# 存储

|  |
| --- |
| 我们经常需要对业务中的一些数据进行存储，通常可以分为 短暂性存储 和 持久性储存。  短暂性的时候，我们只需要将数据存在内存中，只在运行时可用  持久性存储，可以分为 浏览器端 与 服务器端  1.浏览器:  cookie: 通常用于存储用户身份，登录状态等  http 中自动携带， 体积上限为 4K， 可自行设置过期时间  localStorage / sessionStorage: 长久储存/窗口关闭删除， 体积限制为 4~5M  indexDB  2.服务器: 分布式缓存 redis、数据库 |

# Web Worker

|  |
| --- |
| 现代浏览器为JavaScript创造的 多线程环境。可以新建并将部分任务分配到worker线程并行运行，两个线程可 独立运行，互不干扰，可通过自带的 消息机制 相互通信。  **基本用法:**  // 创建 workerconst worker = new Worker('work.js');  // 向主进程推送消息  worker.postMessage('Hello World');  // 监听主进程来的消息  worker.onmessage = *function* (*event*) {  *console*.log('Received message ' + event.data);  }  **限制:**  同源限制  无法使用 document / window / alert / confirm  无法加载本地资源 |

# websocket如何兼容低浏览器

|  |
| --- |
| Adobe Flash *Socket* 、  ActiveX HTMLFile (IE) 、  基于 multipart 编码发送 XHR 、  基于长轮询的 XHR |

# 文字要与进度条反色，怎么实现

|  |
| --- |
| 。。。当时我给的是js的方案，在进度条宽度变化的时候，计算盖过每一个文字的50%，如果超过，设置文字相反颜色。  当然css也有对应的方案，也就是 mix-blend-mode，我并没有接触过。  对应html也有对应方案，也就设置两个相同位置但是颜色相反的dom结构在重叠在一起，顶层覆盖底层，最顶层的进度条取overflow为hidden，其宽度就为进度。 |

# javascript 的单线程机制解释一下

|  |
| --- |
|  |

# 前端持久化的方式、区别

|  |
| --- |
|  |

# ES6怎么编译成ES5,css-loader原理,过程

|  |
| --- |
|  |

# 手动实现parseInt

|  |
| --- |
|  |

# eval是做什么的

|  |
| --- |
|  |

# GPU加速原理，哪些用了 GPU 加速(以空间换时间)

|  |
| --- |
|  |

# CSS实现宽度自适应100%，宽高16:9的比例的矩形

|  |
| --- |
|  |

# 抽离样式模块怎么写，说出思路，有无实践经验？[阿里航旅的面试题]

# 全屏滚动的原理是什么？用到了CSS的那些属性？

# 什么是响应式设计？响应式设计的基本原理是什么？如何兼容低版本的IE？

# 视差滚动效果，如何给每页做不同的动画？（回到顶部，向下滑动要再次出现，和只出现一次分别怎么做？）

# 如何修改chrome记住密码后自动填充表单的黄色背景 ？

# font-style属性可以让它赋值为“oblique” oblique是什么意思？

# position:fixed;在android下无效怎么处理？

# overflow: scroll时不能平滑滚动的问题怎么处理？

# 有一个高度自适应的div，里面有两个div，一个高度100px，希望另一个填满剩下的高度。

# png、jpg、gif 这些图片格式解释一下，分别什么时候用。有没有了解过webp？

# 什么是Cookie 隔离？（或者说：请求资源的时候不要让它带cookie怎么做）

# 实现一个圆形的可点击区域

# setTimeout和setInterval的机制

|  |
| --- |
| 因为js是单线程的。浏览器遇到etTimeout和setInterval会先执行完当前的代码块，在此之前会把定时器推入浏览器的待执行时间队列里面，等到浏览器执行完当前代码之后会看下事件队列里有没有任务，有的话才执行定时器里的代码 |

作用域 作用域链 原型链

# token、cookie、session三者的理解

|  |
| --- |
| · 1、token就是令牌，比如你授权(登录)一个程序时,他就是个依据,判断你是否已经授权该软件（最好的身份认证，安全性好，且是唯一的）用户身份的验证方式  · · 2、cookie是写在客户端一个txt文件，里面包括登录信息之类的，这样你下次在登录某个网站，就会自动调用cookie自动登录用户名服务器生成，发送到浏览器、浏览器保存，下次请求再次发送给服务器（存放着登录信息）  · · 3、session是一类用来客户端和服务器之间保存状态的解决方案，会话完成被销毁（代表的就是服务器和客户端的一次会话过程）cookie中存放着sessionID，请求会发送这个id。sesion因为request对象而产生。  ·  作者：hello等风来 链接：https://juejin.im/post/5b68f384f265da0fa00a3df0 来源：掘金 著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。 |

# 基于Token的身份验证：（最简单的token: uid用户唯一的身份识别 + time当前事件戳 + sign签名）

|  |
| --- |
| 1、用户通过用户名和密码发送请求  2、服务器端验证  3、服务器端返回一个带签名的token，给客户端  4、客户端储存token，并且每次用于发送请求  5、服务器验证token并且返回数据  每一次请求都需要token  cookie与session区别  1、cookie数据存放在客户的浏览器上，session数据放在服务器上。  2、cookie不是很安全，别人可以分析存放在本地的COOKIE并进行COOKIE欺骗考虑到安全应当使用session。  3、session会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能考虑到减轻服务器性能方面，应当使用COOKIE。  4、单个cookie保存的数据不能超过4K，很多浏览器都限制一个站点最多保存20个cookie。 |

# session与token区别

|  |
| --- |
| · 1、session认证只是把简单的User的信息存储Session里面，sessionID不可预测，一种认证手段。只存在服务端，不能共享到其他的网站和第三方App  · 2、token是oAuth Token，提供的是认证和授权，认证针对用户，授权是针对App，目的就是让某APP有权访问某用户的的信息。Token是唯一的，token不能转移到其他的App，也不能转到其他用户上。（适用app）  · 3、session的状态是存在服务器端的，客户端只存在session id， Token状态是存储在客户端的  作者：hello等风来 链接：https://juejin.im/post/5b68f384f265da0fa00a3df0 来源：掘金 著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。 |

# 、Cookie的弊端有哪些？？？（优势：保存客户端数据，分担了服务器存储的负担）

|  |
| --- |
| 1、数量和长度的限制。每个特定的域名下最多生成20个cookie（chorme和safari没有限制）2、安全性问题。 |

# 浏览器渲染原理及流程 DOM -> CSSOM -> render -> layout -> print

|  |
| --- |
| 流程：解析html以及构建dom树 -> 构建render树 -> 布局render树 -> 绘制render树概念：1.构建DOM树： 渲染引擎解析HTML文档，首先将标签转换成DOM树中的DOM node(包括js生成的标签)生成内容树2.构建渲染树： 解析对应的css样式文件信息（包括js生成的样式和外部的css）3.布局渲染树：从根节点递归调用，计算每一个元素的大小，位置等。给出每个节点所在的屏幕的精准位置4.绘制渲染树：遍历渲染树，使用UI后端层来绘制每一个节点重绘：当盒子的位置、大小以及其他属性，例如颜色、字体大小等到确定下来之后，浏览器便把这些颜色都按照各自的特性绘制一遍，将内容呈现在页面上触发重绘的条件：改变元素外观属性。如：color，background-color等重绘是指一个元素外观的改变所触发的浏览器行为，浏览器会根据元素的新属性重新绘制，使元素呈现新的外观注意：table及其内部元素需要多次计算才能确定好其在渲染树中节点的属性值，比同等元素要多发时间，要尽量避免使用table布局重排（重构/回流/reflow）： 当渲染书中的一部分（或全部）因为元素的规模尺寸，布局，隐藏等改变而需要重新构建，这就是回流。每个页面都需要一次回流，就是页面第一次渲染的时候重排一定会影响重绘，但是重绘不一定会影响重排 |

# js怎么控制一次加载一张图片，加载完后再加载下一张

|  |
| --- |
| 方法1         <script type="text/javascript">  *var* obj=new Image();              obj.src="http://www.phpernote.com/uploadfiles/editor/201107240502201179.jpg";              obj.onload=*function*(){              alert('图片的宽度为：'+obj.width+'；图片的高度为：'+obj.height);              document.getElementById("mypic").innnerHTML="<img src='"+this.src+"' />";              }              </script>              <div id="mypic">onloading……</div>   1. 方法2      <script type="text/javascript">  *var* obj=new Image();              obj.src="http://www.phpernote.com/uploadfiles/editor/201107240502201179.jpg";              obj.onreadystatechange=*function*(){              if(this.readyState=="complete"){              alert('图片的宽度为：'+obj.width+'；图片的高度为：'+obj.height);              document.getElementById("mypic").innnerHTML="<img src='"+this.src+"' />";              }              }              </script>              <div id="mypic">onloading……</div> |

# 如何实现sleep的效果（es5或者es6）

|  |
| --- |
| while循环的方式 *function* sleep(*ms*){  *var* start=*Date*.now(),expire=start+ms;  while(*Date*.now()<expire);  *console*.log('1111');  return;  }  执行sleep(1000)之后，休眠了1000ms之后输出了1111。上述循环的方式缺点很明显，容易造成死循环。 (2)通过promise来实现 function sleep(ms){  var temple=new Promise(  (resolve)=>{  console.log(111);setTimeout(resolve,ms)  });  return temple  }  sleep(500).then(function(){  //console.log(222)  })  //先输出了111，延迟500ms后输出222复制代码 (3)通过async封装 function sleep(ms){  return new Promise((resolve)=>setTimeout(resolve,ms));  }  async function test(){  var temple=await sleep(1000);  console.log(1111)  return temple  }  test();  //延迟1000ms输出了1111复制代码  ####(4).通过generate来实现  function\* sleep(ms){  yield new Promise(function(resolve,reject){  console.log(111);  setTimeout(resolve,ms);  })  }  sleep(500).next().value.then(function(){console.log(2222)}) |

# 定时器注意

|  |
| --- |
| setInterval(fn1,1000);          var num=0;          function fn1(){          console.log("hello world!"+num);          num++;          }            setInterval("fn2(100)",1000); // 没有双引号只调用一次          function fn2(a){          a++;          console.log("hello world!"+a);          }            var num2 = 10000;          setInterval(function () {          console.log("hello world!"+num2);          num2++          },1000); |

    URL(Uniform Resource Locator，统一资源定位器)

    例如：http://php.itcast.cn/about/20140706/230.html

    一个URL(网址)由四个部分构成：

        第一部分：协议，比如：http:// ftp://

            如何判断是什么服务器？主要是看安装了什么样的服务器软件？Apache软件是WWW服务器

            如果安装了FTP的软件的服务器，FTP服务器；

            如果安装了邮箱服务器的软件，那就是“邮件服务器”；

            如果安装了MySQL软件的服务器，那就是“数据库服务器”；

        第二个部分：主机名，比如：php.itcast.cn就是一个主机的二级域名(一级域名是以WWW开头的)。

        第三部分：目录名，比如：about/20140706/

        第四部分：文件名，比如：230.html

.com 顶级域名

baidu.com 一级域名

www.baidu.com 二级域名

bbs.baidu .com 二级域名

tieba.baidu .com 二级域名

最后一个"."的右边部分称为顶级域名（TLD，也称为一级域名），最后一个"."的左边部分称为二级域名（SLD），

二级域名的左边部分称为三级域名，以此类推，每一级的域名控制它下一级域名的分配

        jQuery AJAX 请求,后台收到为乱码

        可以显式设置contentType的编码为utf-8, 后台一般都能正常解码例如

        contentType: "application/x-www-form-urlencoded; charset=utf-8",

        jQuery(form).ajaxSubmit({

        url: "doLogin?user=a",

        type: "post",

        dataType: "json",

        contentType: "application/x-www-form-urlencoded; charset=utf-8",

        success: showLoginResponse

        });

        如果不加contentType: "application/x-www-form-urlencoded; charset=UTF-8", 而使用escape(),则会出现

        post 主体包含的字节比内容长度指定的字节要少

        的错误.

        2. jQuery AJAX 返回乱码

        你是否发现无论你怎么设置编码, 客户端收到的数据总是乱码呢, 请看下面的例子,你发现有什么问题吗?

        void saveTempKind(HttpServletRequest req, HttpServletResponse resp)

        throws ServletException, IOException {

        PrintWriter pw = resp.getWriter();

        resp.setContentType("text/text;charset=GBK");

        try {

        //DO PROCESS

        pw.write("成功");

        }catch(Exception e){

        pw.write("失败");

        }

        }

        问题在于resp.setContentType("text/text;charset=GBK"); 语句应该在

        PrintWriter pw = resp.getWriter();

        之前,而不是之后。如果象上例这样，无论你怎么设置response的编码，前端乱码的问题都依然如故。 可能的原因是pw在获得PriintWriter 之前,编码就已经确定了。所以编码的声明应该在获取PrintWriter对象之前

# 伪类nth-child和nth-of-type区别

|  |
| --- |
|  |

http1.0 http1.1 http2.0 https

介绍SSL和TLS

HTTPS的工作原理

HTTPS <https://github.com/airuikun/Weekly-FE-Interview/issues/2>

|  |
| --- |
| 非对称加密与对称加密双剑合璧，使用非对称加密算法传递用于对称加密算法的密钥，  然后使用对称加密算法进行信息传递。这样既安全又高效 |

https中公钥和私钥

https对称加密和非对称加密

对称加密：

发送方和接收方需要持有同一把密钥，发送消息和接收消息均使用该密钥。

相对于非对称加密，对称加密具有更高的加解密速度，但双方都需要事先知道密钥，密钥在传输过程中可能会被窃取，因此安全性没有非对称加密高。

非对称加密：

接收方在发送消息前需要事先生成公钥和私钥，然后将公钥发送给发送方。发送放收到公钥后，将待发送数据用公钥加密，发送给接收方。

接收到收到数据后，用私钥解密。

在这个过程中，公钥负责加密，私钥负责解密，数据在传输过程中即使被截获，攻击者由于没有私钥，因此也无法破解。

非对称加密算法的加解密速度低于对称加密算法，但是安全性更高。

几个名词要理清

RSA：非对称加密

AES：对称加密 生成一个随机字符串key 只有客户端和服务端有 他们两个通过这个key对数据加密和传输跟解密 这一个统称对称加密

CA：权威认证机构 服务器在建站的时候 去CA认证机构认证 得到对应的数字签名 相当于身份证号 客户端每次安装浏览器的时候 都会下载最新的CA列表 这个列表有对应的数字签名和服务器IP一一对应的列表 这就是为什么我们自己搭建的localhost没法发https的原因 因为没法进行CA认证

数字证书：包含了数字签名跟RSA公钥

数字签名：保证数字证书一定是服务器传给客户端的 相当于服务器的身份证ID

对称密钥： 对数据进行加密的key

非对称密钥： （k1， k2） k1加密的数据 只有k2能解开 k1位非对称公钥 k2为非对称私钥

非对称公钥：RSA公钥 k1加密的数据 只有k2能解开

非对称私钥：RSA私钥 k1加密的数据 只有k2能解开

|  |
| --- |
| <https://github.com/airuikun/Weekly-FE-Interview/issues/3>  对称加密： 发送方和接收方需要持有同一把密钥，发送消息和接收消息均使用该密钥。相对于非对称加密，对称加密具有更高的加解密速度，但双方都需要事先知道密钥，密钥在传输过程中可能会被窃取，因此安全性没有非对称加密高。  非对称加密： 接收方在发送消息前需要事先生成公钥和私钥，然后将公钥发送给发送方。发送放收到公钥后，将待发送数据用公钥加密，发送给接收方。接收到收到数据后，用私钥解密。 在这个过程中，公钥负责加密，私钥负责解密，数据在传输过程中即使被截获，攻击者由于没有私钥，因此也无法破解。 非对称加密算法的加解密速度低于对称加密算法，但是安全性更高。 |

HTTPS的加密过程

TCP/IP网络分层