# 浏览器

## 16ms渲染帧

屏幕刷新率60HZ，1秒刷新60次，则1000/60=16ms

## 重绘和重排（回流）

* 重排：是改变DOM元素的位置和大小（即改变了几何属性）时，会导致浏览器重新生成渲染树，这个过程叫重排。
* 重绘：当重新生成渲染树后，就要将渲染树每个节点绘制到屏幕，这个过程叫重绘。

不是所有的动作都会导致重排，比如改变字体颜色，只会导致重绘。

重排会导致重绘，重绘不会导致重排

重绘和重排这两个操作都是非常昂贵的，因为JavaScript引擎线程和GUI渲染线程是互斥的，他们同时只能一个在工作。

**如何减少重排和重绘？**

1. 用JavaScript修改样式时，最好不要直接写样式，而是替换class来改变样式
2. 如果要将DOM元素执行一系列操作，可以将DOM元素脱离文档流，修改完成后，再将它带回文档。

推荐使用隐藏元素（display：none）或文档碎片（DocumentFragement），都能很好的实现这个方案。

## ****防抖和节流****

## 浏览器渲染机制

**9.1 浏览器是多进程的，包含哪些进程？**

**9.2 浏览器的渲染进程是多线程的，包含哪些线程？**

**9.3 浏览器的渲染流程**

## **常见的浏览器兼容性问题**

1. **不同浏览器的默认margin和padding不一样**
2. **图片的默认间距不一样**
3. **获取视口的宽高window.innerHeight/width**
4. **css的动画、过渡、渐变、flex也有，grid**
5. **Canvas、SVG**
6. **IE9以下不能的opacity，使用filter：alpha（opacity = 50）**
7. **event.offsetX/Y**
8. **绑定事件（addEventListener）IE9才支持**

## **web安全XSS、CSRF**

xss：跨站脚本攻击，xss的重点不在于跨站点，而在于监本的执行

原理：恶意攻击者在web页面中会插入一些恶意的script代码；

当用户浏览该页面的时候，那么嵌入在web页面的script代码会执行；

以此达到恶意攻击用户的目的。

分类：

反射型XSS：在url后面的参数中加入恶意攻击代码

存储型XSS：将恶意代码存储在服务器上

防范：后端需要对提交的数据进行过滤

DOM-based型XSS：客户端的js对页面dom节点进行动态操作

SQL注入：

防范：cookie安全策略

http-only：只允许http或https请求读取cookie，js代码是无法读取cookie的；

secure-only：只允许https请求读取

host-only：只允许主机域名和domain设置完成一致的网站才能访问该cookie

防御HTML、HTML Attribute、javascript、URL、CSS编码

CSRF：跨域请求伪造； 用户登录一个正常的网站后，由于没有退出该正常网站，cookie信息还保留，然后用户去点击一个危险的网站页面，那么这个时候危险网站就可以拿到你之前登录的cookie信息，然后使用cookie信息去做其他事情。

防范：

验证HTTP Referer字段：判断来源域名是否是本地网站

加验证码

使用Token

CSRF攻击并没有违反同源策略

网页不能请求不同源服务器上的资源，也不能读取不同源网页的cookie、localStorage中的数据，但是以下资源的请求不受同源策略的限制：

* <script src=”…”></script>
* <link rel=”stylesheet” href=”…”>
* <img src=”…” />
* <video>和<audio>嵌入的多媒体资源
* <object>、<embed>和<applet>的插件
* @font-face引入的字体。一般浏览器允许跨域字体(cross-origin fonts),一些需要同源字体(same-origin fonts)
* <frame>和<iframe>载入的任何资源(可以使用X-Frame-Options阻止这种形式的跨域交互)

我们平常说的最多的跨域受限的方式是采用ajax的显示请求数据。

而CRSF可以利用上叙不受跨域资源影响的标签对目标服务器进行攻击。

## **浏览器缓存**

当浏览器访问过后的资源，会被浏览器缓存在本地，当下次再访问页面的时候，如果没有过期，就直接读取资源，加快浏览器的加载速度

http缓存机制：

1. Expires：设置最大缓存时间，当时间超过了就去服务器下载
2. http1.1 ，cache-control： max-age =time ， 当time过期后，检测etag 带上etag往服务器发请求，如果etag没变，直接告诉浏览器读本地缓存， 如果没有etag就会检测last-modified，判断 如果上一次更改的时候，距离本次访问时间比较久，说明文件没有发生变化，返回304

## 协商缓存和强制缓存

协商缓存的标识有2种：ETag/if-None-Match和Last-Modified/if-Modify-Since

浏览器第一次请求一个资源的时候，服务器会告知浏览器一个最后修改时间，当浏览器再次请求的时候，request的请求头会加上if-Modify-Since，服务器收到浏览器发送来的最后修改时间和当前自己的最后修改时间进行对比，一致的话就会返回304，否则，当前自己的最后修改时间大于浏览器携带的最后时间，将返回200，并给浏览器传递新的最后修改时间。

Etag是唯一标识，

文中"只要资源有变化，ETag会重新生成的"。也不能说作者大大说错。而是不严谨。Etag生成规则是由服务器的生成算法决定的，像nginx服务器的Etag生成规则就是由response的 Last-Modified 与 Content-Length 表示为十六进制组合而成，而last-modified又是由mtime生成。当编辑文件但是没修改内容时资源是没有变化的，但是此时mtime已经改变所以Etag也会跟着改变，但是资源文件并没有改变，Etag也会重新生成

协商缓存有两种，一种是Last-Modified，就是第一次请求资源的时候，服务器会在响应头里面设置该字段，表示该资源的最后修改时间，浏览器再第二次请求该资源的时候，会在请求头里面加上一个字段If-Modified-Since，值为第一次请求的时候服务器返回的Last-Modified的值，服务器会判断当前资源的最后修改时间和请求头里面的If-Modified-Since字段是否相同，如果相同，则告诉客户端使用缓存，否则重新下载资源。

另外一种协商缓存是使用Etag，原理与Last-Modified类似，就是第一次请求的时候，服务器会根据资源的内容或者最好修改时间生成一个标识，然后再响应头里面设置Etag返回给客户端，客户端第二次请求的时候会在请求头里面带上这个Etag，也就是再请求头里面加上If-None-Match字段，服务器接收到Etag之后会判断是否与原来第一次的标识相同，如果相同，则告诉客户端使用缓存。

强制缓存：

浏览器在第一次请求资源的时候，服务器响应头里可以设置expires字段，该字段表示该资源的缓存过期时间，第二次请求的时候，如果时间还在该缓存时间之内，则会直接使用缓存，否则重新加载资源。这个expires字段有一个缺陷，就是它必须服务器和客户端的时间严格同步才能生效，所以现在很多人不会使用该方案，另外一种方案是第一次请求资源的时候，服务器设置响应头cache-control：max-age，这样设置的意思是告诉浏览器，这个资源什么时候过期，等第二次请求资源的时候，判断是否超过过期时间，如果没有超出，直接使用缓存。

Cache-control是服务器设置的。

cache-control的其他值： public 客户端和服务端都可以缓存

private 只能客服端缓存

no-store 不使用缓存

no-cache 使用协商缓存

# **http**

## 什么是http？

http是应用层的协议；

应用层协议是客户端和服务端通信的一个约束；

http是基于TCP协议之上

## 什么是TCP ？

面向**连接**的、**可靠的**传输协议；

什么是连接？

三次握手：

TCP是传输控制层的协议；

TCP协议由操作系统实现，操作系统对上提供Socket接口；

## 什么是socket?

## https工作原理

1. **客户端发送HTTPS请求、传送加密方式列表、传递client\_random**

SSL连接总是由客户端启动，在SSL会话开始时，执行SSL握手。用户在浏览器里输入一个https网址，然后连接到server的443端口。

客户端发送以下：

* 列出客户端密支持的加密方式列表（以客户端首选项顺序排列），如SSL的版本、客户端支持的加密算法和客户端支持的数据压缩方法（hash算法）；
* 包含28字节的随机数，client\_random

1. **服务端回应算法、传送server\_random、传送SSL证书**

服务器端返回以下：

* 服务器端选出的一套加密算法和Hash算法；
* 服务器生成的随机数server\_random
* SSL数字证书（服务器使用带有SSL的X.509 V3数字证书），这个证书包含网站地址、公匙public\_key，证书的颁发机构、过期时间等。

**服务端的配置**

采用HTTPS协议的服务器必须要有一套数据证书，可以自己制作，也可以像组织申请。区别是自己颁发的证书需要客户端验证通过，才可以继续访问，而使用受信任公司申请的证书则不会弹出提示页面。这台证书其实就是一对公匙和私匙。

\*\*如果对公匙和私匙不太理解，可以想象成一把钥匙和一个锁，只是全世界只有你一个人有这把钥匙，你可以把锁给别人，别人可以用这个锁把重要的东西锁起来，然后发给你，因为只有你一个人有这把钥匙，所以只有你才能看到被这把锁锁起来的东西。

**使用非对称加密算法生成公匙和私匙。**

**非对称私密算法有：RSA、DSA/DSS**

1. **客户端TLS 解析证书并传送加密信息**

* **解析证书**

这部分工作是由客户端的TLS来完成的

首先会验证证书是否有效，这是对服务端的一种认证，比如颁发机构、过期时间等，如果发现异常，则会弹出一个警告框，提示证书存在问题。

* **根据server\_random生成随机值premaster\_secre**

如果证书没有问题，那么浏览器根据步骤3的server\_random生成一个随机值premaster\_secret（前2个字节是协议版本号，后26字节是用在对称加密密匙的随机数字）

* **根据premaster\_secre、client\_random、server\_random生成master\_secret**

master\_secret的生成需要premaster\_secret，并需要client\_random和server\_random作为种子。

master\_secret = PRF(premaster\_secret,”master secret”, client\_random + server\_random)

* **发送公匙public\_key加密的premaster\_secret给服务端**

客户端发送公匙public\_key加密的premaster\_secret。目的就是让服务器得到这个随机数，以后客户端和服务器的通信就可以通过这个随机数进行加密解密来的。

**4、服务器解密信息并传输加密后的信息**

服务器用私匙private\_key解密后，得到客户端传过来的随机数premaster\_secret，又由于服务器在步骤1中收到了client\_random，所以服务器根据相同的生成算法，在相同输入参数的情况下，得到相同他的master\_secret。然后把内容通过该值进行对称加密。

非对称加密算法：RSA、DSA/DSS

对称加密算法：AES、RC4/3DES

HASH算法：MD5、SHA1、SHA256

非对称加密算法生成公匙和私匙； 对称加密算法用于对真正传输的数据进行加密；HASH算法用于验证数据的完整性。

## 一个完整的http请求包含：

* DNS查找

1. 浏览器要将域名解析为IP地址，所以需要向本地DNS发送请求
2. 本地DNS依次向根服务器、顶级域名服务器、权限服务器发送请求，得到网站服务器的IP地址
3. 本地DNS将IP地址发回给浏览器

* TCP握手
* 浏览器发出http请求
* 服务器接收请求
* 服务器处理请求并发回响应
* 浏览器接收响应
* …….

请求到资源后开始渲染

## 浏览器的渲染过程

* 解析HTML生成DOM树
* 解析CSS生成CSSOM规则树
* 将DOM树和CSSOM规则树合并在一起生成渲染树
* 遍历渲染树开始布局，计算每个节点的位置大小信息，

生成布局，即将所有渲染树的所有节点进行平面合成

* 将渲染树每个节点绘制到屏幕

将布局绘制到屏幕上

[重排和重绘](#_重绘和回流)

抓取一个HTTP请求

Queueing：在请求队列中的时间

Stalled：从TCP链接建立完成，到真正可以传输数据之间的时间差，此时间包括代理协商时间。

Proxy negotiation：与代理服务器连接进行协商所花费的时间

DNS Lookup：执行DNS查找所花费的时间，页面上每个不同的域都需要进行DNS查找

Initial Connection/Connecting：建立连接所花费的时间，包括TCP握手/重试和协商SSl

SSL：完成SSL握手所花费的时间

Request sent：发出网络请求所花费的时间，通常为1ms的时间

Waiting（TFFB）：TFFB是发出页面请求到接收应答数据第一个字节的时间总和，它包含了DNS解析时间和获取响应消息第一个字节的时间。

Content Download：接收响应数据所花费的时间。

## 三次握手、四次挥手

**三次握手**

三次握手的本质是确认通信双方收发数据的能力。

首先，我让信使运输一份信件给对方，对方收到了，那么他就知道了我的发件能力和他的收件能力是可以的；

于是他给我回信，我若收到了，我便知我的发件能力和他的收件能力是可以的，并且他的发件能力和我的收件能力是可以的；

然而此时他还不知道他的发件能力和我的收件能力到底可不可以，于是我最后反馈一次，他若收到了，他便清楚他的发件能力和我的收件能力是可以的。

|  |  |
| --- | --- |
| 我 | 发件能力 |
| 收件能力 |
| 对方 | 发件能力 |
| 发件能力 |

**四次挥手**

## http常见状态码

200 请求成功，并返回数据

301 永久重定向

302 临时移动，可使用原有URL

304 资源未修改，可使用缓存

404 找不到资源

403 没有权限访问等

500 服务器内部错误

## 长链接Connection: keep-alive

## 断点续传、

## cache缓存策略、

## 多路复用、服务器推送

## **cookie、sessionStorage、localStorage**

cookie是在HTML4中使用的给客户端保存数据的，也可以和session配合实现跟踪浏览器用户身份；

webstorage（包括：localStorage和sessionStorage）是在HTML5提出来的，纯粹为了保存数据，不会与服务器端通信。WebStorage两个主要目标：（1）提供一种在cookie之外存储会话数据的路径。（2）提供一种存储大量可以跨会话存在的数据的机制。

相同点：

都是保存在浏览器端，且同源的。  
cookie，localStorage，sessionStorage都是在客户端保存数据的，存储数据的类型：都是字符串。

不同点：

1、生命周期：  
1).cookie如果不设置有效期，那么就是临时存储（存储在内存中），是会话级别的，会话结束后，cookie也就失效了，如果设置了有效期，那么cookie存储在硬盘里，有效期到了，就自动消失了。  
2).localStorage的生命周期是永久的，关闭页面或浏览器之后localStorage中的数据也不会消失。localStorage除非主动删除数据，否则数据永远不会消失。  
3）sessionStorage仅在当前会话下有效。sessionStorage引入了一个“浏览器窗口”的概念，sessionStorage是在同源的窗口中始终存在的数据。只要这个浏览器窗口没有关闭，即使刷新页面或者进入同源另一个页面，数据依然存在。但是sessionStorage在关闭了浏览器窗口后就会被销毁。同时独立的打开同一个窗口同一个页面，sessionStorage也是不一样的。

2、网络流量：cookie的数据每次都会发给服务器端，而localstorage和sessionStorage不会与服务器端通信，纯粹为了保存数据，所以，webstorage更加节约网络流量。

3、大小限制：cookie大小限制在4KB，非常小；localstorage和sessionStorage在5M

4、安全性：WebStorage不会随着HTTP header发送到服务器端，所以安全性相对于cookie来说比较高一些，不会担心截获。

5、使用方便性上：WebStorage提供了一些方法，数据操作比cookie方便;  
6. 数据与服务器之间的交互方式  
cookie的数据会自动的传递到服务器，服务器端也可以写cookie到客户端  
sessionStorage和localStorage不会自动把数据发给服务器，仅在本地保存。

**cookie和session的区别**  
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  
5）建议将登录信息等重要信息存放为session，其他信息如果需要保留，可以放在cookie中  
6）session保存在服务器，客户端不知道其中的信心；cookie保存在客户端，服务器能够知道其中的信息  
7）session中保存的是对象，cookie中保存的是字符串  
8）session不能区分路径，同一个用户在访问一个网站期间，所有的session在任何一个地方都可以访问到，而cookie中如果设置了路径参数，那么同一个网站中不同路径下的cookie互相是访问不到的

## DNS解析

将域名解析为IP

1，检查本机DNS缓存 可以是使用 ipconfig /displaydns 查看du

2. 检查本机 hosts文件，可以使用 CMD %windir%\System32\drivers\etc\hosts 查看

3. 向DNS查询 nslookup

当访问一个网站时系统将从DNS缓存中读取该域名所对应的IP地址，

当查找不到时就会到系统中查找hosts文件，

如果还没有那么才会向DNS服务器请求一个DNS查询，

DNS服务器将返回该域名所对应的IP，

在你的系统收到解析地址以后将使用该IP地址进行访问，

同时将解析缓存到本地的DNS缓存中。

如果DNS地址无法解析，或者是DNS缓存中的地址错误，一般才会使用ipconfig/flushdns来清除所有的DNS缓存。

ipconfig /displaydns 显示系统中已经缓存的DNS域名

ipconfig /flushdns 这是清除DNS缓存用的

## DNS预解析 dns-prefetch

DNS prefetch是一种DNS预解析技术，当你浏览网页时，浏览器会在加载网页时对网页中的域名进行解析缓存，这样在你单击当前网页中的链接时，就无需进行DNS解析，减少用户等待时间，提高用户体验。

目前每次DNS解析，通常在200ms以下。

针对DNS解析耗时问题，一些浏览器通过DNS Prefetch来提高访问的流畅性

* **实现步骤及原理**

1、<meta>信息高速浏览器，当前页面要做DNS预解析

<meta http-equiv=”x-dns-prefetch-control” content=”on” />

2、<head>使用<link>标签来强制对DNS预解析

<link rel=”dns-prefetch” href=”//static.huishenghuo.com” />

* **注意实现**

Dns-prefetch需慎用，多页面重复DNS预解析会增加重复DNS查询次数

* **浏览器对网站第一次的域名DNS解析查找流程：**

浏览器缓存 -> 系统缓存 -> 路由器缓存 -> ISP -> DNS缓存 -> 递归搜索

* **禁止隐式的DNS prefetch**

<meta http-equiv=”x-dns-prefetch-control” content=”off” />

# Html

## Contenteditable与textarea

区别：contenteditbale随着内容增加自动增高；textarea不能自动增高，需要配合js监听文本高度的变化来实现；

## line-height与height

## Node和Element的区别

Element是node的一种element\_node

Node是一个基类，DOM中的Element、Text、Comment都继承它

**parentNode 和 parentElement**

parentNode是w3c标准，parentElement只ie支持

当父节点的nodeType不是1，即不是element节点的话，他的parentElement就会是null

一般情况parentNode可以取代parentElement的所有功能

ParentElement匹配的是parent为element的情况，而parentNode匹配的是parent为node的情况。Element是包含在node里的，它的nodeType是1。

## transition

<transition  name="move" mode="out-in">

</transition>

.move-enter-active, .move-leave-active {

  transition: opacity .5s

}

.move-enter, .move-leave-active {

  opacity: 0

}

## clientHeight、offsetHeight、scrollHeight、offsetTop、scrollTop

scrollHeight：网页正文全文高

clientHeight：网页可见区域高；包括padding；不包括border、水平滚动条、margin

offsetHeight：网页可见区域，包括padding、border、水平滚动条、margin

scrollTop：网页被卷去的高

offsetTop：当前元素顶部具体最近父元素顶部的距离，和滚动没有关系

## script标签的defer和async

async：可选属性，表示应该立即下载脚本，但不应该妨碍页面中的其他操作，比如下载其他资源和等待加载其他脚本。只对外部脚本文件生效。

Defer：可选属性，表示脚本可以延迟到文档完全被解析和显示之后再执行。只对外部脚本文件有效。

# CSS

## 盒子模型

* margin 外边距
* border 边框
* padding 内边距
* content（内容区域）

width属性等于是content的宽度

总宽度 = 左右外边距 + 左右边框 + 左右内边距 + 内容宽度

总高度 = 上线外边距 + 上下边框 + 上下内边距 + 内容高度

IE5.X和6的width 属性等于 内边距+边框+content的宽度

## Flex弹性布局

布局的传统解决方案，基于盒装模型，依赖display属性 + position属性 + float属性。它对于那些特殊布局非常不方便，比如，垂直居中就不容易实现。

使用flex布局以后，子元素的float、clear和vertical-align属性将失效。

主轴：X轴

交叉轴：Y轴

**Flex容器的属性**

**1、flex-directio**n属性决定X周的方向，row | row-reverse | column | column-reverse

**2、flex-wrap** 默认情况下，项目都排在一条线上，如果一条轴线排不下，如何换行。

nowrap（默认），不换行

wrap 换行，第一行再上方

wrap-reverse 换行，第一行在下方

**3、flex-flow** 是flex-direction和flex-wrap的简写形式，默认值为row || nowrap

**4、justify-content** 定义项目再主轴上的对齐方式

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

space-around 每个项目两侧的间隔相等。

所以，项目之间的间隔比项目与边框的间隔大一倍。

**5、align-items** 定义项目在y轴上如何对齐

flex-start 起点对齐

flex-end 终点对齐

center 终点对齐

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者设置为auto，将占满整个容器的高度

**6、aligin-content** 定义了很多轴线的对齐方式。

如果项目只有一根轴线，该属性不起作用。

**Flex项目的属性**

1. **order** 定义项目的排列顺序，数值越小，排列越靠前，默认为0
2. **flex-grow** 定义项目的放大比例，默认为0，即如果存在剩余空间，也不放大

如果所有项目的flex-grow属性都为1，则它们将等分剩余空间（如果有的话）；

如果一个项目的flex-grow属性为2，其他都为1，则前者占据的剩余空间比其他项多一倍。

1. **flex-shrink** 定义项目的缩小比例，默认为1，即如果空间不足，将项目缩小
2. **flex-basis**
3. **flex**
4. **flex-self** 允许单个项目有与其他项目不一样的水平对齐方式，可以覆盖align-items属性。

默认为auto，表示继承父元素的align-items属性。

如果没有父元素，则等同于stretch

## px、em、rem，VW，VH

em相对长度单位，相对于当前对象内文字的字体尺寸，如当前对行内文字的字体尺寸未被人为设置，则相对于浏览器的默认字体尺寸。

rem是css3新增的一个相对单位（root em，根em），rem为元素设定字体大小时，依然是相对大小，但相对的只是HTML根元素。

VM：视口的宽度为100VM，相对于把视口分为100份；

VH：视口的高度为100VH，同理。

## line-height：2 、 line-height：2em、line-height：2%

浏览器有一个默认文字大小是16px，

 1、line-height:2em、line-height:200%

　　　  根据父元素的字体大小（font-size）计算行高

         子元素继承父元素计算后的行高，如果子元素的字体很大，而父元素字体小，这样计算出来的行高有可能无法容量子元素的显示，会被截掉(或是重叠)。

     2、line-height:2（推荐）

　　    根据子元素的font-size计算行高（取最大字体值\*2）

如何父元素和子元素都没有设置font-size，就按照浏览器默认的16px\*2

## 移动端适配

<https://www.runoob.com/w3cnote/moible-web-front-source.html>

meta篇

1、视图宽度

<meta name="viewport" content="width=device-width,initial-scale=1.0,minimum-scale=1.0,maximum-scale=1.0,user-scalable=no"/>

width=device-width 设置视图宽度为设备视图宽度，还可以固定宽度，例如：width=640则是640px的宽度（常见于微信）；

initial-scale=1.0 设置缩放比例为1.0

minimum-scale=1.0 和maximum-scale=1.0 最小缩放比例和最大缩放比例

user-scalable=no 禁止用户自由缩放，默认值为yes

提示：刚刚那个是带全部参数的，一般常用的，有user-scalable=no就不用使用minimum-scale=1.0和maximum-scale=1.0来强制禁止缩放了

2、自动识别格式

<meta name="format-detection" content="telephone=no"/>

Content里面的参数：telephone=no是禁止浏览器自动识别手机号码；email=no是禁止浏览器自动识别email。

Onorientationchange 手机屏幕旋转

Rezie 浏览器窗口缩放

document.documentElement 返回的是<html>……..</html>

document.documentElement.getBoundingClientRect().width （html的宽）

window.screen.width（设备的宽）

window.getComputedStyle(document.documentElement, null).getPropertyValue('font-size')

获取html标签设置的font-size

当浏览器没有最大化的时候，html的宽度小于设备的宽度

大多数浏览器的默认字号是16px，所以1rem=16px

设计稿750

Flexible将设计稿分为10份，可以理解为页面的width=10rem，即1rem=75px，所以根font-size（基准值）=75px

返利根节点data-screenwidth和font-size的设置

data-screenwidth（html的宽度）

data-screenwidth > 450时，font-size=60

data-screenwidth < 450时，font-size= data-screenwidth /750 \* 100

定时器实时监听屏幕大小，然后重新设置屏幕宽度和font-size

返利默认设置body的font-size=14px

当根html的font-size = 60px

则相当于1rem=60px

当设置某一个div的font-size=0.25rem时，font-size换算后为60\*0.25=15px

## Css样式权重

 !important > 行内 > 头部 > 引入

## line-height/font-size/height

## css伪类

input:checked 所有选中的表单元素

input:disabled 所有禁用的表单元素

input:enabled 所有启用的表单元素

input:in-range 元素指定范围内的值

input-invalid 无效的元素

p:empty 没有子元素的p元素

p:first-of-type 第一个p元素

p:last-child p元素下的最后一个子元素

## css三大预处理器（Sass、Less、stylus）

预处理器是一种用来为CSS增加一些编程特性的语言，无需考虑浏览器的兼容问题。

## BFC

## 清除浮动的方法

# <https://www.cnblogs.com/plsmile/p/11125641.html>

# JavaScript

## Js执行机制

Js是单线程语言，同一时间内只能干一件事情；

**为什么JS不能是多线程？**

是否多线程取决于这个语言的用途。

一个简单的例子，如果同一时间，一个添加DOM，一个删除了DOM，这时候语言就不知道是该添还是删了。

**同步：**在主线程上排队执行的任务，只有前一个任务执行完毕，才能执行后一个任务。

**异步：**不进入主线程，而进入“任务队列”（task queue）的任务，只有“任务队列”通知主线程，该任务才会进入主线程执行

**运行机制**

1. 所有任务都在主线程上执行，形成一个执行栈
2. 主线程之外，还存在一个“任务队列”。只要异步任务有一个运行结果，就在“任务队列”之中放置一个事件。
3. 一旦“执行栈”中的所有同步任务执行完毕，系统就会读取“任务队列”，看看里面有哪些事件。于是那些对应的异步任务结束等待状态，进入执行栈，开始执行。
4. 主线程不断重复上面三步。

## 事件循环

Promise是

## **懒加载图片的原理**

在图片没有进入可视化区域时，先不给src赋值，这样浏览器就不会发送请求了，

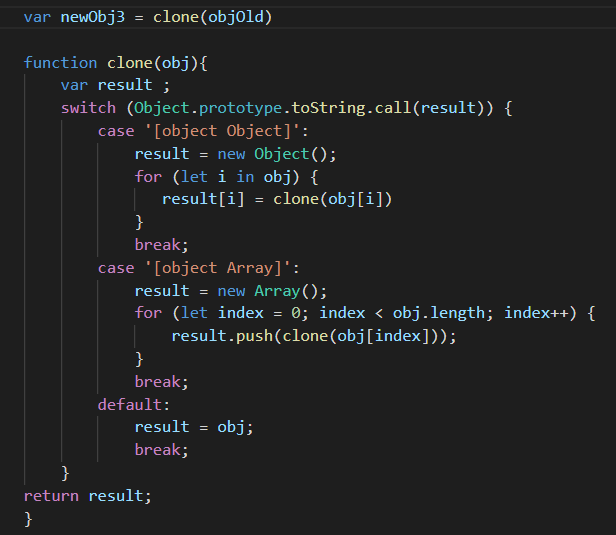
等到图片进入可视化区域再给src赋值，

图片的真实地址需要存储在data-src中，

图片没有进入可视区域，也就是说图片的offsetTop需要小于页面的可视高度，

当页面存在滚动条的时候，则需要满足img.scrollTop小于页面的可视化区域高度 + 页面滚动的高度。

## 写一个深拷贝，考虑 正则，Date这种类型的数据



## Call()、apply()、bind()

## 函数的防抖、节流

防抖和节流都是为了提升运行效率，减少函数的执行次数

防抖：把多次函数执行，合并成一次执行。给定一个间隔时间，当两次函数执行的间隔时间大于给定的间隔时间，就执行一次函数。

节流：减少函数执行的频率。规定一个单位时间，在单位时间内触发一个事件回调，触发时超过间隔则执行，否则不执行。

## 通过new操作符调用构造函数，会经历哪些阶段？

1. 创建一个新的对象
2. 链接到原型
3. 将构造函数的this指向这个新对象，执行构造函数的代码，为这个对象添加属性、方法等
4. 返回新对象

## 什么是polyfill？

Polyfill是用于实现浏览器并不支持的原生API的代码

比如说Object.assign()是很多现代浏览器都支持的原生Web API，但是古老的浏览器并不支持，那么假设有人写了一段代码来实现这个功能是这些浏览器也支持了这个功能，那么这久可以成为一个polyfill。

## 事件代理/委托

是JavaScript中常用的绑定事件的常用技巧。

即是把原来需要绑定在子元素的响应事件（click、keydown…）委托给父元素，

让父元素担当事件监听的职务

事件代理的原理是DOM元素的事件冒泡

## 事件冒泡

有两个fatherDiv和sonDiv，fatherDiv是sonDiv的父节点；

fatherDiv绑定fatherClickEvent，sonDiv绑定sonClickEvent;

当点击sonDiv后会先执行sonClickEvent再执行fatherClickEvent，

这是因为sonDiv将click事件传递给了fatherDiv；类似这种事件向上传递的现象，就叫做**事件冒泡；**

**如何取消事件冒泡？**

 function stopBubble(e){

        e = e?e:window.event

        if(e && e.stopPropagation){

            e.stopPropagation()

        }

        e.cancelBubble = true

    }

# ES6

## Proxy

## Reflect

## Babel、polyfill

Babel默认只转换新的JavaScript句法（syntax），二不转化新的API，比如Iterator、Generator、Set、Map、Proxy、Reflect、Symbol、Promise等全局对象，以及一些定义在全局对象上的方法（比如Object.assign）都不会转码。

## 实现一个promise

厄运回调金字塔、链式回调

# Nodejs

## http

http模块是node.js中非常重要的一个核心模块。

可以使用http.createServer方法创建一个http服务器；

可以使用http.reuqest方法来创建一个http客户端；

HTTP是一个用来传送数据的应用层协议，其底层使用TCP传输协议；

HTTP是互联网数据通信的基础，要相应客户端请求（如浏览器访问），首先需要部署HTTP服务器。

<https://itbilu.com/nodejs/core/N1okQ7Eh.html>

Nodejs如何创建HTTP服务器？

https://itbilu.com/nodejs/core/N1okQ7Eh.html

var http = require('http');

// 创建server对象，并添加request事件监听器

var server = http.createServer(function(req, res) {

    res.writeHeader(200,{'Content-Type':'text/plain'});

    res.end('http://erp.office.51fanli.com/');

});

server.listen(3000)

Nodejs对客户端的支持

var http = require('http');

var options = {

    host: '10.200.137.53',

    port: '8080',

    method: 'GET',

    path: '/Okr/OKpi/getProjectList?page=1&name=&type='

}

var req = http.request(options);

req.on('response', function(res){

    res.setEncoding('utf8');

    res.on('data', function(chunk){

        console.log('收到数据：%s', chunk);

    });

});

req.end();

返回结果：

# Webpack

## 四个核心概念（entry、output、loader、plugins）

入口（entry）：指示webpack应该使用哪个模块，来作为构建其内部依赖图的开始；

输出（output）：在哪里输出它所创建的bundles；

Loader：让webpack能够去处理那些非JavaScript文件；

插件（plugins）：用于执行范围更广的任务。

## css-loader和style-loader

css-loader用来解释@import和url();

style-loader用来将css-loader生成的样式表通过<style>标签，插入到页面中去。

## sass-loader、 less-loader和postcss-loader

**postcss-loader**

主要功能：

* 把css解析成JS可以操作的抽象语法树AST
* 调用插件来处理AST并得到结果

配合autoprefixer插件，打包后的css自动加上-moz、-ms、-webkit等浏览器私有前缀

## babel-**loader**

配合babel将高版本的ES6甚至ES7转化为ES5

## file-loader和url-loader

都是用来处理图片、字体图标等文件

分两种情况：

1. 当文件大小小于limit参数，url-loader将文件转化为base-64编码，用于减少http请求；
2. 当文件大小大于limit参数，调用file-loader进行处理

## vue-loader

## webpack和webpack-dev-server

webpack-dev-server的用法和webpack一样，但他会额外的启动一个express的服务器。

Webpack-dev-server打包到内存，访问速度快

## HMR（热更新）

## devtool

# Vue

## 生命周期

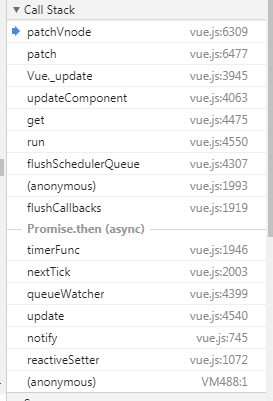
|  |  |  |
| --- | --- | --- |
| 生命周期钩子 | 组件状态 | 最佳实践 |
| BeforeCreate | 实例创建之后，this指向创建的实例，不能访问到data，computed、watch、methods上的方法和数据 | 常用于初始化非响应式变量 |
| Created | 实例创建完成，可访问data、computed、watch、methods上的方法和数据，未挂载到DOM，不能访问$el属性，$ref属性内容为空数组。 | 常用于简单的ajax请求，页面的初始化 |
| BeforeMount | 在挂载开始之前被调用，beforeMount之前，会找到对应的template，并编译成render函数 |  |
| Mounted | 实例挂载到DOM上，此时可以通过DOM API访问到DOM节点，$ref属性可以访问 | 常用于获取VNode信息和操作，ajax请求 |
| BeforeUpdate | 响应式数据更新时调用，发生在虚拟DOM打补丁之前 | 适合更新之前访问现有的DOM，比如手动移除已添加的事件监听器 |
| Updated | 虚拟DOM重新渲染和打补丁之后调用，组件DOM已经更新，可执行依赖于DOM的操作 | 避免在这个钩子函数中操作数据，可能陷入死循环 |
| BeforeDestroy | 实例销毁之前调用。这一步，实例依然可用，this仍能获取实例 | 常用于销毁定时器，解绑全局变量、销毁插件对象等操作 |
| Destroyed | 实例销毁后调用，调用后，vue实例指示的所有东西都会解绑定，所有的事件监听器会被移除，所有的子实例也会被销毁。 |  |

注意:  
created阶段的ajax请求与mounted请求的区别：前者页面视图未出现，如果请求信息过多，页面会长时间处于白屏状态。

## 虚拟dom，diff算法

virtual-dom完全使用js实现的，和宿主浏览器没有任何关系

* 1. vue的diff算法
     1. 调用栈



## 什么是单向数据流？

数据通常按照一个方向来传递，通常从父传递子组件，子组件要修改父组件数据需要通知父组件自己修改，目的是便于状态的管理和解耦。

如果不加以限制，任何子组件内部直接修改父组件数据就会影响到其他引用相同数据的组件的显示，从而造成数据混乱。

## Vue如何实现数据的双向绑定

Vue2.0使用Object.defineProperty()实现数据响应

Vue遍历data中的数据，使用Object.defineProperty()劫持getter和setter

在getter中做数据依赖收集

在setter中监听数据的变化，并通知订阅了当前数据的地方

Vue通过数据劫持（Object.defineProperty）+ 订阅发布模式

监听器Observer：用来劫持并通过Object.defineProperty监听所有属性（转变成setter和getter形式），如果属性发生变化，就通知订阅者

订阅器Dep：用来收集订阅者，对监听器Observer和订阅者Watcher进行统一管理

订阅者Watcher：监听器Observer和解释器Compile之间通信的桥梁；如果收到属性变化通知，就会执行相应的方法，从而更新视图。

解析器Compile：可以解析每个节点的相关指令，对模板数据和订阅器进行初始化。

主要做的事情是：

1. 在自身实例化时往属性订阅器（Dep）里面添加自己
2. 自身有一个update方法
3. 待属性变动dep.notice()通知时，就触发自身的update方法，并触发解析器（compile）中绑定的回调。

## 异步更新队列

当触发数据更新通知时，dep通知watcher进行数据更新，这时watcher会将自己加入到一个异步的更新队列中；

然后更新队列会将传入的更新操作进行批量处理；

这样就达到了多次更新同时完成，提高了用户体验，减少了浏览器的开销，增强了性能。

采用微任务执行刷新队列函数。

## v-model的实现原理

|  |
| --- |
| <input v-model=”searchText” >  等价于:  <input v-bind:value=”searchText” v-on:input=”searchText = $event.target.value” > |

获取到页面表单上的v-model属性，拿到属性值val，为节点绑定input事件，同时把data中属性为val的属性值赋值给当前的表单value；

当前{{}}节点的时候，就创建一个观察者（Watcher），把这个观察者添加到当前val属性的发布者上。

Vue3.0采用proxy

Object.defineProperty()的问题有：

1. 检测不到对象属性的添加和删除

Vue的set是能让vue知道你添加了属性

1. vue2.0不监控数组下标的变化，导致直接通过数组的下标给数组设置值，不能实时响应。

Js可以通过Object.defineProperty()监控数据下标的变化，但vue的设计者出于性能考虑取消了这个功能，并不是原生不支持。

3、只能劫持对象的属性，因此我们需要对每个对象的每个属性进行遍历，如果属性值也是对象，就需要深度遍历，当data中的数据比较多且层级很深的时候，会有性能问题，因为要遍历data中所有的数据并给其设置成响应式。

## v-for的key为什么不建议使用数据下标

## <https://blog.csdn.net/qq_28766729/article/details/98346565>

## computed、watch、methods

computed是计算属性，它会根据所依赖的数据动态显示计算结果，该计算结果会被缓存起来。

computed应用场景

1. 适用于一些重复使用数据或复杂及费时的运算
2. 如果我们需要的数据依赖其他的数据的话

computed的成员可以只定义一个函数作为只读属性，也可以定义为get/set变成可读写属性

watch 是对data的数据监听回调，当依赖的data的数据变化时，会执行回调。在回调中传入newVal和oldVal两个参数。

Watch应用场景：

当在data中的某个数据发生变化时，我们需要做一些操作

watch 普通监听和深度监听

computed适合一个数据被多个数据影响，watch使用于一个数据影响多个数据。

Computed实现原理：

Watchers 也就是vm.\_computedWatchers对象的引用

存放每个计算属性的观察者watcher实例

## <keep-alive></keep-alive>的作用是什么?

<keep-alive></keep-alive> 包裹动态组件时，会缓存不活动的组件实例,主要用于保留组件状态或避免重新渲染。

大白话: 比如有一个列表和一个详情，那么用户就会经常执行打开详情=>返回列表=>打开详情…这样的话列表和详情都是一个频率很高的页面，那么就可以对列表组件使用<keep-alive></keep-alive>进行缓存，  
这样用户每次返回列表的时候，都能从缓存中快速渲染，而不是重新渲染

<keep-alive>

<router-view v-if="$route.meta.alive"></router-view>

</keep-alive>

  <router-view v-if="!$route.meta.alive"></router-view>

结合vue-router使用

export default new Router({

    routes: [

        {

            path: '/banner',

            name: 'banner',

            component: Banner,

            meta:{

                alive: true,

            }

        },

]

})

**activated**::keep-alive组件激活时调用

## 父子组件各声明周期的执行顺序

## 父子组件、兄弟组件传递

父到子，在子组件中使用props声明，

子到父，在使用子组件的时候绑定自定义事件，在子组件内部拿到事件名称，再通过this.$emit(‘事件名’)触发并传递参数

其他的方式还有全局bus、provide/inject、refs/parent/children、attrs/$listeners

## vuex

vuex是vue的一个插件，用来做状态管理

其实我们可以把vuex看做是一个仓库，我们把需要共享的数据统一存放在仓库中，谁需要，直接取就可以。然后提供好一个统一修改机制（提交mutation）来修改仓库里的数据。

仓库解决了数据传递层级和同级组件数据传递问题。

State：存放状态（数据）的地方

Getters：派生状态的地方

Mutation：修改状态的地方

Action：做异步操作，并提交mutation修改状态的地方

vue v-cloak 指令

可以使用 v-cloak 指令设置样式，这些样式会在 Vue 实例编译结束时，从绑定的 HTML 元素上被移除。

当网络较慢，网页还在加载 Vue.js ，而导致 Vue 来不及渲染，这时页面就会显示出 Vue 源代码。我们可以使用 v-cloak 指令来解决这一问题。

在简单项目中，使用 v-cloak 指令是解决屏幕闪动的好方法。但在大型、工程化的项目中（webpack、vue-router）只有一个空的 div 元素，元素中的内容是通过路由挂载来实现的，这时我们就不需要用到 v-cloak 指令咯。

## **vue-rouder原理**

**13、 什么是单页应用？**

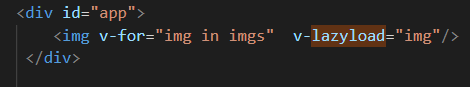
单页应用的全称是 single-page application，简称 SPA，它是一种网站应用的模型，它可以动态重写当前的页面来与用户交互，而不需要重新加载整个页面。

**工作原理：**

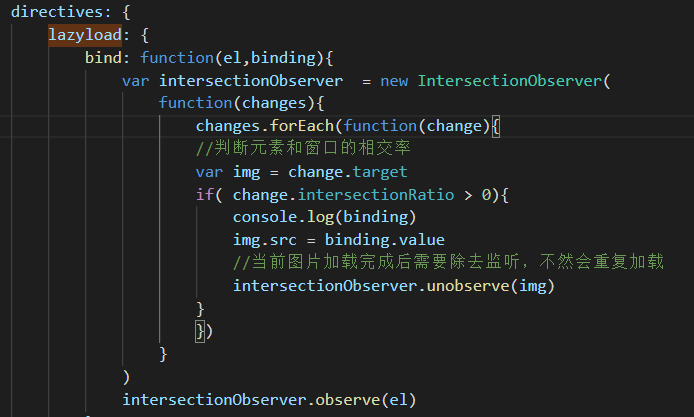
SPA 的一个重要实现就是改变路由时，页面不刷新。实现这个功能，通常有两种方式：使用 window.history 对象或 location.hash。

## ****vm.$refs****

## Vue自定义指令懒加载







## CSS scoped穿透

[https://vue-loader.vuejs.org/zh/guide/scoped-css.html#](https://vue-loader.vuejs.org/zh/guide/scoped-css.html)

**什么是scoped？**

在vue文件中的style标签上有一个特殊的属性，scoped。当一个style标签有scoped属性的时候，它的css样式只能用于当前的vue组件，可以使组件的样式不相互污染。如果一个项目的所有style标签都加上了scoped属性，相当于实现了样式的模块化。

**Scoped的实现原理**

vue中国的scoped属性的效果主要是通过postCSS实现的，

|  |
| --- |
| <style scoped>  .example {  color: red;  }  </style>  <template>  <div class="example">hi</div>  </template> |

通过postCSS转译后

|  |
| --- |
| <style>  .example[data-v-f3f3eg9] {  color: red;  }  </style>  <template>  <div class="example" data-v-f3f3eg9>hi</div>  </template> |

## nextTick

[JS的运行机制](#_Js执行机制)

实现原理：基于语言执行机制实现，直接创建一个异步任务，那么nextTick自然就达到再同步任务后执行的目的。

nextTick是vue的更新策略，也是性能优化手段，基于JS执行机制实现。

Vue中我们改变数据时不会立即触发视图，如果需要实时获取最新的DOM，这个时候可以手动调用nextTick。

$nextTick()会将我们传入的函数加入到异步更新队列中，

## Jquery Ajax/Fetch/Axios

XHR（XML HttpRequest）是最早的发送后端请求的技术，属于原始JS

1、Jquery Ajax 是对原生XHR的封装，除此之外还增添了对JSONP的支持；

2、Fetch 是一种HTTP数据请求的方式，是XML httpRequest的一种替代方案；

不是Ajax的进一步封装，而是原生JS，没有使用XML httpRequest对象。

1. Axios 是一个基于promise的HTTP库，也是对原生XML httpRequest的封装，可以在浏览器和nodejs中使用。

Ajax本身是针对MVC的编程，不符合现在前端MVVC的趋势；

很多情况下我们只需要使用ajax，但是却需要引入整个jquery，这是非常不合理的，于是便有了fetch的解决方案。

## 跨域JSONP、CORS

* **JSONP**

最大特点是简单适用，兼容性好，缺点是只支持get请求，不支持post请求。

核心思想：网页通过添加一个<script>元素，向服务器请求JSON数据，服务器收到请求后，将数据放在一个指定名称的回调函数的参数位置传回来。

1、原生实现

<script src=”http://test.com/data.php?callback=dosomething”></script>

<script type=”text/javascript”>

function dosomething(res){

console.log(res.data)

}

</script>

* jQuery ajax

$.ajax({

url:’’,

type: ‘get’,

datatype: ‘jsonp’,

jsonpCallback: ‘handleCallback’,

data: {}

})

3、vue.js

this.$http.jsonp(‘’,{

params: {},

jsonp: ‘handleCallback’

}).then((res)=>{

console.log(res)

})

* **CORS（跨域资源分享）**

1. 普通跨域请求：只需要服务器端设置Access-Control-Allow-Origin
2. 带cookie跨域请求，前后端都需要进行设置

【前端设置】根据xhr.withCredentials字段判断是否带有cookie

CORS支持所有类型的HTTP请求，是跨域HTTP请求的根本解决方案

JSONP只支持GET请求，JSONP的优势在于支持老式浏览器，以及可以向不支持CORS  
CORS的网站请求数据。

* **Nodejs中间件代理**

重定向

Proxy就是利用nodejs中间件代理做跨域的

* **Nginx代理**

# 前端性能优化

1. 减少http请求
2. 使用HTTP2
3. 使用服务器端渲染
4. 静态资源使用CDN
5. 将CSS放在文件头部，JavaScript文件放在底部
6. 使用字体图标iconfont替换图片图标
7. 善用缓存，不重复加载相同的资源
8. 压缩文件
9. 图片优化
10. 通过webpack按需加载代码，提取第三库代码，减少ES6转为ES5的冗余代码
11. 减少重绘重排
12. 使用事件委托
13. 注意程序的局部性
14. If-else对比switch
15. 查找表
16. 避免页面卡顿
17. 使用requestAnimationFrame来实现视觉变化
18. 使用Web Workers
19. 使用位操作
20. 不要覆盖原生方法
21. 降低css选择器的复杂性
22. 使用flexbox而不是最早的布局模型
23. 使用transform和opacity属性更改来实现动画
24. 合理使用规则，避免过度优化

FCP（首屏渲染时间）

RTT（HTTP的往返时间）

前后端分离导致首屏需要请求更多的内容，导致更多HTTP的往返时间，这造成了白屏。

**优化首屏渲染时间，减少白屏时间的一些方案**

* 加速或减少HTTP请求耗损

1. 使用CDN加载公用库
2. 使用强缓存和协商缓存
3. 使用域名收敛
4. 小图片使用Base64代替
5. 使用Get请求代替Post请求
6. 设置Access-Control-Max-Age减少预检请求，
7. 页面内跳转其他域名或请求使用其他域名的资源时使用浏览器prefetch预解析等

延迟加载

* 减少请求内容的体积

非重要的库、非首屏图片延迟加载、SPA的组件懒加载等；

* 浏览器渲染原理

1. 开启服务器Gzip压缩
2. JS、CSS文件压缩合并
3. 减少cookies大小
4. SSR直接输出渲染后的HTML等

* 优化用户等待体验

白屏使用加载进度条、菊花图、骨架屏代替等。

# 项目介绍

## 人事管理系统

**项目难点**

* SKU
* 骨架屏（Skeleton Screen）
* 长列表优化
* Vue大文件上传和断点续传
* 前端监控系统
* 异常问题排查

**遇到的难点、问题及解决办法**

|  |  |
| --- | --- |
| **1** | * 大文件上传，如视频200M   <1>上传文件类型的判断；不是通过后缀名判断；而是转化为二进制流，判断头信息  <2>文件切片 Blob.slice  <3>断点续传  <4>文件算hash，用hash做唯一标识  方式1：webwork 防止卡顿线程  this.worker = new Worker(‘/’)  方法二：浏览器每一帧都有空闲时间  流畅的动画或者网页60FPS，也就是1秒渲染60次，1帧大概16.6ms  卡顿造成的原因：同步任务的耗时远大于16ms  我们可以利用空闲来计算，一旦有一个优先级更高的同步任务，返回浏览器控制权，等待下一次空闲，所谓浏览器的fiber架构，就是这个理念。  Fiber：   1. 虚拟dom从树拆成链表 易于中断 2. Diff过程用空闲时间来做   Time-slice  requestIdleCallback  方法三、  布隆过滤器 牺牲一点进度 获取更高的效率  <5>时间切片  <6>控制并发数，异步任务并发 + 报错重试  <7> |
| **2** | * 动态生成页面，页面input、select太多，直接传参数（组件类型，id、方法等），通过if判断生成，无需重复编写html代码 |
| **3** | * 页面使用easyUI的输入框、下拉框、按钮太多了，导致页面渲染非常的慢。   这个页面是一个员工信息管理页面，主要包含几十列的table列表、几十个筛选项、几十个编辑项；  解决办法：  方法一：按需加载  easyui是通过class来标识的，首先将所有的对应的easyui class换成与easyui无关的class；  其次，在每次需要使用的时候，再转化为easyui对应的class，比如：点击编辑以后，再将编辑框里面的所有项的class加上easyui对应的class；  方法二：能不是使用easyui的地方就不要使用，这样会导致整体的样式不一致，但我们可以将样式修改成一样的。 |
| **4** | * 我们这个系统很多的页面涉及到上传的功能，一般有图片、pdf、excel的上传   它存在的问题是，每一处的上传都由各自的代码，很多处的代码都是复制粘贴过去的；  我需要解决的是，去除重复代码，将上传功能提取到公共文件里面，前前后后有做过大约3次大的变动；  第一次就是将以前的上传代码保留一份，放入公共模块，供使用的地方调取；  第二次改动是因为发现前面写的上传公共方法不支持一个页面有多处上传的情况  第三次改动是因为ios的钉钉不支持图片的上传  再后面是写了vue版本的公共上传组件。 |
| **5** | * 一个vue页面，它引用了三个子组件；其中两个子组件的接口需要依赖另外一个子组件接口的返回值。   B组件需要依赖A组件里面的接口返回值；  A组件创建（created）和B组件挂载（mounted）时间间隔，比A组件接口返回时间短，  所以在A组件创建阶段请求接口，B组件挂载阶段请求接口不行。  Vuex设置一个uid，默认值为0，A组件接口接口返回后给uid赋值，  B组件接口判断uid是否为0，如果是0的话等待100ms，再判断，如此一直循环，直到uid不为0，再继续请求B接口 |
| **6** | * 异常问题排查   北京分公司的一位同事访问erp.office.51fanli.com的广告排期相关的页面返回空白页； |
| **7** | * 使用jOrgChart实现组织架构树 * Dom节点生成png，支持将架构树转化为图片下载，要熟悉下原理   <https://github.com/tsayen/dom-to-image/blob/master/src/dom-to-image.js> |

## 爱予亲子阅读微信小程序

在这个项目是我自己独立开发的，我做过的事情有：

购买服务器、域名、

搭建运行环境，linux系统，使用nginx部署，配置https，

在这个项目中使用到微信接口有：

1. 小程序更新
2. 获取用户信息
3. 微信支付功能
4. 微信模板发送信息

遇到的难点、问题及解决办法

* ios8卡顿问题

scroll-view的bindscrolltolower下拉刷新确实会存在异常卡顿

解决方式：使用view再使用onReachBottom（页面上拉触底事件的处理函数）

* 登录保持

## 广告排期系统

这个系统里面涉及到了一个排期面板，这个是比较复杂的

## 同事圈

详情页返回列表页保持

keep-alive

路由的地方增加meta

实现滚动行为的代码：router/index.js

|  |
| --- |
| scrollBehavior(to, from, savePosition){  if(savePosition){  return savePosition  }else{  return {x:0,y:0}  }  } |

表情插件

1. 使用css 雪碧图（精灵图）将所有的小表情合并为一张图片，使用background-position显示指定的小表情；
2. 数据库存储小表情英文别名，取出来后，替换为dom
3. 使用div的contenteditable=”true”来实现小表情的输入

# 网上学习链接

<https://blog.csdn.net/weixin_42623421/article/details/106856141>

# 题目

# 场景

## 登录

第一步：服务器配置

在php.ini文件中配置（session.name可以自定义）

|  |
| --- |
| Session.name =PHPSESSID22  Session.auto\_start=0 |

第二步：浏览器发送第一个http请求时，服务器返回一个sessionID给浏览器

具体形式为：http请求的response中设置set-cookie

则sessionID保存在了浏览器的cookie中

|  |
| --- |
| Set-Cookie: PHPSESSID22=aqu7hrh85rhsrc4qe821behvm6; path=/ |

第三步：浏览器发送第二个http请求，http请求的request中携带cookie

|  |
| --- |
| Cookie: PHPSESSID22=aqu7hrh85rhsrc4qe821behvm6 |

第四步：服务器验证是否登录，没有登录跳转到登录页

第五步：浏览器post参数携带用户名和MD5加密的密码，服务器通过用户名、密码及其他参数生成一个key存储在服务器的session中，验证是否登录就是判断这个session是否存在

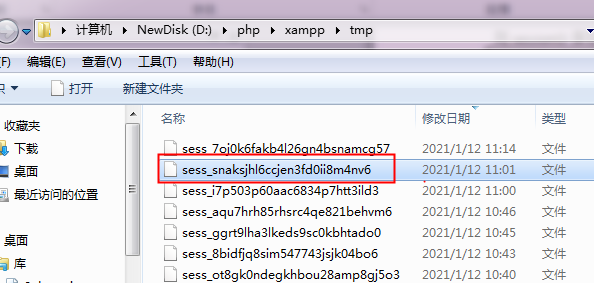
## PHP的session机制《PHPSESSID》

1. 默认机制，用磁盘文件来实现PHP会话。Php.ini配置如下

|  |
| --- |
| session.save\_handler=files  session.save\_path="D:\php\xampp\tmp" |

1. session\_start()

A、session\_start()是session机制的开始，



## JAVA的session机制 《JSESSIONID》

<https://www.cnblogs.com/barneywill/p/10116996.html>

## 单点登录（SSO）

上图是CAS官网上的标准流程，具体流程如下：

1. 用户访问app系统，app系统是需要登录的，但用户现在没有登录。
2. 跳转到CAS server，即SSO登录系统，**以后图中的CAS Server我们统一叫做SSO系统。** SSO系统也没有登录，弹出用户登录页。
3. 用户填写用户名、密码，SSO系统进行认证后，将登录状态写入SSO的session，浏览器（Browser）中写入SSO域下的Cookie。
4. SSO系统登录完成后会生成一个ST（Service Ticket），然后跳转到app系统，同时将ST作为参数传递给app系统。
5. app系统拿到ST后，从后台向SSO发送请求，验证ST是否有效。
6. 验证通过后，app系统将登录状态写入session并设置app域下的Cookie。

至此，跨域单点登录就完成了。以后我们再访问app系统时，app就是登录的。接下来，我们再看看访问app2系统时的流程。

1. 用户访问app2系统，app2系统没有登录，跳转到SSO。
2. 由于SSO已经登录了，不需要重新登录认证。
3. SSO生成ST，浏览器跳转到app2系统，并将ST作为参数传递给app2。
4. app2拿到ST，后台访问SSO，验证ST是否有效。
5. 验证成功后，app2将登录状态写入session，并在app2域下写入Cookie。

这样，app2系统不需要走登录流程，就已经是登录了。SSO，app和app2在不同的域，它们之间的session不共享也是没问题的。

**有的同学问我，SSO系统登录后，跳回原业务系统时，带了个参数ST，业务系统还要拿ST再次访问SSO进行验证，觉得这个步骤有点多余。他想SSO登录认证通过后，通过回调地址将用户信息返回给原业务系统，原业务系统直接设置登录状态，这样流程简单，也完成了登录，不是很好吗？**

**其实这样问题时很严重的，如果我在SSO没有登录，而是直接在浏览器中敲入回调的地址，并带上伪造的用户信息，是不是业务系统也认为登录了呢？这是很可怕的。**

# 总结

单点登录（SSO）的所有流程都介绍完了，原理大家都清楚了。总结一下单点登录要做的事情：

* **单点登录（SSO系统）是保障各业务系统的用户资源的安全 。**
* **各个业务系统获得的信息是，这个用户能不能访问我的资源。**
* **单点登录，资源都在各个业务系统这边，不在SSO那一方。 用户在给SSO服务器提供了用户名密码后，作为业务系统并不知道这件事。 SSO随便给业务系统一个ST，那么业务系统是不能确定这个ST是用户伪造的，还是真的有效，所以要拿着这个ST去SSO服务器再问一下，这个用户给我的ST是否有效，是有效的我才能让这个用户访问。**

智力题：

25匹马，5个人一组跑，跑几次可以找到前三名

7 前5次