**前端开发规范**

## 1.目录构建规范

#### *1.1命名原则*

#### *1.2根目录结构按职能划分*

#### *1.3根据业务逻辑进行文件夹划分*

## 2.HTML编写规范

#### *2.1 通用约定*

#### *2.2 HTML内容*

## 3.CSS编写规范

#### *3.1通用约定*

## 4.JS编写规范

#### *4.1通用约定*

#### *4.2性能优化*

## 代码规范校验工具——csslint，htmlhint，eslint (sublime和vscode均可安装)

## 1.目录构建规范

#### 1.1命名原则

- 简洁明了（如下：）

\* src 源代码

\* img 图片资源

\* js JavaScript脚本

\* dep 第三方依赖包

- 不使用复数（如下：）

\* 不使用 imgs docs

-html文件名应使用首字母小写驼峰命名，如userList.html。且该html对应的js和css的命名应保持一致。

-英文较长时，应使用简写：[paymentManage](http://www.baidu.com/link?url=pvLDSVDrVvL2pyUgFLP3UmdMYs0xeJ9Il5exHf1IC0MJYXDQS_QptoyuJVOKbE7cVRJQciPe5rdNccnpUySmRRzcJQ_81-W06ChIIGphTj2sAM6SJMGhlBYiw4Nuq-2W" \t "_blank)——>payManage，并在项目的doc文档中备注好该html对应的是哪个功能页面

#### 1.2根目录结构按职能划分（如下：）

- src 源代码（逻辑）

- doc 文档

- dep 第三方依赖包

- test 测试

#### 1.3根据业务逻辑进行文件夹划分（如下：）

├── src 源码目录

│ ├── main.js 入口js文件

│ ├── app.vue 根组件

│ ├── components 公共组件目录

│ │ └── title.vue

│ ├── assets 资源目录，这里的资源会被wabpack构建

│ │ └── images

│ │ └── logo.png

│ ├── routes 前端路由

│ │ └── index.js

│ ├── store 应用级数据（state）

│ │ └── index.js

│ └── views 页面目录

│ ├── hello.vue

│ └── notfound.vue

├── static 纯静态资源，不会被wabpack构建。

以上为vue 工程目录结构

├──js

│ ├──index.js

│ ├──home.js

├──css

│ ├──index.css

│ ├──home.css

├──img

│ ├──logo.png

├──html（除index.html外的其他html）

│ ├──home.html

├──index.html

以上为h5工程目录结构

## 2.HTML编写规范

#### 2.1 通用约定

###### 2.1.1标签

- 自闭合标签，无需闭合 ( 例如： img input br hr 等 )；

- 可选的闭合标签，需闭合 ( 例如：</li> 或 </body> )；

- 尽量减少标签数量；

###### 2.1.2 Class与ID命名规范——BEM规范

- BEM是定义了一种css class的命名规范，每个名称及其组成部分都是存在一定的含义。

- 由拉丁字母, 数字, -组成css的单个名称.

- Block    Element Modifier

- 独立且有意义的实体, e.g. header, container, menu, checkbox, etc.

- Block的一部分且没有独立的意义, e.g. header title, menu item, list item, etc.

- Blocks或Elements的一种标志，可以用它改变其表现形式、行为、状态. e.g. disabled, checked, fixed, etc.

- Naming

由拉丁字母, 数字, -组成css的单个名称.

- Block

使用简洁的前缀名字来命名一个独立且有意义的抽象块或组件。

```css

<!-- e.g. -->

.block

.header

.site-search

```

- Element

使用\_\_连接符来连接Block 和 Element。

```css

<!-- e.g. -->

.block\_\_element

.header\_\_title

.site-search\_\_field

```

- Modifier

使用--连接符来连接Block 或 Element 和 Modifier。

```css

<!-- e.g. -->

.block--modifier

.block\_\_element--modifier

.header--hide

.header\_\_title--color-red

.site-search\_\_field--disabled

```

- 实例

```html

<!-- HTML文件 -->

<form class="form form--theme-xmas form--simple">

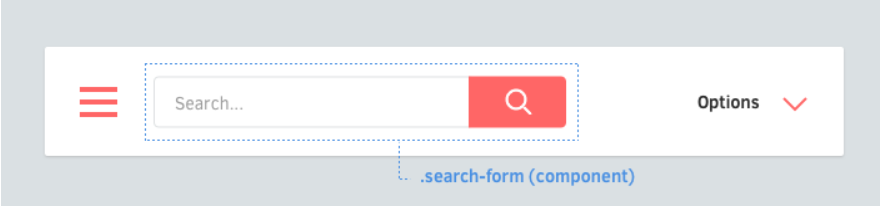
<input class="form\_\_input" type="text" />

<input class="form\_\_submit form\_\_submit--disabled" type="submit" />

</form>

```

Block的命名也可从组件的角度思考，将网站的模块都作为一个独立的Components



通过 - 分离，例如：

-点赞按钮 (.like-button)

-搜索框 (.search-form)

-文章卡片 (.article-card)

###### 2.1.3属性顺序

HTML 属性应该按照特定的顺序出现以保证易读性。

-class

-id

-name

-data-xxx

-src, for, type, href

-title, alt

-aria-xxx, role

如:

<a class="..." id="..." data-modal="toggle" href="###"></a>

###### 2.1.4引号

属性的定义，统一使用双引号。

如：

<span id="j-hook" class="text">Google</span>

###### 2.1.5布尔值属性

HTML5 规范中 disabled、checked、selected 等属性不用设置值。

如：

<input type="text" disabled>

<input type="checkbox" value="1" checked>

#### 2.2 HTML内容

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document</title>

<link rel="stylesheet" href="test.css">

</head>

<body>

<script type="text/javascript" src="test.js"></script>

</body>

</html>

-基本结构如上（css放head中，js放body中）

-公共方法抽离在公共js中，业务js中应只书写与业务相关代码，避免有过多的公用方法

## 3.CSS编写规范

#### 3.1 通用约定

###### 3.1.1代码组织

以组件为单位组织代码段，必须做好注释：/\* 对组件块的描述 \*/

如：

/\* 这是选择模块 \*/

.selector {

padding: 15px;

margin-bottom: 15px;

}

###### 3.1.2 Class和ID

-使用语义化、通用的命名方式；

-使用连字符 - 作为 ID、Class 名称界定符，不要驼峰命名法和下划线；

-避免选择器嵌套层级过多，尽量少于 3 级；

-避免选择器和 Class、ID 叠加使用；

如：

/\* 错误 \*/

.box\_green {}

.page .header .login #username input {}

ul#example {}

/\* 正确 \*/

.box-video {}

#username input {}

#example {}

###### 3.1.3 声明顺序

相关属性应为一组，推荐的样式编写顺序

1.Positioning

2.Box model

3.其它属性

由于定位（positioning）可以从正常的文档流中移除元素，并且还能覆盖盒模型（box model）相关的样式，因此排在首位。盒模型决定了组件的尺寸和位置，因此排在第二位。

其他属性只是影响组件的内部（inside）或者是不影响前两组属性，因此排在后面。

如：

.declaration-order {

/\* Positioning \*/

position: absolute;

top: 0;

right: 0;

bottom: 0;

left: 0;

z-index: 100;

/\* Box model \*/

display: block;

box-sizing: border-box;

width: 100px;

height: 100px;

padding: 10px;

border: 1px solid #e5e5e5;

border-radius: 3px;

margin: 10px;

float: right;

overflow: hidden;

/\* Other \*/

font: normal 13px "Helvetica Neue", sans-serif;

line-height: 1.5;

text-align: center;

cursor: pointer;

background-color: #f5f5f5;

color: #fff;

opacity: .8;

}

###### 3.1.4引号使用

url() 、属性选择符、属性值使用双引号。

如：

html {

font-family: "open sans", arial, sans-serif;

}

.selector[type="text"] {

}

###### 3.1.5媒体查询放置的位置

将媒体查询放在尽可能相关规则的附近。不要将他们打包放在一个单一样式文件中或者放在文档底部。易于维护

如：

.element { ... }

.element-avatar { ... }

.element-selected { ... }

@media (max-width: 768px) {

.element { ...}

.element-avatar { ... }

.element-selected { ... }

}

## 4.JS编写规范

#### 4.1 通用约定

###### 4.1.1函数/方法需添加注释说明

-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。；

-参数和返回值注释必须包含类型信息和说明；

/\*\*

\* 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

};

}

###### 4.1.2 命名

变量，函数，函数的参数都使用驼峰命名

**常量**, 使用全部字母大写，单词间下划线分隔的命名方式，如：

var HTML\_ENTITY = {};

**boolean**类型的变量使用 is 或 has 开头，如：

var isReady = false;

var hasMoreCommands = false;

#### 4.2 性能优化

4.2.1避免不必要的DOM操作

当一个元素出现多次时，将它保存在一个变量中，可避免多次查询 DOM 树

// 不推荐

for (var i = 0; i < 100; i++) {

document.getElementById("myList").innerHTML += "<span>" + i + "</span>";

}

// 推荐

var myList = "";

var myListHTML = document.getElementById("myList").innerHTML;

for (var i = 0; i < 100; i++) {

myList += "<span>" + i + "</span>";

}

myListHTML = myList;

4.2.2异步加载第三方内容

当无法保证嵌入第三方内容可以正常工作的时候，可考虑用异步加载这些代码，避免阻塞整个页面加载。

(function() {

var script,

scripts = document.getElementsByTagName('script')[0];

function load(url) {

script = document.createElement('script');

script.async = true;

script.src = url;

scripts.parentNode.insertBefore(script, scripts);

}

load('//apis.google.com/js/plusone.js');

load('//platform.twitter.com/widgets.js');

load('//s.widgetsite.com/widget.js');

}());