2

cookies localStorage sessionStorage

在同源的http请求中携带 是 否 否

储存地 浏览器端 浏览器端 浏览器端

大小 少于4k 5m或更大 5m或更大

作用域 所有同源窗口 所有同源窗口 当前窗口及

同源子窗口

有效期 过期时间之前 始终有效 当前窗口及

同源子窗口

关闭之前

3 iframe的调用包括以下几个方面

1. iframe页面调用主页面：parent.
2. 主页面调用iframe页面：通过选择器选择iframe页面.contentWindow.
3. 主页面包含的iframe们之间相互调用：

parent.通过选择器选择iframe页面.contentWindow.

4 http状态码

1XX 信息响应类，表示接受到请求并且继续处理

2XX 处理成功响应类，表示动作被成功接收，理解和接受

3XX 重定向响应类，为了完成指定的动作，必须接受进一步处理

4XX 客户端错误，客户请求包含语法错误或者是不能正确执行

5XX 服务端错误，服务器不能正确执行一个正确的请求

100 "continue"

101 "switching protocols"

102 "processing"

200 "ok"

201 "created"

202 "accepted"

203 "non-authoritative information"

204 "no content"

205 "reset content"

206 "partial content"

207 "multi-status"

300 "multiple choices"

301 "moved permanently"

302 "moved temporarily"

303 "see other"

304 "not modified"

305 "use proxy"

307 "temporary redirect"

400 "bad request"

401 "unauthorized"

402 "payment required"

403 "forbidden"

404 "not found"

405 "method not allowed"

406 "not acceptable"

407 "proxy authentication required"

408 "request time-out"

409 "conflict"

410 "gone"

411 "length required"

412 "precondition failed"

413 "request entity too large"

414 "request-uri too large"

415 "unsupported media type"

416 "requested range not satisfiable"

417 "expectation failed"

418 "i'm a teapot"

422 "unprocessable entity"

423 "locked"

424 "failed dependency"

425 "unordered collection"

426 "upgrade required"

428 "precondition required"

429 "too many requests"

431 "request header fields too large"

500 "internal server error"

501 "not implemented"

502 "bad gateway"

503 "service unavailable"

504 "gateway time-out"

505 "http version not supported"

506 "variant also negotiates"

507 "insufficient storage"

509 "bandwidth limit exceeded"

510 "not extended"

511 "network authentication required"

日注连升经缓转

主引范授从 接字编语 如匹没改不范 用大代

位服变年警三代再

允过刷e扩上 内类范编长语位5

5 http的头域包含通用头域，请求头域，响应头域和实体信息的实体头域四个部分

通用头域：请求和响应消息都支持的头域

通用头域包含以下字段

Date消息发送的时间

Pragma实现指定的指令，最常用的是Pragma:no-cache在HTTP/1.1协议中，它

的含义和Cache-Control:no-cache相同

Connection是否需要持久连接

Upgrade向服务器指定某种传输协议以便服务器进行转换

Via通知中间网关或代理服务器地址何种通信协议

Cache-Control缓存机制

Transfer-Encoding 传输编码

请求消息的第一行为下面的格式

Method Request-URL HTTP-Version

Method包括GET，POST，HEAD，OPTIONS，PUT，DELETE，TRACE

请求头域可能包含下列字段

Host 请求资源的Internet主机和端口号

Referer 请求uri的源资源地址

Range 请求实体的一个或者多个子范围

Authorization 授权信息

From 请求发送者的email地址

Accept 浏览器可接受的MIME类型

Accept-Charset 浏览器可接受的字符集

Accept-Encoding 浏览器能够进行解码的数据编码方式，比如gzip

Accept-Language 浏览器所希望的语言种类，当服务器能够提供一种以上的语言

版本时要用到

If-Match 只有请求内容与实体相匹配才有效

If-None-Match 如果内容未改变返回304代码，参数为服务器先前发送的Etag，

与服务器回应的Etag比较判断是否改变

If-Modified-Since 如果请求的部分在指定时间之后被修改则请求成功，未被修

改则返回304代码

If-Unmodified-Since 只有实体在指定时间之后未被修改才请求成功

If-Range 如果实体未改变，服务器发送客户端丢失的部分，否则发送整个实体。

参数也为Etag

User-Agent 包含发出请求的用户信息

Max-Forwards 限制信息通过代理和网关传送的时间

Proxy-Authorization 连接到代理的授权证书

响应消息的第一行为下面的格式

HTTP-Version Status-Code Reason-Phrase

响应头域可能包含下列字段

Location 重定向接受者到一个新URL地址

Server 处理请求的原始服务器的软件信息

Vary 告诉下游代理是使用缓存响应还是从原始服务器请求

Age 从原始服务器到代理缓存形成的估算时间（以秒计，非负）

Warning 警告实体可能存在的问题

WWW-Authenticate 客户端应该在Authorization头中提供何种类型的授权信

息，在包含401的应答中必需

Proxy-Authenticate 它指出认证方案和可应用到代理的该URL上的参数

Retry-After 如果实体暂时不可取，通知客户端在指定时间之后再次尝试

实体信息：请求消息和响应消息都可以包含实体信息，实体信息一般由实体头域和实体组成

实体信息的实体头域可能包含下列字段

Allow 服务器的请求方法（如GET，POST等）

Expires 文档过期时间，之后不再缓存

Refresh 表示浏览器应该在多少时间之后刷新文档，以秒计

Etag 请求变量的实体标签的当前值

extension-header 允许客户端定义新的实体头域

Last-Modified 服务器上保存内容的最后修订时间

Content-Type 向接收方指示实体的介质类型

Content-Range 整个实体中的一部分的插入位置及整个实体的长度

Content-Encoding 文档编码

Content-Length 表示内容长度

Content-Language 响应体的语言

Content-Location 请求资源可替代的备用的另一地址

Content-MD5 返回资源的MD5校验值

6 浏览器安全

保护cookie：加上HTTPOnly属性的cookie字段，document.cookie无法进行读

写

控制台注入代码

XSS攻击：在输入内容中嵌入js代码，浏览器显示时执行该js代码

防范：对script标签尖括号进行转义

CSRF（也叫作XSRF）攻击：跨站请求伪造

防范：使用验证码或者token进行校验

网络劫持攻击：数据在中间代理层被截获

防范：（1）使用https进行加密

（2）使用非对称加密即客户端加密，只有服务器能解开

钓鱼：（1）诱使用户通过链接访问自己网站并输入重要信息

（2）诱使用户通过链接访问自己网站a页，在用户浏览a页时通过

window.opener.location=””;将原正常网页修改成自己网站仿制

正常网页的b页，诱使用户输入重要信息

7 TCP和UDP区别

TCP UDP

首部开销 **2**0字节 8字节

是否连接 **连**接（发送数据之前 无连接（发送数据之前

需要建立连接） 不需要建立连接）

是否可靠 **可**靠（无差错，不丢失，不重复） 不可靠（有差错，会丢失，有重复）

面向对象 **字**节流 报文

使用对象 **一**对一 一对一，一对多，多对一，多对多

2连可字一

8 正则表达式

^

$

\*

? （1）匹配零次或一次 （2）非贪婪

+

{n} n是非负整数。匹配n次

{n , } n是非负整数。至少匹配n次

{n , m} m和n均为非负整数，n<m，最少匹配n次且最多匹配m次

(pattern) 匹配pattern并获取，最多使用9次，程序获取为RegExp.$1到

RegExp.$9。正则表达式内获取为\1到\9。例如(.)\1匹配两个连续

的相同字符

(?:pattern) 匹配pattern不获取

(?=pattern)和(?!pattern) 是正向预查和负向预查

|

[] 只匹配其中任意一个字符

[^] 只匹配其中不包含的任意一个字符

- a-z或A-Z或0-9表示从a到z或A到Z或0到9任意一个

\f 匹配一个换页符

\n 匹配一个换行符

\r 匹配一个回车符

\t 匹配一个制表符

\v 匹配一个垂直制表符

\cx 匹配由x指明的控制字符，x的值必须为a-z或A-Z之一，否则将c视为一

个原义的c字符

\

\s 匹配任何空白字符，等价于[\f\n\r\t\v]

\S 匹配任何非空白字符，等价于[^\f\n\r\t\v]

\d 匹配一个数字字符

\D 匹配一个非数字字符

\w 匹配任何单词字符，等价于[A-Za-z0-9\_]

\W 匹配任何非单词字符，等价于[^A-Za-z0-9\_]

\b 匹配一个单词边界，指单词和空格间的位置

\B 匹配非单词边界

. 匹配除\n之外的任何单个字符

9 web前端优化（减设合资精嵌懒 外异头请跳避 到维易作访拼选）

一页面级优化

1减少http请求数

1. 从设计实现层面简化页面
2. 合理设置http缓存：缓存越多越久越好。很少变化的资源通过HTTP Header中的Expires设置长过期，可能会变的资源通过Last-Modified做请求验证
3. 资源合并与压缩
4. css spirit：把网页中背景图整合到一张图片文件中，再利用css的”background-image”,”background-repeat”,”background-position”的组合进行背景定位
5. inline images：将图片嵌入到页面或者css中。缺点：增大页面体积，无法利用浏览器缓存
6. lazy load images/javascript

2将外部脚本置底

3异步执行inline脚本

4将css放在head中

5异步请求callback

6减少不必要的http跳转

7避免重复的资源请求

二代码级优化

1javascript

1. dom：dom操作是脚本中最耗性能的一类操作
2. HTMLCollection集合在访问性能比数组差很多，尽量转化为数组后再访问或少访问，如遍历时将length属性，成员保存到局部变量后再使用
3. with(obj){}，代码块中访问非局部变量都从obj开始查找，没有再按作用域链向上查找
4. 避免使用eval，作用于字符串表示的源代码时，脚本引擎要将源代码转成可执行代码
5. 减少作用域链查找，如在循环中访问非本作用域下的变量时可在遍历之前用局部变量缓存该变量，在遍历结束后再重写该变量
6. 数据访问，javascript中的数据访问包括直接量（字符串，正则表达式），变量，对象属性以及数组，其中对直接量和局部变量的访问是最快的，对对象属性和数组的访问需要很大开销。当出现以下情况，建议将数据放入局部变量：

（a）对任何对象属性的访问超过1次

（b）对任何数组成员的访问超过1次

1. 字符串拼接，使用”+”拼接字符串效率较低，因为每次都开辟新内存。当拼接字符串较多时，可将拼接的字符串放在数组中并用join方法

2css选择符，浏览器对选择符的解析是从右向左

10 json是一种描述信息的格式

jsonp是一种调用方法，可用于跨域传输数据。利用凡是拥有src属性的标签拥有跨域的能力

原理：

服务器上有文件flightResult.aspx内容为

flightHandle({

”code”:”ca1998”,

”price”:”1780”,

”tickets”:5

})

网页上有

<script type=”text/javascript”>

var flightHandler=function(data){

alert(”你查询的航班结果是：票价”+data.price+”元”);

};

var url=”http://flightQuery.com/jsonp/flightResult.aspx?

code=ca1998&callback=flightHandler”;

var script=document.createElement(”script”);

script.setAttribute(”src”,url);

document.getElementsByTagName(”head”)[0].appendChild(script);

</script>

ajax实现:

$.ajax({

type:”get”,

async:false,

url:”http://flightQuery.com/jsonp/flightResult.aspx?

code=ca1998”,

dataType:”jsonp”,

jsonp:”callback”, //对应原理url后的callback

jsonpCallback:”flightHandler”, //对应原理url后的flightHandler

success:function(json){

alert(”你查阅到航班信息票价：”+json.price+”元”);

},

error:function(){

}

})

11 标签

<script src=””></script>

<link href=”” rel=”styleSheet” />

<script type=”text/javascript”>

</script>

<style type=”text/css”>

</style>

12

图片左移 图片上移

background:red url(””) -50px -50px no-repeat;

13定位

absolute:相对于static定位以外的第一个父元素定位 识别top，

fix: 相对于窗口定位 bottom，

relative:对默认值static的相对位移：若top，left，都为0， left，

则与static一致，若top，left不为0，则在 right，

static上相对位移。 z-index

static: 默认值 不识别top，bottom，left，right，z-index

13 css中opacity:0 到 1

无 实

14 js和jquery对比

js

静态方法：类调用的方法

实例方法：类实例调用的方法

静态方法启动时生成内存，静态内存是连续且有限制的

实例方法在程序运行过程中生成内存，实例内存是离散的

function A(){

}

1. 静态方法名=function(){

}

1. prototype.实例方法名=function(){

}

jquery:方法都是实例方法，工具函数都是静态方法

添加静态方法

$.extend(方法名)

添加实例方法

$.prototype.extend(方法名)=$.fn.extend(方法名)=$.fn.方法名

15 图片的DataURL技术

1. 原理：将图片对应的二进制代码，通过base64转换成可打印的字母数字形式
2. 源格式：data：图片类型(images/gif或images/jpg或image/png);

base64,base64转换成的数据

1. 优点

（a）当图片很小，用一个http请求太浪费，可用此

（b）当图片是服务器生成，每个用户不一致，如验证码，头像，可用此

1. 缺点

（a）base64转换过后的图片比原来大1/3

（b）DataURL形成的图片不会被浏览器缓存

16 commonjs 通过require导入模块

commonjs中 var path = require(”path”) //找到path对象

path.resolve(”a”,”b”,”c”);

//cd到a后cd到b后cd到c

17 source map是位置信息文件，存储压缩后的代码对应压缩前代码的位置。是.map文件，与源码在同一目录下。目前chrome支持，

Developer Tools ->Settings->Enable source maps

18

<meta charset=”utf-8”/> 定义文档字符编码

<meta http-equiv=”content-type” content=””/> 关联http头

set-cookie

expires

refresh

<meta name=”author” content=”” /> 关联名称

keywords

description

…

19 css直接子选择器是>

20 使用html5中history解决分享选中图片问题。不刷新操作url。history.pushState/replaceState(自定义对象，标题信息，放入的url可相对路径如”?preview=2”可绝对路径如”http://localhost/history/?preview=2”)

21 a继承b三种方式

1. prototype子类在创建对象的同时传递参数到父类不适用

a.prototype=new b();

1. apply子类在创建对象的同时传递参数到父类适用。

instanceof属于子类不属于父类

function a(构造参数1,构造参数2){

b.apply(this,[构造参数1,构造参数2]);

}

1. call+prototype子类在创建对象的同时传递参数到父类适用

instanceof属于子类也属于父类

function a (构造参数1,构造参数2){

b.call(this,构造参数1,构造参数2);

}

a.prototype=new b() //call方法只实现了方法的替换而没有进行对象属

//性的复制

a.prototype.constructor=a //a原有的对象构造信息还原

22 display

1. block 独占一行，width，height有效，margin，padding有效
2. inline 不独占一行，width，height无效，margin-top，margin-bottom，padding-top，padding-bottom无效
3. inline-block 外部呈inline，内部呈block

23 纯div上下左右居中

<body>

<div>上下左右居中</div>

</body>

div{

position:absolute;

top:50%;

left:50%;

margin-left:-200px;

margin-top:-150px;

width:400px;

height:300px;

border:1px solid red;

}

24 typeof 返回 undefined

boolean

number

string

object

Function

Symbol

25 闭包：能够读取其他函数内部变量的函数

26 js两种垃圾回收机制

（1）标记清除

（2）引用计数

27 null和undefined

null表示“没有对象”即“该处不应该有值”

undefined表示“缺少值”即“该处应该有一个值，但没有定义”

null 转数值为0

undefined 转数值为NaN

28 margin-top和margin-left 当值为%形式时都是以父元素宽为标准

29 数组去重：去掉重复

30 B继承A： B.prototype = new A()

31 call和apply：方法被谁用

1. function add(a,b){

alert(a+b);

}

function reduce(a,b){

alert(a-b);

}

add.call(reduce,1,3) //将add方法运用到reduce，结果为4

方法 被 谁用

1. 改变this指向

function b(){

alert(this);

}

b(); //window

b.call(); //window

b.call(”a”,2,3); //a

1. 实现继承

function Animal(name){

this.name=name;

this.showName = function(){

alert(this.name);

}

}

function Cat(name){

Animal.call(this.name); //将Animal用到Cat上，因此Cat有了

//Animal的所有属性和方法

}

var cat = new Cat(”Black Cat”);

cat.showName(); //弹出Black Cat

1. a.call(b,arg1,arg2);

a.apply(b,[arg1,arg2]);

两者除第二个参数外其余均一样

32 js事件委托：事件目标自身不处理事件，把处理委托给父元素或祖先元素。

33 -moz- 火狐

-ms- ie

-webkit- chrome safari

-o- opera

34 二进制

字节（一个字节等于8位二进制） 其他进制

35 编码：哪些二进制表示哪些符号

36

scroll-behavior: smooth; 滚动条平滑过渡

37



38

import Loadable from 'react-loadable';

const GenerateTags = Loadable({

loader: () =>

import(/\* webpackChunkName: "generateTags" \*/ "./GenerateTags"),

loading: LoadingSpinner

});

40 React高阶组件就是一个函数，且该函数接受一个组件作为参数，并返回一个新的组件。

41 hash 和 history

（一）

hash是url中#及后面的内容

history是在back、forward、go基础上，HTML5新增pushState()和replaceState()方法，功能是修改历史记录且不向后端发送请求

（二）

刷新hash模式，仅#之前内容包含在请求中，不会返回404

刷新history模式，全部内容包含在请求中，可能返回404

（三）

pushState() 可设置与当前url同源的任意url。hash可设置#后面的部分

pushState() 设置的url与当前url一样，会把记录加到栈中。而 hash设置的url与当前url不一样，会把记录加到栈中；

pushState() 可添加任意类型的数据。hash可添加字符串；

pushState() 可额外设置title属性供后续使用。

43

macro-task(宏任务)：包括整体代码script，setTimeout，setInterval

micro-task(微任务)：Promise，process.nextTick

44

mouseover与moseenter区别

mouseover鼠标移入元素及其子元素时会触发

mouseenter鼠标移入元素时会触发

45

0.1+0.2===0.3 false

46数组鉴别

var arr = [1,2,3];

arr instanceof Array// true

arr.constructor === Array// true

Object.prototype.toString.call(arr) === '[object Array]'// true

47

css 宽度高度等于屏幕宽度

容器的padding比例值是相对于容器的宽度值来计算的

div{

width:100%;

padding-bottom:100%;

border:solid 1px red;

}

48左定宽，右自适应布局

(1)利用position

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Document</title>

<style>

\*{

margin: 0;

padding: 0;

}

.sitebar{

width: 200px;

background-color: green;

}

.content{

position: absolute;

left: 200px;

right: 0;

top: 0;

background-color: red;

}

</style>

</head>

<body>

<div class="content">右侧主体自适应区块</div>

<div class="sitebar">左侧定宽200px区块</div>

</body>

</html>

(2)flex

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Document</title>

<style>

\*{

margin: 0;

padding: 0;

}

.main{

display: flex;

}

.content{

flex:1;

background-color: red;

}

.sitebar{

flex:0 0 200px;

order:-1;

background-color: green;

}

</style>

</head>

<body>

<div class="main">

<div class="content">右侧主体自适应区块</div>

<div class="sitebar">左侧定宽200px区块</div>

</div>

</body>

</html>

49css 三个div 宽度不定 每个间距20

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Document</title>

<style>

\*{

margin: 0;

padding: 0;

}

.main{

display: flex;

}

.content1{

flex:1;

background-color: red;

}

.content2{

margin-left:60px;

flex:1;

background-color: green;

}

.content3{

flex:1;

margin-left:60px;

background-color: yellow;

}

</style>

</head>

<body>

<div class="main">

<div class="content1">右侧主体自适应区块</div>

<div class="content2">右侧主体自适应区块</div>

<div class="content3">右侧主体自适应区块</div>

</div>

</body>

</html>

50 react router 中link 和a标签区别

对比<a>,Link组件避免了不必要的重渲染

51react component 和pure component区别

PureComponent自带通过props和state的浅对比来实现 shouldComponentUpate()，而Component没有。

51 h5 storage事件监听

（1）

网页A：监听了storage事件：

<!DOCTYPE html>

<html>

<head lang="en">

<title>A</title>

</head>

<body>

<script>

window.addEventListener("storage", function (e) {

alert(e.newValue);

});

</script>

</body>

</html>

网页B：修改了localStorage

<!DOCTYPE html>

<html>

<head lang="en">

<title>B</title>

</head>

<body>

<script>

localStorage.clear();

localStorage.setItem('foo', 'bar');

</script>

</body>

</html>

（2）如果非得要在同一网页监听怎么办？可以重写localStorage的方法，抛出自定义事件：

<!DOCTYPE html>

<html>

<head lang="en">

<title>A</title>

</head>

<body>

<script>

var orignalSetItem = localStorage.setItem;

localStorage.setItem = function(key,newValue){

var setItemEvent = new Event("setItemEvent");

setItemEvent.newValue = newValue;

window.dispatchEvent(setItemEvent);

orignalSetItem.apply(this,arguments);

}

window.addEventListener("setItemEvent", function (e) {

alert(e.newValue);

});

localStorage.setItem("nm","1234");

</script>

</body>

</html>

52 linux 新建文件 touch

53 apply，call和bind方法区别

apply 改变函数运行时上下文，调用之后立即执行，参数是数组

call 改变函数运行时上下文，调用之后立即执行，参数是列表

call 改变函数运行时上下文，调用之后返回一个新的函数需要时再调用

54

function test(){

function a(){}

function b(){}

console.log(a===a); //true

console.log(a===b); //false

console.log(a.bind(this)===a.bind(this)); //false

}

55 Promise用ok代替then;

let test=new Promise((resolve, reject) => {

console.log("1");

resolve();

});

Promise.prototype.ok=Promise.prototype.then;

test.ok(function(data){

console.log("2");

})

56 浏览器断网重连 navigator.onLine 返回true和false

移动端布局

https://www.cnblogs.com/liuranRan/p/11178066.html

设计稿宽度px(640px) clientWidth真实宽度px (500px) 真实rem（6.4rem）

------------------ = ---------------------------- = ----------------

设计稿字体px(100px) clientWidth真实字体px (78.125px) 真实1rem

定78.125px为1rem

在js代码中是无法直接获取到设备的物理像素的，需要通过document.documentElement.clientWidth \* window.devicePixelRatio来间接的获取

webpack打包速度优化

https://blog.csdn.net/mhbsoft/article/details/90213541

（1）减少搜索范围使用resolve写明全路径

（2）精确test & include & exclude

（3）使用更快插件，如ParallelUglifyPlugin替换UglifyJsPlugin

（4）设置babel的cacheDirectory 为true

（5）设置 noParse

浏览器缓存

（1）https://mp.weixin.qq.com/s?\_\_biz=MzI5MjU0Mjk5MA==&mid=2247483826&idx=2&sn=e38f5e5b928a6aa5c15371505b4e8d71&chksm=ec7e8ed3db0907c545393ea90ee8bfd7dc944ff013957228b200d66929767251b1982340d036&token=816776520&lang=zh\_CN#rd

（2）强缓存和协商缓存

浏览器加载资源，浏览器判断是否命中强缓存，如果命中强缓存（http的status为200，size为from cache），从缓存中加载资源，不发请求到服务器。如果没命中强缓存，发请求到服务器。服务器判断是否命中协商缓存，如果命中协商缓存（http的status为304），服务器返回请求，不返回资源，浏览器从缓存中加载资源。如果没命中协商缓存，浏览器从服务器加载资源

（3）强缓存

（3.1）判断是否命中强缓存用Expires和Cache-Control，Cache-Control优先级更高

（3.2）浏览器首次请求资源，服务器返回资源和在respone的header加上Expires（http1.0提出GMT格式的绝对时间），浏览器把资源和所有response的 header缓存。浏览器再次请求该资源，若再次请求时间在缓存Expires时间前，命中缓存，否则没有命中缓存。Expires是绝对时间，修改客户端时间，能影响命中缓存。所以Cache-Control（http1.1提出秒为单位的相对时间）

（3.3）浏览器首次请求资源，服务器返回资源和在respone的header加上Cache-Control，浏览器把资源和所有response的 header缓存。浏览器再次请求该资源，根据首次请求时间和Cache-Control计算出过期时间，若再次请求时间在该过期时间前，命中缓存，否则没有命中缓存

（3.4）Pragma设置页面是否缓存，值为Pragma缓存，值为no-cache不缓存

（3.5）强缓存静态资源使用，动态资源慎用。引用静态资源的html相当于动态资源，若html被缓存，该html更新后，没有机制通知浏览器该html有更新

（4）协商缓存

（4.1）判断是否命中协商缓存用Last-Modified，If-Modified-Since对和ETag、If-None-Match对

（4.2）浏览器首次请求资源，服务器返回资源和在respone的header加上Last-Modified（该资源在服务器最后修改时间）。浏览器再次请求该资源，request的header加If-Modified-Since（值是首次请求时返回的Last-Modified的值）。服务器根据该If-Modified-Since和该资源在服务器最后修改时间判断该资源是否有变化，若没有变化，命中协商缓存，在response的header不加上Last-Modified。若有变化，没命中协商缓存。Last-Modified，If-Modified-Since有时服务器资源有变化，但最后修改时间却没变，问题不容易被定位，所以ETag、If-None-Match

（4.3）浏览器首次请求资源，服务器返回资源和在respone的header加上ETag（服务器根据该资源生成唯一字符串标识）。浏览器再次请求该资源，request的header加If-None-Match（值是首次请求时返回的ETag的值）。服务器根据该If-None-Match和服务器根据该资源再一次生成唯一字符串标识判断该资源是否有变化，若没有变化，命中协商缓存，在response的header加上ETag。若有变化，没命中协商缓存。

（4.4）分布式系统多台机器间Last-Modified必须一致，以免负载均衡到不同机器导致比对失败。分布式系统尽量关掉ETag，每台机器生成的ETag不一样

（4.5）协商缓存需配合强缓存使用，如果不启用强缓存，协商缓存没有意义

（4.6）

当ctrl+f5强刷网页，跳过强缓存和协商缓存，从服务器加载资源

当f5刷新网页，跳过强缓存，判断是否命中协商缓存

选择器 https://www.runoob.com/cssref/css-selectors.html

（1）\*

表示所有

（2）#id

（3）.class

（4）标签

（5），分隔

表示左侧和右侧都

（6）空格分隔

表示左侧内所有右侧

（7）>分隔

表示左侧子第一层右侧

（8）+分隔

表示左侧后面兄弟第一个右侧

（9）~分隔

表示左侧后面兄弟所有右侧

（10）[属性名]

表示有属性名的所有

（11）[属性名=属性值]

表示属性名为属性值的所有

（12）[属性名~=属性值]

表示属性名包含属性值单词的所有

（13）[属性名\*=属性值]

表示属性名包含属性值的所有

（14）[属性名|=属性值]

表示属性名以属性值单词开头的所有

（15）[属性名^=属性值]

表示属性名以属性值开头的所有

（16）[属性名$=属性值]

表示属性名以属性值结尾的所有

（17）同级无杂:first-child

表示同级无杂第一个

（18）同级:first-of-type

表示同级第一个

（19）同级无杂:last-child

表示同级无杂最后一个

（20）同级:last-of-type

表示同级最后一个

（21）同级无杂:nth-child(n)

表示同级无杂正数第n个

（22）同级:nth-of-type(n)

表示同级正数第n个

（23）同级无杂:nth-last-child(n)

表示同级无杂倒数第n个

（24）同级:nth-last-of-type(n)

表示同级倒数第n个

（25）同级无杂:only-child

表示同级无杂父元素只有

（26）同级:only-of-type

表示同级父元素只有

（27）选择:before

表示每个选择之前插入内容

（28）选择:after

表示每个选择之后插入内容

（29）选择:first-letter

表示每个选择内第一个字母

（30）选择:first-line

表示每个选择内第一行

（31）:link

表示未访问链接

（32）:active

表示点击一瞬链接

（33）:visited

表示访问后链接

（34）:target

表示跳转到同页该位置

（35）:enabled

表示已启用的输入

（36）:disabled

表示已禁用的输入

（37）:read-write

表示可读及可写的输入

（38）:read-only

表示只读的输入

（39）:optional

表示选填的输入

（40）:required

表示必填的输入

（41）:valid

表示合法的输入

（42）:invalid

表示非法的输入

（43）:in-range

表示区间内的输入

（44）:out-of-range

表示区间外的输入

（45）:focus

表示获得焦点的输入

（46）:checked

表示已选中的输入

（47）:hover

表示悬浮

（48）::selection

表示选中

（49）选择:lang(属性值)

表示每个选择lang属性为属性值单词的所有

（50）:root

表示文档的根元素

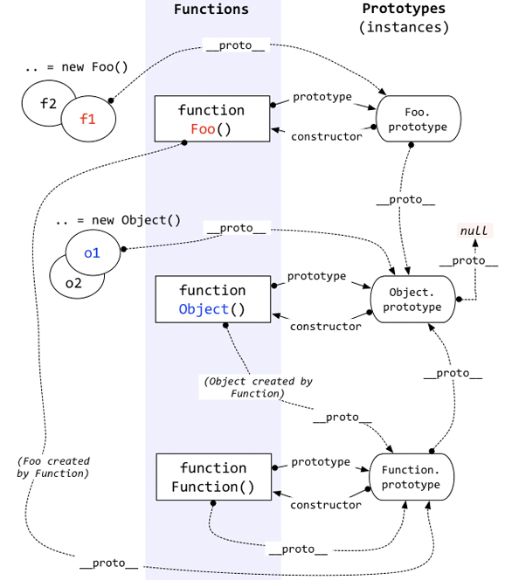
（51）选择:empty

表示每个选择没有子

（52）:not(选择)

表示不是选择的元素

js原型链 <https://www.zhihu.com/question/34183746/answer/58155878>



state直接修改的影响

react：地址未变，地址上的值改变，不触发渲染。触发渲染会将当前地址上的值正确渲染，PureComponent有作用

redux：地址未变，地址上的值改变，不触发渲染。触发渲染会将当前地址上的值正确渲染，PureComponent无作用，原因redux每次返回新对象

HTTP1.0和HTTP1.1和HTTP2.0的区别

https://segmentfault.com/a/1190000015316332

https://blog.csdn.net/ailunlee/article/details/97831912

1 HTTP1.0和HTTP1.1的区别

1.1 长连接

HTTP1.0每次请求都要创建tcp连接进行三次握手，必须用Connection：keep-alive建立长连接

HTTP1.1默认建立长连接（创建tcp连接后不关闭，后续请求继续使用，大多数浏览器允许同时建立6个长连接），不必须用Connection：keep-alive建立长连接

1.2 节约带宽

HTTP1.0客户端只需某个对象一部分，服务器将整个对象传过来，且不支持断点续传功能。

HTTP1.1支持只发header不发body，若服务器认为客户端有权限则返回100（Continue），客户端收到100发body。若服务器认为客户端没权限则返回401（Unauthorized），客户端收到401不发body

1.3 HOST域

HTTP1.0认为一台服务器绑定唯一IP，因此不支持Host请求头。

虚拟主机技术发展，一台服务器可以存在多个虚拟主机，共享一个IP。支持Host请求头，且请求头若没有Host会报400（Bad Request）。

1.4缓存处理

HTTP1.0使用Expires，If-Modified-Since判断缓存

HTTP1.1使用Etag，If-None-Match判断缓存

1.5错误通知的管理

HTTP1.1新增24个错误状态响应码，如409（Conflict）表示请求的资源与资源的当前状态发生冲突；410（Gone）表示服务器上的某个资源被永久性的删除。

2 HTTP1.1和HTTP2.0的区别

2.1 多路复用

HTTP1.1创建tcp连接后，新的http请求要在老的http请求完成后进行

HTTP2.0创建tcp连接后，可以并发进行多个http请求

2.2 头部数据压缩

HTTP1.1不支持header压缩，支持body压缩

HTTP2.0支持header压缩，支持body压缩

2.3 服务器推送

在客户端请求前，允许服务端推送资源给客户端。客户端可从本地加载该资源，不用再发请求

懒加载（lazy）实现原理

https://www.cnblogs.com/forcheng/p/13132582.html

https://blog.csdn.net/chenyu1105811473/article/details/107642435

function import(url) {

return new Promise((resolve, reject) => {

const script = document.createElement("script");

const tempGlobal = "\_\_tempModuleLoadingVariable" + Math.random().toString(32).substring(2);

script.type = "module";

script.textContent = `import \* as m from "${url}"; window.${tempGlobal} = m;`;

script.onload = () => {

resolve(window[tempGlobal]);

delete window[tempGlobal];

script.remove();

};

script.onerror = () => {

reject(new Error("Failed to load module script with URL " + url));

delete window[tempGlobal];

script.remove();

};

document.documentElement.appendChild(script);

});

}

css函数

calc()动态计算宽高

如

width:calc(100%-100px);

height:calc(100vh-100px);

post方式导出

import axios from 'axios';

axios

.post(url,data,{

responseType:'blob',

})

.then(res=>{

if(!res||!res.data){

return false;

}

const url=URL.createObjectURL(res.data);

const a=document.createElement('a');

const filename=res.headers['content-disposition']?decodeURI(res.headers['content-disposition'].split('filename=')[1]):'export';

const evt=document.createEvent('HTMLEvents'); //对firefox的兼容

evt.initEvent('click',false,false); //对firefox的兼容

a.href=url;

a.download=filename;

a.dispatchEvent(evt); //对firefox的兼容

a.click();

URL.revokeObjectURL(url);

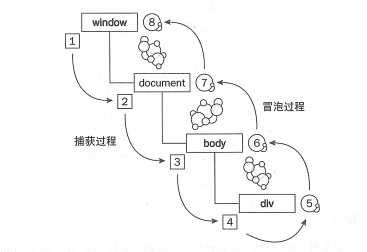
a.remove();

})

.catch(err=>{

});

js事件捕获及冒泡



addEventListener的第三个参数

第三个参数为true在js事件捕获中执行，第三个参数为false在js事件冒泡中执行，第三个参数默认为false

js事件

（一）Event.preventDefault()和Event.stopPropagation()和Event.stopImmediatePropagation()区别

Event.preventDefault()阻止默认行为，如（1）点击form submit后提交表单（2）点击a后链接跳转

Event.stopPropagation()阻止捕获或冒泡后一级及之后的传播

Event.stopImmediatePropagation()阻止捕获或冒泡同一级后一个及之后的传播

（二）React合成事件和DOM原生事件

https://juejin.im/post/6844903502729183239

https://www.jianshu.com/p/601d0cbbbc71

（三）测试代码

<html>

<head>

<meta charset="utf-8">

<title>react的合成事件</title>

</head>

<body>

<div id="app"></div>

<script crossorigin src="https://unpkg.com/react@16/umd/react.development.js"></script>

<script crossorigin src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>

<script src="https://cdn.bootcss.com/babel-standalone/6.22.1/babel.min.js"></script>

<script type="text/babel">

class MessageBox extends React.Component {

componentDidMount() {

document.addEventListener('click', (e) => {

alert('document 原生事件2')

e.stopImmediatePropagation()

})

document.addEventListener('click', (e) => {

alert('document 原生事件3')

})

document.body.addEventListener('click', (e) => {

e.stopPropagation()

alert('body 原生事件1')

})

document.body.addEventListener('click', (e) => {

// e.stopImmediatePropagation()

alert('body 原生事件2')

})

document.getElementById('parent').addEventListener('click', (e) => {

alert('parent 原生事件')

}, false)

document.getElementById('child').addEventListener('click', (e) => {

alert('child 原生事件')

// e.stopPropagation() //原生事件可以阻止合成事件

}, false)

}

parent = (e) => {

alert('父组件')

}

child = (e) => {

alert('子组件')

//e.preventDefault()

//e.stopPropagation()

//e.stopImmediatePropagation()

//e.nativeEvent.stopImmediatePropagation()

}

render() {

return (

<div id='parent' onClick={this.parent}>

父组件

<div id='child' onClick={this.child}>子组件</div>

</div>

);

}

}

document.addEventListener('click', () => {

alert('document 原生事件1')

})

ReactDOM.render(<MessageBox />, document.getElementById('app'), function () {

//console.log('渲染完成啦！！');

});

document.addEventListener('click', () => {

alert('document 原生事件4')

})

</script>

</body>

</html>

浏览器进程线程及fiber及对reac生命周期的影响

（1）浏览器进程线程

https://segmentfault.com/a/1190000012925872

（2）fiber

https://segmentfault.com/a/1190000018250127

（3）对reac生命周期的影响

https://segmentfault.com/a/1190000020035950

https://www.jianshu.com/p/0d9f558959c3

（1）浏览器进程线程

（1.1）

进程是cpu资源分配的最小单位

线程是cpu调度的最小单位

（1.2）浏览器包含的进程

（1.2.1）Browser进程：主进程，只有一个，作用：

（a）界面显示，与用户交互，如前进，后退等

（b）每个Tab页的管理，创建和销毁其他进程

（c）将Renderer进程得到的内存中的Bitmap，绘制到界面

（d）网络资源的管理，下载等

（1.2.2）第三方插件进程：每种类型的插件对应一个进程，仅当使用该插件时才创建

（1.2.3）GPU进程：最多一个，用于3D绘制等

（1.2.4）Renderer进程：渲染进程，浏览器内核，默认每个Tab页一个Renderer进程，互不影响，内部多线程，作用：

（a）页面渲染，js执行，事件循环等

（1.2.5）SharedWorker进程

（a）所有Tab页共享，所有Renderer进程共享

（1.3）浏览器有时会将多个进程合并。比如打开多个空白Tab页，浏览器会将多个空白Tab页合并成一个进程

（1.4）浏览器多进程优势

（1.4.1）避免单个page crash影响整个浏览器

（1.4.2）避免第三方插件crash影响整个浏览器

（1.4.3）多进程充分利用多核优势

（1.4.4）方便使用沙盒模型隔离插件等进程，提高浏览器稳定性

（1.5）Renderer进程包含的线程：

（1.5.1）GUI渲染线程

（a）页面渲染，解析html，css，构建dom树、style树、render树，布局和绘制等。

（b）Repaint、reflow

（c）GUI渲染线程与JS引擎线程是互斥的，GUI渲染被保存在一个队列中等到JS引擎空闲时被立即执行。如果JS引擎执行时间过长，GUI渲染阻塞，fiber解决此问题。

（1.5.2）JS引擎线程

（a）即JS内核，如V8引擎

（b）解析Javascript，运行代码。

（c）一直等待任务队列中任务到来，加以处理

（1.5.3）事件触发线程

（a）控制事件循环（JS引擎线程忙不过来，需另开线程协助）

（b）当JS引擎执行代码块如setTimeOut（也可来自Renderer进程的其他线程，如鼠标点击、AJAX异步请求等），会将对应事件添加到事件触发线程中

（c）当对应事件符合触发条件被触发时，事件触发线程会把对应事件添加到任务队列的队尾，等待JS引擎空闲时加以处理

（1.5.4）定时触发器线程

（a）setInterval与setTimeout计时

（b）W3C在HTML标准中规定setTimeout低于4ms的时间间隔算为4ms。

（1.5.5）异步http请求线程

（a）XMLHttpRequest在连接后新开一个线程请求

（1.6）GUI渲染线程与JS引擎线程是互斥的原因

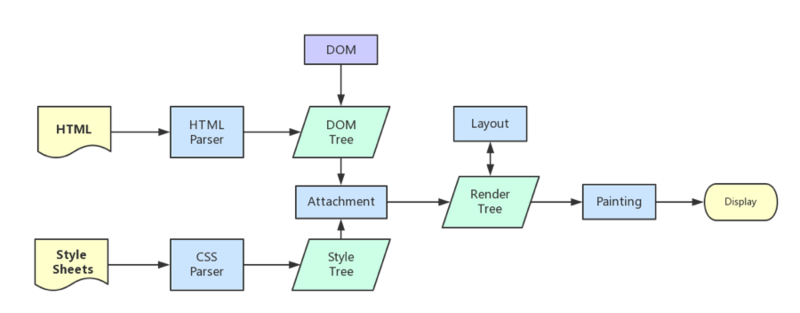
js可操作dom元素，如果js操作dom元素同时渲染界面，那么渲染结果不可预期

（1.7）WebWorker

（1.7.1）JS引擎线程的子线程，完全受JS引擎线程控制，且不能操作DOM

（1.7.2）JS引擎线程与WebWorker线程通过postMessage序列化对象通信

（1.8）Renderer进程页面渲染过程



（1.9）setTimeout不能准时执行原因：（1）事件触发线程将其添加到任务队列的队尾，队列前面还有事件未被执行（2）JS引擎未空闲

（1.10）setInterval

（1.10.1）事件触发线程每次隔精确时间将其添加到任务队列的队尾

（1.10.2）累计效应，如果下次setInterval任务已到，本次setInterval任务还没有执行完，就会导致两次任务没有间隔连续执行。

（1.10.3）就算正常间隔执行，两次setInterval任务间隔可能比预期小，因为任务执行需要一段时间

（1.10.4）浏览器最小化时，事件触发线程每次隔精确时间将其添加到任务队列的队尾，浏览器再次打开时，一瞬间全部执行

（1.11）宏任务与微任务与渲染执行顺序

一个宏任务->该宏任务所有微任务->渲染->...

（2）fiber

（2.1）Fiber指一种数据结构，可用一个纯JS对象表示：

const fiber = {

stateNode, // 节点实例

child, // 子节点

sibling, // 兄弟节点

return, // 父节点

}

（2.2）React框架内部运作分为3层：

Virtual DOM层：描述页面长什么样。

Reconciler层：调用组件生命周期方法，进行Diff计算。

Renderer层：根据不同的平台，渲染出对应的页面，如ReactDOM和ReactNative。

（2.3）fiber改动主要在Reconciler层，以前Reconciler叫Stack Reconciler，运行过程不能被打断，现在Reconciler叫Fiber Reconciler，运行过程可以被打断，让优先级更高的任务先执行。

（2.4）Fiber Reconciler执行过程中，分为2个阶段。

阶段1：render/reconciliation。生成Fiber树，得出需要更新的节点信息，是一个渐进过程，可以被打断，让优先级更高的任务先执行。包括生命周期

componentWillMount

componentWillReceiveProps

shouldComponentUpdate

componentWillUpdate

阶段2：commit。将需要更新的节点一次性批量更新，不能被打断。包括生命周期

componentDidMount

componentDidUpdate

componentWillUnmount

（2.5）fiber树

（2.5.1）Fiber Reconciler在阶段1进行Diff计算时，生成一棵Fiber树。该树在Virtual DOM树基础上增加额外信息生成，本质是一个链表。

（2.5.2）Fiber树首次渲染时一次性生成。后续需要Diff时，根据已有树和最新Virtual DOM信息，生成新树。该新树每生成一个新节点，会将控制权交还给JS引擎线程，去检查有没有优先级更高的任务需要执行。如果没有，继续构建Fiber树。如果有，丢弃正在生成的Fiber树，执行优先级更高的任务，执行完后空闲时重新生成Fiber树。

（3）对reac生命周期的影响

（3.1）由于Fiber树的生成机制，Fiber Reconciler阶段1各生命周期可能被重复执行。由于不是纯函数，执行结果不可预测。换成static getDerivedStateFromProps，是纯函数，执行结果可预测。