一阶段

网络中使用最多的图片格式有哪些?

- JPEG、GIF、PNG

请简述css盒子模型

- CSS盒子模型也叫做框模型，具备内容(content)、填充(padding)、边框(border)、边界(margin)这些属性。在CSS中，每一个元素都被视为一个框，而每个框都有三个属性：1.border:元素的边框（可能不可见），用于将框的边缘与其他框分开； 2.margin：外边距，表示框的边缘与相邻框之间的距离，也称为页边空白； 3.padding:内边距，表示框内容和边框之间的空间。 4.width 和 height 指的是内容区域的宽度和高度。增加内边距、边框和外边距不会影响内容区域的尺寸，但是会增加元素框的总尺寸。

视频/音频标签的使用

- 音频

```
<audio src="课堂示例/梦然-少年.mp3" controls autoplay loop muted></audio> 内联
<audio controls>
    <source src="课堂示例/梦然-少年.ogg" type="audio/ogg"></source>
    <source src="课堂示例/梦然-少年.mp3" type="audio/mpeg"></source>
    你的浏览器不支持此文件，换个电脑吧
</audio>
```

  - controls 控件
  - autoplay 自动播放 谷歌不支持，火狐静音下支持 ie支持
  - loop 循环播放
  - muted 静音播放

- 视频

```
<video src="课堂示例/qr.ogg" controls height="300" autoplay loop muted>
</video>
 <video controls>
    <source src="课堂示例/wje.mp4" type="video/mp4"></source>
    <source src="课堂示例/wje.ogg"  type="video/ogg"></source>
    <source src="课堂示例/wje.webm" type="video/webm"></source>
</video>
```

  - controls 控件
  - autoplay: 自动播放 谷歌和火狐静音下支持 ie支持

- muted 静音播放
- loop 循环
- poster 未播放前显示的内容

HTML5新增的内容有哪些?

- H5表单新增属性
  - required (必填) placeholder (提示信息) autofocus(自动聚焦) autocomplete = "on/off" (自动提示)
- h5的语法
  - DOCTYPE可以使用小写
  - 单标签没有结束标签
  - 可以省略结束的标签: li、dt、dd、p、option、colgroup、thead、tbody、tfoot、tr、td、th
  - 可以省略的标签 html head body tbody
- 新增标签 **特点：语义化**，ie8及以下不生效
  - header标签 头部
  - section 划分板块
  - article 与上下文无关的内容
  - aside 可以放侧边工具栏
  - nav 导航
  - figure和figcaption 类似于dl标签
  - footer 底部
  - main 比较重要的部分
  - mark 标记 默认是黄色的高亮，可以修改样式 内联

Css3新增的特性

- 1、CSS3选择器

  2、CSS3边框(Borders)

  3、CSS3背景

  4、CSS3渐变

  5、CSS3文本效果

  6、CSS3字体

  7、CSS3转换和变形

1）2D转换方法

2）3D转换属性

8、CSS3过度

9、CSS3动画

10、CSS3多列

11、CSS3盒模型

12、CSS3伸缩布局盒模型(弹性盒子)

13、CSS3多媒体查询

清除浮动的方式有哪些？请说出各自的有点

- 额外标签法 (在最后一个浮动元素的后面新加一个标签如

  ，并在其CSS样式中设置clear: both;)

  1.优点：简单，通俗易懂，写少量代码，兼容性好 2.缺点：额外增加无语义html元素，代码语义化差，后期维护成本大

- 给父级设置高度 1.优点：简单，写少量代码，容易掌握 2.缺点：不够灵活，只适用于高度固定的布局

- 触发父级BFC (如给父元素设置overflow:hidden，特别注意的是：在IE6中还需要触发hasLayout，例如给父元素设置zoom:1。原理是触发父级BFC后，父元素在计算高度时，浮动的子元素也会参与计算)

  1.优点：简单，代码简洁 2.缺点：设置overflow:hidden容易造成不会自动换行导致超出的尺寸被隐藏掉，无法显示要溢出的元素

- 使用after伪元素

  ```
  .clearfix::after {
      content: ".";
      display: block;
      height: 0;
      line-height: 0;
      clear: both;
      visibility:hidden;
      font-size: 0;
  }
  ```

  1.优点：符合闭合浮动思想，结构语义化正确 2.缺点：代码量多，因为IE6-7下不支持after伪元素

定位的属性值有何区别?

- static(默认)
- relative(相对定位)
  - 参考物：自己本身位置
  - 特点：移动后仍然占据空间
- absolute(绝对定位)
  - 参考物： 参考有定位属性(除了static)的祖先元素,定位元素通过一层一层想上找，找到有定位的祖先元素，如果找到body依然没有找到有定位的祖先元素，参考浏览器窗口(**子绝父相**)。
  - 特点： 1、脱离文档流，且文字能被遮挡 2、块级元素设置margin:0 auto;会失效 3、内联元素设置定位转换成块元素
- fixed(固定定位)
  - 参考物：浏览器的窗口
  - 特点：1、不会跟随滚动条的滚动而滚动 2、脱离文档流 3、宽度自适应的时候，宽度不显示，可以通过设置width:100%;
- sticky(粘性定位)
  - 参考物：浏览器的窗口
  - 方向：top
  - 特点：没有达到top值之前正常显示，达到top值之后类似于固定定位，不会跟随滚动条滚动而滚动

Border-box与content-box的区别?

- content-box是符合w3c标准的盒模型，也是默认的盒模型，它是先根据设定的样式确定元素content的宽高，有border和padding的情况下再额外增加盒子的宽高，内容宽高不受影响，设定多少就是多少。
- border-box不符合w3c标准盒模型，也称为怪异盒模型，它是先根据设定的样式固定盒子的宽高，如果有border和padding的情况下，再根据盒子的宽高减去border或者padding，内容的宽高会受到影响，会被减去border或者padding。

如何让chrome浏览器显示小于12px的文字?

- 使用transform:scale()进行缩放

Css选择器有哪些，那些属性可以继承，优先级如何计算？Css3新增的伪类有哪些?

- 1、标签选择器 2、类选择器 3、id选择器 4、全局选择器 5、群组选择器
- css3新增： 1、属性选择器 2、伪类选择器
  - 结构性伪类选择器(child系列)
    - E:first-child E必须是父元素里面的第一个孩子 E:last-child E必须是父元素里面的最后一个孩子 **E:nth-child(n)** 不匹配前面的元素类型，如果对应的位置是该元素才匹配 E:only-child 必须只有他自己一个孩子
  - 结构性伪类选择(type系列)
    E:first-of-type 匹配到该元素中第一个孩子 E:last-of-type 匹配到该元素的最后一个孩子 **E:nth-of-type(n)** 匹配到该元素的第几个孩子 n可以是表达式 2n 3n 2n+1 even(偶数) odd(奇数) E:nth-last-of-type(n) 该元素的倒数第几个
  - 目标伪类： 结合锚点使用

```html
  <a href="#box">点击</a>
 <div id="box">
     这里是要跳转过来的
 </div>
```

```css
  #box:target {
      color: pink;
  }
```

  - 状态伪类选择器
    a. :enabled 元素可编辑
    b. :disabled 元素不可编辑
    c. :checked 选中
    d. ::selection 高亮状态 一般修改字体颜色和背景色
  - 动态伪类选择器
    a. :link 未访问前
    b. :visited 访问过后的
    c. :hover 鼠标滑过
    d. :active 鼠标点击

3、层级选择器

  - 后代选择器
  - 子项选择器 （选择器>选择器）
  - 相邻的兄弟 （选择器+选择器）
  - 相邻的兄弟们 （选择器~选择器）

- 可继承的样式： font-size font-family color, UL LI DL DD DT;　不可继承的样式：border padding margin width height ;
- 优先级算法: !important > 行内样式 > id > class > 标签 > 通配 > 继承

网页中有大量图片加载很慢 你有什么办法进行优化?

- 1、图片懒加载，在页面上的未可视区域可以添加一个滚动条事件，判断图片位置与浏览器顶端的距离与页面的距离，如果前者小于后者，优先加载。

  2、如果为幻灯片、相册等，可以使用图片预加载技术，将当前展示图片的前一张和后一张优先下载。

  3、如果图片为css图片，可以使用CSSsprite，SVGsprite，Iconfont、Base64等技术。

  4、如果图片过大，可以使用特殊编码的图片，加载时会先加载一张压缩的特别厉害的缩略图，以提高用户体验。

  5、如果图片展示区域小于图片的真实大小，则因在服务器端根据业务需要先行进行图片压缩，图片压缩后大小与展示一致。

行内元素/块级元素有哪些?

- 行类：span b strong i em a u del img
- 块元素: div p ul li ol h1-h6 dl dd dt

浏览器的标准模式和怪异模式区别?

- 标准模式：是浏览器按照W3C标准解析执行代码，这样用规定的语法去渲染，就可以兼容各个浏览器，保证以正确的形式展示网页。
- 怪异模式：是使用浏览器自己的方式解析执行代码，因为不同浏览器解析执行的方式不一样，所以我们称之为怪异模式。

Margin和padding在什么场合下使用?

- margin:

  需要在border外侧添加空白时；

  空白处不需要背景（色）时；

  上下相连的两个盒子之间的空白，需要相互抵消时。

- padding：

  需要在border内测添加空白时；

空白处需要背景（色）时；

上下相连的两个盒子之间的空白，希望等于两者之和时。

弹性盒子布局属性有那些请简述?

- 父元素
  - 形成弹性盒 display:flex;
  - 主轴方向 flex-direction
    - row 主轴从左向右 默认值
    - row-revese 主轴从右向左
    - column 主轴从上到下
    - column-reverse 主轴从下到上
  - 主轴方向排列方式 justify-content
    - flex-start 主轴起点 默认值
    - flex-end 主轴终点
    - center 居中
    - space-between 两端对齐
    - space-around 中间的留白是两边的2倍
    - space-evenly 平均分配留白
  - 交叉轴排列方式 align-items
    - stretch 拉伸 默认值 **去掉子元素的高度**
    - flex-start 交叉轴的起点
    - flex-end 交叉轴的终点
    - center 居中
  - 换行 flex-wrap
    - nowrap 不换行,默认值，会将子元素压缩
    - wrap 换行
    - wrap-reverse 反向换行
  - **多行之间**的排列方式 align-content
    - stretch 拉伸 默认值 需要去掉子元素的高
    - flex-start 主轴起点依次排列
    - flex-end 主轴终点依次排列
    - center 居中
    - space-between 两端对齐
    - space-around 中间的两端的2倍
    - space-evenly 平均分配
  - *flex-flow*: flex-direction(主轴方向) flex-wrap(换行);
- 子元素
  - 重写子项对应的交叉轴的对齐方式 align-self
    - stretch 拉伸 默认值 **去掉子元素的高度**
    - flex-start 交叉轴的起点

        - flex-end 交叉轴的终点
        - center 居中
    - *放大 flex-grow*
        - 0 不放大
        - 数值 填充剩余的空间
    - 压缩 flex-shrink
        - 1 压缩
        - 0 不压缩
        - 实现导航的滚动效果
            a. 子项的宽度超出了父容器的宽度
            b. 设置子项不压缩 flex-shrink:0;
            c. 父元素设置溢出显示滚动条 overflow-x:auto;
    - *子项的宽度* flex-basis:数值+px 类似于宽度
    - *排序 order*
        - 数值 值越大越像后，可以设置负数
    - 复合写法 flex: flex-grow(放大) flex-shrink(压缩) flex-basis(子项的宽度)
        - flex: 0(不放大) 1(压缩) auto(本身设置的宽度);
        - **flex: 1; 等同于flex-grow:1;放大**

怎么实现标签的禁用

- style="pointer-events: none;"

Flex布局原理?

- 1、采用 Flex 布局的元素，称为 Flex 容器（flex container），简称"容器"。它的所有子元素自动成为容器成员，称为 Flex 项目（flex item），简称"项目"

  2、Flex 容器默认存在两根轴：水平的主轴（main axis）和垂直的交叉轴（cross axis）。项目默认沿主轴排列，当然项目的排列方向也可以通过改变属性来控制。

  3、主轴的开始位置（与边框的交叉点）叫做main start，结束位置叫做main end；交叉轴的开始位置叫做cross start，结束位置叫做cross end。单个项目占据的主轴空间叫做main size，占据的交叉轴空间叫做cross size。

Px与rem的区别

- px：绝对单位，页面按精确像素展示。

  em：相对单位，基准点为父节点字体的大小，如果自身定义了font-size按自身来计算（浏览器默认字体是16px），整个页面内1em不是一个固定的值。

rem：相对单位，可理解为”root em”，相对根节点html的字体大小来计算，CSS3新加属性，chrome/firefox/IE9+支持。

网页的三层结构有哪些

```
结构层(HTML)、表示层(CSS)、行为层(JavajScript)
```

请简述媒体查询?

- 媒体指的就是各种设备 (移动设备, PC设备) 查询指的是要检测属于哪种设备 媒体查询: 通过查询当前属于哪种设备, 让网页能够在不同的设备下正常的预览
- 作用：实现页面在不同设备下正常预览 [判断当前设备]
- 媒体类型：将不同的设备划分为不同的类型 all (所有的设备) print (打印设备) screen (电脑屏幕, 平板电脑,智能手机)
- 语法 @media 关键字 设备类型 and (媒体特性){}
- 关键字：all(所有设备类型)/only(限定某种设备)/not(排除设备)

Doctype作用

- DOCTYPE是document type (文档类型) 的缩写。<!DOCTYPE >声明位于文档的最前面，处于标签之前，它不是html标签。主要作用是告诉浏览器的解析器使用哪种HTML规范或者XHTML规范来解析页面。

常见的兼容性一阶段内容中记几个

Rem缺点

- 目前ie不支持 对pc页面来讲使用次数不多；
- 数据量大：所有的图片，盒子都需要我们去给一个准确的值；才能保证不同机型的适配；

垂直与水平居中的方式

- 1、设置该元素 left:50% , margin-left:当前盒子宽度的一半(负值) 2、设置该元素 top:50% , margin-top:当前盒子高度的一半(负值)
- 1、设置left:0;top:0;right:0;bottom:0; 2、设置margin:auto

三栏布局方式两边固定中间自适应

- 利用浮动:

```
<div>中间自适应,左边和右边固定宽度为300px,高度为100px</div>
    <div>
        <div class="main_box">
            <div class="left_section"></div>
            <div class="right_section"></div>
            <div class="center_section"></div>
        </div>
    </div>
    <style>
        html * {
            padding: 0;
            margin: 0
        }
        .main_box {
            height: 100px;
        }
        .left_section {
            float: left;
            width: 300px;
            background: yellow;
            height: 100%;
        }
        .right_section {
            float: right;
            width: 300px;
            background: blue;
            height: 100%;
        }
        .center_section {
            background: red;
            height: 100%;
        }
    </style>
```

- flex布局

```
<div>中间自适应,左边和右边固定宽度为300px,高度为100px</div>
    <div>
        <div class="main_box">
            <div class="left_section"></div>
            <div class="center_section"></div>
            <div class="right_section"></div>
        </div>
    </div>
    <style>
        .main_box {
            display: flex;
        }
        .left_section {
            width: 300px;
            background: yellow;
            height: 100px;
```

```
        }
        .right_section {
            width: 300px;
            background: blue;
            height: 100px;
        }
        .center_section {
            background: red;
            height: 100px;
            flex: 1;
        }
    </style>
```

第二阶段

Js基本数据类型有哪些?

- Number,String,Boolean,Function,Object,undefined

Ajax如何使用

- 原生:

```
    //1.实例化一个XMLHttpRequest对象
    let http = new XMLHttpRequest();
    //2.规划一个请求（三要素）
    // (1).请求方式GET||POST (2).请求地址 (3).同步还是异步
    http.open("get","http://127.0.0.1")
    //3.真实发送请求
    http.send()
    //4.接收来自服务器端的响应
    http.onreadystatechange = function(){
        //服务器端已将返回的内容交付给客户端手里
        if(http.readyState === 4){
            console.log(http.responseText)
        }
    }
```

- jquery

```
    $.ajax({
        url:"http://127.0.0.1",
        data:{
            name:"hl",
            age:"18"
        },
```

```
        //请求方式为jsonp
        dataType:"jsonp"
        async:false,
        method:"get",
        success(data){
            console.log(data)
        }
    })
```

如何判断一个数据是NaN

- 用isNaN()方法
- 利用NaN不等于NaN的特殊性

```
var a = NaN;
a == a;
```

Js中null与undefined区别

- Undefined类型只有一个值，即undefined。当声明的变量还未被初始化时，变量的默认值为undefined。
- Null类型也只有一个值，即null。null用来表示尚未存在的对象，常用来表示函数企图返回一个不存在的对象。

闭包是什么有什么特性，对页面会有什么影响

- 闭包可以简单理解成“定义在一个函数内部的函数“。当其中一个内部函数在包含它们的外部函数之外被调用时，就会形成闭包。 特点： 1.函数嵌套函数。 2.函数内部可以引用外部的参数和变量。 3.参数和变量不会被垃圾回收机制回收。 用处 常驻内存 会增大内存的使用量 使用不: 1.读取函数内部的变量； 2.这些变量的值始终保持在内存中，不会在外层函数调用后被自动清除。 优点：1:变量长期驻扎在内存中； 2:避免全局变量的污染； 3:私有成员的存在 ； 缺点： 当会造成内存泄露

事件委托是什么？如何确定事件源

- 事件委托就是利用冒泡的原理，把事件加到父级上，触发执行效果。

ES6新特性

- ES6新特性 const和let 模板字符串 箭头函数 函数的参数默认值 对象和数组解构 for...of 和 for...in ES6中的类

Let与var与const的区别

- var声明的变量会挂载在window上，而let和const声明的变量不会： var声明变量存在变量提升，let和const不存在变量提升 let和const声明形成块作用域 同一作用域下let和const不能声明同名变量，而var可以 let 暂存死区

数组方法有哪些请简述

- push() 从后面添加元素，返回值为添加完后的数组的长度 arr.pop() 从后面删除元素，只能是一个，返回值是删除的元素 arr.shift() 从前面删除元素，只能删除一个 返回值是删除的元素 arr.unshift() 从前面添加元素, 返回值是添加完后的数组的长度 arr.splice(i,n) 删除从i(索引值)开始之后的那个元素。返回值是删除的元素 arr.concat() 连接两个数组 返回值为连接后的新数组 str.split() 将字符串转化为数组 arr.sort() 将数组进行排序,返回值是排好的数组，默认是按照最左边的数字进行排序，不是按照数字大小排序的 arr.reverse() 将数组反转,返回值是反转后的数组 arr.slice(start,end) 切去索引值start到索引值end的数组，不包含end索引的值，返回值是切出来的数组 arr.forEach(callback) 遍历数组,无return 即使有return，也不会返回任何值，并且会影响原来的数组 arr.map(callback) 映射数组(遍历数组),有return 返回一个新数组 。 arr.filter(callback) 过滤数组，返回一个满足要求的数组

ES5 数组对象的全局方法

```
* forEach(function(item,index,array){}):帮助我们循环数组，这个方法的返回值是undefined。
* map(function(item,index,array){}):如果需要操作元素里面的每一个值并且返回一个新数组的话，那么map就是最好的选择。这个方法的返回值是一个新数组，新数组内部的成员要看函数体内部的return
* reduce(function(a,b,c){})
* filter(function(item,index,array){}):有返回值，返回一个新数组，新数组的长度不确定，但一定小等于原数组的长度，新数组内部的成员取决于执行函数时return的是true还是false,如果是false那么新数组内部将过滤掉该成员。
```

字符串的方法

-
    - indexOf()：根据字符查找对应的下标。
    - lastIndexOf()：根据字符查找对应的下标(从后往前找)。
    - charAt() 根据下标查找字符
    - concat() 拼接字符串
    - replace() 替换
    - slice() 通过起始下标和终止下标截取一段字符串
    - substring() 同slice功能和参数都一样，不过第二个参数不允许为负值。
    - split() 根据所传参数的字符做为分隔将原字符串切割为长度若干的一个数组
    - substr() 通过起始下标和截取数量来截取一段字符串
    - toLowerCase() 转换字符串为小写
    - toUpperCase() 转换....为大写

Json如何新增/删除键值对

什么是面向对象请简述

-
  - 使用对象的时候只关注对象提供的一些功能，不关注内部的一些细节
  - 面向对象是一种通用的思想，并非只有编程中能使用。任何事情都可以使用
  - 面向对象的三大基本特征
  - 抽象：抓住核心问题
  - 封装：不考虑内部实现，只考虑功能使用
  - 继承：从已有的对象上继承出新的对象
  - 面向对象思维主张的是：团队---配合---分工---协作，将大问题拆分成诺干个小问题，并试图用分工协作来完成

普通函数和构造函数上的区别

- 在命名规则上，构造函数一般是首字母大写，普通函数遵照小驼峰式命名法。 在函数调用的时候： function fn() { } 构造函数：1. new fn( ) 2 .构造函数内部会创建一个新的对象，即f的实例 3. 函数内部的this指向 新创建的实例 4. 默认的返回值是f的实例 普通函数：1. fn( ) 2. 在调用函数的内部不会创建新的对象 3. 函数内部的this指向调用函数的对象（如果没有对象调用，默认是window） 4. 返回值由return语句决定

  构造函数的返回值： 有一个默认的返回值，新创建的对象（实例）； 当手动添加返回值后（return语句）： 1. 返回值是基本数据类型-->真正的返回值还是那个新创建的对象（实例） 2. 返回值是复杂数据类型（对象）-->真正的返回值是这个对象

- 构造函数的this指向实例化后的那个对象 普通函数的this指向调用该函数的那个对象

请简述原型/原型链/继承

- 继承：继承就是在子类构造函数中继承父类构造函数的私有属性和原型属性。我们在子类构造函数中使用call或apply方法调用父类构造函数并改变其this指向为子类构造函数的this，此时子类的构造函数就继承了父类的私有属性和私有方法。将父类的实例化对象赋值给子类的原型对象，此时子类就继承了父类的原型属性和原型方法。

- 原型：原型是 function 对象的一个属性，它定义了构造函数制造出的对象的公共祖先。通过该构造函数产生的对象，可以继承该原型的属性和方法。原型也是对象。

- 原型链：在实例化对象(构造函数)时，寻找某个属性(如demo中的name属性),在当前属性中无法找到属性，会从xxx.__proto__中进行寻找。若还未找到，会继续向xxx.**proto**.**proto** 中进行寻找。利用上述原理，可通过继承方法(如demo2,Programmer函数中的prototype继承于Person中的属性)，使子继承于父对象中的属性，若存在同名属性，则取子对象中的值。原型链其实就是通过__proto__属性，以层层递推的方式，一层一层的寻找需要的属性。

Promise的理解

- Promise 是异步编程的一种解决方案，相比传统的解决方案——回调函数和事件更合理和更强大。它由社区最早提出和实现，ES6将其写进了语言标准，统一了语法，原生提供了Promise。Promise，简单说就是一个容器，里面保存着某个未来才会结束的事件(通常是一个异步操作）的结果。从语法上说，Promise是一个对象，从它可以获取异步操作的消息。 Promise 对象的状态不受外界影响

Promise在哪里使用过

- 一般用在发送ajax请求

请简述async的用法

- async函数返回一个 Promise 对象，可以使用then方法添加回调函数。当函数执行的时候，一旦遇到await就会先返回，等到异步操作完成，再接着执行函数体内后面的语句

Css预处理sass less是什么？为什么使用他们

- Sass 和 LESS 都是是 CSS 预处理器，是 CSS 上的一种抽象层，是一种特殊的 语法/语言 最终会编译成 CSS，他们是一种动态样式语言，将 CSS 赋予了动态语言的特性，如变量，继承，运算， 函数。

- 结构清晰，便于扩展。可以方便地屏蔽浏览器私有语法差异。这个不用多说，封装对浏览器语法差异的重复处理，减少无意义的机械劳动。可以轻松实现多重继承。

Js中.call()与.apply()区别

- 每一个函数都有call和apply的方法:都是帮助我们调用函数的,但是在调用的时候可以改变本次执行函数中this的指向。他们的不同点就在于call是无限多个参数，而apply有两个参数，第二个参数是数组，代表实参集合。一般来说我们也就是在用JS实现面向对象的继承特点时会使用到这个方法。

为什么会造成跨域/请简述同源策略

- 不同域的页面之间不能相互访问各自的页面内容。
- 同源组策略：就是浏览器保护浏览器安全的一种机制，它不允许客户端请求从A服务器请求过来的页面往B服务器去发送Ajax请求。

This指向

- 出现在一般函数中或者是全局作用域下，this指向的window对象。 出现在事件处理函数（事件句柄）中，this指向的是触发事件的元素。 出现在对象的方法中，指向该对象。 出现在构造函数中，指向的实例化对象

请输出三种减少页面加载时间的方式

- 减少http请求（合并文件、合并图片） 优化图片 css样式的定义放置在文件头部 Javascript脚本放在文件末尾 压缩合并Javascript、CSS代码 网址后加斜杠（如www.campr.com/目录，会判断这个“目录是什么文件类型，或者是目录。） 服务器开启gzip压缩

什么是jsonp工作原理是什么？他为什么不是真正的ajax

- 前端使用一个script标签来创建一个HTTP请求，并且事先声明一个回调函数，该回调函数的名字由callback参数的参数值发送给后端。后端接收到来自前端的请求后利用callback参数的参数值和要给前端返回的数据拼接成一段JS执行函数的代码段，此时实参就是要返回给前端的数据。前端接收到后端返回的结果后，会自动的执行事先声明好的回调函数，此时函数的形参就代表了后端要返回的数据。
- 实质不同 ajax的核心是通过xmlHttpRequest获取非本页内容 jsonp的核心是动态添加script标签调用服务器提供的js脚本 jsonp只支持get请求，ajax支持get和post请求

请掌握2种以上数组去重的方式

- 穷举法:定义一个新数组，并存放原数组的第一个元素，然后将元素组一一和新数组的元素对比，若不同则存放在新数组中。

```
function unique(arr) {
        let newArr = [arr[0]];
        for (let i = 1; i < arr.length; i++) {
            let repeat = false;
            for (let j = 0; j < newArr.length; j++) {
                if (arr[i] === newArr[j]) {
                    repeat = true;
                    break;
                }else{

                }
            }
            if (!repeat) {
                newArr.push(arr[i]);
            }
        }
        return newArr;
    }

    console.log(unique([1, 1, 2, 3, 5, 3, 1, 5, 6, 7, 4]));
    // 结果是[1, 2, 3, 5, 6, 7, 4]
```

- Methods 2: 思路：先将原数组排序，在与相邻的进行比较，如果不同则存入新数组。

```
 function unique2(arr) {
          var formArr = arr.sort()
          var newArr=[formArr[0]]
          for (let i = 1; i < formArr.length; i++) {
              if (formArr[i]!==formArr[i-1]) {
                  newArr.push(formArr[i])
              }
          }
          return newArr
      }
      console.log(unique2([1, 1, 2, 3, 5, 3, 1, 5, 6, 7, 4]));
      // 结果是[1, 2, 3,4,5, 6, 7]
```

- Methods 3: 利用 ES6的set 方法。

```
function unique3(arr) {
          //Set数据结构，它类似于数组，其成员的值都是唯一的
          return Array.from(new Set(arr)); // 利用Array.from将Set结构转换成数组
      }
      console.log(unique10([1, 1, 2, 3, 5, 3, 1, 5, 6, 7, 4]));
      // 结果是[1, 2, 3, 5, 6, 7, 4]
```

深浅拷贝是什么？如何实现

- 浅拷贝：就是拷贝对象的引用，而不深层次的拷贝对象的值，多个对象指向堆内存中的同一对象，任何一个修改都会是使得所有对象的值被修改，因为它们公用一条数据。
- 深拷贝：深拷贝不会拷贝引用类型的引用，拷贝的是引用类型的值，形成一个新的引用类型。

为什么js是弱类型语言

- 别的编程语言会对变量的类型有严格的限制，之间的转换也有规定。你开始定义一个变量，是整型它就只能是整型，是字符串它就必须是字符串。而JS就不同了。变量声明的时候不用规定是什么类型的，用的时候它自己根据你赋的值判断。

怎么转换less为css

- 1.使用 npm 安装 lessc，命令行：npm install -g less

  2.然后，进入需要转换的less文件的目标位置。

  3.最后，你只需输入以下两条命令： npm install -g less lessc less文件名.less 生成的css文件名.css

echarts使用最多的是什么

- 数据可视化

For循环与map循环有什么区别

- 1.map只能遍历数组，不能中断，返回值是修改后的数组 2.map方法不会对空数组进行检测，map方法不会改变原始数组。

请写出一个简单的类与继承

```
function Person(name,age,sex){
            this.name = name;
            this.age = age;
            this.sex = sex;
        }
Person.prototype.sayName = function(){
            alert(this.name)
        }
let wdw = new Person("王大伟",18,"男")
console.log(wdw)

function SuperMan(name,age,sex,skill){
    // 继承了Person类的私有属性和私有方法
    Person.apply(this,[name,age,sex]);

    this.skill = skill;
}

// 子类继承父类的原型属性和原型方法
SuperMan.prototype = new Person()


SuperMan.prototype.sk = function(){
    alert(this.skill);
}

let spiderMan = new SuperMan("蜘蛛侠",20,"男","吐丝");
```

同步与异步的区别/阻塞与非阻塞区别

- 阻塞调用是指调用结果返回之前，当前线程会被挂起。调用线程只有在得到结果之后才会返回。非阻塞调用指在不能立刻得到结果之前，该调用不会阻塞当前线程。

重绘和回流是什么

- 当渲染树中的一部分或者全部因为元素的尺寸、布局、隐藏等改变而需要重新构建的时候，这时候就会发生回流。每个页面都至少发生一次回流，也就是页面第一次加载的时候。在回流的时候，浏览器会使渲染树中受到影响的元素部分失效，并重新绘制这个部分的渲染树，完成回流以后，浏览器会重新绘制受到影响的部分元素到屏幕中，这个过程就是重绘。
- 什么时候会发生回流？1、添加或者删除可见的DOM元素的时候 2、元素的位置发生改变

http是什么？有什么特点

- HTTP 就是 “超文本传输协议”
- 特点：1.支持客户/服务器模式。 2.简单快速 3.无连接 4.无状态

HTTP协议和HTTPS区别

- HTTPS和HTTP的区别主要如下：

1、https协议需要到ca申请证书，一般免费证书较少，因而需要一定费用。

2、http是超文本传输协议，信息是明文传输，https则是具有安全性的ssl加密传输协议。

3、http和https使用的是完全不同的连接方式，用的端口也不一样，前者是80，后者是443。

4、http的连接很简单，是无状态的；HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，比http协议安全。

call和apply继承的区别（第一个参数是相同的，第二个的区别在哪）

- call:第一个参数是this的指向，第二个以及后面的所有参数需要一个个进行传递 apply:第一个参数是this的指向，第二个参数是一个数组

箭头函数与普通函数的区别

- 箭头函数是匿名函数，不能作为构造函数，不能使用new 箭头函数不能绑定arguments，取而代之用rest参数...解决 箭头函数没有原型属性 箭头函数的this永远指向其上下文的this，没有办改变其指向， 普通函数的this指向调用它的对象 箭头函数不绑定this，会捕获其所在的上下文的this值，作为自己的this值

什么是js内存泄露?

- 内存泄漏说白了就是本该被回收的内存因为一些异常没有被回收掉，而一直存在于内存中占用内存， 虽然说js有垃圾自动回收机制，但是如果我们的代码写法不当，会让变量一直处于“进入环境”的状态，无法被正常回收。

你如何对网站的文件和资源进行优化?

1.文件合并（目的是减少http请求）：使用css sprites合并图片，一个网站经常使用小图标和小图片进行美化，但是很遗憾这些小图片占用了大量的HTTP请求，因此可以采用sprites的方式把所有的图片合并成一张图片，可以通过相关工具在线合并，也可以在ps中合并。

2.使用CDN（内容分发网络）加速，降低通信距离。

3.缓存的使用，添加Expire/Cache-Control头。

4.启用Gzip压缩文件。

5.将css放在页面最上面。

6.将script放在页面最下面。

7.避免在css中使用表达式。

8.将css, js都放在外部文件中。

9.减少DNS查询。

10.文件压缩：最小化css, js，减小文件体积。

11.避免重定向。

12.移除重复脚本。

13.配置实体标签ETag。

14.使用AJAX缓存，让网站内容分批加载，局部更新。

请简述ajax的执行过程 以及常见的HTTP状态码

- ajax过程 1、创建XMLHttpRequest对象

  XMLHttpRequest()

  ActiveXObject('Microsoft.XMLHTTP') IE下兼容（IE7一下 6）

  2、准备发送

  open()

  三个参数的含义 1、提交方式 Form-method 2、提交地址 Form-action 3、异步（同步）

  3、执行发送动作

  send()

4、指定回调函数

onreadystatechange事件

readyState属性：请求状态 0 （初始化）还没有调用open()方法 1 （载入）已调用send()方法，正在发送请求 2 （载入完成）send()方法完成，已收到全部响应内容 3 （解析）正在解析响应内容 4 （完成）响应内容解析完成，可以在客户端调用了

status属性：服务器(请求资源)的状态 http状态码

5，返回内容

responseText：返回以文本形式存放的内容 responseXML：返回XML形式的内容

- http状态码 1##：请求收到，继续处理

  2##：操作成功收到，分析、接受

  3##：完成此请求必须进一步处理

  4##：请求包含一个错误语法或不能完成

  5##：服务器执行一个完全有效请求失败

  100——客户必须继续发出请求

  101——客户要求服务器根据请求转换HTTP协议版本

  200——交易成功

  201——提示知道新文件的URL

  202——接受和处理、但处理未完成

  203——返回信息不确定或不完整

  204——请求收到，但返回信息为空

  205——服务器完成了请求，用户代理必须复位当前已经浏览过的文件

  206——服务器已经完成了部分用户的GET请求

  300——请求的资源可在多处得到

  301——删除请求数据

  302——在其他地址发现了请求数据

  303——建议客户访问其他URL或访问方式

  304——客户端已经执行了GET，但文件未变化

  305——请求的资源必须从服务器指定的地址得到

306——前一版本HTTP中使用的代码，现行版本中不再使用

307——申明请求的资源临时性删除

400——错误请求，如语法错误

401——请求授权失败

402——保留有效ChargeTo头响应

403——请求不允许 404——没有发现文件、查询或URl

405——用户在Request-Line字段定义的方法不允许

406——根据用户发送的Accept拖，请求资源不可访问

407——类似401，用户必须首先在代理服务器上得到授权

408——客户端没有在用户指定的饿时间内完成请求

409——对当前资源状态，请求不能完成

410——服务器上不再有此资源且无进一步的参考地址

411——服务器拒绝用户定义的Content-Length属性请求

412——一个或多个请求头字段在当前请求中错误

413——请求的资源大于服务器允许的大小

414——请求的资源URL长于服务器允许的长度

415——请求资源不支持请求项目格式

416——请求中包含Range请求头字段，在当前请求资源范围内没有range指示值，请求也不包含If-Range请求头字段

417——服务器不满足请求Expect头字段指定的期望值，如果是代理服务器，可能是下一级服务器不能满足请求

500——服务器产生内部错误

501——服务器不支持请求的函数 502——服务器暂时不可用，有时是为了防止发生系统过载

503——服务器过载或暂停维修

504——关口过载，服务器使用另一个关口或服务来响应用户，等待时间设定值较长

505——服务器不支持或拒绝支请求头中指定的HTTP版本

预加载和懒加载的区别，预加载在什么时间加载合适

- 预加载是指在页面加载完成之前，提前将所需资源下载，之后使用的时候从缓存中调用；懒加载是延迟加载，按照一定的条件或者需求等到满足条件的时候再加载对应的资源
- 预加载增加了服务器压力，换来的是用户体验的提升，典型例子是在一个图片较多的网页中，如果使用了预加载就可以避免网页加载出来是时，图片的位置一片空白（图片可能还没加载出来），造成不好的用户体验；懒加载的作用减少不要的请求，缓解了服务器压力

Jquery选择器有哪些

一、基本选择器 基本选择器是jQuery中最常用也是最简单的选择器，它通过元素的id、class和标签名等来查找DOM元素。 1、ID选择器 #id 描述：根据给定的id匹配一个元素， 返回单个元素（注：在网页中，id名称不能重复） 示例：$("#test") 选取 id 为 test 的元素 2、类选择器 .class 描述：根据给定的类名匹配元素，返回元素集合 示例：$(".test") 选取所有class为test的元素 3、元素选择器 element 描述：根据给定的元素名匹配元素，返回元素集合 示例：$("p") 选取所有的

元素 4、* 描述：匹配所有元素，返回元素集合 示例：$("*") 选取所有的元素 5、selector1，selector2,...,selectorN 描述：将每个选择器匹配到的元素合并后一起返回，返回合并后的元素集合 示例：$("p,span,p.myClass") 选取所有

,和class为myClass的

标签的元素集合

二、层次选择器 三、过滤选择器 四、表单选择器（返回元素集合，使用相似）

Jquery插入节点的方法

- append() 对标的是appendChild
- appendTo() 对标的是appendChild，将调用方法的对象和参数对象做了调转。
- prepend() 作为子元素插入到开始标签之后
- prependTo()
- after() 插入到某个元素之后
- before() 插入到某个元素之前

Js的函数节流和函数防抖的区别

函数节流是指一定时间内js方法只跑一次。比如人的眨眼睛，就是一定时间内眨一次。这是函数节流最形象的解释。

函数防抖是指频繁触发的情况下，只有足够的空闲时间，才执行代码一次。比如生活中的坐公交，就是一定时间内，如果有人陆续刷卡上车，司机就不会开车。只有别人没刷卡了，司机才开车。

Get和post不同

1.GET请求参数是在URL中存在的,POST请求的参数是在请求主体中的。 2.POST的安全性要好GET请求。 3.GET请求会造成一次浏览器浏览记录，而POST请求不会有浏览记录。 4.GET请求有长度限制。POST没有长度限制。

什么是csrf攻击

CSRF（Cross-site request forgery）也被称为 one-click attack或者 session riding，中文全称是叫跨站请求伪造。 一般来说，攻击者通过伪造用户的浏览器的请求，向访问一个用户自己曾经认证访问过的网站发送出去，使目标网站接收并误以为是用户的真实操作而去执行命令。常用于盗取账号、转账、发送虚假消息等。攻击者利用网站对请求的验证漏洞而实现这样的攻击行为，网站能够确认请求来源于用户的浏览器，却不能验证请求是否源于用户的真实意愿下的操作行为。

Js数据类型的分类

JavaScript支持的数据类型可分为基本数据类型和引用数据类型（引用数据类型也称为复杂类型）。其中基本数据类型包含了数字（number）类型、字符串（string）类型、布尔（boolean）类型、未定义（undefined）类型、空（null）类型；引用数据类型就是对象类型。在 JavaScript 中，数组、函数都属于对象类型。

Js中手写一个深拷贝

```
function deepCopy(obj){
    //判断是否是简单数据类型,
    if(typeof obj == "object"){
        //复杂数据类型
        var result = obj.constructor == Array ? [] : {};
        for(let i in obj){
            result[i] = typeof obj[i] == "object" ? deepCopy(obj[i]) : obj[i];
        }
    }else {
        //简单数据类型 直接 == 赋值
        var result = obj;
    }
    return result;
}
```

什么时候用深拷贝 /浅拷贝

如何遍历一个多维数组

```
<!DOCTYPE html>
<html lang="zh-cn">
<head>
    <meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>js遍历多维数组</title>
</head>
<body>
    <script>
    //实现一个Array each方法，实现遍历多维数组
    var arr = [1,2,3,[4,5,[6,7]]];  //arr.length
    Array.prototype.each = function(fn){
        try{
            //1、目的 遍历数组的每一项
             //计数器  记录当前遍历的元素位置
            this.i ||(this.i = 0);
            //2、判断什么时候使用each核心方法
            //当数组的长度大于0的时候 && 传递的参数必须为函数
            if(this.length>0 && fn.constructor == Function){
                //循环遍历数组的每一项
                while(this.i<this.length){
                    //获取数组的每一个值
                    var e = this[this.i];  //数组的每一项
                    //如果当前元素获取到了并且当前元素是一个数组
                    if(e && e.constructor == Array){
                        //直接递归操作
                        e.each(fn);
                    }else{
                        //如果不是数组（那就是一个单个元素）
                        //把数组的当前元素传递给fn函数，并且让函数执行
                        fn.call(e,e);
                    }
                    this.i++
                }
                this.i == null;  //释放内存，垃圾回收机制回收变量
            }
        }catch(err){
                //do something
        }
        return this;
    }
    arr.each(function(item){
        console.log(item);
    })
    </script>
</body>
</html>
```

第三阶段

Vue的核心是什么

- 数据驱动和组件化

请简述你对vue的理解

- 是一套构建用户界面的渐进式的自底向上增量开发MVVM框架，Vue 的核心库只关注视图层，它不仅易于上手，还便于与第三方库或既有项目整合。通过尽可能简单的 API 实现响应的数据绑定和组合的视图组件

请简述vue的单向数据流

- 数据从父组件流向（传递）给子组件，只能单向绑定。父级 prop 的更新会向下流动到子组件中，但是反过来则不行。这样会防止从子组件意外改变父级组件的状态，从而导致你的应用的数据流向难以理解。

Vue常用的修饰符有哪些

- 按键修饰符 .up, .down, .ctrl, .enter, .space等等 语法：@click.修饰符='fn()'
- 事件修饰符 prevent修饰符：阻止事件的默认行为(submit提交表单) stop修饰符：阻止事件冒泡 capture修饰符：与事件冒泡的方向相反，事件捕获由外到内 self：只会触发自己范围内的事件，不包含子元素 once：只会触发一次

v-text与{{}}区别

- v-text与{{}}等价，{{}}叫模板插值，v-text叫指令。有一点区别就是，在渲染的数据比较多的时候，可能会把大括号显示出来，俗称屏幕闪动，为了解决这种问题，可以采用以下两种方式：①使用v-text渲染数据 ②使用{{}}语法渲染数据，但是同时使用v-cloak指令（用来保持在元素上直到关联实例结束时候进行编译），v-cloak要放在什么位置呢，v-cloak并不需要添加到每个标签，只要在el挂载的标签上添加就可以

v-on可以绑定多个方法吗

- 可以，同对象将事件包起来

```
<p v-on="{click:dbClick,mousemove:MouseClick}">v-on绑定多个方法</p>
```

Vue循环的key作用

- key的作用是为了在diff算法执行时更快的找到对应的节点，提高diff速度;key具有唯一性;vue中循环需加 :key= “唯一标识” ，唯一标识可以使item里面id index 等，因为vue组件高度复用增加key可以标识组件的唯一性，为了更好地区别各个组件key的作用主要是为了高效的更新虚拟DOM

什么是计算属性

- 顾名思义，首先它是一种属性，其次它有“计算”这个特殊性质。每次取得它的值得时候，它并不像普通属性那样直接返回结果，而是经过一系列的计算之后再返回结果。同时只要在它的当中里引用了data中的某个属性，当这个属性发生变化时，计算属性仿佛可以嗅探到这个变化，并自动重新执行

Vue单页面的优缺点

- 单页面的优点：
  - 用户体验好，快，内容的改变不需要重新加载整个页面，基于这一点spa对服务器压力较小。
  - 前后端分离。
  - 页面效果会比较炫酷（比如切换页面内容时的专场动画）。
- 单页面缺点：
  - 不利于seo。
  - 导航不可用，如果一定要导航需要自行实现前进、后退。（由于是单页面不能用浏览器的前进后退功能，所以需要自己建立堆栈管理）。
  - 初次加载时耗时多。
  - 页面复杂度提高很多。

Vuex是什么？怎么使用？在那种场景下使用

- Vuex 是一个专为 Vue.js 应用程序开发中管理的一个模式。 通过创建一个集中的数据存储，方便程序中的所有组件进行访问
- vuex只能用于单个页面中不同组件（例如兄弟组件）的数据流通。

Vue中路由跳转方式（声明式/编程式）

- 声明式：使用<router-link to=“/url” ></router-link>标签设置路由跳转 to属性用来设置跳转链接
- 编程式：使用this.$touter全局路由的push()方法进行路由跳转

```
fun(){
    this.$router.push("/index")
}
```

跨域的解决方式

Vue的生命周期请简述

vue的生命周期就是vue实例从创建到销毁的过程，一共有四个阶段。 每个阶段有两个钩子函数，一共八个钩子函数。 一、创建 1、beforeCreate：这个阶段实例已经初始化，只是数据观察与事件机制尚未形成，不能获取DOM节点（没有data，没有el） 使用场景：因为此时data和methods都拿不到，所以通常在实例以外使用 2、created：实例已经创建，仍然不能获取DOM节点（有data，没有el） 使用场景：模板渲染成html前调用，此时可以获取data和methods， 可以初始化某些属性值，然后再渲染成视图，异步操作可以放在这里

二、载入 1、beforeMount：是个过渡阶段，此时依然获取不到具体的DOM节点，但是vue挂载的根节点已经创建（有data，有el） 2、mounted：数据和DOM都已经被渲染出来了 使用场景：模板渲染成html后调用，通常是初始化页面完成后再对数据和DOM做一些操作，需要操作DOM的方法可以放在这里

三、更新 1、beforeUpdate：检测到数据更新时，但在DOM更新前执行 2、updated：更新结束后执行 使用场景：需要对数据更新做统一处理的；如果需要区分不同的数据更新操作可以使用$nextTick

四、销毁 1、beforeDestroy：当要销毁vue实例时，在销毁前执行 2、destroyed：销毁vue实例时执行

第一次页面加载会触发哪些钩子 beforeCreate、created、beforeMount、mounted

Vue生命周期的作用

- 可以在不同阶段添加自己的代码，例如，如果要通过某些插件操作DOM节点，如想在页面渲染完后弹出广告窗， 那我们最早可在mounted 中进行。

DOM渲染在那个生命周期阶段内完成

- mounted阶段

Vue路由模式hash和history，简单讲一下

- hash模式url里面永远带着#号，我们在开发当中默认使用这个模式
- history模式没有#号，是个正常的url适合推广宣传
- 考虑url的规范那么就需要使用history模式，因为当然其功能也有区别，在开发app的时候有分享页面，这个分享出去的页面就是用vue做的，把这个页面分享到第三方的app里，有的app里面url是不允许带有#号的，所以要将#号去除那么就要使用history模式，history模式还有一个问题就是，做刷新操作，会出现404错误，那么就需要和后端人配合让他配置一下apache或是nginx的url重定向，重定向到你的首页路由上。

history模式与hash模式区别

```
* url显示：hash：有#  history：没有#
* 回车刷新：hash：可以加载到hash值对应的页面 history：一般就是404掉了
```

```
* 支持版本：hash：支持低版本浏览器和ie浏览器 history：HTML5新推出的api
```

Vue路由传参的两种方式，prams和query方式与区别

- 用法上的：query要用path来引入，params要用name来引入，接收参数都是类似的，分别是this.$route.query.name和this.$route.params.name。
- url展示上:params类似于post，query更加类似于我们ajax中get传参，说的再简单一点，前者在浏览器地址栏中不显示参数，后者显示，所以params传值相对安全一些。

Vue数据绑定的几种方式

- 1：普通文本绑定，mustach语法【双大括号】 {{}} 或者 使用v-text 绑定【常用】
- 2：解释HTML标签的绑定：使用v-html绑定【慎用】
  - 2.1 注意：除非是信任的内容使用这样的方式进行数据绑定
  - 2.2 这样的绑定方式，如果内容不是被信任的，有可能会造成XSS攻击
- 3：将数据绑定到标签的属性上，使用 v-bind:属性名称=”变量” 来绑定【常用】

Vue的路由钩子函数/路由守卫有哪些

- 全局前置守卫：router.beforeEach((to，from，next)=>{})
- 全局后置钩子: router.afterEach((to, from) => {})
- 路由独享的守卫: beforeEnter（在路由配置上直接定义 ）
- 组件内的守卫:在组件中使用beforeRouteEnter(to, from, next) {}来进行进入组建前的钩子,在组件中使用beforeRouteLeave(to, from, next) {}来进行离开组件的钩子

Vue中如何进行动态路由设置？有哪些方式？怎么获取传递过来的数据？

动态路由匹配

```
*  动态路由也可以叫做路由传参
*  组件的显示内容经常会根据用户选择的内容不同来在同一个组件中渲染不同内容。那么在这个时候就需要动态路由
```

动态路径参数

```
*  使用动态路由匹配中的 动态路径参数来进行路由配置。注意：动态路径参数 以冒号：开头
```

```
    {name:"tema",path:"/index/:id/:name",component:tema}
```

绑定参数

* 路由导航绑定参数的两种方式 但是注意 params只能通过路由配置中的name属性来引用路由

//第一种格式 <router-link to="/index/参数1/参数2">index</router-link> //第二种格式 <router-link :to="{name:'tema',params:{id:'参数1',name:'参数二'}}">index</router-link> * js方式进行参数绑定

```
fun(){
    this.$router.push("/index/js参数1/js参数2");
    //第二种
    this.$router.push({name:'tema',params:{id:'js参数1',name:'js参数2'}})
}
```

获取路由传入参数

* 如果想得到路径参数那么使用$route.params.id

```
    const tema = {template:"<p>index{{this.$route.params.id}}----
{{this.$route.params.name}}</p>"}
```

* 或者是使用this实例中的this.$route.params.id进行调用

```
    created(){
        //路由信息都被挂载到this实例中去
        console.log("路由参数是: "+this.$route.params.id+"---"+this.$route.params.name)
    }
```

动态路由--知识点扩展query传参

* 路由参数不需要添加内容

```
    {name:"tema",path:"/home",component:temb}
```

路由导航绑定参数的方式

```
    <router-link to="/home?id=参数1&name=参数2">home</router-link>
<!-- 使用name引入路由 -->
    <router-link :to={name:'tema',query:{id:'js参数1',name:'js参数2'}}>home</router-link>
<!-- 使用path引入路由 -->
    <router-link :to={path:'/home',query:{id:'js参数1',name:'js参数2'}}>home</router-
link>
```

```
*  js方式进行参数绑定
```

```
    fun(){
        this.$router.push("/home?id=参数1&name=参数2");
        // 使用path引入路由
        this.$router.push({path:'/home',query:{id:'js参数1',name:'js参数2'}})
        // 使用name引入路由
        this.$router.push({name:'tema',query:{id:'js参数1',name:'js参数2'}})
    }
```

params与query区别

```
*  用法上的：query要用path来引入，params要用name来引入，接收参数都是类似的，分别是
this.$route.query.name和this.$route.params.name。
*  url展示上的：params类似于post，query更加类似于我们ajax中get传参，说的再简单一点，前者在浏览
器地址栏中不显示参数，后者显示，所以params传值相对安全一些
```

Elementui中的常用组件有哪些？请简述你经常使用的 并且他们的属性有哪些?

Vue-cli中如何自定义指令

自定义指令

```
* 除了内置指令外， 用户自定义的指令
* 局部指令定义：
```

```
    directives:{
      自定义指令的名字：{
        自定义指令钩子函数(el){
          操作逻辑
        }
      }
    },
```

自定义指令钩子函数

```
* bind：绑定指令到元素上，只执行一次

* inserted：绑定了指令的元素插入到页面中展示时调用，基本上都是操作这个钩子函数

* update：所有组件节点更新时调用

* componentUpdated：指令所在组件的节点及其子节点全部更新完成后调用

* unbind：解除指令和元素的绑定，只执行一次
```

Vue中指令有哪些

- v-model 指令（主要是用于表单上数据的双向绑定，v-model 指令必须绑定在表单元素上）
- v-show 指令（控制切换一个元素的显示和隐藏，根据表达式结果的真假，确定是否显示当前元素true表示显示该元素；false(默认)表示隐藏该元素，元素一直存在只是被动态设置了display:none）
- v-on 指令（为 HTML 元素绑定事件监听）
- v-for 指令（遍历 data 中的数据，并在页面进行数据展示）
- v-bind 指令（绑定 HTML 元素的属性，绑定多个属性不能使用简写）
- v-if指令（判断是否加载固定的内容，根据表达式结果的真假，确定是否显示当前元素，true表示加载该元素；false表示不加载该元素，元素的显示和隐藏是对Dom元素进行添加和删除）
- v-show与v-if区别：v-if有更高的切换消耗（安全性高）。v-show有更高的初始化的渲染消耗（对安全性无要求选择）
- v-else 指令（必须配合v-if使用否则无效。当v-if条件不成立的时候执行）
- v-else-if 指令（当有一项成立时执行。）
- v-text 指令（操作网页元素中的纯文本内容。{{}}是他的另外一种写法）
- v-text与{{}}区别（v-text与{{}}等价，{{}}叫模板插值，v-text叫指令。有一点区别就是，在渲染的数据比较多的时候，可能会把大括号显示出来，俗称屏幕闪动）
- v-html 指令（双大括号会将数据解释为纯文本，而非 HTML 。为了输出真正的 HTML ，你需要使用 v-html 指令）
- v-once 指令（当数据改变时，插值处的内容不会更新(会影响到该节点上的所有属性)）

Vue如何定义一个过滤器

- 全局过滤器的定义方法：位置：创建实例之前

```
Vue.filter("过滤器名称", function(val){
    //val:需要处理的值
    //逻辑代码
    //return 处理后的数据
```

```
        return val + 4;
    });
```

- 局部过滤器：只能在当前vue注册内容中使用，在vue实例中与el属性data属性同级定义

```
    filters: {
    "过滤器名字":function(val){
    return "输出内容"
    }
}
```

对vue 中keep-alive的理解

keep-alive

```
* 在上一个demo中我们不停的切换两个标签页的内容时候，会发现在练习我们中选择好的内容，切换路由之后会恢复初始化。也就是说之前的状态丢失。原因是每次切换路由的时候，Vue 都创建了一个新的 组件实例。
* 解决这个问题，我们可以用一个 <keep-alive> 元素将其路由出口包裹起来。在切换过程中将状态保留在内存中，防止重复渲染DOM，减少加载时间及性能消耗，提高用户体验性
```

keep-alive 属性

```
* 在vue 2.1.0 版本之后，keep-alive新加入了两个属性：include(包含的组件缓存) 与 exclude(排除的组件不缓存，优先级大于include) 。
```

```
    <keep-alive include="tema,temb">
        <component :is="text"></component>
    </keep-alive>
```

keep-alive 的钩子函数

```
* 这两个生命周期函数一定是要在使用了keep-alive组件之上。activated  类型：func  触发时机：keep-alive组件激活时使用；
* deactivated  类型：func  触发时机：keep-alive组件停用时调用；
```

如何让组件中的css在当前组件生效

- 在style标签中添加属性scoped

Vue生命周期一共几个阶段

实例生命周期

```
* 什么是实例的生命周期?
        - 实例在创建到销毁的一系列过程叫生命周期
    * 什么是生命周期钩子
        - 在生命周期中被自动调用的函数叫生命周期的钩子
    * 生命周期钩子函数的用途
        - 每个 Vue 实例在被创建时都要经过一系列的初始化过程—例如，需要设置数据监听、编译模板、将实例挂载到 DOM 并在数据变化时更新 DOM 等。同时在这个过程中也会运行一些叫做生命周期钩子的函数，这给了用户在不同阶段添加自己的代码的机会。
    * 钩子函数有哪些?
        - beforeCreate（创建实例）、created（创建完成）、beforeMount（开始创建模板）、mounted（创建完成）、beforeUpdate（开始更新）、updated（更新完成）、beforeDestroy（开始销毁）、destroyed（销毁完成）

        - 钩子函数的书写位置在data与methods同级位置书写。
    * 扩展设置数据请求的钩子
        - created里面，如果涉及到需要页面加载完成之后的话就用 mounted。
        - 在created的时候，视图中的html并没有渲染出来，所以此时如果直接去操作html的dom节点，一定找不到相关的元素
        - 而在mounted中，由于此时html已经渲染出来了，所以可以直接操作dom节点
    * 扩展常见问题:
        - 什么是vue生命周期?
            * Vue 实例从创建到销毁的过程，就是生命周期。也就是从开始创建、初始化数据、编译模板、挂载Dom→渲染、更新→渲染、卸载等一系列过程，我们称这是 Vue 的生命周期。
        - vue生命周期的作用是什么?
            * 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完成指定逻辑。
        - vue生命周期总共有几个阶段?
            * 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
        - 第一次页面加载会触发哪几个钩子?
            * 第一次页面加载时会触发 beforeCreate, created, beforeMount, mounted 这几个钩子
        - DOM 渲染在 哪个周期中就已经完成?
            * DOM 渲染在 mounted 中就已经完成了。
        - 简单描述每个周期?
            * beforeCreate（创建前） 在数据观测和初始化事件还未开始
            * created（创建后） 完成数据观测，属性和方法的运算，初始化事件，实例中的el属性还没有显示出来
            * beforeMount（载入前） 在挂载开始之前被调用，相关的render函数首次被调用。实例已完成以下的配置：编译模板，把data里面的数据和模板生成html。注意此时还没有挂载html到页面上。
            * mounted（载入后） 在el 被新创建的 vue.el 替换，并挂载到实例上去之后调用。实例已完成以下的配置：用上面编译好的html内容替换el属性指向的DOM对象。完成模板中的html渲染到html页面中。此过程中进行ajax交互。
            * beforeUpdate（更新前） 在数据更新之前调用，发生在虚拟DOM重新渲染和打补丁之前。可以在该钩子中进一步地更改状态，不会触发附加的重渲染过程。
            * updated（更新后） 在由于数据更改导致的虚拟DOM重新渲染和打补丁之后调用。调用时，组件DOM已经更新，所以可以执行依赖于DOM的操作。然而在大多数情况下，应该避免在此期间更改状态，因为这可能会导致更新无限循环。该钩子在服务器端渲染期间不被调用。
            * beforeDestroy（销毁前） 在实例销毁之前调用。实例仍然完全可用。
            * destroyed（销毁后） 在实例销毁之后调用。调用后，所有的事件监听器会被移除，所有的子实例也会被销毁。该钩子在服务器端渲染期间不被调用。
```

Mvvm与mvc的区别

- MVVM实现了View和Model的自动同步，也就是当Model的属性改变时，我们不用再自己手动操作Dom元素，来改变View的显示，而是改变属性后该属性对应View层显示会自动改变。MVC模式是MVP,MVVM模式的基础，这两种模式更像是MVC模式的优化改良版,他们三个的MV即Model，view相同，不同的是MV之间的纽带部分。

Vue组件中的data为什么是函数

- 如果是函数的话，因为数据是一个函数返回值的形式，所以每个组件实例可以维护一份被返回对象的独立拷贝，组件内的数据被复用后，如果改动数据，其他组件不会受到影响；如果是单纯的一个对象，组件实例共用了一份data数据，因此，无论在哪个组件实例中修改了data,都会影响到所有的组件实例。

Vue双向绑定的原理

- vue数据双向绑定是通过数据劫持结合发布者-订阅者模式的方式来实现的。 我们已经知道实现数据的双向绑定，首先要对数据进行劫持监听，所以我们需要设置一个监听器Observer，用来监听所有属性。如果属性发上变化了，就需要告诉订阅者Watcher看是否需要更新。因为订阅者是有很多个，所以我们需要有一个消息订阅器Dep来专门收集这些订阅者，然后在监听器Observer和订阅者Watcher之间进行统一管理的。接着，我们还需要有一个指令解析器Compile，对每个节点元素进行扫描和解析，将相关指令（如v-model，v-on）对应初始化成一个订阅者Watcher，并替换模板数据或者绑定相应的函数，此时当订阅者Watcher接收到相应属性的变化，就会执行对应的更新函数，从而更新视图。

Vue中组件怎么传值

- 父组件向子组件传值：在子组件中使用 props就可以实现
- 子组件向父组件传值：用自定义事件，在子组件向父组件传值时需要使用 vue 中的 $on 和 $emit ，使用$emit 可以触发 $on 中监听的事件
- 兄弟组件之间的传值：用中央事件总线Eventbus，也可以用vuex，vuex可以包含这三种情况的传值

Bootstrap的原理

- 网格系bai统的实现原理，是通过定义容器大du小，平分12份(也有zhi平分成24份或32份，但12份是最常见dao的)，再调整内外边距，最后结合媒体查询，就制作出了强大的响应式网格系统。Bootstrap框架中的网格系统就是将容器平分成12份。

如果一个组件在多个项目中使用怎么办

- 将vue组件开发成一个包，然后提交到git上进行统一管理。

槽口请简述

- 用来混合父组件的内容与子组件自己的模板

Watch请简述

- 可以监听模型数据 当模型数据改变的时候就会触发,watch初始化的时候不会运行,只有数据被改变之后才会运行,当需要在数据变化时执行异步或开销较大的操作时，watch这个方式是最有用的

计算属性与侦听器的区别

- 当watch监听的值发生改变就会被调用，watch可以在数据变化时做一些异步处理或者开销大的操作,计算属性是计算依赖的值，当依赖的值发生改变才会触发。

mvvm框架是什么？它和其它框架（jquery）的区别是什么？哪些场景适合？

- 一个model+view+viewModel框架，数据模型model，viewModel连接两个
- 区别：vue数据驱动，通过数据来显示视图层而不是节点操作。
- 场景：数据操作比较多的场景，更加便捷

Vue首屏加载慢的原因，怎么解决的，白屏时间怎么检测，怎么解决白屏问题

- 原因：页面在打包后如果不进行相关配置会导致资源文件特别的大，一次想要全部加载完成会特别的耗时。
- 解决方案：路由懒加载，gzip压缩，异步加载组件，服务端渲染
- 白屏时间：可使用 Performance API 时，白屏时间 = firstPaint - performance.timing.navigationStart;不可使用 Performance API 时，白屏时间 = firstPaint - pageStartTime;
- 解决白屏：为了避免白屏，可以进来第一件事，快速生成一套loading的渲染树（前端骨架屏），服务器的SSR骨架屏所提高的渲染就是避免了客户端再次单独请求数据，而不是样式

Vue双数据绑定过程中，这边儿数据改变了怎么通知另一边改变

Vuex流程

- 页面通过mapAction异步提交事件到action。action通过commit把对应参数同步提交到mutation。mutation会修改state中对于的值。 最后通过getter把对应值跑出去，在页面的计算属性中，通过mapGetter来动态获取state中的值

- Vuex的异步操作流程 1.在组件中通过this.$store.dispath("actions名字")调用一个actions进行异步操作 2.在vuex的acions创建一个方法进行异步操作 他的形参就是store对象 把请求来的数据通过 commit（）传递给mutations 3.mutations把传递过来的数据 进行state的修改

Vuex怎么请求异步数据

- 结合action

```
// 异步操作全部放到actions
 actions: {
   axioslink(store){
       demoapi.demo().then((ok)=>{
         console.log(ok.data.data.commentList)
         // 把请求来的数据给mutations进行state的修改
         store.commit("uparr",ok.data.data.commentList)
       })
   }
}
```

Vuex中action如何提交给mutation的

Route与router区别

- $router是VueRouter的一个对象，router的实例对象，这个对象中是一个全局的对象，他包含了所有的路由包含了许多关键的对象和属性。举例：history对象
- $route是一个跳转的路由对象，每一个路由都会有一个route对象，是一个局部的对象，可以获取对应的name,path,params,query等

vuex有哪几种状态和属性

- 有五种,分别是State , Getter , Mutation , Action , Module (就是mapAction)

vuex的State特性是?

- stae就是存放数据的地方，类似一个仓库，特性就是当mutation修改了state的数据的时候，他会动态的去修改所有的调用这个变量的所有组件里面的值（ 若是store中的数据发生改变，依赖这个数据的组件也会发生更新 ）

vuex的Getter特性是?

- getter用来获取数据，mapgetter经常在计算属性中被使用

vuex的Mutation特性是?

- 在vuex中如果想进行数据的修改 必须必须必须必须必须在mutations中进行操作，调用mutations需要使用commit()来进行调用

vuex 是什么？怎么使用？哪种功能场景使用它，vuex的优势

- Vuex是vue框架中状态管理。
- 使用场景：兄弟之间组件有大量通信的
- 优势：状态管理工具 核心是响应式的做到数据管理, 一个页面发生数据变化。动态的改变对应的页面

路由懒加载

- 懒加载简单来说就是延迟加载或按需加载，即在需要的时候的时候进行加载。
- 为给客户更好的客户体验，首屏组件加载速度更快，解决白屏问题。做的一些项目越来越大。vue打包后的js文件也越来越大，这会是影响加载时间的重要因素。当构建的项目比较大的时候，懒加载可以分割代码块，提高页面的初始加载效率。

常用的懒加载方式有两种：即使用vue异步组件懒加载 和ES中的import

- ES 提出的import(推荐使用):const HelloWorld = （）=>import('需要加载的模块地址')
- vue异步组件懒加载-- resolve :component：resolve=>(require(["引用的组件路径"],resolve))

v-for与v-if优先级

- 首先不要把v-if与用在同一个元素上，原因：v-for比v-if优先，如果每一次都需要遍历整个数组，将会影响速度，尤其是当之需要渲染很小一部分的时候。v-for 比 v-if 具有更高的优先级