Fron-End Book

陈磊

2018 年 6 月 29 日

目录

第一章　FUNDAMENTAL

1.1　HTML

1.1.1　HTML5 新标签

标签	描述
<article>	定义文章。
<aside>	定义页面内容之外的内容。
<audio>	定义声音内容。
<bdi>	定义文本的文本方向，使其脱离其周围文本的方向设置。
<canvas>	定义图形。
<command>	定义命令按钮。
<datalist>	定义下拉列表。
<details>	定义元素的细节。
<dialog>	定义对话框或窗口。
<embed>	定义外部交互内容或插件。
<figcaption>	定义 figure 元素的标题。
<figure>	定义媒介内容的分组，以及它们的标题。
<footer>	定义 section 或 page 的页脚。
<header>	定义 section 或 page 的页眉。
<keygen>	定义生成密钥。
<mark>	定义有记号的文本。
<meter>	定义预定义范围内的度量。
<nav>	定义导航链接。
<output>	定义输出的一些类型。
<progress>	定义任何类型的任务的进度。

标签	描述
`<rp>`	定义若浏览器不支持 ruby 元素显示的内容。
`<rt>`	定义 ruby 注释的解释。
`<ruby>`	定义 ruby 注释。
`<section>`	定义 section。
`<source>`	定义媒介源。
`<summary>`	为 `<details>` 元素定义可见的标题。
`<time>`	定义日期/时间。
`<video>`	定义视频。
`<wbr>`	定义视频。

reference: w3school, HTML 参考手册

1.1.2 内联元素和块级元素

1. 块级元素

```
<address> <caption> <dd> <div> <dl> <dt> <fieldset>
<form> <h1> <h2> <h3> <h4> <h5> <h6> <hr> <legend>
<li> <noframes> <noscript> <ol> <ul> <p> <pre>
<table> <tbody> <td> <tfoot> <th> <thead> <tr>
```

2. 内联元素

```
<a> <abbr> <acronym> <b> <bdo> <big> <br> <cite> <code>
<dfn> <em> <i> <img> <input> <kbd> <label> <q> <samp> <select>
<small> <span> <strong> <sub> <sup> <textarea> <tt> <var>
```

3. 可变元素：根据上下文确定是块级元素还是内联元素

```
<applet> <button> <del> <iframe> <ins> <map> <object> <script>
```

reference: A1ter, 块级元素和行内元素的区别, segmentfault

1.1.3 defer 和 async

```
<script async src="script.js"></script>
<script defer src="script.js"></script>
```

1. defer 和 async 在网络读取（下载）这块儿是一样的，都是异步的（相较于 HTML 解析）

2. 它俩的差别在于脚本下载完之后何时执行，显然 defer 是最接近我们对于应用脚本加载和执行的要求的
3. 关于 defer，此图未尽之处在于它是按照加载顺序执行脚本的，这一点要善加利用
4. async 则是一个乱序执行的主，反正对它来说脚本的加载和执行是紧紧挨着的，所以不管你声明的顺序如何，只要它加载完了就会立刻执行
5. async 对于应用脚本的用处不大，因为它完全不考虑依赖（哪怕是最低级的顺序执行），不过它对于那些可以不依赖任何脚本或不被任何脚本依赖的脚本来说却是非常合适的，最典型的例子：Google Analytics

reference: nightire, defer 和 async 的区别, segmentfault

1.1.4 img 属性 alt 和 title

- alt：当图片无法显示时，将显示 alt 指定的文字。
- title：鼠标滑过时，会显示 title 指定文本。

1.1.5 Web 存储

- localStorage - 没有时间限制的数据存储
- sessionStorage - 针对一个 session 的数据存储

  sessionStorage, in practice, is best used for temporary data storage.

(Cookie 4kb, userData 64kb, Flash 100kb, SQLite, HTML5 localStorage 5MB)

reference: Web Storage 初探, Web 前端实现本地存储, Web Storage API

1.2 CSS

定位, 盒模型, 浮动, 媒体查询 (自适应布局).

1.2.1 定位

CSS 有三种基本的定位机制：普通流、浮动和绝对定位。任何元素都可以定位，不过绝对或固定元素会生成一个块级框，而不论该元素本身是什么类型。

说明

1. 默认值: static
2. 继承性: no
3. 版本: CSS2
4. JavaScript 语法: `object.style.position="absolute"`

值	描述
absolute	生成绝对定位的元素，相对于 static 定位以外的第一个父元素进行定位。元素的位置通过 “left”, “top”, “right” 以及 “bottom” 属性进行规定。
fixed	生成绝对定位的元素，相对于浏览器窗口进行定位。元素的位置通过 “left”, “top”, “right” 以及 “bottom” 属性进行规定。
relative	生成相对定位的元素，相对于其正常位置进行定位。因此，“left:20” 会向元素的 LEFT 位置添加 20 像素。
static	默认值。没有定位，元素出现在正常的流中（忽略 top, bottom, left, right 或者 z-index 声明）。
inherit	规定应该从父元素继承 position 属性的值。

- 如果一个标签的位置是绝对的，它又不在其他任何设定了 absolute、relative 或 fixed 定位的标签里面，那它就是相对于浏览器窗口进行定位。
- 如果一个标签处在另一个设定了 absolute、relative 或 fixed 定位的标

签里面，那它就是相对于另一个元素的边沿进行定位。

reference: `http://www.w3school.com.cn/cssref/pr_class_position.asp`，CSS 实战手册（第 2 版）

1.2.2 盒模型

IE 盒模型和标准存在区别。

1.2.3 优先级

优先级**从高到低**，分三个层次描述。

1. 作者/用户/浏览器样式。作者样式指网页本身的样式，或者开发者编写的样式；用户样式指浏览网页的用户自己添加的样式表 (通过浏览器设置)；浏览器样式指浏览器提供的默认样式。
    1. 标有 `!important` 的用户样式。
    2. 标有 `!important` 的作者样式。
    3. 作者样式。
    4. 用户样式。
    5. 浏览器样式。
2. 作者样式：内部样式（internal）、内联样式（inline）、外部样式（external）。内部样式 style 标签中声明的样式；内联样式指元素属性 style 中的样式；外部样式指通过 link 链接的外部文件中样式。
    1. 内联样式（行内样式）
    2. 内部样式
    3. 外部样式

外部样式和内部样式在优先级相同的情况下，后定义的会覆盖先定义的。

3. CSS 选择器
    1. ID 选择器：`#idname`
    2. 伪类：`:hover`
    3. 属性选择器：`input[type="text"]`
    4. 类选择器：`.classname`
    5. 元素 (类型) 选择器（包括伪元素）：`input`, `:after`
    6. 通用选择器：`*`

reference: Cameron Moll, 精通 CSS（第 2 版），优先级, mdn, INLINE VS INTERNAL VS EXTERNAL CSS

1.2.4 link 和 @import

- `@import` 和 link 混用时，可能会出现不同时下载的情况。
- 考虑两者混合使用的浏览器实现的不一样（下载次序）。

reference: [Steve Souders, don't use @import](http://www.stevesouders.com/blog/2009/04/09/use-import/), 外部引用 CSS 中 link 与 @import 的区别

1.2.5 浮动和文档流

1.2.6 两栏等高布局

父元素设置 `overflow:hidden;`，子元素设置 `padding-bottom:10000px; margin-bottom:-10000px;`。`overflow:hidden;` 截除超出的高度，`margin-bottom:-10000px;` 抵消较高内容超出的 `padding`。

```
<div id="fa">
  <div class="col">
    <p>1231321321</p>
    <p>1313213</p>
    <p>1313213</p>
    <p>1313213</p>
    <p>1313213</p>
    <p>1313213</p>
  </div>
  <div class="col">45645456456</div>
</div>
```

```
#fa {
  width: 800px;
  margin: 0 auto;
  background-color: #1524e5;
  overflow: hidden;
}

.col {
  float: left;
  width: 50%;
  padding-bottom: 10000px;
  margin-bottom: -10000px;
}

```

```
.col:first-child {
  background-color: #34ef34;
}
.col:last-child {
  background-color: #ef34ef;
}
```

refernce: CSS/两栏并列等高布局

1.2.7 动画

动画的实现方式可以分为几种：CSS3 原生实现；JavaScript 实现 (操作 CSS、canavs、svg)；动画文件 (gif, flash)。

CSS3 的实现可以通过 @keyframes 和 animation 完成，@keyframes 定义动画，animation 调用动画。

```
/* 定义 */
@keyframes changebackcolor{
  from {background: red;}
  to {background: yellow;}
}
/* 调用 */
div{
  animation: changebackcolor 5s;
}
```

reference: `http://www.w3school.com.cn/css3/css3_animation.asp`

1.2.8 垂直居中和水平居中

1.2.8.1 块元素垂直居中 (transform)

此方案存在兼容性问题 (Firefox 43 不支持, Chrome 46 支持)

```
{
  width: 250px;
  height: 250px;
  position: relative;
  top: 50%;
  transform: translateY(-50%);
}
```

1.2.8.2 块元素垂直居中 (绝对定位)

```
{
  width: 250px;
  height: 250px;
  position: absolute;
  top: 50%;
  margin-top: -125px;
}
```

1.2.8.3 内联元素垂直居中

```
{
  /* 方案一: 块元素内容会居中, 需要设置高度 */
  display: table-cell;
  vertical-align: middle;

  /* 方案二: 一般单行文本的上下居中 */
  line-height: 50px;
}
```

1.2.8.4 块元素的水平居中

```
{
  width: 100px;
  margin-left: auto;
  margin-right: auto;
}
```

1.2.9 扩展

1. atomiks/30-seconds-of-css: 一些实用的代码片段收集.

1.3 JavaScript

1.3.1 JavaScript 组成

一个完整的 JavaScript 实现是由以下 3 个不同部分组成的：

- 核心（ECMAScipt）
- 文档对象模型（DOM）
- 浏览器对象模型（BOM）
- JavaScript 实现
- JavaScript 学习总结（三）BOM 和 DOM 详解

1.3.2 类型、值和变量

1.3.2.1 数据类型

- 基本数据类型：String 字符串；Number 数字；Boolean 布尔。
- 复合数据类型：Object 对象；Array 数组。
- 特殊数据类型：Null 空对象；Undefined 未定义。
- 数据类型 (JavaScript), msdn

1.3.2.2 null, NaN, undefined

JavaScript 中有 6 个值为 “假”: `false`, `null`, `undefined`, `0`, `''`(空字符串), `NaN`. 其中 `NaN` 是 JavaScript 中唯一不等于自身的值, 即 `NaN == NaN` 为 `false`.

```
console.log( false == null ) // false
console.log( false == undefined ) // false
console.log( false == 0 )    // true
console.log( false == '' )   // true
console.log( false == NaN )  // false

console.log( null == undefined ) // true
console.log( null == 0 )    // false
console.log( null == '' )   // false
console.log( null == NaN )  // false

console.log( undefined == 0 ) // false
console.log( undefined == '' ) // false
console.log( undefined == NaN ) // false

console.log( 0 == '' )      // true
console.log( 0 == NaN )     // false
```

对于 `===` 以上全为 `false`。对于 `==`，以下几组为 `true`: `null` 和 `undefined`; `false`、`0`、`''`。

- JavaScript 中奇葩的假值
- 阮一峰, undefined 与 null 的区别, 2014

1.3.3 函数

1.3.3.1 常用函数

- `string.slice(start,end)` 复制 string 中的一部分。
- `string.indexOf(searchString, position)` 在 string 中查找 searchString。如果被找到，返回第一个匹配字符的位置，否则返回 -1。可选参数 position 可设置从 string 的某个指定的位置开始查找。
- `object.hasOwnProperty(name)`

1.3.3.2 Array

改变原数组

- `array.pop()` 移除最后一个元素。
- `array.push(item...)` 将一个或多个参数附加到数组的尾部。
- `array.reverse()` 反转 array 元素的顺序。
- `array.sort(camprefn)` 数组排序。
- `array.shift()` 移除数组中的第一个元素并返回该元素。
- `array.splice(start, deleteCount, item)` 从 array 中移除一个或多个元素并用新的 item 替换他们。

不改变原数组

- `array.concat(item...)` 产生一个新数组，它包含一份 array 的前复制，并把一个或多个参数 item 附加在其后面。
- `array.join(separator)` join 方法把一个 array 构成一个字符串。它先把 array 中的每个元素构造成一个字符串，接着用一个 separator 分隔符把它们连接在一起。
- `array.map(item => )` 数组映射, 依次处理数组中的每一项.
- `array.filter(item => )` 数组过滤, 保存返回为真的项.
- forEach(callback(currentValue[, index[, array]]){}), 循环.
- some, 是否有某个元素满足条件 (回掉返回 true).
- every, 是否全部元素满足条件.

1.3.3.3 Date

```
var t = new Date();
var tt = [
    t.getFullYear(), '年', // 不是 getYear()
    t.getMonth() + 1, '月',
    t.getDate(), '日', ' ',
    t.getHours(), '时',
    t.getMinutes(), '分',
    t.getSeconds(), '秒'
].join('');
console.log(tt); // 2015年10月30日 22时6分21秒
```

注意 `getMonth()` 和 `getDay()` 都是从 0 开始的，需要加 1。

- Date, mdn

1.3.3.4 typeof 和 instanceof

1. typeof: `typeof operand`
   typeof 操作符返回一个字符串, 表示未经求值的操作数 (unevaluated operand) 的类型。typeof 只有一个实际应用场景，就是用来检测一个对象是否已经定义或者是否已经赋值。而这个应用却不是来检查对象的类型。除非为了检测一个变量是否已经定义，我们应尽量避免使用 typeof 操作符。

   ```
   typeof foo == 'undefined' // 若 foo 未定义, 返回 true
   ```

类型	结构
Undefined	“undefined”
Null	“object”
布尔值	“boolean”
数值	“number”
字符串	“string”
Symbol (ECMAScript 6 新增)	“symbol”
宿主对象 (JS 环境提供的，比如浏览器)	Implementation-dependent
函数对象 (implements [[Call]] in ECMA-262 terms)	“function”
任何其他对象	“object”

2. instanceof: `object instanceof constructor`
   instanceof 运算符可以用来判断某个构造函数的 prototype 属性是否存在另外一个要检测对象的原型链上。(instanceof 运算符用来检测 constructor.prototype 是否存在于参数 object 的原型链上。)

```
function C(){} // 定义一个构造函数
function D(){} // 定义另一个构造函数

var o = new C();
o instanceof C; // true,因为:Object.getPrototypeOf(o) === C.
    ↪ prototype
o instanceof D; // false,因为D.prototype不在o的原型链上
o instanceof Object; // true,因为Object.prototype.isPrototypeOf(o)返
    ↪ 回true
C.prototype instanceof Object // true,同上

C.prototype = {};
var o2 = new C();
o2 instanceof C; // true
o instanceof C; // false,C.prototype指向了一个空对象,这个空对象不在o
    ↪ 的原型链上.

D.prototype = new C();
var o3 = new D();
o3 instanceof D; // true
o3 instanceof C; // true
```

- typeof 和 instanceOf 的区别
- instanceof
- typeof

1.3.3.5 setInterval 和 setTimeout

- `setTimeout` 只执行一次
- `setInterval` 连续执行多次

1.3.3.6 call 和 apply

通过 call 和 apply, 可以实现自定义函数调用的上下文.

1. call: `fun.call(thisArg, arg1, arg2, ...)`

call() 方法在使用一个指定的 this 值和若干个指定的参数值的前提下调用某个函数或方法.

thisArg 在 fun 函数运行时指定的 this 值。需要注意的是，指定的 this 值并不一定是该函数执行时真正的 this 值，如果这个函数处于非严格模式下，则指定为 null 和 undefined 的 this 值会自动指向全局对象 (浏览器中就是 window 对象)，同时值为原始值 (数字，字符串，布尔值) 的 this 会指向该原始值的自动包装对象。

arg1, arg2, ... 指定的参数列表。

```
/* 将函数的参数 arguments 转换为数组 */
function listFirst(){
    // this 指向 arguments
    // 下面的语句相当于 arguments.slice(0)，但由于 arguments 不是数
        ↪ 组，不能直接调用 slice 方法
    var arr = Array.prototype.slice.call(arguments, 0);
    for (var i=0; i < arr.length; i++){
        console.log(arr[i]);
    }
}

listFirst(1,2,3); // 调用，输出 1,2,3
```

2. apply: `fun.apply(thisArg, [argsArray])`
   apply() 方法在指定 this 值和参数（参数以数组或类数组对象的形式存在）的情况下调用某个函数。

   thisArg 在 fun 函数运行时指定的 this 值。需要注意的是，指定的 this 值并不一定是该函数执行时真正的 this 值，如果这个函数处于非严格模式下，则指定为 null 或 undefined 时会自动指向全局对象（浏览器中就是 window 对象），同时值为原始值（数字，字符串，布尔值）的 this 会指向该原始值的自动包装对象。

   argsArray 一个数组或者类数组对象，其中的数组元素将作为单独的参数传给 fun 函数。如果该参数的值为 null 或 undefined，则表示不需要传入任何参数。从 ECMAScript 5 开始可以使用类数组对象。浏览器兼容性请参阅本文底部内容。

   在调用一个存在的函数时，你可以为其指定一个 this 对象，无需此参数时第一个参数可用 null（比如对于 add）。this 指当前对象，也就是正在调用这个函数的对象。使用 apply，你可以只写一次这个方法然后

在另一个对象中继承它，而不用在新对象中重复写该方法。

```
/* 例一: 将函数的参数 arguments 转换为数组 */
function listFirst(){
    // this 指向 arguments
    var arr = Array.prototype.slice.apply(arguments, [0]);
    for (var i=0; i < arr.length; i++){
        console.log(arr[i]);
    }
}

listFirst(1,2,3); // 调用, 输出 1,2,3

/* 例二: push 一个数组 */
var arr1=new Array("1","2","3");
var arr2=new Array("4","5","6");
Array.prototype.push.apply(arr1,arr2);
```

- Function.prototype.apply()
- apply 和 call 的用法

1.3.3.7 new/构造函数

构造函数只是一些使用 new 操作符时被调用的普通函数。使用 new 来调用函数，或者说发生构造函数调用时，会自动执行下面的操作。

1. 创建（或者说构造）一个全新的对象。
2. 这个新对象会被执行 [[prototype]] 链接。
3. 这个对象会绑定到函数调用的 this。
4. 如果函数没有返回其他对象，那么 new 表达式中的函数调用会自动返回这个新对象。

1.3.3.8 函数调用模式

除了声明时定义的形式参数, 每个函数还接收两个附加的参数: this 和 arguments. 在 JavaScript 中一共有 4 种调用模式: 方法调用模式、函数调用模式、构造器调用模式和 apply 调用模式. 函数调用的模式不同, 对应的 this 值也会不同。

1. 方法调用模式
   当一个函数被保存为对象的一个属性时，我们称它为一个方法。当一个方法被调用时，this 被绑定到该对象。

```
// print 作为 obj 属性被保存，当 print 被调用时，this 指向 obj
var obj = {
    value: 'I am a string.',
    print: function (){
        console.log(this.value);
    }
}

obj.print(); // 调用
```

2. 函数调用模式
   当一个函数并非一个对象的属性时, 那么它就是被当做一个函数来调用。以此模式调用函数时，this 被绑定到全局对象。

```
var func = function(){
    console.log('Hello here.');
}

func(); // 调用
```

3. 构造器调用模式
   如果在一个函数前面带上 new 来调用，那么背地里将会创建一个连接到该函数的 prototype 成员的新对象，同时 this 会绑定到那个新对象上。

```
var Quo = function(string){
    this.status = string;
}
Quo.prototype.get_status = function(){
    return this.status;
}

var myQuo = new Quo('new Quo.') // 调用
myQuo.get_status(); // "new Quo."
```

4. apply 调用模式
   apply 方法让我们构建一个参数数组传递给调用函数。它允许我们选择 this 的值。apply 方法接收两个参数，第 1 个是要绑定给 this 的值，第 2 个就是一个参数数组.

- 《JavaScript 语言精粹 (修订版)》，第 4 章函数

1.3.3.9 this

this 在运行时绑定，它的上下文取决于函数调用时的各种条件。this 的绑定和函数声明的位置没有任何关系，只取决于函数的调用方式。

判断 this 的优先级，可以按照下面的顺序进行判断：

1. 函数是否在 new 中调用 (new 绑定)？如果是的话 this 绑定的是新创建的对象。

```
var bar = new foo(); // 绑定 bar
```

2. 函数是否通过 call、apply (显示绑定) 或者硬绑定调用？如果是的话，this 绑定的是指定的对象。

```
var bar = foo.call(obj2); // 绑定 obj2
```

3. 函数是否在某个上下文对象中调用 (隐式调用)？如果是的话，this 绑定的是那个上下文对象。(调用时是否被某个对象拥有或包含，对象属性引用链中只有最顶层或者最后一层会影响调用位置 `obj1.obj2.foo()` ↪ `; // 绑定 obj2`)

```
var bar = obj1.foo(); // 绑定 obj1
```

4. 如果都不是的话，使用默认绑定。如果在严格模式下，就绑定到 undefined，否则绑定到全局对象 (window)。

```
var bar = foo(); // 绑定 undefined 或 window
```

可总结为:

1. 作为方法调用, 上下文为方法的拥有者.
2. 作为全局函数调用, 上下文为 window.
3. 作为构造函数调用, 上下文为新创建的实例对象.
4. call 和 apply 可以自定义上下文.
5. 箭头函数的上下文为其所在作用域的上下文.

- 你不知道的 JavaScript（上卷）
- JavaScript 忍者秘籍, p52

1.3.3.10 闭包

当函数可以记住并访问所在的词法作用域时，就产生了闭包，即使函数是在当前词法作用域之外执行[1]。(闭包是发生在定义时的。)

[1] 你不知道的 JavaScript（上卷）

```
// foo() 定义的中括号内就是 bar 的词法作用域
function foo() {
  var a = 2;
  function bar() {
    console.log( a );
  }
  return bar;
}
var baz = foo();
baz(); // 2, 这就是闭包, 用到变量 a
```

```
var fn;
function foo() {
  var a = 2;
  function baz() {
    console.log( a ); // 2
  }
  fn = baz; // 将 baz 赋值给全局变量
  // 调用 fn 相当于执行的 baz, 而其词法作用域在 foo() 定义函数内.
}

function bar(fn) {
  fn(); // 这就是闭包, 用到变量 a
}

foo();
bar(); // 2
```

1.3.4 继承方法

- 原型链继承
- 构造继承
- 实例继承
- 拷贝继承
- 继承与原型链, mdn, RayChase, JavaScript 实现继承的几种方式

1.3.5 正则表达式

参数

- `i`, 忽略大小写.
- `g`, 全局匹配, 找到所有匹配，而不是在第一个匹配后停止.

```
// 替换所有 a 字符，不区分大小写
"aAopa".replace(/a/gi, '')
```

特殊字符

- `\d` 任意一个数字，等价于 `[0-9]`
- `\D` 任意一个非数字，等价于 `[^0-9]`
- `\w` 任意一个字母、数字或下划线字符，等价于 `[a-zA-Z_]`
- `\W` 任意一个非字母、数字和下划线字符，等价于 `[^a-zA-Z_]`
- `\s` 任意一个空白字符，包括换页符、换行符、回车符、制表符和垂直制表符，等价于 `[\f\n\r\t\v]`
- `\S` 任意一个非空白符，等价于 `[^\f\n\r\t\v]`
- `.` 换行和回车以外的任意一个字符，等价于 `[^\n\r]`

次数匹配

- `?` 最多一次 (零次或一次)
- `+` 至少一次
- `*` 任意次
- `{n}` 只能出现 n 次
- `{n,m}` 至少 n 次，最多 m 次

reference

- mozilla, RegExp
- 正则总结：JavaScript 中的正则表达式

1.3.6 ES5

新功能包括：原生 JSON 对象、继承的方法、高级属性的定义以及引入严格模式。

- 梦禅, ECMAScript 各版本简介及特性, segmentfault

1.3.7 ES6

模块，类，块级作用域，Promise，生成器...

1.3.7.1 箭头函数

1. 优点: 箭头函数可省略 `function`, 单行的情况下可省略 `return`, 单个参数的情况括号也可省略.

```
function func1(a) {
  return a + 1;
}

var func1 = a => a + 1;
```

2. 优点: 箭头函数不会创建自己的上下文 (context), 意味着函数内 `this` 和箭头函数所定义的上下文一样.

```
var _this = this;
axios.get(api).then(function(res){
  _this.model = res.data;
});

// 无需 _this 或者 bind 传递 this
axios.get(api).then(res => {
  this.model = res.data;
})
```

3. 技巧: 返回对象, 需要在对象外层加括号 (对象大括号会被解析为函数体).

```
// 错误
var func1 = a => {a: a};
console.log(func1(a));

// 正确
var func1 = a => ({a: a});
console.log(func1(a));
```

4. 技巧: 通过 `||` 添加日志, 而不用单独一行输出日志.

```
a => {
  console.log(a);
  return {
    a: a
  }
}

```

```
a => console.log(a) || ({
  a: a
});
```

- Arrow Functions in JavaScript

1.3.8 异步请求

1.3.8.1 Ajax (Asynchronous Javascript And XML)

Ajax 实现

1. 原生 Ajax 实现: GET

```
var xmlhttp = new XMLHttpRequest();
xmlhttp.open("GET","test1.txt",true);
xmlhttp.send();
```

2. 原生 Ajax 实现: POST

```
var xmlhttp = new XMLHttpRequest();
xmlhttp.onreadystatechange = function(){
  if (xmlhttp.readyState==4 && xmlhttp.status==200){
    document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
  }
}
xmlhttp.open("POST","ajax_test.asp",true);
xmlhttp.setRequestHeader("Content-type","application/x-www-form-
    ↪ urlencoded");
xmlhttp.send("fname=Bill&lname=Gates");
```

- 原生 JS 与 jQuery 对 AJAX 的实现
- AJAX - 向服务器发送请求

Ajax 的 5 个状态

新建对象 -> 建立连接 -> 接收响应原始数据 -> 解析原始数据 -> 响应就绪 (待后续处理)

1. 0: 请求未初始化
   此阶段确认 XMLHttpRequest 对象是否创建，并为调用 open() 方法进行未初始化作好准备。值为 0 表示对象已经存在，否则浏览器会报错——对象不存在。
2. 1: 服务器连接已建立

此阶段对 XMLHttpRequest 对象进行初始化，即调用 open() 方法，根据参数 (method,url,true) 完成对象状态的设置。并调用 send() 方法开始向服务端发送请求。值为 1 表示正在向服务端发送请求。

3. 2: 请求已接收
   此阶段接收服务器端的响应数据。但获得的还只是服务端响应的原始数据，并不能直接在客户端使用。值为 2 表示已经接收完全部响应数据。并为下一阶段对数据解析作好准备。
4. 3: 请求处理中
   此阶段解析接收到的服务器端响应数据。即根据服务器端响应头部返回的 MIME 类型把数据转换成能通过 responseBody、responseText 或 responseXML 属性存取的格式，为在客户端调用作好准备。状态 3 表示正在解析数据。
5. 4: 请求已完成，且响应已就绪
   此阶段确认全部数据都已经解析为客户端可用的格式，解析已经完成。值为 4 表示数据解析完毕，可以通过 XMLHttpRequest 对象的相应属性取得数据。

- Panda, Ajax readyState 的五种状态, LOFTER

1.3.8.2 Fetch

fetch 和 Ajax (`jQuery.ajax()`) 的区别

- The Promise returned from `fetch()` **won't reject on HTTP error status** even if the response is an HTTP 404 or 500. Instead, it will resolve normally (with ok status set to false), and it will only reject on network failure or if anything prevented the request from completing.
- By default, `fetch` **won't send or receive any cookies** from the server, resulting in unauthenticated requests if the site relies on maintaining a user session (to send cookies, the credentials init option must be set).

```
fetch(api, config)
.then(response => response.json())
.then(data => {
  // handle
})
```

```
6 .catch(err => console.error(err));
```

- Using Fetch, MDN web docs

1.3.9 异步

1. 回调, 适应于异步处理较少的情况

```
1 get(url, callback);
```

2. Promise, 链式方式写异步处理

```
1 promise(somthing).then(res => handle)
```

3. async 和 await, 同步的方式写异步, 可适用于异步处理之间存在依赖的情况.

```
1 const promiseFunc = async () => {
2   const result1 = await getSomething();
3   const result2 = await dependOnResult1(result1);
4   return result2;
5 };
6 promiseFunc().then(result => {
7   console.log(result);
8 })
```

1.3.9.1 async 和 await

async, await

扩展

1. 理解 JavaScript 的 async/await, 边城, 2016/11/19

1.3.10 文档对象 DOM

1.3.10.1 浏览器事件冒泡和捕获

事件分为三个阶段:

- 捕获阶段
- 目标阶段
- 冒泡阶段

TODO:IE 和 w3c 标准的区别;阻止事件传播 (捕获,冒泡) `e.stopPropagation();` 阻止事件默认行为 `e.preventDefault()`。

```
function registerEventHandler(node, event, handler) {
  if (typeof node.addEventListener == "function")
    node.addEventListener(event, handler, false);
  else
    node.attachEvent("on" + event, handler);
}

registerEventHandler(button, "click", function(){print("Click (2)");});
```

- 本期节目, 浏览器事件模型中捕获阶段、目标阶段、冒泡阶段实例详解, segmentfault, 2015.8

1.3.11 浏览器对象 BOM

1.3.11.1 弹框

- `prompt(text,defaultText)` 提示用户输入的对话框。`text` 对话框中显示的纯文本，`defaultText` 默认的输入文本。返回值为输入文本。
- `alert(message)` 警告框。`message` 对话框中要实现的纯文本。
- `confirm(message)` 显示一个带有指定消息和 OK 及取消按钮的对话框。`message` 对话框中显示的纯文本。点击确认返回 true，点击取消返回 false。
- `http://www.w3school.com.cn/jsref/met_win_prompt.asp`

1.3.11.2 localStorage

> 存储在浏览器中的数据,如 localStorage 和 IndexedDB,以源进行分割。每个源都拥有自己单独的存储空间,一个源中的 Javascript 脚本不能对属于其它源的数据进行读写操作。

localStorage 在当前源下设置的值，只能在当前源下查看。

```
localStorage.setItem('key',value) // 设置值, 或 localStorage.key = value
localStorage.getItem('key')    // 获取值, 或 localStorage.key
localStorage.removeItem('key') // 删除值
localStorage.clear()           // 清空 localStorage
```

- JavaScript 的同源策略, MDN
- localStorage, MDN
- 杜若, localStorage 介绍

1.3.12 jQuery

1.3.12.1 jQuery 知识结构

- jQuery 基础: 选择器
- jQuery 效果: hide, show, toggle, fadeIn, fadeOut, animate
- jQuery 操作 HTML: text, html, val, attr, append, after, prepend, before, remove, empty
- jQuery 遍历 Dom: parent, parents, children, find

1.3.12.2 jQuery 和 Dom 对象相互转换

1. jQuery 对象转换成 Dom 对象: 下标和 get 方法

```
/* 方法一: 下标 */
var $v = $("#v") ; //jQuery对象
var v = $v[0];  //DOM对象

/* 方法二: get */
var $v = $("#v"); //jQuery对象
var v = $v.get(0); //DOM对象
```

2. Dom 对象转 jQuery 对象: 通过 `$()` 把 Dom 对象包装起来实现

```
var v = document.getElementById("v"); //DOM对象
var $v = $(v); //jQuery对象
```

- jQuery 对象与 dom 对象的区别与相互转换

1.3.12.3 jQuery 命名冲突解决

1. 使用 `jQuery.noConflict()`

```
jQuery.noConflict();
// 之后使用 jQuery 调用, jQuery("#id").methodname()
```

2. 自定义别名

```
var $j = jQuery.noConflict();
// $j("#id").methodname();
```

3. 传入参数，继续使用 $

```
jQuery.noConflict();
(function($){
    $("#id").methodname();
})(jQuery)
```

- 邦彦, 谈谈 jQuery 中的防冲突（noConflict）机制, TaoBaoUED
- RascallySnake, Jquery 的 $ 命名冲突, cnblogs
- TerryChen, Jquery 命名冲突解决的五种方案, cnblogs

1.3.13 其他

1. switch 以 === 匹配。
2. 函数会首先被提升，然后才是变量。

1.3.14 进阶

1.3.14.1 深拷贝和浅拷贝

浅拷贝只拷贝一层对象的属性，深拷贝则递归拷贝了所有层级。而 JavaScript 存储对象都是存地址的，这就导致如果拷贝的是一个对象，而不是值，原来的对象和拷贝到的目标变量会指向同一个地址，更改其中任一一个，另一个也会更改。或者可以说，浅拷贝是拷贝存值的地址，深拷贝是拷贝值。一下是浅拷贝的示例。

```
// 例1: 数值数组
var a = [1, 2, 3];
var b = a;
b[1] = 12344;
console.log(a);
// [1, 12344, 3]

// 例2: 对象
var obj = {a: 1, b: {p: 2}};
var newObj = {};
for (var i in obj) {
  newObj[i] = obj[i];
}
newObj.b.p = 3;
console.log(obj.b.p);
```

```
16 // 3
```

对象数组，比如 `[{a: 1}, {b: 2}]`，由于数组内部的值为引用对象，用 `slice` 拷贝仍然是浅拷贝。

实现深拷贝的方法，对于只有数值的数组可以直接用 `slice` 和 `concat` 方法实现，通用的方法是循环加递归。外层为循环，内层判断属性对应值是否为对象（`typeof obj === 'object'`），如果是则递归，如果不是则直接赋值。

```
1 var obj = {a: 1, b: {p: 2}};
2 function deepClone(obj) {
3   var result = obj.constructor === Array ? [] : {};
4   for (var key in obj) {
5     result[key] = typeof obj[key] !== 'object' ? obj[key] :
        ↪ deepClone(obj[key]);
6   }
7   return result;
8 }
```

总结：

1. 数组浅拷贝：`=`
2. 对象浅拷贝：`=` 或 `Object.assign({}, obj)`
3. 含值数组深拷贝：`[1,2,3].slice()` 或 `[].concat([1,2,3])`
4. 对象或数组深拷贝：`JSON.parse(JSON.stringify(obj))` 或者循环加递归。

备注：

`JSON.parse(JSON.stringify(obj))` 对于嵌套的对象或数组只包含 Primitive（只含值），但对于属性值有 function 或者 Date 等不应采用此法。

```
1 var a = {a: 1, b: () => {}, c: new Date()}
2 // {a: 1, b: f, c: Fri Jun 15 2018 11:10:49 GMT+0800 (China
    ↪ Standard Time)}
3 var o = JSON.parse(JSON.stringify(a))
4 // {a: 1, c: "2018-06-15T03:10:49.965Z"}
```

深拷贝辅助工具

```
1 import * as cloneDeep from 'lodash/cloneDeep';
2 var obj = {a: 1, b: 2};
3 var clone = cloneDeep(obj);
```

- javascript 中的深拷贝和浅拷贝？
- Deepcopy of JavaScript Objects and Arrays using lodash's cloneDeep method

1.4 浏览器

1.4.1 浏览器兼容性

1.4.1.1 IE6/IE7/IE8 支持 html5 新标签

创建标签

```
document.createElement('section'); // 其他标签一样处理
```

使用 JS 方案

1. html5shiv:https://github.com/aFarkas/html5shiv/ (用的也是 createElement())
2. modernizr: https://github.com/modernizr/modernizr/ (这个功能要多一点，还兼顾 CSS3，更多)

1.4.1.2 IE 兼容性测试

Modern.IE 提供的虚拟机测试。下载 virtualbox (免费) 或者 VMware (收费)，然后导入对应的下载包 (不同系统 IE 版本不同)。如果是 Windows 10, 虚拟机可启用 Hyper-V.

1.4.1.3 提示不支持 JavaScript

noscript 元素用来定义在脚本未被执行时的替代内容（文本）。此标签可被用于可识别 `<script>` 标签但无法支持其中的脚本的浏览器。

```
<noscript>Your browser does not support JavaScript!</noscript>
```

reference: HTML noscript 标签, w3school

1.4.1.4 浏览器 hack

以 IE 为例，展示几个 CSS hack 方法，更多的见参考链接。

IE6

```
.selector { _property: value; }
.selector { -property: value; }
```

IE <= 7 `! $ & * ( )= % + @ , . / ` [ ] # ~ ? : < > |`

```
.selector { !property: value; }
.selector { $property: value; }
.selector { &property: value; }
.selector { *property: value; }
.selector { )property: value; }
.selector { =property: value; }
.selector { %property: value; }
.selector { +property: value; }
.selector { @property: value; }
.selector { ,property: value; }
.selector { .property: value; }
.selector { /property: value; }
.selector { `property: value; }
.selector { ]property: value; }
.selector { #property: value; }
.selector { ~property: value; }
.selector { ?property: value; }
.selector { :property: value; }
.selector { |property: value; }
```

IE 6-8

```
.selector { property: value\9; }
.selector { property/*\**/: value\9; }
```

reference: `http://browserhacks.com/`

1.4.1.5 CSS3 前缀

虽然目前 CSS3 得到广泛支持，但各个浏览器厂商对标准实现并不完全一样，可以对 CSS3 使用前缀。建议把特殊的 CSS 语句放在前面，一般的语句放置在后面。可以使用 Autoprefixer 自动添加前缀。

- `-moz-` Firefox,
- `-webkit-` Safari, Chrome
- `-o-` Opera (不过, Opera 和 Chrome 现在都采用 Blink 内核)
- `-ms-` Internet Explorer

1.4.1.6 浏览器/设备检测

通过 js 检测浏览器方法包括：条件语句检测, 依靠浏览器各自的特性检测；通过 `userAgent` 检测。

IE 条件语句 可以添加条件语句检测浏览器，针对性的编写样式。

```
<!--[if IE 8]>
<body class="ie8">
<![endif]-->
<!--[if !IE]>
<body class="notie">
<![endif]-->
```

reference: About conditional comments, msdn

userAgent

userAgent 示例

```
Chrome 60
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML
    ↪ , like Gecko) Chrome/60.0.3112.113 Safari/537.36

360 安全浏览器/极速模式
Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML,
    ↪ like Gecko) Chrome/55.0.2883.87 Safari/537.36 QIHU 360SE

360 安全浏览器/IE 兼容模式
Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; .NET4.0C; .NET4.0
    ↪ E; .NET CLR 2.0.50727; .NET CLR 3.0.30729; .NET CLR
    ↪ 3.5.30729; rv:11.0) like Gecko

IE 11
Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; .NET4.0C; .NET4.0
    ↪ E; .NET CLR 2.0.50727; .NET CLR 3.0.30729; .NET CLR
    ↪ 3.5.30729; rv:11.0) like Gecko

Edge 40
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML
    ↪ , like Gecko) Chrome/52.0.2743.116 Safari/537.36 Edge
    ↪ /15.15063
```

userAgent 判别浏览器 可直接查看此插件的 js 代码:jquery-browser-plugin。实测了 IE、Firefox、Opera、Chrome，输出结果都是对的。

```
<html>
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script src="jquery.browser.js"></script>
</head>
<body>
<script>
    console.log("jQBrowser.webkit: " + jQBrowser.webkit);
    console.log("jQBrowser.mozilla: " + jQBrowser.mozilla);
    console.log("jQBrowser.msie: " + jQBrowser.msie);
    console.log("jQBrowser.version: " + jQBrowser.version);
</script>
</body>
</html>
```

userAgent 判别设备 通过检测设备, 可以针对不同设备提供不同功能, 并处理不同设备的兼容需求.

```
// 检测 IOS
ios = /iphone|ipod|ipad/i.test(window.navigator.userAgent);
// 检测 Android
android = /android/i.test(window.navigator.userAgent);
```

通过特性检测判断 reference: 露兜博客, 检测访客浏览器的 2 种方法, 2009

1.4.1.7 IE 图标字体未加载降级处理 (IE icon)

IE 下图标字体未加载, 导致图标无法显示, `element-ui` 就存在这个情况. 可以判断浏览器是否是 IE, 若果是则加载覆盖的样式, 用文本或其他字符代替图标. 例如通过下面的方式, 修复表格中操作图标无法展示.

```
.el-table .el-icon-delete:before {
  content: '删除';
}
```

```
const ua = window.navigator.userAgent.toLowerCase()
// 不包含 Edge
let ifIE = ua.indexOf('msie') > -1 || ua.indexOf('trident') > -1
if (ifIE) {
  // 依赖 loadjs
  loadjs('/static/css/iefix.css')
}
```

1.4.2 一些兼容性解决方法

1.4.2.1 scrollTop

在应用滚动加载时,发现 `document.documentElement.scrollTop` 在 Chrome 老版本(比如 56)存在 bug,值为 0,可参见 document.documentElement.scrollTop/Left is always zero (body is the scrollingElement even in strict mode)。

```
// [关于scrolltop 兼容 IE6/7/8, Safari,FF的方法](http://www.cnblogs.com/ckmouse/archive/2012/01/30/2332076.html)
const dE = document.documentElement;
const scrollTop = dE.scrollTop || window.pageYOffset || document.body.scrollTop;
```

1.4.3 本地缓存

- 详说 Cookie, LocalStorage 与 SessionStorage

1.5 HTTP

- http 中 get 和 post;
- HTTP 格式: 方案://服务器位置/路径, `<scheme>://<user>:<password>@<host>:<port>/<path>;<params>?<query>#<frag>`

1.5.1 同源

> 如果两个页面拥有相同的**协议**（protocol），**端口**（如果指定），和**主机**，那么这两个页面就属于同一个源（origin）。

> 来自 about:blank，javascript: 和 data:URLs 中的内容，继承了将其载入的文档所指定的源，因为它们的 URL 本身未指定任何关于自身源的信息。

注意 IE 的区别（比如：IE 未将端口号加入到同源策略的组成部分之中）。

reference: JavaScript 的同源策略, MDN

1.5.2 从输入 URL 到页面加载完成的过程

1. 输入地址
2. 浏览器查找域名的 IP 地址。这一步包括 DNS 具体的查找过程，包括：浏览器缓存-> 系统缓存-> 路由器缓存...
3. 浏览器向 web 服务器发送一个 HTTP 请求
4. 服务器的永久重定向响应（从 http://example.com 到 http://www.example.com）
5. 浏览器跟踪重定向地址
6. 服务器处理请求
7. 服务器返回一个 HTTP 响应
8. 浏览器显示 HTML
9. 浏览器发送请求获取嵌入在 HTML 中的资源（如图片、音频、视频、CSS、JS 等等）
10. 浏览器发送异步请求

reference: 从输入 URL 到页面加载完成的过程中都发生了什么事情？, segmentfault

1.5.3 状态码

整体范围	已定义范围	分类
100 ~ 199	100 ~ 101	信息提示
200 ~ 299	200 ~ 206	成功
300 ~ 399	300 ~ 305	重定向
400 ~ 499	400 ~ 415	客户端错误
500 ~ 599	500 ~ 505	服务器错误

状态码	原因短语	含义
100	Continue	说明收到了请求的初始部分, 轻客户端继续. 发送了这个状态码之后，服务器在收到请求之后必须进行响应。
101	Switching Protocols	说明服务器正在根据客户端的指定，将协议切换成 Update 首部所列的协议
200	OK	请求没问题，实体的主体部分包含了所请求的资源
201	Created	用于创建服务器对象的请求 (比如, PUT)。响应的实体主体部分中应该包含各种引用了已创建的资源 URL, Location 首部包含的则是最具体的引用。
202	Accepted	请求已被接受，但服务器还未对其执行任何动作。不能保证服务器会完成这个请求; 这只是意味着接收请求时, 他看起来是有效的，服务器应该在实体的主体中包含对请求状态的描述，或许还应该有队请求完成时间的估计

状态码	原因短语	含义
203	Non-Authoritative Information	实体首部包含的信息不是来自于源服务器，二是来自资源的一份副本。如果中间节点上有一份资源副本，但无法或者没有对它所发送的与资源有关的元星系进行验证，就会出现这种情况。
204	No Content	响应报文中包含若干首部和一个状态行，但没有实体的主体部分。主要用于在浏览器不转为显示新文档的情况下，对其进行更新
205	Reset Content	另一个主要用于浏览器的代码。负责告知浏览器清除当前页面中的所有 HTML 表单元素
206	Partial Content	成功执行了一个部分或 Range 请求。
300	Multiple Choices	客户端请求一个实际指向多个资源的 URL 时会返回这个状态码，比如服务器上有某个 HTML 文档的英语和法语版本。
301	Moved Permanently	在请求的 URL 已被移除时使用。响应的 Location 首部中应该包含资源现在所处的 URL

状态码	原因短语	含义
302	Found	与 301 状态码类似; 但是，客户端应该使用 Location 首部给出的 URL 来临时定位资源。将来的请求仍使用老的 URL
303	See Other	告知客户端应该用另外一个 URL 来获取资源。新的 URL 位于响应报文的 Location 首部。其主要目的是允许 POST 请求的响应将客户端丁香到某个资源上去
304	Not Modified	客户端可以通过所包含的请求首部，使其请求变成有条件的。如果客户端发起了一个 GET 请求，而最近资源未被修改的话，就可以用这个状态码来说明资源未被修改。带有这个状态码的响应不应该包含实体的主体部分。

状态码	原因短语	含义
305	Use Proxy	用来说明必须通过一个代理来访问资源; 代理的位置有 location 首部给出。很重要的一点是，客户端是相对某一个特定的资源来解析这条响应的，不能假定所有请求，甚至所有持有所请求资源的服务器的请求都通过这个代理进行。如果客户端错误的让代理介入了某条请求，可能会引发破坏性的行为，而且会造成安全漏洞。
306	(未使用)	当前未使用
307	Temporary Redirect	与 301 状态码类似; 但客户端应该使用 Location 首部给出 URL 来临时定位资源。将来的请求应该使用老的 URL
400	Bad Request	用于告知客户端它发送了一个错误的请求
401	Unauthorized	与适当的首部一起返回，在这些首部中请求客户端在获取对资源的访问权之前，对自己进行认证。
402	Payment Required	现在这个状态码还未使用，但已经被保留，一做未来之用

状态码	原因短语	含义
403	Forbidden	用于说明请求被服务器拒绝了。如果服务器想说明为什么拒绝请求，可以包含实体的主体部分来对原因进行描述。但这个状态码通常是在服务器不想说明拒绝原因的使用使用的。
404	Not Found	用于说明服务器无法找到所请求的 url。通常会包含一个实体一般客户端应用程序显示给用户看。
405	Mehtod Not Allowed	发起的请求中带有所请求的 url 不支持的方法时，使用此状态码。应该在响应中包含 Allow 首部，已告知客户端对所请求的的资源可以使用哪些方法。
406	Not Acceptable	客户端可以指定参数来说明他们愿意接收什么类型的实体。服务器没有与客户端可以接受的 url 相匹配的资源时，使用此代码。通常服务器会包含一些首部，以便客户端弄清楚为什么请求无法得到满足。

状态码	原因短语	含义
407	Proxy Authentication Required	与 401 类似，但用于要求对资源进行认证的代理服务器。
408	Request Timeout	如果客户端完成请求所化的时间太长，服务器可以回送此状态码，并关闭连接。超时时常随服务器的设置不同而不同，但通常对所有的合法请求来说，都是够长的。
409	Confict	用于说明请求可能在资源上引发一些冲突。服务器担心会引发一些冲突时，可以发送此状态码。响应中应该包含描述冲突的主体。
410	Gone	与 404 类似，只是服务器曾经拥有过此资源。主要用于 Web 站点的维护，这样服务器的管理者就可以在资源被移除的情况下通知客户端了。
411	Length Required	服务器要求在请求报文中包含 Content-Length 首部时使用。
412	Precondition Failed	客户端发起了条件请求，且其中一个条件失败了的时候使用。

状态码	原因短语	含义
413	Request Entity Too Large	客户端所发送请求中的请求 url 比服务器能够或者希望处理的要大时，使用此状态码。
414	Request Uri Too Long	客户端所发送请求中的请求 url 比服务器能够或者希望处理的要长时，使用此状态码。
415	Unsupported Media Type	服务器无法理解或无法支持客户端所发实体的内容类型时，使用此状态码。
416	Requested Range Not Satisfiable	请求报文所请求的是指定资源的某个范围，而此范围无效或无法满足时，使用此状态码
417	Expectation Failed	请求的 Expect 请求首部包含了一个期望，但服务器无法满足此期望时，使用此状态码
500	Internal Server Error	服务器遇到了一个妨碍它为请求提供服务的错误时，使用此状态码
501	Not Implemented	客户端发起的请求超出服务器的能力范围时比如使用了服务器不支持的请求方法)，使用此状态码

状态码	原因短语	含义
502	Bad Gateway	作为代理或网关使用的服务器从请求响应链的链路上收到了一条伪响应 (比如，它无法连接到其父网关) 时，使用此状态码
503	Service Unavailable	用来说明服务器现在无法为请求提供服务，当将来可以。如果服务器知道什么时候资源会变为可使用的，可以在响应中包含一个 Retry-After 首部。
504	Gateway Timeout	与状态码 408 类似，只是这里的响应来自一个网关或代理，他们在等待另一个服务器对其请求进行相应是超时了
505	http Version Not Supported	服务器收到的请求使用了它无法或不愿支持的协议版本时，使用此状态码。有些服务器应用程序会选择不支持协议的早期版本

第二章　FRAMEWORKS

2.1　介绍

1. jQuery: Dom 工具.
2. lodash: A modern JavaScript utility library delivering modularity, performance, & extras.

2.2　Vue.js

Vue.js 开发简单直观, 简单实用的东西通常寿命会比较长.

2.2.1　周边配套

1. 开发小程序: Meituan-Dianping/mpvue
2. 开发原生 APP: weex

2.2.2　Tips

2.2.2.1　本地服务通过 IP 无法访问

1. 方案 1，更改 package.json 中的命令：`webpack-dev-server --port 3000 --hot --host 0.0.0.0`。
2. 方案 2，更改 `config/index.js` 中 `host: 'localhost'` 为 `host: '0.0.0.0'`。

2.2.2.2　动态组件加载

场景：根须不同的条件加载不同的组件，效果类似 React 中，根据条件 Return 不同的视图。

```
<component :is='ComponentName'></component>

<!-- 组件需要传参的场景 -->
<component :is='ComponentName' :yourPropName="binddingIt"></
    ↪ component>
```

如果需要异步加载组件，则采用

```
data () {
  return {
    // 无法异步加载
    // ComponentName: MyComponent,
    ComponentName: () => import('@/components/dynamic/MyComponent')
  }
}
```

参考vuejs-dynamic-async-components-demo

2.2.2.3 组件内事件添加额外的参数

封装的组件提供的事件已经有返回的数据，需要添加额外的参数作预处理。

```
// myComponent 组件内定义的事件
this.$emit('on-change', val);
```

```
<!-- 使用组件，关键在于添加 $event -->
<myComponent @on-change="myChangeEvent($event, myParams)" />
```

```
// 第一个参数为 myComponent 组件内的返回数据，第二个参数为自定义参数
myChangeEvent(val, myParams) {

}
```

2.2.2.4 watch 对象变化

```
watch: {
  form: {
    handler(val) {
      this.$emit('data-change', val);
    },
    // here
```

```
    deep: true,
  },
},
```

2.2.2.5 extend 实现 JS 调用的组件封装

```
// MyComponent/index.js
import Vue from 'vue';
// MyComponent/main.vue 用一般的组件写法编写
import mainVue from './main';
const ConfirmBoxConstructor = Vue.extend(mainVue);
const MyComponent = (options) => {
  const instance = new ConfirmBoxConstructor({
    el: document.createElement('div'),
    // 参数将赋值到 main.vue 中的 data 中，实现配置
    data: options,
  });

  document.body.appendChild(instance.$el);
};

MyComponent.myMethod = () => {
  // define here
}

export default MyComponent;
```

调用组件

```
import MyComponent from 'MyComponent'
export default {
  mounted() {
    const options = {
      // custom here
    };
    MyComponent(options);
    // 方法调用
    // MyComponent.myMethod();
  }
}
```

2.2.2.6 ES6

以下几个 ES6 功能应用于 Vue.js 将获得不错的收益[1], 特别是对于无需构建工具的情况.

1. 箭头函数: 让 this 始终指向到 Vue 实例上.
2. 模板字符串: 应用于 Vue 行内模板, 可以方便换行, 无需用加号链接. 也可以应用于变量套入到字符串中.

```
Vue.component({
  template: `<div>
             <h1></h1>
             <p></p>
           </div>`
  data: {
    time: `time: ${Date.now()}`
  }
});
```

3. 模块 (Modules): 应用于声明式的组件 `Vue.component`, 甚至不需要 webpack 的支持.

```
import component1 from './component1.js';
Vue.component('component1', component1);
```

4. 解构赋值: 可应用于只获取需要的值, 减少不必要的赋值, 比如只获取 Vuex 中的 commit 而不需要 store.

```
actions: {
  increment ({ commit }) {
    commit(...);
  }
}
```

5. 扩展运算符: 数组和对象等批量导出, 而不需要用循环语句. 比如, 将路由根据功能划分为多个文件, 再用扩展展运算符在 index 中合在一起.

2.2.2.7 组件重新渲染

通过设置 `v-if` 实现, 从 Dom 中剔除再加入.

```
<demo-component v-if="ifShow"></demo-component>
```

[1]ANTHONY GORE, 4 Essential ES2015 Features For Vue.js Development, 2018-01-22

2.2.2.8 绑定数据后添加属性视图未重新渲染

如果存在异步请求, 在数据上添加属性的情况, 需要先预处理好获取的数据, 然后在将其赋值到 data 中变量. 数据绑定后, 再添加属性, 不会触发界面渲染.

```
API.getSomething().then(res => {
  // 1. 先添加属性
  // handle 表示对数据的处理, 包括对象中属性的添加
  const handledRes = handle(res);
  // 2. 然后绑定到 data 中的变量
  this.varInDate = handledRes;
});
```

2.2.2.9 全局引入 SCSS 变量文件[2]

场景: 将常用的变量存储到 `vars.scss`, 应用变量时需要在每个需要的地方 `import`.

1. `npm install sass-resources-loader --save-dev`
2. 更改 `build/webpack.base.conf.js`, 适用于 vue-cli.

```
{
   test: /\.vue$/,
   loader: 'vue-loader',
   options: {
      loaders: {
         sass: ['vue-style-loader', 'css-loader', {
            loader: 'sass-loader',
            options: {
               indentedSyntax: true
            }
         }, {
            loader: 'sass-resources-loader',
            options: {
               resources: path.resolve(__dirname, "./styles/vars.
                   ↪ scss")
            }
         }],
         scss: ['vue-style-loader', 'css-loader', 'sass-loader', {
```

[2] https://www.reddit.com/r/vuejs/comments/7o663j/sassscss_in_vue_where_to_store_variables/?st=JC9T45PB&sh=4f87ec9d

```
            loader: 'sass-resources-loader',
            options: {
                resources: path.resolve(__dirname, "./styles/vars.
                    ↪ scss")
            }
        }]
    }
    // other vue-loader options go here
  }
}
```

2.2.3 Compatible

2.2.3.1 IE vuex requires a promise polyfill in this browser

```
npm install --save-dev babel-polyfill
```

```
// build/webpack.base.conf.js
entry: {
  app: [
    'babel-polyfill',
    './src/main.js'
  ]
}
```

vuex requires a promise polyfill in this browser

2.3 React

```
npm install -g create-react-app
create-react-app my-app
```

React 与 Vue 类似，主要专注于视图层，结合其他的库实现扩展，比如路由、数据状态维护等。

2.3.1 常用依赖

1. 路由: React Router
2. 类型检查: prop-types
3. 数据管理: React Redux

2.3.1.1 UI

- element-react
- ant-design

2.3.1.2 优化

- reselector
- immutable.js
- seamless-immutable.js

2.3.2 文件组织

2.3.3 React 与 Redux

引入 Redux 用于应用状态维护，为了方便与 React 结合，因此会同时引入 React Redux。用户在视图层（React）操作，触发事件（action），事件触发 reducer，在 reducer 中决定如何返回新的状态，状态的更新触发视图的更新。从而实现了单向的数据流。

reducer 和 action 分文件夹放置，reducer 通过 combineReducers 合并，然后再通过 createStore 创建 store。store 通过 Provider 放置在 App 的顶层元素，并分发给所有子组件。对每个具体的功能，分别按需引入需要的 state（reducer 中定义）和 action，并用 connect 将此两项和视图层（react 实现）连接起来，实现完整的组件。由此，实现了数据、视图、事件之间的分离，通过 connect 实现连接。

reducer 是什么呢？reducer 按照 Redux 的输入书写。每次输入确定的数据，通过 reducer 处理，能够得到固定的输出。也就是说 reducer 中不包含随机因素或者环境因素，因此称之为纯函数。

2.3.4 扩展

1. 关于 actionTypes, actions, reducer 文件分割的提议:GitHub, erikras/ducks-modular-redux
2. React 生命周期及方法图:wojtekmaj/react-lifecycle-methods-diagram

2.4 React Native

1. 主页: `https://facebook.github.io/react-native`
2. GitHub: `https://github.com/facebook/react-native`
3. 示例项目: amazing-react-projects
4. Demo Project: react-native

2.4.1 环境配置

2.4.1.1 系统环境

1. 安装 nodejs.
2. `npm install -g react-native-cli`.

Android

1. JDK (并配置环境变量)
2. 安装 Android Studio `http://www.android-studio.org`
3. 通过 SDK Manager 下载 SDK, 并配置环境变量.

```
REM set var
set ANDROID_HOME=C:\Users\chenl\AppData\Local\Android\Sdk

REM set Android home path
setx /m ANDROID_HOME "%ANDROID_HOME%"

REM set path
setx /m path "%path%;%ANDROID_HOME%\tools;%ANDROID_HOME%\platform-
    ↪ tools;"
```

iOS

1. App Store 安装 XCode.
2. 其他工具安装

```
brew install node
brew install watchman
npm install -g react-native-cli
```

2.4.1.2 编辑器

1. Visual Studio Code. 安装扩展 `React Native Tools` 用于调试.
2. Atom. 安装nuclide.

2.4.1.3 参考

1. `https://facebook.github.io/react-native/docs/getting-started.html`

2.4.2 基本命令

1. 新建工程: `react-native init demo-project`.
2. Android 运行: `react-native run-android`.
3. iOS 运行: `react-native run-ios`.

新建工程后首先 `npm install` 安装依赖. 示例项目 python 和 node-gyp-bin 相关错误可以尝试先执行 `yarn add node-sass` 或者 `npm install -f node-sass` (`https://github.com/sass/node-sass/issues/1980`).

2.4.3 打包

2.4.3.1 Android 打包

生成签名密钥

```
$ keytool -genkey -v -keystore my-release-key.keystore -alias my-key-alias -keyalg RSA -keysize 2048 -validity 10000
Enter keystore password:
Keystore password is too short - must be at least 6 characters
Enter keystore password: chenlei
Re-enter new password: chenlei
What is your first and last name?
  [Unknown]: HereChen
What is the name of your organizational unit?
  [Unknown]: HereChen
What is the name of your organization?
  [Unknown]: HereChen
What is the name of your City or Locality?
  [Unknown]: Chengdu
What is the name of your State or Province?
  [Unknown]: Sichuan
```

```
What is the two-letter country code for this unit?
  [Unknown]: 51
Is CN=HereChen, OU=HereChen, O=HereChen, L=Chengdu, ST=Sichuan, C
    ↪ =51 correct?
  [no]: yes

Generating 2,048 bit RSA key pair and self-signed certificate (
    ↪ SHA256withRSA) with a validity of 10,000 days
       for: CN=HereChen, OU=HereChen, O=HereChen, L=Chengdu, ST=
           ↪ Sichuan, C=51
Enter key password for <my-key-alias>
       (RETURN if same as keystore password):
[Storing my-release-key.keystore]
```

gradle 设置

1. `my-release-key.keystore` 文件放到工程 `android/app` 文件夹下.
2. 编辑 `android/app/gradle.properties`, 添加如下信息.

```
MYAPP_RELEASE_STORE_FILE=my-release-key.keystore
MYAPP_RELEASE_KEY_ALIAS=my-key-alias
MYAPP_RELEASE_STORE_PASSWORD=chenlei
MYAPP_RELEASE_KEY_PASSWORD=chenlei
```

3. 编辑 `android/app/build.gradle`, 添加如下信息.

```
...
android {
    ...
    defaultConfig { ... }
    signingConfigs {
        release {
            storeFile file(MYAPP_RELEASE_STORE_FILE)
            storePassword MYAPP_RELEASE_STORE_PASSWORD
            keyAlias MYAPP_RELEASE_KEY_ALIAS
            keyPassword MYAPP_RELEASE_KEY_PASSWORD
        }
    }
    buildTypes {
        release {
            ...
            signingConfig signingConfigs.release
```

```
        }
    }
}
...
```

生成 apk

```
cd android && ./gradlew assembleRelease
```

打包后在 `android/app/build/outputs/apk/app-release.apk`.

安装 apk 方式

1. Genymotion 可以拖拽 apk 进行安装.
2. `adb install app-release.apk` 安装.
   如果报签名错误, 可先卸载之前的 debug 版本.

2.4.3.2 iOS 打包

iOS 版本编译需要在 Mac 上进行.

签名 没有证书….

生成 ipa 以下流程以 Xcode 9 为例.

1. 打开工程: Xcode 打开 `ios` 文件夹下 `*.xcodeproj` 文件 (工程).
2. 选择编译机型: Xcode 虚拟机选择栏中选择 `Generic iOS Device`.
3. 编译设置: Xcode -> Product -> Scheme -> Edit Scheme -> Run -> Info -> Build Configuration 选择 Rlease
4. JS 改为离线 (打包进 APP)???
   TODO: 命令行打包

2.4.3.3 参考

1. Generating Signed APK, Facebook Open Source
2. 打包 APK, React Native 中文网
3. ReactNative 之 Android 打包 APK 方法(趟坑过程), ZPengs, 2017.02.09, 简书

2.4.4 入口文件更改

从 0.49 开始, 只有一个入口, 不区分 ios 和 android. `https://github.com/facebook/react-native/releases/tag/v0.49.0`

React Native CLI 新建的工程, 默认入口是 `index.js`. 在 `android\app\build.gradle` 中更改入口.

```
project.ext.react = [
    entryFile: "index.android.js"
]
```

对应更改 `android\app\src\main\java\com\**\MainApplication.java`.

```
protected String getJSMainModuleName() {
  return "index.android";
}
```

2.4.5 工具/依赖 (dependencies)

2.4.5.1 导航

`https://facebook.github.io/react-native/docs/navigation.html`

1. react-navigation 提供了常用的导航方式 (Stack, Tab, Drawer), 推荐.
2. NavigatorIOS 为内建的导航, 仅在 IOS 上可用.

2.4.5.2 UI

尚未找到两端 (Web, Native) 完整好用的 UI, 若后端采用 ant-design 可用 ant-design-mobile.

1. ant-design-mobile 每个组件是否支持 Native 有说明.
2. react-native-elements
3. NativeBase

2.4.5.3 HTTP 请求

`https://facebook.github.io/react-native/docs/network.html`

1. fetch 为内建接口.
2. **axios** 为使用校广泛的第三方请求库, 推荐使用.

2.4.6 调试

`https://facebook.github.io/react-native/docs/debugging.html`

根据提示, 可以菜单按钮选择重新加载或热加载. Android 可摇晃手机显示菜单.

2.4.6.1 虚拟机

1. Genymotion, 需要先注册, 然后选择 for personal 使用. 如果系统开启了 Hyper-V, 需要先关闭.
2. Android Studio 内建虚拟机, 同样需要关闭 Hyper-V.
3. Visual Studio Emulator for Android 需要开启 Hyper-V.

2.4.6.2 调试工具: Chrome

1. `Remote JS Debugging` 开启 JS 调试.
2. 浏览器端进去 `http://localhost:8081/debugger-ui/`, 并开启开发工具.
3. 可在 Sources 中设置断点或者代码中写入 `debugger`.

2.4.6.3 调试工具: VSCode

1. 安装扩展: React Native Tools.
2. F5 生成 launch.json 文件.
3. 进入调试菜单 (Ctrl + Shift + D), 选择 Debug Android.
4. 设置断点或者写入 `debugger` 开始调试, 在 output 栏输出.

2.4.6.4 HTTP 调试问题备注

应用 Fiddler 调试 HTTP, 模拟器设置了代理后, APP 无法热加载 JS bundle. 目前只有用 Chrome 或者断点的方式来调试.

2.4.7 工程结构

2.4.7.1 结构

```
android/       # Android 工程
ios/           # IOS 工程
```

```
src/          # 开发前端资源
  -- assets/   # 静态资源
  -- components/ # 组件
  -- api/      # 接口
  -- route/    # 导航(路由)
  -- config/   # 常量配置
  -- pages/    # 页面/功能
  -- utils/    # 常用工具
  -- reducers 相关
  -- index.js  # APP 入口
index.js       # 入口文件
```

2.4.7.2 参考

1. Organizing a React Native Project
2. React native project setup—a better folder structure

2.4.8 Tips

1. Android 查看当前的 Android 设备 `adb devices`.
2. Android 虚拟机: Ctrl + M 打开菜单 (Android Studio 自带虚拟机没有菜单和摇晃手机, 可以这种方式打开菜单).
3. iPhone 虚拟机啊重新加载资源: command + R.

2.4.9 问题及解决

1. VSCode Debug 无法加载的情况, 首先重启 VSCode 再启动项目.
2. 添加`antd-mobile`后报错, 无法解析 `react-dom`, 依赖中加入`react-dom`并安装即可.
3. 集成`react-native-navigation`需要注意 Android SDK 版本, 版本过低可能出现编译错误 (`Error:Error retrieving parent for item: No resource found`).

2.4.10 原理

1. React Native 将代码由 JSX 转化为 JS 组件，启动过程中利用 instantiateReactComponent 将 ReactElement 转化为复合组件 React-

CompositeComponent 与元组件 ReactNativeBaseComponent，利用 ReactReconciler 对他们进行渲染[3]。

2. UIManager.js 利用 C++ 层的 Instance.cpp 将 UI 信息传递给 UIManagerModule.java，并利用 UIManagerModule.java 构建 UI[4]。
3. UIManagerModule.java 接收到 UI 信息后，将 UI 的操作封装成对应的 Action，放在队列中等待执行。各种 UI 的操作，例如创建、销毁、更新等便在队列里完成，UI 最终得以渲染在屏幕上[5]。

2.5 Weex

1. 主页: `http://weex.apache.org`
2. GitHub: `https://github.com/apache/incubator-weex/`

问题: 入口在哪儿?

案例

1. 网易严选
2. 点我达骑手 Weex 最佳实践
3. weexteam/weex-hackernews

2.5.1 搭建开发环境

```
npm install -g weex-toolkit
```

2.5.2 Demo

web

```
weex create weex
cd weex
npm install
npm run dev & npm run serve
```

命令

https://github.com/weexteam/weex-pack

[3]ReactNative 源码篇：渲染原理
[4]ReactNative 源码篇：渲染原理
[5]ReactNative 源码篇：渲染原理

```
# debug
weex debug

# add platform
weex platform add android
weex platform add ios

# run
weex run web
weex run android
weex run ios

# build
weex build web
```

2.5.3 问题及解决

1. `https://maven.google.com/` 链接不上, 更改`\platforms\android\build.gradle` ↪ 文件, 换成 `https://dl.google.com/dl/android/maven2/`。
2. `adb: failed to stat app/build/outputs/apk/playground.apk: No such file` ↪ `or directory`, 替换 `platforms/android/app/build.gradle` 文件中的 `weex-app.apk` 为 `playground.apk`.
3. `weex debug` 报错可先安装 `npm install -g weex-devtool`.

2.6 React Native vs Weex

2.6.1 对比表格

属性	React Native	Weex
开源时间	2015/03	2016/06
开源企业	Facebook	Alibaba
协议	BSD 3-clause	Apache License 2.0
主页标语	Build native mobile apps using JavaScript and React	A framework for building Mobile cross-paltform UIs

属性	React Native	Weex
核心理念	Learn Once, Write Anywhere	Write Once, Run Everywhere
前端框架	React	Vue.js
JS Engine	JavaScriptCore(iOS/Android)	JavaScriptCore(iOS) /v8(Android)
三端开发	部分组件需要区分平台开发	强调三端统一
代码写法	JSX(JavaScript + XML)	Web 写法
调试	虚拟机	可用 Chrome 查看效果
社区支持	社区活跃, 有多个流行产品的实践	目前, 开发者主要在国内, 没有太多的实践案例
优势	生态好, 第三方依赖多, 有可借鉴的经验	基于 Vue.js, 上手快, 能更好的保证三端一致

以下参考都是 2016 年文章.

1. compare weex to react native
2. Weex 简介
3. Weex & React Native

2.6.2 评论摘抄

> After a few days of experimentation, I realized Weex and its documentation were not yet developed enough to for us to use to deliver top-quality apps. This was my experience with Weex. Sam Landfried, 2017.10.20, Is VueJS' Weex a Suitable Alternative to React Native?

2.7 Element

`https://github.com/ElemeFE/element`

2.7.1 组件使用

2.7.1.1 自定义表单校验

```
export default {
  data: function () {
    var checkVars = function (rule, value, callback) {
      if (!value) {
        callback(new Error('不能为空'));
      } else {
        callback();
      }
    };
    return {
      rules: {
        vars: [{
          required: true,
          trigger: 'change',
          validator: checkVars
        }]
      }
    }
  }
}
```

2.7.2 兼容性

2.7.2.1 IE 图标不显示

可用文字替代伪元素中的内容.

2.8 Electron

Electron 构建应用，主要包括两部分：一是构建 Web App；二是 Electron 提供扩展能力（内置了 Node，可用 node 的模块），Web App 实现本地交互能力。构建 Electron 应用工程，可首先构建 Web App 工程，稍作修改适应客户端最后打包的需求。

2.8.1 技术栈

应用 (TypeScript + React) + 打包 (electron-builder)。

2.8.1.1 准备

1. Electron 支持 TypeScript，Announcing TypeScript support in Electron，模板工程参考 electron-quick-start-typescript。
2. React TypeScript 模板工程参考 TypeScript-React-Starter。
3. 打包工具 electron-builder。

思路，首先建立一个 Web 应用工程（通常为单页应用，无需管理多窗口），然后建一个桌面应用入口文件，入口文件加载 Web App 应用，最后用工具将 electron 和 Web 应用打包在一起。结合以上的两个模板工程，优先构建 Web 应用，然后加入 Electron 相关依赖的脚本。

2.8.1.2 工程搭建

创建工程流程，可先用 create-react-app 创建 React 应用工程，然后作调整，并加入 Electron 相关依赖的脚本。

1. 创建 React 工程

```
create-react-app my-app --scripts-version=react-scripts-ts
```

2. 更改 tsconfig：解决 main.ts 编译问题

```
{
   "compilerOptions": {
      "module": "commonjs",
      // 关闭窗口时设置为 null 禁用 null 检查
      "strictNullChecks": false
   }
}
```

3. 静态资源路径需要改为相对路径。
4. 将 electron-quick-start-typescript 中的 main.ts 移到当前工程 `src` 下。

注意 main.ts 编译后 index.html 以及静态资源路径问题。

2.8.2 资源及扩展阅读

1. Technical Differences Between Electron and NW.js(formerly node-webkit)

第三章　TOOLS

3.1　parcel 与 React 搭建项目

parcel 支持热加载、source map、code split (需要按既定格式书写代码).

3.1.1　基础

1. 初始化项目

```
npm init -y
```

2. 安装资源依赖: react 相关依赖

```
npm i --save react react-dom
```

3. 安装工具依赖: parcel

```
npm i --save-dev parcel-bundler
```

4. 启动命令: `npm run start`

```
"scripts": {
  "start": "parcel src/index.html"
}
```

`src/index.js`

```
import React from "react";
import ReactDOM from "react-dom";

const App = () => {
  return <h1>Hello World! HOT</h1>;
};

ReactDOM.render(<App />, document.getElementById("root"));
```

src/index.html

```
<!DOCTYPE html>
<html lang="zh">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale
        ↪ =1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>APP</title>
</head>
<body>
    <div id="root"></div>
    <script src="./index.js"></script>
</body>
</html>
```

3.2 Visual Studio Code

1. 主页: https://code.visualstudio.com/
2. 快捷键: https://code.visualstudio.com/shortcuts/keyboard-shortcuts-windows.pdf

3.2.1 基本用法

1. 命令搜索工具: F1.
2. 扩展安装: ext install extensionname

3.2.2 Vue.js 配置

3.2.2.1 代码高亮

1. 安装 Vetur 扩展.

3.2.2.2 ESLint 问题自动修复

```
{
  "eslint.enable": true,
  "eslint.options": {
```

```
    "extensions": [
      ".html",
      ".js",
      ".vue"
    ]
  },
  "eslint.validate": [{
     "language": "html",
     "autoFix": true
    },
    {
      "language": "vue",
      "autoFix": true
    },
    {
      "language": "javascript",
      "autoFix": true
    }
  ]
}
```

3.3 Vue-CLI

1. GitHub: `https://github.com/vuejs/vue-cli`

3.3.1 代理

配置代理可解决跨域问题, 需要服务端配置跨域.

```
// config/index.js
proxyTable: {
  '/api': {
     target: 'http://stage.xxxx.com',
     changeOrigin: true,
     pathRewrite: {
     '^/api': '/api'
     }
  }
}
```

第四章　PERFORMANCE

4.1　文件

4.1.1　图片

4.1.1.1　图片格式的选择

webp, gif, png, jpg, icon font

4.1.1.2　icon font

字体图片有两个优点: 矢量图放大后不失真; 起到图片精灵的作用, 减少图片请求次数.

图片转成字体文件, 作为矢量图, 常用于图标. 工具可用 iconfont 上传后生成 CSS 文件和字体.

4.1.1.3　图片延迟加载/懒加载 (lazy load)

思路　延迟加载通常是将暂不需要的资源延后加载. 懒加载是延迟加载的一种, 即达到某个条件 (或某个事件触发) 时加载图片.

延迟加载可处理为, 当必要的资源加载完后再加载其余资源. 懒加载基本思路:

1. 暂存一张图片, 显示该默认图片.
2. 显示图片的元素在可视区域时, 加载该图片.

实例　具体到技术, 飞猪 H5 的实现方法是:

```
1 <div class="base-bg base-bg-m regular-product__image___Bu73a" data-
      ↪ reactid=".0.$=1$trip_home_arbitrary_gate_product_0.0.$=1
      ↪ $regular_item_1.0.$=10">
```

```
    <div data-lazyloadid="lazyload_item_36" class="fade" style="
        ↪ opacity: 1;background-image: url(&quot;//gw.alicdn.com/
        ↪ tips/i3/638737216/TB2vvwZtVXXXXX0XXXXXXXXXXXX_
        ↪ !!638737216.jpg_400x400q75.jpg_.webp&quot;);"
       data-reactid=".0.$=1$trip_home_arbitrary_gate_product_0.0.$=1
           ↪ $regular_item_1.0.$=10.$=11" data-imageloaded="true">
           ↪ </div>
</div>
```

1. 父元素上设置默认的背景图片.

```
.skin-yellow .base-bg {
  background: #f2f3f4 url(
      ↪ iVBORw0KGgoAAAANSUhEUgAAALkAAABPCAMAAACAuJRqAAAAq1BMV···
      ↪ mgg7e+vIXHxHbzIMosU7LAtcvNOAUKpxf6kSUl8MPvAnj+
      ↪ AYRcPQeahlKYAAAAAElFTkSuQmCC) 50% no-repeat;
  background-size: auto .8rem;
}
```

2. 子元素内联样式背景图片链接, 外链样式图片相关属性. 初始化时 `opacity: 0`, 并且不包含背景设置.

```
opacity: 1;
background-image: url(//gw.alicdn.com/tips/i3/638737216/
    ↪ TB2vvwZtVXXXXX0XXXXXXXXXXXX_!!638737216.jpg_400x400q75.jpg_.
    ↪ webp);
```

```
.base-bg>div {
  width: 100%;
  height: 100%;
  background-repeat: no-repeat;
  background-position: 50%;
  background-size: cover;
}
```

3. **满足条件时**, 设置子元素的背景图片 (或者设置 img src 属性), 然后标识已加载的标签一个属性 (比如 `data-imageloaded="true"`), 如果是 img 标签, 加载后删除 `data-src`.

关键点 这里的**满足条件时**, 可用以下逻辑. 检查元素是否在可视区域, 可全局循环检查, 至于是否有性能问题, 待考察.

```
loadIfVisible() // 如果在可视区域则加载
```

```
onScroll(loadIfVisible()); // 滚动事件触发时, 检查
```

判断元素是否在可视区域

```
// 判断元素是否在可视区域
function isInView(obj) {
  var e = obj.getBoundingClientRect();
  return !(e.top > window.innerHeight || e.bottom < 0 || e.left >
      ↪ window.innerWidth || e.right < 0)
}
```

参考扩展

1. stackoverflow, How to tell if a DOM element is visible in the current viewport?
2. mozilla, Element.getBoundingClientRect()

4.2 体验优化

对页面性能的优化算起来都是体验优化, 这里主要指具有进一步提升性质的. 比如, 骨架屏实际上也可以用转圈圈来替代, 但其使用感受更好.

4.2.1 骨架屏/Skeleton Screen

骨架屏指的是数据呈现之前, 一般用浅色的色条勾勒渲染后的轮廓. 相对通常的空白区域或者加 loading, 体验会好一些. 其次还起到了占位的作用.

文章

1. Skeleton Screen – 骨架屏
2. How to Speed Up Your UX with Skeleton Screens
3. Building Skeleton Screens with CSS Custom Properties

实例

Ant Design 的 loading card, `https://ant.design/components/card/`

第五章　SOLUTIONS

5.1　跨域资源共享/CORS (Cross-origin resource sharing)

- TAT.Johnny, iframe 跨域通信的通用解决方案, alloyteam
- JasonKidd,「JavaScript」四种跨域方式详解, segmentfault

5.2　跨站请求伪造/CSRF (Cross-site request forgery)

5.3　权限管理

5.3.1　AngularJS 分角色登录

不同角色/权限登录后所见菜单不一样. 方案如下:

1. 给不同的路由配置其角色/权限属性.
2. 登录进入时, 记录角色/权限.
3. 进入主页, 根据角色/权限构建菜单 (view 中包含全部菜单, 非此角色菜单移除 Dom).
4. 点击菜单进入到对应路由时, 根据判断路由的角色/权限属性是否和登录进入时记录的一样.

此方案包括两部分的权限限制, 其一是将不必要的菜单移除 Dom, 但菜单对应的路由依然可用, 只是在页面上没有对应可操作的视图, 其二是路由和登录的角色/权限匹配. 以 AngularJS 为例, 对应每个步骤的代码如下.

1. 路由属性.

```
1 $stateProvider.state('Registration.Instructors', {
```

```
    url: "/Instructors",
    templateUrl: '/Scripts/App/Instructors/Templates/instructors.
        ↪ html',
    controller: 'InstructorController',
    data: { auth: "Admin"}
})
```

2. 登录用户权限/角色信息可记录到 rootScope 中, 比如 rootScope.adminType = "Admin".
3. 菜单保留与移除. ng-if="adminType==='Admin'"[1].
4. 路由和登录角色/权限匹配[2].

```
app.run(function($rootScope){
  $rootScope.$on('$stateChangeStart', function(event, toState,
      ↪ toParams, fromState, fromParams){
    if ( toState.data.auth !== $rootScope.adminType ) {
       event.preventDefault();
       return false;
    }
  })
});
```

5.3.2 Vue.js 权限管理

- 基于 Vue 实现后台系统权限控制
- 用 addRoutes 实现动态路由
- 手摸手，带你用 vue 撸后台系列二 (登录权限篇)
- Vue 后台管理控制用户权限的解决方案?
- 自定义指令
- https://codepen.io/diemah77/pen/GZGxPK

5.4 代码规范

5.4.1 编辑器文本规范

规范文件采用的换行符、缩进方式以及编码等等.

1. EditorConfig: http://editorconfig.org/

[1] what is the difference between ng-if and ng-show/ng-hide

[2] angularjs: conditional routing in app.config

2. VSCode 插件: EditorConfig for VS Code, 可生成配置样本.
   `.editorconfig`配置样例

```
root = true

[*]
indent_style = space
indent_size = 2
charset = utf-8
end_of_line = lf
insert_final_newline = true
trim_trailing_whitespace = true
max_line_length = 80
```

5.4.2 命名规范

5.4.2.1 CSS 命名

1. BEM, Block Element Modifier.
2. SMACSS, Scalable and Modular Architecture for CSS.
3. OOCSS, Object-Oriented CSS.
4. Atomic CSS.

5.4.3 代码检查

5.4.3.1 CSS 格式化

1. CSScomb: `http://csscomb.com`
2. 配置文件`.csscomb.json`示例:`https://github.com/htmlacademy/codeguide/blob/master/csscomb.json`
3. VSCode 插件: CSScomb

5.4.3.2 JS 静态代码检查工具

1. ESLint: `https://github.com/eslint/eslint`
2. JSLint: `https://github.com/jshint/jshint/`

5.4.3.3 JS 语法规范

1. airbnb: `https://github.com/airbnb/javascript`

2. standard: `https://github.com/standard/standard`

5.4.4 扩展

1. 贺师俊, Myths of CSS Frameworks, 2015
2. 白牙, [译] 结合智能选择器的语义化的 CSS, 2013-10-06

5.5 兼容性问题解决

5.5.1 IE: 盒模型

IE 默认情况下长宽包含 `padding` 和 `border`, 和其他浏览器的长宽存在区别, 建议添加 `border-sizing` 属性. 保持多个浏览器的一致性.

```
box-sizing: border-box;
```

5.5.2 IE 8: map

IE8 不支持 JavaScript 原生 map 函数, 可在任意地方加入如下的代码片段[3].

```
(function (fn) {
    if (!fn.map) fn.map = function (f) {
        var r = [];
        for (var i = 0; i < this.length; i++)
            if (this[i] !== undefined) r[i] = f(this[i]);
        return r
    }
    if (!fn.filter) fn.filter = function (f) {
        var r = [];
        for (var i = 0; i < this.length; i++)
            if (this[i] !== undefined && f(this[i])) r[i] = this[i];
        return r
    }
})(Array.prototype);
```

或者用 jQuery 的 map 函数.

[3] Is the javascript .map() function supported in IE8?

```
// array.map(function( ) { });
jQuery.map(array, function( ) {
}
```

5.5.3 IE 8: fontawesome 图标显示为方块

对于修饰性不影响功能的图标, 可以做降级处理, 仅在非 IE 或者 IE9+ (条件注释[4]) 情况下引入 fontawesome 图标库. (谷歌搜索了一堆方案都没用, 最后应用这种方式来解决).

```
<!--[if (gt IE 8) | !IE]><!-->
<link rel="stylesheet" href="font-awesome.min.css">
<!--<![endif]-->
```

5.5.4 IE 10+ 浏览器定位

IE 10+ 不支持条件注释, 因此需要其他方式定位这些浏览器. 如果只增加 CSS, 可采用以下方式定位[5].

```
/*IE 9+, 以及 Chrome*/
@media screen and (min-width:0\0) {
}

/*IE 10*/
@media all and (-ms-high-contrast: none), (-ms-high-contrast:
    ↪ active) {
}

/*Edge*/
@supports (-ms-accelerator:true) {
}
```

另一种方式是 JavaScript 检测浏览器版本, 在 `body` 标签为特定浏览器添加 `class` 属性标识.

5.5.5 IE 6-8 CSS3 媒体查询 (Media Query)

引入 Respond.js.

[4] About conditional comments

[5] How do I target only Internet Explorer 10 for certain situations like Internet Explorer-specific CSS or Internet Explorer-specific JavaScript code?

```
<!--[if lt IE 9]>
<script src="respond.min.js"></script>
<![endif]-->
```

5.6 npm 包及私有库

5.6.1 npm 包编写

npm 包通常会兼容不同的应用场景: nodejs、require.js 和浏览器. 所以会包含一段用于判断运行环境的代码, 如下.

```
// from Vue.js
(function (global, factory) {
  typeof exports === 'object' && typeof module !== 'undefined' ?
      ↪ module.exports = factory() :
  typeof define === 'function' && define.amd ? define(factory) : (
      ↪ global.NPM_THING = factory());
}(this, (function () {
  'use strict';
  // code
  // return NPM_THING;
})));
```

1. 判断是否是 nodejs 环境: `typeof exports === 'object' && typeof module !== 'undefined'`.
2. 判断是否是 require.js: `typeof define === 'function' && define.amd`

其中的 `this` 指向全局变量, nodejs `this` 为 `global`, 浏览器中 `this` 为 `window`.

5.6.2 私有库方案

为了避免重复造轮子, 提供编码效率, 同时又可以避免企业内部的业务逻辑暴露, 于是对私有库有需求. 期望, 如果私有库中有, 则从私有库中下载, 否则从公开的库中下载.

npm 的包都是公开的, 提供的企业私有化方案是收费的. 开源方案有:

1. cnpm: `https://github.com/cnpm/cnpmjs.org`
2. sinopia: `https://github.com/rlidwka/sinopia`

两者的对比 (企业私有 npm 服务器):

–	cnpm	sinopia
系统支持	非 windows	全系统
安装	复杂	简单
配置	较多，适合个性化需求较多的	较少
配置——修改默认镜像	不支持	支持
存储	mysql	文件格式，直观
服务托管	默认后台运行	pm2, doker, forever
文档资料	较多	较少

5.7 HTML

5.7.1 参数 (input) 在 form 之外

input 在 form 之外时, 在 input 元素内添加 form 属性值为 form 的 ID[6]. 这样 input 仍然可以看做隶属于此表单, jQuery `$('#formid').serialize();` 能够获取 form 之外的输入框值. 或者在提交 (`submit`) 表单时会同样提交 `outside` 这个值.

```
<form id="formid" method="get">

    <label>Name:</label>
    <input type="text" id="name" name="name">

    <label>Email:</label>
    <input type="email" id="email" name="email">
    <input type="submit" form="contact_form" value="send form" />
</form>
<input type="text" name="outside" form="formid">
```

注意: IE8 `$('#formid').serialize();` 无法获取 `outside` 值.

5.8 模块化与组件化

模块化，强调内聚，包含完整的业务逻辑，可以方便业务的复用。组件化，强调复用，重点在于接口的暴露，和构件的概念类似。

[6]PLACING FORM FIELDS OUTSIDE THE FORM TAG

5.8.1 实现模块化

业务相关的特殊性都应包含在同一个模块内，具体到前端，这些特性包括与业务相关的接口、状态、路由等。

5.8.2 实现组件化

关键是如何暴露接口，方便外部复用。

5.9 前后端分离

内容简述: 简单描述前后端分离的历史状况, 明确前后端划分的原则: 后端面向数据, 前端面向用户, 在服务端引入 nodejs 可以解决的问题 (应用的场景).

5.9.1 历史

1. 前后端耦合. 例如, ASP.NET Webform 和 jsp 的标记语言的写法, 每次请求由后端返回, 且后端的语言变量混在 HTML 标签中.
2. 前后端半分离. 例如, ASP.NET MVC 和 Spring MVC 视图由后端控制, V (视图) 由前端人员开发. 开发新的页面需要后端新建接口, 编程语言通常在一个工程中, and so on.
3. 前后端完全分离. 前后端通过接口联系. 前后端会有部分逻辑重合, 比如用户输入的校验, 通常后端接口也会处理一次. 前端获取数据后渲染视图, SEO 困难.

5.9.2 目标/方法

1. 后端: 数据处理; 前端: 用户交互[7].
2. 前端向后扩展 (服务端 nodejs): 解决 SEO、首屏优化、部分业务逻辑复用等问题; 前端向前扩展: 实现跨终端 (iOS 和 Android, H5, PC) 代码复用.

解决的问题:

1. core: 优化交互体验, 提高编码效率.
2. SEO.

[7] Balint Sera, On the separation of front-end and backend, 2016-06-15

3. 性能优化.
4. 首屏优化.
5. 代码复用 (业务逻辑, 路由, 模板).

5.9.3 应用

1. 服务端框架: Koa, Express.
2. 同构框架支持: nuxt.js(可用 Koa 替换 Express).
3. 路由用 history mode (Vue.js), 如果后端不配置, 直接进入页面无法访问. 可复用模板, 直接访问时后端渲染, 路由访问时前端渲染[8].
4. 服务端, 浏览器端及 Native 端都可应用的第三方库: axios, moment.js.

5.9.4 引入 nodejs 层的应用场景

除了上面所述的性能和 SEO 等问题, 还可以作为中间件, 抹平同类型系统的差异, 构建统一的平台.

比如, 对于多个定制化的产品, 每个产品都对应有运营平台, 用于观测用户使用情况. 由于历史上造成的差异, 每个运营平台都需要重新构建一套运行于浏览器端的前端工程. 对于这种业务相似度较高的情况, 就可以在服务端引入 nodejs, 构建统一的平台, 抹平已有系统之前的差异 (比如接口有不同的风格), 只需要实现一套 Web APP. 同时也方便了后期其他定制产品的扩展.

5.9.5 扩展

1. 美团点评点餐, 美团点评点餐 Nuxt.js 实战, 2017-08-09
2. Jason Strimpel, Maxime Najim, 同构 JavaScript 应用开发, 2017
3. Nicholas C. Zakas, Node.js and the new web front-end, 2013-10-07

5.10 JavaScript 同构

承接前后端分离的主题，详述同构相关内容。

5.10.1 同构相关依赖

1. isomorphic-fetch：异步数据访问。

[8]赫门, 淘宝前后端分离实践, 2014

第六章　TEST

6.1　介绍

6.1.1　功能测试工具

1. puppeteer：Headless Chrome Node API。
2. selenium：A browser automation framework and ecosystem。

6.1.2　单元测试工具

1. Jasmine:Simple JavaScript testing framework for browsers and node.js。
2. mocha：simple, flexible, fun javascript test framework for node.js & the browser。
3. karma：Spectacular Test Runner for JavaScript。

6.1.3　扩展

1. Getting Started with Headless Chrome