目录

[高阶函数 3](#_Toc536018400)

[ajax 3](#_Toc536018401)

[1创建xhr对象 3](#_Toc536018402)

[2绑定监听函数 3](#_Toc536018403)

[3打开连接 4](#_Toc536018404)

[4发送请求 4](#_Toc536018405)

[常用属性 4](#_Toc536018406)

[常用方法 5](#_Toc536018407)

[模块 6](#_Toc536018408)

[es6 6](#_Toc536018409)

[其他 6](#_Toc536018410)

[amd requirejs 6](#_Toc536018411)

[cmd seajs 6](#_Toc536018412)

[commonjs 6](#_Toc536018413)

[@import 7](#_Toc536018414)

[严格模式 7](#_Toc536018415)

[arguments.callee 7](#_Toc536018416)

[arguments.callee.caller 7](#_Toc536018417)

[文件上传 7](#_Toc536018418)

[input.file 8](#_Toc536018419)

[multiple 8](#_Toc536018420)

[files 8](#_Toc536018421)

[accept 8](#_Toc536018422)

[window.URL.createObjectURL 8](#_Toc536018423)

[注意 9](#_Toc536018424)

[总结 9](#_Toc536018425)

[两种上传方式 9](#_Toc536018426)

[img 9](#_Toc536018427)

[宽高 9](#_Toc536018428)

[onload/oncomplete 10](#_Toc536018429)

[onerror 10](#_Toc536018430)

[构造函数 10](#_Toc536018431)

[formdata 10](#_Toc536018432)

[fileReader 10](#_Toc536018433)

[onload 10](#_Toc536018434)

[上传到服务器 10](#_Toc536018435)

[fileReader的result 10](#_Toc536018436)

[直接form 11](#_Toc536018437)

[自定义form 11](#_Toc536018438)

[手动转换格式 11](#_Toc536018439)

[图片压缩 12](#_Toc536018440)

[编码相关 12](#_Toc536018441)

[blob 12](#_Toc536018442)

[window.atob 12](#_Toc536018443)

[window.btoa 12](#_Toc536018444)

[ArrayBuffer 12](#_Toc536018445)

[ArrayBuffer 12](#_Toc536018446)

[view 13](#_Toc536018447)

[常用 13](#_Toc536018448)

[dataURItoBlob 13](#_Toc536018449)

[http缓存机制 14](#_Toc536018450)

[modified 14](#_Toc536018451)

[Last-Modified 14](#_Toc536018452)

[If-Modified-Since 14](#_Toc536018453)

[if-Unmodified-Since 15](#_Toc536018454)

[etag 15](#_Toc536018455)

[配置 15](#_Toc536018456)

[使用 15](#_Toc536018457)

[浏览器机制 16](#_Toc536018458)

[If-None-Match 16](#_Toc536018459)

[If-Match 16](#_Toc536018460)

[Cache-control 17](#_Toc536018461)

[内容指令 17](#_Toc536018462)

[expires 17](#_Toc536018463)

[Pragma 17](#_Toc536018464)

[Range和If-Range 18](#_Toc536018465)

[类型相关 18](#_Toc536018466)

[函数 18](#_Toc536018467)

[NaN 18](#_Toc536018468)

[typeof 19](#_Toc536018469)

[instanceof 19](#_Toc536018470)

[constructor 19](#_Toc536018471)

[类型判断 19](#_Toc536018472)

[数组 19](#_Toc536018473)

[对象 19](#_Toc536018474)

[位运算 19](#_Toc536018475)

[按位与(&) 20](#_Toc536018476)

[按位或(|) 20](#_Toc536018477)

[按位非(~) 20](#_Toc536018478)

[按位异或(^) 20](#_Toc536018479)

[有符号左移(<<) 20](#_Toc536018480)

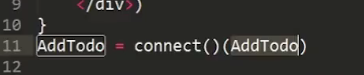
[有符号右移(>>) 20](#_Toc536018481)

[无符号右移(>>>) 21](#_Toc536018482)

[常见错误 21](#_Toc536018483)

[事件绑定赋值 21](#_Toc536018484)

# 高阶函数



函数里的返回值本身又是一个函数,addtodo传进它的返回值里面

# ajax

## 1创建xhr对象

var xmlhttp;

if(Window.X**ML**HttpRequest){ //前三个字母都是大写

xmlhttp=new XmlHttpRequest(); //是一个函数，有()

}else{

xmlhttp=new ActivObjextX(new XmlHttpRequest)

//应该是ActiveXObject(“Microsoft.XMLHTTP”)

}

## 2绑定监听函数

xmlhttp.on**r**eady**s**tatuschange=function(){

if(xmlhttp.readyS**tate**=4&&xmlhttp.status=200){

//这里可以接收的响应有：readyState、status、responseText、responseXML；

//如：alert(xmlhttp.responseText);

}

}

**注意**：当要设置返回内容格式为xml时；需要在后台在设置返回内容时，要严格设置XML内容，不然用responseXML获取返回值为null。如：

① 需要加xml文件的头部：

<?xml version='1.0' encoding='UTF-8'?>

② 设置返回值内容类型为：

response.setContentType("text/xml;charset=UTF-8");

③ 注意xml中的各个标签需要开始和结束标签对应

## 3打开连接

xmlhttp.open(type(要有引号),url,Boolean(ture为异步，false为同步))

## 4发送请求

post可以设置xmlhttp.setRequestHeader("Content-type","application/x-www-form-urlencoded");//此句代码应该在open方法之后，send方法之前

xmlhttp.send(data(get为null，不是空))

## 常用属性

|  |  |
| --- | --- |
| **属性** | **描述** |
| onreadystatechange | 存储函数（或函数名），每当 readyState 属性改变时，就会调用该函数。 |
| readyState | 存有 XMLHttpRequest 的状态。从 0 到 4 发生变化。  0: 请求未初始化。 还没有调用 open 方法  1: 服务器连接已建立。 open 方法已被调用，但 send 方法还没有被调用  2: 请求已接收。send 已被调用。请求已经开始  3: 请求处理中。服务器正在发送响应  4: 请求已完成，且响应已就绪 |
| responseText | 服务器的响应文本内容；如纯文本、html等 |
| responseXML | 服务器的响应内容，xml格式，xml dom对象；如可使用xmlDocument.getElementsByTagName("root") |
| status | 服务器发送的每一个响应的首部信息中的一个属性  常用状态码及其含义：  404 没找到页面(not found)  403 禁止访问(forbidden)  500 内部服务器出错(internal service error)  200 一切正常(ok)  304 没有被修改(not modified) |
| statusText | Http的状态响应文本（OK或Not Found等） |

## 常用方法

|  |  |
| --- | --- |
| **方法** | **描述** |
| open(method,url,async) | 规定请求的类型、URL 以及是否异步处理请求。  method：请求的类型；GET 或 POST  url：文件在服务器上的位置  async：true（异步）或 false（同步） |
| send(string) | 将请求发送到服务器。  string：仅用于 POST 请求；当为GET时可不填string对应值或者null |
| setRequestHeader(header,value) | 向请求添加 HTTP 头。  header: 规定头的名称  value: 规定头的值  **注意：**POST请求时如需提交在send方法中提交表单值；一般设置：xmlhttp.setRequestHeader("Content-type","application/x-www-form-urlencoded"); |
| abort() | 停止当前请求 |
| getAllResponseHeaders() | 把Http请求的所有响应首部作为键/值对返回 |
| getResponseHeader(header) | 返回指定首部属性的值 |

# 模块

## es6

见es6文件。

## 其他

### amd requirejs

AMD 是提前执行，CMD 是延迟执行。不过 RequireJS 从 2.0 开始，也改成可以延迟执行（根据写法不同，处理方式不同）。CMD 推崇 as lazy as possible.

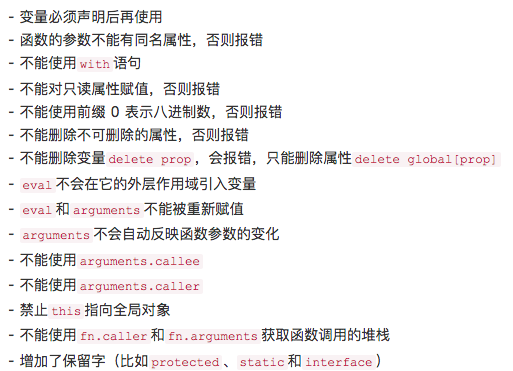
### cmd seajs

### commonjs

见node 文档。

### @import

# 严格模式



## arguments.callee

因为es3支不允许使用命名函数表达式，所以arguments.callee就不是很必要了，而且这个属性会改变函数里this的指向，本来函数里的this是指向作用域，结果变成指向Arguments.

## arguments.callee.caller

这个和f

# 文件上传

webuploader

chrome不支持两次上传同一个文件，火狐支持。

## input.file

当表单被提交, 每个选中的文件名将被添加到 URL 参数中`?file=file1.txt&file=file2.txt`

### multiple

多个文件可以被同时选中. 只要用户所在的平台允许 (e.g. 摁住 Shift 或者 Control), 用户可以选择多个文件.

### files

HTMLInputElement.files 属性返回一个 FileList 对象,这个对象是一个包含了许多 File 文件的列表

每个 File 对象包含了下列信息:

name: 文件名.

lastModified: UNIX timestamp 形式的最后修改时间.

lastModifiedDate: Date 形式的最后修改时间.

size: 文件的字节大小.

type: DOMString 文件的 MIME 类型.

### accept

accept 属性接受一个逗号分隔的 MIME 类型字符串，选择时只能选择指定类型。不使用这个属性时默认接受所有文件。

accept="image/png" or accept=".png" — 只接受 png 图片.

accept="image/png, image/jpeg" or accept=".png, .jpg, .jpeg" — PNG/JPEG 文件.

accept="image/\*" — 接受任何图片文件类型.

accept=".doc,.docx,.xml,application/msword,application/vnd.openxmlformats-officedocument.wordprocessingml.document" — 接受任何 MS Doc 文件类型.

### window.URL.createObjectURL

把blob格式转换成dataUrl格式。

image.src = window.URL.createObjectURL(curFiles[i]);

### 注意

#### chrome卡顿

'image/\*' 改成blob:http://testshop.home.mi.com/1abbf670-3924-412b-b9dc-deb9e76299b8

g'

#### 隐藏input

使用 opacity 而不是 visibility: hidden / display: none的原因是因为visibility和display会让input不可选择

## 总结

### 两种上传方式

#### window.URL.createObjectURL

const imgUrl = window.URL.createObjectURL(file);

filesDom.push(<div

*key*={`${i}${file.name}`}

*className*="item">

<img *src*={imgUrl} *alt*=""/>

</div>);

缺点：不能把握img加载的进度。通过onload可以把握进度，但还是**不能主动保存加载后的文件。**

## img

### 宽高

// 创建对象

var img = new Image();

// 改变图片的src

img.src = img\_url;

// 加载完成执行

img.onload = function(){

 // 打印

 alert('width:'+img.width+',height:'+img.height);

};

### onload/oncomplete

返回一个event对象，数据在event.target.src里面。

### onerror

### 构造函数

new Image(width, height)

## formdata

------WebKitFormBoundarydZ36hJhSN0geXC7u

Content-Disposition: form-data; name="name"; filename="name"

Content-Type: application/octet-stream

354b6be3fb49e71a8ffbca6495e38b92.jpg

------WebKitFormBoundarydZ36hJhSN0geXC7u

## fileReader

最新的safari是支持fileReader的。

### onload

结果在e.target.result。也可以调用this.result(如果不是用的箭头函数的话)

## 上传到服务器

### fileReader的result

默认是转换成base64

var reader = new FileReader();

reader.onload = function(evt){

document.getElementById('image').src = evt.target.result;

**image** = evt.target.result;

}

reader.readAsDataURL(file.files[0]);

### 直接form

form有个属性是enctype，会将文件转换成这个属性的二进制值，然后提交到后台。

### 自定义form

var formData = new FormData();

formData.append("upload\_file", file);// formData上传时自己会转换成二进制。

### 手动转换格式

function convertBase64UrlToBlob(urlData, filetype){

//去掉url的头，并转换为byte

var bytes = window.atob(urlData.split(',')[1]);

//处理异常,将ascii码小于0的转换为大于0

var ab = new ArrayBuffer(bytes.length);

var ia = new Uint8Array(ab);

var i;

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

ia[i] = bytes.charCodeAt(i);

}

return new Blob([ab], {type : filetype});

}

var formData = new FormData();

formData.append(name, convertBase64UrlToBlob(base64, fileType), filename); // 最后一个参数是上传到服务器的名称,name（键）存在时会把新值添加到已有值集合的后面。

## 图片压缩

canvas压缩是质量压缩，并没有压缩实际像素。

压缩出来的dataUrl要转成blob格式再上传。

# 编码相关

## blob

new Blob(blobParts[, options])

blobParts: 数组

从Blob中读取内容的唯一方法是使用 FileReader

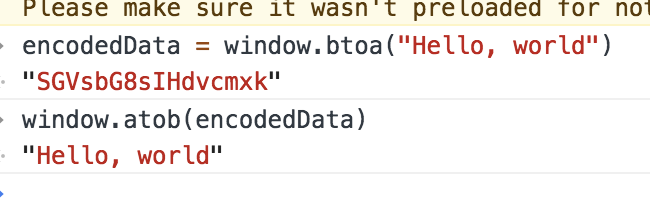
## base64

浏览器提供的base64不支持中文，涉及中文是要和后端定好格式，比如用encodeURIComponent。

可以自己实现支持中文的base64解码函数。

### window.atob

对 base64 编码过的字符串进行解码。



### window.btoa

对一个 ASCII 编码的字符串进行 base64 编码（注意：只支持 ASCII 编码，不支持汉字）

## ArrayBuffer

### ArrayBuffer

new ArrayBuffer(length)

创建一个指定了字节长度的新ArrayBuffer对象。你不能直接操纵ArrayBuffer的内容; 相反，你可以创建一个类型化的数组对象(typed array objects)或一个DataView对象，它以特定的格式表示缓冲区，并使用它读取和写入缓冲区的内容。

buffer和视图必须匹配，比如创建5个字节的buffer，创建Int16Array(1个占2个字节)，就会出错。

### view

实际是一个对象，不是数组。

直接创建

const f64a = new Float64Array(8); 8个成员，不是8个字节（64个字节）。这样创建复制时会生成新的内存。（只有基于buffer，才会共享内存）。

const x = new Int8Array([1, 1]);

const y = new Int8Array(x.buffer);



## 常用

### dataURItoBlob

function dataURItoBlob(*dataURI*) {

// convert base64 to raw binary data held in a string

// doesn't handle URLEncoded DataURIs - see SO answer #6850276 for code that does this

var byteString = atob(dataURI.split(',')[1]);

// 先转换成字符串，这样才能一个字节一个字节的拷贝

var mimeString = dataURI.split(',')[0].split(':')[1].split(';')[0]

var ab = **new** *ArrayBuffer*(byteString.length);

var ia = **new** *Uint8Array*(ab);

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

ia[i] = byteString.charCodeAt(i);

}

var blob = **new** *Blob*([ab], {type: mimeString});

return blob;

}

# http缓存机制

响应头如果是POST模式递交数据，则返回的页面大部分不会被浏览器缓存

## modified

1）在客户端第一次输入url是，服务器端会返回内容和状态码200，表示请求成功，**同时会添加一个“Last-Modified”的响应头**，表示此文件在服务器上的最后更新时间；

2）客户端第二次请求此URL时，客户端会像服务器发送请求头“If-Modified-Since”，询问服务器该时间之后当前请求内容是否有被修改过，如果服务器的内容没有变化，则自动返回HTTP304状态码（只要响应头，内容为空，这样就节省了网络带宽）

Spring提供的对Last-Modified机制的支持，只需要实现LastModified接口，并重写getLastModified方法，保证当内容发生改变时返回最新的修改时间即可。

### Last-Modified

<day-name>, <day> <month> <year> <hour>:<minute>:<second>（除了day-name和month是字母，其他都是数字。）

响应头。精确度比 ETag 要低，所以这是一个备用机制。**包含有 If-Modified-Since 或 If-Unmodified-Since 首部的条件请求会使用这个字段。**

### If-Modified-Since

请求头，值为时间。带有这个请求头的前提是以前的响应有last-modified。

当与 If-None-Match 一同出现时，它会被忽略掉，除非服务器不支持 If-None-Match。

可以用在 GET 或 HEAD 请求中。

服务器只在所请求的资源在给定的日期时间之后对内容进行过修改的情况下才会将资源返回，状态码为 200。如果请求的资源从那时起未经修改，那么返回一个不带有消息主体的304 响应，而在 Last-Modified 首部中会带有上次修改时间。

### if-Unmodified-Since

用于确保请求的资源在这段时间里没有被修改。比如断点续传。

只有当资源在指定的时间之后没有进行过修改，或者请求方式是post等非安全请求。服务器才会返回请求的资源

往往搭配If-Range使用。

## etag

http1.1的标准。服务器

W/开头表示弱验证。

### 配置

在Apache中设置Etag的支持比较简单，只需要在apache的配置中加入下面的内容就可以了：

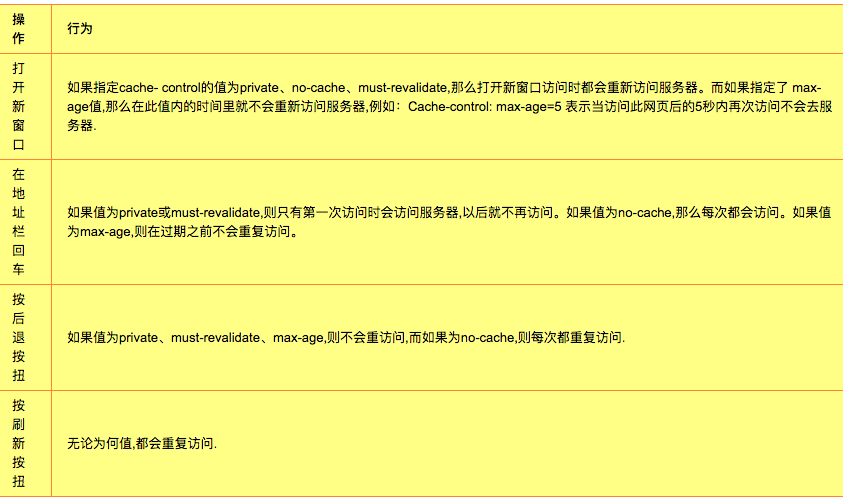
FileETag MTime Size

### 使用

对于某些资源，先用Head请求判断资源是否更新。

采用其他方法，尤其是 PUT，将 If-None-Match used 的值设置为 \* ，用来生成事先并不知道是否存在的文件，可以确保先前并没有进行过类似的上传操作，防止之前操作数据的丢失。这个问题属于更新丢失问题的一种。

## 浏览器机制



### If-None-Match

内容为etag返回回来的标志。

默认弱比较算法。(只要文件相同就相同，哪怕时间注脚不同，比如修改后又改回来。)

\*代表任意资源。

请求首部

对于 GETGET 和 HEAD 请求方法来说，当且仅当服务器上没有任何资源的 ETag 属性值与这个首部中列出的相匹配的时候，服务器端会才返回所请求的资源，响应码为200。对于其他方法来说，当且仅当最终确认没有已存在的资源的ETag 属性值与这个首部中所列出的相匹配的时候，才会对请求进行相应的处理。

对于 GET 和 HEAD 方法来说，当验证失败的时候，服务器端必须返回响应码 304 （Not Modified，未改变）。对于能够引发服务器状态改变的方法，则返回 412 （Precondition Failed，前置条件失败）。

### If-Match

默认强比较。（只有在每一个比特都相同的情况下，才可以认为两个文件是相同的。）

匹配时，也就是没有更新时才返回。

在请求方法为 GET 和 HEAD 的情况下，服务器仅在请求的资源满足此首部列出的 ETag 之一时才会返回资源。

而对于 PUT 或其他非安全方法来说，只有在满足条件的情况下才可以将资源上传。

## Cache-control

级别最高。

这个是单向的, 这意味着在请求设置的指令，在响应中不一定包含相同的指令。

### 内容指令

请求：

Cache-Control: max-age=<seconds>

Cache-Control: max-stale[=<seconds>]

Cache-Control: min-fresh=<seconds>

Cache-control: no-cache

Cache-control: no-store

Cache-control: no-transform

Cache-control: only-if-cached

响应：

Cache-control: must-revalidate

Cache-control: no-cache// 不使用缓存，即使之前已经缓存了。

Cache-control: no-store

Cache-control: no-transform

Cache-control: public //可以被任何对象缓存，包括客户端，代理服务器等。

Cache-control: private//只能被单个用户缓存，不能作为共享缓存（即代理服务器不能缓存它）。

Cache-control: proxy-revalidate

Cache-Control: max-age=<seconds>

Cache-control: s-maxage=<seconds>

## expires

指定一个日期/时间， 在这个日期/时间之后，HTTP响应被认为是过时的。

如果还有一个设置了 "max-age" 或者 "s-max-age" 指令的Cache-Control响应头，那么Expires 头就会被忽略。

## Pragma

类似cache-control,不过在HTTP 响应中的行为没有确切规范

## Range和If-Range

#### Range

请求首部，告知服务器返回文件的哪一部分。

在一个 Range 首部中，可以一次性请求多个部分，服务器会以 multipart 文件的形式将其返回。

Range: <unit>=<range-start>-, <range-start>-<range-end>

Range: bytes=200-1000, 2000-6576, 19000-

如果服务器返回的是范围响应，需要使用 206 Partial Content 状态码。

假如所请求的范围不合法，那么服务器会返回 416 Range Not Satisfiable 状态码，表示客户端错误。

服务器允许忽略 Range 首部，从而返回整个文件，状态码用 200 。

#### If-Range

请求头字段用来使得 Range 头字段在一定条件下起作用。

If-Range: <星期>, <日> <月> <年> <时>:<分>:<秒> GMT

If-Range: <etag>

当字段值中的条件得到满足时，Range 头字段才会起作用，同时服务器回复206 部分内容状态码，以及Range 头字段请求的相应部分

如果字段值中的条件没有得到满足，服务器将会返回 200 OK 状态码，并返回完整的请求资源。

# 类型相关

## 函数

### NaN

保留字（表明数据类型不能转换成数字）将某些不能强制转换为数值的非数值转换为数值的时候，也会得到NaN

如果把 NaN 与任何值（包括其自身）相比得到的结果均是 false，所以要判断某个值是否是 NaN，不能使用 == 或 === 运算符。

isNaN() 函数通常用于检测 parseFloat() 和 parseInt() 的结果,不能转换成数字为true，其他都是false

isNaN('5') false

isNaN('x') true

判断： a == a 为false,则是NaN,也可能是对象{} === {}

### typeof

返回字符串："number"，"string"，"boolean"，"object"，"function"，"undefined"（可用于判断变量是否存在）。 但 typeof 的能力有限，其对于Date、RegExp类型返回的都是"object"。注意null也是object。

### instanceof

object instanceof constructor(构造函数)

检测 constructor.prototype 是否存在于参数 object 的原型链上

({}) instanceof Object true

[] instanceof Object true

### constructor

每个对象都有一个constructor属性，它引用了初始化该对象的构造函数

## 类型判断

### 数组

Array.isArray(obj)

### 对象

obj instanceof Object && obj.length === undefined

# 位运算

平时的数值运算，其实是要先转换成二进制再进行运算的，而位运算就是直接进行二进制运算，所以位运算的执行效率肯定是更高的。

**优先级比===还要低，所以一定要加括号。**

### 按位与(&)

function assert(n) {

if (n & 1) {

console.log("n是奇数");

} else {

console.log("n是偶数");

}

}

assert(3); // "n是奇数"

对的，因为1其他位都是0，相与后都是0；

### 按位或(|)

### 按位非(~)

### 按位异或(^)

### 有符号左移(<<)

如果要求2的n次方，可以这样：

function power(n) {

return 1 << n;

}

power(5); // 32

1的二进制是01，左移5位就是0100000，十进制就是2的5次方32。

### 有符号右移(>>)

求一个数的二分之一：

var num = 64 >> 1; // 32

**有符号左移与右移不会影响符号位。**

### 无符号右移(>>>)

正数的无符号右移与有符号右移结果是一样的。负数的无符号右移会把符号位也一起移动，而且无符号右移会把负数的二进制码当成正数的二进制码：

所以，我们可以利用无符号右移来判断一个数的正负：

function isPos(n) {

return (n === (n >>> 0)) ? true : false;

}

isPos(-1); // false

isPos(1); // true

-1>>>0虽然没有向右移动位数，但-1的二进制码已经变成了正数的二进制码：

11111111111111111111111111111111

所以-1>>>0的值为4294967295。

# 常见错误

## 事件绑定赋值

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

pageItem[i].onclick=function(e){

index=i;

scrollPage();

}

}

**最后每一个index都等于4**

**原因：做循环的时候 ，只是对function进行静态解析，没有执行，所以index并没有赋值**

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

**var m=i;**

pageItem[i].onclick=function(e){

index=m;

scrollPage();

}

}

这样是**一样的**，也有错误。

**解决**

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

**pageItem[i].index=i;**

pageItem[i].onclick=function(e){

index=**this**.index;//必须通过this，不用涉及i相关。

console.log("21")

scrollPage();

}

}