前端开发

HTML

文档结构

`doc--document` 、 `html--html5` 告诉浏览器要按照 `html5` 标准文档来解释下面的代码

`<head>` 一般放置整篇文档的配置信息

```
<!DOCTYPE html>
<html>
    <!-- 代表整篇文档开始 -->
<head>
    <!-- 文档的头部 -->
</head>
<body>
    <!-- 浏览器窗口显示内容 -->
</body>
</html>
<!-- 代表整篇文档结束 -->
```

meta标签

> 存储的是网页的元数据，搜索引擎会通过查找meta值来给网页分类，是判断网页内容的基础

- charset 属性

  UTF-8 是全球通用的**国际标准编码**

  GB2312 一般是**简体中文**

  GBK 扩展了**繁体中文**，适合香港台湾

- 移动端

  `width`：可以指定的一个值，如 600。 `device-width` 为设备的宽度

  `initial-scale`：初始缩放比例，也即是当页面第一次 load 的时候缩放比例

  `maximum-scale`：允许用户缩放到的最大比例

  `minimum-scale`：允许用户缩放到的最小比例

  `user-scalable`：用户是否可以手动缩放

  `X-UA-Compatible` 是针对 IE8 新加的一个设置，对于 IE8 之外的浏览器是不识别的。为了避免制作出的页面在IE8下面出现错误，直接将 IE8 以最高级别的可用模式显示内容。

```
<title>整篇文档的标题</title>
<meta charset="UTF-8">
<!-- 设置字符集 -->
<meta charset="gb2312">
<meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0, user-scalable=no"/>
<meta http-equiv="X-UA-Compatible" content="ie=edge">
```

标签

单标签（6）

`<img>` `<input>` `<link>` `<br>` `<hr>` `<meta>`

块状元素

`display: block`

`<div>` `<h1>` `<h6>` `<ul>` `<ol>` `<li>` `<dl>` `<dt>` `<dd>` `<hr>` `<ul>` `<p>` `<tabel>` `<thead>` `<tbody>` `<tfoot>`

内联元素

`display: inline`

`<a>` `<b>` `<i>` `<em>` `<span>` `<label>` `<strong>` `<img>` `<input>` `<tr>` `<th>` `<td>`

内联块状元素

`display: inline-block`

特殊标识符

` ` `&copy;` `<` (<) `>` (>)

锚点定点跳转

`<a href="#bottom">到大白鹅</a>`

`<a name="bottom">我是大白鹅</a>`

h1-6标签

一个网页只能有一个 `h1` 标签，一般用于网页大标题，过多的 `h1` 标签会被浏览器认为是在恶意提高权重。

文字修饰效果

加粗：**加粗 加粗**

倾斜：*倾斜 倾斜*

下划线：<u>下划线</u>

删除线：~~删除线~~

地址线：

地址线

上标：x^2

下标：x_2

段落缩进

```
<blockquote>hello
```

预格式化

```
 sdv    wids           avkn   sd
```

忽略 html 标记

定义列表

```
dd{
    direction: rtl; // ltr | rtl | inherit
}
```

```
<dl>
    <dt>前端工程师技术栈</dt> // margin: 0
    <dd>html5+css+js</dd> // margin-inline-start: 40px
    <dd>angular react vue</dd>
    <dd>git svn</dd>
</dl>
```

writing-mode

- 定义了文本在水平或垂直方向上如何排布
  - horizontal-tb：水平方向自上而下的书写方式
  - vertical-rl：垂直方向自右而左的书写方式
  - vertical-lr：垂直方向内内容从上到下，水平方向从左到右
  - sideways-rl：内容垂直方向从上到下排列
  - sideways-lr：内容垂直方向从下到上排列

表单

```
<form name="" action="" method="get">
    1.输入文本框:
    <input type="text" name="" value="你好吗" size="150"
    maxlength="15"><br>
    2.密码:
```

```
    <input type="password" name="" size="30"><br>
    3.单选按钮:
    <input type="radio" name="sex" value="nan">男
    <input type="radio" name="sex" value="nv" checked>女<br>
    4.复选按钮:
    <input type="checkbox" name="">语文
    <input type="checkbox" name="">数学
    <input type="checkbox" name="">生物
    <input type="checkbox" name="" checked>历史<br>
    5.按钮:
    <input type="button" name="" value="按钮"><br>
    6.单选下拉菜单:
    <select>
        <option value="">杭州</option>
        <option value="">北京</option>
        <option value="" selected>上海</option>
        <option value="">天津</option>
    </select><br>
    7.多选列表:
    <select multiple>
        <option value="">杭州</option>
        <option value="">北京</option>
        <option value="">上海</option>
        <option value="" selected>天津</option>
    </select><br>
    8.文件域:
    <input type="file" name=""><br>
    9.文本域:
    <textarea cols="50" rows="10"></textarea>
    <input type="reset" name="" value="哈哈重置按钮">
    <input type="submit" name="" value="提交哈哈"><br>
    10.图像域:
    <input type="image" name="" src="img/flower7.jpg"><br>
    11.隐藏域:
    <input type="hidden" name="" value="100"><br>
    12.验证URL格式:
    <input type="url" name="url_name">
    13.添加数字类型的数字框:
    <input type="number" name="" step="5" min="5" max="20" value="-5">
    14.音量控制:
    <input type="range" name="" min="0" max="10" step="1" value="-1">
    15.日期数据:
    Date:
    <input type="date" name="">
    month :
    <input type="month" name="">
    week:
    <input type="week" name="">
    time:
    <input type="time" name="">
    datetime:
    <input type="datetime" name="">
    datetime-local:
```

```
    <input type="datetime-local" name="">
    16.请选择颜色：
    <input type="color" name="">
    17.搜索文本框
    <input type="search" name="search_name">
    18.datalist列表
    <input type="text" list="mydata" name="" placeholder="--计算机--">
    <datalist id="mydata">
        <option label="top1" value="计算机基础"></option>
        <option value="XML实例欣赏"></option>
        <option label="top1" value="Java应用实例"></option>
        <option label="top1" value="Flash学习指南"></option>
        <option label="top1" value="HTML5+CSS3"></option>
    </datalist>
</form>
```

```
// 不验证：
<form novalidate>
    E-mail:
    <input type="email" name="email">
    <input type="submit" name="">
</form>
```

表格

`<table>` `<thead>` `<tr>` `<th>` `<tbody>` `<td>` `<tfoot>`

`thead` 元素应该与 `tbody` 和 `tfoot` 元素结合起来使用。

```
<table border="1">
  <thead>
    <tr>
      <th>Month</th>
      <th>Savings</th>
    </tr>
  </thead>

  <tfoot>
    <tr>
      <td>Sum</td>
      <td>$180</td>
    </tr>
  </tfoot>

  <tbody>
    <tr>
      <td>January</td>
      <td>$100</td>
    </tr>
    <tr>
```

```
      <td>February</td>
      <td>$80</td>
    </tr>
  </tbody>
</table>
```

Browser

五大主流浏览器

- IE
- 内核 Trident
- Firefox
- 内核

html5 新增

- 语义化标签
- 。。。。。

CSS

基本css样式

display

浮动

css3

flex

值	描述
flex-grow	一个数字，规定项目将相对于其他灵活的项目进行扩展的量。
flex-shrink	一个数字，规定项目将相对于其他灵活的项目进行收缩的量。
flex-basis	项目的长度。合法值："auto"、"inherit" 或一个后跟 "%"、"px"、"em" 或任何其他长度单位的数字。
auto	与 1 1 auto 相同。
none	与 0 0 auto 相同。
initial	设置该属性为它的默认值，即为 0 1 auto。请参阅 *initial*。
inherit	从父元素继承该属性。请参阅 *inherit*。

属性选择器

- a[href] 所有的a 标签，这个a需要有 href 属性
- a[href='demo'] 所有的a 标签，这个a需要有 href=demo
- a[href*='demo'] 包含demo
- a[href^='demo'] 以demo 开始
- a[href$='demo'] demo 结束

伪类选择器

- a:active a:link a:hover a:visited
- li:first-child
- li:last-child
- li:nth-child() 根据n 去取值,索引是从1开始计算
- li:nth-last-child 从后向前计算，倒着算 n 的用法. 取值0,1,2,3,4...
- div:empty 选中没有子元素的div
- div:target 结合锚点进行使用，处于当前锚点的元素会被选中

伪元素选择器

:before 和 ::before 写法是等效的

- :before 添加一个子元素在最前面
- :after 添加一个子元素在最后面
- first-letter 选中第一个字或者字母。
- first-line 选中第一行
- ::selection 可改变选中文本的样式

透明度

- rgba （ red，green，blue，Alpha 透明通道）
- hsla（Hue 色调，Saturation 饱和度，Lightness 亮度，Alpha 透明度）

文本阴影

- text-shadow: 1px 1px 1px red;
- 第一个向右移动，负值的是向左移动（正值向右偏移，负值向左偏移）
- 第二个向下移动 ，负值是向上移动(正值是向下偏移，负值向上偏移)
- 第三个代表的是模糊度，不能为负数，值越大，越模糊
- 第四个red 代表模糊的颜色. 影子的颜色

盒模型

- 我们可以改变盒子计算宽高的方式，通过设置盒子的这个属性
- box-sizing: content-box | border-box | inherit
- content-box 盒子的宽度=width+padding+border
- border-box 盒子的宽度=width

边框圆角

- border-radius
- box-shadow: h-shadow v-shadow blur spread color inset
- border-image: 可以为边框设置图片
    - border-image-slice 裁剪的方式
    - border-image-width 边框的宽度
    - border-image-repeat:round,stretch,repeat
    - border-image-source 边框图片的路径

渐变

- **线性渐变:**
    - liner-gradient(135deg, yellow 25%, green 50%)
    - linear-gradient(to right, red , blue)
    - linear-gradient(to bottom right, red , blue)
- **径向渐变：**
    - radius-gradient(100px at center center, yellow, green)
    - 半径 at 中心点的位置 起始颜色 终止颜色

背景

- **背景尺寸：**

    - background-size: contain cover
    - contain: 背景图片始终完全显示，等比例缩放.
    - cover: 也会缩放，背景始终填充整个区域
- **背景原点：**
    - background-origin: 可以设置背景图片的位置.
    - background-origin: border-box, padding-box, content-box
    - border-box 背景图像相对于边框盒来定位
    - padding-box 背景图像相对于内边距框来定位
    - content-box 背景图像相对于内容框来定位
- **背景裁剪：**
    - background-clip: border-box, padding-box, content-box
    - border-box 背景被裁剪到边框盒
    - padding-box 背景被裁剪到内边距框
    - content-box 背景被裁剪到内容框
- **多重背景：**
    - url("images/bg5.png") center center no-repeat

过渡

transition 属性

- transition-property 需要过渡的属性
- transition-duration 过渡这些属性需要执行的时间
- transition-timing-function 过渡的速度
- transition-delay 延迟多少秒

2D—3D转换

- 2d 转换
    - **translate()**
    - **rotate()**
    - **scale()**
    - skew()
    - matrix()
- 3d 转换
    -

动画

animation 属性:

- 动画执行的时间
- 动画的速度
- 延迟时间

- 循环的次数
- 动画的结束状态: animation-fill-mode : none | forwards | backwards | both;
  - forwards
    - 当动画完成后，保持最后一个属性值
  - backwards
    - 在 animation-delay 到动画显示之前，应用开始属性值（在第一个关键帧中定义）
  - 动画的状态
    - animation-play-state: paused | running

JavaScript

ECMAScript

数据类型

- Number
  - 数值转换

    `Number()`

```
console.log(Number(100));//100
console.log(Number(100.5));//100.5
console.log(Number(-1.5));//-1.5
console.log(Number(null));//0
console.log(Number(undefined));//NaN Not a Number
// 如果是字符串
console.log(Number('abc'));//NaN
console.log(Number(''));//0
console.log(Number('123'));//123
console.log(Number('123+123'));//NaN
console.log(Number('070'));//56 忽略八进制前导零0  转成十进制  !!!!
console.log(Number('0xA'));//10 不忽略十六进制0x  转成十进制  !!!!
console.log(Number('070.5'));//70.5
console.log(Number('0xA.5'));//NaN 十六进制没有小数
```

    `parseInt()` 、 `parseFloat()`

```
// parseInt()
console.log(parseInt('123.5ss1s'));//123
console.log(parseInt('ss123'));//NaN
console.log(parseInt(070));//56   !!!!
console.log(parseInt(0xA));//10   !!!!
console.log(parseInt('070', 8));//56  转成十进制
console.log(parseInt('0xA', 16));//10  转成十进制
console.log(parseInt('0xB', 16));//11
console.log(parseInt(070, 8));//46  转成十进制
console.log(parseInt(0xA, 16));//16  转成十进制
console.log(parseInt(0xB, 16));//17  ????
console.log(parseFloat('123.5ss1s'));//123.5
```

- String
  - `String()`、`toString()`

```
// String()、toString()两种方法
var a = String(123);//能将任何类型转换成字符串
console.log(a);//'123'

var b = true;
console.log(b.toString());//"true"

var c = 16;
console.log(c.toString(16));//十六进制基地 十转成十六 10
console.log(c.toString(8));//八进制基地 十转成八 20

// 二进制转成十六进制？
```

- Boolean
  - **undefined、null、NaN、false、0、""** 转换成 boolean 值均为 false
  - `&&`、`||`、`!`
- Null

```
var car = null
console.log(typeof(car)) // object
```

- Undefined

  undefined 和 null 都不能 `.toString()` 转换

```
console.log(null==undefined)//true undefined派生于null
console.log(null===undefined)//false
```

- Object
  - Array、Object、Funtion
- Symbol

```
let name = Symbol()
{
    var person = {}
    person[name] = 'file1'
    console.log(person[name])//file1
}
{
    let name = Symbol()
    person[name] = 'file2'
    console.log(person[name])//file2
}
console.log(person[name])//file1
console.log(person)//Symbol(): "file1" Symbol(): "file2"
```

typeof

检测基本数据类型，会返回六种数据类型：

`undefined`、`string`、`number`、`boolean`、`function`、`object`

instanceof

```
// 检测引用数据类型
var a = new Error()
a instanceof Error //true
```

内置对象

`Array` `Object` `function` `Date` `Math` `Error` `RegExp` `Global` `String` `Number` `Boolean`

数组方法

`push()`、`pop()` --- 对于最后一个进行操作

```
var arr = new Array();
var arr = [11, 23, 43, 10, 78, 90]
console.log(arr.pop());//435 arr = [11, 23, 43, 10, 78]
console.log(arr.push(200));//6 arr = [11, 23, 43, 10, 78, 200]
```

`shift()`、`unshift()` --- 对第一个元素进行操作

```
var arr = [11, 23, 43, 10, 78, 90]
console.log(arr.unshift('添加项'));//7 arr = ["添加项", 11, 23, 43, 10, 78, 90]
console.log(arr.shift());//"添加项" arr = [11, 23, 43, 10, 78, 90]
```

`reverse()` --- 逆序

```
var arr = [11, 23, 43, 10, 78, 90]
console.log(arr.reverse());//arr = [90, 78, 10, 43, 23, 11]
```

`sort()` --- 排序

```
var arr = [11, 23, 43, 10, 78, 90]
console.log(arr.sort());
for(var i = 0;i < arr.length - 1;i++){
    for(var j = 0;j < arr.length - 1 - i ;j++){
        if(arr[i] > arr[j]){
            min = arr[j];
            arr[j] = arr[i];
            arr[i] = min;
        }
        count = count + 1;
    }
}
console.log(arr);
```

`concat()` --- 扩充

```
var arr = [11, 23, 43, 10, 78, 90]
var arr2 = arr.concat('111', [7, 14]);
console.log(arr2);//[11, 23, 43, 10, 78, 90, "111", 7, 14]
```

`slice()` --- 截取 [起始点 结束点)

```
var arr = [11, 23, 43, 10, 78, 90]
var arr2 = arr.slice(0, 3);
console.log(arr2);//arr = [11, 23, 43]
```

`splice()` --- [操作的位置 删除的项数 | 插入(替换)的项 项 项 项 项]

```
var arr = [11, 23, 43, 10, 78, 90]
var arr2 = arr.splice(1, 2, '1', '1');
console.log(arr2);//[23, 43] 删除的内容
console.log(arr);//arr = [11, 1, 1, 10, 78, 90]
```

`indexOf()` --- 从前向后查找 --- 两个参数：[查找项 和 起始点0 1 2]

```
var arr = [1, 2, 3, 4, 5, 6, 7];
console.log(arr.indexOf(2));//1
console.log(arr.indexOf(0));//-1 找不到的
console.log(arr.indexOf(7, 2);//返回查找项的数组索引位置 6
```

`lastIndexOf()` --- 从后向前查找 --- 两个参数：[查找项 和 起始点0 1 2...]

```
var arr = ["Ads", "Bds", "ccc", "Dsdf", "asdf", "ccc", "re", "sdf"];
console.log(arr.indexOf("ccc", 3));// 5
console.log(arr.lastIndexOf("ccc", 6));//5
```

字符串方法

`charAt()` --- 返回指定位置的字符 [index]

```
var str = "Stephanie"
console.log(str.charAt(0))//S
```

`charCodeAt()` --- 返回指定位置的字符编码 [index]

```
var str = "Stephanie"
console.log(str.charCodeAt(0))//83
```

`concat()` --- 拼接

```
var str = "Stephanie"
console.log(str.concat(" Hi"))//"Stephanie Hi"
console.log(str)//"Stephanie"
```

`slice()` --- 截取 [起始点 结束点)

```
var str = "Stephanie"
console.log(str.slice(3, 4))//p
console.log(str.slice(-3))//nie str.length-3
console.log(str.slice(3, -4))//ph str.slice(3, 5)
```

`substring()` --- 截取 [起始点 结束点)

```
var str = "Stephanie"
console.log(str.substring(3, 4))//p
console.log(str.substring(-3))//Stephanie 没有负数
console.log(str.substring(3, -4))//Ste str.substring(0, 3)
```

`substr()` --- 截取 [起始点 返回个数]

```
var str = "Stephanie"
console.log(str.substr(3, 7))//phan
console.log(str.substr(-3))//nie str.length-3
console.log(str.substr(3, -4))//""
```

`indexOf()`、`lastIndexOf()`

对象

什么是对象？..... 属性和方法的集合 封装 继承 多态

- 作用域
  - 预编译
  - 变量提升
  - 自执行函数
- 闭包

    - 什么是闭包
    - this
    - call、apply、bind
    - 解决内存泄漏
- 原型、原型链
    - `__proto__`、 `prototype`、`constructor`
- 原型、原型链

DOM

选择器

```
document.getElementById('')
document.getElementsByTagName('')[0]
document.getElementsByClassName('')[0]
document.getElementsByName('')[0]
document.querySelector('')
document.querySelectorAll('')
```

节点 (12)

- 元素节点 (1)
- 属性节点 (2)
- 文本节点 (3)
- 注释节点 (8)
- 文档节点 (9)

事件

- 阻止冒泡事件，事件冒泡与捕获
- 阻止默认事件
- 普通事件和事件监听，IE的事件监听

BOM

window

location

navigator

screen

history

COOKIE

local storage

session storage

cookie

session

AJAX

AJAX 是什么

AJAX 的全称是 `Asynchronous JavaScript and XML`，它是一种基于 JavaScrip t的网页应用技术。传统的提交方式会重载整个网页，而利用 AJAX 技术可以使J avaScript 与 Web 服务器异步传输数据，从而实现不重载整个页面的情况下，更新局部页面局部内容。

AJAX 实现过程

① 实例化 XMLHTTPRequest() 对象

XMLHttpRequest 是 AJAX 的基础，XMLHTTPRequest() 是核心对象，首先要实例化一个 XMLHTTPRequest() 对象

```
var xhr = new XMLHTTPRequest();
```

② AJAX 进行请求，规定请求的类型、URL 以及是否异步处理请求。

```
xhr.open(method, url, async);
```

`open()` 方法接收三个参数：

method：请求的类型；GET 或 POST

url：文件在服务器上的位置

async：true（异步）或 false（同步）

AJAX 的原理就是 Asynchronous 异步的，所以第三个参数 async 为 true。

第一个参数method决定了请求类型，即传输数据的方式。

GET

与POST相比，GET 更简单也更快。但是传输数据时会将数据放在地址栏的后面，对客户端而言不安全。除此之外，GET传输数据的大小受限，一般只有 2k-8k ，因浏览器而异。所以在传输不敏感信息并且传输文件小的情况下，我们可以选择用GET方式传输。

POST

POST传输是传输数据体，是隐式的，相对客户端较为安全。但是从另一层面上来说，相对服务器端就有一定的风险了。POST传输数据没有大小限制，但是服务器对上传的数据有限制，需要手动修改。

③ AJAX向服务器发送请求

```
xhr.send(string);
xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
```

用GET方法传输时，send(null)，用POST方法传输时，需要用setRequestHeader方法设置请求头。

④ AJAX响应服务器状态

AJAX响应服务器状态涉及XMLHttpRequest对象的三个重要的属性：onreadystatechange、readyState、status。

readyState存有XMLHttpRequest的状态。从0到4发生变化：

0: 请求未初始化

1: 服务器连接已建立

2: 请求已接收

3: 请求处理中

4: 请求已完成，且响应已就绪

每当readyState改变时，就会触发onreadystatechange事件，在事件中判断请求是否成功，响应是否就绪，当readyState等于4且状态为200时，表示响应已就绪：

```
xhr.onreadystatechange = function () {
    if (xhr.readyState === 4) {
        if (xhr.status === 200) {
            obj.success(xhr.responseText);
        }
    }
}
```

⑤ 使用 Callback 函数

```
obj.success(xhr.responseText);
```

AJAX响应就绪后，对接收到的数据进行后续的操作。

AJAX响应就绪后，对接收到的数据进行后续的操作。

可以看出调用的形式与jQuery的实现原理相似。

Ajax原生封装

```
var ajax = {
    create: function () {
        var xhr;
        if (XMLHttpRequest) {
            xhr = new XMLHttpRequest();
        } else{
            xhr = new ActiveXObject('Microsoft.XMLHTTP');
        }
        return xhr;
```

```
    },
    request: function (obj) {
        var xhr = this.create();
        var data = this.changeData(obj.data);
        if (obj.type === 'GET') {
            xhr.open(obj.type, obj.url+'?rand='+Math.random()+'&'+data, true);
            xhr.send(null);
        } else if (obj.type === 'POST') {
            xhr.open(obj.type, obj.url, true);
            xhr.setRequestHeader('Content-Type', 'application/x-www-form-
urlencoded')
            xhr.send(data);
        }
        xhr.onreadystatechange = function () {
            if (xhr.readyState === 4) {
                if (xhr.status === 200) {
                    obj.success(xhr.responseText);
                }
            }
        }
    },
    changeData: function (data) {
        var arr = [];
        for (var i in data) {
            arr.push(i+'='+data[i]);
        }
        return arr.join('&');
    }
}
// 调用
ajax.request({
    type: 'POST',
    url: 'weibo.php',
    data: 'act=update',
    success: function(res){
        console.log(res);
    }
});
```

ES6

let

let为 JavaScript 新增了块级作用域。使得立即执行函数表达式不再必要。

暂时性死区的：只要一进入当前作用域，所要使用的变量就已经存在了，但是不可获取，只有等到声明变量的那一行代码出现，才可以获取和使用该变量。

let**不允许**在相同作用域内，**重复声明**同一个变量。也不能重新声明参数。

ES6明确允许在块级作用域之中声明函数，但是浏览器要兼容老版本，考虑到环境导致的行为差异太大，应该避免在块级作用域内声明函数。如果确实需要，也应该写成函数表达式，而不是函数声明语句。

const

const声明一个只读的常量，且具有块级作用域，不提升，存在暂时性死区。**一旦声明，常量的值就不能改变**。所以一旦声明变量，就**必须立即初始化**。

const对于复合类型的数据，只能保证指针指向的地址是固定的，它的数据结构是可变的。解决方法：用Object.freeze({})冻结对象，就不可给对象添加属性方法。

```
const foo = Object.freeze({});
// 常规模式时，下面一行不起作用；
// 严格模式时，该行会报错
foo.prop = 123;
```

声明变量的方法有`var`、`function`、`let`、`const`、`import`、`class`。

let命令、const命令、class命令声明的全局变量，不属于顶层对象的属性。

结构赋值

是一种"模式匹配"的写法，只要等号两边的模式相同，左边的变量就会被赋予对应的值。解构赋值允许指定默认值。解构不仅可以用于数组，还可以用于对象。

对象的解构与数组有一个重要的不同。数组的元素是按次序排列的，变量的取值由它的位置决定；而对象的属性没有次序，变量必须与属性同名，才能取到正确的值。

对象的解构赋值的内部机制，是先找到同名属性，然后再赋给对应的变量。真正被赋值的是后者，而不是前者。

```
let {a: A=1, b=2} = {a: 10}
console.log(A) // 10
console.log(b) // 2
```

```
let {floor, pow, random} = Math
let a = 2.1
console.log(floor(a))
console.log(pow(2, 3))
```

```
let {length} = 'Steph'
console.log(length)
```

```
var arr = [1, 2]
var obj = {b: 1}
function test({a = 5, b = 3}) {
    console.log(a, b)
}
test(arr)
```

解构赋值的规则是，只要等号右边的值不是对象或数组，就先将其转为对象。由于undefined和null无法转为对象，所以对它们进行解构赋值，都会报错。

以下三种解构赋值不得使用圆括号：（1）变量声明语句、（2）函数参数、（3）赋值语句的模式。可以使用圆括号的情况只有一种：赋值语句的非模式部分，可以使用圆括号。

用途：（1）交换变量的值、（2）从函数返回多个值、（3）函数参数的定义（4）提取 JSON 数据、（5）函数参数的默认值、（6）遍历 Map 结构、（7）输入模块的指定方法

字符串扩展

`include()`、`startsWith()`、`endsWith()`、`repeat()`

```
// 查找字符是否存在
let str = 'naosiuf'
str.include('f') // true
// 是否以此字符开头
str.startsWith('s') // false
// 是否以此字符结尾
str.endsWith('o') // false
str.repeat(3) // 拷贝3次

```

模板语法

``

Symbol

ES6 引入了一种新的原始数据类型`Symbol`，表示独一无二的值。`Symbol`函数前不能使用`new`命令。

`Symbol`函数可以接受一个字符串作为参数，表示对 Symbol 实例的描述，主要是为了在控制台显示，或者转为字符串时，比较容易区分。

```
let s1 = Symbol('foo');
let s2 = Symbol('bar');

s1 // Symbol(foo)
s2 // Symbol(bar)

s1.toString() // "Symbol(foo)"
s2.toString() // "Symbol(bar)"
```

如果 Symbol 的参数是一个对象，就会调用该对象的`toString`方法，将其转为字符串，然后才生成一个 Symbol 值。

```
const obj = {
    toString() {
        return 'abc';
    }
};
const sym = Symbol(obj);
sym // Symbol(abc)
```

由于每一个 Symbol 值都是不相等的，这意味着 Symbol 值可以作为标识符，用于对象的属性名，就能保证不会出现同名的属性。这对于一个对象由多个模块构成的情况非常有用，能防止某一个键被不小心改写或覆盖。

```

```

```
let mySymbol = Symbol();

// 第一种写法
let a = {};
a[mySymbol] = 'Hello!';

// 第二种写法
let a = {
    [mySymbol]: 'Hello!'
};

// 第三种写法
let a = {};
Object.defineProperty(a, mySymbol, { value: 'Hello!' });

// 以上写法都得到同样结果
a[mySymbol] // "Hello!"
```

并非不能重新赋值，只是每个值都唯一.....

```
let name = Symbol()
{
    var person = {}
    person[name] = 'file1'
    console.log(person[name]) // file1
}
{
    let name = Symbol()
    person[name] = 'file2'
    console.log(person[name]) // file2
}
console.log(person[name]) // file1
console.log(person) // Symbol(): "file1" Symbol(): "file2"
```

Vue

指令

v-html v-text

```
// 可以解析html语法
<p v-html="msg"></p>
// 不能及解析
<p v-text="msg"></p>
```

class

```
<span v-bind:class="{active: isActive]"></span> // 对象语法
<span v-bind:class="[active, errorClass]"></span> // 数组语法
<span v-bind:class="classObject"></span>
<script>
    data: {
        classObject: {
            active: true,
            'text-danger': false
        }
    }
</script>
```

style

```
<div v-bind:style="{ color: activeColor, fontSize: fontSize + 'px' }"></div>
<script>
    data: {
        activeColor: 'red',
        fontSize: 30
    }
</script>
// 多重值
// 只渲染最后一个浏览器支持的属性
<div :style="{ display: ['-webkit-box', '-ms-flexbox', 'flex'] }"></div>
```

v-if v-show

```
// v-if v-else-if v-else
// v-if="no"在页面上没有位置
<div v-if="type === 'A'">A</div>
<div v-else-if="type === 'B'">B</div>
<div v-else-if="type === 'C'">C</div>
<div v-else>Not A/B/C</div>
// v-show 始终有位置，仅切换display属性
// v-show不支持<template>模板
// 如果需要非常频繁地切换，则使用 v-show 较好；如果在运行时条件很少改变，则使用 v-if 较好。
```

v-for

当在组件中使用 `v-for` 时，`key` 现在是必须的

```
<ul id="example-2">
    <li v-for="(item, index) in items">
        {{ parentMessage }} - {{ index }} - {{ item.message }}
    </li>
</ul>
<script>
var example2 = new Vue({
    el: '#example-2',
    data: {
        parentMessage: 'Parent',
```

```
        items: [
            { message: 'Foo' },
            { message: 'Bar' }
        ]
    }
})
</script>
```

动态添加

```
<script>
var vm = new Vue({
    data: {
        a: 1
    }
})
// `vm.a` 现在是响应式的
vm.b = 2
// `vm.b` 不是响应式的
</script>
```

可以使用 `Vue.set(object, key, value)` 方法向嵌套对象添加响应式属性

```
<script>
var vm = new Vue({
    data: {
        userProfile: {
            name: 'Anika'
        }
    }
})
// 添加 age 属性到 userProfile 对象
Vue.set(vm.userProfile, 'age', 27)
// vm.$set是全局 Vue.set 的别名
vm.$set(vm.userProfile, 'age', 27)
</script>
```

CDN

```
// 不支持IE8
// 2.x
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
```

搭建脚手架

vue cli2

```
// 查看node版本
node -v
// 查看npm版本
npm -v
# vue cli2
// 安装全局的cli
npm install --global vue-cli
// 查看vue版本
vue --version
// 搭建基于webpack的项目
vue init webpack ProjectName
// 运行
cd ProjectName
npm install
npm run dev
```

vue cli3

```
# vue cli3
// 安装全局
npm install -g @vue/cli
// 查看版本
vue -V
// 查看命令
vue --help
// 创建项目
vue create ProjectName
// 可以默认和自选安装 Babel、Router、Vuex
cd ProjectName
npm run serve
```

```
// 安装路由
npm install vue-router --save-dev
npm install axios --save
```

--save

--save 和 --save-dev 区别：

表面上的区别是 `--save` 会把依赖包名称添加到 package.json 文件 dependencies 下，`--save-dev` 则添加到 package.json 文件 devDependencies 下。

不过这只是它们的表面区别。dependencies 是运行时依赖，devDependencies 是开发时的依赖。比如 我们安装 js 的压缩包 gulp-uglify 时，我们采用的是 “npm install –save-dev gulp-uglify” 命令安装，因为我们在发布后用不到它，而只是在我们开发才用到它。dependencies 下的模块，则是我们发布后还需要依赖的模块，譬如像jQuery 库或者 Angular 框架类似的，我们在开发完后后肯定还要依赖它们，否则就运行不了。

组件父传子

将数据放在父级 App.vue 的 data 里，在组件上绑定自定义属性传值，在子级用 props 接收传值

```
// props: ['users']
props: {
    users: {
        type: Array,
        required: true
    }
}
```

传值 传引用

如果给多个子级传引用，其中一个子级改变引用值，其余都改变，传值不会。

组件子传父

以事件为驱动

```
// 在子级组件Header.vue中声明一个传参方法
// 注册titleChange
// 命名 传的内容
methods: {
    change () {
        this.$emit('titleChange', '子向父传值')
    }
}

// 在父级组件App.vue中调用方法
// $event一定
<app-header v-on:titleChange="updateTitle($event)"></app-header>
// 在父级组件App.vue中实现updateTitle()方法
methods: {
    updateTitle (target) {
        this.title = target;
    }
}
```

路由过程

```
// 1.安装
npm install vue-router --save-dev
// 2.导入路由和组件
import VueRouter from 'vue-router'
import Header from './components/Header' // ...
// 3.使用路由
Vue.use(VueRouter)
```

```
// 4.配置路由
const routes = [
    {path: '/header', component: Header, name: '', children: '',redirect: ''},
    {path: '', component: '', name: '', children: '',redirect: ''}
    // ...
]
// 5.创建路由
const router = new VueRouter({
    routes,
    mode: 'history' // 去除#
})
// 6.将路由注入到vue的根实例中
new Vue({
    el: '#app',
    router,
    render: h => h(App)
})
```

局部注册组件

```
import Users from './Users.vue'
import Header from './Header.vue'
import Footer from './Footer.vue'
export default {
    name: 'home',
    data () {
        return {
        }
    },
    components: {
        'users': Users,
        'app-header': Header,
        'app-footer': Footer
    }
}
```

全局注册组件

```
// main.js文件内注册全局组件
import Users from './compoment/Users
Vue.component('users', Users)
```

prop、$ref区别 ☆

prop 着重于数据的传递，它并不能调用子组件里的属性和方法。像创建文章组件时，自定义标题和内容这样的使用场景，最适合使用prop。

$ref 着重于索引，主要用来调用子组件里的属性和方法，其实并不擅长数据传递。而且ref用在dom元素的时候，能使到选择器的作用，这个功能比作为索引更常有用到。

全局守卫

```
// 全局守卫
// to进入的路由、from从哪离开、next()决定是否展示进入的路由页面
router.beforeEach((to, from, next) => {
    console.log(to)
    console.log(from)
    if (to.path == '/login' || to.path == '/register') {
        next();
        //next(false)表示不跳转
    } else {
        alert('还没登录')
        next('/login')
    }
})
```

全局后置钩子

```
// 后置钩子
router.afterEach((to, from) => {
    alert('each')
})
```

路由独享守卫

```
//在某个路由的配置里面
{path: '/admin', name: 'adminLink', component: Admin, beforeEnter: (to, from, next)
=> { alert('快去登录') }},
```

组件内的守卫

```
//在Admin.vue组件内
<template>
    <div>
        <h1>Admin</h1>
    </div>
</template>

<script>
```

```
export default {
    name: '',
    data () {
        return {
            name: 'Henry'
        }
    },
    beforeRouteEnter: (to, from, next) => {
        console.log(this.name)// undefined
        // 定义回调
        // 异步的？
        next(vm => {
            console.log(vm.name)
        })
    },
    beforeRouteLeave: (to, from, next) => {
        if (confirm('Leave？？？')) {
            next()
        } else {
            next(false)
        }
    }
}
</script>
```

路由控制滚动

```
// 控制浏览器的展示位置，仅当前进后退才可触发
// 在main.js里
const router = new VueRouter({
    routes,
    mode: 'history',
    // 展示的位置
    scrollBehavior (to, from, savedPosition) {
        // return {x:0, y:0} 期望滚动到的位置
        // return {selector: '.btn'}
        // 保留上次一的位置
        if (savePosition) {
            return savePosition
        } else {
            return {x:0, y:0}
        }
    }
})
```

组件复用

```
// App.vue里复用组件
```

```
<template>
    <div id="app">
        <div class="container">
            <app-header></app-header>
        </div>
        <div class="container">
            <router-view></router-view>
        </div>
        <br>
        <div class="container">
            <div class="row">
                <div class="col-sm-12 col-md-4">
                    <router-view name="orderingGuide"></router-view>
                </div>
                <div class="col-sm-12 col-md-4">
                    <router-view name="delivery"></router-view>
                </div>
                <div class="col-sm-12 col-md-4">
                    <router-view name="history"></router-view>
                </div>
            </div>
        </div>
    </div>
</template>
// routes.js中注册组件
// 在Home.vue中显示
{path: '/', name: 'homeLink', components: {
    default: Home, // 默认显示
    orderingGuide: OrderingGuide,
    delivery: Delivery,
    history: History
}},
```

动态路由

`this.$route` 是当前路由，`this.$router` 是路由实例。还有 `$route.query` 、 `$route.hash` 等等。你可以查看 API 文档 的详细说明。

```
const router = new VueRouter({
  routes: [
    // 动态路径参数 以冒号开头
    { path: '/user/:id', component: User }
  ]
})
```

```
/user/:username/post/:post_id
/user/evan/post/123
{ username: 'evan', post_id: '123' }
```

复用组件

复用组件时，想对路由参数的变化作出响应的话，你可以简单地 watch (监测变化) `$route` 对象，或者使用 2.2 中引入的 `beforeRouteUpdate` 导航守卫：

```
const User = {
    template: '<div>User{{$route.params.id}}</div>',
    watch: {
        '$route' (to, from) {
        // 对路由变化作出响应...
        }
    },
    beforeRouteUpdate (to, from, next) {
        // react to route changes...
        // don't forget to call next()
    }
}
```

编程式导航

> `router.push(location, onComplete?, onAbort?)`

想要导航到不同的 URL，用 `router.push` 方法。这个方法会向 history 栈添加一个新的记录。当你点击 `<router-link>` 时，这个方法会在内部调用，所以说，点击 `<router-link :to="...">` 等同于调用 `router.push(...)`。前者是声明式，后者是编程式。

```
// 等同于router-link的路径写法
// 字符串
router.push('home')
// 对象
router.push({path: 'home'})
// 命名的路由
router.push({name: 'user', params: {userId: 123}})
// 带查询参数，相当于 /register?plan=private
router.push({path: 'register', query: {plan: 'private'}})
```

注意：如果提供了 path，params 会被忽略。

如果目的地和当前路由相同，只有参数发生了改变 ，需要使用 `beforeRouteUpdate` 来响应变化 。

```
const userId = 123
router.push({ name: 'user', params: { userId }}) // -> /user/123
router.push({ path: `/user/${userId}` }) // -> /user/123
// 这里的 params 不生效
router.push({ path: '/user', params: { userId }}) // -> /user
```

> `router.replace(location, onComplete?, onAbort?)`

跟 `router.push` 很像，唯一的不同就是，它不会向 history 添加新记录，而是替换掉当前的 history 记录。他的声明式 `<router-link :to="..." replace>`

> `router.go(n)`

类似 `window.history.go(n)`，是在 history 记录中向前或者后退多少步。

命名路由和视图

```
<router-view class="view one"></router-view>
<router-view class="view two" name="a"></router-view>
<router-view class="view three" name="b"></router-view>
<script>
const router = new VueRouter({
    routes: [
        {
            path: '/',
            components: {
                default: Foo,
                a: Bar,
                b: Baz
            }
        }
    ]
})
</script>
```

Vee-Validate

```
// 安装验证
npm install vee-validate --save
// __WEBPACK_IMPORTED_MODULE_2_vee_validate__.a.addLocale is not a function 报错
npm uninstall vee-validate
npm install vee-validate@2.0.0-rc.25
npm install vee-validate@next --save
```

新建 Validate.js 配置信息

```
import Vue from 'vue'
import VeeValidate, {Validator}  from 'vee-validate';
import zh from 'vee-validate/dist/locale/zh_CN';
Validator.addLocale(zh);
// 设置中文
const config = {
  locale: 'zh_CN'
};
Vue.use(VeeValidate,config);
// 自定义validate
```

```
const dictionary = {
    zh_CN: {
        // 提示语
        messages: {
            email: () => '请输入正确的邮箱格式',
            required: ( field )=> "请输入" + field
        },
        attributes:{
            email:'邮箱',
            password:'密码',
            name: '账号',
            phone: '手机'
        }
    }
};
Validator.updateDictionary(dictionary);
// 扩展
Validator.extend('phone', {
  messages: {
    zh_CN:field => field + '必须是11位手机号码',
  },
  validate: value => {
    return value.length == 11 && /^((13|14|15|17|18)[0-9]{1}\d{8})$/.test(value)
  }
});

// 提交验证
this.$validator.validateAll().then((result)=>{
    if(result){
        //...
    }
})
```

form.vue 组件中的代码：

- 首先在 input 中你得有 name 属性。
- v-validate 属性：管道形式进行过滤，验证条件。
- span 就是错误提示 。

```
<div class="layui-form-item">
    <label class="layui-form-label">账户</label>
    <div class="layui-block">
        <input v-model="name" v-validate="'required|min:3|alpha'" :class="{'input': true, 'is-danger': errors.has('name') }" type="text" name="name" class="layui-input" placeholder="账户">
        <span v-show="errors.has('name')" class="text-style" v-cloak> {{ errors.first('name') }} </span>
    </div>
</div>
```

```
errors.first('field') // 获取关于当前field的第一个错误信息
collect('field') // 获取关于当前field的所有错误信息(list)
has('field') // 当前filed是否有错误(true/false)
all() // 当前表单所有错误(list)
any() // 当前表单是否有任何错误(true/false)

```

参数语法：

```
// required 必填 验证规则是email
<input v-validate="'required|email'" type="email" name="email">
// alpha 只包含字母字符 长度6-16
<input v-validate="'alpha|min:6|max：16'" type="text" name="username">
```

Vuex

安装和使用

```
// npm
npm install vuex --save
// 使用
import Vuex from 'vuex'
Vue.use(Vuex)
```

创建一个 store

```
// npm
npm install vuex --save
// 使用
import Vuex from 'vuex'
Vue.use(Vuex)
```

GIT

创建和编辑文件

mkdir "name" 创建文件

vi(visual interface) "name" 编辑文件，也可创建

i(Insert mode) 切换到编辑模式

esc 退出编辑模式

:wq(write and quite) enter 保存并退出编辑

创建本地仓

git init 创建本地仓

mkdir learngit 创建文件夹

git add readme.md 创建文件

添加到暂存区

git add *** 添加指定文件到暂存区

git add -A 添加所有内容

git add . 添加新增和编辑的内容，不包括删除的文件

git add -u 添加编辑和删除的文件，不添加新增文件

关联远程

git remote add origin "address" 关联远程仓

git push -u origin master 初始推送至远程仓

git push origin master 推送新修改

git clone "address" 克隆

git remote -v 查看关联信息

git pull 推送失败，先抓取远程的新提交

版本回退

git reset --hard HEAD^ 回退到上一次commit的版本（注意本地文件可能会被删除）

git reset --hard "codeNum" 回退到指定commit版本

git log 查看commit的历史版本记录

git reflog 查看命令历史记录，包括回退命令

分支

git branch 查看分支

git branch "name" 创建分支

git checkout "name" 切换到指定分支

git checkout -b "name" 切换并创建分支

初始创建分支后，此时分支还在本地，推送到远程仓需要以下命令：

1. git add ...
2. git commit -m '...'
3. git push -u origin "name"，此时分支同步到远程
4. git push 此后可以只用push

合并分支

git merge "branch name" 合并指定分支到当前分支

git branch -d "branch name" 删除本地分支

git branch -D "name" 强行删除

git push origin -d "branch name" 删除远程分支

准备合并`dev`分支，请注意`--no-ff`参数，表示禁用`Fast forward`：

git merge --no-ff -m "merge with no-ff" dev

标签

git tag 查看所有标签

git tag “tagname” 用于新建一个标签，默认为 HEAD，也可以知道一个 commit.id

git tag -a “tagname” -m “aaaa” 可以指定标签信息