

目錄

Introduction	1.1
CSS编码规范	1.2
HTML编码规范	1.3
HEAD	1.3.1
JavaScript编码规范	1.4
其他规范	1.5
开发工具的使用	1.6
Chrome Dev Tool使用教程	1.6.1
Github与git的使用	1.6.2
好书推荐篇	1.7
非编程类书籍推荐	1.7.1
免费的计算机编程类中文书籍	1.7.2
学习指导&资源篇	1.8
前端开发指南	1.8.1
Web Developer 成长路线图	1.8.2
前端StuQ技能图谱	1.8.3
前端资源汇总	1.8.4
JavaScript资源大全	1.8.5
前端优化	1.9
性能优化篇一	1.9.1
性能优化资源列表(en)	1.9.2
[其他干货]	1.10
HTTP API 设计指南	1.10.1
完美判断IE各版本	1.10.2

front-end-manual

前端手册

目录

1. [CSS编码规范](#)
2. [HTML编码规范](#)
3. [JavaScript编码规范](#)
4. [其他规范](#)
5. [开发工具的使用](#)
6. [好书推荐篇](#)
7. [学习指导篇](#)
8. [前端优化](#)
9. [其他](#)
10. [更多知识点见issues](#)

目录不全，请阅读[在线版](#)

下载离线版：[pdf](#)、[epub](#)、[mobi](#)

文档会不定时更新，可以 [watch](#) 或 [star](#) 本文档——[Github](#)，更欢迎您加入编辑维护

声明：编码规范参考[\[百度前端开发规范\]](#)修改，或整理于互联网，如有侵权请联系
giscafer@outlook.com。

整理人列表：[contributors](#)

欢迎关注微信公众号：[giscafer](#)



| giscafer.com · GitHub [@giscafer](https://github.com/giscafer) · Weibo [@Nickbing Lao](https://weibo.com/u/1311511713)

CSS编码规范

1、前言

CSS指层叠样式表 (Cascading Style Sheets)，是网页样式语言。任何网页都离不开CSS样式，作为前端开发工程师也必须要掌握的编程语言，使得在开发过程中，你能随意的更改样式和做页面UI调整，增强界面体验等。[CSS语言掌握情况测试](#)，[基础教程](#)

本编码规范目的是使开发人员编写的CSS代码风格保持一致，容易被理解和维护。

2、代码风格

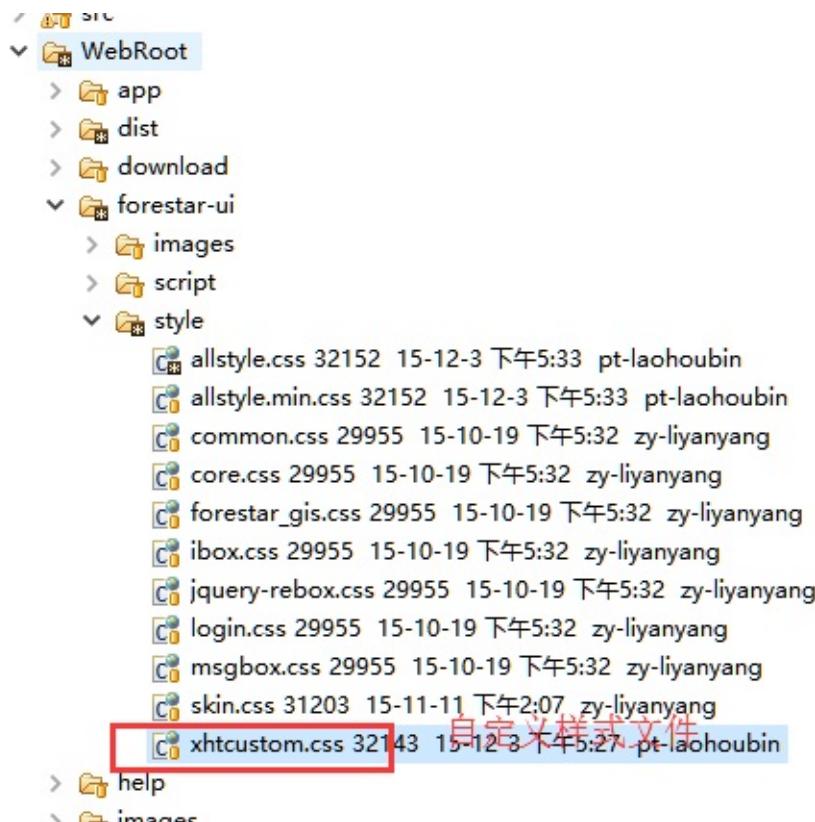
2.1 文件

[强制] 自定义CSS 文件 请单独创建文件

解释：单独创建文件有利于维护，同时方便ui的样式替换和升级。

[建议] CSS 文件使用无 BOM 的 UTF-8 编码

解释：UTF-8 编码具有更广泛的适应性。BOM在使用程序或工具处理文件时可能造成不必要的干扰。



2.2 缩进

[建议] 使用 4 个空格做为一个缩进层级，不允许使用 2 个空格或 tab 字符。示例：

```
.map {
    margin: 0;
    padding: 0;
}
```

2.3 空格

[强制] 选择器 与 { 之间必须包含空格。

示例：

```
.selector {
```

[强制] 属性名 与之后的 : 之间不允许包含空格， : 与 属性值 之间必须包含空格。

示例：

```
margin: 0;
```

[强制] 列表型属性值 书写在单行时，**,** 后必须跟一个空格。

示例：

```
font-family: Arial, sans-serif;
```

2.4 行长度

[强制] 每行不得超过 **120** 个字符，除非单行不可分割。

解释：

常见不可分割的场景为URL超长。

[建议] 对于超长的样式，在样式值的 空格 处或 **,** 后换行，建议按逻辑分组。

示例：

```
/* 不同属性值按逻辑分组 */
background:
    transparent url(aVeryVeryVeryLongUrlIsPlacedHere)
    no-repeat 0 0;

/* 可重复多次的属性，每次重复一行 */
background-image:
    url(aVeryVeryVeryLongUrlIsPlacedHere)
    url(anotherVeryVeryVeryLongUrlIsPlacedHere);

/* 类似函数的属性值可以根据函数调用的缩进进行 */
background-image: -webkit-gradient(
    linear,
    left bottom,
    left top,
    color-stop(0.04, rgb(88, 94, 124)),
    color-stop(0.52, rgb(115, 123, 162))
);
```

2.5 选择器

[强制] 当一个 **rule** 包含多个 **selector** 时，每个选择器声明必须独占一行。

示例：

```
/* good */
.post,
.page,
.comment {
    line-height: 1.5;
}

/* bad */
.post, .page, .comment {
    line-height: 1.5;
}
```

[强制] > 、 + 、 ~ 选择器的两边各保留一个空格。

示例：

```
/* good */
main > nav {
    padding: 10px;
}

label + input {
    margin-left: 5px;
}

input:checked ~ button {
    background-color: #69C;
}

/* bad */
main>nav {
    padding: 10px;
}

label+input {
    margin-left: 5px;
}

input:checked~button {
    background-color: #69C;
}
```

[强制] 属性选择器中的值必须用双引号包围。

解释：

不允许使用单引号，不允许不使用引号。

示例：

```
/* good */
article[character="juliet"] {
    voice-family: "Vivien Leigh", victoria, female;
}

/* bad */
article[character='juliet'] {
    voice-family: "Vivien Leigh", victoria, female;
}
```

2.6 属性

[强制] 属性定义必须另起一行。

示例：

```
/* good */
.selector {
    margin: 0;
    padding: 0;
}

/* bad */
.selector { margin: 0; padding: 0; }
```

[强制] 属性定义后必须以分号结尾。

示例：

```
/* good */
.selector {
    margin: 0;
}

/* bad */
.selector {
    margin: 0
}
```

说明：以上**2.2—2.6** 的要求可以通过**IDE美化CSS**达到。推荐 `sublime` 或者 `webstrom`

3 通用

3.1 选择器

[强制] 如无必要，不得为 **id** 、 **class** 选择器添加类型选择器进行限定。

解释：

在性能和维护性上，都有一定的影响。

示例：

```
/* good */
#error,
.danger-message {
    font-color: #c00;
}

/* bad */
dialog#error,
p.danger-message {
    font-color: #c00;
}
```

[建议] 选择器的嵌套层级应不大于 **3** 级，位置靠后的限定条件应尽可能精确。

示例：

```
/* good */
#username input {}
.comment .avatar {}

/* bad */
.page .header .login #username input {}
.comment div * {}
```

3.2 属性缩写

[建议] 在可以使用缩写的情况下，尽量使用属性缩写。

示例：

```

/* good */
.post {
    font: 12px/1.5 arial, sans-serif;
}

/* bad */
.post {
    font-family: arial, sans-serif;
    font-size: 12px;
    line-height: 1.5;
}

```

[建议] 使用 **border / margin / padding** 等缩写时，应注意隐含值对实际数值的影响，确实需要设置多个方向的值时才使用缩写。

解释：

border / margin / padding 等缩写会同时设置多个属性的值，容易覆盖不需要覆盖的设定。如某些方向需要继承其他声明的值，则应该分开设置。

示例：

```

/* centering <article class="page"> horizontally and highlight featured ones */
article {
    margin: 5px;
    border: 1px solid #999;
}

/* good */
.page {
    margin-right: auto;
    margin-left: auto;
}

.featured {
    border-color: #69c;
}

/* bad */
.page {
    margin: 5px auto; /* introducing redundancy */
}

.featured {
    border: 1px solid #69c; /* introducing redundancy */
}

```

3.3 属性书写顺序

[建议] 同一 **rule set** 下的属性在书写时，应按功能进行分组，并以 **Formatting Model**（布局方式、位置）>**Box Model**（尺寸）>**Typographic**（文本相关）>**Visual**（视觉效果）的顺序书写，以提高代码的可读性。

解释：

- **Formatting Model** 相关属性包括：`position / top / right / bottom / left / float / display / overflow` 等
- **Box Model** 相关属性包括：`border / margin / padding / width / height` 等
- **Typographic** 相关属性包括：`font / line-height / text-align / word-wrap` 等
- **Visual** 相关属性包括：`background / color / transition / list-style` 等

另外，如果包含 `content` 属性，应放在最前面。

示例：

```
.sidebar {
    /* formatting model: positioning schemes / offsets / z-indexes / display / ... */
    position: absolute;
    top: 50px;
    left: 0;
    overflow-x: hidden;

    /* box model: sizes / margins / paddings / borders / ... */
    width: 200px;
    padding: 5px;
    border: 1px solid #ddd;

    /* typographic: font / aligns / text styles / ... */
    font-size: 14px;
    line-height: 20px;

    /* visual: colors / shadows / gradients / ... */
    background: #f5f5f5;
    color: #333;
    -webkit-transition: color 1s;
    -moz-transition: color 1s;
    transition: color 1s;
}
```

3.4 清除浮动

[建议] 当元素需要撑起高度以包含内部的浮动元素时，通过对伪类设置 `clear` 或触发 `BFC` 的方式进行 `clearfix`。尽量不使用增加空标签的方式。

解释：

触发 `BFC` 的方式很多，常见的有：

- `float` 非 `none`
- `position` 非 `static`
- `overflow` 非 `visible`

如希望使用更小副作用的清除浮动方法，参见 [A new micro clearfix hack](#) 一文。

另需注意，对已经触发 `BFC` 的元素不需要再进行 `clearfix`。

3.5 !important

[建议] 尽量不使用 `!important` 声明。

[建议] 当需要强制指定样式且不允许任何场景覆盖时，通过标签内联和 `!important` 定义样式。

解释：

必须注意的是，仅在设计上 确实不允许任何其它场景覆盖样式 时，才使用内联的 `!important` 样式。通常在第三方环境的应用中使用这种方案。下面的 `z-index` 章节是其中一个特殊场景的典型样例。

3.6 z-index

[建议] 将 `z-index` 进行分层，对文档流外绝对定位元素的视觉层级关系进行管理。

解释：

同层的多个元素，如多个由用户输入触发的 `Dialog`，在该层级内使用相同的 `z-index` 或递增 `z-index`。

建议每层包含100个 `z-index` 来容纳足够的元素，如果每层元素较多，可以调整这个数值。

[建议] 在可控环境下，期望显示在最上层的元素，`z-index` 指定为 `999999`。

解释：

可控环境分成两种，一种是自身产品线环境；还有一种是可能会被其他产品线引用，但是不会被外部第三方的产品引用。

不建议取值为 `2147483647`。以便于自身产品线被其他产品线引用时，当遇到层级覆盖冲突的情况，留出向上调整的空间。

[建议] 在第三方环境下，期望显示在最上层的元素，通过标签内联和 `!important`，将 `z-index` 指定为 `2147483647`。

解释：

第三方环境对于开发者来说完全不可控。在第三方环境下的元素，为了保证元素不被其页面其他样式定义覆盖，需要采用此做法。

4 值与单位

4.1 文本

[强制] 文本内容必须用双引号包围。

解释：

文本类型的内容可能在选择器、属性值等内容中。

示例：

```

/* good */
html[lang|="zh"] q:before {
    font-family: "Microsoft YaHei", sans-serif;
    content: "〃";
}

html[lang|="zh"] q:after {
    font-family: "Microsoft YaHei", sans-serif;
    content: "〃";
}

/* bad */
html[lang|=zh] q:before {
    font-family: 'Microsoft YaHei', sans-serif;
    content: '〃';
}

html[lang|=zh] q:after {
    font-family: "Microsoft YaHei", sans-serif;
    content: "〃";
}

```

4.2 数值

[强制] 当数值为 **0 - 1** 之间的小时时，省略整数部分的 **0**。

示例：

```

/* good */
panel {
    opacity: .8;
}

/* bad */
panel {
    opacity: 0.8;
}

```

4.3 url()

[强制] **url()** 函数中的路径不加引号。

示例：

```
body {
    background: url(bg.png);
}
```

[建议] `url()` 函数中的绝对路径可省去协议名。

示例：

```
body {
    background: url(//baidu.com/img/bg.png) no-repeat 0 0;
}
```

4.4 长度

[强制] 长度为 `0` 时须省略单位。**(也只有长度单位可省)**

示例：

```
/* good */
body {
    padding: 0 5px;
}

/* bad */
body {
    padding: 0px 5px;
}
```

4.5 颜色

[强制] **RGB**颜色值必须使用十六进制记号形式 `#rrggbb`。不允许使用 `rgb()`。

解释：

带有alpha的颜色信息可以使用 `rgba()`。使用 `rgba()` 时每个逗号后必须保留一个空格。

示例：

```

/* good */
.success {
    box-shadow: 0 0 2px rgba(0, 128, 0, .3);
    border-color: #008000;
}

/* bad */
.success {
    box-shadow: 0 0 2px rgba(0,128,0,.3);
    border-color: rgb(0, 128, 0);
}

```

[强制] 颜色值可以缩写时，必须使用缩写形式。

示例：

```

/* good */
.success {
    background-color: #aca;
}

/* bad */
.success {
    background-color: #aaccaa;
}

```

[强制] 颜色值不允许使用命名色值。

示例：

```

/* good */
.success {
    color: #90ee90;
}

/* bad */
.success {
    color: lightgreen;
}

```

[建议] 颜色值中的英文字符采用小写。如不用小写也需要保证同
一项目内保持大小写一致。

示例：

```

/* good */
.success {
    background-color: #aca;
    color: #90ee90;
}

/* good */
.success {
    background-color: #ACA;
    color: #90EE90;
}

/* bad */
.success {
    background-color: #ACA;
    color: #90ee90;
}

```

4.6 2D 位置

[强制] 必须同时给出水平和垂直方向的位置。

解释：

2D 位置初始值为 `0% 0%`，但在只有一个方向的值时，另一个方向的值会被解析为 `center`。为了避免理解上的困扰，应同时给出两个方向的值。[background-position属性值的定义](#)

示例：

```

/* good */
body {
    background-position: center top; /* 50% 0% */
}

/* bad */
body {
    background-position: top; /* 50% 0% */
}

```

5 文本编排

5.1 字体族

[强制] **font-family** 属性中的字体族名称应使用字体的英文 **Family Name**，其中如有空格，须放置在引号中。

解释：

所谓英文 Family Name，为字体文件的一个元数据，常见名称如下：

字体	操作系统	Family Name
宋体(中易宋体)	Windows	SimSun
黑体(中易黑体)	Windows	SimHei
微软雅黑	Windows	Microsoft YaHei
微软雅黑	Windows	Microsoft JhengHei
华文黑体	Mac/iOS	STHeiti
冬青黑体	Mac/iOS	Hiragino Sans GB
文泉驿正黑	Linux	WenQuanYi Zen Hei
文泉驿微米黑	Linux	WenQuanYi Micro Hei

示例：

```
h1 {
    font-family: "Microsoft YaHei";
}
```

[强制] **font-family** 按「西文字体在前、中文字体在后」、「效果佳(质量高/更能满足需求)的字体在前、效果一般的字体在后」的顺序编写，最后必须指定一个通用字体族(**serif / sans-serif**)。

解释：

更详细说明可参考[本文](#)。

示例：

```

/* Display according to platform */
.article {
    font-family: Arial, sans-serif;
}

/* Specific for most platforms */
h1 {
    font-family: "Helvetica Neue", Arial, "Hiragino Sans GB", "WenQuanYi Micro Hei", "Microsoft YaHei", sans-serif;
}

```

[强制] `font-family` 不区分大小写，但在同一个项目中，同样的 `Family Name` 大小写必须统一。

示例：

```

/* good */
body {
    font-family: Arial, sans-serif;
}

h1 {
    font-family: Arial, "Microsoft YaHei", sans-serif;
}

/* bad */
body {
    font-family: arial, sans-serif;
}

h1 {
    font-family: Arial, "Microsoft YaHei", sans-serif;
}

```

5.2 字号

[强制] 需要在 **Windows** 平台显示的中文内容，其字号应不小于 **12px**。

解释：

由于 Windows 的字体渲染机制，小于 `12px` 的文字显示效果极差、难以辨认。

5.3 字体风格

[建议] 需要在 Windows 平台显示的中文内容，不要使用除 `normal` 外的 `font-style`。其他平台也应慎用。

解释：

由于中文字体没有 `italic` 风格的实现，所有浏览器下都会 `fallback` 到 `oblique` 实现（自动拟合为斜体），小字号下（特别是 Windows 下会在小字号下使用点阵字体的情况下）显示效果差，造成阅读困难。

5.4 字重

[强制] `font-weight` 属性必须使用数值方式描述。

解释：

CSS 的字重分 100 – 900 共九档，但目前受字体本身质量和浏览器的限制，实际上支持 `400` 和 `700` 两档，分别等价于关键词 `normal` 和 `bold`。

浏览器本身使用一系列[启发式规则](#)来进行匹配，在 `<700` 时一般匹配字体的 Regular 字重，`>=700` 时匹配 Bold 字重。

但已有浏览器开始支持 `=600` 时匹配 Semibold 字重（见[此表](#)），故使用数值描述增加了灵活性，也更简短。

示例：

```
/* good */
h1 {
    font-weight: 700;
}

/* bad */
h1 {
    font-weight: bold;
}
```

5.5 行高

[建议] `line-height` 在定义文本段落时，应使用数值。

解释：

将 `line-height` 设置为数值，浏览器会基于当前元素设置的 `font-size` 进行再次计算。在不同字号的文本段落组合中，能达到较为舒适的行间间隔效果，避免在每个设置了 `font-size` 都需要设置 `line-height`。

当 `line-height` 用于控制垂直居中时，还是应该设置成与容器高度一致。

示例：

```
.container {
    line-height: 1.5;
}
```

6 变换与动画

[强制] 使用 `transition` 时应指定 `transition-property`。

示例：

```
/* good */
.box {
    transition: color 1s, border-color 1s;
}

/* bad */
.box {
    transition: all 1s;
}
```

[建议] 尽可能在浏览器能高效实现的属性上添加过渡和动画。

解释：

见[本文](#)，在可能的情况下应选择这样四种变换：

- `transform: translate(npx, npx);`
- `transform: scale(n);`
- `transform: rotate(ndeg);`
- `opacity: 0..1;`

典型的，可以使用 `translate` 来代替 `left` 作为动画属性。

示例：

```
/* good */
.box {
    transition: transform 1s;
}
.box:hover {
    transform: translate(20px); /* move right for 20px */
}

/* bad */
.box {
    left: 0;
    transition: left 1s;
}
.box:hover {
    left: 20px; /* move right for 20px */
}
```

7 响应式

[强制] **Media Query** 不得单独编排，必须与相关的规则一起定义。

示例：

```

/* Good */
/* header styles */
@media (...) {
    /* header styles */
}

/* main styles */
@media (...) {
    /* main styles */
}

/* footer styles */
@media (...) {
    /* footer styles */
}

/* Bad */
/* header styles */
/* main styles */
/* footer styles */

@media (...) {
    /* header styles */
    /* main styles */
    /* footer styles */
}

```

[强制] Media Query 如果有多个逗号分隔的条件时，应将每个条件放在单独一行中。

示例：

```

@media
(-webkit-min-device-pixel-ratio: 2), /* Webkit-based browsers */
(min--moz-device-pixel-ratio: 2),      /* Older Firefox browsers (prior to Firefox 16) */
/
(min-resolution: 2dppx),              /* The standard way */
(min-resolution: 192dpi) {           /* dppx fallback */
    /* Retina-specific stuff here */
}

```

[建议] 尽可能给出在高分辨率设备 (**Retina**) 下效果更佳的样式。

8 兼容性

8.1 属性前缀

[强制] 带私有前缀的属性由长到短排列，按冒号位置对齐。

解释：

标准属性放在最后，按冒号对齐方便阅读，也便于在编辑器内进行多行编辑。

示例：

```
.box {  
    -webkit-box-sizing: border-box;  
    -moz-box-sizing: border-box;  
    box-sizing: border-box;  
}
```

8.2 Hack

[建议] 需要添加 **hack** 时应尽可能考虑是否可以采用其他方式解决。

解释：

如果能通过合理的 HTML 结构或使用其他的 CSS 定义达到理想的样式，则不应该使用 **hack** 手段解决问题。通常 **hack** 会导致维护成本的增加。

[建议] 尽量使用 **选择器 hack** 处理兼容性，而非 **属性 hack**。

解释：

尽量使用符合 CSS 语法的 **selector hack**，可以避免一些第三方库无法识别 **hack** 语法的问题。

示例：

```
/* IE 7 */
*:first-child + html #header {
    margin-top: 3px;
    padding: 5px;
}

/* IE 6 */
* html #header {
    margin-top: 5px;
    padding: 4px;
}
```

[建议] 尽量使用简单的 **属性 hack**。

示例：

```
.box {
    _display: inline; /* fix double margin */
    float: left;
    margin-left: 20px;
}

.container {
    overflow: hidden;
    *zoom: 1; /* triggering hasLayout */
}
```

8.3 Expression

[强制] 禁止使用**CSS表达式 Expression**。

解释：原因见前端优化篇章

HTML 编码规范

1 前言

HTML 是超文本标记语言。本文档的目标是使 HTML 代码风格保持一致，容易被理解和被维护。

[HTML 语言掌握情况测试](#)，[基础教程](#)

2 代码风格

2.1 缩进与换行

[强制] 使用 **4** 个空格做为一个缩进层级，不允许使用 **2** 个空格或 **tab** 字符。

示例：

```
<ul>
  <li>first</li>
  <li>second</li>
</ul>
```

[建议] 每行不得超过 **120** 个字符。

解释：

过长的代码不容易阅读与维护。但是考虑到 HTML 的特殊性，不做硬性要求。

2.2 命名

[强制] **class** 必须单词全字母小写，单词间以 **-** 分隔。

[强制] **class** 必须代表相应模块或部件的内容或功能，不得以样式信息进行命名。

示例：

```
<!-- good -->
<div class="sidebar"></div>

<!-- bad -->
<div class="left"></div>
```

[强制] 元素 **id** 必须保证页面唯一。

解释：

同一个页面中，不同的元素包含相同的 **id**，不符合 **id** 的属性含义。并且使用 `document.getElementById` 时可能导致难以追查的问题。

[建议] **id** 建议单词全字母小写，单词间以 **-** 分隔。同项目必须保持风格一致。**[建议]** **id**、**class** 命名，在避免冲突并描述清楚的前提下尽可能短。

示例：

```
<!-- good -->
<div id="nav"></div>
<!-- bad -->
<div id="navigation"></div>

<!-- good -->
<p class="comment"></p>
<!-- bad -->
<p class="com"></p>

<!-- good -->
<span class="author"></span>
<!-- bad -->
<span class="red"></span>
```

[强制] 禁止为了 **hook** 脚本，创建无样式信息的 **class**。

解释：

不允许 **class** 只用于让 JavaScript 选择某些元素，**class** 应该具有明确的语义和样式。否则容易导致 CSS **class** 泛滥。

使用 **id**、属性选择作为 **hook** 是更好的方式。

[强制] 同一页面，应避免使用相同的 `name` 与 `id`。

解释：

IE 浏览器会混淆元素的 `id` 和 `name` 属性，`document.getElementById` 可能获得不期望的元素。所以在对元素的 `id` 与 `name` 属性的命名需要非常小心。

一个比较好的实践是，为 `id` 和 `name` 使用不同的命名法。

示例：

```
<input name="foo">
<div id="foo"></div>
<script>
// IE6 将显示 INPUT
alert(document.getElementById('foo').tagName);
</script>
`
```

2.3 标签

[强制] 标签名必须使用小写字母。

示例：

```
<!-- good -->
<p>Hello StyleGuide!</p>

<!-- bad -->
<P>Hello StyleGuide!</P>
```

[强制] 对于无需自闭合的标签，不允许自闭合。

解释：

常见无需自闭合标签有 `input`、`br`、`img`、`hr` 等。

示例：

```
<!-- good -->
<input type="text" name="title">

<!-- bad -->
<input type="text" name="title" />
```

[强制] 对 **HTML5** 中规定允许省略的闭合标签，不允许省略闭合标签。

解释：

对代码体积要求非常严苛的场景，可以例外。比如：第三方页面使用的投放系统。

示例：

```
<!-- good -->
<ul>
  <li>first</li>
  <li>second</li>
</ul>

<!-- bad -->
<ul>
  <li>first
  <li>second
</ul>
```

[强制] 标签使用必须符合标签嵌套规则。

解释：

比如 `div` 不得置于 `p` 中，`tbody` 必须置于 `table` 中。

详细的标签嵌套规则参见[HTML DTD](#)中的 `Elements` 定义部分。

[建议] **HTML** 标签的使用应该遵循标签的语义。

解释：

下面是常见标签语义

- `p` - 段落
- `h1,h2,h3,h4,h5,h6` - 层级标题
- `strong,em` - 强调
- `ins` - 插入
- `del` - 删除
- `abbr` - 缩写
- `code` - 代码标识
- `cite` - 引述来源作品的标题
- `q` - 引用
- `blockquote` - 一段或长篇引用

- **ul** - 无序列表
- **ol** - 有序列表
- **dl,dt,dd** - 定义列表

示例：

```
<!-- good -->
<p>Esprima serves as an important <strong>building block</strong> for some JavaScript
language tools.</p>

<!-- bad -->
<div>Esprima serves as an important <span class="strong">building block</span> for som
e JavaScript language tools.</div>
```

[建议] 在 **CSS** 可以实现相同需求的情况下不得使用表格进行布局。

解释：

在兼容性允许的情况下应尽量保持语义正确性。对网格对齐和拉伸性有严格要求的场景允许例外，如多列复杂表单。

[建议] 标签的使用应尽量简洁，减少不必要的标签。

示例：

```
<!-- good -->


<!-- bad -->
<span class="avatar">
    
</span>
```

2.4 属性

[强制] 属性名必须使用小写字母。

示例：

```
<!-- good -->
<table cellspacing="0">...</table>

<!-- bad -->
<table cellSpacing="0">...</table>
```

【强制】属性值必须用双引号包围。

解释：

不允许使用单引号，不允许不使用引号。

示例：

```
<!-- good -->
<script src="esl.js"></script>

<!-- bad -->
<script src='esl.js'></script>
<script src=esl.js></script>
```

【建议】布尔类型的属性，建议不添加属性值。

示例：

```
<input type="text" disabled>
<input type="checkbox" value="1" checked>
```

【建议】自定义属性建议以 `xxx-` 为前缀，推荐使用 `data-`。

解释：

使用前缀有助于区分自定义属性和标准定义的属性。

示例：

```
<ol data-ui-type="Select"></ol>
```

3 通用

3.1 DOCTYPE

[强制] 使用 **HTML5** 的 **doctype** 来启用标准模式，建议使用大写的 **DOCTYPE**。

示例：

```
<!DOCTYPE html>
```

[建议] 启用 **IE Edge** 模式。

示例：

```
<meta http-equiv="X-UA-Compatible" content="IE=Edge">
```

[建议] 在 **html** 标签上设置正确的 **lang** 属性。

解释：

有助于提高页面的可访问性，如：让语音合成工具确定其所应该采用的发音，令翻译工具确定其翻译语言等。

示例：

```
<html lang="zh-CN">
```

3.2 编码

[强制] 页面必须使用精简形式，明确指定字符编码。指定字符编码的 **meta** 必须是 **head** 的第一个直接子元素。

解释：

见 [HTML5 Charset能用吗](#) 一文。

示例：

```

<html>
  <head>
    <meta charset="UTF-8">
    .....
  </head>
  <body>
    .....
  </body>
</html>

```

[建议] **HTML** 文件使用无 **BOM** 的 **UTF-8** 编码。

解释：

UTF-8 编码具有更广泛的适应性。**BOM** 在使用程序或工具处理文件时可能造成不必要的干扰。

3.3 CSS 和 JavaScript 引入

[强制] 引入 **css** 时必须指明 **rel="stylesheet"**。

示例：

```
<link rel="stylesheet" href="page.css">
```

[建议] 引入 **css** 和 **JavaScript** 时无须指明 **type** 属性。

解释：

text/css 和 **text/javascript** 是 **type** 的默认值。

[建议] 展现定义放置于外部 **css** 中，行为定义放置于外部 **JavaScript** 中。

解释：

结构-样式-行为的代码分离，对于提高代码的可阅读性和维护性都有好处。

[建议] 在 **head** 中引入页面需要的所有 **css** 资源。

解释：

在页面渲染的过程中，新的CSS可能导致元素的样式重新计算和绘制，页面闪烁。

[建议] **JavaScript** 应当放在页面末尾，或采用异步加载。

解释：

将 `script` 放在页面中间将阻断页面的渲染。出于性能方面的考虑，如非必要，请遵守此条建议。

示例：

```
<body>
    <!-- a lot of elements -->
    <script src="init-behavior.js"></script>
</body>
```

[建议] 移动环境或只针对现代浏览器设计的 **Web** 应用，如果引用外部资源的 **URL** 协议部分与页面相同，建议省略协议前缀。

解释：

使用 `protocol-relative URL` 引入 CSS，在 `IE7/8` 下，会发两次请求。是否使用 `protocol-relative URL` 应充分考虑页面针对的环境。

示例：

```
<script src="//s1.bdstatic.com/cache/static/jquery-1.10.2.min_f2fb5194.js"></script>
```

4 head

4.1 title

[强制] 页面必须包含 `title` 标签声明标题。

[强制] `title` 必须作为 `head` 的直接子元素，并紧随 `charset` 声明之后。

解释：

`title` 中如果包含 ASCII 之外的字符，浏览器需要知道字符编码类型才能进行解码，否则可能导致乱码。

示例：

```
<head>
  <meta charset="UTF-8">
  <title>页面标题</title>
</head>
```

4.2 favicon

[强制] 保证 **favicon** 可访问。

解释：

在未指定 favicon 时，大多数浏览器会请求 Web Server 根目录下的 `favicon.ico`。为了保证 favicon 可访问，避免 404，必须遵循以下两种方法之一：

1. 在 Web Server 根目录放置 `favicon.ico` 文件。
2. 使用 `link` 指定 favicon。

示例：

```
<link rel="shortcut icon" href="path/to/favicon.ico">
```

4.3 viewport

[建议] 若页面欲对移动设备友好，需指定页面的 **viewport**。

解释：

`viewport meta tag` 可以设置可视区域的宽度和初始缩放大小，避免在移动设备上出现页面展示不正常。

比如，在页面宽度小于 `980px` 时，若需 iOS 设备友好，应当设置 `viewport` 的 `width` 值来适应你的页面宽度。同时因为不同移动设备分辨率不同，在设置时，应当使用 `device-width` 和 `device-height` 变量。

另外，为了使 `viewport` 正常工作，在页面内容样式布局设计上也要做相应调整，如避免绝对定位等。关于 `viewport` 的更多介绍，可以参见 [Safari Web Content Guide](#) 的介绍

5 图片

[强制] 禁止 `img` 的 `src` 取值为空。延迟加载的图片也要增加默认的 `src`。

解释：

`src` 取值为空，会导致部分浏览器重新加载一次当前页面，参考：<https://developer.yahoo.com/performance/rules.html#emptysrc>

[建议] 避免为 `img` 添加不必要的 `title` 属性。

解释：

多余的 `title` 影响看图体验，并且增加了页面尺寸。

[建议] 为重要图片添加 `alt` 属性。

解释：

可以提高图片加载失败时的用户体验。

[建议] 添加 `width` 和 `height` 属性，以避免页面抖动。

[建议] 有下载需求的图片采用 `img` 标签实现，无下载需求的图片采用 **CSS** 背景图实现。

解释：

1. 产品 logo、用户头像、用户产生的图片等有潜在下载需求的图片，以 `img` 形式实现，能方便用户下载。
2. 无下载需求的图片，比如：icon、背景、代码使用的图片等，尽可能采用 CSS 背景图实现。

6 表单

6.1 控件标题

[强制] 有文本标题的控件必须使用 `label` 标签将其与其标题相联。

解释：

有两种方式：

1. 将控件置于 `label` 内。
2. `label` 的 `for` 属性指向控件的 `id`。

推荐使用第一种，减少不必要的 `id`。如果 DOM 结构不允许直接嵌套，则应使用第二种。

示例：

```
<label><input type="checkbox" name="confirm" value="on"> 我已确认上述条款</label>

<label for="username">用户名：</label> <input type="textbox" name="username" id="username">
```

6.2 按钮

[强制] 使用 `button` 元素时必须指明 `type` 属性值。

解释：

`button` 元素的默认 `type` 为 `submit`，如果被置于 `form` 元素中，点击后将导致表单提交。为显示区分其作用方便理解，必须给出 `type` 属性。

示例：

```
<button type="submit">提交</button>
<button type="button">取消</button>
```

[建议] 尽量不要使用按钮类元素的 `name` 属性。

解释：

由于浏览器兼容性问题，使用按钮的 `name` 属性会带来许多难以发现的问题。具体情况可参考[此文](#)。

6.3 可访问性 (A11Y)

[建议] 负责主要功能的按钮在 DOM 中的顺序应靠前。

解释：

负责主要功能的按钮应相对靠前，以提高可访问性。如果在 CSS 中指定了 `float: right` 则可能导致视觉上主按钮在前，而 DOM 中主按钮靠后的情况。

示例：

```

<!-- good -->
<style>
.buttons .button-group {
    float: right;
}
</style>

<div class="buttons">
    <div class="button-group">
        <button type="submit">提交</button>
        <button type="button">取消</button>
    </div>
</div>

<!-- bad -->
<style>
.buttons button {
    float: right;
}
</style>

<div class="buttons">
    <button type="button">取消</button>
    <button type="submit">提交</button>
</div>

```

[建议] 当使用 **JavaScript** 进行表单提交时，如果条件允许，应使原生提交功能正常工作。

解释：

当浏览器 JS 运行错误或关闭 JS 时，提交功能将无法工作。如果正确指定了 `form` 元素的 `action` 属性和表单控件的 `name` 属性时，提交仍可继续进行。

示例：

```

<form action="/login" method="post">
    <p><input name="username" type="text" placeholder="用户名"></p>
    <p><input name="password" type="password" placeholder="密码"></p>
</form>

```

[建议] 在针对移动设备开发的页面时，根据内容类型指定输入框的 `type` 属性。

解释：

根据内容类型指定输入框类型，能获得友好的输入体验。

示例：

```
<input type="date">
```

7 多媒体

[建议] 当在现代浏览器中使用 `audio` 以及 `video` 标签来播放音频、视频时，应当注意格式。

解释：

音频应尽可能覆盖到如下格式：

- MP3
- WAV
- Ogg

视频应尽可能覆盖到如下格式：

- MP4
- WebM
- Ogg

[建议] 在支持 `HTML5` 的浏览器中优先使用 `audio` 和 `video` 标签来定义音视频元素。

[建议] 使用退化到插件的方式来对多浏览器进行支持。

示例：

```
<audio controls>
  <source src="audio.mp3" type="audio/mpeg">
  <source src="audio.ogg" type="audio/ogg">
  <object width="100" height="50" data="audio.mp3">
    <embed width="100" height="50" src="audio.swf">
  </object>
</audio>

<video width="100" height="50" controls>
  <source src="video.mp4" type="video/mp4">
  <source src="video.ogg" type="video/ogg">
  <object width="100" height="50" data="video.mp4">
    <embed width="100" height="50" src="video.swf">
  </object>
</video>
```

[建议] 只在必要的时候开启音视频的自动播放。

[建议] 在 **object** 标签内部提供指示浏览器不支持该标签的说明。

示例：

```
<object width="100" height="50" data="something.swf">DO NOT SUPPORT THIS TAG</object>
```

8 模板中的 HTML

比如FreeMarker中的HTML模板

[建议] 模板代码的缩进优先保证 **HTML** 代码的缩进规则。

示例：

```

<!-- good -->
{if $display == true}
<div>
    <ul>
        {foreach $item_list as $item}
            <li>{$item.name}</li>
        {/foreach}
    </ul>
</div>
{/if}

<!-- bad -->
{if $display == true}
    <div>
        <ul>
            {foreach $item_list as $item}
                <li>{$item.name}</li>
            {/foreach}
        </ul>
    </div>
{/if}

```

[建议] 模板代码应以保证 **HTML** 单个标签语法的正确性为基本原则。

示例：

```

<!-- good -->
<li class="{if $item.type_id == $current_type}focus{/if}">{ $item.type_name }</li>

<!-- bad -->
<li {if $item.type_id == $current_type} class="focus"{/if}>{ $item.type_name }</li>

```

[建议] 在循环处理模板数据构造表格时，若要求每行输出固定的个数，建议先将数据分组，之后再循环输出。

示例：

```
<!-- good -->





```

HEAD

A collection of HTML head elements.

Recommended Minimum

Below are the essential tags for basic, minimalist websites:

```
<meta charset="utf-8">
<meta http-equiv="x-ua-compatible" content="ie=edge">
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>Page Title</title>
```

Elements

```
<title>Page Title</title>
<!-- The <base> tag defines a base URL for all relative links in the document -->
<base href="https://example.com/page.html">
<style>
  body { color: red; }
</style>
<script src="script.js"></script>
```

Meta Element

```
<meta charset="utf-8">
<meta http-equiv="x-ua-compatible" content="ie=edge">
<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
<!-- The above 3 meta tags *must* come first in the head; any other head content must
come *after* these tags --&gt;
&lt;meta name="application-name" content="Application Name"&gt;
&lt;meta name="keywords" content="your, keywords, here, comma, separated, no, spaces"&gt;
&lt;meta name="description" content="150 chars"&gt;
&lt;meta name="subject" content="your website's subject"&gt;
&lt;meta name="robots" content="index,follow,noodp"&gt;
&lt;meta name="googlebot" content="index,follow"&gt;
&lt;meta name="google" content="nositelinkssearchbox"&gt;
&lt;meta name="google-site-verification" content="verification_token"&gt;
&lt;meta name="abstract" content=""&gt;
&lt;meta name="topic" content=""&gt;
&lt;meta name="summary" content=""&gt;
&lt;meta name="classification" content="business"&gt;
&lt;meta name="url" content="https://example.com/"&gt;
&lt;meta name="identifier-URL" content="https://example.com/"&gt;
&lt;meta name="directory" content="submission"&gt;
&lt;meta name="category" content=""&gt;
&lt;meta name="coverage" content="Worldwide"&gt;
&lt;meta name="distribution" content="Global"&gt;
&lt;meta name="rating" content="General"&gt;
&lt;meta name="referrer" content="never"&gt;
&lt;meta http-equiv="Content-Security-Policy" content="default-src 'self'"&gt;</pre>
```

Not Recommended

Below are the meta attributes which are not recommended for use:

```
<!-- Used to declare the document language, but not well supported. Better to use <htm  
l lang="" -->  
<meta name="language" content="en">  
  
<!-- No evidence of current use in any search engines -->  
<meta name="revised" content="Sunday, July 18th, 2010, 5:15 pm">  
  
<!-- Provides an easy way for spam bots to harvest email addresses -->  
<meta name="reply-to" content="email@example.com">  
  
<!-- Better to use <link rel="author"> or humans.txt file -->  
<meta name="author" content="name, email@example.com">  
<meta name="designer" content="">  
<meta name="owner" content="">  
  
<!-- Tells search bots to revisit the page after a period. This is not supported because most Search Engines now use random intervals for re-crawling a webpage -->  
<meta name="revisit-after" content="7 days">  
  
<!-- Google strongly advises not to use this. Better to set up server-side (e.g. Apache, nginx) redirects instead -->  
<meta http-equiv="refresh" content="300;url=https://example.com/">  
  
<!-- Cache Control -->  
<!-- Better to configure cache control server side -->  
<meta http-equiv="Expires" content="0">  
<meta http-equiv="Pragma" content="no-cache">  
<meta http-equiv="Cache-Control" content="no-cache">
```

Link Element

```

<link rel="copyright" href="copyright.html">
<link rel="stylesheet" href="https://example.com/styles.css">
<link rel="alternate" href="https://feeds.feedburner.com/martini" type="application/rss+xml" title="RSS">
<link rel="alternate" href="https://example.com/feed.atom" type="application/atom+xml" title="Atom 0.3">
<link rel="alternate" href="https://es.example.com/" hreflang="es">
<link rel="me" href="https://google.com/profiles/thenextweb" type="text/html">
<link rel="me" href="mailto:name@example.com">
<link rel="me" href="sms:+15035550125">
<link rel="archives" href="https://example.com/2003/05/" title="May 2003">
<link rel="index" href="https://example.com/" title="DeWitt Clinton">
<link rel="start" href="https://example.com/photos/pattern_recognition_1_about/" title="Pattern Recognition 1">
<link rel="prev" href="https://example.com/opensearch/opensearch-and-openid-a-sure-way-to-get-my-attention/" title="OpenSearch and OpenID? A sure way to get my attention.">
<link rel="search" href="/open-search.xml" type="application/opensearchdescription+xml" title="Search Title">
<link rel="self" type="application/atom+xml" href="https://example.com/atomFeed.php?page=3">
<link rel="first" href="https://example.com/atomFeed.php">
<link rel="next" href="https://example.com/atomFeed.php?page=4">
<link rel="previous" href="https://example.com/atomFeed.php?page=2">
<link rel="last" href="https://example.com/atomFeed.php?page=147">
<link rel="shortlink" href="https://example.com/?p=43625">
<link rel="canonical" href="https://example.com/2010/06/9-things-to-do-before-entering-social-media.html">
<link rel="amphtml" href="https://www.example.com/url/to/amp-version.html">
<link rel="EditURI" href="https://example.com/xmlrpc.php?rsd" type="application/rsd+xml" title="RSD">
<link rel="pingback" href="https://example.com/xmlrpc.php">
<link rel="webmention" href="https://example.com/webmention">
<link rel="manifest" href="manifest.json">
<link rel="author" href="humans.txt">
<link rel="import" href="component.html">

<!-- Prefetching, preloading, prebrowsing -->
<link rel="dns-prefetch" href="//example.com/">
<link rel="preconnect" href="https://www.example.com/">
<link rel="prefetch" href="https://www.example.com/">
<link rel="prerender" href="https://example.com/">
<link rel="subresource" href="styles.css">
<link rel="preload" href="image.png">
<!-- More info: https://css-tricks.com/prefetching-preloading-prebrowsing/ -->

```

Not Recommended

Below are the link relations which are not recommended for use:

```
<link rel="shortcut icon" href="path/to/favicon.ico">
```

Favicons

```
<!-- For IE 10 and below -->
<!-- No link, just place a file called favicon.ico in the root directory -->

<!-- For IE 11, Chrome, Firefox, Safari, Opera -->
<link rel="icon" href="path/to/favicon-16.png" sizes="16x16" type="image/png">
<link rel="icon" href="path/to/favicon-32.png" sizes="32x32" type="image/png">
<link rel="icon" href="path/to/favicon-48.png" sizes="48x48" type="image/png">
<link rel="icon" href="path/to/favicon-62.png" sizes="62x62" type="image/png">
<!-- More info: https://bitsofco.de/all-about-favicons-and-touch-icons/ -->
```

- All About Favicons (And Touch Icons)

Social

OEmbed

```
<link rel="alternate" type="application/json+oembed"
      href="http://example.com/services/oembed?url=http%3A%2F%2Fexample.com%2Ffoo%2F&format=json"
      title="oEmbed Profile: JSON">
<link rel="alternate" type="text/xml+oembed"
      href="http://example.com/services/oembed?url=http%3A%2F%2Fexample.com%2Ffoo%2F&format=xml"
      title="oEmbed Profile: XML">
```

- oEmbed format

Facebook / Open Graph

```

<meta property="fb:app_id" content="123456789">
<meta property="og:url" content="https://example.com/page.html">
<meta property="og:type" content="website">
<meta property="og:title" content="Content Title">
<meta property="og:image" content="https://example.com/image.jpg">
<meta property="og:description" content="Description Here">
<meta property="og:site_name" content="Site Name">
<meta property="og:locale" content="en_US">
<meta property="article:author" content="">
<!-- Facebook: https://developers.facebook.com/docs/sharing/webmasters#markup -->
<!-- Open Graph: http://ogp.me/ -->

```

- Facebook Open Graph Markup
- Open Graph protocol

Facebook / Instant Articles

```

<meta charset="utf-8">
<meta property="op:markup_version" content="v1.0">

<!-- The URL of the web version of your article -->
<link rel="canonical" href="http://example.com/article.html">

<!-- The style to be used for this article -->
<meta property="fb:article_style" content="myarticlestyle">

```

- Facebook Instant Articles: Creating Articles
- Instant Articles: Format Reference

Twitter

```

<meta name="twitter:card" content="summary">
<meta name="twitter:site" content="@site_account">
<meta name="twitter:creator" content="@individual_account">
<meta name="twitter:url" content="https://example.com/page.html">
<meta name="twitter:title" content="Content Title">
<meta name="twitter:description" content="Content description less than 200 characters">
<meta name="twitter:image" content="https://example.com/image.jpg">
<!-- More info: https://dev.twitter.com/cards/getting-started -->
<!-- Validate: https://dev.twitter.com/docs/cards/validation/validator -->

```

- Twitter Cards: Getting Started Guide
- Twitter Card Validator

Google+ / Schema.org

```
<link href="https://plus.google.com/+YourPage" rel="publisher">
<meta itemprop="name" content="Content Title">
<meta itemprop="description" content="Content description less than 200 characters">
<meta itemprop="image" content="https://example.com/image.jpg">
```

Browser/Platform

Apple iOS

```
<!-- Smart App Banner -->
<meta name="apple-itunes-app" content="app-id=APP_ID,affiliate-data=AFFILIATE_ID,app-a
rgument=SOME_TEXT">

<!-- Disable automatic detection and formatting of possible phone numbers -->
<meta name="format-detection" content="telephone=no">

<!-- Add to Home Screen -->
<meta name="apple-mobile-web-app-capable" content="yes">
<meta name="apple-mobile-web-app-status-bar-style" content="black">
<meta name="apple-mobile-web-app-title" content="App Title">

<!-- Touch Icons -->
<link rel="apple-touch-icon" href="apple-touch-icon.png">
<link rel="apple-touch-icon-precomposed" href="apple-touch-icon-precomposed.png">
<!-- In most cases, one 180x180px touch icon in the head is enough -->
<!-- If you use art-direction and/or want to have different content for each device, y
ou can add more touch icons -->

<!-- Startup Image -->
<link rel="apple-touch-startup-image" href="startup.png">

<!-- More info: https://developer.apple.com/safari/library/documentation/appleapplications/reference/safarihtmlref/articles/metatags.html -->
```

- Apple Meta Tags

Apple Safari

```
<!-- Pinned Site -->
<link rel="mask-icon" href="icon.svg" color="red">
```

Google Android

```
<meta name="theme-color" content="#E64545">

<!-- Add to homescreen -->
<meta name="mobile-web-app-capable" content="yes">
<!-- More info: https://developer.chrome.com/multidevice/android/installtohomescreen --
->
```

Google Chrome

```
<link rel="chrome-webstore-item" href="https://chrome.google.com/webstore/detail/APP_I
D">

<!-- Disable translation prompt -->
<meta name="google" value="notranslate">
```

Microsoft Internet Explorer

```
<meta http-equiv="x-ua-compatible" content="ie=edge">
<meta http-equiv="cleartype" content="on">
<meta name="skype_toolbar" content="skype_toolbar_parser_compatible">

<!-- Disable link highlighting on IE 10 on Windows Phone (https://blogs.windows.com/bu
ildingapps/2012/11/15/adapting-your-webkit-optimized-site-for-internet-explorer-10/) -
->
<meta name="msapplication-tap-highlight" content="no">

<!-- Pinned sites (https://msdn.microsoft.com/en-us/library/dn255024(v=vs.85).aspx) -->

<meta name="application-name" content="Contoso Pinned Site Caption">
<meta name="msapplication-tooltip" content="Example Tooltip Text">
<meta name="msapplication-starturl" content="/">

<meta name="msapplication-config" content="http://example.com/browserconfig.xml">

<meta name="msapplication-allowDomainApiCalls" content="true">
<meta name="msapplication-allowDomainMetaTags" content="true">
<meta name="msapplication-badge" content="frequency=30; polling-uri=http://example.com
/id45453245/polling.xml">
<meta name="msapplication-navbutton-color" content="#FF3300">
<meta name="msapplication-notification" content="frequency=60; polling-uri=http://examp
le.com/livetile">
<meta name="msapplication-square150x150logo" content="images/logo.png">
<meta name="msapplication-square310x310logo" content="images/largelogo.png">
<meta name="msapplication-square70x70logo" content="images/tinylogo.png">
<meta name="msapplication-wide310x150logo" content="images/widelogo.png">
<meta name="msapplication-task" content="name=Check Order Status;action-uri=../orderSta
tus.aspx?src=IE9;icon-uri=../favicon.ico">
<meta name="msapplication-task-seperator" content="1">
<meta name="msapplication-TileColor" content="#FF3300">
<meta name="msapplication-TileImage" content="images/tileimage.jpg">
<meta name="msapplication-window" content="width=1024;height=768">
```

Microsoft Internet Explorer (LEGACY DO NOT USE)

```
<!-- Disable the image toolbar when you mouse over images in IE 6 (https://msdn.microsoft.com/en-us/library/ms532986(v=vs.85).aspx) -->
<meta http-equiv="imagetoolbar" content="no">

<!-- Disable Windows theming to form inputs/buttons (https://support.microsoft.com/en-us/kb/322240) -->
<meta name="MSThemeCompatible" content="no">

<!-- Disable a feature that only appeared on IE 6 beta (https://stackoverflow.com/q/2167301) -->
<meta name="MSSmartTagsPreventParsing" content="true">

<!-- Interpage Transitions (https://msdn.microsoft.com/en-us/library/ms532847(v=vs.85).aspx) -->
<meta http-equiv="Page-Enter" content="revealtrans(duration=2,transition=2)">
<meta http-equiv="Page-Exit" content="revealtrans(duration=3,transition=12)">
<meta http-equiv="Site-Enter" content="revealtrans(duration=2,transition=2)">
<meta http-equiv="Site-Exit" content="revealtrans(duration=3,transition=12)">
```

360 Browser

```
<!-- select rendering engine in order -->
<meta name="renderer" content="webkit|ie-comp|ie-stand">
```

UC Mobile Browser

```
<!-- Locks the screen into the specified orientation -->
<meta name="screen-orientation" content="landscape/portrait">
<!-- Display this page in fullscreen -->
<meta name="full-screen" content="yes">
<!-- UC browser will display images even if in "text mode" -->
<meta name="imagemode" content="force">
<!-- Page will be displayed in "application mode"(fullscreen,forbidding gesture, etc.) -->
<meta name="browsermode" content="application">
<!-- Disabled the UC browser's "night mode" in this page -->
<meta name="nightmode" content="disable">
<!-- Simplify the page to reduce data transfer -->
<meta name="layoutmode" content="fitscreen">
<!-- Disable the UC browser's feature of "scaling font up when there are many words in this page" -->
<meta name="wap-font-scale" content="no">
```

- UC Browser Docs

QQ Mobile Browser

```
<!-- Locks the screen into the specified orientation -->
<meta name="x5-orientation" content="landscape/portrait">
<!-- Display this page in fullscreen -->
<meta name="x5-fullscreen" content="true">
<!-- Page will be displayed in "application mode"(fullscreen, etc.) -->
<meta name="x5-page-mode" content="app">
```

App Links

```
<!-- iOS -->
<meta property="al:ios:url" content="applinks://docs">
<meta property="al:ios:app_store_id" content="12345">
<meta property="al:ios:app_name" content="App Links">
<!-- Android -->
<meta property="al:android:url" content="applinks://docs">
<meta property="al:android:app_name" content="App Links">
<meta property="al:android:package" content="org.applinks">
<!-- Web Fallback -->
<meta property="al:web:url" content="http://applinks.org/documentation">
<!-- More info: http://applinks.org/documentation/ -->
```

- [App Links Docs](#)

Other Resources

- [HTML5 Boilerplate Docs: The HTML](#)
- [HTML5 Boilerplate Docs: Extend and customize](#)

Contributing

Open an issue or a pull request to suggest changes or additions.

Author

[Josh Buchea](#)

License

[MIT License](#)

JavaScript编码规范

1 前言

JavaScript 是最火的前端脚本语言（近些年因为 CommonJS 规范的完善和 NodeJS 的诞生，JavaScript 使用了包和模块管理的方式，走前端开发方向的请自行去了解）。本文档的目标是使 JavaScript 代码风格保持一致，容易被理解和被维护。

虽然本文档是针对 JavaScript 设计的，但是在使用各种 JavaScript 的预编译语言时(如 TypeScript 等)时，适用的部分也应尽量遵循本文档的约定。

JavaScript 语言掌握情况[测试](#)，基础[教程](#)

2 代码风格

2.1 文件

[建议] JavaScript 文件使用无 `BOM` 的 `UTF-8` 编码。

解释：

UTF-8 编码具有更广泛的适应性。BOM 在使用程序或工具处理文件时可能造成不必要的干扰。

[建议] 在文件结尾处，保留一个空行。

2.2 结构

2.2.1 缩进

[强制] 使用 `4` 个空格做为一个缩进层级，不允许使用 `2` 个空格或 `tab` 字符。

[强制] `switch` 下的 `case` 和 `default` 必须增加一个缩进层级。

示例：

```
// good
switch (variable) {

    case '1':
        // do...
        break;

    case '2':
        // do...
        break;

    default:
        // do...

}

// bad
switch (variable) {

    case '1':
        // do...
        break;

    case '2':
        // do...
        break;

    default:
        // do...

}
```

2.2.2 空格

[强制] 二元运算符两侧必须有一个空格，一元运算符与操作对象之间不允许有空格。

示例：

```
var a = !arr.length;
a++;
a = b + c;
```

[强制] 用作代码块起始的左花括号 { 前必须有一个空格。

示例：

```
// good
if (condition) {
}

while (condition) {
}

function funcName() {
}

// bad
if (condition){
}

while (condition){
}

function funcName(){
}
```

[强制] `if / else / for / while / function / switch / do / try / catch / finally` 关键字后，必须有一个空格。

示例：

```
// good
if (condition) {
}

while (condition) {
}

(function () {
})();

// bad
if(condition) {
}

while(condition) {
}

(function() {
})();
```

[强制] 在对象创建时，属性中的 `:` 之后必须有空格，`:` 之前不允许有空格。

示例：

```
// good
var obj = {
  a: 1,
  b: 2,
  c: 3
};

// bad
var obj = {
  a : 1,
  b:2,
  c :3
};
```

【强制】 函数声明、具名函数表达式、函数调用中，函数名和 `(` 之间不允许有空格。

示例：

```
// good
function funcName() {
}

var funcName = function funcName() {
};

funcName();

// bad
function funcName () {
}

var funcName = function funcName () {
};

funcName ();
```

【强制】 `,` 和 `;` 前不允许有空格。如果不位于行尾，`,` 和 `;` 后必须跟一个空格。

示例：

```
// good
callFunc(a, b);

// bad
callFunc(a , b) ;
```

【强制】 在函数调用、函数声明、括号表达式、属性访问、`if / for / while / switch / catch` 等语句中，`()` 和 `[]` 内紧贴括号部分不允许有空格。

示例：

```
// good

callFunc(param1, param2, param3);

save(this.list[this.indexes[i]]);

needIncream && (variable += increment);

if (num > list.length) {
}

while (len--) {

}

// bad

callFunc( param1, param2, param3 );

save( this.list[ this.indexes[ i ] ] );

needInrement && ( variable += increment );

if ( num > list.length ) {

}

while ( len-- ) {
```

【强制】单行声明的数组与对象，如果包含元素，`{}` 和 `[]` 内紧贴括号部分不允许包含空格。

解释：

声明包含元素的数组与对象，只有当内部元素的形式较为简单时，才允许写在一行。元素复杂的情况，还是应该换行书写。

示例：

```
// good
var arr1 = [];
var arr2 = [1, 2, 3];
var obj1 = {};
var obj2 = {name: 'obj'};
var obj3 = {
    name: 'obj',
    age: 20,
    sex: 1
};

// bad
var arr1 = [ ];
var arr2 = [ 1, 2, 3 ];
var obj1 = { };
var obj2 = { name: 'obj' };
var obj3 = {name: 'obj', age: 20, sex: 1};
```

【强制】行尾不得有多余的空格。

2.2.3 换行

【强制】每个独立语句结束后必须换行。

【强制】每行不得超过 120 个字符。

解释：

超长的不可分割的代码允许例外，比如复杂的正则表达式。长字符串不在例外之列。

【强制】运算符处换行时，运算符必须在新行的行首。

示例：

```
// good
if (user.isAuthenticated()
    && user isInRole('admin')
    && user.hasAuthority('add-admin')
    || user.hasAuthority('delete-admin'))
) {
    // Code
}

var result = number1 + number2 + number3
+ number4 + number5;

// bad
if (user.isAuthenticated() &&
    user.isInRole('admin') &&
    user.hasAuthority('add-admin') |||
    user.hasAuthority('delete-admin')) {
    // Code
}

var result = number1 + number2 + number3 +
number4 + number5;
```

【强制】在函数声明、函数表达式、函数调用、对象创建、数组创建、`for`语句等场景中，不允许在`,`或`;`前换行。

示例：

```
// good
var obj = {
  a: 1,
  b: 2,
  c: 3
};

foo(
  aVeryVeryLongArgument,
  anotherVeryLongArgument,
  callback
);

// bad
var obj = {
  a: 1
  , b: 2
  , c: 3
};

foo(
  aVeryVeryLongArgument
  , anotherVeryLongArgument
  , callback
);
```

【建议】 不同行为或逻辑的语句集，使用空行隔开，更易阅读。

示例：

```
// 仅为按逻辑换行的示例，不代表setStyle的最优实现
function setStyle(element, property, value) {
  if (element == null) {
    return;
  }

  element.style[property] = value;
}
```

【建议】 在语句的行长度超过 **120** 时，根据逻辑条件合理缩进。

示例：

```
// 较复杂的逻辑条件组合，将每个条件独立一行，逻辑运算符放置在行首进行分隔，或将部分逻辑按逻辑组合进行分隔。
// 建议最终将右括号 ) 与左大括号 { 放在独立一行，保证与 `if` 内语句块能容易视觉辨识。
if (user.isAuthenticated()
  && user isInRole('admin'))
```

```

    && user.hasAuthority('add-admin')
    || user.hasAuthority('delete-admin')
) {
    // Code
}

// 按一定长度截断字符串，并使用 + 运算符进行连接。
// 分隔字符串尽量按语义进行，如不要在一个完整的名词中间断开。
// 特别的，对于 HTML 片段的拼接，通过缩进，保持和 HTML 相同的结构。
var html = '' // 此处用一个空字符串，以便整个 HTML 片段都在新行严格对齐
+ '<article>'
+   '<h1>Title here</h1>'
+   '<p>This is a paragraph</p>'
+   '<footer>Complete</footer>'
+ '</article>';

// 也可使用数组来进行拼接，相对 `+` 更容易调整缩进。
var html = [
    '<article>',
    '<h1>Title here</h1>',
    '<p>This is a paragraph</p>',
    '<footer>Complete</footer>',
    '</article>'
];
html = html.join('');

// 当参数过多时，将每个参数独立写在一行上，并将结束的右括号 ) 独立一行。
// 所有参数必须增加一个缩进。
foo(
    aVeryVeryLongArgument,
    anotherVeryLongArgument,
    callback
);

// 也可以按逻辑对参数进行组合。
// 最经典的是 baidu.format 函数，调用时将参数分为“模板”和“数据”两块
baidu.format(
    dateFormatTemplate,
    year, month, date, hour, minute, second
);

// 当函数调用时，如果有 1 个或以上参数跨过多行，应当每一个参数独立一行。
// 这通常出现在匿名函数或者对象初始化等作为参数时，如 `setTimeout` 函数等。
setTimeout(
    function () {
        alert('hello');
    },
    200
);

order.data.read(
    'id=' + me.model.id,
    function (data) {

```

```

        me.attchToModel(data.result);
        callback();
    },
    300
);

// 链式调用较长时采用缩进进行调整。
$('#items')
    .find('.selected')
    .highlight()
    .end();

// 三元运算符由3部分组成，因此其换行应当根据每个部分的长度不同，形成不同的情况。
var result = thisIsAVeryVeryLongCondition
    ? resultA : resultB;

var result = condition
    ? thisIsAVeryVeryLongResult
    : resultB;

// 数组和对象初始化的混用，严格按照每个对象的 `{' 和结束 `}` 在独立一行的风格书写。
var array = [
{
    // ...
},
{
    // ...
}
];

```

[建议] 对于 `if...else...`、`try...catch...finally` 等语句，推荐使用在 `}` 号后添加一个换行的风格，使代码层次结构更清晰，阅读性更好。

示例：

```

if (condition) {
    // some statements;
}

else {
    // some statements;
}

try {
    // some statements;
}
catch (ex) {
    // some statements;
}

```

2.2.4 语句

【强制】不得省略语句结束的分号。

【强制】在 `if / else / for / do / while` 语句中，即使只有一行，也不得省略块 `{...}`。

示例：

```
// good
if (condition) {
    callFunc();
}

// bad
if (condition) callFunc();
if (condition)
    callFunc();
```

【强制】函数定义结束不允许添加分号。

示例：

```
// good
function funcName() {
}

// bad
function funcName() {
};

// 如果是函数表达式，分号是不允许省略的。
var funcName = function () {
};
```

【强制】 `IIFE` 必须在函数表达式外添加 `(`，非 `IIFE` 不得在函数表达式外添加 `(`。

解释：

`IIFE = Immediately-Invoked Function Expression.`

额外的 `(` 能够让代码在阅读的一开始就能判断函数是否立即被调用，进而明白接下来代码的用途。而不是一直拖到底部才恍然大悟。

示例：

```
// good
var task = (function () {
    // Code
    return result;
})();

var func = function () {
};

// bad
var task = function () {
    // Code
    return result;
}();

var func = (function () {
});
```

2.3 命名

【强制】 变量 使用 **Camel**命名法 。

示例：

```
var loadingModules = {};
```

【强制】 常量 使用 全部字母大写，单词间下划线分隔 的命名方式。

示例：

```
var HTML_ENTITY = {};
```

【强制】 函数 使用 **Camel**命名法 。

示例：

```
function stringFormat(source) {
}
```

【强制】 函数的 参数 使用 **Camel**命名法 。

示例：

```
function hear(theBells) {
}
```

【强制】 类 使用 **Pascal**命名法。

示例：

```
function TextNode(options) {
}
```

【强制】 类的 方法 / 属性 使用 **Camel**命名法。

示例：

```
function TextNode(value, engine) {
    this.value = value;
    this.engine = engine;
}

TextNode.prototype.clone = function () {
    return this;
};
```

【强制】 枚举变量 使用 **Pascal**命名法，枚举的属性 使用 全部字母大写，单词间下划线分隔 的命名方式。

示例：

```
var TargetState = {
    READING: 1,
    READED: 2,
    APPLIED: 3,
    READY: 4
};
```

【强制】 命名空间 使用 **Camel**命名法。

示例：

```
equipments.heavyWeapons = {};
```

【强制】 由多个单词组成的缩写词，在命名中，根据当前命名法和出现的位置，所有字母的大小写与首字母的大小写保持一致。

示例：

```

function XMLParser() {
}

function insertHTML(element, html) {
}

var httpRequest = new XMLHttpRequest();

```

[强制] 类名 使用 名词。

示例：

```

function Engine(options) {
}

```

[建议] 函数名 使用 动宾短语。

示例：

```

function getStyle(element) {
}

```

[建议] boolean 类型的变量使用 is 或 has 开头。

示例：

```

var isReady = false;
var hasMoreCommands = false;

```

[建议] Promise对象 用 动宾短语的进行时 表达。

示例：

```

var loadingData = ajax.get('url');
loadingData.then(callback);

```

2.4 注释

2.4.1 单行注释

[强制] 必须独占一行。 // 后跟一个空格，缩进与下一行被注释说明的代码一致。

2.4.2 多行注释

[建议] 避免使用 `/*...*/` 这样的多行注释。有多行注释内容时，使用多个单行注释。

2.4.3 文档化注释

[强制] 为了便于代码阅读和自文档化，以下内容必须包含以 `/**...*/` 形式的块注释中。

解释：

1. 文件
2. namespace
3. 类
4. 函数或方法
5. 类属性
6. 事件
7. 全局变量
8. 常量
9. AMD 模块

[强制] 文档注释前必须空一行。

[建议] 自文档化的文档说明 **what**，而不是 **how**。

2.4.4 类型定义

[强制] 类型定义都是以 `{` 开始，以 `}` 结束。

解释：

常用类型如：`{string}`, `{number}`, `{boolean}`, `{Object}`, `{Function}`, `{RegExp}`, `{Array}`, `{Date}`。

类型不仅局限于内置的类型，也可以是自定义的类型。比如定义了一个类 `Developer`，就可以使用它来定义一个参数和返回值的类型。

[强制] 对于基本类型 `{string}`, `{number}`, `{boolean}`，首字母必须小写。

类型定义	语法示例	解释
String	{string}	--
Number	{number}	--
Boolean	{boolean}	--
Object	{Object}	--
Function	{Function}	--
RegExp	{RegExp}	--
Array	{Array}	--
Date	{Date}	--
单一类型集合	{Array.<string>}	string 类型的数组
多类型	{(number boolean)}	可能是 number 类型, 也可能是 boolean 类型
允许为null	{?number}	可能是 number, 也可能是 null
不允许为null	{!Object}	Object 类型, 但不是 null
Function类型	{function(number, boolean)}	函数, 形参类型
Function带返回值	{function(number, boolean):string}	函数, 形参, 返回值类型
Promise	Promise.<resolveType, rejectType>	Promise, 成功返回的数据类型, 失败返回的错误类型
参数可选	@param {string=} name	可选参数, =为类型后缀
可变参数	@param {...number} args	变长参数, ...为类型前缀
任意类型	{*}	任意类型
可选任意类型	@param {*=} name	可选参数, 类型不限
可变任意类型	@param {...*} args	变长参数, 类型不限

2.4.5 文件注释

[强制] 文件顶部必须包含文件注释，用 `@file` 标识文件说明。

示例：

```
/**
 * @file Describe the file
 */
```

[建议] 文件注释中可以用 `@author` 标识开发者信息。

解释：

开发者信息能够体现开发人员对文件的贡献，并且能够让遇到问题或希望了解相关信息的人找到维护人。通常情况文件在被创建时标识的是创建者。随着项目的进展，越来越多的人加入，参与这个文件的开发，新的作者应该被加入 `@author` 标识。

`@author` 标识具有多人时，原则是按照 责任 进行排序。通常的说就是如果有问题，就是找第一个人应该比找第二个人有效。比如文件的创建者由于各种原因，模块移交给了其他人或其他团队，后来因为新增需求，其他人在新增代码时，添加 `@author` 标识应该把自己的名字添加在创建人的前面。

`@author` 中的名字不允许被删除。任何劳动成果都应该被尊重。

业务项目中，一个文件可能被多人频繁修改，并且每个人的维护时间都可能不会很长，不建议为文件增加 `@author` 标识。通过版本控制系统追踪变更，按业务逻辑单元确定模块的维护责任人，通过文档与wiki跟踪和查询，是更好的责任管理方式。

对于业务逻辑无关的技术型基础项目，特别是开源的公共项目，应使用 `@author` 标识。

示例：

```
/**
 * @file Describe the file
 * @author author-name(mail-name@domain.com)
 *         author-name2(mail-name2@domain.com)
 */
```

2.4.6 命名空间注释

[建议] 命名空间使用 `@namespace` 标识。

示例：

```
/**
 * @namespace
 */
var util = {};
```

2.4.7 类注释

[建议] 使用 `@class` 标记类或构造函数。

解释：

对于使用对象 `constructor` 属性来定义的构造函数，可以使用 `@constructor` 来标记。

示例：

```
/**  
 * 描述  
 *  
 * @class  
 */  
function Developer() {  
    // constructor body  
}
```

[建议] 使用 `@extends` 标记类的继承信息。

示例：

```
/**  
 * 描述  
 *  
 * @class  
 * @extends Developer  
 */  
function Fronteer() {  
    Developer.call(this);  
    // constructor body  
}  
util.inherits(Fronteer, Developer);
```

[强制] 使用包装方式扩展类成员时，必须通过 `@lends` 进行重新指向。

解释：

没有 `@lends` 标记将无法为该类生成包含扩展类成员的文档。

示例：

```
/**  
 * 类描述  
 *  
 * @class  
 * @extends Developer  
 */  
function Fronteer() {  
    Developer.call(this);  
    // constructor body  
}  
  
util.extend(  
    Fronteer.prototype,  
    /** @lends Fronteer.prototype */{  
        getLevel: function () {  
            // TODO  
        }  
    }  
);
```

[强制] 类的属性或方法等成员信息使用 `@public` / `@protected` / `@private` 中的任意一个，指明可访问性。

解释：

生成的文档中将有可访问性的标记，避免用户直接使用非 `public` 的属性或方法。

示例：

```

    /**
     * 类描述
     *
     * @class
     * @extends Developer
     */
    var Fronterer = function () {
        Developer.call(this);

        /**
         * 属性描述
         *
         * @type {string}
         * @private
         */
        this.level = 'T12';

        // constructor body
    };
    util.inherits(Fronterer, Developer);

    /**
     * 方法描述
     *
     * @private
     * @return {string} 返回值描述
     */
    Fronterer.prototype.getLevel = function () {
    };

```

2.4.8 函数/方法注释

[强制] 函数/方法注释必须包含函数说明，有参数和返回值时必须使用注释标识。

解释：

当 `return` 关键字仅作退出函数/方法使用时，无须对返回值作注释标识。

[强制] 参数和返回值注释必须包含类型信息，且不允许省略参数的说明。

[建议] 当函数是内部函数，外部不可访问时，可以使用 `@inner` 标识。

示例：

```

    /**
     * 函数描述
     *
     * @param {string} p1 参数1的说明
     * @param {string} p2 参数2的说明，比较长
     *      那就换行了。
     * @param {number=} p3 参数3的说明（可选）
     * @return {Object} 返回值描述
     */
    function foo(p1, p2, p3) {
        var p3 = p3 || 10;
        return {
            p1: p1,
            p2: p2,
            p3: p3
        };
    }
}

```

[强制] 对 **Object** 中各项的描述，必须使用 `@param` 标识。

示例：

```

    /**
     * 函数描述
     *
     * @param {Object} option 参数描述
     * @param {string} option.url option项描述
     * @param {string=} option.method option项描述，可选参数
     */
    function foo(option) {
        // TODO
    }
}

```

[建议] 重写父类方法时，应当添加 `@override` 标识。如果重写的形参个数、类型、顺序和返回值类型均未发生变化，可省略 `@param`、`@return`，仅用 `@override` 标识，否则仍应作完整注释。

解释：

简而言之，当子类重写的方法能直接套用父类的方法注释时可省略对参数与返回值的注释。

2.4.9 事件注释

[强制] 必须使用 `@event` 标识事件，事件参数的标识与方法描述的参数标识相同。

示例：

```

/**
 * 值变更时触发
 *
 * @event Select#change
 * @param {Object} e e描述
 * @param {string} e.before before描述
 * @param {string} e.after after描述
 */
this.fire(
  'change',
  {
    before: 'foo',
    after: 'bar'
  }
);

```

[强制] 在会广播事件的函数前使用 `@fires` 标识广播的事件，在广播事件代码前使用 `@event` 标识事件。

[建议] 对于事件对象的注释，使用 `@param` 标识，生成文档时可读性更好。

示例：

```

/**
 * 点击处理
 *
 * @fires Select#change
 * @private
 */
Select.prototype.clickHandler = function () {

  /**
   * 值变更时触发
   *
   * @event Select#change
   * @param {Object} e e描述
   * @param {string} e.before before描述
   * @param {string} e.after after描述
   */
  this.fire(
    'change',
    {
      before: 'foo',
      after: 'bar'
    }
  );
};

```

2.4.10 常量注释

[强制] 常量必须使用 `@const` 标记，并包含说明和类型信息。

示例：

```
/**
 * 常量说明
 *
 * @const
 * @type {string}
 */
var REQUEST_URL = 'myurl.do';
```

2.4.11 复杂类型注释

[建议] 对于类型未定义的复杂结构的注释，可以使用 `@typedef` 标识来定义。

示例：

```
// `namespaceA~` 可以换成其它 namepaths 前缀，目的是为了生成文档中能显示 `@typedef` 定义的类型
// 和链接。
/**
 * 服务器
 *
 * @typedef {Object} namespaceA~Server
 * @property {string} host 主机
 * @property {number} port 端口
 */

/**
 * 服务器列表
 *
 * @type {Array.<namespaceA~Server>}
 */
var servers = [
  {
    host: '1.2.3.4',
    port: 8080
  },
  {
    host: '1.2.3.5',
    port: 8081
  }
];
```

2.4.12 AMD 模块注释

[强制] AMD 模块使用 `@module` 或 `@exports` 标识。

解释：

@exports 与 @module 都可以用来标识模块，区别在于 @module 可以省略模块名称。而只使用 @exports 时在 namepaths 中可以省略 module: 前缀。

示例：

```
define(
    function (require) {
        /**
         * foo description
         *
         * @exports Foo
         */
        var foo = {
            // TODO
        };

        /**
         * baz description
         *
         * @return {boolean} return description
         */
        foo.baz = function () {
            // TODO
        };

        return foo;
    }
);
```

也可以在 exports 变量前使用 @module 标识：

```

define(
  function (require) {

    /**
     * module description.
     *
     * @module foo
     */
    var exports = {};

    /**
     * bar description
     *
     */
    exports.bar = function () {
      // TODO
    };

    return exports;
  }
);

```

如果直接使用 factory 的 exports 参数，还可以：

```

/**
 * module description.
 *
 * @module
 */
define(
  function (require, exports) {

    /**
     * bar description
     *
     */
    exports.bar = function () {
      // TODO
    };
    return exports;
  }
);

```

[强制] 对于已使用 `@module` 标识为 **AMD**模块 的引用，在 `namepaths` 中必须增加 `module:` 作前缀。

解释：

`namepaths` 没有 `module:` 前缀时，生成的文档中将无法正确生成链接。

示例：

```
/**
 * 点击处理
 *
 * @fires module:Select#change
 * @private
 */
Select.prototype.clickHandler = function () {
    /**
     * 值变更时触发
     *
     * @event module:Select#change
     * @param {Object} e e描述
     * @param {string} e.before before描述
     * @param {string} e.after after描述
     */
    this.fire(
        'change',
        {
            before: 'foo',
            after: 'bar'
        }
    );
};
```

[建议] 对于类定义的模块，可以使用 `@alias` 标识构建函数。

示例：

```
/**
 * A module representing a jacket.
 * @module jacket
 */
define(
    function () {

        /**
         * @class
         * @alias module:jacket
         */
        var Jacket = function () {
        };

        return Jacket;
    }
);
```

[建议] 多模块定义时，可以使用 `@exports` 标识各个模块。

示例：

```
// one module
define('html/utils',
  /**
   * Utility functions to ease working with DOM elements.
   * @exports html/utils
   */
  function () {
    var exports = {
    };

    return exports;
  }
);

// another module
define('tag',
  /** @exports tag */
  function () {
    var exports = {
    };

    return exports;
  }
);
```

[建议] 对于 `exports` 为 `Object` 的模块，可以使用 `@namespace` 标识。

解释：

使用 `@namespace` 而不是 `@module` 或 `@exports` 时，对模块的引用可以省略 `module:` 前缀。

[建议] 对于 `exports` 为类名的模块，使用 `@class` 和 `@exports` 标识。

示例：

```
// 只使用 @class Bar 时，类方法和属性都必须增加 @name Bar#methodName 来标识，与 @exports 配合
// 可以免除这一麻烦，并且在引用时可以省去 module: 前缀。
// 另外需要注意类名需要使用 var 定义的方式。

/**
 * Bar description
 *
 * @see foo
 * @exports Bar
 * @class
 */
var Bar = function () {
    // TODO
};

/**
 * baz description
 *
 * @return {string|Array} return description
 */
Bar.prototype.baz = function () {
    // TODO
};

```

2.4.13 细节注释

对于内部实现、不容易理解的逻辑说明、摘要信息等，我们可能需要编写细节注释。

[建议] 细节注释遵循单行注释的格式。说明必须换行时，每行是一个单行注释的起始。

示例：

```
function foo(p1, p2, opt_p3) {
    // 这里对具体内部逻辑进行说明
    // 说明太长需要换行
    for (...) {
        ....
    }
}
```

[强制] 有时我们会使用一些特殊标记进行说明。特殊标记必须使用单行注释的形式。下面列举了一些常用标记：

解释：

1. TODO: 有功能待实现。此时需要对将要实现的功能进行简单说明。
2. FIXME: 该处代码运行没问题，但可能由于时间赶或者其他原因，需要修正。此时需要对如何修正进行简单说明。
3. HACK: 为修正某些问题而写的不太好或者使用了某些诡异手段的代码。此时需要对思路或诡异手段进行描述。
4. XXX: 该处存在陷阱。此时需要对陷阱进行描述。

3 语言特性

3.1 变量

【强制】 变量、函数在使用前必须先定义。

解释：

不通过 `var` 定义变量将导致变量污染全局环境。

示例：

```
// good
var name = 'MyName';

// bad
name = 'MyName';
```

原则上不建议使用全局变量，对于已有的全局变量或第三方框架引入的全局变量，需要根据检查工具的语法标识。

示例：

```
/* globals jQuery */
var element = jQuery('#element-id');
```

【强制】 每个 `var` 只能声明一个变量。

解释：

一个 `var` 声明多个变量，容易导致较长的行长度，并且在修改时容易造成逗号和分号的混淆。

示例：

```
// good
var hangModules = [];
var missModules = [];
var visited = {};

// bad
var hangModules = [],
    missModules = [],
    visited = {};
```

【强制】 变量必须 **即用即声明**，不得在函数或其它形式的代码块起始位置统一声明所有变量。

解释：

变量声明与使用的距离越远，出现的跨度越大，代码的阅读与维护成本越高。虽然JavaScript的变量是函数作用域，还是应该根据编程中的意图，缩小变量出现的距离空间。

示例：

```
// good
function kv2List(source) {
    var list = [];

    for (var key in source) {
        if (source.hasOwnProperty(key)) {
            var item = {
                k: key,
                v: source[key]
            };

            list.push(item);
        }
    }

    return list;
}

// bad
function kv2List(source) {
    var list = [];
    var key;
    var item;

    for (key in source) {
        if (source.hasOwnProperty(key)) {
            item = {
                k: key,
                v: source[key]
            };

            list.push(item);
        }
    }

    return list;
}
```

3.2 条件

[强制] 在 **Equality Expression** 中使用类型严格的 `==`。仅当判断 `null` 或 `undefined` 时，允许使用 `== null`。

解释：

使用 `==` 可以避免等于判断中隐式的类型转换。

示例：

```
// good
if (age === 30) {
    // .....
}

// bad
if (age == 30) {
    // .....
}
```

【建议】尽可能使用简洁的表达式。

示例：

```
// 字符串为空

// good
if (!name) {
    // .....
}

// bad
if (name === '') {
    // .....
}
```

```
// 字符串非空

// good
if (name) {
    // .....
}

// bad
if (name !== '') {
    // .....
}
```

```
// 数组非空

// good
if (collection.length) {
    // .....
}

// bad
if (collection.length > 0) {
    // .....
}
```

```
// 布尔不成立

// good
if (!notTrue) {
    // .....
}

// bad
if (notTrue === false) {
    // .....
}
```

```
// null 或 undefined

// good
if (noValue == null) {
    // .....
}

// bad
if (noValue === null || typeof noValue === 'undefined') {
    // .....
}
```

[建议] 按执行频率排列分支的顺序。

解释：

按执行频率排列分支的顺序好处是：

1. 阅读的人容易找到最常见的情况，增加可读性。
2. 提高执行效率。

[建议] 对于相同变量或表达式的多值条件，用 `switch` 代替 `if`。

示例：

```
// good
switch (typeof variable) {
    case 'object':
        // .....
        break;
    case 'number':
    case 'boolean':
    case 'string':
        // .....
        break;
}

// bad
var type = typeof variable;
if (type === 'object') {
    // .....
}
else if (type === 'number' || type === 'boolean' || type === 'string') {
    // .....
}
```

【建议】如果函数或全局中的 `else` 块后没有任何语句，可以删除 `else`。

示例：

```
// good
function getName() {
    if (name) {
        return name;
    }

    return 'unnamed';
}

// bad
function getName() {
    if (name) {
        return name;
    }
    else {
        return 'unnamed';
    }
}
```

3.3 循环

【建议】不要在循环体中包含函数表达式，事先将函数提取到循环体外。

解释：

循环体中的函数表达式，运行过程中会生成循环次数个函数对象。

示例：

```
// good
function clicker() {
    // .....
}

for (var i = 0, len = elements.length; i < len; i++) {
    var element = elements[i];
    addListener(element, 'click', clicker);
}

// bad
for (var i = 0, len = elements.length; i < len; i++) {
    var element = elements[i];
    addListener(element, 'click', function () {});
}
```

【建议】对循环内多次使用的不变值，在循环外用变量缓存。

示例：

```
// good
var width = wrap.offsetWidth + 'px';
for (var i = 0, len = elements.length; i < len; i++) {
    var element = elements[i];
    element.style.width = width;
    // .....
}

// bad
for (var i = 0, len = elements.length; i < len; i++) {
    var element = elements[i];
    element.style.width = wrap.offsetWidth + 'px';
    // .....
}
```

【建议】对有序集合进行遍历时，缓存 `length`。

解释：

虽然现代浏览器都对数组长度进行了缓存，但对于一些宿主对象和老旧浏览器的数组对象，在每次 `length` 访问时会动态计算元素个数，此时缓存 `length` 能有效提高程序性能。

示例：

```
for (var i = 0, len = elements.length; i < len; i++) {  
    var element = elements[i];  
    // ....  
}
```

[建议] 对有序集合进行顺序无关的遍历时，使用逆序遍历。

解释：

逆序遍历可以节省变量，代码比较优化。

示例：

```
var len = elements.length;  
while (len--) {  
    var element = elements[len];  
    // ....  
}
```

3.4 类型

3.4.1 类型检测

[建议] 类型检测优先使用 `typeof`。对象类型检测使用 `instanceof`。`null` 或 `undefined` 的检测使用 `== null`。

示例：

```

// string
typeof variable === 'string'

// number
typeof variable === 'number'

// boolean
typeof variable === 'boolean'

// Function
typeof variable === 'function'

// Object
typeof variable === 'object'

// RegExp
variable instanceof RegExp

// Array
variable instanceof Array

// null
variable === null

// null or undefined
variable == null

// undefined
typeof variable === 'undefined'

```

3.4.2 类型转换

[建议] 转换成 `string` 时，使用 `+ ''`。

示例：

```

// good
num + '';

// bad
new String(num);
num.toString();
String(num);

```

[建议] 转换成 `number` 时，通常使用 `+`。

示例：

```
// good
+str;

// bad
Number(str);
```

[建议] `string` 转换成 `number`，要转换的字符串结尾包含非数字并期望忽略时，使用 `parseInt`。

示例：

```
var width = '200px';
parseInt(width, 10);
```

[强制] 使用 `parseInt` 时，必须指定进制。

示例：

```
// good
parseInt(str, 10);

// bad
parseInt(str);
```

[建议] 转换成 `boolean` 时，使用 `!!`。

示例：

```
var num = 3.14;
!!num;
```

[建议] `number` 去除小数点，使用 `Math.floor` / `Math.round` / `Math.ceil`，不使用 `parseInt`。

示例：

```
// good
var num = 3.14;
Math.ceil(num);

// bad
var num = 3.14;
parseInt(num, 10);
```

3.5 字符串

[强制] 字符串开头和结束使用单引号 '。

解释：

1. 输入单引号不需要按住 shift , 方便输入。
2. 实际使用中，字符串经常用来拼接 HTML。为方便 HTML 中包含双引号而不需要转义写法。

示例：

```
var str = '我是一个字符串';
var html = '<div class="cls">拼接HTML可以省去双引号转义</div>';
```

[建议] 使用 `数组` 或 `+` 拼接字符串。

解释：

1. 使用 `+` 拼接字符串，如果拼接的全部是 StringLiteral，压缩工具可以对其进行自动合并的优化。所以，静态字符串建议使用 `+` 拼接。
2. 在现代浏览器下，使用 `+` 拼接字符串，性能较数组的方式要高。
3. 如需要兼顾老旧浏览器，应尽量使用数组拼接字符串。

示例：

```
// 使用数组拼接字符串
var str = [
    // 推荐换行开始并缩进开始第一个字符串，对齐代码，方便阅读。
    '<ul>',
    '<li>第一项</li>',
    '<li>第二项</li>',
    '</ul>'
].join('');

// 使用 `+` 拼接字符串
var str2 = '' // 建议第一个为空字符串，第二个换行开始并缩进开始，对齐代码，方便阅读
+ '<ul>',
+   '<li>第一项</li>',
+   '<li>第二项</li>',
+ '</ul>';
```

[建议] 使用字符串拼接的方式生成**HTML**，需要根据语境进行合理的转义。

解释：

在 `JavaScript` 中拼接，并且最终将输出到页面中的字符串，需要进行合理转义，以防止安全漏洞。下面的示例代码为场景说明，不能直接运行。

示例：

```
// HTML 转义
var str = '<p>' + htmlEncode(content) + '</p>';

// HTML 转义
var str = '<input type="text" value="' + htmlEncode(value) + '">';

// URL 转义
var str = '<a href="/?key=' + htmlEncode(urlEncode(value)) + '">link</a>';

// JavaScript字符串 转义 + HTML 转义
var str = '<button onclick="check(\'' + htmlEncode(strLiteral(name)) + '\')">提交</button>';
```

【建议】 复杂的数据到视图字符串的转换过程，选用一种模板引擎。

解释：

使用模板引擎有如下好处：

1. 在开发过程中专注于数据，将视图生成的过程由另外一个层级维护，使程序逻辑结构更清晰。
2. 优秀的模板引擎，通过模板编译技术和高质量的编译产物，能获得比手工拼接字符串更高的性能。
3. 模板引擎能方便的对动态数据进行相应的转义，部分模板引擎默认进行HTML转义，安全性更好。
4. **artTemplate:** 体积较小，在所有环境下性能高，语法灵活。
5. **dot.js:** 体积小，在现代浏览器下性能高，语法灵活。
6. **etpl:** 体积较小，在所有环境下性能高，模板复用性高，语法灵活。
7. **handlebars:** 体积大，在所有环境下性能高，扩展性高。
8. **hogan:** 体积小，在现代浏览器下性能高。
9. **nunjucks:** 体积较大，性能一般，模板复用性高。

3.6 对象

【强制】 使用对象字面量 `{}` 创建新 `object`。

示例：

```
// good
var obj = {};

// bad
var obj = new Object();
```

【建议】 对象创建时，如果一个对象的所有 属性 均可以不添加引号，建议所有 属性 不添加引号。

示例：

```
var info = {
    name: 'someone',
    age: 28
};
```

【建议】 对象创建时，如果任何一个 属性 需要添加引号，则所有 属性 建议添加 '。

解释：

如果属性不符合 Identifier 和 NumberLiteral 的形式，就需要以 StringLiteral 的形式提供。

示例：

```
// good
var info = {
    'name': 'someone',
    'age': 28,
    'more-info': '...'
};

// bad
var info = {
    name: 'someone',
    age: 28,
    'more-info': '...'
};
```

【强制】 不允许修改和扩展任何原生对象和宿主对象的原型。

示例：

```
// 以下行为绝对禁止
String.prototype.trim = function () {
};
```

【建议】 属性访问时，尽量使用 . 。

解释：

属性名符合 **Identifier** 的要求，就可以通过 `.` 来访问，否则就只能通过 `[expr]` 方式访问。

通常在 JavaScript 中声明的对象，属性命名是使用 **Camel** 命名法，用 `.` 来访问更清晰简洁。部分特殊的属性（比如来自后端的 JSON），可能采用不寻常的命名方式，可以通过 `[expr]` 方式访问。

示例：

```
info.age;
info['more-info'];
```

【建议】 `for in` 遍历对象时，使用 `hasOwnProperty` 过滤掉原型中的属性。

示例：

```
var newInfo = {};
for (var key in info) {
    if (info.hasOwnProperty(key)) {
        newInfo[key] = info[key];
    }
}
```

3.7 数组

【强制】 使用数组字面量 `[]` 创建新数组，除非想要创建的是指定长度的数组。

示例：

```
// good
var arr = [];

// bad
var arr = new Array();
```

【强制】 遍历数组不使用 `for in`。

解释：

数组对象可能存在数字以外的属性，这种情况下 `for in` 不会得到正确结果。

示例：

```

var arr = ['a', 'b', 'c'];

// 这里仅作演示，实际中应使用 Object 类型
arr.other = 'other things';

// 正确的遍历方式
for (var i = 0, len = arr.length; i < len; i++) {
    console.log(i);
}

// 错误的遍历方式
for (var i in arr) {
    console.log(i);
}

```

[建议] 不因为性能的原因自己实现数组排序功能，尽量使用数组的 `sort` 方法。

解释：

自己实现的常规排序算法，在性能上并不优于数组默认的 `sort` 方法。以下两种场景可以自己实现排序：

1. 需要稳定的排序算法，达到严格一致的排序结果。
2. 数据特点鲜明，适合使用桶排。

[建议] 清空数组使用 `.length = 0`。

3.8 函数

3.8.1 函数长度

[建议] 一个函数的长度控制在 `50` 行以内。

解释：

将过多的逻辑单元混在一个大函数中，易导致难以维护。一个清晰易懂的函数应该完成单一的逻辑单元。复杂的操作应进一步抽取，通过函数的调用来体现流程。

特定算法等不可分割的逻辑允许例外。

示例：

```

function syncViewStateOnUserAction() {
    if (x.checked) {
        y.checked = true;
        z.value = '';
    }
    else {
        y.checked = false;
    }

    if (a.value) {
        warning.innerText = '';
        submitButton.disabled = false;
    }
    else {
        warning.innerText = 'Please enter it';
        submitButton.disabled = true;
    }
}

// 直接阅读该函数会难以明确其主线逻辑，因此下方是一种更合理的表达方式：

function syncViewStateOnUserAction() {
    syncXStateToView();
    checkAAvailability();
}

function syncXStateToView() {
    y.checked = x.checked;

    if (x.checked) {
        z.value = '';
    }
}

function checkAAvailability() {
    if (a.value) {
        clearWarnignForA();
    }
    else {
        displayWarningForAMissing();
    }
}

```

3.8.2 参数设计

【建议】一个函数的参数控制在 **6** 个以内。

解释：

除去不定长参数以外，函数具备不同逻辑意义的参数建议控制在 6 个以内，过多参数会导致维护难度增大。

某些情况下，如使用 AMD Loader 的 `require` 加载多个模块时，其 `callback` 可能会存在较多参数，因此对函数参数的个数不做强制限制。

[建议] 通过 `options` 参数传递非数据输入型参数。

解释：

有些函数的参数并不是作为算法的输入，而是对算法的某些分支条件判断之用，此类参数建议通过一个 `options` 参数传递。

如下函数：

```
/**
 * 移除某个元素
 *
 * @param {Node} element 需要移除的元素
 * @param {boolean} removeEventListeners 是否同时将所有注册在元素上的事件移除
 */
function removeElement(element, removeEventListeners) {
    element.parent.removeChild(element);

    if (removeEventListeners) {
        element.clearEventListeners();
    }
}
```

可以转换为下面的签名：

```
/**
 * 移除某个元素
 *
 * @param {Node} element 需要移除的元素
 * @param {Object} options 相关的逻辑配置
 * @param {boolean} options.removeEventListeners 是否同时将所有注册在元素上的事件移除
 */
function removeElement(element, options) {
    element.parent.removeChild(element);

    if (options.removeEventListeners) {
        element.clearEventListeners();
    }
}
```

这种模式有几个显著的优势：

- `boolean` 型的配置项具备名称，从调用的代码上更易理解其表达的逻辑意义。

- 当配置项有增长时，无需无休止地增加参数个数，不会出现 `removeElement(element, true, false, false, 3)` 这样难以理解的调用代码。
- 当部分配置参数可选时，多个参数的形式非常难处理重载逻辑，而使用一个 `options` 对象只需判断属性是否存在，实现得以简化。

3.8.3 闭包

[建议] 在适当的时候将闭包内大对象置为 `null`。

解释：

在 JavaScript 中，无需特别的关键词就可以使用闭包，一个函数可以任意访问在其定义的作用域外的变量。需要注意的是，函数的作用域是静态的，即在定义时决定，与调用的时机和方式没有任何关系。

闭包会阻止一些变量的垃圾回收，对于较老旧的 JavaScript 引擎，可能导致外部所有变量均无法回收。

首先一个较为明确的结论是，以下内容会影响到闭包内变量的回收：

- 嵌套的函数中是否有使用该变量。
- 嵌套的函数中是否有直接调用 `eval`。
- 是否使用了 `with` 表达式。

Chakra、V8 和 SpiderMonkey 将受以上因素的影响，表现出不尽相同又较为相似的回收策略，而 JScript.dll 和 Carakan 则完全没有这方面的优化，会完整保留整个 `LexicalEnvironment` 中的所有变量绑定，造成一定的内存消耗。

由于对闭包内变量有回收优化策略的 Chakra、V8 和 SpiderMonkey 引擎的行为较为相似，因此可以总结如下，当返回一个函数 `fn` 时：

- 如果 `fn` 的 `[[Scope]]` 是 `ObjectEnvironment` (`with` 表达式生成 `ObjectEnvironment`，函数和 `catch` 表达式生成 `DeclarativeEnvironment`)，则：
 - 如果是 V8 引擎，则退出全过程。
 - 如果是 SpiderMonkey，则处理该 `ObjectEnvironment` 的外层 `LexicalEnvironment`。
- 获取当前 `LexicalEnvironment` 下的所有类型为 `Function` 的对象，对于每一个 `Function` 对象，分析其 `FunctionBody`：
 - 如果 `FunctionBody` 中含有直接调用 `eval`，则退出全过程。
 - 否则得到所有的 `Identifier`。
 - 对于每一个 `Identifier`，设其为 `name`，根据查找变量引用的规则，从 `LexicalEnvironment` 中找出名称为 `name` 的绑定 `binding`。
 - 对 `binding` 添加 `notSwap` 属性，其值为 `true`。
- 检查当前 `LexicalEnvironment` 中的每一个变量绑定，如果该绑定有 `notSwap` 属性且值为 `true`，则：

- i. 如果是 V8 引擎，删除该绑定。
- ii. 如果是 SpiderMonkey，将该绑定的值设为 `undefined`，将删除 `notSwap` 属性。

对于 Chakra 引擎，暂无法得知是按 V8 的模式还是按 SpiderMonkey 的模式进行。

如果有 非常庞大的对象，且预计会在 老旧的引擎 中执行，则使用闭包时，注意将闭包不需要的对象置为空引用。

[建议] 使用 `IIFE` 避免 `Lift` 效应 。

解释：

在引用函数外部变量时，函数执行时外部变量的值由运行时决定而非定义时，最典型的场景如下：

```
var tasks = [];
for (var i = 0; i < 5; i++) {
  tasks[tasks.length] = function () {
    console.log('Current cursor is at ' + i);
  };
}

var len = tasks.length;
while (len--) {
  tasks[len]();
}
```

以上代码对 `tasks` 中的函数的执行均会输出 `current cursor is at 5`，往往不符合预期。

此现象称为 `Lift` 效应。解决的方式是通过额外加上一层闭包函数，将需要的外部变量作为参数传递来解除变量的绑定关系：

```
var tasks = [];
for (var i = 0; i < 5; i++) {
  // 注意有一层额外的闭包
  tasks[tasks.length] = (function (i) {
    return function () {
      console.log('Current cursor is at ' + i);
    };
  })(i);
}

var len = tasks.length;
while (len--) {
  tasks[len]();
}
```

3.8.4 空函数

[建议] 空函数不使用 `new Function()` 的形式。

示例：

```
var emptyFunction = function () {};
```

[建议] 对于性能有高要求的场合，建议存在一个空函数的常量，供多处使用共享。

示例：

```
var EMPTY_FUNCTION = function () {};

function MyClass() {}

MyClass.prototype.abstractMethod = EMPTY_FUNCTION;
MyClass.prototype.hooks.before = EMPTY_FUNCTION;
MyClass.prototype.hooks.after = EMPTY_FUNCTION;
```

3.9 面向对象

[强制] 类的继承方案，实现时需要修正 `constructor`。

解释：

通常使用其他 library 的类继承方案都会进行 `constructor` 修正。如果是自己实现的类继承方案，需要进行 `constructor` 修正。

示例：

```
/**
 * 构建类之间的继承关系
 *
 * @param {Function} subClass 子类函数
 * @param {Function} superClass 父类函数
 */
function inherits(subClass, superClass) {
    var F = new Function();
    F.prototype = superClass.prototype;
    subClass.prototype = new F();
    subClass.prototype.constructor = subClass;
}
```

[建议] 声明类时，保证 `constructor` 的正确性。

示例：

```

function Animal(name) {
    this.name = name;
}

// 直接prototype等于对象时，需要修正constructor
Animal.prototype = {
    constructor: Animal,

    jump: function () {
        alert('animal ' + this.name + ' jump');
    }
};

// 这种方式扩展prototype则无需理会constructor
Animal.prototype.jump = function () {
    alert('animal ' + this.name + ' jump');
};

```

[建议] 属性在构造函数中声明，方法在原型中声明。

解释：

原型对象的成员被所有实例共享，能节约内存占用。所以编码时我们应该遵守这样的原则：原型对象包含程序不会修改的成员，如方法函数或配置项。

```

function TextNode(value, engine) {
    this.value = value;
    this.engine = engine;
}

TextNode.prototype.clone = function () {
    return this;
};

```

[强制] 自定义事件的 事件名 必须全小写。

解释：

在 JavaScript 广泛应用的浏览器环境，绝大多数 DOM 事件名称都是全小写的。为了遵循大多数 JavaScript 开发者的习惯，在设计自定义事件时，事件名也应该全小写。

[强制] 自定义事件只能有一个 `event` 参数。如果事件需要传递较多信息，应仔细设计事件对象。

解释：

一个事件对象的好处有：

1. 顺序无关，避免事件监听者需要记忆参数顺序。

2. 每个事件信息都可以根据需要提供或者不提供，更自由。
3. 扩展方便，未来添加事件信息时，无需考虑会破坏监听器参数形式而无法向后兼容。

[建议] 设计自定义事件时，应考虑禁止默认行为。

解释：

常见禁止默认行为的方式有两种：

1. 事件监听函数中 `return false`。
2. 事件对象中包含禁止默认行为的方法，如 `preventDefault`。

3.10 动态特性

3.10.1 eval

[强制] 避免使用直接 `eval` 函数。

解释：

直接 `eval`，指的是以函数方式调用 `eval` 的调用方法。直接 `eval` 调用执行代码的作用域为本地作用域，应当避免。

如果有特殊情况需要使用直接 `eval`，需在代码中用详细的注释说明为何必须使用直接 `eval`，不能使用其它动态执行代码的方式，同时需要其他资深工程师进行 Code Review。

[建议] 尽量避免使用 `eval` 函数。

3.10.2 动态执行代码

[建议] 使用 `new Function` 执行动态代码。

解释：

通过 `new Function` 生成的函数作用域是全局使用域，不会影响当当前的本地作用域。如果有动态代码执行的需求，建议使用 `new Function`。

示例：

```
var handler = new Function('x', 'y', 'return x + y;');
var result = handler($('#x').val(), $('#y').val());
```

3.10.3 with

[建议] 尽量不要使用 `with`。

解释：

使用 `with` 可能会增加代码的复杂度，不利于阅读和管理；也会对性能有影响。大多数使用 `with` 的场景都能使用其他方式较好的替代。所以，尽量不要使用 `with`。

3.10.4 delete

[建议] 减少 `delete` 的使用。

解释：

如果没有特别的需求，减少或避免使用 `delete`。`delete` 的使用会破坏部分 JavaScript 引擎的性能优化。

[建议] 处理 `delete` 可能产生的异常。

解释：

对于有被遍历需求，且值 `null` 被认为具有业务逻辑意义的值的对象，移除某个属性必须使用 `delete` 操作。

在严格模式或 IE 下使用 `delete` 时，不能被删除的属性会抛出异常，因此在不确定属性是否可以删除的情况下，建议添加 `try-catch` 块。

示例：

```
try {
    delete o.x;
}
catch (deleteError) {
    o.x = null;
}
```

3.10.5 对象属性

[建议] 避免修改外部传入的对象。

解释：

JavaScript 因其脚本语言的动态特性，当一个对象未被 `seal` 或 `freeze` 时，可以任意添加、删除、修改属性值。

但是随意地对 非自身控制的对象 进行修改，很容易造成代码在不可预知的情况下出现问题。因此，设计良好的组件、函数应该避免对外部传入的对象的修改。

下面代码的 `selectNode` 方法修改了由外部传入的 `datasource` 对象。如果 `datasource` 用在其它场合（如另一个 Tree 实例）下，会造成状态的混乱。

```

function Tree(datasource) {
    this.datasource = datasource;
}

Tree.prototype.selectNode = function (id) {
    // 从datasource中找出节点对象
    var node = this.findNode(id);
    if (node) {
        node.selected = true;
        this.flushView();
    }
};

```

对于此类场景，需要使用额外的对象来维护，使用由自身控制，不与外部产生任何交互的 **selectedNodeIndex** 对象来维护节点的选中状态，不对 **datasource** 作任何修改。

```

function Tree(datasource) {
    this.datasource = datasource;
    this.selectedNodeIndex = {};
}

Tree.prototype.selectNode = function (id) {

    // 从datasource中找出节点对象
    var node = this.findNode(id);

    if (node) {
        this.selectedNodeIndex[id] = true;
        this.flushView();
    }
};

```

除此之外，也可以通过 `deepClone` 等手段将自身维护的对象与外部传入的分离，保证不会相互影响。

【建议】 具备强类型的设计。

解释：

- 如果一个属性被设计为 `boolean` 类型，则不要使用 `1` 或 `0` 作为其值。对于标识性的属性，如对代码体积有严格要求，可以从一开始就设计为 `number` 类型且将 `0` 作为否定值。
- 从 DOM 中取出的值通常为 `string` 类型，如果有对象或函数的接收类型为 `number` 类型，提前作好转换，而不是期望对象、函数可以处理多类型的值。

4 浏览器环境

4.1 模块化

4.1.1 AMD

[强制] 使用 `AMD` 作为模块定义。

解释：

AMD 作为由社区认可的模块定义形式，提供多种重载提供灵活的使用方式，并且绝大多数优秀的 Library 都支持 AMD，适合作为规范。

目前，比较成熟的 AMD Loader 有：

- 官方实现的 [requirejs](#)
- 百度自己实现的 [esl](#)

[强制] 模块 `id` 必须符合标准。

解释：

模块 `id` 必须符合以下约束条件：

1. 类型为 `string`，并且是由 `/` 分割的一系列 `terms` 来组成。例如：`this/is/a/module`。
2. `term` 应该符合 `[a-zA-Z0-9_-]+` 规则。
3. 不应该有 `.js` 后缀。
4. 跟文件的路径保持一致。

4.1.2 define

[建议] 定义模块时不要指明 `id` 和 `dependencies`。

解释：

在 AMD 的设计思想里，模块名称是和所在路径相关的，匿名的模块更利于封装和迁移。模块依赖应在模块定义内部通过 `local require` 引用。

所以，推荐使用 `define(factory)` 的形式进行模块定义。

示例：

```
define(
  function (require) {
  }
);
```

[建议] 使用 `return` 来返回模块定义。

解释：

使用 `return` 可以减少 `factory` 接收的参数（不需要接收 `exports` 和 `module`），在没有 AMD Loader 的场景下也更容易进行简单的处理来伪造一个 Loader。

示例：

```
define(
  function (require) {
    var exports = {};

    // ...

    return exports;
  }
);
```

PS: 目前流行的js模块化编程，Forestar前端如果使用的不是AngularJS或者其他模块化前端框架，此处不需要继续往下看

4.1.3 require

[强制] 全局运行环境中，`require` 必须以 `async require` 形式调用。

解释：

模块的加载过程是异步的，同步调用并无法保证得到正确的结果。

示例：

```
// good
require(['foo'], function (foo) {
});

// bad
var foo = require('foo');
```

[强制] 模块定义中只允许使用 `local require`，不允许使用 `global require`。

解释：

1. 在模块定义中使用 `global require`，对封装性是一种破坏。
2. 在 AMD 里，`global require` 是可以被重命名的。并且 Loader 甚至没有全局的 `require` 变量，而是用 Loader 名称做为 `global require`。模块定义不应该依赖使用的 Loader。

[强制] Package 在实现时，内部模块的 `require` 必须使用 `relative id`。

解释：

对于任何可能通过发布-引入的形式复用的第三方库、框架、包，开发者所定义的名称不代表使用者使用的名称。因此不要基于任何名称的假设。在实现源码中，`require` 自身的其它模块时使用 `relative id`。

示例：

```
define(
  function (require) {
    var util = require('./util');
  }
);
```

[建议] 不会被调用的依赖模块，在 `factory` 开始处统一 `require`。

解释：

有些模块是依赖的模块，但不会在模块实现中被直接调用，最为典型的是 `css` / `js` / `tpl` 等 `Plugin` 所引入的外部内容。此类内容建议放在模块定义最开始处统一引用。

示例：

```
define(
  function (require) {
    require('css!foo.css');
    require('tpl!bar.tpl.html');

    // ...
  }
);
```

4.2 DOM

4.2.1 元素获取

[建议] 对于单个元素，尽可能使用 `document.getElementById` 获取，避免使用 `document.all`。

[建议] 对于多个元素的集合，尽可能使用 `context.getElementsByTagName` 获取。其中 `context` 可以为 `document` 或其他元素。指定 `tagName` 参数为 `*` 可以获得所有子元素。

[建议] 遍历元素集合时，尽量缓存集合长度。如需多次操作同一集合，则应将集合转为数组。

解释：

原生获取元素集合的结果并不直接引用 DOM 元素，而是对索引进行读取，所以 DOM 结构的改变会实时反映到结果中。

示例：

```
<div></div>
<span></span>

<script>
var elements = document.getElementsByTagName('*');

// 显示为 DIV
alert(elements[0].tagName);

var div = elements[0];
var p = document.createElement('p');
document.body.insertBefore(p, div);

// 显示为 P
alert(elements[0].tagName);
</script>
```

[建议] 获取元素的直接子元素时使用 `children`。避免使用 `childNodes`，除非预期是需要包含文本、注释和属性类型的节点。

4.2.2 样式获取

[建议] 获取元素实际样式信息时，应使用 `getComputedStyle` 或 `currentStyle`。

解释：

通过 `style` 只能获得内联定义或通过 JavaScript 直接设置的样式。通过 CSS class 设置的元素样式无法直接通过 `style` 获取。

4.2.3 样式设置

[建议] 尽可能通过为元素添加预定义的 `className` 来改变元素样式，避免直接操作 `style` 设置。

[强制] 通过 `style` 对象设置元素样式时，对于带单位非 **0** 值的属性，不允许省略单位。

解释：

除了 IE，标准浏览器会忽略不规范的属性值，导致兼容性问题。

4.2.4 DOM 操作

[建议] 操作 `DOM` 时，尽量减少页面 `reflow`。

解释：

页面 `reflow` 是非常耗时的行为，非常容易导致性能瓶颈。下面一些场景会触发浏览器的 `reflow`：

- DOM元素的添加、修改（内容）、删除。
- 应用新的样式或者修改任何影响元素布局的属性。
- Resize浏览器窗口、滚动页面。
- 读取元素的某些属性（`offsetLeft`、`offsetTop`、`offsetHeight`、`offsetWidth`、`scrollTop/Left/Width/Height`、`clientTop/Left/Width/Height`、`getComputedStyle()`、`currentStyle`(in IE)）。

[建议] 尽量减少 `DOM` 操作。

解释：

DOM 操作也是非常耗时的一种操作，减少 DOM 操作有助于提高性能。举一个简单的例子，构建一个列表。我们可以用两种方式：

1. 在循环体中 `createElement` 并 `append` 到父元素中。
2. 在循环体中拼接 HTML 字符串，循环结束后写父元素的 `innerHTML`。

第一种方法看起来比较标准，但是每次循环都会对 DOM 进行操作，性能极低。在这里推荐使用第二种方法。

4.2.5 DOM 事件

[建议] 优先使用 `addEventListener` / `attachEvent` 绑定事件，避免直接在 `HTML` 属性中或 `DOM` 的 `expando` 属性绑定事件处理。

解释：

`expando` 属性绑定事件容易导致互相覆盖。

[建议] 使用 `addEventListener` 时第三个参数使用 `false`。

解释：

标准浏览器中的 `addEventListener` 可以通过第三个参数指定两种时间触发模型：冒泡和捕获。而 IE 的 `attachEvent` 仅支持冒泡的事件触发。所以为了保持一致性，通常 `addEventListener` 的第三个参数都为 `false`。

[建议] 在没有事件自动管理的框架支持下，应持有监听器函数的引用，在适当时候（元素释放、页面卸载等）移除添加的监听器。

其他规范

1、前言

前端的开发规范不仅仅是上边三章提到的内容，更多的JavaScript规范学习推荐阅读[airbnb/javascript](#)，下面补充其他方面的规范要求

2、数据库相关

2.1 【强制】数据库命名规则

根据实际情况而定

- 表（或者视图）名命名请统一按照系统和功能模块命名，要求看表和视图能快速知道这是哪个模块下的表；业务表和后台表请明确放置位置，为了便于维护。（如果项目没有规定规则的话采用）
- 请项目经理在项目启动设计的时候，定下表名称和字段名称的命名规范，字段名称是统一英文缩写。避免出现个人习惯导致不统一规范的情况。开发人员在不明确的时候也应该先咨询一下规范要求。
- 字段别名备注必须填写，便于维护

开发工具介绍

这里所说的开发工具，指的是前端开发使用的工具

一、浏览器调式工具

1、浏览器开发调式工具推荐三种

- Google Chrome 推荐 ★★★★★
- Firefox Firebug 推荐 ★★★★
- Internet Explorer 推荐 ★★★

2、说明：

以上是本人个人观点，当然，也是网上调查统计的结果。

使用 Google Chrome 开发者调式工具之前，我曾用过FireFox的 Firebug 调式工具， Firebug 对于前端开发也可谓神器。但在使用了chrome开发者调式工具 后，我就喜欢上它的简洁、快速，无论是启动速度还是页面解析速度还是Javascript执行速度。

但是，作为前端开发，功能和页面样式等必须兼容各个主流浏览器，包括 Internet Explorer 各个版本的兼容。所以，不管你喜欢使用哪款调式工具， Internet Explorer 浏览器的调式使用方法也必须掌握。

3、总结

工具看个人喜好和习惯，为的只是提供开发效率而已。（最后保证网站功能在各浏览器都有个良好的兼容性）

4、教程

[Google Chrome Dev Tool使用教程](#)

[网站性能优化 之 Chrome DevTools（一）](#)

[网站性能优化 之 Chrome DevTools（二）](#)

[网站性能优化 之 Chrome DevTools（三）](#)

[Firefox Firebug Tool 使用教程](#)

Internet Explorer 9 调式教程（临时没找到好的教程，自行去了解，作为FED，不管你多么不喜欢IE，你都必须熟悉IE，因为IE是主流浏览器之一）

PS: Chrome Developer Tools视频教程在[freecodecamp](#)站点有，可以注册账号观看一下，中文站点[freecodecamp.cn](#)

二、代码编辑器（IDE）

Java Web项目开发的话，MyEclipse (eclipse) 当然算是优先选用的。至于版本问题，建议使用 MyEclipse10.7 代替 10.0 版本，效率问题，10.0的太卡。或者能选择 MyEclipse2015 也不错的。

但是对于CSS、HTML、JavaScript代码的编写，作为一名FED（Front-End Developer），你需要更快更方便的代码编辑器。

强烈推荐使用以下工具替代MyEclipse编写你的前端代码

- [Sublime Text](#) 速度快，轻且小，可以通过安装插件来满足你想要的效果，可以看官网介绍就知道他有多方便。
- [WebStorm](#) 前端神器，你想要的任何效果他都有，不过比较重，电脑配置好的或者是不想安装插件的，可以用这个。
- [Visual Studio Code \(Node.js\)](#) 的话推荐，微软出品并开源，越来越好)

等其他类似的前端代码编辑器，看个人习惯和喜好。

以上三款都支持[Zen Coding](#)，用习惯后，熟悉使用快捷键，你编码的速度会快上几倍。

教程：[Sublime Text](#)视频教程

三、文档&工具

在开发中经常使用的文档工具；下边整理的一些网站：

- [开源API文档大全](#)
- [Open 开发经验](#)
- [开源社区工具](#)
- [在线工具](#)
- [JS 代码检查工具 JSHint 或 JSVerb](#)

（很多，不一一列举，针对前端开发工程师来说，这只是小部分，详情请去了解【学习指导&资源篇】）

Chrome Dev Tool使用教程

得益于 Google v8 的快速，和对 HTML5 和 CSS3 的支持也算比较完善，HTML类的富客户端应用 在 Chrome上无论是流畅性还是呈现的效果，都是比较出色的，这对于开发者，特别是对于那些喜欢研究前沿技术的前端开发者来说，是很重要的。

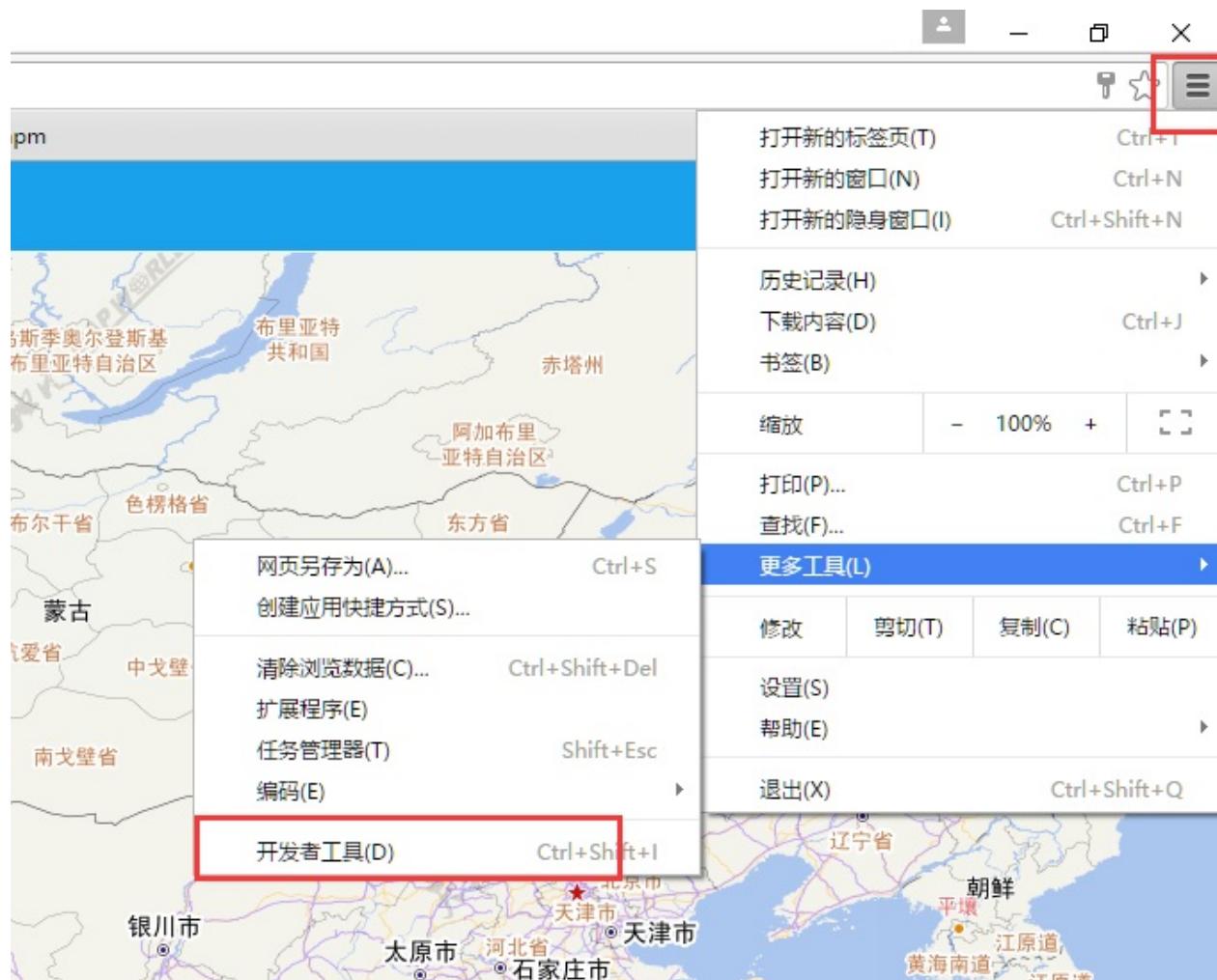
对于本文，作为一个Web开发人员，除了上面的原因以外，与我们开发相关的，就是Chrome的开发者工具。而本文，就是要详细说说Chrome的开发者工具，说说我为什么认为它比 Firebug要好用。

怎样打开**Chrome**的开发者工具？

你可以直接在页面上点击右键，然后选择审查元素：



或者在Chrome的工具中找到：

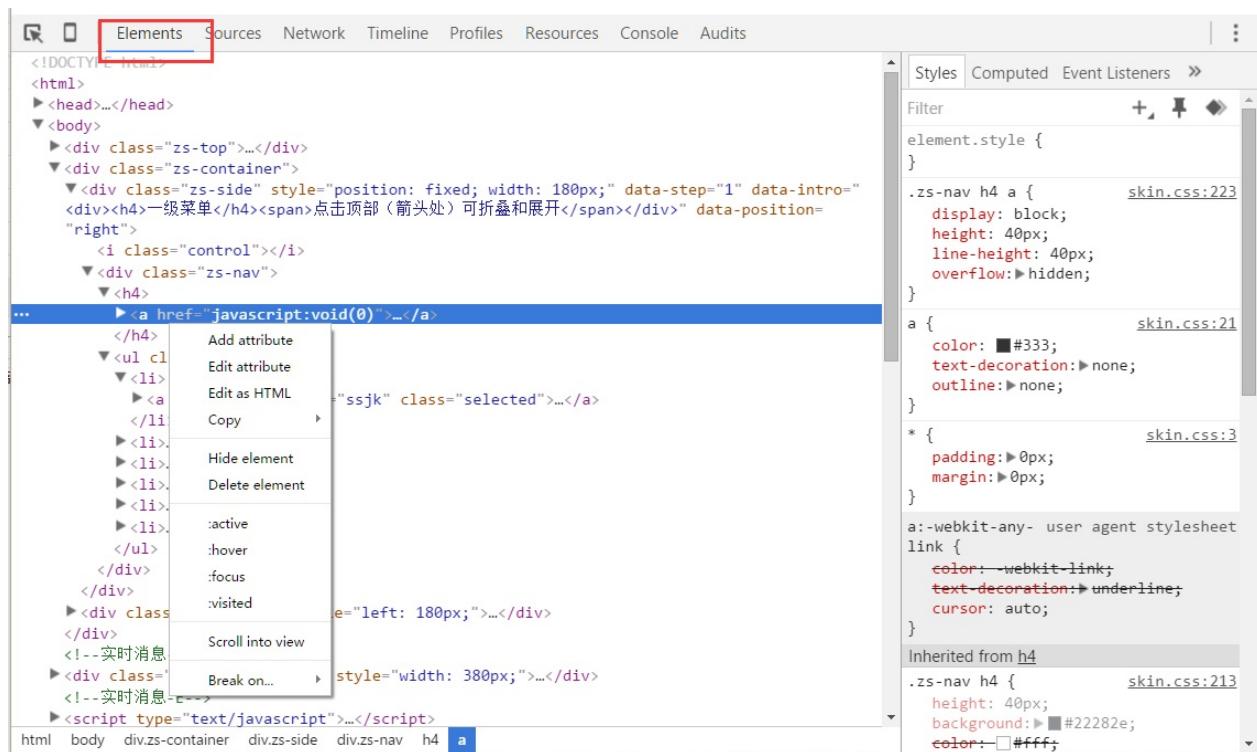


或者，你直接记住这个快捷方式：`Ctrl+Shift+I` (或者 `Ctrl+Shift+J` 直接打开控制台)，或者直接按 `F12`。

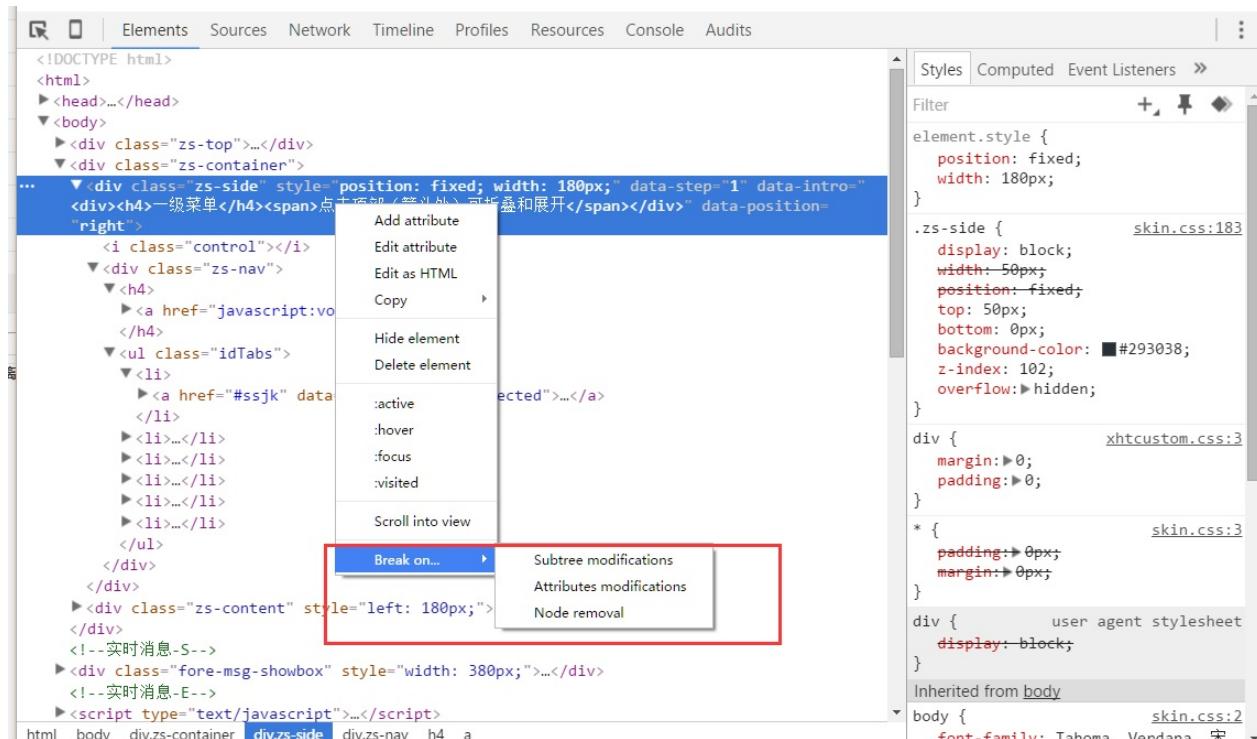
下面来分别说下每个Tab的作用。

Elements 标签页

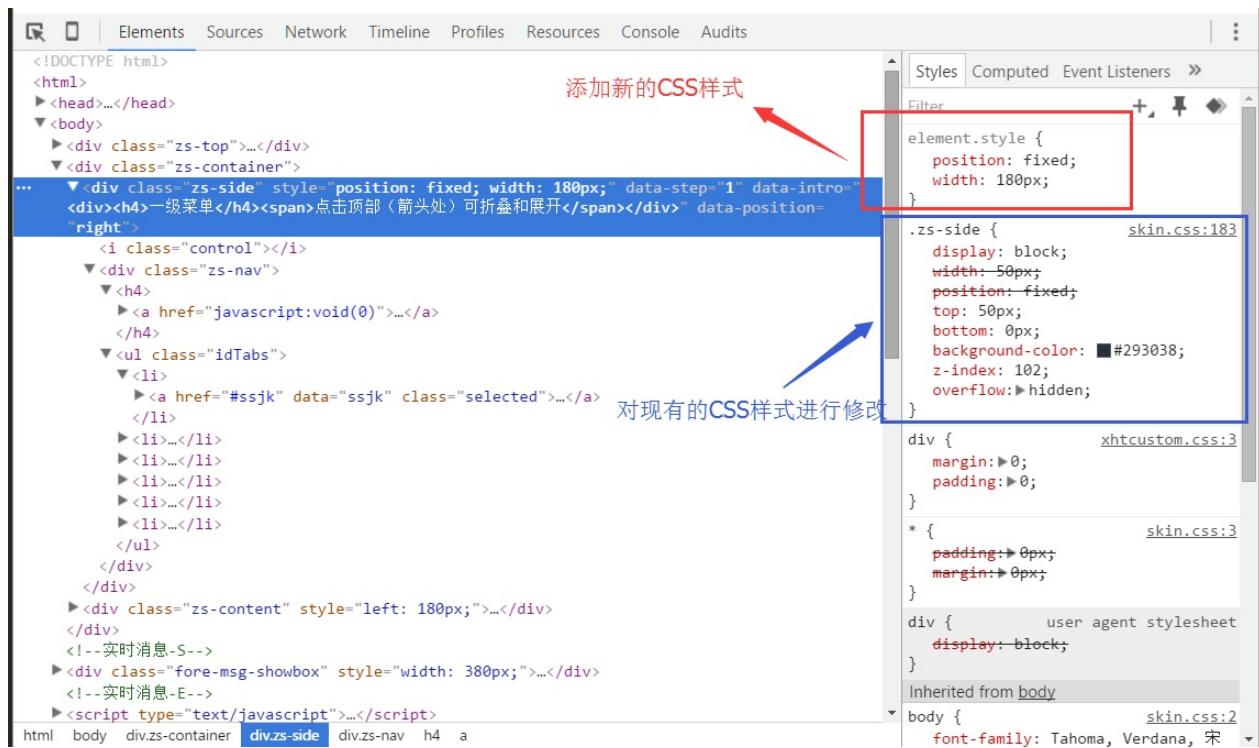
这个就是查看、编辑页面上的元素，包括**HTML**和**CSS**：



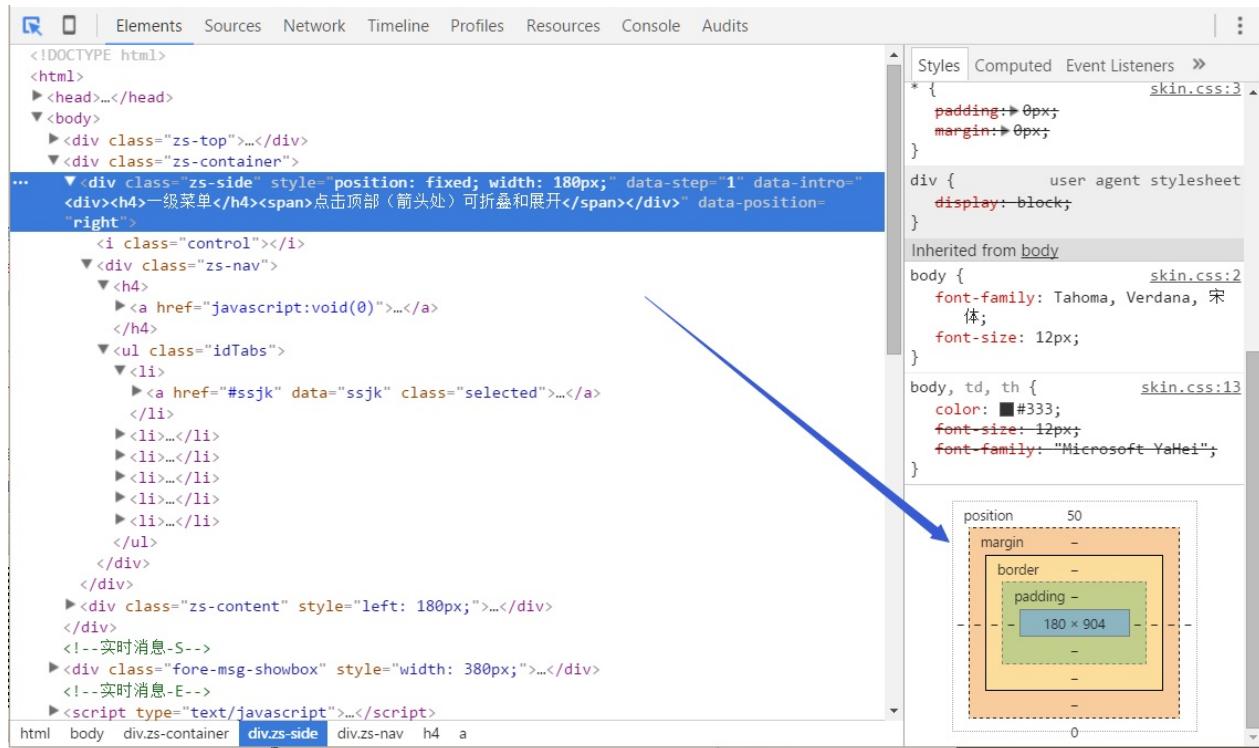
左侧就是对页面HTML结构的查看与编辑，你可以直接在某个元素上双击修改元素的属性，或者你点右键选>Edit as Html"直接对元素的HTML进行编辑，或者删除某个元素，所有的修改都会即时在页面上得到呈现。你还可以对某个元素进行监听，在JS对元素的属性或者HTML进行修改的时候，直接触发断点，跳转到对改元素进行修改的JS代码处：



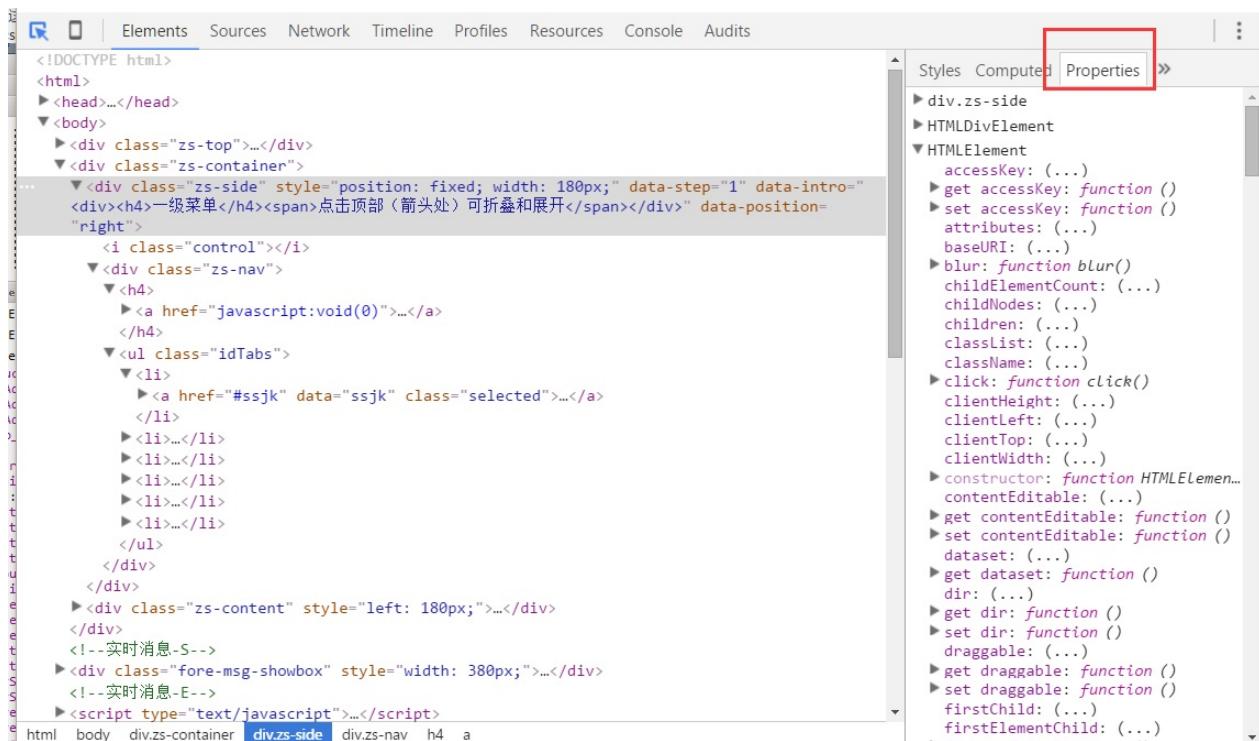
Elements标签页的右侧可以对元素的 **CSS**进行查看与编辑修改：



你还可以通过这里看到各CSS选择器设置的css值的覆盖情况。下面的Metrics可以看到元素占的空间情况（宽、高、Padding、Margin神马的）：

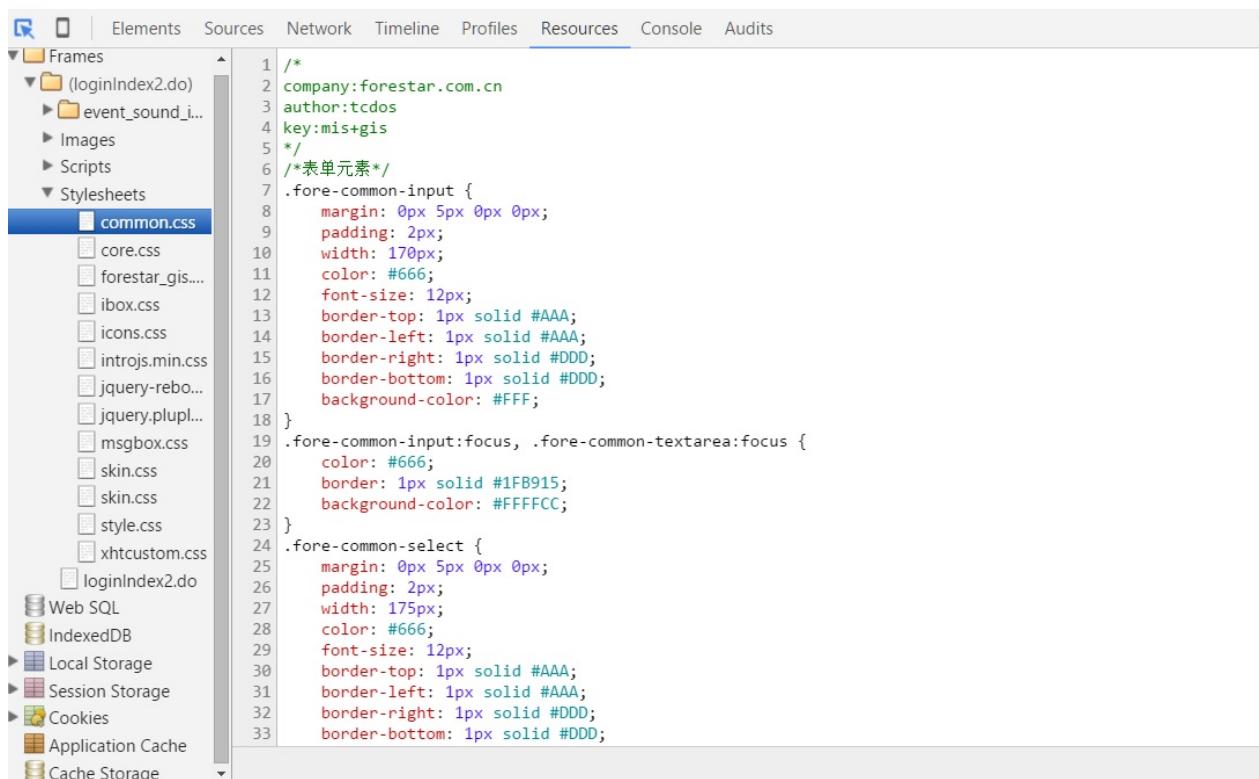


注意到上面的Properties没有？这个很有用哦，可以让你看到元素具有的方法与属性，比查API手册要方便得多哦（要注意某些方法和属性在IE、FireFox等其他浏览器下面的支持情况哦）。



Resources 标签页

Resources 标签页可以查看到请求的资源情况，包括 CSS、JS、图片等的内容，同时还可以查看到存储相关的如 Cookies、HTML5 的 database 和 LocalStore 等，你可以对存储的内容编辑和删除。这里的CSS文件有一个好玩的特性，你可以直接修改 css 文件，并且修改即时生效哦：



Network 标签页

Network 标签页对于分析网站请求的网络情况、查看某一请求的请求头和响应头还有响应内容很有用，特别是在查看 Ajax 类请求的时候，非常有帮助。注意是在你打开 Chrome 开发者工具后发起的请求，才会在这里显示的哦。点击左侧某一个具体去请求 URL，可以看到该请求的详细 HTTP 请求情况：

The screenshot shows the Network tab in the Chrome Developer Tools. A specific request for '2016-01-21' is selected. A red box highlights the 'Headers' section, which displays the following details:

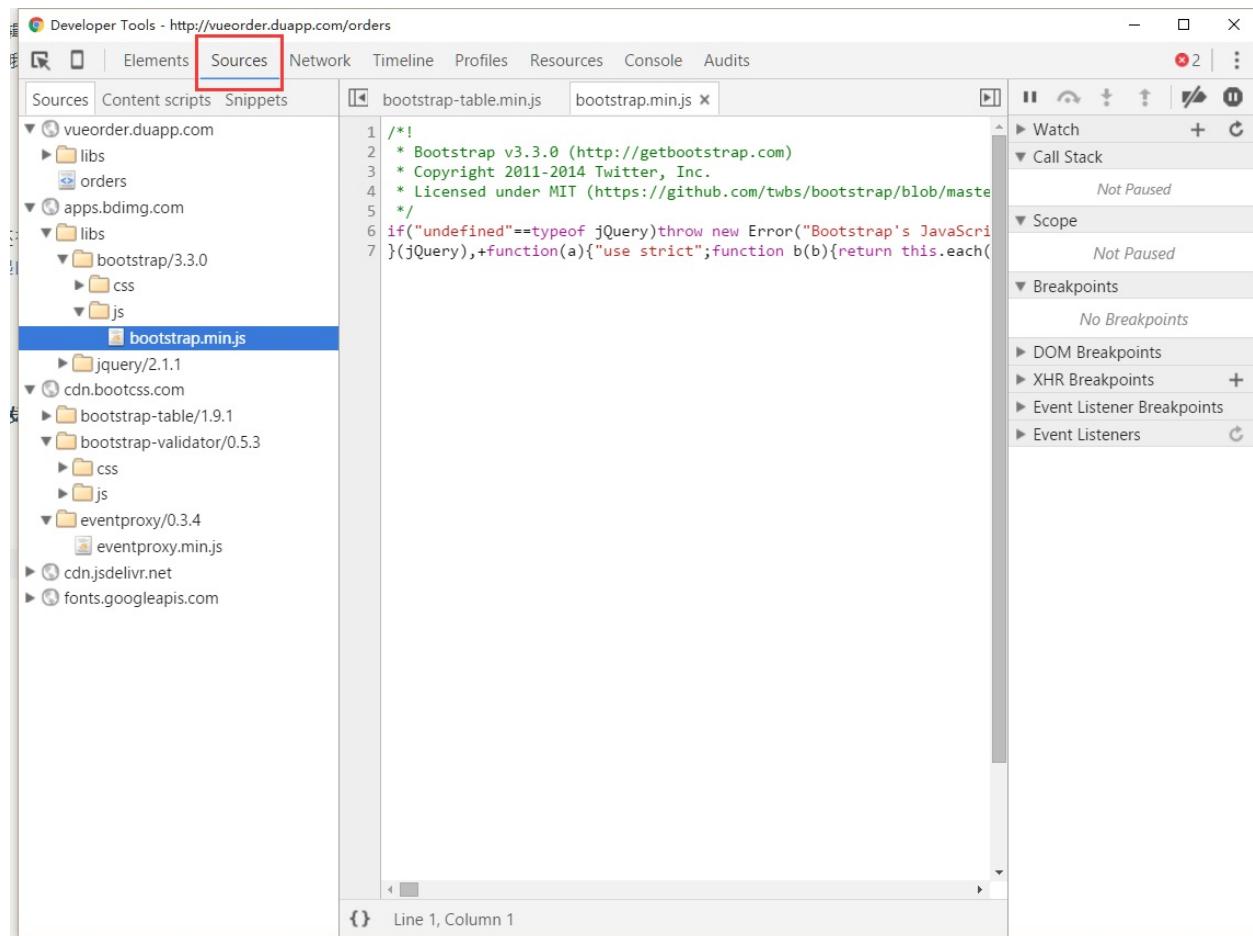
- General**
- Response Headers** (view source)
 - Cache-Control: no-cache
 - Date: Thu, 21 Jan 2016 12:01:30 GMT
 - ETag: W/"601-1q6CxyEYrs3b53THm9XQyA"
 - Server: BWS/1.0
 - X-Powered-By: Express
- Request Headers** (view source)
 - Accept: */*
 - Accept-Encoding: gzip, deflate, sdch
 - Accept-Language: zh-CN,zh;q=0.8
 - Connection: keep-alive
 - Cookie: BAEID=BE5FC4381FC6ECFEA4E88291AE0CAA66:FG=1; vueorder_cookie=s%3AwqW70Vt6dH-C1pgb1y37BSuYzvB8phk0.KSzN60mmLt%2BFKYLf4X2rwn5q%2FkxEJ5TiSU3wNGu3Y
 - Host: vueorder.duapp.com
 - If-None-Match: W/"601-1q6CxyEYrs3b53THm9XQyA"
 - Referer: http://vueorder.duapp.com/orders
 - User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.106 Safari/537.36
 - X-Requested-With: XMLHttpRequest

At the bottom left, it says '1 requests | 169 B transferred'.

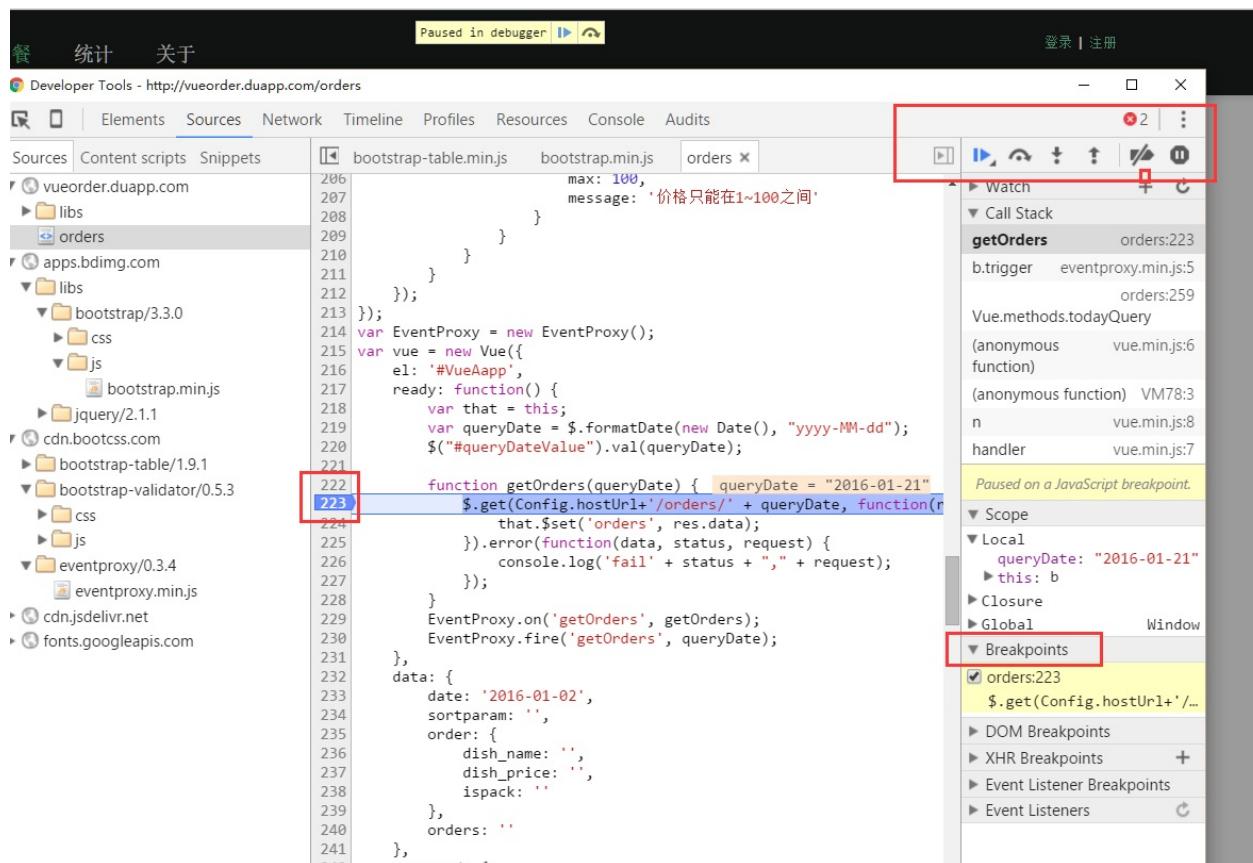
我们可以在这里看到 HTTP 请求头、HTTP 响应头、HTTP 返回的内容等信息，对于开发、调试，都是很有用的。

Sources 标签页

很明显，这个标签页就是查看 JS 文件、调试 JS 代码的，直接看图：

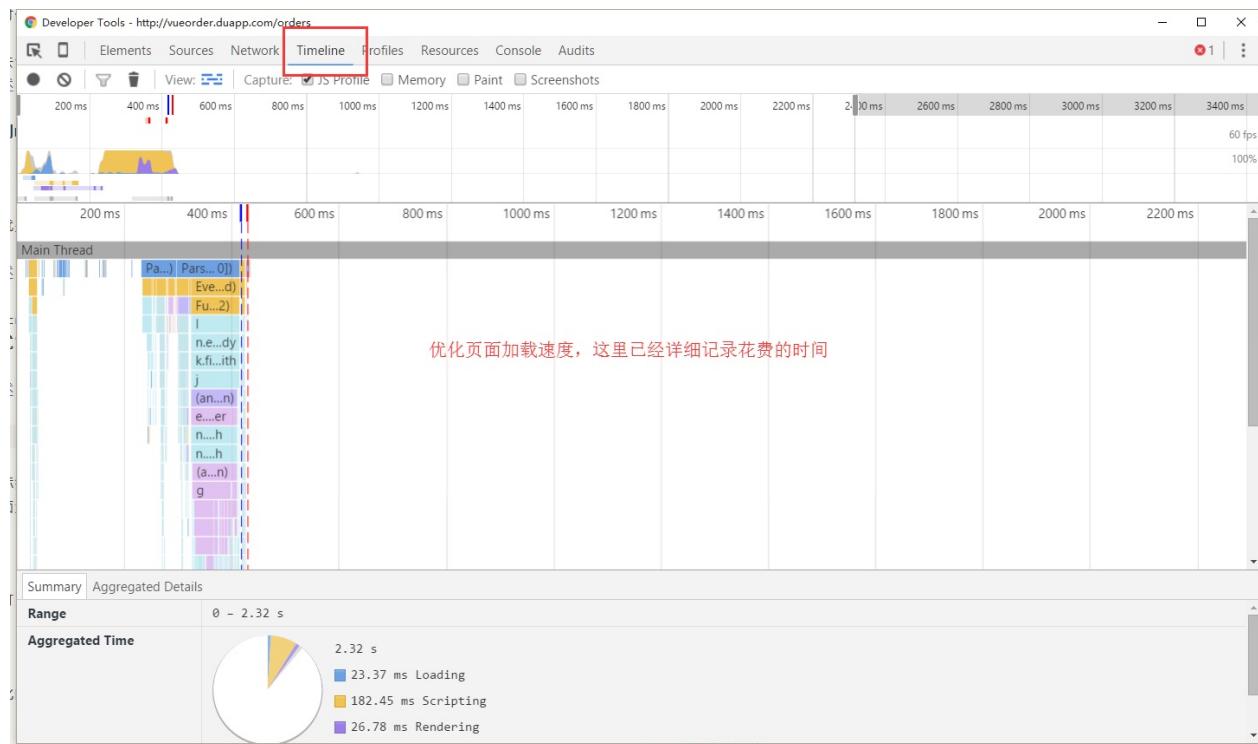


还有你可以打开Javascript控制台，做一些其他的查看或者修改，你甚至还可以为某一XHR请求或者某一事件设置断点（这是前端调试常用，必须熟练的）：



Timeline 标签页

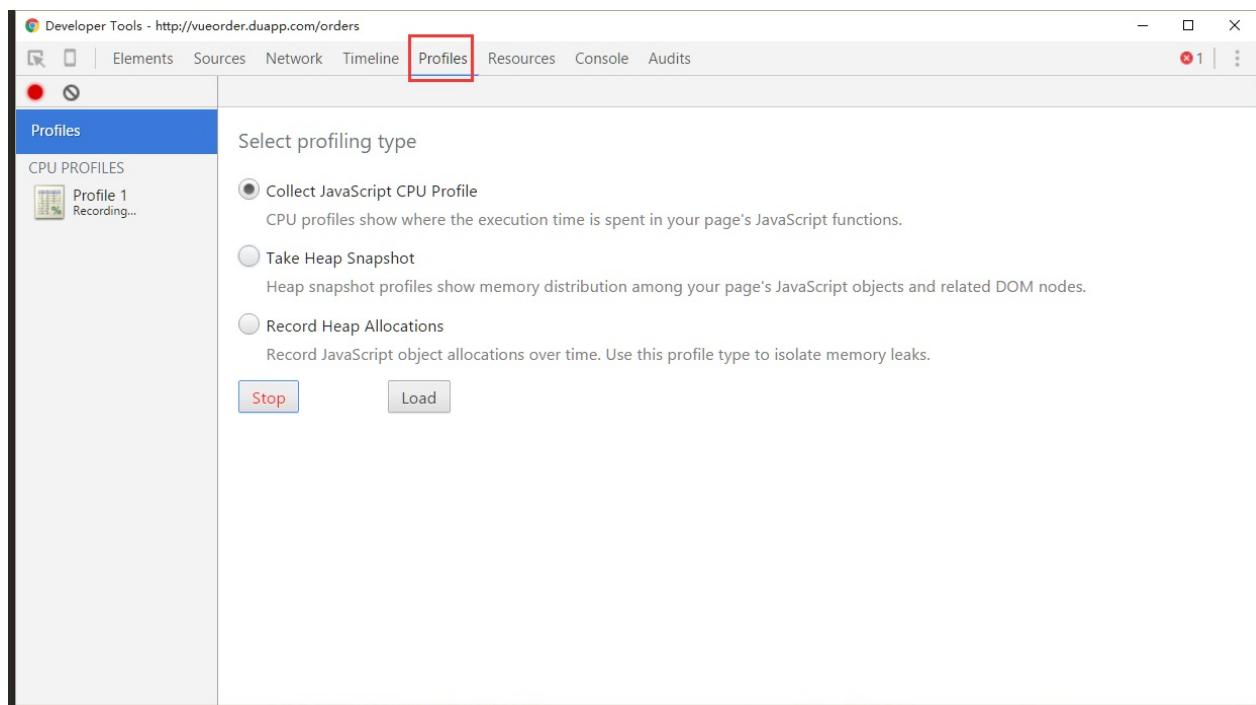
注意这个 Timeline 的标签页不是指网络请求的时间响应情况哦（这个在 Network 标签页里查看），这个 Timeline 指的是 JS 执行时间、页面元素渲染时间，对页面或者前端优化的时候，这个是常用 tab，详细使用教程可以自行搜索或者看官方介绍文档：



点击底部的 Record 就可以开始录制页面上执行的内容。（这个不熟悉，请参考文末链接）

Profiles 标签页

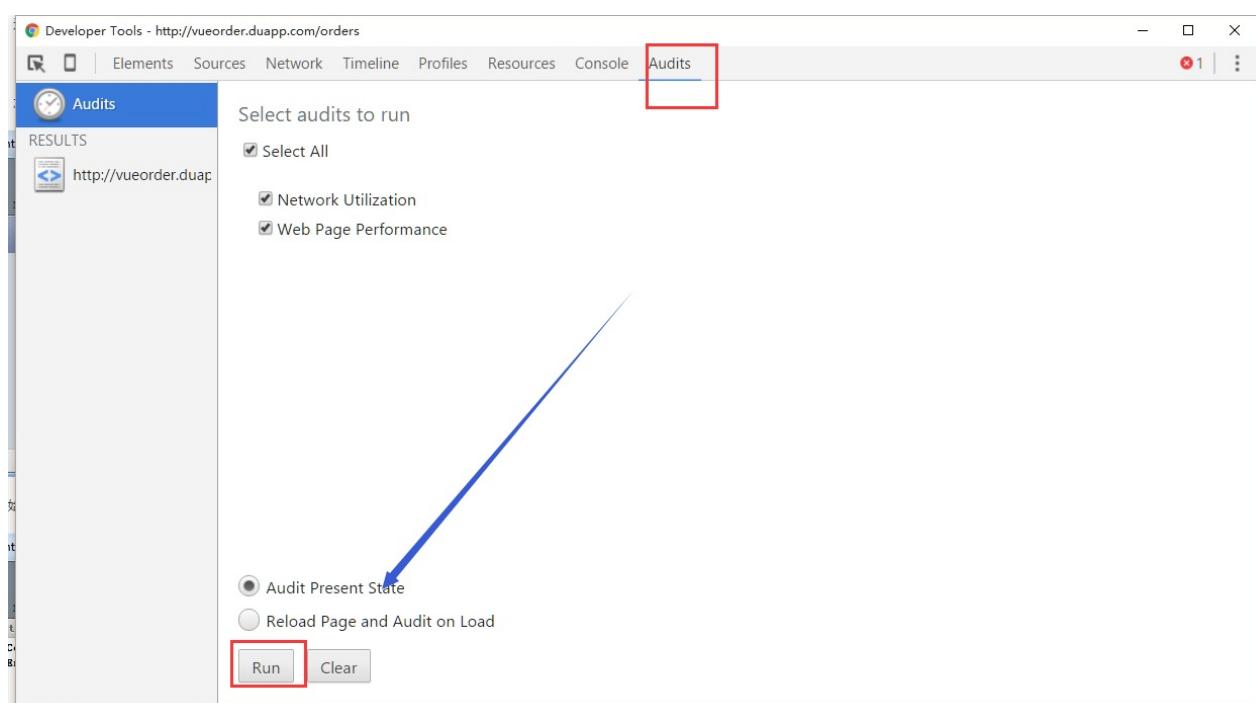
这个主要是做性能优化的，包括查看 CPU 执行时间与内存占用：



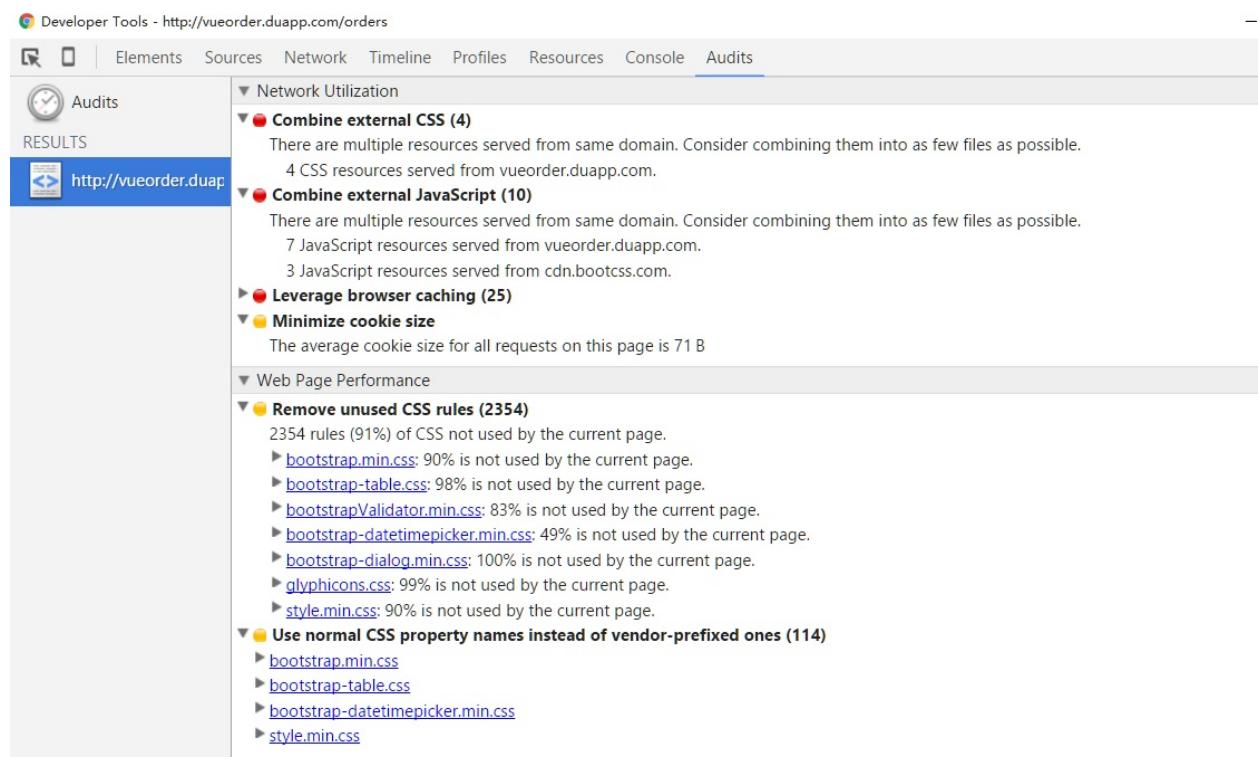
详细使用请自行搜索教程或者看官方文档。

Audits 标签页

这个对于优化前端页面、加速网页加载速度很有用哦（相当与Yslow），点击run按钮，就可以开始分析页面，分析完了就可以看到分析结果了：



它甚至可以分析出页面上样式表中有哪些CSS是没有被使用的哦：

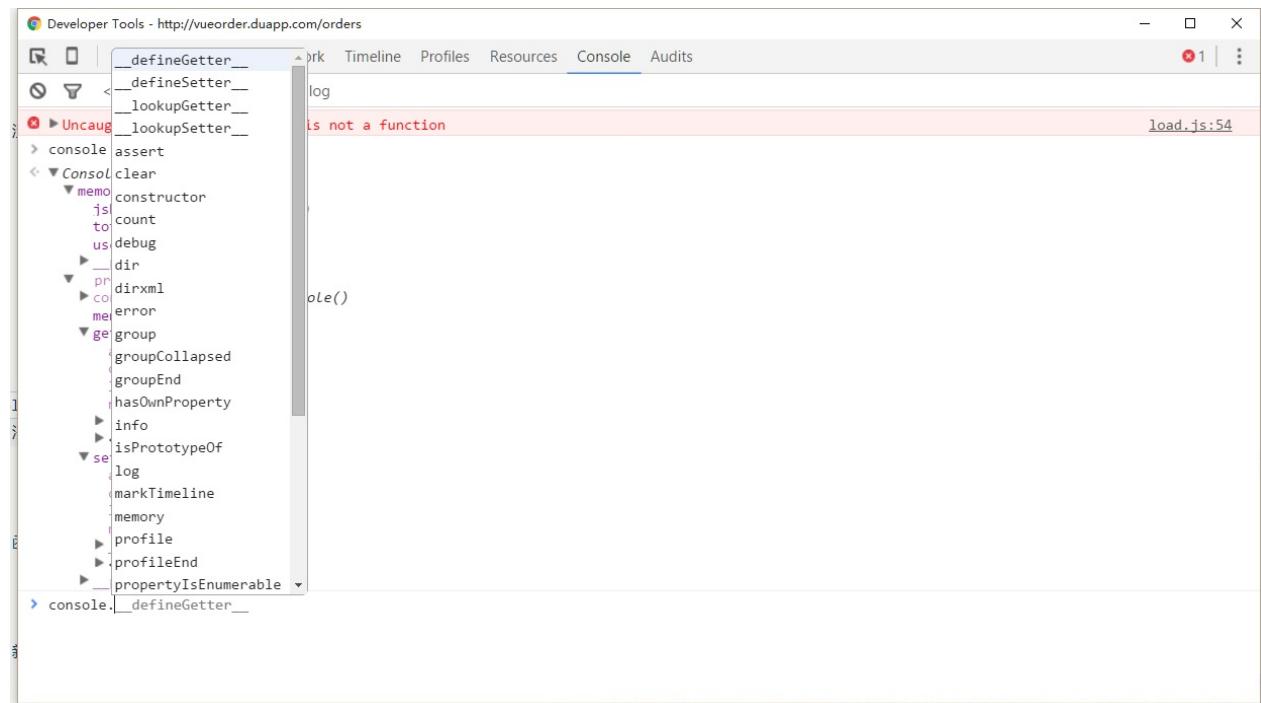


Console 标签页

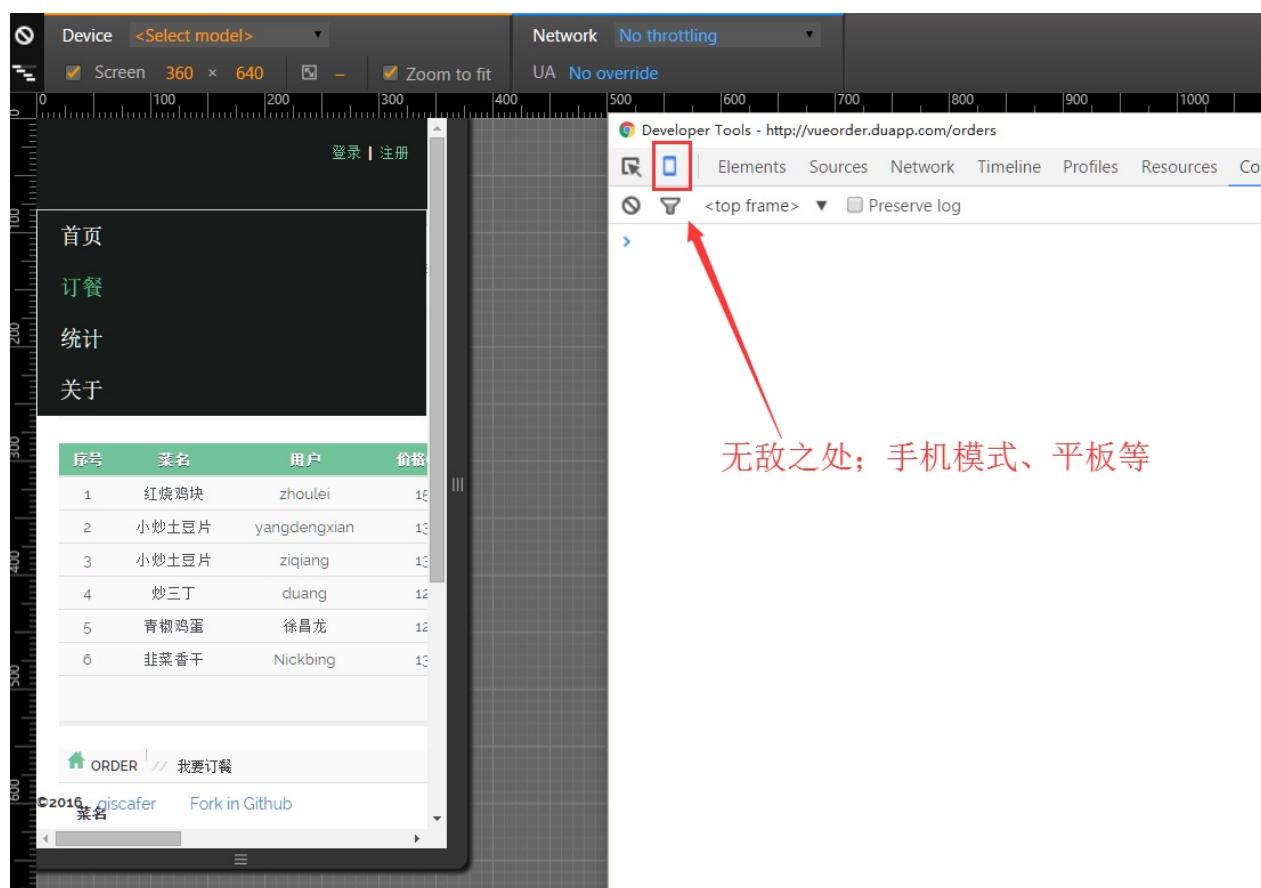
就是 Javascript 控制台了：



这个除了查看错误信息、打印调试信息（`console.log()`）写一些测试脚本以外，还可以当作 Javascript API 查看用。例如我想查看 `console` 都有哪些方法和属性，我可以直接在 Console 中输入 "console" 或者想查看都有哪些方法：



移动设备



结语

Google Chrome除了简洁、快速，现在的Chrome的插件也非常的丰富了。而对于web开发者来说，Chrome对于HTML5、CSS3等一些新标准的支持也是比较完善的，而且Chrome的开发者工具我个人认为真的非常好用，这就是为什么我向web开发者推荐使用Chrome的原因，另外开发中，需要了解浏览器之间的一些差异性，或者低版本和高版本的差异性，比如IE一般都的是非主流形式，你就必须了解IE。避免开发的过程中出现IE不兼容的情况等。

注1：本文截图的Chrome版本为 47.0.2526.106，版本不一样的话可能截图会出现差异。

注2：Chrome开发者工具更详细的说明请参考：<http://code.google.com/intl/zh-CN/chrome/devtools/docs/overview.html>（需要墙）

注3：本文参考《[Chrome浏览器超强调试工具](#)》文章进行重新截图和修改。

awesome-github

目录

- 教程
 - [通用教程](#)
 - [Git使用](#)
 - [GitHub Pages](#)
 - [GitBook](#)
 - [GitHub API](#)
 - [Travis CI](#)
- 文章
- 网站
 - 常用网站
 - [GitHub Rank](#)
 - [Star管理](#)
- 工具
 - 常用工具
 - 桌面工具
 - [App](#)
 - 插件
 - 命令行
- 项目
 - 项目
 - 库
- 其他的awesome

教程

通用教程

- [《Github 帮助文档》 中文翻译](#) - 包含了官方文档以及其他文章
- [GitHub Guides](#) - 官方的GitHub使用指引
- [GitHub 秘籍](#) - 本书为 Github 中级教程，适用在 Github 上做开源项目、制作自己的博客和协同做项目的开发者。
- [Github全程指南-如何高效使用？](#) - 作为一名开发者，Github上面有很多东西值得关注学习，可是刚刚接触github，怎样一步步学习使用Github？怎样更高效的利用Github？
- [GotGitHub](#) - 全面介绍 GitHub 网站的书

- [GitHub 开发指南 - 极客学院](#) - 本指南的目的是教会你如何实际运用 Github API，本指南是 GitHub 官方文档 Development Guides 的中文翻译版本。
- [GitHub Pages 指南 - 极客学院](#) - 本指南是 GitHub Pages 官网 GitHub Pages Basics 的中文翻译版本。
- [怎样使用 GitHub？ - 知乎问题](#)
- [如何高效利用 GitHub](#)
- [gitignore - GitHub官方的.gitignore模板集合](#)
- [GitHub 漫游指南 - phodal的GitHub故事与教程](#)
- [Github MarkDown语法指南 - 使用Github必备](#)
- [git/github guide - 使用Git和GitHub的知道](#)
- [如何编辑 Wiki - 非常好的GitHub Wiki使用教程](#)
- [如何在GitHub上为开源项目做贡献？ - 14 小节课教你如何使用 GitHub，并为开源项目做贡献。](#)
- [GitHub Pull Request Tutorial - 学习提交你的第一个pr](#)
- [learn-with-open-source - 开放文档：《借助开源项目，学习软件开发》](#)
- [open_source_analysis - 借助openhub.net分析开源项目，列举了开源历史上那些的成功项目](#)
- [《GitHub入门与实践》 - 本书从Git的基本知识和操作方法入手，详细介绍了GitHub的各种功能，GitHub与其他工具或服务的协作，使用GitHub的开发流程以及如何将GitHub引入到企业中。在讲解GitHub的代表功能Pull Request时，本书专门搭建了供各位读者实践的仓库，邀请各位读者进行Pull Request并共同维护。](#)

Git使用

- [Git教程 - 廖雪峰的官方网站](#) - 史上最浅显易懂的Git教程！
- [git - 简明指南](#) - 助你入门 git 的简明指南，木有高深内容 ;)
- [常用 Git 命令清单](#) - 来自阮一峰的网络日志，列出了 Git 最常用的命令。
- [Pro Git \(中文版 \) - 书](#)
- [Git Submodule使用完整教程](#) - Git Submodule功能刚刚开始学习可能觉得有点怪异，所以本教程把每一步的操作的命令和结果都用代码的形式展现给大家，以便更好的理解。
- [Git权威指南 - 书](#)
- [git-flow 备忘清单](#) - git-flow 是一个 git 扩展集，按 Vincent Driessen 的分支模型提供高层次的库操作。
- [Git Magic -stanford出品](#)
- [Atlassian Git Tutorials - atlassian出品](#)
- [Try Git \(Interactive \) -互动性的教你使用git](#)
- [Git \(简体中文\) -archlinux出品](#)
- [Git Community Book 中文版](#) -这本书汇聚了Git社区的很多精华，其目的就是帮助你尽快的掌握Git.
- [git-recipes -高质量的Git中文教程，来自国外社区的优秀文章和个人实践](#)

- [git-it](http://jlord.us/git-it/index-zhtw.html) - GitHub一位女员工写的Git教程，繁体中文版在这里可以找到: <http://jlord.us/git-it/index-zhtw.html>
- [Git Town](#) - GitTown 定义了很多高级的 git 命令，例如 git ship / git sync 等以方便 git 的使用

GitHub Pages

- [GitHub Pages 指南](#) - 官方文档翻译版
- [jekyll官方文档中文翻译版](#) - 将纯文本转换为静态博客网站
- [搭建一个免费的，无限流量的Blog---github Pages和Jekyll入门](#) - 示范如何在github上搭建Blog，你可以从中掌握github的Pages功能，以及Jekyll软件的基本用法。更重要的是，你会体会到一种建立网站的全新思路。
- [免费使用Jekyll+Github Page搭建博客入门攻略](#) - Jekyll建站
- [hexo你的博客](#) - hexo出自台湾大学生tommy351之手，是一个基于Node.js的静态博客程序，其编译上百篇文字只需要几秒。
- [如何搭建一个独立博客——简明Github Pages与Hexo教程](#) - 这是一篇很详尽的独立博客搭建教程，里面介绍了域名注册、DNS设置、github和Hexo设置等过程。
- [Hexo 中文版](#) - hexo官网中文版
- [像 geek 一样写博客](#) - 结合了Octopress

GitBook

- [GitBook 简明教程](#) - 本教程主要围绕 GitBook 的安装，使用，集成，书籍发布，个性化以及实用插件几个方面。
- [Gitbook 入门教程](#) - 本书将简单介绍如何安装、编写、生成、发布一本在线图书,且示例全部在windows下展示(其他系统差不多一致):
- [Git教学](#) - GIT版本控制

GitHub API

- [How to Use Github's API with PHP](#) - 通过PHP如何使用GitHub API

Travis CI

- [为 iOS 建立 Travis CI](#) - 在这篇文章中，我将向你展示如何一步步的在项目中集成 Travis 。
- [Travis Ci的最接地气的中文使用教程](#) - Travis Ci的中文文档太少了，于是作者写了一篇简洁的教程

文章

- 如何高效利用GitHub - 本文尝试谈谈GitHub的文化、技巧与影响
- GitHub连击500天：让理想的编程成为习惯 - phodal对于GitHub的看法
- Github 装逼指南——Travis CI 和 codecov - 关于持续集成和统计单测覆盖率
- 如何用Github去管理你的Idea - 用Github的README.md和Issues来管理我的idea
- GitHub开源项目负责人谈开源 - Brandon就其与开源的缘分、当前工作的职责、GitHub及员工与开源的关系等方面的问题一一进行了回答。
- 亲爱的GitHub - 致GitHub的一封公开信
- thank-you-github - 一封从GitHub毕业的公开信
- 用Github issues作为blog的例子
- 2014年GitHub中国开发者年度报告 - 使用python分析数据后的报告
- Gist介绍与用法 - Gist <https://gist.github.com/> 是Github的一个子服务
- 最活跃的GitHub用户 - 想看最活跃用户可以看这里
- 10+ HELPFUL GITHUB HACKS TO IMMEDIATELY BOOST YOUR PRODUCTIVITY - 10个立即提高你生产力的GitHub技能
- Top 10 Git Tutorials for Beginners - 教你使用git最好的10本书
- 使用GitHub进行团队合作 - 译文
- 一键收藏至Github - 通过Rails 收藏文章，并自动提交至github。
- Github Hacking - Github的各种黑客技能
- 如何参与一个GitHub开源项目？ - 本文是Github官方给出的参与Github上开源项目的一些指导，对希望加入开源社区的开发者是一个不错的参考。
- 试译：开源项目成功的十条准则 - 作者将自己30年来的开发经验，总结为开源软件的十条成功法则。
- 漫谈Github与开源 - 本文作者为大二在读Geek学生关于GitHub与开源的理解。
- 关于Pull Request的十个建议 - 作者Mark Seemann
- Github上都有哪些有用但不为大家熟知的小功能？ - 知乎问题
- 如果你用GitHub，可以这样提高效率 - 基于Github，搭建一整套代码管理服务

网站

常用网站

- GitHub Trending - GitHub官方的仓库和开发者流行榜
- GitHuber.info - 最好用的GitHub人才挖掘工具
- Code Review - 利用GitHub进行codereview的网站
- GitHub Resume - 根据GitHub的信息生成简历
- GitBook - Github上写书，发布到GitBook
- choose a license - GitHub发布了choosealicense.com网站，在呼吁开源项目开发者选择一个许可证的同时，还提供了许可证的一些简要说明。

GitHub Rank

- [GitHub Rank \(China\)](#) - GitHub上中国程序员的排名网站，根据follower
- [GitHub Ranking | GitHub Awards](#) - GitHub上程序员的排名网站，根据star
- [GitHub Ranking](#) - GitHub用户和仓库排名，根据star，不区分语言

Star管理

- [My Git Star](#) - My Git Star 是一个开源项目，旨在提供个人 Github Star 管理服务，目前部署在 [Heroku](#) 之上，由 [@Sidong](#) 维护。
- [GitRep](#) - 国外免费网站，目前打算试用。就像官网介绍的那样，不仅能管理star还能发现
- [Astral](#) - 功能更加简洁。感觉不方便的地方就是需要先添加tag，才能给项目添加tag。
- [CODELF](#) - 基于Google Lovefiled，简洁快速，从开发者角度考虑，用完就走，不给开发者更多的管理负担。开源在GitHub上的链接[unbug/codelf](#)
- [gitconstellation](#) - GitHub star 管理

工具

常用工具

- [http://shields.io/](#) - 开源项目的徽章
- [Classroom for GitHub](#) - Classroom for GitHub 可以自动创建代码仓库和访问控制，可以让老师很方便的在 GitHub 上发布代码任务和收集作业。
- [Hexo](#) - 通过Github Pages写博客的Node.js框架
- [octicons](#) - GitHub的图标字体
- [markdown-editor](#) - GitHub味道的markdown编辑器
- [backup-utils](#) - backup-utils 是 Github 企业备份工具，它包括一些备份和恢复工具。这些备份工具实现了多项用于备份主机的高级功能，还原功能也已经包括在 GitHub Enterprise 中。
- [gistblog](#) - gistblog 是一个简单的 Node.js 应用，使用 Github 的认证系统和 gist 提供的后台存储来实现博客的功能。可使用 Markdown 编写博客。
- [openspace](#) - Openspace 是一个用来将你在 Github 上的项目汇总显示在一个网页里的应用。
- [primer](#) - Primer 是 Github 工具包，用于 Github 前端设计。

桌面工具

- [ohmystar](#) - Mac上管理你GitHub star的工具
- [GithubPulse](#) - OS X状态栏的APP，帮你记住你在GitHub每天的贡献
- [githubtrending](#) - OS X状态栏的APP，显示GitHub Trending，也有iOS端
- [ghstatus](#) - OS X状态栏的APP，显示GitHub Status
- [pophub](#) - OS X状态栏的APP，显示GitHub 的activities

App

- [MVVMReactiveCocoa](#) - GitBucket iOS App，一个GitHub第三方客户端
- [Monkey](#) - Monkey是一个GitHub第三方iOS客户端，主要是用来展示GitHub上的开发者的排名，以及仓库的排名。
- [react-native-gitfeed](#) - 一个React Native写的GitHub客户端，支持iOS和Android
- [githot](#) - GitHot是一个Android App,用来发现世界上最流行的项目和人
- [CodeHub](#) - CodeHub是C#写的，它是iOS设备上最好的GitHub仓库浏览和维护工具。
- [ioctocat](#) - GitHub的iOS客户端
- [napcat](#) - 一个比较全面的GitHub的iOS客户端
- [RepoStumble](#) - 查看GitHub仓库的手机客户端
- [GithubTrends](#) - Material Design风格的查看GitHub仓库trending app
- [ForkHub](#) - Android平台的GitHub客户端

插件

- [octotree](#) - 浏览器扩展，树状格式显示GitHub的代码
- [octo-linker](#) - 这款谷歌 Chrome 扩展允许您轻松地浏览 GitHub.com 上的文件和包。
- [github-hovercard](#) - GitHub Hovercard 是一个浏览器扩展，实现了展示用户在 GitHub 上信息的信息卡功能，支持 Firefox 和 Chrome 浏览器。
- [notifier-for-github-chrome](#) - 一个浏览器扩展，它能显示 Github 通知的未读数量
- [github-menu-back](#) - 一款修改 GitHub 导航栏为之前状态的 Chrome 插件
- [gitsense-extensions](#) - GitSense 是一个 Chrome 插件，可以让你在浏览 GitHub 的时候体验更好。
- [git-draw](#) - 谷歌 Chrome 扩展，给GitHub提交历史画个画
- [ShowInGitHub](#) - Xcode插件，打开选中行的GitHub提交页面
- [Reveal-In-GitHub](#) - 有关GitHub的Xcode插件
- [Visual Studio](#) - 有关GitHub的Visual Studio插件
- [github-sublime-theme](#) - GitHub Sublime 主题
- [GitHubinator](#) - sublime插件，显示选中文本上的远程GitHub仓库
- [alfred-github-workflow](#) - Alfred 2上使用GitHub命令
- [ZenHub](#) - ZenHub 能优化你的 GitHub 工作流，是轻量级的 Chrome 浏览器插件。
- [github-gmail](#) - 在Gmail内快速打开GitHub的通知
- [chrome-github-avatars](#) - 谷歌Chrome扩展，可以让你的GitHub主页显示用户的头像。
- [tab-size-on-github](#) - 谷歌Chrome和Opera扩展，让代码缩进为4个空格而不是8个
- [hide-files-on-github](#) - 谷歌Chrome和Opera扩展，隐藏点文件
- [github-highlight-selected](#) - 谷歌Chrome和Safari扩展，代码高亮，看起来像sublime
- [github-awesome-autocomplete](#) - 谷歌Chrome和Safari以及Firefox扩展，在GitHub的搜索栏加入自动补全功能
- [chrome-github-mate](#) - 谷歌Chrome扩展，下载单个文件

- [Pages2Repo](#) - 谷歌Chrome扩展，通过GitHub Pages网站就能访问仓库。
- [lovely-forks](#) - 谷歌Chrome扩展，显示fork你仓库中star最多的
- [github-pr-filter](#) - 谷歌Chrome扩展，在pr中过滤文件
- [github-ast-viewer](#) - 谷歌Chrome扩展，增加代码的抽象语法树
- [github-canned-responses](#) - 谷歌Chrome扩展，评论pr或者issue的时候有一些可选项
- [categoric](#) - 谷歌Chrome扩展，为你的通知分类
- [octo-preview](#) - 谷歌Chrome扩展，预览你评论的markdown内容
- [GifHub](#) - 谷歌Chrome扩展，GifHub一个往GitHub评论里边插入Gif动画的Chrome插件

命令行

- [gh](#) -gh 是一个用 Go 语言开发的 Github 命令行客户端。
- [node-gh](#) -Node GH 是基于 Node.js 编写的 Github 命令行工具。

项目

项目

- [github-trending](#) - 记录下GitHub历史上的每日trending
- [GitHub-Dark](#) - 黑色的GitHub网站风格
- [github-gists](#) - 拿到一个GitHub用户的所有gist
- [Get-Your-GitHub-Card](#) - 基于jquery拿到你的GitHub用户资料
- [ohmyrepo](#) - 一个 GitHub 仓库分析工具
- [greenhat](#) - 一个让GitHub全绿的“旁门左道”的东西。
- [gitfifi](#) - 滥用github提交历史
- [Github-profile-name-writer](#) - 把github提交历史变成你的名字
- [github-contributions](#) - 可以让你的 github 提交日历排出有趣的图案
- [github-corners](#) - 显示 "Fork me on GitHub"
- [GitHub-jQuery-Repo-Widget](#) - 一个GitHub风格的挂件，方便在页面中展示GitHub项目
- [GitHub Archive](#) - GitHub Archive 是一个记录GitHub时间线的项目
- [github-cards](#) - GitHub Cards 用来展示你的简介
- [githut](#) - 可视化了GitHub Archive的数据，网站链接，<http://githut.info/>
- [lolcommits](#) - 每次提交Git都自拍一张
- [github-selfies](#) - Github Selfies 可以在你 Github 的需求和贡献上加上你的自拍照。
- [badges](#) - 收集GitHub上readme页显示的与javascript有关的各种徽章

库

- [octokit](#) - GitHub API的官方封装库
- [GitHub Java API \(org.eclipse.egit.github.core\)](#) - eclipse出品，Java写的GitHub API的封

装库

- [github - michael](#) - JavaScript写的GitHub API的封装库
- [PyGithub](#) - Python的GitHub API封装库
- [UAGithubEngine](#) - Objective-C的GitHub API封装库
- [RxGitHubAPI](#) - 基于RxSwift的GitHub API封装库
- [GitHub API for Java](#) - 面向对象的GitHub API库
- [GitHubObjC](#) - Objective-C实现的GitHub API库
- [go-github](#) - Go实现的GitHub API库
- [ruby-github](#) - Ruby实现的GitHub API库

其他的awesome

- [awesome-github](#) - [phillipadsmith](#)的awesome-github
- [awesome-browser-extensions-for-github](#) - GitHub浏览器扩展收集列表
- [github-cheat-sheet](#) - 一些酷酷的Git和GitHub功能收集

License



文章版权采用[CC Attribution-NonCommercial](#) 中文：署名-非商业性使用协议。

awesome-github 是[coderyi](#)创建的，现在由他和AntBranch组织维护，也欢迎每一个人加入进来。

copyright (c) 2016 coderyi.all rights reserved.

序言

列举一下好书，推荐大家抽空读一下

书单

说明：破折号后边文字是个人简单加的注释，可能出现个人见解不同；另外，未阅读过书籍的朋友，也不要被加的描述干扰，书籍内容肯定会比描述的精彩且有用

《代码整洁之道》——按作者说的去做，你就能写出高质量和优雅的代码

《重构-改善既有代码的设计》——如果从写码一开始就设计好你的方法或者类，就减少了维护时间和项目成本，提高了代码质量和开发效率，你就是优秀的程序员

[更多免费的中文书索引](#)

程序员应该阅读的非编程类书籍有哪些

在 stackoverflow 上有人提问 [程序员应该阅读的非编程类书籍有哪些？](#) 本来只想整理编程类书籍，不过突然眼前一亮，发现了《The Art of War - Sun Tzu》回答者的推荐说明引用 Wikipedia 上的：

亚马逊提供免费的 Kindle 版读本：[孙子兵法](#)

Much of the text is about how to fight wars without actually having to do battle: it gives tips on how to outsmart one's opponent so that physical battle is not necessary. As such, it has found application as a training guide for many competitive endeavors that do not involve actual combat.

This knowledge would surely be useful in the everyday "battles" we have to fight in and out of the office. It's also filled with quotes you can impress your fellow programmers with... :)

《哥德尔、艾舍尔、巴赫——集异璧之大成》 **Gödel, Escher, Bach: an Eternal Golden Braid,**

这本书通常被称为 [《GEB》](#)，它绝对是一本神书，一本奇书，一本神奇的书。在豆瓣读书的科普类排名中稳居第一。我在博客中，和即将出版的书中，也一而再，再而三的提及此书。

书有点儿厚，而且价格不菲，大概五六十吧。我也曾经不止一次的向朋友们推荐此书，并赠书此书。

作者也乘中文版出版之际，为自己取了一个雅致的汉名——侯世达（Douglas Richard Hofstadter）。侯世达应该是 Hofstadter 的音译。

如果你喜爱理科，此书必读。如果你是文科，那就读读《银河系漫游指南》。

《银河系漫游指南》**The Hitchhiker's Guide to the Galaxy**

亚马逊翻译为 [《银河系搭车客指南》](#)，略带喜感。

突如其来的寂静笼罩了地球。

这事实上比噪音更加可怕。

有一会儿，什么也没有发生。

巨大的飞船一动不动地挂在空中，覆盖了地球上的每个国家。

在黯然退场之前，地球首先被改造成了最终极的声音重放器件，这是有史以来建造过的最伟大的播音系统。

但伴之而来的不是演奏会，不是音乐，没有开场号曲，而仅仅是一条简短的信息。

“地球人，请注意了。”一个声音说，这声音堪称完美，仿佛来自四声道系统，完美得无懈可击，失真度低得能让勇敢的男人洒下眼泪。

“这里是银河超空间规划委员会。诸位无疑已经知道，银河系边远地区的开发规划要求建造一条穿过贵恒星系的超空间快速通道，令人遗憾的是，贵行星属于计划中预定毁灭的星球之一。毁灭过程将在略少于贵地球时间两分钟后开始。谢谢合作。”

《人性的弱点》 How to Win Friends and Influence People

《人性的弱点 Kindle版》只售 2.9 元。

《人性的弱点》的作者戴尔·卡耐基，美国“成人教育之父”。20世纪早期，美国经济陷入萧条，战争和贫困导致人们失去了对美好生活的愿望，而卡耐基独辟蹊径地开创了一套融演讲、推销、为人处世、智能开发于一体的教育方式，他运用社会学和心理学知识，对人性进行了深刻的探讨和分析。《人性的弱点》讲述的许多普通人通过奋斗获得成功的真实故事，激励了无数陷和迷茫和困境的人，帮助他们重新找到了自己的人生。

接受卡耐基教育的有社会各界人士，其中不乏军政要员，甚至包括几位美国总统。千千万万的人从卡耐基的教育中获益匪浅。

《人性的弱点》汇集了卡耐基的思想精华和最激动人心的内容，是作者最成功的励志经典，出版后立即获得了广大读者的欢迎，成为西方世界最持久的人文畅销书。无数读者通过阅读和实践中介绍的各种方法，不仅走出困境，有的还成为世人仰慕的杰出人士。只要不断研读《人性的弱点全集》，相信你也可以发掘自己的无穷潜力，创造辉煌的人生。

《别逗了，费曼先生!》 Surely You're Joking, Mr. Feynman!

《别逗了，费曼先生》是一本很棒的读物：挥霍无忌、惊世骇俗，却仍然温馨，很有人情味儿。

R·P·费曼，他因量子电动力学方面的研究荣获诺贝尔物理学奖。除了作为一个物理学家外，费曼在不同时期还曾是故事大王、艺术家、鼓手和密码破译专家。

“费曼的一生，或可比作连锁反应。从一点儿临界质量的灰质开始，这个生命向四面八方炸开，产生出热和光。”——《时代》

“费曼以其才华和怪癖，在他的同事们中间，成了一个传奇人物——您在阅读本书的时候，不从头笑到尾，是很难的。”——《新闻周刊》

“眉飞色舞，肆意笑闹……费曼的语言，生动活泼，直率真朴——真正令人耳目一新。”
——《芝加哥太阳报》

“如果您以为物理学或物理学家中间没有什么乐子——那么来会会费曼吧——一个用一团原子变戏法的最令人捧腹的伙计。”——《联合日报》

“科学家都是枯燥无味之人，这样一种老生常谈，一本书就能打破，这本书就是。”——
《底特律自由报》

《尽管去做》 Getting Things Done

如果你增加听说过一个词——GTD，没错，就是这本书 Getting Things Done，还有一种译法是《搞定1:无压工作的艺术》也很信、达，至于雅嘛，呵呵。

在今天这个信息量和工作量倍增的世界，一些老的工作方法已经失去了效用。每一个职场中人或多或少都有这样的体验：压力重重；太多事情都理不清头绪；似乎永远被各种任务和目标追赶着……

时间管理大师戴维·艾伦将指导你走出规划和执行工作中的泥沼，通向高效、轻松的彼岸。要想让事情井井有条，关键便是——从容、放松。

《别让我思考》 Don't Make Me Think

先推荐一篇知乎上的文章：[Chrome 浏览器的哪些设计符合「Don't make me think」原则？](#)

在豆瓣和亚马逊搜索了很久中文版，居然没找到，不得已求助 Google，原来被翻译成了《[点石成金:访客至上的网页设计秘笈](#)》。

如果你在进行网站设计，为网站编程，或者管理网站，那么一定要读一读此书。

- 有些网站看起来很杂乱；
- 有些网站能让你轻松地找到资料；
- 有些网站让你犹如置身迷宫，

为什么网站的可用性会有如此大的反差？用户在访问网站时有怎样的心理？遵循什么样的原则来设计网站才能吸引访客？这本全球 Web 设计人员的必读经典会给出答案。

《禅与摩托车维修艺术》Zen and the Art of Motorcycle Maintenance

这是什么书？

[《禅与摩托车维修艺术》](#)：累积销量超过一千万册，美国大学“禅与现代美国文学”课程的必读参考书。

70年代的梭罗——罗伯特·M. 波西格，《时代》周刊评选20世纪70年代十本最有影响力的书之一。

《禅与摩托车维修艺术》主要内容简介：在一个炎热的夏天，父子两人和约翰夫妇骑摩托车从明尼苏达到加州，跨越美国大陆，旅行的过程与一个青年斐德洛研修科学技术与西方经典，寻求自我的解脱，以及探寻生命的意义的过程相互穿插。

一路上父亲以一场哲学肖陶扩的形式，将见到的自然景色，野外露营的经历，夜晚旅店的谈话，机车修护技术等等日常生活与西方从苏格拉底以来的理性哲学的深入浅出的阐述与评论相结合，进行了对形而上学传统的主客体二元论的反思，以及对科学与艺术，知识与价值，古典主义与浪漫主义，精神与物质，机械论与神秘主义，西方与东方等西方二分法划分下的事物间的关系的思考。

并潜入自己的过去，探寻在现代文明下自己精神的分裂的起源，完成了一次自我心灵与人类文明的探索。

《编码宝典》(Cryptonomicon)

貌似没有中国版。

免费的编程中文书籍索引

- 国外程序员在 [stackoverflow](#) 推荐的程序员必读书籍，[中文版](#)。
- [stackoverflow](#) 上的程序员应该阅读的非编程类书籍有哪些？[中文版](#)
- [github](#) 上的一个流行的编程书籍索引 [中文版](#)

目录

- 语言无关类
 - 操作系统
 - 智能系统
 - 分布式系统
 - 编译原理
 - 函数式概念
 - 计算机图形学
 - WEB服务器
 - 版本控制
 - 编辑器
 - NoSQL
 - PostgreSQL
 - MySQL
 - 管理和监控
 - 项目相关
 - 设计模式
 - Web
 - 大数据
 - 编程艺术
 - 其它
- 语言相关类
 - Android
 - APP
 - AWK
 - C/C++
 - C#
 - Clojure
 - CSS/HTML

- Dart
 - Elixir
 - Erlang
 - Fortran
 - Go
 - Groovy
 - Haskell
 - iOS
 - Java
 - JavaScript
 - LaTeX
 - LISP
 - Lua
 - OCaml
 - Perl
 - PHP
 - Prolog
 - Python
 - R
 - Ruby
 - Rust
 - Scala
 - Shell
 - Swift
- 读书笔记及其它
 - 测试相关

语言无关类

操作系统

- 开源世界旅行手册
- 鸟哥的Linux私房菜
- The Linux Command Line (中英文版)
- Linux 设备驱动 (第三版)
- 深入分析Linux内核源码
- UNIX TOOLBOX
- Docker中文指南
- Docker —— 从入门到实践

- Docker入门实战
- Docker Cheat Sheet
- FreeRADIUS新手入门
- Mac 开发配置手册
- FreeBSD 使用手册
- Linux 命令行(中文版)
- Linux 构建指南
- Linux工具快速教程
- Linux Documentation (中文版)
- 嵌入式 Linux 知识库 (eLinux.org 中文版)
- 理解Linux进程
- 命令行的艺术
- SystemTap新手指南

智能系统

- 一步步搭建物联网系统

分布式系统

- 走向分布式

编译原理

- 《计算机程序的结构和解释》公开课 翻译项目

函数式概念

- 傻瓜函数编程

计算机图形学

- OpenGL 教程
- WebGL自学网

WEB服务器

- Nginx开发从入门到精通 (淘宝团队出品)
- Nginx教程从入门到精通(PDF版本，运维生存时间出品)
- OpenResty最佳实践
- Apache 中文手册

版本控制

- [Git教程](#)（本文由 廖雪峰 创作，如果觉得本教程对您有帮助，可以去 [iTunes 购买](#)）
- [git - 简易指南](#)
- [猴子都能懂的GIT入门](#)
- [Git 参考手册](#)
- [Pro Git](#)
- [Pro Git 中文版](#) (整理在[gitbook上](#))
- [Git Magic](#)
- [GotGitHub](#)
- [Git Community Book 中文版](#)
- [Mercurial 使用教程](#)
- [HgInit \(中文版\)](#)
- [沉浸式学 Git](#)
- [Git-Cheat-Sheet](#) (感谢 @flyhigher139 翻译了中文版)
- [GitHub秘籍](#)
- [GitHub帮助文档](#)
- [git-flow 备忘清单](#)
- [svn 手册](#)
- [GitHub漫游指南](#)

编辑器

- [exvim--vim 改良成IDE项目](#)
- [笨方法学Vimscript 中译本](#)
- [Vim中文文档](#)
- [所需即所获：像 IDE 一样使用 vim](#)
- [Atom飞行手册中文版](#)
- [Markdown·简单的世界](#)

NoSQL

- [NoSQL数据库笔谈 \(PDF\)](#)
- [Redis 设计与实现](#)
- [Redis 命令参考](#)
- [带有详细注释的 Redis 3.0 代码](#)
- [带有详细注释的 Redis 2.6 代码](#)
- [The Little MongoDB Book](#)
- [The Little Redis Book](#)
- [Neo4j 简体中文手册 v1.8](#)
- [Neo4j .rb 中文資源](#)

- [Disque 使用教程](#)

PostgreSQL

- [PostgreSQL 8.2.3 中文文档](#)
- [PostgreSQL 9.3.1 中文文档](#)

MySQL

- [MySQL 索引背后的数据结构及算法原理](#)
- [21分钟MySQL入门教程](#)

管理和监控

- [ELKstack 中文指南](#)
- [Mastering Elasticsearch\(中文版\)](#)
- [ElasticSearch 权威指南](#)
- [Elasticsearch 权威指南 \(中文版\)](#)
- [Logstash 最佳实践](#)
- [Puppet 2.7 Cookbook 中文版](#)

项目相关

- [持续集成 \(第二版\) \(译言网\)](#)
- [让开发自动化系列专栏](#)
- [追求代码质量](#)
- [selenium 中文文档](#)
- [Selenium Webdriver 简易教程](#)
- [Joel谈软件\)](#)
- [約耳談軟體\(Joel on Software\)](#)
- [Gradle 2 用户指南](#)
- [Gradle 中文使用文档](#)
- [编码规范](#)
- [开源软件架构](#)
- [GNU make 指南](#)
- [GNU make 中文手册](#)

设计模式

- [图说设计模式](#)
- [史上最全设计模式导学目录](#)

Web

- 关于浏览器和网络的 20 项须知
- 浏览器开发工具的秘密
- Chrome 开发者工具中文手册
- Chrome扩展开发文档
- Grunt中文文档
- gulp中文文档
- Gulp 入门指南
- 移动Web前端知识库
- 正则表达式30分钟入门教程
- 前端开发体系建设日记
- 移动前端开发收藏夹
- JSON风格指南
- HTTP 接口设计指北
- 前端资源分享（一）
- 前端资源分享（二）
- 前端代码规范 及 最佳实践
- 前端开发者手册
- 前端工程师手册
- w3school教程整理
- Wireshark用户手册
- 一站式学习Wireshark
- HTTP 下午茶
- HTTP/2.0 中文翻译
- RFC 7540 - HTTP/2 中文翻译版
- http2讲解
- 3 Web Designs in 3 Weeks

大数据

- 大数据/数据挖掘/推荐系统/机器学习相关资源
- 面向程序员的数据挖掘指南
- 大型集群上的快速和通用数据处理架构
- 数据挖掘中经典的算法实现和详细的注释
- Spark 编程指南简体中文版

编程艺术

- 程序员编程艺术

- 每个程序员都应该了解的内存知识(译)【第一部分】
- 取悦的工序：如何理解游戏 (豆瓣阅读，免费书籍)

其它

- OpenWrt智能、自动、透明翻墙路由器教程
- SAN 管理入门系列
- Sketch 中文手册
- 深入理解并行编程
- 程序员的自我修养
- Growth: 全栈增长工程师指南

语言相关类

Android

- Android Design(中文版)
- Google Material Design 正體中文版 (译本一 译本二)
- Material Design 中文版
- Google Android官方培训课程中文版
- Android学习之路
- Android开发技术前线(android-tech-frontier)
- Point-of-Android Android 一些重要知识点解析整理
- Android6.0新特性详解

APP

- Apache Cordova 开发指南

AWK

- awk程序设计语言
- awk中文指南

C/C++

- C/C++ 中文参考手册 (欢迎大家参与在线翻译和校对)
- C 语言编程透视
- C++ 并发编程指南

- Linux C编程一站式学习 (宋劲杉, 北京亚嵌教育研究中心)
- CGDB中文手册
- 100个gdb小技巧
- 100个gcc小技巧
- ZMQ 指南
- How to Think Like a Computer Scientist (中英文版)
- 跟我一起写Makefile(PDF)
- GNU make中文手册
- GNU make 指南
- Google C++ 风格指南
- C/C++ Primer (by @andycui)
- 简单易懂的C魔法
- Cmake 实践 (PDF版)
- C++ FAQ LITE(中文版)
- C++ Primer 5th Answers
- C++ 并发编程(基于C++11)
- QT 教程
- DevBean的《Qt学习之路2》(Qt5)
- C++ Template 进阶指南
- libuv中文教程
- Boost 库中文教程

C#

- MSDN C# 中文文档
- .NET 类库参考
- ASP.NET MVC 5 入门指南
- 超全面的 .NET GDI+ 图形图像编程教程
- .NET控件开发基础

Clojure

- Clojure入门教程

CSS

- 学习CSS布局
- 通用 CSS 笔记、建议与指导
- CSS参考手册
- Emmet 文档

- 前端代码规范 (腾讯 AlloyTeam 团队)
- HTML和CSS编码规范
- Sass Guidelines 中文
- CSS3 Tutorial 《CSS3 教程》
- MDN HTML 中文文档
- MDN CSS 中文文档

Dart

- Dart 语言导览

Elixir

- Elixir编程入门

Erlang

- 21天学通Erlang

Fortran

- Fortran77和90/95编程入门

Go

- Go编程基础
- Go入门指南
- 学习Go语言
- Go Web 编程 (此书已经出版，希望开发者们去购买，支持作者的创作)
- Go实战开发 (当我收录此项目时，作者已经写完第三章，如果读完前面章节觉得有帮助，可以给作者捐赠，以鼓励作者的继续创作)
- Network programming with Go 中文翻译版本
- Effective Go
- Go 语言标准库
- Revel 框架手册
- Java程序员的Golang入门指南
- Go命令教程
- Go语言博客实践
- Go 官方文档翻译
- 深入解析Go
- Go语言圣经(中文版) (GitBook)

Groovy

- 实战 Groovy 系列

Haskell

- Real World Haskell 中文版
- Haskell趣学指南

iOS

- iOS开发60分钟入门
- iOS7人机界面指南
- Google Objective-C Style Guide 中文版
- iPhone 6 屏幕揭秘
- Apple Watch开发初探
- 马上着手开发 iOS 应用程序
- 网易斯坦福大学公开课：iOS 7应用开发字幕文件

Java

- Apache Shiro 用户指南
- Jersey 2.x 用户指南
- Spring Framework 4.x参考文档
- Spring Boot参考指南 (翻译中)
- MyBatis中文文档
- MyBatis Generator 中文文档
- 用jersey构建REST服务
- Activiti 5.x 用户指南
- Google Java编程风格指南
- Netty 4.x 用户指南
- Netty 实战(精髓)
- REST 实战
- Java 编码规范
- Apache MINA 2 用户指南
- H2 Database 教程
- Java Servlet 3.1 规范
- JSSE 参考指南
- Java开源实现及最佳实践
- Java 编程要点
- Think Java

JavaScript

- [Google JavaScript 代码风格指南](#)
- [Google JSON 风格指南](#)
- [Airbnb JavaScript 规范](#)
- [JavaScript 标准参考教程 \(alpha\)](#)
- [Javascript编程指南 \(源码\)](#)
- [javascript 的 12 个怪癖](#)
- [JavaScript 秘密花园](#)
- [JavaScript核心概念及实践 \(PDF\)](#) (此书已由人民邮电出版社出版发行，但作者依然免费提供PDF版本，希望开发者们去购买，支持作者)
- [《JavaScript 模式》“JavaScript patterns”中译本](#)
- [命名函数表达式探秘](#) (注:原文由[为之漫笔翻译](#)，原始地址无法打开，所以此处地址为我博客上的备份)
- [学用 JavaScript 设计模式 \(开源中国\)](#)
- [深入理解JavaScript系列](#)
- [ECMAScript 6 入门 \(作者：阮一峰\)](#)
- [JavaScript Promise迷你书](#)
- [You-Dont-Know-JS \(深入JavaScript语言核心机制的系列图书\)](#)
- [JavaScript 教程 廖雪峰](#)
- [MDN JavaScript 中文文档](#)
- [jQuery
 - \[jQuery 解构\]\(#\)
 - \[简单易懂的JQuery魔法\]\(#\)
 - \[How to write jQuery plugin\]\(#\)
 - \[You Don't Need jQuery\]\(#\)
 - \[如何实现一个类jQuery ?\]\(#\)](#)
- [Node.js
 - \[Node入门\]\(#\)
 - \[七天学会NodeJS\]\(#\)
 - \[Nodejs Wiki Book \\(繁体中文\\)\]\(#\)
 - \[express.js 中文文档\]\(#\)
 - \[koa 中文文档\]\(#\)
 - \[一起学koa\]\(#\)
 - \[使用 Express + MongoDB 搭建多人博客\]\(#\)
 - \[Express框架\]\(#\)
 - \[Node.js 包教不包会\]\(#\)
 - \[Learn You The Node.js For Much Win! \\(中文版\\)\]\(#\)
 - \[Node debug 三法三例\]\(#\)
 - \[nodejs中文文档\]\(#\)](#)

- [orm2 中文文档](#)
- [underscore.js](#)
 - [Underscore.js中文文档](#)
- [backbone.js](#)
 - [backbone.js中文文档](#)
 - [backbone.js入门教程 \(PDF\)](#)
 - [Backbone.js入门教程第二版](#)
 - [Developing Backbone.js Applications\(中文版\)](#)
- [AngularJS](#)
 - [AngularJS最佳实践和风格指南](#)
 - [AngularJS中译本](#)
 - [AngularJS入门教程](#)
 - [构建自己的AngularJS](#)
 - [在Windows环境下用Yeoman构建AngularJS项目](#)
- [Zepto.js](#)
 - [Zepto.js 中文文档](#)
- [Sea.js](#)
 - [Hello Sea.js](#)
- [React.js](#)
 - [React.js 中文文档](#)
 - [React webpack-cookbook](#)
 - [React 入门教程](#)
 - [React Native 中文文档\(含最新Android内容\)](#)
- [impress.js](#)
 - [impress.js的中文教程](#)
- [CoffeeScript](#)
 - [CoffeeScript Cookbook](#)
 - [The Little Book on CoffeeScript中文版](#)
 - [CoffeeScript 编码风格指南](#)
- [TypeScript](#)
 - [TypeScript Handbook](#)
- [ExtJS](#)
 - [Ext4.1.0 中文文档](#)
- [Meteor](#)
 - [Discover Meteor](#)
 - [Meteor 中文文档](#)
 - [Angular-Meteor 中文教程](#)
- [Chrome扩展及应用开发](#)

LaTeX

- 一份其实很短的 LaTeX 入门文档
- 一份不太简短的 LATEX 2 ϵ 介绍 (PDF版)

LISP

- Common Lisp
 - [ANSI Common Lisp 中文翻译版](#)
 - [On Lisp 中文翻译版本](#)
- Scheme
 - [Yet Another Scheme Tutorial Scheme入门教程](#)
 - [Scheme语言简明教程](#)
 - Racket
 - [Racket book](#)

Lua

- [Lua 编程入门](#)
- [Lua 5.1 参考手册 中文翻译](#)
- [Lua 5.3 参考手册 中文翻译](#)
- [Lua 源码欣赏](#)

OCaml

- [Real World OCaml](#)

Perl

- [Modern Perl 中文版](#)
- [Perl 程序员应该知道的事](#)

PHP

- [PHP 官方手册](#)
- [PHP 调试技术手册\(PDF\)](#)
- [PHP 之道 : php-the-right-way \(@wulijun 版 PHPHub 版\)](#)
- [PHP 最佳实践](#)
- [PHP 开发者实践](#)
- [深入理解 PHP 内核](#)
- [PHP 扩展开发及内核应用](#)
- [Laravel5 中文文档](#)
- [Laravel 入门](#)

- [Symfony2 Cookbook 中文版](#)(版本 2.7.0 LTS)
- [Symfony2中文文档](#) (未译完)
- [YiiBook几本Yii框架的在线教程](#)
- [深入理解 Yii 2.0](#)
- [Yii 框架中文文檔](#)
- [简单易懂的PHP魔法](#)
- [swoole 文档及入门教程](#)
- [Composer 中文网](#)
- [Slim 中文文档](#)
- [Lumen 中文文档](#)
- [PHPUnit 中文文档](#)

Prolog

- [笨办法学Prolog](#)

Python

- [廖雪峰 Python 2.7 中文教程](#)
- [廖雪峰 Python 3 中文教程](#)
- [简明Python教程](#)
- [零基础学 Python 第一版](#)
- [零基础学 Python 第二版](#)
- [可爱的 Python](#)
- [Python 2.7 官方教程中文版](#)
- [Python 3.3 官方教程中文版](#)
- [Python Cookbook 中文版](#)
- [Python3 Cookbook 中文版](#)
- [深入 Python](#)
- [深入 Python 3](#)
- [PEP8 Python代码风格规范](#)
- [Google Python 风格指南 中文版](#)
- [Python入门教程 \(PDF\)](#)
- [Python的神奇方法指南](#)
- [笨办法学 Python \(PDF EPUB\)](#)
- [Python自然语言处理中文版](#) (感谢陈涛同学的翻译，也谢谢 @shwley 联系了作者)
- [Python 绘图库 matplotlib 官方指南中文翻译](#)
- [Scrapy 0.25 文档](#)
- [ThinkPython](#)
- [ThinkPython 2ed](#)

- [Python 快速教程](#)
- [Python 正则表达式操作指南](#)
- [python 初级教程：入门详解](#)
- [Twisted 与异步编程入门](#)
- [TextGrocery 中文 API \(基于svm算法的一个短文本分类 Python 库 \)](#)
- [Requests: HTTP for Humans](#)
- [Pillow 中文文档](#)
- [PyMOTW 中文版](#)
- [Python 官方文档中文版](#)
- [Fabric 中文文档](#)
- [Beautiful Soup 4.2.0 中文文档](#)
- [用 Python 做科学计算](#)
- [Sphinx 中文文档](#)
- [精通 Python 设计模式](#)
- [python 安全编程教程](#)
- [程序设计思想与方法](#)
- [知乎周刊·编程小白学 Python](#)
- [Scipy 讲义](#)
- [Python 学习笔记 基础篇](#)
- [Python 学习笔记 模块篇](#)
- [Python 标准库 中文版](#)
- [Python 进阶](#)
- [Python 核心编程 第二版 CPyUG 译](#)
- [Django](#)
 - [Django 1.5 文档中文版](#) 正在翻译中
 - [Django 1.7 文档中文版](#) 正在翻译中，目前只翻译了目录
 - [Django 1.8.2 文档中文版](#) 正在翻译中
 - [Django 最佳实践](#)
 - [Django 搭建简易博客教程](#)
 - [The Django Book 中文版](#)
 - [Django 设计模式与最佳实践](#)
 - [Django 网站开发 Cookbook](#)
 - [Django Girls 學習指南](#)
- [Flask](#)
 - [Flask 文档中文版](#)
 - [Jinja2 文档中文版](#)
 - [Werkzeug 文档中文版](#)
 - [Flask 之旅](#)
 - [Flask 扩展文档汇总](#)
 - [Flask 大型教程](#)

- [SQLAlchemy 中文文档](#)
- [web.py](#)
 - [web.py 0.3 新手指南](#)
 - [Web.py Cookbook 简体中文版](#)
- [Tornado](#)
 - [Introduction to Tornado 中文翻译](#)
 - [Tornado源码解析](#)
 - [Tornado 4.3 文档中文版](#)

R

- [R语言忍者秘笈](#)

Ruby

- [Ruby 风格指南](#)
- [Rails 风格指南](#)
- [笨方法學 Ruby](#)
- [Ruby on Rails 指南](#)
- [Ruby on Rails 實戰聖經](#)
- [Ruby on Rails Tutorial 原书第 3 版](#) (本书网页版免费提供，电子版以 PDF、EPub 和 Mobi 格式提供购买，仅售 9.9 美元)
- [Rails 实践](#)
- [Rails 5 开发进阶\(Beta\)](#)
- [Rails 102](#)
- [编写Ruby的C拓展](#)
- [Ruby 源码解读](#)
- [Ruby中的元编程](#)

Rust

- [rust book 中文翻译](#)
- [rust primer](#)

Scala

- [Scala课堂 \(Twitter的Scala中文教程\)](#)
- [Effective Scala \(Twitter的Scala最佳实践的中文翻译\)](#)
- [Scala指南](#)

Shell

- [Shell脚本编程30分钟入门](#)
- [Bash脚本15分钟进阶教程](#)
- [Linux工具快速教程](#)
- [shell十三问](#)
- [Shell编程范例](#)

Swift

- [The Swift Programming Language 中文版](#)
- [Swift 语言指南](#)
- [Stanford 公开课，Developing iOS 8 Apps with Swift 字幕翻译文件](#)

读书笔记及其它

- [编译原理（紫龙书）中文第2版习题答案](#)
- [把《编程珠玑》读薄](#)
- [Effective C++读书笔记](#)
- [Golang 学习笔记、Python 学习笔记、C 学习笔记 \(PDF\)](#)
- [Jsoup 学习笔记](#)
- [学习笔记: Vim、Python、memcached](#)
- [图灵开放书翻译计划--C++、Python、Java等](#)
- [蒂姆·奥莱利随笔（由译言网翻译，电子版免费）](#)
- [SICP 解题集](#)
- [精彩博客集合](#)
- [正则表达式简明参考](#)
- [中文文案排版指南](#)
- [Standard C 语言标准函数库速查 \(Cheat Sheet\)](#)
- [Git Cheatsheet Chs](#)
- [GitBook简明教程](#)
- [JavaScript语言精粹](#)
- [制造开源软件](#)
- [提问的智慧](#)
- [Markdown 入门参考](#)
- [AsciiDoc简明指南](#)
- [背包问题九讲](#)
- [老齐的技术资料](#)
- [前端技能汇总](#)
- [借助开源项目，学习软件开发](#)
- [前端工作面试问题](#)

- leetcode/lintcode题解/算法学习笔记
- 前端开发笔记本

测试相关

- 移动APP自动化测试优秀框架Appium API Reference V1.2.0 CN

License

collected from [free-programming-books-zh_CN](#)

序言

整理一些好的网站和文档，或其他有用的资料

学习路线

不知如何学？什么都懂了，不知道学啥了？应该怎么学？前端到底还需要学什么？.....
有学习疑问的都建议看一下

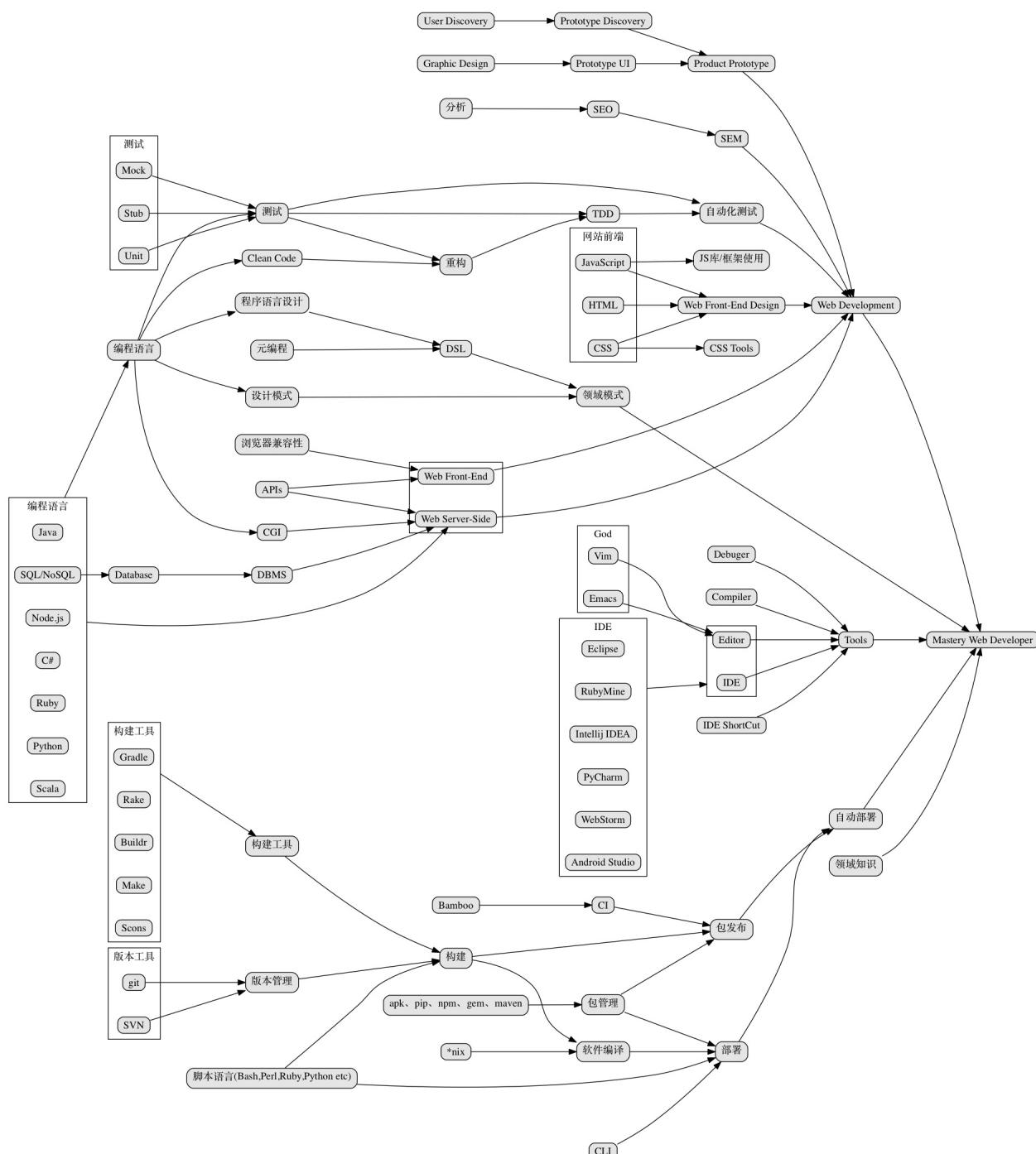
- [Web Developer 成长路线图](#)
- [前端StuQ技能图谱](#)

编程学习网站

- [stackoverflow](#)（无需解释）
- [Github](#)（借助github：阅读优秀框架源码，编写开源项目，有能力尝试去造轮子）
- [慕课网](#)
- [FreeCodeCamp](#)
- [极客学院](#)
- [汇智网](#)
- [StuQ](#)

Awesome Developer

Web Developer 成长路线图

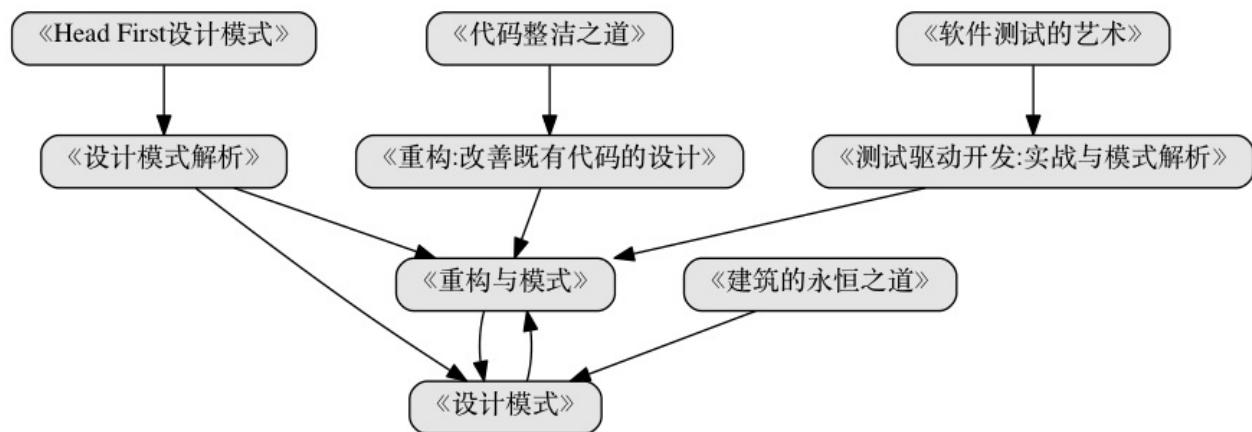


持续交付

- 《敏捷软件开发：原则、模式与实践》
- 《测试驱动开发：实战与模式解析》
- 《实现领域驱动设计》
- 《领域驱动设计：软件核心复杂性应对之道》
- 《敏捷武士：看敏捷高手交付卓越软件》

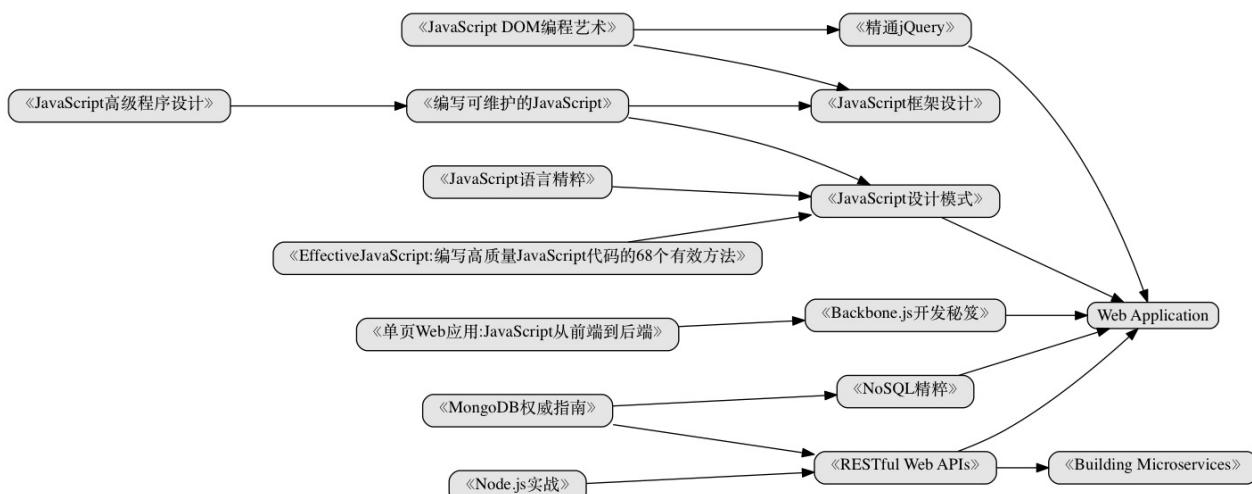
编程技能成长路线

- 《重构与模式》
- 《代码整洁之道》
- 《重构：改善既有代码的设计》

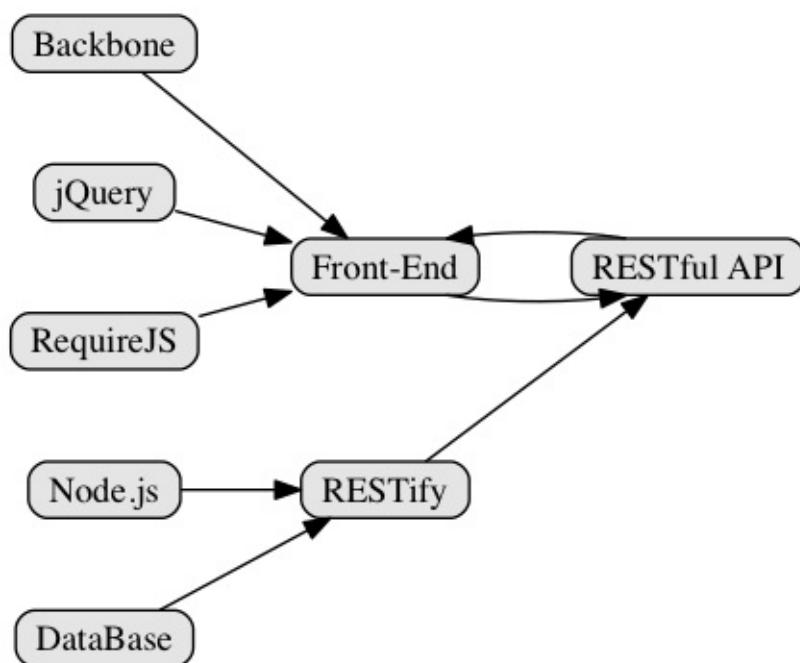


JavaScript读书路线

- 《编写可维护的JavaScript》
- 《JavaScript设计模式》
- 《EffectiveJavaScript：编写高质量JavaScript代码的68个有效方法》
- 《JavaScript语言精粹》



JavaScript Application Example



Web 杂项

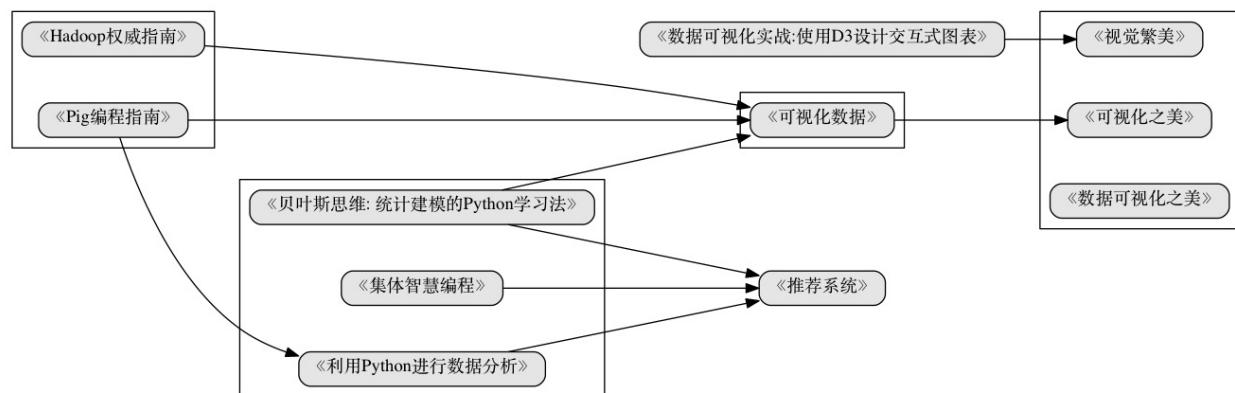
- 《实用负载均衡技术:网站性能优化攻略》
- 《网站性能监测与优化》
- 《构建高性能Web站点》
- 《SEO艺术》

架构学习路线



- 《架构之美》
- 《面向模式的软件架构》(1,4)
- 《软件框架设计的艺术》
- 《程序员必读之软件架构》
- 《架构实战：软件架构设计的过程》

机器学习读书路线



- 《贝叶斯思维:统计建模的Python学习法》
- 《利用Python进行数据分析》
- 《Python自然语言处理》
- 《机器学习:实用案例分析》
- 《驾驭文本:文本的发现、组织和处理》

License

© 2015 Phodal Huang.

各类别都列举的一些关联的知识点或者工具。

- 浏览器

- IE6/7/8/9/10/11 (Trident)
- Firefox (Gecko)
- Chrome/Chromium (Blink)
- Safari (WebKit)
- Opera (Blink)

- 编程语言

- JavaScript/Node.js
- CoffeeScript
- TypeScript

- 切页面

- HTML/HTML5
- CSS/CSS3
- Sass/LESS/Stylus
- PhotoShop/Paint.net/Fireworks/GIMP/Sketch

- 开发工具

- 编辑器和**IDE**

- VIM/Sublime Text2
- Notepad++/EditPlus
- WebStorm
- Emacs EmacsWiki
- Brackets
- Atom
- Lime Text
- Light Table
- Codebox
- TextMate
- Neovim
- Komodo IDE / Edit
- Eclipse

- Visual Studio/Visual Studio Code
- NetBeans
- Cloud9 IDE
- HBuilder
- Nuclide

- 调试工具

- Firebug/Firecookie
- YSlow
- IEDeveloperToolbar/IETester
- Fiddler
- Chrome Dev Tools
- Dragonfly
- DebugBar
- Venkman

- 版本管理

- Git/SVN/Mercurial
- Github/GitLab/Bitbucket/Gitorious/GNU Savannah/Launchpad/SourceForge/TeamForge

- 代码质量

- **Coding style**

- JSLint/JSHint/jscs
- CSSLint
- Markup Validation Service
- HTML Validators

- 单元测试

- QUnit/Jasmine
- Mocha/Should/Chai/Expect
- Unit JS

- 自动化测试

- WebDriver/Protractor/Karma Runner/Sahi
- phantomjs
- SourceLabs/BrowserStack

- 前端库/框架

- jQuery/Underscore/Mootools/Prototype.js
- YUI3/Dojo/ExtJS/KISSY
- Backbone/KnockoutJS/Emberjs

- **AngularJS**

- Batarang
- Bootstrap
- Semantic UI
- Juice UI
- Web Atoms
- Polymer
- Dhtmlx
- qooxdoo
- React
- Brick

- 前端标准/规范

- HTTP/1.1: RFCs 7230-7235
- HTTP/2
- ECMAScript 5/6
- W3C: DOM/BOM/XHTML/XML/JSON/JSONP/...
- CommonJS Modules/AMD
- HTML5/CSS3

- **Semantic Web**

- MicroData
- RDFa

- **Web Accessibility**

- WCAG
- Role Attribute
- WAI-ARIA

- 性能

- JSPerf
- YSlow 35 rules
- PageSpeed
- HTTPWatch
- DynaTrace's Ajax
- 高性能JavaScript

- SEO

- 编程知识储备

- 数据结构
- OOP/AOP
- 原型链/作用域链
- 闭包
- 编程范型
- 设计模式
- Javascript Tips

- 部署流程

- 压缩合并

- YUI Compressor
- Google Clousure Complier
- UglifyJS
- CleanCSS

- 文档输出

- JSDoc
- Dox/Doxmate/Grunt-Doxmate

- 项目构建工具

- make/Ant
- GYP
- Grunt

- Gulp
- Yeoman
- FIS
- Mod

- 代码组织

- 类库模块化

- CommonJS/AMD
 - YUI3模块
 - CMD
 - UMD

- 业务逻辑模块化

- bower/component

- 文件加载

- LABjs
 - SeaJS/Require.js

- 模块化预处理器

- Browserify
 - Webpack

- 安全

- CSRF/XSS
 - CSP
 - Same-origin policy
 - ADsafe/Caja/Sandbox

- 移动Web

- HTML5/CSS3
 - 响应式网页设计
 - Zeptojs/iScroll

- V5/Sencha Touch
- PhoneGap
- jQuery Mobile
- W3C Mobile Web Initiative
- W3C mobileOK Checker
- Open Mobile Alliance

- 前沿技术社区/会议

- D2/WebRebuild
- NodeParty/W3CTech/HTML5梦工厂
- JSConf/沪JS(JSConf.cn)
- QCon/Velocity/SDCC
- JSConf/NodeConf
- CSSConf
- YDN/YUIConf
- HybridApp
- WHATWG
- MDN
- codepen
- w3cplus
- CNode

- 计算机知识储备

- 编译原理
- 计算机网络
- 操作系统
- 算法原理
- 软件工程/软件测试原理
- Unicode

- 软技能

- 知识管理/总结分享
- 沟通技巧/团队协作
- 需求管理/PM
- 交互设计/可用性/可访问性知识

- 可视化

- SVG/Canvas/VML
- SVG: D3/Raphaël/Snap.svg/DataV
- Canvas: CreateJS/KineticJS
- WebGL/Three.JS

Collect From:<http://www.stuq.org/subject/skill-map/>

资源教程：

1. 综合类

- 前端知识体系
- 前端知识结构
- Web前端开发大系概览
- Web前端开发大系概览-中文版
- Web Front-end Stack v2.2
- 免费的编程中文书籍索引
- 前端书籍
- 前端免费书籍大全
- 前端知识体系
- 免费的编程中文书籍索引
- 智能社 - 精通JavaScript开发
- 重新介绍 JavaScript (JS 教程)
- 麻省理工学院公开课：计算机科学及编程导论
- JavaScript中的this陷阱的最全收集--没有之一
- JS函数式编程指南
- JavaScript Promise迷你书（中文版）
- 腾讯移动Web前端知识库
- Front-End-Develop-Guide 前端开发指南
- 前端开发笔记本
- 大前端工具集 - 聂微东
- 前端开发者手册

2. 入门类

- 前端入门教程
- 瘦雪峰的Javascript教程
- jQuery基础教程
- 前端工程师必备的PS技能——切图篇
- 结合个人经历总结的前端入门方法

3. 效果类

- 弹出层
- 焦点图轮播特效

4. 工具类

- css sprite 雪碧图制作
- 版本控制入门 – 搬进 Github
- Grunt-beginner前端自动化工具

5. 慕课专题

- 张鑫旭 - 慕课系列
- lyn - 慕课系列
- 艾伦 - 慕课系列
- 碧仔 - Hello，移动WEB

6. 周报类

- 平安科技移动开发二队技术周报

六. API:

1. 总目录

1. 开发中心

- mozilla js参考
- chrome开发中心 (chrome的内核已转向blink)
- safari开发中心
- microsoft js参考 .aspx)
- js秘密花园
- js秘密花园
- w3help 综合Bug集合网站

2. 综合搜索

- javascripting
- 各种流行库搜索

3. 综合API

- runoob.com-包含各种API集合
- 开源中国在线API文档合集
- devdocs 英文综合API网站

2. jQuery

- jQuery API 中文文档
- hemin 在线版
- css88 jq api
- css88 jqui api
- 学习jquery
- jquery 源码查找

3. Ecmascript

- Understanding ECMAScript 6 - Nicholas C. Zakas
- exploring-es6
- exploring-es6 翻译
- exploring-es6 翻译后预览
- 阮一峰 es6
- 阮一峰 Javascript
- ECMA-262，第 5 版
- es5

4. Js template

- template-chooser
- artTemplate
- tomajs
- 淘宝模板juicer模板
- Fxtpl v1.0 繁星前端模板引擎
- laytpl
- mozilla - nunjucks
- Juicer
- dustjs
- etpl

5. 弹出层

- artDialog 最新版
- artDialog 文档
- google code 下载地址
- 贤心弹出层
- 响应式用户交互组件库
- sweetalert-有css3动画弹出层

6. CSS

- CSS 语法参考
- CSS3动画手册
- 腾讯css3动画制作工具
- 志爷css小工具集合
- css3 js 移动大杂烩
- bouncejs 触摸库
- css3 按钮动画

- [animate.css](#)
- [全局CSS的终结\(狗带\) \[译\]](#)

7. Angularjs

- [Angular.js 的一些学习资源](#)
- [angularjs中文社区](#)
- [Angularjs源码学习](#)
- [Angularjs源码学习](#)
- [angular对bootstrap的封装](#)
- [angularjs + nodejs](#)
- [吕大豹 Angularjs](#)
- [AngularJS 最佳实践](#)
- [Angular的一些扩展指令](#)
- [Angular数据绑定原理](#)
- [一些扩展Angular UI组件](#)
- [Ember和AngularJS的性能测试](#)
- [带你走近AngularJS - 基本功能介绍](#)
- [Angularjs开发指南](#)
- [Angularjs学习](#)
- [不要带着jQuery的思维去学习AngularJS](#)
- [angularjs 学习笔记](#)
- [angularjs 开发指南](#)
- [angularjs 英文资料](#)
- [angular bootstrap](#)
- [angular jq mobile](#)
- [angular ui](#)
- [整合jQuery Mobile+AngularJS经验谈](#)
- [有jQuery背景，该如何用AngularJS编程思想](#)
- [AngularJS在线教程](#)
- [angular学习笔记](#)

8. React

- [react.js 中文论坛](#)
- [react.js 官方网址](#)
- [react.js 官方文档](#)
- [react.js material UI](#)
- [react.js TouchstoneJS UI](#)
- [react.js amazeui UI](#)

- [React 入门实例教程 - 阮一峰](#)
- [React Native 中文版](#)
- [Webpack 和 React 小书 - 前端乱炖](#)
- [Webpack 和 React 小书 - gitbook](#)
- [webpack](#)
- [Webpack，101入门体验](#)
- [webpack入门教程](#)
- [基于webpack搭建前端工程解决方案探索](#)
- [React原创实战视频教程](#)

9. 移动端API

1. API

- [99移动端知识集合](#)
- [移动端前端开发知识库](#)
- [移动前端的一些坑和解决方法（外观表现）](#)
- [【原】移动web资源整合](#)
- [zepto 1.0 中文手册](#)
- [zepto 1.0 中文手册](#)
- [zepto 1.1.2](#)
- [zepto 中文注释](#)
- [jqmobile 手册](#)
- [移动浏览器开发集合](#)
- [移动开发大杂烩](#)
- [微信webview中的一些问题](#)

2. 框架

- [特色的HTML框架可以创建精美的iOS应用](#)
- [淘宝SUI](#)

10. avalon

- [avalonjs](#)
- [Avalon新一代UI库： OniUI](#)
- [avalon.oniui-基于avalon的组件库](#)
-

11. Requiejs

- [Javascript模块化编程（一）：模块的写法](#)
- [Javascript模块化编程（二）：AMD规范](#)

- [Javascript模块化编程（三）：require.js的用法](#)
- [RequireJS入门（一）](#)
- [RequireJS入门（二）](#)
- [RequireJS进阶（三）](#)
- [requrie源码学习](#)
- [requrie 入门指南](#)
- [requrieJS 学习笔记](#)
- [requriejs 其一](#)
- [require backbone结合](#)

12. Seajs

- [seajs](#)
- [seajs 中文手册](#)

13. Less,sass

- [sass](#)
- [sass教程-sass中国](#)
- [Sass 中文文档](#)
- [less](#)

14. Markdown

- [Markdown 语法说明 \(简体中文版\)](#)
- [markdown入门参考](#)
- [gitbook 国外的在线markdown可编辑成书](#)
- [mdeditor 一款国内的在线markdown编辑器](#)
- [stackedit 国外的在线markdown编辑器，功能强大，同步云盘](#)
- [mditor 一款轻量级的markdown编辑器](#)
- [lepture-editor](#)
- [markdown-editor](#)

15. D3

- [d3 Tutorials](#)
- [Gallery](#)
- [lofter](#)
- [iteye](#)
- [ruanyifeng](#)

16. 兼容性

- esma 兼容列表
- W3C CSS验证服务
- caniuse
- csscreator
- microsoft.aspx)
- 在线测兼容-移动端
- emulators

17. UI相关

- bootcss
- MetroUICSS
- semantic
- Buttons
- kitecss
- pintuer
- amazeui
- worldhello
- linuxtoy
- gitmagic
- rogerdudler
- gitref
- book
- gogojimmy

18. HTTP

- HTTP API 设计指南

19. 其它API

- javascript流行库汇总
- 验证api
- underscore 中文手册
- underscore源码分析
- underscore源码分析-亚里士朱德的博客
- underscrejs en api
- lodash - underscore的代替品
- ext4api

- backbone 中文手册
- qwrap 手册
- 缓动函数
- svg 中文参考
- svg mdn 参考
- svg 导出 canvas
- svg 导出 png
- ai-to-svg
- localStorage 库

20. 图表类

- Highcharts 中文 API
- Highcharts 英文 API
- ECharts 百度的图表软件
- 高德地图
- 开源的矢量图脚本框架
- svg 地图

21. vue

- Vue
- Vue 论坛
- Vue 入门指南
- Vue 的一些资源索引
-

21. 正则

- JS 正则表达式元字符
- 正则表达式 30 分钟入门教程
- MDN - 正则表达式
- ruanyifeng - RegExp 对象
- 小胡子哥 - 进阶正则表达式
- is.js
- 正则在线测试
-

22. ionic

- ionic

23. 其它

- Mock.js 是一款模拟数据生成器

七. 开发规范

1. 前端

- 通过分析github代码库总结出来的工程师代码书写习惯
- HTML&CSS编码规范 by @mdo
- 团队合作的css命名规范-腾讯AlloyTeam前端团队
- 前端编码规范之js - by yuwenhui
- 前端编码规范之js - by 李靖
- 前端开发规范手册
- Airbnb JavaScript 编码规范（简体中文版）
- AMD与CMD规范的区别
- AMD与CMD规范的区别
- KISSY 源码规范
- bt编码规范
- 规范加强版
- 前端代码规范 及 最佳实践
- 百度前端规范
- 百度前端规范
- 百度前端规范
- ECMAScript6 编码规范--广发证券前端团队
- JavaScript 风格指南/编码规范（Airbnb公司版）
- 网易前端开发规范
- css模块
- 前端规范资源列表

2. PHP

- 最流行的PHP 代码规范
- 最流行的PHP 代码规范

3. Android

- 【敏捷开发】Android团队开发规范
- Android 开发规范与应用

八. 其它收集

1. 各大公司开源项目

- [Facebook Projects](#)
- [百度web前端研发部](#)
- [百度EFE](#)
- [百度github](#)
- [alloyteam](#)
- [alloyteam-github](#)
- [alloyteam-AlloyGameEngine](#)
- [AlloyDesigner](#) 即时修改，即时保存，设计稿校正，其它开发辅助工具
- [H5交互页编辑器AEditor介绍](#) H5动画交互页开发的工具介绍
- [AEditor](#) H5动画交互页开发的工具
- [maka](#)
- [值得订阅的weekly](#)
- [腾讯html5](#)
- [奇舞团开源项目](#)
- [Qunar UED](#)

2. Javascript

1. 常用

- [ieBetter.js-让IE6-IE8拥有IE9+,Chrome等浏览器特性](#)
- [模拟键盘](#)
- [拼音](#)
- [中国个人身份证号验证](#)

2. 算法

- [数据结构与算法 JavaScript 描述. 章节练习](#)
- [常见排序算法（JS版）](#)
- [经典排序](#)
- [常见排序算法js版本](#)
- [JavaScript 算法与数据结构 精华集](#)
- [面试常考算法题精讲](#)
-

3. 移动端

- [fastclick](#)
- [no-click-delay](#)

4. JSON

- [模拟生成JSON数据](#)
- [返回跨域JSONAPI](#)

3. HTML5

- HTML5 有哪些让你惊艳的 demo ?

4. CSS

- browserhacks
-

5. jQuery

1. 焦点图

- myfocus
- myfocus-官方演示站
- SuperSlidev2.1 -- 大话主席
- soChange

6. Ext, EasyUI, J-UI 及其它各种UI方案

1. Ext

- extjs
- ext4英文api
- ext4中文api
-

2. EasyUI

- jquery easyui 未压缩源代码

3. J-UI

- J-UI

4. Other

- MUI-最接近原生APP体验的高性能前端框架
- Amaze UI | 中国首个开源 HTML5 跨屏前端框架
- 淘宝 HTML5 前端框架
- KISSY - 阿里前端JavaScript库
- 网易Nej - Nice Easy Javascript
- Kendo UI MVVM Demo
- Bootstrap
- Smart UI
- 雅虎UI - CSS UI

7. 页面社会化分享功能

- 百度分享 pc端
- JiaThis pc端
- 社会化分享组件 移动端
- ShareSDK 轻松实现社会化功能 移动端
- 友盟分享 移动端

8. 富文本编辑器

- 百度 ueditor
- 经典的ckeditor
- 经典的kindeditor
- wysiwyg
- 一个有情怀的编辑器。Bach's Editor
- tower用的编辑器
- summernote 编辑器
- html5编辑器
- XEditor
- wangEditor

9. 日历

1. PC

- 经典my97
- 强大的独立日期选择器
- fullcalendar
- fullcalendar日历控件知识点集合
- 中文api
- 农历日历
- 超酷的仿百度带节日日历老黄历控件
- 日期格式化
- 大牛日历控件
- 我群某管理作品
- input按位替换-官网
- input按位替换-github
- bootstrap-daterangepicker
- 国外30个插件集合
- JavaScript datepicker
- Datepair.js

- 一个风格多样的日历
- 弹出层式的全日历
- [jquery双日历](#)

2. 移动

- 大气实用jQuery手机移动端日期选择插件
- [jQuery Mobile 移动开发中的日期插件Mobiscroll](#)

1. Date library

- [Datejs](#)
- [sugarjs](#)

10. 综合效果搜索平台

- 效果网
- [17素材](#)
- 常用的JavaScript代码片段

11. 前端工程化

1. 概述

- [前端工具大全](#)
- [什么是前端工程化](#)

2. Gulp

- [Gulp官网](#)
- [Gulp中文网](#)
- [gulp资料收集](#)
- [Gulp：任务自动管理工具 - ruanyifeng](#)
- [Gulp插件](#)
- [Gulp不完全入门教程](#)
- [为什么使用gulp?](#)
- [Gulp安装及配合组件构建前端开发一体化](#)
- [Gulp 入门指南](#)
- [Gulp 入门指南 - nimojs](#)
- [Gulp入门教程](#)
- [Gulp in Action](#)
- [Gulp开发教程（翻译）](#)
- [前端构建工具gulpjs的使用介绍及技巧](#)

3. Grunt

- [gruntjs](#)
- [Grunt中文网](#)

4. Fis

- [fis 官网](#)
- [fis](#)

12. 轮播图

1. pc图轮

- [单屏轮播sochange](#)
- [左右按钮多图切换](#)
- [fullpage全屏轮播](#)

2. 移动端

- [无缝切换](#)
- [滑屏效果](#)
- [全屏fullpage](#)
- [单个图片切换](#)
- [单个全屏切换](#)
- [百度的切换库](#)
- [单个全屏切换](#)
- [滑屏效果](#)
- [旋转拖动设置](#)
- [类似于swipe切换](#)
- [支持多种形式的触摸滑动](#)
- [滑屏效果](#)
- [大话主席pc移动图片轮换](#)
- [滑屏效果](#)
- [基于zepto的fullpage](#)
- [\[WebApp\]定宽网页设计下，固定宽度布局开发WebApp并实现多终端下WebApp布局自适应](#)
- [判断微信客户端的那些坑](#)
- [可以通过javascript直接调用原生分享的工具](#)
- [JiaThis 分享到微信代码](#)
- [聊聊移动端跨平台开发的各种技术](#)
- [前端自动化测试](#)
- [多种轮换图片](#)
- [滑动侧边栏](#)

13. 文件上传

- 百度上传组件
- 上传
- flash 头像上传
- 图片上传预览
- 图片裁剪
- 图片裁剪-shearphoto
- jQuery图片处理
-

14. 模拟select

- 糖饼 select
- flexselect
- 双select
- select2
-

15. 取色插件

- 类似 Photoshop 的界面取色插件
- jquery color
- 取色插件集合
- farbtastic 圆环+正方形
-

16. 城市联动

- jquery.cityselect.js基于jQuery+JSON的省市或自定义联动效果
-

17. 剪贴板

- 剪贴板
- clipboard 最新的剪切方案
- 不是Flash的剪贴板

18. 简繁转换

- 简繁转换

19. 表格 Grid

- [facebook表格](#)
- [类似于Excel编辑表格-handsontable](#)
- [bootstrap-table插件](#)
- [datatables](#)

20. 在线演示

- [js 在线编辑 - runjs](#)
- [js 在线编辑 - jsbin](#)
- [js 在线编辑 - codepen](#)
- [js 在线编辑 - jsfiddle](#)
- [java 在线编辑 - runjs](#)
- [js 在线编辑 - hcharts](#)
- [js 在线编辑 - jsdm](#)
- [sql 在线编辑 - sqlfiddle](#)
- [mozilla 在线编辑器](#)

21. 播放器

- [Html5 VideoPlayer](#)

22. 粒子动画

- [Proton 烟花](#)

九. Nodejs

- [nodejs 篇幅比较大](#)
- [Node.js 包教不包会](#)
- [篇幅比较少](#)
- [node express 入门教程](#)
- [nodejs定时任务](#)
- [一个nodejs博客](#)
- [【NodeJS 学习笔记04】新闻发布系统](#)
- [过年7天乐，学nodejs 也快乐](#)
- [七天学会NodeJS](#)
- [Nodejs学习笔记（二）--- 事件模块](#)
- [nodejs入门](#)
- [angularjs nodejs](#)

- 从零开始nodejs系列文章
- 理解nodejs
- nodejs事件轮询
- node入门
- nodejs cms
- Node初学者入门，一本全面的NodeJS教程
- NodeJS的代码调试和性能调优

十. 性能优化

1. 常规优化

- Javascript高性能动画与页面渲染
- 移动H5前端性能优化指南
- 5173首页前端性能优化实践
- 给网页设计师和前端开发者看的前端性能优化
- 复杂应用的 CSS 性能分析和优化建议
- 张鑫旭——前端性能
- 前端性能监控总结
- 网站性能优化之CSS无图片技术
- web前端性能优化进阶路
- 前端技术：网站性能优化之CSS无图片技术
- 浏览器的加载与页面性能优化
- 页面加载中的图片性能优化
- Hey——前端性能
- html优化
- 99css——性能
- Yslow——性能优化
- YSLOW中文介绍
- 转一篇Yahoo关于网站性能优化的文章，兼谈本站要做的优化
- Yahoo!团队实践分享：网站性能
- 网站性能优化指南：什么使我们的网站变慢？
- 网站性能优化实践，减少加载时间，提高用户体验
- 浅谈网站性能优化 前端篇
- 前端重构实践之如何对网站性能优化？
- 前端性能优化：使用媒体查询加载指定大小的背景图片
- 网站性能系列博文
- 加载，不只是少一点点
- 前端性能的测试与优化
- 分享网页加载速度优化的一些技巧？
- 页面加载中的图片性能优化

- [web前端优化\(基于Yslow\)](#)
- [网站性能优化工具大全](#)
- [【高性能前端1】高性能HTML](#)
- [【高性能前端2】高性能CSS](#)
- [由12306谈谈网站前端性能和后端性能优化](#)
- [AlloyTeam——前端优化](#)
- [毫秒必争，前端网页性能最佳实践](#)
- [网站性能工具Yslow的使用方法](#)
- [前端工程与性能优化（上）：静态资源版本更新与缓存](#)
- [前端工程与性能优化（下）：静态资源管理与模板框架](#)
- [HTTPS连接的前几毫秒发生了什么](#)
- [Yslow](#)
- [Essential Web Performance Metrics — A Primer, Part 1](#)
- [Essential Web Performance Metrics — Part 2](#)
- [YUISlide,针对移动设备的动画性能优化](#)
- [Improving Site Performance](#)
- [让网站提速的最佳前端实践](#)
- [Why Website Speed is Important](#)
- [Need for Speed – How to Improve your Website Performance](#)
- [阿里无线前端性能优化指南 \(Pt.1 加载期优化\)](#)
-

2. 优化工具

- [JavaScript 性能分析新工具 OneProfile](#)
- [JavaScript 堆内存分析新工具 OneHeap](#)

3. 在线工具

- [google在线工具](#)
- [阿里测](#)
- [阿里-免费测试服务](#)
- [阿里-F2etest多浏览器兼容性测试解决方案](#)
- [js性能测试](#)
-

十一. 前端架构

- [技术架构](#)
- [前端架构](#)
- [如何成为前端架构师](#)
- [关于前端架构-张克军](#)
- [百度腾讯offer比较（腾讯游戏VS百度基础架构）](#)

-

十二. 个人作品

1. 推荐作品

- winter代码片段需要翻墙
- fgm
- 岑安作品集
- 当耐特demo集合
- 米空格 js作品
- myFocus
- SeaJS组件库
- 颜海镜作品
- 脚儿网作品
- javascript个人作品
- 妙味的雷东升游戏作品
- javascript作品集
- 云五笔，灰度产生生成工具
- 项目主页
- 个性的作品主页
- 播放器
- ucren js demos 集
- 智能社
- 实例陈列架
- zoye demo
- 王员外
- 平凡
- jyg 游戏案例
- 很多jquery插件
- 不羁虫 - soJs 作品系列
- frozenui
- 黑白棋
- fromone

2. 群员作品

- MDialog - [合肥-M.J]
- 轮播图 - [上海-冷静]
- [广州—坚壳]

- [成都 - 无痕] 感恩节专题
- [球霸天]
- [北京-小数]
- [ptf] Magix 工具
- [杭州-Pft] Magix 基于 MVC 结构和 Hash 驱动的 OPOA (One Page One Application) 应用
- [上海-剧中人]-实验室
- [上海-豪情] 作品集合
- [成都-feeling]
- [上海-angela]
- [海南-hank]作品
- [上海-张力]博客
- [上海-zenki]作品
- 移动端图案解锁
- [合肥-M.J] - MPreview 移动端图片预览组
- [合肥-M.J] - Mexam 移动端在线做题组
- [北京-苏瑞] - dancer小人
- [上海-玄沐]- 个人网站
- [厦门-二哲]- 个人博客

3. 国外大牛精品

- [pazguille](#)

十三. 简历模板

- 不错的个人简历
- 简历
- 张伦
- 简历
- 翁天信
- 动画方式的简历
- 组件丰富简历
- 简历池
- [haorooms](#)博客
- [Justin Young](#)
-

十四. 面试题

- 那几个月在找工作（百度，网易游戏）
- 2014最新面试题
- 阿里前端面试题
- 2016校招内推 -- 阿里巴巴前端 -- 三面面试经历
- 腾讯面试题
- 年后跳槽那点事：乐视+金山+360面试之行
- 阿里前端面试题上线
- 拉勾网js面试题
- 前端面试
- Web开发笔试面试题 大全
- 前端开发面试题
- 2014最新前端面试题
- 百度面试
- 面试题
- 前端工作面试问题
- 前端开发面试题
- 5个经典的前端面试问题
- 最全前端面试问题及答案总结
- 如何面试一名前端开发工程师？
- 史上最全 前端开发面试问题及答案整理
- 前端实习生面试总结
- 史上最全 前端开发面试问题及答案整理
- BAT及各大互联网公司2014前端笔试面试题：JavaScript篇
- 前端开发面试题大收集
- 收集的前端面试题和答案
- 如何面试前端工程师
- 前端开发面试题
- 牛客网-笔试面经

十五. iconfont

- 中文字体
- 淘宝字库
- 字体
- 制作教程
- zhangxinxu-iconfont
- iconfont
- 用字体在网页中画ICON图标(推荐教程)
- 字体压缩工具 感谢初级群 [深圳-小鱼] 的推荐

十六. 开发工具类

1. 前端开发工具

- [IntelliJ IDEA 简体中文专题教程](#)
- [Webstorm,IntelliIdea,Phpstorm](#)
- [SublimeText](#)
- [Atom](#)
- [visual studio code](#)

2. Chrome, Firebug, Filddle 调试

i. Fiddler

- [Fiddler调式使用知多少\(一\)深入研究](#)
- [微信fiddle](#)
- [微信fiddle](#)
- [...](#)

ii. Chrome

- [Google Chrome 官方](#)
- [Chrome - 基础](#)
- [Chrome - 进阶](#)
- [Chrome - 性能](#)
- [Chrome - 性能进阶](#)
- [Chrome - 移动](#)
- [Chrome - 使用技巧](#)
- [Chrome - Console控制台不完全指南](#)
- [Chrome - Workspace使浏览器变成IDE](#)
- [network面板](#)
- [chrome开发工具快捷键](#)
- [chrome调试工具常用功能整理](#)
- [Chrome 开发工具 Workspace 使用](#)
- [Chrome神器Vimium快捷键学习记录](#)
- [sass调试-w3cplus](#)
- [如何更专业的使用Chrome开发者工具-w3cplus](#)
- [chrome调试canvas](#)
- [chrome profiles1](#)
- [chrome profiles2](#)
- [chrome profiles3](#)
- [chrome移动版调试](#)
- [chrome调试](#)
- [chrome的调试](#)

- chrome console 命令详解
- 查看事件绑定1
- 查看事件绑定2
- 神器——Chrome开发者工具(一)
- 奇趣百科性能优化(Chrome DevTools 中的 Timeline Profils 等工具使用介绍)
- chrome 开发者工具的 15 个小技巧
- Chrome开发者工具不完全指南
- Chrome 开发者工具使用技巧

iii. Firebug

- firebug视频教程
- firefox 模拟器
- console.log 命令详解
- Firebug入门指南
- Firebug控制台详解
-

iv. 移动,微信调试

- 浏览器端调试安卓
- 移动端前端开发调试
- 使用 Chrome 远程调试 Android 设备
- mac移动端调试
- mac移动端调试
- 无线调试攻略
- 无线调试攻略
- 扁爆了,完美调试 微信webview(x5)
- 微信调试的那些事
- 远程console
- 微信调试工具
- 各种真机远程调试方法汇总

v. iOS Simulator

- Simulator
- Xcode 中的iOS模拟器(iOS Simulator)的介绍和使用心得

3. img

- loading img
- 智图-图片优化平台
- 在线png优化

4. 生成二维码

- 生成二维码

5. 浏览器同步

- [puer](#)
- [liveReload](#)
- [f5](#)
- [File Watchers](#)

6. 在线PPT制作

- [nodePPT](#)
- [PPT](#)
- [reveal](#)
- [slippy](#)

十七. 前端导航网站

- [界面清爽的前端导航](#)
- [前端导航](#)
- [前端网址导航](#)
- [前端名录](#)
- [前端导航](#)
- [前端开发资源](#)
- [网址导航](#)
- [前端开发仓库 - 众多效果的收集地](#)
- [前端资源导航](#)
- [F2E 前端导航](#)

十八. 常用CDN

- [新浪CDN](#)
- [百度静态资源公共库](#)
- [360网站卫士常用前端公共库CDN服务](#)
- [Bootstrap中文网开源项目免费 CDN 服务](#)
- [开放静态文件 CDN - 七牛](#)
- [CDN加速 - jq22](#)
- [jQuery CDN](#)
- [Google jQuery CDN](#)
- [微软CDN](#)

十九. Git,SVN,Github

1. Git

- [git-scm](#)

- 廖雪峰-Git教程
- git-for-windows
- GitHub 添加 SSH keys
- gogithub
- git常规命令练习
- git的资料整理
- 我所记录的git命令（非常实用）
- 企业开发git工作流模式探索部分休整
- GitHub 漫游指南
- GitHub秘籍
- 使用git和github进行协同开发流程
- 动画方式练习git

来自：[Front-end-tutorial](#)

目录

- [JavaScript资源大全中文版](#)
 - [包管理器](#)
 - [加载器](#)
 - [打包工具](#)
 - [测试框架](#)
 - [QA 工具](#)
 - [MVC 框架和库](#)
 - [基于 Node 的 CMS 框架](#)
 - [模板引擎](#)
 - [数据可视化](#)
 - [时间轴](#)
 - [编辑器](#)
 - [工具](#)
 - [文件](#)
 - [函数式编程](#)
 - [响应式编程](#)
 - [数据结构](#)
 - [日期](#)
 - [字符串](#)
 - [数字](#)
 - [存储](#)
 - [颜色](#)
 - [国际化和本地化](#)
 - [类](#)
 - [控制流](#)
 - [路由](#)
 - [安全性](#)
 - [日志](#)
 - [正则表达式](#)
 - [媒体](#)
 - [语言命令](#)
 - [API](#)
 - [视觉检测](#)
 - [浏览器检测](#)
 - [UI](#)
 - [代码高亮](#)

- 加载状态
- 验证
- 键盘封装器
- 浏览和引导
- 通知
- 幻灯片
- 滑块控件
- 表单组件
- 提示
- 模态框和弹出框
- 滚动条
- 菜单
- 表格/栅格
- 框架
- 移动
 - 手势
- 地图
- 视频/音频
- 动画
- 图片处理
- ES6
- SDK
- 大杂烩
- 精品阅读
- 资源
 - 社区
 - 有影响力的书
 - 微博、微信公众号
 - 知名网站
 - 博客

包管理器

管理着 javascript 库，并提供读取和打包它们的工具。

- npm : npm 是 javascript 的包管理器。[官网](#)
- Bower : 一个 web 应用的包管理器。[官网](#)
- component : 能构建更好 web 应用的客户端包管理器。[官网](#)
- spm : 全新的静态包管理器。[官网](#)

- jam : 一个专注于浏览器端和兼容 RequireJS 的包管理器。[官网](#)
- jspm : 流畅的浏览器包管理器。[官网](#)
- Ender : 没有库文件的程序库。[官网](#)
- volo : 以项目模板、添加依赖项与自动化生成的方式创建前端项目。[官网](#)
- Duo : 一个整合 Component、Browserify 和 Go [官网](#)的最佳思想，使开发者能快速方便地组织和编写前端代码的下一代包管理器。

加载器

JavaScript 的模块或加载系统。

- RequireJS : JavaScript 文件和模块的加载器。[官网](#)
- browserify : 在浏览器端以 node.js 的方式 require()。[官网](#)
- SeaJS : 用于 Web 的模块加载器。[官网](#)
- HeadJS : HEAD 的唯一脚本。[官网](#)
- curl : 小巧、快速且易扩展的模块加载器，它能处理 AMD、CommonJS Modules/1.1、CSS、HTML/text 和历史脚本。[官网](#)
- lazyload : 小巧且无依赖的异步 JavaScript 和 CSS 加载器。[官网](#)
- script.js : 异步 JavaScript 加载器和依赖管理器。[官网](#)
- systemjs : AMD、CJS (commonJS) 和符合 ES6 规范的模块加载器。[官网](#)
- LodJS : 基于 AMD 的模块加载器。[官网](#)
- ESL : 浏览器端的模块加载器，支持延迟定义和 AMD。[官网](#)
- modulejs : 轻量的 JavaScript 模块系统。[官网](#)

打包工具

- browserify : Browserify 让你能在浏览器端使用 require('modules')，打包所有依赖。[官网](#)
- webpack : 为浏览器打包 CommonJs/AMD 模块。[官网](#)

测试框架

框架

- mocha : 适用于 node.js 和浏览器、简易、灵活、有趣的 JavaScript 测试框架。[官网](#)
- jasmine : 简单无 DOM 的 JavaScript 测试框架。[官网](#)
- qunit : 一个易于使用的 JavaScript 单元测试框架。[官网](#)
- jest : 简单的 JavaScript 单元测试框架。[官网](#)
- prova : 基于 Tape 和 Browserify 的测试运行器，它适用于 Node & 浏览器。[官网](#)

- DalekJS：自动化且跨浏览器的 JavaScript 功能测试框架。[官网](#)

断言

- chai：适用于 node.js 和浏览器的 BDD / TDD 断言框架，并能搭配其它测试框架使用。[官网](#)
- Sinon.JS：对 JavaScript 进行 spies、stubs 和 mock 测试。[官网](#)
- expect.js：简约的、适用于 Node.js 和浏览器端的 BDD 式断言工具。[官网](#)
- should.js：适用于 Node.js 的 BDD 式断言工具。[官网](#)

覆盖率

- istanbul：另一个 JS 代码覆盖率检测工具。[官网](#)
- blanket：一个简单的代码覆盖率检测库。它的设计理念是易于安装和使用，且可用于浏览器端和 node.js。[官网](#)
- JSCover：JSCover 是一个检测 JavaScript 程序代码覆盖率的工具。[官网](#)

运行器

- phantomjs：脚本化的 Headless WebKit。[官网](#)
- slimerjs：一个内核为 Gecko 的类似 PhantomJS 工具。[官网](#)
- casperjs：基于 PhantomJS 和 Slimer JS 的导航脚本和测试工具。[官网](#)
- zombie：基于 node.js、快速、全栈且无图形界面的浏览器的测试工具。[官网](#)
- totoro：一个简单可靠且能跨浏览器运行的测试工具。[官网](#)
- karma：一个优秀的的 JavaScript 测试运行器。[官网](#)
- nightwatch：基于 node.js 和 selenium webdriver 的图形界面自动化测试框架。[官网](#)
- intern：下一代 JavaScript 代码测试栈。[官网](#)
- yolpo：在浏览器逐句执行的 JavaScript 解释器。[官网](#)

QA 工具

- JSHint：JSHint 是一个有助于发现 JavaScript 代码错误和潜在问题的工具。[官网](#)
- jscs：JavaScript 代码风格检测工具。[官网](#)
- jsfmt：格式化、搜索和改写 JavaScript。[官网](#)
- jsinspect：检测复制粘贴和结构类似的代码。[官网](#)
- buddy.js：发现 JavaScript 代码里的 魔术数字。[官网](#)
- ESLint：完全插件化的工具，能在 JavaScript 中识别和记录模式。[官网](#)
- JSLint：高标准、严格和固执的代码质量工具，旨在只保持语言的优良部分。[官网](#)

MVC 框架和库

- [angular.js](#)：为网络应用增强 HTML。[官网](#)
- [aurelia](#)：一个适用于移动设备、桌面电脑和 web 的客户端 JavaScript 框架。[官网](#)
- [backbone](#)：给你的 JS 应用加入带有 Models、Views、Collections 和 Events 的 Backbone。[官网](#)
- [batman.js](#)：最适合 Rails 开发者的 JavaScript 框架。[官网](#)
- [ember.js](#)：一个旨在创建非凡 web 应用的 JavaScript 框架。[官网](#)
- [meteor](#)：一个超简单的、数据库无处不在的、只传输数据的纯 JavaScript web 框架。[官网](#)
- [ractive](#)：新一代 DOM 操作。[官网](#)
- [vue](#)：一个用于构建可交互界面的、直观快速和可组合的 MVVM 框架。[官网](#)
- [knockout](#)：Knockout 用 JavaScript 让创建响应式的富 UI 更加容易。[官网](#)
- [spine](#)：构建 JavaScript 应用的轻量 MVC 库。[官网](#)
- [espresso.js](#)：一个极小的、用于制作用户界面的 JavaScript 库。[官网](#)
- [canjs](#)：让 JS 更好、更快、更简单。[官网](#)
- [react](#)：用于建构用户界面的库。它是声明式的、高效的和极度灵活的，并使用虚拟 DOM 作为其不同的实现。[官网](#)
- [react-native](#)：一个用 React 构建原生应用的框架。[官网](#)
- [riot](#)：类 React 库，但很轻量。[官网](#)
- [thorax](#)：加强你的 Backbone。[官网](#)
- [chaplin](#)：使用 Backbone.js 库的 JavaScript 应用架构。[官网](#)
- [marionette](#)：一个 Backbone.js 的复合应用程序库，旨在简化大型 JavaScript 应用结构。[官网](#)
- [ripple](#)：一个小巧的、用于构建响应界面的基础框架。[官网](#)
- [rivets](#)：轻量却拥有强大的数据绑定和模板解决方案[官网](#)
- [derby](#)：让编写实时和协同应用更简单的 MVC 框架，能够在 Node.js 和浏览器同时运行。[官网](#)
 - [derby-awesome](#)：很棒的 derby 组件集合。[官网](#)
- [way.js](#)：简单、轻量、持久化的双向数据绑定。[官网](#)
- [mithril.js](#)：Mithril 是一个客户端 MVC 框架（轻量、强大和快速）[官网](#)
- [jsblocks](#)：jsblocks 是一个更好的 MV-ish 框架。[官网](#)
- [LiquidLava](#)：易懂的、用于构建用户界面的 MVC 框架。[官网](#)

基于 Node 的 CMS 框架

- [KeystoneJS](#)：强大的 CMS 和 web 应用框架。[官网](#)
- [Reaction Commerce](#)：拥有实时的架构和设计的响应式（reactive）CMS。[官网](#)
- [Ghost](#)：简单、强大的发布平台。[官网](#)

- Apostrophe : 提供内容编辑和基本服务的 CMS。[官网](#)
- We.js : 适用于实时应用、网站或博客的框架。[官网](#)
- Hatch.js : 拥有社交特性的 CMS 平台。[官网](#)
- TaracotJS : 拥有快速、极简风格特点且基于Node.js 的 CMS。[官网](#)
- Nodizecms : 为 CoffeeScript 爱好者准备的 CMS。[官网](#)
- Cody : 拥有所见即所得的编辑器的 CMS。[官网](#)
- PencilBlue : CMS 和博客平台。[官网](#)

模板引擎

模板引擎允许您执行字符串插值。

- mustache.js : 是 JavaScript 中带有的最简模板。[官网](#)
- handlebars.js : 是 Mustache 模板语言的扩展。[官网](#)
- hogan.js : 是 Mustache 模板语言的编译器。[官网](#)
- doT : 最快速简洁的 JavaScript 模板引擎，适用于 nodejs 和浏览器。[官网](#)
- dustjs : 适用于浏览器和 node.js 的异步模板。[官网](#)
- eco : 嵌入式的 CoffeeScript 模板。[官网](#)
- JavaScript-Templates : 轻量（小于 1KB）、快速且无依赖的强大 JavaScript 模版引擎。[官网](#)
- t.js : 小巧的 JavaScript 模板框架，压缩后约为 400 字节。[官网](#)
- Jade : 健壮的、优雅且功能丰富的 nodejs 模板引擎。[官网](#)
- EJS : 高效的 JavaScript 模板。[官网](#)
- xtemplate : 可扩展的模板引擎，适用于 node 和浏览器。[官网](#)
- marko : 快速轻量且基于 HTML 的模板引擎，支持异步、流、自定义标签和 CommonJS 模编译后输出。适用于 Node.js [官网](#) 和浏览器。

数据可视化

Web 数据可视化工具

- d3 : 一个对 HTML 和 SVG 进行可视化的 JavaScript 库。[官网](#)
- metrics-graphics : 更简洁和拥有更规范的数据图表布局优化算法的库。[官网](#)
- pykcharts.js : 经过精心设计后，去除 d3.js 复杂性的 d3.js 图表库。[官网](#)
- three.js : JavaScript 3D 库。[官网](#)
- Chart.js : 简单的、基于 canvas 标签的 HTML5 图表库。[官网](#)
- paper.js : 是矢量图形脚本中的瑞士军刀 —— 使用 HTML5 Canvas 将 Scriptographer 移植到 JavaScript [官网](#) 和浏览器。
- fabric.js : JavaScript Canvas 库，SVG 与 Canvas 可以相互解析。[官网](#)
- peity : 进度条、线状和饼状图。[官网](#)

- raphael : JavaScript 矢量库。[官网](#)
- echarts : 商业产品图表。[官网](#)
- vis : 动态的、基于浏览器的可视化库。[官网](#)
- two.js : 一个渲染器无关的适用于 web 的二维绘图 api 。[官网](#)
- graphael : 基于 Raphaël 图表库。[官网](#)
- sigma.js : 一个致力于图形绘画的 JavaScript 库。[官网](#)
- arbor : 一个使用 web workers 和 jQuery 的图形可视化库。[官网](#)
- cubism : 可视化时间序列的 D3 插件。[官网](#)
- dc.js : 与 crossfilter 无缝合作的多维图表绘制库，使用 d3.js 渲染。[官网](#)
- vega : 一套可视化语法。[官网](#)
- processing.js : Processing.js 基于 Web 标准使数据可视化，而无需任何插件。[官网](#)
- envisionjs : 动态的 HTML5 可视化。[官网](#)
- rickshaw : 用于构建交互式实时图表的 JavaScript 工具包。[官网](#)
- flot : 吸引人的、基于 jQuery 的 JavaScript 图表库。[官网](#)
- morris.js : 漂亮的时间序列线框图。[官网](#)
- nvd3 : 一个为 D3.js 构建可复用图表和图表组件的库。[官网](#)
- svg.js : 一个轻量的、用于操作和添加 SVG 动画的库。[官网](#)
- heatmap.js : 基于 HTML5 canvas 的热力图 JavaScript 库。[官网](#)
- jquery.sparkline : 一个直接在浏览器端生成小型走势图的 jQuery 插件。[官网](#)
- xCharts : 一个基于 D3、用于构建自定义图表和图形的库。[官网](#)
- trianglify : 基于 d3.js 的低多边形 (low poly) 风格背景图片生成器。[官网](#)
- d3-cloud : 创建词云 (word cloud) 效果的 JavaScript 库。[官网](#)
- d4 : 一个基于 D3 、友好、可复用的 DSL 图表库。[官网](#)
- dimple.js : 基于 d3 的简易商业分析图表库。[官网](#)
- chartist-js : 简单的响应式图表。[官网](#)
- epoch : 一个通用的实时图表库。[官网](#)
- c3 : 基于 D3 的可复用图表库。[官网](#)
- BabylonJS : 一个运用 HTML5 和 WebGL 构建 3D 游戏的框架。[官网](#)

也有一些很棒的收费库，如 [amchart](#)、[plotly](#) 和 [highchart](#)。

时间轴

- TimelineJS : 一个用 JavaScript 编写的可叙事时间轴库。[官网](#)
- timesheet.js : 用于构建简单的 HTML5 & CSS3 时间表的 JavaScript 库。[官网](#)

编辑器

- ace : Ace (Ajax.org Cloud9 Editor) 。[官网](#)
- CodeMirror : 浏览器端的代码编辑器。[官网](#)

- **esprima**：用于综合分析的 ECMAScript 解析器。[官网](#)
- **quill**：一个带有 API 的跨浏览器富文本编辑器。[\(官网\)](#)
- **medium-editor**：Medium.com 所见即所得编辑器的克隆版。[官网](#)
- **pen**：享受在线编辑（支持 markdown）。[官网](#)
- **jquery-notebook**：一个易用的、简洁优雅的文本编辑器。灵感来源于 Medium 的魅力。[官网](#)
- **bootstrap-wysiwyg**：小巧的、兼容 bootstrap 的所见即所得的富文本编辑器。[官网](#)
- **ckeditor-releases**：适用于每个人的 web 文本编辑器。[官网](#)
- **editor**：一个 markdown 编辑器，但仍在开发中。[官网](#)
- **EpicEditor**：一个可嵌入的 JavaScript Markdown [官网](#)的编辑器，拥有全屏编辑、即时预览、自动保存草稿和离线支持等功能。
- **jsoneditor**：查看、编辑和格式化 JSON 的 web 工具。[官网](#)
- **vim.js**：拥有持久化 `~/.vimrc` 的 Vim 编辑器的 JavaScript 移植版本。[官网](#)
- **Squire**：HTML5 富文本编辑器。[官网](#)
- **TinyMCE**：JavaScript 富文本编辑器。[官网](#)
- **trix**：由 Basecamp 制作，适用于每天写作的富文本编辑器。[官网](#)

文件

处理文件的库。

- **Papa Parse**：一款强大的 CSV 库，支持解析 CSV 文件/字符串，也能导出 CSV。[官网](#)
- **jBinary**：对用声明式语法描述文件类型和数据结构的二进制文件，进行高级 I/O（加载、解析、操作、序列化、存储）操作。[官网](#)

函数式编程

函数式编程库扩展了 JavaScript 的能力。

- **underscore**：JavaScript 的实用工具。[官网](#)
- **lodash**：提供一致性、可定制、高性能和额外功能的实用库。[官网](#)
- **Sugar**：一个扩展了原生对象功能的 JavaScript 库。[官网](#)
- **lazy.js**：类似 Underscore，但性能更优越[官网](#)
- **ramda**：一个针对 JavaScript 程序员的实用函数库。[官网](#)
- **mout**：模块化的 JavaScript 工具库。[官网](#)
- **mesh**：流数据同步工具。[官网](#)

响应式编程

响应式程序库扩展了 JavaScript 的能力。

- **RxJs**：对 JavaScript 进行响应式扩展。[官网](#)

- Bacon : JavaScript 的 FPR (函数式响应式编程) 库。[官网](#)
- Kefir : 受 Bacon.js 和 RxJS 启发的 FRP 库，专注于高性能和低内存消耗。[官网](#)
- Highland : 对 JavaScript 实用工具的重新思考，Highland 能轻易地管理同步和异步信息，而且仅使用标准 JavaScript 和类 Node 流。[官网](#)
- Most.js : 高性能 FRP 库。[官网](#)

数据结构

数据结构库用于构建一个更复杂的应用。

- immutable-js : 不可变的数据集合，包括 Sequence、Range、Repeat、Map、OrderedMap、Set 和 sparse Vector。[官网](#)
- mori : 使用 ClojureScript 持久化数据结构和支持原生 JavaScript API 的库。[官网](#)
- buckets : 完整的、经过充分测试和记录数据结构的 JavaScript 库。[官网](#)
- hashmap : 简单的 hashmap 实现，支持任何类型的键值。[官网](#)

日期

日期库。

- moment : 解析、验证、操作和显示日期。[官网](#)
- moment-timezone : 基于 moment.js 的时区库。[官网](#)
- jquery-timeago : 一款支持自动更新模糊时间戳的 jQuery 插件（如："4 分钟之前"）。[官网](#)
- timezone-js : 让 JavaScript Date 对象拥有时区功能。使用 Olson zoneinfo 文件记录着时区数据。[官网](#)
- date : 拥有人性化的 Date() 方法。[官网](#)
- ms.js : 小巧的毫秒转换工具。[官网](#)

字符串

字符串库。

- selecting : 一个允许你获取用户选定文本的库。[官网](#)
- underscore.string : 扩展了 Underscore.js 的字符串操作。[官网](#)
- string.js : 额外的 JavaScript 字符串方法。[官网](#)
- he : 健壮的 HTML 实体编码/解码器。[官网](#)
- multiline : 多行字符串。[官网](#)
- query-string : 解析和字符串化 URL 查询字符串。[官网](#)
- URI.js : URL 操作库。[官网](#)
- jsurl : 轻量的 URL 操作库。[官网](#)
- sprintf.js : 实现字符串格式化。[官网](#)

- **url-pattern**：让 url 和其它字符串进行比正则表达式匹配更简单。字符串和数据可相互转化。[官网](#)

数字

- **Numeral.js**：对数字进行格式化和操作的库。[官网](#)
- **odometer**：流畅的数字过渡效果。[官网](#)
- **accounting.js**：对数字、金钱、货币进行格式化的轻量库——完全本地化和无依赖。[官网](#)
- **money.js**：一个小巧（1kb）的货币转换库，适用于 web 和 nodeJS。[官网](#)
- **Fraction.js**：一个有理数库。[官网](#)
- **Complex.js**：一个复数库。[官网](#)
- **Polynomial.js**：一个多项式库。[官网](#)

存储

- **store.js**：为所有浏览器封装了 `LocalStorage`，而没有使用 `cookies` 和 `flash`。隐秘地使用 `localStorage`、`globalStorage` 和用户数据。[官网](#)
- **localForage**：改善后的离线存储。其封装了 `IndexedDB`、`WebSQL` 和 `localStorage`，拥有操作简单和强大的 API。[官网](#)
- **jStorage**：`jStorage` 是一个简单的键值对数据库，用于在浏览器端存储数据。[官网](#)
- **cross-storage**：获得权限后，能跨域名本地存储。[官网](#)
- **basket.js**：用 `localStorage` 加载和缓存脚本的资源加载器。[官网](#)
- **bag.js**：可以缓存脚本和加载资源，与 `basket.js` 相似，但增加了键值对接口和对 `localStorage` / `websql` / [undexedDB](#) 的支持。
- **basil.js**：智能的 JavaScript 数据持久层库。[官网](#)
- **jquery-cookie**：轻量简单的、用于读取、编辑和删除 cookie 的 jQuery 插件。[官网](#)
- **Cookies**：客户端 Cookie 操作库。[官网](#)
- **DB.js**：基于 Promise 的、封装了 `IndexDB` 的库。[官网](#)
- **lawnchair.js**：简单的客户端 JSON 存储。[官网](#)

颜色

- **randomColor**：JavaScript 颜色生成器。[官网](#)
- **chroma.js**：拥有各种各样颜色操作的 JavaScript 库。[官网](#)
- **color**：JavaScript 颜色转换和操作库。[官网](#)
- **colors**：更智能的默认 web 颜色。[官网](#)
- **PleaseJS**：随机创建出赏心悦目的颜色和配色方案。[官网](#)
- **TinyColor**：快速、轻巧的颜色操作和转换库。[官网](#)
- **Vibrant.js**：从图像提取主要颜色。[官网](#)

国际化和本地化 (I18n And L10n)

本地化和国际化 JavaScript 库

- i18next : JavaScript 最简单的国际化 (i18n) 方法。[官网](#)
- polyglot : 小巧的国际化助手库。[官网](#)
- babelfish : i18n 提供友好易懂的 API，并且内置多种支持。[官网](#)

类

- ClassManager : 世界上最快、最方便的类系统之一。[官网](#)
- klass : 用于创建极富表现力的类工具库。[官网](#)
- augment : 世界上最小且最快的一流 JavaScript 继承模式。[官网](#)

控制流

- async : 适用于 node 和浏览器的异步工具库。[官网](#)
- q : 实现异步的 promise JavaScript 库。[官网](#)
- step : 让逻辑顺序合理化的异步控制流库。[官网](#)
- contra : 利用函数风格实现的异步流控制。[官网](#)
- Bluebird : 专注于革新功能和性能的，功能齐全的 promise 库。[官网](#)
- when : 快速可靠的、Promises/A+ 规范的 when() 实现，而且拥有异步其它的优秀特性。[官网](#)
- ObjectEventTarget : 提供增加了事件监听的原型（与 DOMElement 的 EventTarget 在浏览器行为一致）。[官网](#)

路由

- director : 一个小巧的、与 URL 同构的路由器。[官网](#)
- page.js : 受 Express router 启发的小型客户端路由器（约为1200字节）。[官网](#)
- pathjs : 简单、轻量的 web 路由器。[官网](#)
- crossroads : JavaScript 路由。[官网](#)
- davis.js : 使用 pushState、RESTful 风格和可降级的 JavaScript 路由器。[官网](#)
- angular-ui-router : 基于AngularJS的可嵌套路由。[官网](#)

安全性

- DOMPurify : 针对 HTML、MathML 和 SVG 的仅支持DOM、快速、高容错的 XSS 过滤器。[官网](#)
- js-xss : 通过白名单配置，即可过滤不信任的 HTML（防止 XSS 攻击）。[官网](#)

日志

- log：带有样式的 Console.log。[官网](#)
- Conzole：对 JavaScript 原生 console 对象方法和功能进行封装的 debug 面板，使其显示在页面内。[官网](#)
- console.log-wrapper：将日志清晰地记录到 console，且兼容所有浏览器。[官网](#)
- loglevel：最轻量的 JavaScript 日志记录工具库，向封装后可用的 console.log 方法增加可靠的日志等级。[官网](#)
- minilog：轻量的、用流式 API 显示的、可用于客户端和服务器端的日志记录库。[官网](#)

正则表达式

- RegEx101：在线的 JavaScript 正则表达式测试器和调试器。同时支持 Python、PHP 和 PCRE。[官网](#)
- RegExr：用于创建、测试和学习正则表达式的 HTML/JS 工具。[官网](#)
- RegExpBuilder：使用链式方法创建正则表达式。[官网](#)

媒体

- Ion.Sound：可用于任何网页上简单音频。[官网](#)

语音命令

- annyang：向网站添加语音命令的语音识别库。[官网](#)
- voix.js：向网站、app 或游戏添加语音命令的 JavaScript 库。[官网](#)

API

- bottleneck：强大的频率限制器，使调节流量变得更容易。[官网](#)
- oauth-signature-js：适用于 node 和 浏览器的 OAuth 1.0a 签名生成器。[官网](#)
- amygdala：为 Web 应用提供 RESTful HTTP 客户端解决方案。[官网](#)
- jquery.rest：一个让 RESTful API 更易使用的 jQuery 插件。[官网](#)

视觉检测

- tracking.js：在 web 上实现计算视觉的一种现代方法。[官网](#)
- ocrad.js：通过 Emscripten 用 JavaScript 实现 OCR（光学字符识别）。[官网](#)

浏览器检测

- bowser：一个浏览器检测器。[官网](#)

UI

代码高亮

- Highlight.js : JavaScript 语法高亮器。[官网](#)
- PrismJS : 轻量、健壮和优雅的语法高亮器。[官网](#)

加载状态

指示加载状态的库。

- Mprogress.js : 创建谷歌 Material 设计风格的线性进度条。[官网](#)
- NProgress : 在 Ajax'y 应用显示细长型进度条[官网](#)
- Spin.js : 一个旋转的进度指示器。[官网](#)
- progress.js : 为页面任何对象创建和管理进度条。[官网](#)
- progressbar.js : 用 SVG path 动画制作的、漂亮和响应式的进度条。[官网](#)
- pace : 自动向你的网站添加一个进度条。[官网](#)
- topbar : 小巧漂亮的、与网站同宽的进度指示器。[官网](#)
- nanobar : 非常轻量的进度条。不依赖 jQuery。[官网](#)
- PageLoadingEffects : 使用 SVG 动画展现新内容的现代方式。[官网](#)
- SpinKit : 运用 CSS 动画的加载指示器集合。[官网](#)
- Ladda : 内置在按钮的加载指示器。[官网](#)
- css-loaders : 运用 CSS 动画的旋转加载指示器的集合。[官网](#)

除了上述这些库，还有收藏在 [Codepen](#) 的，另外还有 [Ajaxload](#)，[Preloaders](#) 和 [CSSLoad](#) 这些生成器。

验证

- Parsley.js : 不用写一行 JavaScript 代码即可在前端验证表单。[官网](#)
- jquery-validation : jQuery 验证插件。[官网](#)
- validator.js : 字符串验证和过滤（在使用用户输入之前清理用户输入中的有害或危险字符的操作）。[官网](#)
- validate.js : 受 CodeIgniter 启发的轻量表单验证 JavaScript 库。[官网](#)
- validatr : 跨浏览器的 HTML5 表单验证库。[官网](#)
- BootstrapValidator : 是验证表单域中最好的 jQuery 插件。要与 Bootstrap 3 一起使用。[官网](#)
- is.js : 检查类型、正则表达式、是否存在、时间等。[官网](#)
- FieldVal : 多用途验证库。同时支持同步和异步验证。[官网](#)

键盘封装器

- **mousetrap**：处理键盘快捷键的 JavaScript 库。[官网](#)
- **keymaster**：定义和调度键盘快捷键的小型库。[官网](#)
- **Keypress**：键入捕捉工具库，任何键都可以成为一个修饰键。[官网](#)
- **KeyboardJS**：一个用于绑定键盘组合的 JavaScript 库，让你脱离快捷键和快捷键组合冲突的痛苦。[官网](#)
- **jquery.hotkeys**：jQuery Hotkeys 能让你在代码任何的地方监听键盘事件，并几乎支持所有按键组合。[官网](#)
- **jwerty**：令人惊叹的键盘事件处理库。[官网](#)

浏览和引导

- **intro.js**：这是一个介绍新功能的很好方式，能一步步地引导用户浏览你的网站和项目。[官网](#)
- **shepherd**：通过引导让用户浏览你的应用程序。[官网](#)
- **bootstrap-tour**：应用 Twitter Bootstrap 弹出框对产品进行快速简单的引导。[官网](#)
- **tourist**：简单、灵活的应用引导介绍库。[官网](#)
- **chardin.js**：简单的应用遮罩层介绍。[官网](#)
- **pageguide**：使用 jQuery 和 CSS3 的 web 页面元素交互引导库。[官网](#)
- **hopscotch**：让开发者更容易向其页面产品添加引导的框架。[官网](#)
- **joyride**：基于 jQuery 的功能引导插件。[官网](#)
- **focusable**：通过向页面其余部分添加遮罩层，使焦点聚集在特定 DOM 元素。[官网](#)

通知

- **messenger**：为你的应用添加 Growl-style 弹框和信息（Growl 是 Mac OS X 下的一个通知系统）。[官网](#)
- **noty**：jQuery 通知插件。[官网](#)
- **pnotify**：适用于 Bootstrap、jQuery UI 和 Web Notifications Draft 的 JavaScript 通知库。[官网](#)
- **toastr**：用来显示简单的，会自动到期的信息窗口）简单的弹出框通知（[toast notifications](#)[官网](#)
- **humane-js**：一个简单、时髦的浏览器通知系统。[官网](#)
- **smoke.js**：与框架无关的、能够自定义样式的 JavaScript 弹框系统。[官网](#)

幻灯片

- **Swiper**：使用硬件加速过渡的移动设备触控滑块框架。[官网](#)
- **slick**：你所需要的最后一个轮播插件。[官网](#)
- **slidesJS**：响应式的 jQuery (1.7.1+) 幻灯片插件，具有触摸、CSS3 过渡等特性。[官网](#)
- **FlexSlider**：一款令人惊叹的、全响应式的幻灯片 jQuery 插件。[官网](#)

- unslider：最简单的幻灯片 jQuery 插件。[官网](#)
- colorbox：轻量、可自定义的灯箱 jQuery 插件。[官网](#)
- fancyBox：提供了良好优雅的方式，为页面上的图片、html 内容和多媒体添加缩放功能的工具。[官网](#)
- sly：基于项导航的、支持单向滚动的 JavaScript 库。[官网](#)
- vegas：向页面添加漂亮的全屏背景的 jQuery 插件，甚至允许幻灯片。[官网](#)
- Sequence：用于创建响应式的幻灯片、演示、旗帜广告和以步骤为基础的应用的 CSS 动画框架。[官网](#)
- baguetteBox.js：易于使用的、用纯 JavaScript 实现的遮罩层脚本。[官网](#)
- reveal.js：用 HTML 创建漂亮演示控件的框架。[官网](#)
- PhotoSwipe：适用于移动设备和桌面电脑的、模块化和不无依赖框架的 JavaScript 画廊控件。[官网](#)
- jcSlider：用 CSS 动画实现的响应式幻灯片 jQuery 插件。[官网](#)
- basic-jquery-slider：易于使用、指定主题和定制化。[官网](#)
- unslider：这是最简单的幻灯片 jQuery 插件。[官网](#)
- jQuery.adaptive-slider：带有自适应颜色标题和导航的幻灯片 jQuery 插件。[官网](#)
- slidr：可添加一些幻灯片效果。[官网](#)
- Flickity：可触摸的、响应式的和可轻弹的画廊。[官网](#)

滑块控件

- Ion.RangeSlider：强大的、易于自定义的范围滑块选择库，支持很多配置和皮肤。[官网](#)
- jQRangeSlider：支持日期的滑块选择库。[官网](#)
- noUiSlider：轻量无冗余的、高度定制化的滑块选择库。[官网](#)
- rangeslider.js：HTML5 input 区域滑块元素。[官网](#)

表单组件

输入

- typeahead.js：快速的、功能齐全的自动补全库。[官网](#)
- tag-it：处理多标签字段以及标签建议/自动完成的 jQuery UI 插件。[官网](#)
- At.js：向你的应用添加类似 Github 的自动完成提示功能。[官网](#)
- Placeholders.js：JavaScript 补全 HTML5 占位符的属性。[官网](#)
- fancyInput：利用 CSS3 效果让输入更有趣。[官网](#)
- jQuery-Tags-Input：利用这个 jQuery 插件，可奇妙地将一个简单的文本输入转换成一个酷炫的标签列表。[官网](#)
- vanilla-masker：一个纯 JavaScript 实现的输入控制库。[官网](#)
- Ion.CheckRadio：一个为复选框和单选按钮添加样式的 jQuery 库，支持多种皮肤。[官网](#)

日历

- **pickadate.js**：对移动设备友好的、响应式的和轻量的 **jQuery** 日期 & 时间输入选择器。
[官网](#)
- **bootstrap-datepicker**：基于 **bootstrap** 的日历选择器。[官网](#)
- **Pikaday**：一个崭新的 **JavaScript** 日期选择器 —— 轻量、无依赖和模块化的 **CSS**。[官网](#)
- **fullcalendar**：全尺寸、支持拖放事件的日历 (**jQuery** 插件)。[官网](#)
- **rome**：可定制的日期（和时间）选择器。无依赖，可选 UI。[官网](#)
- **datedropper**：**datedropper** 是一个 **jQuery** 插件，它提供了快速简易的方式去管理日期输入框。[官网](#)

选择

- **selectize.js**：**Selectize** 是文本框和选择框的混合体。它基于**jQuery**，拥有自动完成和键盘感应下拉列表功能，可用于标签、联系人列表等。[官网](#)
- **select2**：它基于 **jQuery**，是选择框（**select box**）的替代品。支持搜索、远程数据集和无限滚动。[官网](#)
- **chosen**：可以让冗长不便的选择框更友好的库。[官网](#)

文件上传

- **jQuery-File-Upload**：**File Upload** 是一个支持多文件选择、文件拖放、进度条、验证和图片、音频、视频预览的[官网](#)**jQuery** 插件。
- **dropzone**：**Dropzone** 是一个易于使用且支持多文件拖放的库。其支持图片预览并且拥有很好的进度条效果。[官网](#)
- **flow.js**：一个通过 **HTML5** 的 **File API**，提供多个同时链接的、稳定的、容错的、可恢复的/可重新开始的文件上传库。[官网](#)
- **fine-uploader**：一个带有进度条、拖放功能和支持直接上传到 **S3** (**Amazon Simple Storage Service**，亚马逊简易存储服务) 的多文件上传插件。[官网](#)
- **FileAPI**：**JavaScript** 文件工具集合。支持多文件上传、拖放和文件分块上传。对于图像，支持裁剪、调整大小和根据 **EXIF** 自动调整方向。[官网](#)
- **plupload**：处理文件上传的 **JavaScript** [官网](#)**API**，其支持多文件选择、文件类型过滤、分块请求、客户端图片缩放和根据不同的运行环境选择 **HTML5**、**Silverlight** 和 **Flash**。

其它

- **form**：**jQuery** 表单插件。[官网](#)
- **Garlic.js**：自动在本地保存表单文本和选择框的值，直到表单被提交。[官网](#)
- **Countable**：对某个 **HTML** 元素包含文本的段落数、单词数和字符数进行统计的 **JavaScript** 函数。[官网](#)
- **card**：只需一行代码，让信用卡表单变得更友好。[官网](#)

- stretchy：自适应大小的 form 元素，表单本应该是这样的。[官网](#)
- list.js：向表格、列表等 HTML 元素添加搜索、排序、过滤和自适应功能的库。在已有 HTML 上增加可视化。[官网](#)

提示

- tipsy：基于 jQuery 的 Facebook 风格的提示工具（tooltip）。[官网](#)
- opentip：开源且基于 prototype 框架的 JavaScript 工具提示库。[官网](#)
- qTip2：非常强大的工具提示库。[官网](#)
- tooltipster：一个工具提示 jQuery 插件。[官网](#)
- simptip：用 Sass 制作的、简单的工具提示。[官网](#)
- jquery-popup-overlay：是一个响应式的和可访问性强的模态框（modal）和工具提示框 jQuery 插件。[官网](#)

模态框和弹出框

- Magnific-Popup：专注于性能、轻量、响应式的灯箱（lightbox）脚本。[官网](#)
- jquery-popbox：jQuery 提示框插件。[官网](#)
- jquery.avgrund.js：一种新的定于弹出的模态框 jQuery 插件。[官网](#)
- vex：新的、拥有高度可配置和易于改变样式功能的对话框库。[官网](#)
- bootstrap-modal：对 Bootstrap 默认的模态框类进行扩展。其支持响应式、可堆叠和 ajax 等。[官网](#)
- css-modal：纯 CSS 打造的模态框。[官网](#)
- jquery-popup-overlay：是一个响应式的和可访问性强的模态框和工具提示框（tooltips）jQuery 插件。[官网](#)

滚动

- scrollMonitor：滚动发生时，可以监听元素的、简单、快速的 API。[官网](#)
- headroom：除非你需要显示页面头部（header），否则将隐藏它，以腾出页面头部空间。[官网](#)
- onepage-scroll：创建一个类似 Apple 的单页面滚动网站（iPhone 5S 网站）。[官网](#)
- iscroll：高性能、轻量、无依赖、兼容多平台的 JavaScript 滚动组件。[官网](#)
- skrollr：独立（不依赖 jQuery）的视差滚动库，适用于移动设备（Android + iOS）和桌面电脑。[官网](#)
- parallax：面向智能设备的视差引擎。[官网](#)
- stellar.js：让视差滚动变简单。[官网](#)
- plax：基于 jQuery 的视差库。[官网](#)
- jparallax：创建可交互视差效果的 jQuery 插件。[官网](#)
- fullPage：简单和易于使用的、用于创建全屏滚动网站的插件（也被称为单页面网站）。

[官网](#)

- ScrollMenu：让老旧无聊的滚动条焕然一新。[官网](#)

菜单

- jQuery-menu-aim：当用户光标放在特定下拉菜单项时触发事件。可制作响应式的、大数据量的下拉菜单，如 Amazon 的。[官网](#)
- jQuery contextMenu：右键菜单（contextMenu）管理工具。[官网](#)
- Slideout：为移动设备的 web 应用制作出响应式的、可触摸滑出的导航菜单。[官网](#)
- Slide and swipe：一个基于 touchSwipe 库的滑出菜单插件。[官网](#)

表格/栅格

- jTable：基于 CRUD 表创建 AJAX 的 jQuery 插件。[官网](#)
- DataTables：这是一个非常灵活的工具，在渐进增强的基础上，将高级的交互效果加到 HTML 表格。（jQuery 插件）[官网](#)
- floatThead：（jQuery 插件）锁定表格头部，只允许表格内容滚动。适用于任何表格，而且不需要额外的 html 或 css。[官网](#)
- Masonry：瀑布流式的网格布局库。[官网](#)
- Packery：使用装箱算法（bin-packing）的网格布局库。支持拖拽布局。[官网](#)
- Isotope：可过滤和可排序的网格布局的库，它能实现 Masonry、Packery 等布局。[官网](#)

框架

- Semantic UI：拥有大量主题和元素的 UI 套件。[官网](#)

手势

- hammer.js：拥有多 种触摸手势的 JavaScript 库。[官网](#)
- touchemulator：在桌面电脑模仿触摸输入。[官网](#)
- Dragula：超级易于使用的拖拽库。[官网](#)

地图

- Leaflet：对移动设备友好的、可交互的地图 JavaScript 库。[官网](#)
- Cesium：开源的、基于 WebGL 实现的虚拟地球仪和地图引擎。[官网](#)
- gmaps：以最简单的方式使用 Google 地图。[官网](#)
- polymaps：一个免费的、兼容现代 web 浏览器的、用于制作动态可交互的地图 JavaScript 库。[官网](#)
- kartograph.js：开源的 Kartograph SVG 地图渲染器。[官网](#)
- mapbox.js：Mapbox 的 API，Leaflet 的插件。[官网](#)

- [jqvmap](#)：矢量地图 jQuery 插件。[官网](#)
- [OpenLayers3](#)：高性能的、功能丰富的库，能满足你对地图所有需求。[官网](#)

视频/音频

- [prettyembed.js](#)：更完美地嵌入 YouTube —— 拥有很好的选项，如高分辨率的预览图、嵌入选项的高级定制和可选的 [FitVids](#) 支持。
- [html5media](#)：能在所有主流浏览器播放多媒体标签中定义的多媒体文件。[官网](#)
- [Play-em JS](#)：Play'em 是一个 JavaScript 组件，它能管理音乐/视频播放顺序，通过在一个 DIV 元素里嵌入几个播放器（Youtube、Soundcloud 和 Vimeo）来控制一系列歌曲的播放。[官网](#)
- [polyplayer](#)：将 YouTube、Soundcloud 和 Vimeo 播放器的 API 统一成一套。[官网](#)
- [flowplayer](#)：HTML5 视频播放器 [官网](#)、[Github](#)
- [mediaelement](#)：让 HTML5、Flash 播放器和模仿 HTML5 媒介元素 API 的 Silverlight shim，在所有浏览器拥有一致的 UI。[官网](#)、[Github](#)
- [SoundJS](#)：让音频在 web 上运行更简单的库。它为不同浏览器提供了一致的 API。[官网](#)

动画

- [velocity](#)：加速 JavaScript 动画。[官网](#)
- [jquery.transit](#)：拥有超级流畅的 CSS3 变换和过渡的 jQuery 插件。[官网](#)
- [impress.js](#)：在 HTML 文档里，运用 CSS3 变换和过渡制作类似 Prezi 的展现效果。[官网](#)
- [bounce.js](#)：可以立刻创建有趣的 CSS3 动画。[官网](#)
- [GreenSock-JS](#)：适用于所有主流浏览器的高性能 HTML5 动画。[官网](#)
- [TransitionEnd](#)：TransitionEnd 是一个运用 transitionend 事件的、跨浏览器的库。[官网](#)
- [Dynamics.js](#)：用于创建基于物理知识的 CSS 动画库。[官网](#)

图片处理

- [lena.js](#)：拥有滤镜和实用功能的图像处理库。[官网](#)
- [pica](#)：高质量地调整图片大小（拥有快速的、纯 JS 实现的 Lanczos 滤镜算法）。[官网](#)
- [cropper](#)：一个简单的图像裁剪 jQuery 插件。[官网](#)

ECMAScript 6

- [es6features](#)：ECMAScript 6 特性概述。[官网](#)
- [es6-features](#)：ECMAScript 6: 特性概述和比较。[官网](#)
- [ECMAScript 6 compatibility table](#)：Compatibility tables 展示了各种平台上所有 ECMAScript 6 特性的支持程度。[官网](#)
- [Babel \(Formerly 6to5\)](#)：将 ES6+ 代码转换成纯 ES5。[官网](#)
- [Traceur compiler](#)：ES6 特性转 ES5。包括 classes、generators、promises、

destructuring 官网 patterns、default parameters 等。

软件开发工具包(SDK)

- javascript-sdk-design：从工作和个人经验中提炼出来的 JavaScript SDK 设计指导。[官网](#)

大杂烩

- echo：利用 data-* 属性延迟加载图片。[官网](#)
- picturefill：响应式图片显示插件，使浏览器支持 srcset、size 属性。[官网](#)
- platform.js：一个平台检测库，几乎适用于所有 JavaScript 平台。[官网](#)
- json3：一个现代 JSON 实现库，几乎兼容所有 JavaScript 平台。[官网](#)
- Logical Or Not：一个关于 JavaScript 特性的游戏。[官网](#)
- BitSet.js：实现位向量的 JavaScript 库。[官网](#)

精品阅读

- braziljs/js-the-right-way：[官网](#)
- JSbooks：[官网](#)
- Superhero.js：关于创建、测试和维护一个大型 JavaScript 代码库的资源集。[官网](#)
- 《前端开发者都应知道的 jQuery 小技巧》
- 《常用的 Javascript 设计模式》
- 《10 个 jQuery 图表插件推荐》
- 《理解 JavaScript 原型》
- 《只有20行Javascript代码！手把手教你写一个页面模板引擎》
- 《编写快速、高效的JavaScript代码》
- 《45个实用的JavaScript技巧、窍门和最佳实践》
- 《为现代JavaScript开发做好准备》
- 《给JavaScript初学者的24条最佳实践》

资源

有影响力的书

具有广泛影响且值得阅读的前端经典书籍。

* 《[Limu：JavaScript 的那些书](#)》

from:<https://github.com/jobbole/awesome-javascript-cn>

前言

了解前端必备的优化内容，这作为开发的标准的话，会让我们养成好的编码风格，并做到后期优化内容更少。下边是在学习前端优化记录的笔记内容。推荐书《高性能网站建设指南》，以下只是简单介绍。

1、尽可能的减少 **HTTP** 请求数

前端页面初始化的时候，会在服务器下载外部文件，比如图片、js、css文件等，Http请求次数越多，消耗的时间越长，有效的将图片和js\css文件合并，减少Http的请求可以提升性能优化

2、使用**CDN**（内容分发网络）

CDN 意思是尽可能避免互联网上可能影响数据传输速度和稳定性的瓶颈和环节，使内容传输的更快、更稳定。

3、添加**Expire/Cache-Control**头

`expire`头 的内容是一个时间值，值就是资源的本地的过期时间、存在本地。在本地缓存阶段，找到对应的资源值，当前时间还没超过资源的过期时间，就直接使用这个资源，不会发送http请求。

`Cache-Control` 是http协议中常用的头部之一，顾名思义，他是负责控制页面缓存机制，如果该头部指示缓存，缓存的内容也会缓存在本地，操作流程和`expire`相似，但是也有不同的地方，`cache-control`有更多的选项，而且也有更多的处理方式。

4、启用**Gzip**压缩

HTML、php、js、css、xml、txt都可以使用Gzip压缩使得文件变小，提升文件传输速度。

5、把**CSS**放在页面最上面

为了提高浏览器的渲染性能，避免页面出现空白或者闪烁问题，应该将CSS放在页面顶部，或者head里边。

6、将**script**放在页面最下面

和DOM加载顺序有关，由于script标签的引入和加载消耗时间，而浏览器渲染的时候，CSS和script标签其实都是阻塞的，浏览器加载script的时候，下载完毕并执行，这个过程会阻塞进程。文件大的时候影响着页面的渲染速度和效果，所以将script标签放到页面底部，不影响页面的加载

7、避免在CSS中使用Expressions

CSS中使用Expressions，就是CSS表达式（CSS属性和JavaScript表达式）。在页面显示和缩放、页面滚动、移动的时候浏览器会计算CSS Expressions，移动鼠标的时候也会计算，这个网上可以搜到计算的方法，可以测试一下就发现很恐怖。

8、把JavaScript和CSS都放到外部文件中

提出的好处：提高了js和css的复用性，减小页面体积，提高了js和css的可维护性，可作为文件被浏览器单独缓存；坏处：文件多的时候会增加请求数量

放到页面好处：减少页面请求、加快页面渲染速度；坏处：不利于维护，可复用性差

写在页面内的情况：

- > 1、只应用于一个页面
- > 2、不经常被访问到
- > 3、脚本和样式很少

9、减少DNS查询

当浏览器访问www.giscafer.com的时候，浏览器会将域名解析为IP地址，然后根据DNS查找网站服务器，这个期间会消耗时间，有些网站是将js和img等放在多域下，所以DNS查询多的时候就需要考虑减少dns查询或者缓存来解决

10、压缩JavaScript和CSS

通过网上一些在线或者其他根据来压缩js、css文件，已减少文件体积，提升网页下载文件速度；

11、避免重定向

http协议中重定向有301（页面被移动到了另外的位置）和302（被找到了）两种状态码。另外301表示永久重定向，302表示临时重定向。对搜索引擎来说，爬虫会不定期的搜索站点，如果站点使用了301重定向，搜索引擎会将（a——>b）b记录下来，删除a不再去爬虫。当

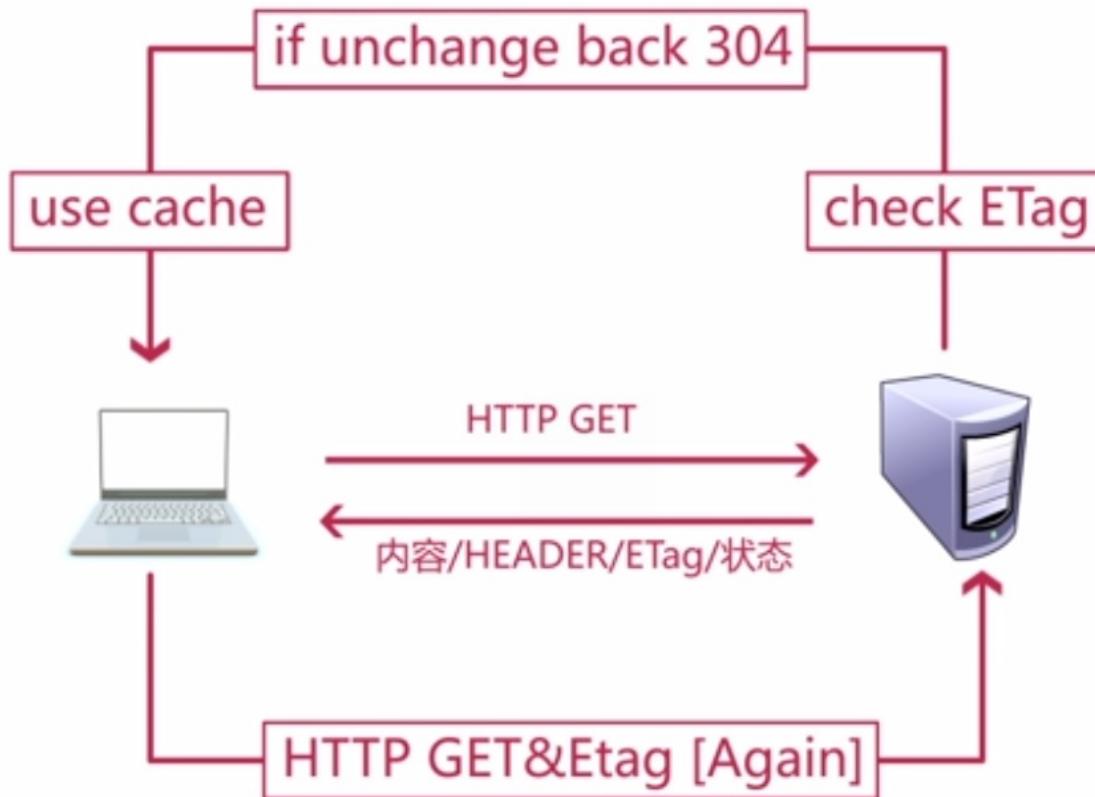
然，简单的从重定向来看，在这个过程中会消耗时间，所以不使用的话就尽量避免；另外特别注意的细微差别是，比如 `http://giscafer.com` 这个地址，就暗藏了重定向了，正确的为 `http://giscafer.com/` 才是直接访问的。其他的地址类似，一般后边加上 /

12、移除重复的脚本

这也是个编码习惯和质量的问题；当重复的脚本下有相同的全局变量时，这个会影响到其他代码，或者是被全局污染了。js 前端代码如果使用模块化管理的方式的话就避免了污染的情况。

13、配置实体标签（ETag）

ETag——Entity Tag（实体标签），属于HTTP协议，受Web服务支持。使用特殊的字符串，来标识某个请求资源版本



当每次Http请求的时候，服务器会判断该次请求的数据是否做过修改，如果没有修改过，则返回304状态码告诉浏览器，浏览器继续使用原来版本的资源，如果修改过了，则重新ETag标识，此时返回状态码是200。

14、使AJAX缓存

Ajax请求也是会消耗时间的，所以有时候也是需要缓存，但AJAX的POST请求的时候，是不被缓存的，而GET请求时可以被缓存的。

GET	POST
把参数数据队列加到提交表单的ACTION属性所指的URL中，值和表单内各个字段一一对应，在URL中可以看到。	通过HTTP POST机制，将表单内各个字段与其内容放置在HTML HEADER内一起传送到ACTION属性所指的URL地址。用户看不到这个过程。
服务器端用Request.QueryString获取变量的值。	服务器端用Request.Form获取提交的数据。
传送的数据量较小，不能大于2KB	传送的数据量较大，一般被默认为不受限制。但理论上，因服务器的不同而异。
安全性非常低。	安全性较高。
<form method="get" action="a.asp?b=b"> 跟<form method="get" action="a.asp"> 是一样的，也就是说，method为get时 action页面后边带的参数列表会被忽视。	<form method="post" action="a.asp?b=b"> 跟<form method="post" action="a.asp"> 是不一样的。
它会将数据添加到URL中，通过这种方式传递到服务器，通常利用一个问号？代表URL地址的结尾与数据参数的开端，后面的参数每一个数据参数以“名称=值”的形式出现，参数与参数之间利用一个连接符&来区分。	数据是放在HTTP主体中的，其组织方式不只一种，有&连接方式，也有分隔符方式，可隐藏参数，传递大批数据，比较方便。

15、Yslow网站性能优化工具

最后推荐Yslow工具，其实基于Firebug上使用的，如果是使用Firefox浏览器可以直接安装插件，Google chrome的话先安装FirebugLiteforChrome插件再安装Yslow

JIT 与 GC 优化

>

- untyped (无类型) 。
 - JAVASCRIPT 是个无类型的语言，这导致了如 `x=y+z` 这种表达式可以有很多含义。
 - `y` , `z` 是数字，则 `+` 表示加法。
 - `y` , `z` 是字符串，则 `+` 表示字符串连接。
 - 而 JS 引擎内部则使用“细粒度”的类型，比如：
 - 32-bit* integer。
 - 64-bit* floating-point。
 - 这就要求 js 类型 -js 引擎类型，需要做“boxed/unboxed（装箱/解箱）”，在处理一次 `x=y+z` 这种计算，需要经过的步骤如下。
 - 从内存，读取 `x=y+z` 的操作符。
 - 从内存，读取 `y` , `z` 。
 - 检查 `y` , `z` 类型，确定操作的行为。
 - `unbox y, z` 。
 - 执行操作符的行为。
 - `box x` 。
 - 把 `x` 写入内存。> 只有第 5 步骤是真正有效的操作，其他步骤都是为第 5 步骤做准备/收尾，JAVASCRIPT 的 untyped 特性很好用，但也为此付出了很大的性能代价。
- JIT 。
 - 先看看 JIT 对 untyped 的优化，在 JIT 下，执行 `x=y+z` 流程。
 1. 从内存，读取 `x=y+z` 的操作符。
 2. 从内存，读取 `y` , `z` 。
 3. 检查 `y` , `z` 类型，确定操作的行为。
 4. `unbox y, z` 。
 5. 执行操作符的行为。
 6. `box x` 。
 7. 把 `x` 写入内存。

其中 1 , 2 步骤由 CPU 负责，7 步骤 JIT 把结果保存在寄存器里。但可惜不是所有情况都能使用 JIT，当 `number+number` , `string+string` 等等可以使用 JIT，但特殊情况，如：`number+undefined` 就不行了，只能走旧解析器。
 - 新引擎还对“对象属性”访问做了优化，解决方案叫 inline caching，简称：IC。简单的说，就是做 cache。但如果当 list 很大时，这种方案反而影响效率。>

- Type-specializing JIT 引擎用来处理 typed 类型（声明类型）变量，但 JAVASCRIPT 都是 untype 类型的。
- Type-specializing JIT 的解决方案是：
 - 先通过扫描，监测类型。
 - 通过编译优化（优化对象不仅仅只是“类型”，还包括对JS代码的优化，但核心是类型优化），生成类型变量。
 - 再做后续计算。
- Type-specializing JIT 的执行 `x=y+z` 流程：
 - 从内存，读取 `x=y+z` 的操作符。
 - 从内存，读取 `y`，`z`。
 - 检查 `y`，`z` 类型，确定操作的行为。
 - `unbox y, z`。
 - 执行操作符的行为。
 - `box x`。
 - 把 `x` 写入内存。

代价是：

- 前置的扫描类型
- 编译优化。

所以·Type-specializing JIT·的应用是有选择性，选择使用这个引擎的场景包括：

- 热点代码。
- 通过启发式算法估算出来的有价值的代码。

另外，有2点也需要注意：

- 当变量类型发生变化时，引擎有2种处理方式：
 - 少量变更，重编译，再执行。
 - 大量变更，交给JIT执行。
- 数组，object properties，闭包变量 不在优化范畴之列。

AJAX 优化

>

- 缓存 AJAX：
 - 异步 并不等于 即时。
- 请求使用 GET：
 - 当使用 XMLHttpRequest 时，而URL长度不到 2K，可以使用 GET 请求数据，GET 相

比 POST 更快速。

- POST 类型请求要发送两个 TCP 数据包。
 - 先发送文件头。
 - 再发送数据。
- GET 类型请求只需要发送一个 TCP 数据包。
 - 取决于你的 cookie 数量。

COOKIE 专题

>

- 减少 COOKIE 的大小。
- 使用无 COOKIE 的域。
 - 比如图片 css 等静态文件放在静态资源服务器上并配置单独域名，客户端请求静态文件的时候，减少 COOKIE 反复传输时对主域名的影响。

DOM 优化

>

- 优化节点修改。
 - 使用 cloneNode 在外部更新节点然后再通过 replace 与原始节点互换。>

```
var orig = document.getElementById('container');
var clone = orig.cloneNode(true);
var list = ['foo', 'bar', 'baz'];
var content;
for (var i = 0; i < list.length; i++) {
  content = document.createTextNode(list[i]);
  clone.appendChild(content);
}
orig.parentNode.replaceChild(clone, orig);
```

- 优化节点添加多个节点插入操作，即使在外面设置节点的元素和风格再插入，由于多个节点还是会引发多次 reflow。
- 优化的方法是创建 DocumentFragment，在其中插入节点后再添加到页面。
 - 如 JQuery 中所有的添加节点的操作如 append，都是最终调用 DocumentFragment 来实现的，>

```
createSafeFragment(document) {
    var list = nodeNames.split( " | " ),
        safeFrag = document.createDocumentFragment();
```

```
if (safeFrag.createElement) {
    while (list.length) {
        safeFrag.createElement(
            list.pop());
    }
}
return safeFrag;
};
```

- 优化 `css` 样式转换。如果需要动态更改 `CSS` 样式，尽量采用触发 `reflow` 次数较少的方式。
- 如以下代码逐条更改元素的几何属性，理论上会触发多次 `reflow` 。>

```
element.style.fontWeight = 'bold' ;
element.style.marginLeft= '30px' ;
element.style.marginRight = '30px' ;
```

- 可以通过直接设置元素的 `className` 直接设置，只会触发一次 `reflow` 。>

```
element.className = 'selectedAnchor' ;
```

- 减少 `DOM` 元素数量
- 在 `console` 中执行命令查看 `DOM` 元素数量。>
- `document.getElementsByTagName('*').length`
- 正常页面的 `DOM` 元素数量一般不应该超过 `1000` 。
- `DOM` 元素过多会使 `DOM` 元素查询效率，样式表匹配效率降低，是页面性能最主要的瓶颈之一。
 - `DOM` 操作优化。
- `DOM` 操作性能问题主要有以下原因。
 - `DOM` 元素过多导致元素定位缓慢。
 - 大量的 `DOM` 接口调用。
 - `JAVASCRIPT` 和 `DOM` 之间的交互需要通过函数 `API` 接口来完成，造成延时，尤其是在循环语句中。
 - `DOM` 操作触发频繁的 `reflow(layout)` 和 `repaint` 。
 - `layout` 发生在 `repaint` 之前，所以 `layout` 相对来说会造成更多性能损耗。
 - `reflow(layout)` 就是计算页面元素的几何信息。

- `repaint` 就是绘制页面元素。
- 对 `DOM` 进行操作会导致浏览器执行回流 `reflow`。
- 解决方案。
 - 纯 `JAVASCRIPT` 执行时间是很短的。
 - 最小化 `DOM` 访问次数，尽可能在 `js` 端执行。
 - 如果需要多次访问某个 `DOM` 节点，请使用局部变量存储对它的引用。
 - 谨慎处理 `HTML` 集合（`HTML` 集合实时连系底层文档），把集合的长度缓存到一个变量中，并在迭代中使用它，如果需要经常操作集合，建议把它拷贝到一个数组中。
 - 如果可能的话，使用速度更快的 `API`，比如 `querySelectorAll` 和 `firstElementChild`。
 - 要留意重绘和重排。
 - 批量修改样式时，离线操作 `DOM` 树。
 - 使用缓存，并减少访问布局的次数。
 - 动画中使用绝对定位，使用拖放代理。
 - 使用事件委托来减少事件处理器的数量。
 - 优化 `DOM` 交互在 `JAVASCRIPT` 中，`DOM` 操作和交互要消耗大量时间，因为它们往往需要重新渲染整个页面或者某一个部分。
- 最小化 现场更新。
 - 当需要访问的 `DOM` 部分已经已经被渲染为页面中的一部分，那么 `DOM` 操作和交互的过程就是再进行一次 现场更新。
 - 现场更新 是需要针对 现场（相关显示页面的部分结构）立即进行更新，每一个更改（不管是插入单个字符还是移除整个片段），都有一个性能损耗。
 - 现场更新进行的越多，代码完成执行所花的时间也越长。
- 多使用 `innerHTML`。
 - 有两种在页面上创建 `DOM` 节点的方法：
 - 使用诸如 `createElement()` 和 `appendChild()` 之类的 `DOM` 方法。
 - 使用 `innerHTML`。
 - 当使用 `innerHTML` 设置为某个值时，后台会创建一个 `HTML` 解释器，然后使用内部的 `DOM` 调用用来创建 `DOM` 结构，而非基于 `JAVASCRIPT` 的 `DOM` 调用。由于内部方法是编译好的而非解释执行，故执行的更快。
 - 对于小的 `DOM` 更改，两者效率差不多，但对于大的 `DOM` 更改，`innerHTML` 要比标准的 `DOM` 方法创建同样的 `DOM` 结构快得多。
 - 回流 `reflow`。
 - 发生场景。
 - 改变窗体大小。
 - 更改字体。

- 添加移除stylesheet块。
- 内容改变哪怕是输入框输入文字。
- CSS虚类被触发如 :hover。
- 更改元素的className。
- 当对DOM节点执行新增或者删除操作或内容更改时。
- 动态设置一个style样式时（比如element.style.width="10px"）。
- 当获取一个必须经过计算的尺寸值时，比如访问offsetWidth、clientHeight或者其他需要经过计算的CSS值。
- 解决问题的关键，就是限制通过DOM操作所引发回流的次数。
 - 在对当前DOM进行操作之前，尽可能多的做一些准备工作，保证N次创建，1次写入。
 - 在对DOM操作之前，把要操作的元素，先从当前DOM结构中删除：
 - 通过removeChild()或者replaceChild()实现真正意义上的删除。
 - 设置该元素的display样式为“none”。
 - 每次修改元素的style属性都会触发回流操作。

```
element.style.backgroundColor = "blue";
```

- 使用更改 className 的方式替换 style.xxx=xxx 的方式。
- 使用 style.cssText = ''；一次写入样式。
- 避免设置过多的行内样式。
- 添加的结构外元素尽量设置它们的位置为 fixed 或 absolute。
- 避免使用表格来布局。
- 避免在 css 中使用 JavaScript expressions(IE only)。
- 将获取的 DOM 数据缓存起来。这种方法，对获取那些会触发回流操作的属性（比如 offsetWidth 等）尤为重要。
- 当对HTMLCollection对象进行操作时，应该将访问的次数尽可能的降至最低，最简单的，你可以将length属性缓存在一个本地变量中，这样就能大幅度的提高循环的效率。

eval优化

>

- 避免 eval ：
 - eval 会在时间方面带来一些效率，但也有很多缺点。
 - eval 会导致代码看起来更脏。
 - eval 会需要消耗大量时间。
 - eval 会逃过大多数压缩工具的压缩。

HTML 优化

- 插入 `HTML` 。
 - `JavaScript` 中使用 `document.write` 生成页面内容会效率较低，可以找一个容器元素，比如指定一个 `div`，并使用 `innerHTML` 来将 `HTML` 代码插入到页面中。
- 避免空的 `src` 和 `href` 。
 - 当 `link` 标签的 `href` 属性为空、`script` 标签的 `src` 属性为空的时候，浏览器渲染的时候会把当前页面的 `URL` 作为它们的属性值，从而把页面的内容加载进来作为它们的值。
- 为文件头指定 `Expires` 。
 - 使内容具有缓存性，避免了接下来的页面访问中不必要的HTTP请求。
- 重构HTML，把重要内容的优先级提高。
- Post-load（次要加载）不是必须的资源。
- 利用预加载优化资源。
- 合理架构，使DOM结构尽量简单。
- 利用 `LocalStorage` 合理缓存资源。
- 尽量避免CSS表达式和滤镜。
- 尝试使用`defer`方式加载JS脚本。
- 新特性：`will-change`，把即将发生的改变预先告诉浏览器。
- 新特性Beacon，不堵塞队列的异步数据发送。
- 不同之处：网络缓慢，缓存更小，不令人满意的浏览器处理机制。
- 尽量多地缓存文件。
- 使用HTML5 Web Workers来允许多线程工作。
- 为不同的Viewports设置不同大小的Content。
- 正确设置可Tap的目标的大小。
- 使用响应式图片。
- 支持新接口协议（如HTTP2）。
- 未来的缓存离线机制：`Service Workers`。
- 未来的资源优化Resource Hints(`preconnect`, `preload`, 和`prerender`)。
- 使用Server-sent Events。
- 设置一个Meta Viewport。

js 载入优化

- 加快JavaScript装入速度的工具：
 - Lab.js
 - 借助LAB.js（装入和阻止JavaScript），你就可以并行装入JavaScript文

件，加快总的装入过程。此外，你还可以为需要装入的脚本设置某个顺序，那样就能确保依赖关系的完整性。此外，开发者声称其网站上的速度提升了2倍。

- 使用适当的CDN：

- 现在许多网页使用内容分发网络（CDN）。它可以改进你的缓存机制，因为每个人都可以使用它。它还能为你节省一些带宽。你很容易使用ping检测或使用Firebug调试那些服务器，以便搞清可以从哪些方面加快数据的速度。选择CDN时，要照顾到你网站那些访客的位置。记得尽可能使用公共存储库。
- 网页末尾装入JavaScript：

- 也可以在头部分放置需要装入的一些JavaScript，但是前提是它以异步方式装入。
- 异步装入跟踪代码：脚本加载与解析会阻塞HTML渲染，可以通过异步加载方式来避免渲染阻塞，步加载的方式很多，比较通用的方法如下。

```

var _gaq = _gaq || [];
_gaq.push(['_setAccount', 'UA-XXXXXX-XX']);
_gaq.push(['_trackPageview']);
(function() {
    var ga = document.createElement('script'); ga.type = 'text/JavaScript';
    ga.async = true;
    ga.src = ('https:' == document.location.protocol ? 'https://ssl' : 'htt
p://www') + '.google-analytics.com/ga.js';
    var s = document.getElementsByTagName('script')[0]; s.parentNode.insert
Before(ga, s);
})();

```

或者

```

function loadjs (script_filename){
    var script = document.createElement( 'script' );
    script.setAttribute( 'type' , 'text/javascript' );
    script.setAttribute( 'src' , script_filename);
    script.setAttribute( 'id' , 'script-id' );

    scriptElement = document.getElementById( 'script-id' );
    if (scriptElement){
        document.getElementsByTagName( 'head' )[0].removeChild(scriptEleme
nt);
    }
    document.getElementsByTagName( 'head' )[0].appendChild(script);
}

var script = 'scripts/alert.js' ;
loadjs(script);

```

- 把你的JavaScript打包成PNG文件

- 将JavaScript/css数据打包成PNG文件。之后进行拆包，只要使用画布API的 `getImageData()`。可以在不缩小数据的情况下，多压缩35%左右。而且是无损压缩，对比较庞大的脚本来说，在图片指向画布、读取像素的过程中，你会觉得有“一段”装入时间。
- 设置Cache-Control和Expires头

通过Cache-Control和Expires头可以将脚本文件缓存在客户端或者代理服务器上，可以减少脚本下载的时间。

Expires格式：

```
Expires = "Expires" ":" HTTP-date
Expires: Thu, 01 Dec 1994 16:00:00 GMT
Note: if a response includes a Cache-Control field with the max-age directive
      that directive overrides the
      Expires field.
```

Cache-Control格式：

```
Cache-Control = "Cache-Control" ":" 1#cache-directive
Cache-Control: public
```

具体的标准定义可以参考http1.1中的定义，简单来说Expires控制过期时间是多久，Cache-Control控制什么地方可以缓存。

with 优化

- 尽可能地少用 `with` 语句，因为它会增加 `with` 语句以外的数据的访问代价。
- 避免使用 `with`

`with` 语句将一个新的可变对象推入作用域链的头部，函数的所有局部变量现在处于第二个作用域链对象中，从而使局部变量的访问代价提高。

```

var person = {
    name: "Nicholas",
    age: 30
}
function displayInfo() {
    var count = 5;
    with (person) {
        alert(name + ' is ' + age);
        alert('count is ' + count);
    }
}

```

事件优化

- 使用事件代理

- 当存在多个元素需要注册事件时，在每个元素上绑定事件本身就会对性能有一定损耗。
- 由于DOM Level2事件模型中所有事件默认会传播到上层文档对象，可以借助这个机制在上层元素注册一个统一事件对不同子元素进行相应处理。

捕获型事件先发生。两种事件流会触发DOM中的所有对象，从document对象开始，也在document对象结束。

```

<li id="post-1">Item 1
<li id="post-2">Item 2
<li id="post-3">Item 3
<li id="post-4">Item 4
<li id="post-5">Item 5
<li id="post-6">Item 6

```

```

</li></ul> // Get the element, add a click listener...
document.getElementById("parent-list").addEventListener("click",function(e) {

    // e.target is the clicked element!
    // If it was a list item
    if(e.target && e.target.nodeName == "LI") {
        // List item found! Output the ID!
        console.log("List item ",e.target.id.replace("post-")," was clicked
    });
})

```

代码优化

- 优化原则：

JS与其他语言不同在于它的执行效率很大程度是取决于 JS engine 的效率。除了引擎实现的优劣外，引擎自己也会为一些特殊的代码模式采取一些优化的策略。例如 FF、Opera 和 Safari 的 JAVASCRIPT 引擎，都对字符串的拼接运算（+）做了特别优化。所以应该根据不同引擎进行不同优化。

而如果做跨浏览器的web编程，则最大的问题是在于IE6（JScript 5.6），因为在不打hotfix的情况下，JScript引擎的垃圾回收的bug，会导致其在真实应用中的 performance 跟其他浏览器根本不在一个数量级上。因此在这种场合做优化，实际上就是为JScript做优化，所以第一原则就是只需要为IE6（未打补丁的JScript 5.6或更早版本）做优化。

- JS优化总是出现在大规模循环的地方：

这倒不是说循环本身有性能问题，而是循环会迅速放大可能存在的性能问题，所以第二原则就是以大规模循环体为最主要优化对象。

以下的优化原则，只在大规模循环中才有意义，在循环体之外做此类优化基本上是没有意义的。

目前绝大多数JS引擎都是解释执行的，而解释执行的情况下，在所有操作中，函数调用的效率是较低的。此外，过深的prototype继承链或者多级引用也会降低效率。JScript中，10级引用的开销大体是一次空函数调用开销的1/2。这两者的开销都远远大于简单操作（如四则运算）。

- 尽量避免过多的引用层级和不必要的多次方法调用：

特别要注意的是，有些情况下看似是属性访问，实际上是方法调用。例如所有DOM的属性，实际上都是方法。在遍历一个NodeList的时候，循环 条件对于nodes.length的访问，看似属性读取，实际上是等价于函数调用的。而且IE DOM的实现上，childNodes.length每次是要通过内部遍历重新计数的。（My god，但是这是真的！因为我测过，childNodes.length的访问时间与childNodes.length的值成正比！）这非常耗费。所以 预先把nodes.length保存到js变量，当然可以提高遍历的性能。

同样是函数调用，用户自定义函数的效率又远远低于语言内建函数，因为后者是对引擎本地方法的包装，而引擎通常是c,c++,java写的。进一步，同样的功能，语言内建构造的开销通常又比内建函数调用要效率高，因为前者在JS代码的parse阶段就可以确定和优化。

- 尽量使用语言本身的构造和内建函数：

这里有一个例子是高性能的String.format方法。String.format传统的实现方式是用String.replace(regex, func)，在pattern包含n个占位符（包括重复的）时，自定义函数func就被调用n次。而这个高性能实现中，每次format调用所作的只是一次Array.join然后一次String.replace(regex, string)的操作，两者都是引擎内建方法，而不会有任何自定义函数调用。两次内建方法调用和n次的自定义方法调用，这就是性能上的差别。

同样是内建特性，性能上也还是有差别的。例如在JScript中对于arguments的访问性能就很差，几乎赶上一次函数调用了。因此如果一个 可变参数的简单函数成为性能瓶颈的时候，可以将其内部做一些改变，不要

访问`arguments`，而是通过对参数的显式判断来处理，比如：

```
function sum() {
    var r = 0;
    for (var i = 0; i < arguments.length; i++) {
        r += arguments[i];
    }
    return r;
}
```

这个`sum`通常调用的时候个数是较少的，我们希望改进它在参数较少时的性能。如果改成：

```
function sum() {
    switch (arguments.length) {
        case 1: return arguments[0];
        case 2: return arguments[0] + arguments[1];
        case 3: return arguments[0] + arguments[1] + arguments[2];
        case 4: return arguments[0] + arguments[1] + arguments[2] + arguments[3];
    }
    default:
        var r = 0;
        for (var i = 0; i < arguments.length; i++) {
            r += arguments[i];
        }
        return r;
    }
}
```

其实并不会有多少提高，但是如果改成：

```
function sum(a, b, c, d, e, f, g) {
    var r = a ? b ? c ? d ? e ? f ? a + b + c + d + e + f : a + b + c + d + e : a + b + c + d : a + b + c : a + b : a : 0;
    if (g === undefined) return r;
    for (var i = 6; i < arguments.length; i++) {
        r += arguments[i];
    }
    return r;
}
```

就会提高很多（至少快1倍）。

代码压缩

- 代码压缩工具

精简代码就是将代码中的 空格 和 注释 去除，也有更进一步的会对变量名称混淆、精简。根据统计精简后文件大小会平均减少 21%，即使 Gzip 之后文件也会减少 5%。

- YUICompressor
- Dean Edwards Packer
- JSMin
- GZip 压缩

- GZip 缩短在浏览器和服务器之间传送数据的时间，缩短时间后得到标题是 Accept-Encoding : gzip , deflate 的一个文件。不过这种压缩方法同样也有缺点。
 - 它在服务器端和客户端都要占用处理器资源（以便压缩和解压缩）。
 - 占用磁盘空间。
- Gzip 通常可以减少 70% 网页内容的大小，包括脚本、样式表、图片等任何一个文本类型的响应，包括 XML 和 JSON 。 Gzip 比 deflate 更高效，主流服务器都有相应的压缩支持模块。
- Gzip 的工作流程为
 - 客户端在请求 Accept-Encoding 中声明可以支持 Gzip 。
 - 服务器将请求文档压缩，并在 Content-Encoding 中声明该回复为 Gzip 格式。
 - 客户端收到之后按照 Gzip 解压缩。

- Closure compiler

作用域链和闭包优化

>

- 作用域。

- 作用域(scope)是 JAVASCRIPT 编程中一个重要的 运行机制，在 JAVASCRIPT 同步和异步编程以及 JAVASCRIPT 内存管理中起着至关重要的作用。
- 在 JAVASCRIPT 中，能形成作用域的有如下几点。
 - 函数的调用
 - with 语句
 - with 会创建自己的作用域，因此会增加其中执行代码的作用域的长度。

■ 全局作用域。 > 以下代码为例： > var foo = function() {

```
var local = {};
```

```
; foo(); console.log(local); //=> undefined > var bar = function() {
```

```
local = {};
```

}; bar(); console.log(local); //=> {} > /这里我们定义了**foo()**函数和**bar()**函数，他们的意图都是为了定义一个名为**local**的变量。在**foo()**函数中，我们使用**var**语句来声明定义了一个**local**变量，而因为函数体内部会形成一个作用域，所以这个变量便被定义到该作用域中。而且**foo()**函数体内并没有做任何作用域延伸的处理，所以在该函数执行完毕后，这个**local**变量也随之被销毁。而在外层作用域中则无法访问到该变量。而在**bar()**函数内，**local**变量并没有使用**var**语句进行声明，取而代之的是直接把**local**作为全局变量来定义。故外层作用域可以访问到这个变量。/ > local = {}; // 这里的定义等效于 global.local = {};

- 作用域链

- 在 **JAVASCRIPT** 编程中，会遇到多层函数嵌套的场景，这就是典型的作用域链的表示。>

```
function foo() {
    var val = 'hello';
    function bar() {
        function baz() {
            global.val = 'world';
        };
        baz();
        console.log(val); //=> hello
    };
    bar();
}
foo();
```

>

/**在`JAVASCRIPT`中，变量标识符的查找是从当前作用域开始向外查找，直到全局作用域为止。所以`JAVASCRIPT`代码中对变量的访问只能向外进行，而不能逆而行之。**baz()**函数的执行在全局作用域中定义了一个全局变量**val**。而在**bar()**函数中，对**val**这一标识符进行访问时，按照从内到外的查找原则：在**bar**函数的作用域中没有找到，便到上一层，即**foo()**函数的作用域中查找。然而，使大家产生疑惑的关键就在这里：本次标识符访问在**foo()**函数的作用域中找到了符合的变量，便不会继续向外查找，故在**baz()**函数中定义的全局变量**val**并没有在本次变量访问中产生影响。**/

- 减少作用域链上的查找次数

- **JAVASCRIPT** 代码在执行的时候，如果需要访问一个变量或者一个函数的时候，它需要遍历当前执行环境的作用域链，而遍历是从这个作用域链的前端一级一级的向后遍历，直到全局执行环境。>

```

/**效率低*/
for(var i = 0; i < 10000; i++){
    var but1 = document.getElementById("but1");
}
/**效率高*/
/**避免全局查找*/
var doc = document;
for(var i = 0; i < 10000; i++){
    var but1 = doc.getElementById("but1");
}
/**上面代码中，第二种情况是先把全局对象的变量放到函数里面先保存下来，然后直接访问这个变量
，而第一种情况是每次都遍历作用域链，直到全局环境，我们看到第二种情况实际上只遍历了一次，而第一种情况却是每次都遍历了，而且这种差别在多级作用域链和多个全局变量的情况下还会表现的非常明显
。在作用域链查找的次数是`O(n)`。通过创建一个指向`document`的局部变量，就可以通过限制一次全局查找来改进这个函数的性能。**/

```

■ 闭包

- JAVASCRIPT 中的标识符查找遵循从内到外的原则。 >

```

function foo() {
    var local = 'Hello';
    return function() {
        return local;
    };
}
var bar = foo();
console.log(bar()); //=> Hello

```

>

/**这里所展示的让外层作用域访问内层作用域的技术便是闭包(Closure)。得益于高阶函数的应用，使foo()函数的作用域得到‘延伸’。foo()函数返回了一个匿名函数，该函数存在于foo()函数的作用域内，所以可以访问到foo()函数作用域内的local变量，并保存其引用。而因这个函数直接返回了local变量，所以在外层作用域中便可直接执行bar()函数以获得local变量。**/

- 闭包是 JAVASCRIPT 的高级特性，因为把带有内部变量引用的函数带出了函数外部，所以该作用域内的变量在函数执行完毕后的并不一定会被销毁，直到内部变量的引用被全部解除。所以闭包的应用很容易造成内存无法释放的情况。
- 良好的闭包管理。
 - 循环事件绑定、私有属性、含参回调等一定要使用闭包时，并谨慎对待其中的细节。
 - 循环绑定事件，我们假设一个场景：有六个按钮，分别对应六种事件，当用户点击按钮时，在指定的地方输出相应的事件。 >

```

var btns = document.querySelectorAll('.btn'); // 6 elements
var output = document.querySelector('#output');
var events = [1, 2, 3, 4, 5, 6];
// Case 1
for (var i = 0; i < btns.length; i++) {
    btns[i].onclick = function(evt) {
        output.innerText += 'Clicked ' + events[i];
    };
}
/**这里第一个解决方案显然是典型的循环绑定事件错误，这里不细说，详细可以参照我给一个网友的回答；而第二和第三个方案的区别就在于闭包传入的参数。**/
// Case 2
for (var i = 0; i < btns.length; i++) {
    btns[i].onclick = (function(index) {
        return function(evt) {
            output.innerText += 'Clicked ' + events[index];
        };
    })(i);
}
/**第二个方案传入的参数是当前循环下标，而后者是直接传入相应的事件对象。事实上，后者更适合在大量数据应用的时候，因为在JavaScript的函数式编程中，函数调用时传入的参数是基本类型对象，那么在函数体内得到的形参会是一个复制值，这样这个值就被当作一个局部变量定义在函数体的作用域内，在完成事件绑定之后就可以对events变量进行手工解除引用，以减轻外层作用域中的内存占用了。而且当某个元素被删除时，相应的事件监听函数、事件对象、闭包函数也随之被销毁回收。**/
// Case 3
for (var i = 0; i < btns.length; i++) {
    btns[i].onclick = (function(event) {
        return function(evt) {
            output.innerText += 'Clicked ' + event;
        };
    })(events[i]);
}

```

■ 避开闭包陷阱

- 闭包是个强大的工具，但同时也是性能问题的主要诱因之一。不合理的使用闭包会导致内存泄漏。
- 闭包的性能不如使用内部方法，更不如重用外部方法。
 - 由于IE 9 浏览器的DOM 节点作为COM 对象来实现，COM 的内存管理是通过引用计数的方式，引用计数有个难题就是循环引用，一旦DOM 引用了闭包(例如event handler)，闭包的上层元素又引用了这个DOM，就会造成循环引用从而导致内存泄漏。

■ 善用函数

- 使用一个匿名函数在代码的最外层进行包裹。> ;(function() {

```
// 主业务代码
```

}());> 有的甚至更高级一点：> ;(function(win, doc, \$, undefined) {

```
// 主业务代码

})(window, document, jQuery); > 甚至连如RequireJS, SeaJS, OzJS 等前端模块化加载解决方案，都是采用类似的形式：> /RequireJS/ define(['jquery'], function($) {

// 主业务代码

}); /SeaJS/ define('module', ['dep', 'underscore'], function($, _) {

// 主业务代码

}); > 被定义在全局作用域的对象，可能是会一直存活到进程退出的，如果是一个很大的对象，那就麻烦了。比如有的人喜欢在JavaScript中做模版渲染：> <?php

$db = mysqli_connect(server, user, password, 'myapp');
$topics = mysqli_query($db, "SELECT * FROM topics;");

?> <!doctype html>

<meta charset="UTF-8">
<title>你是猴子请来的逗比么？</title>

</head>

<ul id="topics"></ul>
<script type="text/tmpl" id="topic-tmpl">
<li class="topic">
<h1><%=title%></h1>
<p><%=content%></p>
</li>
</script>
<script type="text/javascript">
var data = <?php echo json_encode($topics); ?>;
var topicTmpl = document.querySelector('#topic-tmpl').innerHTML;
var render = function(tmpl, view) {
    var complied = tmpl
        .replace(/\n/g, '\\\\n')
        .replace(/<%=([\s\S]+?)%>/g, function(match, code) {
            return '' + escape(' + code + ') + '';
        });
    >
```

```

complied = [
    'var res = "";',
    'with (view || {}) {',
        'res = "' + complied + '";',
    '}',
    'return res;'
].join('\n');

>

var fn = new Function('view', complied);
return fn(view);
};

>

var topics = document.querySelector('#topics');
function init()
{
    data.forEach(function(topic) {
        topics.innerHTML += render(topicTpl, topic);
    });
}
init();
</script>

```

</body> </html> > 在从数据库中获取到的数据的量是非常大的话，前端完成模板渲染以后，`data`变量便被闲置在一边。可因为这个变量是被定义在全局作用域中的，所以 `JAVASCRIPT` 引擎不会将其回收销毁。如此该变量就会一直存在于老生代堆内存中，直到页面被关闭。可是如果我们作出一些很简单的修改，在逻辑代码外包装一层函数，这样效果就大不同了。当UI渲染完成之后，代码对`data`的引用也就随之解除，而在最外层函数执行完毕时，`JAVASCRIPT` 引擎就开始对其中的对象进行检查，`data`也就可以随之被回收。

内存专题

- `JAVASCRIPT` 的内存回收机制
 - 以`Google`的 `v8` 引擎为例，在 `v8` 引擎中所有的 `JAVASCRIPT` 对象都是通过 `堆` 来进行内存分配的。当我们在代码中 声明变量 并 赋值 时，`v8` 引擎就会在 `堆内存` 中分配一部分给这个 变量。如果已申请的 内存 不足以存储这个 变量 时，`v8` 引擎就会继续申请 内存 ，直到 堆 的大小达到了 `v8` 引擎的内存上限为止（默认情况下，`v8` 引擎的 `堆内存` 的大小上限在 `64位系统` 中为 `1464MB`，在 `32位系统` 中则为 `732MB`）。

- 另外，v8 引擎对堆内存中的 JAVASCRIPT 对象进行分代管理。
 - 新生代。
 - 新生代即存活周期较短的 JAVASCRIPT 对象，如临时变量、字符串等
 - 老生代。
 - 老生代则为经过多次垃圾回收仍然存活，存活周期较长的对象，如主控制器、服务器对象等。
- 垃圾回收算法。
 - 垃圾回收算法一直是编程语言的研发中是否重要的一环，而 v8 引擎所使用的垃圾回收算法主要有以下几种。
 - Scavange 算法：通过复制的方式进行内存空间管理，主要用于新生代的内存空间；
 - Mark-Sweep 算法和 Mark-Compact 算法：通过标记来对堆内存进行整理和回收，主要用于老生代对象的检查和回收。
- 对象进行回收。

- 引用。

- 当函数执行完毕时，在函数内部所声明的对象不一定就会被销毁。
- 引用(Reference)是 JAVASCRIPT 编程中十分重要的一个机制。
- 是指代码对对象的访问这一抽象关系，它与 C/C++ 的指针有点相似，但并非同物。引用同时也是 JAVASCRIPT 引擎在进行垃圾回收中最关键的一个机制。

```

var val = 'hello world';
function foo() {
    return function() {
        return val;
    };
}
global.bar = foo();

```

- 当代码执行完毕时，对象 val 和 bar() 并没有被回收释放，JAVASCRIPT 代码中，每个变量作为单独一行而不做任何操作，JAVASCRIPT 引擎都会认为这是对对象的访问行为，存在了对对象的引用。为了保证垃圾回收的行为不影响程序逻辑的运行，JAVASCRIPT 引擎不会把正在使用的对象进行回收。所以判断对象是否正在使用中的标准，就是是否仍然存在对该对象的引用。
- JAVASCRIPT 的引用是可以进行转移的，那么就有可能出现某些引用被带到了全局作用域，但事实上在业务逻辑里已经不需要对其进行访问了，这个时候就应该被回收，但是 JAVASCRIPT 引擎仍会认为程序仍然需要它。

- IE 下闭包引起跨页面内存泄露。
- JAVASCRIPT 的内存泄露处理
 - 给 DOM 对象添加的属性是一个对象的引用。

```
var MyObject = {};
document.getElementById('myDiv').myProp = MyObject;
```

解决方法：在window.onunload事件中写上：

```
document.getElementById('myDiv').myProp = null;
```

- DOM对象与JS对象相互引用。

```
function Encapsulator(element) {
    this.elementReference = element;
    element.myProp = this;
}
new Encapsulator(document.getElementById('myDiv'));
```

解决方法：在onunload事件中写上：

```
document.getElementById('myDiv').myProp = null;
```

- 给DOM对象用attachEvent绑定事件。

```
function doClick() {}
element.attachEvent("onclick", doClick);
```

解决方法：在onunload事件中写上：

```
element.detachEvent('onclick', doClick);
```

- 从外到内执行appendChild。这时即使调用removeChild也无法释放。

```
var parentDiv = document.createElement("div");
var childDiv = document.createElement("div");
document.body.appendChild(parentDiv);
parentDiv.appendChild(childDiv);
```

解决方法：从内到外执行appendChild:

```

var parentDiv = document.createElement("div");
var childDiv = document.createElement("div");
parentDiv.appendChild(childDiv);
document.body.appendChild(parentDiv);

```

- 反复重写同一个属性会造成内存大量占用(但关闭IE后内存会被释放)。

```

for(i = 0; i < 5000; i++) {
    hostElement.text = "asdfasdfsdf";
}

```

这种方式相当于定义了5000个属性，解决方法：无。

- 内存不是缓存。
 - 不要轻易将内存当作缓存使用。
 - 如果是很重要的资源，请不要直接放在内存中，或者制定过期机制，自动销毁过期缓存。
- CollectGarbage。
 - CollectGarbage是IE的一个特有属性，用于释放内存的使用方法，将该变量或引用对象设置为null或delete然后在进行释放动作，在做CollectGarbage前，要必需清楚的两个必备条件：（引用）。
 - 一个对象在其生存的上下文环境之外，即会失效。
 - 一个全局的对象在没有被执用(引用)的情况下，即会失效

动画优化

- 动画效果在缺少硬件加速支持的情况下反应缓慢，例如手机客户端。
 - 特效应该只在确实能改善用户体验时才使用，而不应用于炫耀或者弥补功能与可用性上的缺陷。
 - 至少要给用户一个选择可以禁用动画效果。
 - 设置动画元素为absolute或fixed。
 - position: static或position: relative元素应用动画效果会造成频繁的reflow。
 - position: absolute或position: fixed的元素应用动画效果只需要repaint。
 - 使用一个timer完成多个元素动画。
 - setInterval和setTimeout是两个常用的实现动画的接口，用以间隔更新元素的风格与布局。。

- 动画效果的帧率最优化的情况是使用一个 `timer` 完成多个对象的动画效果，其原因在于多个 `timer` 的调用本身就会损耗一定性能。

```
setInterval(function() {
    animateFirst('');
}, 10);
setInterval(function() {
    animateSecond('');
}, 10);
```

使用同一个 `timer`。

```
setInterval(function() {
    animateFirst('');
    animateSecond('');
}, 10);
```

- 以脚本为基础的动画，由浏览器控制动画的更新频率。

原型优化

- 通过原型优化方法定义。
 - 如果一个方法类型将被频繁构造，通过方法原型从外面定义附加方法，从而避免方法的重复定义。
 - 可以通过外部原型的构造方式初始化值类型的变量定义。（这里强调值类型的原因是，引用类型如果在原型中定义，一个实例对引用类型的更改会影响到其他实例。）
 - 这条规则中涉及到 JAVASCRIPT 中原型的概念，构造函数都有一个 `prototype` 属性，指向另一个对象。这个对象的所有属性和方法，都会被构造函数的实例继承。可以把那些不变的属性和方法，直接定义在 `prototype` 对象上。
 - 可以通过对象实例访问保存在原型中的值。
 - 不能通过对对象实例重写原型中的值。
 - 在实例中添加一个与实例原型同名属性，那该属性就会屏蔽原型中的属性。
 - 通过`delete`操作符可以删除实例中的属性。

变量专题

- 全局变量
 - 当一个变量被定义在全局作用域中，默认情况下 JAVASCRIPT 引擎就不会将其回收销毁。如此该变量就会一直存在于老生代堆内存中，直到页面被关闭。
 - 全局变量 缺点。
 - 使变量不易被回收。
 - 多人协作时容易产生混淆。
 - 在作用域链中容易被干扰。
 - 可以通过包装函数来处理 全局变量。
- 局部变量。
 - 尽量选用局部变量而不是全局变量。
 - 局部变量的访问速度要比全局变量的访问速度更快，因为全局变量其实是 window 对象的成员，而局部变量是放在函数的栈里的。
- 手工解除变量引用
 - 在业务代码中，一个变量已经确定不再需要了，那么就可以手工解除变量引用，以使其被回收。

```
var data = { /* some big data */ };
// ...
data = null;
```

- 变量查找优化。
 - 变量声明带上 var，如果声明变量忘记了 var，那么 JAVASCRIPT 引擎将会遍历整个作用域查找这个变量，结果不管找到与否，都会造成性能损耗。
 - 如果在上级作用域找到了这个变量，上级作用域变量的内容将被无声的改写，导致莫名其妙的错误发生。
 - 如果在上级作用域没有找到该变量，这个变量将自动被声明为全局变量，然而却都找不到这个全局变量的定义。
 - 慎用全局变量。
 - 全局变量需要搜索更长的作用域链。
 - 全局变量的生命周期比局部变量长，不利于内存释放。
 - 过多的全局变量容易造成混淆，增大产生bug的可能性。
 - 具有相同作用域变量通过一个var声明。

```

jQuery.extend = jQuery.fn.extend = function () {
    var options,
        name,
        src,
        copy,
        copyIsArray,
        clone, target = arguments[0] || {},
        i = 1,
        length = arguments.length,
        deep = false ;
}

```

- 缓存重复使用的全局变量。

- 全局变量要比局部变量需要搜索的作用域长
- 重复调用的方法也可以通过局部缓存来提速
- 该项优化在IE上体现比较明显

```

var docElem = window.document.documentElement,
    selector_hasDuplicate,
    matches = docElem.webkitMatchesSelector || docElem.mozMatchesSelector ||
    docElem.oMatchesSelector || docElem.msMatchesSelector,
    selector_sortOrder = function ( a, b ) {
        // Flag for duplicate removal
        if ( a === b ) {
            selector_hasDuplicate = true ;
            return 0;
        }
    }

```

- 善用回调。

- 除了使用闭包进行内部变量访问，我们还可以使用现在十分流行的回调函数来进行业务处理。

```

function getData(callback) {
    var data = 'some big data';

    callback(null, data);
}

getData(function(err, data) {
    console.log(data);
});

```

- 回调函数是一种后续传递风格(Continuation Passing Style , CPS)的技术，这种风格的程序编写将函数的业务重点从返回值转移到回调函数中去。而且其相

比闭包的好处也有很多。

- 如果传入的参数是基础类型（如字符串、数值），回调函数中传入的形参就会是复制值，业务代码使用完毕以后，更容易被回收。
- 通过回调，我们除了可以完成同步的请求外，还可以用在异步编程中，这也就是现在非常流行的一种编写风格。
- 回调函数自身通常也是临时的匿名函数，一旦请求函数执行完毕，回调函数自身的引用就会被解除，自身也得到回收。

同域跨域

- 避免跳转
 - 同域：注意避免反斜杠 “/” 的跳转；
 - 跨域：使用 Alias 或者 mod_rewrite 建立 CNAME （保存域名与域名之间关系的 DNS 记录）

字符串专题

- 对字符串进行循环操作。
 - 替换、查找等操作，使用正则表达式。
 - 因为 JAVASCRIPT 的循环速度较慢，而正则表达式的操作是用 c 写成的 API ，性能比较好。
- 字符串的拼接。
 - 字符串的拼接在我们开发中会经常遇到，所以我将其放在首位，我们往往习惯的直接用 += 的方式来拼接字符串，其实这种拼接的方式效率非常的低，我们可以用一种巧妙的方法来实现字符串的拼接，那就是利用数组的 join 方法，具体请看我整理的：[Web 前端开发规范文档](#) 中的 javascript 书写规范 倒数第三条目。
 - 不过也有另一种说法，通常认为需要用 Array.join 的方式，但是由于 SpiderMonkey 等引擎对字符串的“ + ”运算做了优化，结果使用 Array.join 的效率反而不如直接用“ + ”，但是如果考虑 IE6 ，则其他浏览器上的这种效率的差别根本不值一提。具体怎么取舍，诸君自定。

对象专题

- 减少不必要的对象创建：
 - 创建对象本身对性能影响并不大，但由于 JAVASCRIPT 的垃圾回收调度算法，导致随着对象个数的增加，性能会开始严重下降（复杂度 $O(n^2)$ ）。

- 如常见的字符串拼接问题，单纯的多次创建字符串对象其实根本不是降低性能的主要原因，而是是在对象创建期间的无谓的垃圾回收的开销。
而 `Array.join` 的方式，不会创建中间字符串对象，因此就减少了垃圾回收的开销。
- 复杂的 `JAVASCRIPT` 对象，其创建时时间和空间的开销都很大，应该尽量考虑采用缓存。
- 尽量作用 `JSON` 格式来创建对象，而不是 `var obj=new Object()` 方法。前者是直接复制，而后者需要调用构造器。
- 对象查找
 - 避免对象的嵌套查询，因为 `JAVASCRIPT` 的解释性，`a.b.c.d.e` 嵌套对象，需要进行 4 次查询，嵌套的对象成员会明显影响性能。
 - 如果出现嵌套对象，可以利用局部变量，把它放入一个临时的地方进行查询。
- 对象属性
 - 访问对象属性消耗性能过程（`JAVASCRIPT` 对象存储）。
 - 先从本地变量表找到 `对象`。
 - 然后遍历 `属性`。
 - 如果在 `当前对象` 的 `属性列表` 里没找到。
 - 继续从 `prototype` 向上查找。
 - 且不能直接索引，只能遍历。

```
function f(obj) {
    return obj.a + 1;
}
```

常规优化

>

- 传递方法取代方法字符串 > 一些方法例如 `setTimeout()`、`setInterval()`，接受 `字符串` 或者 `方法实例` 作为参数。直接传递方法对象作为参数来避免对字符串的二次解析。
- 传递方法 >

```
setTimeout(test, 1);
```

>

- 传递方法字符串 >

```
setTimeout('test()', 1);
```

■ 使用原始操作代替方法调用

方法调用一般封装了原始操作，在性能要求高的逻辑中，可以使用原始操作代替方法调用来提高性能。

◦ 原始操作 >

```
var min = a<b?a:b;
```

◦ 方法实例 >

```
var min = Math.min(a, b);
```

■ 定时器

如果针对的是不断运行的代码，不应该使用 `setTimeout`，而应该是用 `setInterval`。`setTimeout` 每次要重新设置一个定时器。

■ 避免双重解释

当 `JAVASCRIPT` 代码想解析 `JAVASCRIPT` 代码时就会存在双重解释惩罚，双重解释一般在使用 `eval` 函数、`new Function` 构造函数和 `setTimeout` 传一个字符串时等情况下会遇到，如。

```
eval("alert('hello world');");
var sayHi = new Function("alert('hello world');");
setTimeout("alert('hello world');", 100);
```

上述 `eval('hello world');` 语句包含在字符串中，即在JS代码运行的同时必须新启运一个解析器来解析新的代码，而实例化一个新的解析器有很大的性能损耗。我们看看下面的例子：

```
var sum, num1 = 1, num2 = 2; //效率低/ for(var i = 0; i < 10000; i++){
```

```
    var func = new Function("sum+=num1;num1+=num2;num2++;");
    func();
    //eval("sum+=num1;num1+=num2;num2++");
```

```
} //效率高/ for(var i = 0; i < 10000; i++){
```

```
    sum+=num1;
    num1+=num2;
    num2++;
```

```
}
```

第一种情况我们是使用了 `new Function` 来进行双重解释，而第二种是避免了双重解释。

■ 原生方法更快

- 只要有可能，使用原生方法而不是自己用JS重写。原生方法是用诸如C/C++之类的编译型语言写出来的，要比JS的快多了。

■ 最小化语句数

JS代码中的语句数量也会影响所执行的操作的速度，完成多个操作的单个语句要比完成单个操作的多个语句块快。故要找出可以组合在一起的语句，以减来整体的执行时间。这里列举几种模式

- 多个变量声明 >

```
/**不提倡*/
var i = 1;
var j = "hello";
var arr = [1,2,3];
var now = new Date();
/**提倡*/
var i = 1,
    j = "hello",
    arr = [1,2,3],
    now = new Date();
```

>

- o 插入迭代值 >

```
/**不提倡*/
var name = values[i];
i++;
/**提倡*/
var name = values[i++];
```

>

- o 使用数组和对象字面量，避免使用构造函数Array(),Object() >

```
/**不提倡*/
var a = new Array();
a[0] = 1;
a[1] = "hello";
a[2] = 45;
var o = new Object();
o.name = "bill";
o.age = 13;
/**提倡*/
var a = [1, "hello", 45];
var o = {
    name : "bill",
    age : 13
};
```

>

- 避免使用属性访问方法

- o JavaScript不需要属性访问方法，因为所有的属性都是外部可见的。
- o 添加属性访问方法只是增加了一层重定向，对于访问控制没有意义。> 使用属性访问方法示例 >

```

function Car() {
    this .m_tireSize = 17;
    this .m_maxSpeed = 250;
    this .GetTireSize = Car_get_tireSize;
    this .SetTireSize = Car_put_tireSize;
}

```

>

```

function Car_get_tireSize() {
    return this .m_tireSize;
}

```

>

```

function Car_put_tireSize(value) {
    this .m_tireSize = value;
}
var ooCar = new Car();
var iTireSize = ooCar.GetTireSize();
ooCar.SetTireSize(iTireSize + 1);

```

> 直接访问属性示例 >

```

function Car() {
    this .m_tireSize = 17;
    this .m_maxSpeed = 250;
}
var perfCar = new Car();
var iTireSize = perfCar.m_tireSize;
perfCar.m_tireSize = iTireSize + 1;

```

>

■ 减少使用元素位置操作

- 一般浏览器都会使用增量reflow的方式将需要reflow的操作积累到一定程度然后再一起触发，但是如果脚本中要获取以下属性，那么积累的reflow将会马上执行，已得到准确的位置信息。>

```

offsetLeft
offsetTop
offsetHeight
offsetWidth
scrollTop/Left/Width/Height
clientTop/Left/Width/Height
getComputedStyle()

```

循环专题

- 循环是一种常用的流程控制。
 - `JAVASCRIPT` 提供了三种循环。
 - `for(;;)` 。
 - 推荐使用`for`循环，如果循环变量递增或递减，不要单独对循环变量赋值，而应该使用嵌套的`++`或`--`运算符。
 - 代码的可读性对于`for`循环的优化。
 - 用`-=1`。
 - 从大到小的方式循环（这样缺点是降低代码的可读性）。

```
/**效率低*/
var divs = document.getElementsByTagName("div");
for(var i = 0; i < divs.length; i++){
    ...
}

/**效率高，适用于获取DOM集合，如果纯数组则两种情况区别不到*/
var divs = document.getElementsByTagName("div");
for(var i = 0, len = divs.length; i < len; i++){
    ...
}

/**在`IE6.0`下，`for(;;)`循环在执行中，第一种情况会每次都计算一下长度
，而第二种情况却是在开始的时候计算长度，并将其保存到一个变量中，所以其执行效率要高点，所以在我们使用`for(;;)`循环的时候，特别是需要计算长度的情况，我们应该开始将其保存到一个变量中。*/

```

- `while()` 。
 - `for(;;)`、`while()` 循环的性能基本持平。
- `for(in)` 。
 - 在这三种循环中 `for(in)` 内部实现是构造一个所有元素的列表，包括`array`继承的属性，然后再开始循环，并且需要查询`hasOwnProperty`。所以 `for(in)` 相对 `for(;;)` 循环性能要慢。
- 选择正确的方法
 - 避免不必要的属性查找。
 - 访问`变量`或`数组`是`O(1)`操作。
 - 访问`对象`上的`属性`是一个`O(n)`操作。

对象上的任何属性查找都要比访问变量或数组花费更长时间，因为必须在原型链中对拥有该名称的属性进行一次搜索，即属性查找越多，执行时间越长。所以针对需要多次用到对象属性，应将其存储在局部变量。

- 优化循环。
 - 减值迭代。
 - 大多数循环使用一个从0开始，增加到某个特定值的迭代器。在很多情况下，从最大值开始，在循环中不断减值的迭代器更加有效。
 - 简化终止条件。
 - 由于每次循环过程都会计算终止条件，故必须保证它尽可能快，即避免属性查找或其它O(n)的操作。
 - 简化循环体。
 - 循环体是执行最多的，故要确保其被最大限度地优化。确保没有某些可以很容易移出循环的密集计算。
 - 使用后测试循环。
 - 最常用的for和while循环都是前测试循环，而如do-while循环可以避免最初终止条件的计算，因些计算更快。

```
for(var i = 0; i < values.length; i++) {
    process(values[i]);
}
```

优化1：简化终止条件

```
for(var i = 0, len = values.length; i < len; i++) {

    process(values[i]);
}

}
```

优化2：使用后测试循环（注意：使用后测试循环需要确保要处理的值至少有一个）

```
var i = values.length - 1; if(i > -1) {

    do {
        process(values[i]);
    }while(--i >= 0);

}
```

- 展开循环。

- 当循环的次数确定时，消除循环并使用多次函数调用往往更快。
- 当循环的次数不确定时，可以使用Duff装置来优化。

- Duff装置的基本概念是通过计算迭代的次数是否为8的倍数将一个循环展开为一系列语句。

```
// Jeff Greenberg for JS implementation of Duff's Device // 假设：
values.length > 0 function process(v) { alert(v); } var values =
[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17]; var iterations =
Math.ceil(values.length / 8); var startAt = values.length % 8; var i = 0;
do { switch(startAt) {

    case 0 : process(values[i++]);
    case 7 : process(values[i++]);
    case 6 : process(values[i++]);
    case 5 : process(values[i++]);
    case 4 : process(values[i++]);
    case 3 : process(values[i++]);
    case 2 : process(values[i++]);
    case 1 : process(values[i++]);

} startAt = 0; }while(--iterations > 0);
```

如上展开循环可以提升大数据集的处理速度。接下来给出更快的Duff装置技术，将do-while循环分成2个单独的循环。（注：这种方法几乎比原始的Duff装置实现快上40%。）

```
// Speed Up Your Site(New Riders, 2003) function process(v) {

    alert(v);

}

var values = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17]; var iterations =
Math.floor(values.length / 8); var leftover = values.length % 8; var i = 0;
if(leftover > 0) {

    do {
        process(values[i++]);
    }while(--leftover > 0);

}

do {
```

```

process(values[i++]);
process(values[i++]);
process(values[i++]);
process(values[i++]);
process(values[i++]);
process(values[i++]);
process(values[i++]);
process(values[i++]);

```

```
}while(--iterations > 0);
```

针对大数据集使用展开循环可以节省很多时间，但对于小数据集，额外的开销则可能得不偿失。

- 避免在循环中使用 `try-catch`。

- `try-catch-finally` 语句在 `catch` 语句被执行的过程中会动态构造变量插入到当前域中，对性能有一定影响。
- 如果需要异常处理机制，可以将其放在循环外层使用。

- 循环中使用 `try-catch`

```

for ( var i = 0; i < 200; i++) {
    try {} catch (e) {}
}

```

- 循环外使用 `try-catch`

```

try {
    for ( var i = 0; i < 200; i++) {}
} catch (e) {}

```

- 避免遍历大量元素：

- 避免对全局 `DOM` 元素进行遍历，如果 `parent` 已知可以指定 `parent` 在特定范围查询。

```

var elements = document.getElementsByTagName( '*' );
for (i = 0; i < elements.length; i++) {
    if (elements[i].hasAttribute( 'selected' )) {}
}

```

如果已知元素存在于一个较小的范围内，

```

var elements = document.getElementById( 'canvas' ).getElementsByTagName
( '*' );
for ( i = 0; i < elements.length; i++ ) {
    if ( elements[i].hasAttribute( 'selected' ) ) {}
}

```

性能测试工具

- js性能优化和内存泄露问题及检测分析工具
 - 性能优化 ajax 工具 `diviefirebug`
 - web 性能分析工具 [YSlow](#)
 - `performance` 性能评估打分，右击箭头可看到改进建议。
 - `stats` 缓存状态分析，传输内容分析。
 - `components` 所有加载内容分析，可以查看传输速度，找出页面访问慢的瓶颈。
 - `tools` 可以查看js和css，并打印页面评估报告。
 - 内存泄露检测工具 [sIEve](#)
 - `sIEve` 是基于 IE 的内存泄露检测工具，需要下载运行，可以查看dom孤立节点和内存泄露及内存使用情况。
 1. 列出当前页面内所有dom节点的基本信息(html id style 等)
 2. 页面内所有dom节点的高级信息(内存占用,数量,节点的引用)
 3. 可以查找出页面中的孤立节点
 4. 可以查找出页面中的循环引用
 5. 可以查找出页面中产生内存泄露的节点
 - 内存泄露提示工具 `leak monitor`
 - `leak monitor` 在安装后，当离开一个页面时，比如关闭窗口，如果页面有内存泄露，会弹出一个文本框进行即时提示。
 - 代码压缩工具
 - YUI压缩工具
 - Dean Edwards Packer
 - JSMin
 - `Blink/Webkit` 浏览器
 - 在 `Blink/Webkit` 浏览器中 (`Chrome`, `Safari`, `Opera`)，我们可以借助其中的 `Developer Tools` 的 `Profiles` 工具来对我们的程序进行内存检查。
- Node.js 中的内存检查
 - `Developer Tools - Profiles`

- 在 `Node.js` 中，我们可以使用 `node-heapdump` 和 `node-memwatch` 模块进行内存检查。

```
var heapdump = require('heapdump');
var fs = require('fs');
var path = require('path');
fs.writeFileSync(path.join(__dirname, 'app.pid'), process.pid);
```

在业务代码中引入 `node-heapdump` 之后，我们需要在某个运行时期，向 `Node.js` 进程发送 `SIGUSR2` 信号，让 `node-heapdump` 抓拍一份堆内存的快照。

```
$ kill -USR2 (cat app.pid)
```

这样在文件目录下会有一个以 `heapdump-<sec>. <usec>.heapsnapshot` 格式命名的快照文件，我们可以使用浏览器的 `Developer Tools` 中的 `Profiles` 工具将其打开，并进行检查。

- 分析浏览器提供的 `Waterfall` 图片来思考优化入口。
- 新的测试手段（`Navigation`, `Resource`, 和 `User timing`）。
- ...

数组专题

- 当需要使用数组时，可使用 `JSON` 格式的语法
 - 即直接使用如下语法定义数组：`[param, param, param...]`，而不是采用 `new Array(param, param, param...)` 这种语法。使用 `JSON` 格式的语法是引擎直接解释。而后者则需要调用 `Array` 的构造器。
- 如果需要遍历数组，应该先缓存数组长度，将数组长度放入局部变量中，避免多次查询数组长度。
 - 根据字符串、数组的长度进行循环，而通常这个长度是不变的，比如每次查询 `a.length`，就要额外进行一个操作，而预先把 `var len=a.length`，则每次循环就少了一次查询。

服务端优化

- 避免404。
 - 更改404错误响应页面可以改进用户体验，但是同样也会浪费服务器资源。

- 指向外部 JAVASCRIPT 的链接出现问题并返回404代码。
 - 这种加载会破坏并行加载。
 - 其次浏览器会把试图在返回的404响应内容中找到可能有用的部分当作 JavaScript代码来执行。
- 删除重复的 JAVASCRIPT 和 CSS 。
 - 重复调用脚本缺点。
 - 增加额外的HTTP请求。
 - 多次运算也会浪费时间。在IE和Firefox中不管脚本是否可缓存，它们都存在重复运算 JAVASCRIPT 的问题。
- ETags 配置 Entity 标签。
 - ETags 用来判断浏览器缓存里的元素是否和原来服务器上的一致。
 - 与 last-modified date 相比更灵活。如某个文件在1秒内修改了10次，ETags 可以综合 Inode (文件的索引节点 inode 数), MTime (修改时间)和 Size 来精准的进行判断，避开 UNIX 记录 MTime 只能精确到秒的问题。服务器集群使用，可取后两个参数。使用 ETags 减少 Web 应用带宽和负载
- 权衡DNS查找次数
 - 减少主机名可以节省响应时间。但同时也会减少页面中并行下载的数量。
 - IE 浏览器在同一时刻只能从同一域名下载两个文件。当在一个页面显示多张图片时，IE 用户的图片下载速度就会受到影响。
- 通过Keep-alive机制减少TCP连接。
- 通过CDN减少延时。
- 平行处理请求（参考BigPipe）。
- 通过合并文件或者Image Sprites减少HTTP请求。
- 减少重定向（HTTP 301和40x/50x）。

类型转换专题

- 把数字转换成字符串。
 - 应用 ""+1 ，效率是最高的。
 - 性能上来说：""+字符串 > String() > .toString() > new String() 。
 - String() 属于内部函数，所以速度很快。
 - .toString() 要查询原型中的函数，所以速度略慢。
 - new String() 最慢。
- 浮点数转换成整型。
 - 错误使用使用 parseInt() 。
 - parseInt() 是用于将 字符串 转换成 数字 ，而不是 浮点数 和 整型 之间的转换。
 - 应该使用 Math.floor() 或者 Math.round() 。

- `Math` 是内部对象，所以 `Math.floor()` 其实并没有多少查询方法和调用的时间，速度是最快的。

运算符专题

- 使用运算符时，尽量使用 `+=`、`-=`、`*=`、`\=` 等运算符号，而不是直接进行赋值运算。
- 位运算。
 - 当进行数学运算时 位运算 较快， 位运算 操作要比任何 布尔运算 或 算数运算 快，如 取模， 逻辑与 和 逻辑或 也可以考虑用 位运算 来替换。

逻辑判断优化

- `switch` 语句。
 - 若有一系列复杂的 `if-else` 语句，可以转换成单个 `switch` 语句则可以得到更快的代码，还可以通过将 `case` 语句按照最可能的到最不可能的顺序进行组织，来进一步优化。

重绘专题

- 减少页面的 重绘。
 - 减少页面 重绘 虽然本质不是 JAVASCRIPT 优化，但 重绘 往往是由 JAVASCRIPT 引起的，而 重绘 的情况直接影响页面性能。

```

var str = "<div>这是一个测试字符串</div>";
/**效率低*/
var obj = document.getElementsByTagName("body");
for(var i = 0; i < 100; i++){
    obj.innerHTML += str + i;
}
/**效率高*/
var obj = document.getElementsByTagName("body");
var arr = [];
for(var i = 0; i < 100; i++){
    arr[i] = str + i;
}
obj.innerHTML = arr.join("");

```

一般影响页面重绘的不仅仅是innerHTML，如果改变元素的样式，位置等情况都会触发页面重绘，所以在平时一定要注意这点。

- 使用HTML5和CSS3的一些新特性。
- 避免在HTML里面缩放图片。
- 避免使用插件。
- 确保使用正确的字体大小。
- 决定当前页面是不是能被访问。

收集于:[kahn1990/web_performance_optimization](#)

Awesome WPO

A curated list of Web Performance Optimization. Everyone can contribute here!

Categories

:memo: [Articles](#)

:newspaper: [Blogs](#)

:books: [Books](#)

:book: [Docs](#)

:movie_camera: [Talks](#)

Tools

- [Analyzers](#)
- [Analyzers API](#)
- [Benchmark - CSS](#)
- [Benchmark - JS](#)
- [Benchmark - PHP](#)
- [Bookmarklets](#)
- [CDN](#)
- [CDN - Monitor](#)
- [CDN - Utilities](#)
- [Image Optimizers](#)
- [Lazyloaders](#)
- [Loaders](#)
- [Metrics Monitor](#)
- [Minifiers HTML](#)
- [Minifiers JS & CSS](#)
- [Miscellaneous](#)
- [Sprite Generators](#)
- [SVG](#)
- [Web Components](#)
- [Webserver Benchmarks](#)
- [Webserver Modules](#)

- Specs
- Stats
- Web Performance Meetup Groups
- Other Awesome Lists
- Contributing

Articles

| For a better categorization of the articles, they were categorized into [ARTICLES.md](#).

Blogs

| For a better categorization of the Blogs, they were categorized into [BLOGS](#).

Books

| Best books about WPO

- [Book of Speed](#) - Stoyan Stefanov
- [Even Faster Web Sites: Performance Best Practices for Web Developers](#) - Steve Souders
- [Getting Started with Web Performance](#) - Daniel Austin - early release
- [High Performance Browser Networking: What every web developer should know about networking and web performance](#) - Ilya Grigorik
- [High Performance JavaScript](#) - Nicholas C. Zakas
- [High Performance Web Sites: Essential Knowledge for Front-End Engineers](#) - Steve Souders
- [Using WebPagetest](#) - Rick Viscomi, Andy Davies, Marcel Duran - early release
- [Web Page Size, Speed, and Performance](#) - Terrence Dorsey
- [Web Performance Daybook Volume 2](#) - Stoyan Stefanov
- [Web Performance Tuning](#) - Patrick Killelea
- [You Don't Know JS: Async & Performance](#) - Kyle Simpson

Case studies

- [WPOStats](#) - Case studies and experiments demonstrating the impact of web performance optimization (WPO) on user experience and business metrics..

Documentation

- [Browser Diet](#) - A collaborative guide about front-end performance.
- [PageSpeed Insights Rules](#) - A guide created by PageSpeed Team.
- [Best Practices for Speeding Up Your Web Site](#) - The list includes 35 best practices divided into 7 categories, created by Yahoo! Exceptional Performance team.
- [Image Optimization Techniques](#) - A collection of tools and techniques to optimize your images and improve your page load times.

Talks

For a better categorization of the talks, they were categorized into [TALKS.md](#).

Tools

Analyzers

- [Confess](#) - Uses PhantomJS to headlessly analyze web pages and generate manifests.
- [Page Speed](#) - The PageSpeed family of tools is designed to help you optimize the performance of your website. PageSpeed Insights products will help you identify performance best practices that can be applied to your site, and PageSpeed optimization tools can help you automate the process.
- [YSlow](#) - YSlow analyzes web pages and suggests ways to improve their performance based on a set of rules for high performance web pages.
- [YSlow for PhantomJS](#) - YSlow for PhantomJS also introduces new output formats for automated test frameworks: TAP (Test Anything Protocol) and JUnit.
- [Grunt-WebPageTest](#) - Grunt plugin for continuously measurement of WebPageTest. ([Demo](#))
- [Grunt-yslow](#) - Grunt task for testing page performance using PhantomJS, a headless WebKit browser.
- [Grunt-perfbudget](#) - A Grunt.js task for enforcing a performance budget ([more on performance budgets](#)).
- [Web Tracing Framework](#) - Web Tracing Framework is a set of libraries, tools, and visualizers for the tracing and investigation of complex web applications
- [HP LoadRunner](#) - An automated performance and test automation product from Hewlett-Packard for application load testing: examining system behaviour and performance, while generating actual load.

- [Yandex.Tank](#) - An extendable open source load testing tool for advanced linux users which is especially good as a part of automated load testing suit.
- [Yellow Lab Tools](#) - Online quick and easy tool that audits front-end bad practices, reveals performance issues and profiles JavaScript.
- [PerfAudit](#) - Performance audit of web applications by professionals to help make web applications faster.
- [Pagelocity](#) - A web performance optimization and analysis tool. It covers resources and code analysis as well as content and social audit.

Analyzers - API

- [Node-yslowjs](#) - YSlow.js on Node.js is a simple Node.js wrapper for programmatically running phantomjs yslow.js.
- [PSI](#) - PageSpeed Insights for Node - with reporting.

Benchmark - CSS

- [CSS-perf](#) - Completely unscientific way of testing CSS performance. Most of these tests will revolve around methodologies and techniques for determining effective CSS architecture. Put another way, I want to know what works best given a particular comparison of CSS strategies.

Benchmark - Javascript

A set of tools for creating test cases and compare different implementations in JavaScript.

- [JSPerf](#) - jsPerf aims to provide an easy way to create and share test cases, comparing the performance of different JavaScript snippets by running benchmarks.
- [Benchmark.js](#) - A robust benchmarking library that works on nearly all JavaScript platforms, supports high-resolution timers, and returns statistically significant results.
- [JSLitmus](#) - JSLitmus is a lightweight tool for creating ad-hoc JavaScript benchmark tests.
- [Matcha](#) - Matcha allow you to design experiments that will measure the performance of your code. It is recommended that each bench focus on a specific point of impact in your application.
- [Timing.js](#) - Timing.js is a small set of helpers for working with the Navigation Timing API to identify where your application is spending its time. Useful as a standalone script, DevTools Snippet or bookmarklet.

- [Stats.js](#) - This class provides a simple info box that will help you monitor your code performance.
- [PerfTests](#) - Performance tests of JavaScript inheritance models.
- [Memory-stats.js](#) - minimal monitor for JS Heap Size via performance memory.

Benchmark - PHP

- [PHPench](#) - PHPench creates a graphical output for a PHP benchmark. Plot the runtime of any function in realtime with GnuPlot and create an image out of the result.
- [php-bench](#) - Benchmark and profile PHP code blocks whilst measuring the performance footprint.

Bookmarklets

- [Yahoo YSlow for Mobile/Bookmarklet](#) - YSlow analyzes web pages and suggests ways to improve their performance based on a set of rules for high performance web pages.
- [PageSpeed \[Deprecated\]](#) - PageSpeed Insights is available as an open-source browser extension for Google Chrome. Webmasters and web developers can use PageSpeed Insights to evaluate the performance of their web pages and to get suggestions on how to improve them - Workaround by [@Kiquenet](#) - <https://gist.github.com/davidsonellipe/f8e3723b4dc0aaf9f6d4>.
- [PerfMap](#) - A bookmarklet to create a front-end performance heatmap of resources loaded in the browser using the Resource Timing API.
- [DOM Monster](#) - A cross-platform, cross-browser bookmarklet that will analyze the DOM & other features of the page you're on, and give you its bill of health.
- [CSS Stress](#) - CSS Stress is a Testing and Performance Profiling.
- [Performance-Bookmarklet](#) - Analyze the current page through the Resource Timing API, Navigation Timing API and User-Timing - Sort of a light live WebPageTest. As [Chrome Extension](#) and [Firefox Add-on](#) under the name Performance-Analyser.

CDN

A content delivery network or content distribution network (CDN) is a large distributed system of servers deployed in multiple data centers across the Internet. The goal of a CDN is to serve content to end-users with high availability and high performance. See a large list of CDN in [Wikipedia](#).

- [jsDelivr](#) - Similar to Google Hosted Libraries, jsDelivr is an open source CDN that allows

developers to host their own projects and anyone to link to our hosted files in their websites.

- [Google Hosted Libraries](#) - Google Hosted Libraries is a content distribution network for the most popular, open-source JavaScript libraries.
- [CDNjs](#) - An open source CDN for Javascript and CSS sponsored by CloudFlare that hosts everything from jQuery and Modernizr to Bootstrap.
- [Microsoft Ajax Content Delivery Network](#) - Microsoft Ajax Content Delivery Network (CDN) hosts popular third party JavaScript libraries such as jQuery and enables you to easily add them to your Web applications.
- [jQuery](#) - jQuery CDN – Latest Stable Versions, powered by MaxCDN.
- [Bootstrap](#) - The recommended CDN for Bootstrap, Font Awesome, and Bootswatch.
- [Handpicked jQuery Plugins Repository CDN](#) - Content Delivery Network for popular jQuery plugins.
- [OSSCDN](#) - OSSCDN is a free CDN powered by MaxCDN that hosts all kinds of Open Source projects.
- [CDNify](#) - A Content Delivery Network for distributing all your static assets with ease around the globe.
- :cn: [Baidu public CDN](#) - Baidu public CDN serves stable, reliable, high-speed services, including all of the world's most popular open source JavaScript libraries.
- :cn: [UpYun CDN](#) - CDN provide by upyun.
- :cn: [Sina Public Resources](#) - CDN provide by sinaapp.com.
- :cn: [Bootstrap中文网开放CDN服务](#) - Bootstrap Chinese net open CDN service (only HTTP).
- :ru: [Yandex CDN](#) - Yandex Content Delivery Network hosts popular third party JavaScript and CSS libraries (best for use in Russia).

To find useful informations and powerful tools in an easy-to-use online environment for you make the right choice between paid CDNs, please visit [CDNPlanet](#).

CDN - Monitor

- [CDNperf](#) - finds you fast and reliable JavaScript CDNs that make your websites snappy and happy.

CDN - Utilities

- [Cdnex](#) - Prepends asset urls with custom CDN urls.
- [Gulp-google-cdn](#) - Replaces script references with Google CDN ones.

Image Optimizers

How to remove all this unnecessary data and give you a file without degrading quality.

- [Grunt-smushit](#) - Grunt plugin to remove unnecessary bytes of PNG and JPG using Yahoo Smushit.
- [Gulp-smushit](#) - Gulp plugin to optimize PNG and JPG using Yahoo Smushit. Made on top of smosh.
- [Smush.it](#) - Smush.it uses optimization techniques specific to image format to remove unnecessary bytes from image files. It is a "lossless" tool, which means it optimizes the images without changing their look or visual quality.
- [Imagemin](#) - Minify images seamlessly with Node.js.
- [Sharp](#) - The typical use case for this high speed Node.js module is to convert large images of many formats to smaller, web-friendly JPEG, PNG and WebP images of varying dimensions.
- [Gm](#) - GraphicsMagick and ImageMagick for node.
- [Exexif](#) - Pure elixir library to extract tiff and exif metadata from jpeg files.
- [OptiPNG](#) - OptiPNG is a PNG optimizer that recompresses image files to a smaller size, without losing any information.
- [Grunt-contrib-imagemin](#) - Minify PNG and JPEG images for Grunt.
- [Gulp-imagemin](#) - Minify PNG, JPEG, GIF and SVG images with imagemin for Gulp.
- [Grunt-WebP](#) - Convert your images to WebP format.
- [Gulp-WebP](#) - Convert images to WebP for Gulp.
- [Imageoptim](#) - Free app that makes images take up less disk space and load faster, without sacrificing quality. It optimizes compression parameters, removes junk metadata and unnecessary color profiles.
- [Grunt-imageoptim](#) - Make ImageOptim, ImageAlpha and JPEGmini part of your automated build process.
- [ImageOptim-CLI](#) - Automates ImageOptim, ImageAlpha, and JPEGmini for Mac to make batch optimisation of images part of your automated build process.
- [Tinypng](#) - Advanced lossy compression for PNG images that preserves full alpha transparency.
- [Kraken Web-interface](#) - Optimize your images and will be available for download for 12 hours.
- [Shrinkray](#) - One-click optimization for images in your Github repos
- [mozjpeg](#) - Improved JPEG encoder.
- [Jpegoptim](#) - Utility to optimize/compress JPEG files.
- [ZopfliPNG](#) - A command line program to optimize PNG images.
- [AdvPNG](#) - Recompress png files to get the smallest possible size.
- [Leanify](#) - Lightweight lossless file minifier/optimizer.

- [Trimage](#) - A cross-platform tool for losslessly optimizing PNG and JPG files.

Lazyloaders

- [lazyload](#) - Lazyload images, iframes, widgets with a standalone JavaScript lazyloader
~1kb

Loaders

- [HeadJS](#) - The only script in your HEAD. for Responsive Design, Feature Detections, and Resource Loading.
- [RequireJS](#) - RequireJS is a JavaScript file and module loader. It is optimized for in-browser use, but it can be used in other JavaScript environments, like Rhino and Node.
- [Labjs](#) - is an open-source (MIT license) project supported by Getify Solutions. The core purpose of LABjs is to be an all-purpose, on-demand JavaScript loader, capable of loading any JavaScript resource, from any location, into any page, at any time.
- [Defer.js](#) - Async Everything: Make the meat of your pages load faster with this JS morsel.
- [InstantClick](#) - InstantClick makes following links in your website instant.
- [JIT](#) - A JIT (Just In Time) plugin loader for Grunt. Load time of Grunt does not slow down even if there are many plugins.

Metrics Monitor

- [Phantomas](#) - PhantomJS-based web performance metrics collector and monitoring tool.
- [Showslow](#) - open source tool that helps monitor various website performance metrics over time. It captures the results of YSlow, Page Speed Insights, WebPageTest and dynaTrace AJAX Edition.
- [Bench](#) - Using Phantomas (a PhantomJS backed client performance metrics scrapper). Benchmark a page, store results in MongoDB and display result via the built in server.
- [Keepfast](#) - Tool to monitor indicators related to performance of a web page.
- [GTmetrix](#) - GTmetrix uses Google Page Speed and Yahoo! YSlow to grade your site's performance and provides actionable recommendations to fix these issues.
- [Pingdom Website Speed Test](#) - Test the load time of that page, analyze it and find bottlenecks.
- [Dotcom-tools](#) - analyze your website's speed in real browsers from 20 locations worldwide.
- [Libra](#) - Libra is a service to measure the weight along the time, written in Python.

- [WebPageTest](#) - Run a free website speed test from multiple locations around the globe using real browsers (IE and Chrome) and at real consumer connection speeds. You can run simple tests or perform advanced testing including multi-step transactions, video capture, content blocking and much more. Your results will provide rich diagnostic information including resource loading waterfall charts, Page Speed optimization checks and suggestions for improvements.
- [Sitespeed.io](#) - Sitespeed.io is an open source tool that will check your site against web performance best practice rules and use the Navigation Timing API to collect metrics. It will create XML & HTML output of the result.
- [Grunt-phantomas](#) - Grunt plugin wrapping phantomjs to measure frontend performance.
- [Perfjankie](#) - Runtime Browser Performance regression suite ([Demo](#)).
- [BrowserView Monitoring](#) - Continually checks web page load times in Internet Explorer, Chrome and Firefox from multiple points around the world.
- [PerfBar](#) - Simple way to collect and look at your website performance metrics quickly, that supports budgeting and adding custom metrics.
- [New Relic Browser Monitoring](#) - Monitor your browser-side applications in real time, with a focus on real end users' experiences.
- [DareBoost](#) - Real Browser Monitoring. Offers complete reports about web performance and quality using YSlow, Page Speed and numerous custom tips.

Metrics Monitor - API

- [WebPageTest API Wrapper for NodeJS](#) - WebPageTest API Wrapper is a NPM package that wraps WebPageTest API for NodeJS as a module and a command-line tool.

Minifiers - HTML

- [HTMLCompressor](#) - HtmlCompressor is a small, fast and very easy to use Java library that minifies given HTML or XML source by removing extra whitespaces, comments and other unneeded characters without breaking the content structure. As a result pages become smaller in size and load faster. A command-line version of the compressor is also available.
- [Django-htmlmin](#) - django-htmlmin is an HTML minifier for Python with full support for HTML. It supports Django, Flask and any other Python web framework. It also provides a command line tool that can be used for static websites or deployment scripts.
- [HTMLMinifier](#) - HTMLMinifier is a highly configurable, well-tested, Javascript-based HTML minifier, with lint-like capabilities.
- [Grunt-contrib-htmlmin](#) - A grunt plugin to minify HTML that uses HTMLMinifier.

- [Gulp-htmlmin](#) - A gulp plugin to minify HTML that uses HTMLMinifier.
- [Grunt-htmlcompressor](#) - Grunt plugin for html compression, using htmlcompressor.
- [HTML_minifier](#) - Ruby wrapper for kangax html-minifier.
- [HTML_press](#) - Ruby gem for compressing html, that removes all whitespace junk, and leaves only HTML.
- [Koa HTML Minifier](#) - Middleware that minifies your HTML responses using html-minifier. It uses html-minifier's default options which are all turned off by default, so you have to set the options otherwise it's not going to do anything.
- [HTML Minifier Online](#) - A HTML min tool by kangax (HTMLMinifier Creator).
- [Minimize](#) - Minimize is a HTML minifier based on the node-htmlparser, currently, HTML minifier is only usable server side. Client side minification will be added in a future release.
- [Html-minifier](#) - A simple Windows command line tool to minify your HTML, Razor views & Web Forms views

Minifiers - JS & CSS

- [YUI Compressor](#) - JavaScript compressor which, in addition to removing comments and white-spaces, obfuscates local variables using the smallest possible variable name. This obfuscation is safe, even when using constructs such as 'eval' or 'with' (although the compression is not optimal in those cases) Compared to jsmin, the average savings is around 20%.
- [UglifyJS2](#) - UglifyJS is a JavaScript parser, minifier, compressor or beautifier toolkit, written in JavaScript.
- [CSSO](#) - CSS minimizer unlike others. In addition to usual minification techniques it can perform structural optimization of CSS files, resulting in smaller file size compared to other minifiers.
- [CSSmin.js](#) - cssmin.js is a JavaScript port of YUICompressor's CSS minifier.
- [Grunt-contrib-concat](#) - A Grunt plugin to concatenate files.
- [Grunt-contrib-uglify](#) - A Grunt plugin to concatenate and minify Javascript files.
- [Clean-css](#) - A fast, efficient, and well tested CSS minifier for node.js.
- [Django-compressor](#) - Compresses linked and inline javascript or CSS into a single cached file.
- [Django-pipeline](#) - Pipeline is an asset packaging library for Django, providing both CSS and JavaScript concatenation and compression, built-in JavaScript template support, and optional data-URI image and font embedding.
- [JShrink](#) - JShrink is a PHP class that minifies javascript so that it can be delivered to the client quicker.
- [JSCompress](#) - The most minimalistic online JS Compress tool.

- [CSSshrink](#) - Because CSS is ospon the critical path to rendering pages. It must be small! Or else!
- [Grunt-cssshrink](#) - This is just a grunt wrapper for CSS Shrink.
- [Gulp-cssshrink](#) - Shrinks CSS files using cssshrink for Gulp.
- [Prettyugly](#) - Uglify (strip spaces) or prettify (add consistent spaces) CSS code.
- [Grunt-contrib-cssmin](#) - CSS Minifier for Grunt.
- [Gulp-cssmin](#) - CSS Minifier for Gulp.
- [Grunt-uncss](#) - A grunt task for removing unused CSS from your projects.
- [Gulp-uncss](#) - A gulp task for removing unused CSS from your projects.

Miscellaneous

- [Socialite.js](#) - Socialite provides a very easy way to implement and activate a plethora of social sharing buttons — any time you wish. On document load, on article hover, on any event.
- [uCSS](#) - uCSS is made for crawling (large) websites to find unused CSS selectors, but not remove unused CSS.
- [HTTPInvoke](#) - A no-dependencies HTTP client library for browsers and Node.js with a promise-based or Node.js-style callback-based API to progress events, text and binary file upload and download, partial response body, request and response headers, status code.
- [Critical](#) - Extract & Inline Critical-path CSS in HTML pages (alpha).
- [Csscolormin](#) - Utility that minifies CSS colors, example: min("white"); // minifies to "#fff".
- [StyleStats](#) - StyleStats is a Node.js library to collect CSS statistics.
- [Lazysizes](#) - High performance lazy loader for images (responsive and normal), iframes and scripts, that detects any visibility changes triggered through user interaction, CSS or JavaScript without configuration.
- [Perf-Tooling](#) - Perf Tooling is a shared resource to keep track of new and existent performance tools.
- [TMI](#) - TMI (Too Many Images) - discover your image weight on the web.

Sprite Generators

- [Glue](#) - Glue is a simple command line tool to generate sprites:
- [Pitomba-spriter](#) - Spriter is a simple and flexible dynamic sprite generator for CSS, using Python. It can process CSS both synchronous and asynchronous as it provides classes to be used in your python code and also a watcher that listens to your filesystem and changes CSS and sprite as soon as a static is changed.
- [Grunt.spritesmith](#) - Grunt task for converting a set of images into a spritesheet and

corresponding CSS variables.

- [Grunt-sprite-css-replace](#) - Grunt task that generates a sprite from images referenced in a stylesheet and then updates the references with the new sprite image and positions.
- [Grunt-svg-sprite](#) - SVG sprites & stacks galore — Grunt plugin wrapping around svg-sprite that reads in a bunch of SVG files, optimizes them and creates SVG sprites and CSS resources in various flavours.
- [Gulp-sprite](#) - gulp task for creating a image sprite and the corresponding stylesheets for Gulp.
- [Gulp-svg-sprites](#) - gulp task for creating svg sprites.
- [SvgToCSS](#) - Optimizes and renders SVG files in css / sass sprites.
- [Assetgraph-sprite](#) - Assetgraph transform for auto generating sprites based on the CSS dependency graph.
- [Sprite Cow](#) - Sprite Cow helps you get the background-position, width and height of sprites within a spritesheet as a nice bit of copyable css.
- [Spriteme](#) - Create, integrate, and maintain CSS sprites with ease.
- [ZeroSprites](#) - ZeroSprites is a CSS sprites generator aimed at area minimization using algorithms used in the field of VLSI floorplanning.
- [CSS Sprite Generator](#) - CSS sprites allow you to combine multiple images into a single file.
- [Sprity](#) - A modular image sprite generator with a lot of features: supports retina sprites, supports different output formats, generates sprites and proper style files out of a directory of images, etc...

SVG

- [SVGO](#) - SVGO is a Nodejs-based tool for optimizing SVG vector graphics files.
- [SVG OMG](#) - SVGOMG is SVGO's Missing GUI, aiming to expose the majority, if not all the configuration options of SVGO.
- [Grunt-svgmin](#) - Minify SVG using SVGO for Grunt.
- [Gulp-svgmin](#) - Minify SVG with SVGO for Gulp.
- [Scour](#) - Scour is an open-source Python script that aggressively cleans SVG files, removing a lot of 'cruft' that certain tools or authors embed into their documents.
- [SVG Cleaner](#) - SVG Cleaner could help you to clean up your SVG files from unnecessary data. It has a lot of options for cleanup and optimization, works in batch mode, provides threaded processing on the multicore processors.

Web Components

- [Vulcanize](#) - Concatenate a set of Web Components into one file, a Build tool for

HTMLImports and Web Components.

- [Grunt-vulcanize](#) - Grunt task for Polymer's Vulcanize.
- [Gulp-vulcanize](#) - Concatenate a set of Web Components into one file that use Vulcanize.

Webserver Benchmarks

- [HTTPPerf](#) - httpperf is a tool for measuring web server performance. It provides a flexible facility for generating various HTTP workloads and for measuring server performance.
- [Apache JMeter](#) - Open source load testing tool: It is a Java platform application.
- [Locust](#) - An open source load testing tool. Define user behaviour with Python code, and swarm your system with millions of simultaneous users.
- [Autoperf](#) - Autoperf is a ruby driver for httpperf, designed to help you automate load and performance testing of any web application - for a single end point, or through log replay.
- [HTTPPerf.rb](#) - Simple Ruby interface for httpperf, written in Ruby.
- [PHP-httpf](#) - PHP Port of HTTPPerf.rb.
- [HTTPPerf.js](#) - JS Port of HTTPPerf.rb.
- [HTTPPerf.py](#) - Python Port of HTTPPerf.rb.
- [Gohttpf](#) - Go Port of HTTPPerf.rb.
- [wrk](#) - A HTTP benchmarking tool (with optional Lua scripting for request generation, response processing, and custom reporting)
- [beeswithmachineguns](#) - A utility for arming (creating) many bees (micro EC2 instances) to attack (load test) targets (web applications).

Webserver Modules

- [PageSpeed Module](#) - PageSpeed speeds up your site and reduces page load time. This open-source webserver module automatically applies web performance best practices to pages and associated assets (CSS, JavaScript, images) without requiring that you modify your existing content or workflow. PageSpeed is available as a module for Apache 2.x and Nginx 1.x.
- [WebP-detect](#) - WebP with Accept negotiation.

Specs

- [Web Performance Working Group](#) - The mission of the Web Performance Working Group, part of the Rich Web Client Activity, is to provide methods to measure aspects of

application performance of user agent features and APIs.

- [Page Visibility](#) - This specification defines a means for site developers to programmatically determine the current visibility state of the page in order to develop power and CPU efficient web applications.
- [Navigation Timing](#) - This specification defines a unified interface to store and retrieve high resolution performance metric data related to the navigation of a document.
- [Resource Timing](#) - This specification defines an interface for web applications to access the complete timing information for resources in a document.
- [User Timing](#) - This specification defines an interface to help web developers measure the performance of their applications by giving them access to high precision timestamps.
- [Performance Timeline](#) - This specification defines an unified interface to store and retrieve performance metric data. This specification does not cover individual performance metric interfaces.
- [CSS will-change](#) - This specification defines the `will-change` CSS property which allows an author to declare ahead-of-time what properties are likely to change in the future, so the UA can set up the appropriate optimizations some time before they're needed. This way, when the actual change happens, the page updates in a snappy manner.
- [Resource Hints](#) - This specification defines the dns-prefetch, preconnect, prefetch, and prerender relationships of the HTML Link Element (`<link>`). These primitives enable the developer, and the server generating or delivering the resources, to assist the user agent in the decision process of which origins it should connect to, and which resources it should fetch and preprocess to improve page performance.

Stats

- [HTTP Archive](#) - It's a permanent repository of web performance information such as size of pages, failed requests, and technologies utilized. This performance information allows us to see trends in how the Web is built and provides a common data set from which to conduct web performance research.

Web Performance Meetup Groups

- [Web Performance Meetup Groups](#) - Full list on www.meetup.com.

Other Awesome Lists

- bayandin/awesome-awesomeness.
- sindresorhus/awesome.

Contributing

For contributing, [open an issue](#) and/or a [pull request](#).

HTTP API 设计指南

概述

该指南讲解了一系列 HTTP+JSON API 设计经验。这些经验最初来自 Heroku 平台 API 的实践。

该指南对此 API 进行了补充，并且对 Heroku 的新的内部 API 起到了指导作用。我们希望在 Heroku 之外的 API 设计者也会对此感兴趣。

本文的目标是在保持一致性，且关注业务逻辑的同时，避免设计歧义。我们一直在寻找一种良好的、一致的、文档化的方法来设计 API，但没必要是唯一的/理想化的方法。

本文假设读者已经对 HTTP+JSON API 的基本知识有所了解，因此不会在指南中涵盖所有的基础概念。

欢迎对该指南给与贡献。

目录

- [基础](#)
 - [必须使用 TLS](#)
 - [用 Accept 头指定版本](#)
 - [利用 Etag 支持缓存](#)
 - [通过 Request-Id 跟踪请求](#)
 - [使用 Content-Range 进行分页](#)
- [请求](#)
 - [返回适当的状态码](#)
 - [尽可能提供完整的资源](#)
 - [允许 JSON 编码的请求体](#)
 - [使用一致的路径格式](#)
 - [小写的路径和属性](#)
 - [为了方便支持非 id 的引用](#)
 - [最少的路径嵌套](#)
- [响应](#)
 - [为资源提供 \(UU\)ID](#)
 - [提供标准的时间戳](#)
 - [使用 ISO8601 格式化的 UTC 时间](#)
 - [嵌套的外键关系](#)

- 生成结构化的错误
 - 显示请求频度限制的状态
 - 在所有请求中都保持 JSON 简洁
- 辅助
 - 提供机器可识别的 JSON schema
 - 提供可读的文档
 - 提供可执行的例子
 - 对稳定性进行描述

基础

必须使用 TLS

必须使用 TLS 来访问 API，没有例外。任何试图阐明或解释什么时候用它合适，什么时候用它不合适都是徒劳。让任何请求都需要使用 TLS。

理想情况下，为了避免任何不安全的数据交换，对任何 HTTP 或端口 80 的非 TLS 的请求都应当不进行响应。实际环境中，这不太可能，所以需要响应 `403 Forbidden`。

由于马虎的/恶意的客户端行为无法提供任何明确的保障，所以不建议使用重定向。重定向的客户端使得服务器的流量成倍增长，并且会在第一次调用的时候让敏感的数据暴露出来，使得 TLS 不起作用。

用 `Accept` 头指定版本

从一开始就对 API 添加版本。使用 `Accept` 头和自定义的内容类型来指定版本，例如：

```
Accept: application/vnd.heroku+json; version=3
```

最好不要用默认的版本，让客户端明确指出它们需要使用的版本。

利用 `Etag` 支持缓存

在所有响应中包含 `ETag` 头，用以标识返回资源的特定版本。用户应当可以在随后的请求中，通过在 `If-None-Match` 头中指定该值来检查过期。

通过 `Request-Id` 跟踪请求

在每个 API 响应中包含 `Request-Id` 头，并附加一个 UUID 值。如果服务器和客户端都对该值进行了记录，那么在跟踪和调试请求的时候会非常有用。

使用 Content-Range 进行分页

对任何响应都进行分页，使得大量数据容易被处理。使用 `Content-Range` 头来传递分页请求。参阅 [Heroku Platform API on Ranges](#) 中的例子来了解请求和响应的头、状态码、上限、排序和跳转的细节。

请求

返回适当的状态码

对每一个请求都返回适当的 HTTP 状态码。根据本指南，成功的响应当使用以下代码：

- 200：对于 `GET` 以及完全同步的 `DELETE` 或 `PATCH` 的请求成功时
- 201：对于完全同步的 `POST` 请求成功时
- 202：对于异步的 `POST`、`DELETE` 或 `PATCH` 请求被接受
- 206：`GET` 请求成功，不过只有部分内容被返回：参阅[前面关于分页的内容](#)

在使用身份验证与身份验证错误码时务必当心：

- 401 Unauthorized：由于用户未进行身份验证，所以请求失败
- 403 Forbidden：由于用户无权对特定资源进行访问，所以请求失败

当遇到错误的时候，需要返回合适的代码里提供附加的信息：

- 422 Unprocessable Entity：请求可以被解析，但包含了错误的参数
- 429 Too Many Requests：请求达到频度限制，稍候再试
- 500 Internal Server Error：服务器发生了一些错误，检查状态站点或提交一个 issue

参阅 [HTTP response code spec](#) 了解用户错误与服务器错误的情况下状态码。

尽可能提供完整的资源

在可能的情况下，在响应中提供完整的资源（例如对象和其所有属性）。在 200 和 201 响应中提供完整的资源，包括 `PUT` / `PATCH` 和 `DELETE` 请求，例如：

```
$ curl -X DELETE \
https://service.com/apps/1f9b/domains/0fd4

HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
...
{
  "created_at": "2012-01-01T12:00:00Z",
  "hostname": "subdomain.example.com",
  "id": "01234567-89ab-cdef-0123-456789abcdef",
  "updated_at": "2012-01-01T12:00:00Z"
}
```

202 响应将不会包含完整的资源，例如：

```
$ curl -X DELETE \
https://service.com/apps/1f9b/dynos/05bd

HTTP/1.1 202 Accepted
Content-Type: application/json; charset=utf-8
...
{}
```

允许 JSON 编码的请求体

对于 `PUT / PATCH / POST` 允许使用 JSON 编码的请求体，可以看作是对表单数据的替换或补充。这与 JSON 编码的响应体对称，例如：

```
$ curl -X POST https://service.com/apps \
-H "Content-Type: application/json" \
-d '{"name": "demoapp"}'

{
  "id": "01234567-89ab-cdef-0123-456789abcdef",
  "name": "demoapp",
  "owner": {
    "email": "username@example.com",
    "id": "01234567-89ab-cdef-0123-456789abcdef"
  },
  ...
}
```

使用一致的路径格式

资源名

使用附带版本的资源名称，除非该资源在系统中仅有一个实例（例如，在大多数系统里，一个给定的用户只能有一个账户）。这与引用特定资源的方法一致。

操作

对于个别无须特定操作的资源，宁可使用直接的布局。而需要特定操作的情况下，将其放置在标准的 `actions` 前缀后，来描述它们：

```
/resources/:resource/actions/:action
```

例如：

```
/runs/{run_id}/actions/stop
```

小写的路径和属性

使用小写的、横线分隔的路径名称，与主机名一致，例如：

```
service-api.com/users  
service-api.com/app-setups
```

属性也小写，但是使用下划线分隔，这样属性名在 JavaScript 里无须转义，例如：

```
service_class: "first"
```

为了方便支持非 `id` 的引用

在某些情况下，让最终用户提供 ID 来标识一个资源可能不是那么方便。例如，用户可能想的是 HeroKu 的应用名称，但是那个应用可能是用 UUID 标识的。在这种情况下，可能需要同时接受 ID 和名称，例如：

```
$ curl https://service.com/apps/{app_id_or_name}  
$ curl https://service.com/apps/97addcf0-c182  
$ curl https://service.com/apps/www-prod
```

不要仅接受名字，而将 ID 排除在外。

最少的路径嵌套

在数据模型中有着父子嵌套关系的资源，路径可能会深层嵌套，例如：

```
/orgs/{org_id}/apps/{app_id}/dynos/{dyno_id}
```

限制嵌套的深度，让资源相对于根路径来定位。使用嵌套来表示域集合。例如，上面的例子中 `dyno` 属于一个 `app`，`app` 属于一个 `org`：

```
/orgs/{org_id}  
/orgs/{org_id}/apps  
/apps/{app_id}  
/apps/{app_id}/dynos  
/dynos/{dyno_id}
```

响应

为资源提供 (UU)ID

给每个资源一个默认的 `id` 属性。除非有一个好理由，否则还是使用 `UUID` 吧。不要使用那些在跨服务器实例或服务的其他资源中不是全局唯一的 ID，特别是不要使用自增 ID。

将 `UUID` 定义为小写的 `8-4-4-4-12` 格式，例如：

```
"id": "01234567-89ab-cdef-0123-456789abcdef"
```

提供标准的时间戳

为资源默认提供 `created_at` 和 `updated_at` 时间戳，例如：

```
{  
  ...  
  "created_at": "2012-01-01T12:00:00Z",  
  "updated_at": "2012-01-01T13:00:00Z",  
  ...  
}
```

这些时间戳对于某些资源来说可能没有实际意义，在这些情况下它们可以被省略。

使用 ISO8601 格式化的 UTC 时间

只使用 UTC 接收或返回时间。用 ISO8601 格式表达时间，例如：

```
"finished_at": "2012-01-01T12:00:00Z"
```

嵌套的外键关系

用嵌套的对象来表达外键关系，例如：

```
{  
  "name": "service-production",  
  "owner": {  
    "id": "5d8201b0..."  
  },  
  ...  
}
```

而不是：

```
{  
  "name": "service-production",  
  "owner_id": "5d8201b0...",  
  ...  
}
```

这一机制允许嵌入更多相关资源的信息，而无须修改响应的数据结构，或引入更多的顶级字段，例如：

```
{  
  "name": "service-production",  
  "owner": {  
    "id": "5d8201b0...",  
    "name": "Alice",  
    "email": "alice@heroku.com"  
  },  
  ...  
}
```

生成结构化的错误

生成一致的、结构化的错误响应。包括机器可识别的错误 `id`，人工可读的错误 `信息`，以及可选的 `url` 引导客户了解关于错误的更进一步的信息和解决方案，例如：

```
HTTP/1.1 429 Too Many Requests
```

```
{
  "id": "rate_limit",
  "message": "Account reached its API rate limit.",
  "url": "https://docs.service.com/rate-limits"
}
```

对错误格式和客户端可能遇到的错误 `id` 编写文档。

显示请求频度限制的状态

限制客户端的请求频度可以保护服务，并保持其他客户端较高的服务质量。可以使用 [token bucket algorithm](#) 来验证请求的频度。

在每个请求里都用 `RateLimit-Remaining` 响应头返回请求 token 的剩余请求数。

在所有请求中都保持 **JSON** 简洁

额外的空白字符会增加响应的大小，这是不必要的，而许多人工的客户端都会自动“美化” JSON 的输出。所以最好让 JSON 的响应保持最小，例如：

```
{"beta":false,"email":"alice@heroku.com","id":"01234567-89ab-cdef-0123-456789abcdef","last_login":"2012-01-01T12:00:00Z", "created_at":"2012-01-01T12:00:00Z", "updated_at":"2012-01-01T12:00:00Z"}
```

而不是：

```
{
  "beta": false,
  "email": "alice@heroku.com",
  "id": "01234567-89ab-cdef-0123-456789abcdef",
  "last_login": "2012-01-01T12:00:00Z",
  "created_at": "2012-01-01T12:00:00Z",
  "updated_at": "2012-01-01T12:00:00Z"
}
```

也可以考虑为客户端增加可选的方式来输出更详细的响应，不论是通过请求参数（例如 `?pretty=true`）或者通过 `Accept` 头参数（例如 `Accept: application/vnd.heroku+json; version=3; indent=4;`）。

辅助

提供机器可识别的 **JSON schema**

提供机器可识别的 schema 来明确你的 API。使用 `prmd` 来管理这些模式，并用 `prmd verify` 来验证。

提供可读的文档

提供可读的文档来让客户端开发者了解你的 API。

如果用上面提到的 `prmd` 创建了一个 schema，就可以很容易的通过 `prmd doc` 为所有接口创建 Markdown 文档。

作为接口的附加细节，为 API 提供以下信息的概述：

- 身份验证，包括获得和使用身份验证 token；
- API 的稳定程度与版本状况，包括如何选择目标版本的 API；
- 通用的请求和响应头；
- 错误的格式；
- 不同客户端语言的使用示例。

提供可执行的例子

提供用户可以直接在终端中输入来了解 API 调用情况的可执行的例子。为了最大程度的可扩展性，这些例子应当每行都可以使用，以降低用户尝试这些 API 的工作量，例如：

```
$ export TOKEN=... # acquire from dashboard  
$ curl -is https://$TOKEN@service.com/users
```

如果你使用 `prmd` 来生成 Markdown 文档，你可以很容易的获得每个接口的例子。

对稳定性进行描述

对你的 API 的稳定程度进行描述，包括不同接口的成熟度和稳定性。例如，使用 `prototype/development/production` 标识。

参阅 [Heroku API compatibility policy](#) 了解可能的稳定性管理和变更管理的方法。

一旦 API 被定义为生产环境适用且为稳定的，就不要对那个版本的 API 进行任何会破坏向后兼容性的改变。如果需要进行向后不兼容的修改，创建一个具有更高版本号的新 API。

collect from [github](#)

JavaScript判断IE各版本最完美解决方案

jQuery在1.9版本之前，提供了一个浏览器对象检测的属性`$.browser`，使用率极高。但是在1.9版本发布之后，大家钟爱的这个属性被jQuery无情的抛弃了。大家开始着手寻找`$.browser`的替代方案。于是各种利用IE bug的检测方法被搜了出来：

```
// shortest from a Russian
var ie = !-[1,]

// Option from Dean Edwards:
var ie = /*@cc_on!@*/false

// Use the commented line:
var ie//@cc_on=1

// Variation (shorter variable):
var ie = '\v'=='\v'

// Option to Gareth Hayes (former record-holder):
var ie = !+"\v1"
```

还有一些读取`navigator.userAgent`的方式，在苹果看来这些写法都不够友好，也不容易记忆，在搜索和挑选之后，终于找到了一个容易理解且友好方便的写法！

解决方案

IE知道自身毛病很多，于是提供的一套官方的HTML hack方式：

```
<!--[if IE]>
// 全部IE版本可见
<! [endif]-->
<!--[if IE 6]>
// IE6可见
<! [endif]-->
```

依次等等。这样的写法在其它浏览器里，完全就是一坨注释而直接遭到无视，但在IE里却不会。IE会分析里面的提到的版本号，并根据版本号确定要不要解析里面的DOM元素和文本内容。等一下！DOM元素？那岂不是可以使用js来获取里面的DOM元素？反正谁看到了，谁就是IE！于是，国外大神就有了下面的写法：

```

var isIE = function(){
    var b = document.createElement('b')
    b.innerHTML = '<!--[if IE]><i></i><![endif]-->'
    return b.getElementsByTagName('i').length === 1
}

```

这也太巧妙了！首先生成了一个b元素，设置它的innerHTML为一坨只有IE才认识的注释，注释里只有一个空的标签，然后读取里面的出现的元素i的个数是不是等于1，是不是等于1，是不是等于1。。。在大苹果看来，这样的写法比其它任何一种都要好。至于为什么生成一个b元素并且里面写一个i元素而不是div或者strong，更多是考虑到前者字节量更小。检测各个IE版本的方法也就顺理成章了：

```

var isIE6 = function(){
    var b = document.createElement('b')
    b.innerHTML = '<!--[if IE 6]><i></i><![endif]-->'
    return b.getElementsByTagName('i').length === 1
}
// var isIE7
// ...

```

更进一步

在苹果看来，还可以进一步将版本号提取成参数，就能生成一个通用的检测IE版本的函数了：

```

var isIE = function(ver){
    var b = document.createElement('b')
    b.innerHTML = '<!--[if IE ' + ver + ']><i></i><![endif]-->'
    return b.getElementsByTagName('i').length === 1
}
if(isIE(6)){
    // IE 6
}
// ...
if(isIE(9)){
    // IE 9
}

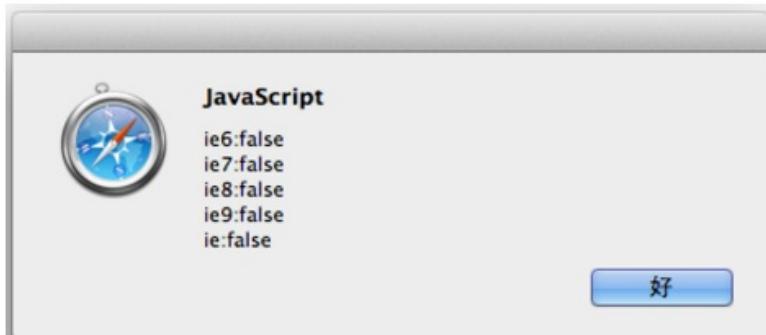
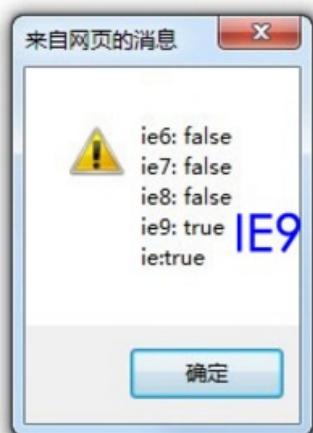
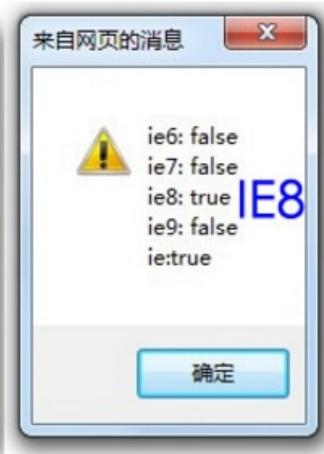
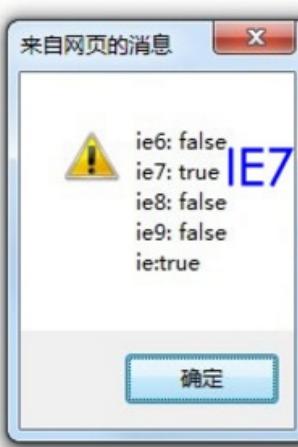
```

这样想检测哪个版本都毫无压力。但是，如果只想检测是不是IE，而不关心浏览器版本，那只需要在调用函数的时候，不传递参数即可。

```
var ie = isIE()
```

最后依次贴下在各大浏览器里测试代码的截图。

```
alert('ie6:' + isIE(6) + '\n' + 'ie7:' + isIE(7) + '\n' + 'ie8:' + isIE(8) + '\n' + 'i  
e9:' + isIE(9) + '\n' + 'ie:' + isIE())
```



PS：此方法仅适合IE9及以下版本浏览器，IE10+以后不支持[if IE]检查。

collect from [jquery-plugins](#)