## css-title

#### **说说你对盒子模型的理解？**

是指网页中的每个元素都可以看作是一个盒子，这个盒子由内容区域、内边距、边框和外边距四部分组成，它决定了元素在页面中的大小、位置、边框和内外边距等样式属性。

1. 内容区域：即元素的实际内容，比如文本、图片、视频等。
2. 内边距：指内容区域和边框之间的空白区域，可以用padding属性来控制。
3. 边框：围绕内容和内边距的线条，可以用border属性来设置样式、宽度和颜色。
4. 外边距：指边框和相邻元素之间的空白区域，可以用margin属性来控制。

box-sizing 属性定义引擎应该如何计算一个元素的总宽度和总高度

1. content-box 默认值，元素的 width/height 不包含padding，border，与标准盒子模型表现一致
2. border-box 元素的 width/height 包含 padding，border，与怪异盒子模型表现一致
3. inherit 指定 box-sizing 属性的值，应该从父元素继承

#### **谈谈对BFC的理解？**

是一个独立的块级渲染区域，用来规定内部块级元素的布局及其与外部元素的关系。

BFC的形成有多种方式，其中包括：

1. 根元素或包含根元素的元素；
2. 浮动元素（float不为none）；
3. 绝对定位元素（position为absolute或fixed）；
4. 行内块元素（display为inline-block）；
5. 表格单元格（display为table-cell）；
6. overflow不为visible的块级元素；

BFC具有以下特性：

1. 内部元素垂直排列：BFC内部的块级元素会垂直排列，一个元素的边框会和其它元素的边框重叠，但是不会和外部元素的边框重叠。
2. 包含浮动元素：BFC会将浮动元素包含在内部，避免浮动元素对外部元素的影响。
3. 阻止边距重叠：在同一个BFC中，相邻元素的垂直边距会发生重叠，但是不同BFC之间的元素不会发生边距重叠。
4. 自适应高度：BFC会自适应其内部元素的高度，使得元素的高度可以自动撑开。

BFC是CSS中非常重要的概念，它可以用来解决一些布局问题，比如清除浮动、阻止边距重叠等。在实际的开发过程中，我们可以通过设置元素的display、position和overflow等属性来创建BFC。

1. **CSS3新特性有哪些？常用哪些？**
2. 选择器（属性选择器、伪类选择器、伪元素选择器、子元素选择器）
3. 盒子模型（box-sizing）
4. 阴影（box-shadow、text-shadow）
5. 渐变（linear-gradient()、radial-gradient()）
6. 动画（@keyframes、animation ）
7. 转换（transform ）、过渡（transition）、字体（@font-face）
8. 多列布局（column-count、column-width）、媒体查询（@media）
9. **动画是怎么做的？举例做过什么样的动画。**

CSS动画使用关键帧（Keyframes）来描述元素在一段时间内的动画效果。关键帧描述了在动画持续时间内元素应该呈现的不同状态，浏览器会在这些状态之间自动进行平滑的过渡。

1. **Transform有哪些属性？scal( )方法用来做什么的？**

用于对元素进行变换操作，可以实现平移、缩放、旋转和倾斜等多种效果。

常用属性：

1. translate：平移元素，即将元素沿着X轴和Y轴移动一定距离。例如：translateX(50px)，将元素沿着X轴向右移动50个像素，translateY(-30px)，将元素沿着Y轴向上移动30个像素
2. scale：缩放元素，即增大或减小元素的尺寸。例如：scale(2)，将元素沿着X轴和Y轴同时放大2倍，scaleX(0.5)，将元素沿着X轴缩小为原来的一半
3. rotate：旋转元素，即将元素围绕中心点旋转一定角度。例如：rotate(45deg)，将元素顺时针旋转45度
4. skew： 倾斜元素，即将元素沿着X轴或Y轴倾斜一定角度。例如：skewX(30deg)，将元素沿着X轴倾斜30度
5. matrix：通过一个2D变换矩阵来变换元素。这个属性需要指定六个值，分别代表变换矩阵中的6个元素

以上是 transform 常用的属性，还有一些其他的属性可以用来实现更复杂的变换效果，例如 perspective、rotateX、rotateY、rotateZ 等

1. **css如何实现一个三角形？**

使用 CSS 的 border 属性。通过指定一个元素的宽度、高度和边框颜色来创建一个三角形

我们将 border-top 和 border-bottom 属性的宽度设置为 50px，并将它们的颜色设置为透明。这将创建一个高度为 50px 的等腰三角形。我们还将 border-right 属性的宽度设置为 100px，这将创建一个底边长度为 100px 的三角形。最后，我们将 border-right 属性的颜色设置为红色，这将使三角形的右侧变为红色

1. **页面是怎么做适配的？**
2. 使用响应式布局（Responsive Web Design，RWD）：响应式布局是指通过使用 CSS3 媒体查询（@media）来自适应不同的屏幕尺寸和分辨率。在响应式布局中，页面中的元素可以使用百分比、em 或 rem 等单位来设置宽度和高度等样式属性，从而实现页面的自适应
3. 使用弹性布局（Flexible Box Layout，Flexbox）：弹性布局是指通过使用 CSS3 弹性盒模型（display: flex）来实现页面的自适应。在弹性布局中，页面中的元素可以根据父容器的大小自动调整位置和大小，从而实现页面的自适应
4. 使用流式布局（Fluid Layout）：流式布局是指通过设置元素的宽度为百分比来实现页面的自适应。在流式布局中，页面中的元素可以根据浏览器窗口的大小自动调整位置和大小，从而实现页面的自适应
5. 使用视口（Viewport、vw\vh）：视口是指浏览器中用来显示网页的区域，通过设置视口的大小和缩放等属性，可以实现页面的自适应。在移动设备中，可以使用 meta 标签来设置视口的大小和缩放等属性，从而实现页面的自适应
6. **移动端布局方案有哪些？分别介绍原理以及优缺点。（高频）**
7. 基于百分比的布局：通过将元素的宽度和高度设置为相对于父元素的百分比来实现布局。这种布局方案的原理比较简单，而且能够适应不同尺寸的设备，但是缺点是对于复杂的布局效果不够灵活，而且需要手动计算百分比值
8. 基于 rem 的布局：通过将元素的宽度和高度设置为相对于根元素 html 的字体大小 font-size 的倍数来实现布局。这种布局方案的优点是能够自适应不同的设备，而且使用相对单位可以避免在不同设备上显示效果不一致的问题。缺点是需要在代码中设置根元素的字体大小，而且不同设备的像素密度可能会影响字体大小的计算(REM 值 = 像素值 ÷ 根元素字体大小)
9. 基于 flex 布局：通过使用 display: flex 和相关的属性来实现弹性盒子布局。这种布局方案的优点是非常灵活，可以实现各种复杂的布局效果，而且支持响应式设计。缺点是需要比较熟悉 flex 布局的相关属性，而且一些旧版的浏览器可能不支持这种布局方案
10. 基于网格布局：通过使用 display: grid 和相关的属性来实现网格布局。这种布局方案的优点是可以实现复杂的布局效果，而且支持响应式设计。缺点是需要比较熟悉网格布局的相关属性，而且一些旧版的浏览器可能不支持这种布局方案

基于百分比和基于 rem 的布局方案适用于相对简单的布局效果，而基于 flex 和网格布局方案适用于相对复杂的布局效果

1. **em rem px有什么区别？**

都是用来定义文本和其他元素的大小的单位

1. px（像素）是相对于显示器的分辨率而言的，是一个固定的单位，不能缩放。通常情况下，1个像素等于1个物理像素点，但是在高分辨率的屏幕上，1个像素可能对应多个物理像素点，这会导致元素在高分辨率屏幕上显示得更小
2. em（字号）是相对于父元素的字号而言的，是一个相对单位。如果一个元素的字号为 16px，那么它的 1em 就等于 16px。如果子元素的字号为 1.5em，那么它的实际字号就是 24px（1.5 \* 16px）
3. rem（根字号）是相对于根元素（通常是 <html> 元素）的字号而言的，也是一个相对单位。与 em 不同的是，rem 的值不会随着父元素字号的变化而变化。因此，使用 rem 单位可以更好地控制整个页面的布局和大小
4. **雪碧图(**CSS Sprites**)的作用和功能？**

是将多个小图标或图片合并到一张大图上，通过 CSS 的 background-position 属性和尺寸来显示指定的图片区域，从而达到减少网页加载时间和HTTP请求数的目的

1. 减少 HTTP 请求：每一个图片文件都需要一次 HTTP 请求，而每次 HTTP 请求都需要额外的时间和资源。通过将多个小图标或图片合并到一张大图上，可以减少 HTTP 请求的次数，提高网页加载速度
2. 提高网页性能：减少 HTTP 请求不仅可以提高网页加载速度，还可以减少服务器的负载和网络带宽的占用，提高网站的性能和稳定性
3. 优化图片加载方式：将多个小图标或图片合并到一张大图上，可以使用 CSS 的 background-position 属性和尺寸来显示指定的图片区域，这种方式比单独加载每个小图标或图片更加高效，可以减少图片文件的大小和下载时间
4. 管理图片资源：使用雪碧图可以将多个小图标或图片合并到一张大图上，便于管理和维护图片资源，减少文件数量和混乱
5. **Flex:1 是什么意思？ 分别介绍每个属性的作用？控制放大缩小的属性是哪个？**

设置 flex-grow: 1 flex-shrink: 1 flex-basis: 0%

1. flex-grow: 1; - 它指定项目在弹性容器中的放大比例，如果有剩余空间，该项目将按比例放大，此处 "flex: 1" 表示放大比例为 1 默认值是 0，如果没有指定放大比例，则项目不会在空间充足时放大
2. flex-shrink: 1; - 它指定项目在弹性容器中的缩小比例，当空间不足时，该项目将按比例缩小，此处 "flex: 1" 表示缩小比例为 1 默认值是 1，如果没有指定缩小比例，则项目在空间不足时会按比例缩小
3. flex-basis: 0%; - 它指定项目在弹性容器中的初始大小，"flex: 1" 表示初始大小为 0% 默认值是 auto，如果没有指定初始大小，则会根据项目的内容自动计算出初始大小

控制一个元素的放大缩小属性的属性是 flex-grow 和 flex-shrink

1. **定位（position）有哪几种方式？及其区别？**
2. “static”（静态定位）：默认值。元素按照文档流排列，不受 top、bottom、left、right 等属性的影响。
3. “relative”（相对定位）：元素的位置相对于它原来的位置进行偏移。通过 top、bottom、left、right 属性可以控制元素的偏移量。
4. “absolute”（绝对定位-脱离文档流）：元素的位置相对于它最近的一个已经定位的父元素进行定位。如果没有已定位的父元素，则相对于文档（<html>）进行定位。通过 top、bottom、left、right 属性可以控制元素的偏移量
5. “fixed”（固定定位-脱离文档流）：元素的位置相对于浏览器窗口进行定位，即使页面滚动，元素的位置也不会改变。通过 top、bottom、left、right 属性可以控制元素的偏移量。
6. “sticky”（粘性定位）：元素在滚动到指定位置时变为固定定位，否则为相对定位。通过 top、bottom、left、right 属性可以控制元素的偏移量。
7. **元素垂直居中方式有哪些？**
8. 使用 vertical-align 属性：这个属性可以设置行内元素和表格单元格元素的垂直对齐方式。比如，将图片垂直居中可以设置 vertical-align: middle。
9. 使用 line-height 属性：将行高设置为容器高度一样可以使内部文本垂直居中。这种方法只适用于单行文本。
10. 使用 display: flex 属性：使用 Flexbox 布局，通过设置容器的 align-items 属性为 center 可以实现子元素的垂直居中。
11. 使用 display: grid 属性：使用 CSS Grid 布局，通过设置容器的 align-items 属性为 center 可以实现子元素的垂直居中。
12. 使用绝对定位（position: absolute）和 transform 属性：将要垂直居中的元素使用 position: absolute 定位，然后使用 transform: translateY(-50%) 将元素上移一半高度的距离实现垂直居中。这种方法需要知道元素的高度。
13. 使用 CSS 表格布局：将容器设置为 display: table，然后将要垂直居中的元素设置为 display: table-cell，并使用 vertical-align: middle 属性实现垂直居中
14. **元素水平垂直居中的方式有哪些？**
15. 使用 display: flex 和 justify-content: center， align-items: center 属性：使用 Flexbox 布局，将容器设置为 display: flex，然后设置 justify-content: center 属性实现水平居中，设置 align-items: center 属性实现垂直居中
16. 使用绝对定位（position: absolute）和 transform 属性：将要水平垂直居中的元素使用 position: absolute 定位，然后使用 top: 50% 和 left: 50% 将元素定位到容器的中心位置，最后使用 transform: translate(-50%, -50%) 将元素上移一半高度和左移一半宽度的距离实现水平垂直居中
17. 使用 CSS Grid 布局：使用 CSS Grid 布局，通过设置容器的 place-items: center 属性可以实现子元素的水平垂直居中
18. 使用 CSS 表格布局：将容器设置为 display: table，然后将要水平垂直居中的元素设置为 display: table-cell，并使用 vertical-align: middle 和 text-align: center 属性实现水平垂直居中
19. 使用绝对定位和margin属性(需要知道父元素的宽度和高度)：将元素设置为position: absolute;top:50%;left:50%;margin-top:-50px;margin-left:-50px;
20. **清除浮动有哪些方式？**

为了使元素脱离文档流而浮动，往往会给我们带来一些问题，如高度塌陷、重叠等，这时我们就需要进行清除浮动

以下是几种清除浮动的方式：

1. 使用空元素清除浮动：在浮动元素的最后一个元素后插入一个空的 div 元素，并为其设置 clear: both，这样就可以清除前面浮动元素的影响
2. 使用 overflow 属性清除浮动：给浮动元素的容器设置 overflow: auto 或 overflow: hidden，因为浮动元素不会溢出容器的高度，所以设置了 overflow 属性的容器可以自动适应其子元素的高度
3. 使用父元素的伪类清除浮动：给浮动元素的容器设置 ::after 伪类，并设置 clear: both 属性，从而清除浮动元素的影响
4. 使用 display 属性清除浮动：给浮动元素的容器设置 display: table，可以清除浮动元素的影响，并自动适应子元素的高度
5. **定义一个50%宽度的元素，大小跟随页面变化并且始终是居中的，这种的怎么实现？**
6. Flex
7. Grid
8. 绝对定位
9. **遇到过移动端兼容性问题？怎么解决的？**
10. iOS 上 fixed 定位的元素会出现跳动问题。

解决方法是将 fixed 定位的元素改为 absolute 定位，然后使用 JavaScript 实现滚动时改变元素位置的效果

1. iOS 上 input[type="number"] 控件无法自定义样式。

解决方法是使用 JavaScript 或 CSS 实现自定义样式，并使用其他类型的控件（如 text）来代替 input[type="number"] 控件

1. iOS 上 font-size 设置为 16px 以下时会自动放大。

解决方法是使用 meta 标签中的 viewport 属性来禁用缩放功能，或者将 font-size 设置为 16px 以上

1. iphone输入框默认内阴影

解决方法用-webkit-appearance:none解决

1. IOS端input光标高度，在安卓手机上显示没有问题，但是在苹果手机上当点击输入的时候，光标的高度和父盒子的高度一样

解决方案：高度height和行高line-height内容用padding撑开

微信浏览器常见兼容性问题

1. 填写表单信息弹出键盘会把页面顶上去，键盘消失的时候页面不自动下来

解决方案：使用onblur方法在表单失去焦点的时候让页面滚动到最顶部

1. Input框默认样式

解决方案：设置透明背景，设置appearance:none去除系统默认appearance的样式

1. 音视频不自动播放

解决方法：使用微信的JS-SDK，先引入微信的JS-SDK，然后在wx.ready中调用play方法播放

1. **移动端1像素问题是怎么解决的？**
2. 使用 viewport + rem 或者 vw/vh 来设置宽度、高度、字体大小等，将 CSS 像素与设备像素一一对应，避免了 1 像素问题。但是需要注意的是，使用 rem 或者 vw/vh 有可能会导致布局错乱等问题
3. 使用 transform 缩放来解决 1 像素问题。将元素缩放为 0.5 倍，然后再缩放回来，即可避免 1 像素问题
4. 使用 CSS3 边框渐变来模拟 1 像素边框。将边框样式设置为渐变色，颜色从浅到深，可以使得边框看起来更加平滑
5. 使用第三方 CSS 框架或者插件，如 border.css、postcss-px-to-viewport 等。这些框架或插件会自动将 CSS 像素转换为设备像素，从而避免了 1 像素问题
6. **使用canvas过程中有遇到过什么兼容性问题吗？怎么解决的？**

性能问题：Canvas 通常是使用 CPU 进行绘制，因此在某些设备上可能会存在性能问题，例如在移动设备上绘制大量的复杂图形可能会导致卡顿或者明显的延迟

解决方法：在编写代码时应该尽量避免频繁的绘制操作，可以使用缓存技术

1. **重排和重绘的区别？并举例说明。**

重排/回流（reflow）是指浏览器重新计算元素的位置和大小，然后将它们放在正确的位置上。重排会改变元素的几何属性，包括宽度、高度、位置、边距和排列顺序等。由于重排需要重新计算布局，所以它的代价比较高，可能会引起页面卡顿或者闪烁等问题

重绘（repaint）是指浏览器重新绘制元素的样式，但不会改变它们的几何属性。重绘只是重新绘制元素的背景、边框和文本等样式，但不会改变它们的位置和大小。重绘的代价比较低，因为它只需要重新绘制样式而不需要重新计算布局

1. **项目中音视频是怎么做的？Audio video是怎么使用的？ios andriod遇到过什么兼容性问题吗？**

使用 HTML5 中的 <audio> <video> 标签来实现音视频的播放和控制，也可以使用第三方库来实现音视频的处理和渲染。比如，可以使用howler.js库来实现音频的播放和处理，使用video.js库来实现视频的播放和处理

其中 controls 属性表示添加控制面板，让用户可以控制音视频的播放、暂停、音量等

1. 音视频格式支持：不同的平台支持不同的音视频格式，需要在不同平台上使用不同的格式来保证兼容性。例如，iOS设备通常支持MP3和AAC格式的音频，而Android设备则更倾向于支持OGG和MPEG-4格式的音频。对于视频，iOS设备通常支持H.264格式的视频，而Android设备则支持更多的格式，包括H.264、VP8和VP9等
2. 自动播放：iOS设备通常不允许在网页中自动播放音视频，必须用户手动触发播放操作。而在Android设备上，自动播放行为则更加灵活
3. 浏览器支持：不同的浏览器可能存在不同的支持程度，需要进行适配和测试。例如，在iOS设备上，Safari浏览器通常是最好的选择，因为它原生支持音视频功能
4. 事件触发：iOS和Android平台上对于音视频播放过程中的事件触发机制不同。例如，iOS设备上的音视频播放完成事件需要监听ended事件，而Android设备则需要监听ended和complete两个事件
5. **CSS选择器优先级？**
6. 内联样式（行内样式）：使用style属性直接为元素指定样式，它的优先级最高
7. ID选择器（ID selectors）：使用#符号指定，比如#example。一个ID选择器的优先级高于多个类选择器、属性选择器或标签选择器的组合
8. 类选择器、属性选择器和伪类选择器（Class selectors, Attribute selectors, and Pseudo-class selectors）：使用“.”符号指定类选择器，使用“[ ]”符号指定属性选择器，使用“:”符号指定伪类选择器。它们的优先级相等，但比标签选择器优先级高
9. 标签选择器和伪元素选择器（Tag selectors and Pseudo-element selectors）：指定元素类型的选择器，比如“div”，以及使用“::”符号指定伪元素选择器，比如“::before”。它们的优先级最低。
10. !important 例外规则：当在一个样式声明中使用一个 !important 规则时，此声明将覆盖任何其他声明。使用 !important 是一个坏习惯，应该尽量避免，因为这破坏了样式表中的固有的级联规则 使得调试找 bug 变得更加困难了。当两条相互冲突的带有 !important 规则的声明被应用到相同的元素上时，拥有更大优先级的声明将会被采用（根据来源决定：内联、内部样式表、外部样式表）

如果有多个选择器具有相同的权重，那么最后定义的样式将覆盖先前定义的样式，这也称为“层叠”

1. **19.CSS实现100%-60PX**

使用calc()函数、calc()函数允许在CSS属性中使用数学表达式，从而动态计算属性值

height: calc(100% - 60px);

这样，元素的高度就会被限制在60px以内，并占据其父元素的全部宽度。请注意，在使用calc()函数时，数学表达式中的运算符和括号都需要使用空格进行分隔，以避免被CSS解析器误解

1. **实现0.5像素有哪些实现方法？**
2. 使用 transform：transform: scale(0.5);
3. 使用 box-shadow：box-shadow: 0 0 0.5px rgba(0, 0, 0, 0.5);
4. 使用 border-image：border-image: linear-gradient(to bottom, black 50%, transparent 50%);
5. 使用 SVG：可以通过使用 SVG 图像来实现 0.5 像素的效果

## JavaScript-title

1. **JS有哪些数据类型？区别？检测数据类型方法？**

7种数据类型

原始类型：

1. Number（数字）: 用于表示数字，包括整数和浮点数
2. String（字符串）: 用于表示文本
3. Boolean（布尔值）: 用于表示真或假
4. undefined: 表示未定义的值，通常用于未赋值的变量
5. null: 表示空值，通常用于已赋值但没有值的变量
6. Symbol（符号）: 一种新的数据类型，用于表示唯一标识符
7. BigInt（大整数）: 一种新的数据类型，用于表示更大范围的整数

引用类型：

1. Object（对象）
2. Function (函数)
3. Array (数组)
4. RegExp (正则表达式)

区别：

原始类型值是不可变的（immutable），一旦创建，就无法修改。值存放在栈中，按值访问。

引用类型其地址（指针）存在栈中，数据存放在堆中，通过栈中的这个地址访问堆内的数据，因为引用类型的大小会改变，所以不能将其存放在栈中，否则会降低变量查询速度

检测数据类型方法：

1. typeof：用于检测基本数据类型
2. instanceof：检测一个对象是否属于一个指定类（对象）的实例。返回值是一个布尔值。
3. Array.isArray：用于检测一个值是否为数组类型。返回值是一个布尔值
4. Object.prototype.toString.call：可以检测任何数据类型的数据类型，返回值是一个字符串
5. **怎么检测数组？数组合并有哪些方法？数组怎么去重？（高频）**

检测数组：Array.isArray()、Object.prototype.toString.call()、instanceof

数组合并：concat()、展开运算符、push()方法

数组去重：Set 对象、filter() 方法、reduce() 方法

1. **数组有哪些方法？数组怎么转字符串？**
2. push()：在数组末尾添加一个或多个元素
3. pop()：删除数组末尾的元素，并返回该元素
4. shift()：删除数组第一个元素，并返回该元素
5. unshift()：在数组开头添加一个或多个元素
6. slice()：从数组中截取一段，返回一个新的数组
7. splice()：在数组中插入或删除元素，并返回被删除的元素
8. concat()：将多个数组合并为一个数组
9. join()：将数组中的所有元素转为字符串，并使用指定的分隔符连接起来
10. toString()：将数组转为字符串，并返回该字符串
11. reverse()：反转数组中的元素
12. sort()：对数组中的元素进行排序
13. indexOf()：返回数组中指定元素的索引位置
14. lastIndexOf()：返回数组中指定元素最后出现的索引位置
15. forEach()：对数组中的每个元素执行指定操作

数组转字符串方法

join()、toString()、两种方法作用相同，都可以将数组中的所有元素转为字符串，并使用指定的分隔符连接起来

1. **字符串有哪些方法？字符串怎么转数组？**
2. charAt()：返回指定索引位置的字符
3. concat()：将多个字符串合并为一个字符串
4. includes()：判断字符串中是否包含指定的子字符串，返回布尔值
5. indexOf()：返回指定子字符串在字符串中第一次出现的索引位置
6. lastIndexOf()：返回指定子字符串在字符串中最后一次出现的索引位置
7. match()：返回一个数组，包含字符串中匹配某个正则表达式的部分
8. replace()：替换字符串中的指定部分，并返回新的字符串
9. slice()：截取字符串的一部分，返回一个新的字符串
10. split()：将字符串分割成多个子字符串，并返回一个数组
11. substr()：截取字符串的一部分，返回一个新的字符串
12. substring()：截取字符串的一部分，返回一个新的字符串
13. toLowerCase()：将字符串中的所有字符转为小写
14. toUpperCase()：将字符串中的所有字符转为大写
15. trim()：去除字符串两端的空格字符

字符串转数组

split()、将一个字符串按照指定的分隔符分割成多个子字符串，并将这些子字符串存储在一个数组中

1. **map和object区别？（数据类型）**

都是用于存储键值对的数据结构

1. 键类型：Map可以使用任何类型的值作为键，包括对象、数组、函数等，而Object的键只能是字符串或符号
2. 键值对数量：Map没有键值对数量的限制，而Object的键值对数量是有限制的，受制于JavaScript引擎和内存的限制
3. 键值对顺序：Map保留插入顺序，而Object不保证键值对的顺序
4. 遍历性能：Map的性能在大型数据集上比Object更好，因为Map是基于哈希表实现的，可以快速查找键值对，而Object的查找速度取决于键的数量
5. 方法：Map提供了一些额外的方法，如set()、get()、has()、delete()、clear()，这些方法使得对Map的操作更加方便。而Object则提供了一些基本的方法，如Object.keys()、Object.values()、Object.entries()、Object.assign()等
6. 内存消耗：Map比Object占用更多的内存空间，因为Map内部维护了额外的哈希表等数据结构

因此，在选择使用Map或Object时，需要根据具体的需求进行判断。如果需要存储大量键值对、需要按照插入顺序遍历或者需要使用非字符串或符号类型的键，那么使用Map可能更合适。如果只需要存储少量的键值对、不需要按照插入顺序遍历或者键只能是字符串或符号类型，那么使用Object也是一个不错的选择

1. **对象合并有哪些方法？**
2. 展开运算符（ES6+）
3. Object.assign()
4. 循环遍历(hasOwnProperty判断是否是自有属性，确认对象obj1中不存在该属性) for-in
5. 第三方库：underscore、lodash(\_assign(obj1,obj2)、\_extend(obj1,obj2))
6. **map和foreach的区别？**

相同点：都是常见的数组方法，它们都可以用来遍历数组并对数组的每个元素进行操作

不同点：

1. 返回值：map()方法会返回一个新的数组，而forEach()方法没有返回值，仅仅是对每个元素进行操作
2. 对原数组的影响：map()方法不会改变原数组，而forEach()方法会对原数组进行修改
3. 使用场景：map()方法通常用于对数组中的元素进行变换或映射，例如将数组中的每个元素都加倍，而forEach()方法则通常用于对数组中的元素进行遍历并执行一些副作用操作，例如输出数组中的每个元素或在每个元素上执行一些函数操作
4. 回调函数参数：map()和forEach()方法都接受一个回调函数作为参数，但这两个回调函数的参数不同。map()方法的回调函数有三个参数：当前元素、当前索引和原数组；而forEach()方法的回调函数只有两个参数：当前元素和当前索引

如果需要对数组进行变换并得到一个新的数组，可以使用map()方法；如果只是需要对数组进行遍历并执行一些操作，可以使用forEach()方法

1. **js是如何实现继承的？有哪些继承方式？**

通过原型链来实现的

1. 原型继承：通过将一个对象的原型指向另一个对象来实现继承
2. 构造函数继承：通过在子类构造函数中调用父类构造函数来实现继承
3. 组合继承：通过原型继承和构造函数继承的结合来实现继承
4. 原型式继承：通过浅拷贝一个对象来实现继承
5. 寄生式继承：通过在原型式继承的基础上添加方法或属性来实现继承
6. 寄生组合式继承：通过将组合继承中的父类实例替换为原型式继承的对象来实现继承，从而避免调用两次父类构造函数
7. **深拷贝、浅拷贝，分别有哪些方法？原理？**

深拷贝：

1. JSON.parse(JSON.stringify(object)): 将对象转换成 JSON 格式的字符串，再将字符串转换成对象。这种方式可以复制对象及其嵌套的所有子对象，但是会忽略对象中的 undefined、symbol 和 function 类型的属性，而且对于包含循环引用的对象会抛出错误
2. 递归遍历：递归遍历对象及其嵌套的所有子对象，创建一个新的对象并复制每个属性的值。这种方式可以复制对象及其嵌套的所有子对象，但是会增加函数调用栈的开销，可能会导致栈溢出的问题
3. Lodash 库中的 cloneDeep() 方法，lodash.cloneDeep(object)这种方式可以复制对象及其嵌套的所有子对象，而且对于包含循环引用的对象也可以正确地处理
4. jQuery 库中的 extend() 方法，jQuery.extend(true, {}, object)这种方式可以复制对象及其嵌套的所有子对象，而且对于包含循环引用的对象也可以正确地处理

浅拷贝：

1. Object.assign(target, source): 将 source 对象的属性复制到 target 对象中，返回 target 对象。这种方式只复制对象的第一层属性，不会复制对象的嵌套属性
2. Object.create(obj)：创建一个新对象，并将 obj 对象作为新对象的原型。这种方式可以用来复制对象，但是会将 obj 对象的属性和方法设置为新对象的原型，而不是直接复制
3. Array.prototype.slice()：返回一个数组的浅拷贝，可以用来复制数组。
4. 扩展运算符(...): 在数组或对象字面量中使用扩展运算符可以将一个对象展开成一系列对象字面量或数组元素。这种方式只复制对象的第一层属性，不会复制对象的嵌套属性

原理：

1. 浅拷贝是指复制对象的引用，新的变量只是原对象的一个别名，修改其中任意一个变量的属性都会影响到另外一个变量。浅拷贝的实现原理通常是通过遍历对象的属性，将属性值复制给新对象，但是对于对象的嵌套属性，只会复制引用而不是值
2. 深拷贝是指复制对象及其嵌套的所有子对象的值，新的变量和原对象没有任何关联，修改其中任意一个变量的属性不会影响到另外一个变量。深拷贝的实现原理通常是通过递归遍历对象及其嵌套的所有子对象，创建一个新的对象并复制每个属性的值。对于包含循环引用的对象，需要特殊处理，避免死循环
3. **说一下对this的理解？改变this指向有哪些方法？**

js的一个关键字，用来引用当前执行上下文中的对象，this的值在运行时动态绑定，取决于调用函数的方式和上下文

全局作用域中的 this：

1. 在全局作用域中，this 指向全局对象 window

函数中的 this：

1. 如果函数是作为普通函数调用，this 指向全局对象 window
2. 如果函数是作为对象的方法调用，this 指向调用该方法的对象
3. 如果函数使用 call() 或 apply() 方法调用，this 指向作为第一个参数传递的对象
4. 如果函数是作为构造函数使用，this 指向新创建的对象
5. 如果使用箭头函数，this 指向箭头函数所在的上下文对象，是在创建函数时就已经确定，不能通过 call()、apply() 或 bind() 方法来修改

改变this指向方法

call(context，arg1,arg2,arg3) ：传递的参数是一个个逐个列举出来的，如果你知道函数需要传递多少个参数，而且这些参数都已经确定，使用call

apply(context，[arg1,arg2,arg3]) ：传递的参数是以数组形式传递的,不知道函数需要传递多少个参数，或者需要使用数组来传递参数,使用apply

bind()：创建一个新的函数，其中 this 的值被绑定到指定的对象。绑定后的函数可以在以后的任何时候调用，而不会改变 this 的指向

1. **什么是作用域？**

作用域是指在程序中定义变量时所指定的一定范围，该范围内变量可被访问或使用

JavaScript 中的作用域可以分为全局作用域和函数作用域，对象不构成单独的作用域

1. 全局作用域是指在整个 JavaScript 程序中都可访问的变量，这些变量定义在函数外部。全局作用域中的变量可以被任何函数访问
2. 函数作用域是指函数内部可以访问的变量，这些变量只在函数内部可见，称为局部变量。函数作用域中的变量只有在函数内部才能被访问，而在函数外部是不可访问的
3. 在 ES6 中引入了块级作用域，通过 let 和 const 关键字可以创建块级作用域。块级作用域中定义的变量只在当前代码块中可见，不会被外部作用域访问到
4. **闭包是什么？用来做什么？会带来什么问题？怎么解决？**

指有权访问另一个函数作用域内的变量的函数,即使这个函数已经执行完毕，它的内部变量仍然可以被访问

实现私有变量和私有方法、模块化代码、异步编程

闭包会带来一些问题，例如内存泄漏和变量共享。在创建闭包时，内部函数会引用外部函数的变量，导致这些变量不能被垃圾回收机制释放，从而造成内存泄漏。另外，多个闭包之间可能会共享同一个变量，导致变量的值被不同的闭包修改，产生不可预期的结果

解决方案：

1. 尽量避免创建不必要的闭包，如果不需要访问外部函数的变量，可以不使用闭包
2. 及时释放闭包，可以在函数执行完毕之后，将闭包赋值为 null，释放内存
3. 使用立即执行函数，可以在函数内部创建闭包，并立即释放闭包，避免闭包带来的内存泄漏问题
4. 使用模块化代码，可以避免变量共享的问题，每个模块都有自己的作用域，不会和其他模块产生冲突
5. **原型是什么？什么是原型链？**

在 JavaScript 中，每个对象都有一个原型对象。它包含了一些通用的属性和方法，这些属性和方法可以被其它对象继承和共享。每个对象都有一个 \_\_proto\_\_ 属性，指向其原型对象

JavaScript 中的原型继承是通过原型链实现的。每个对象都有一个指向其原型对象的 \_\_proto\_\_ 属性，形成了一个层层嵌套的结构。当一个对象需要访问某个属性或方法时，它首先会在自身的属性列表中查找，如果找不到，则会去它的原型对象中查找。如果在原型对象中还找不到，则继续在原型对象的原型对象中查找，直到找到该属性或方法或者原型链的末端为止，这时没有找到就会返回null，因为Object.prototype.\_\_proto\_\_没有prototype属性

通过使用原型链，我们可以实现对象的继承、共享属性和方法等功能。在 JavaScript 中，所有的对象都是通过原型链来实现继承的，这使得 JavaScript 的对象模型变得非常灵活和强大

1. **对正则的理解？怎么使用？**

正则表达式是一种用来匹配字符串模式的工具，常用于文本处理、搜索和替换等操作中。

通过RegExp对象来实现的，两种方式创建：字面量方式和构造函数方式，可以用于字符串的搜索和替换

字面量方式：let regex = /pattern/flags;

构造函数方式：let regex = new RegExp(pattern, flags);

常用方法：

1. test()：测试一个字符串是否符合某个模式，返回布尔值
2. exec()：匹配一个字符串中的模式，并返回匹配结果的数组，如果没有匹配到，则返回null
3. match()：和exec()类似，但是是在字符串对象上调用，返回匹配结果的数组
4. replace()：替换一个字符串中的模式，返回替换后的新字符串
5. search()：在一个字符串中搜索某个模式，返回模式的索引位置，如果没有找到，则返回-1
6. **ajax原生请求是怎么实现的？**

XMLHttpRequest对象

实现步骤：

1. 创建 XMLHttpRequest 对象：new XMLHttpRequest()
2. 指定请求方法和 URL：open() 方法指定请求的方法和 URL
3. 设置请求头部信息：setRequestHeader()方法
4. 发送请求：send() 方法
5. 处理响应：onreadystatechange 事件监听请求的状态变化，使用 status 和 responseText 属性获取响应的状态码和响应数据
6. **什么是跨域？解决方式有哪些？如何实现？**

跨域本质是浏览器基于同源策略的一种安全手段，同源：（协议相同、域名相同、端口相同）

当一个请求url的协议、域名、端口三者之间的任意一个与当前页面url不同即为跨域

解决方式：

1. JSONP：通过动态创建script标签，将JSON数据作为callback函数的参数返回到客户端
2. CORS：通过设置Access-Control-Allow-Origin头部，允许指定域名的请求访问资源
3. 代理服务器：在同一域名下搭建代理服务器，将请求代理到目标域名，再将响应返回给客户端
4. WebSocket协议：建立一条基于Socket的双向通信管道，可以在不同域名之间进行实时通信

Vue项目一般使用Proxy

1. **Cookie，LocalStorage，sessionStorage三者区别？如何使用？**

都是本地存储

Cookie:

1. 存储大小限制：4KB左右
2. 有效期：可以设置一个固定的过期时间，在过期时间之前都有效；也可以设置一个会话级别的 Cookie，这种 Cookie 在用户关闭浏览器后就会失效
3. 作用范围：可以设置作用于整个域名或者某个子域名，也可以在服务器端设置为 HttpOnly，禁止客户端 JavaScript 访问
4. 主要用途：用于记录用户状态和行为，例如登录状态、购物车、网站偏好等

LocalStorage:

1. 存储大小限制：一般为 5MB 左右
2. 有效期：永久存储，除非用户手动删除或清除浏览器缓存
3. 作用范围：只能在同源的页面之间共享数据，即只能在同一个协议、主机名、端口号的页面之间共享数据
4. 主要用途：用于本地存储数据，例如表单数据、用户偏好、用户行为数据等

sessionStorage:

1. 存储大小限制：一般为 5MB 左右
2. 有效期：只在用户当前会话中有效，即当用户关闭浏览器标签页或窗口后数据就会被清除
3. 作用范围：只能在同源的页面之间共享数据，即只能在同一个协议、主机名、端口号的页面之间共享数据
4. 主要用途：用于临时保存数据，例如在表单多页面间共享数据、页面间数据传递等
5. **localStorage和vuex区别？**

都是在前端开发中用于存储数据的工具

localStorage 是浏览器提供的一种本地存储机制，可以用来在用户的本地计算机上存储和获取键值对数据，数据的存储是永久的，直到被手动清除。localStorage 可以用于存储一些简单的应用数据，比如用户的偏好设置、应用的配置信息等。localStorage 只能存储字符串类型的数据，如果需要存储其他类型的数据，需要先将其转换为字符串

vuex 是一个专门为 Vue.js 应用程序开发的状态管理库。它提供了一种集中式存储管理应用的所有组件的状态的方法，使得状态的变化可以被追踪和管理。vuex 中的状态可以被组件读取和修改，但它们只能在应用程序的内存中存储，不会被持久化到本地存储中。vuex 可以用于存储复杂的应用状态，比如登录状态、购物车数据等

1. **说下事件循环？宏任务、微任务有哪些？执行顺序是什么样的？**

是JavaScript的一种运行机制，主要用来处理异步事件。JavaScript运行时会创建一个主线程和一个事件循环队列，将所有任务按照执行顺序加入到事件循环队列中。主线程执行完同步任务后，会不断地从事件循环队列中取出任务执行，直到队列为空为止

宏任务（Macrotask）：包括script、setTimeout、setInterval、setImmediate、I/O、UI rendering等任务

微任务（Microtask）：包括Promise、then()、catch()、Object.observe、MutationObserver等任务

执行顺序：

1. 执行完主线程的同步任务，检查微任务队列，如果有微任务则依次取出并执行，直到微任务队列为空为止
2. 执行当前宏任务中的异步任务，将产生的微任务加入到微任务队列中
3. 如果当前宏任务的异步任务执行完后，产生了新的宏任务，则将新的宏任务加入到事件循环队列的末尾
4. 重复步骤1-3，直到事件循环队列为空

微任务的优先级高于宏任务，即微任务的执行优先级高于宏任务的执行。这是因为微任务可以立即执行，而宏任务必须等待当前宏任务中的异步任务执行完后再执行。因此，在代码编写中需要注意避免过多的宏任务嵌套，以提高Web应用程序的响应速度

1. **new一个对象过程，发生了什么？**
2. 创建一个新对象
3. 将该对象的原型链接到构造函数的prototype属性上
4. 将构造函数的作用域赋给新对象（因此this关键字指向新对象）
5. 执行构造函数中的代码（给新对象添加属性和方法）
6. 返回新对象
7. **网页常见攻击方式？如何解决？**

XSS（Cross-site scripting）跨站脚本攻击：攻击者通过注入恶意脚本代码到Web页面中，使得用户在访问该页面时，执行这些脚本并造成一定的危害，例如窃取用户Cookie、密码等敏感信息。解决方法：

1. 对用户输入的数据进行过滤和转义，避免插入脚本代码
2. 使用HTTP Only属性限制Cookie只能在服务器端读取，防止窃取Cookie

CSRF（Cross-Site Request Forgery）跨站请求伪造攻击：攻击者通过伪造用户请求，实现对用户账户、密码等敏感信息的盗取、修改或删除。解决方法包括：

1. 在用户操作时，增加CSRF Token等验证机制，防止伪造请求
2. 在敏感操作时，增加验证码等二次确认机制，确保用户是本人操作

点击劫持攻击：攻击者通过在Web页面上盖住一个透明的按钮或链接，引导用户误操作，从而达到攻击的目的。解决方法包括

1. 通过设置X-Frame-Options HTTP头，禁止在iframe中加载页面，从而避免被嵌入到其他页面中
2. 在Web页面中使用JavaScript脚本检测当前页面是否被嵌入到其他页面中

iframe注入攻击：攻击者通过在Web页面中嵌入iframe标签，引导用户点击或提交敏感信息，实现攻击目的。解决方法包括：

1. 对于不必要的iframe，可通过设置X-Frame-Options HTTP头禁止其被嵌入到其他页面中
2. 对于必须使用iframe的页面，要通过JavaScript脚本控制其行为，避免被攻击者利用
3. **ES5和ES6的区别？ES6常用的有哪些？**
4. 语法差异：ES6引入了一些新的语法特性，例如let和const关键字、箭头函数、模板字符串、默认参数值、解构赋值、类和模块等
5. 内置对象扩展：ES6扩展了一些内置对象的功能，例如Array、Object、String等，引入了一些新的方法，例如Array.from()、Array.of()、Object.assign()、String.startsWith()等
6. Promise对象：ES6引入了Promise对象，用于管理异步操作，使得异步编程更加简单和可读
7. Iterator和Generator：ES6引入了迭代器（Iterator）和生成器（Generator）的概念，使得对数据集合的遍历和操作更加灵活和高效
8. 其他新特性：ES6还引入了一些其他新特性，例如Symbol类型、Map和Set集合、Proxy和Reflect对象、模块化等

es6常用：

let 和 const 声明变量、箭头函数、模板字符串、解构赋值、Promise、类和继承、模块化：import 和 export、迭代器和生成器、展开运算符、默认参数、Rest 参数、Map 和 Set 数据结构、Proxy 和 Reflect、Symbol、async 和 await

1. **箭头函数和普通函数区别？箭头函数怎么获取传进来的未知参数呢？**

语法、作用域和 this 的指向不同

箭头函数没有自己的 this，它会继承上一层作用域的 this，而普通函数的 this 指向在函数调用时根据调用方式不同而不同

通过 rest 参数语法（...args）来获取传递进来的未知个数的参数

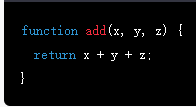
1. **箭头函数this指向？**

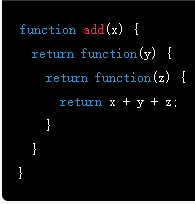
指向箭头函数被创建时上层作用域中的this，也就是说，箭头函数内部的this指向是固定的

因为箭头函数的 this 值是在函数定义时确定的，而不是在函数调用时动态确定的

1. **函数柯里化是什么？写一个例子？**

函数柯里化是一种将接受多个参数的函数转换为接受单个参数并且返回一个新函数的技术。这个新函数可以继续接受参数，返回另一个新函数，直到最后一个参数被传入并返回最终结果





这种技术可以使函数更加灵活和可复用，因为我们可以部分应用函数，只传递一部分参数，而不必传递所有参数。它还可以用于函数组合和函数式编程

1. **promise是什么？解决了什么问题？怎么实现一个promise呢？**

是JS的异步编程的一种解决方案

解决了回调函数嵌套过多、代码可读性差、错误处理困难等问题

1. 创建Promise对象：使用Promise构造函数创建一个Promise对象，构造函数的参数是一个函数，这个函数接受两个参数(resolve,reject)，它们分别是成功和失败的回调函数
2. 执行异步操作：在Promise对象中，执行异步操作，使用setTimeout模拟异步操作
3. 处理Promise结果：在异步操作结束后，将结果传递给resolve或reject函数，Promise对象会根据结果自动调用对应的回调函数
4. **promise和async await有什么区别？**

基于Promise 的一种语法糖，用于简化 Promise 的使用和处理。

通过使用 async 关键字来修饰函数，在函数内部使用 await 关键字来等待异步操作完成，并返回异步操作的结果

Promise 需要在 then 方法中传入回调函数来处理异步操作的结果

1. **Promise.then的原理是什么？**

.then()方法本质上是将回调函数添加到Promise对象的回调队列中，等待异步操作完成后执行。如果Promise对象已经完成，则回调函数会立即执行；否则回调函数会等待Promise对象的状态变为fulfilled或rejected后再执行

1. **说一下es6模块化？**
2. 静态引入：ES6 模块化的引入是静态的，在编译阶段就会确定模块的依赖关系，而不是在运行时根据代码动态加载
3. 只读接口：ES6 模块化中，导出的接口是只读的，不允许修改导出的值，确保了模块之间的隔离性
4. 默认导出：ES6 模块化中，一个模块可以通过 export default 关键字来导出一个默认值，这个默认值可以被其他模块使用
5. 命名导出：ES6 模块化中，可以通过 export 关键字来导出一个或多个具名的值，其他模块可以通过 import 关键字来引入这些具名的值

ES6 模块化需要在支持的浏览器或者 Node.js 环境中使用 import 和 export 关键字来进行模块的导入和导出。同时，在浏览器环境中，需要使用 <script type="module"> 标签来加载模块化脚本

1. **说一下对es6 class的理解？**

语法糖，它提供了一种更简单、更易读的方式来定义 JavaScript 中的对象类型使用 class 定义的类，可以包含 constructor 构造函数、属性、方法和 getter/setter 等成员，而且可以继承其他类（extends）

1. **typescript是什么？用来做什么？项目中有用到吗？说一下泛型？**

JavaScript 的超集。引入类型系统来提高代码的可靠性和可维护性，可以在编码时发现类型相关的错误，避免了一些常见的运行时错误。此外，TypeScript 还支持面向对象编程、模块化、装饰器等高级语言特性，使得代码的组织和管理更加清晰和规范

特点：

1. 类型系统：TypeScript 强制变量、参数、函数的类型定义，支持静态类型检查，可以提前发现类型错误
2. 类和接口：TypeScript 支持类和接口的定义，使得代码更加面向对象化，提高了代码的可读性和可维护性
3. 泛型：TypeScript 支持泛型编程，可以提高代码的复用性和可扩展性
4. 编译时检查：TypeScript 在编译时会对代码进行检查和转换，以生成符合 ECMAScript 标准的代码，可以兼容所有主流浏览器和运行环境

用来做什么：

TypeScript 主要是用来提高代码的可靠性和可维护性的

泛型：

泛型是一种在编程语言中实现参数化类型的技术。在定义泛型时，使用一个或多个类型变量来代替具体的数据类型，这些类型变量可以在后续的代码中被实际的数据类型替换，泛型通常使用尖括号（<>）来标识。

优点在于可以提高代码的复用性和可扩展性，同时还可以使代码更加类型安全。泛型可以在很多场景下使用，例如数组、函数、类等

1. **ajax和axios区别？**
2. Ajax是一种基于原生XMLHttpRequest对象的技术，而Axios是基于Promise的HTTP客户端库
3. Ajax只支持XML和JSON格式的数据传输，而Axios支持更多格式，如二进制、文本、JSON等
4. Axios可以在浏览器和Node.js中使用，而Ajax只能在浏览器中使用
5. Axios可以拦截请求和响应，并且可以全局配置，而Ajax只能局部配置
6. **什么是防抖和节流？有什么区别？如何实现？**

**防抖（debounce）**，事件被触发 n 秒后再执行回调函数，如果在这 n 秒内又触发了该事件，则重新计时。也就是说，防抖的作用是确保在一定时间内只执行一次回调函数，从而减少回调函数的执行次数。比如说，在用户输入搜索关键词时，我们通常会对输入框进行防抖处理，避免用户在快速输入时频繁触发搜索请求。

**节流（throttle）**，事件被触发后的一段时间内只执行一次回调函数。也就是说，节流的作用是确保一定时间内只执行一次回调函数，从而降低回调函数的执行频率。比如说，在页面滚动时，我们通常会对滚动事件进行节流处理，避免滚动过程中频繁触发回调函数，导致页面性能下降。

区别：防抖是在事件触发后一定时间内执行回调函数，如果在这个时间内再次触发了事件，则重新计时。节流是在事件触发后的一定时间内执行一次回调函数，然后在该时间内再次触发事件时不执行回调函数，直到时间过去后重新开始计时。

原理：防抖的实现原理是利用定时器，每次事件触发时先清除之前的定时器，然后重新设置一个新的定时器，在一定时间后执行回调函数。节流的实现原理是利用定时器，每次事件触发时先判断定时器是否存在，如果不存在，则设置一个定时器，在一定时间后执行回调函数。如果定时器已经存在，则不执行任何操作，直到定时器时间过去后重新设置定时器。

1. **jsBrige是什么？用来做什么？怎么使用的？**

是一种实现不同编程语言之间相互通信的技术

通常用于不同环境之间的交互，例如在前端和后端之间进行通信，或在前端 JavaScript 与原生移动应用程序代码之间进行通信

定义一个JavaScript对象或方法，将需要暴露给后端的方法作为对象的属性或方法。通过调用window.来调用后端代码。由于jsBrige技术涉及到跨越不同的环境和安全性问题，使用时需要谨慎处理，避免出现安全漏洞和其他问题

## Vue-title

1. **Vue原理是什么？说一下MVVM？钩子函数有哪些？**

数据驱动、组件化、模板编译、生命周期、插件机制

MVVM是一种架构模式（软件设计模式），它将应用程序分为三个部分：模型（Model）、视图（View）和视图模型（ViewModel）

1. beforeCreate：实例刚被创建，数据观测和事件配置之前被调用
2. created：实例已经创建完成，数据观测和事件配置已经完成
3. beforeMount：在挂载开始之前被调用，此时模板编译已经完成
4. mounted：实例挂载完成后被调用，此时DOM已经渲染完成
5. beforeUpdate：数据更新时调用，但是DOM还没有重新渲染
6. updated：数据更新且DOM重新渲染完成后调用
7. beforeDestroy：实例销毁之前调用
8. destroyed：实例销毁后调用
9. **Vue双向绑定原理是什么？有什么缺陷？**

采用数据劫持结合发布者订阅者模式的方式来实现响应式，通过object.defineProperty来劫持数据的getter，setter,在数据变化时发送消息给订阅者，从而触发相应的回调来更新视图。

object.defineProperty 作用：用来定义对象属性

响应式原理：

1. 获取属性值会触发getter方法
2. 设置属性值会触发setter方法
3. 在setter方法中调用修改dom的方法

object.defineProperty缺点

1. 一次性递归到底开销很大，如果数据很大，大量的递归导致调用栈溢出
2. 不能监听对象的新增属性和删除属性
3. 当监听的下标对应的数据发生改变时,无法正确的监听数组的方法
4. **Vue2和Vue3的区别？Vue3生命周期有哪些？**

主要区别：

1. 响应式原理（Object.defineProperty 和 Proxy）
2. 生命周期（setup 和 vue2生命周期+”on”）
3. 多个根节点(fragment)
4. 组合式API->Composition API(同一逻辑的内容写到一起，增强代码可读性、内聚性，还提供了较为完美的逻辑复用性方案)
5. 异步组件（suspense:允许程序在等待异步组件加载完成前渲染兜底的内容，如 loading ）
6. Teleport(可将部分DOM 移动到 Vue app 之外的位置。比如项目中常见的 Dialog 弹窗)
7. Diff 算法优化（虚拟 DOM 上增加 patchFlag 字段，帮助 diff 时区分静态节点，以及不同类型的动态节点。一定程度地减少节点本身及其属性的比对）
8. 打包优化（更小的体积，优化了全局API，Vue.nextTick、Vue.set、Vue.observable）
9. TypeScript 支持

Vue3生命周期：

1. setup() :开始创建组件之前，在beforeCreate和created之前执行，创建的是data和method
2. onBeforeMount() : 组件挂载到节点上之前执行的函数
3. onMounted() : 组件挂载完成后执行的函数
4. onBeforeUpdate(): 组件更新之前执行的函数
5. onUpdated(): 组件更新完成之后执行的函数
6. onBeforeUnmount(): 组件卸载之前执行的函数
7. onUnmounted(): 组件卸载完成后执行的函数
8. onActivated(): 被 keep-alive 缓存的组件激活时调用
9. onDeactivated(): 被 keep-alive 缓存的组件停用时调用
10. **Proxy是操作对象还是操作对象的属性呢？**

是用来操作对象的，包括读取、修改、添加、删除等操作。它可以拦截对象的所有基本操作，比如获取属性值、设置属性值、删除属性等，而且可以对这些操作进行自定义处理

1. **说一下vue的发布订阅模式？**

发布订阅模式（Publish/Subscribe Pattern）也叫做观察者模式（Observer Pattern），是一种常用的组件通信方式。它基于一个主题（Subject）和多个观察者（Observer）之间的关系，当主题的状态发生变化时，所有观察者都会收到通知并进行相应的处理

在 Vue 中，发布订阅模式被称为事件总线（Event Bus），它是一个全局的 Vue 实例，用于在组件之间传递事件和数据。事件总线通过 Vue 实例的 $on()、$emit() 和 $off() 方法实现了订阅、发布和取消订阅的功能

1. **Vue组件通信方式/传值？**

父子组件通信：使用 props 和 events 来实现。父组件通过 props 属性向子组件传递数据，子组件通过 $emit() 方法触发事件，从而将数据传递回父组件

子触发父：在子组件中，使用this.$parent访问父组件实例，然后使用$refs属性访问父组件中的方法

父触发子：通过this.$refs.child获取子组件实例，然后调用子组件中的同名方法

兄弟组件通信：借助一个共同的父组件来实现。父组件作为中介，通过 props 属性将数据传递给子组件 A，子组件 A 再通过 $emit() 方法将数据传递给父组件，父组件再通过 props 属性将数据传递给子组件B

跨级组件通信：祖先组件通过 provide 来提供数据，子孙组件通过 inject 来注入数据，从而实现跨级组件之间的通信

非父子组件通信：Event Bus/Vuex。事件总线是一个空的 Vue 实例，可以在任何组件中通过 $emit() 方法触发事件，从而将数据传递给其他组件。需要注意的是，事件总线应该在组件的父级或者根级中创建，以便于管理和维护

Vuex状态管理：用于解决大型应用中组件之间共享状态的问题。通过创建一个全局的 store，任何组件都可以通过 this.$store 访问到该 store，并且可以在 store 中定义 state、mutations、actions、getters 等属性来管理应用的状态。通过 commit() 方法触发 mutations 来修改状态，通过 dispatch() 方法触发 actions 来异步处理业务逻辑

### inheritAttrs：默认为true，父作用域的不被认作props的属性作为普通的HTML特性显示在子组件根元素上。改为false，会禁止这个默认行为。通过实例属性 $attrs ,可以将这些特性生效，且可以通过v-bind 绑定到子组件的非根元素上。$listeners(包含了父作用域中不含.native修饰器的事件监听器，它可以通过 v-on="$listeners" 传入内部组件)

1. **父子组件生命周期执行顺序？**

父组件挂载

父beforeCreate->父created->父beforeMount->子beforeCreate->子created->子beforeMount->子mounted->父mounted

父组件更新

父beforeUpdate->子beforeUpdate->子updated->父updated

父子组件异步传值的坑：子组件的生命周期只会执行一次，但是当子组件渲染的时候父组件的数据还没接受完就会造成子组件没有任何内容渲染。

解决方法：给子组件添加渲染条件，使用v-if，当父组件数据接收完毕后在渲染子组件。在子组件中添加watch监听，当父组件数据传输过来时，改变原有的默认数据，重新渲染页面。

1. **组件data为什么是函数？**

因为Vue组件在创建时可能会被多次使用，如果data是一个对象，则每个组件实例都会共享相同的数据对象。这会导致一个组件的数据更改会影响到所有其他使用相同data对象的组件

1. **computed和watch区别？**

都是用来观察数据变化

1. computed是基于依赖的缓存。只有当计算属性所依赖的数据发生变化时，计算属性才会重新计算，否则它会直接返回缓存的结果。一般用来处理一些需要频繁使用但不经常更新的数据
2. watch则是在数据发生变化时立即执行回调函数，无论这个数据有没有被使用，一般用来处理一些需要异步处理或需要进行复杂逻辑的数据
3. **Vuex是什么？有哪些状态值？**

Vuex是一个专门为Vue.js设计的状态管理库。它可以用来集中管理Vue应用程序中的状态，并使这些状态的变化可预测和可调试

1. state：Vuex使用单一状态树来管理应用程序的所有状态，而state就是这个状态树中的根节点
2. getters：可以看作是state的计算属性，用于从state中派生出一些状态
3. mutations：用于修改state中的状态。必须是同步函数
4. actions：类似于mutations，但是可以包含异步操作，并且可以通过mutations来修改state中的状态
5. modules：允许将Vuex状态树分解成多个模块，每个模块可以拥有自己的state、getters、mutations和actions

在Vuex中，state包含了应用程序的所有状态值。这些状态值可以是任何JavaScript类型，包括数字、字符串、数组、对象等等。在Vue组件中，可以通过Vuex提供的mapState辅助函数或this.$store.state来访问和修改Vuex中的状态值

1. **VueRouter是什么？传参方式？有哪几种路由守卫？beforeEach原理？**

Vue.js官方的路由管理器，用于管理Vue应用程序中的路由，它可以实现路由的跳转、传参和保护等功能

Vue Router提供了以下三种传参方式

Params（动态路由、命名路由）：动态路由：需要在路由定义中使用path属性，并在路径中使用:符号来指定参数的位置，path: '/users/:id'；命名路由：通过在路由定义中给路由命名，然后在程序中使用name属性来传递参数；

query(查询参数)：URL后面添加?key=value的方式来传递参数，this.$router.push({ path: '/user', query: { id: '123' }})

Vue Router提供了三种路由守卫

1. 全局前置守卫（beforeEach）：在跳转之前被调用，可以用来验证用户是否已登录或拥有足够的权限等
2. 组件内的守卫（beforeRouteEnter、beforeRouteUpdate和beforeRouteLeave）：在组件内定义的路由守卫，可以用来控制组件的渲染和销毁，以及检查路由参数和状态等
3. 全局后置钩子（afterEach）：在路由跳转完成后被调用，可以用来记录用户的行为日志等

全局前置守卫（beforeEach）是Vue Router的核心之一。它会在每次路由跳转之前被调用，可以用来拦截路由、验证用户身份等。其基本原理是，在路由跳转之前，Vue Router会按照注册的路由守卫顺序依次执行beforeEach函数，如果在其中某个守卫函数中调用了next(false)，则路由跳转被取消；如果调用了next()，则路由跳转继续进行。如果最后一个守卫函数执行完毕，且没有调用next(false)，则路由跳转被允许

在beforeEach函数中，可以通过to和from参数获取当前跳转的路由信息和上一个路由信息，可以通过next函数控制路由跳转，可以传入一个参数来指定跳转的路径或命名路由，也可以不传参，表示路由跳转继续进行。可以在beforeEach函数中执行异步操作，例如从服务器端获取用户权限信息等。但需要注意的是，在执行异步操作时，需要保证调用next函数的时机和顺序，否则可能会导致路由跳转被中断或者多次执行

1. **vue路由跳转方式有哪些？**

router-link（声明式路由）

使用<router-link>组件实现导航，它会自动渲染成一个<a>标签，并且会监听点击事件，通过给其添加 to 属性，然后使用路由器进行跳转、<router-link to="/home">Go to Home</router-link>

$router.push(编程式路由)

跳转到指定url路径，并想history栈中添加一个记录，点击后退会返回到上一个页面、this.$router.push('/home')

this.$router.replace(编程式路由)

跳转到指定url路径，history栈中不会有记录，点击返回会跳转到上上个页面 (直接替换当前页面)、this.$router.replace('/home')

this.$router.go(n)

向前或者向后跳转n个页面，n可为正整数或负整数，类似 window.history.go(n)

1. **路由原理是什么？history 和hash有什么区别？怎么实现的？自己有实现过吗？**

Vue 路由的原理是基于浏览器的 history API 另外也支持使用 Hash（默认）模式来实现，通过设置 Vue Router 实例的 mode 属性来指定路由模式

1. URL 格式不同：Vue 路由的 URL 格式为 /path/to/page，而 Hash 模式的 URL 格式为 /#/path/to/page
2. 兼容性不同：Vue 路由使用的是 HTML5 的 history API，需要浏览器支持该 API，因此在一些旧版本的浏览器上可能会出现兼容性问题。而 Hash 模式不需要浏览器支持任何特定的 API，因此具有更好的兼容性
3. 美观度不同：由于 Hash 模式的 URL 中包含 # 号，因此相对于 Vue 路由的 URL 来说，Hash 模式的 URL 不太美观
4. SEO 友好性不同：由于 Hash 模式的 URL 中不包含实际的路径信息，因此相对于 Vue 路由的 URL 来说，Hash 模式的 URL 不太 SEO 友好

通过 history.pushState() 和 history.replaceState() 方法，可以在不刷新页面的情况下改变浏览器的 URL 地址，并将新的 URL 地址添加到浏览器的历史记录中。当用户点击浏览器的后退或前进按钮时，浏览器会根据历史记录中的 URL 地址进行页面跳转，而不是向服务器发起新的请求

1. **用户登录态失效（长时间没有操作，刷新页面），重新登录成功后，重定向到之前的页面怎么做？再点浏览器返回按钮呢？**
2. 使用 Vue Router 的导航守卫。当用户在未登录状态下访问需要登录的页面时，可以使用 beforeEnter 导航守卫将用户重定向到登录页面。在用户登录成功后，可以在登录成功的回调函数中，通过 next 方法将用户重定向回之前的页面
3. 在登录页面中使用 $route.query.redirect 属性来记录用户之前访问的页面，然后在登录成功后将用户重定向回该页面

点返回按钮：

当您点击浏览器的返回按钮时，浏览器将返回到您的登录页面，而不是返回到之前的页面。这是因为在您登录并重定向回个人中心页面时，浏览器已经将该页面添加到历史记录中。当您点击浏览器的返回按钮时，浏览器会返回到您的登录页面

如果您希望在点击浏览器的返回按钮时返回到之前的页面，您可以使用 HTML5 的 history API，在用户登录并重定向回个人中心页面时，将该页面添加到浏览器的历史记录中，不使用浏览器默认的历史记录管理方式。这样，当用户点击浏览器的返回按钮时，浏览器将返回到您之前访问的个人中心页面

window.history.replaceState({ page: '/dashboard' }, '', '/dashboard'); // 将个人中心页面添加到浏览器的历史记录中

1. **nextTick()是什么？原理？**

是 Vue.js 提供的一个异步方法，它的作用是在 DOM 更新完成后执行回调函数。具体来说，当我们修改了 Vue.js 组件的数据后，Vue.js 会异步地去更新组件的 DOM，而 nextTick() 可以让我们在 DOM 更新完成后再执行一些操作，例如获取 DOM 元素的位置和大小等

利用了浏览器的事件循环机制，将回调函数放到一个微任务队列中，在下一个事件循环周期中执行

1. **单页应用 多页应用区别？优缺点？单页应用seo怎么做？**

单页应用（SPA）是指在一个页面内部通过 Ajax 和 JavaScript 等技术动态加载页面内容，从而实现页面的无刷新切换。在单页应用中，页面的切换不会导致浏览器的刷新，从而能够提升用户体验，同时也能减少服务器的压力

多页应用（MPA）则是指每个页面都是独立的，每次页面的跳转都会导致浏览器的刷新，从而加载新的页面内容。在多页应用中，每个页面都需要加载一次 HTML、CSS 和 JavaScript 等资源，相对于单页应用来说，多页应用的性能要差一些

单页应用优点：

1. 更好的用户体验：无需重新加载页面，快速响应用户操作
2. 较小的服务器压力：由于无需重新加载页面，服务器只需返回数据，减少了网络流量和服务器压力
3. 更容易实现应用的复杂交互和动画效果

单页应用的缺点：

1. 首屏加载时间较长：由于需要加载所有的 JavaScript 和 CSS 文件，首屏加载时间会比较长
2. 对 SEO 不友好：由于单页应用只有一个 HTML 页面，搜索引擎难以识别页面内容，对于 SEO 不友好
3. 对于低性能设备不友好：较老的设备或网络较差的用户，可能会遇到性能问题

优化：

1. 在服务器端渲染：服务器端渲染可以让搜索引擎识别页面内容，并提高网页的加载速度和 SEO
2. 静态站点生成器：使用静态站点生成器，可以生成静态 HTML 页面，让搜索引擎能够更好地理解页面内容
3. 为每个页面添加描述：为每个页面添加 title 和 meta 描述等信息，让搜索引擎更好地理解页面内容
4. **vue中的key是起什么作用？**

用于给每个节点设置唯一的标识，以便 Vue.js 能够识别节点的身份，从而更高效地渲染组件

1. **说一下diff算法？**

Vue.js 通过 Virtual DOM 和 diff 算法来实现高效的 DOM 更新。在每次数据变化时，Vue.js 会先生成一个新的 Virtual DOM 树，然后将其与旧的 Virtual DOM 树进行比较，找出两者之间的差异，并将差异应用到真实的 DOM 树中

1. 首先，对比新旧 Virtual DOM 树的根节点，如果它们不相同，则直接替换整个 DOM 树。这个过程可以通过比较节点的标签名和关键属性等信息来完成，比如标签名、key 属性等
2. 如果根节点相同，Vue.js 会依次比较新旧 Virtual DOM 树中的子节点，找出差异。这个过程是递归进行的，从 Virtual DOM 树的根节点开始，依次比较它们的子节点
3. 在比较子节点时，Vue.js 会先比较它们的 key 属性。如果两个子节点的 key 属性相同，则它们很可能是同一个节点，Vue.js 就会进行精细化的比较。否则，它们可能是不同的节点，Vue.js 就会将旧的节点删除，再插入新的节点
4. 在比较同一个节点时，Vue.js 会依次比较它们的标签名、属性、文本内容等信息。如果这些信息有变化，则需要更新真实的 DOM 节点
5. 在比较子节点时，Vue.js 会使用一些启发式算法来尽可能地减少比较次数，比如设置一些规则和限制条件，以避免不必要的比较
6. **keep-alive是什么？用来做什么？自定义写过吗？举例项目中的使用场景？**

是 Vue.js 内置的一个组件

它用于缓存组件实例，可以在组件切换时保留它们的状态，避免频繁的销毁和重新创建，提升组件渲染的性能，还提供钩子函数可以让我们控制缓存的行为，例如 activated 和 deactivated 钩子函数，它们会在组件被激活和停用时调用，我们可以在这些钩子函数中进行一些逻辑处理

使用场景：

1. 缓存有状态的组件，例如表单、列表等，避免因为组件的频繁销毁和重新创建而导致状态的丢失
2. 提高动态路由的性能，对于一些数据量比较大的页面，使用 <keep-alive> 可以避免每次进入页面都重新加载数据
3. 在一些需要复用的组件中使用 <keep-alive>，例如 modal 弹窗等。这样可以避免每次打开 modal 都需要重新创建组件，提升性能和用户体验
4. **看过Vue源码吗？说一下你觉得做的比较好的地方？**
5. 数据劫持：Vue.js 使用 Object.defineProperty() 实现了对数据的劫持，实现了双向数据绑定的功能。这种劫持数据的方式非常巧妙，可以在数据发生变化时，自动触发更新视图的操作，大大简化了代码的编写
6. Virtual DOM：Vue.js 使用 Virtual DOM 优化了 DOM 更新的效率，避免了频繁的 DOM 操作。通过对 Virtual DOM 的抽象，Vue.js 实现了一种高效的 DOM 更新机制，同时也为跨平台开发提供了支持
7. 组件化：Vue.js 将 UI 组件化，使得开发者可以将代码划分为独立的组件，提高了代码的可复用性和可维护性。组件化的实现方式也非常巧妙，通过将组件的模板、数据和行为等信息打包在一起，实现了一种高度封装的组件开发模式
8. 生命周期：Vue.js 定义了一套完整的组件生命周期，包括创建、挂载、更新和销毁等不同阶段。通过这种生命周期的机制，开发者可以方便地控制组件的行为和状态，同时也为 Vue.js 内部的一些操作提供了支持
9. **怎么设计一个组件呢？业务组件和通用组件？**
10. 功能划分：根据组件的功能，将其划分为多个子组件，每个子组件职责单一，便于维护和扩展
11. 数据设计：明确组件需要的数据类型和数据结构，并根据组件的复杂度选择适合的状态管理方案，例如 Vuex 或者简单的 Props 和 Events
12. API 设计：提供简单、易用、灵活的 API 接口，使组件可以适应不同的使用场景。同时，为了避免 API 滥用和错误使用，可以对 API 进行限制和验证
13. 样式设计：根据设计规范和用户体验，编写符合语义化的 HTML 和 CSS 样式，保证组件可用性和易用性
14. 可测试性设计：为了保证组件的质量和可靠性，需要设计合理的测试用例，包括单元测试和集成测试，保证组件的正确性和稳定性
15. 性能优化设计：在组件设计过程中，需要考虑性能优化问题，例如避免不必要的渲染和更新，避免频繁的 DOM 操作等
16. 文档和示例设计：提供完善的文档和示例，帮助用户快速上手和使用组件，同时也能提高组件的推广和普及度

通用组件：为了解决一类通用问题而设计的，要考虑更广泛的应用场景，要充分考虑组件的可配置性和可定制性，以便适应不同的应用场景和需求。通用组件的设计需要更加注重组件的独立性、可扩展性、可维护性和兼容性等方面

业务组件：业务组件是为了解决特定业务场景下的问题而设计的，通常具有一定的专业性和特殊性，不同的业务场景会有不同的实现方式和需求。因此，在设计业务组件时需要充分考虑业务需求和业务逻辑，并根据实际业务场景进行灵活调整和优化

1. **axios是什么？有什么特点？和fetch区别？**

是一个基于 Promise 的 HTTP 客户端，可以用在浏览器和 Node.js 中

1. 支持浏览器和 Node.js 环境
2. 支持 Promise API，可以使用 async/await 等语法进行异步编程
3. 支持请求拦截器（请求被发送出去之前拦截它，修改请求的配置或者添加自定义的头部信息）和响应拦截器（接收到响应之后拦截，修改响应的数据或者错误信息）通过使用 axios.interceptors 属性来添加，属性是一个对象，包含了两个属性：request 和 response，分别对应请求拦截器和响应拦截器。还支持取消请求的拦截器（请求被取消时执行一些自定义的操作）使用 CancelToken 可以实现请求的取消。
4. 支持自动转换请求和响应的数据格式，如 JSON、FormData 等
5. 支持取消请求，可以通过 cancelToken 取消请求，避免不必要的网络请求

区别：

1. Axios 支持浏览器和 Node.js 环境，而 fetch 仅支持浏览器环境
2. Axios 提供了更加完善的 API，如请求和响应的拦截器、自动转换请求和响应的数据格式、取消请求等
3. Axios 对错误处理和请求超时等问题进行了更加全面的处理
4. Axios 的支持者和使用者更多，社区活跃度更高，更新和维护更加稳定
5. **Vue项目如何兼容低版本浏览器的？**
6. 使用 polyfill 库：例如 babel-polyfill、core-js 等。这些库可以在运行时动态地将缺失的 ES6/7 特性进行填充，从而实现对低版本浏览器的兼容
7. 使用 transpile-only 编译：使用 webpack 或者 Rollup 等打包工具，可以将项目中的 ES6/7 语法通过 babel 进行编译，但不进行 polyfill，从而减小打包后的文件体积，提高页面加载速度
8. 使用 CDN 引入 Vue.js：如果项目不需要支持 IE8 及以下版本的浏览器，可以直接通过 CDN 引入 Vue.js，而不需要打包 Vue.js 到项目中，减小文件体积
9. 手动打包 Vue.js：在项目中手动下载并打包 Vue.js，使用 rollup 或者 webpack 进行处理，可以通过配置 target、useBuiltIns、exclude 等选项实现对低版本浏览器的兼容
10. 避免使用某些特性：在项目中避免使用一些不兼容低版本浏览器的特性，如 ES6 的箭头函数、let/const、模板字符串等。如果必须使用这些特性，可以通过使用 polyfill 或者手动打包的方式进行兼容处理
11. **用element-UI做过二次封装吗？遇到过什么问题，怎么解决的呢？**
12. 组件样式覆盖问题：在进行二次封装时，我需要对一些 Element UI 的组件进行样式覆盖，但是由于 Element UI 的组件样式具有较高的优先级，因此我的样式无法生效。我解决这个问题的方法是通过加上一个更高优先级的 class，或者在组件中加入 !important 这样的样式修饰符
13. 组件 API 风格问题：Element UI 的组件 API 风格和我自己的风格有所不同，我需要对一些组件的 API 进行调整以符合我的需求。我的解决方法是在二次封装时对组件的 API 进行修改，或者自行编写一个新的组件
14. 组件尺寸适应问题：在进行二次封装时，我需要让 Element UI 的组件适应我的项目的尺寸，但是 Element UI 的组件默认是按照设计稿上的尺寸进行的开发。我的解决方法是通过在组件中加入适应不同尺寸的样式代码，或者通过动态设置组件的属性来达到适应不同尺寸的效果
15. 组件功能不足问题：在二次封装过程中，我发现有些 Element UI 的组件的功能不够完善，或者无法满足我的需求。我的解决方法是通过编写新的代码或者使用其他的组件库来解决这些问题
16. **简述mixin extends的覆盖逻辑**

在Vue中，Mixin也是用来实现组件代码复用的一种机制，而extends选项则是用来扩展和覆盖组件选项的

具体来说，Vue中的Mixin通过extends选项将其属性和方法合并到组件中。当组件和Mixin中存在同名选项时，Vue会按照以下顺序进行覆盖：

1. 对于数据对象，将组件和Mixin中的数据对象合并，如果存在重复的键名，则组件中的数据会覆盖Mixin中的数据
2. 对于钩子函数，Mixin中的钩子函数会先于组件自身的钩子函数调用，且Mixin中的钩子函数按照被注入的顺序依次调用
3. 对于其他选项，例如computed和methods，如果组件和Mixin中存在同名选项，则组件中的选项会覆盖Mixin中的选项
4. **什么是自定义事件？自定义事件实现一个页面置顶功能？**

自定义事件是指开发者可以自己定义事件并在组件内部进行触发和监听的事件。可以用来在组件内部进行通信和交互，从而增强组件的灵活性和可重用性。

事件总线：在Vue实例中添加一个新的Vue实例，称之为事件总线。在需要置顶的组件中注册自定义事件，然后在其他组件中使用事件总线的$emit方法来触发自定义事件

使用Vuex：在Vuex store中定义一个state来保存当前是否需要置顶，然后在需要置顶的组件中通过mapState将该状态映射为computed属性，在其他组件中通过调用mapMutations中定义的方法来改变该状态

使用provide/inject：在根组件中使用provide来提供一个goTop方法，然后在需要置顶的组件中使用inject来注入该方法

window.scrollTo（{top:0}）

1. **选项式API和Composition API 区别？**
2. 组合式 API 提供了更好的逻辑复用性
3. 组合式 API 可以更好地解决代码复杂性问题
4. 组合式 API 更易于进行类型推导和 IDE 支持
5. 组合式 API 更易于进行代码重构和迁移

Composition API 相比选项式 API，提供了更好的逻辑复用性、更好的代码组织能力、更易于进行类型推导和 IDE 支持、更易于进行代码重构和迁移等优点。但是也需要注意，在使用 Composition API 时，需要掌握其特殊的语法和使用方式，避免出现一些常见的错误和问题

1. **设计一个popup组件需要注意哪些?怎么实现？**
2. 功能需求：需要明确该组件的作用，例如是用来显示提示信息、展示图片、输入表单等等。根据不同的需求，组件的功能和样式可能会有所不同
3. 可用性：弹出窗口应该易于使用，可以通过添加关闭按钮、点击外部区域关闭等方式提高用户体验。同时，应该注意避免过度使用弹出窗口，否则可能会影响页面的可用性和易用性
4. 样式设计：组件的样式需要与网站或应用的整体风格保持一致，例如颜色、字体、边框等。同时，应该考虑到弹出窗口的大小、位置、动画效果等，以便更好地展示其内容和提高用户体验
5. 兼容性：考虑不同浏览器和设备的兼容性，需要确保组件在各种情况下都能正常工作
6. 安全性：弹出窗口可能会涉及到用户的隐私信息或者执行一些敏感操作，因此需要保证组件的安全性
7. 组件复用：要考虑组件的复用性，使其可以在不同的场景和项目中都能够使用

设计和实现一个高质量的弹出窗（popup）组件需要综合考虑各种因素，包括功能需求、可用性、样式设计、兼容性、安全性、响应式设计、可访问性、浏览器支持、性能优化和组件复用等

1. **SPA首屏加载速度慢是什么原因呢？怎么解决呢？**

网络请求慢、资源体积过大、资源重复请求、加载脚本堵塞了渲染

解决办法：

**压缩图片（使用适当格式如 WebP）：**

使用[image-webpack-loader](https://www.npmjs.com/package/image-webpack-loader" \t "https://shuliqi.github.io/2021/06/21/vue%E9%A1%B9%E7%9B%AE%E9%A6%96%E5%B1%8F%E5%8A%A0%E8%BD%BD%E4%BC%98%E5%8C%96/_blank)对大图片进行压缩，对页面上使用到的icon，可以使用在线字体图标，或者雪碧图，将众多小图标合并到同一张图上，用以减轻http请求压力

**路由懒加载（延迟加载）：**

懒加载则可以将页面进行划分，需要的时候加载页面，可以有效的分担首页所承担的加载压力，减少首页加载用时。原理：将路由对应的组件打包成一个个的js代码块，只有在这个路由被访问到的时候，才加载对应的组件，否则不加载。使用ES6 import（）

**静态资源本地缓存（或者部署到 CDN上 ）：**

**CDN：**CDN是指内容分发网络（Content Delivery Network），是一种通过在多个地理位置上放置服务器来加快对互联网上静态和动态内容传输的技术。CDN通过缓存网站内容并将其分发到离用户最近的服务器上，以提高内容加载速度、减少带宽占用和网络延迟

1. 采用HTTP缓存，设置Cache-Control，Last-Modified，Etag等响应头
2. 采用Service Worker离线缓存、合理使用localStorage

**UI框架按需加载（element-UI、antd）按需引用，不引入整个UI库**

import { Button, Input, Pagination, Table, TableColumn, MessageBox } from 'element-ui'

**GZip压缩（服务器压缩、前端webpack打包生成gz文件）：**

服务器：浏览器请求文件时，服务器对该文件进行压缩后传输给浏览器

前端：浏览器请求文件时，服务器返回相应的的文件的 .gz 文件。安装 compression-webpack-plugin，在vue.congig.js中引入并修改webpack配置，在服务器我们也要做相应的配置 如果发送请求的浏览器支持gzip，就发送给它gzip格式的文件

**webpack打包体积优化（减少首屏加载的请求次数）**

使用SplitChunks（Webpack中一个提取或分离代码的插件，主要作用是提取公共代码，防止代码被重复打包，拆分过大的js文件，合并零散的js文件）

**使用SSR：**

组件或页面通过服务器生成html字符串，再发送到浏览器

**HTTP-title**

# **Http是什么？有哪些字段呢？**

是一种用于传输超文本（HTML）和其他资源的协议。HTTP是基于客户端-服务器模型工作的，其中客户端向服务器发送请求，服务器响应请求并返回相应的资源

字段：

1. 请求行：包括请求方法、请求URI和HTTP版本
2. 请求头部：包括很多字段，例如Host、User-Agent、Accept、Accept-Encoding、Content-Type等，用于告诉服务器一些请求的信息
3. 请求正文：请求正文中包含了客户端向服务器传输的实际数据，例如表单数据、JSON数据等
4. 响应行：包括HTTP版本、状态码和状态码描述
5. 响应头部：包括很多字段，例如Server、Content-Type、Content-Length、Cache-Control等，用于告诉客户端一些响应的信息
6. 响应正文：响应正文中包含了服务器返回给客户端的实际数据，例如HTML、CSS、JavaScript、图片、视频等

HTTP协议通过请求头和响应头传递各种类型的信息，从而支持各种不同的功能和用途，例如身份验证、缓存控制、内容协商等

1. **说下HTTP常用状态码？分别是什么意思？**

是HTTP协议用于表示Web服务器响应请求的结果的数字代码

1. 200 OK：表示服务器成功处理了客户端请求，并返回了请求的内容。这是最常见的状态码之一
2. 401 Unauthorized：认证失败，表示请求没有被认证或者认证失败，通常由web服务器返回，场景（token失效、token缺失、token伪造，导致服务端无法识别身份）
3. 403 Forbidden：缺少指定权限，表示服务器拒绝客户端访问请求的资源，因为客户端没有足够的权限访问该资源，由web应用返回，场景（用户登录成功，但是无权进行读写操作）
4. 404 Not Found：表示客户端请求的资源不存在，服务器无法找到请求的资源
5. 500 Internal Server Error：表示服务器内部出现错误，无法完成请求。这通常是由于服务器端的错误导致的，客户端无法解决
6. 302 Found：表示请求的资源已经被临时移动到其他位置。这个状态码常用于重定向功能
7. 304 Not Modified 表示客户端请求的资源在服务器上没有修改过，客户端可以使用本地缓存的版本

# **说一下HTTP缓存？**

是指浏览器或者代理服务器在第一次请求资源时将该资源缓存起来，在后续请求时直接从缓存中读取，避免了再次请求服务器获取资源的开销，从而提高页面性能和减少服务器压力

强缓存

强缓存是指在第一次请求资源时，浏览器会将该资源的一些信息存储在本地，例如过期时间（Expires）和最大缓存时间（max-age）。在下一次请求该资源时，浏览器会根据这些信息判断该资源是否过期，如果未过期则直接从本地缓存中读取该资源，否则重新向服务器请求该资源。强缓存的优点是在过期时间内直接使用缓存，不需要与服务器通信，速度更快

协商缓存

协商缓存是指在第一次请求资源时，服务器会返回一个响应头，包含该资源的一些信息，例如资源的最后修改时间（Last-Modified）和标识该资源的唯一标识符（ETag）。在下一次请求该资源时，浏览器会将这些信息带上，发送给服务器。服务器会根据这些信息判断该资源是否发生了变化，如果未发生变化，则返回304状态码，告诉浏览器可以使用缓存，否则返回新的资源。协商缓存的优点是能够精确地判断资源是否发生了变化，避免了不必要的数据传输

# **HTTP如何设置缓存标识呢？**

主要是通过响应头的Cache-Control字段和Expires字段来实现

1. Cache-Control字段：Cache-Control字段是控制缓存的最重要的响应头之一，它的值包括多个指令，例如max-age、no-cache、no-store等。其中，max-age指令可以设置缓存的最长时间，例如设置Cache-Control: max-age=3600表示缓存时间为3600秒，当超过这个时间后就需要重新请求数据了。no-cache指令表示不使用缓存，每次都需要向服务器请求最新数据。no-store指令表示不允许缓存任何内容
2. Expires字段：Expires字段是HTTP/1.0协议中定义的一个响应头字段，它的值是一个日期，表示缓存的过期时间。例如设置Expires: Wed, 21 Oct 2023 07:28:00 GMT表示缓存会在这个日期之后失效，需要重新请求数据。但是，由于Expires字段的日期是根据服务器的时钟设置的，而客户端的时钟可能与服务器的时钟存在差异，因此缓存的过期时间可能会出现问题

在实际开发中，一般同时使用Cache-Control和Expires字段来设置缓存标识，以提高缓存的准确性和可靠性。例如，可以设置Cache-Control: max-age=3600, public和Expires: Wed, 21 Oct 2023 07:28:00 GMT来同时控制缓存的过期时间和是否可以被公共缓存

# **HTTP和HTTPS的区别？**

相同点：都是用于在客户端和服务器之间传输数据的协议

不同点：

1. 安全性：HTTP协议是明文传输数据的，数据容易被窃听和篡改，而HTTPS协议通过SSL/TLS加密传输数据，可以保证数据的安全性和完整性
2. 数据传输方式：HTTP协议使用TCP连接传输数据，而HTTPS协议则是在TCP连接上使用SSL/TLS进行加密传输数据
3. 端口号：HTTP协议默认使用80端口进行通信，而HTTPS协议默认使用443端口进行通信
4. 证书验证：HTTPS协议需要使用数字证书来验证服务器的身份，以保证通信的安全性，而HTTP协议没有证书验证机制
5. 性能：HTTPS协议的加密和解密过程需要消耗一定的计算资源，相比HTTP协议会稍微慢一些，但是随着硬件技术的不断发展，HTTPS的性能问题已经不再是问题

HTTP协议主要用于普通的网站数据传输，而HTTPS协议则用于涉及到敏感信息（例如登录密码、银行账号等）的网站数据传输，以保障数据的安全性

# **HTTPS安全性是怎么做到的呢？**

1. 加密传输：HTTPS使用SSL/TLS协议进行加密传输，保护数据在传输过程中不被窃听或篡改。SSL/TLS使用对称加密和非对称加密两种方式来实现加密传输，对称加密用于加密数据，非对称加密用于对加密密钥进行加密和解密
2. 数字证书验证：HTTPS使用数字证书来验证服务器的身份，防止中间人攻击和伪造。数字证书由受信任的第三方机构（CA）颁发，证书中包含了服务器的公钥和证书信息，客户端使用该公钥进行加密通信，并且可以验证证书的合法性
3. 安全协商：在HTTPS握手过程中，客户端和服务器会进行安全协商，确定加密算法、密钥长度、协议版本等参数，保证双方使用最佳的安全配置进行通信
4. HTTP头信息：HTTPS在HTTP头信息中添加了一些安全相关的字段，例如Strict-Transport-Security（HSTS）和Content-Security-Policy（CSP），用于增强HTTPS的安全性

# **HTTP2有了解吗？**

HTTP/2 通过减少延迟和提高网络通信效率来提高 Web 应用程序的性能。它通过引入几个新功能来实现这一点，包括二进制帧、服务器推送、流优先级和标头压缩

1. 二进制分帧意味着 HTTP 消息被分成更小的单元，称为帧，可以并行发送和接收。这减少了与通过网络发送大型 HTTP 消息相关的延迟
2. 服务器推送允许服务器主动向客户端发送资源，而无需客户端请求它们。这有助于减少加载网页所需的往返次数，并可以提高 Web 应用程序的感知性能
3. 流优先级允许客户端指定加载资源的顺序，这有助于确保首先加载最重要的资源
4. 标头压缩减少了 HTTP 标头的大小，这在某些情况下可能很重要，并且有助于减少需要通过网络发送的数据量

**Webpack**

# **Webpack是什么？是用来做什么的？原理是什么？常见功能有哪些？常用的配置有哪些？**

Webpack是一个现代化的JavaScript应用程序静态模块打包器。

用来将一个或多个JavaScript文件打包成一个或多个静态资源文件，以便在浏览器中使用。Webpack不仅支持JavaScript，还可以通过适当的加载器（loaders）来处理CSS、图片、字体等资源。

原理是将所有的资源文件看做是一个个模块，通过各个模块之间的依赖关系，打包成一个或多个静态资源文件。Webpack会通过入口文件来构建依赖图，然后根据依赖关系进行代码分割，最终生成目标代码。

常见功能：

1. 模块打包
2. 加载器（CSS、图片、字体）
3. 插件（代码压缩、自动部署）
4. 代码分割（配置自动分割代码，从而实现按需加载）
5. 开发服务器（启动一个开发服务器，支持自动刷新和热更新）

常见配置：

1. entry：应用程序入口文件。
2. output：打包后的文件输出路径和文件名。
3. module：定义各种类型文件的加载器。
4. plugins：配置各种插件。
5. resolve：定义模块解析规则。
6. devServer：配置开发服务器。
7. optimization：配置代码优化相关的选项。
8. mode：设置构建模式，可以是development、production或none等。
9. **做过哪些webpack性能优化？**
10. 使用 Tree Shaking 和 Scope Hoisting 等技术来减少打包的代码量，减小打包后的文件体积
11. 配置 babel-loader 和 terser-webpack-plugin，使用多线程或者缓存等技术来加速编译和压缩 JavaScript 代码
12. 使用 splitChunks 和 webpack.optimize.CommonsChunkPlugin 等插件，对公共代码进行提取和分离，减少冗余代码，提高代码的复用性
13. 针对生产环境的构建，启用 webpack.DefinePlugin 来设置环境变量，从而可以在构建时进行一些条件判断和优化，例如启用代码压缩等

# **Webpack 、grunt、gulp 区别？**

都是前端构建工具，用于自动化前端项目的构建、优化和部署

区别：

1. Webpack：是一个模块打包工具，主要用于打包处理 JavaScript、CSS、图片等文件。它将所有资源视为模块，通过配置可以将这些模块打包成最终的静态资源文件。Webpack 还可以进行代码拆分、压缩混淆、按需加载等优化操作
2. Grunt：是一个基于任务的构建工具，可以定义一系列需要执行的任务，比如压缩、合并、编译等。它可以自动化执行这些任务，提高开发效率
3. Gulp：也是一个基于任务的构建工具，与 Grunt 相似，但它的优点在于更快的构建速度和更简洁的代码风格。它使用流式处理，可以避免中间文件的产生，同时具有更好的可读性和可维护性

# **webpack 打包后的map是用来做什么的?**

通常是打包后生成的source map文件

用于调试：可以将编译后的代码映射回源代码，提供有关编译后源代码中每个模块的位置，编译后代码中每个模块的位置，以及编译后代码的每个代码行的位置

注意：在生产环境中，为了安全性和性能，我们通常不会将 source map 文件公开，而是只在需要进行调试的时候才启用它们

## Git-title

1. **Git的项目管理工作流**
2. 创建主分支：通常情况下，开发团队会创建一个主分支（通常为master或main），所有的开发工作都在这个分支上进行
3. 创建新的分支：每个新的功能或修复问题都应该在一个新的分支上开发。在这个分支上进行开发，这样可以避免直接对主分支进行修改
4. 开发代码：在新的分支上开发代码，使用git add和git commit命令来提交修改
5. 测试和审查：在提交代码之前，需要进行测试和审查。这可以帮助识别潜在的错误和问题，以便及早解决
6. 合并到主分支：在完成功能或修复问题之后，需要将分支合并到主分支上。这可以使用git merge命令完成
7. 发布：一旦代码准备好发布，需要将代码推送到远程仓库中，以供团队中的其他人员进行使用。这可以使用git push命令完成
8. 解决冲突：如果在合并分支时发生冲突，需要解决冲突。通常情况下，解决冲突需要合并代码、手动编辑代码并提交修改
9. 回退：如果需要回退到之前的版本，可以使用git checkout命令
10. 标记版本：为了跟踪版本，可以使用git tag命令在特定的提交上打标签。这可以帮助团队更好地管理和跟踪发布版本
11. **Git-flow工作流是什么？Git 提交规范？**

是一种基于Git版本控制工具的工作流程，它旨在优化代码的开发和管理流程Git-flow将代码的开发和发布分成多个阶段，使用不同的分支来管理和协调这些阶段，从而帮助团队更好地组织代码的开发和发布

Git-flow的基本分支结构包括两个永久性分支(master和develop)和三个短期分支(feature、release和hotfix)

1. master 分支：代表了当前生产环境中的代码，包含了所有经过测试并且可用的代码。只有从develop或release分支合并过来的代码才能进入master分支
2. develop 分支：代表了下一版本的开发状态，包含了所有经过测试并且可用的代码。所有的feature分支和release分支都是从develop分支分出来的，并且只有当这些分支的代码被测试通过之后，才能合并回develop分支
3. feature 分支：代表了单个的功能开发分支，从develop分支分出来，用于开发单个功能。开发完毕后，feature分支会合并回develop分支，然后被删除
4. release 分支：代表了准备发布一个新版本时的状态。release分支通常是在完成所有功能开发并进行测试之后，从develop分支分出来的。在release分支上进行bug修复和文档编写等，直到该分支准备发布为止。一旦发布，release分支会合并回develop分支和master分支，并打上对应的版本号标签
5. hotfix 分支：代表了修复在生产环境中出现的问题所需的分支，从master分支分出来。开发完成后，hotfix分支会合并回master分支和develop分支，并打上对应的版本号标签

规范：

feat：新功能、fix：修复 bug、docs：文档更新、style：代码格式调整、refactor：代码重构、test：测试代码更新、chore：构建或辅助工具的变动

1. **Git常用命令？**
2. git init：将当前目录初始化为Git仓库
3. git clone：从远程仓库克隆代码到本地
4. git add：将修改的文件添加到暂存区
5. git commit：将暂存区的文件提交到本地仓库
6. git push：将本地仓库的代码推送到远程仓库
7. git pull：从远程仓库拉取最新代码到本地
8. git status：查看当前仓库的状态，包括哪些文件被修改、添加或删除等
9. git log：查看提交历史记录，包括每次提交的作者、时间、提交信息等
10. git branch：查看分支信息，包括当前分支、已有分支等
11. git checkout：切换分支或恢复文件
12. git merge：将不同分支的代码合并到一起
13. git stash：将修改的文件暂时保存起来，以便在未来的某个时间再次使用

## 性能优化-title

1. **说一下浏览器渲染原理？**
2. 构建DOM树：浏览器会解析HTML文档，将HTML标签转换为DOM节点，并建立起DOM树
3. 构建CSSOM树：浏览器会解析CSS样式，将CSS样式表转换为CSSOM树
4. 合并DOM树和CSSOM树：浏览器将DOM树和CSSOM树合并，生成一棵渲染树(Render Tree)，渲染树只包含需要显示的节点和相关的样式信息
5. 布局计算：浏览器会根据渲染树中每个节点的大小、位置等信息，计算出每个节点在屏幕中的精确位置
6. 绘制渲染树：浏览器将渲染树中每个节点绘制出来，生成一张位图(bitmap)或者图层(layer)，并将其存储到显存中
7. 合成图层：如果页面中有多个图层，浏览器会将这些图层进行合成，最终生成一张完整的页面图像，并将其显示在屏幕上

# **说说地址栏输入URL敲下回车后发生了什么?**

1. 解析URL：浏览器首先会解析URL，以确定它包含哪个协议（如HTTP或HTTPS），哪个域名以及可能包含的其他路径或查询参数
2. DNS解析：浏览器会尝试查找URL中指定的域名的IP地址。为此，它会向DNS服务器发送请求，以获取与该域名相关联的IP地址
3. 建立TCP连接：一旦浏览器获得了目标服务器的IP地址，它会使用TCP协议建立一个连接
4. 发送HTTP请求：一旦TCP连接建立，浏览器会向服务器发送一个HTTP请求，该请求包含浏览器想要获取的资源的类型、位置和其他相关信息
5. 服务器处理请求并发送响应：一旦服务器接收到请求，它会查找并检索请求的资源，并使用HTTP响应将其发送回浏览器。响应可能包含HTML代码、CSS样式表、JavaScript代码、图像或其他类型的数据
6. 浏览器渲染页面：一旦浏览器接收到响应，它会解析HTML代码并构建一个文档对象模型（DOM），然后使用CSS和JavaScript来呈现和交互页面
7. 断开TCP连接：一旦浏览器完成对资源的请求和渲染，它会断开与服务器的TCP连接

# **设计模式了解吗？**

是前端开发中常用的一些解决问题的方法论，旨在提高代码的可重用性、可维护性和可扩展性

1. MVC（Model-View-Controller）模式：将应用程序分为三个主要部分，即数据模型（Model）、视图（View）和控制器（Controller），以便实现应用程序的分离和解耦
2. MVVM（Model-View-ViewModel）模式：是MVC模式的改进版，通过使用数据绑定，将视图和数据模型分离，以便实现数据的自动同步
3. 观察者模式（Observer Pattern）：一种对象间的一对多依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都会得到通知并自动更新
4. 单例模式（Singleton Pattern）：确保一个类只有一个实例，并提供一个全局访问点来访问该实例
5. 工厂模式（Factory Pattern）：使用一个工厂方法来创建对象，以便将对象的创建过程和使用过程分离，从而提高代码的可维护性和可扩展性
6. 装饰者模式（Decorator Pattern）：在不改变对象接口的情况下，动态地给对象添加新的功能
7. 适配器模式（Adapter Pattern）：将一个类的接口转换成客户希望的另外一个接口，以便复用旧代码

# **页面内容加载缓慢怎么解决？**

常用优化策略：

1. 压缩文件大小（HTML、CSS、JavaScript等文件进行压缩）
2. 合并文件请求（将多个CSS、JavaScript等文件合并成一个文件，减少文件请求次数）
3. 使用CDN加速（使用CDN来分发常用的静态文件，例如CSS、JavaScript等）
4. 使用图片压缩（使用图片压缩工具对其进行压缩，从而减少文件大小）
5. 使用缓存（HTTP缓存、本地缓存）
6. 懒加载（延迟加载一些不是页面初始化时必须要加载的资源，例如图片、视频等。可以在用户滚动页面到某个区域时再进行加载，从而减少页面初始化的加载时间）
7. 优化代码（使用压缩、去重、缓存等技术来优化代码）

# **Hybrid app了解吗？怎么调用原生API的？**

是一种混合式移动应用程序，它结合了原生应用程序和Web应用程序的优点，使用Web技术（如HTML，CSS和JavaScript）构建应用程序的用户界面，同时使用原生技术（如Java或Objective-C）来访问设备的原生功能（如相机或通讯录）

JavaScript Bridge（在Web应用程序和原生应用程序之间建立通信桥梁，允许Web应用程序使用JavaScript代码来调用原生应用程序中的功能和方法）

1. 在原生应用程序中创建一个接口，用于暴露原生方法和功能
2. 在Web应用程序中，使用JavaScript代码来调用原生方法和功能
3. 在原生应用程序中，实现接口方法，以便在接收到来自Web应用程序的请求时执行相应的操作
4. 在Web应用程序中，处理来自原生应用程序的响应，并采取适当的行动

# **Uni-app生命周期执行顺序？**

应用生命周期

1. onLaunch：应用程序启动时触发，可以在此处进行全局初始化等操作
2. onShow：应用程序显示在前台时触发，可以在此处进行数据更新等操作
3. onHide：应用程序隐藏到后台时触发，可以在此处进行数据保存等操作
4. onUnload：页面关闭时触发，可以在此处进行清理工作等操作
5. onError :当 uni-app 报错时触发

页面生命周期

1. onLoad：页面加载时触发，可以在此处进行数据初始化等操作
2. onShow：页面显示时触发，可以在此处进行页面数据更新等操作
3. onReady：页面初次渲染完成时触发，可以在此处进行与页面渲染相关的操作
4. onHide：页面隐藏时触发，可以在此处进行页面数据保存等操作
5. onUnload：页面卸载时触发，可以在此处进行页面数据清理等操作

# **对前端工程化的理解？为什么要做工程化？**

前端工程化是指使用一系列工具和流程来规范化和优化前端开发的过程和产出，以提高项目开发和维护的效率和质量。

因为现代Web应用已经变得越来越复杂，需要处理大量的业务逻辑和数据，同时还要考虑多浏览器兼容性、性能优化、代码维护等问题。如果没有工程化的支持，前端开发就会变得混乱、低效、容易出错，同时也会导致项目交付周期长、质量不稳定。

前端工程化可以带来以下好处：

1. 提高效率：自动化构建、测试、部署等流程可以减少手动操作，缩短开发时间，提高开发效率。
2. 保证质量：自动化测试和代码规范检查可以减少人为失误，提高代码质量和可维护性。
3. 降低成本：前端工程化可以减少人工成本、加快上线周期，从而降低开发和维护成本。
4. 提升用户体验：性能优化和页面渲染速度的提升可以让用户获得更好的使用体验。
5. 方便协作：前端工程化可以统一团队开发流程，方便团队协作和代码管理。

# **项目中是怎么调试页面的？做过哪些页面性能优化？使用过哪些性能调试工具？**

1. 使用 Vue Devtools：Vue Devtools 是一个浏览器插件，可以帮助我们实时调试 Vue 应用，查看组件树、数据、事件等信息，快速定位和解决问题
2. 使用 Chrome 开发者工具：Chrome 开发者工具是一个内置于 Chrome 浏览器中的调试工具，可以帮助我们查看页面的 HTML、CSS、JavaScript 代码，进行调试和性能分析等操作
3. 使用 console.log()：console.log() 是一个简单而又常用的调试工具，在 Vue 项目中也可以使用它来输出调试信息，查看数据的变化、方法的执行情况等
4. 使用Vue.config.devtools和Vue.config.debug：可以通过设置 Vue.config.devtools 和 Vue.config.debug 来开启或关闭 Vue 的调试模式，方便调试和排查问题
5. 使用 VS Code 插件：VS Code 提供了一些插件，例如 Vetur、Vue Peek、Vue.js Devtools 等，可以帮助我们更好地编辑、调试和维护 Vue 代码
6. 使用模拟数据和 Mock 数据：在开发过程中，可以使用模拟数据和 Mock 数据来模拟真实的数据和情况，方便调试和测试

页面性能优化：

1. 图片优化：对于图片的大小、格式、数量等进行优化，尽可能减少图片的加载时间。可以使用压缩图片的工具，例如 TinyPNG、Kraken 等
2. 前端缓存：使用浏览器缓存、LocalStorage 等技术来减少网络请求，提高前端页面的加载速度
3. HTTP 请求优化：合并和压缩 JavaScript、CSS 等静态资源，减少 HTTP 请求次数。使用 DNS 预解析、CDN 等技术优化资源加载速度
4. 代码优化：减少 DOM 操作和 JavaScript 运算等操作，避免使用不必要的插件和库，减少不必要的计算和渲染操作
5. 懒加载路由组件：Vue-Router 支持路由懒加载，可以将路由组件拆分成独立的代码块，按需加载，减少首页的加载时间
6. 使用 Keep-Alive 组件：Keep-Alive 组件可以缓存已经渲染的组件，减少组件的重复渲染，提高组件的加载速度
7. 合理使用 Vuex：Vuex 是 Vue 的状态管理库，可以帮助我们更好地管理和共享状态，但是在使用时需要注意避免不必要的计算和更新操作，否则会影响性能和响应速度
8. 使用异步组件：使用 Vue 的异步组件可以将组件的加载过程放在后台进行，减少页面的加载时间和提高用户体验

性能调试工具：

1. Chrome 开发者工具：Chrome 开发者工具是一个内置于 Chrome 浏览器中的调试工具，可以帮助我们查看页面的 HTML、CSS、JavaScript 代码，进行调试和性能分析等操作。它可以查看页面的资源、性能、网络、内存、CPU 使用情况等，还可以模拟不同的设备和网络环境，进行调试和测试
2. WebPageTest：WebPageTest 是一个在线的性能测试工具，可以测试页面的加载速度、响应时间、DOM 渲染时间等指标，并提供详细的报告和分析
3. Lighthouse：Lighthouse 是一个由 Google 开发的性能测试工具，可以对网站的性能、可访问性、最佳实践等方面进行评估和优化。它可以提供一些实用的建议和指导，帮助我们优化页面性能和用户体验
4. YSlow：YSlow 是一个性能测试工具，可以评估页面的性能、结构、优化等方面，并给出一些实用的建议和指导。它可以测量页面的加载时间、HTTP 请求数量、缓存设置、JavaScript 和 CSS 优化等方面，并提供相应的优化建议
5. **项目中遇到的难点？或者项目中自己觉得做的比较好的地方？**
6. **项目是怎么做用户登录验证的？举例：人脸识别整个过程做了哪些操作？**

路由守卫：在路由配置中添加一个路由守卫，在需要验证登录的路由中添加 meta 标记（requiresAuth: true）在登录成功后将用户信息保存到本地存储中，在用户访问需要登录验证的路由时，路由守卫会判断用户是否已登录，如果未登录则跳转到登录页面。如果已登录，则继续访问目标路由

Axios拦截 HTTP 请求：在每个请求中添加验证信息，例如 token，如果验证信息未通过，则返回 401 错误码。在前端拦截到 401 错误码后，可以跳转到登录页面

人脸：

1. 采集：首先需要采集图像或视频数据，以获取人脸数据。这可以通过使用摄像头、监控摄像头、图像库等方式来实现
2. 预处理：采集的图像或视频数据需要进行预处理，包括图像增强、图像归一化、噪声去除等操作，以提高人脸识别的准确性和鲁棒性
3. 人脸检测：对于一张图像或视频帧，需要首先检测其中是否存在人脸。这可以通过使用基于 Haar 特征或深度学习算法的人脸检测器来实现
4. 人脸对齐：在检测到人脸后，需要对人脸进行对齐，以保证人脸在后续的处理中具有相同的尺度、姿态和表情。这可以通过使用人脸关键点检测算法来实现
5. 特征提取：对于每个检测到的人脸，需要提取其特征向量，以便后续的识别操作。这可以通过使用基于神经网络的特征提取算法，例如卷积神经网络 (CNN) 或循环神经网络 (RNN) 来实现
6. 特征匹配：对于一张待识别的图像或视频帧，需要将其特征向量与已知的人脸特征向量进行比对，以确定其身份。这可以通过使用一些距离度量算法，例如欧氏距离或余弦相似度来实现
7. 身份识别：最后，需要对比对结果进行决策，确定待识别的人脸的身份。这可以通过使用一些分类算法，例如支持向量机 (SVM) 或随机森林 (Random Forest) 来实现
8. **说一下微信小程序登录流程？**
9. 用户进入小程序，点击登录按钮或者需要授权的功能，如微信支付等
10. 小程序发送登录请求到微信服务器，微信服务器返回一个登录凭证code给小程序
11. 小程序将登录凭证code发送到开发者服务器
12. 开发者服务器接收到登录凭证code后，调用微信提供的接口，使用开发者的appid和appsecret以及登录凭证code向微信服务器发起请求，获取用户的唯一标识openid和会话密钥session\_key
13. 开发者服务器将获取到的openid和session\_key返回给小程序
14. 小程序将openid和session\_key保存在本地，以便后续调用需要用户身份信息的接口，如获取用户信息、支付等

整个登录流程中，小程序通过登录凭证code获取到openid和session\_key，openid是用户在当前小程序的唯一标识，而session\_key用于后续的数据加密和解密。同时，为了保证用户信息的安全性，开发者服务器需要对openid和session\_key进行存储和加密

1. **小程序主包分包是是什么？具体怎么做的？**

小程序主包分包是指将小程序中的一些页面或组件分离到不同的子包中，以减少主包的大小，提高小程序的启动速度和性能。

实现步骤：

1. 在小程序的 app.json 文件中，使用 "subPackages" 字段来定义需要分离出来的子包
2. 在小程序项目中创建对应的子包目录，并将需要分离的页面和组件放入相应的子包目录中
3. 在需要加载子包的页面中，使用 wx.loadSubPackage() 方法来加载子包

需要注意的是，加载子包会有一定的网络请求延迟，因此建议在小程序启动时就预加载子包，以提高用户体验。此外，由于子包中的页面和组件是独立的，因此需要在子包中引入所需的外部资源，如图片、CSS 样式等

1. **H5怎么调用原生功能？**

使用JavaScript Bridge

在移动端开发中，通常会使用JavaScript Bridge来实现H5与原生应用的交互。JavaScript Bridge是一个桥梁，可以让H5与原生应用之间进行相互调用。通过调用JavaScript Bridge提供的接口，H5页面可以调用原生应用提供的功能。例如，在iOS中，可以使用JavaScriptCore框架提供的JSContext对象来实现JavaScript Bridge。在JSContext中，可以使用evaluateScript方法来执行JavaScript代码，并通过调用JSContext对象的方法，将原生应用中的方法暴露给JavaScript使用

使用WebVieW

WebView是一个内置浏览器控件，可以在原生应用中嵌入H5页面。在WebView中，H5页面可以调用原生应用提供的方法。在Android中，可以使用WebView的addJavascriptInterface方法将Java对象暴露给JavaScript使用。在iOS中，可以使用WKWebView的WKScriptMessageHandler协议实现H5与原生应用之间的交互

1. **PC端项目怎么做权限管理的？**

⻚⾯权限和按钮权限

前端

路由控制：通过路由守卫要求⽤户登录，⽤户登录后根据⻆⾊过滤出路由表。⽐如我会配置⼀个 asyncRoutes 数组，需要认证的⻚⾯在其路由的 meta 中添加⼀个 roles 字段，等获取⽤户⻆⾊之后取两者的交集，若结果不为空则说明可以访问。此过滤过程结束，剩下的路由就是该⽤户能访问的⻚⾯，最后通过 router.addRoutes(accessRoutes) ⽅式动态添加路由

接口鉴权：后端接口中添加鉴权机制，在请求头中附加用户的token信息，然后在后端接口中根据该token获取用户信息。如果用户具有访问该接口的权限，则返回数据，否则返回无权访问的提示。在每次请求前，我们需要获取当前用户的token信息，并将其附加在请求头中，然后后端根据该token进行身份验证

权限管理组件：

**---待办**

Vuex状态管理

1. **怎么做用户登录鉴权的？**

用户登录：用户在客户端输入账号和密码进行登录，客户端将账号和密码发送到后端进行校验，如果校验通过，则后端生成一个 token 并返回给客户端

1. token存储：客户端在本地存储该token，通常使用浏览器的 LocalStorage/sessionStorage  或者客户端的内存进行存储
2. token 发送：客户端在后续的请求中，将该 token 作为请求的一个参数或者请求头发送给后端
3. token 鉴权：后端在接收到客户端请求时，首先要对该 token 进行鉴权，验证该 token 是否有效和合法。如果 token 无效或者已过期，则拒绝该请求
4. 接口权限控制：对于不同的接口，可以设置不同的访问权限，例如只有登录用户才能访问某些接口，或者只有管理员才能访问某些接口。这样可以对用户进行更精细的权限控制
5. **A页面>B页面>C页面，从C返回到A是怎么做的？C页面做完操作如何保存状态同步到B页面呢？**

**---待办**

1. **lodash常用哪些方法？原理是什么？**
2. \_.map：对数组或对象的每个元素应用一个函数，返回处理后的新数组或对象
3. \_.reduce：对数组或对象进行累加计算，返回计算结果
4. \_.filter：对数组或对象进行过滤，返回符合条件的元素组成的新数组或对象
5. \_.sortBy：对数组或对象进行排序，返回排序后的新数组或对象
6. \_.debounce：防抖函数，用于减少函数的调用次数
7. \_.throttle：节流函数，用于限制函数的调用频率
8. \_.cloneDeep：深拷贝函数，用于复制一个对象或数组
9. \_.isEmpty：判断一个对象或数组是否为空
10. \_.isEqual：判断两个对象或数组是否相等

Lodash 的原理是基于函数式编程思想，将常见的函数和方法封装成单独的函数，通过组合这些函数来实现更复杂的功能。Lodash 还通过对一些常用方法的优化，如对数组、对象的遍历优化，提升了函数执行效率和性能，让开发者能够更快速、更便捷地完成开发任务

1. **html2canvas是什么？原理是什么？**

html2canvas是一个JavaScript库，它可以将HTML页面中的内容（包括CSS样式和JavaScript交互效果）转换为Canvas图像。通过html2canvas，开发者可以实现网页截图、保存网页为图片、实现图形报表等功能。

html2canvas的原理是将HTML元素解析为Canvas元素，然后绘制在一个临时的Canvas上，并最终导出为图片格式

1. 解析HTML页面，将每个HTML元素转换为Canvas元素，根据元素的样式设置Canvas的样式
2. 创建一个临时的Canvas，大小与当前页面相同
3. 遍历所有的Canvas元素，绘制在临时Canvas上
4. 将临时Canvas上的图像导出为图片格式，如PNG或JPEG
5. 需要注意的是，html2canvas有一些局限性，例如它不能处理跨域图片、使用CSS3动画或transition的元素、使用WebGL或SVG的元素等。此外，由于html2canvas会将整个HTML页面转换为Canvas图像，因此对于大型页面或者有复杂交互效果的页面，可能会导致性能问题
6. **H5是怎么做支付的？怎么调起支付呢？有哪些必传参数？**

原生JavaScript代码来调起第三方支付SDK，HTML5+(plus.payment)

必传参数：

1. 商户订单号：商户系统生成的订单号，保证唯一性
2. 订单名称：订单的名称或描述
3. 交易金额：订单的总金额，单位为元
4. 支付渠道：支付宝或微信支付
5. 支付方式：支付宝支持支付宝钱包、余额、银行卡等方式，微信支付支持微信钱包、银行卡等方式
6. 通知URL：支付结果通知地址，支付平台将支付结果异步通知到该地址
7. 回调URL：支付完成后用户返回商户的地址，用于页面跳转
8. 商品描述：商品或支付项目的描述信息
9. **做过虚拟列表吗？后端返回数据量大怎么处理？**

虚拟列表是一种优化技术，用于处理大量数据的列表展示，它只会渲染屏幕内可见的部分数据，而非全部数据，从而提高列表的渲染性能和用户体验

1. 分页加载：将数据按照一页一页地返回，然后在前端进行分页处理。当用户滚动列表时，再动态加载下一页的数据
2. 后端数据过滤：在后端进行数据过滤，只返回当前列表需要的数据。例如，可以使用类似 SQL 的 limit 和 offset 语句来指定要返回的数据行数和偏移量
3. 前端数据缓存：将后端返回的数据缓存在前端，然后在前端进行数据分页和过滤处理。这样可以减少对后端的请求次数，提高列表的渲染性能和用户体验
4. 虚拟滚动：虚拟滚动是一种针对虚拟列表的优化技术，它只渲染当前可见区域的数据，并且在用户滚动时动态加载新的数据。这样可以减少需要渲染的数据量，提高列表的渲染性能

对于大量数据的列表展示，前端性能的提升并不是唯一的解决方案。后端也应该尽可能地优化数据查询和返回的性能，以便更快地响应前端的请求

1. **文件上传是怎么做的？怎么限制文件大小？大文件是怎么处理的？loading是怎么做的？**

el-upload

before-upload：limit （限制上传文件数量）、file.size（ 限制上传文件大小

**）**

大文件处理:

1. 分片上传：将大文件分割成小块，分别上传。这样可以避免上传过程中出现网络中断等问题时需要重新上传整个文件的情况
2. 断点续传：在上传过程中，如果网络中断或上传失败，可以尝试断点续传，即从上一次上传的位置继续上传，而不是重新上传整个文件
3. 进度条：在上传过程中，可以使用进度条来显示上传进度，让用户了解上传进度

Element UI提供的Loading组件，可以用来显示全屏或指定区域的loading效果

1. **小程序项目是如何获取用户信息的？**

通过微信开放平台提供的接口获取用户信息。当用户打开小程序时，小程序会向用户请求授权获取用户的基本信息，如昵称、头像、地理位置等

通过wx.getUserInfo()方法获取用户信息。在调用该方法前，需要先判断用户是否已经授权，如果用户未授权，则需要调用wx.authorize()方法进行授权。另外，如果需要获取用户的手机号码等敏感信息，则需要先通过微信开放平台申请相应权限，并在小程序中进行相应配置。在用户授权同意后，可以通过wx.request()方法向后端服务器发送请求，获取用户的敏感信息

1. **项目里是怎么保存数据的？**
2. 使用组件的 data 属性：在 Vue 组件中，可以通过 data 属性定义组件内部的数据。这些数据会被 Vue 实例化后自动挂载到组件实例上，可以在组件内部直接使用和修改。在组件中修改 data 属性的值，Vue 会自动检测数据变化并更新视图
3. 使用 Vuex 管理状态：Vuex 是一个专为 Vue.js 应用程序开发的状态管理模式，可以方便地集中管理和共享应用程序的所有组件的状态。使用 Vuex，可以将共享的状态（例如用户登录状态、购物车中的商品列表等）存储在一个全局的 Store 中，通过 mutations 来更新 Store 中的数据，同时使用 getters 来获取 Store 中的数据
4. 使用浏览器的本地存储：Vue 项目可以使用浏览器的本地存储，如 localStorage 和 sessionStorage，来保存一些简单的数据，例如用户的配置信息、表单数据等。使用本地存储可以避免因为刷新页面而导致数据丢失的问题
5. 发送请求保存到后端数据库：对于需要长期保存的数据，例如用户注册信息、文章内容等，可以通过发送请求将数据保存到后端数据库中，保证数据的长期存储和可靠性
6. **node是什么？常用来做些什么操作？**

是一个基于Chrome V8 引擎的 JavaScript 运行环境，使 JavaScript 可以在服务器端运行，具有高效、轻量级、跨平台等优势。Node.js 支持事件驱动、非阻塞 I/O 模型，可以处理大量并发连接，是构建高性能、可扩展、分布式、实时应用的首选技术

1. Web 开发：可以使用 Node.js 搭建 Web 服务器，处理客户端请求，构建 Web 应用程序
2. 后端开发：可以使用 Node.js 搭建服务器端应用程序，处理数据库操作，提供 API 接口等
3. 工具开发：Node.js 提供了丰富的模块和 API，可以开发各种工具，如构建工具、自动化测试工具等
4. 网络编程：Node.js 提供了 TCP、UDP、HTTP 等网络编程接口，可以用于网络通信、实现自定义协议等
5. 实时通信：Node.js 通过 WebSocket、Socket.IO 等实现实时通信，如实时聊天、在线游戏等应用
6. 大数据处理：Node.js 可以处理大数据量的计算和分析，如数据挖掘、机器学习等
7. **PM2了解吗？**

是一个 Node.js 应用程序的进程管理工具，它允许您轻松地启动、停止、重启、监视和自动部署 Node.js 应用程序。PM2 可以自动监视您的应用程序，如果应用程序崩溃或由于任何原因停止运行，它会自动重启它。PM2 还提供了日志管理和负载均衡等其他有用的功能

特点：

1. 自动守护进程：如果您的应用程序崩溃或由于任何原因停止运行，PM2 会自动重启它
2. 监视：PM2 可以监视您的应用程序，以确保它始终在运行
3. 热重启：您可以在不停止应用程序的情况下重新加载更改的代码
4. 集群模式：PM2 可以在多个 CPU 上分配应用程序，以实现负载均衡
5. 日志管理：PM2 可以捕获和管理应用程序生成的日志
6. 自动部署：PM2 可以通过简单的命令来自动部署您的应用程序
7. **node中间层了解吗？**

前端应用程序和后端应用程序进行通信、交换数据以及进行数据加工和处理。Node.js 中间层可以根据具体的业务需求，对来自前端的请求进行适当的处理和解析，然后将其转发给后端应用程序进行处理，同时也可以对后端返回的数据进行加工和处理，然后将其返回给前端应用程序

使用场景：

1. 处理请求和响应：Node.js 中间层可以接收来自前端应用程序的请求，并根据具体的业务需求进行处理和解析，然后将请求转发给后端应用程序进行处理。同时，Node.js 中间层也可以对后端返回的数据进行加工和处理，然后将数据返回给前端应用程序
2. 实现业务逻辑：Node.js 中间层可以实现一些通用的业务逻辑，如身份验证、权限控制、数据格式化等，这些业务逻辑可以帮助后端应用程序更加高效地处理请求，并且可以提高前端应用程序的响应速度
3. 提供 API 接口：Node.js 中间层可以提供一些 API 接口，使得前端应用程序可以直接调用后端应用程序的服务，同时也可以将一些通用的业务逻辑封装成 API 接口，方便前端应用程序调用
4. 缓存数据：Node.js 中间层可以通过缓存数据，提高前端应用程序的响应速度，减轻后端应用程序的负载压力

总的来说，Node.js 中间层是一种将前端应用程序和后端应用程序进行连接和协调的软件层，它可以根据具体的业务需求进行定制和开发，以实现更加高效、灵活、可扩展的软件系统

1. 贴两个题：

