**有关于浏览器缓存：**

强缓存：规定时间内不询问服务器强制使用浏览器缓存。使用响应头的expires（绝对时间：客户端服务器时间不同步会出现用不了缓存现象）字段和cache-control（相对时间：能在有效时间里使用强缓存）字段

协商缓存：if-Modified-Since字段询问服务器是否使用缓存，服务器根据传来的if-Modified-Since时间来判断是否需要使用缓存。，返回304告诉浏览器从缓存中加载资源；

常见的前端缓存：cookie。session。localstorage和sessionstorage

**CDN--解析域名获得相应的CNAME域名-对CNAME域名解析获得缓存服务器的IP地址。--向IP发送请求**

**跨域：jsonp cors机制**

1.cors(cross-origin resource sharing跨域资源共享)：使用自定义的头部让浏览器和服务器进行沟通从而决定请求或响应应该成功还是失败。（Access-Control-Allow-Origin）

缺点：Firefox Chrome 安卓平台的webkit都通过XHR对象实现了对CORS的原生支持，要请求位于另一个域的资源，只需要使用标准的XHR对象，并在open方法传入绝对URL即可。缺点：不能发送和接收cookie，不能使用setRequestHeader()设置自定义头 调用getAllResponseHeaders()总返回空字符串。而CORS又通过preflight requests的机制支持开发者使用自定义头，get/post之外的方法和不同类型的主题内容。而设置withCredentials:true可以允许跨域请求带凭据（cookie http认证等）

2.jsonp:json with padding:前端< script>标签发送跨域请求同时约定好回调函数，后端返回一个字符串模拟js调函数完成跨域。支持浏览器与服务器间的双向通信，简单易用。但是只能处理get请求；存在安全隐患：jsonp是从其他域中加载代码执行，如果其他域不安全，可能会在响应中夹带恶意代码。

3.图像ping:一个网页可以从任何网页加载图像，不存在跨域问题。特点：简单，单向跨域通信。缺点：只能发送get请求，无法访问服务器响应文本。常用在跟踪用户点击页面或动态广告曝光次数。

**window.name传值：打开的浏览器内不同页面间加载后始终存在的一个window对象的属性**

**React是什么？**

react是一个用于视图的js框架；是Facebook打造的，致力于更快的dom渲染和更易于维护的dom层，用于视图的用户交互；很多人认为react是MVC的View

**React Element**

virtual DOM和React Element分别对应是浏览器DOM和HTML元素；virtualDOM不是像HTML一样放在浏览器中而是在内存中的；DOM是由一个个的HTML element组成的，而virtual DOM是由一个个的React element组成的。React element多了之后层层嵌套就形成了DOM树，即virtual DOM。而ReactDOM.render将虚拟DOM转化成真是的DOM。

**对vue生命周期的理解？**

组件创建前/后：

beforeCreate/created

beforeCreate阶段vue实例的挂载元素$el（页面已经存在的一个dom元素）和数据对象data都为undefined，未初始化；

**created阶段**vue实例的data对象有了即完成数据观测，但是$el还不可用；

组件渲染前后 beforeMount/mounted 。

beforeMount阶段，vue的$el和data都已经初始化。vue实例已经完成编译模板把data中的数据和模板生成HTML，但是HTML还没渲染到页面上；

**mounted阶段**vue实例挂载完成，html成功渲染。此过程可以进行ajax交互

数据更新前后 beforeUpdate/updated

销毁前后**beforeDestory**/destoryed 执行destory方法后对data的改变不会再触发周期函数，说明vue实例已经解除了事件监听以及和dom的绑定，但是dom结构依然存在。

beforeDestory:主要用于解绑一些使用addEventListener监听的事件等。

1.什么是vue生命周期？  
答： Vue 实例从创建到销毁的过程，就是生命周期。从开始创建、初始化数据、编译模板、挂载Dom→渲染、更新→渲染、销毁等一系列过程，称之为 Vue 的生命周期。

2.vue生命周期的作用是什么？  
答：它的生命周期中有多个事件钩子，让我们在控制整个Vue实例的过程时更容易形成好的逻辑。

3.vue生命周期总共有几个阶段？  
答：它可以总共分为8个阶段：创建前/后, 载入前/后,更新前/后,销毁前/销毁后。

4.第一次页面加载会触发哪几个钩子？  
答：会触发 下面这几个beforeCreate, created, beforeMount, mounted 。

5.DOM 渲染在 哪个周期中就已经完成？  
答：DOM 渲染在 mounted 中就已经完成了。

补充：

**css3的animation**

1.关键帧：@keyframes:定义动画名以及关键帧（动画在不同阶段的状态）

2.animation:名+时长+何种函数方式播放动画(linear全程一个速度，ease默认，ease-in低速开始...)+播放次数（n,infinite）

**canvas与SVG区别：**

canvas和SVG都允许在浏览器创建图形；

|  |  |
| --- | --- |
| SVG：使用xml描述2D图形的语言  基于xml，则svg DOM中的每个元素都是可用的。可以为某个元素添加js事件处理器  每个被绘制的图形都被视作一个对象，如果svg对象的属性发生变化，浏览器自动重现图形 | canvas通过js绘制2D图形  逐像素渲染  上屏像素化，如果位置发生变化，则需要擦掉重画 |
| * 不依赖分辨率 * 支持事件处理器 * 复杂度高会减慢渲染速度（然和过度使用dom的应用都不快） * 适合带有大型渲染区域的应用程序 * 不适合游戏应用 | * 依赖分辨率 * 不支持事件处理器 * 弱的文本渲染能力 * 能够以png jpg格式保存结果图像 * 适合图像密集型游戏，许多图像会被频繁重绘 |

# 浏览器reflow和repaint

|  |  |
| --- | --- |
| reflow | repaint |
| render Tree的元素因为改变了自身的宽高，布局，显示或隐藏或者元素内部文字结构变化导致需要重新构建页面 | 只是改变了元素的外观风格时如background-color |
| 回流必定触发重绘，重绘不一定触发回流 | |
| 盒子模型相关属性触发重布局 width height padding margin display  定位属性及浮动触发重布局 top left right bottom float clear  改变节点内部文字结构触发重布局 text-align font-size font-weight vertical-align | boder-radius  box-shadow  color  visibility |

注意：渲染树不包含隐藏的节点如display:none的节点，因为其不用于呈现；visibility:hidden隐藏的元素还是包含在渲染树中，因为其会影响布局，会占有空间。

避免回流和重绘：

1.不要使用js对dom设置多条样式，用className代替el.addClass('changeStyle'); .changeStyle{width: 200px;height: 200px;}

2.如果确实需要使用js对dom设置多条样式，可以先将dom display：none，修改--再display:block

3.cssText合并所有修改的样式一次性写入el.style.cssText += "; left: " + left + "px; top: " + top + "px;";

4当前文档之外创建一个子树，修改后再插回去：

var fragment = document.createDocumentFragment();  
var list = document.getElementById("list");

list.appendChild(fragment);

5.transform代替top left margin-left等位移属性

原因：CSS的最终表现分为以下四步：Recalculate Style -> Layout -> Paint Setup and Paint -> Composite Layers（组合层）

 由于transform是位于Composite Layers层，而width、left、margin等则是位于Layout（排布）层，在Layout层发生的改变必定导致Paint Setup and Paint -> Composite Layers，所以相对而言使用transform实现的动画效果肯定比使用改变位置(margin-left等)这些更加流畅。

6. 不要在循环内获取dom 的样式例如：offsetWidth, offsetHeight, clientWidth, clientHeight... 这些浏览器不会马上操作而是缓存在一个栈中，栈满了会一次性触发所有样式的更改且刷新这个栈，如果多次获取就会让browser不断刷新这个栈导致回流增多

|  |  |  |
| --- | --- | --- |
| clientWidth | offsetWidth | scrollWidth |
| width+padding | width+padding+border+（滚动条） | 获取指定标签内容层的真实宽度（可视区域宽度+被隐藏区域宽度） |

有关opacity待定；

**http消息报头:普通报头实体报头请求报头响应报头**

cache-control/accept/host/authorizaiton/content-type/content-encoding/expires/user-agent/server/location/www-authenticate

**js观察者模式**

观察者模式又叫发布订阅模式，（publish/subscribe），它让多个观察者对象同时监听某一个主题对象，这个主题对象状态发生变化时就会通知所有的观察者对象，使他们自动更新自己。适用于：当一个对象改变需要同时改变其他对象，并且他不知道具体多少对象需要改变时

使用观察者模式的好处：

1. 支持简单的广播通信，自动通知所有已经订阅过的对象。
2. 页面载入后目标对象很容易与观察者存在一种动态关联，增加了灵活性。
3. 目标对象与观察者之间的抽象耦合关系能够单独扩展以及重用。

**http与https：**

http缺点：

1.通信使用不加密明文，内容可能被窃听

2.无法验证通信方身份，可能遭遇伪装

3.无法证明接收报文的完整性，可能已经遭遇篡改。

针对1：

通信的加密，http没有加密机制但可以通过和SSL（secure socket layer）或TLS(transport layer security)的组合使用加密http通信内容；与SSL组合使用的HTTP就叫HTTPS（http secure）超文本传输安全协议。（SSL建立安全通信线路后就可以在这条线路上进行http通信）

针对2：

SSL不仅提供加密处理还使用了第三方颁发的证书用于确定通信方身份

针对3：

SSL提供认证和加密处理以及摘要功能保证完整性。

https常用场景：登录页面；购物车清算。

|  |  |
| --- | --- |
| http | https |
| 直接和TCP通信 | http-ssl-tcp |
|  | 通信慢，消耗CPU和内存资源导致处理速度变慢（必须进行加密处理，服务端和客户端都需要进行加密解密的运算处理。） |
|  | 需要购买证书，小网站并不必要。 |

将网站从http--https

申请CA证书--配置服务器配置文件，开放443端口--项目里引用的js和css相对路径从原先的http转换为https--(配置CDN)

**有关node**

node的全局对象：global（node所处的全局环境，类似于浏览器的window对象），process（node所处的当前进程，允许开发者与该进程互动）console（node内置的console模块，提供命令行环境的标准输入标准输出）。

全局函数：定时器函数： setTimeout clearTimeout setInterval clearInterval

require:加载模块；

Buffer()：操作二进制数据

全局变量:\_\_filename 当前运行脚本文件名 \_\_dirname所在的目录

**node特点：**高并发，所有IO请求都是异步的。不擅长CPU密集型的操作。

**node如何实现高并发？**

传统的网络服务技术是每新增一个请求就生成一个新线程，这样多个线程会占用系统内存，而node是单线程，使用非阻塞的异步IO调用，所有连接都由该线程处理，在libuv的加分下，可以将数万个连接挂到该线程的事件循环中

**promise的特点：**

1、对象的状态不受外界影响。

Promise对象有三种状态：pending（进行中）、fulfilled（已成功）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fulfilled和从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，这时就称为 resolved（已定型）。

**问题一：axios、fetch与ajax有什么区别？**

主要区别是 axios、fetch请求后都支持Promise对象API，ajax只能用回调函数

Axios本质上也是对原生XHR的封装，只不过它是Promise的实现版本，符合最新的ES规范，它有以下几条特性：

* 从 node.js 创建 http 请求
* 支持 Promise API
* 客户端支持防止CSRF
* 提供了一些并发请求的接口（重要，方便了很多的操作）

**有关vue-cli下的跨域问题：**

通过node.js代理服务器来实现跨域：在vue-cli项目中的config文件夹下的index.js文件里配置dev对象的proxyTable对象

**axios特点**

1、Axios 是一个基于 promise 的 HTTP 库，支持promise所有的API  
2、它可以拦截请求和响应  
3、它可以转换请求数据和响应数据，并对响应回来的内容自动转换成 JSON类型的数据  
4、安全性更高，客户端支持防御 XSRF

# vue 的spa

**单页Web应用（single page web application，SPA）：**

就是只有一张Web页面的应用。单页应用程序 (SPA) 是加载单个HTML 页面并在用户与应用程序交互时动态更新该页面的Web应用程序。浏览器一开始会加载必需的HTML、CSS和JavaScript，所有的操作都在这张页面上完成，都由JavaScript来控制。因此，对单页应用来说模块化的开发和设计显得相当重要。

**单页Web应用的优点：**

1、提供了更加吸引人的用户体验：具有桌面应用的即时性、网站的可移植性和可访问性。

2、单页应用的内容的改变不需要重新加载整个页面，web应用更具响应性和更令人着迷。

3、单页应用没有页面之间的切换，就不会出现“白屏现象”,也不会出现假死并有“闪烁”现象

4、单页应用相对服务器压力小，服务器只用出数据就可以，不用管展示逻辑和页面合成，吞吐能力会提高几倍。

5、良好的前后端分离。后端不再负责模板渲染、输出页面工作，后端API通用化，即同一套后端程序代码，不用修改就可以用于Web界面、手机、平板等多种客户端。

**单页Web应用的缺点：**

1、首次加载耗时比较多。

2、SEO问题，不利于百度，360等搜索引擎收录。

3、容易造成Css命名冲突。

4、前进、后退、地址栏、书签等，都需要程序进行管理，页面的复杂度很高，需要一定的技能水平和开发成本高。

**为什么要按需加载？**

在于一次性的加载所有功能对应的代码，但其实用户每一阶段只可能使用其中一部分功能。

**如何按需加载**

* 把整个网站划分成一个个小功能，再按照每个功能的相关程度把它们分成几类。
* 把每一类合并为一个 Chunk，按需加载对应的 Chunk。
* 对于用户首次打开你的网站时需要看到的画面所对应的功能，不要对它们做按需加载，而是放到执行入口所在的 Chunk 中，以降低用户能感知的网页加载时间。
* 对于个别依赖大量代码的功能点，例如依赖 Chart.js 去画图表、依赖 flv.js 去播放视频的功能点，可再对其进行按需加载。

import 返回一个 Promise，当文件加载成功时可以在 Promise 的  then  方法中获取到  show.js  导出的内容

 /\* webpackChunkName: "show" \*/  的含义是为动态生成的 Chunk 赋予一个名称，以方便我们追踪和调试代码。 如果不指定动态生成的 Chunk 的名称，默认名称将会是  [id].js。 /\* webpackChunkName: "show" \*/  是在 Webpack3 中引入的新特性，在 Webpack3 之前是无法为动态生成的 Chunk 赋予名称的。

分离css:引入-实例化，同时在下面的plugins数组里添加-rules里面配置把刚刚处理css的用plugin.extract()方法包起来-

**展示型组件和容器型组件**

展示组件

- 只关心它们的样子。

- 可能同时包含子级容器组件和展示组件，一般含DOM标签和自定的样式。

- 通常用this.props.children来包含其他组件

- 不依赖app其它组件，比如flux的actions和stores

- 不会定义数据如何读取，如何改变

- 只通过this.props接受数据和回调函数

- 很少有自己的状态变量，即使有，也是UI的状态变量，比如toggleMenuOpen,InputFocus

- 一般是函数级组件，除非它们需要状态，lifecycle hooks，优化处理。

容器组件

- 只关心它们的运作方式。

- 可能同时包含子级容器组件和展示组件，但大都不含DOM标签，而含他们自己所用的wrapping div，从不用自己的样式。

- 为展示组件或其他组件提供数据和方法。

- 调用Flux的actions，并且将其作为展示组件的回调函数。

- 维持许多状态变量，通常充当一个数据源。

- 通常由高阶组件生成，比如Redux里的connect()，Relay里的createContainer()，Flux Utils里的Container.create()，而非手工写出（译者：可能在meteor中数据是例外吧）

- 例子有UserPage, FollowersSidebar, StoryContainer, FollowedUserList。