一、HTML篇

1、HTML5 新增那些标签?

2、常见的块元素、行内元素以及行内块元素，三者有何不同？

3、HTML、XML、XHTML它们之间有什么区别？

4、DOCTYPE(⽂档类型) 的作⽤

5、localStorage、sessionStorage、cookie 三者明细区别？

6、网站TDK三大标签以及SEO优化

7、img标签的title属性与alt属性的区别是什么？

8、src 和 href 的区别？

9、title 与 h1 的区别、b 与 strong 的区别、i 与 em 的区别？

10、iframe的优点和缺点 ？

二、CSS篇

1、说一下 link 与 @import 的区别和用法？

2、rgba和opacity的透明效果有什么不同？

3、display:none与visibility:hidden的区别？

4、定位布局 position中的relative、absolute、fixed、sticky它们之间的区别？

5、如何用CSS3画一条0.5px的直线？

6、如何用CSS3画一个三角形？

7、CSS3盒子模型：标准盒模型、怪异盒模型

8、浮动（float)以及清除浮动的方法

9、Flex布局

10、CSS3中“transform”属性~平面转换

11、CSS3中 “子绝父相” 定位布局

12、盒子居中的几种方法：“子绝父相”、“Flex布局”、“transform”

13、CSS3中有哪些新特性？

14、CSS3选择器及其优先级

15、CSS3中 “transition” 过渡属性

16、结构伪类选择器&伪元素选择器

17、display的block、inline和inline-block的区别？

18、定位堆叠顺序z-index

三、HTML&&CSS混合篇

1、Localstorage、sessionStorage、cookie 的区别

2、如何实现双飞翼（圣杯）布局？

3、伪元素和伪类的区别和作用？

4、img 的 alt 与 title 的异同，还有实现图片懒加载的原理？

5、BFC 是什么？

四、JavaScript篇

1、JS基础类型和复杂类型

2、箭头函数与普通函数的区别？

3、JS中null和undefined的判断方法和区别？

4、原型链

5、v-show 与 v-if 的区别？

6、keep-alive 的作用是什么?

7、闭包的理解?

8、JS垃圾回收机制

9、nextTick的实现？

10、混入mixin的原理？

11、列举和数组操作相关的方法

12、typeof和instanceof的区别是什么？

13、JS中 “==“和“===“的区别详解

14、如何用原生 JS给一个按钮绑定两个 onclick 事件？

15、var、let和const的区别？

16、讲解js的call、apply和bind区别？

17、谈谈你对webpack的理解？

18、 const定义的对象属性是否可以改变？

19、栈溢出及解决方法？

20、JS如何实现多线程？

21、浅拷贝和深拷贝区别概念常见情况？

22、事件循环，Promise和async/await的详解

23、JS中数组常用方法详解

五、Vue2篇

​1、什么是前端构建工具?比如（Vue2的webpack，Vue3的Vite）

​2、Vue 组件之间的通信方式

​3、Vuex的理解及使用场景

4、vue 的生命周期 八个阶段

5、简述Vue每个周期具体适合哪些场景？

6、简述MVVM 和MVC的原理以及区别？

7、vue常见指令

8、vue中的data为什么是一个函数？起到什么作用？

9、vue中ref的作用？

10、vue中hash和history的区别 ？

六 、Vue3篇

1、Vue2.0和Vue3.0的区别？

2、Vue3带来了什么改变？

3、生命周期（vue2和vue3的生命周期对比）有哪些？

4、Vue3.0中的响应式原理是什么？vue2的响应式原理是什么？

5、vue3的常用 Composition API有哪些？

6、Vue3中的ref函数

七、React篇

1、讲讲什么是 JSX ？

2、React 的生命周期方法有哪些？

3、React的Hooks详解

4、React和Vue.js的相似性和差异性是什么？

5、React的功能是什么？

6、React的一些主要优点是？

7、React的局限性是什么？

8、如何理解React State不可变性的原则

9、React的特点和关键版本区别？

八、uniapp篇

1、uniapp优缺点

2、uniapp目录结构

3、跨端适配—条件编译

4、uniapp 常用的指令语句

5、uniapp 应用的生命周期、页面的生命周期、组件的生命周期

九、Webpack篇

1、Webpack是什么？

2、Webpack的打包过程/打包原理/构建流程？

3、Webpack中loader的作用/ loader是什么？

4、常见的loader有哪些？

5、Plugin有什么作用？Plugin是什么？

6、常见的Plugin有哪些？

7、Webpack中Loader和Plugin的区别

8、如何利用webpack来优化前端性能？

9、Webpack如何配置压缩代码？压缩了什么？

10、如何提高webpack的构建速度？

十、Git/SVN篇

1、Git是什么？

2、简述Git中如何将一个新增文件添加到本地仓库？

3、Git常用命令

4、Git和SVN有什么区别?

十一、B站开源视频大汇总，前端从入门到精通

1、web 入门（HTML5+CSS3）

2、js（JavaScript+jQuery+Ajax）

3、Vue 开发（Vue2+Vue3+实战项目）

4、React &&微信小程序开发&&uniapp

5、华为鸿蒙进阶

6、ssr（Nuxt+Next.js）

7、工程化（webpack+vite）

8、可视化（echarts ）

9、源码（虚拟DOM和diff算法）

10、移动App跨端开发（Uni-App）

11、桌面应用开发（Electron）

12、网络知识（网络概述+结构算法）

13、浏览器底层原理

一、HTML篇

1、HTML5 新增那些标签?

（1）布局标签（语义化标签）

<header> : 头部标签

<nav> : 导航标签

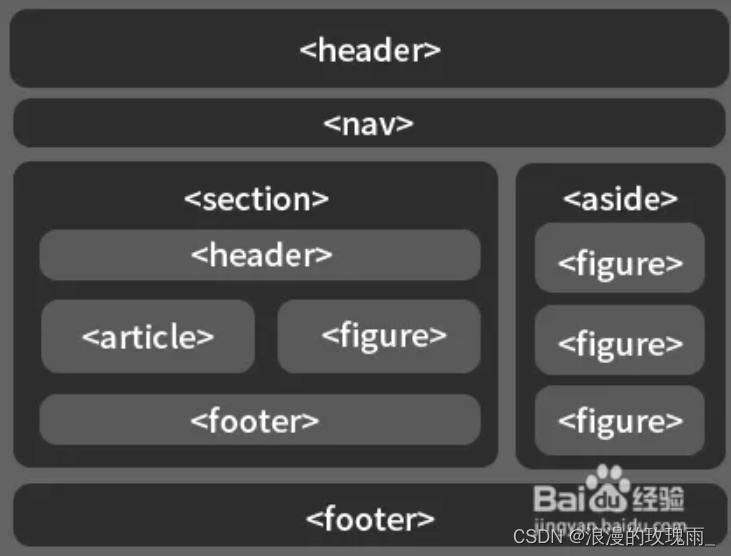
<article> : 内容标签

<section> : 定义文档某个区域

<aside> : 侧边栏标签

<footer> : 底部标签

图形结合记忆法：

（2）视频标签(video)



（3）audio(音频)



（4）新的 API

音视频：audio 和 video 元素

绘图图形：canvas 元素

本地存储：localStorage，sessionStorage

多线程操作：Web Worker （Web Worker 是HTML5 新特性，允许我们在 js 主线程之外开辟新线程，并将一段 js 脚本运行其中，它赋予了开发者利用 js 操作多线程的能力 ）

更多HTML5新属性=>W3Cschool官网：开始学习HTML5\_w3cschool

语义化的优点如下：

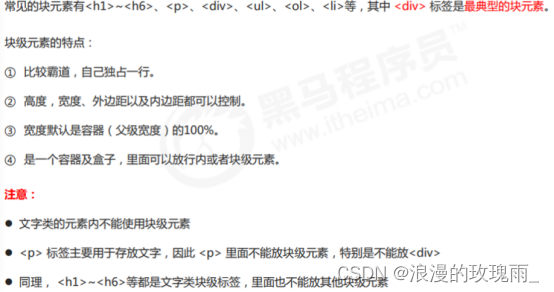
在没有CSS样式情况下也能够让页面呈现出清晰的结构

有利于SEO和搜索引擎建立良好的沟通，有助于爬虫抓取更多的有效信息，爬虫是依赖于标签来确定上下文和各个关键字的权重

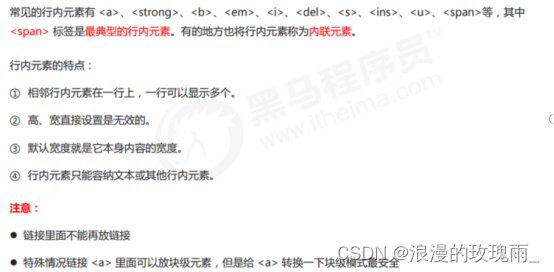
方便团队开发和维护，语义化更具可读性，遵循W3C标准的团队都遵循这个标准，可以减少差异化

2、常见的块元素、行内元素以及行内块元素，三者有何不同？

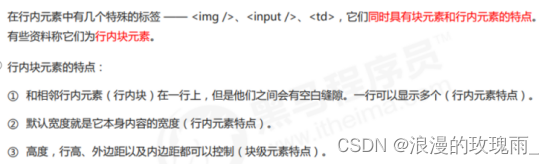
（1）块元素



（2）行内元素：a，span，em(i)，strong(b)，u



（3）行内块元素



3、HTML、XML、XHTML它们之间有什么区别？



4、DOCTYPE(⽂档类型) 的作⽤

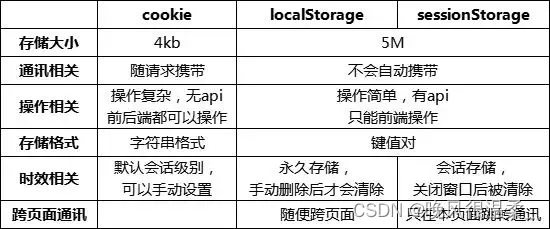
Doctype是HTML5的文档声明，通过它可以告诉浏览器，使用哪一个HTML版本标准解析文档。在浏览器发展的过程中，HTML出现过很多版本，不同的版本之间格式书写上略有差异。如果没有事先告诉浏览器，那么浏览器就不知道文档解析标准是什么？此时，大部分浏览器将开启最大兼容模式来解析网页，我们一般称为怪异模式，这不仅会降低解析效率，而且会在解析过程中产生一些难以预知的bug，所以文档声明是必须的。

浏览器渲染页面的两种模式（可通过document.compatMode获取）：

CSS1Compat：标准模式（Strick mode），默认模式，浏览器使用W3C的标准解析渲染页面。在标准模式中，浏览器以其支持的最高标准呈现页面。

BackCompat：怪异模式(混杂模式)(Quick mode)，浏览器使用自己的怪异模式解析渲染页面。在怪异模式中，页面以一种比较宽松的向后兼容的方式显示。

5、localStorage、sessionStorage、cookie 三者明细区别？



cookie是网站为了标示用户身份而储存在用户本地终端（Client Side）上的数据（通常经过加密）。

cookie数据始终在同源的http请求中携带（即使不需要），记会在浏览器和服务器间来回传递。

sessionStorage和localStorage不会自动把数据发给服务器，仅在本地保存。

存储大小

cookie数据大小不能超过4k。

sessionStorage和localStorage 虽然也有存储大小的限制，但比cookie大得多，可以达到5M或更大。

有期时间

localStorage 存储持久数据，浏览器关闭后数据不丢失除非主动删除数据；

sessionStorage 数据在当前浏览器窗口关闭后自动删除。

cookie 设置的cookie过期时间之前一直有效，即使窗口或浏览器关闭

6、网站TDK三大标签以及SEO优化



TDK是网站标题（title）、描述（description）、关键词（keywords）这三者的英文首字母缩写，主要用来对当前网页进行总结和概况。

在SEO界，标题、描述、关键词通常也被称为三大标签。



7、img标签的title属性与alt属性的区别是什么？

alt是给搜索引擎识别，在图像无法显示时的替代文本

title属性是关于元素的注释信息，主要是给用户解读

在定义 img 对象时，将 alt 和 title 属性写全，可以保证在各种浏览器中都能正常使用

注意：（IE已淘汰，仅做了解即可）

当鼠标放到文字或是图片上时有 title 文字显示。

（因为IE不标准）在 IE 浏览器中 alt 起到了 title 的作用，变成文字提示。

8、src 和 href 的区别？

src和href都是HTML中特定元素的属性，都可以用来引入外部的资源。两者区别如下：

src：全称source，它通常用于img、video、audio、script元素，通过src指向请求外部资源的来源地址，指向的内容会嵌入到文档中当前标签所在位置，在请求src资源时，它会将资源下载并应用到文档内，比如说：js脚本、img图片、frame等元素。当浏览器解析到该元素时，会暂停其它资源下载，直到将该资源加载、编译、执行完毕。这也是为什么将js脚本放在底部而不是头部的原因。

href：全称hyper reference，意味着超链接，指向网络资源，当浏览器识别到它指向的⽂件时，就会并⾏下载资源，不会停⽌对当前⽂档的处理，通常用于a、link元素。

9、title 与 h1 的区别、b 与 strong 的区别、i 与 em 的区别？

title与h1的区别

从网站角度看，title 更重于网站信息。title可以直接告诉搜索引擎和用户这个网站是关于什么主题和内容的。

从文章角度看，h1则是用于概括文章主题。

一个网站可以有多个title,最好一个单页用一个title，以便突出网站页面主体信息，从seo看，title权重比h1高，适用性比h1广。

标记了h1的文字页面给予的权重会比页面内其他权重高很多。一个好的网站是h1和title并存，既突出h1文章主题，又突出网站主题和关键字。达到双重优化网站的效果。

b与strong 的区别

b 是只是对文本的简单加粗， strong 是一个语义化标签，对相关文本具有强调作用

b 标签只是侧重于字体加粗， strong标签加强字体的语气都是通过粗体来实现的，相比之下，搜索引擎更喜欢侧重于strong标签

strong标签更注重于内容上的应用，在html中，对关键词的标明，然而还有一些网站上，也有使用strong标签登对小标题进行强调，但是在页面中，如果出现过多的strong标签，可能会对排名不利。

i 与 em 的区别

i(italic)是实体标签，用来使字符倾斜，em(emphasis)是逻辑标签，作用是强调文本内容

i标签只是斜体的样式，没有实际含义，常用来表达无强调或着重意味的斜体，比如生物学名、术语、外来语；

em表示标签内字符重要，用以强调，其默认格式是斜体，但是可以通过CSS添加样式。

建议：为了符合CSS3的规范，i 标签应尽量少用而应改用 em

10、iframe的优点和缺点 ？

iframe的缺点：

页面样式调试麻烦，出现多个滚动条；

浏览器的后退按钮失效；

过多会增加服务器的HTTP请求；

小型的移动设备无法完全显示框架；

产生多个页面，不易管理；

不容易打印；

代码复杂，无法被一些搜索引擎解读。

搜索引擎的检索程序无法解读这种页面，不利于SEO;

iframe的优点：

iframe能够原封不动的把嵌入的网页展现出来。

如果有多个网页引用iframe，那么你只需要修改iframe的内容，就可以实现调用的每一个页面内容的更改，方便快捷。

网页如果为了统一风格，头部和版本都是一样的，就可以写成一个页面，用iframe来嵌套，可以增加代码的可重用。

如果遇到加载缓慢的第三方内容如图标和广告，这些问题可以由iframe来解决。

重载页面时不需要重载整个页面，只需要重载页面中的一个框架页(减少了数据的传输，增加了网页下载速度)

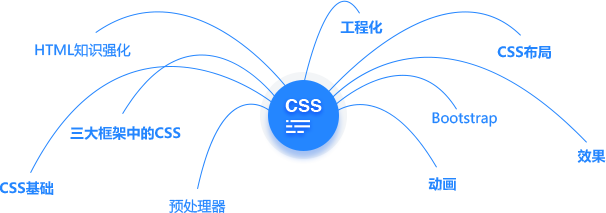
总而言之：

运维性网站或继承性开发的网站，可以使用iframe;

销售内，官网、展示性网站等建议不使用iframe;

标准的网页设计是不使用iframe的。

二、CSS篇



1、说一下 link 与 @import 的区别和用法？

页面导入外部css文件的方法通常有两种，一种在网页中直接link标签加入，另一种在页面中@import引入css文件。两种引入形式如下：

link引入形式：

<link href="styles.css" type="text/css" />

@import引用形式：

<style type="text/css">

@import url("styles.css");

</style>

1、适用范围不同

@import可以在网页页面中使用，也可以在css文件中使用，用来将多个css文件引入到一个css文件中；而link只能将css文件引入到网页页面中。

2、功能范围不同

link属于XHTML标签，而@import是CSS提供的一种方式，link标签除了可以加载CSS外，还可以定义rel连接属性，定义RSS等，@import就只能加载CSS。

3、加载顺序不同

页面被加载的时候，link引用的CSS会同时被加载，而@import引用的CSS会等到页面全部被下载完再被加载。所以有时候浏览@import加载CSS的页面时开始会没有样式（就是闪烁）

4、兼容性

由于@import是css2.1提出的，所以老的浏览器不支持，@import只有在IE5以上的才能识别，而link标签无此问题。

5、控制样式时的差别

使用link方式可以让用户切换CSS样式.现代浏览器如Firefox,Opera,Safari都支持rel=”alternate stylesheet”属性(即可在浏览器上选择不同的风格),当然你还可以使用Javascript使得IE也支持用户更换样式。

6、使用DOM控制样式时的差别

当使用JavaScript控制DOM去改变样式的时候，只能使用link标签，因为@import不是DOM可以控制的。

2、rgba和opacity的透明效果有什么不同？

opacity

opacity是一个属性。opacity属性的值，可以被其子元素继承，给父级div设置opacity属性，那么所有子元素都会继承这个属性，并且，该元素及其继承该属性的所有子元素的所有内容透明度都会改变。

rgba（0，0，0，0.5）

rgba是一个属性值。rgba设置的元素，只对该元素的背景色有改变，并且，该元素的后代不会继承该属性。

3、display:none与visibility:hidden的区别？

这两个属性都是让元素隐藏，不可见。两者区别如下：

（1）在渲染树中

display:none会让元素完全从渲染树中消失，渲染时不会占据任何空间；

visibility:hidden不会让元素从渲染树中消失，渲染的元素还会占据相应的空间，只是内容不可见。

（2）是否是继承属性

display:none是非继承属性，子孙节点会随着父节点从渲染树消失，通过修改子孙节点的属性也无法显示；

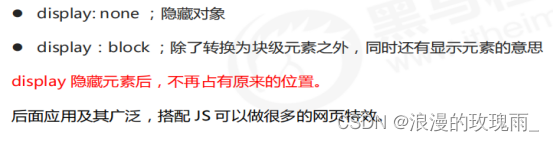
visibility:hidden是继承属性，子孙节点消失是由于继承了hidden，通过设置visibility:visible可以让子孙节点显示；

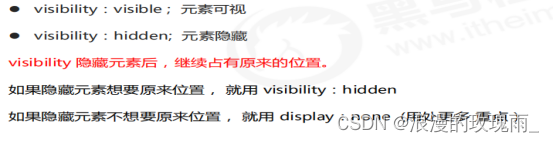
（3）修改常规文档流中元素的 display 通常会造成文档的重排，但是修改visibility属性只会造成本元素的重绘；

（4）如果使用读屏器，设置为display:none的内容不会被读取，设置为visibility:hidden的内容会被读取。

（5）display：none 隐藏对应的元素，在文档布局中不再给它分配空间，它各边的元素会合拢，就当他从来不存在。

（6）visibility：hidden 隐藏对应的元素，但是在文档布局中仍保留原来的空间。



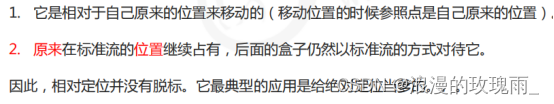




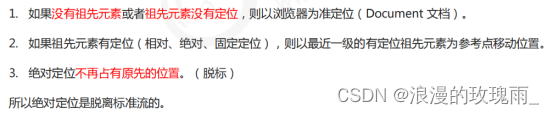
4、定位布局 position中的relative、absolute、fixed、sticky它们之间的区别？

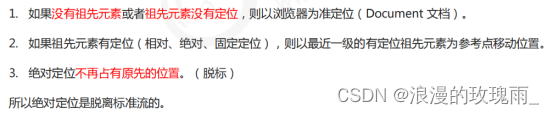


（1）relative:相对定位，相对于自己本身在正常文档流中的位置进行定位。

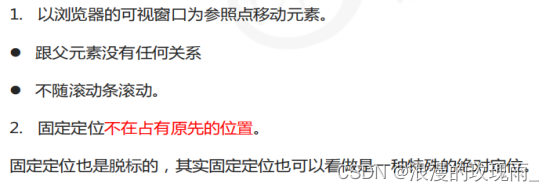


（2）absolute:生成绝对定位，相对于最近一级定位不为static的父元素进行定位。





（3）fixed: 生成绝对定位，相对于浏览器窗口或者frame进行定位。（老版本IE不支持）



（4）static:默认值，没有定位，元素出现在正常的文档流中。（很少用）

（5）sticky:生成粘性定位的元素，容器的位置根据正常文档流计算得出。（很少用）

5、如何用CSS3画一条0.5px的直线？

height: 1px;

transform: scale(0.5);

6、如何用CSS3画一个三角形？

<style>

.up{

width:0;

height:0;

border: 100px solid transparent;

border-top: 100px solid red;/\*红色\*/

}

.down{

width:0;

height:0;

border: 100px solid transparent;

border-bottom: 100px solid blue;/\*蓝色\*/

}

.left{

width:0;

height:0;

border: 100px solid transparent;

border-left: 100px solid pink;/\*黑色\*/

}

.right{

width:0;

height:0;

border: 100px solid transparent;

border-right: 100px solid pink;/\*黄色\*/

}

</style>

<body>

<div class="up"></div>

<div class="down"></div>

<div class="left"></div>

<div class="right"></div>

</body>

7、CSS3盒子模型：标准盒模型、怪异盒模型

盒子模型分为两种:

第一种是 W3C 标准的盒子模型（标准盒模型）

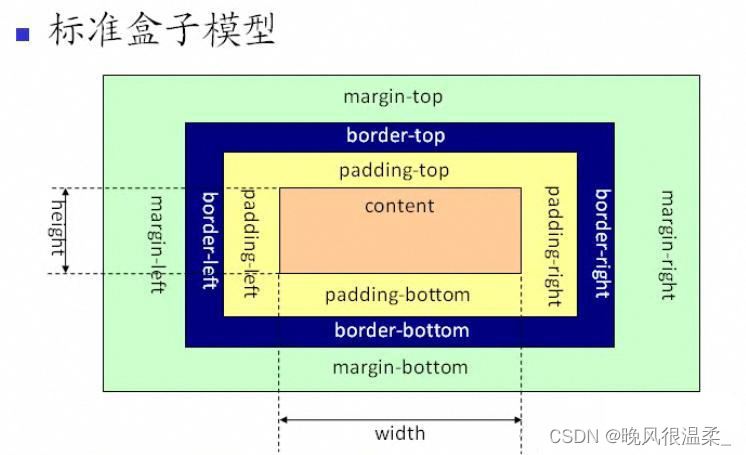
第二种 IE 标准的盒子模型（怪异盒模型）

标准盒模型与怪异盒模型的表现效果的区别之处：

1、标准盒模型中 width 指的是内容区域 content 的宽度

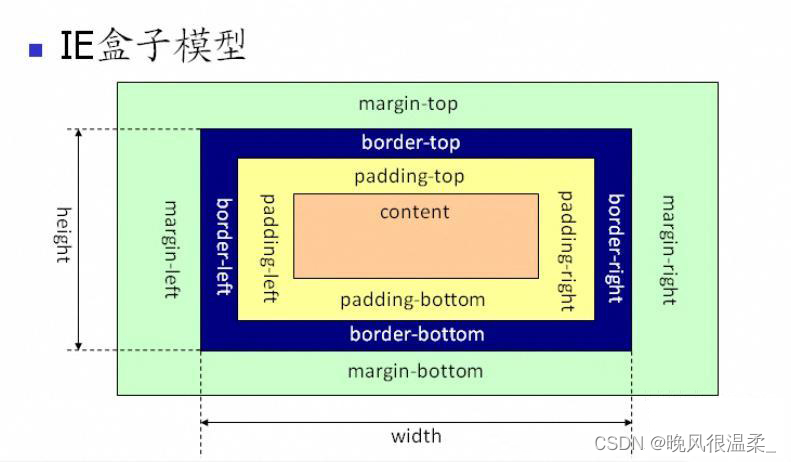
height 指的是内容区域 content 的高度

标准盒模型下盒子的大小 = content + border + padding + margin



2、怪异盒模型中的 width 指的是内容、边框、内边距总的宽度（content + border + padding）；height 指的是内容、边框、内边距总的高度

怪异盒模型下盒子的大小=width（content + border + padding） + margin



除此之外，我们还可以通过属性 box-sizing 来设置盒子模型的解析模式

可以为 box-sizing 赋两个值：

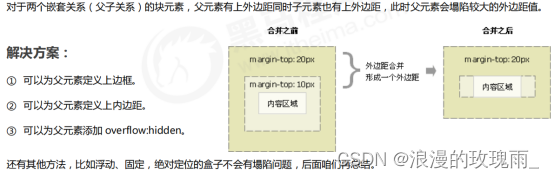
1、box-sizing: content-box：默认值，border 和 padding 不算到 width 范围内，可以理解为是 W3c 的标准模型(default)。总宽=width+padding+border+margin

2、box-sizing: border-box：border 和 padding 划归到 width 范围内，可以理解为是 IE 的怪异盒模型，总宽=width+margin

相邻块元素垂直外边距的合并

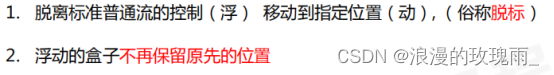


嵌套块元素垂直外边距的塌陷

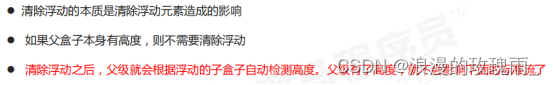


8、浮动（float)以及清除浮动的方法

(1) 脱标



(2) 清除浮动



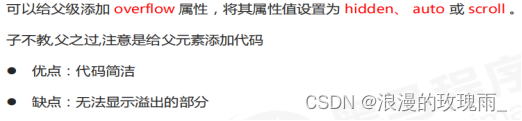


(3) 清除浮动的方法

清除浮动 —— 额外标签法



清除浮动 —— 父级添加 overflow



清除浮动 —— :after 伪元素法



清除浮动 —— 双伪元素清除浮动



9、Flex布局

1、主轴对齐方式

使用justify-content调节元素在主轴的对齐方式



2、侧轴对齐方式

使用align-items调节元素在侧轴的对齐方式



3、换轴

使用flex-direction改变元素排列方向



4、弹性盒子换行

目标：使用flex-wrap实现弹性盒子多行排列效果



10、CSS3中“transform”属性~平面转换

（1）位移：transform: translate(水平移动距离, 垂直移动距离)



（2）旋转

transform: rotate(角度); 注意：角度单位是deg

取值为正, 则顺时针旋转 Ø 取值为负, 则逆时针旋转

（3）缩放

transform: scale(x轴缩放倍数, y轴缩放倍数);

transform: scale(缩放倍数);

scale值大于1表示放大, scale值小于1表示缩小

（4）transition的基本用法

transition：[属性名] [持续时间] [速度曲线] [延迟时间]

我们可以很方便的用这个过渡来给某一个属性加上好看的动效。

例如，高度属性的值改变时，延迟 0.5 秒后以 ease 曲线进行过渡，持续2秒：

transition:height 2s ease 0.5s

或者一个属性不够，想要监听所有属性。

transition: all 2s ease .5s

11、CSS3中 “子绝父相” 定位布局

弄清楚这个口诀，就明白了绝对定位和相对定位的使用场景。

这个“子绝父相”太重要了，是我们学习定位的口诀，是定位中最常用的一种方式这句话的意思是：子级是绝对定位的话，父级要用相对定位。

① 子级绝对定位，不会占有位置，可以放到父盒子里面的任何一个地方，不会影响其他的兄弟盒子。

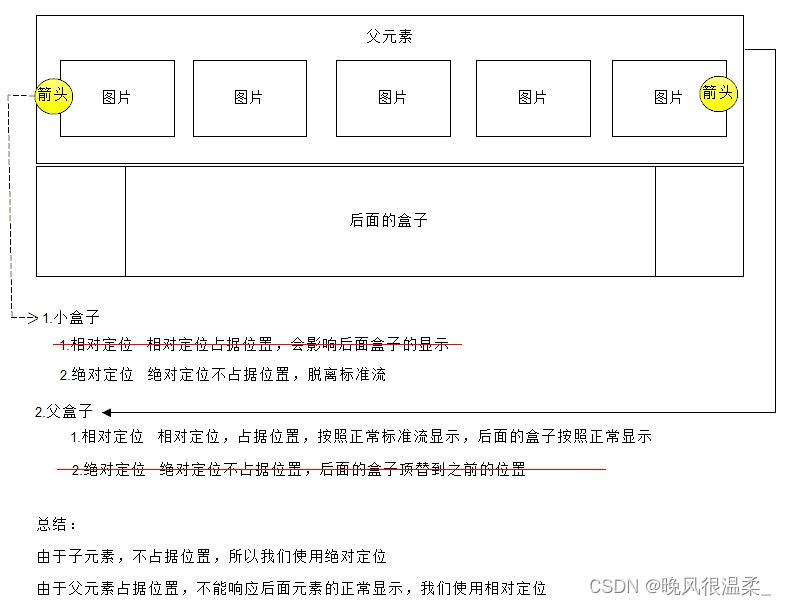
② 父盒子需要加定位限制子盒子在父盒子内显示。

③ 父盒子布局时，需要占有位置，因此父亲只能是相对定位。

这就是子绝父相的由来，所以相对定位经常用来作为绝对定位的父级。

总结： 因为父级需要占有位置，因此是相对定位， 子盒子不需要占有位置，则是绝对定位

当然，子绝父相不是永远不变的，如果父元素不需要占有位置，子绝父绝也会遇到。



12、盒子居中的几种方法：“子绝父相”、“Flex布局”、“transform”

（1）利用定位（子绝父相）、margin-left、margin-top实现



（2）利用定位（子绝父相）、transform属性实现



（3）利用flex布局实现盒子居中



13、CSS3中有哪些新特性？

新增各种CSS选择器 （: not(.input)：所有 class 不是“input”的节点）

圆角 （border-radius:8px）

多列布局 （multi-column layout）

阴影和反射 （Shadoweflect）

文字特效 （text-shadow）

文字渲染 （Text-decoration）

线性渐变 （gradient）

旋转 （transform）

增加了旋转,缩放,定位,倾斜,动画,多背景

14、CSS3选择器及其优先级



对于选择器的优先级：

标签选择器、伪元素选择器：1

类选择器、伪类选择器、属性选择器：10

id 选择器：100

内联样式：1000

注意事项：

!important声明的样式的优先级最高；

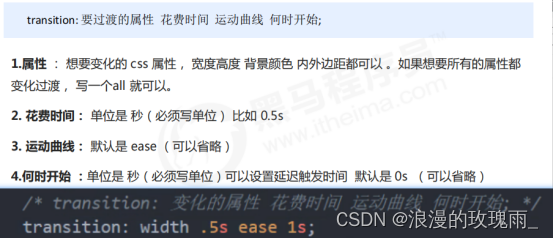
如果优先级相同，则最后出现的样式生效；

继承得到的样式的优先级最低；

通用选择器（\*）、子选择器（>）和相邻同胞选择器（+）并不在这四个等级中，所以它们的权值都为 0 ；

样式表的来源不同时，优先级顺序为：内联样式 > 内部样式 > 外部样式 > 浏览器用户自定义样式 > 浏览器默认样式。

15、CSS3中 “transition” 过渡属性





16、结构伪类选择器&伪元素选择器

1、结构伪类选择器：可方便的选取一个或多个特定的元素

:first-child 选取属于其父元素的首个子元素

:last-child 选取属于其父元素的最后一个子元素

:nth-child(n) 选择第n个子元素

　　　 n=even / 2n ：选取偶数孩子

　　　　n=odd / 2n+1 ：选取奇数孩子

2、伪元素选择器：

::first-letter / line: 文本第一个单词 / 第一行

::selection: 改变选中文本的样式

::before & ::after

这两兄弟特性一样：1.必须要带content属性（可以为空）

　　　　　　　　　2.属于行内盒子

3、属性选择器：

div[class=xx]: 选择类名为xx的div

div[class^=xx]: 选择以类名为xx开头的div

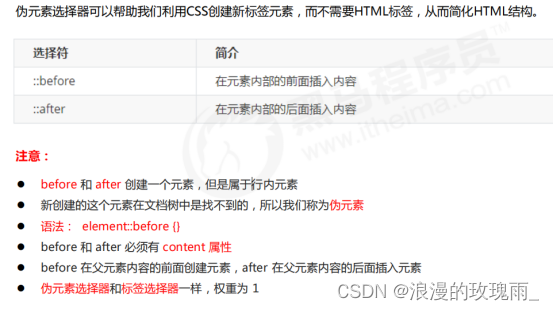
div[class$=xx]: 选择类名是以xx结束的div

div[class\*=xx]: 选择类名带有xx的div

（1）结构伪类选择器



（2）伪元素选择器



17、display的block、inline和inline-block的区别？

（1）block： 会独占一行，多个元素会另起一行，可以设置width、height、margin和padding属性；

（2）inline： 元素不会独占一行，设置width、height属性无效。但可以设置水平方向的margin和padding属性，不能设置垂直方向的padding和margin；

（3）inline-block： 将对象设置为inline对象，但对象的内容作为block对象呈现，之后的内联对象会被排列在同一行内。

对于行内元素和块级元素，其特点如下：

行内元素

设置宽高无效；

可以设置水平方向的margin和padding属性，不能设置垂直方向的padding和margin；

不会自动换行；

块级元素

可以设置宽高；

设置margin和padding都有效；

可以自动换行；

多个块状，默认排列从上到下。

18、定位堆叠顺序z-index

在使用定位布局时，可能会出现盒子重叠的情况。此时，可以使用 z-index 来控制盒子的前后次序 (z轴)

语法： 选择器 { z-index: 1; }

数值可以是正整数、负整数或 0, 默认是 auto，数值越大，盒子越靠上

如果属性值相同，则按照书写顺序，后来居上

数字后面不能加单位

只有定位的盒子才有 z-index 属性

三、HTML&&CSS混合篇

1、Localstorage、sessionStorage、cookie 的区别

共同点：都是保存在浏览器端、且同源的

三者区别：

1、cookie 数据始终在同源的 http 请求中携带（即使不需要），即 cookie 在浏览器和服务器间来回传递，而 sessionStorage 和 localStorage 不会自动把数据发送给服务器，仅在本地保存。 cookie 数据还有路径（path）的概念，可以限制 cookie 只属于某个路径下

2、存储大小限制也不同，cookie 数据不能超过 4K，同时因为每次 http 请求都会携带 cookie、 所以 cookie 只适合保存很小的数据，如会话标识。sessionStorage 和 localStorage 虽然也有存 储大小的限制，但比 cookie 大得多，可以达到 5M 或更大。

3、数据有效期不同

sessionStorage：仅在当前浏览器窗口关闭之前有效；

localStorage：始终有效，窗口或浏览器关闭也一直保存，因此用作持久数据；

cookie：只在设置的 cookie 过期时间之前有效，即使窗口关闭或浏览器关闭

4、作用域不同

sessionStorage 不在不同的浏览器窗口中共享，即使是同一个页面；

localstorage 在所有同源窗口中都是共享的；

cookie 也是在所有同源窗口中都是共享的

5、web Storage 支持事件通知机制，可以将数据更新的通知发送给监听者。

6、web Storage 的 api 接口使用更方便。

2、如何实现双飞翼（圣杯）布局？

1、利用定位实现两侧固定中间自适应

父盒子设置左右 padding 值

给左右盒子的 width 设置父盒子的 padding 值,然后分别定位到 padding 处.

中间盒子自适应

2、利用 flex 布局实现两侧固定中间自适应

父盒子设置 display：flex

左右盒子设置固定宽高

中间盒子设置 flex：1

3、伪元素和伪类的区别和作用？

伪元素：在内容元素的前后插入额外的元素或样式，但是这些元素实际上并不在文档中生成。它们只在外部显示可见，但不会在文档的源代码中找到它们，因此，称为“伪”元素。例如：

p::before {content:"第一章：";}

p::after {content:"Hot!";}

p::first-line {background:red;}

p::first-letter {font-size:30px;}

伪类：将特殊的效果添加到特定选择器上。它是已有元素上添加类别的，不会产生新的元素。例如：

a:hover {color: #FF00FF}

p:first-child {color: red}

总结： 伪类是通过在元素选择器上加⼊伪类改变元素状态，⽽伪元素通过对元素的操作进⾏对元素的改变。

4、img 的 alt 与 title 的异同，还有实现图片懒加载的原理？

异同

alt 是图片加载失败时，显示在网页上的替代文字； title 是鼠标放上面时显示的文字,title

是对图片的描述与进一步说明;

这些都是表面上的区别，alt 是 img 必要的属性，而 title 不是 对于网站 seo 优化来说，title 与 alt 还有最重要的一点： 搜索引擎对图片意思的判断，主 要靠 alt 属性。所以在图片 alt 属性中以简要文字说明，同时包含关键词，也是页面优化的 一部分。条件允许的话，可以在 title 属性里，进一步对图片说明 由于过多的图片会严重影响网页的加载速度，并且移动网络下的流量消耗巨大，所以 说延迟加载几乎是标配了。

原理

图片懒加载的原理很简单，就是我们先设置图片的 data-set 属性（当然也可以是其他任意的， 只要不会发送 http 请求就行了，作用就是为了存取值）值为其图片路径，由于不是 src，所 以不会发送 http 请求。 然后我们计算出页面 scrollTop 的高度和浏览器的高度之和， 如果 图片举例页面顶端的坐标 Y（相对于整个页面，而不是浏览器窗口）小于前两者之和，就说明图片就要显示出来了（合适的时机，当然也可以是 其他情况），这时候我们再将data-set 属性替换为 src 属性即可。

5、BFC 是什么？

定义

BFC (Block formatting context) 直译为 "块级格式化上下文"。它是一个独立的渲染区域，

只有 Block-level box 参与，它规定了内部的 Block-level Box 如何布局，并且与这个区

域外部毫不相干。

布局规则

1、内部的 Box 会在垂直方向，一个接一个地放置

2、Box 垂直方向的距离由 margin 决定。属于同一个 BFC 的两个相邻 Box 的 margin

会发生重叠

3、每个元素的 margin box 的左边， 与包含块 border box 的左边相接触(对于从左往

右的格式化，否则相反)。即使存在浮动也是如此

4、BFC 的区域不会与 float box 重叠

5、BFC 就是页面上的一个隔离的独立容器，容器里面的子元素不会影响到外面的元素。反之也如此

6、计算 BFC 的高度时，浮动元素也参与计算

哪些元素会生成 BFC

1、根元素

2、float 属性不为 none

3、position 为 absolute 或 fixed

4、display 为 inline-block， table-cell， table-caption， flex， inline-flex

5、overflow 不为 visible

四、JavaScript篇

1、JS基础类型和复杂类型

JS数据基础类型有：

String、Number、Boolean、Null、undefined五种基本数据类型，加上新增的两种ES6的类型Symbol、BigInt

JS有三种 复杂类型 （引用数据类型）：

Object（对象）、Array（数组）、function（函数）

2、箭头函数与普通函数的区别？

(1) 箭头函数比普通函数更加简洁

如果没有参数 , 就直接写一个空括号即可 , 如果只有一个参数 , 可以省去参数的括号 如果有多个参数 , 用逗号分割 , 如果函数体的返回值只有一句 , 可以省略大括号。

(2) 箭头函数没有自己的this

箭头函数不会创建自己的this, 所以它没有自己的this, 它只会在自己作用域的上一层继承this。所以箭头函数中this的指向在它在定义时已经确定了, 之后不会改变。

(3) 箭头函数继承来的this指向永远不会改变

(4) call()、apply()、bind()等方法不能改变箭头函数中this的指向

(5) 箭头函数不能作为构造函数使用

由于箭头函数时没有自己的this，且this指向外层的执行环境，且不能改变指向，所以不能当做构造函数使用。

(6) 箭头函数没有自己的arguments对象。在箭头函数中访问arguments实际上获得的是它外层函数的arguments值。

(7) 箭头函数没有prototype

(8) 补充：箭头函数的this指向哪⾥？

箭头函数不同于传统JavaScript中的函数，箭头函数并没有属于⾃⼰的this，它所谓的this是捕获其所在上下⽂的 this 值，作为⾃⼰的 this 值，并且由于没有属于⾃⼰的this，所以是不会被new调⽤的，这个所谓的this也不会被改变。

3、JS中null和undefined的判断方法和区别？

1.undefined 的判断

(1) undefined表示缺少值，即此处应该有值，但是还没有定义

(2) 变量被声明了还没有赋值，就为undefined

(3) 调用函数时应该提供的参数还没有提供，该参数就等于undefined

(4) 对象没有赋值的属性，该属性的值就等于undefined

(5) 函数没有返回值，默认返回undefined

2.null 的判断

(1) null表示一个值被定义了，但是这个值是空值

(2) 作为函数的参数，表示函数的参数不是对象

(3) 作为对象原型链的终点 （Object.getPrototypeOf(Object.prototype)）

(4) 定义一个值为null是合理的，但定义为undefined不合理（var name = null）

3.typeof 类型不同

typeof null; // 'object'

typeof undefined; // 'undefined'

运行运行

4.Number() 转数字也不同

Number(null); // 0

Number(undefined); // NaN

运行运行

4、原型链

前提须知：

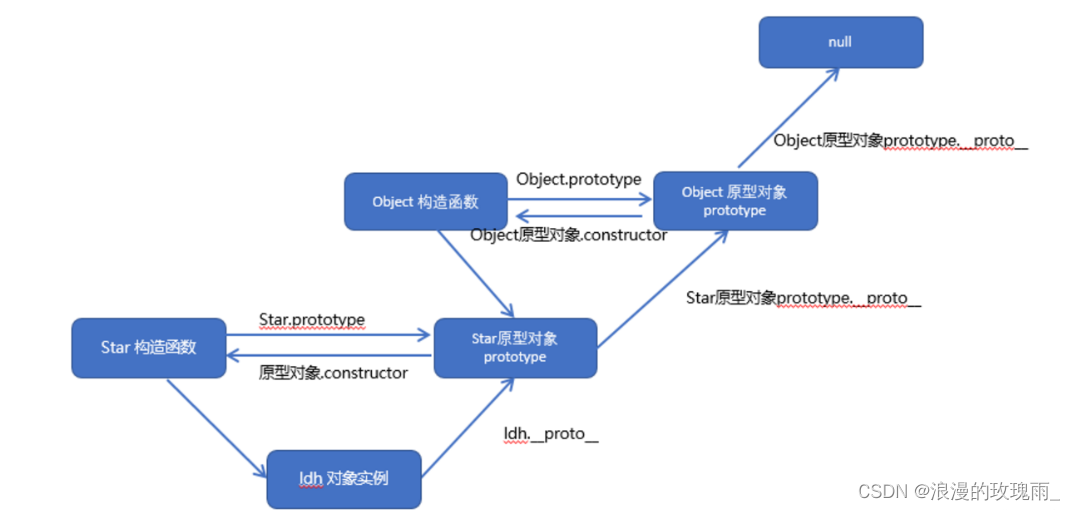
(1) prototype：所有的函数都有原型prototype属性，这个属性指向函数的原型对象。

(2) \_\_proto\_\_，这是每个对象(除null外)都会有的属性，叫做\_\_proto\_\_，这个属性会指向该对象的原型。

(3) constructor: 每个原型都有一个constructor属性，指向该关联的构造函数。

原型链：获取对象时，如果这个对象上本身没有这个属性时，它就会去它的原型\_\_proto\_\_上去找，如果还找不到，就去原型的原型上去找...一直直到找到最顶层（Object.prototype）为止，Object.prototype对象也有\_\_proto\_\_属性，值为null

此外，每一个prototype原型上都会有一个constructor属性，指向它关联的构造函数。



5、v-show 与 v-if 的区别？

v-show指令是通过修改元素的display的CSS属性让其显示或者隐藏；

v-if指令是直接 销毁 和 重建 DOM达到让元素显示和隐藏的效果；

使用 v-show会更加节省性能上的开销；当只需要一次显示或隐藏时，使用v-if更加合理。

6、keep-alive 的作用是什么?

官网解释：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。

作用：实现组件缓存，保持这些组件的状态，以避免反复渲染导致的性能问题。 需要缓存组件 频繁切换，不需要重复渲染。

场景：tabs标签页 后台导航，vue性能优化。

原理：Vue.js内部将DOM节点抽象成了一个个的VNode节点，keep-alive组件的缓存也是基于VNode节点的而不是直接存储DOM结构。它将满足条件（pruneCache与pruneCache）的组件在cache对象中缓存起来，在需要重新渲染的时候再将vnode节点从cache对象中取出并渲染。

7、闭包的理解?

概念：有权访问另一个函数内部变量的函数。

本质：是指有权访问另一个函数作用域中变量的函数，创建闭包的最常见的方式就是在一个函数内创建另一个函数，通过另一个函数访问这个函数的局部变量，利用闭包可以突破作用链域，将函数内部的变量和方法传递到外部。

面试：什么是闭包？

通俗的来说：闭包是在一个函数内部在定一个函数，然后内部函数访问外部函数的一个变量就会形成闭包，闭包的话会形成一个私有空间，然后避免全局变量的一个污染，然后会持久化存储数据到内存中，但是闭包也有弊端，它会导致内存泄漏

拓展：内存泄漏怎么解决？

首先避免它的使用，其次的话就是变量执行完以后，可以让它赋值为null，最后利用JS的一个垃圾回收机制进行回收

闭包用处：

读取内部函数的变量；

这些变量的值始终会保持在内存中，不会在外层函数调用后被自动清除

闭包优点：

变量会一直在内存中；

避免全局变量的污染；

私有变量的存在；

闭包缺点：

变量长期储存在内存中，会增大内存的使用量，使用不当会造成内存泄露

判断闭包的3个特点：

　　1.函数嵌套函数；

　　2.内部函数一定操作了外部函数的局部变量；

　　3.外部函数一定将内部函数返回到外部并保存在一个全局变量中；

判断闭包的执行结果：

　　1.外部函数被调用了几次就有几个受保护的局部变量的副本；

　　2.来自一个闭包的函数被调用几次，受保护的局部变量就变化几次；

闭包特性：

函数嵌套函数；

内部函数可以直接使用外部函数的局部变量；

变量或参数不会被垃圾回收机制回收；

8、JS垃圾回收机制

垃圾回收机制（Garbage Collection）简称GC：是JavaScript中使用的内存管理系统的基本组成部分

JavaScript 是在创建变量（对象，字符串等）时自动进行了分配内存，并且在不使用它们时会 “自动” 释放、释放的过程成为垃圾回收

内存在不使用的时候会被垃圾回收器自动回收

9、nextTick的实现？

nextTick是Vue提供的一个全局API,是在下次DOM更新循环结束之后执行延迟回调，在修改数据之后使用$nextTick，则可以在回调中获取更新后的DOM。

Vue在更新DOM时是异步执行的。只要侦听到数据变化，Vue将开启1个队列，并缓冲在同一事件循环中发生的所有数据变更。如果同一个watcher被多次触发，只会被推入到队列中-次。这种在缓冲时去除重复数据对于避免不必要的计算和DOM操作是非常重要的。nextTick方法会在队列中加入一个回调函数，确保该函数在前面的dom操作完成后才调用

比如，我在干什么的时候就会使用nextTick，传一个回调函数进去，在里面执行dom操作即可。简单了解nextTick的实现，它会在callbacks里面加入我们传入的函数，然后用timerFunc异步方式调用它们，首选的异步方式会是Promise。这让我明白了为什么可以在nextTick中看到dom操作结果。

实现原理：在下次 DOM 更新循环结束之后执行延迟回调，在修改数据之后立即使用 nextTick 来获取更新后的 DOM。 nextTick主要使用了宏任务和微任务。 根据执行环境分别尝试采用Promise、MutationObserver、setImmediate，如果以上都不行则采用setTimeout定义了一个异步方法，多次调用nextTick会将方法存入队列中，通过这个异步方法清空当前队列。

10、混入mixin的原理？

mixin 项目变得复杂的时候，多个组件间有重复的逻辑就会用到mixin

多个组件有相同的逻辑，抽离出来，其实mixin并不是完美的解决方案，会存在一些问题

如：vue3提出的Composition API旨在解决这些问题【追求完美是要消耗一定的成本的，如开发成本】

场景：PC端新闻列表和详情页一样的右侧栏目，可以使用mixin进行混合

劣势：

1. 变量来源不明确，不利于阅读

2. 多mixin可能会造成命名冲突

3. mixin和组件可能出现多对多的关系，使得项目复杂度变高

11、js列举和数组操作相关的方法(常用)

在JavaScript中，数组操作是编程的常见任务之一。JavaScript提供了一系列与数组操作相关的方法，这些方法可以用来操纵数组，如添加、删除、查找、排序和遍历元素等。以下是一些常用的数组方法：

添加元素

push(): 在数组的末尾添加一个或多个元素，并返回新的长度。

unshift(): 在数组的开头添加一个或多个元素，并返回新的长度。

删除元素

pop(): 删除数组的最后一个元素，并返回那个元素。

shift(): 删除数组的第一个元素，并返回那个元素。

splice(): 通过删除、替换或添加新元素来改变数组的内容。

查找元素

indexOf(): 查找元素在数组中的索引。

lastIndexOf(): 从数组的末尾开始查找元素。

find(): 查找第一个满足测试函数的元素。

findIndex(): 查找第一个满足测试函数的元素的索引。

排序和翻转

sort(): 对数组元素进行排序。

reverse(): 颠倒反转数组中元素的顺序。

遍历和过滤

forEach(): 对数组的每个元素执行一次提供的函数。

map(): 创建一个新数组，其结果是该数组中的每个元素都调用一次提供的函数后的返回值。

filter(): 创建一个新数组，包含通过所提供函数实现的测试的所有元素。

reduce(): 对数组中的每个元素执行一个由您提供的reducer函数（升序执行），将其结果汇总为单个返回值。

reduceRight(): 类似reduce()，但是从数组的末尾开始累加。

归并数组

concat(): 合并两个或多个数组。

slice(): 提取原数组的一部分，返回一个新数组。

其它方法

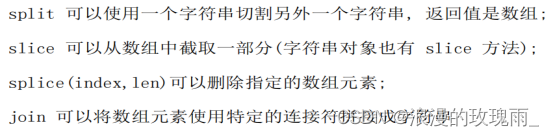
join(): 将数组（或一个类数组对象）的所有元素连接成一个字符串并返回这个字符串。

toString(): 返回一个表示数组内容的字符串。

split(): 把字符串转换成数组。

toLocaleString(): 返回一个表示数组内容的本地化字符串。

这些方法可以在日常编程中大大简化数组操作的复杂性。使用这些方法时，需要注意它们的返回值和副作用。例如，push 和 pop 方法会改变原数组，而 concat 方法则会返回一个新数组。在编写代码时，了解这些方法的特性对于避免不必要的错误和优化性能至关重要。



12、typeof和instanceof的区别是什么？

（1）typeof 的返回值是一个字符串，用来说明变量的数据类型；

typeof用于数据类型的判断，返回值有number、string、boolean、function、undefined、object 六个。但是，在其中你会发现，typeof判断null、array、object以及函数的实例（new + 函数）时，它返回的都是object。这就导致在判断这些数据类型的时候得不到真实的数据类型。所以，typeof 存在的弊端——它虽然可以判断基本数据类型（null 除外），但是引用数据类型中，除了function 类型以外，其他的也无法判断。

（2）instanceof的返回值是布尔值，用于判断一个变量是否属于某个对象的实例。instanceof 可以准确地判断复杂引用数据类型，但是不能正确判断基础数据类型。

13、JS中 “==“和“===“的区别详解

区别

===：三个等号我们称为等同符，当等号两边的值为相同类型的时候，直接比较等号两边 的值，值相同则返回 true，若等号两边的值类型不同时直接返回 false。也就是说三个等号既要判断值也要判断类型是否相等。

==：两个等号我们称为等值符，当等号两边的值为相同类型时比较值是否相同，类型不同时会发生类型的自动转换，转换为相同的类型后再作比较。也就是说两个等号只要值相等就可以。

14、如何用原生 JS给一个按钮绑定两个 onclick 事件？

<button id="btn">点击</button>

<script>

//通过事件监听 绑定多个事件

let btn = document.getElementById('btn')

btn.addEventListener('click', one)

btn.addEventListener('click', two)

function one() {

alert('第一个')

}，

function two() {

alert('第二个')

}，

</script>

15、var、let和const的区别？

|  |  |  |  |
| --- | --- | --- | --- |
| 区别 | var | let | const |
| 块级作用域 | ❌ | ✔️ | ✔️ |
| 是否存在变量提升 | ✔️ | ❌ | ❌ |
| 是否添加全局属性 | ✔️ | ❌ | ❌ |
| 重复声明同名变量 | ✔️ | ❌ | ❌ |
| 是否存在暂时性死区 | ❌ | ✔️ | ✔️ |
| 是否必须设置初始值 | ❌ | ❌ | ✔️ |
| 能否改变指针方向 | ✔️ | ✔️ | ❌ |

（1）块级作用域： 块作用域由 { }包括，let和const具有块级作用域，var不存在块级作用域。块级作用域解决了 ES5 中的两个问题：

内层变量可能覆盖外层变量

用来计数的循环变量泄露为全局变量

（2）变量提升： var存在变量提升，let和const不存在变量提升，即在变量只能在声明之后使用，否在会报错。

（3）给全局添加属性： 浏览器的全局对象是window，Node的全局对象是global。var声明的变量为全局变量，并且会将该变量添加为全局对象的属性，但是let和const不会。

（4）重复声明： var声明变量时，可以重复声明变量，后声明的同名变量会覆盖之前声明的遍历。const和let不允许重复声明变量。

（5）暂时性死区： 在使用let、const命令声明变量之前，该变量都是不可用的。这在语法上，称为暂时性死区。使用var声明的变量不存在暂时性死区。

（6）初始值设置： 在变量声明时，var 和 let 可以不用设置初始值。而const声明变量必须设置初始值。

（7）指针指向： let和const都是ES6新增的用于创建变量的语法。 let创建的变量是可以更改指针指向（可以重新赋值）。但const声明的变量是不允许改变指针的指向。

16、讲解js的call、apply和bind区别？

首先，call apply bind三个方法都可以用来改变函数的this指向，具体区别如下：

call( ) 是接收一个及其以上的参数，第一个参数表示this要指向的对象，其余参数表示调用函数需要传入的参数，返回调用函数的返回结果，属于立即执行函数；

apply( ) 是接收两个参数，第一个参数表示this要指向的对象，第二参数表示调用函数需要传入的参数所组成的数组，返回调用函数的返回结果，属于立即执行函数；

bind( ) 是接收一个及其以上的参数，和call(）一致，但是其返回是一个函数，而不是调用函数的返回结果

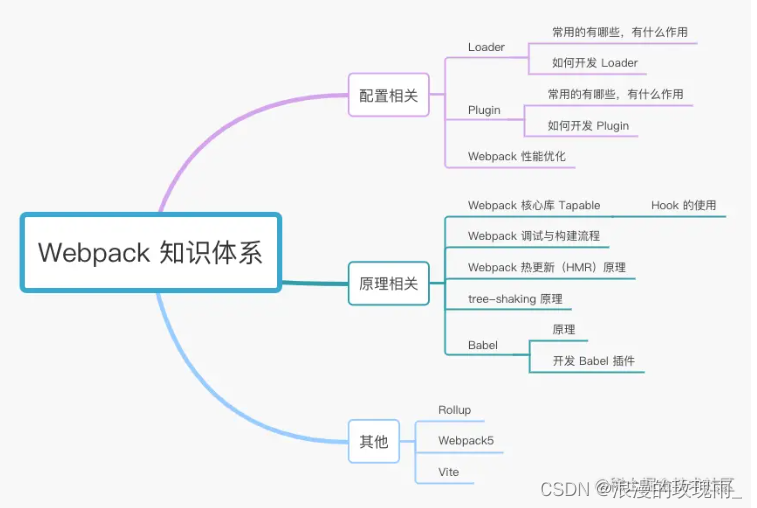
call、apply、bind相同点：都是改变this的指向，传入的第一个参数都是绑定this的指向，在非严格模式中，如果第一个参数是nul或者undefined，会把全局对象（浏览器是window）作为this的值，要注意的是，在严格模式中，null 就是 null，undefined 就是 undefined

call和apply唯一的区别是：call传入的是参数列表，apply传入的是数组，也可以是类数组

bind和call、apply的区别： bind返回的是一个改变了this指向的函数，便于稍后调用，不像call和apply会立即调用；bind和call很像，传入的也是参数列表，但是可以多次传入，不需要像call，一次传入

值得注意：当 bind 返回的函数 使用new作为构造函数时，绑定的 this 值会失效，this指向实例对象，但传入的参数依然生效 （new调用的优先级 > bind调用）

17、谈谈你对webpack的理解？



1、谈谈你对Webpack的理解

Webpack是一个模块打包工具，可以使用它管理项目中的模块依赖，并编译输出模块所需的静态文件。

它可以很好地管理、打包开发中所用到的HTML,CSS,JavaScript和静态文件（图片，字体）等，让开发更高效。

对于不同类型的依赖，Webpack有对应的模块加载器，而且会分析模块间的依赖关系，最后合并生成优化的静态资源。

2、Webpack的基本功能有哪些？

代码转换：TypeScript 编译成 JavaScript、SCSS 编译成 CSS 等等

文件优化：压缩 JavaScript、CSS、HTML 代码，压缩合并图片等

代码分割：提取多个页面的公共代码、提取首屏不需要执行部分的代码让其异步加载

模块合并：在采用模块化的项目有很多模块和文件，需要构建功能把模块分类合并成一个文件

自动刷新：监听本地源代码的变化，自动构建，刷新浏览器

代码校验：在代码被提交到仓库前需要检测代码是否符合规范，以及单元测试是否通过

自动发布：更新完代码后，自动构建出线上发布代码并传输给发布系统。

3、Webpack构建过程？

从entry里配置的module开始递归解析entry依赖的所有module

每找到一个module，就会根据配置的loader去找对应的转换规则

对module进行转换后，再解析出当前module依赖的module

这些模块会以entry为单位分组，一个entry和其所有依赖的module被分到一个组Chunk

最后Webpack会把所有Chunk转换成文件输出在整个流程中Webpack会在恰当的时机执行plugin里定义的逻辑

4、有哪些常见的Loader?

optimize-css-assets-plugin：压缩css；

file-loader：把文件输出到一个文件夹中，在代码中通过相对 URL 去引用输出的文件 (处理图片和字体)

url-loader：与 file-loader 类似，区别是用户可以设置一个阈值，大于阈值会交给 file-loader 处理，小于阈值时返回文件 base64 形式编码 (处理图片和字体)

css-loader：加载 CSS，支持模块化、压缩、文件导入等特性

style-loader：把 CSS 代码注入到 JavaScript 中，通过 DOM 操作去加载 CSS

json-loader: 加载 JSON 文件（默认包含）

ts-loader: babel-loader：把 ES6 转换成 ES5

ts-loader: 将 TypeScript 转换成 JavaScript

less-loader：将less代码转换成CSS

eslint-loader：通过 ESLint 检查 JavaScript 代码

vue-loader:加载 Vue单文件组件

5、有哪些常见的Plugin？

html-webpack-plugin：自动创建一个html文件，并把打包好的js插入到html中

uglifyjs-webpack-plugin：不支持 ES6 压缩 ( Webpack4 以前)

mini-css-extract-plugin: 在每一次打包之前，删除整个输出文件夹下的内容；

clean-webpack-plugin: 抽离css代码放到一个单独的文件中；

copy-webpack-plugin: 拷贝文件

webpack-bundle-analyzer: 可视化 Webpack 输出文件的体积 (业务组件、依赖第三方模块)

optimize-css-assets-plugin：压缩css；

6、那你再说一说Loader和Plugin的区别？

Loader 本质就是一个函数，在该函数中对接收到的内容进行转换，返回转换后的结果。 因为 Webpack 只认识 JavaScript，所以 Loader 就成了翻译官，对其他类型的资源进行转译的预处理工作。

Plugin 就是插件，基于事件流框架 Tapable，插件可以扩展 Webpack 的功能，在 Webpack 运行的生命周期中会广播出许多事件，Plugin 可以监听这些事件，在合适的时机通过 Webpack 提供的 API 改变输出结果。

Loader 在 module.rules 中配置，作为模块的解析规则，类型为数组。每一项都是一个 Object，内部包含了 test(类型文件)、loader、options (参数)等属性。

Plugin 在 plugins 中单独配置，类型为数组，每一项是一个 Plugin 的实例，参数都通过构造函数传入。

7、如何优化 Webpack 的构建速度？

（1）使用高版本的 Webpack 和 Node.js

（2）压缩代码

通过 uglifyjs-webpack-plugin 压缩JS代码

通过 mini-css-extract-plugin 提取 chunk 中的 CSS 代码到单独文件，通过 css-loader 的 minimize 选项开启 cssnano 压缩 CSS。

（3）多线程/多进程构建：thread-loader, HappyPack

（4）压缩图片: image-webpack-loader

（5）缩小打包作用域

exclude/include (确定 loader 规则范围)

resolve.modules 指明第三方模块的绝对路径 (减少不必要的查找)

resolve.mainFields 只采用 main 字段作为入口文件描述字段 (减少搜索步骤，需要考虑到所有运行时依赖的第三方模块的入口文件描述字段)

resolve.extensions 尽可能减少后缀尝试的可能性

noParse 对完全不需要解析的库进行忽略 (不去解析但仍会打包到 bundle 中，注意被忽略掉的文件里不应该包含 import、require、define 等模块化语句)

ignorePlugin (完全排除模块)

8、说一下 Webpack 的热更新原理吧？

Webpack 的热更新又称热替换（Hot Module Replacement），缩写为 HMR。 这个机制可以做到不用刷新浏览器而将新变更的模块替换掉旧的模块。

HMR的核心就是客户端从服务端拉去更新后的文件，准确地说是 chunk diff (chunk 需要更新的部分)，实际上 WDS 与浏览器之间维护了一个 Websocket，当本地资源发生变化时，WDS 会向浏览器推送更新，并带上构建时的 hash，让客户端与上一次资源进行对比。客户端对比出差异后会向 WDS 发起 Ajax 请求来获取更改内容(文件列表、hash)，这样客户端就可以再借助这些信息继续向 WDS 发起 jsonp 请求获取该chunk的增量更新。

后续的部分(拿到增量更新之后如何处理？哪些状态该保留？哪些又需要更新？)由 HotModulePlugin 来完成，提供了相关 API 以供开发者针对自身场景进行处理，像react-hot-loader 和 vue-loader 都是借助这些 API 实现 HMR。

9、什么是bundle？什么是chunk？什么是module？

bundle：是webpack打包后的一个文件；

chunk：代码块，一个chunk 可能有很多的模块组成，用于合并和分割代码；

module：模块，在webpack中，一切都是模块，一个文件就是一个模块，她从入口开始查找webpack依赖的所有模块

10、webpack和grunt以及gulp有什么不同？

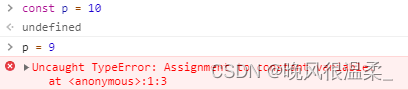
grunt和gulp是基于任务处理的工具，我们需要把我们要做的事分配成各种各样的任务，grunt和gulp会自动执行各种分配的任务，像流水线一样，把资源放上去通过不同的插件进行加工，他的插件非常丰富，能够为我们打造各种工作流；

webpack是模块化打包工具，把所有文件都当作模块进行处理，也就是说webpack和grunt和gulp是两种完全不一样的东西；

18、 const定义的对象属性是否可以改变？

情况一：const定义的变量存在块级作用域，且不存在变量提升，一般用于定义常量，定义的时候必须初始化。

答：不可以，const定义的如果是基本数据类型（string，number，boolean，null，undifined，symbol），定义后就不可再修改，如果修改，会报错。



情况二：那么如果是const定义的对象呢？是否可以修改对象中的属性？

答案：可以

原因：对象是引用类型的，const定义的对象t中保存的是指向对象t的指针，这里的“不变”指的是指向对象的指针不变，而修改对象中的属性并不会让指向对象的指针发生变化，所以用const定义对象，对象的属性是可以改变的。



19、栈溢出及解决方法？

栈溢出(stack Overflow)

缓冲区溢出是由于C语言系列设有内置检查机制来确保复制到缓冲区的数据不得大于缓冲区的大小，因此当这个数据足够大的时候，将会溢出缓冲区的范围。

栈溢出就是缓冲区溢出的一种。 由于缓冲区溢出而使得有用的存储单元被改写, 往往会引发不可预料的后果。程序在运行过程中，为了临时存取数据的需要，一般都要分配一些内存空间，通常称这些空间为缓冲区。如果向缓冲区中写入超过其本身长度的数据，以致于缓冲区无法容纳，就会造成缓冲区以外的存储单元被改写，这种现象就称为缓冲区溢出。缓冲区长度一般与用户自己定义的缓冲变量的类型有关。

由于缓冲区溢出而使得有用的存储单元被改写，往往会引发不可预料的后果。向这些单元写入任意的数据，一般只会导致程序崩溃之类的事故，对这种情况我们也至多说这个程序有Bug。但如果向这些单元写入的是精心准备好的数据，就可能使得程序流程被劫持，致使不希望的代码被执行，落入攻击者的掌控之中，这就不仅仅是bug，而是漏洞(exploit)了。

栈溢出的解决方法

减少栈空间的需求，不要定义占用内存较多的auto变量，应该将此类变量修改成指针，从堆空间分配内存。

函数参数中不要传递大型结构/联合/对象，应该使用引用或指针作为函数参数。

减少函数调用层次，慎用递归函数，例如A->B->C->A环式调用。

20、JS如何实现多线程？

什么是JavaScript的多线程？

JavaScript本身是单线程的，但是可以通过实现多线程来提高性能和用户体验。多线程允许JavaScript在等待用户交互或网络请求时，执行其他任务，从而提高页面加载速度和响应速度。

JavaScript中有哪些实现多线程的方式？

JavaScript有多种实现多线程的方式，包括Web Workers、SharedArrayBuffer、WebAssembly等。其中，Web Workers允许在后台线程中运行JavaScript代码，而SharedArrayBuffer和BufferSource API则允许在多个线程之间共享数据。

如何使用Web Workers实现多线程？

使用Web Workers实现多线程需要创建一个新的worker线程，并将需要执行的代码作为字符串传递给worker。worker线程可以访问全局对象messageChannel的postMessage方法来发送消息，主线程可以使用onmessage方法来接收消息并执行相应的操作。

如何保证多线程安全？

多线程环境下的安全问题主要包括数据竞争和死锁等。为了解决这些问题，需要使用同步机制，如使用Promise、async/await等异步编程方式，或者使用事件循环、共享内存等机制来保证数据的一致性和安全性。

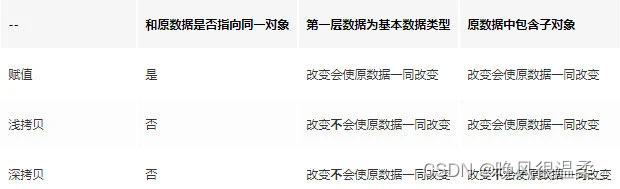
描述一个实际的多线程应用场景。

在实际应用中，多线程可以用于提高页面加载速度和响应速度，例如在电商网站中，可以使用Web Workers在后台线程中加载和处理商品图片，从而提高页面加载速度和用户体验。同时，多个并发请求也可以使用Web Workers并行处理，提高系统性能和响应速度。

21、浅拷贝和深拷贝区别概念常见情况？

浅拷贝：在栈内存中重新开辟一块内存空间，并将拷贝对象储存在栈内存中的数据存放到其中。

深拷贝：基于浅拷贝的过程如果栈内存里面存储的是一个地址，那么其在堆内存空间中的数据也会被重新拷贝一个并由栈空间新创建的地址指向。



1、基本类没有问题

因为，基本类型赋值时，赋的是数据（所以，不存在深拷贝和浅拷贝的问题）。

例如1：

var x = 100;

var y = x; //此时x和y都是100;

如果要改变y的值，x的值不会改变。

2、引用类型有问题

因为，引用类型赋值时，赋的值地址（就是引用类型变量在内存中保存的内容）

例如2：

var arr1 = new Array(12,23,34)

var arr2 = arr1; //这就是一个最简单的浅拷贝

如果要改变arr2所引用的数据：arr2[0]=100时，那么arr1[0]的值也是100。

原因就是 arr1和arr2引用了同一块内存区域（以上的第二点中有体现）。

这是最简单的浅拷贝，因为，只是把arr1的地址拷贝的一份给了arr2，并没有把arr1的数据拷贝一份。所以，拷贝的深度不够。

一、常见的 “浅” 拷贝方法：

除了上面我们演示的对于赋值操作，下面将介绍一些开发中可能会用到，当然也可以会被面试官问到的实现深浅拷贝的方法。

1. Object.assign()

方法解释：方法用于将所有可枚举属性的值从一个或多个源对象分配到目标对象,它将返回目标对象可以实现一个浅拷贝的效果。

参数一：目标对象

参数二：源对象

var obj1 = {

a: 1,

b: 2,

c: ['c', 't', 'r']

}

var obj2 = Object.assign({}, obj1);

obj2.c[1] = 5;

obj2.b = 3

console.log(obj1); // {a:1,b:2,c:["c", 5, "r"]}

console.log(obj2); // {a:1,b:3,c:["c", 5, "r"]}

console.log(obj1.c); // ["c", 5, "r"]

console.log(obj2.c); // ["c", 5, "r"]

运行运行

注意：可见Object.assign()方法对于一维数据是深拷贝效果，但是对于多维数据是浅拷贝效果。Object.assign是一个浅拷贝,它只是在根属性(对象的第一层级)创建了一个新的对象，但是对于属性的值是仍是对象的话依然是浅拷贝，

2. slice()

方法解释：数组进行截取，如果不传参数,会使用默认值,得到一个与原数组元素相同的新数组。

参数一：截取的起始位置

参数二：截取的结束位置

var a = [1, [1, 2], 3, 4];

var b = a.slice();

a[0] = 99

b[1][0] = 2;

console.log(a); // [99,[2,2],3,4]

console.log(b); // [1,[2,2],3,4]

运行运行

注意：可见slice()方法也只是对一维数据进行深拷贝，但是对于多维的数据还是浅拷贝效果。

3. concat()方法

方法解释：数组的拼接(将多个数组或元素拼接形成一个新的数组)，不改变原数组，如果不传参数,会使用默认值，得到一个与原数组元素相同的新数组 (复制数组)。

var a = [1, 2, [3, 4]]

var c = [];

var b = c.concat(a);

a[0] = 99

b[2][1] = 88

console.log(a); // [99,2,[3,88]]

console.log(b); // [1,2,[3,88]]

运行运行

注意：可见concat()方法也只对一维数据具有深拷贝效果，对于多维的数据任然只是浅拷贝

4. ES6拓展运算符

var a = [1, 2, [3, 4]]

var b = [...a];

a[2][1] = 88

b[1] = 99

console.log(a); // [1,2,[3,88]]

console.log(b); // [1,99,[3,88]]

运行运行

注意: 可见ES6的展开运算符对于一维数据是深拷贝效果，但是对于多维数据任然是浅拷贝效果。

二、实现 “深” 拷贝常见方法：

1. JSON.parse(JSON.stringify(obj))

JSON.stringify()是目前前端开发过程中最常用的深拷贝方式，原理是把一个对象序列化成为一个JSON字符串，将对象的内容转换成字符串的形式再保存在磁盘上，再用JSON.parse()反序列化将JSON字符串变成一个新的对象

JSON.stringfy() 将对象序列化成json对象

JSON.parse() 反序列化——将json对象反序列化成js对象

function deepCopy(obj1){

let \_obj = JSON.stringify(obj1);

let obj2 = JSON.parse(\_obj);

return obj2;

}

var a = [1, [1, 2], 3, 4];

var b = deepCopy(a);

b[1][0] = 2;

console.log(a); // 1,1,2,3,4

console.log(b); // 1,2,2,3,4

运行运行

注意：它会抛弃对象的constructor，深拷贝之后，不管这个对象原来的构造函数是什么，在深拷贝之后都会变成Object类型，这种方法能正确处理的对象只有 Number, String, Boolean, Array, 扁平对象，

也就是说，只有可以转成JSON格式的对象才可以这样用，像function没办法转成JSON。

2. 使用第三方库实现对象的深拷贝，比如：lodash、jQuery

import lodash from 'lodash'

var objects = [1,{ 'a': 1 }, { 'b': 2 }];

var deep = lodash.cloneDeep(objects);

deep[0] = 2;

deep[1].a = 2;

console.log(objects); // [1,{ 'a': 1 }, { 'b': 2 }]

console.log(deep); //[2,{ 'a': 2 }, { 'b': 2 }]

3. 递归

这里简单封装了一个deepClone的函数，for in遍历传入参数的值，如果值是引用类型则再次调用deepClone函数，并且传入第一次调用deepClone参数的值作为第二次调用deepClone的参数，如果不是引用类型就直接复制

var obj1 = {

a:{

b:1

}

};

function deepClone(obj) {

var cloneObj = {}; //在堆内存中新建一个对象

for(var key in obj){ //遍历参数的键

if(typeof obj[key] ==='object'){

cloneObj[key] = deepClone(obj[key]) //值是对象就再次调用函数

}else{

cloneObj[key] = obj[key] //基本类型直接复制值

}

}

return cloneObj

}

var obj2 = deepClone(obj1);

obj1.a.b = 2;

console.log(obj2); //{a:{b:1}}

'

运行运行

但是还有很多问题

首先这个deepClone函数并不能复制不可枚举的属性以及Symbol类型

这里只是针对Object引用类型的值做的循环迭代，而对于Array,Date,RegExp,Error,Function引用类型无法正确拷贝

对象循环引用成环了的情况

22、事件循环，Promise和async/await的详解

事件循环event loop它的执行顺序：

一开始整个脚本作为一个宏任务执行

执行过程中同步代码直接执行，宏任务进入宏任务队列，微任务进入微任务队列

当前宏任务执行完出队，检查微任务列表，有则依次执行，直到全部执行完

执行浏览器UI线程的渲染工作

检查是否有Web Worker任务，有则执行

执行完本轮的宏任务，回到2，依此循环，直到宏任务和微任务队列都为空

微任务包括：MutationObserver、Promise.then()或catch()、Promise为基础开发的其它技术，比如fetch API、V8的垃圾回收过程、Node独有的process.nextTick。

宏任务包括：script 、setTimeout、setInterval 、setImmediate 、I/O 、UI rendering。

注意⚠️：在所有任务开始的时候，由于宏任务中包括了script，所以浏览器会先执行一个宏任务，在这个过程中你看到的延迟任务(例如setTimeout)将被放到下一轮宏任务中来执行。

Promise和async/await是JavaScript中处理异步操作的两种方式。

Promise是一种用于处理异步操作的对象。它可以表示一个异步操作的最终完成或失败，并返回相应的结果或错误信息。Promise有三种状态：pending（进行中）、fulfilled（已完成）和rejected（已拒绝）。通过调用Promise的then()方法可以注册回调函数来处理异步操作的结果。

async/await是ES8引入的一种更加简洁的处理异步操作的方式。async函数是一个返回Promise对象的函数，其中可以使用await关键字来等待一个Promise对象的解决。await关键字可以暂停async函数的执行，直到Promise对象解决为止，并返回解决后的结果。

区别：

- 语法上，Promise使用then()和catch()方法来处理异步操作的结果，而async/await使用async函数和await关键字来等待异步操作的结果。

- 可读性上，async/await更加直观和易于理解，代码结构更加清晰，而Promise则需要通过链式调用then()方法来处理多个异步操作。

- 错误处理上，Promise使用catch()方法来捕获错误，而async/await可以使用try-catch语句来捕获错误。

详细解答：

JavaScript的异步机制包括以下几个步骤：

（1）所有同步任务都在主线程上执行，行成一个执行栈

（2）主线程之外，还存在一个任务队列，只要异步任务有了结果，就会在任务队列中放置一个事件

（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，看看里面还有哪些事件，哪些对应的异步任务，于是异步任务结束等待状态，进入执行栈，开始执行

（4）主线程不断的重复上面的第三步

promise的用法

Promise,简单来说就是一个容器，里面保存着某个未来才会结束的时间(通常是一个异步操作的结果)

基本语法

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

//...

resolve('success')

});

obj.then(result => {

console.log(result); //success

});

'

运行运行

promise共有三个状态

pending（执行中）、resolve（成功）、rejected（失败）

链式调用

Promise 链式调用是一种编程模式，允许在异步操作之间顺序执行多个操作。在每个操作中，可以使用 `.then()` 方法返回一个新的 Promise，从而在异步流程中继续执行下一个操作。这样可以避免回调函数地狱，提高代码的可读性和可维护性。

链式调用的基本步骤包括：

创建一个新的 Promise 对象，并调用 `resolve` 或 `reject` 来变更其状态。

在 `then` 或 `catch` 方法中，处理成功或失败的状态。

在 `then` 方法中，可以使用 `return` 关键字返回一个新的 Promise 对象，或者直接返回普通值。

继续在下一个 `then` 方法中处理返回的 Promise 对象，或者直接处理返回的普通值。

例如，以下代码展示了如何使用 `.then()` 和 `.catch()` 方法进行链式调用：

let promise1 = new Promise((resolve, reject) => {

resolve('new promise111111');

});

promise1.then(res => {

console.log(res); // 输出: 'new promise111111'

return '链式调用的方式';

}).then(value => {

console.log(value); // 输出: '链式调用的方式'

});

'

运行运行

在这个例子中，`promise1` 被成功地 resolve 并返回了一个值，然后 `then` 方法被调用，返回了一个新的 Promise 对象，并返回了 `'链式调用的方式'`。这个新的 Promise 对象又被继续 `.then` 处理，最终返回了 `'链式调用的方式'`。

需要注意的是，每次 `.then` 方法调用都会返回一个新的 Promise 对象，因此链式调用的结果取决于最后 `.then` 方法中返回的值或新的 Promise 对象。

错误捕获

Promise.prototype.catch用于指定Promise状态变为rejected时的回调函数，可以认为是.then的简写形势，返回值跟.then一样

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

reject('error');

});

obj.catch(result => {

console.log(result);

})

运行运行

async、await的用法

特点简洁：异步编程的最高境界就是不关心它是否是异步。async、await很好的解决了这一点，将异步强行转换为同步处理。

async/await与promise不存在谁代替谁的说法，因为async/await是寄生于Promise，Generater的语法糖。

用法

async用于申明一个function是异步的，而await可以认为是async wait的简写，等待一个异步方法执行完成。

规则：

1 async和await是配对使用的，await存在于async的内部。否则会报错

2 await表示在这里等待一个promise返回，再接下来执行

3 await后面跟着的应该是一个promise对象，（也可以不是，如果不是接下来也没什么意义了…）

写法

async function demo() {

let a= await sleep(100); //上一个await执行之后才会执行下一句

let b= await sleep(a+ 100);

let c= await sleep(b+ 100);

return c; // console.log(c);

}

demo().then(result => {

console.log(result);

});

错误捕获

如果是reject状态，可以用try-catch捕捉

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

setTimeout(() => {

reject('error');

},1000);

});

async function demo(item) {

try {

let result = await obj;

} catch(e) {

console.log(e);

}

}

demo();

'

运行运行

两者区别

1、promise是ES6，async/await是ES7

2、async/await相对于promise来讲，写法更加优雅

3、reject状态：

（1）promise错误可以通过catch来捕捉，建议尾部捕获错误，

（2）async/await既可以用.then又可以用try-catch捕捉

推荐一篇Promise文章：https://juejin.cn/post/6844904181627781128

23、JS中数组常用方法详解

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **顺序** | **方法名** | **功能** | **返回值** | **是否改变原数组** | **版本** |
| 1 | push() | (在结尾)向数组添加一或多个元素 | 返回新数组长度 | Y | ES5- |
| 2 | unshift() | （在开头)向数组添加一或多个元素 | 返回新数组长度 | Y | ES5- |
| 3 | pop() | 删除数组的最后一位 | 返回被删除的数据 | Y | ES5- |
| 4 | shift() | 移除数组的第一项 | 返回被删除的数据 | Y | ES5- |
| 5 | reverse() | 反转数组中的元素 | 返回反转后数组 | Y | ES5- |
| 6 | sort() | 以字母顺序(字符串Unicode码点)对数组进行排序 | 返回新数组 | Y | ES5- |
| 7 | splice() | 在指定位置删除指定个数元素再增加任意个数元素 （实现数组任意位置的增删改) | 返回删除的数据所组成的数组 | Y | ES5- |
| 8 | concat() | 通过合并（连接）现有数组来创建一个新数组 | 返回合并之后的数组 | N | ES5- |
| 9 | join() | 用特定的字符,将数组拼接形成字符串 (默认",") | 返回拼接后的新数组 | N | ES5- |
| 10 | slice() | 裁切指定位置的数组 | 被裁切的元素形成的新数组 | N | ES5- |
| 11 | toString() | 将数组转换为字符串 | 新数组 | N | ES5- |
| 12 | valueOf() | 查询数组原始值 | 数组的原始值 | N | ES5- |
| 13 | indexOf() | 查询某个元素在数组中第一次出现的位置 | 存在该元素,返回下标,不存在 返回 -1 | N | ES5- |
| 14 | lastIdexOf() | 反向查询数组某个元素在数组中第一次出现的位置 | 存在该元素,返回下标,不存在 返回 -1 | N | ES5- |
| 15 | forEach() | (迭代) 遍历数组,每次循环中执行传入的回调函数 | 无/(undefined) | N | ES5- |
| 16 | map() | (迭代) 遍历数组, 每次循环时执行传入的回调函数,根据回调函数的返回值,生成一个新的数组 | 有/自定义 | N | ES5- |
| 17 | filter() | (迭代) 遍历数组, 每次循环时执行传入的回调函数,回调函数返回一个条件,把满足条件的元素筛选出来放到新数组中 | 满足条件的元素组成的新数组 | N | ES5- |
| 18 | every() | (迭代) 判断数组中所有的元素是否满足某个条件 | 全都满足返回true 只要有一个不满足 返回false | N | ES5- |
| 19 | some() | (迭代) 判断数组中是否存在,满足某个条件的元素 | 只要有一个元素满足条件就返回true,都不满足返回false | N | ES5- |
| 20 | reduce() | (归并)遍历数组, 每次循环时执行传入的回调函数,回调函数会返回一个值,将该值作为初始值prev,传入到下一次函数中 | 最终操作的结果 | N | ES5- |
| 21 | reduceRight() | (归并)用法同reduce,只不过是从右向左 | 同reduce | N | ES5- |
| 22 | includes() | 判断一个数组是否包含一个指定的值. | 是返回 true，否则false | N | ES6 |
| 23 | Array.from() | 接收伪数组,返回对应的真数组 | 对应的真数组 | N | ES6 |
| 24 | find() | 遍历数组,执行回调函数,回调函数执行一个条件,返回满足条件的第一个元素,不存在返回undefined | 满足条件第一个元素/否则返回undefined | N | ES6 |
| 25 | findIndex() | 遍历数组,执行回调函数,回调函数接受一个条件,返回满足条件的第一个元素下标,不存在返回-1 | 满足条件第一个元素下标,不存在=>-1 | N | ES6 |
| 26 | fill() | 用给定值填充一个数组 | 新数组 | Y | ES6 |
| 27 | flat() | 用于将嵌套的数组“拉平”，变成一维的数组。 | 返回一个新数组 | N | ES6 |
| 28 | flatMap() | flat()和map()的组合版 , 先通过map()返回一个新数组,再将数组拉平( 只能拉平一次 ) | 返回新数组 | N | ES6 |

1.push（）在数组最后一位添加一个或多个元素,并返回新数组的长度,改变原数组.(添加多个元素用逗号隔开)

语法: 数组名.push(数据)

作用: 就是往数组末尾添加数据

返回值: 就是这个数组的长度

//push

var arr = [10, 20, 30, 40]

res = arr.push(20)

console.log(arr);//[10,20,30,40,20]

console.log(res);//5

'

运行运行

2. pop（） 末尾出删除数据

语法:数组名.pop()

作用: 就是从数组的末尾删除一个数据

返回值: 就是你删除的那个数据

//pop

var arr = [10, 20, 30, 40]

res =arr.pop()

console.log(arr);//[10,20,30]

console.log(res);//40

'

运行运行

3.unshift（） 头部添加数据

语法: 数组名.unshift(数据)

作用: 就是在数组的头部添加数据

返回值: 就是数组的长度

//pop

var arr = [10, 20, 30, 40]

res=arr.unshift(99)

console.log(arr);//[99,10,20,30,40]

console.log(res);//5

'

运行运行

4.shift（） 头部删除数据

语法: 数组名.shift()

作用: 头部删除一个数据

返回值: 就是删除掉的那个数据

//shift

var arr = [10, 20, 30, 40]

res=arr.shift()

console.log(arr);[20,30,40]

console.log(res);10

'

运行运行

5.reverse（） 翻转数组

语法: 数组名.reverse()

作用: 就是用来翻转数组的

返回值: 就是翻转好的数组

//reverse

var arr = [10, 20, 30, 40]

res=arr.reverse()

console.log(arr);//[40,30,20,10]

console.log(res);//[40,30,20,10]

'

运行运行

6.sort（） 排序

语法一: 数组名.sort() 会排序 会按照位排序

语法二: 数组名.sort(function (a,b) {return a-b}) 会正序排列

语法三: 数组名.sort(function (a,b) {return b-a}) 会倒序排列

//sort()

var arr = [2, 63, 48, 5, 4, 75, 69, 11, 23]

arr.sort()

console.log(arr);

arr.sort(function(a,b){return(a-b)})

console.log(arr);

arr.sort(function(a,b){return(b-a)})

console.log(arr);

'

运行运行

打印结果：

7.splice（） 截取数组

语法一: 数组名.splice(开始索引,多少个)

作用: 就是用来截取数组的

返回值: 是一个新数组 里面就是你截取出来的数据

语法二: 数组名.splice(开始索引,多少个,你要插入的数据)

作用: 删除并插入数据

注意: 从你的开始索引起

返回值: 是一个新数组 里面就是你截取出来的数据

//splice() 语法一

var arr = [2, 63, 48, 5, 4, 75]

res = arr.splice(1,2)

console.log(arr); // [2, 5, 4, 75]

console.log(res); // [63, 48]

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//splice() 语法二

var arr = [2, 63, 48, 5, 4, 75]

res = arr.splice(1,1,99999,88888)

console.log(arr); //[2, 99999, 88888, 48, 5, 4, 75]

console.log(res); // [63]

'

运行运行

二、不改变原数组的方法

1.concat（） 合并数组

功能： 数组的拼接(将多个数组或元素拼接形成一个新的数组),不改变原数组

如果拼接的是数组 则将数组展开,之后将数组中的每一个元素放到新数组中.

如果是其他类型, 直接放到新数组中

另外，如果不给该方法任何参数，将返回一个和原数组一样的数组（复制数组）

var arr1 = [1, 2, 3];

var arr2 = ["a", "b", "c"];

var arr3 = ["A", "B", "C"];

var rel = arr1.concat(arr2, arr3);

console.log(arr1); //原数组

console.log(rel); //新数组

'

运行运行

打印结果：

2.join（） 数组转字符串 语法:数组名.join('连接符') 作用: 就是把一个数组转成字符串

功能：用特定的字符,将数组拼接形成字符串 (默认",")

var list = ["a", "b", "c", "d"]; // "a-b-c-d"

var result = list.join("-"); //"a-b-c-d"

var result = list.join("/"); //"a/b/c/d"

var result = list.join(""); //"abcd"

var result = list.join(); // a,b,c,d

console.log(result);

'

运行运行

find()返回匹配的元素，findIndex()返回匹配元素的索引，find()如果没有匹配到元素则返回undefined, 而findIndex()返回-1

1、数组方法find()

返回数组中满足提供的测试函数的第一个元素的值。否则返回 undefined。

find方法对数组中的每一项元素执行一次 callback 函数，直至有一个 callback 返回 true。当找到了这样一个元素后，该方法会立即返回这个元素的值，否则返回 undefined。

注意 callback 函数会为数组中的每个索引调用即从 0 到 length - 1，而不仅仅是那些被赋值的索引，这意味着对于稀疏数组来说，该方法的效率要低于那些只遍历有值的索引的方法。

callback函数带有3个参数：当前元素的值、当前元素的索引，以及数组本身。

如果提供了 thisArg参数，那么它将作为每次 callback函数执行时的this ，如果未提供，则使用 undefined。

find方法不会改变数组。

实战举例：

var arr = [

{name: '张三', age: 18},

{name: '李四', age: 20},

]

var arrTest = arr.find((item, index, array) => {

console.log('当前值:' + JSON.stringify(item), '当前值的索引：' + index, '当前数组：' + array)

return item.age >= 18

})

console.log(arrTest)

'

运行运行

打印结果：

当前值:{"name":"张三","age":18} 当前值的索引：0 当前数组：[object Object],[object Object]

{name: '张三', age: 18}

2、数组方法findIndex()

返回数组中满足提供的测试函数的第一个元素的索引。否则返回-1。

findIndex方法对数组中的每个数组索引0..length-1（包括）执行一次callback函数，直到找到一个callback函数返回真实值（强制为true）的值。

如果找到这样的元素，findIndex会立即返回该元素的索引。

如果回调从不返回真值，或者数组的length为0，则findIndex返回-1。

与某些其他数组方法（如Array#some）不同，在稀疏数组中，即使对于数组中不存在的条目的索引也会调用回调函数。

回调函数调用时有三个参数：元素的值，元素的索引，以及被遍历的数组。

如果一个 thisArg 参数被提供给 findIndex, 它将会被当作this使用在每次回调函数被调用的时候。如果没有被提供，将会使用 undefined。findIndex不会修改所调用的数组。

实战举例：

var arr = [

{name: '张三', age: 18},

{name: '王二', age: 20},

]

var arrTest = arr.findIndex((item, index, array) => {

console.log('当前值:' + JSON.stringify(item), '当前值的索引：' + index, '当前数组：' + array)

return item.name === '张三'

})

console.log(arrTest)

'

运行运行

打印结果：

当前值:{"name":"张三","age":18} 当前值的索引：0 当前数组：[object Object],[object Object]

0

3、JSON.stringify( ) 和 JSON.parse( )

JSON.stringify()：将对象、数组转换成字符串

JSON.parse()：将字符串转成json对象

JSON.parse(JSON.stringify(obj))我们一般用来深拷贝，其过程说白了 就是利用JSON.stringify 将js对象序列化（JSON字符串），再使用JSON.parse来反序列化(还原)js对象；序列化的作用是存储(对象本身存储的只是一个地址映射，如果断电，对象将不复存在，因此需将对象的内容转换成字符串的形式再保存在磁盘上 )

应用场景

a、浏览器创建、获取（sessionStorage、localStorage）数组内容

b、路由（浏览器地址）传参、获取数组内容

创建、传参的时候使用JSON.stringify()深拷贝

(如果不使用JSON.stringify()存进去的将是[object object]，所以如果我们开发中遇到了获取内容的时候是[object object]不妨试试JSON.stringify())

4、instanceof( )

instanceof运算符返回一个布尔值，如下：

var array = [1, 2, 3];

var obj = {};

console.log( array instanceof Array ) // true

console.log( obj instanceof Object) // true

'

运行运行

5、every()方法的定义与用法：

every()方法用于检测数组中的所有元素是否都满足指定条件（该条件为一个函数）。

every()方法会遍历数组的每一项，如果有有一项不满足条件，则表达式返回false,剩余的项将不会再执行检测；如果遍历完数组后，每一项都符合条，则返回true。

标准用法：

array.every(function(currentValue,index,arr), thisValue)

参数说明：

第一个参数为一个回调函数，必传，数组中的每一项都会遍历执行该函数。

currentValue：必传，当前项的值

index：选传，当前项的索引值

arr：选传，当前项所属的数组对象

第二个参数thisValue为可选参数，回调函数中的this会指向该参数对象。

Tips:

every() 不会对空数组进行检测

every() 不会改变原始数组

代码实例：

var arr = [1000, 2000, 3000]

var flag = arr.every(function (a, b, c) {

console.log(a + "===" + b + "====" + c) //1000===0====1000,2000,3000

return a > 2000;//数组中的每个元素的值都要大于2000的情况,最后才返回true

})

console.log(flag) //false

'

运行运行

6、数组some方法作用

判断数组中是否有满足条件的元素

语法：array.some( function ( item, index, arr) {} ,thisValue)

function ：必须,数组中的每个元素都会执行这个函数

第一个参数:item,必须,当前元素的值

第二个参数:index,可选,当前元素在数组中的索引值

第三个参数:arr,当前元素属于的数组对象

thisValue ：可选，对象作为该执行回调时使用，传递给函数，用作 "this" 的值。

如果省略了 thisValue ，"this" 的值为 "undefined"

some方法特点

(1)函数执行次数 !== 数组长度

(2)函数内部的return

return true : 循环结束,找到了满足条件的元素

return false : 循环继续,没找到循环继续,如果所有元素全部遍历还是没找到,最终结果为false

(3) some()方法不会对空数组进行检测

(4) some()方法不会改变原始数组

应用场景 : 这个方法与every()方法极为相似，但又有所不同，every要求每一项都要符合函数里面的条件，而some只要求有数组中的某一项符合即可，这点类似于或与和。返回的是布尔值，使用事例如下：

var num = [1,2,3,4,5];

var eve = num.some(function(item, index, arr){

return (item > 2);

});

console.log(eve); //true

'

运行运行

7、数组filter()方法

filter()方法的回调函数接受三个参数：当前元素、当前元素的索引和数组本身。其中，当前元素是必需的，而索引和数组是可选的。过滤数组中，符合条件的元素并返回一个新的数组

const people = [

{ name: 'Alice', age: 25 },

{ name: 'Bob', age: 17 },

{ name: 'Charlie', age: 30 },

{ name: 'David', age: 16 },

{ name: 'Eva', age: 18 }

];

const adults = people.filter(person => person.age >= 18);

console.log(adults); // [{ name: 'Alice', age: 25 }, { name: 'Charlie', age: 30 }, { name: 'Eva', age: 18 }]

'

运行运行

五、Vue2篇

​1、什么是前端构建工具?比如（Vue2的webpack，Vue3的Vite）

①首先要明白：浏览器它只认识html, css, js

企业级项目里都可能会具备哪些功能？

1. typescript:·如果遇到ts文件我们需要使用tsc将typescript代码转换为js代码

2. React/Vue:安装react-compiler / vue-complier，将我们写的jsx文件或者.vue文件转换为render函数

3. less/postcss/component-style:我们又需要安装less-loader, sass-loader等一系列编译工具4. 语法降级:babel --->将es的新语法转换旧版浏览器可以接受的语法

5. 体积优化:uglifyjs --->将我们的代码进行压缩变成体积更小性能更高的文件

②前因后果：

因为稍微改一点点东西，非常麻烦！

将App.tsx --->tsc --->· App.jsx --->React-complier --->js文件

但是有一个东西能够帮你把tsc,react-compiler, less, babel,uglifyjs全部集成到一起，我们只需要关心我们写的代码就好了

我们写的代码变化-->有人帮我们自动去tsc, react-compiler,less, babel, uglifyjs全部挨个走一遍-->js

这个东西就叫做 前端构建工具

③前端构建工具的工作：

打包: 将我们写的浏览器不认识的代码交给构建工具进行编译处理的过程就叫做打包，打包完成以后会给我们一个浏览器可以认识的文件。

一个构建工具他到底承担了哪些脏活累活:

1.模块化开发支持：支持直接从node\_modules里引入代码＋多种模块化支持

2.处理代码兼容性：比始babel语法降级，less,ts 语法转换(\*\*不是构建工具做的，构建工具将这些语法对应的处理工具集成进来自动化处理)

3.提高项目性能：压缩文件，\*\*代码分割\*

4.优化开发体验:

（1）构建工具会帮你自动监听文件的变化，当文件变化以后自动帮你调用对应的集成工具进行重新打包，然后再浏览器重新运行（整个过程叫做热更新, hot replacement）

（2）开发服务器:跨域的问题，用react-cli create-react-element-vue-cli·解决跨域的问题,

④构建工具总结：

构建工具它让我们可以不用每次都关心我们的代码在浏览器如何运行，我们只需要首次给构建工具提供一个配置文件(这个配置文件也不是必须的，如果你不给他他会有默认的帮你去处理)，有了这个集成的配置文件以后，我们就可以在下次需要更新的时候调用一次对应的命令就好了，如果我们再结合热更新，我们就更加不需要管任何东西，这就是构建工具去做的东西。

构建工具它让我们不用关心生产的代码，也不用去关心代码如何在浏览器运行，只需要关心我们的开发怎么写的爽怎么写就好了

​2、Vue 组件之间的通信方式

（1）父组件向子组件传值

//App.vue父组件

<template>

<div id="app">

<users :users="users"></users>//前者自定义名称便于子组件调用，后者要传递数据名

</div>

</template>

<script>

import Chilren from "./components/chilren "

export default {

name: 'App',

data(){

return{

users:["HelloKK"]

}

},

components:{

Chilren:Chilren

}

}

//chilren子组件

<template>

<div class="hello">

<ul>

<li v-for="user in users">{{user}}</li>//遍历传递过来的值，然后呈现到页面

</ul>

</div>

</template>

<script>

export default {

name: 'chilren',

props:{

users:{ //这个就是父组件中子标签自定义名字

type:Array,

required:true

}

}

}

</script>

（2）子组件向父组件传值

// 子组件

<template>

<header>

<h1 @click="changeTitle">{{title}}</h1>//绑定一个点击事件

</header>

</template>

<script>

export default {

name: 'app-header',

data() {

return {

title:"Vue.js Demo"

}

},

methods:{

changeTitle() {

this.$emit("titleChanged","子向父组件传值");//自定义事件 传递值“子向父组件传值”

}

}

}

</script>

// 父组件

<template>

<div id="app">

<app-header v-on:titleChanged="updateTitle" ></app-header>

//与子组件titleChanged自定义事件保持一致

// updateTitle($event)接受传递过来的文字

<h2>{{title}}</h2>

</div>

</template>

<script>

import Header from "./components/Header"

export default {

name: 'App',

data(){

return{

title:"传递的是一个值"

}

},

methods:{

updateTitle(e){ //声明这个函数

this.title = e;

}

},

components:{

"app-header":Header,

}

}

</script>

​3、Vuex的理解及使用场景

Vuex 是一个专为 Vue 应用程序开发的状态管理模式。每一个 Vuex 应用的核心就是 store（仓库）。

Vuex 的状态存储是响应式的；当 Vue 组件从 store 中读取状态的时候，若 store 中的状态发生变化，那么相应的组件也会相应地得到高效更新 2. 改变 store 中的状态的唯一途径就是显式地提交 (commit) mutation， 这样使得我们可以方便地跟踪每一个状态的变化 Vuex主要包括以下几个核心模块：

1.State：定义了应用的状态数据

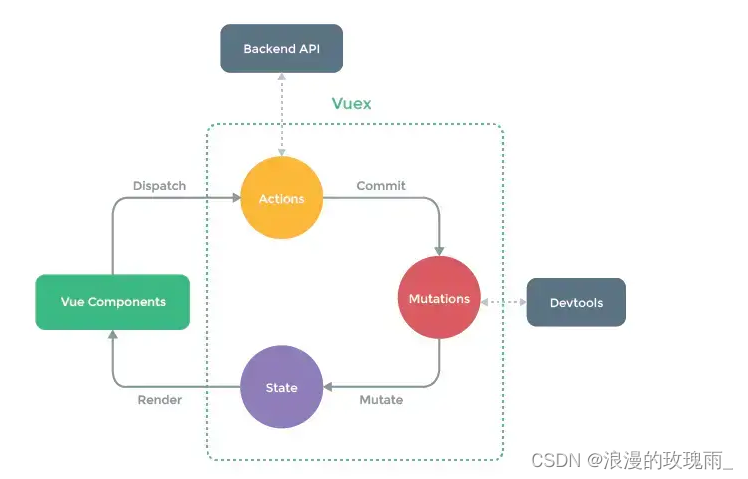
2.Getter：在 store 中定义“getter”（可以认为是 store 的计算属性），

就像计算属性一样，getter 的返回值会根据它的依赖被缓存起来， 且只有当它的依赖值发生了改变才会被重新计算

3. Mutation：是唯一更改 store 中状态的方法，且必须是同步函数

4. Action：用于提交 mutation，而不是直接变更状态，可以包含任意异步操作

5. Module：允许将单一的 Store 拆分为多个 store 且同时保存在单一的状态树中



4、vue 的生命周期 八个阶段

1.beforeCreate

在实例创建之间执行，数据是未加载状态。

创建一个Vue实例，此时实例上只有一些生命周期函数和默认的事件

此时data computed watch methods上的方法和数据均不能访问

2.created

在实例创建、数据加载后，能初始化数据，DOM渲染之前执行。

可以对data数据进行操作，可进行一些请求，请求不易过多，避免白屏时间太长。

3.beforeMount

虚拟DOM已创建完成，在数据渲染前最后一次更改数据。el未挂载。

判断el的挂载方式

判断是否有template设置

将template进行渲染保存到内存当中，还未挂载在页面上

4.mounted

页面、数据渲染完成。el挂载完毕。可以访问DOM节点。

将内存中的模版挂载到页面上

此时可以操作页面上的DOM节点

此时组件从创建阶段进入运行阶段

5.beforeUpdate

重新渲染之前触发。不会造成重渲染。

页面显示的数据是旧的，此时data里面的数据是最新，页面数据和data数据暂未同步6.updated

数据已经更新完成，DOM也重新render完成，更改数据会陷入死循环。

根据data里的最新数据渲染出最新的DOM树，然后将最新的DOM挂载到页面

此时data和页面数据一致，都是最新的

7.beforeDestroy

实例销毁前执行，实例仍然完全可用。

此时组件从运行阶段进入到销毁阶段

组件上的data和methods以及过滤器等都出于可用状态，销毁还未执行

8.Destroyed

实例销毁后执行，这时候只剩下DOM空壳。

组件已经被完全销毁，组件中所有的数据、方法、指令、过滤器等，都已不可用

5、简述Vue每个周期具体适合哪些场景？

beforeCreate： 在new一个vue实例后，只有一些默认的生命周期钩子和默认事件，其他的东西都还没创建。在beforeCreate生命周期执行的时候，data和methods中的数据都还没有初始化。不能在这个阶段使用data中的数据和methods中的方法

created： data 和 methods都已经被初始化好了，如果要调用 methods 中的方法，或者操作 data 中的数据，最早可以在这个阶段中操作

beforeMount： 执行到这个钩子的时候，在内存中已经编译好了模板了，但是还没有挂载到页面中，此时，页面还是旧的

mounted： 执行到这个钩子的时候，就表示Vue实例已经初始化完成了。此时组件脱离了创建阶段，进入到了运行阶段。如果我们想要通过插件操作页面上的DOM节点，最早可以在和这个阶段中进行

beforeUpdate： 当执行这个钩子时，页面中的显示的数据还是旧的，data中的数据是更新后的， 页面还没有和最新的数据保持同步

updated： 页面显示的数据和data中的数据已经保持同步了，都是最新的

beforeDestory： Vue实例从运行阶段进入到了销毁阶段，这个时候上所有的 data 和 methods ， 指令， 过滤器 ……都是处于可用状态。还没有真正被销毁

destroyed： 这个时候上所有的 data 和 methods ， 指令， 过滤器 ……都是处于不可用状态。组件已经被销毁了。

6、简述MVVM 和MVC的原理以及区别？

MVVM视图模型双向绑定，是Model-View-ViewModel的缩写

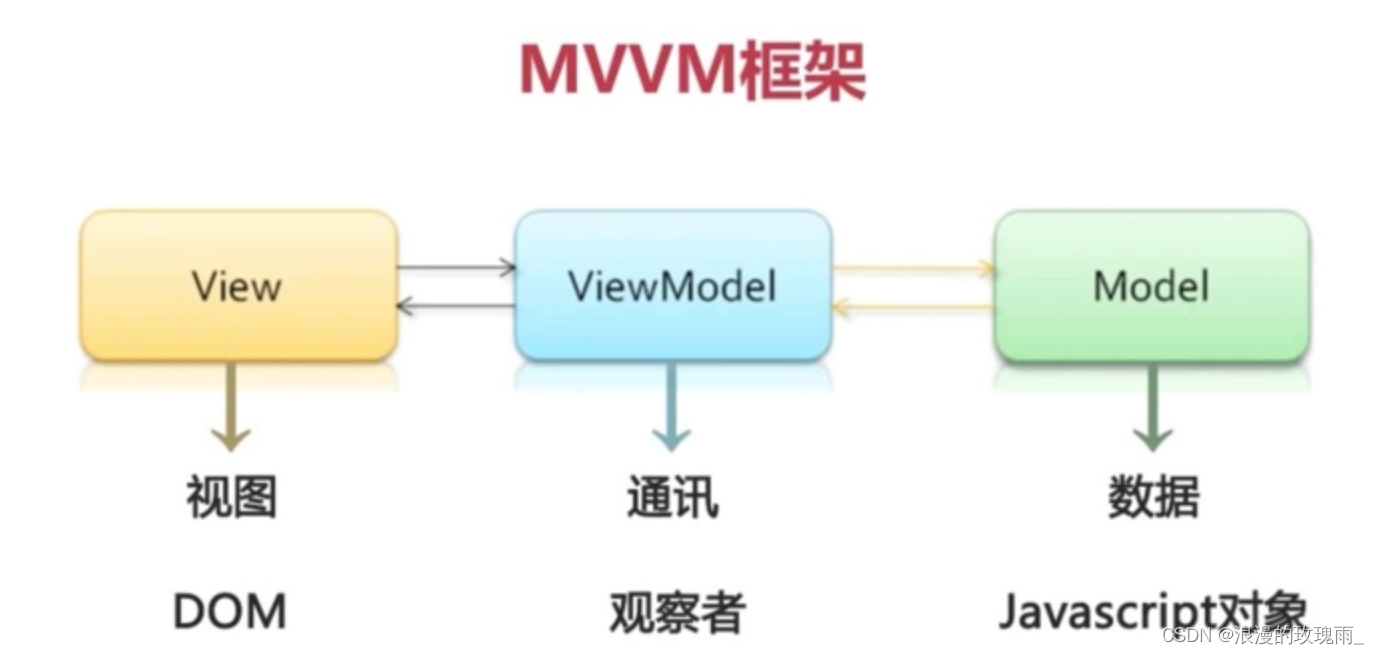
1、MVVM的优点：

低耦合。视图（View）可以独立于Model变化和修改，一个Model可以绑定到不同的View上，当View变化的时候Model可以不变化，当Model变化的时候View也可以不变；

可重用性。你可以把一些视图逻辑放在一个Model里面，让很多View重用这段视图逻辑。

独立开发。开发人员可以专注于业务逻辑和数据的开发(ViewModel)，设计人员可以专注于页面设计。

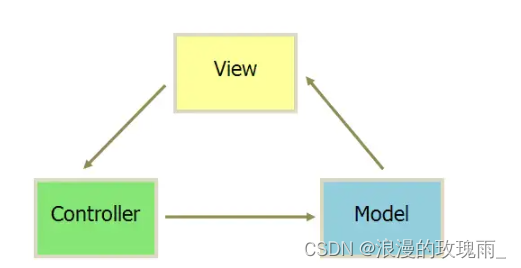
可测试。





2、什么是MVC?

MVC是应用最广泛的软件架构之一,一般MVC分为:Model(模型),View(视图),Controller(控制器)。 这主要是基于分层的目的,让彼此的职责分开.View一般用过Controller来和Model进行联系。Controller是Model和View的协调者,View和Model不直接联系。基本都是单向联系。M和V指的意思和MVVM中的M和V意思一样。C即Controller指的是页面业务逻辑。MVC是单向通信。也就是View跟Model，必须通过Controller来承上启下。



Model（模型）表示应用程序核心（如数据库）。

View（视图）显示效果（HTML页面）。

Controller（控制器）处理输入（业务逻辑）。

MVC 模式同时提供了对 HTML、CSS 和 JavaScript 的完全控制。

Model（模型）是应用程序中用于处理应用程序数据逻辑的部分。（通常模型对象负责在数据库中存取数据)

View（视图）是应用程序中处理数据显示的部分。(通常视图是依据模型数据创建的)

Controller（控制器）是应用程序中处理用户交互的部分。(通常控制器负责从视图读取数据，控制用户输入，并向模型发送数据）

优点:

1、低耦合

2、重用性高

3、生命周期成本低

4、部署快

5、可维护性高

6、有利软件工程化管理

3、MVC与MVVM的区别？

MVC和MVVM的区别并不是VM完全取代了C，ViewModel存在目的在于抽离Controller中展示的业务逻辑，而不是替代Controller，其它视图操作业务等还是应该放在Controller中实现。也就是说MVVM实现的是业务逻辑组件的重用。

MVC中Controller演变成MVVM中的ViewModel

MVVM通过数据来显示视图层而不是节点操作

MVVM主要解决了MVC中大量的dom操作使页面渲染性能降低,加载速度变慢,影响用户体验

7、vue常见指令

v-model 多用于表单元素实现双向数据绑定

v-bind：简写为冒号：“：”，动态绑定一些元素的属性，类型可以是：字符串、对象或数组。

v-on:click 给标签绑定函数，可以缩写为：“@”，例如绑定一个点击函数 函数必须写在methods里面

v-for 格式： v-for="字段名 in(of) 数组json" 循环数组或json，记得加上key

v-show 显示内容

v-if 指令：取值为true/false，控制元素是否需要被渲染

v-else 指令：和v-if指令搭配使用，没有对应的值。当v-if的值false，v-else才会被渲染出来

v-else-if 必须和v-if连用

v-text 解析文本

v-html 解析html标签 （一般常见的解决后台的富文本内容）

v-bind:class 三种绑定方法

对象型 "{red：isred}"

三元型 " isred？"red"："blue"

数组型 " [{red："isred" }，{blue："isblue"}] "

v-once 进入页面时 只渲染一次 不在进行渲染

v-cloak 防止闪烁

v-pre 把标签内部的元素原位输出

8、vue中的data为什么是一个函数？起到什么作用？

在Vue组件中，data选项必须是一个函数，而不能直接是一个对象。这是因为Vue组件可以同时存在多个实例，如果直接使用对象形式的data选项，那么所有的实例将会共享同一个data对象，这样就会造成数据互相干扰的问题。

因此，将data选项设置为函数可以让每个实例都拥有自己独立的data对象。当组件被创建多次时，每个实例都会调用该函数并返回一个新的data对象，从而保证了数据的隔离性。

另外，data选项作为一个函数还具有一个重要的特性，就是它可以接收一个参数，这个参数是组件实例本身。这个特性在一些场景下非常有用，例如在定义组件时需要使用组件实例的一些属性或方法来计算初始数据。

因此，为了避免数据共享和保证数据隔离性，以及方便使用组件实例的属性和方法，Vue组件中的data选项必须是一个函数。

以下是一个简单的Vue组件示例，其中data被定义为一个函数：

<template>

<div>

<p>{{ message }}</p>

<button @click="increment">{{ count }}</button>

</div>

</template>

<script>

export default {

data() {

return {

message: 'Hello, Vue!',

count: 0

}

},

methods: {

increment() {

this.count++

}

}

}

</script>

在这个例子中，data函数返回一个包含message和count两个属性的对象。每次创建组件实例时，Vue都会调用该函数返回一个新的数据对象，确保每个组件实例都有它自己的数据对象。

9、vue中ref的作用？

1、获取DOM元素的引用。

ref 加在普通的元素上，用this.ref.name 获取到的是dom元素

vue给我们提供一个操作dom的属性，ref绑定在dom元素上时，用起来与id差不多，通过this.$refs来调用

<template>

<div id="app">

<div ref="testDom">11111</div>

<button @click="getTest">获取test节点</button>

</div>

</template>

<script>

export default {

methods: {

getTest() {

console.log(this.$refs.testDom)

}

}

};

</script>

2、获取子组件的引用。

在Vue组件中使用ref可以获取子组件的引用，从而可以在父组件内部调用子组件的方法或访问其数据。

在父组件中将子组件引入，并在子组件标签上添加ref属性，然后就可以通过this.$refs.myChild获取子组件的引用，在父组件内部调用子组件的sayHello方法。

<template>

<div>

<child ref="myChild"></child>

</div>

</template>

<script>

import Child from './Child.vue';

export default {

components: { Child },

mounted() {

// 使用this.$refs获取子组件的引用

this.$refs.myChild.sayHello();

}

}

</script>

3、利用 v-for 和 ref 获取一组数组或者dom 节点

注意事项：

ref 需要在dom渲染完成后应用，在使用时确保dom已经渲染完成。比如在生命周期 mounted(){} 钩子中调用，或者在 this.$nextTick(()=>{}) 中调用。

如果ref 是循环出来的，有多个重名，那么ref值会是一个数组 ，此时要拿到单个ref 只需要循环就可以。

10、vue中hash和history的区别 ？

Vue-router的路由分为hash和history模式

1、hash方式

hash方式是指url中存在 # 的一种方式，是vueRouter的默认模式，

当#后面的url地址发生变化时，浏览器不会向服务器发送请求，故不会刷新页面

当#后面的url地址发生变化时，会触发hashChange（hash模式得核心实现原理）事件，从而，我们可以通过监听hashChange事件来知道路由发生变化，从而进一步去更新我们的页面

只可修改hash部分，

当浏览器刷新时，浏览器只会向服务器去请求# 前面的域名服务器下根目录下的index.html文件

hash模式会创建hashHistory对象,hashHistory对象有两个方法，push() 和 replace()

HashHistory.push()会将新的路由添加到浏览器访问的历史的栈顶,而HasHistory.replace()会替换到当前栈顶的路由

2、history模式

history模式得路由和域名之间直接通过/连接，无#符分隔，就是普通url形式

history模式当发生路由跳转时，通过HTML5的history.pushState()方法或者history.replaceState() 方法改变地址栏地址，并将地址的改变记录到浏览器访问栈中。（这里有一点需要注意，它只改变了浏览器地址栏中的地址，但并不会像服务器去发送请求）

当浏览器前进，后台，或者调用back(),forward(), go()等方法时，会触发popstate事件。故，我们可以通过监听popstate事件来获取最新的路由地址，从而更新页面

通过pushstate() 修改的url可以是与当前url同源的任意url。

需要和服务器配合使用，否则容易出现页面404的情况

总结如下：

hash模式带#号比较丑，history模式比较优雅；

pushState设置的新的URL可以是与当前URL同源的任意URL；而hash只可修改#后面的部分，故只可设置与当前同文档的URL；

pushState设置的新URL可以与当前URL一模一样，这样也会把记录添加到栈中；而hash设置的新值必须与原来不一样才会触发记录添加到栈中；

pushState通过stateObject可以添加任意类型的数据到记录中；而hash只可添加短字符串；

pushState可额外设置title属性供后续使用；

hash兼容IE8以上，history兼容IE10以上；

history模式需要后端配合将所有访问都指向index.html，否则用户刷新页面，会导致404错误。

六 、Vue3篇

1、Vue2.0和Vue3.0的区别？

前言

vue经历从2.0到3.0更新之后，简⽽⾔之就是变得更轻，更快，使⽤起来更加⽅便，每⼀次的版本迭代都是对上⼀个版本的升级优化，不管 是对于我们开发者还是对于⽤户体验都是不断地在越来越⽅便，接下来我会着重于开发者来说⼀下两个不同版本的区别

详解

Vue2和vue3的初始化就存在着⼀定区别，⽐如vue3.0可以在安装脚⼿架同时提前安装好⼀些项⽬开发必备的插件，并且3.0提供了可视化创建脚⼿架，可以更加⽅便的对插件和依赖进⾏管理和配置，同时两个版本的⽬录结构也是有些许差别的。（比如：Vue3相对于Vue2，打包工具Vite替代了webpack；TS替代了JS，pinia替代了vuex；Element-plus替代了Element等等）

在开发过程中两个版本的使⽤⽅法虽然在表⾯上没有太⼤的⼀个区别，但是在他的底层⽅⾯去看的话区别还是很⼤的，其中就包括渲染⽅式，数据监听，双向绑定，⽣命周期，vue3更精准变更通知，这⾥着重说⼀下关于双向绑定的更新，

vue2 的双向数据绑定是利⽤ES5的⼀个 API ，Object.definePropert()对数据进⾏劫持 结合发布订阅模式的⽅式来实现的。

vue3 中使⽤了 ES6 的 ProxyAPI 对数据代理，通过 reactive() 函数给每⼀个对象都包⼀层 Proxy，通过 Proxy 监听属性的变化，从⽽实现对数据的监控。

2、Vue3带来了什么改变？

性能的提升

打包大小减少41%

初次渲染快55%, 更新渲染快133%

内存减少54%

源码的升级

使用Proxy代替defineProperty实现响应式

重写虚拟DOM的实现和Tree-Shaking

拥抱TypeScript

Vue3可以更好的支持TypeScript

新的特性

Composition API（组合API）

（1）setup配置

（2）ref与reactive

（3）watch与watchEffect

（4）provide与inject

新的内置组件

（1）Fragment

（2）Teleport

（3）Suspense

其他改变

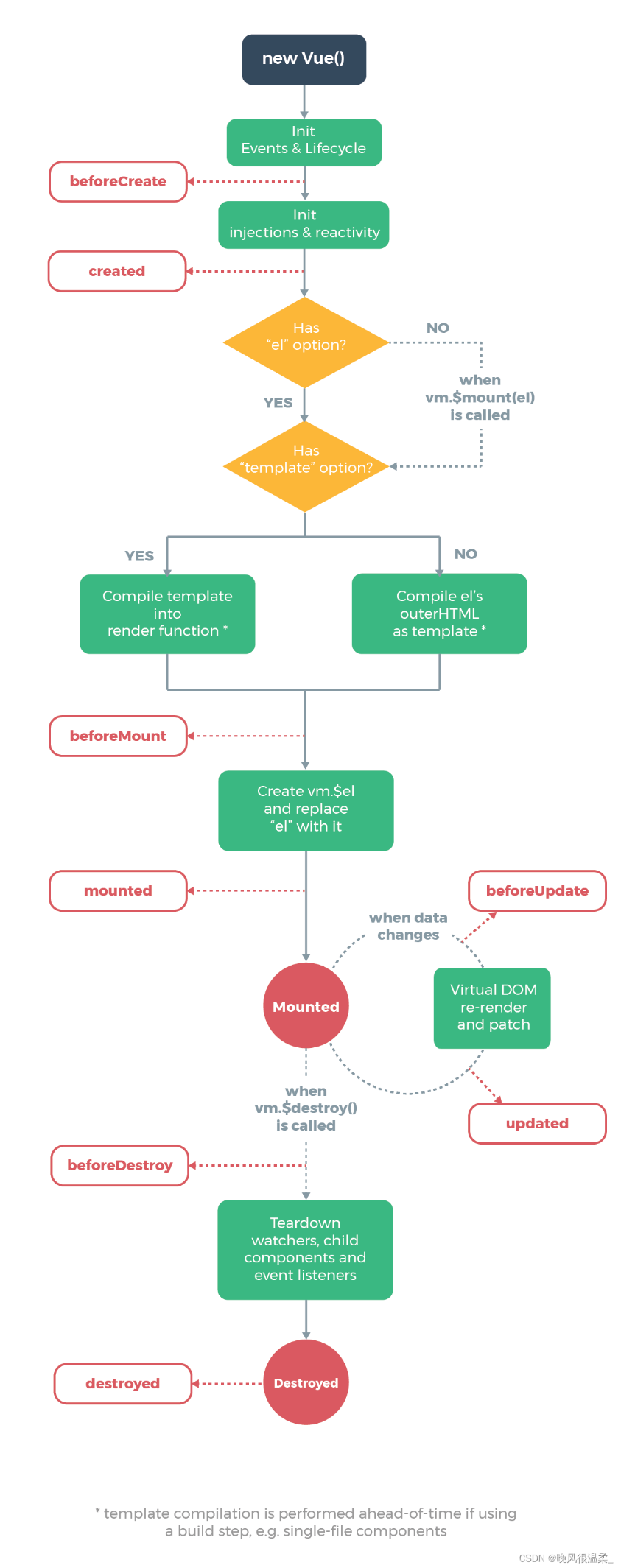
（1）新的生命周期钩子

（2）data 选项应始终被声明为一个函数

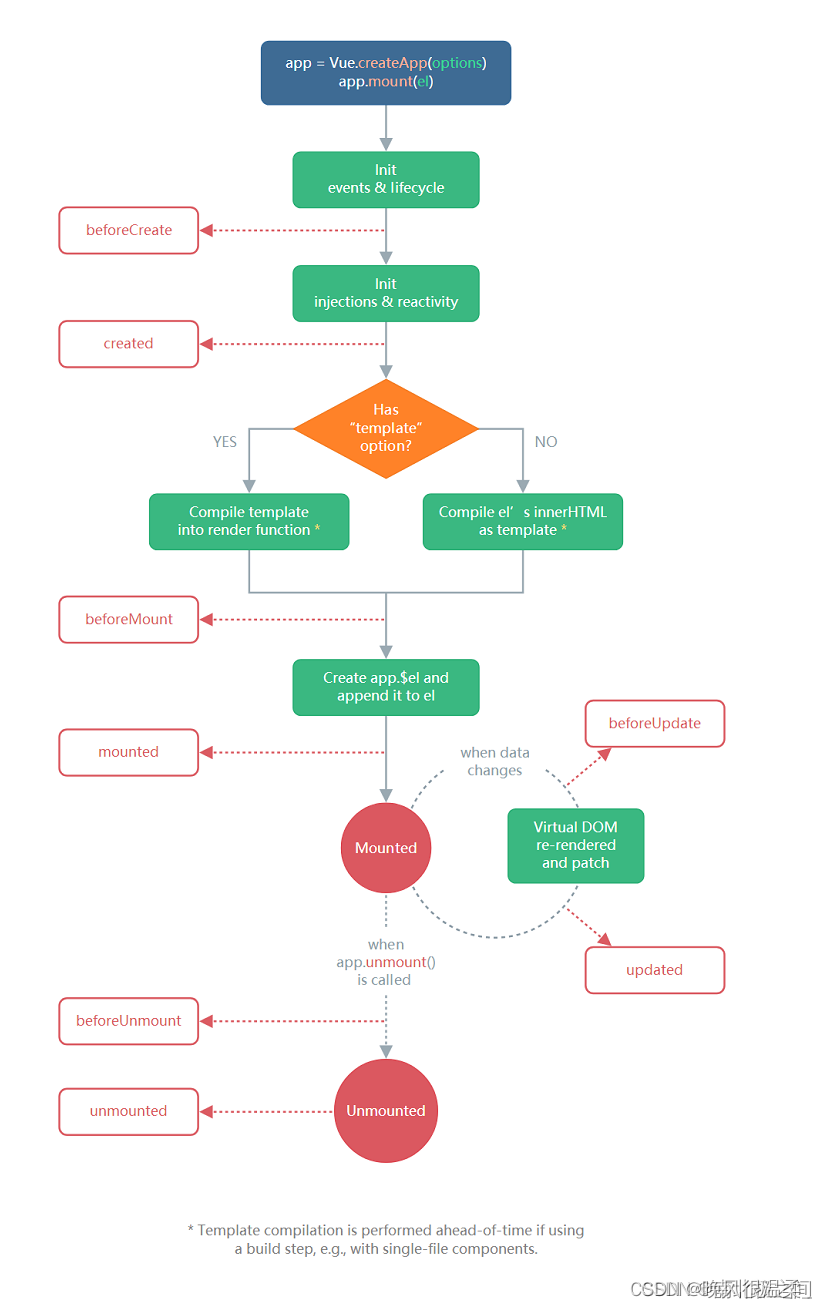
（3）移除keyCode支持作为 v-on 的修饰符

3、生命周期（vue2和vue3的生命周期对比）有哪些？

vue2.x的生命周期



vue3.0的生命周期



Vue3.0中可以继续使用Vue2.x中的生命周期钩子，但有有两个被更名：

beforeDestroy改名为 beforeUnmount

destroyed改名为 unmounted

Vue3.0也提供了 Composition API 形式的生命周期钩子，与Vue2.x中钩子对应关系如下：

beforeCreate===>setup()

created=======>setup()

beforeMount ===>onBeforeMount

mounted=======>onMounted

beforeUpdate===>onBeforeUpdate

updated =======>onUpdated

beforeUnmount ==>onBeforeUnmount

unmounted =====>onUnmounted

4、Vue3.0中的响应式原理是什么？vue2的响应式原理是什么？

vue2.x的响应式

实现原理：

对象类型：通过Object.defineProperty()对属性的读取、修改进行拦截（数据劫持）。

数组类型：通过重写更新数组的一系列方法来实现拦截。（对数组的变更方法进行了包裹）。

Object.defineProperty(data, 'count', {

get () {},

set() {}

})

存在问题：

新增属性、删除属性, 界面不会更新。

直接通过下标修改数组, 界面不会自动更新。

Vue3.0的响应式

实现原理:

通过Proxy（代理）: 拦截对象中任意属性的变化, 包括：属性值的读写、属性的添加、属性的删除等。

通过Reflect（反射）: 对源对象的属性进行操作。

MDN文档中描述的Proxy与Reflect：

Proxy：Proxy - JavaScript | MDN

Reflect：Reflect - JavaScript | MDN

new Proxy(data, {

// 拦截读取属性值

get (target, prop) {

return Reflect.get(target, prop)

},

// 拦截设置属性值或添加新属性

set (target, prop, value) {

return Reflect.set(target, prop, value)

},

// 拦截删除属性

deleteProperty (target, prop) {

return Reflect.deleteProperty(target, prop)

}

})

proxy.name = 'tom'

vue3响应式数据的判断

isRef: 检查一个值是否为一个 ref 对象

isReactive: 检查一个对象是否是由 reactive 创建的响应式代理

isReadonly: 检查一个对象是否是由 readonly 创建的只读代理

isProxy: 检查一个对象是否是由 reactive 或者 readonly 方法创建的代理

5、vue3的常用 Composition API有哪些？

1.拉开序幕的setup

理解：Vue3.0中一个新的配置项，值为一个函数。

setup是所有Composition API（组合API）“ 表演的舞台 ”。

组件中所用到的：数据、方法等等，均要配置在setup中。

setup函数的两种返回值：

若返回一个对象，则对象中的属性、方法, 在模板中均可以直接使用。（重关注！）

若返回一个渲染函数：则可以自定义渲染内容。（了解）

5.setup的几个注意点

setup执行的时机

在beforeCreate之前执行一次，this是undefined。

setup的参数

props：值为对象，包含：组件外部传递过来，且组件内部声明接收了的属性。

context：上下文对象

attrs: 值为对象，包含：组件外部传递过来，但没有在props配置中声明的属性, 相当于 this.$attrs。

slots: 收到的插槽内容, 相当于 this.$slots。

emit: 分发自定义事件的函数, 相当于 this.$emit。

尽量不要与Vue2.x配置混用

Vue2.x配置（data、methos、computed...）中可以访问到setup中的属性、方法。

但在setup中不能访问到Vue2.x配置（data、methos、computed...）。

如果有重名, setup优先。

setup不能是一个async函数，因为返回值不再是return的对象, 而是promise, 模板看不到return对象中的属性。（后期也可以返回一个Promise实例，但需要Suspense和异步组件的配合）

6、Vue3中的ref函数

作用: 定义一个响应式的数据

语法: const xxx = ref(initValue)

创建一个包含响应式数据的引用对象（reference对象，简称ref对象）。

JS中操作数据： xxx.value

模板中读取数据: 不需要.value，直接：<div>{{xxx}}</div>

备注：

接收的数据可以是：基本类型、也可以是对象类型。

基本类型的数据：响应式依然是靠Object.defineProperty()的get与set完成的。

对象类型的数据：内部 “ 求助 ” 了Vue3.0中的一个新函数—— reactive函数。

Vue2与vue3的区别

1.双向数据绑定原理不同

Vue2 的双向数据绑定是利用ES5的一个APIObject.definePropert() 对数据进行劫持，结合发布订阅模式的方式来实现的。

Vue3 中使用ES6的Proxy API对数据代理。

Vue3 使用数据代理的优势有以下几点：1）definePropert 只能监听某个属性，不能对整个对象进行监听 2）可以省去for in，闭包等内容来提升效率（直接绑定整个对象即可）3）可以监听数组，不用再单独的对数组做特异性操作，Vue3可以检测到数组内部数据的变化

2.是否支持碎片

Vue2 不支持碎片。Vue3 支持碎片，就是说可以拥有多个根节点

3.API 类型不同

Vue2 使用选项类型api,选项型api 在代码里分割了不同的属性：data,computed,method等。

Vue3 使用合成型api,新的合成型api 能让我们使用方法来分割，相比于旧的api 使用属性来分组，这样代码会更加简便和整洁。

4定义数据变量和方法不同

Vue2是把数据放到了data 中，在 Vue2中 定义数据变量是data(){},创建的方法要在method:{}

Vue3 就需要使用一个新的setup()方法，此方法在组件初始化构造的时候触发。使用以下三个步骤来建立反应性数据：1）从vue 引入 reactive；2）使用 reactive ()方法来声明数据为响应性数据；3） 使用setup()方法来返回我们的响应性数据，从而template 可以获取这些响应性数据。

5.生命周期钩子函数不同

Vue2 中的生命周期：beforeCreate 组件创建之前；created 组建创建之后；beforeMount 组件挂载到页面之前执行；Mounted 组件挂载到页面之后执行，beforeUpdate 组件更新之前；updated组件更新之后

Vue3 中的生命周期：setup 开始创建组件；onBeforeMount 组件挂载到页面之前执行；onMounted 组件挂载到页面之后执行；onBeforeUpdate 组件更新之前；onUpdated 组件更新之后；

而且 Vue3 生命周期在调用前需要先进行引入。除了这些钩子函数外，Vue3 还增加了 onRenderTracked 和onRenderTriggered 函数。

6.父子传参不同

Vue2 父传子，用props ；子传父用事件Emitting Events。在Vue2 中，会调用this$emit 然后传入事件名和对象。

Vue3 父传子，用props;子传父用Emitting Events 。在Vue3 中的setup()中的第一参数content 对象中就有 emit,那么我们只要在setup()接收第二个参数中使用分解对象法取出emit 就可以在setup 方法中随意使用了。

7.指令与插槽不同

Vue2 中使用slot 可以直接使用slot ;v-for 与v-if 在Vue2中优先级高的是v-for 指令，而且不建议一起使用。

Vue3 中必须是使用v-slot的形式；vue 3中v-for 与v-if ，只会把当前v-if 当作v-for 的一个判断语句，不会相互冲突；

Vue3 中移除keyCode 作为v-on 的修饰符，当然也不支持config.keyCodes，取而代之的是使用键名来作为事件的修饰符来使用，于是Vue.config.keyCodes 也被弃用了-vue2我们在监听按键事件的时候，是可以通过在事件后面加上按键码来实现监听某一个按键的

Vue3 中移除v-on.native 修饰符-这个修饰符的作用是：如果想要在一个组件的根元素上直接监听一个原生事件，需要使用v-on的.native修饰符Vue3中移除掉之后，取而代之的是，在新增的emits 选项中定义当前组件真正触发的事件（即，组件事件）。此外，Vue 现在将所有未在组件 emits 选项中定义的事件作为原生事件添加到子组件的根元素中（除非子组件选项中设置了inheritAttrs:false）

Vue3 中移除过滤器filter，官方推荐，如果要使用过滤器可以使用 computed 计算属性和method 函数来代替。

8.Main.js 文件不同

Vue2中我们可以使用pototype(原型)的形式去进行操作，引入的是构造函数。

Vue3 中需要使用结构的形式进行操作，引入的是工厂函数；

Vue3中app组件中可以没有根标签。

Setup() 函数特性

setup() 函数接收两个参数：props、context（包含attrs、slots、emit）

setup()函数处于生命周期 beforeCreated 两个钩子函数之前

执行setup时，组件实例尚未被创建（在setup（）内部，this不会是该活跃实例的引用，即不指向vue 实例，Vue 为了避免我们错误的使用，直接将setup 函数中的this 修改成了undefined）

与模版一起使用时，需要返回一个对象

因为setup函数中，props时响应式得，当传入新的prop时，它就会被更新，所以不能使用es6解构，因为它会消除 prop得响应性，如果需要解构prop,可以通过使用setup函数中torefs 来完成此操作。

在setup（）内使用响应式数据时，需要通过.value 获取

从setup（）中返回得对象上得property 返回并且可以在模版中被访问时，它将自动展开为内部值。不需要在模版中追加.value.

Setup 函数只能是同步的不能是异步的。

七、React篇

1、讲讲什么是 JSX ？

当 Facebook 第一次发布 React 时，他们还引入了一种新的 JS 方言 JSX，将原始 HTML 模板嵌入到 JS 代码中。JSX 代码本身不能被浏览器读取，必须使用Babel和webpack等工具将其转换为传统的JS。很多开发人员就能无意识使用 JSX，因为它已经与 React 结合在一直了。

class MyComponent extends React.Component {

render() {

let props = this.props;

return (

<div className="my-component">

<a href={props.url}>{props.name}</a>

</div>

);

}

}

JSX的特点

可以将HTML语言直接写在JavaScript语言之中，不加任何引号，这就是JSX的语法，它允许HTML与JavaScript的混写。

JSX允许直接在模板插入JavaScript变量。如果这个变量是一个数组，则会展开这个数组的所有成员。

防注入攻击

在JSX中嵌入用户输入是安全的；

React DOM在渲染之前默认会过滤所有传入的值。它可以确保应用不会被注入攻击。所有的内容在渲染之前都被转换成了字符串。这样可以有效地防止XSS(跨站脚本攻击)

Babel转译器会把JSX转换成一个名为React.createElement()的方法调用。

如果在普通的html里面要写jsx语法，要将script的type改成text/jsx，这是因为React独有的JSX语法跟JavaScript不兼容。凡是使用JSX的地方，都要加上type=“text/jsx”。其次，React提供俩个库:react.js和JSXTransformer.js，它们必须首先加载。其中，JSXTransformer.js的作用是将JSX语法转为JavaScript语法。这一步很消耗时间，实际上线的时候，应该把它放到服务器完成。

2、React 的生命周期方法有哪些？

React的生命周期方法有很多，并且在不同的React版本中可能会有一些变化。为了更好地理解React的生命周期，我将它们按照React 16版本及其之后的版本进行整理，以帮助你更好地理解。

在React 16版本及其之后，React的生命周期方法可分为三个阶段：挂载阶段、更新阶段和卸载阶段。以下是React 16版本及其之后的生命周期方法列表：

挂载阶段：

constructor：组件实例化时调用，用于初始化状态和绑定方法。

static getDerivedStateFromProps：在渲染之前调用，用于根据新的属性值计算并返回一个新的状态。

render：渲染组件的内容。

componentDidMount：组件挂载后调用，可以进行异步操作、订阅事件等。

更新阶段：

static getDerivedStateFromProps：在渲染之前调用，用于根据新的属性值计算并返回一个新的状态。

shouldComponentUpdate：在渲染之前调用，用于决定是否重新渲染组件，默认返回true。

render：渲染组件的内容。

getSnapshotBeforeUpdate：在最终将内容渲染到DOM之前调用，用于获取DOM更新前的快照。

componentDidUpdate：组件更新后调用，可以进行DOM操作、发起网络请求等。

卸载阶段：

componentWillUnmount：组件卸载前调用，可以进行清理操作，如取消订阅、清除定时器等。

另外，React 16.3版本后引入了以下生命周期方法：

static getDerivedStateFromError：在子组件渲染过程中，如果发生错误，会调用该方法，返回一个新的状态。

componentDidCatch：在子组件渲染过程中，如果发生错误，会调用该方法，用于记录错误信息或上报错误。

需要注意的是，React 17版本之后，一些生命周期方法被标记为过时，并推荐使用其他替代方法来实现相应的功能。在使用React时，可以根据具体的需求和React版本来选择合适的生命周期方法。同时，React还提供了钩子函数的方式（如useEffect钩子）来完成与生命周期相关的操作，这也是React 16.8版本及其之后的新特性。

3、React的Hooks详解

Hook是什么

Hook 是一些可以让你在函数组件里“钩入” React state 及生命周期等特性的函数。Hook 不能在 class 组件中使用 —— 这使得你不使用 class 也能使用 React。

React 内置了一些像 useState 这样的 Hook。你也可以创建你自己的 Hook 来复用不同组件之间的状态逻辑。

React提供了多个常用的Hooks，用于在函数组件中管理状态、处理副作用和访问React的上下文等。以下是React所有的常用Hooks列表：

useState：用于在函数组件中添加状态。

useEffect：用于在函数组件中执行副作用操作。

useContext：用于在函数组件中访问React的上下文。

useReducer：用于在函数组件中使用Reducer模式来管理状态。

useRef：用于在函数组件中创建可变的引用。

useMemo：用于在函数组件中缓存计算的值。

useCallback：用于在函数组件中缓存函数。

useLayoutEffect：类似于useEffect，但在DOM更新之后同步执行。

useImperativeHandle：用于在函数组件中自定义外部组件实例的暴露。

useDebugValue：用于在自定义Hooks中显示自定义的调试值。

这些Hooks可以帮助你更方便地编写和管理React函数组件。同时，你还可以根据需要自定义自己的Hooks来封装和复用逻辑。React的Hooks特性在React 16.8版本及其之后引入，它们提供了一种更简洁、灵活的方式来编写可复用的React组件逻辑。

4、React和Vue.js的相似性和差异性是什么？

相似性如下。

（1）都是用于创建UI的 JavaScript库。

（2）都是快速和轻量级的代码库（这里指 React核心库）。

（3）都有基于组件的架构。

（4）都使用虚拟DOM。

（5）都可以放在单独的HTML文件中，或者放在 Webpack设置的一个更复杂的模块中。

（6）都有独立但常用的路由器和状态管理库。

它们最大的区别在于 Vue. js通常使用HTML模板文件，而 React完全使用 JavaScript创建虚拟DOM。 Vue. js还具有对于“可变状态”的“ reactivity”的重新渲染的自动化检测系统。（React一般适用于大型项目）

5、React的功能是什么？

React的主要功能如下：

它使用虚拟DOM而不是真实DOM。

它使用服务器端渲染。

它遵循单向数据流或数据绑定。

6、React的一些主要优点是？

它提高了应用程序的性能

它可以方便地在客户端和服务器端使用

由于有了JSX，代码的可读性提高了

React易于与其他框架（如Meteor，Angular等）集成

使用React，编写UI测试用例变得非常容易

7、React的局限性是什么？

React只是一个库，而不是一个成熟的框架

它的图书馆很大，需要花费一些时间来理解

对于新手程序员而言，理解起来可能有点困难

由于使用内联模板和JSX，编码变得复杂

8、如何理解React State不可变性的原则

在 React 中，不可变性是指数据一旦被创建，就不能被修改。React 推崇使用不可变数据的原则，这意味着在更新数据时，应该创建新的数据对象而不是直接修改现有的数据。

以下是理解 React 中不可变性原则的几个关键点：

数据一旦创建就不能被修改：在 React 中，组件的状态（state）和属性（props）应该被视为不可变的。一旦创建了状态或属性对象，就不应该直接修改它们的值。这样可以确保组件的数据在更新时是不可变的，从而避免意外的数据改变和副作用。

创建新的数据对象：当需要更新状态或属性时，应该创建新的数据对象。这可以通过使用对象展开运算符、数组的 concat()、slice() 等方法，或者使用不可变数据库（如Immutable.js、Immer 等）来创建新的数据副本。

比较数据变化：React 使用 Virtual DOM 来比较前后两个状态树的差异，并仅更新需要更新的部分。通过使用不可变数据，React 可以更高效地进行比较，因为它可以简单地比较对象引用是否相等，而不必逐个比较对象的属性。

性能优化：使用不可变数据可以带来性能上的优势。由于 React 可以更轻松地比较前后状态的差异，可以减少不必要的重新渲染和组件更新，提高应用的性能和响应性。

不可变性的原则在 React 中有以下好处：

简化数据变更追踪：由于数据不可变，可以更轻松地追踪数据的变化。这样可以更好地理解代码的行为和数据的流动。

避免副作用：可变数据容易引发副作用和难以追踪的 bug。通过使用不可变数据，可以避免许多与副作用相关的问题。

方便的历史记录和回滚：不可变数据使得记录和回滚应用状态的历史变得更容易。可以在不改变原始数据的情况下，创建和保存不同时间点的数据快照

9、React的特点和关键版本区别？

react的关键版本

React16之前对virtural dom的更新和渲染是同步的。就是当一次更新或者一次加载开始以后，diff virtual dom并且渲染的过程是一口气完成的。如果组件层级比较深，相应的堆栈也会很深，长时间占用浏览器主线程，一些类似用户输入、鼠标滚动等操作得不到响应。

React16开始用了分片的方式解决上面的问题

就是把一个任务分成很多小片，当分配给这个小片的时间用尽的时候，就检查任务列表中有没有新的、优先级更高的任务，有就做这个新任务，没有就继续做原来的任务。这种方式被叫做异步渲染(Async Rendering)。

React的特点

1. React速度很快

与其他框架相比，React采取了一种独特操作DOM的方式。

它并不直接对DOM进行操作。它引入了一个叫虚拟DOM的概念，安插在JavaScript逻辑和实际的DOM之间。这一概念提高了Web性能。在UI渲染过程中，React通过在虚拟DOM中的微操作来实现对实际DOM的局部更新。

将视图与数据进行单向绑定，绝大部分操作都可以不再直接操作DOM，而是通过改变数据来更新视图，这对于前端是具有里碑意义的。

2. 跨浏览器兼容

虚拟DOM帮助我们解决了跨浏览器问题，它为我们提供了标准化的API，甚至在IE8中都是没问题的。

3. 模块化

为程序编写独立的模块化UI组件，并且它们可以引入其他组件。这等同于提高了代码的可维护性。

4. 单向数据流，让事情一目了然

Flux是一个用于在JavaScript应用中创建单向数据层的架构，它随着React视图库的开发而被Facebook概念化。它只是一个概念，而非特定工具的实现。它可以被其它框架吸纳。例如，Alex Rattray有一个很好的Flux实例，在React中使用了Backbone的集合和模型。

5. 纯粹的JavaScript

现代Web应用程序与传统的Web应用有着不同的工作方式。

例如，视图层的更新需要通过用户交互而不需要请求服务器。因此视图和控制器非常依赖彼此。

许多框架使用Handlebars或Mustache等模板引擎来处理视图层。但React相信视图和控制器应该相互依存在一起而不是使用第三方模板引擎，而且最重要的是，它是纯粹的JavaScript程序。

6. 同构的JavaScript

单页面JS应用程序的最大缺陷在于对搜索引擎的索引有很大限制。React对此有了解决方案。

React可以在服务器上预渲染应用再发送到客户端。它可以从预渲染的静态内容中恢复一样的记录到动态应用程序中。

因为搜索引擎的爬虫程序依赖的是服务端相应而不是JavaScript的执行，预渲染你的应用有助于搜索引擎优化。

7. React与其他框架/库相比兼容性好

比如使用RequireJS来加载和打包，而Browserify和Webpack适用于构建大型应用。它们使得那些艰难的任务不再让人望而生畏。

8. 相比于其他的框架【vue】更灵活

没有那么多的条条框框，提供了一系列基础api自己随意组合。如果要我把他比作积木的话那么react提供的就是小型的积木，vue提供的是大型的积木。这就势必造成了react会比vue更加的灵活。

八、uniapp篇

1、uniapp优缺点

uniapp官网：组件使用的入门教程 | uni-app官网

优点

一套代码可以生成多端

学习成本低,语法是vue的,组件是小程序的

拓展能力强

使用HBuilderX开发,支持vue语法

突破了系统对H5调用原生能力的限制

缺点

问世时间短,很多地方不完善

社区不大

官方对问题的反馈不及时

在Android平台上比微信小程序和iOS差

文件命名受限

2、uniapp目录结构

一个uni-app工程，默认包含如下目录及文件：

┌─uniCloud 云空间目录，阿里云为uniCloud-aliyun,腾讯云为uniCloud-tcb（详见uniCloud）

│─components 符合vue组件规范的uni-app组件目录

│ └─comp-a.vue 可复用的a组件

├─utssdk 存放uts文件

├─pages 业务页面文件存放的目录

│ ├─index

│ │ └─index.vue index页面

│ └─list

│ └─list.vue list页面

├─static 存放应用引用的本地静态资源（如图片、视频等）的目录，注意：静态资源只能存放于此

├─uni\_modules 存放[uni\_module](/uni\_modules)。

├─platforms 存放各平台专用页面的目录，详见

├─nativeplugins App原生语言插件 详见

├─nativeResources App端原生资源目录

│ └─android Android原生资源目录 详见

├─hybrid App端存放本地html文件的目录，详见

├─wxcomponents 存放小程序组件的目录，详见

├─unpackage 非工程代码，一般存放运行或发行的编译结果

├─AndroidManifest.xml Android原生应用清单文件 详见

├─main.js Vue初始化入口文件

├─App.vue 应用配置，用来配置App全局样式以及监听 应用生命周期

├─manifest.json 配置应用名称、appid、logo、版本等打包信息，详见

├─pages.json 配置页面路由、导航条、选项卡等页面类信息，详见

└─uni.scss 这里是uni-app内置的常用样式变量

pages.json

配置文件，全局页面路径配置，应用的状态栏、导航条、标题、窗口背景色设置等

main.js

入口文件，主要作用是初始化vue实例、定义全局组件、使用需要的插件如 vuex，注意uniapp无法使用vue-router，路由须在pages.json中进行配置。如果开发者坚持使用vue-router，可以在插件市场找到转换插件。

App.vue

是uni-app的主组件，所有页面都是在App.vue下进行切换的，是页面入口文件。但App.vue本身不是页面，这里不能编写视图元素。除此之外，应用生命周期仅可在App.vue中监听，在页面监听无效。

pages

页面管理部分用于存放页面或者组件

应用配置 manifest.json

文件是应用的配置文件，用于指定应用的名称、图标、权限等，HBuilderX 创建的工程此文件在根目录，CLI 创建的工程此文件在 src 目录。我们也可以在这里为Vue 为H5设置跨域拦截处理器

package.json

文件用来对 uni-app 进行全局配置，决定页面文件的路径、窗口样式、原生的导航栏、底部的原生tabbar 等。

pages数组中第一项表示应用启动页，参考官网：pages.json 页面路由 | uni-app官网

3、跨端适配—条件编译

开发者按照 uni-app 规范开发即可保证多平台兼容，大部分业务均可直接满足。但每个平台有自己的一些特性，因此会存在一些无法跨平台的情况。因此就有了条件编译这个模式，不仅是js逻辑代码，template和css样式都可以设置成在某个环境中生效，在其他环境不生效。更多详细信息及平台适配请看官方文档

//template

<!-- #ifdef MP-WEIXIN -->

<!-- 只在小程序中生效 -->

<view>我是微信小程序</view>

<!-- #endif -->

<!-- #ifdef APP-PLUS -->

<!-- 只在 app 中生效 -->

<view>我是 app </view>

<!-- #endif -->

//js

// #ifndef H5

// 表示只有 h5 不使用这个 api

uni.createAnimation(OBJECT)

// #endif

//css

/\* #ifdef MP-WEIXIN \*/

/\* 只在小程序中生效 \*/

.header {

color:red

}

/\* #endif \*/

4、uniapp 常用的指令语句

v-for：循环渲染 （注意加：key）

v-if ：控制元素的删除添加

v-show：控制元素的显示隐藏

v-model：双向数据绑定

v-on：事件绑定（简写@）

v-bind：属性绑定（简写：）

5、uniapp 应用的生命周期、页面的生命周期、组件的生命周期

应用的生命周期

onLaunch——当uni-app 初始化完成时触发（全局只触发一次）

onShow——当 uni-app 启动，或从后台进入前台显示

onHide——当 uni-app 从前台进入后台

onError——当 uni-app 报错时触发

onUniNViewMessage——对 nvue 页面发送的数据进行监听，可参考 nvue 向 vue 通讯

onUnhandledRejection——对未处理的 Promise 拒绝事件监听函数（2.8.1+）

onPageNotFound——页面不存在监听函数

onThemeChange——监听系统主题变化

页面的生命周期

onInit——监听页面初始化，其参数同 onLoad 参数，为上个页面传递的数据，参数类型为 Object（用于页面传参），触发时机早于 onLoad

onLoad——监听页面加载，其参数为上个页面传递的数据，参数类型为 Object（用于页面传参），参考示例

onShow——监听页面显示。页面每次出现在屏幕上都触发，包括从下级页面点返回露出当前页面

onReady——监听页面初次渲染完成。注意如果渲染速度快，会在页面进入动画完成前触发

onHide——监听页面隐藏

onUnload——监听页面卸载

onResize——监听窗口尺寸变化

组件的生命周期

uni-app 组件支持的生命周期，与vue标准组件的生命周期相同

beforeCreate——在实例初始化之后被调用。

created——在实例创建完成后被立即调用。

beforeMount——在挂载开始之前被调用。

mounted——挂载到实例上去之后调用。详见 注意：此处并不能确定子组件被全部挂载，如果需要子组件完全挂载之后在执行操作可以使用$nextTickVue官方文档

beforeUpdate——数据更新时调用，发生在虚拟 DOM 打补丁之前。

updated——由于数据更改导致的虚拟 DOM 重新渲染和打补丁，在这之后会调用该钩子。

beforeDestroy——实例销毁之前调用。在这一步，实例仍然完全可用。

destroyed——Vue 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。

九、Webpack篇

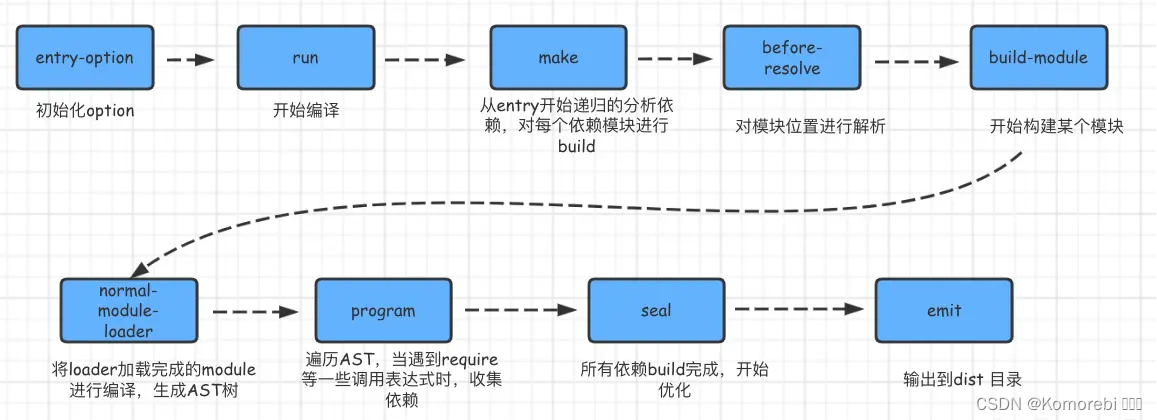
1、Webpack是什么？

webpack 是一个静态模块打包器，当 webpack 处理应用程序时，会递归构建一个依赖关系图，其中包含应用程序需要的每个模块，然后将这些模块打包成一个或多个 bundle。

webpack 就像一条生产线,要经过一系列处理流程(loader)后才能将源文件转换成输出结果。 这条生产线上的每个处理流程的职责都是单一的,多个流程之间有存在依赖关系,只有完成当前处理后才能交给下一个流程去处理。

插件就像是一个插入到生产线中的一个功能,在特定的时机对生产线上的资源做处理。 webpack 在运行过程中会广播事件,插件只需要监听它所关心的事件,就能加入到这条生产线中,去改变生产线的运作。

2、Webpack的打包过程/打包原理/构建流程？



webpack 的运行流程是一个串行的过程，它的工作流程就是将各个插件串联起来。命令行执行npx webpack打包命令开始

1.初始化编译参数:从配置文件和shell命令中读取与合并参数

2.开始编译:根据上一步得到的参数初始化Compiler对象，加载所有配置的Plugin，执行对象的 run 方法开始执行编译。

3.确定入口:根据配置中的 entry 找出所有的入口文件

4.编译模块:从入口文件触发，调用所有配置的Loader对模块进行翻译，再找出该模块依赖的模块，然后递归本步骤直到所有入口依赖的文件都进行翻译。

5.完成模块编译:在经过第4步使用 Loader 翻译完所有模块后，得到了每个模块被翻译后的最终内容以及它们之间的依赖关系图。

6.输出资源：根据依赖关系图，组装成一个个包含多个模块的Chunk，再把每个Chunk转化成一个单独的文件加入到输出列表，根据配置确定输出的路径和文件名输出。

在以上过程中，Webpack 会在特定的时间点广播出特定的事件，插件在监听到感兴趣的事件后会执行特定的逻辑。

总结归纳：

初始化:从配置文件和shell命令中读取与合并参数，根据参数初始化Compiler实例，加载Plugin(注册所有配置的插件)，调用Compiler实例的run方法开始执行编译。

Compiler编译对象掌控者webpack生命周期，不执行具体的任务，只是进行一些调度工作。比如执行模块创建、依赖收集、分块、打包等

调用run之后，创建Compiltation实例，每次构建都会新创建一个Compiltation实例，包含了这次构建的基本信息

Webpack 会在特定的时间点广播出特定的事件，插件在监听到感兴趣的事件后会执行特定的逻辑。

编译:从entry 触发，对每个Module 串行调用对应的 Loader对模块进行翻译，再找出该模块依赖的模块，递归进行编译处理。

从配置文件（ webpack.config.js ）中指定的 entry 入口，开始解析文件构建 AST 语法树

根据依赖关系图，组装成包含多个模块的Chunk，将个Chunk转换成文件输出。

不同entry生成不同chunk，动态导入也会生成自己的chunk

3、Webpack中loader的作用/ loader是什么？

Loader 是webpack中提供了一种处理多种文件格式的机制，因为webpack只认识JS和JSON，所以Loader相当于翻译官，将其他类型资源进行预处理。

用于对模块的"源代码"进行转换。

loader支持链式调用，原型调用的顺序是从右往左。原型链中的每个loader会处理之前已处理过的资源，最终变为js代码。

可以通过 loader 的预处理函数，为 JavaScript 生态系统提供更多能力。

4、常见的loader有哪些？

less-loader: 将less文件编译成css文件，开发中，我们常常会使用less预处理器编写css样式，使开发效率提高

css-loader: 将css文件变成commonjs模块加载到js中，模块内容是样式字符串

style-loader: 创建style标签，将js中的样式资源插入标签内，并将标签添加到head中生效

ts-loader: 打包编译Typescript文件

5、Plugin有什么作用？Plugin是什么？

Plugin功能更强大，主要目的就是解决loader 无法实现的事情，比如打包优化和代码压缩等。

Plugin加载后，在webpack构建的某个时间节点就会触发plugin定义的功能，帮助webpack做一些事情。实现对webpack的功能扩展。

6、常见的Plugin有哪些？

html-webpack-plugin 处理html资源，默认会创建一个空的HTML，自动引入打包输出的所有资源（js/css）

mini-css-extract-plugin 打包过后的css在js文件里，该插件可以把css单独抽出来

clean-webpack-plugin 每次打包时候，CleanWebpackPlugin 插件就会自动把上一次打的包删除

7、Webpack中Loader和Plugin的区别

webpack 就像一条生产线,要经过一系列处理流程(loader)后才能将源文件转换成输出结果。 这条生产线上的每个处理流程的职责都是单一的,多个流程之间有存在依赖关系,只有完成当前处理后才能交给下一个流程去处理。

插件就像是一个插入到生产线中的一个功能,在特定的时机对生产线上的资源做处理。 webpack 在运行过程中会广播事件,插件只需要监听它所关心的事件,就能加入到这条生产线中,去改变生产线的运作。

8、如何利用webpack来优化前端性能？

代码压缩

按需加载

代码分割 splitChunks - 在optimization配置项中配置

1.可以将node\_\_mudules中代码单独打包成一个chunk输出（比如使用了jqury？）

2.会自动分析多入口chunk中，有没有公共的文件，如果有会打包成单独的一个chunk不会重复打包

使用Dll进行分包

正常情况下node\_module会被打包成一个文件

使用dll技术，对可以将那些不常更新的框架和库进行单独打包，生成一个chunk

使用路由懒加载

在代码中所有被 import()函数引用的模块，都将打成一个单独的包，放在 chunk 存储的目录下。在浏览器运行到这一行代码时，就会自动请求这个资源，实现异步加载。

9、Webpack如何配置压缩代码？压缩了什么？

在optimization配置项中来配置该插件作为压缩器进行压缩。

在plugins里使用该插件进行压缩

js压缩:利用terser-webpack-plugin

css压缩:利用了optimize-css-assets-webpack-plugin 插件

删除了console、注释、空格、换行、没有使用的css代码等

10、如何提高webpack的构建速度？

思路1：减少需要构建的文件或代码

HMR(Hot Module Replacement) 模块热替换只重新构建发生变化的模块 – 开发环境中

缩小处理范围:合理利用这两个属性exclude：不需要处理的文件 和 include：需要处理的文件

babel缓存 第二次构建时，会读取之前的缓存，只重新构建变化的文件

使用Dll进行分包 --> 分包方便按需加载

正常情况下node\_module会被打包成一个文件

使用dll技术，对可以将那些不常更新的框架和库进行单独打包，生成一个chunk

项目源代码也需要拆分，可以根据路由来划分打包文件 --> 怎么实现路由懒加载？webpack中如何实现组件异步加载？

思路2：多进行进行构建

多进程打包 thread-loader,将其放在费时的loader之前

进程启动和进程通信都有开销，工作时间比较长，才需要多进程打包

十、Git/SVN篇

1、Git是什么？

Git是目前世界上最先进的分布式版本控制系统，旨在快速高效地处理从小型到大型项目的所有事务

特性：易于学习，占用内存小，具有闪电般快速的性能

使用Git和Gitlab搭建版本控制环境是现在互联网公司最流行的版本控制方式

2、简述Git中如何将一个新增文件添加到本地仓库？

git config --global user.name '开发人员名称'

git add 文件名称/\* //将当前文件或当前文件夹下未添加到栈存区的文件添加栈存区

git commit 文件名称/\* -m '完善的注释'

//将栈存区当前文件或当前文件夹下的全部文件添加到版本本地仓库

git log //查看文件版本号

3、Git常用命令

文字整理(可直接复制使用)

1、git config - - 可以配置git的参数，可以使用 git config --list查看已经配置的git参数。

其中有三个级别的保存位置，

–system（本系统）

–global（当前用户，全局）

–local（本地配置，当前目录）

默认使用–local

配置用户名及邮箱

git config --global user.name “\*\*\*\*”

git config --global user.email 1\*\*\*\*\*\*\*\*@qq.com

2、git init - - 初始化代码仓库

3、git clone - - 克隆远程仓库

PS:如果希望在克隆的时候，自己定义要新建的项目目录名称，可以在上面的命令末尾指定新的名字：

$ git clone git://github.com/schacon/grit.git mygrit

4、git add - - 把需要提交的所有修改放到暂存区（Stage）

git add file – 提交指定文件

git add . || git add -A – 提交所有文件

git add \*.js – 提交所有.js格式文件

git add -f file – 强制添加

5、git diff - - 查看当前目录的所有修改(#当暂存区中没有文件时，git diff比较的是，工作区中的文件与上次提交到版本库中的文件。

#当暂存区中有文件时，git diff则比较的是，当前工作区中的文件与暂存区中的文)

git diff HEAD - - file – 比较工作区中的文件与版本库中文件的差异。HEAD指向的是版本库中的当前版本，而file指的是当前工作区中的文件。

6、git commit -m “message” - - 提交代码

7、git rm - - 会把文件从当前目录删除（不会保存删除的文件）。如果需要从Git仓库中删除，但保留在当前工作目录中，亦即从跟踪清单中删除，可以使用git rm -r --cached readme.md

PS:如果一个文件已经被提交到版本库，那么你永远不用担心误删，但是要小心，你只能恢复文件到最新版本，你会丢失最近一次提交后你修改的内容。

8、git log - - 查看历史记录，git log命令显示从最近到最远的提交日志

git log --graph – 查看分支合并图

9、git reflog - - 用来记录你的每一次命令

10、git remote - - 查看当前的远程库

11、git remote -v - - 可以显示对应的克隆地址（对于多个远程仓库很有用）

12、git remote add [short\_name][url] - - 可以添加新的远程仓库

13、git remote add origin < address > - - 关联一个远程库

14、git fetch [remote-name] - - 可以从远程仓库抓取数据到本地。【git fetch origin release/sprint-70】

15、git pull - - 更新数据

16、git push [remote\_name] [branch\_name] - - 推送数据到远程仓库 默认使用origin和master

17、git push -u origin master [-f] - - 第一次将本地库的所有内容推送到远程库上

18、git remote show origin - - 查看远程仓库信息

19、git remote rename [old\_name][new\_name] - - 远程仓库重命名

20、git remote rm [remote\_name] - - 删除远程仓库

21、git branch -d < name > - - 删除本地分支

22、git tag - - 显示当前库中的标签

23、git branch - - 可显示当前所有分支。可以使用–merged和–no-merged查看已经合并、未合并的分支。

24、git branch <branch\_name> - - 创建新分支

25、git branch -r - - 查看远程仓库分支

26、git checkout <branch\_name> - - 切换到指定的分支

27、git checkout -b <branch\_name> - - 创建新分支并切换到该分支

28、git merge 合并分支

举例：

将hotfix分支合并到master上需要：

git checkout master

git merge hotfix

合并之后可以使用git branch -d hotfix删除分支。

如果合并时存在冲突，需要手工修改

合并分支时，加上—no-ff参数就可以用普通模式合并，合并后的历史有分支，能看出来曾经做过合并，而fast forward合并就看不出来曾经做过合并。

29、git checkout . --恢复上次提交状态

git checkout --file - - 文件在工作区的修改全部撤销

一种是file修改后还没有被放到暂存区，现在，撤销修改就回到和版本库一模一样的状态；

一种是file已经添加到暂存区后，又作了修改，现在，撤销修改就回到添加到暂存区后的状态。

30、git status – 用于显示工作目录和暂存区的状态。使用此命令能看到那些修改被暂存到了, 哪些没有, 哪些文件没有被Git tracked到。git status不显示已经commit到项目历史中去的信息。

31、git reset --hard HEAD^ - - #版本回退

git reset --hard commitId - - 取消回退，commitId为你想要回到的未来版本号

PS:Git必须知道当前版本是哪个版本，在Git中，用HEAD表示当前版本，上一个版本就是HEAD，上上一个版本就是HEAD^，当回退版本较早时可以写成HEAD~100。

32、git stash - - 储藏可以获取你工作目录的中间状态——也就是你修改过的被追踪的文件和暂存的变更——并将它保存到一个未完结变更的堆栈中，随时可以重新应用。

现在你想切换分支，但是你还不想提交你正在进行中的工作；所以你储藏这些变更。为了往堆栈推送一个新的储藏，只要运行git stash。把所有未提交的修改（包括暂存的和非暂存的）都保存起来，用于后续恢复当前工作目录。

PS:需要说明一点，stash是本地的，不会通过git push命令上传到git server上。

33、git stash list - - 查看现有的所有储藏，此命令显然暗示了git stash可以多次保存工作进度，并用在恢复时候选择。

34、git stash pop [–index] [ < stash > ] - - 重新应用已经实施的储藏（删除储藏）

如果不使用任何参数，会恢复最新保存的工作进度，并将恢复的工作进度从存储的工作进度列表中清除。

如果提供< stash>参数（来自git stash list显示的列表），则从该< stash>中恢复。恢复完毕也将从进度列表中删除< stash>。

选项–index除了恢复工作区的文件外，还尝试恢复暂存区。

35、git stash drop [< stash >] - - 删除一个存储的进度。（默认删除最新的进度）

36、git stash clear - - 清空当前所有的stash

37、git stash branch < branchname > < stash > - - 基于储藏进度创建分支。

4、Git和SVN有什么区别?

git是分布式版本控制，svn是集中式版本控制（核心区别）

git相对于svn的优势就是不需要网络即可版本控制

git把内容按数据方式存储，而svn是按文件

git可以是公用的，可以分享，svn基本是公司内部才能访问，网外不方便访问

git不依赖中央服务器，即使服务器有问题也不受影响，svn依赖服务器，一旦服务器有问题就会受影响

git没有一个全局的版本号，svn有