第三版：前端 60 道

HTML语义化

HTML 语义化就是让页面内容结构化，它有如下优点

1、 易于用户阅读，样式丢失的时候能让页面呈现清晰的结构。

2、 有利于 SEO，搜索引擎根据标签来确定上下文和各个关键字的权重。

3、 方便其他设备解析，如盲人阅读器根据语义渲染网页

4、 有利于开发和维护，语义化更具可读性，代码更好维护，与 CSS3 关系更和谐

如：

```
<header>代表头部
<nav>代表超链接区域
<main>定义文档主要内容
<article>可以表示文章、博客等内容
<aside>通常表示侧边栏或嵌入内容
<footer>代表尾部
```

HTML5 新标签

```
有<header>、<footer>、<aside>、<nav>、<video>、<audio>、<canvas>等...
```

盒子模型

盒模型分为标准盒模型和怪异盒模型(IE 模型)

```
box-sizing：content-box    //标准盒模型
box-sizing：border-box     //怪异盒模型
```

标准盒模型：元素的宽度等于 style 里的 width+margin+border+padding 宽度

如下代码，整个宽高还是 120px

```
div{
    box-sizing: content-box;
    margin: 10px;
    width: 100px;
    height: 100px;
    padding: 10px;
}
```

怪异盒模型：元素宽度等于 style 里的 width 宽度

![91_4.png][91_4.png]

如下代码，整个宽高还是 100px

```
div{
    box-sizing: border-box;
    margin: 10px;
    width: 100px;
```

```
    height: 100px;
    padding: 10px;
}
```

注意：如果你在设计页面中，发现内容区被撑爆了，那么就先检查一下 border-sizing 是什么，最好在引用 reset.css 的时候，就对 border-sizing 进行统一设置，方便管理

rem 与 em 的区别

rem 是根据根的 font-size 变化，而 em 是根据父级的 font-size 变化

rem：

相对于根元素 html 的 font-size，假如 html 为 font-size：12px，那么，在其当中的 div 设置为 font-size：2rem,就是当中的 div 为 24px

em：

相对于父元素计算，假如某个 p 元素为 font-size:12px,在它内部有个 span 标签，设置 font-size：2em,那么，这时候的 span 字体大小为：12*2=24px

CSS 选择器

css 常用选择器

```
通配符：*
ID 选择器：#ID
类选择器：.class
```

元素选择器：p、a　　等
后代选择器：p span、div a　　等
伪类选择器：a:hover 等
属性选择器：input[type="text"]　等

css 选择器权重

!important -> 行内样式 -> #id -> .class -> 元素和伪元素 -> * -> 继承 -> 默认

CSS 新特性

transition：过渡
transform：旋转、缩放、移动或者倾斜
animation：动画
gradient：渐变
shadow：阴影
border-radius：圆角

行内元素和块级元素

行内元素（display: inline）

宽度和高度是由内容决定，与其他元素共占一行的元素，我们将其叫行内元素，例如：、 <i> 、 <a>等

块级元素（display: block）

默认宽度由父容器决定，默认高度由内容决定，独占一行并且可以设置宽高的元素，我们将其叫做块级元素，例如：<p> 、 、<ul>等

在平时，我们经常使用 CSS 的 display: inline-block，使它们拥有更多的状态

绝对定位和相对定位的区别

position: absolute

绝对定位：是相对于元素最近的已定位的祖先元素

position: relative

相对定位：相对定位是相对于元素在文档中的初始位置

Flex 布局

[juejin.im/post/5d428c...][juejin.im_post_5d428c]

BFC

什么是 BFC?

BFC 格式化上下文，它是一个独立的渲染区域，让处于 BFC 内部的元素和外部的元素相互隔离，使内外元素的定位不会相互影响

如何产生 BFC?

display: inline-block

position: absolute/fixed

BFC 作用

BFC 最大的一个作用就是：在页面上有一个独立隔离容器，容器内的元素和容器外的元素布局不会相互影响

```
解决上外边距重叠;重叠的两个 box 都开启 bfc;
解决浮动引起高度塌陷;容器盒子开启 bfc
解决文字环绕图片;左边图片 div,右边文字容器 p,将 p 容器开启 bfc
```

水平垂直居中

Flex 布局

```
display: flex   //设置 Flex 模式
flex-direction: column   //决定元素是横排还是竖着排
flex-wrap: wrap        //决定元素换行格式
justify-content: space-between   //同一排下对齐方式，空格如何隔开各个元素
align-items: center        //同一排下元素如何对齐
align-content: space-between      //多行对齐方式
```

水平居中

```
行内元素：display: inline-block;
块级元素：margin: 0 auto;
Flex: display: flex; justify-content: center
```

垂直居中

行高 = 元素高：line-height: height
flex: display: flex; align-item: center

less,sass,styus 三者的区别

变量

Sass 声明变量必须是『$』开头，后面紧跟变量名和变量值，而且变量名和变量值需要使用冒号：分隔开。

Less 声明变量用『@』开头，其余等同 Sass。

Stylus 中声明变量没有任何限定，结尾的分号可有可无，但变量名和变量值之间必须要有『等号』。

作用域

Sass：三者最差，不存在全局变量的概念

Less：最近的一次更新的变量有效，并且会作用于全部的引用！

Stylus：Sass 的处理方式和 Stylus 相同，变量值输出时根据之前最近的一次定义计算，每次引用最近的定义有效；

嵌套

三种 css 预编译器的「选择器嵌套」在使用上来说没有任何区别，甚至连引用父级选择器的标记 & 也相同

继承

Sass和 Stylus的继承非常像,能把一个选择器的所有样式继承到另一个选择器上。使用『@extend』开始，后面接被继承的选择器。Stylus 的继承方式来自 Sass，两者如出一辙。 Less 则又「独树一帜」地用伪类来描述继承关系；

导入@Import

Sass 中只能在使用 url() 表达式引入时进行变量插值

```
$device: mobile;
@import url(styles.#{$device}.css);
```

Less 中可以在字符串中进行插值

```
@device: mobile;
@import "styles.@{device}.css";
```

Stylus 中在这里插值不管用，但是可以利用其字符串拼接的功能实现

```
device = "mobile"
@import "styles." + device + ".css"
```

总结

Sass 和 Less 语法严谨、Stylus 相对自由。因为 Less 长得更像 css，所以它可能学习起来更容易。

Sass 和 Compass、Stylus 和 Nib 都是好基友。

Sass 和 Stylus 都具有类语言的逻辑方式处理：条件、循环等，而 Less 需要通过 When 等关键词模拟这些功能，这方面 Less 比不上 Sass 和 Stylus

Less 在丰富性以及特色上都不及 Sass 和 Stylus，若不是因为 Bootstrap 引入了 Less，可能它不会像现在这样被广泛应用（个人愚见）

link 与@import 区别与选择

```
<style type="text/css">
    @import url(CSS 文件路径地址);
</style>
<link href="CSSurl 路径" rel="stylesheet" type="text/css" /
```

link 功能较多，可以定义 RSS，定义 Rel 等作用，而@import 只能用于加载 css；

当解析到 link 时，页面会同步加载所引的 css，而@import 所引用的 css 会等到页面加载完才被加载；

@import 需要 IE5 以上才能使用；

link 可以使用 js 动态引入，@import 不行

多行元素的文本省略号

```
overflow : hidden;
text-overflow: ellipsis;
display: -webkit-box;
-webkit-line-clamp: 3;
-webkit-box-orient: vertical
```

JS 的几条基本规范

1、 不要在同一行声明多个变量

2、 请使用===/!==来比较 true/false 或者数值

3、 使用对象字面量替代 new Array 这种形式

4、 不要使用全局变量

5、 Switch 语句必须带有 default 分支

6、 函数不应该有时候有返回值，有时候没有返回值

7、 For 循环必须使用大括号

8、 IF 语句必须使用大括号

9、 for-in 循环中的变量 应该使用 var 关键字明确限定作用域，从而避免作用域污染

JS 引用方法

行内引入

```
<body>
  <input type="button" onclick="alert('行内引入')" value="按钮"/>
  <button onclick="alert(123)">点击我</button>
```

```
</body>
```

内部引入

```
<script>
  window.onload = function() {
    alert("js 内部引入！");
  }
</script>
```

外部引入

```
<body>
  <div></div>

  <script type="text/javascript" src="./js/index.js"></script>
</body>
```

注意

1，不推荐写行内或者 HTML 中插入<script>,因为浏览器解析顺序缘故，如果解析到死循环之类的 JS 代码，会卡住页面
2，建议在 onload 事件之后，即等 HTML、CSS 渲染完毕再执行代码

JS 的基本数据类型

Undefined、Null、Boolean、Number、String、新增:Symbol

数组操作

在 JavaScript 中，用得较多的之一无疑是数组操作，这里过一遍数组的一些用法

map: 遍历数组，返回回调返回值组成的新数组
forEach: 无法 break，可以用 try/catch 中 throw new Error 来停止
filter: 过滤
some: 有一项返回 true，则整体为 true
every: 有一项返回 false，则整体为 false
join: 通过指定连接符生成字符串
push / pop: 末尾推入和弹出，改变原数组， 返回推入/弹出项【有误】
unshift / shift: 头部推入和弹出，改变原数组，返回操作项【有误】
sort(fn) / reverse: 排序与反转，改变原数组
concat: 连接数组，不影响原数组， 浅拷贝
slice(start, end): 返回截断后的新数组，不改变原数组
splice(start, number, value...): 返回删除元素组成的数组，value 为插入项，改变原数组
indexOf / lastIndexOf(value, fromIndex): 查找数组项，返回对应的下标
reduce / reduceRight(fn(prev, cur)， defaultPrev): 两两执行，prev 为上次化简函数的 return 值，cur 为当前值(从第二项开始)

JS 有哪些内置对象

Object 是 JavaScript 中所有对象的父对象

数据封装对象：Object、Array、Boolean、Number 和 String
其他对象：Function、Arguments、Math、Date、RegExp、Error

get 请求传参长度的误区

误区：我们经常说 get 请求参数的大小存在限制，而 post 请求的参数大小是无限制的

实际上 HTTP 协议从未规定 GET/POST 的请求长度限制是多少。对 get 请求参数的限制是来源与浏览器或 web 服务器，浏览器或 web 服务器限制了 url 的长度。为了明确这个概念，我们必须再次强调下面几点：

1、 HTTP 协议 未规定 GET 和 POST 的长度限制

2、 GET 的最大长度显示是因为 浏览器和 web 服务器限制了 URI 的长度

3、 不同的浏览器和 WEB 服务器，限制的最大长度不一样

4、 要支持 IE，则最大长度为 2083byte，若只支持 Chrome，则最大长度 8182byte

补充 get 和 post 请求在缓存方面的区别

get 请求类似于查找的过程，用户获取数据，可以不用每次都与数据库连接，所以可以使用缓存。

post 不同，post 做的一般是修改和删除的工作，所以必须与数据库交互，所以不能使用缓存。因此 get 请求适合于请求缓存。

闭包

什么是闭包？

函数 A 里面包含了 函数 B，而 函数 B 里面使用了 函数 A 的变量，那么 函数 B 被称为闭包。

又或者：闭包就是能够读取其他函数内部变量的函数

```
function A() {
  var a = 1;
  function B() {
    console.log(a);
  }
  return B();
}
```

闭包的特征

1、 函数内再嵌套函数

2、 内部函数可以引用外层的参数和变量

3、 参数和变量不会被垃圾回收制回收

对闭包的理解

使用闭包主要是为了设计私有的方法和变量。闭包的优点是可以避免全局变量的污染，缺点是闭包会常驻内存，会增大内存使用量，使用不当很容易造成内存泄露。在 js 中，函数即闭包，只有函数才会产生作用域的概念

闭包 的最大用处有两个，一个是可以读取函数内部的变量，另一个就是让这些变量始终保持在内存中

闭包的另一个用处，是封装对象的私有属性和私有方法

闭包的好处

能够实现封装和缓存等

闭包的坏处

就是消耗内存、不正当使用会造成内存溢出的问题

使用闭包的注意点

由于闭包会使得函数中的变量都被保存在内存中，内存消耗很大，所以不能滥用闭包，否则会造成网页的性能问题，在 IE 中可能导致内存泄露

解决方法是：在退出函数之前，将不使用的局部变量全部删除

闭包的经典问题

```
for(var i = 0; i < 3; i++) {
  setTimeout(function() {
    console.log(i);
  }, 1000);
}
```

这段代码输出

答案：3 个 3
解析：首先，for 循环是同步代码，先执行三遍 for，i 变成了 3；然后，再执行异步代码 setTimeout，这时候输出的 i，只能是 3 个 3 了

有什么办法依次输出 0 1 2

第一种方法

使用 let

```
for(let i = 0; i < 3; i++) {
  setTimeout(function() {
    console.log(i);
  }, 1000);
}
```

在这里，每个 let 和代码块结合起来形成块级作用域，当 setTimeout() 打印时，会寻找最近的块级作用域中的 i，所以依次打印出 0 1 2

如果这样不明白，我们可以执行下边这段代码

```
for(let i = 0; i < 3; i++) {
  console.log("定时器外部：" + i);
  setTimeout(function() {
    console.log(i);
  }, 1000);
}
```

此时浏览器依次输出的是：

```
定时器外部：0
定时器外部：1
定时器外部：2
0
1
2
```

即代码还是先执行 for 循环，但是当 for 结束执行到了 setTimeout 的时候，它会做个标记，这样到了 console.log(i) 中，i 就能找到这个块中最近的变量定义

第二种方法

使用立即执行函数解决闭包的问题

```
for(let i = 0; i < 3; i++) {
  (function(i){
    setTimeout(function() {
      console.log(i);
    }, 1000);
  })(i)
}
```

JS 作用域及作用域链

作用域

在 JavaScript 中，作用域分为 全局作用域 和 函数作用域

全局作用域

代码在程序的任何地方都能被访问，window 对象的内置属性都拥有全局作用域

函数作用域

在固定的代码片段才能被访问

例子：

![91_5.png][91_5.png]作用域有上下级关系，上下级关系的确定就看函数是在哪个作用域下创建的。如上，fn 作用域下创建了 bar 函数，那么“fn 作用域”就是“bar 作用域”的上级。

作用域最大的用处就是隔离变量，不同作用域下同名变量不会有冲突。

变量取值：到创建 这个变量 的函数的作用域中取值

作用域链

一般情况下，变量取值到 创建 这个变量 的函数的作用域中取值。

但是如果在当前作用域中没有查到值，就会向上级作用域去查，直到查到全局作用域，这么一个查找过程形成的链条就叫做作用域链

![91_6.png][91_6.png]

原型和原型链

原型和原型链的概念

每个对象都会在其内部初始化一个属性，就是 prototype(原型)，当我们访问一个对象的属性时，如果这个对象内部不存在这个属性，那么他就会去 prototype 里找这个属性，这个 prototype 又会有自己的 prototype，于是就这样一直找下去

原型和原型链的关系

instance.constructor.prototype = instance.__proto__

原型和原型链的特点

JavaScript 对象是通过引用来传递的，我们创建的每个新对象实体中并没有一份属于自己的原型副本。当我们修改原型时，与之相关的对象也会继承这一改变

当我们需要一个属性的时，Javascript 引擎会先看当前对象中是否有这个属性，如果没有的

就会查找他的 Prototype 对象是否有这个属性，如此递推下去，一直检索到 Object 内建对象

组件化和模块化

组件化

为什么要组件化开发

有时候页面代码量太大，逻辑太多或者同一个功能组件在许多页面均有使用，维护起来相当复杂，这个时候，就需要组件化开发来进行功能拆分、组件封装，已达到组件通用性，增强代码可读性，维护成本也能大大降低

组件化开发的优点

很大程度上降低系统各个功能的耦合性，并且提高了功能内部的聚合性。这对前端工程化及降低代码的维护来说，是有很大的好处的，耦合性的降低，提高了系统的伸展性，降低了开发的复杂度，提升开发效率，降低开发成本

组件化开发的原则

1、 专一

2、 可配置性

3、 标准性

4、 复用性

5、 可维护性

模块化

为什么要模块化

早期的 javascript 版本没有块级作用域、没有类、没有包、也没有模块，这样会带来一些问题，如复用、依赖、冲突、代码组织混乱等，随着前端的膨胀，模块化显得非常迫切

模块化的好处

1、 避免变量污染，命名冲突

2、 提高代码复用率

3、 提高了可维护性

4、 方便依赖关系管理

模块化的几种方法

函数封装

```
var myModule = {
    var1: 1,

    var2: 2,

    fn1: function(){

    },

    fn2: function(){

    }
}
```

总结：这样避免了变量污染，只要保证模块名唯一即可，同时同一模块内的成员也有了关系

缺陷：外部可以睡意修改内部成员，这样就会产生意外的安全问题

立即执行函数表达式(IIFE)

```
var myModule = (function(){
    var var1 = 1;
    var var2 = 2;

    function fn1(){

    }
```

```
    function fn2(){

    }

return {
    fn1: fn1,
    fn2: fn2
};
})();
```

总结：这样在模块外部无法修改我们没有暴露出来的变量、函数

缺点：功能相对较弱，封装过程增加了工作量，仍会导致命名空间污染可能、闭包是有成本的

图片的预加载和懒加载

预加载：提前加载图片，当用户需要查看时可直接从本地缓存中渲染

懒加载：懒加载的主要目的是作为服务器前端的优化，减少请求数或延迟请求数

两种技术的本质：两者的行为是相反的，一个是提前加载，一个是迟缓甚至不加载。预加载则会增加服务器前端压力，懒加载对服务器有一定的缓解压力作用。

mouseover 和 mouseenter 的区别

mouseover：当鼠标移入元素或其子元素都会触发事件，所以有一个重复触发，冒泡的过程。对应的移除事件是 mouseout

mouseenter：当鼠标移除元素本身（不包含元素的子元素）会触发事件，也就是不会冒泡，对应的移除事件是 mouseleave

解决异步回调地狱

promise、generator、async/await

对 This 对象的理解

this 总是指向函数的直接调用者（而非间接调用者）

如果有 new 关键字，this 指向 new 出来的那个对象

在事件中，this 指向触发这个事件的对象，特殊的是，IE 中的 attachEvent 中的 this 总是指向全局对象 Window

vue 生命周期

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?

它的生命周期中有多个事件钩子，让我们在控制整个 Vue 实例的过程时更容易形成好的逻辑

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

每个生命周期适合哪些场景？

生命周期钩子的一些使用方法：

beforecreate：可以在这加个 loading 事件，在加载实例时触发

created：初始化完成时的事件写在这里，如在这结束 loading 事件，异步请求也适宜在这里调用

mounted：挂载元素，获取到 DOM 节点

updated：如果对数据统一处理，在这里写上相应函数

beforeDestroy：可以做一个确认停止事件的确认框

nextTick：更新数据后立即操作 dom

v-show 与 v-if 区别

v-show 是 css 切换，v-if 是完整的销毁和重新创建

使用 频繁切换时用 v-show，运行时较少改变时用 v-if

v-if= ‘false’ v-if 是条件渲染，当 false 的时候不会渲染

开发中常用的指令有哪些

v-model :一般用在表达输入，很轻松的实现表单控件和数据的双向绑定

v-html: 更新元素的 innerHTML

v-show 与 v-if: 条件渲染，注意二者区别

使用了 v-if 的时候，如果值为 false，那么页面将不会有这个 html 标签生成
v-show 则是不管值为 true 还是 false，html 元素都会存在，只是 CSS 中的 display 显示或隐藏

v-on：click: 可以简写为@click,@绑定一个事件。如果事件触发了，就可以指定事件的处理函数 v-for:基于源数据多次渲染元素或模板块 v-bind: 当表达式的值改变时，将其产生的连带影响，响应式地作用于 DOM

语法: v-bind:title="msg" 简写: :title="msg"

绑定 class 的数组用法

对象方法 v-bind:class="{'orange': isRipe, 'green': isNotRipe}"

数组方法 v-bind:class="[class1, class2]"

行内 v-bind:style="{color: color, fontSize: fontSize+'px' }"

组件之间的传值通信

父组件给子组件传值

使用 props，父组件可以使用 props 向子组件传递数据

父组件 vue 模板 father.vue

```
<template>
    <child :msg="message"></child>
</template>

<script>
import child from './child.vue';
export default {
    components: {
        child
    },
    data () {
        return {
            message: 'father message';
        }
    }
}
```

```
</script>
```

子组件 vue 模板 child.vue:

```
<template>
    <div>{{msg}}</div>
</template>

<script>
export default {
    props: {
        msg: {
            type: String,
            required: true
        }
    }
}
</script>
```

子组件向父组件通信

父组件向子组件传递事件方法，子组件通过$emit 触发事件，回调给父组件

父组件 vue 模板 father.vue:

```
<template>
    <child @msgFunc="func"></child>
</template>

<script>
import child from './child.vue';
```

```
export default {
    components: {
        child
    },
    methods: {
        func (msg) {
            console.log(msg);
        }
    }
}
</script>
```

子组件 vue 模板 child.vue:

```
<template>
    <button @click="handleClick">点我</button>
</template>

<script>
export default {
    props: {
        msg: {
            type: String,
            required: true
        }
    },
    methods () {
        handleClick () {
            //.....
            this.$emit('msgFunc');
        }
```

```
    }
}
</script>
```

非父子，兄弟组件之间通信

可以通过实例一个 vue 实例 Bus 作为媒介，要相互通信的兄弟组件之中，都引入 Bus，然后通过分别调用 Bus 事件触发和监听来实现通信和参数传递

Bus.js 可以是这样：

```
import Vue from 'vue'
export default new Vue()
```

在需要通信的组件都引入 Bus.js:

```
<template>
    <button @click="toBus">子组件传给兄弟组件</button>
</template>

<script>
import Bus from '../common/js/bus.js'
export default{
    methods: {
        toBus () {
            Bus.$emit('on', '来自兄弟组件')
        }
    }
}
</script>
```

另一个组件也 import Bus.js 在钩子函数中监听 on 事件

```
import Bus from '../common/js/bus.js'
export default {
    data() {
      return {
        message: ''
      }
    },
    mounted() {
        Bus.$on('on', (msg) => {
          this.message = msg
        })
     }
  }
```

路由跳转方式

<router-link to='home'> router-link 标签会渲染为<a>标签，咋填 template 中的跳转都是这种；

另一种是编程是导航 也就是通过 js 跳转 比如 router.push('/home')

MVVM

M - Model，Model 代表数据模型，也可以在 Model 中定义数据修改和操作的业务逻辑

V - View，View 代表 UI 组件，它负责将数据模型转化为 UI 展现出来

VM - ViewModel，ViewModel 监听模型数据的改变和控制视图行为、处理用户交互，简单理解就是一个同步 View 和 Model 的对象，连接 Model 和 View

computed 和 watch 有什么区别?

computed:

1、 computed 是计算属性,也就是计算值,它更多用于计算值的场景

2、 computed 具有缓存性,computed 的值在 getter 执行后是会缓存的，只有在它依赖的属性值改变之后，下一次获取 computed 的值时才会重新调用对应的 getter 来计算

3、 computed 适用于计算比较消耗性能的计算场景

watch:

1、 更多的是「观察」的作用,类似于某些数据的监听回调,用于观察 props $emit 或者本组件的值,当数据变化时来执行回调进行后续操作

2、 无缓存性，页面重新渲染时值不变化也会执行

小结:

1、 当我们要进行数值计算,而且依赖于其他数据，那么把这个数据设计为 computed

2、 如果你需要在某个数据变化时做一些事情，使用 watch 来观察这个数据变化

key

key 是为 Vue 中的 vnode 标记的唯一 id，通过这个 key，我们的 diff 操作可以更准确、更快速

准确:

如果不加 key,那么 vue 会选择复用节点(Vue 的就地更新策略),导致之前节点的状态被保留下来，会产生一系列的 bug

快速:

key 的唯一性可以被 Map 数据结构充分利用

组件中的 data 为什么是函数?

为什么组件中的 data 必须是一个函数，然后 return 一个对象，而 new Vue 实例里，data 可以直接是一个对象?

```
// data
data() {
  return {
    message: "子组件",
    childName:this.name
  }
}

// new Vue
new Vue({
  el: '#app',
```

```
  router,
  template: '<App/>',
  components: {App}
})
```

因为组件是用来复用的，JS 里对象是引用关系，这样作用域没有隔离，而 new Vue 的实例，是不会被复用的，因此不存在引用对象问题

Class 与 Style 如何动态绑定?

Class 可以通过对象语法和数组语法进行动态绑定：

对象语法

```
<div v-bind:class="{ active: isActive, 'text-danger': hasError }"></div>

data: {
  isActive: true,
  hasError: false
}
```

数组语法

```
<div v-bind:class="[isActive ? activeClass : '', errorClass]"></div>

data: {
  activeClass: 'active',
  errorClass: 'text-danger'
}
```

Style 也可以通过对象语法和数组语法进行动态绑定：

对象语法

```
<div v-bind:style="{ color: activeColor, fontSize: fontSize +
'px' }"></div>

data: {
  activeColor: 'red',
  fontSize: 30
}
```

数组语法

```
<div v-bind:style="[styleColor, styleSize]"></div>

data: {
  styleColor: {
      color: 'red'
   },
  styleSize:{
      fontSize:'23px'
  }
}
```

vue 的单项数据流

所有的 prop 都使得其父子 prop 之间形成了一个单向下行绑定：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

额外的，每次父级组件发生更新时，子组件中所有的 prop 都将会刷新为最新的值。这意味着你不应该在一个子组件内部改变 prop。如果你这样做了，Vue 会在浏览器的控制台中发出警告。子组件想修改时，只能通过 $emit 派发一个自定义事件，父组件接收到后，由父组件修改

有两种常见的试图改变一个 prop 的情形 ：

这个 prop 用来传递一个初始值；这个子组件接下来希望将其作为一个本地的 prop 数据来使用

在这种情况下，最好定义一个本地的 data 属性并将这个 prop 用作其初始值：

```
props: ['initialCounter'],
data: function () {
  return {
    counter: this.initialCounter
  }
}
```

这个 prop 以一种原始的值传入且需要进行转换

在这种情况下，最好使用这个 prop 的值来定义一个计算属性

```
props: ['size'],
computed: {
  normalizedSize: function () {
    return this.size.trim().toLowerCase()
  }
}
```

keep-alive

keep-alive 是 Vue 内置的一个组件，可以使被包含的组件保留状态，避免重新渲染，其有以下特性：

1、 一般结合路由和动态组件一起使用，用于缓存组件；

2、 提供 include 和 exclude 属性，两者都支持字符串或正则表达式， include 表示只有名称匹配的组件会被缓存，exclude 表示任何名称匹配的组件都不会被缓存 ，其中 exclude 的优先级比 include 高；

3、 对应两个钩子函数 activated 和 deactivated ，当组件被激活时，触发钩子函数 activated，当组件被移除时，触发钩子函数 deactivated。

v-model 的原理

vue 项目中主要使用 v-model 指令在表单 input、textarea、select 等元素上创建双向数据绑定，我们知道 v-model 本质上不过是语法糖，v-model 在内部为不同的输入元素使用不同的属性并抛出不同的事件：

1、 text 和 textarea 元素使用 value 属性和 input 事件；

2、 checkbox 和 radio 使用 checked 属性和 change 事件；

3、 select 字段将 value 作为 prop 并将 change 作为事件；

以 input 表单元素为例：

```
<input v-model='something'>
```

相当于

```
<input v-bind:value="something" v-on:input="something = $event.target.value">
```

如果在自定义组件中，v-model 默认会利用名为 value 的 prop 和名为 input 的事件，如下所示：

```
父组件:
<ModelChild v-model="message"></ModelChild>

子组件:
<div>{{value}}</div>

props:{
    value: String
},
methods: {
  test1(){
     this.$emit('input', '小红')
  },
},
```

nextTick()

在下次 DOM 更新循环结束之后执行延迟回调。在修改数据之后，立即使用的这个回调函数，获取更新后的 DOM

```
// 修改数据
```

```
vm.msg = 'Hello'
// DOM 还未更新
Vue.nextTick(function () {
  // DOM 更新
})
```

vue 插槽

单个插槽

1、 当子组件模板只有一个没有属性的插槽时，
2、 父组件传入的整个内容片段将插入到插槽所在的 DOM 位置，
3、 并替换掉插槽标签本身

命名插槽

1、 solt 元素可以用一个特殊的特性 name 来进一步配置如何分发内容。
2、 多个插槽可以有不同的名字。 这样可以将父组件模板中 slot 位置，
3、 和子组件 slot 元素产生关联，便于插槽内容对应传递

作用域插槽

1、 可以访问组件内部数据的可复用插槽(reusable slot)
2、 在父级中，具有特殊特性 slot-scope 的 元素必须存在，
3、 表示它是作用域插槽的模板。slot-scope 的值将被用作一个临时变量名，
4、 此变量接收从子组件传递过来的 prop 对象

vue-router 有哪几种导航钩子

第一种：是全局导航钩子：router.beforeEach(to,from,next)，作用：跳转前进行判断拦截

第二种：组件内的钩子

第三种：单独路由独享组件

vuex

vuex 是什么？

vuex 就是一个仓库，仓库里放了很多对象。其中 state 就是数据源存放地，对应于一般 vue 对象里面的 data

state 里面存放的数据是响应式的，vue 组件从 store 读取数据，若是 store 中的数据发生改变，依赖这相数据的组件也会发生更新

它通过 mapState 把全局的 state 和 getters 映射到当前组件的 computed 计算属性

Vuex 有 5 种属性：分别是 state、getter、mutation、action、module;

state

Vuex 使用单一状态树,即每个应用将仅仅包含一个 store 实例，但单一状态树和模块化并不冲突。存放的数据状态，不可以直接修改里面的数据

mutations

mutations 定义的方法动态修改 Vuex 的 store 中的状态或数据

getters

类似 vue 的计算属性，主要用来过滤一些数据

action

actions 可以理解为通过将 mutations 里面处里数据的方法变成可异步的处理数据的方法，简单的说就是异步操作数据。view 层通过 store.dispath 来分发 action

总结

vuex 一般用于中大型 web 单页应用中对应用的状态进行管理，对于一些组件间关系较为简单的小型应用，使用 vuex 的必要性不是很大，因为完全可以用组件 prop 属性或者事件来完成父子组件之间的通信，vuex 更多地用于解决跨组件通信以及作为数据中心集中式存储数据

你有对 Vue 项目进行哪些优化?

代码层面的优化

v-if 和 v-show 区分使用场景
computed 和 watch 区分使用场景
v-for 遍历必须为 item 添加 key，且避免同时使用 v-if
长列表性能优化
事件的销毁
图片资源懒加载
路由懒加载
第三方插件的按需引入
优化无限列表性能
服务端渲染 SSR or 预渲染

Webpack 层面的优化

Webpack 对图片进行压缩
减少 ES6 转为 ES5 的冗余代码
提取公共代码
模板预编译
提取组件的 CSS
优化 SourceMap
构建结果输出分析
Vue 项目的编译优化

基础的 Web 技术的优化

开启 gzip 压缩
浏览器缓存
CDN 的使用
使用 Chrome Performance 查找性能瓶颈