只有没有条件的情况下才能直接写else{}，有条件的情况下要加要用else if{**

**}**

**括号里的真语句可以为已经定义过的真语句。**

**括号里写真语句，然后自动返回一个bool值。**

初学者一定不要省略花括号，把花括号写完整，按规范的模式来。

## For循环

**For（初始条件语句，初始条件语句；循环条件语句；执行完本每次循环之后的执行语句）{**

**执行语句；**

**}**

**这个结构不能出错，如果初始条件有多个，那么初始条件用“，”隔开，而不能用“；”隔开。**

**i++等价于i=i+1.先使用，再自增。**

**++i是先自增，后使用。**

**i--等价于 i=i，i--。先使用，再赋值。**

**i=i++; i=i, i++先赋值，再自增；**

**i=++i; i++, i=i先自增，再赋值；**

**break：跳出本循环体，结束所有循环。相当于终止本循环体。**

**Continue：在本循环体内，跳出本次循环，进入下一次循环。**

**如果需要打印所有的结果，那么要把console.log放到for循环里面，如果得出一个结果就行，那么就将console.log放到for循环外面。所以for循环通常用于循环滚动或多个结果的输出。**

**For循环嵌套：外部循环执行一次，内部循环执行内部循环条件的次数。**

**For（var i=0;i<10;i++）{**

**For(var j=0;i<10;j++){**

**Console.log(j);**

**}**

**}**

**for（var 变量 in 数组名）{**

**}**

**变量：是自己定义的一个关键字，相当于for循环中的i。**

**关联数组：他是个JSON对象（JSON格式的JS对象）**

**用处：当所处理的数组是以键值对的形式存在的时候，用关联数组的方法可以分割数组，从而将键值对打印出来，更适合处理字符串。但性能比普通for循环速度较慢。**

**For循环内尽量不要声明变量，否则每次循环结束，被声明的变量会被清空。**

**For循环与判断的先顺序（全选与反选的实现原理）：**

**共同点：二者控制方式相同。**

**①全选：是单个控制多个；**

**②反选：也是单个控制多个。**

**二者的单个按钮都是个指示灯，作用都是给状态。**

**不同点：二者先决条件（前提）不同。**

**①全选的前提条件是以全选按钮（单个个体）为准则，所以不必管群体按钮的样式，只需要跟他保持一致就行了，所以先判断再循环；**

**②反选的前提条件是群体按钮中部分按钮的状态，所以要先for循环再判断。**

**For循环只是对数组中的每一项进行遍历，它的对象是对数组中的每一项而言，循环条件有几次，就循环几次，每次都是一个。这点很重要，不要理解成一次性循环所有的。**

## Break

break 语句用于跳出循环。

continue 用于跳过循环中的一个迭代。

## Break 语句

break 语句可用于跳出循环。

在for循环中，如果碰到break条件语句的话，for循环会将i依次累加，直到满足break所指定的条件的上一个下标（不包括break的条件）。如果碰到满足break的条件，那么就不会再往下循环，而是立即跳出当前整个for循环体，。

break 语句跳出循环后，会继续执行该循环体之后的代码（如果有的话）

## Continue 语句

continue 语句中断循环中的迭代。如果碰到continue的条件语句，就会跳过本次循环，然后继续循环中的下一个迭代。

## While循环

While（表达式）{

执行语句；

}

While循环是对while后面括号的表达式进行判断，相当于一个条件。当执行语句符合这个条件时，那么执行while循环体。

do

{

}while（）

## 遍历筛选——循环+if判断

循环遍历的时候，如果想用循环遍历筛选合适的条件，可以在循环遍历中添加if语句。但是不能写else。但是不能写else。但是不能写else。重要的事情说三遍！！因为既然是筛选，总有不符合的，if执行完之后else总会执行，所以筛选就无效了。

## Switch ...case语句

Switch...case语句：多条件判断语句：

这是一个分支语句，表示只匹配符合表达式里的语句里的内容。Default表示除了你列出的所有条件之外的情况。

用法如下：

Switch（条件表达式）{

Case常量表达式1：语句1；

//Break；

Case常量表达式2：语句2；

//Break；

Case常量表达式3：语句3；

//Break；

Case常量表达式4：语句4；

//Break；

Default 语句5；

}

原理：首先设置表达式 n（通常是一个变量）。随后表达式的值会与结构中的每个 case 的值做比较。如果存在匹配，则与该 case 关联的代码块会被执行。请使用 break 来阻止代码自动地向下一个 case 运行。

注意：case后面只能跟常量表达式，不能跟变量。

try...catch...finally    语句

try...catch...finally 语句 为 Jscript 实现错误处理。

try {

tryStatements

}

catch(exception){

catchStatements

}

finally{

finallyStatements

}

**参数  ：**

   tryStatement     必选项。可能发生错误的语句。       
   exception     必选项。任何变量名。exception    的初始化值是扔出的错误的值。

  exception属性：

exception.name: 错误类型.

    exception.message：错误信息    
   catchStatement       
   可选项。处理在相关联的    tryStatement    中发生的错误的语句。       
   finallyStatements       
   可选项。就是无论try与catch语句执行结果如何，都会执行的语句。

**说明   :**

   try...catch...finally    语句提供了一种方法来处理可能发生在给定代码块中的某些或全部错误，同时仍保持代码的运行。如果发生了程序员没有处理的错误，Jscript    只给用户提供它的普通错误信息，就好象没有错误处理一样。

tryStatements    参数包含可能发生错误的代码，而    catchStatement    则包含处理任何发生了的错误的代码。如果在    tryStatements    中发生了一个错误，则程序控制被传给    catchStatements    来处理。exception    的初始化值是发生在    tryStatements    中的错误的值。如果错误不发生，则不执行    catchStatements。

如果在与发生错误的    tryStatements    相关联的    catchStatements    中不能处理该错误，则使用    throw    语句来传播、或重新扔出这个错误给更高级的错误处理程序。

在执行完    tryStatements    中的语句，并在    catchStatements    的所有错误处理发生之后，可无条件执行    finallyStatements    中的语句。

请注意，即使在    try    或    catch    块中返回一个语句，或在    catch    块重新扔出一个错误，仍然会执行    finallyStatements    编码。一般将确保    finallyStatments    的运行，除非存在未处理的错误。（例如，在    catch    块中发生运行时错误。）。

# 函数的性质

## 1.复习

1.1复习：if/for等这些都是流程控制语句，是编译语言通用的语句。

If语句：

选择语句，给程序添加多种执行线路，有且只有一条出路。注意**跳楼现象。所以我们发现计算机的两个基本功能：1：计算能力。2：流程控制能力。**

**1.2 for循环语句：**

**顾名思义，就是将结构类似的语句重复执行。For语句可简化程序的书写，不用大量的复制粘贴。For语句充分地体现了计算机的“奴隶性”。**

**比如题目：寻找1—1000之内所有能被5或6整除的数字。他是把所有的结果都打印出来，而不是单单一个结果。**

**上面这个例子的算法叫“穷举法”或“枚举法”，就是一个一个的尝试列出，实验，判断每一个可能的结果是否符合。**

**计算机就是人类的奴隶，它没有自己的思维，就是一个不吃饭，有着极强运算能力的机器而已。所以不能用人类想当然的思维来让计算机工作。**

**While和do while（他们不常用）：**

**While（条件表达式）**

**{**

**}**

**Switch case语句：**

**Switch（条件表达式）{**

**Case1：常量表达式：执行语句1；**

**break；**

**Case2：常量表达式：执行语句2；**

**break；**

**Case3：常量表达式：执行语句3；**

**break；**

**Case4：常量表达式：执行语句4；**

**break；**

**Case5：常量表达式：执行语句5；**

**break；**

**}**

## 2.函数

2.1整体感知：

函数的定义：

**function 函数名（）{**

**函数体语句；**

**Return 结果；**

**}**

函数的调用：

定义变量

**函数名（）；**

**Console.log()**

函数在使用前要先定义或先声明，然后再执行调用。函数名称是自己定义的。

函数就是一种代码封装，就是将一些语句封装到函数里面，通过函数的执行，来执行这些语句。

Function就是英语”函数“的意思。是指将一些功能封装到里面。

Function是一个关键字，和var，typeof一样，都是关键字，后面要加空格。

123.html:19 undefined函数的名字有明明规范，和变量名一样，只能是字母，数字，下划线，美元符号，不能以数字或他们的组合开头。

后面的小括号，里面存放参数，大括号里存放函数体语句。函数后面不跟分号。

函数的调用：

如果函数不调用，等于白写。调用一个函数，就是**函数名（）；**

**Return 的作用：**[JavaScript](http://lib.csdn.net/base/javascript" \o "JavaScript知识库" \t "http://blog.csdn.net/gchonghavefun/article/details/_blank)在事件中调用函数时用return返回值实际上是对window.event.returnvalue进行设置。

而该值决定了当前操作是否继续。  
当返回的是true时，将继续操作。  
当返回是false时，将中断操作。

而直接执行时（不用return）。将不会对window.event.returnvalue进行设置

关于javascript中的 return false和return true  
return 是javascript里函数返回值的关键字，  
一个函数内处理的结果可以使用return 返回，  
这样在调用函数的地方就可以用变量接收返回  
结果。return 关键字内任何类型的变量数据或表达式  
都可以进行返回，甚至什么都不返回也可以比如return ;

**函数的好处是什么？**

**1.大量重复的语句将会被写入函数里面，这样以后我们需要用的时候，直接调用函数，不用谢重复语句，简化编程。**

**调用时候如果不加小括号（），那么就不显示函数结果，而是函数体。**

**小括号里面的是参数。注意：函数体的结果要有返回值，一个函数只能有一个返回值。如果一个函数有多个返回值，那么只有第一个返回值有效，其他的返回值都无效。**

**例如：var a，b，c；**

**Function sum（a，b）{**

**c=a\*b；**

**Return c；**

**}//函数定义完成。**

**Var x=1，y=2，z；**

**z=sum（x，y）；调用函数sum。**

**Console.log(c);**

**我们的调用函数与定义函数的参数个数要相同，如果不同，若是少了参数，那么将会把没有定义的值称为undefined。**

**如果两个或多个名字相同但是函数体不同的函数同时出现，那么结果将只打印最后一个函数的结果，不管console.log出现在哪里。**

**注意：函数体里面声明的变量为局部变量，不能被外部所访问，但for循环体和if语句定义的能被外部访问。**

**全局变量如果与局部变量名称相同，那么全局变量在本函数中会被覆盖， 被谁覆盖了呢？被本函数体内新建的一个与局部变量名相同的变量所覆盖。也就是说，全局变量在函数体内不能被直接调用**

**同名函数的重载：两个函数的执行顺序。**

**当浏览器看到多个同名函数时，它会从上下依次浏览，但不会执行，只有当看到最后一个函数时，他再执行函数结果。**

**函数的形参和实参：**

**形参：形式参变量，是指函数定义时的变量，相当于公式中的字母，只是把函数原理表示出来，并不参加函数的调用。**

**实参：实际参变量，是指函数调用时的变量，用于参与实际运算。**

**Arguments的长度是由实参个数所决定，function.length由形参所决定。**

## JS变量与隐式转换：

我们知道，JS是一种 弱变量（只有一种 var 变量声明类型），弱类型（只有6种类型）的一种语言。

若变量固然简单，但也带来了一定的困扰：变量的隐式转换。

NaN：不是一个数字的数字，用来解决隐式转换。如果不同类型的两个变量做数学运算，那么就可能涉及隐式转换。如果不能隐式转换，那么就是NaN。

Infinity ：Infinity 属性用于存放表示正无穷大的数值。

Nall，undefined。

**注意：1：形参变量不需要定义，可以直接拿来用。2：”+“的特殊性：在函数定义时如果要用到”+“，那么在函数调用的时候要注意，实参变量如果要相加，那么就要用（parseInt（a），parseInt（b））。这是因为”+“在html中不仅有算数运算符”+“的作用，而且有连字符”+“的作用。如果不加parseInt，那么就是两个字符串相连接。其他运算符都可以进行隐式转换**

形参是对实参直接负责的，也就是说形参能把函数逻辑关系传递到实参当中去。

## 函数类型的声明与调用

函数表达式就是JS'中的一个工具类型。它有两种表达方式

①函数声明：

function 函数名 (){

执行语句

}

alert (函数名.name);也就是说函数名称是可以被打印出来的。但是这个功能并不是所有浏览器都支持的。

②函数声明提升

就是先调用，再声明

nihao();

function nihao(){

alert("我是一个函数声明");

}

但是用匿名函数赋值给一个变量，那么就不能先调用。

a();

var a=function (){

alert("我是一个函数声明");

}

浏览器报错：a is not afunction。

而应该把调用放到声明之后。

var a=function (){

alert("我是一个函数声明");

}

a();

这样就可以了。

匿名函数表达式也称之为lameda表达式。像这样，通过一个变量动态生成一个函数表达式，就体现了函数的灵活性。

理解函数提升的定义：就是要求大家理解函数声明与函数表达式之间的差异。

二 递归函数

function factorial(num){

if (num<1) {

return 1;

}

else{

return num\*factorial(num-1);

}

}

alert(factorial(6));

打印结果为720.

自定义函数

UnDefined与null：

Undefined：是指未定义的变量，但是占用了计算机内存。

Null：是指连内存都没有占用的变量。

Js分为三个模块：

1：EcmaScript：就是JS的一些基本语法：赋值，循环，判断，函数等。

2：Bom ，浏览器对象模型。就是用前面学习到的JS语法去操作浏览器。

3：Dom 对html标签进行操作

补充;

1：.length 表示显示字符串的个数。

数据类型：Object：null和undefined，string，number，boolean（true/false），isNaN：判断是否是数值类型（true/fslse）

函数声明：function sum=（a，b）{ }

匿名函数 var sum=function（）{ }

自执行函数：（function（）{ }）

## javascript变量

作用域：如果几个变量都在同一个环境下，那么他们都称为在一个作用域下面。

循环和函数的作用域不一样。循环变量本来就是全局变量。

全局变量：

是指在函数外声明的变量或者在函数内部定义的无var 声明的变量。

调用特点：在任何情况下都可以调用。

局部变量：

是指在函数体内部用var声明的变量或者函数的参数变量，包括形参与实参。

调用特点：只能在当前函数内调用，出了该函数就是undefined。

作用域：

函数变量的作用域：

变量的优先级：

1：局部变量高于同名参数变量。

2：参数变量高于同名全局变量。

3：局部变量高于同名全局变量。

变量的提升：

在函数体内部声明变量会把该声明提升到该函数体内部的最顶端，只提升变量不赋值，因为在函数体语句内，函数体是从上往下执行的。在函数体外部声明的变量都是该函数的全局变量

在函数体内部出现的变量都是在函数体内部声明的。

作用域链：

1：内层函数可访问外层函数的局部变量。

2：外层函数不能访问内层函数的局部变量。

对象/属性链：JavaScript的变量都是对象的属性，而该对象可能又是其它对象的属性，而所有的对象都是全局对象的属性，所以这些对象的关系可以看作是一条链，链头就是变量所处的对象，链尾就是全局对象.

## 函数闭包

要学习函数闭包，一下这几点必须知道：

闭包是指有权利访问另一个作用域于内的函数变量。创建闭包的常见方式就是在一个函数内部创建另一个函数。

函数的闭包实际上是作用域的问题，在工作上牵涉到的主要问题就是浏览器内存泄露。函数（）如果加了（），就执行，如果不加（），就不执行，只是个函数体。

比如alert（a）和alert（a（））是不一样的，前一个是打印函数体，后一个打印的是a（）的结果。

**函数闭包问题，说白了就是变量作用域的问题。**

**1：函数可以理解为既可以存放，又可以赋值给变量的一个内存空间。函数体内部可以调用另外一个函数体的变量，也可以调用全局变量。如果一个全局变量在一个函数体内部被赋值了的话，那么它在该函数内就成为了一个局部变量；在另外一个函数内调用的是该变量在上一个函数内的值。也就是说，函数体跟全局这两个内空间存是不同状态的两个空间，他们可以分别调用或声明变量，二者不一样。**

**2：在函数内部声明的变量永远是局部变量，只能在该函数内部使用，在其他函数内无效。**

**3.如果在函数外面（全局范围）声明了一个变量，再在函数内部声明一个同样的变量，那么该变量的值以函数内部赋值为准。**变量的作用域

## 函数详解

一：this关键字

This 关键字代表着当前对象，

1. 在script标签里面打印出this，那么this是当前的window对象，打印结果是[Object Window]
2. 如果写在script标签里的变量或函数，其实都是我们写在window对象下的对象成员。写在function下面的this指向的是window，而不是指向函数的形参。
3. 事件中的this指向：

指向函数所作用的html元素对象（标签）

二window.SetInterval(“函数（）”，3000)；它调用的函数永远是全局函数，而不是局部函数，说白了必须是window对象下面的函数。我们在window.onload这个事件里卖弄定义的函数，它的作用域只是onlod事件（window对应的事件有很多，如window.alert,window.console.log等），所以SetInterval无法找到并调用。

注意：如果在某个子作用域中调用了setInterval（）方法，那么在在此时，setInterval（）方法里的函数作用域就是全局的了。如果还想调用子作用域的this，就要转换作用域，就要在setInterval（）方法外面写：const \_this=this；在setInterval（）方法里面调用\_this.

## 关于ValueOf( )的用法

ValueOf是返回指定对象的原始值，而tostring是强行转化输出为字符串形式。

1：ValueOf它是我们object对象下的函数。其他的对象也有这个函数，为哦什么呢，因为任何对象都是继承了object！既然Object有，那么孩子也有，理所当然。

## New 方法与构造函数

构造函数不需要返回对象，饮食的返回对象

1：当我们在页面上大写定义一个函数的时候，那么约定俗成的就是大写函数是为了创建类。

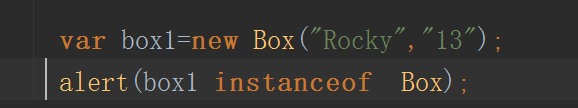
2：当我们在外面通过new 函数名称（构造函数）实例化一个对象的时候。浏览器会隐式的给我们new object()对象

3：然后把我们定义好的function构造函数添加到object对象里面作为他的一个成员。

4：通过New 构造函数实例化的时候，我们就可以得到一个具体的类实例。

使用构造函数的方法解决了重复实例化的问题，又解决了对象识别问题。但是问题是感官上没有new Object()。为什么可以实例化。

所有的构造函数实例都是



可用之处：1 构造函数没有new Object();后台会隐式自动 创建一个 var box=new Object();

2 this 就指向了box

3 构造函数不需要返回对象，隐式的返回对象。

规范： 1 构造函数名首字母大写

2 必须使用new 构造函数名 首字母大写

规范:构造函数首字母大写

对象的伪装：要伪装的对象.Call(“伪装的对象”，要伪装的对象)；

对象方法的比较：

1：如果比较两个函数的运行结果，那么比较的是两个输出结果的值，结果有可能一样。

2：如果比较两个函数的名称，两个函数体也有可能一样，那么打印出来也一样。但是实质上比较的是两个函数体的地址位置。那是肯定不一样的。

堆栈：

栈方法：（LIFO）（先进后出）

队列方法（LIFO）后进先出

对象的伪装： 被伪装的对象.Call(‘伪装的对象’，要伪装对象的实参)。有多少参数写多少参数，可以是函数。

场景：有利于自己的自定义对象成员和实现方式。不被泄漏。

## Call，apply，bind的用法：

### Call的用法：

是自定义一个变量类型a，构造一个函数体，函数名.call(a,属性1，属性2.。。)然后让元素a运行。

//声明一个原型对象方法：

function Box(name,age)

{

this.name=name;

this.age=age;

this.run= function () {

return this.name+this.age+"运行中";

}

}

var o=new Object();//声明一个伪装体对象

Box.call(o,'Rocky',30);//用新声明的函数伪装原函数。

alert(o.run());

注意：call改变了this的指向。

如果在函数体1里面定义了一个函数体2；函数体2描述函数体1的一个对象，那么函数体2就成了局部函数。在函数体1外部直接调用函数2的时候，window会把函数2定义为undefined，一般不能调用。如果想要在函数体外部调用函数体2，那么就要用到指针this。

this.函数1属性=function（）{

执行语句；

}

那么这样就能在函数体1外部调用函数体2了。

在事件函数中，如果要在内部函数体中调用外部函数体中的对象，要另外声明一个变量存储父级函数的“this”，否则直接写对象名。

## apply的用法：

Apply的用法同call一样，区别只是apply以数组形式传参，而call是以单个参数形式传参。

总结：call和apply，bind都是用来改变函数中this的指向  
不同的是call和apply不仅改变了函数中this的指向并且立即调用了函数而bind仅仅是替换了this没有调用

# JS数据类型与Object

## Js数据类型

任何类型都有布尔值，强制将其他类型转化为boolean：

|  |  |  |
| --- | --- | --- |
| 数据类型 | Boolean表现 | 说明 |
| String | True/false | 非空字符为true，空字符为false |
| Number | True/false | 非零为true，零为false |
| Undefined | 只有false | Undefined类型只有一个值undefined，并且为false |
| Null | False | 他是一个空对象。 |

**一：ECMAScript就是语法，它里面包含了6个语法对象**

**①Js语法的初级数据类型有：**

**Undefined（只有一个值undefined）**

**string**

**number（包括数字和NaN）**

**null**

**Objecct(包括function与array)**

**boolean**

**Undefined：JS里面没有被声明的局部变量或已经被声明但未被赋值的变量。它里面只有一个值，那就是Undefined。**

**String：字符串类型。**

**Number：数字和NaN**

**Null：未被开辟空间来存储值的变量成为null。也就是说不存在的变量打印出来就是null。**

Null 是第二个只有一个值的类型，这个特殊值就是null，从逻辑上来说，null值表示一个空对象，而这个对象用typeof检测null值得时候会返回object。也就是说，其类型是object。

**Objecct：对象。书写形式为：**

**Var aa={**

**A：”nini”,**

**B:”wertp”**

**....**

**}**

**Console.log(aa[“A”]);//结果为“nini”**

**Console.log(aa.A);//结果为“nini”**

**Aa[“A”]=”nini”,**

**aa.A=”nini”**

## 判断数据类型的属性与方法：

**1.Typeof（）方法：**

**Typeof（）方法用来检测某个变量属于六种数据类型的哪一种，返回首字母小写的字符串类型的数据名称（“object”，“string”，，，，）。**

1. **constructor属性：**
2. **constructor.name也是检测某个变量属于六种数据类型的哪一种，但要比typeof准确。返回首字母大写的字符串类型的数据名称（“Object”，“String”，，，，）。**

**比如，用typeof判断一个string类型的数据，只能判断到object这一层，但是constructor.name可以准确判断到string这一层。**

# Array类型的方法属性

## 数组的定义：

Var 数组名=new Array（元素个数）;

数组名[1]=”元素1”；

数组名[2]=”元素2”；

数组名[3]=”元素3”；

数组名[4]=”元素4”；

..............;

这种方法比较官方，

也可以：

Var 数组名=new Array（“元素1”，“元素2”，“元素3”，.......）

数组名 .length:该数组的元素个数。

或者是

var 数组名=[“元素 1”,”元素2”,”元素3”,”元素4”,..............] 这种方比较简单，比较常用。

默认的元素的索引位从左到右是第0位，第1位，第2位，第3位，第4位。。。。。

定义一个空数组：var a=[ ];

注意：中括号里的数据类型必须是同类型。

关于数组的长度length:

数组的length属性指的是该元素所包含数据类型的长度或个数，字符串的个数，数组元素的个数等，并且可以动态设置，即初始化元素个数之后还可以设置length的值。

如果你定义的数组元素个数大于真正数组元素，那么数组元素定义的个数可以被你真正写的元素个数所扩大，这种现象是因为在工作中汪汪你不知道你到底要定义多少个。

相反，如果你定义的数组元素个数大于真正数组元素，那么多余的元素就会显示undefined或显示空字符，就是空白，啥都没有，但实际上是打印了。

在工作中，通常要将每一个数组元素单独打印，可以用for循环来实现。

var a=[1,5,8,9,12];

for (vari = 0; i < =a.length; i++) {

i=a[i]

alert(i);

}

如果直接打印数组名称，那么会把数组默认转化为字符串。

注意，结果要注意字符串跟数字还有符号的表达，该用连字符用连字符，该用引号的用引号。

## 数组元素的添加：

1.添加索引：数组名[n+1]=”新元素”；

2.concat：js的Array对象提供了一个叫concat()方法，连接两个或更多的数组，并返回合并后的数组。

用法：

var arr0 = arr1.concat(arr2,arr3,arr4,......)；

Console.log(arr0);

arr1.concat(arr2,arr3,arr4......)--------这是个赋值语句。想要得到新数组，必须用变量接收；

concat()方法可接受任意多的数组，把他们逐个添加到已有的数组，并返回添加后的新数组。无论concat进来的是一个数组还是一个元素，都返回新组成的数组。a.concat(b)返回的是新的数组。

var arr1=["a","b","c","d"];

var arr2=["e","f"];

Var arr3=arr2.concat(arr1);

console.log(arr3);

//返回 ["e", "f", "a", "b", "c", "d"]。

将数组b拓展给a，使a成为a和b合并后的新数组，并返回新数组的长度。

1. 向数组的末尾添加元素：push（）

push（）方法可接受任意多的数组元素，把他们逐个添加到已有的数组，并返回添加后的新数组的长度，不可用变量接受。无论push进来的是一个数组还是一个元素，都返回原数组长度统统+1。

数组在for循环内添加元素的话应该用push，而不应该用concat。

a.push(b)返回的是新数组的长度（length）；而执行完push语句后，打印a,才返回新数组。例如：

var arr1=["a","b","c","d"];

var tt1=1;

arr1.push(tt1);

console.log(arr1.push(tt1));//返回5；

console.log(arr1);//返回["a", "b", "c", "d", 1]；

4.在数组的开头添加新元素:unshift()

Unshift()推入，从开始位置添加元素，并返回被添加的元素。

5.在数组的第2位置添加一个元素 - splice()

## 数组元素的删除与截取：

Arr.join() 把数组的所有元素放入一个字符串。元素通过指定的分隔符进行分隔。

数组名.shift（）移除数组中第一个元素并返回数组并返回被移除的元素，原数组长度-1

②pop（）从数组的末尾移除最后一个元素，返回被移除的这个元素。

Slice（a，b）（自立为王）从一个数组中抽取一个或多个元素来组成一个新的数组。a是从第a个下标开始截取，b是到第b个下标

Slice（i，j）--------从一个数组中按下标抽取i-j元素来组成一个新的数组，自立为王。

王炸：

数组名.splice(i,j)----------从第i索引位开始删除，删除j个元素。i>=0,j>=1。

这个很叼：数组名.splice(i,j，“元素1”，“元素2”。。。)----------从第i索引位开始删除，删除j个元素，并从第i个元素开始添加“元素1”，“元素2”。。。

## 数组的循环遍历

### 1.$.each(prama,function(index,el))

jQuery遍历数组中每一个元素并为每一个元素添加事件：

.each()方法，该对每一个选中的对象添加函数事件，相当于for循环.

.each()的2种写法：

1：$(“”).each(function(index,element){

//添加事件语句

index是索引值，

此时的$(this)与element相同，都表示获取所遍历的每一个节点对象；

但是：$(this).click(function(){

//实现功能语句

此时的$(this)指的是你点击的单个节点对象。

})

})

2$.each($(“”),function(index,element){

//添加事件语句

此时的$(this)与element相同，都表示获取所遍历的每一个节点对象；

})

### 2.循环对象$.each()

$.each()接收两个参数，第一个是需要循环的对象或数组，第二个是回掉函数。

用法：$.each(obj,function(key,value){

})

$.each()方法比较牛逼，可以循环遍历对象，依次输出对象的每个键与值。

index()方法：

jQueryObject.index( [ object ] )或者$(selector).index()

index() 方法返回指定元素相对于其他指定元素的 index 位置。

这些元素可通过 jQuery 选择器或 DOM 元素来指定。

**注释：**如果未找到元素，index() 将返回 -1。

**返回值：**

index()方法的返回值为Number类型，返回指定元素的索引位置(从0开始算起)。

注意：以下文本描述中的当前元素表示当前jQuery对象的第一个元素。

如果没有指定参数object，则返回当前元素在其所有同辈元素中的索引位置。

如果object为String类型，则将其视作选择器，返回当前元素在选择器所匹配的元素中的索引位置。如果该选择器不匹配任何元素或者当前元素不在匹配到的元素内，则返回-1。

如果object为DOM元素或jQuery对象，则返回该元素(或该jQuery对象中的第一个元素)在当前jQuery对象所匹配的元素中的索引位置。

这个写法有个bug，index()方法打印出来的不是相对于你规定的那些元素中的洗标，而是相对于页面boday下所有的Html标签下标。

### 3.Array.map()

map()方法遍历数组的每一个元素。Item是数组的单个元素，index返回item的索引。map循环有一个至关重要的前提条件：那就是map循环的对象必须是颜色意义上的数组（constructor.name==”array”）。另外，map()方法是有返回值得，要把map的结果返回出来。

用法：arr.map(function(value,index,array){

　　//do something

　　return XXX

})

Map方法的性质：

1. map() 不会对空数组进行检测。

2. map() 不会改变原始数组。

3.跳出map循环的方法：return false;

### 4.Array.forEach()

用法：arr[].forEach(function(value,index,array){

　　//do something

})

用法与map相同，只不过没有返回值。

## 数组的搜索：

### 1.Arr.indexOf(item);

indexOf()方法可返回某个指定的字符串或数组值在字符串或数组中首次出现的索引位置，若不在其中则返回-1 。但是这种方法只能搜索基本的数组，复杂的对象数组不起作用，始终返回-1.

### 2.Array.find()/array.findIndex()方法

用法：Array.find(function(){})/array.findIndex(function(){})

find()、findIndex()接收一个回调函数，搜索一个满足函数条件的值。

find()与findIndex()的不同支持就在于：find()方法返回第一个满足函数条件的数组元素值，而findIndex()返回该值的索引。如果未找到的话，find()会返回undefined，findIndex()返回-1。

### 3.arr.includes(item,[index])

Includes()方法接收两个参数，第一个是被搜索的元素，第二个是开始搜索的索引位置，也就是从第几位开始搜索，正数是从该位开始搜索，负数是从右边倒数该位置开始索引。省略之后就是全数组搜索。返回的结果是一个布尔值，找到的话就是true，找不到返回false。

### 数组的筛选——循环遍历+if判断

循环遍历的时候，如果想用循环遍历筛选合适的条件，可以在循环遍历中添加if语句把符合条件的筛选出来。但是不能写else。但是不能写else。但是不能写else。重要的事情说三遍！！因为既然是筛选，总有不符合if条件的，if执行完之后else总会执行，所以筛选就无效了。

## 数组的检测：

### Array some() 方法

some() 方法用于检测数组中的元素是否满足指定条件（函数提供）。

用法:

array.filter(function(currentValue,index,arr), thisValue)；

currentValue：当前在数组中处理的元素。

Index：正在处理元素在数组中的索引。可选。

Arr：调用了filter的数组。可选。

thisValue：执行 callback 时的用于 this 的值。可选。

some() 方法会依次执行数组的每个元素：

如果有一个元素满足条件，则表达式返回true , 剩余的元素不会再执行检测。

如果没有满足条件的元素，则返回false。

some() 不会对空数组进行检测。

some() 不会改变原始数组。

### Array filter() 方法

filter() 方法创建一个新数组, 其包含通过所提供函数实现的测试的所有元素。

array.filter(function(currentValue,index,arr), thisValue)；

currentValue：当前在数组中处理的元素。

Index：正在处理元素在数组中的索引。可选。

Arr：调用了filter的数组。可选。

thisValue：执行 callback 时的用于 this 的值。可选。

filter() 不会对空数组进行检测。

filter() 不会改变原始数组。

## 数组的转义：Array.from()

Array.from()方法就是将一个类数组对象（具有length属性的对象）或者可遍历对象转换成一个真正的数组。

let arrayLike = {

    0: 'tom',

    1: '65',

    2: '男',

    3: ['jane','john','Mary'],

    'length': 4

}

let arr = Array.from(arrayLike)

console.log(arr) // ['tom','65','男',['jane','john','Mary']]

如果把length属性去掉，就会出现问题。

console.log(arr)  // [ undefined, undefined, undefined, undefined ]

由此可见，要将一个类数组对象转换为一个真正的数组，必须具备以下条件：

1、该类数组对象必须具有length属性，用于指定数组的长度。如果没有length属性，那么转换后的数组是一个空数组。

2、该类数组对象的属性名必须为数值型或字符串型的数字

ps: 该类数组对象的属性名可以加引号，也可以不加引号

## 数组与字符串的转换：

Join（）将数组转化为字符串。

.join(“,”“ ”“.””/””：”)以逗号，. / 冒号等将数组中的每个元素拼接成字符串。

语句名.split(“,”“ ”“.””/”””)-----------用逗号,空格，斜杠等分割字符串，并返回一个分割后的数组。

## 数组元素排序：

### Sort排序：

Sort（）方法的排序规则：将数组中的每一个元素toString（）方法调用，然后根据字符串进行排序。字符串排序：按照字符串的ASCII码值，默认是ASCII升序排列。

function sortNumber(a,b){

return a - b；

}

*var* arr1=[3,45,56,12,23,26,78,86,92,71,54];

*console*.log(arr1.sort(sortNumber));

这样，就可以准确地把数组从小到大排列起来了。

### 冒泡排序:

*function* maopao (*arr*) {

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

for (*var* j = 0; j < arr.length-i-1; j++) {

if (arr[j]>arr[j+1]) {

const swap = arr[j];

arr[j]=arr[j+1];

arr[j+1]=swap;

}

}

return arr;

}

*var* arr=[2,5,3,1,7,8,78,89];

*var* temp=arr[j];

arr[j]=arr[j+1];

arr[j+1]=temp;

}

*var* dd= maopao(arr);

*console*.log(dd);

### 对象数组排序：

通常我们需要根据一个对象数组的每个元素的某一个性质（比如id）来进行排序；这个要比上面稍微复杂一些，但是原理思想是相同的，所以写法类似。

var compare = function (obj1={}, obj2={}) {

var val1 = obj1.id;//若是根据id排序 那么就写id，如果根据其他属性排序 就是其他的

var val2 = obj2.id;

if (val1 < val2) {

return -1;

} else if (val1 > val2) {

return 1;

} else {

return 0;

}

}

var objArr=[

{

name:"李二",

id:3,

age:20,

},

{

name:"张一",

id:1,

age:18,

},

{

name:"找四",

id:10,

age:90,

},

{

name:"王三",

id:8,

age:18,

},

{

name:"冯五",

id:5,

age:18,

},

];

console.log(objArr.sort(compare));

## 数组去重：

### new Set()去重

new Set()方法：对基本类型的数组进行去重操作。ES6中新增了Set数据结构，类似于数组，但是 它的成员都是唯一的 ，其构造函数可以接受一个数组作为参数

let array = [1, 1, 1, 1, 2, 3, 4, 4, 5, 3];

let set = new Set(array);

console.log(set);

### 不排列数组的情况下

思路：

1.创建一个新的数组存放结果

2.创建一个空对象

3.for循环时，每次取出一个元素与对象进行对比，如果这个元素不重复，则把它存放到结果数组中，同时把这个元素的内容作为对象的一个属性，并赋值为1，存入到第2步建立的对象中。

说明：至于如何对比，就是每次从原数组中取出一个元素，然后到对象中去访问这个属性，如果能访问到值，则说明重复。

一：function func1(array){

var temp = [];//临时数组

for(var i=0;i<array.length;i++){ //遍历当前数组

//判断array[i]是否在临时数组中，不在则将array[i]的值复制到temp里 //indexOf()方法可返回某个指定的字符串或数组值在字符串或数组中首次出现的位置，若不在其中则返回-1

if (temp.indexOf(array[i]) === -1) {

temp.push(array[i]);

}

}

return temp;

}

二：Array.prototype.unique3 = *function*(){

*var* res = [];

*var json* = {};

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

*console*.log(*json*[*this*[i]]);

if(!*json*[*this*[i]]){

res.push(*this*[i]);

*json*[*this*[i]] = 1;

}

}

return res;

}

*var* arr = [112,112,34,'你好',112,112,34,'你好','str','str1'];

alert(arr.unique3());

### 排列数组情况下：

思路：

1.先将原数组进行排序

2.检查原数组中的第i个元素 与 结果数组中的最后一个元素是否相同，因为已经排序，所以重复元素会在相邻位置

3.如果不相同，则将该元素存入结果数组中

Array.prototype.unique2 = function(){

this.sort(); //先排序

var res = [this[0]];

for(var i = 1; i < this.length; i++){

if(this[i] !== res[res.length - 1]){

res.push(this[i]);

}

}

return res;

}

var arr = [1, 'a', 'a', 'b', 'd', 'e', 'e', 1, 0]

alert(arr.unique2());

数组名.reverse()-------------数组元素进行反转，排头变队尾。

indexOf() -------方法可返回某个指定的字符串值在字符串中首次出现的位置。它的合法取值是 0 到 stringObject.length - 1。如省略该参数，则将从字符串的首字符开始检索。indexOf() 方法对大小写敏感！如果要检索的字符串值没有出现，则该方法返回 -1。

**For循环之后，a[ i ] .index=i:数组当前元素的索引值；通过给index赋值为i来实现获得当前元素的索引值，供下一次调用。**

**说明：i不是索引值，而.index才是索引值。i只是index的值。**

主要处理下拉菜单问题。

### 对象数组去重(特殊数组去重)

思想：因为对象数组不能直接用indexOf()方法判断，所以我们可以简化为普通的临时数组代替，做简化处理。

①for循环arr0，把所有原始数组的每个对象的某个属性（如id）都push到一个新数组arr1内，这样push之后的数组就变成普通的有重复数组了。这样就简化为普通的数组去重问题考虑。

②再声明一个中间数组arr2，for循环arr1，判断arr2.indexOf(arr1[j])==-1?如果不存在，就把arr1[j]push到arr2中去，否则(重点在否则)，arr2.push(0)(这里arr2随便push一个东西就行，目的是跟push进来的arr1[j]做区分),此时的数组arr2.length===arr0.length===arr1.length。

③声明一个目的itemArr，for循环arr2，此时arr2肯定既有arr1的元素又有0，判断arr2[k]!=0;就让itemArr.push(arr0[k]);

④console.Log(itemArr);我们就得到了去重后的数组。

代码如下：

var arr0=[

{

name:"张三",

id:1,

age:18,

},

{

name:"李四",

id:2,

age:20,

},

{

name:"张三",

id:1,

age:18,

},

{

name:"王五",

id:3,

age:90,

},

];

var arr1=[];//id数组

var arr2=[];//中间数组

var itemArr=[];//目的数组

console.log(arr0);

for(var i=0;i<arr0.length;i++){arr1.push(arr0[i].name);}

for(var j=0;j<arr1.length;j++){

if(arr2.indexOf(arr1[j])==-1){

arr2.push(arr1[j]);

}else{

arr2.push(0);

}

}

console.log(arr2);

for(var k=0;k<arr2.length;k++){

if(arr2[k]!=0){

itemArr.push(arr0[k]);

}

}

console.log(itemArr);

## 将伪数组转化为严格意义上的数组

### 伪数组的定义：

因为数组（array）严格意义上来讲是Object类型的一支，所以这里把符合以下条件的对象称为伪数组：  
1，具有length属性  
2，按[索引](http://www.jzxue.com/tag/%CB%F7%D2%FD/" \t "https://blog.csdn.net/diligentcatrich/article/details/_blank)方式存储数据  
3，不具有数组的push,pop等方法

常见类型有：

1. function内的arguments 。  
   2.通过以下几种：

Object JSON对象

document.forms

Form.elements

[Select](http://www.jzxue.com/tag/Select/" \t "https://blog.csdn.net/diligentcatrich/article/details/_blank).options

document.[get](http://www.jzxue.com/tag/get/" \t "https://blog.csdn.net/diligentcatrich/article/details/_blank)ElementsByName()

document.getElementsByTagName()

document.getElementsByClasssNameName()

[childNodes/children](http://www.cnblogs.com/snandy/archive/2011/03/11/1980085.html" \t "https://blog.csdn.net/diligentcatrich/article/details/_blank)

等方式获取的集合（[HTML](http://www.jzxue.com/wangyesheji/html/" \t "https://blog.csdn.net/diligentcatrich/article/details/_blank)Collection，NodeList）等。

### 转换方法：

1. 使用Array.prototype.slice.call();或者[].slice.call();方法
2. 使用ES6中Array.from();

HTML:

<body>

<div class="d1">1</div>

<div class="d1">2</div>

<div class="d1">3</div>

<div class="d1">4</div>

<div class="d1">5</div>

</body>

Javascript:

var d1=document.getElementsByClassName("d1");

console.log(d1);//HTMLCollection(5) [div.d1, div.d1, div.d1, div.d1, div.d1]

var d1Arr=Array.from(d1);

console.log(d1Arr);//[div.d1, div.d1, div.d1, div.d1, div.d1]

## 数组的扩展(...)

扩展运算符（spread）是三个点（...）。它好比 rest 参数的逆运算，将一个数组转为用逗号分隔的参数序列,该运算符主要用于函数调用,这种方法实际上是对数组的深拷贝。

console.log(...[1, 2, 3])

// 1 2 3

var ar=[0,1,2,3,4,5];

console.log(...ar)//0 1 2 3 4 5

console.log([...ar])//[0,1,2,3,4,5]

## 数组元素的迭代

### Foreach(()=>{})方法迭代

arr.forEach()方法接受一个函数作为筛选条件,可返回数组每一个元素被该筛选条件筛选后的所有结果。

### Map(()=>{})和filter(()=>{})方法：

Map和filter也可以接受一个函数。Map返回的是数组每一个元素经过函数筛选的所有结果，而filter返回的是每一个满足函数筛选条件的所有数组元素。

### 数组元素的累加

reduce() 方法接收一个函数作为累加器，数组中的每个值（从左到右）开始缩减，最终计算为一个值。它有4个参数，上个元素（previous）,当前元素（current）,当前元素的索引（currentIndex）和当前数组(arr).如果直接用当前数组的话，那么第4个参数就不用写了。

arr.reduce((previous,current,index)=>{

return previous+current;

})

### 数组元素的循环迭代

Foreach、map、for...of，for...in

### 数组的填充：fill方法

Arr.fill(item,beginIndex,endIndex)

Arr.fill方法会复制内部元素并填充到相应索引位置上。

Fill()方法接收3个参数：item是要填充的元素，beginIndex是开始的index值，endIndex是结束的index。

### 数组的内部潜复制：copyWithin()方法\*晦涩难懂 不明白也罢

array.copyWithin(target,[start, end])

该方法是将数组内部的元素进行复制，然后替换另外一些元素。该方法接收3个参数：

Target：目的索引，复制过来的元素都将从该位置替换原来的元素

Start：从该索引开始找元素(左边闭合)，可省略，省略之后。

End：找到该索引结束（右边开，不包括该元索引），省略之后会将找start开始，到数组结束为止所有的元素来替换。

## 将一般数组转化为对象

let arr = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15];

let jsonObj={};

arr.map((item,index)=>{

jsonObj[index]=item;

})

console.log(jsonObj);

//{0: 1, 1: 2, 2: 3, 3: 4, 4: 5, 5: 6, 6: 7, 7: 8, 8: 9, 9: 10, 10: 11, 11: 12, 12: 13, 13: 14, 14: 15}

## 将特殊数组转化为对象

var Objarry=["name=李狗蛋","fryNumber=0","sex=男"];

var Obj={};

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

Obj[Objarry[i].split("=")[0]]=Objarry[i].split("=")[1];

}

console.log(Obj);

**弹出结果为true。**

**Boolean：布尔值，true或false；**

**②其他数据类型：array Math date function**

**特殊：NaN:不是数字的number类型的数据。**

**Console.log(typeof(NaN))==>number.**

**打印它的数据类型，结果为number，说明它地的数据类型是number；**

**Array：数组。Math：数学工具对象date：日期类型function：函数**

**判断一个变量是不是数字类型：**

If（typeof（a）==“number”）{

}

# Object类型的方法属性

## 对象的属性与方法

### 对象的形式

Object类型的数据是JS数据家族中一个重要的数据类型，它的格式为{键：值，键：值，。。。。。}

获取对象的某一个键，可以用for循环的方法进行判断，而获取对象的某一个value，可以有两种写法。

1.Obj.a //此时a为Obj的一个key，这样写的时候a不要加引号，代表a是key值

2.Obj[“a”]//此时a可以理解为与Obj的属性a值相同的一个字符串。

Var oa={

a:1,b:2,c:3

};

**空对象：{ }**

**判断一个对象是否为空对象：**

var c={ }；

if(JSON.stringify(c)==”{ }”){

console(“true”);

}

### in操作符与delete：

in 操作符用来判断属性是否存在与对象中。

Let obj={

a:1，

b:2，

}

Let c= 3;

Console.log(a in obj);//true

Console.log(c in obj);//false

delete操作符用来删除对象的某个属性。

Let obj={

a:1,

b:2,

}

delete obj[a];

Console.log(obj);

### hasOwnProperty(key)方法

hasOwnProperty(key)方法也可以判断key属性是否在对象中，返回布尔值boolen。

Let o = new Object();

o.prop = 'exists';

function changeO() {

o.newprop = o.prop;

delete o.prop;}

o.hasOwnProperty('prop'); // 返回 truechangeO();

o.hasOwnProperty('prop'); // 返回 false

### 对象的解构赋值

let node = {

type: "Identifier",

name: "foo"

};

let { type, name } = node;

console.log(type); // "Identifier"

console.log(name); // "foo"

在这段代码中，node.type的值被存储在名为type的变量中；node.name的值被存储在名为name的变量中

let node = {

type: "Identifier",

name: "foo"

},

type = "Literal",

name = 5;// 使用解构来分配不同的值

Let{ type, name } = node;

console.log(type); // "Identifier"

console.log(name); // "foo"

var ar=[0,1,2,3,4,5];

var obj ={

name:'aa',

age:22

};

let {name,age}=obj;

console.log(name,age);

console.log({...obj});//{name:'aa',age:22}

console.log([...ar]);//[0,1,2,3,4,5]

### 对象的合并

假设有多个对象，对象的个数是可枚举的，有限的：

const obj2={a:1,b:2}

const obj3={c:3,d:4}

1. 将每个个对象作为属性单元合并到一个对象中：

跟对象的解构过程相反：{obj2,obj3}

const concatObj={obj2,obj3};

console.log(concatObj);//{obj2:{a:1,b:2},obj3:{c:3,d:4}}

1. 将多个对象的每一项拆开共同合并到一个对象中：

{...obj2,...obj3}

const concatObj2={...obj2,...obj3};

console.log(concatObj);//{a:1,b:2,c:3,d:4}

但是这种方法要考虑一种情况，就是多个对象的key如果有重复的话，后面的会覆盖前面的。

const obj2={a:1,b:2}

const obj3={a:3,d:4}

console.log(concatObj);//{a:3,b:2,d:4}

### 对象的循环遍历

对象也可以for循环。如果一个对象Obj的某个属性的键为key，那么Obj[key]是该属性的value值。

1. for(in)循环

用法：对于一个 baseObj，for(let key in baseObj){

console.log(key,baseObj1[key]);

}

此时key就是键，baseObj[key]是value。

1. Object.keys(Obj).forEach();

用法：Object.keys(Obj).forEach(function(key,value){

//do something

})

该方法接收两个参数key与value，value与Obj[key]相同。

1. jQuery内置的Object遍历方法：

$.each(obj,function(key,value){})方法

也是接收key与value两个参数。

### 将对象转化为数组（只有Chrome与Firefox支持）：

一：

Object.keys(obj)方法：返回一个以对象的key值组成的数组

Object.values(obj)方法：返回一个以对象的value值组成的数组

Object.entries(obj)方法：返回一个以对象的每一个[ “key”，”value“ ](数组)为元素值组成的数组。

var obj={

a:1,

b:2,

c:3,

d:4,

e:5,

f:6,

g:7

}

Key返回对象的每个key值组成的数组，value返回key值对应的value值所组成的数组。

var item1=[];

var item2=[];

console.log(Object.keys(obj));

console.log(Object.values(obj));

console.log(Object.entries(obj));

二：$.each(obj,function(key,value){})方法：

var obj={username:"李狗蛋",age:18,isBoy:true};

$.each(obj,function(key,value){

item1.push(key);

item2.push(value);

})

alert(item1);

alert(item2);

### 对象的拷贝

对于基本数据类型来说，复制一个变量值，本质上就是copy了这个变量。一个变量值的修改，不会影响到另外一个变量。

let val = 123;

let copy = val;

console.log(copy); //123

val = 456; //修改val的值对copy的值不产生影响

console.log(copy); //123

而对于复杂数据类型来说，同基本数据类型实现的不太相同。对于复杂数据类型的复制，要注意的是，变量名只是指向这个对象的指针。当我们将保存对象的一个变量赋值给另一个变量时，实际上复制的是这个指针，而两个变量都指向都一个对象。因此，一个对象的修改，会影响到另外一个对象。

// obj只是指向对象的指针

let obj = {

character: 'peaceful'

};//copy变量复制了这个指针，指向同一个对象

let copy = obj;

console.log(copy); //{character: 'peaceful'}

obj.character = 'lovely';

console.log(copy); //{character: 'lovely'}

同理，在复制一个数组时，所有声明的变量名只是指向这个数组对象的指针；

即使arr=null，也不会影响copy。因此此时的arr变量只是一个指向数组对象的指针。同理，在复制一个函数时，函数名只是指向这个函数对象的指针：

function foo () {

return 'hello world';

};

let bar = foo;

console.log(foo());//'hello world'

foo = null; //foo只是指向函数对象的指针,即使fool变成null也不会对bar造成影响

console.log(bar()); //'hello world'

#### **1.对象的浅拷贝**

在ES6中，Object对象新增了一个assign方法，可以实现对象的浅复制。这里谈谈Object.assign方法的具体用法，因为稍后会分析jQuery的extend方法，实现的原理同Object.assign方法差不多

Object.assign的第一个参数是目标对象，可以跟一或多个源对象作为参数，将源对象的所有可枚举（[[emuerable]] === true）复制到目标对象。这种复制属于浅复制，复制对象时只是包含对该对象的引用。

Object.assign(target, [source1, source2, ...])

1.如果目标对象与源对象有同名属性，则后面的属性会覆盖前面的属性

2.如果只有一个参数，则直接返回该参数。即Object.assign(obj) === obj

3.如果第一个参数不是对象，而是基本数据类型（Null、Undefined除外），会调用对应的基本包装类型

4.如果第一个参数是Null和Undefined，则会报错；如果Null和Undefined不是位于第一个参数，则会略过该参数的复制

要实现对象的浅复制，可以使用Object.assign方法,用法与Array.concat方法类似：

let target = {a: 123};

let source1 = {b: 456};

let source2 = {c: 789};

let obj = Object.assign(target, source1, source2);

console.log(obj); //{a: 123,b: 456,c: 789}

#### 对象的深拷贝

一.JSON.parse(JSON.stringify(obj)是一种对象的深度拷贝

let obj = {

a: 123,

b: {

c: 456,

d: {

e: 789,

}

}

};

let copy = JSON.parse(JSON.stringify(obj));// 对obj对象无论怎么修改，都不会影响到copy对象

obj.b.c = 'hello';

obj.b.d.e = 'world';

console.log(copy); // {a: 123, b: {c: 456, d: {e: 789}}}

1. 对象/数组的扩展（...）

const arr = [1, 2, 3];

let copy = [...arr];

copy = [0, 0, 0];

// arr：[1, 2, 3]，copy : [0, 0, 0]

const obj1 = { a: 1 };

const obj2 = { ...obj1 };

obj1.a = 2;

// obj1 :{a:1} obj2 : { a:2 }

# String类型的方法属性

## 字符串的转换与截取：

parseInt(string, radix)：解析一个字符串或一个数字，返回一个整数。

String：必须，需要解析的字符串。

Radix：可选。表示要解析的数字的基数。该值是介于 2 ~ 36 之间的一个整数。（小数的话转化为其它数字，结果有偏差）

如果省略该参数或其值为 0，有两种情况：

如果String以 “0x” 或 “0X” 开头，将转化为16进制数，否则String将转化为10进制数。

如果该参数为1或该参数大于36，则 parseInt() 将返回 NaN。

.substring()与.substr():

.substring(i)-----------从第i个索引位开始截取它之前的字符串，返回它之后的字符串。

语句名.split(“,”)-----------用逗号,空格，斜杠等分割字符串，返回一个分割后的数组。

indexOf()方法：

用法：Arr.indexOf(item);

indexOf()方法可返回某个指定的字符串或数组值在字符串或数组中首次出现的索引位置，若不在其中则返回-1

Arr.charAt() – 返回指定位置的字符。

语句名.substring(i,j)-----------从第i个索引位开始，截取到第j-1个索引位字符，截取长度为j-i（包括空格）。

语句名.substr(i,j)-----------从第i个索引位开始，截取到第j个字符。  
去除所有空格: str = str.replace(/\s\*/g,"");        
去除两头空格: str = str.replace(/^\s\*|\s\*$/g,"");  
去除左空格： str = str.replace( /^\s\*/, “”);  
去除右空格： str = str.replace(/(\s\*$)/g, "");

去除字符串内的空格：str.replace(/\s/ig,'')；

var s="1223 332 "; console.log(s.replace(/\s/ig,''));

new Date()；--------声明日期时间。

Date .parse(时间格式)；-------将日期时间转化为规定的时间格式。

比如一段话：word1=“你是我的小雅小苹果。”

charAt（i）获取第i个索引字符。

charCodeAt（i）获取第i个索引字符的AscII字符。

World1[i]根据字符串数组的[ ]索引找到第i个字符。

Word1.concat（word2）；链接字符串word1和word2。用“+”也是可以的。也可以用于把一个数组添加到另一个数组。

.trim() 函数用于去除字符串两端的空白字符(空格)。

**注意：**$.trim()函数会移除字符串开始和末尾处的所有换行符，空格(包括连续的空格)和制表符。如果这些空白字符在字符串中间时，它们将被保留，不会被移除。

英文的大小写转换;语句名

语句名..toUpperCase();-----转换成大写字母。

语句名.toLowerCase();------转换成小写字母。

语句名.replace(第i个字符,其他元素)-------将第i个字符换成其他字符。

var num=1.37

Number（）强制转换为数字，把字符串里的数字提取出来，如果是布尔类型，那么转换为数字0/1.如果是undefined，那么结果是NaN。如果是Null那么返回0，如果是字符串，那么转换成NaN，如果是字符串类型的数字，那么转化为数字。

toString()：数字强制转换成字符串。

Num.valueOf()数字转化为字符串。

`${expression}`字符串占位符

在一个模板字面量中，你可以把任何合法的JavaScript表达式嵌入到占位符中将其作为字符串的一部分输出到结果中。占位符${expression}中间可以包含任意的JavaScript表达式。` `里占位符${expression}可无缝拼接字符串。例如下面的例子：

const a = '小明';

const b = `我是${a}`;

console.log(b);// (String)我是小明

注意：parseInt（）parseFlour（）Number（）

## Js对字符串进行编解码：

escape(string)

escape(string) 函数可对字符串进行编码，这样就可以在所有的计算机上读取该字符串。

注意：escape(string)方法不会对 ASCII 字母和数字进行编码，也不会对下面这些 ASCII 标点符号进行编码： \* @ - \_ + . / 。其他所有的字符都会被转义序列替换。

unescape(string)

unescape() 函数可对通过 escape() 编码的字符串进行解码。

注意：unescape()数通过找到形式为 %xx 和 %uxxxx 的字符序列（x 表示十六进制的数字），用 Unicode 字符 \u00xx 和 \uxxxx 替换这样的字符序列进行解码。

encodeURI(string)

encodeURI() 函数可把字符串作为 URI 进行编码。

注意：该方法不会对 ASCII 字母和数字进行编码，也不会对这些 ASCII 标点符号进行编码： - \_ . ! ~ \* ' ( ) ，该方法的目的是对 URI 进行完整的编码，因此对以下在 URI 中具有特殊含义的 ASCII 标点符号，encodeURI() 函数是不会进行转义的：;/?:@&=+$,#

decodeURI(string)

decodeURI() 函数可对 encodeURI() 函数编码过的 URI 进行解码。

# Number类型与Math数学工具

Number(object )方法：将其他类型的值强制转化为Number类型。以下几种是转换的结果：

Boolean类型：false转化为0；true转化为1.

Array类型：空数组转化为0，非空数组转化为1.

Object类型：空对象转化为0，非空对象转化为1.

String类型：只含有纯阿拉伯数字的字符串转化为Number类型，纯数字与其他字符串并存时转化为NaN。

toFixed(n) 方法可把 Number 四舍五入为指定小数位数的数字，n为位数。

用法：NumberObject.toFixed(num).例如：

var num = new Number(13.37);

document.write (num.toFixed(1));

isNaN（）判断一个变量是否是一个非数字类型的变量，返回一个布尔值。 如果是非数字类型，那么返回true；否则返回false；

isNaN（123）返回false；isNaN（“123”）也返回false；只有当变量值全为纯字符串时才为true。比如isNaN（“你好”）、isNaN（“adb”）、isNaN（“123adb”）

王炸：

eval(string);eval() 函数可计算某个字符串，并执行其中的的 JavaScript 代码。

**比如说eval（“3/4”）=0.75.**

乘方和开方

Console.log（Math.pow(3,4)）

Math.Pow()乘方函数

.Pow（a，b）表示a的b次方。

Console.log (Math.Pow（3，math.pow(3,4)）)

Console.log (Math.Pow（math.pow(3,4),3）)

圆周率：Math.PI，在js中表示180°对应的弧度。

角度：

n°(角度)的转弧度的js的写法n\*Math.PI/180 。

Js三角函数：

Jiaodu de

Math.sin(n°)

Math.cos(n°)

Math.tan(n°)

自然常数：Math.E（e = 2.718281828459）

开根号：

Math.sqrt()开根号

Math.abs():取绝对值

Math.ceil(num)----天花板，要有小数，就进1

Math .floor(num)； 地板，只取整数部分，和parseInt（）功能一样，返回的数字类型为Number类型。

Math .round(num)；/num.toFixed（）四舍五入，返回的数字类型是String类型。

num.toExponential()转化成指数形式

Math.max(a,b,c);取最大值

Math.min(a,b,c);取最小值。

Math .random();取闭区间[0,1]之间小数点后的位数长达16位的随机数。这个非常有用，用于抽奖，用于抢票，作弊，很叼的。

例如：字母+数字组合：

const num=["1","2","3","4","5","6","7","8" ,"9","0" ];

const abc=["a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p","q","r","s","t","u","v","w","x","y","z" ];

private initData(){

var suiji1 = Math.ceil(Math.random() \* (this.abc.length - 1));

var suiji2 = Math.ceil(Math.random() \* (this.abc.length - 1));

var suiji3 = Math.ceil(Math.random() \* (this.num.length - 1));

var suiji4 = Math.ceil(Math.random() \* (this.num.length - 1));

var suiji5 = Math.ceil(Math.random() \* (this.num.length - 1));

var suiji6 = Math.ceil(Math.random() \* (this.num.length - 1));

var suiji7 = Math.ceil(Math.random() \* (this.num.length - 1));

//this.zhanghao=Math.random().toString(36).substr(2);

this.ZC\_ZhangHao.text= this.abc[suiji1]+this.abc[suiji2]+this.num[suiji3]+this.num[suiji4]+this.num[suiji5]+this.num[suiji6]+this.num[suiji7];

# JS面向对象编程

## this的指向

一：在原生JS的this指向：

a：在全局范围内，this是指window。

b：在对象里或函数里，this指向调用该语句的该对象（直接对象）或该函数（直接函数，PS：函数本身也是一种对象）。

二：在框架内的this指向：

1.箭头数里的 this 指向声明函数的地方，也就是组件class内，所以不用 bind(this)  
2.普通函数里 this 指向调用它的地方，在没有bind 的时候指向函数，而不指向组件class，所以要想让 this 指向组件class必须使用 bind()。

## 二：封装（prototype与\_proto\_）

### prototype的定义

javascript中的每个对象都有prototype属性，Javascript中对象的prototype属性的解释是：返回对象类型原型的引用。Prototype指向它的原型对象。

当我们用obj.xxx访问一个对象的某个属性时，JavaScript引擎先在当前对象上查找该属性，如果没有找到，就到其原型对象上找，如果还没有找到，就一直上溯到原型对象的Object.prototype对象，最后，如果还没有找到，就只能返回undefined。

### prototype的作用

prototype的作用是为原型Object声明一些公共的方法来抛出给实例。

### prototype里的this

Prototype里的this指向该函数的调用者，也就是指向new出来的实例。

### \_\_proto\_\_属性：

一句简单的话概括\_\_proto\_\_属性：Son.\_\_proto\_\_==Father.prototype.

也就是说，new 实例的子对象本身是有一个\_\_proto\_\_属性的，该属性继承自它的原型的prototype属性。

### **ES6的类： Class关键字**

用class关键字来定义一个Object类：

class Point {

constructor(x, y) {//声明自己的私有属性，无法对这些属性进行扩展

this.x = x;

this.y = y;

this.a=1;

this.b=2;

this.sum=function(){

return this.a+this.b

}

}

//声明prototype里的公有方法。

toString() {

return '(' + this.x + ', ' + this.y + ')';

}

}

注意：定义“类”的方法的时候，前面不需要加上function这个关键字，直接把函数定义放进去了就可以了。另外，方法之间不需要逗号分隔，加了会报错。

构造函数的prototype属性，在ES6的“类”上面继续存在。事实上，类的所有方法都定义在类的prototype属性上面。

class Point {

constructor(){

// 声明自己的 私有属性

}

//prototype里的方法：

toString(){

// ... }

toValue(){

// ... }

}

// 等同于

Point.prototype = {

toString(){},

toValue(){}

};

类的内部所有定义的方法，都是不可枚举的（non-enumerable）。这一点与ES5的行为不一致。

constructor方法是类的默认方法，通过new命令生成对象实例时，自动调用该方法。一个类必须有constructor方法，如果没有显式定义，一个空的constructor方法会被默认添加。constructor方法默认返回实例对象（即this），完全可以指定返回另外一个对象。

可以通过实例的\_\_proto\_\_属性为Class添加方法。

var p1 = new Point(2,3);var p2 = new Point(3,2);

p1.\_\_proto\_\_.printName = function () { return 'Oops' };

p1.printName() // "Oops"

p2.printName() // "Oops"

var p3 = new Point(4,2);

p3.printName() // "Oops"

### constructor属性

　　prototype对象有一个constructor属性，默认指向prototype对象所在的构造函数。

要注意的是，prototype是构造函数的属性，而constructor则是构造函数的prototype属性所指向的那个对象，也就是原型对象的属性。注意不要混淆。

constructor属性的作用：

a：抛出一些默认的私有属性给实例

function A(){};

var a=new A();

console.log(a.constructor.name===A) //true

console.log(a.constructor.name===Array) //false

b：从实例新建另一个实例

function A() {};

 var a = new A();

var b = new a.constructor();

console.log(b instanceof A); //true

上面代码中，a是构造函数A的实例对象，可以从a.constructor间接调用构造函数。

### **Static-静态方法**

Static方法是一个class类私有的方法，它声明的方法用于该类直接调用，而不能被子实例继承。

class Book {

constructor(a,b) {

this.a=a;

this.b=b;

}

static open() {

return console.log(12313);

}

}

let mybook=new Book(1,'2');

console.log(mybook);

Book.open();//12313

mybook.open();//error

### instanceof运算符

instanceof运算符返回一个布尔值，表示指定对象是否为某个构造函数的实例。

用法：a instanceof(Object);

function A() {};

var a = new A();

console.log(a instanceof A); //true

因为instanceof对整个原型链上的对象都有效，所以同一个实例对象，可能会对多个构造函数都返回true。

function A() {};

 var a = new A();

console.log(a instanceof A); //true

console.log(a instanceof Object); //true

　注意，instanceof对象只能用于复杂数据类型（数组，对象等），不能用于简单数据类型（布尔值，数字，字符串等）。

var x = [1];

 var o = {};

 var b = true;

 var c = 'string';

 console.log(x instanceof Array); //true

console.log(o instanceof Object); //true

 console.log(b instanceof Boolean); //false

console.log(c instanceof String); //false

此外，null和undefined都不是对象，所以instanceof 总是返回false。

console.log(null instanceof Object); //false console.log(undefined instanceof Object); //false

利用instanceof运算符，还可以巧妙地解决，调用构造函数时，忘了加new命令的问题。

function Keith(name,height) {

 if (! this instanceof Keith) {

 return new Keith(name,height);

 }

 this.name = name;

 this.height = height;

}

　　上面代码中，使用了instanceof运算符来判断函数体内的this关键字是否指向构造函数Keith的实例，如果不是，就表明忘记加new命令，此时构造函数会返回一个对象实例，避免出现意想不到的结果。

　 因为限于篇幅的原因，暂时介绍到这里。

　　我会在下次的分享中谈谈原型（prototype）对象的一些原生方法，如Object.getPrototypeOf()，Object.setPrototypeOf()等，并且介绍获取原生对象方法的比较。

## 取值函数（getter）和存值函数（setter）

与 ES5 一样，在“类”的内部可以使用get和set关键字，对某个属性设置存值函数和取值函数，拦截该属性的存取行为。

class MyClass {

constructor() {}

get prop() {

return 'getter';

}

set prop(value) {

console.log('setter: '+value);

}}

let inst = new MyClass();

inst.prop = 123;

// setter: 123

inst.prop

// 'getter'

## 三：继承与扩展（prototype）

当用构造函数的方法来创建一个对象的时候，这个构造函数将会把该对象的属性prototype赋给新new出来的子对象的内部属性\_\_proto\_\_。这个\_\_proto\_\_被这个对象用来查找它的属性。

JS创建对象的方法有三种：

a 类方法b 对象方法c 原型方法

//构造函数的方法创建一个对象:

function Student(name) {

this.name = name;

this.hello = function () {

alert('Hello, ' + this.name + '!');

}

}

设置对象的原型Prototype：

Student.prototype={

IntroduceChinese:function(){

alert("我的名字是"+this.name);

},

age:18,

}

用new方法实例化一个Student对象，此时，new的新对象就继承了原Student对象内置的所有属性，并且可以在new的子例中可以更改内置属性。

var xiaoming=new Student("小明");

console.log(xiaoming.name);

xiaoming.name="小红";//xiaoming的name属性可以被更改

xiaoming.hello();

原型继承：xiaoming不仅继承了函数内部的对象属性，更继承了Student的prototype属性。xiaoming的IntroduceChinese属性继承自Student的prototype，并且继承自原型的prototype也可以被更改：

xiaoming.IntroduceChinese();

console.log("原型age为",xiaoming.age)

xiaoming.age=20;

xiaoming.IntroduceChinese();

console.log("实例子对象age更改为",xiaoming.age);

//打印输出子实例的\_\_proto\_\_属性，它就继承自它的原型的prototype属性。

console.log(xiaoming.\_\_proto\_\_);结果为（一个对象）：

Object {

IntroduceChinese:function(){

alert("我的名字是"+this.name);

},

age: 18

}

该\_\_prototype\_\_属性与父类对象的prototype属性完全相同。

我们用js的prototype属性可以做一些自己的函数封装，可以对Object，Array，String，Number类型的数据进行自己的实例化封装函数；

Array.prototype.con=function(){

console.log(this);

return this.concat([1,2,3,4]);

}

var arr0=[5,6,7,8];

console.log(arr0.con());

这里的this指向用prototype属性创建的方法的调用者，也就是你实例出来的子类。

使用extends关键字实现类之间的继承。这比在ES5中使用继承要方便很多。

//定义类父类

class Parent {

  constructor(name,age){

    this.name = name;

    this.age = age;

  }

  speakSometing(){

    console.log("I can speek chinese");

  }

}

//定义子类，继承父类

class Child extends Parent {

  constructor(name,age){

    //不调super()，则会报错 this is not defined

    //必须调用super

    super(name,age);

  }

  coding(){

    console.log("coding javascript");

  }

}

var c = new Child("job",30);

//可以调用父类的方法

c.speakSometing(); // I can speek chinese

子类必须在constructor方法中调用super方法，否则新建实例时会报错(this is not defined)。这是因为子类没有自己的this对象，而是继承父类的this对象，然后对其进行加工。如果不调用super方法，子类就得不到this对象。

## 四：多态

多态就是一个变量可以接受多种类型的数据。

## 获取日期、时间对象

通过实例化一个日期对象来获取当前的时间

var date=new Date()；声明一个日期对象

.getFullYear()获取完整年份（浏览器都支持）

.getMonth()获得月份

var date1=date.getDate()获得当日的日期0-31；

var date2=date.getDay()；获得星期0-6，从星期日开始算

.getHours()获取小时0-23

.getMinutes()获得分钟0-59

.getSeconds()获取秒数0-59

.getMillisecond()；获取毫秒0-1000；

New Date().getTime()；或者用Date.parse(new Date())获取指定日期对象的累计毫秒数（据1970年1月1日午夜）目前总共有13位。

Date.parse(datestring)方法：解析一个标准类型的日期字符串，并返回该日期与 1970 年 1 月 1 日午夜之间相差的毫秒数(时间戳)。

创建一个日期对象：

Var date1=newDate（“年数/月数/日数 小时数：分数：秒数”）；

Var date1=new Date(“2017/04/19 17：30：00”);

Var date2=new Date(“2017-04-19 17：30：00”);

得到某个时间戳的官方日期字符串：

New Date.getTime().toLocaleDateString()；

得到某个时间的官方时间字符串：

New Date.getTime().toLocaleTimeString()；

使用 ISO 标准返回 Date 对象的字符串格式:

new Date().toISOString();

获取当前时间的时间戳，有三种方法

第一种方法：

var timestamp = Date.parse(new Date());

结果：1280977330000

第二种方法：

var timestamp = (new Date()).valueOf();

结果：1280977330748

第三种方法：

var timestamp = new Date().getTime();

结果：1280977330748

第一种方法获取的时间戳是把毫秒改成000显示，后两种方法获取了当前毫秒的时间戳。三种结果的类型都是number。

倒计时：

var second = parseInt((endTime.getTime() - nowTime.getTime()) / 1000);

或者是

var second =Math.floor((endTime.getTime() - nowTime.getTime()) / 1000);// 用将来的时间毫秒 - 现在的毫秒 / 1000 得到的 还剩下的秒 可能除不尽 取整

// 一小时 3600 秒

// second / 3600 一共的小时数 /24 天数

var d = parseInt(second / 3600 / 24); //天数

//console.log(d);

或者是

var d = Math.floor(second / 3600 / 24); //天数

//console.log(d);

// 获取剩下的小时

var h = parseInt(second / 3600 % 24)

// console.log(h);

或者是

var h = Math.floor(second / 3600 % 24) // 小时

// console.log(h);

var m = parseInt(second / 60 % 60);

//console.log(m);

或者是

var m =Math.floor(second / 60 % 60);

//console.log(m);

当前的秒：

var s = parseInt(second % 60); //

console.log(s);

或者是

var s = Math.floor(second % 60);

console.log(s);

## 定时器/计数器的声明与调用：

setTimeout（函数名，毫秒数）到数执行，执行1次；

SetInterval（函数名，毫秒数）间隔执行，反复执行；

ClearTimeout（定时器名），clearIntevcal（定时器名）清除定时器；

第一种声明方式：间接声明

var 定时器名=Window.setInterval(“函数名”，记时时间)；

Function 函数名（）{

}

第二种：直接声明：

var 定时器名=Window.setInterval(function（）{

}，记时时间)；

或者这样：

Var myInterval；

myInterval=Window.setInterval(“函数名”，记时时间)；

Function 函数名（）{

}

这样的好处是可以随时调用或清除。

以指定时间间隔调用函数，反复循环调用，没有终止。下面是函数体。

setTimeout（函数名，毫秒数）到数执行，执行1次

SetInterval（函数名，毫秒数）间隔执行，反复执行；

ClearTimeout（定时器名），

clearInterval（定时器名）清除定时器；

在实际应用中，如果定时器需要多次重复调用，那么每次调用之前必须先清除，再重新调用。

声明定时器/计数器，用于设置执行函数语句。这个计时时间单位是毫秒，1s=1000ms。它不是一个单线程的方法。

setInterval的特点：

1：全局性：其内的作用域是全局作用域。

2：异步操作：setInterval（）是异步进程。

window.SetInterval(“函数（）”，1000)；它调用的函数永远是全局函数，而不是局部函数，说白了必须是window对象下面的函数。我们在window.onload这个事件里面弄定义的函数，它的作用域只是onlod事件（window对应的事件有很多，如window.alert,window.console.log等），所以SetInterval无法找到并调用。

注意：如果在某个子作用域中调用了setInterval（）方法，那么在在此时，setInterval（）方法里的函数作用域就是全局的了。如果还想调用子作用域的this，就要转换作用域，就要在setInterval（）方法外面写：const \_this=this；在setInterval（）方法里面调用\_this.

setInterval与for循环配合使用基本思想：

在setInterval内部设置一个变量（i=0/1）；让i自增，当i值增大到某一个数时，i置零；这样这个setInterval就变成了一个for循环；在setInterval内部添加一个for循环（var j=0；j<...），然后判断i与j是否相等，然后再添加事件。

一个for循环同时处理多个事件的基本思想（层级菜单，层级下拉列表）：

for循环结束之后，通过把数组名元素的索引值赋值：数组名[i].index=i；

第一个事件函数用 class名[i];后面的事件函数用class名[this.index];this指向的是第一个函数的第i个索引对象。

排他思想：两个for循环嵌套：

外部for循环找到当前事件对象的索引值；a[i].index=i;内部for循环清空所有对象的抓状态。

总结for循环的几个用途：

循环计数，循环赋值（添加事件），清空状态（CSS属性/innerHTML状态）

Window.clearInterval（方法名）清除执行语句。也不是一个单线程的语句。这个也很操蛋，如果在一个局部区域内使用了这个语句，那么不仅当前的语句被清除，往下的所有的语句都被清除了。因为window是单线程的。如果把它放到全局区域里，那么还是因为window是单线程原因，当前的循环清除就失效了，因为当浏览器执行完所有的语句之后，还会执行当前的循环体，实际上是清除了一次，又执行了一次，相当于不清除。

浏览器的单线程性：如果浏览器看到有阻塞浏览气执行的代码，那么它不会立即执行，而是先从上到下执行完所有的操作之后，才去执行该语句。

setInterval，setTimeout与ajax请求一样，是异步的，所以在这些函数体内执行自己的语句，不会影响函数外代码的执行，也就是说不会影响浏览器的主进程。所以浏览器会同时开辟另一个进程完成这些异步方法。异步方法执行完之后的结果会变成同步进程的状态。

## 堆和栈：

栈的空间是固定的，而堆的空间是不固定的，当用户定义一个变量为一个数字时候，内存在堆里开辟一个确定的空间来存放该变量的值，当用户自定义（动态定义）变量为字符串的时候，内存在栈里开辟一个地址来引用该字符串，再在堆空间里通过地址来找到该字符串。

除了自定义对象之外，其他类型的数据都是保存在栈里。栈内存的内存开辟都是固定大小。堆空间内存大小是动态生成的，所以自定义的面向对象都是堆和栈同时贮存的。

内存在编程过程中其实逻辑上分为两个部分。

栈空间是通过变量的名称开辟了一个空间，并且里面放入了变量的值。比如说var a=”张三”; 就是在栈内存上开辟了一个空间，这个空间的内存地址是ofx0101101,但是内存地址不好记忆，因为它是二进制的，所有我们的变量名称可以作为这个地址空间的别名，方便记忆。在这个空间上存贮的值是字符串“张三”。

堆：动态存贮的是自定义对象的数据的。里面实实在在的是对象的值，通过栈内存开辟的空间变量里面的存放的地址来引用。

引用类型与值类型

引用类型就是指的贮存方式由栈开辟空间贮存的是地址，改名改的是堆空间内存的值。

赋值是copy一个变量，引用类型赋值是两个变量同时拥有堆里的值。

地址与值。

面向对象的语法定义：就是对象将类实例化。

引用函数来创造一个类，再用对象将该类实例化：

例如：

function Cat(name,color){

return{

name:name,

color:color

}

}

var cat1=new Cat( "Tom","Flower");

alert(cat1.name+cat1.color);

# Window对象的方法

1：浏览器是一个对象，对象名成为window。

任何一个对象都有一个方法和属性。

函数就是function定义的方法，属性就是定义特有变量。

一个对象都有其属性。

只要是在script标签下进行的所有编程，其对象都是window。

重要语法：window.alert（window.弹出内容）。

Window对象的函数有两种：一种是window自带的函数，另一种是程序猿定义的函数。任何在window下定义的变量，都属于window下面的变量定义。

当一个变量出现在局部范围内，并且没有在全局范围内规定，那么window会饮食地定义该变量为全局变量，并且其值（属性）为undefined。所以如果在script标签下有一个未命名的变量，那么它就被window定义了。

四：关于window默认的几个重要对象。

Window.alert()

Window.prompt()接受用户输入

Window.prompt(“提示用户输入的语言”,”值”)；通常这个值为空，也就是不设置这个值。

Window.confirm()选择对话框状态，返回一个布尔值true/false。

Window.print();打印

Window.close( )关闭浏览器页面

Window.open(“http://www.baidu.com”)；打开浏览器页面或打开本页面的一个标签，比如登陆，下载等（这个很操蛋，慎用，忌用，容易被浏览器拦截）

navigator对象：用于提供ｗｅｂ服务器的信息

比如：window.navigator.userAgent：获取当前浏览器的版本信息。

screen对象：可以用screen对象获取某些关于用户屏幕信息。

screen对象通常包含下列属性.

       availHeight : 窗口可以使用的屏幕的高度(以像素计),其中包括操作系统元素（如windows工具栏）需要的空间

       availWidth: 窗口可以使用的屏幕的宽度(以像素计)

       colorDepth: 用户表示颜色的位数,大多数采取32位计.

       height : 屏幕的高度

       width : 屏幕的宽度

History对象：用于保存浏览器显示过的URL

## location对象：

window.location：获取当前页面的地址。不光可以获取，还可以赋值。例如：window.location(.href)='http://www.baidu.com/'

BOM之location对象

Location专门来处理URL解析的

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 案例 | 说明 | 备注 |
| Hash | “#contents””页面内的锚节点，会在地址栏中显示 | 返回URL中的hash，如过不包含则返回空字符串 |  |
| Host | “www.BAIDU.com：80” | 端口号和服务器名称 |  |
| Href |  | 当前加载的URL地址 | 跳转 |
| Pathname | “D/Web/1.html” | 返回url中的目录和文件名称 |  |
| Port | “8080” | 返回端口号 |  |
| Protocal | hppt/https | 返回页面使用的协议，通常是hppt/https |  |
| Search | “？Name=age” | 返回参数字符串，问号？开头 | 常用 |

window.location.protocol 返回所使用的 web 协议（http:// 或 <https://）>

window.location.hostname（document.domain） 返回当前的web域名

window.location.port 返回 web 端口 （80，或 443）

window.location.host：web 协议+当前域名+端口号，如果没有显示端口号就是默认80

window.location.pathname 返回当前页面的路径和文件名

window.location.href：当前页面完整URL（web 协议+当前域名+端口号+服务器页面地址+参数）

## Window.location.href=””

跳转网页加载地址，后跟域名。用于网页跳转到下一个页面。这是系统自己定义的。

1. 在当前窗口打开百度,并且使URL地址出现在搜索栏中.

window.open("<http://www.baidu.com/>", "\_search");

window.open("<http://www.baidu.com/>", "\_self");

Window.location.href="<http://www.baidu.com/>";

a href="<http://www.baidu.com/>"

三者的功能都是一样的，都是在本网页上打开新的网页链接。

2. 在一个新的窗口打开百度

window.open("<http://www.baidu.com/>", "\_blank");

3. 打开一个新的窗口,并命名为"hello"

window.open("", "hello");

另外, open函数的第二个参数还有几种选择:

\_top : 如果页面上有framesets,则url会取代framesets的最顶层, 即, 如果没有framesets, 则效果等同于\_self.

\_parent: url所指向的页面加载到当前frame的父亲, 如果没有则效果等同于\_self.

\_media : url所指向的页面加载到Media Bar所包含的HTML代码区域中.如果没有Media Bar则加到本身.

Window.history.go(“页面地址”)跳转到指定的页面。go里面是数字。.。。。1-，-2，-1，1，2，3。。。负数代表上页，正数代表下页。

Window.history.back(“上页页面地址”)跳回上页。

Window.history.forward(“下一页地址”)跳到下页。

## history.back(-1)和history.go(-1)的区别

history.back(-1):直接返回当前页的上一页，数据全部消息，是个新页面

history.go(-1):也是返回当前页的上一页，不过表单里的数据全部还在

history.back(0) 刷新 history.back(1) 前进 history.back(-1) 后退

window.location.reload();    //刷新

window.history.go(1);         //前进

window.history.go(-1);        //返回+刷新

window.history.forward();  //前进

window.history.back();       //返回

window.navigator.userAgent：获取当前浏览器的版本信息。

window.location：获取当前页面的地址。不光可以获取，还可以赋值。例如：window.location(.href)='http://www.baidu.com/'

$(window).resize(function() { }）当浏览器窗口大小改变时的函数

$(dindow).scrollTop()/Left():据页面已经向上/向左卷动的距离。

## Event对象与dom对象的宽高

DOM事件可以传一个event对象，这个event对象就继承自window。

document.element.onclick=function(event){

const event= event ||wndow.evnet;

}

//获得相对于body定位的横标值；

x=event.clientX

//获得相对于body定位的纵标值；

y=event.clientY

当前元素可见区域宽： document.element.clientWidth  
当前元素可见区域高： document.element.clientHeight

可见区域，这个好理解，就是在屏幕内的都算，出去卷去的宽高之外的可见宽高。  
当前元素的宽度： document.element.offsetWidth (包括边线的宽)  
当前元素的高度： document.element.offsetHeight (包括边线的高)

offsetWidth与style.widtht的区别是：

1.若对象的宽度设定值为百分百宽度，无论页面变大或变小，style.width都返回此百分比；而offsetWidth则返回页面中对象的宽度值而不是百分比。

2.如果没在行内style里或style标签里设置height或width；那么style.width返回值为“”，但是offsetWidth 返回值是元素本身初始化的宽度。  
网页正文全文宽： document.body.scrollWidth  
网页正文全文高： document.body.scrollHeight  
网页被卷去的高： document.body.scrollTop  
网页被卷去的左： document.body.scrollLeft

scrollHeight - scrollTop = clientHeight：当这两个条件成立时，也就代表垂直滚动条走到底了

scrollWidth - scrollLeft = clientWidth：当这两个条件成立时，也就代表水平滚动条走到底了  
网页正文部分上： window.screenTop  
网页正文部分左： window.screenLeft  
屏幕分辨率的高： window.screen.height  
屏幕分辨率的宽： window.screen.width  
屏幕可用工作区高度： window.screen.availHeight  
屏幕可用工作区宽度： window.screen.availWidth

# JS复杂数据结构

## 栈——Stack

Js里的栈是一种遵循后进先出(LIFO)原则的有序集合。栈的添加新元素与删除元素都在某一端，称为栈顶。可以理解为一摞砖头，从最上面添加砖头，从最上面移除砖头。

### 创建栈，以数组为例：

function Stack(){

let items = [];

}

### 给栈声明一些属性方法：

由于栈只遵循后入先出的规则，那么就只能用push（）和pop（）两种方法。

1.push(item):往栈里添加元素到栈的最后一项(栈顶)

2.pop():删除栈的最后一项，同时返回被移除的元素。

3.peek():找到栈顶的元素。

4.isEmpty():判断这个栈里有没有元素(是否为空)；如果为空，返回true，否则返回false。

5.clear():把该栈清空。

6.size():返回该栈的元素个数(类似于数组的length)

push()方法:

this.push= function(item){

items.push(item);

}

pop()方法:

this.pop=function(){

return items.pop()

}

Peek()方法:

This.peek=function(){

Return items[items.length-1]

}

isEmpty()方法:

This.empty=function(){

return items.length==0;//return items.length==0?true:false;

}

size()方法:

This.size=function(){

return items.length;

}

clear()方法:

This.clear=function(){

items=[];

}

注意：以上的方法包括items都是栈的私有属性，也称为固有属性，可以直接访问items。如果要写一些公共方法或者不需要访问items的方法或属性，那么就要用到prototype属性。写法为：

Stack.prototype.ok= function(){

alert(“ok”);

}

### 实例化：

Let stack = new Stack()

### 用ES6 class来声明Stack类

Class Stack {

Constructor(){

This.items=[];

}

push(item){

this.items.push(item);

}

//其他方法

}

用法：

let st=new Stack();

console.log(st);//是一个该类的实例化 {"items":[]}

st.push(1111);

console.log(st);//{"items":[1111]}

这种class类声明的类就不区分私有属性还是共有属性了，new出来的实例都可以访问。

## 队列——Queue

队列是遵循FIFO（First in first out先进先出，也称为先来先服务）原则的一组有序的项。队列在尾部添加新元素，并从顶部移除元素。最新添加的元素必须排在队列的末尾。

### 创建队列

function Queue(){

let items = [];

}

### 给队列声明一些属性方法：

由于栈只遵循先入先出的规则，那么就只能用shift（）和pop（）两种方法。

1.enqueue(item):往队列里添加元素到栈的最后一项(栈顶)

2.dequeue():删除队列的最后一项，同时返回被移除的元素。

3.Front():找到队列第一个元素。

4.isEmpty():判断这个队列里有没有元素(是否为空)；如果为空，返回true，否则返回false。

5.clear():把该队列清空。

6.size():返回该队列的元素个数(类似于数组的length)

queue()方法:

this.enqueue= function(item){

items.push(item);

}

dequeue()方法:

this.pop=function(){

return items.shift()

}

Front()方法:

This.peek=function(){

Return items[0]

}

isEmpty()方法:

This.empty=function(){

return items.length==0;//return items.length==0?true:false;

}

size()方法:

This.size=function(){

return items.length;

}

clear()方法:

This.clear=function(){

items=[];

}

### 实例化：

Let queue= new Queue();

### 用ES6 class来声明Queue类

class Queue{

constructor(){

this.items=[];

}

queue(item){

this.items.push(item);

}

//其他方法

}

### 优先队列

function PriorityQueue(){

let items = [];

function QueueElement(element,priority){

this.element=element;

this.priority=priority;

};

this.enqueue=function(element,priority){

let queueElement =new QueueElement(element,priority)；

let added=false;

for(let i=0;i<items.length;i++){

if(queueElement .priority<items[i].priority){

items.splice(i,0,queueElement);

added=true;

break;

}

}

if(!added){

items.push(queueElement);

}

};

this.print=fuction(){

for(let i=0;i<items.length;i++){

console.log(`${items[i].element}-${items[i].priority}`)

}

}

//其他方法和默认Queue实现相同

}

## 链表——LinkedList

链表是一种物理存储单元上非线性、非连续性的数据结构（它在数据逻辑上是线性的），它的每个节点由两个域组成：数据域和指针域。数据域中存储实际数据，指针域则存储着指针信息，指向链表中的下一个元素或者上一个元素。正是由于指针的存在，链表的存储在物理单元是非连续性的。

      链表的优点和缺点同样明显。和线性表相比，链表在添加和删除节点上的效率更高，因为其只需要修改指针信息即可完成操作，而不像线性表（数组）那样需要移动元素。同样的，链表的长度在理论上也是无限的（在存储器容量范围内），并可以动态变化长度，相比线性表优势很大。 相应的，由于线性表无法随机访问节点，只能通过指针顺着链表进行遍历查询来访问，故其访问数据元素的效率比较低。下面是JS部分

这里面封装了的常用方法及描述：

append(element)   向链表尾部添加结点element

insert(position,element)  向位置position处插入结点element

removeAt(position)  按照索引值position删除结点

remove(element)  搜索并删除给定结点element

remove()  删除链表中最后一个结点

indexOf(element) 查找并返回给定结点element的索引值

isEmpty()  判断链表是否为空

size()  获取链表长度

toString()  转换为字符串输出

getHead() 获取头结点

getTail()  获取尾结点

### 单向链表：

function LinkedList(){

/\*节点定义\*/

var Node = function(element){

this.element = element; //存放节点内容

this.next = null; //指针

}

var length = 0, //存放链表长度

head = null; //头指针

this.append = function(element){

var node = new Node(element),

current; //操作所用指针

if (!head){

head = node;

}else {

current = head;

while(current.next){

current = current.next;

}

current.next = node;

}

length++;

return true;

};

this.insert = function(position, element){

if (position >= 0 && position <= length) {

var node = new Node(element),

current = head,

previous,

index = 0;

if(position === 0){

node.next = current;

head = node;

}else{

while(index++ < position){

previous = current;

current = current.next;

}

node.next = current;

previous.next = node;

}

length++;

return true;

}else{

return false;

}

};

this.removeAt = function(position){

if(position > -1 && position < length){

var current = head,

previous,

index = 0;

if (position === 0) {

head = current.next;

}else{

while (index++ < position){

previous = current;

current = current.next;

}

previous.next = current.next;

};

length--;

return current.element;

}else{

return null;

}

};

this.remove = function(element){

var current = head,

previous;

if(element === current.element){

head = current.next;

length--;

return true;

}

previous = current;

current = current.next;

while(current){

if(element === current.element){

previous.next = current.next;

length--;

return true;

}else{

previous = current;

current = current.next;

}

}

return false;

};

this.remove = function(){

if(length < 1){

return false;

}

var current = head,

previous;

if(length == 1){

head = null;

length--;

return current.element;

}

while(current.next !== null){

previous = current;

current = current.next;

}

previous.next = null;

length--;

return current.element;

};

this.indexOf = function(element){

var current = head,

index = 0;

while(current){

if(element === current.element){

return index;

}

index++;

current = current.next;

}

return false;

};

this.isEmpty = function(){

return length === 0;

};

this.size = function(){

return length;

};

this.toString = function(){

var current = head,

string = '';

while(current){

string += current.element;

current = current.next;

}

return string;

};

this.getHead = function(){

return head;

}

}

### 单向循环链表

在单链表的基础上，将尾节点的指针指向头结点，就构成了一个循环链表。环形链表从任意一个节点开始，都可以遍历整个链表。

function CircularLinkedList(){

var Node = function(element){

this.element = element;

this.next = null;

}

var length = 0,

head = null;

this.append = function(element){

var node = new Node(element),

current;

if (!head) {

head = node;

node.next = head;

}else{

current = head;

while(current.next !== head){

current = current.next;

}

current.next = node;

node.next = head;

};

length++;

return true;

};

this.insert = function(position, element){

if(position > -1 && position < length){

var node = new Node(element),

index = 0,

current = head,

previous;

if (position === 0) {

node.next = head;

head = node;

}else{

while(index++ < position){

previous = current;

current = current.next;

}

previous.next = node;

node.next = current;

};

length++;

return true;

}else{

return false;

}

};

this.removeAt = function(position){

if(position > -1 && position < length){

var current = head,

previous,

index = 0;

if (position === 0) {

head = current.next;

}else{

while (index++ < position){

previous = current;

current = current.next;

}

previous.next = current.next;

};

length--;

return current.element;

}else{

return null;

}

};

this.remove = function (element){

var current = head,

previous,

indexCheck = 0;

while(current && indexCheck < length){

if(current.element === element){

if(indexCheck == 0){

head = current.next;

length--;

return true;

}else{

previous.next = current.next;

length--;

return true;

}

}else{

previous = current;

current = current.next;

indexCheck++;

}

}

return false;

};

this.remove = function(){

if(length === 0){

return false;

}

var current = head,

previous,

indexCheck = 0;

if(length === 1){

head = null;

length--;

return current.element;

}

while(indexCheck++ < length){

previous = current;

current = current.next;

}

previous.next = head;

length--;

return current.element;

};

this.indexOf = function(element){

var current = head,

index = 0;

while(current && index < length){

if(current.element === element){

return index;

}else{

index++;

current = current.next;

}

}

return false;

};

this.isEmpty = function(){

return length === 0;

};

this.size = function(){

return length;

};

this.toString = function(){

var current = head,

string = '',

indexCheck = 0;

while(current && indexCheck < length){

string += current.element;

current = current.next;

indexCheck++;

}

return string;

};

}

### 双向链表

单向链表只能向着一个方向遍历链表节点，而在节点指针域中增加了前向指针的双向链表，则可以向着两个方向遍历节点。这使得双向链表也可以在任何一个节点遍历整个链表。

function DoublyLinkedList() {

var Node = function(element) {

this.element = element;

this.next = null;

this.prev = null;

};

var length = 0,

head = null,

tail = null;

this.append = function(element){

var node = Node(element),

current,

previous;

if(!head){

head = node;

tail = node;

}else{

current = head;

while(current){

previous = current;

current = current.next;

}

node.next = current;

current.prev = node;

previous.next = node;

node.prev = previous;

}

length++;

return true;

}

this.insert = function(position,element){

if(position > -1 && position < length){

var node = new Node(element),

current = head,

previous,

index = 0;

if(position === 0){

if(!head){

head = node;

tail = node;

}else{

node.next = current;

current.prev = node;

head = node;

}

}else if (position === length -1){

current = tail;

current.next = node;

node.prev = current;

}else {

while(index++ < position){

previous = current;

current = current.next;

}

node.next = current;

previous.next = node;

current.prev = node;

node.prev = previous;

}

length++;

return true;

}else{

return false;

}

};

this.removeAt = function(position){

if(position > -1 && position < length){

var current = head,

index = 0,

previous;

if (position === 0) {

head = current.next;

if(length === 1){

tail = null;

}else{

head.prev = null;

}

}else if(position === length - 1){

current = tail;

tail = current.prev;

tail.next = null;

} else{

while(index++ < position){

previous = current;

current = current.next;

}

previous.next = current.next;

current.next.prev = previous;

};

length-- ;

return current.element;

}else{

return false;

}

};

this.remove = function(element){

var current = head,

previous;

if(current.element === element){

head = current.next;

}

previous = current;

current = current.next;

while(current){

if (current.element = element) {

previous.next = current.next;

current.next.prev = previous;

}else{

previous = current;

current = current.next;

}

}

return false;

};

this.remove = function(){

if (length === 0) {

return false;

};

var current = head,

previous;

if(length === 1){

head = null;

tail = null;

length--;

return current.element;

}

while(current){

previous = current;

current = current.next;

}

previous.next = null;

length--;

return current.element;

};

this.indexOf = function(element){

var current = head,

index = 0;

while(current && index++ < length){

if (current.element === element) {

return index;

};

current = current.next;

}

return false;

};

this.isEmpty = function(){

return length === 0;

};

this.size = function(){

return length;

};

this.toString = function(){

var current = head,

string = '';

while(current){

string += current.element;

current = current.next;

}

return string;

};

this.getHead = function(){

return head;

};

this.getTail = function(){

return tail;

};

}

### 双向循环链表

将双向链表的头尾指针相连，就构成了双向循环链表。这种链表从任意一个节点都可以同时向两个方向进行节点遍历，查询节点的速度也是最快的。

/\*双向循环链表\*/

function DoublyCircularLinkedList(){

var Node = function(element){

this.element = element;

this.next = null;

this.prev = null;

};

var length = 0,

head = null,

tail = null;

this.append = function(element){

var node = new Node(element),

current,

previous;

if (!head) {

head = node;

tail = node;

head.prev = tail;

tail.next = head;

}else{

current = head;

while(current.next !== head){

previous = current;

current = current.next;

}

current.next = node;

node.next = head;

node.prev = current;

};

length++;

return true;

};

this.insert = function(position, element){

if(position >= 0 && position <= length){

var node = new Node(element),

index = 0,

current = head,

previous;

if(position === 0){

if(!head){

node.next = node;

node.tail = node;

head = node;

tail = node;

}else{

current.prev = node;

node.next = current;

head = node;

node.prev = tail;

}

}else if(position === length){

current = tail;

current.next = node;

node.prev = current;

tail = node;

node.next = head;

}else{

while(index++ < position){

previous = current;

current = current.next;

}

current.prev = node;

node.next = current;

previous.next = node;

node.prev = previous;

}

length++;

return true;

}else{

return false;

}

};

this.removeAt = function(position){

if(position > -1 && position < length){

var current = head,

index = 0,

previous;

if(position === 0){

current.next.previous = tail;

head = current.next;

}else if(position === length - 1){

current = tail;

current.prev.next = head;

head.prev = current.prev;

tail = current.prev;

}else{

while(index++ < position){

previous = current;

current = current.next;

}

previous.next = current.next;

current.next.prev = previous;

}

length--;

return true;

}else{

return false;

}

};

this.remove = function(element){

var current = head,

previous,

indexCheck = 0;

while(current && indexCheck < length){

if(current.element === element){

if(indexCheck === 0){

current.next.prev = tail;

head = current.next;

}else{

current.next.prev = previous;

previous.next = current.next;

}

length--;

return true;

}

previous = current;

current = current.next;

indexCheck++;

}

return false;

};

this.remove = function(){

if(length === 0){

return false;

}

var current = head,

previous,

indexCheck = 0;

if(length === 1){

head = null;

tail = null;

length--;

return current.element;

}

while(indexCheck++ < length){

previous = current;

current = current.next;

}

previous.next = head;

tail = previous.next;

length--;

return current.element;

};

this.indexOf = function(element){

var current = head,

index = 0;

while(current && index++ < length){

if(current.element === element){

return index;

}

current = current.next;

}

return false;

};

this.toString = function(){

var current = head,

indexCheck = 0,

string = '';

while(current && indexCheck < length){

string += current.element;

indexCheck++;

current = current.next;

}

return string;

};

this.isEmpty = function(){

return length === 0;

};

this.getHead = function(){

return head;

};

this.getTail = function(){

return tail;

};

this.size = function(){

return length;

};

}

## 集合——Set

集合是一组无序且唯一（不能重复）的项组成的。这个数据结构使用了与有限集合相同的数学概念，但应用在了计算机科学的数据结构中。集合里的每一项都是用对象Object包裹起来的，包括键与值。

### 创建集合

function Set(){

let items ={};

}

### 给集合声明一些属性方法：

add(value)：向集合内添加一个新的项。

remove(value)：移除集合中某一个值。

has(value)：判断是否含有该值。

clear()：清空该集合

size()：获取集合所包含元素的数量。

values()：返回一个包含集合中所有值所组成的数组。

has(value)方法：可以用js的in操作符

this.has=function(value){

return value in items；

}

add(value)方法：

在add方法中，会先判断该元素是否在集合中，如果不在集合中，那么就添加。否则就return false;。

this.add=function(value){

if(!This.has(value)){

items[value]=value;

}

return false;

}

remove和clear方法：

同add方法，remove也会先判断该元素是否在集合中。

This.remove=function(value){

If(value in items){

delete items[value];

}

Return;

}

Clear()方法：

This.clear=function(){

items = { }

}

Size()方法:

Size()方法有两种实现方式：

1:间接获取Object.keys(items)到的长度。

this.size=function(){

return Object.keys(items).length;

//return Object.values(items).length;

}

2:老老实实用for in循环。

this.size=function(){

For(let key in items){

Let count =0;

If（items.hasOwnProperty(key)){++count;}

}

Return count;

}

Values()方法：也有两种方法实现第一种只有谷歌与火狐兼容。第二种是兼容大众浏览器。

1：this.values=function(){

Return Object.values(items);

}

2：this.valuesAll=function(){

Let vlaues=[];

For(let key in items){

If（items.hasOwnProperty(key)){vlaues.push(items[key]);}

}

return values;

}

Set的遍历：for of

for(let item of set){

}

### 集合的运算：交集，并集，差集与子集

并集：对于给定的两个集合，返回一个包含两个集合中素有元素所组成的不重复的新集合。

交集：对于给定的两个集合，返回一个包含两个集合中共有元素所组成的新集合。

差集：对于给定的两个集合，返回一个包含于第一个集合而且不包含在第二个集合的元素所组成的新集合。

子集：验证一个给定集合是否是另一个集合的子集。

1. 并集(union)：

this.union =function(otherSet){

let unionSet=newSet();

let values=this.values();

for(let i=0;i<values.length;i++){

unionSet.add(values[i]);

}

let values2=otherSet.values();

for(let i=0;i<values2.length;i++){

unionSet.add(values2[i]);

}

return unionSet;

}

1. 交集(intersection)：

This.intersection=function(otherSet){

Let intersectionSet=new Set();

Let values = this.values();

For(let i =0;i<values.length;i++){

If(otherSet.has(values)){

intersectionSet.add(values[i]);

}

}

Return intersectionSet;

}

1. 差集(difference)：

This.difference=function(otherSet){

Let differenceSet=new Set();

Let values=this.values();

For(let i =0;i<values.length;i++){

If(!otherSet.has(values[i])){

differenceSet.add(values[i]);

}

}

Return differenceSet;

}

1. 子集

This.subset=function(otherSet){

If(this.size()>otherSet.size()){

Return false;

}

Else{

Let values=this.values();

For(let i=0;i<values.length;i++){

If(!otherSet.has(values[i]){return false;}

}

Return true;

}

}

### 使用ES6——Set类

Let set = new Set( );

set.add(1);

ES6的Set类已经包含了我们以上封装好的方法，可以直拿来接用。但和我们自己封装的Set类不同的是，ES6的Set类只能打印Iterator，而不是由值构成的数组。

let a = new Set([1, 2, 3]);

let b = new Set([4, 3, 2]);

// 并集

let union = new Set([...a, ...b]);

// Set {1, 2, 3, 4}

// 交集

let intersect = new Set([...a].filter(x => b.has(x)));

// set {2, 3}

// 差集

let difference = new Set([...a].filter(x => !b.has(x)));

// Set {1}

## 字典——Map类

与Set类似，Map是ES6对对象Object的封装，用于存储一组互不相同的元素。在字典中，我们用【键，值】来存储数据。

## 创建字典

fnction Dictionary(){

var items={};

}

## 给字典声明一些方法

set（key，value）：向字典中添加新元素。

Delete（key）：通过键删除已经存在于字典中的键对应的值。

Has（key）：判断该键是否存在于字典中。

Get（key）：通过键值获取对应数据，有的话返回该值，没有的话返回undefined。

Clear（）：将该字典清空。

Size（）：获取该字典的元素数量，与数组的length长度类似。

Keys（）：返回字典中所有key组成的数组。

Values（）：返回字典中所有值组成的数组。

1. has和set方法：

This.has = function(key){

Return items.hasOwnProperty(key);

// return items[key]===undefined;

}

This.set=function(key,value){

Items.key =value;

//Items[ key ]=value;

}

1. delete方法：

This.delete = function(key){

If(item.has(key)) {

delete items.key;

}

}

1. get 和values方法：

This.get=function(key){

Return item.has(key)?Items.key:undefined;

}

This.values=function(){

Const values=[];

For(let k in items){

Values.push(items[k]);

}

Return values;

}

1. clear,size,keys和getItems方法

This.clear=function(){

Items={};

}

This.size=function(){

Return this.values().length;

}

This.keys=function(){

Const keys=[];

For(let k in items){

keys.push(k);

}

return keys;

}

This.getItems=function(){

return items;

}

实例化：

Const myd =**new** Dictionary();

## ES6——Set类

Const myset = new Set();

以上封装的字典方法都适用于Set类。

## 树——Tree

# 常用函数汇总

## 判断两个数组是否相等

function arrayEqual(arr1, arr2) {

    if (arr1 === arr2) return true;

    if (arr1.length != arr2.length) return false;

    for (var i = 0; i < arr1.length; ++i) {

        if (arr1[i] !== arr2[i]) return false;

    }

    return true;

}

## 根据name读取cookie

function getCookie(name) {

    var arr = document.cookie.replace(/\s/g, "").split(';');

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

        var tempArr = arr[i].split('=');

        if (tempArr[0] == name) {

            return decodeURIComponent(tempArr[1]);

        }

    }

    return '';

}

## removeCookie

function removeCookie(name) {

    // 设置已过期，系统会立刻删除cookie

    setCookie(name, '1', -1);

}

## setCookie

function setCookie(name, value, days) {

    var date = new Date();

    date.setDate(date.getDate() + days);

    document.cookie = name + '=' + value + ';expires=' + date;

}

## 获取浏览器类型和版本

function getExplore() {

    var sys = {},

        ua = navigator.userAgent.toLowerCase(),

        s;

    (s = ua.match(/rv:([\d.]+)\) like gecko/)) ? sys.ie = s[1]:

        (s = ua.match(/msie ([\d\.]+)/)) ? sys.ie = s[1] :

        (s = ua.match(/edge\/([\d\.]+)/)) ? sys.edge = s[1] :

        (s = ua.match(/firefox\/([\d\.]+)/)) ? sys.firefox = s[1] :

        (s = ua.match(/(?:opera|opr).([\d\.]+)/)) ? sys.opera = s[1] :

        (s = ua.match(/chrome\/([\d\.]+)/)) ? sys.chrome = s[1] :

        (s = ua.match(/version\/([\d\.]+).\*safari/)) ? sys.safari = s[1] : 0;

    // 根据关系进行判断

    if (sys.ie) return ('IE: ' + sys.ie)

    if (sys.edge) return ('EDGE: ' + sys.edge)

    if (sys.firefox) return ('Firefox: ' + sys.firefox)

    if (sys.chrome) return ('Chrome: ' + sys.chrome)

    if (sys.opera) return ('Opera: ' + sys.opera)

    if (sys.safari) return ('Safari: ' + sys.safari)

    return 'Unkonwn'

}

## 获取操作系统类型

function getOS() {

    var userAgent = 'navigator' in window && 'userAgent' in navigator && navigator.userAgent.toLowerCase() || '';

    var vendor = 'navigator' in window && 'vendor' in navigator && navigator.vendor.toLowerCase() || '';

    var appVersion = 'navigator' in window && 'appVersion' in navigator && navigator.appVersion.toLowerCase() || '';

    if (/mac/i.test(appVersion)) return 'MacOSX'

    if (/win/i.test(appVersion)) return 'windows'

    if (/linux/i.test(appVersion)) return 'linux'

    if (/iphone/i.test(userAgent) || /ipad/i.test(userAgent) || /ipod/i.test(userAgent)) 'ios'

    if (/android/i.test(userAgent)) return 'android'

    if (/win/i.test(appVersion) && /phone/i.test(userAgent)) return 'windowsPhone'

}

## 判断浏览设备（PC，手机，微信）

function IsPC() {  
var userAgentInfo = navigator.userAgent;  
var Agents = ["Android", "iPhone","SymbianOS", "Windows Phone",  
                "iPad", "iPod"];  
    var flag = true;  
    for (var v = 0; v < Agents.length; v++) {  
    if (userAgentInfo.indexOf(Agents[v]) > 0) {  
    flag = false;  
    break;  
    }  
    }  
    return flag;  
}

If(IsPC()){

Console.log(“是PC端”);

}

else{

Console.log(“是手机端”);

}

## 判断是否微信浏览器中打开(是否是微信端)

var ua = navigator.userAgent.toLowerCase();

if(ua.match(/MicroMessenger/i)=="micromessenger") {

return true;

} else {

return false;

}

## 判断手机操作系统(ios或者是Android)

var u = navigator.userAgent, app = navigator.appVersion;

var isAndroid = u.indexOf('Android') > -1 || u.indexOf('Linux')> -1; //安卓终端

var isIOS = !!u.match(/\(i[^;]+;( U;)? CPU.+Mac OS X/); //ios终端

if (isAndroid) {

//这个是安卓操作系统

}

if (isIOS) {

　　//这个是ios操作系统

}

## 获取滚动条距顶部的距离getScrollTop

**function** getScrollTop() {

**return** (document.documentElement && document.documentElement.scrollTop) || document.body.scrollTop;

}

## 获取一个元素（ele）的距离文档offect的位置

**function** getOffset(ele) {

**var** pos = {

        left: 0,

        top: 0

    };

**while** (ele) {

        pos.left += ele.offsetLeft;

        pos.top += ele.offsetTop;

        ele = ele.offsetParent;

    };

**return** pos;

}

## 在${duration}时间内，滚动条平滑滚动到${to}指定位置

**function** scrollTo(to, duration) {

**if** (duration < 0) {

        setScrollTop(to);

**return**

    }

**var** diff = to - getScrollTop();

**if** (diff === 0) **return**

**var** step = diff / duration \* 10;

    requestAnimationFrame(

**function** () {

**if** (Math.abs(step) > Math.abs(diff)) {

                setScrollTop(getScrollTop() + diff);

**return**;

            }

            setScrollTop(getScrollTop() + step);

**if** (diff > 0 && getScrollTop() >= to || diff < 0 && getScrollTop() <= to) {

**return**;

            }

            scrollTo(to, duration - 16);

        });

}

## 设置滚动条距顶部的距离setScrollTop

**function** setScrollTop(value) {

    window.scrollTo(0, value);

**return** value;

}

## 对象深拷贝，支持常见类型

**function** deepClone(values) {

**var** copy;

    // Handle the 3 simple types, and null or undefined

**if** (**null** == values || "object" != **typeof** values) **return** values;

    // Handle Date

**if** (values **instanceof** Date) {

        copy = **new** Date();

        copy.setTime(values.getTime());

**return** copy;

    }

    // Handle Array

**if** (values **instanceof** Array) {

        copy = [];

**for** (**var** i = 0, len = values.length; i < len; i++) {

            copy[i] = deepClone(values[i]);

        }

**return** copy;

    }

    // Handle Object

**if** (values **instanceof** Object) {

        copy = {};

**for** (**var** attr **in** values) {

**if** (values.hasOwnProperty(attr)) copy[attr] = deepClone(values[attr]);

        }

**return** copy;

    }

**throw** **new** Error("Unable to copy values! Its type isn't supported.");

}

## 判断obj是否为空

**function** isEmptyObject(obj) {

**if** (!obj || **typeof** obj !== 'object' || Array.isArray(obj))

**return** **false**

**return** !Object.keys(obj).length

}

## 随机生成颜色

**function** randomColor() {

**return** '#' + ('00000' + (Math.random() \* 0x1000000 << 0).toString(16)).slice(-6);

}

## 判断是否为邮箱地址email

**function** isEmail(str) {

**return** /\w+([-+.]\w+)\*@\w+([-.]\w+)\*\.\w+([-.]\w+)\*/.test(str);

}

## 判断是否为身份证号**isIdCard**

**function** isIdCard(str) {

**return** /^(^[1-9]\d{7}((0\d)|(1[0-2]))(([0|1|2]\d)|3[0-1])\d{3}$)|(^[1-9]\d{5}[1-9]\d{3}((0\d)|(1[0-2]))(([0|1|2]\d)|3[0-1])((\d{4})|\d{3}[Xx])$)$/.test(str)

}

## 判断是否为手机号

**function** isPhoneNum(str) {

**return** /^(0|86|17951)?(13[0-9]|15[012356789]|17[678]|18[0-9]|14[57])[0-9]{8}$/.test(str)

}

## 判断是否为国内手机号

//手机号正则

var r=/^((0\d{2,3}-\d{7,8})|(1([358][0-9]|4[579]|66|7[0135678]|9[89])[0-9]{8}))$/;

if(!r.test(phone)){

alert('号码格式错误,固话需加区号和符号‘-’');

return;

};

## 判断是否为URL地址

**function** isUrl(str) {

**return** /[-a-zA-Z0-9@:%.\_\+~#=]{2,256}\.[a-z]{2,6}\b([-a-zA-Z0-9@:%\_\+.~#?&//=]\*)/i.test(str);

}

## 现金额转大写

**function** digitUppercase(n) {

**var** fraction = ['角', '分'];

**var** digit = [

        '零', '壹', '贰', '叁', '肆',

        '伍', '陆', '柒', '捌', '玖'

    ];

**var** unit = [

        ['元', '万', '亿'],

        ['', '拾', '佰', '仟']

    ];

**var** head = n < 0 ? '欠' : '';

    n = Math.abs(n);

**var** s = '';

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

        s += (digit[Math.floor(n \* 10 \* Math.pow(10, i)) % 10] + fraction[i]).replace(/零./, '');

    }

    s = s || '整';

    n = Math.floor(n);

**for** (**var** i = 0; i < unit[0].length && n > 0; i++) {

**var** p = '';

**for** (**var** j = 0; j < unit[1].length && n > 0; j++) {

            p = digit[n % 10] + unit[1][j] + p;

            n = Math.floor(n / 10);

        }

        s = p.replace(/(零.)\*零$/, '').replace(/^$/, '零') + unit[0][i] + s;

    }

**return** head + s.replace(/(零.)\*零元/, '元')

        .replace(/(零.)+/g, '零')

        .replace(/^整$/, '零元整');

};

## 判断浏览器是否支持webP格式图片isSupportWebP

**function** isSupportWebP() {

**return** !![].map && document.createElement('canvas').toDataURL('image/webp').indexOf('data:image/webp') == 0;

}

## 格式化${startTime}距现在的已过时间

**function** formatPassTime(startTime) {

**var** currentTime = Date.parse(**new** Date()),

        time = currentTime - startTime,

        day = parseInt(time / (1000 \* 60 \* 60 \* 24)),

        hour = parseInt(time / (1000 \* 60 \* 60)),

        min = parseInt(time / (1000 \* 60)),

        month = parseInt(day / 30),

        year = parseInt(month / 12);

**if** (year) **return** year + "年前"

**if** (month) **return** month + "个月前"

**if** (day) **return** day + "天前"

**if** (hour) **return** hour + "小时前"

**if** (min) **return** min + "分钟前"

**else** **return** '刚刚'

}

## 格式化现在距${endTime}的剩余时间

**function** formatRemainTime(endTime) {

**var** startDate = **new** Date(); //开始时间

**var** endDate = **new** Date(endTime); //结束时间

**var** t = endDate.getTime() - startDate.getTime(); //时间差

**var** d = 0,

        h = 0,

        m = 0,

        s = 0;

**if** (t >= 0) {

        d = Math.floor(t / 1000 / 3600 / 24);

        h = Math.floor(t / 1000 / 60 / 60 % 24);

        m = Math.floor(t / 1000 / 60 % 60);

        s = Math.floor(t / 1000 % 60);

    }

**return** d + "天 " + h + "小时 " + m + "分钟 " + s + "秒";

}

## url参数转对象

**function** parseQueryString(url) {

    url = url == **null** ? window.location.href : url

**var** search = url.substring(url.lastIndexOf('?') + 1)

**if** (!search) {

**return** {}

    }

**return** JSON.parse('{"' + decodeURIComponent(search).replace(/"/g, '\\"').replace(/&/g, '","').replace(/=/g, '":"') + '"}')

}

## 对象序列化**stringfyQueryString**

**function** stringfyQueryString(obj) {

**if** (!obj) **return** '';

**var** pairs = [];

**for** (**var** key **in** obj) {

**var** value = obj[key];

**if** (value **instanceof** Array) {

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

                pairs.push(encodeURIComponent(key + '[' + i + ']') + '=' + encodeURIComponent(value[i]));

            }

**continue**;

        }

        pairs.push(encodeURIComponent(key) + '=' + encodeURIComponent(obj[key]));

    }

**return** pairs.join('&');

}

## 删除旧的全量压缩包

rm(path.resolve(rootPath, 'min', `${pkg.name}.min.js`), err => {

**if** (err) **throw** (err)

    webpack(config, **function** (err, stats) {

**if** (err) **throw** (err)

        building.stop()

        process.stdout.write(stats.toString({

            colors: **true**,

            modules: **false**,

            children: **false**,

            chunks: **false**,

            chunkModules: **false**

        }) + '\n\n')

        resolve()

        console.log(chalk.cyan('  Build complete.\n'))

    })

})

OA：办公自动化

CRM管理系统： custormer releation Manager。

ERP ：Enterprise Relationship Planning 企业资源计划

O2O：online to offline

EC（Electronic Commrce电子商务）

SNS （Social Network Site）

# DOM对象模型

一：Dom产生于98年，由ECMAScript组织把原属于XML技术的DOM移植到了HTML平台上。

XML：是一种数据传输方式，08年之前是主流。

二：DOM描绘了一个层次化的节点关系。允许开发人员动态的去增删改查页面的一部分。98年10月，DOM一级规范出炉，目前各大浏览器厂商都支持DOM。

三：DOM1时代 2B青年的操作节点

Node只不过是一个HTML文档给我们提供的一个操作节点的API（Application InterFace 应用程序接口）。当时除了IE之外，任何浏览器都支持Node对象来操作文档。目前新版支持。

1：NodeName和Nodevalevalue

要了解具体信息。最好使用上面这两个属性。

Nodename可以直接可以直接获取元素对象名称，也就是标签名。Doucument.getEl....获得

Nodevalue实在接待你元素上才可找到元素值。、

1. 节点关系

在DOM元素下，每个阶段都与其他节点有关系，节点之之间的关系通常使用家族关系来描述的。家族。在HTMl中，可以将<html>元素看成<body>元素的父亲。每一个节点都有一个childnode属性，其中保存着一个神秘的对象：nodelist。Nodelist是一个数组，可以通过[ ]来访问nodelist的值。既然是数组，那么肯定有length属性，但他并不是Array的实例。

(Window.)document.title=”” 作用于<title></title>标签。给浏览器命名。

document.write(“”);动态输出一段话

Document.write(“”)与innerHTML=（“”）的区别：

1：Document.write(“”)本意是重绘html页面样式，是把原来的页面清空之后再写一遍，而innerHTML=（“”）是对页面样式的书写，可以添加或更改，而不是重绘。

2：在工作中：忘掉innertext，很少用Document.write(“”)，常用innerHTML=（“”）。

3：document.write与window.onload=function(){}

Window.onload含义是当页面html样式全部加载完毕之后再显示它下面的东西，而document.write意思是网页重绘，如果<style>标签在<script>上面的话，那么在window.onload下面书写document.write(“”)，在<style>标签里书写document.write(“”)里的CSS样式，CSS将不起作用。

Document.read()读取一段话

document.getElementsByTagName(“标签名”);通过标签选择器来找到自己的标签，作用于多个标签选择器，返回一个数组。

var divs=document.getElementsByTagName("div");

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

{

console.log(divs[i].innerText);

console.log(divs[i].innerHTML);

}

Document.getElementById(“ID名”)通过id名来找到自己想要的标签。返回一个元素。

⑤： var input= document.getElementById("dy");

alert(input.value);

Document.getElementsByClassNa'me(“标签名”)通过标签名来找到自己想要的标签，作用于多个标签，返回一个数组。

var tags= document.getElementsByClassName("btn");

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

{

tags[i].style.backgroundColor="red";

}

Document.getElementsByName(“name属性名称”)通过name属性名称来找到自己想要的标签。返回一个数组。

var tags= document.getElementsByName("tag");

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

{

tags[i].style.width="500px";

}

1.onabort事件:对象载入被中断时触发

2.onblur事件：元素或窗口本身失去焦点时触发。

3.onchange事件：改变<select>元素中的选项或者其他表单元素失去焦点，并且在其获取焦点后内容发生过改变时触发。

4.onclick事件：单机鼠标左键时候触发。当光变的焦点在按钮上，并且按下Enter键时，也会触发该事件。

5.onblclick事件:双击鼠标左键时候触发。

6.onerror事件:出现错误时候触发。

7.onfocus事件：任何元素或者窗口本身获得焦点时触发。

8.onkeydown事件:键盘上的按键（包括Alt和Shift等键）被按下时候触发，如果一直按着，则会不断触发。返回false时，取消默认动作。

9.onkeypress事件：键盘上的按键被按下，并产生一个字符时候发生，即当按下Shift等键时不断触发。如果一直按下某键时，会不断触发。返回lfalse时候，取消默认动作。

10.onkeyup事件：释放键盘上的按键时候触发。

11.onload事件:页面完全载入后，在windows对象上触发;所有框架都载入后，在框架集上触发;<img>标记指定的图像完全载入后，在其上触发；<object>标记指定的对象完全载入后，在其上触发。

12.onmousedown事件:单击任何一个鼠标按键时候触发。

13.onmousemove事件:鼠标在某个元素上持续移动时候触发。

14.onmouseout事件：将鼠标从指定元素上移开时触发。

15.onmouseover事件:鼠标移动到某个元素上时候触发。

16.onmouseup事件:释放任意一个鼠标按键时候触发。

17:onreset事件：单击重置按钮是在<form>上触发。

18.onresize事件：窗口或者框架大小有所改变时候触发。

19.onscroll事件：在屏幕滚动的时候触发。

20.onselect事件：选中文本时触发。

21.onsubmit事件：单击提交按钮时候，在<form>上触发。

22.onnunload事件：当页面完全卸载后，在Window对象上触发，或者所有框架都卸载后，在框架集上触发。

## Dom事件汇总

事件3要素：

1：事件源（对象）

2：事件：（触发方法）

Window.onload/onunload

Onsubmit：表单提交的事件

Onreset：重置表单

Onclick/bdlclick事件：

Change事件：文本经过或者下拉菜单中的选项发生改变

Onfocus、onblur聚焦/失焦onmouseover：鼠标悬停(包括子元素) onmouseout：鼠标移出（包括子元素）

.focus();使对象获得焦点（处于获得焦点的状态）

.onmouseenter:鼠标悬停(不包括子元素)

.onmouseleave:鼠标移出（不包括子元素）

在拖动目标上触发事件 (目标元素)以鼠标光标事件（event）为标准，而不是被拖拽对象（this）:

1、要想使用拖拽功能，就必须使用draggable属性，标签内设置行内样式设置为true。

2、对象.[ondragstart](http://www.runoob.com/jsref/event-ondragstart.html) - 用户开始拖动元素时触发该事件，此事件作用在被拖拽元素上。

3、ondrag - 元素正在拖动时触发

4、[ondragend](http://www.runoob.com/jsref/event-ondragend.html) - 用户完成元素拖动后触发，此事件作用在被拖拽的元素上。

5、[ondragenter](http://www.runoob.com/jsref/event-ondragenter.html) - 当被鼠标拖动的对象进入其容器范围内时触发此事件，只触发一次，此事件作用在目标元素上，以鼠标光标事件（event）为标准，而不是被拖拽对象（this）

6、[ondragover](http://www.runoob.com/jsref/event-ondragover.html) - 当某被拖动的对象在另一对象容器范围内拖动时触发此事件，此事件作用在目标元素/容器，以鼠标光标事件（event）为标准，而不是被拖拽对象（this）只要被拖拽对象在目标容器内，该事件一直在触发。

7、[ondragleave](http://www.runoob.com/jsref/event-ondragleave.html) - 当被鼠标拖动的对象离开其容器范围内时触发此事件，以鼠标光标事件（event）为标准，而不是被拖拽对象（this）。

8、[ondrop](http://www.runoob.com/jsref/event-ondrop.html) - 在一个拖动过程中，释放鼠标键时触发此事件，作用在目标接纳容器上

9、dataTransfer 对象：拖拽对象用来传递的媒介使用的时候使用event.dataTransfer 。

3：鼠标事件处理程序（结果）

|  |  |
| --- | --- |
| 事件名称 | 说明 |
| onclick | 鼠标左键单击 |
| Ondblclick | 鼠标左键双击 |
| Onkeyup | 按下并松开一个键（微博限制数字） |
| Onkeydown | 按下触发的那一刻（微博输入框限制数字） |
| Onchange | 文本内容或下拉菜单中的选项发生改变。 |
| Onfocus | 获得焦点，表示文本框等获得鼠标焦点 |
| Onblur | 失去标点，表示文本框等失去鼠标焦点 |
| Onmouseomve | 当鼠标按住拖动时 |
| Onmouseover | 就是鼠标悬停在被获得的元素上方时 |
| Window.Onload | 网页加载完毕时显示 |
| window.Onunload | 网页关闭时显示 |
| Ondragstart | 被拖动的对象开始拖动瞬间时 |
| Ondragover | 被拖动的对象悬停在其对象其容器范围内时 |
| Ondragend | 被拖动的对象拖拽结束时 |
| Ondragenter | 当被拖动的对象进入其容器范围内时 |
| Ondrag | 正在拖动时 |
| Ondragleave | 当被拖动的对象离开其容器范围内时 |
| ondragover | 鼠标拖动停止时 |
| Onmousedown | 鼠标左键按下时 |
| Onmouseup | 鼠标左键松开时 |
| Onmouseleave | 鼠标离开元素时 |

事件三要素：

事件源：点击，改变了，双击了，悬浮了

事件处理程序：就是function（）函数。

事件源.事件

This：指的是外部事件所触发的元素。

祖先包含关系的节点查找：Dom树

Dom操作包含级标签就称为节点查找。

父子关系：祖先节点也属于间接父子关系事件就是由外部因素导致的动作和行为。

事件的声明：用函数来声明事件，后面要加上匿名函数。

键盘事件：

3：.onkeydowm鼠标或键盘按下去没有松开时

4：.onkeyup鼠标或键盘按下去再松开时

## 通过DOM来操作节点的属性：

**对象名.setAttribute(“属性名”,”属性值”)；解释一下，这里的Attribute属性是后天具有的，比如name，class，id，href，src等等。**

**设置多个属性：.setAttribute（{“key”：“value”，“key”：“value”，“key”：“value”，......}）**

**对象名.Classlist=””;**

**对象名.ClassName=””;这三种含义是相同的**

对象名.parentNote 通过此标签找到其父亲节点。

Div.getAttribute（“属性名”）获取该节点的某个属性名。

对象名.setAttribute（css属性，属性值）设置一个新的属性和属性值。

对象名.removeAttribute(css属性，属性值)移除某该节点的某一个属性。

兄弟关系：

标签名.nextSibling(标签名)找到下一个兄弟标签。空格也算。

标签名.previousSibling(标签名)找到上一个兄弟标签，空格也算。

选择器不要选错。

还有一种对标签的属性（prototype）操作：

El.property.”属性名称”=”值”；

这里的property属性指的是元素本身具有的属性天生的属性，比如input标签中type为checkbox或者radio样式时，它的checked属性或者value属性。

而style属性是专门指css样式属性。

## Document获取元素并触发的事件：

一 根据id获取元素的值，getElementById（）返回一个HTML元素对象。

二 根据name取得多个元素所组成的数组getElementsByName（）

注意：textContent只获取元素中的文本，而innerHTML不但获取文本，而且获取里面HTML标签。

三：根据类样式的名称来获取一串HTML标签对象getElementsByClassName（）返回一个含有该由类名的标签所组成的一个数组。

四：根据标签名称获取HTMl标签getElementsByTagName（），返回一个由所有该标签组成的数组。

五：js根据需要找到元素对象，找到我们想要的第一个标签类型document.querySelector( “.div” “.p”);返回第一个符合条件的标签。

六java根据需要找到元素对象，找到所有想要的标签类型document.querySelector（“div”）返回所有符合条件的元素。

七：节点操作

1: .parentNode 找到父亲级元素。

1. nextSibling(标签名)找到下一个兄弟标签。但是空格也算。.previousSibling(标签名)找到上一个兄弟标签，空格也算。
2. previousSibling同级的上个兄弟节点。
3. **childNodes返回一个节点下面所有一阶子节点的组合数组（只包含儿子元素，不包含孙子元素）**
4. firstchild第一个孩子
5. lastchild最后一个孩子
6. **div .removeChild(被删除的节点名称)删除节点，空格也算一个节点。**
7. **div.replaceChild（标签1，标签2）替换节点，用标签1替换标签2。**
8. 父节点：</b>parentNode、offsetParent（获取元素用于定位的那个父级）
9. **nodeType单个节点类型。只要是一个标签，其type就是1。**
10. **nodeValue节点的值**
11. **nodeName节点的名称**
12. **document.createElement(“标签名”)新建一个标签。**

**标签名.appendChild（子元素名)；追加一个子元素，不分类型，只要是元素都可以。先创建，再追加。**

**14.标签名.createTextNode（“一段话”）新建一段文本元素（字符串）。**

1. **.insertbefore（标签a，标签b）；把a标签放到标签前面。**

## JS动态创建元素/节点：

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

设置父元素接收并添加：

节点名.appendChild(lis);

JS动态设置所创建对象的样式：

lis.style.width=400+"px";

lis.style.height=20+"px";

lis.style.border="solid 1px red";

lis.style.listStyle="none";

JS动态添加元素的class/id名:  
lis.className="liss";或者lis.setAttribute(“class”,”class名”)；

console.log(lis.className);

li.id="lis1";

getElement方法只能得到，得到之后不能再设置样式。

然后再var liss=document.getElemensByClassName(“liss”);

或者 var lis1=document.getElementById(“lis1”);

console.log(liss.length);

删除节点：

节点名.removeChild(lis);

节点名.children;该节点的所有子节点

节点名.children.length该节点的所有子节点的长度。

JS动态创建元素/节点：

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

设置父元素接收并添加：

节点名.appendChild(lis);

JS动态设置所创建对象的样式：

lis.style.width=400+"px";

lis.style.height=20+"px";

lis.style.border="solid 1px red";

lis.style.listStyle="none";

JS动态添加元素的class/id名:  
lis.className="liss";

console.log(lis.className);

lis.id="lis1";/lis.setAttribute(“id”,”lis1”);

getElement方法只能得到，得到之后不能再设置。

然后再var liss=document.getElemensByClassName(“liss”);

或者 var lis1=document.getElementById(“lis1”);

console.log(liss.length);

删除节点：

节点名.removeChild(lis);

节点名.children;该节点的所有子节点

节点名.children.length该节点的所有子节点的长度。

1. **Dom对象是节点**

**得到指定节点对象的类名：var cN=document.getElementById(“id名”).className;**

**Alert(cN);**

**可以通过改变节点对象的类名来改变节点标签的样式，实现一定的样式效果，如选项卡的制作。**

**找到节点对象的索引值（一般在ul li 中）：**

**.style.index**

**计算机对于操作者来说，就是4件事：增删改查。**

**NodeObj.style.css属性=“值”：动态设置nodeObj的css样式，只能读取或设置行内样式，style标签里的样式不能读取。**

**.getComputedStyle（element，ov）是获取一个对象最终的所有属性列表，返回一个由属性组成的数组，只读。Element：元素节点对象；ov：伪类，如果没有伪类则值为null。**

**一般我们在读取一个元素的摸一个CSS属性时使用的方式是getPropertyCSSValue（返回一个叫做CSS2Properties的对象）或者使用getPropertyValue（）（返回一个PropertyValue的字符串）。**

## 循环为Dom元素添加事件

HTML：

<body>

<div class="d1">1</div>

<div class="d1">2</div>

<div class="d1">3</div>

<div class="d1">4</div>

<div class="d1">5</div>

</body>

### JS第一种写法：普通for循环

window.onload=function(){

var d1=document.getElementsByClassName("d1");

console.log(d1);

for(var i=0;i<d1.length;i++){

d1[i].index=i;//在这里，为d1[i]的对象属性index赋了值为i。

d1[i].onclick=function(){

console.log(this.index);//this指向函数的调用者，即d1[i]。

}

}

}

### JS第二种写法：Arr.map( )方法

因为Arr.map()规定，这里的Arr必须是严格意义上的array，但是通过document选择器获取的并不是严格意义上的array，而是HTMLCollection(5)，所以要用ES6的Array.form()方法处理一下，转换为真正的array。

window.onload=function(){

var d1=document.getElementsByClassName("d1");

console.log(d1);

var d1Arr=Array.from(d1)

console.log(d1Arr);

d1Arr.map(function(item,index){

item.index=index;

item.onclick=function(){

console.log(this.index)

}

})

}

# JS操控浏览器本身事件

一：Onsubmit.function( ){

}

点击提交事件，处理用户表单元素向服务器提交数据。无论它之后有多少个事件，这个提交事件都是最后触发。需要提交的表单元素都写在Onsubmit事件中。

return false 固定语法，阻止标签在该语句之后的默认事件发生。

二：一个链接？跟的是参数

？：参数名称=参数值&参数名称=参数值&参数名称=参数值

一个按钮（或者说一个鼠标操作）可出发多个事件。

Javascript：void（0）；阻止标签属性的运行。

Argument(s)参数，argument.length实际参数的个数.无论形式参数有几个，它只显示实参就打印几个。从先到后。

## 事件冒泡:

事件冒泡：就是子元素触发事件之后，父元素的事件也会随之出发（并不需要父元素出发）

阻止事件冒泡的发生：在事件函数的参数括号里写个e，再在事件结束之前添加一句event.stopPropagation();这种方法只能阻止这一级事件的发生。比如：

div2.onclick=function(e){

Alert(“我是div2！！”)；

e.stopPropagation()

}

Document.getElementById(“div1”).onclick=fuction(){

Alert(“我是div1”)

}

阻止事件默认执行的方法：

return false 不仅阻止了事件往上冒泡，而且阻止了事件本身。event.stopPropagation() 则只阻止事件往上冒泡，不阻止事件本身。

event.stopPropagation()方法

$("#d0").click(*function*(){

*console*.log("d0");

})

$("#d1").click(*function*(*event*){

*event*.stopPropagation();

*console*.log("d1");

})

## 阻止浏览器继续执行js代码深入剖析：

”return false“到底做了什么？

当你每次调用”return false“的时候，它实际上做了3件事情：

•event.preventDefault();  
　　•event.stopPropagation();  
　　•停止回调函数执行并立即返回。

## **js检测复制、粘贴与剪切操作**

检测复制：

$("#textA").bind('copy', function() {

}

检测粘贴：

$("#textA").bind(‘paste’, function() {

}

检测剪切：

$("#textA").bind(‘cut’, function() {

}

## [js禁用页面上左右键菜单、选中和复制](http://www.cnblogs.com/liaidai/p/6737784.html)：

oncontextmenu 事件：当点击右键时触发该事件。

onselectstart事件：当开始选中时触发该事件。

onkeydown 事件：当有键盘被按下时触发该事件。

\* 禁用右键菜单

document.oncontextmenu = function(){

event.returnValue = false;

};

\* 禁用选中功能

document.onselectstart = function(){

event.returnValue = false;

};

禁用复制功能

document.oncopy = function(){

event.returnValue = false;

};

\* 禁用鼠标的左右键 \* @param {Object} e

document.onmousedown = function(){

if(event.which==1){//鼠标左键

return false;

}

if(event.which==3){//鼠标右键

return false;

}

};

\* 获取键盘上的输入值

document.onkeydown = function(){

console.info(event.which);

if(event.which==13){

console.info("回车键");

}

};

# Ajax异步请求

## ajax请求（Asynchronous JavaScript And Xml）

异步javascript和XML（代表产品：G-mail谷歌邮箱）

存在原因是浏览器与服务器之间的数据交互不用通过刷新页面就可以完成。

创建AJAX对象请求服务器一共分为4个步骤（每一步都非常重要）：

①：**创建浏览器的ajax对象**

**var oajax=new XMLHttpRequest();//保留了最原始的xml格式，非IE浏览器适用**

**//IE适用于 var oajax=new ActiveXObject("Microsoft.XMLHTTP");**

**var oajax;**

**if (window.XMLHttpRequest) { //判断是否为非IE浏览器,如果是非IE浏览器，那么就往下执行**

**oajax=new XMLHttpRequest()**

**}**

**else{//如果不是，那么就写为IE浏览器的一部对象声明。**

**oajax=new ActiveXObject("Microsoft.XMLHTTP");**

**}**

②：**连接服务器**

通过 .open（“Get/Post”,端口号，true/false）方法。Open（）方法有3个参数，第1个参数是请求方式(GET/POST)第2个参数是URL地址，即所要传递的服务器端口号，第3个参数是 是否异步，布尔值，true是异步，false为同步。

语法：

**oajax.open(“Get/Post”,“url地址”,true);**

③发送请求，一句话

**oajax.send(data)**

这句话非常关键。

浏览器从读到这句话开始，浏览器与服务器就开始进行数据交互并且执行判断5步状态了，而且是循环执行5次。

④接收返回值：

**oajax.onreadystateChange事件，当请求被发送到服务器端时，我们需要执行一些基于响应的任务。每当readystate改变时，就会触发onreadystateChange事件。**

**oajax.readyState：判断请求状态，返回的是5个整数（0，1，2，3，4），代表5种状态。**

0：未初始化：还没有调用open（）方法。

1：载入中：已经调用send方法，正在发送请求

2：载入完成，send（）方法已经完成，已经接收到响应内容。

3：解析中：浏览器正在解析响应内容

4：响应完成；响应内容已经完成，可以在浏览器调用了。

**oajax.status：**服务器返回结果，返回200或是404.(200:成功，404：未找到)

**ajax.responseText：**返回过来的内容，就是我们做需要读取的数据，responseText属性返回的是一个字符串。

前后台数据交互有4种常用格式

|  |
| --- |
| HTML |
| Text |
| XML |
| Json（主流，最常用） |

## JavaScript Ajax请求的Jason格式（要求必须会手写，熟练运用）：

在工作中，这种请求通常用函数封装起来，作为全局函数，便于以后调用。

在封装函数的时候，要判断是哪种请求（get/post）。如果是get，那么系统默认执行，如果是post请求，那么要加上一句话：oajax.setRequestHeader("content-type","application/x-www-form-urlencoded");

向服务器发送Ajax请求时，可以有两种不同的方式从服务器响应返回的数据：

一：XMLHttpRequest对象的responseXML属性访问XML格式的数据。

二：XMLHttpRequest对象的responseText属性访问字符串格式的数据。

当前，XML是已经被废除的进行数据传输的标准语言。XML操作复杂，解析复杂。

JSOAN（Jacascript Object Notation）是一种轻量级的数据传输格式。我们成为javascript对象表示法。使用JSON进行数据传输的优势之一是实际上JSON是一种javascript表示。

## http协议的服务器错误代码及其含义

|  |  |
| --- | --- |
| 1XX | 服务器已经获取了客户端的请求，但是还没有处理 |
| 2XX | 服务器返回浏览器请求成功 |
| 3XX | 重定向301 临时重定向，302永久重定向 |
| 4XX | 404找不到网页403禁止访问 |
| 5XX 6XX | 服务器内部发生错误 |

## 上传图片形式的文件：

const { uploadMerchantImageAct } = this.props;

var formData = new FormData();

formData.append("Filename", e.target.files[0].name);

formData.append("imgFile",e.target.files[0]);

uploadMerchantImageAct(formData);

# Fetch异步请求

## Fetch使用说明

W3Cschool：https://www.w3cschool.cn/fetch\_api/fetch\_api-5kdv2lfq.html

fetch(url, options).then(function(response) {

// handle HTTP response

}, function(error) {

// handle network error

});

说明：  
a. fetch api返回的是一个promise对象  
b.Options:

* method(String): HTTP请求方法，默认为GET
* body(String): HTTP的请求参数，有两种情况：

1，如果设置了Form-data形式的data,需要做一下处理：

由于fetch的body参数的形式是[[‘a’,1],[‘b’,2]...];

然后用 **new** URLSearchParams()方法处理一下。

所以要把普通的JSON对象转化为上面的形式。方法如下：

**function** **objtoString**(obj, arr = [], idx = 0) {

**for** (**let** item **in** obj) {

arr[idx++] = [item, obj[item]]

//arr.push([item, obj[item]];

)

}

**return** **new** URLSearchParams(arr).toString()

}

由上面的方法我们可以了解到，fetch执行post请求时，实质上是把post请求的参数转化为url参数，写到url里了。

2.如果设置了JSON形式的data，那么就用body: JSON.stringify(data)的形式。

* headers(Object): HTTP的请求头，默认为{}，通常有下面几种：

headers: **new** Headers({

'Accept': 'application/json',

'Content-Type': 'application/x-www-form-urlencoded'

Content-Type:text/plain;charset=UTF-8

}),

* credentials(String): 默认为omit,忽略的意思，也就是不带cookie;还有两个参数，same-origin，意思就是同源请求带cookie；include,表示无论跨域还是同源请求都会带cookie

c.第一个then函数里面处理的是response的格式，这里的response具体如下：

· status(number): HTTP返回的状态码，范围在100-599之间

· statusText(String): 服务器返回的状态文字描述，例如Unauthorized,上图中返回的是Ok。

· ok(Boolean): 如果状态码是以2开头的，则为true

· headers: HTTP请求返回头

· body: 返回体，这里有处理返回体的一些方法

* text(): 将返回体处理成字符串类型
* json()： 返回结果和 JSON.parse(responseText)一样
* blob()： 返回一个Blob，Blob对象是一个不可更改的类文件的二进制数据
* arrayBuffer()
* formData()

## 封装的fetch请求

/\*\*

\* 将对象转成 a=1&b=2的形式

\* @param obj 对象

\*/**function** **obj2String**(obj, arr = [], idx = 0) {

**for** (**let** item **in** obj) {

arr[idx++] = [item, obj[item]]

}

**return** **new** URLSearchParams(arr).toString()

}

/\*\*

\* 真正的请求

\* @param url 请求地址

\* @param options 请求参数

\* @param method 请求方式

\*/**function** **commonFetcdh**(url, options, method = 'GET') {

**const** searchStr = obj2String(options)

**let** initObj = {}

**if** (method === 'GET') { // 如果是GET请求，拼接url

url += '?' + searchStr

initObj = {

method: method,

credentials: 'include'

}

} **else** {

initObj = {

method: method,

credentials: 'include',

headers: **new** Headers({

'Accept': 'application/json',

'Content-Type': 'application/x-www-form-urlencoded'

}),

body: searchStr

}

}

fetch(url, initObj).then((res) => {

**return** res.json()

}).then((res) => {

**return** res

})

}

/\*\*

\* GET请求

\* @param url 请求地址

\* @param options 请求参数

\*/**function** **GET**(url, options) {

**return** commonFetcdh(url, options, 'GET')

}

/\*\*

\* POST请求

\* @param url 请求地址

\* @param options 请求参数

\*/function POST(url, options) {

return commonFetcdh(url, options, 'POST')

}

# 异步转同步

使用new Promise 与async/await是一种异步转同步的好方案：

大致思想：在Promise里的回掉函数里用resolve/reject函数把异步的结果返回出来，然后在另一个函数中用async/await调用，从而避免了陷入回调地狱的尴尬。

//fetch.js:

function fetchGET() {

return new Promise((resolve, reject) => {

//异步成功的回调：

resolve(res);

//异步失败的回调：

reject(res);

});

}

export default {

fget: fetchGET,

};

//Item.js:

import FETCH from '../utils/fetch';

async componentWillMount() {

const myRes = await fget();

console.log(myRes);

}

# JSON语法

JSON是一种数据格式。在JSON中，有两种结构：对象和数组。

1. 一个对象以“{”（左括号）开始，“}”（右括号）结束。每个“名称”后跟一个“:”（冒号）；“‘名称/值’ 对”之间运用 “,”（逗号）分隔。 名称用引号括起来；值如果是字符串则必须用括号，数值型则不须要。例如：

var o={"xlid":"cxh",

"xldigitid":123456,

"topscore":2000,

"topplaytime":"2009-08-20"

}；

2. 数组是值（value）的有序集合。一个数组以“[”（左中括号）开始，“]”（右中括号）结束。值之间运用 “,”（逗号）分隔。例如：

var jsonranklist=[

{"xlid":"cxh",

"xldigitid":123456,

"topscore":2000,

"topplaytime":"2009-08-20"

},{"xlid":"zd",

"xldigitid":123456,

"topscore":1500,

"topplaytime":"2009-11-20"

}

];

JSON格式的字符串:

    var str1 = '{ "name": "cxh", "sex": "man" }';

    JSON格式对象:

    var jsonObj= { "name": "cxh", "sex": "man" };

多个json数据集合用数组封装表示：

Var jsondata2=[

{“变量1”：“值1”，“变量2”：“值2”，“变量3”：“值3”，“变量4”：“值4”},

{“变量1”：“值1”，“变量2”：“值2”，“变量3”：“值3”，“变量4”：“值4”},

{“变量1”：“值1”，“变量2”：“值2”，“变量3”：“值3”，“变量4”：“值4”},

{“变量1”：“值1”，“变量2”：“值2”，“变量3”：“值3”，“变量4”：“值4”}

]

一个对象以{ “ ” }每两个对象后面用逗号“，”隔开，属性与值对之间用冒号“：”隔开,最后一个键值对后面不能加逗号（“，”），多个数据集合之间用逗号（“，”）隔开，最后一个数据集合后面不加逗号（“，”）。

例如：var ad1={

name：“Jordan”，

sex：“m”，

age：18，

job：”程序员”

}

Var ad2={

name：“奇瑞”，

Color：“yellow”，

Coast：5.89，

job：”UI设计师”

}

值的数据类型可以包含以下几种：

字符，“字符串”，数字，布尔值，null，DOM对象，数组。

2：json字面量的表达法:

Var add1={

name：“Jordan”，

sex：“m”，

age：18

}

JSON格式的数据实际上就是我们的JS对象。JSON格式的对象实例：

{

"message": "查找生肖信息列表成功",

"statusCode": 107,

"status": true,

"resource": [

{

"\_id": "59ad028b36c34d55c001d811",

"palaceid": 1,

"uid": "59a12166a6cc1020f06ff61d",

"ripe\_num": 59.76,

"type": 0,

"status": 1,

"futou": 0,

"ctime": 1504510603343,

"utime": 0,

"bonus": 600

},

{

"\_id": "59b3b70c4492f151186d8d9c",

"palaceid": 2,

"uid": "59a12166a6cc1020f06ff61d",

"ripe\_num": 14.94,

"type": 0,

"status": 1,

"futou": 0,

"ctime": 1504950028969,

"utime": 0,

"bonus": 600

},

{

"\_id": "59b3b7174492f151186d8da1",

"palaceid": 3,

"uid": "59a12166a6cc1020f06ff61d",

"ripe\_num": 14.94,

"type": 0,

"status": 1,

"futou": 0,

"ctime": 1504950039383,

"utime": 0,

"bonus": 600

}

]

}

二：使用XMLHttpRequest对象创建JSON数据请求

为了方便处理JSON数据，JSON提供了json。Js包。JSON是以文本，即字符串的形式传递的。所以JSON对象和JSON字符串之间的相互转换式关键。

## 将JSON格式字符串转化为JOSN格式的对象：

使用方法1：var ajsion=JSON.prase(oa);

或者使用方法2：var obj=str.parseJSON();

## 将JSON格式的对象转化为JSON格式的字符串：

可以运用 toJSONString()或者使用JSON.stringify()将JSON对象转化为JSON字符串,用于调试打印或者本地存储。

var last=obj.toJSONString(); //将JSON对象转化为JSON字符串。

var last=JSON.stringify(obj); //将JSON对象转化为JSON字符

# JQuery讲解

Jquery的操作方法包括以下5个方面：

1：JQuery选择器

2：JQUERY 操作Dom

3：jquery的动画操作

4：jquery对表单元素Ajax操作

5：jquery插件

## jQuery选择器与事件

一：认识什么是JavaScript框架

Javascript框架就是为了简化原生JS操作，世界上有很多简化JS的框架。

ProtoType 2004-2008年使用 开源基金会

Dojo Yahoo公司出品

YUI Yahoo公司出品

JQuery

二 下载并测试JQuery代码，并且和们的html文件放在同一个文件夹内。

Jquery1.x版本兼容IE6，7，8

Jquery2.x版本不兼容IE6，7，8

Jquery3.x版本不兼容IE6，7，8，支持移动端代码。

三：开始学习JQuery

1：引入JQuery3.0.0-js框架

<script type="text/javascript" src="jquery-3.0.0/jquery-3.0.0.js"></script>

外部文件的引用：

外部css文件的引用：<link rel="stylesheet" type="text/css" href="css/base.css">

外部jquery框架的引用：<script type="text/javascript" src="jquery-3.0.0/jquery-3.0.0.js"></script>

2：开始编写

$(document).Ready(function(){

});

$非常强大，可以抽取任何Dom对象,$(“”)就是选择器。

当页面Dom部分加载完成之后执行该函数。类似于window.onload=function(){ },但又有所不同。下面介绍两种的不同之处：

|  |  |  |
| --- | --- | --- |
| 不同指之处 | Window.onload=function(){ } | $(document).ready( function(){ }) |
| 执行时机 | 必须等待网页中所有内容加载完毕（包括图片，外部样式，外部JS等）才能执行 | 所有Dom结构绘制完毕后就可以执行，可能与Dom关联的东西没有加载完毕 |
| 编写个数 | 一个页面只能由于一个window.onload，即使有多个，也只执行最后一个 | 一个页面可以有多个$(document).ready( )，并且都可以执行 |
| 简化写法 | 没有 | $(function(){ })常用简写方法 |

四JQuery对象和Dom对象：

Jquery中页面跳转：（function（）{window.location.href=”网页地址”}）,和js一样

JQuery对象是通过JQuery包装Dom对象之后的对象，也就是说，JQuery对象和Dom对象是一样的，本质上都是js对象。把dom对象转化成jquery对象：$(“#one”).把jquery对象转化为对象：var dom=document.getElementById（“one”）；

五：jquery的选择器

### 1：基本选择器

|  |  |  |  |
| --- | --- | --- | --- |
| 选择器 | 描述 | 返回 | 案例 |
| #id名 | Id选择器，通过id选择jquery元素 | 单个id元素 | Var $select=$(“#one”) |
| .class名 | 类选择器 | 所有class类组成的一个数组 | Var $select=$(“.one”); |
| element名 | 标签选择器 | 所有标签名组成的数组 | $selector=$(“span”) |
| Element名 .class名 #id名 | 把标签和标签下面的元素合并在一起选择 | 把标签和标签下面的元素合并在一起的数组 | $(“span.one#one”) |

Jquery的每一个操作都是函数操作，所以都写成函数调用的方式。

把dom对象转化成jquery对象：$(“#one”).

把jquery对象转化为对象：var dom=document.getElementById（“one”）；

Dom=$dom.get(0)/dom=$dom.get[0];固定语法

例如：$(“#one”).text():

### 2：层次选择器

|  |  |
| --- | --- |
| $(“parent children”) | $(“div span”)选中该标签名下的所有后代span元素，不论子元素或孙子元素,返回一个数组 |
| $( “parent>child” ) | $( “div>p” )选取所有一阶p标签返回一个数组 |
| $(“prev+next”) | $( “div1+div2” )选取div1的下一个div |
| $(“prev~siblings”) | 选取所有的silbing元素 |

$(“div span ”).css(“属性” ，“属性值”)。

### 3：过滤选择器：

|  |  |  |
| --- | --- | --- |
| **基本过滤选择器** | | |
| **符号** | 说明 | 用法 |
| **:eq(index)**  **.eq(index)** | index是从0开始的一个数字，选择序号为index的元素。选择第一个匹配的元素。 | $(“li:eq(1)”). css(“background”,”red”)  或者$(“li”).eq(1) |
| **:gt(index)** | Index 是从0开始的一个数字，选择序号大于index的元素 | $(“li:gt(2)”). css(“background”,”red”) |
| **:lt(index)** | Index是从0开始的一个数字，选择小于index 的元素 | $(“li:lt(2)”). css(“background”,”red”) |
| **:odd** | 选择所有序号为奇数行的元素 | $(“li:odd”). css(“background”,”red”) |
| **:even** | 选择所有序号为偶数的元素 | $(“li:even”). css(“background”,”red”) |
| **:first** | 选择匹配第一个元素 | $(“li:first”). css(“background”,”red”) |
| **:last** | 选择匹配的最后一个元素 | $(“li:last”). css(“background”,”red”) |
| **:contains（‘’）** | 选择包含该字符的元素 | $(:contains('尼康')).css("background","pink") |

### 4：属性选择器：

|  |  |  |
| --- | --- | --- |
| **属性选择器** | | |
| **符号** | 说明 | 用法 |
| **$(“a[href]”)** | 选择所有包含href属性的元素 | $(“a[href]”). css(“background”,”red”) |
| **$(“a[href=’itcast’]”)** | 选择href属性值为itcast的所有a标签 | $(“a[href=’itcast’]”). css(“background”,”red”) |
| **$(“a[href!=’baidu’]”)** | 选择所有href属性不等baidu的所有元素，包括没有href的元素 | $(“a[href!=’baidu’]”). css(“background”,”red”) |
| **$(“a[href^=’web’]”)** | 选择所有以web开头的元素 | $(“a[href^=’web’]”). css(“background”,”red”) |
| **$(“a[href$=’cn’]”)** | 选择所有以cn结尾的元素 | $(“a[href$=’cn’]”). css(“background”,”red”) |
| **$(“a[href\*=’i’]”)** | 选择所有包含i这个字符的元素，可以是中英文 | $(“a[href\*=’i’]”). css(“background”,”red”) |
| **$(“a[href][title=’我’]”)** | 选择所有符合指定属性规则的元素，都符合才会被选中。 | $(“a[href][title=’我’]”). css(“background”,”red”) |
| **$("input[name='checked']:checked")/$(":checked")** | 选择属性为：checked的标签 | **$("input[name='checked']:checked")/$(":checked")** |

例：[name=’checked’]

### 5:兄弟选择器：

$(“’).siblings()当前元素的所有同级的兄弟节点

$(“’).prev()当前元素的前一个兄弟节点

$(“’).prevAll()当前元素之前的所有兄弟节点

$(“’).next()当前元素的后一个兄弟节点

$(“’).nextAll()当前元素之后的所有兄弟节点

### 6.后代选择器

.find(“.pep”)找到包含class属性为pep的后代元素并返回该元素组成的数组。

如果给定一个表示 DOM 元素集合的 jQuery 对象，.find() 方法允许我们在 DOM 树中搜索这些元素的后代，并用匹配元素来构造一个新的 jQuery 对象。.find() 与 .children() 方法类似，不同的是.children() 方法仅沿着 DOM 树向下遍历单一层级。

.find() 方法第一个明显特征是，其接受的选择器表达式与我们向 $() 函数传递的表达式的类型相同。将通过测试这些元素是否匹配该表达式来对元素进行过滤。

当调整浏览器窗口的大小时，发生 resize 事件。

resize() 方法触发 resize 事件，或规定当发生 resize 事件时运行的函数。

1： .show() 显示隐藏的元素

2 ： .css({属性：值,属性：值}) 给元素添加样式

3 ： .text() 没有参数的话就是获取标签的文本如果有参数的话就是设置标签的文本

4 ： .filter(exp) 筛选出与指定的选择器匹配的元素集合。可以有多个

5： .hide()隐藏元素

判断方法：

$("#id").is(“:visible”，boolean);       true 为显示 false 为隐藏

$("#id").is(":hidden"，boolean);       true 为隐藏 false 为显示

$("#id").is(":checked"，true);  true为选中

$("#id").is(":checked"，false);  false为未选中

补充css代码：

cursor: pointer;当鼠标悬停在被点击元素上时，鼠标箭头变成手指形态。

.indexOf("字符 ")找到包含该字符的元素

.contains(“字符”)找到含有指定字符的元素。

## jQuery操作Dom

一般来说，DOM的操作主要分3种，即D'OM Core（核心），HTML-DOM和CSS-DOM

1：查找设置标签节点

var $xj=$("ul li:eq(1)");

alert($xj.text());

.val():内容，指获取input标签中的内容或value属性值。

.html()：设置html标签样式。

.text()打印文本内容。

.html()与.text()都是方法，要直接在括号内写内容。

.html()与.text()都可以动态获取并设置dom节点的内容，不同之处是.text()只能获取或设置文本内容，而.html()不仅可以动态获取或者设置文本内容，还可以动态设置标签。

2：查找和设置属性节点

1. attr(“属性名”，”值”)设置所有行内属性和值。针对于对于自己定义的属性（比如src，data，height，width），一般是class,id,name等。

设置单个attr属性：$().attr(“key”,”value”);

设置多个属性：$().attr({“key1”：”value1”，“key2”：”value2”，.........});

②对于html固有的属性，用prop（“属性名”，“值”）方法。

对于表单元素（input，a，form，select（下拉列表），button，datalist），name才是其固有属性，否则是可有可无的。

③.css（属性名”，”值”）是更改style标签里的css样式表里的css属性,一般用于更改某一项css的样式属性。

编辑多个CSS样式：

.css({“key1”:”value1”,“key2”:”value2”,“key3”:”value3”...})

.addClass(“”)添加一个class；

.removeClass(“属性名”)删除该class。

②.prop(“属性名”，boolean)，设置节点的原本属性。

jQueryObject.prop( propertyName [, value ] )

设置或返回指定属性propertyName的值。如果指定了value参数，则表示设置属性propertyName的值为value；如果没有指定value参数，则表示返回属性propertyName的值。

参数value还可以是函数，prop()将根据匹配的所有元素遍历执行该函数，函数中的this指针将指向对应的DOM元素。prop()还会为函数传入两个参数：第一个参数就是该元素在匹配元素中的索引，第二个参数就是该元素propertyName属性当前的值。函数的返回值就是为该元素的propertyName属性设置的值。

用法二：

jQueryObject.prop( object )

以对象形式同时设置任意多个属性的值。对象object的每个属性对应propertyName，属性的值对应value。

注意：prop()函数的所有"设置属性"操作针对的是当前jQuery对象所匹配的每一个元素；所有"读取属性"的操作只针对第一个匹配的元素。

返回值

prop()函数的返回值是任意类型，返回值的类型取决于当前prop()函数执行的是"设置属性"操作还是"读取属性"操作。

如果prop()函数执行的是"设置属性"操作，则返回当前jQuery对象本身；如果是"读取属性"操作，则返回读取到的属性值。

如果当前jQuery对象匹配多个元素，返回属性值时，prop()函数只以其中第一个匹配的元素为准。如果该元素没有指定的属性，则返回undefined。

prop()和attr()的主要区别：prop()函数针对的是DOM元素(JS Element对象)的属性，[attr()](http://www.365mini.com/page/jquery-attr.htm" \o "jQuery.attr() 函数详解" \t "http://www.365mini.com/page/_blank)函数针对的是DOM元素所对应的文档节点的属性。详情请查看[jQuery函数attr()和prop()的区别](http://www.365mini.com/page/jquery-attr-vs-prop.htm" \o "jQuery函数attr()和prop()的区别" \t "http://www.365mini.com/page/_blank)。

**注意事项**

①如果通过prop()函数更改<input>和<button>元素的type属性，在多数浏览器上将会抛出一个错误，因为该属性一般不允许在后期更改。

②如果使用prop()函数操作表单元素的checked、selected、disabled等属性，如果该元素被选中(或禁用)，则返回true，否则(意即HTML中没有该属性)返回false。

③prop()函数还可以设置或返回DOM元素的Element对象上的某些属性，例如：tagName、selectedIndex、nodeName、nodeType、ownerDocument、defaultChecked和defaultSelected等属性。

④在IE9及更早版本中，如果使用prop()函数设置的属性值不是一个简单的原始值(String、Number、Boolean)，并且在对应的DOM元素被销毁之前，该属性没有被移除，则可能会导致内存泄漏问题。如果你只是为了存储数据，建议你使用[data()](http://www.365mini.com/page/jquery-data.htm" \o "jQuery.data() 函数详解" \t "http://www.365mini.com/page/_blank)函数，以避免内存泄漏问题。

3：动态创建和追加节点（文本，元素，属性都可以）

.append(“<></>”)

$li.append(“<p title=“te” name=“pb”>我是一段文字</p>”)

append详解

|  |  |
| --- | --- |
| .append（“”） | 向某个元素里面追加新的元素（最后一位） |
| .appendTo(“”) | 将所有的匹配元素剪切追加到原有标签里边去，原来的被删除了。 |
| .prepend(“<></>”) | 向某个元素里添加新的元素（最前面一位） |
| .prependTo(“”) | 将某个元素追加到原有标签里边去（最前面一位） |
| $(“”).after(<></>“”) | 将所匹配的标签插入到制定标签的前面。 |
| $(“<></>”).insertAfter（“p”） | 将所有匹配的元素插入到指定元素的后面，实际上，使用该方法颠倒了常规的$().after() |
| $(b”).before(“<p></p>”) | b在p标签前面 |
| $(“b”)insertbefore(<></>) | b在p后面 |

4:删除节点

$(“ul li.eq(1)”).removeClass(“属性名”);根据属性名将所选中的标签删除。如果没有属性名称的话，那么就是默认删除所有属性。

被删除的节点看可以被接受，还可以添加到别的别的元素里边去。这就说明，.remove方法不是真的删除。

5：Empty方法：

.empty()

Empty方法并不是删除节点，而是清空节点，它能清空该标签元素所有后代节点

6：复制节点

$(“”).clone()

.clone()通常与.appendTo(“”)配合使用

$(“”).clone().appendTo(“”)；

7:替换节点

$(“”).replacewith(“<></>”)

将后面的标签替换前面的标签。

$(“<></>”)replaceAll(“”)

将前面的标签替换后面的标签

8：$(“strong”).wrap(“<b></b>”)分别包裹$(“strong”).warpAll(“<b></b>”)将所有的标签用b标签包裹。

9：追加设置属性：

$(“p”).attr(“class ”,”on heigh”)这时p标签具有两个属性，on和heigh。

$(“p”).addClass(“类名”)追加样式

设置和获取html文本和值

获取HTML：$(“p”).Html()

获取文字：$(“p”).text()

获取文本内容：$(“p”).val()

10：添加子节点

.children（）选中所有一阶子节点

.next和nextAll()取得匹配元素后面的一个同级元素/所有的同级元素

.prev和prevAll()取得匹配元素前面的一个同级元素/所有的同级元素

$(“p”).closest（“标签名”）获取离鼠标最近的标签的父级标签名。

11：css的编辑

①如果只有一个属性：

.css(“属性1”：“值”）

②如果有多个属性：

.css({

“属性1”：“值”，

“属性2”：“值”，

“属性3”：“值”

。。。。。。

})

.offset();获取或设置当前对象距离它的已经定位了的父级对象的坐标；如果没有定位的父级对象，则以body为准。

.offset().left：距离父级元素的左边框距离

.offset().top：距离父级元素的上边框距离

12：事件函数（event）

.bind(“事件名”，function（）{ })给某个标签绑定函数（写死了，可以多次使用），就相当于js中的事件函数声名和调用。

或者$(slector).on(“click”,function(){ });

.each(function(index,el){ })对每一个选中的对象添加回调函数事件，相当于for循环.

.click /.dblclick

.mouseover/out

.onkeyup

Js方式 :document.getElementById(“id”).onclick

jQuery方式: $(“#id”).click

区别：jQuery的事件不带on

事件处理程序

Js 书写方式:

document.getElementById(“id”).onclick = function(){

// 语句

}

jQuery 书写方式:

$(“#id”).click(function(){

// 语句

});

还有一种写法：

$(“p”).unbind(\_”事件名”，function（）{ })移除某一个函数。

## jQuery里的$(this)

在原生js里，this指向函数的调用者。在操作Dom的时候，this指向选择器选择出来的Html标签。但是jQuery里，$(this)指向jQuery选择器，而不再指向你选择出来的标签。如果想在each（）循环里点击打印标签的index，那么就还是用原生js的index赋值方法。不多比比，上代码：

<div class="dd">1111</div>

<div class="dd">2222</div>

<div class="dd">3333</div>

<div class="dd">4444</div>

$(".dd").each(function(index,item){

item.index=index;

$(item).click(function(){

console.log(this);//Html标签

console.log(this.index);//index下标

console.log($(this));//$选择器：r.fn.init [div.dd]

console.log($(this).index())//n，不确定。这个写法有个bug：index()方法打印出来的不是相对于你规定的那些元素中的洗标，而是相对于页面boday下所有的Html标签下标。

})

})

## 点击失效的解决方法：

Ajax从服务器里获取数据后，动态创建Dom元素并添加点击事件失效问题的解决方法：

$(“父元素”).on(“click”,”被点击的元素”,function(){

})

# JQuery的动画效果

模拟操作

前面学过的所有例子必须都是通过用户出发才能执行。但有时需要通过模拟操作来达到效果，例如在用户进入页面内就能触发click事件，而不需要用户主动点击。

.trigger(“事件名”)；：当页面加载完毕时自动执行事件函数，写在函数体的最后面。

模拟事件不关心事件函数名是否为关键字，只关心时间函数体内的代码。所以这个事件名可以是自定义的事件名（用户触发不可用，仅限于自执行）。

或者直接 $(“#btn”).事件名( );

含参函数的声名与调用：

$(“#btn”).bind(“myclick”,function(eventmesg1,mesg2){ })

为一个元素绑定多个事件

$(“button”).bind(“事件1 事件2”，function（）{ })

这个不常用，常用：

$(“button”).bind(“事件1”，function（）{ }).bind(“事件2”，function（）{ })......

动画效果：

display：none;

display：block;

.is（：”didden”）

.is（：”visiable”）

.is（“：animated”）判断是否是当前的动画效果。

.hide(3000) 3秒之内消失

.show(3000) 3秒之内呈现

.toggle(3000)对选中元素隐藏/显示的切换

$(selector).fadeOut(speed,callback);在一段时间内逐渐消失

可选的 speed 参数规定效果的时长。它可以取以下值："slow"、"fast" 或毫秒数。可选的 callback 参数是 fading 完成后所执行的函数名称。

$(selector).fadeIn(speed,callback);在一段时间内逐渐出现。

可选的 speed 参数规定效果的时长。它可以取以下值："slow"、"fast" 或毫秒数。可选的 callback 参数是 fading 完成后所执行的函数名称。

$(selector).fadeIn();在一段时间内逐渐出现。

$(selector).fadeIn(speed,callback);

可选的 speed 参数规定效果的时长。它可以取以下值："slow"、"fast" 或毫秒数。可选的 callback 参数是 fading 完成后所执行的函数名称。

.slideDown(3000)从上往下卷动显示

slideUp(3000)从下往上卷动隐藏

## jQuery slideDown() 方法

jQuery slideDown() 方法用于向下卷动显示。

$(selector).slideDown(speed,callback);

可选的 speed 参数规定效果的时长。它可以取以下值："slow"、"fast" 或毫秒。

可选的 callback 参数是滑动完成后所执行的函数名称。

slideUp与slideDown用法相同。

## jQuery slideUp() 方法

jQuery slideDown() 方法用于向上卷动隐藏。

$(selector).slideDown(speed,callback);

可选的 speed 参数规定效果的时长。它可以取以下值："slow"、"fast" 或毫秒。

可选的 callback 参数是滑动完成后所执行的函数名称。

slideUp与slideDown用法相同。

## jQuery slideToggle() 方法

jQuery slideToggle() 方法可以在 slideDown() 与 slideUp() 方法之间进行切换。

$(selector).slideToggle(speed,callback);

可选的 speed 参数规定效果的时长。它可以取以下值："slow"、"fast" 或毫秒。

可选的 callback 参数是滑动完成后所执行的函数名称。

## jQuuery animate方法

animate（parms，speed，callback，3000）一坨参数，执行自定义CSS属性集的动画。

1. params:一个包含样式属性以及值的映射，比如{background：}
2. speed速度参数
3. callback：可选，在animate动画效果执行完毕之后要执行的函数，可选

效果：.stop().animate（{”left“：500px}，3000）;3秒室内移动到据左边框500（向右移动500）

.stop().animate（{“left”：”+=500px”}，3000）3秒之内在当前位置基础上向右移500px;

opacity：设置透明度:0-1之间，小数表示。

opacity：1透明度为1

$(“div”).stop( ).animate({ },300);在当前动画效果的时候停止。

.is（“：animated”）判断是否是当前的动画效果。

$(dindow).scrollTop()/Left():据页面已经向上/向左卷动的距离。

.hover(function(){ }，function（）{ }):鼠标移入/移出时的事件。相当于.mouseover(function(){ }).mouseout(function(){ })

$(“”).find（“”）获取某个节点下面的子节点

.height()/width()设置节点的宽/高

$(window).height/width()获取浏览器窗口高度

# jQuery中的ajax

AJAX 是一种用于创建快速动态网页，请求网页数据的技术。

通过在后台与服务器进行少量数据交换，AJAX 可以使网页实现异步更新。这意味着可以在不重新加载整个网页的情况下，对网页的某部分进行更新。

传统的网页（不使用 AJAX）如果需要更新内容，必需重载整个网页面。

有很多使用 AJAX 的应用程序案例：新浪微博、Google 地图、开心网等等。

## 1.回顾原生js中的AJAX

①好处：

1：AJAX的实现不需要任何外部插件。

2：用户体验非常好

3：提高Web程序性能

4：减轻服务器贷款的负担

②不足之处：

1：浏览器对XMLHttpRequest对象的支持不同

2：破坏浏览器的前进/后退按钮的使用。

3：对搜索引擎的支持不足

4：开发和调试困难

原生AJAX代码：

①：**创建浏览器的ajax对象**

**var oajax=new XMLHttpRequest();//保留了最原始的xml格式，非IE浏览器适用**

**//IE适用于 var oajax=new ActiveXObject("Microsoft.XMLHTTP");**

**var oajax;**

**if (window.XMLHttpRequest) { //判断是否为非IE浏览器,如果是非IE浏览器，那么就往下执行**

**oajax=XMLHttpRequest()**

**}**

**else{//如果不是，那么就写为IE浏览器的一部对象声明。**

**oajax=new ActiveXObject("Microsoft.XMLHTTP");**

**}**

②：**连接服务器**

通过 .open（）方法。Open（）方法有3个参数，第1个参数是请求方式(GET/POST)第2个参数是URL地址，即所要传递的服务器地址，第3个参数是 是否异步，布尔值，true是异步，false为同步。

语法：

**oajax.open(“Get/Post”,“url地址”,true);**

③发送请求，一句话：

**oajax.send(data)**

这句话非常关键。

浏览器从读到这句话开始，浏览器与服务器就开始进行数据交互并且执行判断5步状态了，而且是循环执行5次。

④接收返回值：

Oajax.onreadystateChange=function(){

If(oajax.readyState==4){

If(oajax.status==200){

responseText()

}

}

}

**.onreadystatechange事件，当请求被发送到服务器端时，我们需要执行一些基于响应的任务。每当readyState改变时，就会触发onreadystatechange事件。**

**.readyState：判断请求状态，返回的是5个整数（0，1，2，3，4），代表5种状态。**

0：未初始化：还没有调用open（）方法。

1：载入中：已经调用send方法，正在发送请求

2：载入完成，send（）方法已经完成，已经接收到响应内容。

3：解析中：浏览器正在解析响应内容

4：响应完成；响应内容已经完成，可以在浏览器调用了。

.status：服务器返回结果，返回200或是404.(200:成功，404：未找到)

ajax.responseText：返回过来的内容，就是我们做需要读取的数据，responseText属性返回的是一个字符串。

1.open(method,url,[async,username,password]);

method:get/post;

url:指定服务端的链接；

async:false/true(默认)；true表示异步,false表示同步

username，password可选参数,指定服务器的用户名和密码;

2.setRequestHeader(label,value);

给该方法的请求头添加标记/值对;

3.send('content');

核心方法,发送请求,附带相应的数据

4.getAllResponseHeaders

getAllResponseHeaders();

字符串的形式,返回所有http应答的头,内容,服务，

日期等相关的信息

5.getResponseHeader

getResponseHeaders(lable);

字符串的形式,获取指定http应答头的信息

6.abort

abort(),取消当前的请求

1.XMLHttpRequest.open()中的method，常用

get，post；

但是还包含其他的方法

get:服务端请求数据；

post:向服务端发送数据；

delete：删除指定资源的数据；

put：数据保存到指定的资源上;

head，和get相似,但是服务器端应答的只是头而已；

其中最主要使用的还是get/post

2.XMLHttpRequest对象中的属性：

1.onreadystatechange

保存请求的ready状态改变的时候,调用的函数

2.readyState

五个值

0 没有初始化

1 open阶段

2 请求已经发送

3 正在接受应答

4 应答已经收到

注意:大部分情况下,我们只需要关心他的值为4的情况

3.responseText

文本格式的应答信息

4.responseXML

xml格式的应答信息，看做有效的xml处理

5.status

返回http请求的状态

404，500，等等

最期待的是200(因为200表示一切正常)

6.statusText

以文字的形式请求状态

## 处理web请求

发送Ajax请求之后,下一步就要开始处理相应的请求

1.获取服务器返回的数据

XMLHttpRequest.onreadystatechange = function(){}

(后面也可以是函数名,函数另外设定,一般使用回调函数)

注意:其实按照前文所说,是状态改变的时候，后面跟着需要调用的函数

也就是创建对象，发送数据完结之后,需要执行的回调函数而已

它最重要的一点就是函数的调用时机,状态改变的时候调用

2.回调函数的设定

XMLHttpRequest.onreadystatechange = function(){

//按照前文,其实也就是判断应答是否已经收到,传输过程是否是一切正常

if(XMLHttpRequest.readyState == 4 && XMLHttpRequest.status == 200)

//如果应答收到,一切正常,那么执行页面需要执行的程序即可

//如果需要获取服务器返回的数据

var data = XMLHttpRequest.responseText;

}

## 2.JQuery中的Ajax

JQuery中的Ajax是对原生Ajax进行了封装

1：$(“.classname”).load(”url“，{data}，callbackfunction（）{})

Laod(“url“,{参数1，参数2.。。}，服务器返回值)url为外部html文件名，参数为你所需要的标签内容，可选。如果没有参数，那么默认是加载整个外部html文件，浏览器默认的是get请求。如果写了参数，那么就变成了post请求。

通过.load()方法来调用外部的html文件并插入到Dom中。网页上很多评论都是这种方法。

2：$.post( )和$.get( )

$.get/$.post(url,{data} , callback(function(){ }) ,type)

这种方法最简单

|  |  |  |
| --- | --- | --- |
| 参数名称 | 类型 | 说明 |
| url | String | 请求的HTML页面的URL地址 |
| Data(可选) | Object | 发送到服务器key/value数据会作为  url?参数名称=参数值& |
| Callback（可选） | Function | 当载入成功后，自动将请求的结果通过function参数的形式返回给html |

jQuery get() 和 post() 方法用于通过 HTTP GET 或 POST 请求从服务器请求数据。

## HTTP 请求：GET vs. POST

两种在客户端和服务器端进行请求-响应的常用方法是：GET 和 POST。

* **GET** - 从指定的资源请求数据
* **POST** - 向指定的资源提交要处理的数据

GET 基本上用于从服务器获得（取回）数据。注释：GET 方法可能返回缓存数据。

POST 也可用于从服务器获取数据。不过，POST 方法不会缓存数据，并且常用于连同请求一起发送数据。

如需学习更多有关 GET 和 POST 以及两方法差异的知识，请阅读我们的 [HTTP 方法 - GET 对比 POST](http://www.w3school.com.cn/tags/html_ref_httpmethods.asp" \o "HTTP 方法：GET 对比 POST)。

## jQuery $.get() 方法

$.get() 方法通过 HTTP GET 请求从服务器上请求数据。

$.get(*URL*,*callback*);

必需的 *URL* 参数规定您希望请求的 URL。

可选的 *callback* 参数是请求成功后所执行的函数名。

下面的例子使用 $.get() 方法从服务器上的一个文件中取回数据：

### 实例

$("button").click(function(){

$.get("demo\_test.asp",function(data,status){

alert("Data: " + data + "\nStatus: " + status);

});

});

$.get() 的第一个参数是我们希望请求的 URL（"demo\_test.asp"）。

第二个参数是回调函数。第一个回调参数存有被请求页面的内容，第二个回调参数存有请求的状态。

**提示：**这个 ASP 文件 ("demo\_test.asp") 类似这样：

<%

response.write("This is some text from an external ASP file.")

%>

## jQuery $.post() 方法

$.post() 方法通过 HTTP POST 请求从服务器上请求数据。

### 语法：

$.post(*URL*,*data*,*callback*);

必需的 *URL* 参数规定您希望请求的 URL。

可选的 *data* 参数规定连同请求发送的数据。

可选的 *callback* 参数是请求成功后所执行的函数名。

下面的例子使用 $.post() 连同请求一起发送数据：

### 实例

$("button").click(function(){

$.post("demo\_test\_post.asp",

{

name:"Donald Duck",

city:"Duckburg"

},

function(data,status){

alert("Data: " + data + "\nStatus: " + status);

});

});

## jQuery Ajax 操作函数

jQuery 库拥有完整的 Ajax 兼容套件。其中的函数和方法允许我们在不刷新浏览器的情况下从服务器加载数据。

|  |  |
| --- | --- |
| 函数 | 描述 |
| [jQuery.ajax()](http://www.w3school.com.cn/jquery/ajax_ajax.asp" \o "jQuery ajax - ajax() 方法) | 执行异步 HTTP (Ajax) 请求。 |
| jQuery[.ajaxComplete()](http://www.w3school.com.cn/jquery/ajax_ajaxcomplete.asp" \o "jQuery ajax - ajaxComplete() 方法) | 当 Ajax 请求完成时注册要调用的处理程序。这是一个 Ajax 事件。 |
| jQuery[.ajaxError()](http://www.w3school.com.cn/jquery/ajax_ajaxerror.asp" \o "jQuery ajax - ajaxError() 方法) | 当 Ajax 请求完成且出现错误时注册要调用的处理程序。这是一个 Ajax 事件。 |
| jQuery[.ajaxSend()](http://www.w3school.com.cn/jquery/ajax_ajaxsend.asp" \o "jQuery ajax - ajaxSend() 方法) | 在 Ajax 请求发送之前显示一条消息。 |
| [jQuery.ajaxSetup()](http://www.w3school.com.cn/jquery/ajax_ajaxsetup.asp" \o "jQuery ajax - ajaxSetup() 方法) | 设置将来的 Ajax 请求的默认值。 |
| [.ajaxStart()](http://www.w3school.com.cn/jquery/ajax_ajaxstart.asp" \o "jQuery ajax - ajaxStart() 方法) | 当首个 Ajax 请求完成开始时注册要调用的处理程序。这是一个 Ajax 事件。 |
| [.ajaxStop()](http://www.w3school.com.cn/jquery/ajax_ajaxstop.asp" \o "jQuery ajax - ajaxStop() 方法) | 当所有 Ajax 请求完成时注册要调用的处理程序。这是一个 Ajax 事件。 |
| [.ajaxSuccess()](http://www.w3school.com.cn/jquery/ajax_ajaxsuccess.asp" \o "jQuery ajax - ajaxSuccess() 方法) | 当 Ajax 请求成功完成时显示一条消息。 |
| [jQuery.get()](http://www.w3school.com.cn/jquery/ajax_get.asp" \o "jQuery ajax - get() 方法) | 使用 HTTP GET 请求从服务器加载数据。 |
| [jQuery.getJSON()](http://www.w3school.com.cn/jquery/ajax_getjson.asp" \o "jQuery ajax - getJSON() 方法) | 使用 HTTP GET 请求从服务器加载 JSON 编码数据。 |
| [jQuery.getScript()](http://www.w3school.com.cn/jquery/ajax_getscript.asp" \o "jQuery ajax - getScript() 方法) | 使用 HTTP GET 请求从服务器加载 JavaScript 文件，然后执行该文件。 |
| [.load()](http://www.w3school.com.cn/jquery/ajax_load.asp" \o "jQuery ajax - load() 方法) | 从服务器加载数据，然后把返回到 HTML 放入匹配元素。 |
| [jQuery.param()](http://www.w3school.com.cn/jquery/ajax_param.asp" \o "jQuery ajax - param() 方法) | 创建数组或对象的序列化表示，适合在 URL 查询字符串或 Ajax 请求中使用。 |
| [jQuery.post()](http://www.w3school.com.cn/jquery/ajax_post.asp" \o "jQuery ajax - post() 方法) | 使用 HTTP POST 请求从服务器加载数据。 |
| [.serialize()](http://www.w3school.com.cn/jquery/ajax_serialize.asp" \o "jQuery ajax - serialize() 方法) | 将表单内容序列化为字符串。 |
| [.serializeArray()](http://www.w3school.com.cn/jquery/ajax_serializearray.asp" \o "jQuery ajax - serializeArray() 方法) | 序列化表单元素，返回 JSON 数据结构数据。 |

如果需要使用$.ajax()方法进行ajax开发，那么上表这些常用参数必须了解。

Ajax请求：$.ajax({

Type:”POST/GET”;必选。

dataType：“json”，

data:{“type”：num，“key1”:”value1”,”key2”:”value2”,.....},必选，这里的type值是服务器给的

Url:服务器地址，必选。

以上三步是向服务器请求数据。可能与用户输入的数据相同。

Success：function（data）{成功之后要有变量名接收返回成功的数据。

var jsonObject=JSON.parse(data);//数模转换，将数据库里请求的数据转换成json对象。这里的data就是请求的data。

Console.log(data);

Var dat=jsonObject.things; things就是json对象中的子对象。

}

error：function（变量名）{错误之后要有变量名接收返回的信息提示。

Alert（变量名.status）;变量名.status是服务器返回给浏览器的状态。

}

})

如果是“POST”请求：data:{“type”：num，“key1”:”value1”,”key2”:”value2”,.....},必选，这里的type值是服务器给的。

如果是“GET”请求：data数据里的type要写到url地址中。

## JQuery中$.ajax()方法参数详解

1.url: 要求为String类型的参数，（默认为当前页地址）发送请求的地址。

2.type: 要求为String类型的参数，请求方式（post或get）默认为get。注意其他http请求方法，例如put和delete也可以使用，但仅部分浏览器支持。

timeout: 要求为Number类型的参数，设置请求超时时间（毫秒）。此设置将覆盖$.ajaxSetup()方法的全局设 置。

async：要求为Boolean类型的参数，默认设置为true，所有请求均为异步请求。

       如果需要发送同步请求，请将此选项设置为false。注意，同步请求将锁住浏览器，用户其他操作必须等待请求完成才可以执行。

cache：要求为Boolean类型的参数，默认为true（当dataType为script时，默认为false）。设置为false将不会从浏览器缓存中加载请求信息。

data: 要求为Object或String类型的参数，发送到服务器的数据。如果已经不是字符串，将自动转换为字符串格式。get请求中将附加在url后。防止这种自动转换，可以查看processData选项。对象必须为key/value格式，例如{foo1:"bar1",foo2:"bar2"}转换为&foo1=bar1&foo2=bar2。如果是数组，JQuery将自动为不同 值对应同一个名称。例如{foo:["bar1","bar2"]}转换为&foo=bar1&foo=bar2。

dataType: 要求为String类型的参数，预期服务器返回的数据类型。如果不指定，JQuery将自动根据http包mime  信息返回responseXML或responseText，并作为回调函数参数传递。

         可用的类型如下：

          xml：返回XML文档，可用JQuery处理。

          html：返回纯文本HTML信息；包含的script标签会在插入DOM时执行。

          script：返回纯文本JavaScript代码。不会自动缓存结果。除非设置了cache参数。注意在远程请求时（不在同一个域下），所有post请求都将转为get请求。

          json：返回JSON数据。

          jsonp：JSONP格式。使用SONP形式调用函数时，例如myurl?callback=?，JQuery将自动替换后一个?”为正确的函数名，以执行回调函数。

   text：返回纯文本字符串。

beforeSend：要求为Function类型的参数，发送请求前可以修改XMLHttpRequest对象的函数，例如添加自定义 HTTP头。在beforeSend中如果返回false可以取消本次ajax请求。XMLHttpRequest对象是惟一的参  数。

  function(XMLHttpRequest){

     this;   //调用本次ajax请求时传递的options参数

  }

complete：要求为Function类型的参数，请求完成后调用的回调函数（请求成功或失败时均调用）。

 参数：XMLHttpRequest对象和一个描述成功请求类型的字符串。

function(XMLHttpRequest, textStatus){

this;    //调用本次ajax请求时传递的options参数

          }

success：要求为Function类型的参数，请求成功后调用的回调函数，有两个参数。

         (1)由服务器返回，并根据dataType参数进行处理后的数据。

         (2)描述状态的字符串。

         function(data, textStatus){

            //data可能是xmlDoc、jsonObj、html、text等等

            this;  //调用本次ajax请求时传递的options参数

error：要求为Function类型的参数，请求失败时被调用的函数。该函数有3个参数，即XMLHttpRequest对象、错 误信息、捕获的错误对象(可选)。

       ajax事件函数如下：

       function(XMLHttpRequest, textStatus, errorThrown){

          //通常情况下textStatus和errorThrown只有其中一个包含信息

          this;   //调用本次ajax请求时传递的options参数

       }

contentType：要求为String类型的参数，当发送信息至服务器时，内容编码类型默认为"application/x-www-form-urlencoded"。该默认值适合大多数应用场合。

dataFilter：要求为Function类型的参数，给Ajax返回的原始数据进行预处理的函数。 提供data和type两个参数。data是Ajax返回的原始数据，type是调用jQuery.ajax时提供的dataType参数。函数返回的值将由jQuery进一步处理。

  function(data, type){

                //返回处理后的数据

                return data;

            }

global：要求为Boolean类型的参数，默认为true。表示是否触发全局ajax事件。设置为false将不会触发全局

  ajax事件，ajaxStart或ajaxStop可用于控制各种ajax事件。

ifModified：要求为Boolean类型的参数，默认为false。仅在服务器数据改变时获取新数据。 服务器数据改变判断的依据是Last-Modified头信息。默认值是false，即忽略头信息。

jsonp：要求为String类型的参数，在一个jsonp请求中重写回调函数的名字。

该值用来替代在"callback=?"这种GET或POST请求中URL参数里的"callback"部分，例如

{jsonp:'onJsonPLoad'}会导致将"onJsonPLoad=?"传给服务器。

username：要求为String类型的参数，用于响应HTTP访问认证请求的用户名。

password：要求为String类型的参数，用于响应HTTP访问认证请求的密码。

processData：要求为Boolean类型的参数，默认为true。默认情况下，发送的数据将被转换为对象（从技术角度

             来讲并非字符串）以配合默认内容类型"application/x-www-form-urlencoded"。如果要发送DOM树信息或者其他不希望转换的信息，请设置为false。

scriptCharset：要求为String类型的参数，只有当请求时dataType为"jsonp"或者"script"，并且type是GET时才会用于强制修改字符集(charset)。通常在本地和远程的内容编码不同时使用。

## 表单序列化

目的是为了解决表单元素的值太多，一一获取组件参数列表太累，随意提供了一个叫做serialize()的表单序列化的方法。

$.post(“”,$(“#form id名”)，返回值)

$（）.load(“url”)方法：

.load方法用于加载新页面内容或引入新页面内容的一部分。

在node.js中，ajax请求的地址url都是路由地址。

一、serialize()定义和用法：

serialize()方法通过序列化表单值，创建标准的URL编码文本字符串，它的操作对象是代表表单元素集合的jQuery 对象。你可以选择一个或多个表单元素（比如input或文本框），或者 form 元素本身。序列化的值可在生成 AJAX 请求时用于 URL 查询字符串中。

语法：

$(selector).serialize()

详细说明

1、.serialize() 方法创建以标准 URL 编码表示的文本字符串。它的操作对象是代表表单元素集合的 jQuery 对象。

2、.serialize() 方法可以操作已选取个别表单元素的 jQuery 对象，比如 <input>, <textarea> 以及 <select>。不过，选择 <form> 标签本身进行序列化一般更容易些

3、只会将”成功的控件“序列化为字符串。如果不使用按钮来提交表单，则不对提交按钮的值序列化。如果要表单元素的值包含到序列字符串中，元素必须使用 name 属性。

4、form里面的name不能够用 Js、jquery里的关键字。

二、JQuery中serialize()实例

1、ajax serialize()

复制代码代码如下:

$.ajax({  
    type: "POST",  
    dataType: "json",  
    [url:ajaxCallBack](http://www.jb51.net/article/ajaxCallBack),  
    data:$('#myForm').serialize(),// 要提交表单的ID

//Console.Log(data),  
    success: function(msg){  
        alert(msg);  
    }  
});

三、serialize是用param方法对serializeArray的一个简单包装

1、$.param()

$.param()方法是serialize()方法的核心，用来对一个数组或对象按照key/value进行序列化。

param方法的js代码

2、serializeArray

serializeArray方法是将一个表单当中的各个字段序列化成一个数组

serializeArray方法的jquery定义

**1、serialize()方法**

描述：序列化表单内容为字符串,用于Ajax请求。

格式：var data = $(form).serialize();

**2.serializeArray()方法**

描述：序列化表单元素(类似'.serialize()'方法)返回JSON数据结构数据。

注意,此方法返回的是JSON对象而非JSON字符串。需要使用插件或者第三方库进行字符串化操作。

格式：var jsonData = $(form).serializeArray();

在使用ajax提交表单数据时,以上两种方法都可以将data参数设置为$(form).serialize()或$(form).serializeArray()。

1.serialize()方法

　　格式：var data = $("form").serialize();

　　功能：将表单内容序列化成一个字符串。

　　这样在ajax提交表单数据时，就不用一一列举出每一个参数。只需将data参数设置为 $("form").serialize() 即可。

2.serializeArray()方法

　　格式：var jsonData = $("form").serializeArray();

　　功能：将页面表单序列化成一个JSON结构的对象。注意不是JSON字符串。

　　比如，[{"name":"lihui", "age":"20"},{...}] 获取数据为 jsonData[0].name

# offset家族

## 1.offset属性

offset家族是建立在元素定位基础之上的，offset家族的性质都以定位为基本。

1.1：.offsetWidth/Height：得到对象的宽/高属性，返回宽/高的值。

offsetWidth/Height=Width/Height+border+padding。

1.2.offsetLeft/.offsetTop:获取距离父亲级盒子（最近已经定位了的父级盒子）左边/顶端距离。

offsetLeft/.offsetTop=padding+margin。如果父级盒子没有定位，则以body为准。

offsetLeft/.offsetTop从最近已经定位了的父级的padding开始计算，不含border，含margin，其实就是距离最近已经定位了父级元素边框的距离

.offsetParent：返回该对象的已经定位过了的祖先元素（不一定是亲爸爸），如果祖先元素都没定位，则以body为准。

总结：offsetTop/Left和style.Top/Left区别：

1：最大的区别在于offsetTop/Left可以返回没有定位的盒子左侧/顶端的位置，而style.Top/Left不能，只有定位了的盒子才有left/top属性。

2：offsetTop/Left返回的是值；而style.Top/Left返回的是字符串。

3：offsetTop/Left只能读取值，不能设置值。而style.Top/Left可以设置值。

4：style.Top/Left只能用于行内属性上，而offsetTop/Left随便，就是这么刚，就是这么强大。

.clientWidth:对象可见区域的宽度

.clientHeight:对象可见区域的高度

获得浏览器宽/高的方法：

①：Window.innerHeight/Width：获得浏览器窗口的宽/高

②：Window.clientWidth:对象可见区域的高度

③：document.documentElement.clientWidth/Height；可见html文档标签的宽度/高度

Document.body.clientWidth/Height

Window.navlgator.userAgent:返回包含浏览器名称的字符。

总结box居中的方法：

Margintop负数、table、js获取窗口高度。

## js缓动公式：

begin=begin+(target-begin)/roateTime;

target：最终距离

begin：元素位置（一般是offsetLeft、offsetTop、left、top等）

roateTime：控制速度的时间，单位是毫秒。值越大，越慢，值越小越快

## 2.event事件对象

事件对象的声明：

Btn.事件名=function(event){

var event1=event||window.event;

}

或function 函数名（event）{

var event1=event||window.event;

}

短路表达式：var c=a||b;如果条件a成立，那么将a赋值给c，否则将b赋值给c。

或逻辑表达式。

event就是事件的对象，在出发dom上的某个事件时，会产生一个事件对象evnet，这个对象包含所有与事件有关的信息。所有的浏览器都支持event事件对象，但支持的方式不同，（主要是IE678）支持window.event，普通的浏览器支持event。

Data 返回拖拽对象的URL字符串

Width：该窗口或框架<iframe>的宽度

PageX/Y 光标相对于该网页的水平/竖直位置（IE无该属性）

ScreenX/y光标相对该屏幕的水平/竖直位置(不常用)

Target 该事件被传送到对象

type 事件的类型

clientX/Y对象相当于网页的水平位置（当前可见区域）

总结pageX，clientX，screenX的区别：

.ClientX/Y，对象当前窗口的左上角为基准点的距离

PageX

低版本浏览器不支持

我们的文档左上角为基准点

以我们文档（绝对定位）基准点对其

ScreenX/Y当前屏幕的左上角

案例：点击浏览器内文档某个位置，盒子跟随鼠标位置移到该位置。

加餐:

网页可见区域宽： document.body.clientWidth  
网页可见区域高： document.body.clientHeight  
网页可见区域宽： document.body.offsetWidth (包括边线的宽)  
网页可见区域高： document.body.offsetHeight (包括边线的高)  
网页正文全文宽： document.body.scrollWidth  
网页正文全文高： document.body.scrollHeight  
网页被卷去的高： document.body.scrollTop  
网页被卷去的左： document.body.scrollLeft  
网页正文部分上： window.screenTop  
网页正文部分左： window.screenLeft  
屏幕分辨率的高： window.screen.height  
屏幕分辨率的宽： window.screen.width  
屏幕可用工作区高度： window.screen.availHeight  
屏幕可用工作区宽度： window.screen.availWidth

# **懒加载实现原理**

在我们闲暇无事的时候，总是看看这个、那个的网站，比如京东、天猫、淘宝什么的，当我们再看的时候会出现照片闪了一下的情况，这是正常的，因为图片刚刚加载出来，这个也就是懒加载，如果你在打开网页的时候，一下把图片都请求出来，你可能没看几秒就退出了，一是浪费了你的流量，二是用户体验差，一进入这个网站会浪费很多流量，所以会造成遗失大量的用户。

### **1. 简介**

懒加载(Load On Demand)是一种独特而又强大的数据获取方法,它能够在用户滚动页面的时候自动获取更多的数据,而新得到的数据不会影响原有数据的显示,同时最大程度上减少服务器端的资源耗用。  
[百度百科](http://baike.baidu.com/link?url=8GNATQflwTtWWVIKrbdN-qi-egWJJPTmT5c7-VF1pKicIJ9cBybQFn4Q2UhdAdvw91nYLhW4XXJmheg09lAzW9RyNOiD7dMV7PpDhSDiLCM99FdHuA6R_tSoVFkkuRbb" \t "http://www.jianshu.com/p/_blank)

### **2. 代码实现**

#### **2.1 CSS样式**

下面这句话是你在写移动端的时候写的

<meta name="viewport" content="width=device-width,initial-

scale=1,minimum-scale=1,maximum-scale=1,user-scalable=no" />

\*{

margin: 0;

padding: 0;

}

ul,li{

list-style: none;

}

img{

width: 100%;

height: 640px;

}

#### **2.2 HTML结构**

实现懒加载的图片多一点

<ul>

<li><img data-src="1.jpg" alt="" /></li>

<li><img data-src="2.jpg" alt="" /></li>

<li><img data-src="3.jpg" alt="" /></li>

<li><img data-src="4.jpg" alt="" /></li>

<li><img data-src="5.jpg" alt="" /></li>

<li><img data-src="1.jpg" alt="" /></li>

<li><img data-src="2.jpg" alt="" /></li>

<li><img data-src="3.jpg" alt="" /></li>

<li><img data-src="4.jpg" alt="" /></li>

<li><img data-src="5.jpg" alt="" /></li>

<li><img data-src="1.jpg" alt="" /></li>

<li><img data-src="2.jpg" alt="" /></li>

<li><img data-src="3.jpg" alt="" /></li>

<li><img data-src="4.jpg" alt="" /></li>

<li><img data-src="5.jpg" alt="" /></li>

<li><img data-src="1.jpg" alt="" /></li>

<li><img data-src="2.jpg" alt="" /></li>

<li><img data-src="3.jpg" alt="" /></li>

<li><img data-src="4.jpg" alt="" /></li>

<li><img data-src="5.jpg" alt="" /></li>

<li><img data-src="1.jpg" alt="" /></li>

<li><img data-src="2.jpg" alt="" /></li>

<li><img data-src="3.jpg" alt="" /></li>

<li><img data-src="4.jpg" alt="" /></li>

<li><img data-src="5.jpg" alt="" /></li>

<li><img data-src="1.jpg" alt="" /></li>

<li><img data-src="2.jpg" alt="" /></li>

<li><img data-src="3.jpg" alt="" /></li>

<li><img data-src="4.jpg" alt="" /></li>

<li><img data-src="5.jpg" alt="" /></li>

</ul>

###### **实现懒加载思路**

1. 提前给所有的图片设置宽和高来占位
2. 不设置图片的src属性，而是自定义一个data-src属性来储存图片的路径
3. 判断当前屏幕的位置（图片的top值和屏幕底部的值），来进行是否显示图片，如果需要显示的话，把data-src的值给src。

#### **2.3 JS代码**

var imgs = document.getElementsbyTagName("img");

//设置滚动事件

window.onscroll = lazyloadFn;

//屏幕的高度

var sh = window.innerHeight;

//刚进来需要调用一下

lazyloadFn();

//图片懒加载的方法

function lazyloadFn() {

//获取滚动的距离

var st = document.body.scrollTop;

//计算滚动中的底部的值

var sn = sh + st;

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

if (imgs[i].offsetTop <= sn) {

imgs[i].src = imgs[i].getAttribute("data-src");

}

}

}

#### **2.4 效果** 你是不是要问我怎么没有闪一下的效果呢？别着急啊，在下面给你实现

#### **2.5 闪一下的效果**

在CSS中添加几行代码，以下样式是为了做渐变效果，在你写项目的时候不要乱用！！！

img{

width: 100%;

height: 640px;

opacity: 0;

transition: opacity 2s;

}

img[src]{

opacity: 1;

}

#### **2.6 以后真正写的代码**

// 利用jquery懒加载插件的话，图片设置data-original属性来存储图片路径

$("img").lazyload({

effect : "fadeIn"

});

# Layui学习笔记

layui对象通过use方法，调用layer模块和form模块，需要实例化layer对象

什么是模块化开发？

我们现在的网站或者系统，越来越像桌面程序一样，需要一个团队分工协作（快），进度管理（非阻塞），单元测试（效率更高）等等，导致开发人员不得不使用软件工程的方法，管理网站业务逻辑（对我们前段开发人员来说，要具有模块化开发思维，以后的要求更高。）

Js的模块化变成已经成为一种趋势或一种需求。

例如layui，开发一个网站，只需要实现核心页面构造和业务逻辑，其他的都可以加载layui已经编写好的模块。

但是，javascript不是一种模块化的编程语言，没有类，更没有模块化概念(ES6会正式支持)

什么是模块？

模块就是实现特定功能的一组方法，只要把不同的函数（以及各种变量）简单放在一起就是模块

CSS模块：

Base.css,common.css,index.css,lawerPerCenter.css

JS模块：$(),setTab(name,index,maxlength)放到一个js文件里，区别网页脚本，以上属于原始级别的模块化开发。

缺点：1、“污染”了全局变量，无法保证不与其他模块发生变量名冲突。

1. 模块成员之间看不出直接关系。

对象写法

为了解决原始方法的缺点，可以把模块写成一个模块对象，所有的模块成员都放到这对象里面。

function Product（name，price，color）{

this.name=name;

this.price=price;

......

}

var product1=new Product();

但是这样的写法会暴露所有模块的属性成员，很多时候都能被外部改写，有时候我们不希望内部属性被改写。

来看下layui的引入方法：

//让layui对象通过use方法，调用layer模块和form模块，需要实例化layer对象

layui.use(["layer","form"],function(){

var layer =layui.layer;//实例化layer对象

var form=layui.form();//实例化form

layer.msg("欢迎使用layer模块",{shift:4})

})

项目经理：使用axure，快速构建出产品的原型，

UI设计师（美工）：根据原型做出PSD效果图

前段开发：根据PSD效果图实现页面的重构及前段功能开发

使用：mulu.html作为开发工作的记录页面，方便检查工作进度

使用：IE css3.htc来帮助解决IE6-IE8实现元素的CSS3效果，要加上下面一句话：



.css3\_style{behavior:url(ie-css3.htc);position:relative;z-index:100;}

* max-width和min-width
* width:65%;

overflow: hidden; /\*自动隐藏文字\*/

text-overflow: ellipsis;/\*文字隐藏后添加省略号\*/

white-space: nowrap;/\*强制不换行\*//\*不允许出现半汉字截断\*/

**草稿箱 企业申报信息**

**LayUI 学习笔记**

什么是**模块化**开发？

我们做的网站或者系统（B/S），越来越像桌面程序，需要一个团队分工协作（快），进度管理（非阻塞），单元测试（效率）等等，导致开发人员不得不使用软件工程的方法，管理网页的业务逻辑（对我们前端开发人员来说，要具有模块化开发思维，以后的要求更高）

js的模块化编程，已经成为一种趋势或者是需求。

拿layui来说，我们开发 一个网站，只需要实现核心的页面构造和业务逻辑，其他的都可以加载layui已经编写的好模块。

但是，**JavaScript 不是一种严格的模块化的编程语言，没有“类”，更没有“模块”**概念（ES6会正式支持）

1. **具体什么是模块**？

“module”，模块就是实现特定功能的一组方法，只要把不同的函数（以及各种变量）简单放一起就算是一模块

1. **模块化的应用**？

css的模块化：base.css，common.css，index.css，lawerPerCenter.css

js的 模块： $(id),setTab(name,indx,maxlength)放到一个js文件里，区别网页脚本

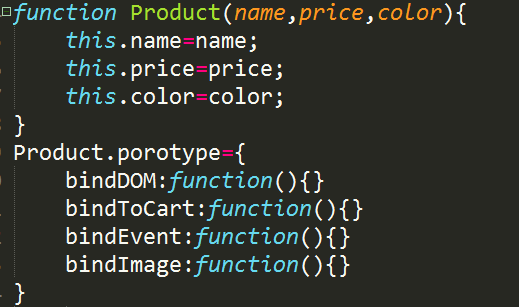
以上属于原始级别的模块化开发。

缺点：1、“污染”了全局变量，无法保证不与其他模块发生变量名冲突

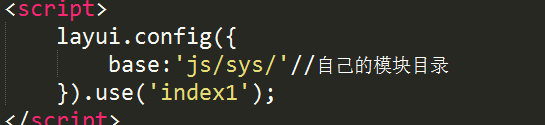
模块成员之间看不出直接关系。

1. **对象写法**

为了解决原始方法的缺点，可以把模块写成一个对象，所有的模块成员都放到这对象里面。

但是，这样的写法会暴露所有模块属性成员，很多时候内部状态都可以被外部改写。有时候我们不希望内部属性被改写。

来看下layui的引入方法：





简单粗暴的layui引入方法：

<script src=“../layui.all.js”><script>

<script >

//当使用了layui.all.js，而无需在执行layui.use()方法

Var form = layui.form();

Var layer=layui.layer;

Layui.msg(“111”)；

<script>

通过使用layui，我们了解到模块化设计，必须的能力：

1. **定义封装的模块。**
2. **定义新模块对其他模块的依赖。**
3. **可对其他模块的引入支持。**

**AMD和CMD**

**Asynchronous Module Definition ，所有的模块将被异步加载（按需加载）**

**面试题：**

**回想js一个特点：单线程的概念（解释执行，一一行往下读着执行），那么如何让js实现多线程或者异步执行呢？set Timeout。**

**For(var i=0;i<100;i++){**

**setTimmeut(function(){**

**Console.log(i)**

**},0)**

**}**

**CMD common Module Definition**

项目经理：使用axure ，快速构建出产品的原型

UI设计师（美工）：根据原型做出PSD效果图

前端开发：根据PSD效果图实现页面的重构及前端功能开发

使用：mulu.html 作为开发的记录页面，方便检查工作进度

使用ie-css3.htc 来帮助IE6-IE8实现元素的CSS3效果



max-width和min-width

width:65%;

overflow: hidden; /\*自动隐藏文字\*/

text-overflow: ellipsis;/\*文字隐藏后添加省略号\*/

white-space: nowrap;/\*强制不换行\*//\*不允许出现半汉字截断\*/

大家都知道，JS是若变量，弱类型的一种脚本语言，更谈不上“模块化”的概念（ES6正式支持）

使用：simplejavascriptinheritance.js来模拟实现js对象的类

使用：jquery.elatage.js来实现

# 本地存储与数据请求

## 本地存储部分

CRUD：

CRUD是指在做计算处理时的增加(Create)、查询(Retrieve)（重新得到数据）、更新(Update)和删除(Delete)几个单词的首字母简写。主要被用在描述软件系统中数据库或者持久层的基本操作功能。

数据处理：

//判断当前数据是否满足格式要求（是不是正确格式的手机号，邮箱等）

if(uNmae.value==''){return;}

if(uNmae.value.length==0){return;}

//进行数据保护，以后在进行后台交互时，不管什么，先加上这两句话。

localStorage.setItem(“对象名”,“值”);//设置本地存储，它里边存储的变量都是全局变量。

localhost本地存储，其值为127.0.0.1。

localStorage.clear()；//把本地存储的键值对都删除

localStorage.removeItem('name')//只删除指定的本地对象（key：value）

key：localStorage.key(index);//得到某个索引的值

1：Storage就是存储，Web Storage就是通过浏览器进行本地存储，Web Storage在HTML5提供了2个对象帮助用户进行存储数据，将数据存储到本地。localStorage和sessionStorage。这两个对象是window属性，方法有：

window.setItem（key，value）设置要存储的数据是以key/value进行保存

window.getItem（key，value）通过key/value取值

localStorage.key(index)/sessionStorage.key(index)是通过index（下标/索引）拿到key值**（而不是value值）**

length：获取当前存储数据的个数，也就是有几个Item

.clear()将当前对应的存储数据全部清除

.remove(key)删除指定的key，同时也把对应的value值删除。

1. localStorage：长期本地存储（将存储到的数据类型保存到本地硬盘中长期保存，用户不清除浏览器缓存的话就一直保存，清除缓存之后删除）
2. sessionStorage：临时本地存储（用户关闭刘浏览器时就被清除。）

#### LocalStorage

* 是对Cookie的优化
* 没有时间限制的数据存储
* 在隐私模式下不可读取
* 大小限制在500万字符左右，各个浏览器不一致
* 在所有同源窗口中都是共享的
* 本质是在读写文件，数据多的话会比较卡（firefox会一次性将数据导入内存）
* 不能被爬虫爬取，不要用它完全取代URL传参
* IE７及以下不支持外，其他标准浏览器都完全支持

#### SessionStorage

* 针对一个 session 的数据存储
* 大小限制在5M左右，各个浏览器不一致
* 仅在当前浏览器窗口关闭前有效（适合会话验证）
* 不在不同的浏览器窗口中共享，即使是同一个页面

### JS下的操作方法

* 获取键值：localStorage.getItem(“key”)
* 设置键值：localStorage.setItem(“key”,”value”)
* 清除键值：localStorage.removeItem(“key”)
* 清除所有键值：localStorage.clear()
* 获取键值2：localStorage.keyName
* 设置键值2：localStorage.keyName = “value”

### JQ下的操作方法（JS方法前加”window.”）

* 获取键值：window.localStorage.getItem(“key”)
* 设置键值：window.localStorage.setItem(“key”,”value”)
* 清除键值：window.localStorage.removeItem(“key”)
* 清除所有键值：window.localStorage.clear()
* 获取键值2：window.localStorage.keyName
* 设置键值2：window.localStorage.keyName = “value”

User表的处理：

当用户登录成功之后存储user表。

在用户登录输入密码的时候移除user表。

判断用户切换设备登录或者第一次登录的方法：

当用户登录成功之后，随便存储一个变量并且不要清除它。

在登录页面的点击登录按钮时候判断该变量是否存在，如果不存在或用户清除浏览器数据了，那么就说明更换了设备或是第一次登录。

## 数据请求

1：用户输入的是数据，键值对形式存在

2：form表单原封不动地向服务器提交的也是用户的数据，以键值对形式存在。

3：服务器返回给浏览器的还是数据，即浏览器从服务器端接受的还是数据。该数据是以键值对的形式存在

4：浏览器将接收到的数据进行数模转换：将数据转化为对象（jsonObject）：var jsonObject=JSON.parse(data);

将对象转化为json字符串：JSON.stringfy(obj)方法。

5：浏览器对对象进行操作，完成表单验证，动态添加数据等操作。方法是ajax请求。

Ajax请求：$.ajax({

Type:”POST/GET”;必选。

data:{“type”：num，“key1”:”value1”,”key2”:”value2”,.....},必选，这里的type值是服务器给的

Url:服务器地址，必选。

以上三步是向服务器请求数据。可能与用户输入的数据相同。

Success：function（data）{成功之后要有变量名接收返回成功的数据。

var jsonObject=JSON.parse(data);//数模转换，将数据库里请求的数据转换成json对象。这里的data就是请求的data。

Console.log(data);

Var dat=jsonObject.things; things就是json对象中的子对象。

}

error：function（变量名）{错误之后要有变量名接收返回的信息提示。

Alert（变量名.status）;变量名.status是服务器返回给浏览器的状态。

}

})

如果是“POST”请求：

data:{“

type”：num，

“key1”:”value1”,

”key2”:”value2”,

.....

},

必选，这里的type值是服务器给的。

如果是“GET”请求：data数据里的type要写到url地址中。

# EcmaScript6语法

## ES6的严格模式：

ES6 的模块自动采用严格模式，不管你有没有在模块头部加上"use strict";。

严格模式主要有以下限制。

* 变量必须声明后再使用
* 函数的参数不能有同名属性，否则报错
* 不能使用with语句
* 不能对只读属性赋值，否则报错
* 不能使用前缀 0 表示八进制数，否则报错
* 不能删除不可删除的属性，否则报错
* 不能删除变量delete prop，会报错，只能删除属性delete global[prop]
* eval不会在它的外层作用域引入变量
* eval和arguments不能被重新赋值
* arguments不会自动反映函数参数的变化
* 不能使用arguments.callee
* 不能使用arguments.caller
* 禁止this指向全局对象
* 不能使用fn.caller和fn.arguments获取函数调用的堆栈

增加了保留字（比如protected、static和interface）

## Promise对象：

Promise是异步编程的一种解决方案，比传统的解决方案（回调函数和事件）更合理更强大。

所谓Promise，简单说就是一个容器，里面保存着某个未来才会结束的事件 (通常是一个异步操作)的结果。从语法上说，Promise是一个对象，从它可以获取异步操作的消息。

### Promise对象的2个特点

对象的状态不受外界影响。Promise对象代表一个异步操作，有三种状态：Pending(进行中)、Resolved(已完成)和Rejected(已失败)。只有异步操作的结果，可以决定当前是哪一种状态，任何其他操作都无法改变这个状态。这也是Promise这个名字的由来，它的英语意思就是“承诺”，表示其他手段无法改变。   
2.一旦状态改变，就不会再变，任何时候都可以得到这个结果。Promise对象的状态改变，只有两种可能：从Pending变为Resolved；从Pending变为Rejected。只要这两种情况发生，状态就凝固了，不会再变了，会一直保持这个结果。就算改变已经发生了，你再对Promise对象田静回调函数，也会立即得到这个结果。这与事件(Event)完全不同，事件的特点是，如果你错过了它，再去监听，是得不到结果的。

有了Promise对象，就可以把异步操作以同步操作的流程表达出来，避免了层层嵌套的回调函数。此外，Promise对象提供了统一的接口，使得控制异步操作更加容易。

Promise构造函数接受一个函数作为参数，该函数的两个参数分别是resolve和reject。它们是两个函数，由 JavaScript 引擎提供，不用自己部署。

const promise = new Promise(function(resolve, reject) {

// ... some code

if (/\* 异步操作成功 \*/){

resolve(value);

} else {

reject(error);

}

}

);

promise.then(function(value) {

// success

},

function(error) {

// failure

}

);

### resolve

resolve函数的作用是，将Promise对象的状态从“未完成”变为“成功”（即从 pending 变为 resolved），在异步操作成功时调用，并将异步操作的结果，作为参数（value）传递出去；reject函数的作用是，将Promise对象的状态从“未完成”变为“失败”（即从 pending 变为 rejected），在异步操作失败时调用，并将异步操作报出的错误，作为参数（err）传递出去。

Promise实例生成以后，可以用then方法分别指定resolved状态和rejected状态的回调函数。

then方法可以接受两个回调函数作为参数。第一个回调函数是Promise对象的状态变为resolved时调用，这个函数接受 resolved（value），第二个回调函数是Promise对象的状态变为rejected时调用，这个函数接受 reject(error)。其中，第二个函数是可选的，不一定要提供。这两个函数都接受Promise对象传出的值作为参数。

## 模块化的JS（export与import）

模块功能主要由两个命令构成：export和import。export命令用于规定模块的对外接口，import命令用于输入其他模块提供的功能。

### export 命令

一个模块就是一个独立的文件。该文件内部的所有变量，外部无法获取。如果你希望外部能够读取模块内部的某个变量，就必须使用export关键字输出该变量。下面是一个 JS 文件，里面使用export命令输出变量。

export var firstName = 'Michael';

export var lastName = 'Jackson';

export var year = 1958;

export的写法，除了像上面这样，还有另外一种。

var firstName = 'Michael';var lastName = 'Jackson';var year = 1958;

export {firstName, lastName, year};

上面代码在export命令后面，使用大括号指定所要输出的一组变量。它与前一种写法（直接放置在var语句前）是等价的，但是应该优先考虑使用这种写法。因为这样就可以在脚本尾部，一眼看清楚输出了哪些变量。

export命令除了输出变量，还可以输出函数或类（class）。

export function multiply(x, y) {

return x \* y;};

上面代码对外输出一个函数multiply。

通常情况下，export输出的变量就是本来的名字，但是可以使用as关键字重命名。

function v1() { ... }function v2() { ... }

export {

v1 as streamV1,

v2 as streamV2,

v2 as streamLatestVersion};

上面代码使用as关键字，重命名了函数v1和v2的对外接口。重命名后，v2可以用不同的名字输出两次。

最后，export命令可以出现在模块的任何位置，只要处于模块顶层就可以。如果处于块级作用域内，就会报错，import命令也是如此。这是因为处于条件代码块之中，就没法做静态优化了，违背了 ES6 模块的设计初衷。

### import 命令

使用export命令定义了模块的对外接口以后，其他 JS 文件就可以通过import命令加载这个模块。

// main.jsimport {firstName, lastName, year} from './profile.js';

function setName(element) {

element.textContent = firstName + ' ' + lastName;}

上面代码的import命令，用于加载profile.js文件，并从中输入变量。import命令接受一对大括号，里面指定要从其他模块导入的变量名。大括号里面的变量名，必须与被导入模块（profile.js）对外接口的名称相同。

如果想为输入的变量重新取一个名字，import命令要使用as关键字，将输入的变量重命名。

import { lastName as surname } from './profile.js';

import命令输入的变量都是只读的，因为它的本质是输入接口。也就是说，不允许在加载模块的脚本里面，改写接口。

import {a} from './xxx.js'

a = {}; // Syntax Error : 'a' is read-only;

上面代码中，脚本加载了变量a，对其重新赋值就会报错，因为a是一个只读的接口。但是，如果a是一个对象，改写a的属性是允许的。

import {a} from './xxx.js'

a.foo = 'hello'; // 合法操作

上面代码中，a的属性可以成功改写，并且其他模块也可以读到改写后的值。不过，这种写法很难查错，建议凡是输入的变量，都当作完全只读，轻易不要改变它的属性。

import后面的from指定模块文件的位置，可以是相对路径，也可以是绝对路径，.js后缀可以省略。如果只是模块名，不带有路径，那么必须有配置文件，告诉 JavaScript 引擎该模块的位置。

import {myMethod} from 'util';

上面代码中，util是模块文件名，由于不带有路径，必须通过配置，告诉引擎怎么取到这个模块。

注意，import命令具有提升效果，会提升到整个模块的头部，首先执行。

foo();

import { foo } from 'my\_module';

上面的代码不会报错，因为import的执行早于foo的调用。这种行为的本质是，import命令是编译阶段执行的，在代码运行之前。

由于import是静态执行，所以不能使用表达式和变量，这些只有在运行时才能得到结果的语法结构。

import 'lodash';

上面代码仅仅执行lodash模块，但是不输入任何值。

如果多次重复执行同一句import语句，那么只会执行一次，而不会执行多次。

import 'lodash';

import 'lodash';

上面代码加载了两次lodash，但是只会执行一次。

import { foo } from 'my\_module';

import { bar } from 'my\_module';

// 等同于import { foo, bar } from 'my\_module';

### 模块的整体加载

除了指定加载某个输出值，还可以使用整体加载，即用星号与“as”（\* as）指定一个对象，所有输出值都加载在这个对象上面。

### export default 命令

export default命令用于指定模块的默认输出，相当于用defalut语句声明了一个数据类型。显然，一个模块只能有一个默认输出，因此export default命令只能使用一次。所以，import命令后面才不用加大括号，因为只可能唯一对应export default命令。正是因为export default命令其实只是输出一个叫做default的变量，所以它后面不能跟变量声明语句。

## async 函数

ES2017 标准引入了 async 函数，使得异步操作变得更加方便。async 函数是什么？一句话，它就是 Generator 函数的语法糖。

基本用法

async函数返回一个 Promise 对象，可以使用then方法添加回调函数。当函数执行的时候，一旦遇到await就会先返回，等到异步操作完成，再接着执行函数体内后面的语句。async函数的语法规则总体上比较简单，难点是错误处理机制。

下面是一个例子。

async function getStockPriceByName(name) {

const symbol = await getStockSymbol(name);

const stockPrice = await getStockPrice(symbol);

return stockPrice;

}

getStockPriceByName('goog').then(function (result) {

console.log(result);});

下面是另一个例子，指定多少毫秒后输出一个值。

function timeout(ms) {

return new Promise((resolve) => {

setTimeout(resolve, ms);

});}

async function asyncPrint(value, ms) {

await timeout(ms);

console.log(value);

}

asyncPrint('hello world', 50);

上面代码指定 50 毫秒以后，输出hello world。

由于async函数返回的是 Promise 对象，可以作为await命令的参数。所以，上面的例子也可以写成下面的形式。

async function timeout(ms) {

await new Promise((resolve) => {

setTimeout(resolve, ms);

});}

async function asyncPrint(value, ms) {

await timeout(ms);

console.log(value);}

asyncPrint('hello world', 50);

### Promise 对象的状态变化

async函数返回的 Promise 对象，必须等到内部所有await命令后面的 Promise 对象执行完，才会发生状态改变，除非遇到return语句或者抛出错误。也就是说，只有async函数内部的异步操作执行完，才会执行then方法指定的回调函数。

下面是一个例子。

async function getTitle(url) {

let response = await fetch(url);

let html = await response.text();

return html.match(/<title>([\s\S]+)<\/title>/i)[1];}getTitle('https://tc39.github.io/ecma262/').then(console.log)

// "ECMAScript 2017 Language Specification"

上面代码中，函数getTitle内部有三个操作：抓取网页、取出文本、匹配页面标题。只有这三个操作全部完成，才会执行then方法里面的console.log。

### await 命令

正常情况下，await命令后面是一个 Promise 对象。如果不是，会被转成一个立即resolve的 Promise 对象。

async function f() {

return await 123;

}

f().then(v => console.log(v))

// 123

上面代码中，await命令的参数是数值123，它被转成 Promise 对象，并立即resolve。

await命令后面的 Promise 对象如果变为reject状态，则reject的参数会被catch方法的回调函数接收到。

async function f() {

await Promise.reject('出错了');}

f().then(v => console.log(v)).catch(e => console.log(e))

// 出错了

注意，上面代码中，await语句前面没有return，但是reject方法的参数依然传入了catch方法的回调函数。这里如果在await前面加上return，效果是一样的。

只要一个await语句后面的 Promise 变为reject，那么整个async函数都会中断执行。

async function f() {

await Promise.reject('出错了');

await Promise.resolve('hello world'); // 不会执行}

上面代码中，第二个await语句是不会执行的，因为第一个await语句状态变成了reject。

有时，我们希望即使前一个异步操作失败，也不要中断后面的异步操作。这时可以将第一个await放在try...catch结构里面，这样不管这个异步操作是否成功，第二个await都会执行。

async function f() {

try {

await Promise.reject('出错了');

} catch(e) {

}

return await Promise.resolve('hello world');}

f().then(v => console.log(v))

// hello world

另一种方法是await后面的 Promise 对象再跟一个catch方法，处理前面可能出现的错误。

async function f() {

await Promise.reject('出错了')

.catch(e => console.log(e));

return await Promise.resolve('hello world');}

f().then(v => console.log(v))

// 出错了// hello world

### 错误处理

如果await后面的异步操作出错，那么等同于async函数返回的 Promise 对象被reject。

async function f() {

await new Promise(function (resolve, reject) {

throw new Error('出错了');

});}

f().then(v => console.log(v)).catch(e => console.log(e))

// Error：出错了

上面代码中，async函数f执行后，await后面的 Promise 对象会抛出一个错误对象，导致catch方法的回调函数被调用，它的参数就是抛出的错误对象。具体的执行机制，可以参考后文的“async 函数的实现原理”。

防止出错的方法，也是将其放在try...catch代码块之中。

async function f() {

try {

await new Promise(function (resolve, reject) {

throw new Error('出错了');

});

} catch(e) {

}

return await('hello world');}

如果有多个await命令，可以统一放在try...catch结构中。

async function main() {

try {

const val1 = await firstStep();

const val2 = await secondStep(val1);

const val3 = await thirdStep(val1, val2);

console.log('Final: ', val3);

}

catch (err) {

console.error(err);

}}

下面的例子使用try...catch结构，实现多次重复尝试。

const superagent = require('superagent');

const NUM\_RETRIES = 3;

async function test() {

let i;

for (i = 0; i < NUM\_RETRIES; ++i) {

try {

await superagent.get('http://google.com/this-throws-an-error');

break;

} catch(err) {}

}

console.log(i); // 3}

test();

上面代码中，如果await操作成功，就会使用break语句退出循环；如果失败，会被catch语句捕捉，然后进入下一轮循环

### 注意：

第一点，前面已经说过，await命令后面的Promise对象，运行结果可能是rejected，所以最好把await命令放在try...catch代码块中。

async function myFunction() {

try {

await somethingThatReturnsAPromise();

} catch (err) {

console.log(err);

}}

// 另一种写法

async function myFunction() {

await somethingThatReturnsAPromise()

.catch(function (err) {

console.log(err);

});}

第二点，多个await命令后面的异步操作，如果不存在继发关系，最好让它们同时触发。

let foo = await getFoo();let bar = await getBar();

上面代码中，getFoo和getBar是两个独立的异步操作（即互不依赖），被写成继发关系。这样比较耗时，因为只有getFoo完成以后，才会执行getBar，完全可以让它们同时触发。

// 写法一let [foo, bar] = await Promise.all([getFoo(), getBar()]);

// 写法二let fooPromise = getFoo();let barPromise = getBar();let foo = await fooPromise;let bar = await barPromise;

上面两种写法，getFoo和getBar都是同时触发，这样就会缩短程序的执行时间。

第三点，await命令只能用在async函数之中，如果用在普通函数，就会报错。

async function dbFuc(db) {

let docs = [{}, {}, {}];

// 报错 docs.forEach(function (doc) {

await db.post(doc);

});}

上面代码会报错，因为await用在普通函数之中了。但是，如果将forEach方法的参数改成async函数，也有问题。

function dbFuc(db) { //这里不需要 async let docs = [{}, {}, {}];

// 可能得到错误结果 docs.forEach(async function (doc) {

await db.post(doc);

});

}

上面代码可能不会正常工作，原因是这时三个db.post操作将是并发执行，也就是同时执行，而不是继发执行。正确的写法是采用for循环。

async function dbFuc(db) {

let docs = [{}, {}, {}];

for (let doc of docs) {

await db.post(doc);

}}

如果确实希望多个请求并发执行，可以使用Promise.all方法。当三个请求都会resolved时，下面两种写法效果相同。

async function dbFuc(db) {

let docs = [{}, {}, {}];

let promises = docs.map((doc) => db.post(doc));

let results = await Promise.all(promises);

console.log(results);}

// 或者使用下面的写法

async function dbFuc(db) {

let docs = [{}, {}, {}];

let promises = docs.map((doc) => db.post(doc));

let results = [];

for (let promise of promises) {

results.push(await promise);

}

console.log(results);}

目前，[esm](https://www.npmjs.com/package/esm)模块加载器支持顶层await，即await命令可以不放在 async 函数里面，直接使用。

// async 函数的写法const start = async () => {

const res = await fetch('google.com');

return res.text();};

start().then(console.log);

// 顶层 await 的写法const res = await fetch('google.com');

console.log(await res.text());

上面代码中，第二种写法的脚本必须使用esm加载器，才会生效。

# JavaScript编译器---babel

Babel 是一个工具链，主要用于在旧的浏览器或环境中将 ECMAScript 2015+ 代码转换为向后兼容版本的 JavaScript 代码。以下是Babel 可以为你做的主要事情：

1.转换语法 babel-cli和babel-preset-env

2.Polyfill 实现目标环境中缺少的功能 （通过 [@babel/polyfill](https://babel.docschina.org/docs/en/babel-polyfill)）

3.源代码转换 (codemods)

下面是一个例子：

// Babel Input: ES2015 arrow function

[1, 2, 3].map((n) => n + 1);

// Babel Output: ES5 equivalent

[1, 2, 3].map(function(n) { return n + 1; });

## 安装 Babel CLI 和 **[env](https://babeljs.cn/docs/plugins/preset-env)** preset

npm install --save-dev babel-cli babel-preset-env

## ES6及其他

Babel 通过语法转换器支持最新版本的 JavaScript 。 这些[插件](https://www.babeljs.cn/docs/plugins/)允许你立刻使用新语法，无需等待浏览器支持。根据你支持的环境自动决定适合你的 Babel 插件的 Babel preset。它使用了 compat-table。

你可以通过以下方式安装 preset:

npm install --save-dev babel-preset-env

在没有任何配置选项的情况下，babel-preset-env 与 babel-preset-latest（或者babel-preset-es2015，babel-preset-es2016和babel-preset-es2017一起）的行为完全相同。

{

"presets": ["env"]

}

你也可以仅仅配置项目所支持浏览器所需的polyfill和transform。只编译所需的代码会使你的代码包更小。

这个例子只包含了支持每个浏览器最后两个版本和safari大于等于7版本所需的polyfill和代码转换。我们使用 [browserslist](https://github.com/ai/browserslist) 来解析这些信息，所以你可以使用 [browserslist 支持的有效的查询格式](https://github.com/ai/browserslist" \l "queries)。

{

"presets": [

["env", {

"targets": {

"browsers": ["last 2 versions", "safari >= 7"]

}

}]

]

}

## Polyfill

由于 Babel 只转换语法(如箭头函数)， 你可以使用 babel-polyfill 支持新的全局变量，例如 Promise 、新的原生方法如 String.padStart (left-pad) 等。 它使用了 [core-js](https://github.com/zloirock/core-js) 和 [regenerator](https://facebook.github.io/regenerator/)。 查看 [babel-polyfill](https://www.babeljs.cn/docs/usage/polyfill) 文档获取更多信息。它会仿效一个完整的 ES2015+ 环境，并意图运行于一个应用中而不是一个库/工具。这个 polyfill 会在使用 babel-node 时自动加载。

这意味着你可以使用新的内置对象比如 Promise 或者 WeakMap, 静态方法比如 Array.from 或者 Object.assign, 实例方法比如 Array.prototype.includes 和生成器函数（提供给你使用 [regenerator](https://www.babeljs.cn/docs/plugins/transform-regenerator/) 插件）。为了达到这一点， polyfill 添加到了全局范围，就像原生类型比如 String 一样。

你可以通过以下方式安装 polyfill

npm install --save-dev babel-polyfill

注意：最新版是[@babel/polyfill](https://babel.docschina.org/docs/en/babel-polyfill),与老版babel-polyfill不兼容

在使用它时需要在你应用程序的入口点顶部或打包配置中引入.在 webpack.config.js 中，将 babel-polyfill 加到你的 entry 数组中：

entry: {

index: ['babel-polyfill','whatwg-fetch','./index.js'],

// index:'./test.mjs'

},

这里的whatwg-fetch是为了让浏览器兼容fetch方法。

新版的[@babel/polyfill](https://babel.docschina.org/docs/en/babel-polyfill):

entry: {

index: ['[@babel/polyfill](https://babel.docschina.org/docs/en/babel-polyfill)','whatwg-fetch','./index.js'],

// index:'./test.mjs'

},

## 在Webpack中使用babel加载器——babel-loader：

npm install --save-dev babel-loader babel-core

Webpack.config.js里：

在你的头部入口里配置：

当你需要用到React的jsx的时候，在你的module规则里配置：

module: {

rules: [

{ test: /\.js|.jsx$/, exclude: /node\_modules/, loader: "babel-loader" }

]

}

或者当你需要用到mjs文件的时候：

module: {

rules: [

//其他规则

{

test: /\.m?js$/,

exclude: /(node\_modules|bower\_components)/,

use: {

loader: 'babel-loader',

options: {

presets: ['@babel/preset-env']

}

}

},

//其他规则

]

}

拓展：也可以使用[options](https://webpack.js.org/configuration/module/" \l "rule-options-rule-query)属性将选项传递给加载程序：

{

test: /\.m?js$/,

exclude: /(node\_modules|bower\_components)/,

use: {

loader: 'babel-loader',

options: {

presets: ['@babel/preset-env'],

plugins:[require('@babel/plugin-proposal-object-rest-spread')]

}

}

}

# Js高级算法

## JS递归算法

JS递归思想：递归就是将一个大问题分解成 n 个相似的小问题，然后不断地调用自身去解决这些小问题，从而求出结果。

### js递归求解实际应用：

// 定义一个函数，用于求 n 的阶乘

function func(n) {

if (n == 1) {

return 1;

}else{

// func(n-1) 因为传递的参数是 n-1,那么就是求 (n-1) 的阶乘 return n \* func(n-1);

}

}

console.log( func(5) );

//公园里有一堆桃子，猴子每天吃掉一半，挑出一个坏的扔掉，第6天的时候发现还剩1个桃子，问原来有多少个桃子

总桃子数与天数就构成了一个递归函数关系。

var sl;//总桃子数

  function shuliang (n){//n为天数

    if(n==6) {

          sl=1;

          }

     else{

           sl=(shuliang(n+1)+1)\*2;

           }

          return sl;

    }

   alert( shuliang(0));

### 2.SKU递归算法

1.将下面的数据源分类数组所有属性的所有排列组合用递归的方法输出出来。

数据源数组：

goodsPropertyCategoryList:[

{

"goodsPropertyCategoryName":"颜色",

"id":0,

"goodsPropertyList":[

{"goodsPropertyName":"红色"},

{"goodsPropertyName":"蓝色"}

]

},

{

"goodsPropertyCategoryName":"容量",

"id":1,

"goodsPropertyList":[

{"goodsPropertyName":"70ml"},

{"goodsPropertyName":"50ml"}

]

}

]

SKU递归算法：

//工具方法1（不带类名，只有属性名）

function sku(list,index,str,arr=[]){

let goodsPropertyList = list[index].goodsPropertyList;

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

let s = str+goodsPropertyList[i].goodsPropertyName;

if (index == list.length - 1) {

console.log(s);

arr.push(s);

} else {

sku(list, index + 1, s + ", ",arr);

}

}

}

输出结果为;（类名a：属性a0，类名b：属性b0）

[

"颜色:红色,容量:70ml",

"颜色:红色,容量:50ml",

"颜色:蓝色,容量:70ml",

"颜色:蓝色,容量:50ml"

]

//工具方法2（带类名与属性名）

function skuName(list, index, str,arr=[]) {

let goodsPropertyList = list[index].goodsPropertyList;

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

let s = str + list[index].goodsPropertyCategoryName + ':' + goodsPropertyList[i].goodsPropertyName;

if (index == list.length - 1) {

console.log(s);

arr.push(s);

} else {

skuName(list, index + 1, s + ',',arr);

}

}

}

输出结果为;（属性a0，属性b0）

[

"红色,:70ml",

"红色,50ml",

"蓝色,70ml",

"蓝色,50ml"

]

var cuarr=[];

this.skuName(goodsPropertyCategoryList,0,"",cuarr);

Console.log(cuarr);

这里的goodsPropertyCategoryList是不同的分类组成的分类数组，而goodsPropertyList 则是每一个分类数组所对应的属性数组。

总结：此方法通常解决多个数组（数组个数不确定）所有元素（每个数组的元素个数不确定）所有可能的排列组合的问题。

淘宝后台管理系统的SKU属性列表原理与此类似。

# HTML5和CSS3新特性

学习目标：

1. 熟练使用web Storage中的Session Storage进行web本地存储
2. 了解draggable拖拽效果
3. 熟练CSS3新属性进行样式布局（重点）
4. 熟练使用CSS3进行2D、3D过度（重点）
5. 熟练使用CSS3完成动画效果（重点）
6. 熟练使用HTML5新标签
7. 熟练掌握CSS3新特性
8. 熟练使用H5本地存储
9. 熟练使用Canvas绘图
10. 熟练使用Svg绘图
11. 熟练使用Video和Audio标签
12. 自己实现一个播放器

## H5标签和属性

H5的发展史：04年开始（自己查看）

WCCC结构标准：结构 表现 行为 严格、过渡、框架

HTML5新特性的目的:

1、书写更简便的HTML代码 腾讯视频

1. 创建更简单的Web程序

简化了得标签：

1. DOCTYPE document type：<!DOCTYPE html>以当前最新的HTML版本进行解析，如果不是，就不解析。简化目的是为了更好地让HTML进行向低版本兼容
2. Meta标签的简化：简化当前的文本解析是以UTF-8格式进行解析
3. Link标签的简化：简化type是让浏览器在解析时候自动判断当前link文件的格式，通过文件的后缀名来判断type
4. Script和link：相似 都是让浏览器在解析的时候判断type
5. HTML5新增标签

新增加的结构标签份额日 主体结构标签和非主体结构标签

语义化标签：<header>头部，块级标签 <nav> 导航栏，块级元素<aside> <footer>

<header>：头部，块级标签，一般被放置在页面顶部，或者页面中某个页面区块元素的顶部，包含整个页面简介信息

<article文章标签></article>：从语义上来说 文档、页面，通常是一片文章、一个页面、一个独立完整的内容的模块。Article在开发中使用频率极高，强调独立性，多注意和header，footer标签配合使用

Arcitle标签可以互相嵌套。

<p>标签：自带上下行间距

<section>:区域声明标签，相当于一个div，容纳文字，将文本区域化，不建议当作div使用

aside标签：从语义上看是旁边，侧边

在article标签内部使用aside，则作为内容的附属信息部分，如 有关的参考资料、名词解释等。

外部使用时，则作为页面或者站点或全局信息附属部分信息，如测边栏、博客的友情链接部分、广告区域等。

Nav标签：通常作为页面导航的链接可以放在头部区域，也可放在侧边栏区域。

Main标签：显示页面主体内容

每个main标签只能包含一个main标签，main标签中不包含网站标题、logo、主导航、版本声明部分信息。

Header：从语义上讲是页眉，通常放置在整个页面或者页面内的一个内容区块的标题。一个页面内并没限制使用header标签个数，他是一种具有引导、导航结构的元素。

Footer标签：从语义上讲是文档的脚注，一个内容块级底部区域，通常内容是联系方式，相关内容，相关信息，版权等。

<mark>文本标记标签，行内标签 使文字高亮显示，默认背景颜色为黄色，可在css中调整

<b>标签也可使字体变粗，但H5赋予了它是起强调作用

<em>标签：（emphasis强调）和<i>标签一样，也是斜体

如果你想单纯地将字体样式设置为粗体或斜体，请不要受用这几个标签，因为w3c建议要在css样式表中定义字体样式。

新属性：

全局属性：style，draggable

Draggable：设置能否拖动，true能，false不能。在行内样式内设置/在HTML。

拖动元素：被托动的元素。

目标元素/容器：接纳被拖动元素的元素/容器。

**在拖动目标上触发事件** (目标元素)以鼠标光标事件（event）为标准，而不是被拖拽对象（this）**:**

1、要想使用拖拽功能，就必须使用draggable属性，标签内设置行内样式设置为true。

2、对象.[ondragstart](http://www.runoob.com/jsref/event-ondragstart.html) - 用户开始拖动元素时触发该事件，此事件作用在被拖拽元素上。

3、ondrag - 元素正在拖动时触发

4、[ondragend](http://www.runoob.com/jsref/event-ondragend.html) - 用户完成元素拖动后触发，此事件作用在被拖拽的元素上。

5、[ondragenter](http://www.runoob.com/jsref/event-ondragenter.html) - 当被鼠标拖动的对象进入其容器范围内时触发此事件，只触发一次，此事件作用在目标元素上，以鼠标光标事件（event）为标准，而不是被拖拽对象（this）

6、[ondragover](http://www.runoob.com/jsref/event-ondragover.html) - 当某被拖动的对象在另一对象容器范围内拖动时触发此事件，此事件作用在目标元素/容器，以鼠标光标事件（event）为标准，而不是被拖拽对象（this）只要被拖拽对象在目标容器内，该事件一直在触发。

7、[ondragleave](http://www.runoob.com/jsref/event-ondragleave.html) - 当被鼠标拖动的对象离开其容器范围内时触发此事件，以鼠标光标事件（event）为标准，而不是被拖拽对象（this）。

8、[ondrop](http://www.runoob.com/jsref/event-ondrop.html) - 在一个拖动过程中，释放鼠标键时触发此事件，作用在目标接纳容器上

9、dataTransfer 对象：拖拽对象用来传递的媒介使用的时候使用event.dataTransfer 。

HTML5中新使用的选择方法：

document.querySelector（）返回第一个CSS类对象

document.querySelectorAll（）返回所有CSS类对象

c# .NET引擎

Java JAVA引擎

C/C++

|  |  |  |
| --- | --- | --- |
| 浏览器 | 引擎名称 | 备注 |
| Safari | Nitro |  |
| Chrome | V8 |  |
| IE | Chakra |  |
| Firefox |  |  |

HTML5表单

<form action=“url” method=“post” enctype=“”>

<input type=“file”>

（22页）

盒子阴影：box-shadow

## HTML5动画与绘图：

linear-gradient：

有3个参数：1、top/left方向。2、起点颜色。3、终点颜色

Cacvas绘图与SVG

视频与音频

Geolocation（定位）

WebSoket

Forms

拖放

WebStorage（网页存储，cookie的增强版）

## Transform 2D动画

2D转换：transform，它有一下几个值，能使用2D属性将HTML元素进行如下变化：

1.移动 translate

translate（x，y）百分比是按自身宽高移动。

X:x轴移动距离

y:y轴移动距离

1. 旋转transform：rotate 坐标轴（度数）

坐标轴有三个值：X，Y，Z，(45deg) deg是度数单位

1. 缩放scale(1.5,0.5)倍数
2. 倾斜

skew(45deg,-45deg)x轴y轴倾斜

注意：一起使用的话会被覆盖。

5：matrix（a,b,c,d,e,f）矩阵 的平移公式：

X+=a\*x+c\*y+e；x的系数是缩放，常数是平移

Y+=b\*x+d\*y+f；y的系数是缩放，常数是平移

矩阵的旋转公式：

matrix（cos0，sin0，-sin0，cos0，0，0）；

0代表角度，cos30°=0.86625

## 背景颜色的变换：linear-gradient（）

1.background:linear-gradient(to bottom,red,yellow);从上到下有红变黄

linear-gradient（）参数有：渐变方向，起始颜色，颜色1，颜色2。。。。。。

1.线性变换：渐变方向从左到右或从上到下

第一种：to right/left/top/buttom/left top/right top/left bottom/right bottom

第二种：角度，范围是0-360deg其中0°-top，90°-rignt，180°-bottom，270°-left，360°-top。

2.径向变换：由内而外的变换颜色

background:radial-gradient( 渐变形状，大小 ，at 位置， 颜色1,颜色2，。。。。。。。);

渐变形状的值有：默认是随着盒子宽高而改变、circle圆形，

注意，一个动画效果只能设置一次旋转或者平移或者倾斜，如果同时设置两个或以上的2D动画，那么先设置的效果会被后设置的所覆盖。

## 动画过渡transition：

一个元素在不同状态之间进行平滑转换，CSS3使用transition实现过渡效果。

过渡属性：执行时间，时间函数（速率），延迟时间。

过渡属性是可选值 缺省就是all

执行时间：执行动画的时间

时间函数：速率，时间比例。比如总时间10秒，距离100米，匀速的话10米每秒先快后慢 前6s跑90m，后4s跑10m。时间函数有：

linear 匀速

ease-in快速进来

ease-out快速出去

ease-in-out快进快出

在开发过程中，transition与transform配合使用，而@keyframes关键帧与animation配合使用。

## @keyframes关键帧：

动画都是由关键帧组成，我们只需要设置几个关键帧，剩下的过渡部分浏览器自动生成连续动画。

animation 动画：animation动画是异步执行的

Webkit内核浏览器要加上-Webkit-前缀

声明关键帧：

@keyframes mybox 动画名{

0%{样式}

25%{样式}

50%{样式}

75%{样式}

100%{样式}

}

例如：

@-webkit-keyframes 动画名（chrome Safrai）

为了解决兼容性问题，通常两种都要写。

-webkit-animation

生成动画步骤：

1. 声明一组关键帧

@keyframes mybox{

from{

起始样式 }

to{

结束样式 }

}

@-webkit-keyframes mybox{

from{

width: 50px;

}

to{

width: 200px;

}

}

或者：

@keyframes mybox{

0%{width: 50px;height: 100px;}

25%{width: 100px;height: 100px;background: blue;}

50%{width: 200px;height: 100px;}

75%{width: 200px;height: 150px;}

100%{width: 200px;height: 200px;}

}

@-webkit-keyframes mybox1{

0%{width: 50px;height: 100px;}

25%{width: 100px;height: 100px;background: blue;}

50%{width: 200px;height: 100px;}

75%{width: 200px;height: 150px;}

100%{width: 200px;height: 200px;}

}

调用：

div:hover{

1、animation-name: mybox1;调用名字

2、animation-duration: 2s;规定动画完成一个周期所花费的秒或毫秒。默认是 0。

3、[animation-timing-function](http://www.w3school.com.cn/cssref/pr_animation-timing-function.asp" \o "CSS3 animation-timing-function 属性)：规定动画的速度曲线。默认是 "ease"

linear 匀速

ease-in快速进来

ease-out快速出去

ease-in-out快进快出

4、animation-delay: 3s;设置延时，规定动画何时开始

5、animation-iteration-count:设置动画播放次数，默认是1次，值为1,2,3.。。，infinite：无限次播放

6、[animation-direction](http://www.w3school.com.cn/cssref/pr_animation-direction.asp" \o "CSS3 animation-direction 属性)：规定动画是否在下一周期逆向地播放。

默认是 "normal",Alternate：动画应该轮流反向播放。

7、animation-play-state:设置当前播放状态：其值为：

running，播放状态

pushed：暂停状态

8、animation-fill-mode：设置播放后的状态。其值为：

backwards：与播放之前的关键帧保持一致；

forwards：与播放之后的关键帧保持一致；

both：=backwards+forwards设置播放前后的样式，

默认值是null

}

连写：

#box2{-webkit-animation:mybox2 1s 1s linear infinite;}

连写规则：{-webkit-animation:动画名->播放时间->动画的速度曲线（时间函数->延迟时间->播放次数->(周期逆向地播放)->当前播放状态 播放过后的状态}

连写时必须注意：name 和动画时间必须有，时间函数可有可无，若有两个时间，则前面为播放时间，后面的为延迟时间。

注意：360包括360浏览器里的IE浏览器所有版本完全不兼容。

## 3D模型转换

使用transform属性将HTML元素在3D空间内

旋转 rotate

移动 translate

缩放 scale

transform: rotateX(60deg);

perspective-origin（X Y Z）：设置当前元素的坐标系视角；值有

X 当前元素自身的X坐标 left  
center：中心

## Canvas绘图：

Canvas 画布、绘画

要在web页面上进行绘制图像、图形

在<canvas>标签内书写

必须书写这样一行代码：

<canvas id="canvas" style="border:1px solid #00FFFF">当前版本过低,请升级浏览器!!!</canvas><canvas id="canvas" style="border:1px solid #00FFFF">当前版本过低,请升级浏览器!!!</canvas><canvas id="canvas" style="border:1px solid #00FFFF">当前版本过低,请升级浏览器!!!</canvas>

var draw\_border=document.getElementById("canvas");//声明画板

draw\_border.width=900;//设置画板宽高

draw\_border.height=600;

var context=draw\_border.getContext("2d");//3.生命笔拿到画纸,上下文

context.lineWidth=5; //笔的属性:

context.strokeStyle="red";//笔的颜色

context.beginPath();//开始绘画

context.moveTo(100,100);//落笔的位置:书写的起点

context.lineTo(200,200);//书写的终点

context.stroke();//绘制

context.closePath();//将两点连接起来,声明绘制完成

//再来一根

context.beginPath();

context.lineWidth=7;

context.strokeStyle="darkblue";//笔的颜色

context.moveTo(200,200);//落笔的位置:书写的起点

context.lineTo(300,200);//默认

context.lineTo(400,500);

context.lineTo(600,500);

context.lineTo(200,200);

context.stroke();//绘制

// context.closePath();//将两点连接起来,声明绘制完成

context.fillStyle="red";//设置填充颜色

context.fill();设置图形填充

Canvas与Svg的区别：

Canvas适合像素处理、动态渲染和大量数据的绘制

Svg试用场景：

1：静态图、高保真

|  |  |  |
| --- | --- | --- |
|  | Canvas | SVG |
| 功能 | 功能简单，2d绘图API | 绘制图形、滤镜、动画 |
| 特点 | 像素、只能用脚本驱动 | HTML、CSS、元素操作 |
| 支持 | 主流浏览器>IE9 | >IE9 |
| 元素 | 单个<canvas>标签 | 多个<svg>元素 |
| 适合 | 小面积，大数据 | 大面积，小数据 |

CSS3媒体查询：

使用 @media 查询，你可以针对不同的媒体类型定义不同的样式。

@media 可以针对不同的屏幕尺寸设置不同的样式，特别是如果你需要设置设计响应式的页面，@media 是非常有用的。

当你重置浏览器大小的过程中，页面也会根据浏览器的宽度和高度重新渲染页面。

@media only screen and (max-width：xxpx){

css样式

}

@media only screen and (min-width：xxpx){

css样式

}

常见错：

①：@media only screen and (

这几个单词之间必须加空格。

②：(min-width：xxpx)中：xxpx)之间不能加分号。

@media only screen and (max-width: 500px) {  
    .gridmenu {  
        width:100%;  
    }  
  
    .gridmain {  
        width:100%;  
    }  
  
    .gridright {  
        width:100%;  
    }  
}

@media only screen and (max-width: 500px) {  
    .gridmenu {  
        width:100%;  
    }  
  
    .gridmain {  
        width:100%;  
    }  
  
    .gridright {  
        width:100%;  
    }  
}

## HTML5 音频和视频

国内：Audio Video Interleavel（.avi）

Flash Video(.flv).swf

MPEG4(MP4)

Motroska(.mkv)

Ogg(.ogv)

音频和视频的视频编码格式

音频：AAC ，MPGE-3，Ogg Virbits

视频编码:H.264，VP8，WebM（Google Twiter）

HTML5Audio与Video的限制

有些功能是HTML5规范所不支持的，如下所示：

流式音频和视频。因为视频规范比特率转换。

先了解2个概念： 容器和编解码器

视频容器

音频文件或者视频文件，都只是一个容器文件。视频文件里面包含了音频轨道、视频轨道和一些其他数据。视频播放的时候，音频、视频轨道视绑定在一起的，其他数据（视频的封面、标题、字幕等相关信息）作为其他数据放在不同渠道。目前主流的视频格式为 .mp4 .mkv .flv .ogg .rmvb .avi 等。

编解码：

因为音频、视频都是通过特殊编码进行播放的，它首先是通过特定的编码进行格式上的变化，然后通过对应的解码进行播放，其实就是 编和解的过程。如果不对齐进行解码，就无法将编码后的数据 组成媒体数据进行播放。

主要的音频编解码有：AAC MPEG-3 Ogg

视频的编解码是：H.264 Ogg

Audio和video标签：

Video：播放视频的标签，poster：设置视频封面，指定数据载入时显示的图片。

Loop：设置反复播放，单曲循环

src：视频资源的url

Width/height:设置播放器宽高。注意：-

Controls：设置后表示显示播放器

Autoplay：设置打开网页后自动播放

Video的方法（js部分）：

Load（）加载视频/音频文件，播放做准备

CanPlayType（type）video是否支持播放给定格式的文件格式

play();播放

pause():暂停

.webkitRequestFullScreen();全屏播放

Audio ：用来播放音频的标签

Src，autoplay，controls与video标签一样。

// pushed属性:表示音乐是否停止播放

ended属性播放完毕

Starttime返回播放的最初时间

error：发生错误的时候返回错误代码

CurrertSrc 以字符串形式返回当前播放的文件名称

// currentTime属性：表示当前音频播放到哪个时间点

// duration属性：整段音频的总时长

Apache WebHTTP Sever 协议

**[embed标签的使用(在网页中播放各种音频视频的插件的使用)](http://www.cnblogs.com/roychen/archive/2007/11/02/946771.html)**

 播放器插件使用说明:

代码：

< embed src=“music.mid”autostart=“true”loop=“2”width=“80”height=“30”>   
src：音乐文件的路径及文件名；（完整的路径或URL）   
ShowTracker:为是否显示播放进度条   
ShowPositionControls:为是否显示播放控制按钮如快进等   
ShowAudioControls: 为控制是否显示音量按钮   
ShowStatusBar: 是否显示咨询窗  
ShowDisplay: 为显示更完整的咨询视窗  
EnableContextMenu: 防止使用右键  
autostart：true为音乐文件上传完后自动开始播放，默认为false（否）   
loop：true为无限次重播，false为不重播，某一具体值（整数）为重播多少次   
volume：取值范围为“0-100”，设置音量，默认为系统本身的音量   
starttime：“分：秒”，设置歌曲开始播放的时间，如，starttime=“00:10”，从第10开始播放   
endtime： “分：秒”，设置歌曲结束播放的时间   
width：控制面板的宽   
height：控制面板的高   
controls：控制面板的外观   
controls=“console/smallconsole/playbutton/pausebutton/stopbutton/volumelever”

·console：正常大小的面板   
·smallconsole：较小的面板   
·playbutton：显示播放按钮   
·pausebutton：显示暂停按钮   
·stopbutton：显示停止按钮   
·volumelever：显示音量调节按钮

hidden：为true时可以隐藏面板

embed标签src后的文件也可以是.swf文件，用来播FLASH

type属性：设置浏览器用什么样的播放器播放。

type="application/x-shockwave-flash"：用flash播放，适用于flash视频

type="application/x-mplayer3"：用MP3播放，适用于音频

## LAMP开发模式

L：Linux

A：Apache

M：MySQL

P：PHP

Localhost/127.0.0.1都表示本地存储

# **SVG图像的绘制与操作**

## SVG简介

### 什么是SVG？

SVG 指可伸缩矢量图形 (Scalable Vector Graphics)

SVG 用来定义用于网络的基于矢量的图形

SVG 使用 XML 格式定义图形

SVG 图像在放大或改变尺寸的情况下其图形质量不会有所损失

SVG 是万维网联盟的标准

SVG 与诸如 DOM 和 XSL 之类的 W3C 标准是一个整体

SVG 是使用 XML 来描述二维图形和绘图程序的语言。

### SVG实例：

<?xml version="1.0" standalone="no"?>

<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"

"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

<svg width="100%" height="100%" version="1.1"

xmlns="http://www.w3.org/2000/svg">

<circle cx="100" cy="50" r="40" stroke="black"

stroke-width="2" fill="red"/>

</svg>

### **SVG属性：**

standalone 属性：该属性规定此 SVG 文件是否是“独立的”，或含有对外部文件的引用。

standalone="no" 意味着 SVG 文档会引用一个外部文件 - 在这里，是 DTD 文件。

第二和第三行引用了这个外部的 SVG DTD。该 DTD 位于 “http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd”。该 DTD 位于 W3C，含有所有允许的 SVG 元素。

<svg> 标签：

SVG 代码以 <svg> 元素开始，包括开启标签 <svg> 和关闭标签 </svg> 。这是根元素。width 和 height 属性可设置此 SVG 文档的宽度和高度。version 属性可定义所使用的 SVG 版本。

关闭标签的作用是关闭 SVG 元素和文档本身。

**注释：**所有的开启标签必须有关闭标签！

xmlns命名空间：xmlns 属性可定义 SVG 命名空间。此属性表示<svg>标记及其子标记属于名称空间为“[http://www.w3.org/2000/svg](https://link.jianshu.com/?t=http://www.w3.org/2000/svg" \t "https://www.jianshu.com/p/_blank)”的XML方言，当然，它是SVG。注意，命名空间声明只需要在根标记上提供一次。

<circle>标签：

SVG 的 <circle> 用来创建一个圆。cx 和 cy 属性定义圆中心的 x 和 y 坐标。如果忽略这两个属性，那么圆点会被设置为 (0, 0)。r 属性定义圆的半径。

stroke 和 stroke-width 属性控制如何显示形状的轮廓。我们把圆的轮廓设置为 2px 宽，黑边框。

fill 属性设置形状内的颜色。

## viewport,viewBox,preserveAspectRatio缩放

### Viewport

就是SVG的宽高，表示SVG可见区域的大小，或者可以想象成舞台大小，画布大小。

<svg width="500" height="300"></svg>

上面的SVG代码定义了一个视区，宽500单位，高300单位。

注意这里的措辞是“单位”，不是“像素”。虽然说，width/height如果是纯数字，使用的就是“像素（px）”作为单位的。

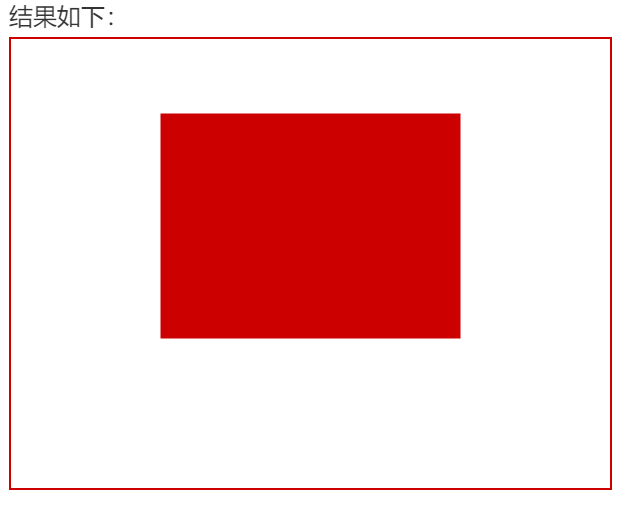
### viewBox属性

“视区盒子”,自己的理解就是在SVG画布里再套一层视图层，然后自适应撑满Viewport。宽高比通常可与Viewport成比例。

<svg width="400" height="300" viewBox="0,0,40,30" style="border:1px solid #cd0000;">

<rect x="10" y="5" width="20" height="15" fill="#cd0000"/>

</svg>



viewBox="x, y, width, height" // x:左上角横坐标，y:左上角纵坐标，width:宽度，height:高度

viewBox顾名思意是“视区盒子”的意思，好比在说：“SVG啊，要不你就让我铺满你吧~”

更形象的解释就是：SVG就像是我们的显示器屏幕，viewBox就是截屏工具选中的那个框框，最终的呈现就是把框框中的截屏内容再次在显示器中全屏显示！

### preserveAspectRatio属性

上面的例子，SVG的宽高比正好和viewBox的宽高比是一样的，都是4:3. 显然，实际应用viewBox不可能一直跟viewport穿同一条开裆裤。此时，就需要preserveAspectRatio出马了，此属性也是应用在<svg>元素上，且作用的对象都是viewBox。

preserveAspectRatio属性的值为空格分隔的两个值组合而成。例如，上面的xMidYMid和meet.

第1个值表示，viewBox如何与SVG viewport对齐；第2个值表示，如何维持高宽比（如果有）。

其中，第1个值又是由两部分组成的。前半部分表示x方向对齐，后半部分表示y方向对齐。家族成员如下：

|  |  |
| --- | --- |
| 值 | 含义 |
| xMin | viewport和viewBox左边对齐 |
| xMid | viewport和viewBox x轴中心对齐 |
| xMax | viewport和viewBox右边对齐 |
| YMin | viewport和viewBox上边缘对齐。注意Y是大写。 |
| YMid | viewport和viewBox y轴中心点对齐。注意Y是大写。 |
| YMax | viewport和viewBox下边缘对齐。注意Y是大写。 |

x, y自由合体就可以了，如：xMaxYMax xMidYMid

preserveAspectRatio属性第2部分的值支持下面3个：

|  |  |
| --- | --- |
| 值 | 含义 |
| meet | 保持纵横比缩放viewBox适应viewport，受 |
| slice | 保持纵横比同时比例小的方向放大填满viewport，攻 |
| none | 扭曲纵横比以充分适应viewport，变态 |

两种属性综合起来就是preserveAspectRatio: "xMinYMin meet",

# Not Afraid

* 专辑：[Recovery](orpheus://orpheus/pub/app.html" \l "/m/album/?id=133852" \o "Recovery)
* 歌手：[Eminem](orpheus://orpheus/pub/app.html" \l "/m/artist/?id=32665" \o "Eminem)
* 来源：[Eminem 热门50单曲](orpheus://orpheus/pub/app.html" \l "/m/playlist/?id=622845532&rid=A_PL_0_622845532&fromSource=1)

I'm not afraid, I'm not afraid!

我从未畏惧，从不惧怕

(Yeah)

耶

To take a stand, to take a stand

诉说我的立场，诉说我的立场

(It's been a ride)

就像是在奔腾

Everybody, everybody

众人齐心，众人齐心

(I guess I had to)

我想我必须要

Come take my hand, come take my hand

拥有属于我的力量，拥有属于我的力量

(Go to that place to get to this one)

避开这个地方

We'll walk this world together through the storm

让我们荣辱与共，冲过暴雨疾风

(Now some of you)

此时此刻的你

Whatever weather, cold or warm

管它什么天气，严冬或许酷暑

(Might still be in that place)

大概还会停滞不前

Just lettin' you know that you're not alone

我只想让你知道，我始终都会在你身旁

(If you're tryin' to get out)

如果你也想逃离出来

Holla if you feel like you've been down the same road

我们肝胆相照，在困难重重的人生当中，携手共行

(Just follow me, I'll get you there)

跟着我，带你去梦想的出生点

You can try read my lyrics off of this paper before I lay 'em

在我还没出专辑，你大可知道我的歌词

But you won't take the sting out these words before I say 'em

但你坚决不会理解它的内涵，除非我讲释给你听

'Cause ain't no way I'ma let you stop me from causing mayhem

我绝对不会让你这样的愚昧之人成为我前进的脚步

When I say I'ma do something, I do it, I don't give a damn what you think

无论我做什么，才不会在意你怎么想

I'm doin' this for me, so fuck the world

我只会为我自己而活着，你快滚蛋吧

Feed it beans, it's gassed up if it thinks it's stopping me

跟你并没有任何关系，这真的太恶心人了，如果有人阻止我的话

I'ma be what I set out to be, without a doubt, undoubtably

我只会永远的做回我自己

And all those who look down on me, I'm tearing down your balcony

至于那些曾经唾弃我的傻子，我会不费吹灰之力的将你踩在脚下

No if, ands, or buts, don't try to ask him why or how can he

并没有假如，他的成就不是那么容易得来的

From "Infinite" down to that last "Relapse" album

从此时此刻，只知道现在

He's still shittin' whether he's on salary, paid hourly

他在鬼混，他曾经拥有着别人有不了的薪酬

Until he bows out or he shits his bowels out of him

但他的负面新闻从未消停过

Whichever comes first, for better or worse

不管是好是坏，他都无所谓

He's married to the game, like a fuck-you for Christmas

他玩弄婚姻，就像过家家

His gift is a curse, forget the earth, he's got

他的才能根本就是诅咒，把一切都忘记，重新振作起来

The urge to pull his dick from the dirt and fuck the whole universe

老天算个屁！只有我才能主宰我自己

I'm not afraid, I'm not afraid

我从未畏惧，从不惧怕

To take a stand, to take a stand

诉说我的立场，诉说我的立场

Everybody, everybody

众人齐心，众人齐心

Come take my hand, come take my hand

拥有属于我的力量，拥有属于我的力量

We'll walk this road together through the storm、

# 网络端口与HTTP协议

1.IP地址是一个32位的二进制数，通常被分割为4个“8位二进制数”（也就是4个字节）。IP地址通常用“点分十进制”表示成（a.b.c.d）的形式，其中，a,b,c,d都是0~255之间的十进制整数。例：点分十进IP地址（100.4.5.6   
），实际上是32位二进制数（01100100.00000100.00000101.00000110）。  
255以内  
:后边是端口号，最高65535

127.0.0.1 本机 只能只本机  
192.168.xx 局域网内任意一台  
192.168.xx 也可叫lan网

127.0.0.1是localhost ，本机电脑的IP地址。192.168.xx是局域网内任意一台电脑的IP地址。

二者一个是针对自己本机访问，一个是针对局域网内其他计算机访问自己的电脑。

# 兼容性问题汇总（http://caniuse.com/#feat=webgl ）

## IE兼容性问题汇总与解决方法

1、IE6怪异解析之padding与border算入宽高   
原因：未加文档声明造成非盒模型解析   
解决方法：加入文档声明<!doctype html>   
2、IE6在块元素、左右浮动、设定marin时造成margin双倍（双边距）   
解决方法：display:inline   
3、以下三种其实是同一种bug，其实也不算是个bug，举个例子：父标签高度20，子标签11，垂直居中，20-11=9，9要分给文字的上面与下面，怎么分？IE6就会与其它的不同，所以，尽量避免。   
1）字体大小为奇数之边框高度少1px   
解决方法：字体大小设置为偶数或line-height为偶数   
2）line-height，文本垂直居中差1px   
解决方法：padding-top代替line-height居中，或line-height加1或减1   
3）与父标签的宽度的奇偶不同的居中造成1px的偏离   
解决方法：如果父标签是奇数宽度，则子标签也用奇数宽度;如果是父标签偶数宽度，则子标签也用偶数宽度   
4、内部盒模型超出父级时，父级被撑大   
解决方法：父标签使用overflow:hidden   
5、line-height默认行高bug   
解决方法：line-height设值   
6、行标签之间会有一小段空白   
解决方法：float或结构并排(可读性差，不建议)   
7、标签高度无法小于19px   
解决方法：overflow: hidden;   
8、左浮元素margin-bottom失效   
解决方法：显示设置高度 or 父标签设置\_padding-bottom代替子标签的margin-bottom or 再放个标签让父标签浮动，子标签   
margin- bottom，即(margin-bottom与float不同时作用于一个标签)   
9、img于块元素中，底边多出空白   
解决方法：父级设置overflow: hidden; 或 img { display: block; } 或 \_margin: -5px;   
10、li之间会有间距   
解决方法：float: left;   
11、块元素中有文字及右浮动的行元素，行元素换行   
解决方法：将行元素置于块元素内的文字前   
12、position下的left，bottom错位   
解决方法：为父级(relative层)设置宽高或添加\*zoom:1   
13、子级中有设置position，则父级overflow失效   
解决方法：为父级设置position:relative  
以下是补充：上面要先看  
1、终极方法：条件注释   
<!--[if lte IE 6]> 这段文字仅显示在 IE6及IE6以下版本。 <![endif]-->   
<!--[if gte IE 6]> 这段文字仅显示在 IE6及IE6以上版本。 <![endif]-->   
<!--[if gt IE 6]> 这段文字仅显示在 IE6以上版本（不包含IE6）。 <![endif]-->   
<!--[if IE 5.5]> 这段文字仅显示在 IE5.5。 <![endif]-->   
<!--在 IE6及IE6以下版本中加载css-->   
<!--[if lte IE 6]> <link type="text/css" rel="stylesheet" href="css/ie6.css" mce\_href="css/ie6.css" /><![endif]-->   
缺点是在IE浏览器下可能会增加额外的HTTP请求数。   
2、CSS选择器区分   
IE6不支持子选择器；先针对IE6使用常规申明CSS选择器，然后再用子选择器针对IE7+及其他浏览器。   
/\* IE6 专用 \*/   
.content {color:red;}   
/\* 其他浏览器 \*/   
div>p .content {color:blue;} -->   
3、PNG半透明图片的问题   
虽然可以通过JS等方式解决，但依然存在载入速度等问题，所以，这个在设计上能避免还是尽量避免为好。以达到网站最大优化。   
4、IE6下的圆角   
IE6不支持CSS3的圆角属性，性价比最高的解决方法就是用图片圆角来替代，或者放弃IE6的圆角。   
5、IE6背景闪烁   
如果你给链接、按钮用CSS sprites作为背景，你可能会发现在IE6下会有背景图闪烁的现象。造成这个的原因是由于IE6没有将背景图缓存，每次触发hover的时候都会重新加载，可以用JavaScript设置IE6缓存这些图片：   
document.execCommand("BackgroundImageCache",false,true);   
6、最小高度   
IE6 不支持min-height属性，但它却认为height就是最小高度。解决方法：使用ie6不支持但其余浏览器支持的属性!important。   
#container {min-height:200px; height:auto !important; height:200px;}   
7、最大高度   
//直接使用ID来改变元素的最大高度   
var container = document.getElementById('container');   
container.style.height = (container.scrollHeight > 199) ? "200px" : "auto";   
//写成函数来运行   
function setMaxHeight(elementId, height){   
var container = document.getElementById(elementId);   
container.style.height = (container.scrollHeight > (height - 1)) ? height + "px" : "auto";   
}   
//函数示例   
setMaxHeight('container1', 200);   
setMaxHeight('container2', 500);   
8、100% 高度   
在IE6下，如果要给元素定义100%高度，必须要明确定义它的父级元素的高度，如果你需要给元素定义满屏的高度，就得先给html和body定义height:100%;。   
9、最小宽度   
同max-height和max-width一样，IE6也不支持min-width。   
//直接使用ID来改变元素的最小宽度   
var container = document.getElementById('container');   
container.style.width = (container.clientWidth < width) ? "500px" : "auto";   
//写成函数来运行   
function setMinWidth(elementId, width){   
var container = document.getElementById(elementId);   
container.style.width = (container.clientWidth < width) ? width + "px" : "auto";   
}   
//函数示例   
setMinWidth('container1', 200);   
setMinWidth('container2', 500);   
10、最大宽度   
//直接使用ID来改变元素的最大宽度   
var container = document.getElementById(elementId);   
container.style.width = (container.clientWidth > (width - 1)) ? width + "px" : "auto";  
//写成函数来运行   
function setMaxWidth(elementId, width){   
var container = document.getElementById(elementId);   
container.style.width = (container.clientWidth > (width - 1)) ? width + "px" : "auto";  
}   
//函数示例   
setMaxWidth('container1', 200);   
setMaxWidth('container2', 500);   
11、双边距Bug   
当元素浮动时，IE6会错误的把浮动方向的margin值双倍计算。个人觉得较好解决方法是避免float和margin同时使用。   
12、清除浮动   
如果你想用div(或其他容器)包裹一个浮动的元素，你会发现必须给div(容器)定义明确的height、width、overflow之中一个属性（除了auto值）才能将浮动元素严实地包裹。  
#container {border:1px solid #333; overflow:auto; height:100%;}   
#floated1 {float:left; height:300px; width:200px; background:#00F;}   
#floated2 {float:right; height:400px; width:200px; background:#F0F;}   
更多：http://www.twinsenliang.net/skill/20090413.html   
13、浮动层错位   
当内容超出外包容器定义的宽度时，在IE6中容器会忽视定义的width值，宽度会错误地随内容宽度增长而增长。   
浮动层错位问题在IE6下没有真正让人满意的解决方法，虽然可以使用overflow:hidden;或overflow:scroll;来修正， 但hidden容易导致其他一些问题，scroll会破坏设计；JavaScript也没法很好地解决这个问题。所以建议是一定要在布局上避免这个问题发 生，使用一个固定的布局或者控制好内容的宽度（给内层加width）。   
14、躲猫猫bug   
在IE6和IE7下，躲猫猫bug是一个非常恼人的问题。一个撑破了容器的浮动元素，如果在他之后有不浮动的内容，并且有一些定义了:hover的链接，当鼠标移到那些链接上时，在IE6下就会触发躲猫猫。   
解决方法很简单：   
1.在（那个未浮动的）内容之后添加一个<span style="clear: both;"> </span>   
2.触发包含了这些链接的容器的hasLayout，一个简单的方法就是给其定义height:1%;   
15、绝对定位元素的1像素间距bug   
IE6下的这个错误是由于进位处理误差造成（IE7已修复），当绝对定位元素的父元素高或宽为奇数时，bottom和right会产生错误。唯一的解决办法就是给父元素定义明确的高宽值，但对于液态布局没有完美的解决方法。   
16、3像素间距bug   
在IE6中，当文本(或无浮动元素)跟在一个浮动的元素之后，文本和这个浮动元素之间会多出3像素的间隔。   
给浮动层添加 display:inline 和 -3px 负值margin   
给中间的内容层定义 margin-right 以纠正-3px   
17、IE下z-index的bug   
在IE浏览器中，定位元素的z-index层级是相对于各自的父级容器，所以会导致z-index出现错误的表现。解决方法是给其父级元素定义z-index，有些情况下还需要定义position:relative。   
18、Overflow Bug   
在IE6/7中，overflow无法正确的隐藏有相对定位position:relative;的子元素。解决方法就是给外包容器.wrap加上position:relative;。   
19、横向列表宽度bug   
如果你使用float:left;把<li>横向摆列，并且<li>内包含的<a>（或其他）触发了 hasLayout，在IE6下就会有错误的表现。解决方法很简单，只需要给<a>定义同样的float:left;即可。   
20、列表阶梯bug   
列表阶梯bug通常会在给<li>的子元素<a>使用float:left;时触发，我们本意是要做一个横向的列表(通常 是导航栏)，但IE却可能呈现出垂直的或者阶梯状。解决办法就是给<li>定义float:left;而非子元素<a>，或者 给<li>定义display:inline;也可以解决。   
21、垂直列表间隙bug   
当我们使用<li> 包含一个块级子元素时，IE6(IE7也有可能)会错误地给每条列表元素（<li>）之间添加空隙。   
解决方法：把<a>flaot并且清除float来解决这个问题；另外一个办法就是触发<a>的hasLayout（如定 义高宽、使用zoom:1;）；也可以给<li> 定义display:inline;来解决此问题；另外还有一个极有趣的方法，给包含的文本末尾添加一个空格。   
22、IE6中的:hover   
在IE6中，除了(需要有href属性)才能触发:hover行为，这妨碍了我们实现许多鼠标触碰效果，但还是有一些法子是可以解决它的。最好是不要用:hover来实现重要的功能，仅仅只用它来强化效果。   
23、IE6调整窗口大小的 Bug   
当把body居中放置，改变IE浏览器大小的时候，任何在body里面的相对定位元素都会固定不动了。解决办法：给body定义position:relative;就行了。   
24、文本重复Bug   
在IE6中，一些隐藏的元素（如注释、display:none;的元素）被包含在一个浮动元素里，就有可能引发文本重复bug。解决办法：给浮动元素添加display:inline;

## 完美解决header,footer等HTML5标签在IE(IE6/IE7/IE8)无效的兼容性问题

HTML5的语义化标签以及属性，可以让开发者非常方便地实现清晰的web页面布局，加上CSS3的效果渲染，快速建立丰富灵活的web页面显得非常简单

HTML5的语义化标签以及属性，可以让开发者非常方便地实现清晰的web页面布局，加上CSS3的效果渲染，快速建立丰富灵活的web页面显得非常简单。

HTML5的新标签元素有：

<header>定义页面或区段的头部；

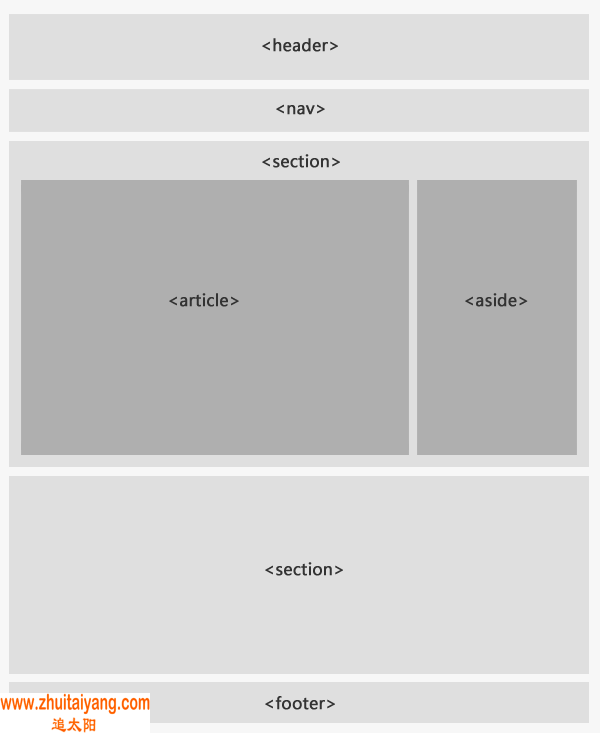
<footer>定义页面或区段的尾部；

<nav>定义页面或区段的导航区域；

<section>页面的逻辑区域或内容组合；

<article>定义正文或一篇完整的内容；

<aside>定义补充或相关内容；



使用他们能让代码语义化更直观,而且更方便SEO优化。但是此HTML5新标签在IE6/IE7/IE8上并不能识别，需要进行JavaScript处理。以下就介绍几种方式。

方式一：Coding JavaScript

<!--[if lt IE9]>

<script>

(function() {

if (!

/\*@cc\_on!@\*/

0) return;

var e = "abbr, article, aside, audio, canvas, datalist, details, dialog, eventsource, figure, footer, header, hgroup, mark, menu, meter, nav, output, progress, section, time, video".split(', ');

var i= e.length;

while (i--){

document.createElement(e[i])

}

})()

</script>

<![endif]-->

如果是IE9以下的IE浏览器将创建HTML5标签， 这样非IE浏览器就会忽视这段代码，也就不会有无谓的http请求了。

第二种方法：使用Google的html5shiv包（推荐）

<!--[if lt IE9]>

<script src="http://html5shiv.googlecode.com/svn/trunk/html5.js"></script>

<![endif]-->

但是不管使用以上哪种方法,都要初始化新标签的CSS.因为HTML5在默认情况下表现为内联元素，对这些元素进行布局我们需要利用CSS手工把它们转为块状元素方便布局

/\*html5\*/

article,aside,dialog,footer,header,section,footer,nav,figure,menu{display:block}

但是如果ie6/7/8 禁用脚本的用户,那么就变成了无样式的"白板"网页,我们该怎么解决呢?

我们可以参照facebook的做法，即引导用户进入带有noscript标识的 “/?\_fb\_noscript=1”页面，用 html4 标签替换 html5 标签，这要比为了保持兼容性而写大量 hack 的做法更轻便一些。

<!--[if lte IE 8]>

<noscript>

<style>.html5-wrappers{display:none!important;}</style>

<div class="ie-noscript-warning">您的浏览器禁用了脚本，请<a href="">查看这里</a>来启用脚本!或者<a href="/?noscript=1">继续访问</a>.

</div>

</noscript>

<![endif]-->

由于IE不兼容H5新标签而造成的布局排版混乱：

IE8下打开布局全乱了，原因是IE8不支持header，nav，aside等h5标签。解决方法：

在script标签里加了这样一个函数，来判断IE9以下版本，使其兼容新标签。

(function() {

if (!

/\*@cc\_on!@\*/

0) return;

var e = "abbr, article, aside, audio, canvas, datalist, details, dialog, eventsource, figure, footer, header, hgroup, mark, menu, meter, nav, output, progress, section, time, video".split(', ');

var i= e.length;

while (i--){

document.createElement(e[i])

}

})()

[在easyUI的datagrid中，IE浏览器报错:SCRIPT5007: 无法获取属性“rowspan”的值: 对象为 null 或未定义](http://blog.csdn.net/civilized/article/details/8588742)，导致数据解析发生错误：

错误：SCRIPT5007: 无法获取属性“rowspan”的值: 对象为 null 或未定义。

造成这种错误的原因大致为：(1) 可能是在datagrid里面下的frozenColumns:[[ {......}]]或者是columns:[[{...},{...}]]   {...}之间多一个逗号(,)。

## 各浏览器常用兼容标记一览表:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 标记 | IE6 | IE7 | IE8 | FF | Opera | Sarari |
| [\*+><] | √ | √ | X | X | X | X |
| \_ | √ | X | X | X | X | X |
| \9 | √ | √ | √ | X | X | X |
| \0 | X | X | √ | X | √ | X |
| @media screen and (-webkit-min-device-pixel-ratio:0){.bb {}} | X | X | X | X | X | √ |
| .bb , x:-moz-any-link, x:default | X | √ | X | √(ff3.5及以下) | X | X |
| @-moz-document url-prefix(){.bb{}} | X | X | X | √ | X | X |
| @media all and (min-width: 0px){.bb {}} | X | X | X | √ | √ | √ |
| \* +html .bb {} | X | √ | X | X | X | X |
| 游览器内核 | Trident | Trident | Trident | Gecko | Presto | WebKit |

（以上 .bb 可更换为其它样式名）

注意点：

网上很多资料中常常把!important也作为一个hack手段，其实这是一个误区。!important常常被我们用来更改样式，而不是兼容hack。造成这个误区的原因是IE6在某些情况下不主动识别!important,以至于常常被人误用做识别IE6的hack。可是，大家注意一下，IE6只是在某些情况下不识别（ie6下，同一个大括号里对同一个样式属性定义，其中一个加important 则important标记是被忽略的，例：{background:red!important; background:green;} ie6下解释为背景色green，其它浏览器解释为背景色red；如果这同一个样式在不同大括号里定义，其中一个加important 则important发挥正常作用，例：div{background:red!important} div{background:green}，这时所有浏览器统一解释为背景色red。）

实例讲解：

Hack应用情境（一）

适用范围：IE:6.0,IE7.0,IE8.0之间的兼容

实例说明：

* 此例中我们使用了渐进识别的方式，从总体中逐渐排除局部。首先，巧妙的使用“\9”这一标记，将IE游览器从所有情况中分离出来。接着，再次使用“+”将IE8和IE7、IE6分离开来，此时，我们的IE8已经独立识别。

实例代码：

.bb{  
height:32px;  
background-color:#f1ee18;/\*所有识别\*/  
.background-color:#00deff\9; /\*IE6、7、8识别\*/  
+background-color:#a200ff;/\*IE6、7识别\*/  
\_background-color:#1e0bd1;/\*IE6识别\*/  
}

/\*一个用于展示的class为bb的div标签\*/

* < div class ="bb"></ div >

Hack应用情境（二）

适用范围：IE:6.0,IE7.0,IE8.0,Firefox之间的兼容

实例说明：

大家很容易的可以看出这是情境（一）的加强版，适用于更广泛的环境。其实情境（一）中也已经做到了把火狐与IE游览器区分开来了，现在我们要做的是把火狐从其它游览器中再次识别出来。大家仔细看下代码，大家会发现其实游览器识别是很简单的。火狐如何识别？对了，IE中对伪类支持不广泛，所以伪类是个不错的途径。(.yourClass,x:-moz-any-link, x:default)注意，这个区分伪类往往IE7也能识别，所以最好还需要把IE7单独识别出来，且此方法对ff3.6 已无效，firefox的区分可以使用@-moz-document url-prefix(){}

实例代码：

.bb{  
height:32px;  
background-color:#f1ee18;/\*所有识别\*/  
background-color:#00deff\9; /\*IE6、7、8识别\*/  
+background-color:#a200ff;/\*IE6、7识别\*/  
\_background-color:#1e0bd1;/\*IE6识别\*/  
}  
.bb, x:-moz-any-link, x:default{background-color:#00ff00;}/\*IE7 firefox3.5及以下 识别 \*/   
@-moz-document url-prefix(){.bb{background-color:#00ff00;}}/\* 仅firefox 识别 \*/   
\* +html .bb{background-color:#a200ff;}/\* 仅IE7 识别 \*/

/\*一个用于展示的class为bb的div标签\*/

* < div class ="bb"></ div >

Hack应用情境（三）

适用范围：IE:6.0,IE7.0,IE8.0,Firefox,Safari(Chrome)之间的兼容

实例说明：

* 我们现在将再次对我们的CSS进行加强了，使其能识别Safari(Chrome)游览器。这是基于它们的内核webkit来识别的，用法为@media screen and (-webkit-min-device-pixel-ratio:0)

实例代码：

.bb{  
height:32px;  
background-color:#f1ee18;/\*所有识别\*/  
background-color:#00deff\9; /\*IE6、7、8识别\*/  
+background-color:#a200ff;/\*IE6、7识别\*/  
\_background-color:#1e0bd1;/\*IE6识别\*/  
}  
@media screen and (-webkit-min-device-pixel-ratio:0){.bb{background-color:#f1ee18}}{} /\*safari(Chrome) 有效 \*/  
.bb, x:-moz-any-link, x:default{background-color:#00ff00;}/\*IE7 firefox3.5及以下 识别 \*/   
@-moz-document url-prefix(){.bb{background-color:#00ff00;}}/\*仅firefox 识别\*/   
\* +html .bb{background-color:#a200ff;}/\* 仅IE7 识别 \*/

/\*一个用于展示的class为bb的div标签\*/

* < div class ="bb"></ div >

Hack应用情境（四）

适用范围：IE:6.0+，FireFox:2.0+，Opera 10.0+，Sarari 3.0+，Chrome全兼容

实例说明：

* 实例的具体代码在下面实例代码中已经列出，具体效果如此页面的顶端部分效果，您可以通过不同游览器检测该效果。这次我们基本把所有的主流游览器都兼容了，大家来看下代码。Opera的识别有一部分归功于“\0”标记，这个标记只被IE8和Opera识别，特殊的标记往往造就的是我们更广泛的hack手段。

## CSS兼容IE6 IE7 和FF的总结

一、CSS HACK  
以下两种方法几乎能解决现今所有HACK.  
1, !important  
随着IE7对!important的支持, !important 方法现在只针对IE6的HACK.(注意写法.记得该声明位置需要提前.)  
<stylegt;  
#wrapper  
{  
width: 100px!important; /\* IE7+FF \*/  
width: 80px; /\* IE6 \*/  
}  
</stylegt;  
2, IE6/IE77对FireFox  
\*+html 与 \*html 是IE特有的标签, firefox 暂不支持.而\*+html 又为 IE7特有标签.  
<stylegt;  
#wrapper  
{  
#wrapper { width: 120px; } /\* FireFox \*/  
\*html #wrapper { width: 80px;} /\* ie6 fixed \*/  
\*+html #wrapper { width: 60px;} /\* ie7 fixed, 注意顺序 \*/  
}  
</stylegt;  
注意:  
\*+html 对IE7的HACK 必须保证HTML顶部有如下声明：  
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"　"" style="color: rgb(0, 136, 204);">[网页链接](http://www.w3.org/TR/html4/loose.dtd)  
二、万能 float 闭合(非常重要!)  
关于 clear float 的原理可参见 [How To Clear Floats Without Structural Markup]  
将以下代码加入Global CSS 中,给需要闭合的div加上 ><stylegt;  
/\* Clear Fix \*/  
.clearfix:after  
{  
content:".";  
display:block;  
height:0;  
clear:both;  
visibility:hidden;  
}  
.clearfix  
{  
display:inline-block;  
}  
/\* Hide from IE Mac \*/  
.clearfix {display:block;}  
/\* End hide from IE Mac \*/  
/\* end of clearfix \*/  
</stylegt;  
三、其他兼容技巧(再次啰嗦)  
1, FF下给 div 设置 padding 后会导致 width 和 height 增加, 但IE不会.(可用!important解决)  
2, 居中问题.  
1).垂直居中.将 line-height 设置为 当前 div 相同的高度, 再通过 vertical-align: middle.( 注意内容不要换行.)  
2).水平居中. margin: 0 auto;(当然不是万能)  
3, 若需给 a 标签内内容加上 样式, 需要设置 display: block;(常见于导航标签)  
4, FF 和 IE 对 BOX 理解的差异导致相差 2px 的还有设为 float的div在ie下 margin加倍等问题.  
5, ul 标签在 FF 下面默认有 list-style 和 padding . 最好事先声明, 以避免不必要的麻烦. (常见于导航标签和内容列表)  
6, 作为外部 wrapper 的 div 不要定死高度, 最好还加上 overflow: hidden.以达到高度自适应.  
7, 关于手形光标. cursor: pointer. 而hand 只适用于 IE.  
1 针对firefox ie6 ie7的css样式  
现在大部分都是用!important来hack，对于ie6和firefox测试可以正常显示，  
但是ie7对!important可以正确解释，会导致页面没按要求显示！找到一个针  
对IE7不错的hack方式就是使用“\*+html”，现在用IE7浏览一下，应该没有问题了。  
现在写一个CSS可以这样：  
#1 { color: #333; } /\* Moz \*/  
\* html #1 { color: #666; } /\* IE6 \*/  
\*+html #1 { color: #999; } /\* IE7 \*/  
那么在firefox下字体颜色显示为#333，IE6下字体颜色显示为#666，IE7下字体颜色显示为#999。  
2 css布局中的居中问题  
主要的样式定义如下：  
body {TEXT-ALIGN: center;}  
#center { MARGIN-RIGHT: auto; MARGIN-LEFT: auto; }  
说明：  
首先在父级元素定义TEXT-ALIGN: center;这个的意思就是在父级元素内的内容居中；对于IE这样设定就已经可以了。  
但在mozilla中不能居中。解决办法就是在子元素定义时候设定时再加上“MARGIN-RIGHT: auto;MARGIN-LEFT: auto; ”  
需要说明的是，如果你想用这个方法使整个页面要居中，建议不要套在一个DIV里，你可以依次拆出多个div，  
只要在每个拆出的div里定义MARGIN-RIGHT: auto;MARGIN-LEFT: auto; 就可以了。  
3 盒模型不同解释.  
#box{ width:600px; //for ie6.0- w\idth:500px; //for ff+ie6.0}  
#box{ width:600px!important //for ff width:600px; //for ff+ie6.0 width /\*\*/:500px; //for ie6.0-}  
4 浮动ie产生的双倍距离  
#box{ float:left; width:100px; margin:0 0 0 100px; //这种情况之下IE会产生200px的距离 display:inline; //使浮动忽略}  
这里细说一下block,inline两个元素,Block元素的特点是:总是在新行上开始,高度,宽度,行高,边距都可以控制(块元素);Inline元素的特点是:和其他元素在同一行上,...不可控制(内嵌元素);  
#box{ display:block; //可以为内嵌元素模拟为块元素 display:inline; //实现同一行排列的的效果 diplay:table;  
5 IE与宽度和高度的问题  
IE不认得min-这个定义，但实际上它把正常的width和height当作有min的情况来使。这样问题就大了，如果只用宽度和高度，  
正常的浏览器里这两个值就不会变，如果只用min-width和min-height的话，IE下面根本等于没有设置宽度和高度。  
比如要设置背景图片，这个宽度是比较重要的。要解决这个问题，可以这样：  
#box{ width: 80px; height: 35px;}html>body #box{ width: auto; height: auto; min-width: 80px; min-height: 35px;}  
6 页面的最小宽度  
min-width是个非常方便的CSS命令，它可以指定元素最小也不能小于某个宽度，这样就能保证排版一直正确。但IE不认得这个，  
而它实际上把width当做最小宽度来使。为了让这一命令在IE上也能用，可以把一个<div> 放到 <body> 标签下，然后为div指定一个类：  
然后CSS这样设计：  
#container{ min-width: 600px; width:expression(document.body.clientWidth < 600? "600px": "auto" );}  
第一个min-width是正常的；但第2行的width使用了Javascript，这只有IE才认得，这也会让你的HTML文档不太正规。它实际上通过Javascript的判断来实现最小宽度。  
7 清除浮动  
.hackbox{ display:table; //将对象作为块元素级的表格显示}或者.hackbox{ clear:both;}  
或者加入:after（伪对象）,设置在对象后发生的内容，通常和content配合使用，IE不支持此伪对象，非Ie 浏览器支持，  
所以并不影响到IE/WIN浏览器。这种的最麻烦的......#box:after{ content: "."; display: block; height: 0; clear: both; visibility: hidden;}  
8 DIV浮动IE文本产生3象素的bug  
左边对象浮动，右边采用外补丁的左边距来定位，右边对象内的文本会离左边有3px的间距.  
#box{ float:left; width:800px;}#left{ float:left; width:50%;}#right{ width:50%;}\*html #left{ margin-right:-3px; //这句是关键}  
HTML代码<div id="box"> <div id="left"></div> <div id="right"></div></div>  
9 属性选择器(这个不能算是兼容,是隐藏css的一个bug)  
p[id]{}div[id]{}  
这个对于IE6.0和IE6.0以下的版本都隐藏,FF和OPera作用  
属性选择器和子选择器还是有区别的,子选择器的范围从形式来说缩小了,属性选择器的范围比较大,如p[id]中,所有p标签中有id的都是同样式的.  
10 IE捉迷藏的问题  
当div应用复杂的时候每个栏中又有一些链接，DIV等这个时候容易发生捉迷藏的问题。  
有些内容显示不出来，当鼠标选择这个区域是发现内容确实在页面。  
解决办法：对#layout使用line-height属性 或者给#layout使用固定高和宽。页面结构尽量简单。  
11 高度不适应  
高度不适应是当内层对象的高度发生变化时外层高度不能自动进行调节，特别是当内层对象使用  
margin 或paddign 时。  
例：  
<div id="box">  
p对象中的内容   
</div>  
CSS：#box {background-color:#eee; }  
#box p {margin-top: 20px;margin-bottom: 20px; text-align:center; }  
解决方法：在P对象上下各加2个空的div对象CSS代码：.1{height:0px;overflow:hidden;}或者为DIV加上border属性。

**99%的人都理解错了HTTP中GET与POST的区别**

GET和POST是HTTP请求的两种基本方法，要说它们的区别，接触过WEB开发的人都能说出一二。

最直观的区别就是GET把参数包含在URL中，POST通过request body传递参数。

你可能自己写过无数个GET和POST请求，或者已经看过很多权威网站总结出的他们的区别，你非常清楚知道什么时候该用什么。

当你在面试中被问到这个问题，你的内心充满了自信和喜悦。

你轻轻松松的给出了一个“标准答案”：

GET在浏览器回退时是无害的，而POST会再次提交请求;

GET产生的URL地址可以被Bookmark，而POST不可以;

GET请求会被浏览器主动cache，而POST不会，除非手动设置;

GET请求只能进行url编码，而POST支持多种编码方式;

GET请求参数会被完整保留在浏览器历史记录里，而POST中的参数不会被保留;

GET请求在URL中传送的参数是有长度限制的，而POST么有;

对参数的数据类型，GET只接受ASCII字符，而POST没有限制;

GET比POST更不安全，因为参数直接暴露在URL上，所以不能用来传递敏感信息。 GET参数通过URL传递，POST放在Request body中。

“很遗憾，这不是我们要的回答!”

请告诉我真相。。。

如果我告诉你GET和POST本质上没有区别你信吗?

让我们扒下GET和POST的外衣，坦诚相见吧!

GET和POST是什么?HTTP协议中的两种发送请求的方法。

HTTP是什么?HTTP是基于TCP/IP的关于数据如何在万维网中如何通信的协议。

HTTP的底层是TCP/IP。所以GET和POST的底层也是TCP/IP，也就是说，GET/POST都是TCP链接。GET和POST能做的事情是一样一样的。你要给GET加上request body，给POST带上url参数，技术上是完全行的通的。

那么，“标准答案”里的那些区别是怎么回事?

在我大万维网世界中，TCP就像汽车，我们用TCP来运输数据，它很可靠，从来不会发生丢件少件的现象。但是如果路上跑的全是看起来一模一样的汽车，那这个世界看起来是一团混乱，送急件的汽车可能被前面满载货物的汽车拦堵在路上，整个交通系统一定会瘫痪。为了避免这种情况发生，交通规则HTTP诞生了。HTTP给汽车运输设定了好几个服务类别，有GET, POST, PUT, DELETE等等，HTTP规定，当执行GET请求的时候，要给汽车贴上GET的标签(设置method为GET)，而且要求把传送的数据放在车顶上(url中)以方便记录。如果是POST请求，就要在车上贴上POST的标签，并把货物放在车厢里。当然，你也可以在GET的时候往车厢内偷偷藏点货物，但是这是很不光彩;也可以在POST的时候在车顶上也放一些数据，让人觉得傻乎乎的。HTTP只是个行为准则，而TCP才是GET和POST怎么实现的基本。

但是，我们只看到HTTP对GET和POST参数的传送渠道(url还是requrest body)提出了要求。“标准答案”里关于参数大小的限制又是从哪来的呢?

在我大万维网世界中，还有另一个重要的角色：运输公司。不同的浏览器(发起http请求)和服务器(接受http请求)就是不同的运输公司。 虽然理论上，你可以在车顶上无限的堆货物(url中无限加参数)。但是运输公司可不傻，装货和卸货也是有很大成本的，他们会限制单次运输量来控制风险，数据量太大对浏览器和服务器都是很大负担。业界不成文的规定是，(大多数)浏览器通常都会限制url长度在2K个字节，而(大多数)服务器最多处理64K大小的url。超过的部分，恕不处理。如果你用GET服务，在request body偷偷藏了数据，不同服务器的处理方式也是不同的，有些服务器会帮你卸货，读出数据，有些服务器直接忽略，所以，虽然GET可以带request body，也不能保证一定能被接收到哦。

好了，现在你知道，GET和POST本质上就是TCP链接，并无差别。但是由于HTTP的规定和浏览器/服务器的限制，导致他们在应用过程中体现出一些不同。

你以为本文就这么结束了?

我们的大BOSS还等着出场呢。。。

这位BOSS有多神秘?当你试图在网上找“GET和POST的区别”的时候，那些你会看到的搜索结果里，从没有提到他。他究竟是什么呢。。。

GET和POST还有一个重大区别，简单的说：

GET产生一个TCP数据包;POST产生两个TCP数据包。

长的说：

对于GET方式的请求，浏览器会把http header和data一并发送出去，服务器响应200(返回数据);

而对于POST，浏览器先发送header，服务器响应100 continue，浏览器再发送data，服务器响应200 ok(返回数据)。

也就是说，GET只需要汽车跑一趟就把货送到了，而POST得跑两趟，第一趟，先去和服务器打个招呼“嗨，我等下要送一批货来，你们打开门迎接我”，然后再回头把货送过去。

因为POST需要两步，时间上消耗的要多一点，看起来GET比POST更有效。因此Yahoo团队有推荐用GET替换POST来优化网站性能。但这是一个坑!跳入需谨慎。为什么?

1. GET与POST都有自己的语义，不能随便混用。

2. 据研究，在网络环境好的情况下，发一次包的时间和发两次包的时间差别基本可以无视。而在网络环境差的情况下，两次包的TCP在验证数据包完整性上，有非常大的优点。

3. 并不是所有浏览器都会在POST中发送两次包，Firefox就只发送一次。

# 正则表达式

正则表达式是对字符串的一种特殊操作。

验证信息：15638252897 11位 1开头

邮箱验证：@符号

## 一：创建正则表达式对象

var box=new RegExp(string,‘i、g、m’);（i，m，g必须加引号）

//new RegExp()对象里面可以传递两个参数

//第一个：字符串，即在该字符串前后加两个斜杠。第二个：可选模式修饰符

String：将来要匹配的字符串。

也可以直接声明 var box=/rocky/ig

创建正则表达式中的可选模式修饰符：

|  |  |
| --- | --- |
| i | 忽略大小写 |
| g | 全局匹配 |
| m | 多行匹配 |

## 二：使用正则对象提供的方法进行测试字符串

RegExp对象包含两种方法：.test（）与.exec（），功能基本相似，用于测试字符串是否匹配。

.test（）方法主要在字符串总共查找是否包含指定的字符串进行返回布尔值。

.exec（）方法是执行查找，返回的是字符串数组.

|  |  |  |
| --- | --- | --- |
| 方法 | 功能 | 结果 |
| Test（） | 查找是否存在匹配的字符串 | 返回布尔值 |
| Exec（） | 搜索匹配的字符串 | 有的话返回字符串组成的数组，没有话返回null |

## 三：使用字符串、数组的方法操作正则表达式

有4个方法：

|  |  |
| --- | --- |
| 方法 | 含义 |
| Match（obj） | 用字符串匹配obj，返回obj中匹配到的字符串串数组，或null |
| Replace（obj，str） | 用后面的字符串替代前面的字符串，有多少代替多少 |
| Search（） | 返回匹配字符串的位置 |
| Split（） | 返回obj拆分的数组 |

## RegExp对象的静态属性

静态属性：指固定不变的属性

|  |  |  |  |
| --- | --- | --- | --- |
| 属性 | 短名 | 含义 | 说明 |
| input | $ | 返回当前匹配的字符串 | Opera不支持 |
| lastMatch | $& | 返回最后一个匹配的字符串 | Opera不支持 |
| lastParen | $+ | 返回最后一个圆括号内的匹配字符串 | Opera不支持 |
| leftContext | $` | 返回最后一个匹配字符串之前的字符串 |  |
| multiline | $\* | 用于指定是否说有表达式 | IE不支持 |
| RightContext | $’ | 最后一次匹配字符串之后的字符串 |  |

## RegExp对象实例属性（没什么卵用，IE也不支持）

|  |  |
| --- | --- |
| global | 返回Boolean表示g是否已设置（是否全局） |
| ignoreCase | 返回Boolean表示i是否已设置（是否忽略大小写） |
| lastIndex | 返回整数代表下次将从哪个字符位置开始 |
| multiline | 返回Boolean，表示m是否设置 |
| source | 返回正则表达式的源字符 |

## 获取控制字符

字符类：匹配单个字符和数字

|  |  |
| --- | --- |
| 元字符 | 匹配情况 |
| . | 匹配除换行符意外的任意单个字符。 |
| [a-z0-9] | 匹配括号中的字符集中的任意字符 |
| [^a-z0-9] | 匹配任意不在括号中的字符 |
| \d | 匹配数字 |
| \D | 匹配非数子 |
| \w | 匹配字母，数字以及下划线 |
| \W | 匹配非字母、数字以及下划线 |

字符类：空白字符

|  |  |
| --- | --- |
| 元字符 | 匹配情况 |
| \0 | 匹配null |
| \b | 空格 |
| \f | 进制 |
| \n | 匹配换行符 |
| \r | 匹配回车 |
| \t | 匹配制表符 |
| \s | 空白字符，空格，制表符，换行字符 |
| \S | 非空白字符，空格，制表符，换行字符 |

字符类：锚字符

|  |  |
| --- | --- |
| ^X | 行首匹配 |
| X$ | 行尾匹配 |
| \A | 匹配字符船开始 |
| \b | 单词边界 |
| \B | 非单词边际 |
| \G | 当前搜索位置 |
| \Z | 匹配字符串结束位置 |
| \z | 匹配字符串结束处 |

字符类：替代字符

|  |  |
| --- | --- |
| 元字符、元符号 | 匹配情况 |
| This|where|logo | 匹配this或者where或logo中任意一个 |

字符类：重复字符

|  |  |
| --- | --- |
| 元字符 | 匹配情况 |
| X？ | 匹配0或1个字符 |
| X\* | 匹配0或任意多个字符 |
| X+ | 至少匹配一个 |
| （XYZ）+ | 匹配至少一个XYZ |
| X{m，n} X{m,} X{,n} | 匹配至少，m个，最多n个X |

字符类：记录字符

|  |  |
| --- | --- |
| 元字符 | 匹配情况 |
| （string） | 用于反向引用的组 |
| \1或$1 | 匹配第1个分组中的内容 |
| \2或$2 | 匹配第2个分组中的内容 |
| \n或$n | 匹配第n个分组中的内容 |

四：贪婪模式与非贪婪模式

|  |  |
| --- | --- |
| 贪婪 | 惰性模式 |
| + | +？ |
| ？ | ？？ |
| \* | \*？ |
| {n} | {n}? |
| {n,} | {n,}? |
| {n,m} | {n,m}? |

**什么是正则表达式的贪婪与非贪婪匹配**

如：var str="abcaxc";

　　　　var p="/ab\*c/";

**贪婪匹配:**正则表达式一般趋向于最大长度匹配，也就是所谓的贪婪匹配。如上面使用模式p匹配字符串str，结果就是匹配到：**abcaxc**(ab\*c)。

**非贪婪匹配**：就是匹配到结果就好，最少的匹配字符。如上面使用模式p匹配字符串str，结果就是匹配到：**abc**(ab\*c)。

下面代码关闭了贪婪模式，只替换了第一个

// var pattern=/[a-z]+/;// 最少一个字符

var pattern=/[a-z]+?/;// 最少一个字符

var str='abcdefghijklmnopqrstuvwxyz';

var resoult=str.replace(pattern,'xxx');

document.write(resoult);

## 正则表达式举例：

**一、校验数字的表达式**

1 数字：^[0-9]\*$

2 n位的数字：^\d{n}$

3 至少n位的数字：^\d{n,}$

4 m-n位的数字：^\d{m,n}$

5 零和非零开头的数字：^(0|[1-9][0-9]\*)$

6 非零开头的最多带两位小数的数字：^([1-9][0-9]\*)+(.[0-9]{1,2})?$

7 带1-2位小数的正数或负数：^(\-)?\d+(\.\d{1,2})?$

8 正数、负数、和小数：^(\-|\+)?\d+(\.\d+)?$

9 有两位小数的正实数：^[0-9]+(.[0-9]{2})?$

10 有1~3位小数的正实数：^[0-9]+(.[0-9]{1,3})?$

11 非零的正整数：^[1-9]\d\*$ 或 ^([1-9][0-9]\*){1,3}$ 或 ^\+?[1-9][0-9]\*$

12 非零的负整数：^\-[1-9][]0-9"\*$ 或 ^-[1-9]\d\*$

13 非负整数：^\d+$ 或 ^[1-9]\d\*|0$

14 非正整数：^-[1-9]\d\*|0$ 或 ^((-\d+)|(0+))$

15 非负浮点数：^\d+(\.\d+)?$ 或 ^[1-9]\d\*\.\d\*|0\.\d\*[1-9]\d\*|0?\.0+|0$

16 非正浮点数：^((-\d+(\.\d+)?)|(0+(\.0+)?))$ 或 ^(-([1-9]\d\*\.\d\*|0\.\d\*[1-9]\d\*))|0?\.0+|0$

17 正浮点数：^[1-9]\d\*\.\d\*|0\.\d\*[1-9]\d\*$ 或 ^(([0-9]+\.[0-9]\*[1-9][0-9]\*)|([0-9]\*[1-9][0-9]\*\.[0-9]+)|([0-9]\*[1-9][0-9]\*))$

18 负浮点数：^-([1-9]\d\*\.\d\*|0\.\d\*[1-9]\d\*)$ 或 ^(-(([0-9]+\.[0-9]\*[1-9][0-9]\*)|([0-9]\*[1-9][0-9]\*\.[0-9]+)|([0-9]\*[1-9][0-9]\*)))$

19 浮点数：^(-?\d+)(\.\d+)?$ 或 ^-?([1-9]\d\*\.\d\*|0\.\d\*[1-9]\d\*|0?\.0+|0)$

**二、校验字符的表达式**

1 汉字：^[\u4e00-\u9fa5]{0,}$或者 /.\*[\u4e00-\u9fa5]+.\*$/

2 英文和数字：^[A-Za-z0-9]+$ 或 ^[A-Za-z0-9]{4,40}$

3 长度为3-20的所有字符：^.{3,20}$

4 由26个英文字母组成的字符串：^[A-Za-z]+$

5 由26个大写英文字母组成的字符串：^[A-Z]+$

6 由26个小写英文字母组成的字符串：^[a-z]+$

7 由数字和26个英文字母组成的字符串：^[A-Za-z0-9]+$

8 由数字、26个英文字母或者下划线组成的字符串：^\w+$ 或 ^\w{3,20}$

9 中文、英文、数字包括下划线：^[\u4E00-\u9FA5A-Za-z0-9\_]+$

10 中文、英文、数字但不包括下划线等符号：^[\u4E00-\u9FA5A-Za-z0-9]+$ 或 ^[\u4E00-\u9FA5A-Za-z0-9]{2,20}$

11 可以输入含有^%&',;=?$\"等字符：[^%&',;=?$\x22]+

12 禁止输入含有~的字符：[^~\x22]+

13 整数或者小数：^[0-9]+\.{0,1}[0-9]{0,2}$

**三、特殊需求表达式**

文本输入框禁用或屏蔽回车：onkeypress="if(event.keyCode==13) return false;"

1 Email地址：^\w+([-+.]\w+)\*@\w+([-.]\w+)\*\.\w+([-.]\w+)\*$

2 域名：

[a-zA-Z0-9][-a-zA-Z0-9]{0,62}(/.[a-zA-Z0-9][-a-zA-Z0-9]{0,62})+/.?

3 InternetURL：[a-zA-z]+://[^\s]\* 或 ^http://([\w-]+\.)+[\w-]+(/[\w-./?%&=]\*)?$

4 手机号码：

^(13[0-9]|14[5|7]|15[0|1|2|3|5|6|7|8|9]|18[0|1|2|3|5|6|7|8|9])\d{8}$

5 电话号码("XXX-XXXXXXX"、"XXXX-XXXXXXXX"、"XXX-XXXXXXX"、"XXX-XXXXXXXX"、"XXXXXXX"和"XXXXXXXX)：^(\(\d{3,4}-)|\d{3.4}-)?\d{7,8}$

6 国内电话号码(0511-4405222、021-87888822)：\d{3}-\d{8}|\d{4}-\d{7}

7 身份证号：

15或18位身份证：^\d{15}|\d{18}$

15位身份证：

^[1-9]\d{7}((0\d)|(1[0-2]))(([0|1|2]\d)|3[0-1])\d{3}$

18位身份证：

^[1-9]\d{5}[1-9]\d{3}((0\d)|(1[0-2]))(([0|1|2]\d)|3[0-1])\d{4}$

8 短身份证号码(数字、字母x结尾)：^([0-9]){7,18}(x|X)?$

或 ^\d{8,18}|[0-9x]{8,18}|[0-9X]{8,18}?$

9 帐号是否合法(字母开头，允许5-16字节，允许字母数字下划线)：^[a-zA-Z][a-zA-Z0-9\_]{4,15}$

10 密码(以字母开头，长度在6~18之间，只能包含字母、数字和下划线)：^[a-zA-Z]\w{5,17}$

11 强密码(必须包含大小写字母和数字的组合，不能使用特殊字符，长度在8-10之间)：^(?=.\*\d)(?=.\*[a-z])(?=.\*[A-Z]).{8,10}$

12 日期格式：^\d{4}-\d{1,2}-\d{1,2}

13 一年的12个月(01～09和1～12)：^(0?[1-9]|1[0-2])$

14 一个月的31天(01～09和1～31)：^((0?[1-9])|((1|2)[0-9])|30|31)$

15 钱的输入格式：

15-1.有四种钱的表示形式我们可以接受:"10000.00" 和 "10,000.00", 和没有 "分" 的 "10000" 和 "10,000"：^[1-9][0-9]\*$

15-2.这表示任意一个不以0开头的数字,但是,这也意味着一个字符"0"不通过,所以我们采用下面的形式：^(0|[1-9][0-9]\*)$

15-3.一个0或者一个不以0开头的数字.我们还可以允许开头有一个负号：^(0|-?[1-9][0-9]\*)$

15-4.这表示一个0或者一个可能为负的开头不为0的数字.让用户以0开头好了.把负号的也去掉,因为钱总不能是负的吧.下面我们要加的是说明可能的小数部分：^[0-9]+(.[0-9]+)?$

15-5.必须说明的是,小数点后面至少应该有1位数,所以"10."是不通过的,但是 "10" 和 "10.2" 是通过的：^[0-9]+(.[0-9]{2})?$

15-6.这样我们规定小数点后面必须有两位,如果你认为太苛刻了,可以这样：^[0-9]+(.[0-9]{1,2})?$

15-7.这样就允许用户只写一位小数.下面我们该考虑数字中的逗号了,我们可以这样：^[0-9]{1,3}(,[0-9]{3})\*(.[0-9]{1,2})?$

15-8. 1到3个数字,后面跟着任意个 逗号+3个数字,逗号成为可选,而不是必须：^([0-9]+|[0-9]{1,3}(,[0-9]{3})\*)(.[0-9]{1,2})?$

备注：这就是最终结果了,别忘了"+"可以用"\*"替代如果你觉得空字符串也可以接受的话(奇怪,为什么?)最后,别忘了在用函数时去掉去掉那个反斜杠,一般的错误都在这里

16 xml文件：^([a-zA-Z]+-?)+[a-zA-Z0-9]+\\.[x|X][m|M][l|L]$

17 中文字符的正则表达式：[\u4e00-\u9fa5]

18 双字节字符：[^\x00-\xff] (包括汉字在内，可以用来计算字符串的长度(一个双字节字符长度计2，ASCII字符计1))

19 空白行的正则表达式：\n\s\*\r (可以用来删除空白行)

20 HTML标记的正则表达式：<(\S\*?)[^>]\*>.\*?</\1>|<.\*? /> (网上流传的版本太糟糕，上面这个也仅仅能部分，对于复杂的嵌套标记依旧无能为力)

21 首尾空白字符的正则表达式：^\s\*|\s\*$或(^\s\*)|(\s\*$) (可以用来删除行首行尾的空白字符(包括空格、制表符、换页符等等)，非常有用的表达式)

22 腾讯QQ号：[1-9][0-9]{4,} (腾讯QQ号从10000开始)

23 中国邮政编码：[1-9]\d{5}(?!\d) (中国邮政编码为6位数字)

24 IP地址：\d+\.\d+\.\d+\.\d+ (提取IP地址时有用)

复习：HTML的年轮：

1993年互联网工程小组成员发布了一个HTML草案（html1.0，很不完善）

1994年：html2版本（相对完善）发布

1996年：CSS1+Javascript问世，作者是布兰登.艾奇(前端工程师的祖师爷)

1997年：HTML4发布

1998年：W3C（世界万维网组织）发布了XML-1.0（一种数据格式，一种扩展性标记语言）70%以上的数据传输都是用的XML（JSON数据的前身）结构化文档和数据结构的通用性，带来了一场风暴。

2000年XHTML-1.0问世作为XML被重新设计的html

2005年：ajax诞生（下一代：fetch要被W3C要被定为标准）

2004 html5Application（外部应用）1.0

2007年W3C同意采用html5作为标准，并且成立了工作团队

2014年 HTML5.0发布了！！！

广义概念：新一代开发web富客户端应用程序整体解决方案：

HTML5≈HTML（提升）+CSS（提升）+Javascript API（提升）

Html新标签、新属性

data-，dragable、**contentedable:true(可在线编辑文本)、role（盲人模式）数据结构标签CSS：transition：4个值：**

**1Transition-duration（总共时间）**

**2Transition-property：过度属性，逗号隔开；（all，width，background）**

**3Transition-timming-function：过渡函数，默认值是（ease，easein，easeout）**

**Ease：开始和结束慢**

**Linear：匀速**

**Ease-in：开始慢**

**Ease-out：结束慢**

**Ease-in-out：夸张版的ease**

**4Transition-delay（延时播放，默认值是0）：2s**

**Step-start：直接处于结束处；**

**Step-end：口位于开始处的经过时间间隔后结束**

**怎么用：以上两种可直接用，或者是step（1，start）/step(1,end)**

**Cubic-bazier(0,0,1,1),贝赛尔曲线**

# 阿里高清布局方案

代码原理

这是阿里团队的高清方案布局代码，所谓高清方案就是根据设备屏幕的DPR（设备像素比，又称DPPX，比如dpr=2时，表示1个CSS像素由4个物理像素点组成） 动态设置 html 的font-size, 同时根据设备DPR调整页面的缩放值，进而达到高清效果。

有何优势

引用简单，布局简便 根据设备屏幕的DPR,自动设置最合适的高清缩放。 保证了不同设备下视觉体验的一致性。（老方案是，屏幕越大元素越大；此方案是，屏幕越大，看的越多） 有效解决移动端真实1px问题（这里的1px 是设备屏幕上的物理像素）

如何使用

重要的事情说三遍！ 绝不是每个地方都要用rem，rem只适用于固定尺寸！ 绝不是每个地方都要用rem，rem只适用于固定尺寸！ 绝不是每个地方都要用rem，rem只适用于固定尺寸！ 在相当数量的布局情境中（比如底部导航元素平分屏幕宽，大尺寸元素），你必须使用百分比或者flex才能完美布局！ 此方案也是默认 1rem=100px，所以你布局的时候，完全可以按照设计师给你的效果图写各种尺寸啦。 比如你在效果图上量取的某个按钮元素长 55px, 宽37px ，那你直接可以这样写样式：

.myBtn {

width: 0.55rem;

height: 0.37rem;

}

示例demo

<!DOCTYPE html><html lang="en"><head>

<meta charset="UTF-8">

<title>ceshi</title><!-- 阿里高清方案 -->

<script>

!function(e){function t(a){if(i[a])return i[a].exports;var n=i[a]={exports:{},id:a,loaded:!1};return e[a].call(n.exports,n,n.exports,t),n.loaded=!0,n.exports}var i={};return t.m=e,t.c=i,t.p="",t(0)}([function(e,t){"use strict";Object.defineProperty(t,"\_\_esModule",{value:!0});var i=window;t["default"]=i.flex=function(e,t){var a=e||100,n=t||1,r=i.document,o=navigator.userAgent,d=o.match(/Android[\S\s]+AppleWebkit\/(\d{3})/i),l=o.match(/U3\/((\d+|\.){5,})/i),c=l&&parseInt(l[1].split(".").join(""),10)>=80,p=navigator.appVersion.match(/(iphone|ipad|ipod)/gi),s=i.devicePixelRatio||1;p||d&&d[1]>534||c||(s=1);

Var u=1/s,m=r.querySelector('meta[name="viewport"]');m||(m=r.createElement("meta"),m.setAttribute("name","viewport"),r.head.appendChild(m)),m.setAttribute("content","width=device-width,user-scalable=no,initial-scale="+u+",maximum-scale="+u+",minimum-scale="+u),r.documentElement.style.fontSize=a/2\*s\*n+"px"},e.exports=t["default"]}]);

flex(100, 1);</script>

<!-- 字形图标 -->

<script src="https://use.fontawesome.com/f47f4563cb.js"></script><style>

\* {

box-sizing: border-box;

/\* 在X5新内核Blink中，在排版页面的时候，会主动对字体进行放大，会检测页面中的主字体，当某一块字体在我们的判定规则中，认为字号较小，并且是页面中的主要字体，就会采取主动放大的操作。然而这不是我们想要的，可以采取给最大高度解决 \*/

max-height: 100000px;

}

\*:before, \*:after {

box-sizing: border-box;

max-height: 100000px;

}

\*, \*:before, \*:after {

-webkit-tap-highlight-color: rgba(0, 0, 0, 0);

}

html, body, div, span, object, iframe, h1, h2, h3, h4, h5, h6, p, blockquote, pre, a, abbr, address, cite, code, del, dfn, em, img, ins, kbd, q, samp, small, strong, sub, sup, var, b, i, dl, dt, dd, ol, ul, li, fieldset, form, label, legend, table, caption, tbody, tfoot, thead, tr, th,textarea, td {

border: 0 none;

font-size: inherit;

color: inherit;

margin: 0;

padding: 0;

vertical-align: baseline;

}

h1, h2, h3, h4, h5, h6 {

font-weight: normal;

}

em, strong {

font-style: normal;

}

ul, ol, li {

list-style: none;

}

body {

font-family: "Helvetica Neue",Helvetica,"PingFang SC","Hiragino Sans GB","Microsoft YaHei","\5FAE\8F6F\96C5\9ED1",Arial,sans-serif;

line-height: 1.5;

color: #333;

background-color: #f2f2f2;

font-size: 0.24rem;

}

a {

text-decoration: none;

}.box {

position: relative;

max-width: 10rem;

margin: 0 auto;

}.navRoot {

position: fixed;

z-index: 50;

bottom: 0;

left:0;

width: 100%;

height: 1rem;

display: -webkit-box;

display: -ms-flexbox;

display: flex;

border-top: 0.01rem solid #ccc;

background:#f2f2f2;

}

.navLink {

font-size: 0.42rem;

-webkit-box-flex: 1;

-ms-flex: auto;

flex: auto;

display: -webkit-box;

display: -ms-flexbox;

display: flex;

-webkit-box-orient: vertical;

-webkit-box-direction: normal;

-ms-flex-direction: column;

flex-direction: column;

-ms-flex-wrap: wrap;

flex-wrap: wrap;

-webkit-box-pack: center;

-ms-flex-pack: center;

justify-content: center;

-webkit-box-align: center;

-ms-flex-align: center;

align-items: center;

line-height: 1;

color: #666;

}.navLink.active {

color: #1abc9c

}.navLinkIco {

display: block;

margin-bottom: 0.1rem;

}

.navLinkText {

display: block;

line-height: 1;

font-size: 0.24rem;

}

.img-responsive {

width: 100%;

}</style></head><body>

<div class="box">

<img src="http://temp.im/640x260/444/fff" class="img-responsive">

<ul style="font-size:0.3rem; padding: 0.2rem">

<li style="font-size:0.4rem"><a href="http://www.jianshu.com/p/985d26b40199">rem布局（进阶版）的一些说明</a></li>

<li>1，大家先把这个页面在不同设备模式下浏览下，感受下。</li>

<li>2，一般来说我们的手机端效果图是640px或750px。</li>

<li>3，对于上图这类需要等屏宽的图片，只需让其宽度设为 100% 即可。</li>

<li>4，该布局方案要求凡是涉及尺寸的，单位都是rem，比如，height width margin paddint top border-radius 等等。你只需要按照效果图上标注的尺寸布局即可。</li>

</ul>

<div class="navRoot">

<a href="#" class="navLink active">

<i class="fa fa-home navLinkIco" aria-hidden="true"></i>

<span class="navLinkText">首页</span>

</a>

<a href="#" class="navLink">

<i class="fa fa-home navLinkIco" aria-hidden="true"></i>

<span class="navLinkText">好店</span>

</a>

<a href="#" class="navLink">

<i class="fa fa-home navLinkIco" aria-hidden="true"></i>

<span class="navLinkText">简单</span>

</a>

<a href="#" class="navLink">

<i class="fa fa-home navLinkIco" aria-hidden="true"></i>

<span class="navLinkText">复杂</span>

</a>

<a href="#" class="navLink">

<i class="fa fa-home navLinkIco" aria-hidden="true"></i>

<span class="navLinkText">异步</span>

</a>

</div>

</div>

</body>

</html>

# rem布局

<!--使用rem为单位--><!DOCTYPE html>

<html lang="en"><head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">

<meta

name="viewport" content="width=device-width,minimum-scale=1.0,maximum-scale=1.0,user-scalable=no"> <script>

(function (doc, win) {

var docEl = doc.documentElement,

resizeEvt = 'orientationchange' in window ? 'orientationchange' : 'resize',

recalc = function () {

var clientWidth = docEl.clientWidth;

if (!clientWidth) return;

if(clientWidth>=640){

docEl.style.fontSize = '100px';

}else{

docEl.style.fontSize = 100 \* (clientWidth / 640) + 'px';

}

};

if (!doc.addEventListener) return;

win.addEventListener(resizeEvt, recalc, false);

doc.addEventListener('DOMContentLoaded', recalc, false);

})(document, window);</script>

/\*你引进的资源\*/<title>标题</title> </head><body>

/\*你的代码\*/</body></html>

# Bootstrap

网址：<http://www.bootcss.com/>

CDN：百度静态资源、bootcdn

前端构建工具：webpack、gulp、grunt

框架（类）：jquery.js、zepto.js、jqueryMobile.js

CSS预编译：less、sass、stylic

Bootstrap环境安装：

框架是个半成品，Bootstrap-7成品

团队协作开发工具：svn，git（github）

Bootstrap兼容到IE8

<metahttp-equiv="X-UA-Compatible"content="IE=edge,chrome=1">

谷歌、IE外挂

这句话表明了以谷歌chrome浏览器引擎解析页面，或者以IE最新版本浏览页面。



使得p标签拥有h1的样式

1. 排版：

h1-h6.

2全局属性：

Font-sise：14px；

line-height：1.4；

<mark>行内元素有特殊背景色

<del> <s>删除线<ins>下划线<strong>加粗<em><var>斜体<code><kbd>按键背景<pre>区域背景

3列表：

行内显示： ul class=list-inline

水平显示：dl class=horizontal 默认垂直

4表格：

基本表格样式：table

条纹间隔：.table-striped

代边框：.table-border

鼠标悬停：.table-hover

紧缩：.table-condensed

1. 表单

Form表单：Div/form上设置 class=“form-group”

控件：为表单加上class=form-control

使form表单内的控件水平一行显示：

rol=“form”;class="form-horizontal "

附加提示部分设置span class="input-group-addon"

为了实现最佳间距多处使用，为表单添加外围容器div class=form-group

为了让表单lable与表单控件都能水平显示，可为表单添加外围容器div class=form-horizontal

内联表单：.form-inline

表单额外添加图标：

需要给form-group 的div加上.hasfeedback

给图标标签加class="glyphicon glyphicon-ok form-control-feedback"

1. 按钮

基础样式：<a href=”javascript:;” class=”btn btn-default”></a>

按钮颜色：primary、danger、warning、info、success、default、disabled。在lable和控件上都添加disabled

按钮大小：btn-lg/sm/xs等

手机按钮显示：btn-block

1. 图片：

圆角：img-rounded

圆形图片：img-circle

标准图片设计：img-thumbnail（边框、圆角、padding）

图片的响应式：img-responsive

1. 网格：

规则1：必须要有父容器class="container" 或container-fluid

规则2：必须要有一个子容器包裹：div class=row

规则2：row下面的子元素不许都是col\*-\*

规则3：不能超过12份  
辅助类：

1. 文字颜色

.text-mued灰色.text-primary 主蓝 .text-success 浅绿

.text-indo 浅蓝.text-warning 黄色 .text-danger危险红

1. 背景

比上面少了个背景灰，其他都一样，格式为bg-\*。

1. 关闭按钮

<button class=”clase”><button>

<!--三角符号-->

<span class="caret"></span>向下三角

1. 浮动、块级居中

Pull-left、pull-right、center-block

1. 清除浮动

.clearfix放到浮动区块的前面

1. 显示和隐藏

.show .hidden

响应式工具

主要用来通过mediaquery来判断大、中、小设备，来显示隐藏

## Bootstrap网格

1:、什么是响应式布局？

流式布局：普通的浮动布局

自适应布局：百分比布局，

在做屏幕适配时，可能需要设计两套样式，根据浏览器设备的特征进行区域区别加载。

响应式布局：上面两种布局的结合

作用：响应式web设计师一个让用户通过各种尺寸的设备，浏览器网站获得良好的视觉体验方案。

1. CSS3实现响应式布局：mediaquery
2. bootstrap的布局使用

网格布局

Bootstrap 提供了一套响应式、移动设备优先的流式栅格系统，随着屏幕或视口（viewport）尺寸的增加，系统会自动分为最多12列。

3.1、创建容器

<div class="container">

<!--创建一个响应式行-->

<div class="row">

<div class="col-sm-4 col-xs-12">当屏幕为small的时候，本容器占4/12</div>

<div class="col-sm-4 col-xs-12">当屏幕为small的时候，本容器占4/12</div>

<div class="col-sm-4 col-sx-12">当屏幕为small的时候，本容器占4/12</div>

</div>

</div>

网格的核心就是12列的column

第一：行（row）必须包含在.continer(固定宽度)或.container-fluid(100%宽度)中。

第二：页面内容应当放置于column内，只有col-可以作为div.row的子元素

第三：当前屏幕下，所有的列数之和不应超过12.

3.2：区分的四种浏览器的宽度值

大屏：>=1200px col-lg 针对PC

中屏：>=992px col-md 针对笔记本

小屏：>=768px col-sm 针对平板paid

超小屏：<768px col-xs 针对手机（一般不用）

3.3 列偏移

使用 .col-md-offset-\* 类可以将列向右侧偏移。这些类实际是通过使用 \* 选择器为当前元素增加了左侧的边距（margin）。例如，.col-md-offset-4 类将 .col-md-4 元素向右侧偏移了4个列（column）的宽度。

## 组件

1. 图标组件

使用<i><span>，具体如下：

<i class=”glyphicon glyphicon-alert”>警告</i>

<span class=”glyphicon glyphicon-phone”></span> 手机图标标签

使用时给span/i标签定义大小

**2、下拉组件：**

**给父亲级div添加class=“**dropdown/dropup**”**

**再给子元素设置**class="dropdown-menu"

<ul class="dropdown-menu">

<li><a href="#">首页</a> </li>

<li><a href="#">新闻</a> </li>

<li><a href="#">关于</a> </li>

<li><a href="#">产品</a> </li>

</ul>

1. 输入框组件

1）再左/右侧添加文字：

<div class="input-group has-feedback">

<span class="input-group-addon">用户名：</span>

<input type="text" class="form-control" />

<span class="input-group-addon"><i class="glyphicon glyphicon-alert"></i> </span>

<span class="input-group-addon">用户名：</span>

<input type="text" class="form-control" />

<span class="glyphicon glyphicon-apple form-control-feedback"></span>

</div>

1. 设置表单尺寸

div class="input-group has-feedback input-group-lg"

左侧提示表单

1. 导航：

<ul class="nav nav-tabs navbar-default navbar-fixed-top">

<li > <a href="#">首页</a></li>

<li class="active"><a href="#">资讯</a></li>

<li class="active"><a href="#">产品</a></li>

<li class="active"><a href="#">新闻</a></li>

<li class="active"><a href="#">地图</a></li>

</ul>

带下拉菜单的导航

<ul class="nav nav-pills nav-stacked ">

<li > <a href="#">首页</a></li>

<li class="active"><a href="#">资讯</a></li>

<li class="active"><a href="#">产品</a></li>

<li class="active"><a href="#">新闻</a></li>

<li class="active"><a href="#">地图</a></li>

<li class="dropdown ">

<button class="btn btn-info" data-toggle="dropdown">123123</button>

<ul class="dropdown-menu">

<li>2131</li>

<li>123</li>

<li>12313</li>

</ul>

</li>

</ul>

完成基本导航

导航条的菜单：

设置折叠菜单：

Button class="navbar-toggle collapsed" data-toggle="collapse" data-target="#mynav"指向id名的区域

span class="glyphicon glyphicon-list"设置button的样式

设置隐藏折叠区域：

加div id=“mynav”将所有要折叠的区域包裹起来

在要折叠的区域加id=“mynav”.

导航条固定在顶部：

nav class="navbar navbar-default navbar-fixed-top"

导航条去圆角：navbar-static-top。

路径导航（面包屑导航）组件：

<ol class="breadcrumb">

<li> <a href="#">首页</a></li>

<li> <a href="#">框架</a></li>

<li class="active">react.js</li>

</ol>

默认分页：

<ul class="pagination">

<li class="disabled"> <a href="javascript:;">上一页</a></li>

<li> <a href="javascript:;">1</a></li>

<li> <a href="javascript:;">2</a></li>

<li> <a href="javascript:;">3</a></li>

<li> <a href="javascript:;">4</a></li>

<li> <a href="javascript:;">5</a></li>

<li class="next"> <a href="javascript:;">下一页</a></li>

<li></li>

</ul>  
7、标签：

<h3>javacsript<spanclass="label label-default">hot</span></h3>

1. 巨幕：

巨幕就是大的div广告区域，放一些网站标题，往往是同屏和圆角两种

<div class="jumbotron">

<div class="container">

<h1>我是前端工程师</h1>

<p>这是我的个人网站</p>

<p>我在4月等你</p>

<p class="text-info">欢迎和我交流</p>

</div>

</div>

10：页头组件

《div class=“page-header”》

<h1>大标题</h1>

<h2>中标题 《smal》小标题《/small》</h2>

<div>

1. 缩略图：

<div class="container">

<div class="row">

<div class="col-xs-6 col-sm-4 col-md-3 col-lg-3">

<div class="thumbnail">

<img src="699.jpg" />

</div>

</div>

<div class="col-xs-6 col-sm-4 col-md-3 col-lg-3">

<div class="thumbnail">

<img src="699.jpg" />

</div>

</div>

<div class="col-xs-6 col-sm-4 col-md-3 col-lg-3">

<div class="thumbnail">

<img src="699.jpg" />

</div>

</div>

<div class="col-xs-6 col-sm-4 col-md-3 col-lg-3">

<div class="thumbnail">

<img src="699.jpg" />

</div>

</div>

</div>

</div>

## Bootstrap JS插件：

一：模态框（弹出框）

两个必备元素：

1. 事件源绑定：直接声明data-toggle=“modal”(dropdown,collapse)

Data-target=”#模态框id”

1. 模态框搭建：

<button data-toggle="modal" data-target="#box">激活按钮</button>

<div class="modal fade" id="box">

<div class="modal-dialog model-dialog-sm">

<div class="modal-content">

<div class="modal-header">我是摩太狂的弹出层标题</div>

<div class="modal-body">我是摩太狂的主体</div>

<div class="modal-body">我是摩太狂的柱脚</div>

</div>

</div>

</div>

2：JS API方法：

$(模态框对象).modal({键值对1，键值对2});

二：滚动监听：

声明方法：

两个对象元素

1. 被监听对象，比如说<body>
2. 监听的反馈媒介。比如说nav

# CSS伪元素和伪类

## 伪类

## 链接伪类：

## :link鼠标未上上去时的链接样式，不太常用 ：visited 鼠标点击过后的链接样式

动态伪类：

：hover 鼠标悬停

：active 鼠标点击下去还没松开

：focus鼠标在某个元素上点击了一次并悬停在该元素上

静态伪类

：first-child 选择某一类元素当中的第一个子元素

## 2、伪元素

::after 在某个元素之后插入内容

::before 在某个元素之前插入

::first-letter 一个**块级元素**的首字符

::first-line一个**块级元素**的首行文本样式

具体用法：伪元素经常要绝对定位在父级元素里面，通常都是“子绝父相”定理。所以在设置的时候，要把父级元素设成相对或绝对定位。把它当成一个绝对定位的元素，不占据父级的空间就OK。

aside::after{

*content*: " > ";

*color*:#fff;

*position*: absolute;

*width*: 40px;

*height*: 20px;

*top*:0;

}

1. 可放置string，使得引号包括一段字符串
2. Attr()返回当前元素的属性
3. Url()用于引用媒体文件
4. 制作聊天气泡
5. 制作▼倒三角:

.paopao::after{

content:“”;

*height*:0;*width*:0;

*border-left*:5px solid transparent;/\*---调节左倾斜距离，transparert：透明---\*/

*border-right*:5px solid transparent;/\*---调节右倾斜距离---\*/

*border-top*:10px solid #666;/\*---调节高度---\*/

*position*:absolute;*right*:7px;*top*:15px;

}

1. 鼠标移入效果美化链接

7)伪元素实现行内元素换行，取代<br>

attr() – 调用当前元素的属性，可以方便的比如将图片的 Alt 提示文字或者链接的 Href 地址显示出来。示例：a:after { content:"(" attr(href) ")"; }

url() / uri() – 用于引用媒体文件。示例： h1::before { content: url(logo.png); }

counter() – 调用计数器，可以不使用列表元素实现序号功能。具体请参见 counter-increment 和 counter-reset 属性的用法。示例：   
h2:before { counter-increment: chapter; content: "Chapter " counter(chapter) ". " }

3、怎么区分伪元素和伪类？

在CSS3中为了区别伪元素和伪类，就为伪元素设置了双冒号：：的写法。伪类还是类，只不过没有具体的元素依托，伪元素有具体的结构和样式存在，但是不占有DOM结构，无法被选中。

# Less学习笔记

Less是一门CSS预处理语言（也叫预编译）

Less特点：

它扩展了CSS语言，增加了变量，

Mixin、嵌套、函数等的特点，使CSS更容易维护和扩展

作用环境：

可以在客户端，也可以在服务器端运行（node.js启动）。

Less用法1：

Less可以像js定义函数那样设置CSS各种属性，也可以像css那样调用该属性。

定义属性：

@属性：值

调用属性：直接写样式表

可以进行数学运算。

图片的引用：然后定义相对或绝对路径再引用。

用法2：mixin混合用法

普通混合：

直接设，直接调用。

#btns{

*color*:red;

.btns;

.bdr;

}

.bdr(@bdr:50%;){

*border-radius*:@bdr;

}

带参混合：

.boxshadow(@x,@y,@width:3px,@col:#000){

*box-shadow*: @arguments;

@x:2px;

@y:2px;

}//定义函数，

#box2{

*width*: 300px;

*height*: 200px;

.boxshadow(2px,2px,4px,#f60);

}//调用函数

函数方法混合调用。

混入里边还有一个重要变量：@arguments，和在预编译里模式一样，都是标准的函数声明与调用。

用法3：嵌套

.foot1{

ul{

*padding*:10px;

*border*:2px solid goldenrod;

li{

*background*: #082;

*border*:1px solid #009999;

a{*display*:block;*background*: grey;*width*:100px;*height*: 20px;*margin*: 10px auto;&:hover{*background*: yellow;}}

}

}

*background*:#245580;

*width*:500px;

*height*:200px;

}

嵌套主要针对于多重级别的元素嵌套的。对应的是函数的嵌套合闭包。

具体用法4：

注意作用域（变量作用范围）scope plugin

# 二维码的生成

Var Img1=document.getElementbyId("Img1");

Img1.resource=

"http://qr.liantu.com/api.php?text="+winodw.location.host+"?username="+userObj.username;

蓝色部分是固定的二维码生成网址，链接到该网址上生成一个二维码。红色部分是二维码的第一个参数，是本地网址，黄色部分+黑色部分是本地网址的参数，也就是本地网址中想要扫除来的东西。

# 移动端

预习：rem、 flex、 逻辑分辨率与物理分辨率。

学习目标：

1、掌握移动端开发的基础概念（理解+记忆）

1. 能够搭建屏幕适配的web页面。

引入概念：

## 一、移动开发正在成为前端技术开发的主流。

手机网页开发（bootstrap）和手机APP前端开发

手机APP前端开发：

主要包括webAPP、HybirdAPP、NativeAPP三种主流

webAPP：纯HTML+CSS+Js搭建的类似手机APP的手机wap网站（bootstrap）做成的APP

NativeAPP：手机上安装的各种应用，基于安卓或IOS开发的应用（如手机淘宝，美团等大部分页面）比如微信，支付宝

HybirdAPP：混合开发，将以上两种开发模式混合起来开发。rectNayive、ionic、AMAZUE（最火热的开发模式）

美工：做好切片交给ios或安卓开发，和前端没多大关系。

Bootstrap满不了移动web需求，web App要求高仿的页面，需要开发至少两套代码。

device.js的对象以及方法和使用。

移动web开发屏幕适配的布局解决方案

1. 百分比传统布局，适合简单的页面（详情页、登录注册）
2. 响应式布局 mediaquery @media（-webkit-device-pixel-ratio：2）{}还可以放到link标签里面，引入不同的样式。

## 二、移动端开发的特点：

1. H5和CSS3是主角，告别IE、主攻safari和chrome和UC、微信、小米、qq
2. 不支持鼠标光标，告别hover，focus
3. 不支持flash，没有falsh
4. 事件：touch替代click，可以理解为PC网页缩小版+一些触摸的特性

5、滚动条不再像预期那样起作用

6、快速搭建，成本低，可以调用各种手机上的应用接口，包括摄像头、联系人，

缺点：对开发人员要求比较高，激动前端又懂后台。

## 脚本侦测

步骤：1.确保Hbuilder（开发软件）与手机/平板在同一个网络下。

1. 打开Hbuilder的运行/设置外部服务器/web服务器/内置web服务器，设置IP地址为192.168.92.xxx。
2. 用手机搜IP地址：8020/html文件的绝对路径

192.168.92.xxx这个IP地址是针对其他移动/PC设备访问而言，而127.0.0.1这个IP地址是针对本地访问。

//判断是否是手机端

*var* isMobile=device.mobile(),

//判断是否是平板

isTable=device.tablet();

if(isMobile||isTable){

*window*.open("mobile.html","\_self");

}

else{

*window*.open("pc.html","\_self");

}

移动、WEB page的区分：

1. 普通的手机网页

meta 用法：

### meta控制：

Dns-prefetch在浏览网页的时候，浏览器在加载网页时会对网页的域名进行解析缓存，这样我们在点击当前网页链接时，无需进行DNS解析，减少浏览器的等待时间，属于网站优化的一个重要手段。

常识：一个是减少DNS请求，一个是进行DNS的预获取。

meta name="viewport" content="width=device-width, initial-scale=1,minimum-scale=1, maximum-scale=1, user-scalable=yes"

content属性值 :

     width:可视区域的宽度，值可为数字或关键词device-width

     height:同width

     intial-scale:页面首次被显示是可视区域的缩放级别，取值1.0则页面按实际尺寸显示，无任何缩放

     maximum-scale=1.0, minimum-scale=1.0;可视区域的缩放级别，

     maximum-scale用户可将页面放大的程序，1.0将禁止用户放大到实际尺寸之上。

     user-scalable:是否可对页面进行缩放，no 禁止缩放

1. 全屏属性：

format-detection翻译成中文的意思是“格式检测”，顾名思义，它是用来检测html里的一些格式的，那关于meta的format-detection属性主要是有以下几个设置：

meta name="format-detection" content="telephone=no"

meta name="format-detection" content="email=no"

meta name="format-detection" content="adress=no"

也可以连写：meta name="format-detection" content="telephone=no,email=no,adress=no"

下面具体说下每个设置的作用：

telephone=no就禁止了把数字转化为拨号链接！  
telephone=yes就开启了把数字转化为拨号链接，要开启转化功能，这个meta就不用写了,在默认是情况下就是开启！

email=no禁止作为邮箱地址！  
email=yes就开启了把文字默认为邮箱地址，这个meta就不用写了,在默认是情况下就是开启！

adress=no禁止跳转至地图！  
adress=yes就开启了点击地址直接跳转至地图的功能,在默认是情况下就是开启！

设置作者姓名及联系方式

<meta name="author" contect="name, xxx@163.com" />

带有苹果设备特效的设计图标（被苹果设备加工）：

<link href= rel="apple-touch-icon-precomposed"/>

常识：拟物化图标和扁平化图标

移动端开发常识：

1. 屏幕尺寸和移动设备的浏览器：

单位：英寸，屏幕对角线长度（inch）

单位：px，像素，pixel

1. 屏幕分辨率

屏幕分辨率是指屏幕图像的精细度，是显示器所能显示的具体数值。

1280\*720 分辨率越大，屏幕越好吗？并不是这样的。

**物理分辨率**：devic pixels

显示屏的最佳分辨率（真实分辨率）这才是决定屏幕清晰度的标准。

**逻辑分辨率**

缩放倍率：device pixel ratio（scale）

Window.devicePixelRatio设备屏幕的缩放倍率。

苹果：sacleFactor缩放倍率

安卓：density屏幕密度

逻辑分辨率=物理分辨率/缩放倍率

总结：物理分辨率是硬件所支持的水平，逻辑分辨率是软件可达到的水平。

移动端开发布局设计

## 屏幕适配的解决方案

1. 采用百分比的传统布局方法

子元素50%，父元素就是它的百

例如：Padding：2%;

范围：只适用一些简单的页面布局

1. 响应式媒体查询

@media only screen and (max-width:640px){}

@media only *screen and* (*min-width*: 100px) *and* (*max-width*: 640px) {}

@media only *screen and* (*min-width*: 100px) *and* (*min-device-pixel-ratio:2*) {}当最小缩放比为 2的时候

1. rem技术的应用

rem是CSS3引进的新单位，即根字体，给html设置字体。

所有的页面标签元素都可以使用rem作为px的替代品，需要配合js来获取屏幕宽度,再赋予根字体的大小（一般都是window.width/10），但是需要字体设置的话，还是要用font-size：14px;

window.innerWidth：获取浏览器屏幕的宽度

*document*.documentElement.clientWidth/*document*.body.clientWidth

都可以动态获取屏幕的宽度。

Html默认字体为16px；

那么1rem=16px；

font size of the root element

Html{font-size:16px;}

## flexible实现的主要功能：屏幕适配

1. 动态改写meta标签里的缩放比（在js里写）
2. 给html元素添加data-dpr属性并且根据屏幕的缩放比（dpr）进行赋值，作为所有元素的属性选择器
3. 给html元素添加font-size属性，设置rem的值
4. 设计规则：一般情况下只给和模型设置rem，文字字体，行高使用px。

*var* wid=*document*.documentElement.clientWidth/10+"px";

*document*.getElementsByTagName("html")[0].*style*.fontSize=wid;

alert(*document*.body.clientWidth)

设置完之后，1rem=1/10设备屏幕宽度

使用规则：不用给自己的meta标签设置viewpot属性了

rem区别em

em参照父亲，由于父亲不固定，em也没有固定的标准。而rem参照祖宗字体，是一个绝对标准。

Rem替代px（标题和详情的时候还是用，还是用px）

flexbile.js淘宝开源的解决方案：

Window.devicePixclRatio获得当前屏幕的缩放比来设置data-dpr

vw：1/10rem，通常为屏幕宽度的1%。

1. 设置全局属性html[data-dpr=]
2. Rem 的设置：1rem=10vw=html.style.fontSize=屏幕宽度/10。

vh、vmin、vmax

CSS3的布局方案

传统布局：盒模型->display、定位、浮动

缺点：垂直居中费劲，需要各种辅助手段

2009年W3C推出了新的布局方式：flex布局

IE10+，谷歌21+，火狐22+，safari6+

那么到底什么是弹性布局？

## Flex弹性布局：

flex布局：“flexible的缩写，弹性布局，用来为盒子模型提供最大的灵活性

用法：div{display：flex；}

具体用法：

-webkit-

### 1.display：flex；与dislpay：block；、display：inline、display：none是平级的。

### 2.父容器有6个属性。Item（子容器）有6个属性。

注意：设置flex布局之后，子元素的float、clear、vertical-align等属性无效，

弹性布局：

采用Flex布局（dispalay：flex；）的元素，称为Flex容器（flex container），简称"容器"。它的所有子元素自动成为容器成员，称为Flex项目（flex item），简称"项目"。

容器默认存在两根轴：水平的主轴（main axis）和垂直的交叉轴（cross axis）。主轴的开始位置（与边框的交叉点）叫做**main start**，结束位置叫做main end；交叉轴的开始位置叫做**cross start**，结束位置叫做**cross end**。

项目默认沿主轴排列。单个项目占据的主轴空间叫做**main size**，占据的交叉轴空间叫做**cross size。**

### 父容器属性1：flex-direction：

*决定item在哪个轴上，默认是row（main轴）。*

*Row：main轴，从左到右。*

row-reverse;main轴反转，从右到左

Column：cross轴，从上到下

column-reverse;cross轴翻转，从下到上

对于父容器来说：设置display：flex;

Width：calc（100%-2\*padding值）；padding：值；这样可以实现既撑满父亲容器高度，又可实现padding。如果不这样，屏幕会出现滚动条。

### 父容器属性2：flex-wrap:

规定items是否换行：

Nowrap（默认）不换行，就是一行排开，不会产生第二行

Wrap:换行，第一行在最上方。

Wrap-reverse：换行反转，第一行在最下方。

### 父容器属性3：Flex-flow

该属性为flex-*direction和flex-wrap属性的缩写，*默认值为row nowrap

### 父容器属性4：justify-content：

定义了项目在主轴上的对齐方式。

flex-start（默认值）：左对齐

flex-end：右对齐

center： 居中

space-between：两端对齐，项目之间的间隔都相等，但与父盒子内边框的间距为0 ，所以两边的紧靠父盒子边框。

space-around：每个项目两侧的间隔相等，距离父盒子内边框也等距离间隔。所以，项目之间的间隔比项目与父盒子边框的间隔大一倍。

### 父容器属性5：align-items：

规定元素在cross轴上如何对齐。

flex-start：交叉轴的起点对齐。

flex-end：交叉轴的终点对齐。

center：交叉轴的中点对齐。即居中

baseline: 项目的第一行文字的基线对齐。

stretch（默认值）：如果项目未设置高度或设为auto，将占满整个容器的高度。

### 父容器属性6：align-content

定义了多根轴线的对齐方式。如果项目只有一根轴线，该属性不起作用。

flex-start：与交叉轴的起点对齐。

flex-end：与交叉轴的终点对齐。

center：与交叉轴的中点对齐。

space-between：与交叉轴两端对齐，轴线之间的间隔平均分布。

space-around：每根轴线两侧的间隔都相等。所以，轴线之间的间隔比轴线与边框的间隔大一倍。

stretch（默认值）：轴线占满整个交叉轴。

## Item的属性：

### 1.order

order属性定义项目的排列，数值越小，排列越靠前，默认值为0，值必须为integer（整数）

### 2.flex-grow

flex-grow属性定义项目的放大比例，默认值0，不放大，值可以是小数。如果所有的item flex-grow属性都为1，则他们将等分充满剩余空间；如果一个项目的flex-grow属性为2，其他项目为1，则前者占据的剩余空间将比其他项目占据的空间多一倍。

### 3.flex-shrink

flex-shrink缩小比例，默认为1，0是不缩小。如果空间不够，只将该项目缩小。如果所有项目的flex-shrink属性都为1，当空间不足时，都将等比例缩小。如果一个项目的flex-shrink属性为0，其他项目都为1，则空间不足时，前者不缩小。

### 4.flex-basis

flex-basis 定义该项目在主轴占据的宽度（类似width，但比width权重高），默认值是auto，item space in main size ，代表该项目本来的大小。

### 4.flex

flex属性 有flex-grow、flex-shrink、flex-basis的简写

复习（flex-flow，有两个属性）

默认值：0 1 auto，第一个属性必填。通常设flex：1，占满额外的空间

flex： 1 1 auto；沾满额外的空间，可缩放。

flex： 0 0 auto；不占满额外空间，不可缩放

### 6.align-self

align-self：垂直方向自甘堕落，允许单个项目与其他项目不一样的对齐方式。可以覆盖父元素align-items属性。

值有：

Auto：默认值。元素继承了它的父容器的 align-items 属性。如果没有父容器则为 "stretch"。

Stretch：元素被拉伸以适应容器。

Center：元素位于容器的中心。

flex-start：元素位于容器的开头。

flex-end：元素位于容器的结尾。

Baseline：元素位于容器的基线上。

实现网格布局：屏幕小于750px，单行分布。

# 移动端事件的基础知识

touch事件

PC端：鼠标点下可能包含的事件：mousedown，mouseup，mousemove。

Click事件主要包含mousedown，mouseup。

移动端（触摸设备）:手指触摸可能发生的事件：touchstart,touchmouve,touchend

## 监听事件：

ele.addEventListener("touchstart",testfun, false);

.addEventListener(“被监听的函数”，“监听函数”，“布尔值”)

被监听的函数：通常是浏览器内置的事件函数，比如onclick，touchstart，

布尔值 true：允许事件冒泡

False：阻止事件冒泡

**touchstart事件**：当手指触摸屏幕时候触发，即使已经有一个手指放在屏幕上也会触发。

**touchmove事件**：当手指在屏幕上滑动的时候连续地触发。在这个事件发生期间，调用preventDefault()事件可以阻止滚动。

**touchend事件**：当手指从屏幕上离开的时候触发。

**touchcancel事件**：当系统停止跟踪触摸的时候触发。关于这个事件的确切出发时间，文档中并没有具体说明，咱们只能去猜测了。

上面的这些事件都会冒泡，也都可以取消。虽然这些触摸事件没有在DOM规范中定义，但是它们却是以兼容DOM的方式实现的。所以，每个触摸事件的event对象都提供了在鼠标实践中常见的属性：bubbles(起泡事件的类型)、cancelable(是否用 preventDefault() 方法可以取消与事件关联的默认动作)、clientX(返回当事件被触发时，鼠标指针的水平坐标)、clientY(返回当事件触发时，鼠标指针的垂直坐标)、screenX(当某个事件被触发时，鼠标指针的水平坐标)和screenY(返回当某个事件被触发时，鼠标指针的垂直坐标)。除了常见的DOM属性，触摸事件还包含下面三个用于跟踪触摸的属性。

**touches**：表示当前跟踪的触摸操作的touch对象的数组。

**targetTouches**：特定于事件目标的Touch对象的数组。

**changeTouches**：表示自上次触摸以来发生了什么改变的Touch对象的数组。

　　每个Touch对象包含的属性如下。

**clientX**：触摸目标在视口中的x坐标。

**clientY**：触摸目标在视口中的y坐标。

**identifier**：标识触摸的唯一ID。

**pageX**：触摸目标在页面中的x坐标。

**pageY**：触摸目标在页面中的y坐标。

**screenX**：触摸目标在屏幕中的x坐标。

**screenY**：触摸目标在屏幕中的y坐标。

**target**：触目的DOM节点目标。

## Touch对点击事件的影响：

PC的<a>默认实在onclick事件中处理，移动设备也用onclick事件，但会有300ms（现在是100-120ms，大于0.1s）的延迟，所以快速响应式UI的重中之重。

延迟带来的后果：

zepto的tap事件

点透事件：弹出层关闭时候，会出现点透事件

解决方法一：用touchend方法代替click：移动端的交互体验全靠触摸的，touchstart或touchend都会干扰其他交互行为，比如说滚动或拖拽。（不推荐）

解决方法二：e.preventDefault来阻止默认事件（有可能把其他默认事件比如a标签不执行，所以也不推荐）

解决方法三：引入fastclick.js，用当单击事件时，click来代替touchstart或touchend。（强烈推荐）

## DPR的问题：

一：CSS里的1px问题

在头部标签

<meta name="viewport"; content="width=device-width,initial-scale=1, minimum-scale=1, maximum-scale=2, user-scalable=no">情况下，

Iphone6获取的viewport为275px，

Div{height：5rem；width：5rem；border：1px solid #000；}

在浏览器上实际显示是2px

解决方法

1：设置initial-scale=0.5；

2：border-width=0.5px（一些安卓手机下，识别为0px）

@media（-webkit-min-device-pixel-ratio：2）{

div{border-width：0.5px；}

}

二背景图片用background-size来实现

1、auto：此值为默认值，保持背景图片的原始高度和宽度;  
2、number：此值设置具体的值，可以改变背景图片的大小;  
3、length/percentage：此值为百分值，可以是0%〜100%之间任何值，但此值只能应用在块元素上，所设置百分值将使用背景图片大小根据所在元素的宽度的百分比来计算。它的值有两个，第一个设置高度，第二个设置宽度，如果只设定一个值，那么第二个值为自动  
4、cover：此值是将图片放大，以适合铺满整个容器，这个主要运用在，当图片小于容器时，又无法使用background-repeat来实现时，我们就可以采用cover;将背景图片放大到适合容器的大小，但这种方法会使用背景图片失真;  
5、contain：此值刚好与cover相反，其主要是将背景图片缩小，以适合铺满整个容器，这个主要运用在，当背景图片大于元素容器时，而又需要将背景图片全部显示出来，此时我们就可以使用contain将图片缩小到适合容器大小为止，这种方法同样会使用图片失真。

小技巧

1、不可选中，禁止复制、选中文本

{

-webkit-user-select：none；

User-selecct：none；

2、H5调用安卓或者ios的拨号功能

<a href=”tel:139XXXXXXX”>一键拨打电话号码</a>

<a href=”sms:139XXXXXXX”>一键发送短信</a>

1. 避免出现ios和android下触摸元素时出现半透明灰色遮罩

-webkit-tap-height-color：rgba（0,0，0,0）；

1. 阻止旋转屏幕时自动调整字体大小

-webkit-text-size-adjust是webkit的私有CSS

html、body、form、dieldest、屏p、div等

{-webkit-text-size-adjust：100%；text-size-adjust：100%；}

1. 去除input自带的默认样式：

Input，textarea{border：0；-webkit-appearance：none；}

1. 模拟hover效果：

*var* myLinks=*document*.getElementsByTagName("a");

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

myLinks[i].addEventListener("touchstart",*function*(){*this*.classList.add("hover");},false); myLinks[i].addEventListener("touchend",*function*(){*this*.classList.remove("hover");},false);

}

.classList.add("属性名")：添加属性名

1. 移动端HTML5audio autoplay失效问题

自动播放网页中的音频或视频，会给用户带来一些困扰或者不必要的流量消耗，所以苹果系统和安卓系统通常会禁止自动播放和使用JS的触发播放，即必须由用户来触发才能播放。

Document.addRventListner("touchstart",*function*(){

document.getELementsbyTagname(“audio”)[0].paly();

document.getELementsbyTagname(“audio”)[0].pause();

},false);

1. 移动端HTML5input data不支持placeholder的问题

CSS：

input[type=“date”]:before{color:#A9A9A9;contant:attr(plceholder)}

input[type=“type=data”]:before{color:black;contant:”!import”}

JS:

$(“#date”).on(“input”,function(){

If($(this).val().length>0){

$(this).addClass(“full”);

}

else{

$(this).removeClass(“full”);

}

})

HTML

<input id=”date” type=”date” placeholder=”请选择出生日期” style=“height：41px;overflow:hidden;”>

1. input type=”number”的两个问题

Type=text maxlength正常，type=number maxlength无效

解决方案：

<input type="number" maxlength=5 oninput="if(value>5)value=value.slice(0,5)"/>

<input type="text" maxlength=5 oninput="if(value>5)value=value.slice(0,5)"/>

Background图片自适应屏幕大小

# npm使用介绍

npm是随同NodeJS一起安装的包管理工具，能解决NodeJS代码部署上的很多问题，常见的使用场景有以下几种：

允许用户从NPM服务器下载别人编写的第三方包到本地使用。

允许用户从NPM服务器下载并安装别人编写的命令行程序到本地使用。

允许用户将自己编写的包或命令行程序上传到NPM服务器供别人使用。

由于新版的nodejs已经集成了npm，所以之前npm也一并安装好了。同样可以通过输入 "npm -v" 来测试是否成功安装。命令如下，出现版本提示表示安装成功:

$ npm -v2.3.0

如果你安装的是旧版本的 npm，可以很容易得通过 npm 命令来升级，命令如下：

$ sudo npm install npm -g/usr/local/bin/npm -> /usr/local/lib/node\_modules/npm/bin/npm-cli.js

npm@2.14.2 /usr/local/lib/node\_modules/npm

如果是 Window 系统使用以下命令即可：

npm install npm -g

使用淘宝镜像的命令：

cnpm install npm -g

## 使用 npm 命令安装模块

npm 安装 Node.js 模块语法格式如下：

$ npm install <Module Name>

以下实例，我们使用 npm 命令安装常用的 Node.js web框架模块 express:

$ npm install express

安装好之后，express 包就放在了工程目录下的 node\_modules 目录中，因此在代码中只需要通过 require('express') 的方式就好，无需指定第三方包路径。

var express = require('express');

升级npm:

npm install npm@ltest -g

升级cnpm：

npm install cnpm@ltest -g

## 使用淘宝 NPM 镜像

大家都知道国内直接使用 npm 的官方镜像是非常慢的，这里推荐使用淘宝 NPM 镜像。

淘宝 NPM 镜像是一个完整 npmjs.org 镜像，你可以用此代替官方版本(只读)，同步频率目前为 10分钟 一次以保证尽量与官方服务同步。

你可以使用淘宝定制的 cnpm (gzip 压缩支持) 命令行工具代替默认的 npm:

$ npm install -g cnpm --registry=https://registry.npm.taobao.org

## 全局安装与本地安装

npm 的包安装分为本地安装（local）、全局安装（global）两种，从敲的命令行来看，差别只是有没有-g而已，比如

npm install express # 本地安装

npm install express -g # 全局安装

如果出现以下错误：

npm err! Error: connect ECONNREFUSED 127.0.0.1:8087

解决办法为：

$ npm config set proxy null

### 本地安装

1. 将安装包放在 ./node\_modules 下（运行 npm 命令时所在的目录），如果没有 node\_modules 目录，会在当前执行 npm 命令的目录下生成 node\_modules 目录。

2. 可以通过 require() 来引入本地安装的包。

### 全局安装

1. 将安装包放在 /usr/local 下或者你 node 的安装目录。

2. 可以直接在命令行里使用。

如果你希望具备两者功能，则需要在两个地方安装它或使用 npm link。

接下来我们使用全局方式安装 express

$ npm install express -g

安装过程输出如下内容，第一行输出了模块的版本号及安装位置。

express@4.13.3 node\_modules/express├── escape-html@1.0.2├── range-parser@1.0.2├── merge-descriptors@1.0.0├── array-flatten@1.1.1├── cookie@0.1.3├── utils-merge@1.0.0├── parseurl@1.3.0├── cookie-signature@1.0.6├── methods@1.1.1├── fresh@0.3.0├── vary@1.0.1├── path-to-regexp@0.1.7├── content-type@1.0.1├── etag@1.7.0├── serve-static@1.10.0├── content-disposition@0.5.0├── depd@1.0.1├── qs@4.0.0├── finalhandler@0.4.0 (unpipe@1.0.0)├── on-finished@2.3.0 (ee-first@1.1.1)├── proxy-addr@1.0.8 (forwarded@0.1.0, ipaddr.js@1.0.1)├── debug@2.2.0 (ms@0.7.1)├── type-is@1.6.8 (media-typer@0.3.0, mime-types@2.1.6)├── accepts@1.2.12 (negotiator@0.5.3, mime-types@2.1.6)└── send@0.13.0 (destroy@1.0.3, statuses@1.2.1, ms@0.7.1, mime@1.3.4, http-errors@1.3.1)

### 查看安装信息

你可以使用以下命令来查看所有全局安装的模块：

$ npm list -g

├─┬ cnpm@4.3.2│ ├── auto-correct@1.0.0│ ├── bagpipe@0.3.5│ ├── colors@1.1.2│ ├─┬ commander@2.9.0│ │ └── graceful-readlink@1.0.1│ ├─┬ cross-spawn@0.2.9│ │ └── lru-cache@2.7.3……

如果要查看某个模块的版本号，可以使用命令如下：

$ npm list grunt

projectName@projectVersion /path/to/project/folder└── grunt@0.4.1

## npm 常用命令

Npm Install

Npm install <package-name> 从npm官方库中下载node\_modules包。

Npm publish

Npm publish用于发布自己的node.js包

Npm login

Npm login 用于发布前的登录

NPM提供了很多命令，例如install和publish，使用npm help可查看所有命令。

NPM提供了很多命令，例如install和publish，使用npm help可查看所有命令。

使用npm help <command>可查看某条命令的详细帮助，例如npm help install。

在package.json所在目录下使用npm install . -g可先在本地安装当前命令行程序，可用于发布前的本地测试。

使用npm update <package>可以把当前目录下node\_modules子目录里边的对应模块更新至最新版本。

使用npm update <package> -g可以把全局安装的对应命令行程序更新至最新版。

使用npm cache clear可以清空NPM本地缓存，用于对付使用相同版本号发布新版本代码的人。

使用npm unpublish <package>@<version>可以撤销发布自己发布过的某个版本代码。

这样就可以使用 cnpm 命令来安装模块了：

$ cnpm install [name]

## ES 模块与 Node.js模块

当 Node.js 开始出现时，那时并没有 ES 模块的提议。于是，Node.js 决定使用 [CommonJS](https://en.wikipedia.org/wiki/CommonJS) 模块。当 CommonJS 这个组织不再活跃，Node.js 和 npm 融入了规范中，一起创造了一个很大的 JavaScript 生态。[Browserify](http://browserify.org/) 与更近期的 [webpack](https://webpack.github.io/) 为浏览器带来了 Node 版的 CommonJS 模块，优雅地解决了浏览器端的模块问题。因此，Node / npm JavaScript 模块生态已经扩散到服务器端与客户端，也发展地非常迅速。

但在一个大环境中，我们该如何处理标准的 ES 模块与 CommonJS 风格的模块的互操作性呢？自 ES 模块规范化进程开始以来，这个问题一值都在激烈地争论中。

Browserify 和 webpack 减少了浏览器端与服务端的差距，使 JavaScript 开发变得容易，也有所统一。如果失去了可互操作性，我们便增加了现有生态与新的标准的不同。如果前端开发人员选择 ES 模块作为他们的默认选择，而服务端工程师则继续使用 Node 版的 CommonJS 的话，它们之前的差距只会越来越大。

## mjs与js

mjs文件与js文件都是js的模块化标识，只是二者遵循的规则不一样。mjs模块遵循ES规则 ，而js遵循CommonJS规则。

例如，如果一个开发者想创建一个模块，为了向后兼容，导出两种模块类型（CommonJS 与 ES 模块），它的 package.json 可能会被定义如下：

{

"name": "test",

"version": "0.0.1",

"description": "",

"main": "./index", // 没有文件扩展名

}

这个包就会同时有 index.mjs 与 index.js。index.mjs 就是 ES 模块，使用新的 export / import 语法：

// index.mjs

export default class Foo {

//..

}

而 index.js 则是 CommonJS 风格的模块，使用 module.exports 对象：

// index.js

class Foo {

// ...

}

module.exports = Foo;

如果你正使用的 Node.js 版本支持通过 .mjs 文件扩展名来识别 ES 模块，它会先尝试找到 index.mjs 文件。然而，如果当前版本的 Node.js 不支持 ES 模块时（如 Node.js v4 和 v6），或者支持但找不到 index.mjs 时，它会找 index.js 文件。

根据这个 EP，你可能可以同时使用 require 和 import 来从你 node\_modules 里找到包：

import mkdirp from 'mkdirp';

require('mkdirp');

为了归约本地项目或包的模块，你无需在你的 require() 或 import 语句中添加文件扩展名，除非你想更准确些。如果你不提供文件扩展名，那就会采用标准的 Node.js 文件归约算法，而且 .mjs 优先于 .js。

require('./foo');

import './foo';// 它们都将寻找

// ./foo.mjs

// ./foo.js

// ./foo/index.mjs

// ./foo/index.js

// 添加 '.js'，显示地加载 CJS 模块import './foo.js';

// 添加 '.mjs'，显示地加载 ES 模块import './bar.mjs'

## 在 ES 模块中加载 CommonJS 模块

// cjs.js

**module**.**exports** = {

**default**:'my-default',

thing:'stuff'

};

// es.mjs

**import** \* **as** baz **from** './cjs.js';

// baz = {

// get **default**() {**return** module.exports;},

// get thing() {**return** **this**.**default**.thing}.bind(baz)

// }

// console.log(baz.**default**.**default**);

// my-**default**

**import** foo **from** './cjs.js';

// foo = {default:'my-default', thing:'stuff'};

**import** {**default** **as** bar} **from** './cjs.js';

// bar = {default:'my-default', thing:'stuff'};

## 在 CommonJS 模块中使用 ES 模块

// es.mjs

let foo = {bar:'my-default'};

**export** **default** foo;

foo = null; // **this** null value does **not** effect **import** value.

// cjs.js

**const** es\_namespace = require('./es');

// es\_namespace ~= {

// get default() {

// return result\_from\_evaluating\_foo;

//}

// }

console.log(es\_namespace.default);// {bar:'my-default'}

## require 与 import 规则：

尽管 Bradley 已经发布的 EP 内建于协同流程，但考虑到提议的替代方案，它收到了来自 EP 流程之外的一个突出的反提议。该反提议名为 "[捍卫 .js（In Defense of .js）](https://github.com/dherman/defense-of-dot-js/blob/master/proposal.md)"，它依赖于对 package.json 的使用，而不是新的文件扩展名。即使这个选择之前已经讨论过，这个新提议仍然还有一些有趣的观点。

捍卫 .js 提出了一下规则来决定以何种格式来加载文件，以及同样的 require 与 import 规则：

* 如果 package.json 有 "main" 字段但没有 "module" 字段，包内的所有文件都以 CommonJS 模块加载。
* 如果 package.json 有 "module" 字段但没有 "main" 字段，包内的所有文件都已 ES 模块加载。
* 如果 package.json 既没有 "main" 字段也没有 "module" 字段，它将依赖于包中存在的是 index.js 或是 module.js 来决定是作为 CommonJS 模块或是 ES 模块加载。
* 如果 package.json 同时有 "main" 字段与 "module" 字段，除了在 "module" 字段的文件将作为 ES 模块加载外，也包括目录，包内的其他文件将作为 CommonJS 模块加载。
* 如果加载处没有 package.json（如 require(‘c:/foo’)），那它将默认作为 CommonJS 模块加载。
* 如果 package.json 中有特殊的 "module.root" 字段，那么该目录下的所有文件都将作为 ES 模块加载。此外，从相对于包自身的路径处加载的文件（如 require('lodash/array')）将在这个目录内加载。

// package.json// 所有文件以 CommonJS 加载

{

"main": "index.js" // 包的默认模块

}

// package.json

// 默认使用 CommonJS，条件地以 ES 模块加载

{

"main": "index.js", // 给老版本的 **Node**.**js** 使用，作为默认模块，CommonJS

"module": "module.js" // 给较新版本的 **Node**.**js** 使用，作为默认模块，ES 模块

}

// package.json// 默认 CommonJS，一些文件目录例外

{

"main": "index.js",

"module": "module.js",

"modules.root": "lib" // 该目录下所有文件以 ES 模块加载

}

# [创建并发布自己的Node.js模块](https://docs.npmjs.com/creating-node-js-modules)

Node.js模块是一种可以发布到npm 的[包](https://docs.npmjs.com/about-packages-and-modules)。

### 步骤

注册一个npm账号，用于发布模块版本的时候登录。

[创建一个package.json文件](https://docs.npmjs.com/creating-node-js-modules" \l "create-a-package-json-file)

[创建在另一个应用程序需要模块时将加载的文件](https://docs.npmjs.com/create-the-file-that-will-be-loaded-when-your-module-is-required-by-another-application)

下载并引用[你的模块](https://docs.npmjs.com/creating-node-js-modules" \l "test-your-module)

### [创建一个package.json文件](https://docs.npmjs.com/creating-node-js-modules" \l "create-a-packagejson-file)

创建一个文件夹目录作为根目录，要package.json在命令行上创建文件，请在Node.js模块的根目录中运行npm init：

对于[范围模块](https://docs.npmjs.com/about-scopes)，请运行npm init --scope=@scope-name

对于未[范围的模块](https://docs.npmjs.com/creating-and-publishing-unscoped-public-packages)，请运行npm init

提供必填字段（name和version）以及字段的响应main：

name：必填，模块的名称。

version：必填，初始模块版本。我们建议遵循[语义版本控制准则](https://docs.npmjs.com/about-semantic-versioning)并[从中](https://docs.npmjs.com/about-semantic-versioning)开始1.0.0。

main：必填，当另一个应用程序需要您的模块时将加载的文件的名称。默认名称是index.js。

### [创建您的模块文件](https://docs.npmjs.com/creating-node-js-modules" \l "create-the-file-that-will-be-loaded-when-your-module-is-required-by-another-application)

一旦你的package.json文件被创建，创建所需的模块时，将加载该文件。该文件的默认名称是index.js。所以要在你创建的根目录里新建一个index.js.你的中心模块文件就写在index.js里。

由于浏览器不支持require写法，但ES6支持import与export写法。而node端支持require方法。所以：node端和Web端各有自己的暴露与引用写法。

1.node端或ES5的写法：

在文件中，添加一个函数作为exports对象的属性。这将使该功能可用于其他代码：

const a= "This is a message from the demo package"

exports.printMsg = function() {

console.log(a);

}

如果需要引入其他依赖项（包括自己写的其他js模块），那么就按照相对路径require其他的依赖项。

1. Web端的ES6写法：

const a= "This is a message from the demo package"

export default Function printMsg() {

console.log(a);

}

如果需要引入其他依赖项（包括自己写的其他js模块），那么就按照相对路径import其他的依赖项。

### [在package.json文件中指定依赖](https://docs.npmjs.com/specifying-dependencies-and-devdependencies-in-a-package-json-file)包

如果你创建的node包本身有依赖，那么就要从命令行向package.json文件添加依赖项和devDependencies ，可以使用--save-prod依赖项标志（默认行为npm install）或--save-devdevDependencies标志将它们安装在程序包的根目录中：

要"dependencies"在package.json文件属性中添加条目，请在命令行上运行以下命令：

npm install <package-name>

要"devDependencies"在package.json文件属性中添加条目，请在命令行上运行以下命令：

npm install <package-name> --save-dev

### [发布你的模块到npm](https://docs.npmjs.com/creating-node-js-modules" \l "test-your-module)

将您的包发布到npm：

在发布前，请先用npm login登录，否则可能会报没有权限的错误。

对于[私有包](https://docs.npmjs.com/creating-and-publishing-private-packages" \l "publishing-private-packages)和未[包装的包](https://docs.npmjs.com/creating-and-publishing-unscoped-public-packages" \l "publishing-unscoped-public-packages)，请使用npm publish。

对于[作用域公共包](https://docs.npmjs.com/creating-and-publishing-scoped-public-packages" \l "publishing-scoped-public-packages)，请使用npm publish --access public

### 从何npm上下载引用你的模块包

在命令行上，在项目目录之外创建一个新的测试目录。

mkdir test-directory

切换到新目录：

cd /path/to/test-directory

在测试目录中，安装模块：

cnpm install <your-module-name> --save

### 引用你的模块：

在测试目录中，创建一个test.js需要模块的文件，并将模块作为方法调用。由于ES6的写法不能直接用node的npm命令行执行，要用babel编译之后才能运行在js上。所以运行在Node端的可直接测试。

1. Node模块或ES5的写法：

Var printMsg = require（‘your-module-name’）；

printMsg（）；

1. Web端的ES模块写法：

Import printMsg from ‘your-package-name’；

printMsg（）；

在命令行上，运行node test.js。将显示发送到console.log的消息。

# 基于webpack[创建react-antd组件](https://docs.npmjs.com/creating-node-js-modules)包

由于react的组件包与基本的node包不一样,它依赖babel、antd、react等复杂的es6语法结构，所以它的创建方法是通过webpack打包来创建的，是一个可打包的react-antd工程项目。

## 创建项目，指定入口文件

创建一个文件夹作为项目的根目录，npm init创建package.json，然后用npm install项目的依赖包，下到package.json中。你的package.json大致如下：

{

"name": "react-modules",

"version": "0.0.1",

"description": "react-modules1",

"main": "lib/index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1",

"build": "webpack --progress",//配置webpack的打包指令

"watch": "webpack --watch --progress"//配置webpack的运行指令

},

"author": "",

"license": "ISC",

"dependencies": {

"@babel/preset-env": "^7.1.6",

"@babel/preset-react": "^7.0.0",

"babel-core": "^6.26.3",

"react-dom": "^16.6.3",

"webpack": "^4.26.0"

},

"devDependencies": {

"@babel/core": "^7.1.6",

"antd": "^3.10.8",

"babel-core": "^6.26.3",

"babel-loader": "^8.0.4",

"babel-preset-env": "^1.7.0",

"babel-preset-react": "^6.24.1",

"clean-webpack-plugin": "^1.0.0",

"css-loader": "^1.0.1",

"less-loader": "^4.1.0",

"react": "^16.6.3",

"react-router-dom": "^4.3.1",

"style-loader": "^0.23.1",

"webpack-cli": "^3.1.2"

}

}

由于是打包项目，故要指定入口文件为lib/index.js，这点尤其重要。

在根目录下创建一个src文件夹，里面创建一个index.js文件作为webpack打包的入口文件。

在src文件夹下再建一个专门写组件的组件文件夹（components）。

在组件文件夹（components）下写我们正式的组件js(index.js)与组件css(index.css)。

你的项目结构应该如下：



## 在根目录下配置webpack与babel的打包环境

1. 配置webpack.cofig.js与.babelrc.js：

由于package.json中指定了node包的入口文件为lib/index.js，所以我们的webpack打包的输出目录与输出文件应与package.json中指定的相同：

//webpack.config.js:

const path = require('path');

const CleanWebpackPlugin = require('clean-webpack-plugin');

module.exports = {

mode: 'production',

entry: {

index: './src/index.js'

},

output: {

filename: '[name].js',

path: path.resolve(\_\_dirname, "lib"),

libraryTarget: 'commonjs2',

},

module: {

rules: [

{

test: /\.js$/,

exclude: /node\_modules/,

loader: 'babel-loader',

options: {//引入babel，支持react（jsx）、es6等语法

presets: ['@babel/preset-env',"@babel/preset-react"]

},

{

test: /\.css$/,

use: ['style-loader','css-loader']

},

// 下面的less-loader是为了支持antd

// less-loader的文档： https://www.npmjs.com/package/less-loader

{

test: /\.less$/,

use: [{

loader: 'style-loader',

},

{

loader: 'css-loader',

},

{

loader: 'less-loader',

options: {

modifyVars: { // 主体颜色

'primary-color': '#40a9ff',

'link-color': '##40a9ff',

'border-radius-base': '4px',

'font-size-base': '12px',

'line-height-base': '1.2',

},

javascriptEnabled: true,

},

}],

},

]

},

plugins: [

new CleanWebpackPlugin(['lib'])

]

}

//.Babelrc.js:

const plugins = [

["@babel/plugin-proposal-class-properties", {

"loose": false

}],

["@babel/plugin-syntax-import-meta"],

// 以下是为了支持antd的按需加载

// https://github.com/ant-design/babel-plugin-import#usage

// https://ant.design/docs/react/getting-started-cn#按需加载

["import", { "libraryName": "antd", "libraryDirectory": "lib", "style": true }, "ant"],

]

module.exports = {

plugins

};

## 3.编写自己的组件

### src/components/index.js:

import React from 'react';

import { Checkbox, DatePicker, Select, Input, Radio, Tree } from 'antd';

import './index.css';

/\*\*ConditionDatePicker

\* dateSelectContent复选框汉语文字

\* dateSelect:选择日期的方法

\* 返回父组件的值：dateString：时间数组，String类型

\*/

class ConditionDatePicker extends React.Component {

constructor(props) {

super(props);

this.state = {

disabled: true,

};

}

componentWillMount() {

// console.log(this.props);

}

componentDidMount() {}

onOk(value) {}

render() {

const { dateSelectContent } = this.props;

const { disabled } = this.state;

return (

<div>

<div className="ConditionDatePicker\_mormal">

<Checkbox

className="ConditionDatePicker\_che"

onChange={(e) => { this.setState({ disabled: !e.target.checked }); }}

>

{dateSelectContent}

</Checkbox>

<DatePicker

style={{ width: '100%', marginLeft: '-12px', marginRight: '-8px' }}

disabled={disabled}

showTime

format="YYYY-MM-DD HH:mm:ss"

placeholder="请选择时间"

onChange={(value, dateString) => {

const { dateSelect } = this.props;

dateSelect(dateString);

}

}

onOk={this.onOk}

/>

</div>

</div>

);

}

}

Class others extends React.Component {

...

}

...

export default {

ConditionDatePicker,

Others,

...

}

### src/components/index.css:

.ConditionDatePicker\_mormal{ height: 32px;line-height: 32px;display: flex; justify-content: flex-start;flex-wrap: nowrap;}

.ConditionDatePicker\_che{width: 200px;}

### Src/index.js:

import index from './components/index';

export default index;

//console.log(index);

## 4.打包

在根目录下运行npm run build。如果打包成功，便说明react-antd包没有语法错误。

引用

# 发布npm包到Nexus私服上

## 1.注册一个Nexus用户。

打开Nexus Repository Manager，并注册一个管理员权限,Nexus Repository Manager里面可能有几层权限，但是你要获取最高权限才能发布。这个用户包括你的username，password，还有你的邮箱并登陆。

## 2.创建私库。

点击 Create repository，选择npm(hosted) 类型，这个是存放私服包的库，这里起名为rupiahone-hosted, 然后点击下面的 Create repository 即可。

3.创建代理库。

点击 Create repository ，选择 npm(proxy)，这里名称为rupiahone-proxy。

在 Remote storage 填入代理源，这里填入的是npm官方的地址https://registry.npmjs.org。如果你有需要，也可以填入淘宝镜像地址https://registry.npm.taobao.org。

点击下面的 Create repository 即可。

4.创建仓库组。

点击 Create repository ，选择 npm(group)，这里名称为rupiahone-group。

在下面Group的视图中，将上面步骤创建的两个仓库添加到右边。

创建完成之后，url列表里会抛出3个url。

## 5.配置publish的npm权限

在设置界面，点击Security下的Realms，将npm Bearer Token Realm 添加到右边，然后保存。(这一步非常重要，否则publish 包到私服的时候会报没有权限的错误)

## 6.在本地用npm-init创建并编写好node包。

## 7.配置本地的.npmrc文件

npm config edit ，自动弹出windows的.npmrc文件，修改该文件，将对应的url配置到npm使用的注册表。

命令行写法：npm config set registry your-url

或者：在.npmrc中直接修改字段为：

registry = your-url。

查看.npmrc文件配置的registry-url：

npm config get registry

你也可以在你的package.json里添加publishConfig，例如：

"publishConfig": {

"registry": "http://192.168.144.184:8081/repository/npm-proxy/"

},

## 8.登录验证用户权限

npm login --registry = your-url

输入你注册的账号、密码与邮箱，成功完成后，会有验证成功的结果返回。npm将自动将用于验证此组合的行添加到.npmrc 特定存储库的配置文件中。

## 发布您的包到Nexus私服上。

由于远程代理的proxy仓库需要远程账号密码，而group仓库整合了hosted仓库跟proxy仓库，所以只能上传到hosted仓库。

npm publish --registry <>your-host-url>。

## 从Nexus私服上npm install你的npm包。

由于私服（Nexus）仓库的包与npm公网仓库的包不能混用，所以如果你的项目文件夹下已经引用了开源的npm公共包，请在你的项目文件夹外部新建一个文件夹，单独下载并抛出你的私有包，来作为项目公用的工具架包使用。所以你的私有架包与你的项目文件夹的目录关系是并列关系，而不是包含关系。

### 方法一：网页直接下载

1. 登录NexusUI界面，登录你的账户密码。
2. 在用户界面上的Brose浏览页上找到npmhosted或者npm-groups条目，单击打开，会出现你发布过的npm包列表，展开你的包，单击子项，出现你包的详细内容。
3. 然后点击详细内容的Path直接在浏览器下载（.tgz格式压缩包）。
4. 由于这种方法只有package.json跟index.js等你自己创建的文件，而不包含你的包本身的依赖——node\_modules包，所以你需要在包文件夹下手动下载你的依赖包。

### 方法二：npm命令行直接下载。

npm install <your-package-name@版本号> --save

# Node.JS

回顾：

一个网络系统无论多复杂，总包含三个部分：

1、Web浏览器：那七个。

2、服务器，常用的有：阿帕奇，IIS、TomCat

3、数据库：MSSQLSever Oracle MySql

Web开发本质上就是Web文件（HTML）通过浏览器请求Web服务器上的文件，Web服务器上的文件请求数据库服务器上的文件。

注意：

1. 域名：本质上就是将我们难以记忆的IP地址（192.168.0.101:80）翻译成

http//www.xxx.com的形式。

1. 浏览器请求的文件是真实存在于服务器上面的文件，是以HTML形式存在。
2. 同一个时间段内请求一个网站的现象叫做并发，请求的人数称为并发数。（进程和线程）

进程：一个软件运行起来就是一个进程。

线程：是主机cpu的性质。当web服务器接收到一个请求，就自动分配一个线程来专门处理该请求。一个进程里能包括多个线程。

1. Web容器：指的是服务器开辟的文件夹来存放服务器文件。

Node.js是一个由专注于实现高性能Web服务器优化的专家几经挫折后，遇到V8引擎而诞生的项目，它专注于单线程处理所有的请求，而其他（.Net）是单线程处理单个请求。

由于Node.js是一个单线程的运行环境，简单易懂，能让Js运行在服务器端的开发平台，它让Javascript的触角衍生到了服务器，可以与PHP，JSP，Python，Ruby，.NET平起平坐；但是成也单线程，败也单线程。由于它是一个单线程方式，所以这也是它的一个缺点，就是一个程序出现问题，所有的程序都会瘫痪。

Node.js不是一种独立的语言，与PHP、JSP、ASP、Perl、Ruby的语言“及时语言，又是平台”不同，Node.js是使用Javascript进行编程，运行在V8（Js）引擎上。

与PHP、JSP、.NET（都需要运行在服务器程序上：Apache，Nginx，Tomcat，IIS）不同，Node.js跳过了这些http服务器程序，自己不用关联在服务器软件之上，直接运行在V8引擎上。Node.js的许多设计理念与经典架构（LAMP ：linux（操作系统）、Apache（Web服务器）、MySql、PHP）有着很大不同。可以提供强大的伸缩功能（根据需要加减服务器）。Node.js没有运行的服务器，也不需要启动一个服务来运行它，因为Node.js本身就可以充当服务器。

Node.js 是一个基于 Chrome V8 引擎的 JavaScript 运行环境。   
Node.js 使用了一个事件驱动、非阻塞式 I/O 的模型，使其轻量又高效。   
Node.js 的包管理器 npm，是全球最大的开源库生态系统。

传统意义上的服务器请求：

服务器向数据库请求数据成为I/O操作（input与output操作），I/O操作的实质是操作数据库的本地文件。

多线程：在Java、PHP或者.NET等服务器语言中，会为每个客户端连接创建一个新的线程。而 每一个线程需要消耗大约2M内存。也就是说，理论上一个8G的服务器可以同时连接大约4000个用户量。要让Web应用支持更多的用户，就需要增加服务器的数量，而web服务器的硬件成本就当然上升了。CPU读取数据的速度：内存>外存（硬盘）容易造成请求阻塞。

而Node.js不是为每一个客户都创建一个县城，而仅仅使用一个线程，当有用户链接了，就发出一个内部事件，通过非阻塞I/O，事件驱动机制，让Node.js程序宏观上也是并行的。使用Node.js，一个8G内存的服务器，可以同时处理4万个用户连接。

另外，单线程的好处是：操作系统不再创建单线程，销毁时间的开销。

坏处：就是当一个用户造成了线程的崩溃，整个服务器都崩溃了。（宕机）。其他人也崩溃了。

可以解决阻塞问题，它是非阻塞的。

阻塞：当访问数据库取得数据的时候，需要一段没时间，在传统web开发中处理机制，在执行了访问数据库代码之后，整个线程都将暂停起来，等待数据执行的后果才能执行后面的代码，也就是说，I/O阻塞了代码的执行，极大降低了程序的执行效率。

由于Node.js采用的非阻塞I/O机制，因此，在执行数据库的代码之后，将立即转而执行其后面的代码，将数据库返回的处理结果存在一个内存中（回掉函数），从未提高了执行效率。

当某个I/O执行完毕时，将以事件的形式通知没有处理的线程。为了处理I/O，必须有事件循环。不断地检查有没有未完成的事件，依次进行处理。

阻塞模式下，一个线程只能处理一项任务，要想提高线程的吞吐量必须通过多线程。而非阻塞模式下，一个线程永远在执行操作，这个线程的CPU利用率是100%，所以，这是一种特有哲理的解决方案，预期人多但是有汗多人闲着，不如一个人玩命干，往死里干。

事件驱动 event-driven：

在Node，客户端请求服务器连接时，提交表单时，查找数据时等等，都会触发事件的回掉函数。但是执行一个回掉函数的中途就可以转而处理其他事件。这种事件处理机制，成为“事件环”机制。

Node.js（基于V8引擎）底层是C++，底层代码中都是事件队列，回掉函数。只有大神级的工程师才能想到，用一个线程，担负起多个线程的使命。  
Node.js善于处理IO操作，因为Node.js就是为了任务任务调度，如果你的项目用很多CPU计算，就不适合Node.js开发。

Node.js的优势：

窗体顶端

动态语言：开发效率非常高，并有能力构建复杂系统，如ql.io。

性能和I/O负载：Nodejs非常好的解决了IO密集的问题，通过异步IO来实现。

连接的内存开销：每个Node.js进程可以支持超过12万活跃的连接，每个连接消耗大约2K的内存。

操作性：实现了Nodejs对于内存堆栈的监控系统。

窗体顶端

Node.js不适合的领域

　每一种语言或平台都有不擅长领域，对于Nodejs来说最不擅长的领域在于CPU和内存的编程操作。

1. 计算密集型应用，让Javascript和C去拼计算性能，估计是不可能赢的。

2. 内存控制，让Javascript和Java比较复杂数据类型定义，也是很困难的。因为 Javascript的面向对象是基于JSON的，而Java是直接使用内存结构。所以，通过JSON序列化和反序列的过程控制内存，Javascript就已经输了。

3. 大内存的应用，由于V8引擎有内存设计的限制，32位环境中最大堆是1G，64位环境中最大堆也不到2G，如果要一次读入10G数据，对于Nodejs来说也无法实现。

4. 静态服务器，虽然Nodejs的优势在IO密集集应用，但是和Nginx的处理静态资源还是有很大的差距。

5. 不需要异步的应用：比如系统管理，自行化脚本等，还是Python更顺手，Nodejs的异步调用可能会给编程带来一些麻烦。

一个网站里的用户表单收集

考试系统

聊天室

图文直播

提供JSON的API（Angular.js）

Node.js的安装：在win10左下角搜索框内输入CMD，弹出电脑的命令程序输入框，在输入框内输入node -v查看当前Node.js的版本。

Node.js运行：在命令输入框内输入node Node.js文件的绝对路径，回车键入，看看下面有没有出现东西，若什么都没有出现，说明CMD命令成功，Node.js文件运行成功。

Dir：命令行里输入查看文件夹里的所有文件。

搜索：在地址栏中搜索js文件里的地址和端口号。

Node.js不存在根目录的概念，因为它没有没有Web服务器，他就是Web服务器。让Node.js

请求一个静态页面根本不可能，也就是说当你看到一个网站地址为127.0.0.1/div，就不用去想，肯定有个文件名称叫做div。

注意：绝不允许在node.js安装目录下的文件夹安装js文件。

HTTP模块：

Node.js中，将很多的功能，划分成了一个个的Module。大陆书翻译成模块，港台数据称之为模组。这是因为一些程序需要使用fs功能（文件读取功能），有一些不用的，所以为了效率，你用什么，你就require什么。

引包http模块：require("http"); 请求http协议的功能以便在地址栏中能找到。

引包url模块：var url=require（“url”）；这里url是个对象，处理url地址，把url地址拆分成很细的枝节。

queryString专门处理url地址路径的。

var server=http.createServer(function(req,res){

}）

http通过回调函数就是函数体内部去完执行的某项功能，并且函数执行完毕后由参数接收结果。服务器创建了两个对象：req对象和res对象。req对象得到请求并执行，res对象回调给浏览器。

http创建服务器

var url=req.url;声明一个变量来接收request请求文件的地址，指的是域名。这里的地址有讲究。

Req的url属性：域名。

Req是指浏览器请求的所有信息。

Res是指服务器相应的所有信息。

Res.end(“”)响应结束语句，在这句话后面再写就会报错。

Node.js不关心响应状态码，响应状态可以自己设置，请求的文件可以不表示http状态码。

Req里的东西：最关键的就是url的地址，表示请求的url地址。所有通过node.js请求的路径都叫做路由 Routing。都是通过reg.url来实现的。

我们不关心url地址，而是识别这个url地址。

识别url地址，用到两个模块，第一个是url模块，第二个是requirestring

Node.js和其他开发方式不同，它本身不存在web文件夹这个概念，它的请求文件机制叫做路由（Routing）机制：

fs模块：require（“fs”）：请求filesysterm（文件系统）来读取服务器上的文件，只要是文件，都能读取到。./：是指当前同一级的父文件夹。

If（url==“路由地址”）{

fs.readField（“url”，function（err，data）{

}）

}

**1、在if（url==“路由地址”）中，这里的url是路由地址，当html页面没有引入外部js/css文件时，这个路由地址可以随便命名，它只是个指路人；**

**2、当html页面引入外部文件时，这个src地址是html引进外部文件的路由地址，是随便命名的，指向html页面引入的外部css/js文件。**

比如：<script type="text/javascript" src="xxx"></script>

或者<link rel="stylesheet" type="text/css" href="cccc">

**有几个js或css文件，就要写几个else if(){fs.readFile}。**

在fs.readField（“url”，function（err，data）{ }）中，这里的url是需要加载的文件的相对/绝对路径地址，是外部js/css文件的真实地址，是告诉服务器应该在哪个地方找到该文件，再返回给浏览器。在这个地址中，应该把要加载的文件的所属文件夹直接放在C/D/E盘下面。

在Node.js写的服务器中，我们在地址栏中搜索的全部都是html文件的路由地址。

Node.js语法：

Node.js本质上就是学习服务器端的编程。

对于谷歌浏览器，每当访问一个服务器页面时，谷歌浏览器会默认访问一个缓存文件（./favicon.ico）,然后在控制台上打印两次结果。

解决办法：if（url==“./favicon.ico”）{

return ；

}

服务器端的服务器就两个功能（两个socket干了两件事）：

1. 接收请求。
2. 响应请求。

有两个socket（套接字，插座）。

第一个socket就是循环监听并接收浏览器客户端的请求。

While（true）{

Socket.accept();

}

第二个socket处理来自第一个socket的请求，生成一个服务器相应报文，来传递给浏览器客户端。

一：Node.js中的http协议

server.listen(8080,"127.0.0.1");

该端口号理论上是只要电脑上没被占用的端口号都可以被监听。127.0.0.1是域名。

一个 URL 字符串是一个结构化的字符串，它包含多个有意义的组成部分。 当被解析时，会返回一个 URL 对象，它包含每个组成部分作为属性。

* urlObject [<Object>](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object) | [<String>](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures" \l "String_type) 一个 URL 对象（就像 url.parse() 返回的）。 如果是一个字符串，则通过 url.parse() 转换为一个对象。

urlString [<String>](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures" \l "String_type) 要解析的 URL 字符串。

## 在这个网址：

## https://www.hao123.com/?tn=99682755\_hao\_pg中：

### urlObject.pathname

pathname 属性包含URL的整个路径部分。它跟在 host （包括 port）后面，排在 query 或 hash 组成部分的前面且被 ASCII 问号（?）或哈希字符（#）分隔。

### urlObject.search

search 属性包含 URL 的整个查询字符串部分，包括开头的 ASCII 问号字符（?）。

例如：'?query=string'

### urlObject.path

path 属性是一个pathname与search组成部分的串接。

例如：'/p/a/t/h?query=string

## url.parse(urlString[, parseQueryString[, slashesDenoteHost]])

urlString [<String>](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures" \l "String_type) 要解析的 URL 字符串。

slashesDenoteHost [<Boolean>](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures" \l "Boolean_type) 如果为 true，则 // 之后至下一个 / 之前的字符串会被解析作为 host。 例如，//foo/bar 会被解析为 {host: 'foo', pathname: '/bar'} 而不是 {pathname: '//foo/bar'}。 默认为 false。

url.parse() 方法会解析一个 URL 字符串并返回一个 URL 对象。

### urlObject.query

query 属性是不含开头 ASCII 问号（?）的查询字符串，或一个被 [querystring](http://nodejs.cn/api/querystring.html) 模块的 parse() 方法返回的对象。 query 属性是一个字符串还是一个对象是由传入 url.parse() 的 parseQueryString 参数决定的。

例如：'query=string' or {'query': 'string'}

如果返回一个字符串，则不会对查询字符串执行解码。 如果返回一个对象，则键和值都会被解码。

parseQueryString [<Boolean>](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures" \l "Boolean_type) 如果为 true，则 query 属性总会通过 [querystring](http://nodejs.cn/api/querystring.html)模块的 parse() 方法生成一个对象。 如果为 false，则返回的 URL 对象上的 query 属性会是一个未解析、未解码的字符串。 默认为 false。

如果设为 true，则 // 之后至下一个 / 之前的字符串会被解析作为 host。 例如，//foo/bar 会被解析为 {host: 'foo', pathname: '/bar'} 而不是 {pathname: '//foo/bar'}。

## queryString：

Node.js学习就是学习很多功能，这些功能都是一个个的模块（module）。

|  |  |
| --- | --- |
| http模块 | 请求http协议，搭建服务器 |
| url | url模块提供了一些实用函数，用于URL处理与解析。 |
| Fs（fieldsysterm） | read write |
| queryString | 字符串（URL地址）查询 |
| Path（路径） | 文件路径 |
| exports | 暴露 |

非阻塞的Node.js什么意思：

当Node.js里面去执行毁掉函数的时候，Node.js并不会等待回调函数执行完毕之后采取执行回调函数里面的代码，而是依次执行下面的代码，并不等待回调函数执行的代码。可以理解为异步执行。和ajax一样，也是非阻塞的，异步的。

事件环：各种回调函数层层嵌套组成事件环，每执行完一个函数，就立即清除，如果再想使用上一个函数，就要再次声明。

MVC前端开发模式

M - Model（模型）

V - View（视图）

C - Controler（控制器）

高内聚，低耦合：

高内聚：指一个工作模块内的许多文件要集中在一个项目文件中。

低耦合：指模块与模块之间的关系，不要太多联系，低耦合。

在Node.js中，js文件与js文件之间有两种合作模式

var msg=“hallo”;

1. 某一个js文件中，提供了函数，要想供别人使用，只需要使用暴露expoets.msg=msg；
2. 另一个文件描述了一个类。Module.exports=People;

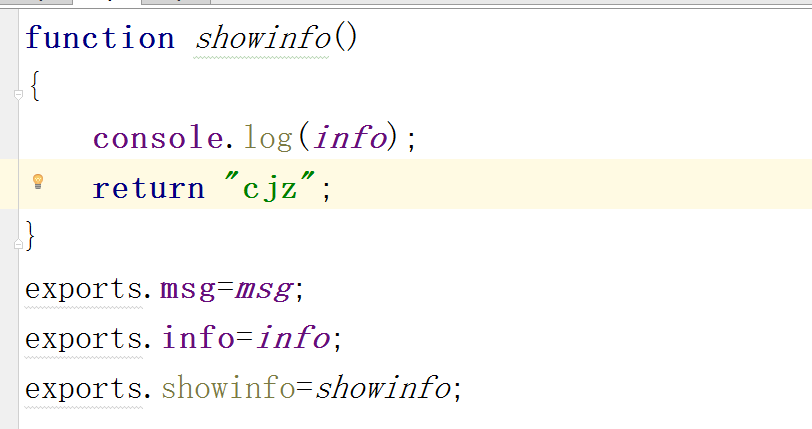
## 二：模块

在划分Node.js中，以模块为单位划分所有功能，并且要有MVC。

狭义的说，每一个JavaScript文件都是一个模块，而多个JavaScript文件之间的引包使用关键字require();.它们共同实现了Node.js的编程。

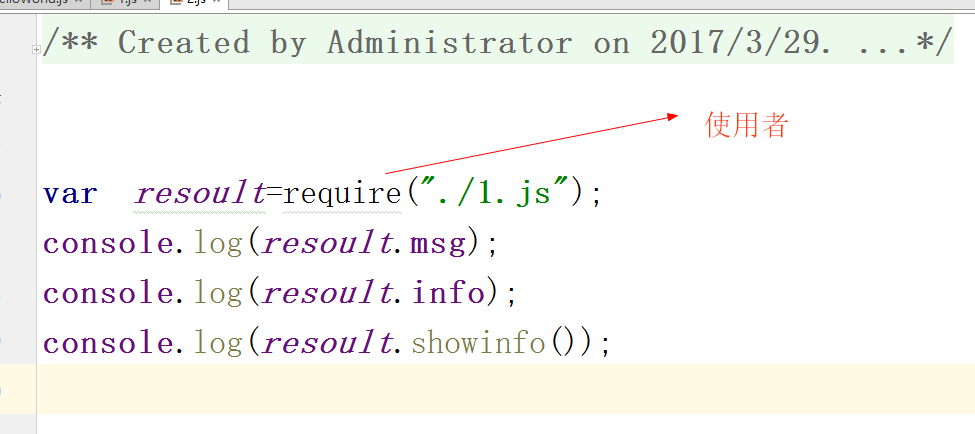
Node.js中，一个JavaScript文件中定义的变量，函数，都只有这个文件内部有效，当需要从此JS文件外部引用这些变量，函数时，必须使用export对象进行暴露，使用者使用require()命令引用这个JS文件。

1. js 定义者



msg这个变量，是一个js文件内部才有作用域的变量。

如果其他想用这个msg变量，那么要用exports暴露。



使用者接受exports对象，也就是说这里的resoult变量，就是1.js中的exports变量。

一个JS文件当中，可以向外exports无数个变量，函数，但是require的时候，仅仅需要一个require这个JS文件一次。使用它的变量，函数的时候，用点(.)即可。所以，无形之中，增加了一个引包。俗称是exports暴露。

Node.js当中，文件与文件之间，就是一个个的exports和require（）构建成为网络的。不是靠HTML a标签进行链接的。

可以将一个文件想象成一个Class。

也就是说，js文件与js文件之间有两种合作的模式

1. 某一个js文件中，提供了函数，供别人使用，只需要暴露函数就exports.msg=msg;
2. 某一个文件，描述了一个类。Module.exports=People

如果在require命令中，写var resoult=require(“1.js”);

上面写法就是错的，不是相对路径，而是一个特殊路径。

正确的如下：var resoult=require(“./1.js”);

var resoult=require(“1.js”);

那么Node.js 将该文件视为node\_Modules目录下的一个文件。

在1.js文件里，

Module是一个类包，

可以创建对象，也可以创建属性。

Module.exports=对象；

而exports.msg=msg;这个exports点出来的是属性。

在2.js文件中，

Require（“./1.js”）;引进的是1.js的类包，等于把1.js中的module引进来了。

注意：绝不允许在node.js安装目录下的文件夹安装其他js文件包。因为默认的是安装目录下node.module这个文件夹，然后默认该文件夹下的js文件，其他的不认。

node\_modules本身也是有个查找顺序，树形查找。自下而上的查找。在当前文件夹找不到，就找上一级，上以及找不到，就找爷爷级，依次往上找，能找64级。如果这个文件你不想费劲，就把他的地址路径配置在全局环境变量（path）里面。

require（“bar”）如果前不带点斜杠，后不带后缀，那么这个bar就是个文件夹。那么就要在bar文件夹下建一个index.js文件，默认找的是node\_modules/bar/index.js。也就是说，node\_modules与index.js这两个是确定好的，而bar这个文件夹可以自定。

如果不想默认请求index.js想require bar文件夹下的其他js文件（比如01.js），那么就要新建一个json文件parkage.json（确定的）,在里面通过“main”：“01.js”启动01.js。

下载npm 文件夹：

Cd：C：\node;

①：npm install xxx

②：下载下来，放到node\_module文件夹下面。

## MD5

计算机世界中，有一些信息数据是需要加密的，让别人看不懂/看不到。

Message Digest Algorithm MD5（中文名为[消息摘要算法](http://baike.baidu.com/item/%E6%B6%88%E6%81%AF%E6%91%98%E8%A6%81%E7%AE%97%E6%B3%95" \t "http://baike.baidu.com/_blank)第五版）为计算机安全领域广泛使用的一种散列函数，用以提供消息的完整性保护。

**MD5**即Message-Digest Algorithm 5（信息-摘要算法5），用于确保信息传输完整一致。是计算机广泛使用的杂凑算法之一（又译[摘要算法](http://baike.baidu.com/item/%E6%91%98%E8%A6%81%E7%AE%97%E6%B3%95" \t "http://baike.baidu.com/_blank)、[哈希算法](http://baike.baidu.com/item/%E5%93%88%E5%B8%8C%E7%AE%97%E6%B3%95" \t "http://baike.baidu.com/_blank)），主流编程语言普遍已有MD5实现。将数据（如汉字）运算为另一固定长度值，是杂凑算法的基础原理，MD5的前身有MD2、[MD3](http://baike.baidu.com/item/MD3" \t "http://baike.baidu.com/_blank)和[MD4](http://baike.baidu.com/item/MD4" \t "http://baike.baidu.com/_blank)。

MD5算法具有以下特点：

1、压缩性：任意长度的数据，算出的MD5值长度都是固定的。

2、容易计算：从原数据计算出MD5值很容易。

3、抗修改性：对原数据进行任何改动，哪怕只修改1个字节，所得到的MD5值都有很大区别。

4、强抗碰撞：已知原数据和其MD5值，想找到一个具有相同MD5值的数据（即伪造数据）是非常困难的。

MD5的作用是让大容量信息在用[数字签名](http://baike.baidu.com/item/%E6%95%B0%E5%AD%97%E7%AD%BE%E5%90%8D" \t "http://baike.baidu.com/_blank)软件签署私人[密钥](http://baike.baidu.com/item/%E5%AF%86%E9%92%A5" \t "http://baike.baidu.com/_blank)前被"[压缩](http://baike.baidu.com/item/%E5%8E%8B%E7%BC%A9/13032501" \t "http://baike.baidu.com/_blank)"成一种保密的格式（就是把一个任意长度的字节串变换成一定长的[十六进制](http://baike.baidu.com/item/%E5%8D%81%E5%85%AD%E8%BF%9B%E5%88%B6" \t "http://baike.baidu.com/_blank)数字串）。

MD5加密后的字符串是根据什么生成的？Mac随机地址，GUID全球唯一标识符，它就是根据你自己的电脑的Mac地址加上当前电脑的时间戳再加上硬件固化电器的地址。

## Nodejs处理Ajax的Post请求：

一般我们向服务器提交post数据的时候，poet数据都是写在form表单里的:

<form action="http：//127.0.0.1:8080" method="post" enctype="multipart/form-data" >

<p>姓名： <input type="text" name="uName" id="uName" /></p>

<p>密码： <input type="password" name="uPass" id="uPass" /></p>

<p>

姓别： 男<input type="radio" id="man" name="man" value="男"/>

女<input type="radio" name="wonan"id="woman" value="女"/>

保密<input type="radio" name="secrecy"id="secrecy" value="保密" />

</p>

<p> <input type="button" id="btn" value="登陆" /></p>

</form>

而我们在ajax里写post请求的时候:

除了以前学过的ajax请求的那一套，还要注意：

oajax.open("POST","http://127.0.0.1:8080",true);

oajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");当用原生js写的时候：

*var* data="uName="+uName+"&uPass="+uPass+"&sex="+sex;

注意：这里的&连字符必须加上，因为不加上的话，node.js会把uName后面的字符串都认为是uName里的值,上面的数据格式很严格必须为

“key1=”+value1+“&key2=”+value2+“&key3=”+value3+。。。。

一点也不能错，错一点服务器接收就会发生错误。将来这个字符串会被转化为node.js对象，格式为：{key1：vaue1，key2：value2，key3：value3...}

或者为：

ajax请求时：

var data={key1：vaue1，key2：value2，key3：value3...}，

但是dataType必须是“json”。

例如：{ uName: 'asd', uPass: '123', sex: '女' }，这是在node.js控制台打印出来的对象。

Oajax.Send(data).//向浏览器发送数据

Oajax.onreadystatechange=function(){

Oajax.statute……

}

oajax.responseText:是指从服务器接收到的**字符串**。

JSON.parse()方法：是将服务器发来的字符串转化为JSON格式的js对象（键值对），然后通过对象及其属性将每个对象和其值分发到dom中去。

而JSON.stringfy()方法：将对象转化为json字符串。

判断一个对象是否为空对象：

Var c={};

If(JSON.stringfy(c)==“{}”){

Console.log(“c为空对象”);

}else{

Console.log(“c为非空对象”);

}

接收Post请求的**字符串**：

**Post请求：post请求本质上是把浏览器传递给服务器的信息(data)做成了一个一个的小包。每一个小包是用一个参数chunk接收，参数chunk就是负责就收请求数据的拆分小包，是以buffer二进制流的形式传递。**

**req.addListener()方法来监听数据请求:**

**var str;**

**req.addListener(“data”,function(chunk)){  
str+=chunk;**

**}**

处理POST请求：

1.Node.js处理post请求是通过querystring.parse()方法来将ajax方法的post请求所发送的json字符串转化为node.js对象;

2.通过JSON.stringfy()方法将node.js对象转化为json字符串返回给浏览器。

3.util.inspect() 方法返回 object 的字符串表示，主要用于调试。

req.addListener("end",*function*(data){//这个data可有可无。

res.writeHead(200,{"Content-type":'application/json'})

*console*.log(reqStr);

第一种传输方式：

res.writeHead(200,{"Content-type":"text/html;charset=utf-8"})

*var* reqObj=querystring.parse(reqStr);

*console*.log(reqObj);

Obj.uPass+"性别："+reqObj.sex;

*var* resStr="姓名："+reqObj.uName+"密码：";

这种方法传给浏览器的是普通的字符串，结果为：

name:sdsf,password:123,sex:man，是不带大括号的普通字符串。

或者是第二种传输方式：

res.writeHead(200,{"Content-type":'application/json'})

//设置返回数据的响应头，规定传输的数据格式

var resStr=JSON.stringify({"姓名：":reqObj.uName,"密码：":reqObj.uPass,"性别：":reqObj.sex});

res.end(resStr);

传给浏览器的结果为：{"uName":"asd","uPass":"sdf","sex":"man"}，浏览器接收结果为：

*{uName: "ewer", uPass: "asd", sex: "保密"}*

sex:"保密"

uName:"ewer"

uPass:"asd"

\_\_proto\_\_:Object

是一个对象。

推荐使用第二种，第二种是正宗的json格式数据传输。这种方法的好处是返回正宗json字符串，浏览器接收后可用JSON.parse(str)来进行数据/对象转换，得到想要的东西。

## querystring.parse(str[, sep[, eq[, options]]])

str [<string>](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures" \l "String_type) 要解析的 URL 查询字符串。

sep [<string>](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures" \l "String_type) 用于界定查询字符串中的键值对的子字符串。默认为 '&'。

eq [<string>](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures" \l "String_type) 用于界定查询字符串中的键与值的子字符串。默认为 '='。

options [<Object>](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object)

decodeURIComponent [<Function>](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function) 当解码查询字符串中百分号编码的字符时使用的函数。默认为 querystring.unescape()。

maxKeys [<number>](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures" \l "Number_type) 指定要解析的键的最大数量。默认为 1000。指定为 0 则移除键数的限制。

querystring.parse() 方法能把一个 URL 查询字符串（str）解析成一个键值对的集合。

例子，查询字符串 'foo=bar&abc=xyz&abc=123' 被解析成：

{

foo: 'bar',

abc: ['xyz', '123']}

注意：querystring.parse() 方法返回的对象不继承自 JavaScript 的 Object。 这意味着典型的 Object 方法如 obj.toString()、obj.hasOwnProperty() 等没有被定义且无法使用。

默认情况下，查询字符串中的百分号编码的字符会被认为使用了 UTF-8 编码。 如果使用的是另一种字符编码，则 decodeURIComponent 选项需要被指定

路径问题：

1. 在同一个文件夹下，只要在一个js文件中引入另一个js文件(“./2.js”) 或通过fs.readFile(“./1.txt”)引入txt文件，那么被引入的那个js文件中的代码自动执行.

如果用require引入的路径问题，我闷可以直接使用相对路径。因为require是从自己所在的文件夹开始找的相对路径。

fs.readFile（）等其他模块方法引入的其他类型文件也可以用相对路径，但这个相对路径都是相对于cmd命令光标的路径，闲麻烦就用绝对路径。Node.js为我们提供了可以使用\_\_dirname变量获取当前模块文件所在目录的完整绝对路径的方法：\_\_dirname.(两个下划线后跟dirname)+“/文件名”。

## fs.readFile(\_\_dirname+“/文件名”);

以后用到模块路径的时候都写\_\_dirname+“/文件名”；

这里有一个小技巧：在性别的那一栏中，只要把name属性设为一样，那么就只能选一个性别了，否则的话男和女都能选。

# Express框架

Express框架是后台的Node框架，所以和jQuery、zepto、yui、bootstrap都不一个东西。

Express在后台的受欢迎的程度，和jQuery一样，就是企业的事实上的标准。

● 原生Node开发，会发现有很多问题。比如：

■ 呈递静态页面很不方便，需要处理每个HTTP请求，还要考虑304问题

■ 路由处理代码不直观清晰，需要写很多正则表达式和字符串函数

■ 不能集中精力写业务，要考虑很多其他的东西

我们自己可以把第一天的作业，就是那个静态文件服务给封装成为模块。封装的越多，就自己做出了类似Express的东西。

● EXPRESS的哲学是在你的想法和服务器之间充当薄薄的一层。这并不意味着他不够健壮，或者没有足够的有用特性，而是尽量少干预你，让你充分表达自己的思想，同时提供一些有用的东西。

英语官网：http://expressjs.com/

中文官网：<http://www.expressjs.com.cn/>

整体感知，Express框架。

安装Express框架，就是使用npm的命令。

|  |
| --- |
| 1. npm install **--save** express |

--save参数，表示自动修改package.json文件，自动添加依赖项。

路由能力：

|  |
| --- |
| 1. var express = require("express"); 2. var app = express(); 3. app.get("/",function(req,res){ 4. res.send("你好"); 5. }); 6. app.get("/haha",function(req,res){ 7. res.send("这是haha页面，哈哈哈哈哈哈"); 8. }); 9. app.get(/^\/student\/([\d]{10})$/,function(req,res){ 10. res.send("学生信息，学号" + req.params[0]); 11. }); 12. app.get("/teacher/:gonghao",function(req,res){ 13. res.send("老师信息，工号" + req.params.gonghao); 14. }); 15. app.listen(3000); |

## 利用 Express 托管静态文件

### 1.返回图片、svg、html等静态文件（String类型）也就是生成一个网络链接

app.get('/svg', (req, res) => {

res.header('Access-Control-Allow-Origin', '\*');

res.header('Content-Type', 'application/json;charset=utf-8');

res.sendfile('./src/assets/wenzhou-section-v2.svg');

});

### 2.返回渲染好的html页面

用ejs模板来渲染html文件：

import ejs from 'ejs';

const app = express();

// ./dist是运行命令所在文件夹与dist文件夹的相对路径

app.set('views', './dist');

app.engine('html', ejs.renderFile);

app.set('view engine', 'html');

// 返回dist/index.html

app.get('/', (req, res) => {

res.header('Access-Control-Allow-Origin', '\*');

res.header('Content-Type', 'text/html');

res.render('index');

});

## 二、路由

当用get请求访问一个网址的时候，做什么事情：

|  |
| --- |
| 1. app.get("网址",function(req,res){ 2. }); |

当用post访问一个网址的时候，做什么事情：

|  |
| --- |
| 1. app.post("网址",function(req,res){ 2. }); |

如果想处理这个网址的任何method的请求，那么写all

|  |
| --- |
| 1. app.all("/",function(){ 2. }); |

这里的网址，不分大小写，也就是说，你路由是

|  |
| --- |
| 1. app.get("/AAb",function(req,res){ 2. res.send("你好"); 3. }); |

实际上小写的访问也行。

所有的GET参数，? 后面的都已经被忽略。 锚点#也被忽略

你路由到/a ， 实际/a?id=2&sex=nan 也能被处理。

正则表达式可以被使用。正则表达式中，未知部分用圆括号分组，然后可以用req.params[0]、[1]得到。

req.params类数组对象

|  |
| --- |
| 1. app.get(**/^\/student\/([\d]{10})$/**,function(req,res){ 2. res.send("学生信息，学号" + **req.params[0]**); 3. }); |

冒号是更推荐的写法。

|  |
| --- |
| 1. app.get("/student/:id",function(req,res){ 2. var id = req.params["id"]; 3. var reg= /^[\d]{6}$/; //正则验证 4. if(reg.test(id)){ 5. res.send(id); 6. }else{ 7. res.send("请检查格式"); 8. } 9. }); |

表单可以自己提交到自己上。

|  |
| --- |
| 1. app.get("/",function(req,res){ 2. res.render("form"); 3. }); 4. app.post("/",function(req,res){ 5. //将数据添加进入数据库 6. res.send("成功"); 7. }); |

适合进行 RESTful路由设计。简单说，就是一个路径，但是http method不同，对这个页面的使用也不同。

/student/345345

get 读取学生信息

add 添加学生信息

delete 删除学生新

## 三、中间件

如果我的的get、post回调函数中，没有next参数，那么就匹配上第一个路由，就不会往下匹配了。

如果想往下匹配的话，那么需要写**next()**

|  |
| --- |
| 1. app.get("/",function(req,res,**next**){ 2. console.log("1"); 3. **next();** 4. }); 5. app.get("/",function(req,res){ 6. console.log("2"); 7. }); |

下面两个路由，感觉没有关系：

|  |
| --- |
| 1. app.get("**/:username/:id**",function(req,res){ 2. console.log("1"); 3. res.send("用户信息" + req.params.username); 4. }); 5. app.get("**/admin/login**",function(req,res){ 6. console.log("2"); 7. res.send("管理员登录"); 8. }); |

但是实际上冲突了，因为admin可以当做用户名 login可以当做id。

解决方法1：交换位置。 也就是说，express中所有的路由（中间件）的顺序至关重要。

匹配上第一个，就不会往下匹配了。 具体的往上写，抽象的往下写。

1. app.get("/admin/login",function(req,res){
2. console.log("2");
3. res.send("管理员登录");
4. });
5. app.get("/:username/:id",function(req,res){
6. console.log("1");
7. res.send("用户信息" + req.params.username);

});

解决方法2：

app.get("/:username/:id",function(req,res,next){

var username = req.params.username;

//检索数据库，如果username不存在，那么next()

if(检索数据库){

console.log("1");

res.send("用户信息");

}else{

next();

}

});

app.get("/admin/login",function(req,res){

console.log("2");

res.send("管理员登录");

});

路由get、post这些东西，就是中间件，中间件讲究顺序，匹配上第一个之后，就不会往后匹配了。next函数才能够继续往后匹配。

app.use()也是一个中间件。与get、post不同的是，他的网址不是精确匹配的。而是能够有小文件夹拓展的。

比如网址： <http://127.0.0.1:3000/admin/aa/bb/cc/dd>

|  |
| --- |
|  |

如果写一个/

1. //当你不写路径的时候，实际上就相当于"/"，就是所有网址
2. app.use(function(req,res,next){
3. console.log(new Date());
4. next();

});

|  |
| --- |
|  |

app.use()就给了我们增加一些特定功能的便利场所。

实际上app.use()的东西，基本上都从第三方能得到。

● 大多数情况下，渲染内容用res.render()，将会根据views中的模板文件进行渲染。如果不想使用views文件夹，想自己设置文件夹名字，那么app.set("views","aaaa");

● 如果想写一个快速测试页，当然可以使用res.send()。这个函数将根据内容，自动帮我们设置了Content-Type头部和200状态码。send()只能用一次，和end一样。和end不一样在哪里？能够自动设置MIME类型。

● 如果想使用不同的状态码，可以：

res.status(404).send('Sorry, we cannot find that!');

● 如果想使用不同的Content-Type，可以：

res.set('Content-Type', 'text/html');

## **四、GET请求和POST请求的参数**

● GET请求的参数在URL中，在原生Node中，需要使用url模块来识别参数字符串。在Express中，不需要使用url模块了。可以直接使用req.query对象。

● 当数据是以formData的形式传递到服务器时，POST携带的请求参数在express中不能直接获得，必须使用body-parser模块。使用后，将可以用req.body得到参数。但是如果表单中含有文件上传，那么还是需要使用formidable模块。

● 当数据是以Request Payload或者FormData的方式传递给后台时的情况下：Request Payload方式是以“流“”的方式出入到后台，需要监听data事件来获取完整的数据。

var str="";

req.on("data",function(chunk){

str+=chunk;

})

上面的方式对chunk进行转化。然后用:

req.on('end',function(){

let pramas = JSON.parse(str);

Logger.info(pramas);

//......请求数据库

})；

接收GET的参数：

app.get('/query/allDevice', (req, res) => {

Logger.info(req.query);

//连接数据库

.......

});

接收POST的参数：

app.post('/query/station', (req, res) => {

let str = '';

let pramas = {};

req.on('data', function(dt) { str += dt; });

req.on('end',function(){

Logger.info(str);//是一个JSON类型的字符串,它就是我们post请求的JSON字符串。这个很重要，这个很重要，这个很重要。

pramas = JSON.parse(str);//将字符串JSON化，转化为Object

// 拿着从页面获取的数据连接数据库

dataStore.queryStation(pramas.lineId)

.then((result) => {

logger.info(result);

res.header('Access-Control-Allow-Origin', '\*');

res.header('Content-Type', 'application/json;charset=utf-8');

res.json(result);

})

.catch((err) => {

logger.info('error',err.message);

});

});

});

## 返回静态资源

res.sendfile(‘path’);

这里的path指的是node-sever命令行运行的文件夹与静态资源文件夹的相对路径。

# 跨域问题

跨域问题本身属于ajax请求的问题，是服务器开发人员解决的问题。可以使用node.js解决。

在ajax应用中，Ajaxi请求的地址必须是服务器上本地或者是同一个服务器下面的URl地址。简单说就是请求同一个域名下的URl地址。

特别注意两点：

1. 如果是协议和端口号造成的跨域问题，那么是无法解决的。
2. 在跨域问题上，域仅仅是用过URL的首部来识别的。而不会去判断两个域名是否都在同一个IP地址上面。URl的首部指的是window.location.protocal+window.location.host,理解为domin,protocols and ports must mach。

跨域问题的解决

Ajax请求请求的是同域请求，默认情况下只能访问包含在它同一页面中的资源。这种策略主要是为了安全。但是，为了实现跨域请求，在某些浏览器应用程序中也是至关重要的。

CORS（Cross-Origin-Recourse-Sharing，跨域资源共享），CORS的本质是一个自定义的HTTP头部让浏览器域服务器通信，从而请求定向是应该成功的，还是失败的。

比如说一个简单的额使用GET或者POST请求，它没有自定义的头部，Content-type：text/plain.在发送跨域请求的时候，我们需要动态添加一个Orign它包含了请求页面的源信息（协议：域名：端口号），以便服务器根据这个头部来决定是不是响应。

Orign:htttp://www.newscape.com

如果服务器认可了，那么久返回，不认可就不返回。

如果服务器接受了这个请求，那么就应该在Access-Control-Allow-Origin头部中回发相同的源信息。

Access-Control-Allow-Origin：

htttp://www.newscape.com

如果没有上述这个信息，就证明浏览器不支持跨域，正常情况下，设置了Origin之后，都会接收请求，但是请求和响应Cookie无用。

一：IE浏览器对 CORS的实现：（并没什么卵用）

微软在IE8以上才开始提供跨于技术。XDR对象来支持，这个对象和XHR对象很相似。但能实现安全跨域。

缺点：服务器如果收到了请求，那么只能发送文本文件，没有办法确定响应头的状态码，而且只有响应成功有效才会触发onload事件，

Cookie问题

Cookie是浏览器保存状态的一门技术。

1：浏览器向服务器请求登录，登陆成功之后，将在服务器端（js文件）产生一个Cookie，本质上是一个字符串，形式是key：value形式，存放用户名或者登录邮箱（不包含密码，因为会被其他人看到）

2：cookie返回给浏览器，浏览器接收这个cookie并且下一次请求的时候就携带着这个cookie来请求。

3：到了服务器端之后，服务器去检查有没有cookie，如果有，那么返回登录者信息。

内存cookie与硬盘cookie

内存级别cookie就是当浏览器的内存，当浏览器关闭的时候，cookie就没有了。

硬盘cookie就是白存在本地硬盘中的，浏览器关闭不会自动删除。

跨域的方法：

1. 通过jsonp跨域

Jsonp的跨域请求原理：JSONP的原理是

通过jsonp数据来实现，说白了，jsonp就是js可执行文件，由于<script>和<link>,<img>加载的文件不受同源限制，所以可以通过设置src和href来加载相应的跨域json或者xml数据，但是这种方式需要服务端相配合（服务端需要取出请求字符串中的回调函数名，并且返回给客户端）。jsonp的格式为：回调函数名（jsonData）。

     jQuery中的$.getJSON()默认实现了这种方式，只需要在url？callback=？即可，当然也可以手动指定回调函数名。

1. 通过修改document.domain来跨子域
2. 使用window.name来进行跨域
3. 使用HTML5中新引进的window.postMessage方法来跨域传送数据

# Webpack

Webpack是我们创模块化Web项目的打包与管理工具。目前我用的是Webpack4.0+。配置webpack的步骤：

1. 新建一个文件夹（myProject）
2. cd进入到该文件夹。打开powershall窗口输入命令 cnpm init
3. 然后编辑一些项目的名字、作者、版本、介绍等。
4. Webpack.config.js

从 webpack v4.0.0 开始，可以不用引入一个配置文件。然而，webpack 仍然还是[高度可配置的](https://www.webpackjs.com/configuration)。在开始前你需要先理解四个核心概念：

入口(entry)、输出(output)、loader转换器和插件(plugins)

### package.json

对于每一个Web项目来说，要想实现模块化开发与生产，都要有一个package.json，我们项目的所有依赖包，包括启动运行命令和打包命令都是在package.json里统一管理的。Webpack项目也不例外。创建一个文件夹作为项目的根目录，然后用windows的cmd指令进入命令行进入该文件夹，然后在确保安装有npm之后用npm命令行创建。创建package.json的npm指令为：npm init。

打包与运行的script命令如下：

"scripts": {

"build": "webpack",

"start": "./node\_modules/.bin/webpack-dev-server --devtool source-map --mode development",

"watch": "webpack --watch --mode development",

"test": "echo \"Error: no test specified\" && exit 1"

},

### 入口(entry)

入口起点(entry)指示 webpack 应该使用哪个js来作为构建其内部依赖图的入口文件。

可以通过在 [webpack 配置](https://www.webpackjs.com/configuration)中配置 entry 属性，来指定一个入口起点（或多个入口起点）。默认值为 ./src。

接下来我们看一个 entry 配置的最简单例子：

module.exports = {

entry: {index: './index.js',},

};

### 出口(outpot)

出口规定了打包出来的js的文件名与文件夹地址。

output: {

filename: 'index.js',

path: path.resolve(\_\_dirname, 'dist'),

publicPath:'/',

},

publicPath用于指定输出解析文件的目录，url 相对于 HTML 页面

### Loader加载器

loader 让 webpack 能够去处理那些非 JavaScript 文件（webpack 自身只理解 JavaScript）。loader 可以将所有类型的文件转换为 webpack 能够处理的有效[模块](https://www.webpackjs.com/concepts/modules)，然后你就可以利用 webpack 的打包能力，对它们进行处理。

本质上，webpack loader 将所有类型的文件，转换为应用程序的依赖图（和最终的 bundle）可以直接引用的模块。

注意，loader 能够 import 导入任何类型的模块（例如 .css 文件），这是 webpack 特有的功能，其他打包程序或任务执行器的可能并不支持。我们认为这种语言扩展是有很必要的，因为这可以使开发人员创建出更准确的依赖关系图。

Webpack常用的webloader有：

babel-loader:js编译器-babel

eslint-loader：代码规范制度

css-loader,lessloader,style-loader,sass-loader处理css,less,style等。

url-loader：处理image图片。在react与babel里会有详细的loader代码，所以在这里不详细列出了。

下面是一个处理url-loader的例子：

{

test: /\.(png|jpg|jpng|eot|ttf)$/,

loader: 'url-loader?limit=8192&name=src/images/[name].[ext]',

},

### 插件plugin

loader 被用于转换某些类型的模块，而插件则可以用于执行范围更广的任务。插件的范围包括，从打包优化和压缩，一直到重新定义环境中的变量。[插件接口](https://www.webpackjs.com/api/plugins)功能极其强大，可以用来处理各种各样的任务。

想要使用一个插件，你只需要 require() 它，然后把它添加到 plugins 数组中。多数插件可以通过选项(option)自定义。你也可以在一个配置文件中因为不同目的而多次使用同一个插件，这时需要通过使用 new 操作符来创建它的一个实例。

const HtmlWebpackPlugin = require('html-webpack-plugin'); // 通过 npm 安装

const CleanWebpackPlugin = require('clean-webpack-plugin');

const ManifestPlugin = require('webpack-manifest-plugin');

const webpack = require('webpack'); // 用于访问内置插件

module.exports = {

module: {

rules: [

{ test: /\.txt$/, use: 'raw-loader' }

]

},

plugins: [

new HtmlWebpackPlugin({

title: '杭州',

template: 'index.html',

inject: true,

chunks: ['index'],

}),

new CleanWebpackPlugin(['dist']),

new ManifestPlugin(),

new webpack.HotModuleReplacementPlugin(),

]};

### 模式mode

通过选择 development 或 production 之中的一个，来设置 mode 参数，你可以启用相应模式下的 webpack 内置的优化

module.exports = {

mode: 'production'

};

### 创建独立sever

Web项目的开发模式中，前后端分离就是通过在web端配置server来单独跑页面，，来请求数据库server端的数据。

通过来自 [webpack-dev-server](https://github.com/webpack/webpack-dev-server) 的这些选项，能够用多种方式改变其行为。这里有一个简单的例子，所有来自 dist/ 目录的文件都做 gzip 压缩和提供为服务：

devtool: 'source-map',

watch: true,

devServer: {

clientLogLevel: 'error',

contentBase: path.resolve(\_\_dirname, 'dist/'),//与output保持同步

inline: true,

hotOnly: true,

historyApiFallback: true,//页面404时转到index.html

overlay: {

warnings: false,

errors: true,

},

port: 8082,//配置端口号

},

historyApiFallback: true是用于当React-Router是BrowserRouter的时候，浏览器会找寻devServer的真实路径，但是Router确实虚拟路径，所以当也拿刷新时，浏览器找不不到页面会报404。该设置是为了当浏览器找不到页面时，自动返回index.html。

# React.Js

React 是一个用于构建用户界面的 javascript 库，与agular、vue一样称为最火的前端三大框架。

React主要用于构建UI，很多人认为 React 是 MVC 中的 V（视图）。

React 起源于 Facebook 的内部项目，用来架设 Instagram 的网站，并于 2013 年 5 月开源。

React 拥有较高的性能，代码逻辑非常简单，越来越多的人已开始关注和使用它。

## React 特点

1.声明式设计 −React采用声明范式，可以轻松描述应用。

2.高效 −React通过对DOM的模拟，最大限度地减少与DOM的交互。

3.灵活 −React可以与已知的库或框架很好地配合。

4.JSX − JSX 是 JavaScript 语法的扩展。React 开发不一定使用 JSX ，但我们建议使用它。

5.组件 − 通过 React 构建组件，使得代码更加容易得到复用，能够很好的应用在大项目的开发中。

6.单向响应的数据流 − React 实现了单向响应的数据流，从而减少了重复代码，这也是它为什么比传统数据绑定更简单。

## React注释

汉语注释：//汉语注释

代码注释：{/\* 注释内容 \*/}，可多行，可单行

## ReactDOM.render()

ReactDOM.render 是 React 的最基本方法，用于将模板转为 HTML 语言，并插入指定的 DOM 节点。

ReactDOM.render(

<h1>Hello, world!</h1>,

document.getElementById('example'));

## React JSX语法

React 使用 JSX 来替代常规的 JavaScript。JSX 是一个看起来很像 XML 的 JavaScript 语法扩展。我们不需要一定使用 JSX，但它有以下优点：

1、JSX 执行更快，因为它在编译为 JavaScript 代码后进行了优化。

2、它是类型安全的，在编译过程中就能发现错误。

3、使用 JSX 编写模板更加简单快速。

我们可以在以上代码中嵌套多个 HTML 标签，需要使用一个 div 元素包裹它，实例中的 p 元素添加了自定义属性 **data-myattribute**，添加自定义属性需要使用 **data-** 前缀。

## JavaScript 表达式

我们可以在 JSX 中使用 JavaScript 表达式。表达式写在花括号 **{}** 中。实例如下：

ReactDOM.render(

<div>

<h1>{1+1}</h1>

</div> ,

document.getElementById('example')

);

在 JSX 中不能使用 **if else** 语句，但可以使用 **conditional (三元运算)** 表达式来替代。以下实例中如果变量 **i** 等于 **1** 浏览器将输出 **true**, 如果修改 i 的值，则会输出 **false**.

ReactDOM.render(

<div>

<h1>{i == 1 ? 'True!' : 'False'}</h1>

</div> ,

document.getElementById('example')

);

## 样式

React 推荐使用内联样式。我们可以使用 **camelCase** 语法来设置内联样式. React 会在指定元素数字后自动添加 **px** 。以下实例演示了为 **h1** 元素添加 **myStyle** 内联样式：

var myStyle = {

fontSize: 100,

color: '#FF0000'

};

ReactDOM.render(

<h1 style = {myStyle}>菜鸟教程</h1>, document.getElementById('example')

);

## 数组

JSX 允许在模板中插入数组，数组会自动展开所有成员。

## this 指向

箭头函数里的 this 指向声明函数的地方，也就是组件内，所以不用 bind(this)  
普通函数里 this 指向调用它的地方，在没有bind 的时候指向函数，而不指向组件，所以要想让 this 指向组件必须使用 bind()

在类的实例上面调用方法，其实就是调用原型上的方法。

class B {}let b = new B();

b.constructor === B.prototype.constructor // true

prototype对象的constructor属性，直接指向“类”的本身，这与 ES5 的行为是一致的。  
Point.prototype.constructor === Point // true

## constructor() 方法

constructor方法是类的默认方法，通过new命令生成对象实例时，自动调用该方法。一个类必须有constructor方法，如果没有显式定义，一个空的constructor方法会被默认添加。  
所有关于组件自身的状态的初始化工作都会放在 constructor 里面去做。

## Props：

props是property的缩写，可以理解为HTML标签的attribute。

不可以使用this.props直接修改props，因为props是只读的，props是用于整个组件树中传递数据和配置。

在当前组件中访问 props，通过this.props.属性名 获取。props属性是父组件控制子组件的单向数据流传输的关键

## super()

组件间的沟通方式（数据传递）

父组件更新组件状态 ——> props ——> 子组件更新

子组件更新父组件状态 ：需要父组件通过 props 传递回调函数 ——> 子组件调用触发

# React 组件

React 允许将代码封装成组件（component），然后像插入普通 HTML 标签一样，在网页中插入这个组件。React.createClass 方法就用于生成一个组件类。

var HelloMessage = React.createClass({

render: function() {

return <h1>Hello World！</h1>;

}

});

ReactDOM.render(

<HelloMessage />,

document.getElementById('example') );

React.createClass 方法用于生成一个组件类 HelloMessage。

<HelloMessage /> 实例组件类并输出信息。

注意，原生 HTML 元素名以小写字母开头，而自定义的 React 类名以大写字母开头，比如 HelloMessage 不能写成 helloMessage。除此之外还需要注意组件类只能包含一个顶层标签，否则也会报错。

## this.props.children

this.props 对象的属性与组件的属性一一对应，但是有一个例外，就是 this.props.children 属性。它表示组件的所有子节点。

var NotesList = React.createClass({

render: function() {

return (

<ol>

{

React.Children.map(this.props.children, function (child) {

return <li>{child}</li>;

})

}

</ol>

);

}});

ReactDOM.render(

<NotesList>

<span>hello</span>

<span>world</span>

</NotesList>,

document.body);

## React PropTypes

组件的属性可以接受任意值，字符串、对象、函数等等都可以。有时，我们需要一种机制，验证别人使用组件时，提供的参数是否符合要求。

组件类的PropTypes属性，就是用来验证组件实例的属性是否符合要求

state 和 props 主要的区别在于 **props** 是不可变的，而 state 可以根据与用户交互来改变。这就是为什么有些容器组件需要定义 state 来更新和修改数据。 而子组件只能通过 props 来传递数据。

var HelloMessage = React.createClass({

render: function() {

return <h1>Hello {this.props.name}</h1>;

}

});

ReactDOM.render(

<HelloMessage name="Runoob" />, document.getElementById('example')

)

注意：在render属性中return出来的html标签，要么不加括号，要么必须用小括号（）包裹起来。

## getDefaultProps() 方法

通过 getDefaultProps() 方法为 props 设置默认值。

var HelloMessage = React.createClass({

getDefaultProps: function() {

return { name: 'Runoob' };

},

render: function() {

return <h1>Hello {this.props.name}</h1>;

}

});

ReactDOM.render( <HelloMessage />, document.getElementById('example') );

## React State(状态)

React 把组件看成是一个状态机（State Machines）。通过与用户的交互，实现不同状态，然后渲染 UI，让用户界面和数据保持一致。React 里，只需更新组件的 state，然后根据新的 state 重新渲染用户界面（不要操作 DOM）。

以下实例中创建了 **LikeButton** 组件，getInitialState 方法用于定义初始状态，也就是一个对象，这个对象可以通过 this.state 属性读取。当用户点击组件，导致状态变化，this.setState 方法就修改状态值，每次修改以后，自动调用 this.render 方法，再次渲染组件。

var LikeButton = React.createClass({

getInitialState: function() {

return {liked: false};

},

handleClick: function(event) {

this.setState({liked: !this.state.liked});

},

render: function() {

var text = this.state.liked ? '喜欢' : '不喜欢';

return (

<p onClick={this.handleClick}>

你<b>{text}</b>我。点我切换状态。

</p>

);

}

});

ReactDOM.render(

<LikeButton />,

document.getElementById('example')

);

随着state的改变，render也会被调用，React会对比render的返回值，如果有变化就会DOM。

state与props类似，只能通过setSate()方法修改。不同的是，state只能在组件内部使用，其是对组件本身状态的一个引用。

setState（）方法是异步的，并且在一个事件中设置了多个setState（）语句，只有最后一个setState（）方法执行。

### setState的回调函数

在使用React过程中，中可以使用this.state来访问需要的某些状态，但是需要更新或者修改state时，一般而言，我们都会使用setState()函数，从而达到更新state的目的，setState()函数执行会触发页面重新渲染UI。但是！！！setState是异步的！！！

我们在一个点击事件里设置了state之后，另起一行打印刚设置的state；会发现打印的state并没有改变。所以state的回调函数就是把setState（）方法的异步方法用同步的方式返回出来。

不在回调中使用参数，我们在设置state后立即使用state：

this.state = {foo: 1};

this.setState({foo: 123});

console.log(this.state.foo);

// 1

在回调中调用设置好的state

this.state = {foo: 2};

this.setState({foo: 123}, ()=> {

 console.log(foo);

 // 123

});

## React props

React中的数据流是单向的，只会从父组件传递到子组件。属性props（properties）是父子组件间进行状态传递的接口，React会向下遍历整个组件树，并重新渲染使用这个属性的组件。

setProps()方法只能在组件外调用，不能在组件内部调用this.setProps()修改组件属性。组件内部的this.props属性是只读的，只能用于访问props，不能用于修改组件自身的属性。

组件接收上级组件的props，并传递props到其下级组件。如：

var myCheckbox = React.createClass({

render: myCheckbox() {

var myClass = this.props.checked ? 'MyChecked' : 'MyCheckbox';

return (

<div className={myClass} onClick={this.props.onClick}>

{this.props.children}

</div>

);

}

}

);

React.render(

<MyCheckbox checked={true} onClick={console.log.bind(console)}>

Hello world!

</MyCheckbox>,

document.getElementById('example')

);

## Props与state的比较

React会根据props或state更新视图状态。虽然二者有些类似，但应用范围确不尽相同。具体表现如下：

1.props会在整个组件数中传递数据和配置，props可以设置任命类型的数据，应该把它当做组件的数据源。不仅可以用于上级组件与下组件的通信，而且可以用其做为事件处理器。

2.state只能在组件内部使用，state只应该用于存储简单的视图状（如：上面示例用于控制下拉框的可见状态）。

3.props和state都不能直接修改，而应该分别使用setProps()和setSate()方法修改。

## React 组件 API

在本章节中我们将讨论 React 组件 API。我们将讲解以下7个方法:

设置状态：setState

setState(object nextState[, function callback])

参数说明：

nextState，将要设置的新状态，该状态会和当前的state合并callback，可选参数，回调函数。该函数会在setState设置成功，且组件重新渲染后调用。

合并nextState和当前state，并重新渲染组件。setState是React事件处理函数中和请求回调函数中触发UI更新的主要方法。

注意：

不能在组件内部通过this.state修改状态，因为该状态会在调用setState()后被替换。

setState()并不会立即改变this.state，而是创建一个即将处理的state。setState()是异步的，为了提升性能React会批量执行state和DOM渲染。

setState()总是会触发一次组件重绘，除非在shouldComponentUpdate()中实现了一些条件渲染逻辑。

1. 替换状态：replaceState

replaceState(object nextState[, function callback])

nextState，将要设置的新状态，该状态会替换当前的state。

callback，可选参数，回调函数。该函数会在replaceState设置成功，且组件重新渲染后调用。

replaceState()方法与setState()类似，但是方法只会保留nextState中状态，原state不在nextState中的状态都会被删除。

1. 设置属性：setProps

setProps(object nextProps[, function callback])

nextProps，将要设置的新属性，该状态会和当前的props合并

callback，可选参数，回调函数。该函数会在setProps设置成功，且组件重新渲染后调用。

设置组件属性，并重新渲染组件。

props相当于组件的数据流，它总是会从父组件向下传递至所有的子组件中。当和一个外部的JavaScript应用集成时，我们可能会需要向组件传递数据或通知React.render()组件需要重新渲染，可以使用setProps()。

更新组件，我可以在节点上再次调用React.render()，也可以通过setProps()方法改变组件属性，触发组件重新渲染。

1. 替换属性：replaceProps

replaceProps(object nextProps[, function callback])

nextProps，将要设置的新属性，该属性会替换当前的props。

callback，可选参数，回调函数。该函数会在replaceProps设置成功，且组件重新渲染后调用。

replaceProps()方法与setProps类似，但它会删除原有

1. 强制更新：forceUpdate

forceUpdate([，function callback])

callback，可选参数，回调函数。该函数会在组件render()方法调用后调用。

forceUpdate()方法会使组件调用自身的render()方法重新渲染组件，组件的子组件也会调用自己的render()。但是，组件重新渲染时，依然会读取this.props和this.state，如果状态没有改变，那么React只会更新DOM。

forceUpdate()方法适用于this.props和this.state之外的组件重绘（如：修改了this.state后），通过该方法通知React需要调用render()

一般来说，应该尽量避免使用forceUpdate()，而仅从this.props和this.state中读取状态并由React触发render()调用。

1. 获取DOM节点：findDOMNode

DOMElement findDOMNode()

返回值：DOM元素DOMElement

如果组件已经挂载到DOM中，该方法返回对应的本地浏览器 DOM 元素。当render返回null 或 false时，this.findDOMNode()也会返回null。从DOM 中读取值的时候，该方法很有用，如：获取表单字段的值和做一些 DOM 操作。

7、判断组件挂载状态：isMounted

bool isMounted()

返回值：true或false，表示组件是否已挂载到DOM中

isMounted()方法用于判断组件是否已挂载到DOM中。可以使用该方法保证了setState()和forceUpdate()在异步场景下的调用不会出错。

## 表单

用户在表单填入的内容，属于用户跟组件的互动，所以不能用 this.props。

var Input = React.createClass({

getInitialState: function() {

return {value: 'Hello!'};

},

handleChange: function(event) {

this.setState({value: event.target.value});

},

render: function () {

var value = this.state.value;

return (

<div>

<input type="text" value={value} onChange={this.handleChange} />

<p>{value}</p>

</div>

);

}});

ReactDOM.render(<Input/>, document.body);

## Ajax请求服务器

组件的数据来源，通常是通过 Ajax 请求从服务器获取，可以使用 componentDidMount 方法设置 Ajax 请求，等到请求成功，再用 this.setState 方法重新渲染 UI （ajax请求例子）：

var UserGist = React.createClass({

getInitialState: function() {

return {

username: '',

lastGistUrl: ''

};

},

componentDidMount: function() {

$.get(this.props.source, function(result) {

var lastGist = result[0];

if (this.isMounted()) {

this.setState({

username: lastGist.owner.login,

lastGistUrl: lastGist.html\_url

});

}

}.bind(this));

},

render: function() {

return (

<div>

{this.state.username}'s last gist is

<a href={this.state.lastGistUrl}>here</a>.

</div>

);

}});

ReactDOM.render(

<UserGist source="<https://api.github.com/users/octocat/gists>" />,

document.body);

但是，React 本身没有任何依赖，完全可以不用jQuery，而使用其他库。

如果Promise对象正在抓取数据（pending状态），组件显示"正在加载"；如果Promise对象报错（rejected状态），组件显示报错信息；如果Promise对象抓取数据成功（fulfilled状态），组件显示获取的数据。

# React-Redux 的用法

为了方便使用，Redux 的作者封装了一个 React 专用的库 [React-Redux](https://github.com/reactjs/react-redux" \t "http://www.ruanyifeng.com/blog/2016/09/_blank)，这个库是可以选用的。实际项目中，你应该权衡一下，是直接使用 Redux，还是使用 React-Redux。后者虽然提供了便利，但是需要掌握额外的 API，并且要遵守它的组件拆分规范。

## UI 组件

React-Redux 将所有组件分成两大类：UI 组件（presentational component）和容器组件（container component）。

UI 组件有以下几个特征：

1.只负责 UI 的呈现，不带有任何业务逻辑

2.没有状态（即不使用this.state这个变量）

3.所有数据都由参数（this.props）提供

4.不使用任何 Redux 的 API。

## 容器组件

容器组件的特征恰恰相反。

1.负责管理数据和业务逻辑，不负责 UI 的呈现

2.带有内部状态

3.使用 Redux 的 API

总之，一句话：UI 组件负责 UI 的呈现，容器组件负责管理数据和逻辑。

React-Redux 规定，所有的 UI 组件都由用户提供，容器组件则是由 React-Redux 自动生成。也就是说，用户负责视觉层，状态管理则是全部交给它。外面是一个容器组件，里面包了一个UI 组件。前者负责与外部的通信，将数据传给后者，由后者渲染出视图。

## connect()

React-Redux 提供connect方法，用于从 UI 组件生成容器组件。connect的意思，就是将UI组件与容器组件连起来。

例如：

import { connect } from 'react-redux'

const VisibleTodoList = connect()(TodoList);

上面代码中，TodoList是 UI 组件，VisibleTodoList就是由 React-Redux 通过connect方法自动生成的容器组件。

但是，因为没有定义业务逻辑，上面这个容器组件毫无意义，只是 UI 组件的一个单纯的包装层。为了定义业务逻辑，需要给出下面两方面的信息。

（1）输入逻辑：外部的数据（即state对象）如何转换为 UI 组件的参数

（2）输出逻辑：用户发出的动作如何变为 Action 对象，从 UI 组件传出去。

connect方法的完整 API：

import { connect } from 'react-redux'

const VisibleTodoList = connect(

mapStateToProps,

mapDispatchToProps)(TodoList)

上面代码中，connect方法接受两个参数：mapStateToProps和mapDispatchToProps。它们定义了 UI 组件的业务逻辑。前者负责输入逻辑，即将state映射到 UI 组件的参数（props），后者负责输出逻辑，即将用户对 UI 组件的操作映射成 Action。

## mapStateToProps()

mapStateToProps是一个函数。它的作用就是像它的名字那样，建立一个从（外部的）state对象到（UI 组件的）props对象的映射关系。

作为函数，mapStateToProps执行后应该返回一个对象，里面的每一个键值对就是一个映射。

const mapStateToProps = (state) => {

return {

todos: getVisibleTodos(state.todos, state.visibilityFilter)

}}

上面代码中，mapStateToProps是一个函数，它接受state作为参数，返回一个对象。这个对象有一个todos属性，代表 UI 组件的同名参数，后面的getVisibleTodos也是一个函数，可以从state算出 todos 的值。

下面就是getVisibleTodos的一个例子，用来算出todos

const getVisibleTodos = (todos, filter) => {

switch (filter) {

case 'SHOW\_ALL':

return todos

case 'SHOW\_COMPLETED':

return todos.filter(t => t.completed)

case 'SHOW\_ACTIVE':

return todos.filter(t => !t.completed)

default:

throw new Error('Unknown filter: ' + filter)

}}

mapStateToProps会订阅 Store，每当state更新的时候，就会自动执行，重新计算 UI 组件的参数，从而触发 UI 组件的重新渲染。

mapStateToProps的第一个参数总是state对象，还可以使用第二个参数，代表容器组件的props对象。

// 容器组件的代码

// <FilterLink filter="SHOW\_ALL">

// All

// </FilterLink>

const mapStateToProps = (state, ownProps) => {

return {

active: ownProps.filter === state.visibilityFilter

}}

使用ownProps作为参数后，如果容器组件的参数发生变化，也会引发 UI 组件重新渲染。

connect方法可以省略mapStateToProps参数，那样的话，UI 组件就不会订阅Store，就是说 Store 的更新不会引起 UI 组件的更新。

## mapDispatchToProps()

mapDispatchToProps是connect函数的第二个参数，用来建立 UI 组件的参数到store.dispatch方法的映射。也就是说，它定义了哪些用户的操作应该当作 Action，传给 Store。它可以是一个函数，也可以是一个对象。

如果mapDispatchToProps是一个函数，会得到dispatch和ownProps（容器组件的props对象）两个参数。

const mapDispatchToProps = (

dispatch,

ownProps) => {

return {

onClick: () => {

dispatch({

type: 'SET\_VISIBILITY\_FILTER',

filter: ownProps.filter

});

}

};

}

从上面代码可以看到，mapDispatchToProps作为函数，应该返回一个对象，该对象的每个键值对都是一个映射，定义了 UI 组件的参数怎样发出 Action。

如果mapDispatchToProps是一个对象，它的每个键名也是对应 UI 组件的同名参数，键值应该是一个函数，会被当作 Action creator ，返回的 Action 会由 Redux 自动发出。举例来说，上面的mapDispatchToProps写成对象就是下面这样。

const mapDispatchToProps = {

onClick: (filter) => {

type: 'SET\_VISIBILITY\_FILTER',

filter: filter

};

}

# React-router4.x与页面传参

## histroy 属性

Router组件的history属性，用来监听浏览器地址栏的变化，并将URL解析成一个地址对象，供 React Router 匹配。

history属性，一共可以设置三种值。

hashHistory

browserHistory

createMemoryHistory

1.如果设为hashHistory，路由将通过URL的hash部分（#）切换，URL的形式类似example.com/#/some/path。

import React, { Component } from 'react';

import { HashRouter, Route, Switch } from 'react-router-dom';

import Login from'./routes/Login';

import Home from'./routes/Home';

export default function RouterConfig() {

return (

<HashRouter>

<Switch>

<Route path='/' exact component={Login} />

<Switch>

<Route path='/home' exact component={Home} />

</Switch>

</Switch>

</HashRouter>

);

}

2如果设为browserHistory，浏览器的路由就不再通过Hash完成了，而显示正常的路径example.com/some/path，背后调用的是浏览器的History API。

import React, { Component } from 'react';

import { BrowserRouter, Route, Switch } from 'react-router-dom';

import Login from'./routes/Login';

import Home from'./routes/Home';

export default function RouterConfig() {

return (

<BrowserRouter>

<Switch>

<Route path='/' exact component={Login} />

<Switch>

<Route path='/home' exact component={Home} />

</Switch>

</Switch>

</BrowserRouter>

);

}

但是，这种情况需要对[服务器改造](https://github.com/reactjs/react-router/blob/master/docs/guides/Histories.md" \l "configuring-your-server" \t "http://www.ruanyifeng.com/blog/2016/05/_blank)。否则用户直接向服务器请求某个子路由，会显示网页找不到的404错误。

如果开发服务器使用的是webpack-dev-server，那么要在devserver里添加historyApiFallback: true,

然后在output里添加：publicPath:’/’,

就能完美解决刷新报404的问题。

## Switch标签与层级式APP

### **Switch标签**

既然我们用了BrowserRouter或者HashRouter，那么我们就必须要用到Switch标签。Switch标签渲染第一个被history的location匹配到的并且作为子元素的<Route>或者<Redirect>。Switch标签还可以嵌套，来突出页面之间的包含关系。

<BrowserRouter>

<Switch>

<Route path='/' exact component={Login} />

<Route path='/home' exact component={Home} />

</Switch>

</BrowserRouter>

### 层级式app

在配置具有层级行性的网站，比如具有多级菜单导航的后管理系统时，这类网站的特点是只有一个登陆页面在最外部，而登陆进去后包括Home页面都是在登陆之后。我们可以构建层级式app，就是把登陆单独看做一个Route，然后把网站内部看成一个大route。这样的话就可以分级写route。

import Login from './routes/Login';

import App from './routes/App';

import HomePage from './routes/HomePage';

import SAM from './routes/SAM';

import Station from './routes/Station';

import TabPage from './routes/TabPage';

import Ticket from './routes/Ticket';

export default function RouterConfig() {

return (

<HashRouter>

<Switch>

<Route path='/' exact component={Login} />

<App>

<Switch>

<Route path='/tabpage' exact component={TabPage} />

<Route path='/sam' exact component={SAM} />

<Route path='/home' exact component={HomePage} />

<Route path='/station' exact component={Station} />

<Route path='/tabpage' exact component={TabPage} />

<Route path='/ticket' exact component={Ticket} />

</Switch>

</App>

</Switch>

</HashRouter>

);

}

其中：app只是个父容器页面，用来包裹所有内部的页面route，把其他页面route看做为children。

App.js:

import React from 'react';

import MainLayout from '../components/MainLayout';

export default class App extends React.Component {

constructor(props) {

super(props);

this.state = {

};

}

componentWillMount() {

console.log(this.props);//已经有chldren了

}

componentDidMount() {}

render() {

const { children, location } = this.props;

return (

<div>

<MainLayout location={location}>{children}</MainLayout>

</div>

);

}

}

然后mainLayout里面部署我们的chcildren页面，是通过路由引进来的。

## path 属性

Route组件的path属性指定路由的匹配规则。这个属性是可以省略的，这样的话，不管路径是否匹配，总是会加载指定组件。

请看下面的例子。

<Route path="inbox" component={Inbox}>

<Route path="messages/:id" component={Message} /></Route>

上面代码中，当用户访问/inbox/messages/:id时，会加载下面的组件。

<Inbox>

<Message/></Inbox>

如果省略外层Route的path参数，写成下面的样子。

<Route component={Inbox}>

<Route path="inbox/messages/:id" component={Message} />

</Route>

现在用户访问/inbox/messages/:id时，组件加载还是原来的样子。

<Inbox>

<Message/></Inbox>

## 页面跳转——Link

Link组件用于取代<a>元素，生成一个链接，允许用户点击后跳转到另一个路由。它基本上就是<a>元素的React 版本，可以接收Router的状态。

import { Link} from 'react-router-dom';

render() {

return (<div>

<ul role="nav">

<li><Link to="/about">About</Link></li>

<li><Link to="/repos">Repos</Link></li>

</ul>

</div>

)}

如果希望当前的路由与其他路由有不同样式，这时可以使用Link组件的activeStyle属性。

<Link to="/about" activeStyle={{color: 'red'}}>About</Link>

<Link to="/repos" activeStyle={{color: 'red'}}>Repos</Link>

上面代码中，当前页面的链接会红色显示。

另一种做法是，使用activeClassName指定当前路由的Class。

<Link to="/about" activeClassName="active">About</Link><Link to="/repos" activeClassName="active">Repos</Link>

上面代码中，当前页面的链接的class会包含active。

在Router组件之外，导航到路由页面，可以使用浏览器的History API，像下面这样写。

import { browserHistory } from 'react-router';

browserHistory.push('/some/path');

## 页面传参——通配符

在router中：

path属性可以使用通配符来传递简单的参数，比如某个字段，某个判断标识等。

<Route path="/suocang/:type">

name字段是页面传参的标识。

在相应页面中

dispatch(routerRedux.push({pathname: '/suocang/1'}));

在suocang页面中

console.log(this.props.match.params.type);

根据this.props.match.params.type来判断穿的参数是1还是2或者其他字符

<Route path="/hello(/:name)">

// 匹配 /hello

// 匹配 /hello/michael

// 匹配 /hello/ryan

<Route path="/files/\*.\*">

// 匹配 /files/hello.jpg

// 匹配 /files/hello.html

<Route path="/files/\*">

// 匹配 /files/

// 匹配 /files/a

// 匹配 /files/a/b

<Route path="/\*\*/\*.jpg">

// 匹配 /files/hello.jpg

// 匹配 /files/path/to/file.jpg

通配符的规则如下。

（1）:paramName

:paramName匹配URL的一个部分，直到遇到下一个/、?、#为止。这个路径参数可以通过this.props.params.paramName取出。

（2）():

()表示URL的这个部分是可选的。

（3）匹配任意字符，直到模式里面的下一个字符为止。匹配方式是非贪婪模式。

（4） 匹配任意字符，直到下一个/、?、#为止。匹配方式是贪婪模式。

path属性也可以使用相对路径（不以/开头），匹配时就会相对于父组件的路径，可以参考上一节的例子。嵌套路由如果想摆脱这个规则，可以使用绝对路由。

路由匹配规则是从上到下执行，一旦发现匹配，就不再其余的规则了。

<Route path="/comments" ... />

<Route path="/comments" ... />

上面代码中，路径/comments同时匹配两个规则，第二个规则不会生效。

设置路径参数时，需要特别小心这一点。

<Router>

<Route path="/:userName/:id" component={UserPage}/>

<Route path="/about/me" component={About}/></Router>

上面代码中，用户访问/about/me时，不会触发第二个路由规则，因为它会匹配/:userName/:id这个规则。因此，带参数的路径一般要写在路由规则的底部。此外，URL的查询字符串/foo?bar=baz，可以用this.props.location.query.bar获取。

## 页面传参——routerRedux

传递：将需要传递的参数都写到对象里，然后在props里的dispatch方法，将参数push到routerRedux里就行了。

turn(){

const{dispatch}=this.props;

const path={

pathname: '/c1',

q:1,

w:2,

}

dispatch(routerRedux.push(path));

}

获取：在目的页面里的props里获取：

componentWillMount(){

console.log(this.props);

}

## 页面传参——history

类似routerRedux，history可以直接push参数与路由。

写法：

const{dispatch,history}=this.props;  
    const path={  
    pathname: '/home',  
    q:1,  
    w:2,  
     arr:this.state.arr,  
    }  
//dispatch(routerRedux.push(path));//dispatch的routerRedux方法  
 history.push(path);

获取方法与routerRedux方法都一样：在目的页面里的props里获取：

componentWillMount(){console.log(this.props);}

## Context对象

Context 可以构建 API 使得父组件和子组件进行相互通信。

在React中，数据可以以流的形式自上而下的传递，每当你使用一个组件的时候，你可以看到组件的props属性会自上而下的传递。但是，在某些情况下，我们不想通过父组件的props属性一级一级的往下传递，我们希望在某一级子组件中，直接得到上N级父组件中props中的值。

## 表单处理

Link组件用于正常的用户点击跳转，但是有时还需要表单跳转、点击按钮跳转等操作。这些情况怎么跟React Router对接呢？

下面是一个表单。

<form onSubmit={this.handleSubmit}>

<input type="text" placeholder="userName"/>

<input type="text" placeholder="repo"/>

<button type="submit">Go</button></form>

第一种方法是使用browserHistory.push

import { browserHistory } from 'react-router'

// ... handleSubmit(event) {

event.preventDefault()

const userName = event.target.elements[0].value

const repo = event.target.elements[1].value

const path = `/repos/${userName}/${repo}`

browserHistory.push(path)

},

第二种方法是使用context对象。

export default React.createClass({

// ask for `router` from context contextTypes: {

router: React.PropTypes.object

},

handleSubmit(event) {

// ... this.context.router.push(path)

},})

## 路由的钩子

每个路由都有Enter和Leave钩子，用户进入或离开该路由时触发。

<Route path="about" component={About} />

＜Route path="inbox" component={Inbox}>

＜Redirect from="messages/:id" to="/messages/:id" /></Route>

上面的代码中，如果用户离开/messages/:id，进入/about时，会依次触发以下的钩子。

/messages/:id的onLeave

/inbox的onLeave

/about的onEnter

下面是一个例子，使用onEnter钩子替代<Redirect>组件。

<Route path="inbox" component={Inbox}>

<Route

path="messages/:id"

onEnter={

({params}, replace) => replace(`/messages/${params.id}`)

}

/></Route>

onEnter钩子还可以用来做认证。

const requireAuth = (nextState, replace) => {

if (!auth.isAdmin()) {

// Redirect to Home page if not an Admin

replace({ pathname: '/' })

}

}

export const AdminRoutes = () => {

return (

<Route path="/admin" component={Admin} onEnter={requireAuth} />

)

}

下面是一个高级应用，当用户离开一个路径的时候，跳出一个提示框，要求用户确认是否离开。

const Home = withRouter(

React.createClass({

componentDidMount() {

this.props.router.setRouteLeaveHook(

this.props.route,

this.routerWillLeave

)

},

routerWillLeave(nextLocation) {

// 返回 false 会继续停留当前页面， // 否则，返回一个字符串，会显示给用户，让其自己决定 if (!this.state.isSaved)

return '确认要离开？';

},

}))

# React生命周期与函数执行顺序

在一个单页面里，通常会有如下几个函数先后执行：

1.constructor(props) {

super(props);

//*this*.state = {}

}

constructor函数是当你在React class中需要设置state的初始值或者绑定事件时触发的，往往在constructor函数里设置本class里面需要的state，进行初始化state。Constructor（）函数必须要配合super（）才能正常使用，因为单独使用的话会导致this的作用域改变，不再指向你所命名的class类。

1. componentWillMount()函数。

componentWillMount()函数是在constructor()函数之后，render()之前执行一次。通常情况下会在这个函数里绑定一些自动执行的一些方法，比如改变state，请求一些页面数据，供自动加载。

1. render()函数。

Render()函数会在componentWillMount()函数执行完之后立即执行一次（第一次执行）。通常render函数里的JSX会将componentWillMount()函数里获取到的state数据渲染到react组件里。

1. componentDidMount()函数。

componentDidMount()函数会在render()函数渲染组件后立即执行。通常componentDidMount()函数里会加载一些render()函数里绑定的点击事件等。

5.componentWillUpdate()函数（如果有的话）

componentWillUpdate()函数会在componentDidMount()函数执行之后，第二次render执行之前执行。通常会在此方法里重新setState（）。

6.render()函数。

第二次执行render()函数，会将componentDidMount()函数里的点击事件返回的数据或状态重新渲染一回，实现刷新render()函数里的JSX组件的效果。

7.componentDidUpdate()函数（如果有的话）

componentDidUpdate()函数会在render（）函数刷新之后执行。

总结起来就是：

constructor(props) {super(props);}--->componentWillMount()--->

--->render()--->componentDidMount()--->componentWillUpdate()--->render()--->componentDidUpdate()。

# React引用样式，图片，工具类的方法总结

## 引用CSS样式：

import styles from './IndexPage.css';

如果页面js与CSS在同一个文件夹内，用./IndexPage.css。

如果在外面，比如css样式表在另一个styles文件夹内，则写为../styles/index.css

Render里使用：

<div className={styles.normal}>6666</div>

## 引入图片：

import movieImg from '../assets/images/bg.png';

import bgImg from "../assets/images/bg.png";

Render里使用：

①：img直接使用

<img src={movieImg } />

②：div背景图片使用

在JSX的style标签里：

<style>{`

.ex{background：url(${bgImg}) no-repeat}

`}</style>

或者background：url({require(“../assets/images/bg.png”)}) no-repeat

行内样式：

<div

style={{backgroundImage:`url(${bgImg})`,backgroundSize: 'cover'}}>

...

</div>

## 引入JS工具类：

有两种情况，1是整个js类，2是某个js类下面的一个子方法。

### 1.引入整个工具类：

例1，比如引入user请求接口文件：

在工具类../services/user.js里所有的函数都写为暴露状态，前面加个export，（白鹭的暴露形式为 public static function dateutils（）{。。。}）。

//请求所有人的加入动态

export async function getJoin(){

......

}

在页面js里引用：import \* as user from '../services/user';

使用：let res=async user.getJoin();

例2，引用时间格式化工具类Dateutil.js：

在DateUtil.js里所有函数都抛出：export function format(fmt,data) {

...  
}

在页面js里引用：import \* as Datautils from '../services/Datautils ;

使用：this.setState({overTime:Datautils.format(res.overTime)})

### 2.引入某个工具类的某一个方法：

例1：引用Fetch.js里的登录方法

Fetch.js里写法，同样把方法写成暴露形式，加个export

//判断有没有本地存储

export *function* loggedIn(){

*var* user = store.get("username");

if(user){return true;}

else{return false;}

}

export *function* loggedOut(){

....

}

在页面js里引用：

import {loggedIn,loggedOut,....} from '../utils/fetch';

在页面js里使用：

If（loggedIn）{

...

}else{

...

}

一个工具类js的方法函数只要写成暴露形式，都可以在页面js里引用。

### 3.引入外部的组件（class类）并通过子组件改变自身的状态：

在项目中，往往会遇到每个不同的页面内会有公共的组件，为了节省代码，使代码结构更清晰，把公共的组件写到一个公共的js类里。比如页面最下部的底部导航条：

在此，以Public为例，引入一个Public的类：

一：父页面向子组件传递参数：

1.import Public from “./Public.js”;

2.在其他页面中render中渲染并传值：

<Public number={0}/>

3.在Public.js里获取参数：

console.log(this.props);或者Es6语法：const{number}=this.props

然后判断number的值，根据number不同的值来做不同的颜色渲染。

二：父页面向子组件传递方法:

1. 父页面里(Father.js):引入Child组件,并把自己的方法写到子组件里。

import Chlid from “./Chlid”;

export default Father extends React.Componment{

Constructor(props){

Super(props);

This.state={

C:0,

a:1,

}

}

changeState(){

This.setState({a:2})

}

Render(){

Return(

<Chlid c={this.state.c} onCh={this.changeState.bind(this)} />

)

}

}

1. 子页(Child.js)面在this.props里获取，调用时直接调用props:

export default Child extends React.Componment{

Constructor(props){

Super(props);

This.state={}

}

ComponmentWillmount(){

Console.log(this.props);

}

Render(){

Return(

<div onClick={this.props.clickDiv.bind(this)}>点我改变父页面的状态a</div>

)

}

}

1. 在父页面为子页面（child）添加React标签，并在子页面中使用：

父页面：import Chlid from “./Chlid”;

export default Father extends React.Componment{

Constructor(props){

Super(props);

This.state={

C:0,

a:1,

}

}

changeState(){

This.setState({a:2})

}

Render(){

Return(

<Chlid c={this.state.c} onCh={this.changeState.bind(this)} >

<Button type=”praimy” ><Button>

<div>123123123</div>

</Child>

)

}

}

子页面：

export default Child extends React.Componment{

Constructor(props){

Super(props);

This.state={}

}

ComponmentWillmount(){

Console.log(this.props);

}

Render(){

const {children}=this.props;

Return(

<div onClick={this.props.clickDiv.bind(this)}>点我改变父页面的状态a</div>

<div>{children}</div>

)

}

}

# React全选与单选

React，vue，Angular等MVC框架的全选与单选的思路都是一样的。大致思路为：

声明一个selectedArr:[ ] 作为记录数组，跟数据源数组dataArr做比较。

在Dom结构中，某个子项的selected的判断依据是该子项的某一属性(如value属性)是否存在selectedArr中（selectedArr.indexOf(item.value)>=0），如果存在，那么就是选中状态，不存在就是非选中状态。全选的selected属性的判断依据是selecteArr.length==dataArr.Length。

在子项按钮的onChange事件中，判断该子项的属性是否在selectedArr中，让selectedArr添加或删除该子项的属性。

在全选按钮的onChange事件中，因为全选按钮只有两种事件：全选与全清空，但是selectedArr对应2种状态，满额与未满额。所以判断selecteArr的长度与dataArr的长度是否相等，如果不相等，那么就先清空selectedArr，然后for循环遍历dataArr，像selecteArr中push(dataArr.value)，反之如果已经满额，那么就清空selectedArr。

代码如下：

this.state={

li0:[

{ value: 0, label: 'Ph.D.'},

{ value: 1, label: 'Bachelor'},

{ value: 2, label: 'College diploma'},

],//数据源数组

selectedArray:[],//记录选中的子项

],

}

//单选

onChange(item,index){

let{selectedArray,li0}=this.state;

console.log(item);

let sInd= selectedArray.indexOf(item.value);//返回item.value在selectedArray种的位置index值，如果存在就返回index，不存在就返回-1.

if(sInd>=0){

selectedArray.splice(sInd,1);

}else{

selectedArray.push(item.value)

}

this.setState({selectedArray});

}

//全选与清空

allC(){

let{selectedArray,li0}=this.state;

if(selectedArray.length==li0.length){

selectedArray=[];

}else{

selectedArray=[];

for(var i=0;i<li0.length;i++){

selectedArray.push(li0[i].value);

}

}

this.setState({selectedArray});

} render(){

Const{li0,selectedArray} =this.state;

return (

<div className={styles.normal}>

{li0.map((data\_item,index) => (

<List key={index}>

{/\*复选框\*/}

<CheckboxItem

onChange={() => this.onChange(data\_item,index)}

checked={selectedArray.indexOf(data\_item.value)>=0}

>

{data\_item.label}

</CheckboxItem>

</List>

))}

....

<div>123123</div>

</div>

)

}

React解决事件冒泡的方法：

React 为提高性能，有自己的一套事件处理机制，相当于将事件代理到全局进行处理，也就是说监听函数并未绑定到DOM元素上。因此，如果你禁止react事件冒泡e.stopPropagation()，你就无法阻止原生事件冒泡；你禁用原生事件冒泡e.nativeEvent.stopPropagation()，React的监听函数就调用不到了。

正确的姿势，应该是判断event.target对象，是否是目标对象、或包含的对象、或被包含的对象，来决定是否触发事件。以下函数就可以用来判断包含性：

阻止冒泡事件分三种情况

1. 阻止合成事件间的冒泡（父元素与子元素都设置事件）用e.stopPropagation();

import React,{ Component } from 'react';

import ReactDOM,{findDOMNode} from 'react-dom';

class Counter extends Component{

constructor(props){

super(props);

this.state = {

count:0,

}

}

handleClick(e) {

// 阻止合成事件间的冒泡

e.stopPropagation();

this.setState({count:++this.state.count});

}

testClick(){ console.log('test'); }

render(){

return(

<div ref="test" onClick={()=>this.testClick()}>

<p>{this.state.count}</p>

<a ref="update" onClick={(e)=>this.handleClick(e)}>更新</a>

</div>

)

}

}

B、阻止原生事件与最外层document上的事件间的冒泡，用e.nativeEvent.stopImmediatePropagation();

componentDidMount(){

document.onclick=function(){alert(1);}

}

con(e){

console.log(e.target.nodeName);

//阻止事件冒泡

e.nativeEvent.stopImmediatePropagation();

var clickEl=e.target;

console.log(clickEl);

console.log(clickEl.className);

}

Render(){

Return(

<Button type="primary" onClick={this.con.bind(this)}>打印target元素</Button>

)}

C、阻止合成事件与除最外层document上的原生事件上的冒泡，通过判断e.target来避免

import React,{ Component } from 'react';

import ReactDOM,{findDOMNode} from 'react-dom';

class Counter extends Component{

constructor(props){

this.state = {

count:0,

}

}

componentDidMount() {

document.body.addEventListener('click',e=>{

// 通过e.target判断阻止冒泡

if(e.target&&e.target.matches('a')){

return;

}

console.log('body');

})

}

handleClick(e){

this.setState({count:++this.state.count});

}

render(){

return(

<div ref="test">

<p>{this.state.count}</p>

<a ref="update" onClick={(e)=>this.handleClick(e)}>更新</a>

</div>

)

}

}

# 基于webpack搭建React+antd项目（待完成）

## 1.创建项目

创建一个文件夹作为根目录，npm init 创建项目的package.json，来配置项目的版本号、作者、项目说明等。

在根目录下创建一个index.html、index.js、src文件夹、webpack.config.js、.babelrs.js。

Src目下：创建src/router.js（页面路路由管理）创建src/components(组件管理) 、src/css （css资源）、src/images （图片资源）、src/routes（页面文件夹）、src/utils（请求或数据转化工具js文件夹）

## 配置webpack.config.js与必要的babel环境

1.在工程项目里shift+右键打开命令行，cnpm install antd --save

2.node\_moduell包里会有antd包，并且会更新到package.json中。

在webpack.config.js里的rules里添加这么一段：连react+antd都可以加载。

//确保每个页面的css都是私有化的，不会出现全局css污染某个页面的样式。

{

test: /\.css$/,

use: [

{ loader: 'style-loader' },

{

loader: 'css-loader',

options:{

modules: true,

localIdentName: '[name]\_\_[local]--[hash:base64:5]',//为每一个scss编写唯一的区分

},

},

],

include: [path.resolve(\_\_dirname, 'src')], // 样式只应用到这个文件夹下面的css文件中

},

//支持React与antd

{

test: /\.js|.jsx$/,

exclude: /node\_modules/,

loader: 'babel-loader',

options: {

presets: ['es2015', 'react'],

plugins: [

// 引入样式为 css

// ['import', { libraryName: 'antd', style: 'css' }],

// 改动: 引入样式为 less

['import', { libraryName: 'antd', "libraryDirectory": "es",style: true }],

]

},

},

// 下面的less-loader是为了支持antd的主题

// less-loader的文档： https://www.npmjs.com/package/less-loader

{

test: /\.less$/,

use: [{

loader: 'style-loader',

}, {

loader: 'css-loader',

}, {

loader: 'less-loader',

options: {

modifyVars: { // 主体颜色

'primary-color': '#40a9ff',

'link-color': '##40a9ff',

'border-radius-base': '4px',

'font-size-base': '14px',

'line-height-base': '1.2',

},

javascriptEnabled: true,

},

}],

},

3.运行webpack：npm run build

# 基于 dva 创建React+ antd-mobile 的项目

### **1.安装 全局**

$ npm install dva-cli -g

### **2.创建项目**

选择一个目录然后创建项目

$ dva new my-project

my-project是我的项目名称

dva-cli 会为你创建好基本的文件架构 其中 .webpackrc 文件中是相关webpack的配置信息，格式为JSON，比如：

{"entry": "src/pages/\*.js"}

如果你喜欢js的写法可以用.webpackrc.js进行配置文件，支持ES6的语法，比如：

export default{

entry: "src/pages/\*.js"

}

相关配置可参考 [https://github.com/sorrycc/roadhog/blob/master/README\_zh-cn.md#extrababelpresets](https://link.jianshu.com/?t=https://github.com/sorrycc/roadhog/blob/master/README_zh-cn.md%23extrababelpresets" \t "https://www.jianshu.com/p/_blank)

### **3. 安装 antd-mobile**

$ npm install antd-mobile --save

安装过程中如果出现类似以下警告信息

peerDependencies WARNING antd-mobile@2.1.3 › react-native-collapsible@^0.9.0 requires a peer of react-native@\* but none was installed

peerDependencies WARNING antd-mobile@2.1.3 › react-native-drawer-layout@~1.3.0 requires a peer of react-native@\* but none was installed

Recently updated (since 2018-01-01): 2 packages (detail see file /Applications/MAMP/htdocs/react/dva-project/node\_modules/.recently\_updates.txt)

因为antd-mobile可以同时制作app所以需要 react-native 依赖

$ npm install react-native --save

### **4. 建议对项目是用按需加载，这样做可以减少打包后的体积大小**

找到根目录下的.webpackrc文件，并在该文件中添加以下代码

"extraBabelPlugins": [

["import", { "libraryName": "antd-mobile", "style": "css" }]

]

并安装以下插件

$ npm install babel-plugin-import --save-dev

# 完成以上操作就可以正确的安装并使用 antd-design 组件库了。

还有一个点需要注意的是，dva 建项目的时候会默认安装redux和react-router，所以在开发中千万不要在去安装，会因为版本不兼容而导致项目无法运行；

# React组件的解决方案

## 一：防重复点击

重复点击：重复点击是指当网络有延迟或者网速不好的情况下，同一个人在极短的时间内连续点击两次或多次，服务器返回了多个结果的现象。产生这种现象的原因是:当一个人点击一次发送请求时，由于网速慢，这时服务器还没来得及返回请求的结果，结果又点击了一次或多次，服务器就返回了多次结果。

防重复点击就是为了防止上面的情况发生。

### 解决办法：

1,在state里设个状态：dumiao：false，

2在点击事件的组件上设置：

onClick={tthis.state.dumiao==true?this.Dumiao.bind(this):this.clickBtn.bind(this)}

Dumiao是个空函数，就是当重复点击的时候什么也不做，防止重复发出请求。

在clickBtn(){//业务逻辑}这个函数里处理逻辑。

clickBtn(){

This.setstate({dumiao:true});

Const context=this；

//业务逻辑

//只要联网成功,只要有返回，不管是成功的回调还是错误的回调；

Context.setState({dumiao:false});

}

## 二：延时请求

延时请求是为了解决同一个时间点内多点请求的问题。就是防止同一时间内多个客户端(手机端与PC端或者多个手机端或者多个PC端)同时发送同一个请求，而导致重复发送请求的问题。

解决办法：

clickBtn(){

var suiji0=parseInt(Math.random()\*300);

var suiji1=parseInt(Math.random()\*200);

Var suiji=suiji0+suiji1;

setTimeout(function(){

//业务逻辑。。。

//联网

},suiji);

}

二者结合可以实现防止多点点击与防重复点击。

## e.target:获取非控组件的元素

在事件里用一个参数e接收，e.target表示所点击的Dom元素。

### 获取非空组件元素的值：

在input的onChange事件里传个event事件参数e，然后在onChange事件中通过获取e.target.value来得到input框的value值。

JSX里的代码：

<input onChange={this.cc.bind(this)} />

cc(e){

Console.log(e.target.value)

}

## Ant-design上传七牛图片获取外链的解决方案

1. 在七牛云上注册一个账号，并实名认证，成功之后在个人中心-秘钥管理中可得到一组AccessKey/SecretKey(ak与ak)
2. 在七牛创建一个项目空间，名字是bucketName；
3. 打开七牛官网提供的uptoken生成的API网址(自己玩的时候)：

<http://jsfiddle.net/gh/get/extjs/4.2/icattlecoder/jsfiddle/tree/master/uptoken>

将bucketName,ak,sk输入API，生成一个uptoken,于是就得到了token；

1. 但是在实际项目中，需要向后台请求uptoken。七牛云为后台生成uptoken提供了丰富的API。
2. 在antd-design的unload组件中引用Upload组件：

import {Upload} from “antd”;

<Upload

action=“http://up.qiniu.com”

data={{token:”你在七牛生成的token”}}

listType="picture-card"

className="upload-list-inline"

onChange={**this**.onChange}

onPreview={**this**.handlePreview}

fileList={fileList}

>

onChange(file){console.log(file)}

在onChange事件内返回的file数组里，有key或hash字段(二者内容相同)，保存下来

1. 登录七牛，在产品-对象存储-内容管理内有外链默认域名
2. [http://+链默认域名+hash(或key)就是](http://+链默认域名+hash(或key)jiushi)生成的图片外链。(在七牛云空间里也可直接查看外链名)。

## Ant-design上传Excel文件解决方案：

与上传图片相似，只是把action的路径改为服务器接口路径就OK了，然后在onChange事件里打印file，里面有我们所需要的所有信息。

代码如下：

 //上传excel  
 handleChange=(file)=>{  
  console.log(file);  
  if(file.file.status=="done"){  
  let questionArr=file.fileList[0].response.data;  
  console.log(questionArr);  
  if(questionArr[0]=="解析Excel出现异常"){  
  message.warning("上传文件错误,请重新上传!!");  
  return;  
  }else{  
  message.success("导入Excel表格成功!!")  
  this.setState({questionArr:questionArr});  
  }  
  }else if(file.file.status=="uploading"){  
  IMG_256message.info("正在上传,请耐心等待..");  
  }else{  
  IMG_257message.info("您已删除,请重新上传..");  
  }  
 }  
 //下载excel  
 downloadExcel(){  
  window.open(reajax.APIhost+"/admin/goods/downLoadExcel","\_blank");  
 }

Render{

Return(

<div className="uploadDiv">

<Upload

name='file'

action={reajax.APIhost+"/admin/goods/analysisExcel"}

onChange={this.handleChange}

>

<Button><Icon type="upload" /> 点击上传商品价格Excel表格</Button>

</Upload>

</div>

<div className="uploadDiv">

<Button

onClick={this.downloadExcel.bind(this)}

>

<Icon type="download"/>点击下载商品价格Excel表格

</Button>

</div>

)

}

## React富文本编辑器插件

1. react-umeditor
2. react-draft-wysiwyg

第一个插件可提供向七牛传送图片并返回前端图片外链的功能。

## React-Router在服务器端刷新404问题

### 1.node端

const express = require('express')

const path = require('path')

const port = process.env.PORT || 8080

const app = express()

// 通常用于加载静态资源

app.use(express.static(\_\_dirname + '/public'))

// 在你应用 JavaScript 文件中包含了一个 script 标签

// 的 index.html 中处理任何一个 route

app.get('\*', function (request, response){

 response.sendFile(path.resolve(\_\_dirname, 'public', 'index.html'))

})

app.listen(port)

console.log("server started on port " + port)

### 2.Nginx服务：

这里提供一种解决React项目放在PHPstudy上的nginx环境下，路由打不开的问题：

1进服务器打开PHPstudy，其他选项菜单

2打开配置文件

3选择最后一个vhosts-conf

1. 找到自己配置的端口文件，

location / {

index index.html index.htm index.php;

#autoindex on;

}

改为

location / {

index index.html index.htm index.php;

try\_files $uri $uri/ /index.html;

#autoindex on;

}

5保存，重启PHPstudy。

这种方法有个弊端，就是知识暂时性的修改保存配置文件，如果有新项目添加进来，保存并生成配置文件的话，该项目的配置文件就会还原，只能再次修改。

### 3.Webpak-dev-server

1.配置路由的根路由publicPath:'/'。

output: {

filename: 'index.js',

path: path.resolve(\_\_dirname, 'dist'),

publicPath:'/',

},

2.在devServer的配置中添加：historyApiFallback: true,使页面报错时找index.html。

devServer: {

clientLogLevel: 'error',

contentBase: path.resolve(\_\_dirname, 'dist/'),

inline: true,

hotOnly: true,

historyApiFallback: true,

overlay: {

warnings: false,

errors: true,

},

port: 8082,

},

# Angular.js

angular中文意思是棱角分明。

Angular是谷歌公司在2009年推出的一款构建HTML应用程序的一个Js框架。

Angular出现的目的就是谷歌公司为了干掉jquery。就目前来说，手机端jquery框架越来越大，用户使用的话就要多下载几十kb的jquery，耗费流量。而PC端就不在乎这几十Kb的流量了。

在浏览网页时，最消耗浏览器内存的有两个方面：①复杂的计算；②DOM的操作。Angular.js的目的是让开发人员尽量少操作DOM，甚至不操作DOM。

Angular.js是一个mvc框架。MVVM这种模块化的思想。

四大特性：①MVC ②模块化 ③双向数据绑定 ④指令系统

四种方式配置angular环境：

1：直接引用下载好的angular文件。

2：找CDN地址

3：npm install angular（小黑框输入指令下载）

4：brower指令下载 gult

开始第一个angular程序：

①配置环境

放置在node\_modules这个文件夹内，在html页面的script标签内引入angular.js文件

语法：

1：激活指令 ng-app=”自己创建的模块名“。ng：angualr。

通知引入的angular，激活所有子容器的angular指令标签，圈定自己的势力范围。

2：ng-model=value指令：model里面储存的是数据，通常作为input的指令，用于存储value值(用户手写或者controller生成得)。ng-mldel=value；value在标签内用占位符表示，如<p>{{value}}</p>。

3： view 指令（ng-指令名称）{{表达式}}表达式就是将model里面的数据放置表达式所在的位置。

1.98+2.98={{1.98+2.98|number：0}}

在计算数值时用了一个“|”管道符号。这个符号在angular中表示过滤器格式化数据，他的语法是{exp|fitername：value，value，value}number：0表示去掉小数点之后的数值，保留整数部分。

4：ng-init=“key=value”：初始化占位符的内容。

5：ng-controller=value：设置控制器，名字必须是以controller结尾。

Angular的执行流程是什么：

①当页面加载完毕的时候，angular执行

②html页面中的ng-xx属性都称为指令。

Angular减少了dom操作，把注意力集中到写业务上面去。

MVC模式：

Model 数据 饭菜

Controller 业务 厨子

View 展示 服务员

MVC本质上是一种开发思想主要目的就是解决应用程序展示结构，业务逻辑之间的解耦问题。

在项目中，angular.js开发的页面只有一个html公共页面,这个页面就是所有页面的公共部分，其他的页面都是由angular生成的。

Angular.js的几个重要组成部分：

Html部分：

<div ng-app ='Rocky' ng-controller="democontroller">

<p><input type="text" name="" id="" value="" ng-model="user.name"/></p>

<p>内容：<span>{{user.name}}</span></p>

</div>

JavaScript部分：

①定义模块 Module

var myfirstapp=angualr.module('Rocky'，[]);

两个参数：第一个是自己创建的模块名，第二个是你自己创建的模块所依赖的angular模块。如果不依赖，那么就是空数组[]。

②定义控制器controller

myfirstapp.controller(“democontroller”，function($scope){

$scope.user={};

$scope.user.name="张三";

})

Controller()方法：两个参数：

1：“变量名”:必须是以controller结尾。

2：匿名函数：在匿名函数内控制和动态操作html页面，用参数$scope接收。$scope是动态操作html页面占位符的，动态添加html页面内容。

$scope.$watch(“监听对象”,function(now,old){

})

Now:现在的监视数据。Old：mow之前的监视数据。

当多个ng-app的module模块都出现的时候，浏览器会默认执行第一个。复习wendow.onload:当有多个window.onload时，会执行最后一个。但是我们就想使多个module生效，怎么办呢？就要在后面一个

angualr.module('Rocky'，[])的[]中分别定义module名：[‘module1’,’module2’]。

动态注册一个指令，要用$scope.directive(“指令名称”，function(){

Return function(scope,element,attrs){

}

})

|  |  |
| --- | --- |
| Component | 存放独立的angular文件 |
| Controller | 存放控制器 |
| Filters | 存放过滤器 |
| Ng-modules | 存放模块 |
| Views | 视图 |

# Vue.js

Vue的配置环境：

Vue-cli是vue项目的脚手架。脚手架帮我们写好了很多vue基础代码工具

目录结构 本地调试 代码部署 热加载

通过全局命令npm install -g vue-cli 安装脚手架。

1.安装vue-cli全局环境：github vue-cli

2.创建项目文件夹 如：d:/VueProgram

3.进入文件夹，vue init webpack-simple my\_progect(工程名)初始化一个vue+webpack项目

4.安装node依赖包：在my\_progect文件夹里执行命令：npm install --save

5.npm run dev 启动项目。

## 一．全局API：

Vue.extend()方法：

Vue.extend(options);

类似于jQuery的extend方法，使用vue构造器，创建一个“子类”，参数是对象。

注意：在Vue.extend（）方法创建的子类中，data必须是函数。创建完子类之后，再new出来挂载到一个元素上，用于动态改变dom元素的内容。

// 创建构造器

var Profile = Vue.extend({

template: '<p>{{firstName}} {{lastName}} aka {{alias}}</p>',

data: function () {

return {

firstName: 'Walter',

lastName: 'White',

alias: 'Heisenberg'

}

}

})

// 创建 Profile 实例，并挂载到一个元素上。

new Profile().$mount('#mount-point')

Vue.nextTick([callback,context])方法：

在下次DOM元素更新循环结束之后执行延迟回调，在修改数据时使用这个方法获取更新后的DOM元素。

参数：callback：function类型；contect：object类型。

用法：Vue.nextTick（function（）{

}）

①

<div id="example">{{message}}</div>

var vm = new Vue({

el: '#example',

data: {

message: '123'

}

})

vm.message = 'new message' // 更改数据

vm.$el.textContent === 'new message' // false

Vue.nextTick(function () {

vm.$el.textContent === 'new message' // true

})

②

Vue.component('example', {

template: '<span>{{ message }}</span>',

data: function () {

return {

message: 'not updated'

}

},

methods: {

updateMessage: function () {

this.message = 'updated'

console.log(this.$el.textContent) // => '没有更新'

this.$nextTick(function () {

console.log(this.$el.textContent) // => '更新完成'

})

}

}

})

Vue.component()方法：

该方法用于注册一个全局DOM对象或局部DOM对象，与templete一起配合使用，该方法会覆盖原DOM元素的内容。注意：data属性在此范围内必须是函数。

// 注册组件，传入一个扩展过的构造器

Vue.component('my-component', Vue.extend({ /\* ... \*/ }))

// 注册组件，传入一个选项对象（自动调用 Vue.extend）

Vue.component('my-component', { /\* ... \*/ })

// 获取注册的组件（始终返回构造器）

var MyComponent = Vue.component('my-component')

局部注册：

Compontmts：{

“tagName”:{

Template：“<div>hhahah</div>”

}

Data：function(){

}

}

比如局部注册：

<div id="com">

<my-component>{{尼克斯电脑防守打法}}</my-component>

</div>

new Vue({

el:"#com",

components:{

"my-component":{

template:"<div>hellwwwlwlwlwl</div>"

}

}

})

全局注册：

<div id="co">

<iss>{{时间的话决定好}}</iss>

</div>

Vue.component("iss",{

template:"<div>memeda</div>"

}

)

new Vue({

el:"#co"

})

[Vue.set( object, key, value )方法：](http://cn.vuejs.org/v2/api/" \l "Vue-set)

[设置对象的属性。如果对象是响应式的，确保属性被创建后也是响应式的，同时触发视图更新。这个方法主要用于避开 Vue 不能检测属性被添加的限制。](http://cn.vuejs.org/v2/api/" \l "Vue-set)

[注意对象不能是 Vue 实例，或者 Vue 实例的根数据对象，即不能对new出来的vue对象或者其对象的根数据对象使用，可以对vue.extend()方法构造出来的对象使用。](http://cn.vuejs.org/v2/api/" \l "Vue-set)

二：Vue的数据与方法：

除了 data 属性， Vue 实例暴露了一些有用的实例属性与方法。el属性，data属性，methods属性等都属于实例属性，也就是vue自带的属性。这些属性与方法都有前缀 $，以便与代理的 data 属性区分。

例如：

var vm = new Vue({

el: '#example',

data: {

objs:[

{message:"你好"},

{message:"你好"},

{message:"你好"},

{message:"你好"}

]

}

vm.$data === data // -> true

vm.$el === document.getElementById('example') // -> true

// $watch 是一个实例方法

vm.$watch('a', function (newVal, oldVal) {

// 这个回调将在 `vm.a` 改变后调用

})

Console.log（vm.$el）;

Console.log（vm.$data）;

Console.log（vm.$data.objs[0].message）;

或者Console.log（vm.objs[0].message）;

new Vue({

el: '#app',

data: {

message: 'Hello Vue.js!'

}

})

el属性：标志vue.js的势力范围。el：“#id名”----通过id名来控制该元素内的vue元素。

el:”#id”,

Data属性：对象属性集合，动态书写占位符里的内容。

Data:{

items：[

{Obj.attr:”值”}，

{Obj.attr:”值”}，

{Obj.attr:”值”}，

{Obj.attr:”值”}，

{Obj.attr:”值”}，

......

]

}

methods属性：方法集合。

methods：{

函数1名：function（）{

}，

函数2名：function（）{

}，

函数3名：function（）{

}，

函数4名：function（）{

}

......

}

## 二．JavaScript 表达式：

{{ number + 1 }}

{{ ok ? 'YES' : 'NO' }}

{{ message.split('').reverse().join('') }}

## 三．Vue的指令：

<div id="app">{{ message }}</div>

<input v-model="message" placeholder="edit me">

v-model指令：v-model=“message”

在行内样式里添加v-model指令来控制双向数据绑定。

V-for指令：控制多个元素中的单个元素。

v-for=“item in object”

<span>{{item}}</span>

①

Object：{

Key1:value1,

Key2:value2,

Key3:value3,

......

}

在这里item是一个对象，key是其属性，可以随便命名。

②v-for=“item in object”

<span>{{item.message}}</span>

Item:{

Item.message=“value1”

Item.message=“value2”

Item.message=“value3”

......

}

二者区别在于：如果占位符里是子对象的属性的话，那么data中的项必须是占位符里的对象属性，值 这样的键值对形式（适用于同一个对象的不同属性）；如果占位符里是子对象的话，那么data中的项可以为任意属性，值 这种键值对形式存在（适用于不同类型的对象）。

V-if指令：根据布尔值插入元素，返回布尔值。true：插入；false：不插入。

在行内样式里添加V-for指令来完成列表的渲染。

v-on：click/mouseover/..=“函数名”。

<span v-once>This will never change: {{ msg }}</span>

## Vue.js组件与数据渲染：

组件（Component）是 Vue.js 最强大的功能之一。组件可以扩展 HTML 元素，封装可重用的代码。在较高层面上，组件是自定义元素， Vue.js 的编译器为它添加特殊功能。在有些情况下，组件也可以是原生 HTML 元素的形式，以 js 特性扩展。

组件的注册：

要注册一个全局组件，你可以使用Vue.component

Vue.component(tagName,template：{ <options>

})

组件在注册之后，便可以在父实例的模块中以自定义元素。例如：

Html部分：

<div id="example1">

<my-component></my-component>

</div>

Js部分：

Vue.component("my-component",{

template:'<div>A custom component!</div>'

})

new Vue({

el:"#example1"

})

## VUE的方法与生命周期：

data：function（）{

return{

//初始化本页面数据

}

},

生命周期：（固定的函数，在不同时段调用）

[beforeCreate](https://cn.vuejs.org/v2/api/" \l "beforeCreate)：function（）{}//在创建本页面之前调用

[created](https://cn.vuejs.org/v2/api/" \l "created)：function（）{}//在创建本页面之后调用

[beforeMount](https://cn.vuejs.org/v2/api/" \l "beforeMount)：function（）{}//在本页面组件渲染之前调用

[mounted](https://cn.vuejs.org/v2/api/" \l "mounted)：function（）{}//在本页面组件渲染之后调用

[beforeUpdate](https://cn.vuejs.org/v2/api/" \l "beforeUpdate)：function（）{}//在本页面组件个更新数据之前调用

[updated](https://cn.vuejs.org/v2/api/" \l "updated)：function（）{}//在本页面组件个更新数据之后调用

各种方法：

methods：{

Click1：function（）{

},

Click2：function（）{

},

。。。。

},

## VUE的页面跳转与页面传参

值得一提的是，与react不同，react里，把想要传递的参数与pathname写到同一个对象里然后push一下就OJ8K了，但VUE的route传参有自己的格式，name单例，params为了传递参数而专门设置了一个pramas对象，与name共同传递到$rout()里面。是固定写法。

传参页面：将参数写到params对象里，与name一起传递。

//跳到第一个页面

Tiaozhuan:function(){

var pathName={

name: 'HelloWorld',

params:{

id:1234321321,

name:"xiandou",

a:1,

b:2,

c:"哈哈哈哈",

},

}

this.$router.push(pathName);

}，

目的页面接收：在本页面里打印一下$route就OJ8K了。

created:function(){

console.log(this.$route);

},

子组件像父页面传值：e.target.innerHtml/value;

除了使用 <router-link> 创建 a 标签来定义导航链接，我们还可以借助 router 的实例方法，通过编写代码来实现。

## router.push(location)

想要导航到不同的 URL，则使用 router.push 方法。这个方法会向 history 栈添加一个新的记录，所以，当用户点击浏览器后退按钮时，则回到之前的 URL。

当你点击 <router-link> 时，这个方法会在内部调用，所以说，点击 <router-link :to="..."> 等同于调用 router.push(...)。

声明式：<router-link :to="...">

编程式：router.push(...)

该方法的参数可以是一个字符串路径，或者一个描述地址的对象。

// 字符串

router.push('home')

// 对象

this.$router.push({path: '/login?url=' + this.$route.path});

// 命名的路由

router.push({ name: 'user', params: { userId: 123 }})

// 带查询参数，变成/backend/order?selected=2

this.$router.push({path: '/backend/order', query: {selected: "2"}});

// 设置查询参数

this.$http.post('v1/user/select-stage', {stage: stage})

   .then(({data: {code, content}}) => {

      if (code === 0) {

        // 对象

        this.$router.push({path: '/home'});

      }else if(code === 10){

        // 带查询参数，变成/login?stage=stage

        this.$router.push({path: '/login', query:{stage: stage}});

      }

});

// 设计查询参数对象

let queryData = {};

if (this.$route.query.stage) {

  queryData.stage = this.$route.query.stage;

}

if (this.$route.query.url) {

  queryData.url = this.$route.query.url;

}

this.$router.push({path: '/my/profile', query: queryData});

**replace**

类型: boolean

默认值: false

设置 replace 属性的话，当点击时，会调用 router.replace() 而不是 router.push()，于是导航后不会留下 history 记录。即使点击返回按钮也不会回到这个页面。

//加上replace: true后，它不会向 history 添加新记录，而是跟它的方法名一样 —— 替换掉当前的 history 记录。

this.$router.push({path: '/home', replace: true})

//如果是声明式就是像下面这样写：

<router-link :to="..." replace></router-link>

// 编程式:

router.replace(...)

例如：

this.$router.push({path: '/coach/' + this.$route.params.id, query: queryData})

# 微信小程序与公众号

小程序API：<https://developers.weixin.qq.com/miniprogram/dev/index.html>

## 1.配置文件app.json

这个json文件相当于react、VUE的路由，存放了该项目的所有页面。

### Pages：

所有的页面路径。

### Window

设置默认页面的窗口表现，即微信头部的标题。它的属性有：

navigationBarBackgroundColor：默认导航头部的背景颜色。

navigationBarTextStyle：微信头部的字体颜色。目前仅支持black和white两种。

navigationBarTitleText String类型， 导航栏标题文字内容

backgroundColor：窗口的背景色。

enablePullDownRefresh：bool类型，是否开启下拉刷新

onReachBottomDistance Number类型，页面上拉触底事件触发时距页面底部距离，单位为px。

### tabBar

如果小程序是一个多 tab 应用（客户端窗口的底部或顶部有 tab 栏可以切换页面），可以通过 tabBar 配置项指定 tab 栏的表现，以及 tab 切换时显示的对应页面。

1.当设置 position 为 top 时，将不会显示 icon

2.tabBar 中的 list 是一个数组，只能配置最少2个、最多5个 tab，tab 按数组的顺序排序。

Color：tab栏的字体颜色。

selectedColor：选中的字体颜色。

backgroundColor：tab的背景颜色

borderStyle:tab的边框颜色，目前仅支持black与white。

List：Array类型，tab 的列表，详见 list 属性说明，最少2个、最多5个 tab。

list 接受一个数组，数组中的每个项都是一个对象，其属性值如下：

pagePath:必填，页面路径，必须在 pages 中先定义

Text：必填，Tabbar上的按钮文字。

iconPath：选填，图片路径，icon 大小限制为40kb，建议尺寸为 81px \* 81px，当 postion 为 top 时，此参数无效，不支持网络图片

selectedIconPath：选中时的图片路径，icon 大小限制为40kb，建议尺寸为 81px \* 81px ，当 postion 为 top 时，此参数无效

Position：tabbar的定位。目前只支持bottom与top

### networkTimeout：

可以设置各种网络请求的超时时间。其参数有：

Request：[wx.request](https://developers.weixin.qq.com/miniprogram/dev/api/network-request.html)的超时时间，单位毫秒，默认为：60000

connectSocket：[wx.connectSocket](https://developers.weixin.qq.com/miniprogram/dev/api/network-socket.html)的超时时间，单位毫秒，默认为：60000

uploadFile：[wx.uploadFile](https://developers.weixin.qq.com/miniprogram/dev/api/network-file.html" \l "wxuploadfileobject)的超时时间，单位毫秒，默认为：60000

downloadFile：[wx.downloadFile](https://developers.weixin.qq.com/miniprogram/dev/api/network-file.html" \l "wxdownloadfileobject)的超时时间，单位毫秒，默认为：60000

例如：

{

"pages": [

"pages/drawguess/index",

"pages/index/index",

"pages/chat/chat",

"pages/upload/index"

],

"window": {

"backgroundTextStyle": "light",

"navigationBarBackgroundColor": "#f8777b",

"navigationBarTitleText": "腾讯云小程序解决方案",

"navigationBarTextStyle": "white"

}

}

## 2.注册程序app()方法

在app.js里，App() 函数用来注册一个小程序。接受一个 object 参数，其指定小程序的生命周期函数等。

Object参数有：

onLaunch：生命周期函数--监听小程序初始化，小程序初始化时执行，我们初始化客户端的登录地址，获取开发者的用户信息。

onShow：生命周期函数--监听小程序显示。当小程序启动，或从后台进入前台显示，会触发 onShow方法。

onHide 生命周期函数--监听小程序隐藏，当小程序从前台进入后台，会触发 onHide

App({

onLaunch: function(options) {

// Do something initial when launch.

},

onShow: function(options) {

// Do something when show.

},

onHide: function() {

// Do something when hide.

},

onError: function(msg) {

console.log(msg)

},

globalData: 'I am global data'

})

## 事件的使用方式

### 1.bindtap

当用户点击该组件的时候会在该页面对应的Page中找到相应的事件处理函数。

<view id="tapTest" data-hi="WeChat" bindtap="tapName"> Click me! </view>

在相应的Page定义中写上相应的事件处理函数，参数是event。

Page({

tapName: function(event) {

console.log(event)

}

});

可以看到log出来的信息大致如下：

{"type":"tap","timeStamp":895,"target": {

"id": "tapTest",

"dataset": {

"hi":"WeChat"

}

},"currentTarget": {

"id": "tapTest",

"dataset": {

"hi":"WeChat"

}

},"detail": {

"x":53,

"y":14

},"touches":[{

"identifier":0,

"pageX":53,

"pageY":14,

"clientX":53,

"clientY":14

}],"changedTouches":[{

"identifier":0,

"pageX":53,

"pageY":14,

"clientX":53,

"clientY":14

}]

}

Touchstart 手指触摸动作开始

Touchmove 手指触摸后移动

Touchcancel 手指触摸动作被打断，如来电提醒，弹窗

Touchend 手指触摸动作结束

Tap 手指触摸后马上离开

Longpress 手指触摸后，超过350ms再离开，如果指定了事件回调函数并触发了这个事件，tap事件将不被触发

Longtap 手指触摸后，超过350ms再离开（推荐使用longpress事件代替）

Transitionend 会在 WXSS transition 或 wx.createAnimation 动画结束后触发

Animationstart 会在一个 WXSS animation 动画开始时触发

Animationiteration 会在一个 WXSS animation 一次迭代结束时触发

Animationend 会在一个 WXSS animation 动画完成时触发

Touchforcechange 在支持 3D Touch 的 iPhone 设备，重按时会触发

### 事件的参数target

Id 事件源组件的id

tagName 当前组件的类型

[dataset](https://developers.weixin.qq.com/miniprogram/dev/framework/view/wxml/event.html" \l "dataset) 事件源组件上由data-开头的自定义属性组成的集合

## WXS 模块

WXS 代码可以编写在 wxml 文件中的 <wxs> 标签内，或以 .wxs 为后缀名的文件内。相当于在html中的script标签与引入外部的js文件一样。

每一个 .wxs 文件和 <wxs > 标签都是一个单独的模块。每个模块都有自己独立的作用域。即在一个模块里面定义的变量与函数，默认为私有的，对其他模块不可见。

一个模块要想对外暴露其内部的私有变量与函数，只能通过 module.exports 实现。

### .wxs 文件

在微信开发者工具里面，右键可以直接创建 .wxs 文件，在其中直接编写 WXS 脚本。

示例代码：

// /pages/comm.wxs

var foo = "'hello world' from comm.wxs";var bar = function(d) {

return d;

}module.exports = {

foo: foo,

bar: bar

};

上述例子在 /pages/comm.wxs 的文件里面编写了 WXS 代码。该 .wxs 文件可以被其他的 .wxs 文件 或 WXML 中的 <wxs> 标签引用。

#### module 对象以及属性：

每个 wxs 模块均有一个内置的 module 对象。

exports: 通过该属性，可以对外共享本模块的私有变量与函数。

示例代码：

// /pages/tools.wxs

var foo = "'hello world' from tools.wxs";var bar = function (d) {

return d;

}module.exports = {

FOO: foo,

bar: bar,

};

module.exports.msg = "some msg";

<!-- page/index/index.wxml -->

<wxs src="./../tools.wxs" module="tools" />

<view> {{tools.msg}} </view>

<view> {{tools.bar(tools.FOO)}} </view>

页面输出：

some msg

'hello world' from tools.wxs

#### require函数

在.wxs模块中引用其他 wxs 文件模块，可以使用 require 函数。

引用的时候，要注意如下几点：

1.只能引用 .wxs 文件模块，且必须使用相对路径。

2.wxs 模块均为单例，wxs 模块在第一次被引用时，会自动初始化为单例对象。多个页面，多个地方，多次引用，使用的都是同一个 wxs 模块对象。

如果一个 wxs 模块在定义之后，一直没有被引用，则该模块不会被解析与运行。

var tools = require("./tools.wxs");

console.log(tools.FOO);

console.log(tools.bar("logic.wxs"));

console.log(tools.msg);

## <wxs> 标签及其属性：

1. module 当前 <wxs> 标签的模块名，必填字段。module 属性是当前 <wxs> 标签的模块名。在单个 wxml 文件内，建议其值唯一。有重复模块名则按照先后顺序覆盖（后者覆盖前者）。不同文件之间的 wxs 模块名不会相互覆盖。

2.Src 引用 .wxs 文件的相对路径。仅当本标签为单闭合标签或标签的内容为空时有效。

引用的时候，要注意如下几点：

1.只能引用 .wxs 文件模块，且必须使用相对路径。

2.wxs 模块均为单例，wxs 模块在第一次被引用时，会自动初始化为单例对象。多个页面，多个地方，多次引用，使用的都是同一个 wxs 模块对象。

3.如果一个 wxs 模块在定义之后，一直没有被引用，则该模块不会被解析与运行。

## 4.在微信小程序中使用Echarts图表库

使用Echarts图表库需要下载微信小程序的echarts项目，然后改源码基础上做改动。

echarts项目地址：<https://github.com/ecomfe/echarts-for-weixin>

循环添加对象属性：这种方法添加不上去

var baseObj={a:0,b:1}  
var curArr=[];  
for(var k=0;k<4;k++){  
curArr.push(baseObj);  
curArr[k].key=k;  
}  
console.log(curArr);

# GitHub与Git命令

Git是目前世界上最先进的分布式版本控制系统（没有之一）。

## 在Widows上安装Git

在我的百度云盘里有Git命令的安装包与[Notepad++](http://notepad-plus-plus.org/" \t "https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/_blank)的安装包，下载，按照默认步骤安装即可。

安装完之后，随便找一个文件夹，单击鼠标右键，鼠标列表中会多出Git GUI Here 和Git Blash Here两个选项。

Git GUI Here是新建Git仓库和复制Git仓库之类的。

Git Blash Here是在这个文件夹下进行Git命令行操作。

鼠标左键单击 Git Blash Here ，弹出一个类似命令窗口之类的小黑框，说明Git安装成功。

然后配置你的全局用户名和邮箱。

$ git config --global user.name "Your Name"

$ git config --global user.email "email@example.com"

## 创建版本仓库

什么是版本库呢？版本库又名仓库，英文名****repository****，你可以简单理解成一个目录，这个目录里面的所有文件都可以被Git管理起来，每个文件的修改、删除，Git都能跟踪，以便任何时刻都可以追踪历史，或者在将来某个时刻可以“还原”。

1.创建一个文件夹learngit作为仓库的根目录，注意不要用汉语名称，以免造成意外的错误。

2.进入文件夹，鼠标右键，然后单击Git Blash Here，出现Git命令窗口

3.$ git init，这样learngit文件夹就称为一个真正的git仓库了。打开隐藏的文件，可以看到隐藏的.git文件夹。不要修改这个文件夹里的东西，否则你的仓库就改乱了。

### 把文件添加到版本库

首先这里再明确一下，所有的版本控制系统，其实只能跟踪文本文件的改动，比如TXT文件，网页，所有的程序代码等等，Git也不例外。版本控制系统可以告诉你每次的改动，比如在第5行加了一个单词“Linux”，在第8行删了一个单词“Windows”。而图片、视频这些二进制文件，虽然也能由版本控制系统管理，但没法跟踪文件的变化，只能把二进制文件每次改动串起来，也就是只知道图片从100KB改成了120KB，但到底改了啥，版本控制系统不知道，也没法知道。

使用Windows的童鞋要特别注意：

千万不要使用Windows自带的****记事本****编辑任何文本文件。原因是Microsoft开发记事本的团队使用了一个非常弱智的行为来保存UTF-8编码的文件，他们自作聪明地在每个文件开头添加了0xefbbbf（十六进制）的字符，你会遇到很多不可思议的问题，比如，网页第一行可能会显示一个“?”，明明正确的程序一编译就报语法错误，等等，都是由记事本的弱智行为带来的。建议你下载[Notepad++](http://notepad-plus-plus.org/" \t "https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/_blank)代替记事本，不但功能强大，而且免费！记得把Notepad++的默认编码设置为UTF-8 without BOM即可：格式-->以UTF-8无 BOM格式编码。

在learngit文件夹（或者其子目录）下编写一个readme.txt文件，随便写点啥。因为这是一个Git仓库，放到其他地方Git再厉害也找不到这个文件，把一个文件放到Git仓库只需要两步。

1.$ git add readme.txt

2.$ git commit -m "写了一个readme文件"。

git add <file>，注意，可反复多次使用，添加多个文件；

git commit -m <message>：版本提交说明，-m后跟你的版本说明文字，可输入任意内容，当然要是有意义的，可以帮助你从历史记录里方便找改动记录。

暂存区与分支区：

第一步是用git add把文件添加进去，实际上就是把文件修改添加到暂存区；

第二步是用git commit提交更改，实际上就是把暂存区的所有内容提交到当前分支。

因为我们创建Git版本库时，Git自动为我们创建了唯一一个master分支，所以，现在，git commit就是往master分支上提交更改。所以，git add命令实际上就是把要提交的所有修改放到暂存区（Stage），然后，执行git commit就可以一次性而且只是把暂存区的所有修改提交到分支，与你的本地文件没关系。

你可以简单理解为，需要提交的文件修改通通放到暂存区，然后，再从暂存区一次性提交所有的修改。所以，每当你在本地修改了文件，你都要重新git add<filename> 和git commit ”message”两个步骤。

## 版本回滚

第一种：git reset --hard HEAD^

HEAD是当前版本，HEAD^是上一个版本，HEAD^^是上上一个版本，以此类推，往前数100个版本是HEAD~100。

第二种：git log +git reset --hard<hashcode>

git log命令显示从最近到最远的提交日志，会显示你历次提交的版本hashcode值和你的版本发布说明。这个hashcode是一个16进制的hash值，有很多位。

git reset --hard<hashcode>：还原到 hashcode的版本，这个hashcode随便前几位就行，git会根据你输入的几位找到对应的hashcode版本。

## 从Git上覆盖原版本（撤销修改）

git checkout -- <filename>

命令git checkout -- readme.txt意思就是，把readme.txt文件在工作区的修改全部覆盖，这里有两种情况：

一种是readme.txt自修改后还没有被放到暂存区，现在，覆盖就回到和版本库一模一样的状态；

一种是readme.txt已经添加到暂存区后，又作了修改，现在，覆盖就回到添加到暂存区后的状态。

总之，就是让这个文件回到最近一次git commit或git add时的状态，让本地文件与暂存区的文件保持一致。

## 删除文件

rm <filename>本地文件资源管理器中的文件就和版本库中的不一样了。此时有两种情况：

1. 你确认删除版本库里的文件：

此时，你可以 git commit 更新git版本库中的文件。

1. 你不小心误删了某个文件：

此时，你可以Git checkout -- <filename>来从git上下载并恢复到你本地。

添加远程仓库

git remote add origin your-git-url

## 推送到远程Github仓库

用Sourcetree

# ESlint代码规范

## .eslintrc.js：

module.exports = {

'env': {

'browser': true,

'es6': true

},

'extends': 'airbnb',

'parserOptions': {

'ecmaFeatures': {

'jsx': true

},

'ecmaVersion': 2018,

'sourceType': 'module'

},

'plugins': [

'react',

],

'rules': {

'indent': [//验证代码每一行缩进,4个空格

'warn',

4

],

'guard-for-in': 'off',//禁止for-in 已修改，循环对象时要用（虽然性能很差）

'no-restricted-syntax': 'off',//禁止某些javascript语法，可配置

'brace-style': [2, 'stroustrup', { 'allowSingleLine': true }],//强制执行Stroustrup风格 允许一个块打开和关闭括号在同一行上 webpack.config.js中--fix可自动修复

'react/jsx-indent':[//验证JSX中的缩进,4个空格

'error',

4

],

'react/jsx-indent-props':[//验证JSX中的props缩进,4个空格

'error',

4

],

'linebreak-style': 'off',//["error", "windows"],换行风格"unix"（默认）强制使用Unix行结尾：\n对于LF。"windows"强制使用Windows行结束符：\r\n用于CRLF。

'quotes': [// 语句强制单引号

'error',

'single'

],

'semi': [// 语句强制分号结尾

'error',

'always'

],

'react/prefer-stateless-function': 'off',//强制将类声明为一个纯函数

'jsx-ally/anchor-is-valid':[// 在组件中不能使用a标签，跳转页面或者跳转锚点时只能用Button或js //已修改

'off',

{

components: ['Link'],

specialLink: ['to'],

aspects: ['noHref', 'invalidHref', 'preferButton']

}

],

'no-const-assign':'error',//已修改 禁止修改const声明的变量

'no-use-before-define':'off',//已修改 改成warn或off符合自己代码习惯

'no-plusplus':'off',//已修改 //禁止使用++、--

'no-unused-expressions':'error',//已修改 //禁止无用的表达式

'no-mixed-operators':'off',//已修改 禁止使用混合运算符（逻辑或数学运算符）

'no-param-reassign':'error',//禁止给参数重新赋值

"max-len":["off", { "code": 200 }], //某行最大的列数 可配置 自相矛盾

'react/jsx-no-undef':'error',//已修改//禁止使用undefined的变量

'no-mixed-spaces-and-tabs':'error',//已修改 禁止使用混合的tab和spaces

'no-console': 'warn',//待修改 禁止console

'no-unused-vars':'off',//不能有声明后未被使用的变量或参数 已修改,自相矛盾

'react/jsx-uses-vars': 'error',// 使用var声明

'class-methods-use-this': 'off',//保持 使用this

'prefer-destructuring': ['off'],//已修改 //强制使用数组/对象解构

'react/destructuring-assignment': ['error'],//已修改，使用解构赋值的形式

'react/jsx-no-bind': ['error'],//禁止使用bind(this)方法

'react/no-unused-state': ['error'],//修改 不使用无用的state

'no-nested-ternary': 'error',//禁止使用嵌套的三目运算 :嵌套三元表达式会使代码更难理解。

'react/prop-types': 'off',//使用propTypes暴露 props里的属性 已修改

'eqeqeq': 'error',//使用全等 已修改

'camelcase': 'error',//强制驼峰法命名

'react/sort-comp': 'off',//已修改 函数排序 没必要固定函数顺序

'jsx-a11y/alt-text': 'error',//已修改 强制所有的alt解释属性。

'no-tabs': 'error',//已修改使用数组索引是一个坏主意，因为它不能唯一地标识您的元素。如果对数组进行排序或将元素添加到数组的开头，则即使表示该索引的元素可能相同，也会更改索引。这会导致不必要的渲染。

'react/no-array-index-key': 'error',//Eslint-plugin-react

'no-useless-concat': 'error',//禁止无用的concat连接

'no-shadow':'error'// 禁止在内部重复声明全局变量

}

};

// eslint内置规则文档：https://eslint.org/docs/rules/

// https://www.npmjs.com/package/eslint-config-airbnb

// airbnb规则文档：https://github.com/yannickcr/eslint-plugin-react/tree/master/docs/rules

## Webpack.config.js:

module: {

// 兼容jsx

rules: [

//...其他规则

{

test: /\.js|.jsx$/,

loader: 'eslint-loader',

enforce: 'pre',

include: [path.resolve(\_\_dirname, 'src')], // 指定检查的目录

options: { // 这里的配置项参数将会被传递到 eslint 的engine

// formatter: require('eslint-friendly-formatter') // 指定错误报告的格式规范

fix: true,

},

},

//...其他规则

]

},

# TypeScript-Javascript的超集

## 安装TypeScript

npm install -g typescript。如果你使用了Visual Studio Code 的话，它自身封装了ts的环境插件。

## Ts举例：

function greeter(person: string) {

return "Hello, " + person;

}

let user = "Jane User";

document.body.innerHTML = greeter(user);

## 运行ts：

tsc greeter.ts

## **类型注解**

TypeScript里的类型注解是一种轻量级的为函数或变量添加约束的方式。它规定你了变量的类型，如果变量的实际类型与定义的类型注解不符，那么ts文件将不会被编辑成功。

function greeter(person: string) {

return "Hello, " + person;

}

let user = "Jane User";

document.body.innerHTML = greeter(user);

## **接口**

让我们开发这个示例应用。这里我们使用接口来描述一个拥有firstName和lastName字段的对象。 在TypeScript里，只在两个类型内部的结构兼容那么这两个类型就是兼容的。 这就允许我们在实现接口时候只要保证包含了接口要求的结构就可以，而不必明确地使用 implements语句。

接口规定了传递的参数或者某些值的类型，只要不违背接口定义的值类型，就不会报错。

interface Person {

firstName: string;

lastName: string;

}

function greeter(person: Person) {

return "Hello, " + person.firstName + " " + person.lastName;

}

let user = { firstName: "Jane", lastName: "User" };

document.body.innerHTML = greeter(user);

### 可选属性

接口里的属性不全都是必需的。 有些是只在某些条件下存在，或者根本不存在。 可选属性在应用“option bags”模式时很常用，即给函数传入的参数对象中只有部分属性赋值了。

interface SquareConfig {

color?: string;

width?: number;

}

function createSquare(config: SquareConfig): {color: string, area: number} {

let newSquare = {color: "white", area: 100};

if (config.color) { newSquare.color = config.color; }

if (config.width) { newSquare.area = config.width \* config.width; }

return newSquare;

}

let mySquare = createSquare({color: "black"});

### 额外的属性检查

在上面的例子中，我们定义了接口SquareConfig ,里面包含了color和width两个可选属性，又在createSquare的传参规则中定义了参数必须遵循SquareConfig 的接口类型。但是如果在createSquare函数里我们多传了其他的属性怎么办?比如我们传了{color: "black",width: 100, a: 1},这样的话就会报错。解决这样的问题，我们可以在定义接口时预留出额外属性的标签签名，并规定多余的参数的类型。即：

interface SquareConfig {

color?: string,

width?: number,

[propname: string]: any;

}

[propname: string]: any;是规定了多余的属性可以传任何类型。

### 只读属性

一些对象属性只能在对象刚刚创建的时候修改其值。 你可以在属性名前用 readonly来指定只读属性:

interface Point { readonly x: number; readonly y: number; }

TypeScript具有ReadonlyArray<T>类型，它与Array<T>相似，只是把所有可变方法去掉了，因此可以确保数组创建后再也不能被修改：

### readonly VS const

最简单判断该用readonly还是const的方法是看要把它做为变量使用还是做为一个属性。 做为变量使用的话用const，若做为属性则使用readonly。

### 函数类型的属性

接口能够描述JavaScript中对象拥有的各种各样的外形。 除了描述带有属性的普通对象外，接口也可以描述函数类型。

为了使用接口表示函数类型，我们需要给接口定义一个调用签名。 它就像是一个只有参数列表和返回值类型的函数定义。参数列表里的每个参数都需要名字和类型。

interface SearchFunc {

(source: string, subString: string): boolean;//冒号之前的括号是函数的参数类型，冒号后面是函数的返回类型。

}

这样定义后，我们可以像使用其它接口一样使用这个函数类型的接口。 下例展示了如何创建一个函数类型的变量，并将一个同类型的函数赋值给这个变量。

let mySearch: SearchFunc;

mySearch = function(source: string, subString: string) {

let result = source.search(subString);

return result > -1;

}

let mySearch: SearchFunc; mySearch = function(src: string, sub: string): boolean {

let result = src.search(sub);

return result > -1;

}

函数的参数会逐个进行检查，要求对应位置上的参数类型是兼容的。 如果你不想指定类型，TypeScript的类型系统会推断出参数类型，因为函数直接赋值给了 SearchFunc类型变量。 函数的返回值类型是通过其返回值推断出来的（此例是 false和true）。 如果让这个函数返回数字或字符串，类型检查器会警告我们函数的返回值类型与SearchFunc接口中的定义不匹配。

let mySearch: SearchFunc; mySearch = function(src, sub) { let result = src.search(sub); return result > -1; }

### 可索引的类型的属性

与使用接口描述函数类型差不多，我们也可以描述那些能够“通过索引得到”的类型，比如a[10]或ageMap["daniel"]。 可索引类型具有一个 索引签名，它描述了对象索引的类型，还有相应的索引返回值类型。 让我们看一个例子：

interface StringArray {

[index: number]: string;

}

let myArray: StringArray;

myArray = ["Bob", "Fred"];

let myStr: string = myArray[0];

上面例子里，我们定义了StringArray接口，它具有索引签名。 这个索引签名表示了当用 number去索引StringArray时会得到string类型的返回值。

TypeScript支持两种索引签名：字符串和数字。 可以同时使用两种类型的索引，但是数字索引的返回值必须是字符串索引返回值类型的子类型。 这是因为当使用 number来索引时，JavaScript会将它转换成string然后再去索引对象。 也就是说用 100（一个number）去索引等同于使用"100"（一个string）去索引，因此两者需要保持一致。

class Animal { name: string; }

class Dog extends Animal { breed: string; } // 错误：使用数值型的字符串索引，有时会得到完全不同的Animal!

interface NotOkay { [x: number]: Animal; [x: string]: Dog; }

字符串索引签名能够很好的描述dictionary模式，并且它们也会确保所有属性与其返回值类型相匹配。 因为字符串索引声明了 obj.property和obj["property"]两种形式都可以。 下面的例子里， name的类型与字符串索引类型不匹配，所以类型检查器给出一个错误提示：

interface NumberDictionary {

[index: string]: number;

length: number; // 可以，length是number类型

name: string // 错误，`name`的类型与索引类型返回值的类型不匹配 }

最后，你可以将索引签名设置为只读，这样就防止了给索引赋值：

interface ReadonlyStringArray {

readonly [index: number]: string;

}

let myArray: ReadonlyStringArray = ["Alice", "Bob"];

myArray[2] = "Mallory"; // error!

你不能设置myArray[2]，因为索引签名是只读的。

## **类-Class**

TypeScript支持JavaScript的新特性，比如支持基于类的面向对象编程。

让我们创建一个Student类，它带有一个构造函数和一些公共字段。 注意类和接口可以一起共作，程序员可以自行决定抽象的级别。

还要注意的是，在构造函数的参数上使用public等同于创建了同名的成员变量。

class Student {

fullName: string;

constructor(public firstName, public middleInitial, public lastName) {

this.fullName = firstName + " " + middleInitial + " " + lastName;

}

}

interface Person { firstName: string; lastName: string; }

function greeter(person : Person) {

return "Hello, " + person.firstName + " " + person.lastName;

}

let user = new Student("Jane", "M.", "User");

document.body.innerHTML = greeter(user);

### 继承

class Animal {

move(distanceInMeters: number = 0) {

console.log(`Animal moved ${distanceInMeters}m.`);

}

}

class Dog extends Animal {

bark() {

console.log('Woof! Woof!');

}

}

const dog = new Dog();

dog.bark();

dog.move(10);

dog.bark();

这个例子展示了最基本的继承：类从基类中继承了属性和方法。 这里， Dog是一个 派生类，它派生自 Animal 基类，通过 extends关键字。 派生类通常被称作 子类，基类通常被称作 超类。

因为 Dog继承了 Animal的功能，因此我们可以创建一个 Dog的实例，它能够 bark()和 move()。

### public、private与protected

Public标记的变量可以被其他的类所调用，但是private标记的变量只是这个类私有的变量，不能被其他类的方法所调用。

class Animal {

public name: string;

public constructor(theName: string) {

this.name = theName;

}

public move(distanceInMeters: number) {

console.log(`${this.name} moved ${distanceInMeters}m.`);

}

}

构造函数也可以被标记成 protected。 **这意味着这个类不能在包含它的类外被实例化，但是能被继承。**

class Person {

protected name: string;

protected constructor(theName: string) {

this.name = theName;

}

}

// Employee 能够继承 Person

class Employee extends Person {

private department: string;

constructor(name: string, department: string) {

super(name);

this.department = department;

}

public getElevatorPitch() {

return `Hello, my name is ${this.name} and I work in ${this.department}.`;

}

}

let howard = new Employee("Howard", "Sales");

let john = new Person("John"); // 错误: 'Person' 的构造函数是被保护的.

### 静态属性-static

到目前为止，我们只讨论了类的实例成员，那些仅当类被实例化的时候才会被初始化的属性。 我们也可以创建类的静态成员，这些属性存在于类本身上面而不是类的实例上。 每个实例想要访问这个属性的时候，直接用类名调用。

静态属性包括静态变量与静态方法：

class MyObj {

static a:number =1;

static b:number =2;

constructor(public a:string,public b:string){

// this.a=a;

// this.b=b;

}

static open():number {

return MyObj.a+MyObj.b;

}

}

const myo=new MyObj('1','2');

console.log(myo);

console.log(MyObj.open());//3

## Ts基础变量类型

### 布尔值（boolean）

最基本的数据类型就是简单的true/false值，在JavaScript和TypeScript里叫做boolean（其它语言中也一样）。

let isDone: boolean = false;

### 数字（number）

和JavaScript一样，TypeScript里的所有数字都是浮点数。 这些浮点数的类型是 number。 除了支持十进制和十六进制字面量，TypeScript还支持ECMAScript 2015中引入的二进制和八进制字面量。

let decLiteral: number = 6;

let hexLiteral: number = 0xf00d;

let binaryLiteral: number = 0b1010;

let octalLiteral: number = 0o744;

### 字符串（string）

JavaScript程序的另一项基本操作是处理网页或服务器端的文本数据。 像其它语言里一样，我们使用 string表示文本数据类型。 和JavaScript一样，可以使用双引号（ "）或单引号（'）表示字符串。

let name: string = "bob";

name = "smith";

### 数组（[ ]）

TypeScript像JavaScript一样可以操作数组元素。 有两种方式可以定义数组。 第一种，可以在元素类型后面接上[]，表示由此类型元素组成的一个数组：

let list: number[] = [1, 2, 3];

或者用数组泛型：

let list: Array<number> = [1, 2, 3];

### 元组 Tuple

元组类型允许表示一个已知元素数量和类型的数组，各元素的类型不必相同。 比如，你可以定义一对值分别为string和number类型的元组。

// 声明一个元素类型

let x: [string, number];

// 初始化

x = ['hello', 10]; // OK

// 错误的初始化

x = [10, 'hello']; // Error

console.log(x[0].substr(1)); // OK

console.log(x[1].substr(1)); // Error, 'number' does not have 'substr'

当访问一个越界的元素，会使用联合类型替代：

x[3] = 'world'; // OK, 字符串可以赋值给(string | number)类型

console.log(x[5].toString()); // OK, 'string' 和 'number' 都有 toString

x[6] = true; // Error, 布尔不是(string | number)类型

### 枚举enum

enum类型是对JavaScript标准数据类型的一个补充。 像C#等其它语言一样，使用枚举类型可以为一组数值赋予友好的名字。

enum Color {Red, Green, Blue}

let c: Color = Color.Green; // c = 1

export enum DialogMode {

Create,

Edit,

View,

}

console.log(DialogMode);

// {

0: "Create",

1: "Edit",

2: "View",

Create: 0,

Edit: 1,

View: 2

}

默认情况下，从0开始为元素编号。 你也可以手动的指定成员的数值。 例如，我们将上面的例子改成从 1开始编号：

enum Color {Red = 1, Green, Blue}

let c: Color = Color.Green;

Console.log(Color.Red)// 1，这里1就是Color的Red的值，也是Color的Red的编号。

### any（任何或不清楚的类型）

有时候，我们会想要为那些在编程阶段还不清楚类型的变量指定一个类型。 这些值可能来自于动态的内容，比如来自用户输入或第三方代码库。 这种情况下，我们不希望类型检查器对这些值进行检查而是直接让它们通过编译阶段的检查。 那么我们可以使用 any类型来标记这些变量：

let notSure: any = 4;

notSure = "maybe a string instead";

notSure = false; // 正确，它可以被声明为bool

在对现有代码进行改写的时候，any类型是十分有用的，它允许你在编译时可选择地包含或移除类型检查。 你可能认为 Object有相似的作用，就像它在其它语言中那样。 但是 Object类型的变量只是允许你给它赋任意值 - 但是却不能够在它上面调用任意的方法，即便它真的有这些方法：

let notSure: any = 4;

notSure.ifItExists(); // okay, ifItExists might exist at runtime

notSure.toFixed(); // okay, toFixed exists (but the compiler doesn't check)

let prettySure: Object = 4;

prettySure.toFixed(); // Error: Property 'toFixed' doesn't exist on type 'Object'.

### void

某种程度上来说，void类型像是与any类型相反，它表示没有任何类型。 当一个函数没有返回值时，你通常会见到其返回值类型是 void：

function warnUser(): void { console.log("This is my warning message"); }

声明一个void类型的变量没有什么大用，因为你只能为它赋予undefined和null

let unusable: void = undefined;

### null 和 undefined

TypeScript里，undefined和null两者各自有自己的类型分别叫做undefined和null。 和 void相似，它们的本身的类型用处不是很大：

// Not much else we can assign to these variables! let u: undefined = undefined; let n: null = null;

默认情况下null和undefined是所有类型的子类型。 就是说你可以把 null和undefined赋值给number类型的变量。

然而，当你指定了--strictNullChecks标记，null和undefined只能赋值给void和它们各自。 这能避免 很多常见的问题。 也许在某处你想传入一个 string或null或undefined，你可以使用联合类型string | null | undefined。 再次说明，稍后我们会介绍联合类型。

注意：我们鼓励尽可能地使用--strictNullChecks，但在本手册里我们假设这个标记是关闭的。

### never

never类型表示的是那些永不存在的值的类型。 例如， never类型是那些总是会抛出异常或根本就不会有返回值的函数表达式或箭头函数表达式的返回值类型； 变量也可能是 never类型，当它们被永不为真的类型保护所约束时。

never类型是任何类型的子类型，也可以赋值给任何类型；然而，没有类型是never的子类型或可以赋值给never类型（除了never本身之外）。 即使 any也不可以赋值给never。

// 返回never的函数必须存在无法达到的终点

function error(message: string): never {

throw new Error(message);

}

// 推断的返回值类型为never

function fail() {

return error("Something failed");

}

// 返回never的函数必须存在无法达到的终点

function infiniteLoop(): never {

while (true) { }

}

### object类型

bject表示非原始类型，也就是除number，string，boolean，symbol，null或undefined之外的类型。

使用object类型，就可以更好的表示像Object.create这样的API

declare function create(o: object | null): void;

create({ prop: 0 }); // OK

create(null); // OK

create(42); // Error

create("string"); // Error

create(false); // Error

create(undefined); // Error

### 类型断言

有时候你会遇到这样的情况，你会比TypeScript更了解某个值的详细信息。 通常这会发生在你清楚地知道一个实体具有比它现有类型更确切的类型。

通过类型断言这种方式可以告诉编译器，“相信我，我知道自己在干什么”。 类型断言好比其它语言里的类型转换，但是不进行特殊的数据检查和解构。 它没有运行时的影响，只是在编译阶段起作用。 TypeScript会假设你，程序员，已经进行了必须的检查。

类型断言有两种形式。 其一是“尖括号”语法：

let someValue: any = "this is a string";

let strLength: number = (<string>someValue).length;

另一个为as语法：

let someValue: any = "this is a string";

let strLength: number = (someValue as string).length;

两种形式是等价的。 至于使用哪个大多数情况下是凭个人喜好；

但是，当你在TypeScript里使用JSX时，只有 as语法断言是被允许的.

## 函数类型

function add(x: number, y: number): number {

return x + y;

}

let myAdd = function(x: number, y: number): number {

return x + y;

};

函数类型包含两部分：参数类型和返回值类型。 当写出完整函数类型的时候，这两部分都是需要的。 我们以参数列表的形式写出参数类型，为每个参数指定一个名字和类型。 这个名字只是为了增加可读性。

### 可选参数和默认参数

TypeScript里的每个函数参数都是必须的。 这不是指不能传递 null或undefined作为参数，而是说编译器检查用户是否为每个参数都传入了值。 编译器还会假设只有这些参数会被传递进函数。 简短地说，传递给一个函数的参数个数必须与函数期望的参数个数一致。

JavaScript里，每个参数都是可选的，可传可不传。 没传参的时候，它的值就是undefined。 在TypeScript里我们可以在参数名旁使用 ?实现可选参数的功能。 比如，我们想让last name是可选的

function buildName(firstName: string, lastName?: string) {

if (lastName) return firstName + " " + lastName;

else return firstName;

}

let result1 = buildName("Bob"); // works correctly now

let result2 = buildName("Bob", "Adams", "Sr."); // error, too many parameters

let result3 = buildName("Bob", "Adams"); // ah, just right

在TypeScript里，我们也可以为参数提供一个默认值当用户没有传递这个参数或传递的值是undefined时。 它们叫做有默认初始化值的参数。 让我们修改上例，把last name的默认值设置为"Smith"。

function buildName(firstName: string, lastName = "Smith") {

return firstName + " " + lastName;

}

let result1 = buildName("Bob"); // works correctly now, returns "Bob Smith"

let result2 = buildName("Bob", undefined); // still works, also returns "Bob Smith"

let result3 = buildName("Bob", "Adams", "Sr."); // error, too many parameters

let result4 = buildName("Bob", "Adams"); // ah, just right

### 剩余参数

必要参数，默认参数和可选参数有个共同点：它们表示某一个参数。 有时，你想同时操作多个参数，或者你并不知道会有多少参数传递进来。 在JavaScript里，你可以使用 arguments来访问所有传入的参数。

在TypeScript里，你可以把所有参数收集到一个变量里：

function buildName(firstName: string, ...restOfName: string[]) {

return firstName + " " + restOfName.join(" ");

}

let employeeName = buildName("Joseph", "Samuel", "Lucas", "MacKinzie");

### this

## 泛型

如果我们想使函数返回any类型，可以这样定义：

function identity（arg：any）：any{

Return arg；

}

使用any类型会导致这个函数可以接收任何类型的arg参数，这样就丢失了一些信息：传入的类型与返回的类型应该是相同的。如果我们传入一个数字，我们只知道任何类型的值都有可能被返回。

因此，我们需要一种方法使返回值的类型与传入参数的类型是相同的。 这里，我们使用了 类型变量，它是一种特殊的变量，只用于表示类型而不是值。

function identity<T>(arg: T): T {

return arg;

}

我们给identity添加了类型变量T。 T帮助我们捕获用户传入的类型（比如：number），之后我们就可以使用这个类型。 之后我们再次使用了 T当做返回值类型。现在我们可以知道参数类型与返回值类型是相同的了。 这允许我们跟踪函数里使用的类型的信息。

我们把这个版本的identity函数叫做泛型，因为它可以适用于多个类型。 不同于使用 any，它不会丢失信息，像第一个例子那像保持准确性，传入数值类型并返回数值类型。

我们定义了泛型函数后，可以用两种方法使用。 第一种是，传入所有的参数，包含类型参数：

let output = identity<string>("myString"); // type of output will be 'string'

这里我们明确的指定了T是string类型，并做为一个参数传给函数，使用了<>括起来而不是()。

第二种方法更普遍。利用了类型推论 -- 即编译器会根据传入的参数自动地帮助我们确定T的类型：

let output = identity("myString"); // type of output will be 'string'

注意我们没必要使用尖括号（<>）来明确地传入类型；编译器可以查看myString的值，然后把T设置为它的类型。 类型推论帮助我们保持代码精简和高可读性。如果编译器不能够自动地推断出类型的话，只能像上面那样明确的传入T的类型，在一些复杂的情况下，这是可能出现的。

## 枚举

### 数字枚举

用法与js的Object和数组方法类似。若用Object的方法来获取某一个成员，那么得到的是成员的索引，若用数组的方法获取某一个成员，那么得到的是成员的值。

默认的枚举：enum Direction { Up, Down, Left, Right, }

Console.log（Direction.Up）//0

Console.Log(Direction[0])// “Up”

初始化下标的枚举：enum Direction { Up = 1, Down, Left, Right }

如上，我们定义了一个数字枚举， Up使用初始化为 1。 其余的成员会从 1开始自动增长。 换句话说，Direction.Up的值为 1， Down为 2， Left为 3， Right为 4。

使用枚举很简单：通过枚举的属性来访问枚举成员，和枚举的名字来访问枚举类型：

enum Response { No = 0, Yes = 1, } function respond(recipient: string, message: Response): void { // ... } respond("Princess Caroline", Response.Yes)

### 字符串枚举

用法与js的Object方法相同，只能用获取属性的方法获取成员，得到成员的值。

enum Direction {

Up = "UP",

Down = "DOWN",

Left = "LEFT",

Right = "RIGHT",

}

Console.log(Direction.Up)//"UP"

由于字符串枚举没有自增长的行为，字符串枚举可以很好的序列化。 换句话说，如果你正在调试并且必须要读一个数字枚举的运行时的值，这个值通常是很难读的 - 它并不能表达有用的信息（尽管 [反向映射](https://www.tslang.cn/docs/handbook/enums.html" \l "enums-at-runtime)会有所帮助），字符串枚举允许你提供一个运行时有意义的并且可读的值，独立于枚举成员的名字。

### 异构枚举

enum BooleanLikeHeterogeneousEnum { No = 0, Yes = "YES", }

为了不混淆，不建议用异构枚举。

### 计算的和常量成员

枚举成员使用 常量枚举表达式初始化。 常数枚举表达式是TypeScript表达式的子集，它可以在编译阶段求值。 当一个表达式满足下面条件之一时，它就是一个常量枚举表达式：

* 一个枚举表达式字面量（主要是字符串字面量或数字字面量）
* 一个对之前定义的常量枚举成员的引用（可以是在不同的枚举类型中定义的）
* 带括号的常量枚举表达式
* 一元运算符 +, -, ~其中之一应用在了常量枚举表达式
* 常量枚举表达式做为二元运算符 +, -, \*, /, %, <<, >>, >>>, &, |, ^的操作对象。 若常数枚举表达式求值后为 NaN或 Infinity，则会在编译阶段报错。

## React&Webpack

### 将ts版本的react包下载到而本地

npm install --save react react-dom @types/react @types/react-dom

使用@types/前缀表示我们额外要获取React和React-DOM的声明文件。 通常当你导入像 "react"这样的路径，它会查看react包； 然而，并不是所有的包都包含了声明文件，所以TypeScript还会查看 @types/react包。 你会发现我们以后将不必在意这些。

### 再添加ts的开发依赖：

npm install --save-dev typescript awesome-typescript-loader source-map-loader

这些依赖会让TypeScript和webpack在一起良好地工作。 awesome-typescript-loader可以让Webpack使用TypeScript的标准配置文件 tsconfig.json编译TypeScript代码。 source-map-loader使用TypeScript输出的sourcemap文件来告诉webpack何时生成 自己的sourcemaps。 这就允许你在调试最终生成的文件时就好像在调试TypeScript源码一样。

### 添加TypeScript配置文件

我们想将TypeScript文件整合到一起 - 这包括我们写的源码和必要的声明文件。

我们需要创建一个tsconfig.json文件，它包含了输入文件列表以及编译选项。 在工程根目录下新建文件tsconfig.json文件，添加以下内容：

{ "compilerOptions": { "outDir": "./dist/", "sourceMap": true, "noImplicitAny": true, "module": "commonjs", "target": "es5", "jsx": "react" }, "include": [ "./src/\*\*/\*" ] }