HTML/CSS开发规范

目录

通用约定

1.项目目录结构

```
|-- 项目名
    |-- src 开发环境
        |-- html    静态页面模板目录
        |-- bgimg   背景图目录(假设有的话)
        |-- image   前景图目录(假设有的话)
        |-- font    字体目录(假设有的话)
        |-- scripts      脚本目录
        |-- styles(Yo)  样式目录
            |-- lib   基础库
                |-- core      核心代码:reset
                |-- element   元素
                |-- fragment  公用碎片
                |-- layout    布局
                |-- widget    组件
            |-- usage  项目具体实现
                |-- project 某个子项目
                    |-- core      核心代码:桥接lib中的core,可以进行项目级扩展
                    |-- fragment  项目公用碎片
                    |-- module    模块
                    |-- page      page桥接文件目录:src-list
                    |-- export    page pack之后的文件目录
    |-- prd 生产环境
        |-- bgimg   背景图目录(假设有的话)
        |-- image   前景图目录(假设有的话)
        |-- font    字体目录(假设有的话)
        |-- scripts      脚本目录
        |-- styles(Yo)  样式目录
            |-- project1 子项目
                |-- index.css
                |-- login.css
                |-- and etc...
            |-- project2 子项目
                |-- index.css
                |-- login.css
                |-- and etc...
            |-- and etc...
```

> `src`，`scripts`，`styles` 三个目录是为了和现有项目保持一致，避免修改过大，所以保持不变。

> `html` 目录，用于存放前端开发做的静态页面，以备查阅、备份、review 或给后端套页面。

> `bgimg`，`image`，`font` 三个目录在Qunar一般不会直接存在，因为我们有source服务器，这些资源都会在那上面管理；不过特殊情况也会有，比如一些独立的项目，没有使用source的，那么就需要遵循这样的目录划分。

> 至于 `html`，`bgimg`，`image`，`font` 这几个目录为什么没有加 `s`，主要是因为不希望大家去想某个目录是否为复数，简单点就好。

> `prd` 为生产环境目录，以 `xxx` 项目中的一个子项目 `mobile` 为例，其生产环境中的某个CSS外链大致如下：`//sitename.com/prd/styles/mobile/index.css`

2.分离

结构（HTML）、表现（CSS）、行为分离（JavaScript）

> 将结构与表现、行为分离，保证它们之间的最小耦合，这对前期开发和后期维护都至关重要。

3.文件命名

- CSS模块文件，其文件名必须与模块名一致；

假定有这样一个模块：

```
.m-detail { sRules; }
.m-detail-hd { sRules; }
.m-detail-bd { sRules; }
.m-detail-ft { sRules; }
```

> 那么该模块的文件名应该为：`m-detail.css`

- CSS页面文件，其文件名必须与HTML文件名一致；

> 假定有一个HTML页面叫 `product.html`，那么其相对应的CSS页面文件名应该为：`product.css`

> 假定现在有一个 `product.html`，里面有2个模块：

```
+<section class="m-list">
+<section class="m-info">
```

那么开发人员能快速找到与该页面相关的3个直接CSS文件，包括：`product.css`，`m-list.css`，`m-info.css`

4.缩进

使用tab（4个空格宽度）来进行缩进，可以在IDE里进行设置

5.编码

- 以 UTF-8 无 BOM 编码作为文件格式；
- 在HTML中文档中用 `<meta charset="utf-8" />` 来指定编码；
- 为每个CSS文档显示的定义编码，在文档首行定义 `@charset "utf-8";`

> 在 Sass 中，如果文档中出现中文，却未显示定义编码，将会编译出错，为了统一各种写法，且提前规避错误几率，统一要求每个CSS文档都需要定义编码

6.小写

- 所有的HTML标签必须小写；
- 所有的HTML属性必须小写；
- 所有的样式名及规则必须小写。

7.注释

尽可能的为你的代码写上注释。解释为什么要这样写，它是新鲜的方案还是解决了什么问题。

8.待办事项

用 TODO 标示待办事项和正在开发的条目

```
<!-- TODO: 图文混排 -->
```

```
<div class="g-imgtext">
<img src="1.png" alt="" />
...

/* TODO: 图文混排 comm: g-imgtext */
.g-imgtext { sRules; }
```

9.行尾空格

删除行尾空格，这些空格没有必要存在

10.省略嵌入式资源协议头

省略图像、媒体文件、样式表和脚本等URL协议头部声明（http:，https:）。如果不是这两个声明的URL则不省略。

省略协议声明，使URL成相对地址，防止内容混淆问题和导致小文件重复下载（这个主要是指http和https交杂的场景中）。

不推荐：

```
<script src="http://www.google.com/js/gweb/analytics/autotrack.js"></script>
```

推荐：

```
<script src="//www.google.com/js/gweb/analytics/autotrack.js"></script>
```

不推荐：

```
.example {
  background: url(http://www.google.com/images/example);
}
```

推荐：

```
.example {
  background: url(//www.google.com/images/example);
}
```

> 注：省略协议头在IE7-8下会有一点小问题，外部CSS文件（link和@import）会被下载两遍，所以该条目的约定看具体项目。

11.代码有效性

- 使用 W3C HTML Validator 来验证你的HTML代码有效性；
- 使用 W3C CSS Validator 来验证你的CSS代码有效性。

> 代码验证不是最终目的，真的目的在于让开发者在经过多次的这种验证过程后，能够深刻理解到怎样的语法或写法是非标准和不推荐的，即使在某些场景下被迫要使用非标准写法，也可以做到心中有数。

HTML约定

1.文档类型

- 统一使用HTML5的标准文档类型：`<!DOCTYPE html>`；

> HTML5文档类型具备前后兼容的特质，并且易记易书写

- 在文档doctype申明之前，不允许加上任何非空字符；

> 任何出现在doctype申明之前的字符都将使得你的HTML文档进入非标准模式

- 不允许添加 `<meta>` 标签强制改变文档模式。

> 避免出现不可控的问题

2.省略type属性

在调用CSS和JavaScript时，可以将type属性省略不写

不允许：

```
<link type="text/css" rel="stylesheet" href="base.css" />
<script type="text/javascript" src="base.js"></script>
```

应该：

```
<link rel="stylesheet" href="base.css" />
<script src="base.js"></script>
```

> 因为HTML5在引入CSS时，默认type值为text/css；在引入JavaScript时，默认type值为text/javascript

3.省略属性值

非必须属性值可以省略

不允许：

```
<input type="text" readonly="readonly" />
<input type="text" disabled="disabled" />
```

应该：

```
<input type="text" readonly />
<input type="text" disabled />
```

> 这里的 readonly 和 disabled 属性的值是非必须的，可以省略不写，我们知道HTML5表单元素新增了很多类似的属性，如：required

4.用双引号包裹属性值

所有的标签属性值必须要用双引号包裹，同时也不允许有的用双引号，有的用单引号的情况

不允许：

```
<a href=http://www.qunar.com class=home>去哪儿网</a>
```

应该：

```
<a href="http://www.qunar.com" class="home">去哪儿网</a>
```

5.嵌套

所有元素必须正确嵌套

- **不允许交叉**；

不允许：

```
<span><dfn>交叉嵌套</span></dfn>
```

应该：

```
<span><dfn>交叉嵌套</dfn></span>
```

- **不允许非法的子元素嵌套。**

不允许：

```
<ul>
    <h3>xx列表</h3>
    <li>asdasdsdasd</li>
    <li>asdasdsdasd</li>
</ul>
```

应该：

```
<div>
    <h3>xx列表</h3>
    <ul>
        <li>asdasdsdasd</li>
        <li>asdasdsdasd</li>
    </ul>
</div>
```

- **不推荐inline元素包含block元素**；

不推荐：

```
<span>
    <h1>这是一个块级h1元素</h1>
    <p>这是一个块级p元素</p>
</span>
```

推荐：

```
<div>
    <h1>这是一个块级h1元素</h1>
    <p>这是一个块级p元素</p>
</div>
```

> 规则可参考：
>
> HTML4/XHTML1.0 Strict: 嵌套规则。
>
> HTML5: 嵌套规则
>
> 举个例子，在HTML5中，a元素同时属于 Flow content, Phrasing content, Interactive content, Palpable content 4个分类，那些子元素是 phrasing 元素的元素可以是 a 的父元素，a 允许的子元素是以它的父元素允许的子元素为准，但不能包含 interactive 元素。

6.标签闭合

所有标签**必**须闭**合**

不允许：

```
<div>balabala...
<link rel="stylesheet" href="*.css">
```

应该：

```
<div>balabala...</div>
<link rel="stylesheet" href="*.css" />
```

> 虽**然有些**标记**没有要求必**须**关**闭，**但是**为**了避免出**错**的几率**，**要求必**须**全部关**闭，**省去判断某**标记**是否需要关**闭**的**时间

7.多媒体替代方案

- 为img**元素加上**alt**属性**；
- 为视频**内容提供音**轨**替代**；
- 为**音**频**内容提供字母替代等等。**

不推荐：

```
<img src="banner.jpg" />
```

推荐：

```
<img src="banner.jpg" alt="520即将到来，爱就大声说出来" />
```

> alt**属性的内容**为对该图**片的**简**要描述**，这对**于盲人用**户**和**图像**损毁**都非常有意**义，**即无障碍。**对**于**纯**粹的装**饰**性**图**片**，alt**属性**值**可以留空**，**如** alt=""

8.有效操作

为表单元素label加上for属性

不允许：

```
<input type="radio" name="color" value="0" /><label>蓝色</label>
<input type="radio" name="color" value="1" /><label>粉色</label>
```

应该：

```
<input type="radio" id="blue" name="color" value="0" /><label for="blue">蓝色</label>
<input type="radio" id="pink" name="color" value="1" /><label for="pink">粉色</label>
```

> for属性能让点击label标签的时候，同时focus到对应的 input 和 textarea上，增加响应区域

9.按模块添加注释

在每个模块开始和结束的地方添加注释

```
<!-- 新闻列表模块 -->
<div class="m-news g-mod"
...
<!-- /新闻列表模块 -->

<!-- 排行榜模块 -->
<div class="m-topic g-mod"
...
<!-- /排行榜模块 -->
```

> 注释内容左右两边保留和注释符号有1个空格位，在注释内容内不允许再出现中划线“-”，某些浏览器会报错。
>
> 注释风格保持与原生HTML的语法相似：成对出现 `<!-- comment --><!-- /comment -->`

10.格式

- 将每个块元素、列表元素或表格元素都放在新行；
- inline元素视情况换行，以长度不超过编辑器一屏为宜；
- 每个子元素都需要相对其父级缩进（参见缩进约定）。

不推荐：

```
<div><h1>asdas</h1><p>dff<em>asd</em>asda<span>sds</span>dasdasd</p></div>
```

推荐：

```
<div>
    <h1>asdas</h1>
    <p>dff<em>asd</em>asda<span>sds</span>dasdasd</p>
</div>
```

11.语义化标签

- 根据HTML元素的本身用途去使用它们；
- 禁止使用被废弃的用于表现的标签，如 center, font 等；
- 部分在XHTML 1中被废弃的标签，在HTML5中被重新赋予了新的语义，在选用时请先弄清其语义，如:b, i, u等。

不允许：

```
<p>标题</p>
```

应该：

```
<h1>标题</h1>
```

> 虽然使用p标签，也可以通过CSS去定义它的外观和标题相同，但p标签本身的并不是表示标题，而是表示文本段落

> 参阅：HTML5 Elements

12.模块化

- 每个模块必须有一个模块名；
- 每个模块的基本组成部分应该一致；
- 模块的子节点类名需带上模块名（防止模块间嵌套时产生不必要的覆盖）；
- 孙辈节点无需再带模块名。

代码如：

```
<section class="m-detail">
    <header class="m-detail-hd">
        <h1 class="title">模块标题</h1>
    </header>
    <div class="m-detail-bd">
        <p class="info">一些实际内容</p>
    </div>
    <footer class="m-detail-ft">
        <a href="#" class="more">更多</a>
    </footer>
</section>
```

> 其中 `.m-detail-hd`，`.m-detail-bd`，`.m-detail-ft` 为可选，视具体模块情况决定是否需要抽象为这种 头，中，尾 的结构

CSS约定

1.文件引用

- 一律使用link的方式调用外部样式
- 不允许在页面中使用 `<style>` 块；
- 不允许在 `<style>` 块中使用 `@import`；
- 不允许使用 `style` 属性写行内样式。

> 一般情况下，在页面中只允许使用 `<link />` 标签来引用CSS文件，

2.命名-组成元素

- 命名必须由单词、中划线①或数字组成；
- 不允许使用拼音（约定俗成的除外，如：youku, baidu），尤其是缩写的拼音、拼音与英文的混合。

不推荐：

```
.xiangqing { sRules; }
.news_list { sRules; }
.zhuti { sRules; }
```

推荐：

```
.detail { sRules; }
.news-list { sRules; }
.topic { sRules; }
```

> ①我们使用中划线 “-” 作为连接字符，而不是下划线 "_"。
>
> 我们知道2种方式都有不少支持者，但 "-" 能让你少按一次shift键，并且更符合CSS原生语法，所以我们只选一种目前业内普遍使用的方式

3.命名-词汇规范

- 不依据表现形式来命名；
- 可根据内容来命名；
- 可根据功能来命名。

不推荐：

```
left, right, center, red, black
```

推荐：

```
nav, aside, news, type, search
```

4.命名-缩写规范

- 保证缩写后还能较为清晰保持原单词所能表述的意思；
- 使用业界熟知的或者约定俗成的。

不推荐：

```
navigation&amp;nbsp;&amp;nbsp;&amp;nbsp;=>  navi
header&amp;nbsp;&amp;nbsp;     =>  head
description&amp;nbsp;&amp;nbsp;=>  des
```

推荐：

```
navigation   =>  nav
header       =>  hd
description  =>  desc
```

5.命名-前缀规范

前缀	说明	示例
g-	全局通用样式命名，前缀g全称为global，一旦修改将影响全站样式	g-mod
m-	模块命名方式	m-detail
ui-	组件命名方式	ui-selector
js-	所有用于纯交互的命名，不涉及任何样式规则。JSer拥有全部定义权限	js-switch

- 选择器必须是以某个前缀开头

不推荐：

```
.info { sRules; }
.current { sRules; }
.news { sRules; }
```

> 因为这样将给我们带来不可预知的管理麻烦以及沉重的历史包袱。你永远也不会知道哪些样式名已经被用掉了，如果你是一个新人，你可能会遭遇，你每定义个样式名，都有同名的样式已存在，然后你只能是换样式名或者覆盖规则。

推荐：

```
.m-detail .info { sRules; }
.m-detail .current { sRules; }
.m-detail .news { sRules; }
```

> 所有的选择器必须是以 g-, m-, ui- 等有前缀的选择符开头的，意思就是说所有的规则都必须在某个相对的作用域下才生效，尽可能减少全局污染。

js- 这种级别的className完全交由JSer自定义，但是命名的规则也可以保持跟重构一致，比如说不能使用拼音之类的

6.id与class

重构工程师只允许使用class（因历史原因及大家的习惯做出妥协）。

7.书写格式

- 选择器与大括号之间保留一个空格；
- 分号之后保留一个空格；
- 逗号之后保留一个空格；
- 所有规则需换行；
- 多组选择器之间需换行。

不推荐：

```
main{
    display:inline-block;
}
h1,h2,h3{
    margin:0;
    background-color:rgba(0,0,0,.5);
}
```

推荐:

```
main {
    display: inline-block;
}
h1,
h2,
h3 {
    margin: 0;
    background-color: rgba(0, 0, 0, .5);
}
```

8.规则与分号

每条规则结**束后都必**须**加上分号**

不推荐:

```
body {
    margin: 0;
    padding: 0;
    font-size: 14px
}
```

推荐:

```
body {
    margin: 0;
    padding: 0;
    font-size: 14px;
}
```

9.0与单位

如果属性值为0, 则**不需要**为0**加**单**位**

不推荐:

```
body {
    margin: 0px;
    padding: 0px;
}
```

推荐:

```
body {
    margin: 0;
    padding: 0;
}
```

10.0与小数

如果是0开始的小数，前面的0可以省略不写

不推荐：

```
body {
    opacity: 0.6;
    text-shadow: 1px 1px 5px rgba(0, 0, 0, 0.5);
}
```

推荐：

```
body {
    opacity: .6;
    text-shadow: 1px 1px 5px rgba(0, 0, 0, .5);
}
```

11.去掉uri中引用资源的引号

不要在url()里对引用资源加引号

不推荐：

```
body {
    background-image: url("sprites.png");
}
@import url("global.css");
```

推荐：

```
body {
    background-image: url(sprites.png);
}
@import url(global.css);
```

12.HEX颜色值写法

- **将所有的颜色值小写**；
- **可以缩写的缩写至3位。**

不推荐：

```
body {
    background-color: #FF0000;
}
```

推荐:

```
body {
    background-color: #f00;
}
```

13.属性书写顺序

- **遵循先布局后内容的顺序。**

```
.g-box {
   display: block;
   float: left;
   width: 500px;
   height: 200px;
   margin: 10px;
   padding: 10px;
   border: 10px solid;
   background: #aaa;
   color: #000;
   font: 14px/1.5 sans-serif;
}
```

> 这个应该好理解，比如优先布局，我们知道布局属性有 display, float, overflow 等等；内容次之，比如 color, font, text-align 之类。

- 组概念。

拿上例的代码来说，如果我们还需要进行定位及堆叠，规则我们可以改成如下：

```
.g-box {
   display: block;
   position: relative;
   z-index: 2;
   top: 10px;
   left: 100px;
   float: left;
   width: 500px;
   height: 200px;
   margin: 10px;
   padding: 10px;
   border: 10px solid;
   background: #aaa;
   color: #000;
   font: 14px/1.5 sans-serif;
}
```

> 从代码中可以看到，我们直接将z-index, top, left 紧跟在 position 之后，因为这几个属性其实是一组的，如果去掉position，则后3条属性规则都将失效。

- 私有属性在前标准属性在后

```
.g-box {
   -webkit-box-shadow: 1px 1px 5px rgba(0, 0, 0, .5);
   -moz-box-shadow: 1px 1px 5px rgba(0, 0, 0, .5);
   -o-box-shadow: 1px 1px 5px rgba(0, 0, 0, .5);
   box-shadow: 1px 1px 5px rgba(0, 0, 0, .5);
}
```

> 当有一天你的浏览器升级后，可能不再支持私有写法，那么这时写在后面的标准写法将生效，避免无法向后兼容的情况发生。

14.注释规范

保持注释内容与星号之间有一个空格的距离

普通注释（单行）

```
/* 普通注释 */
```

区块注释

```
/**
 * 模块: m-detail
 * 描述:酒店详情模块
 * 应用:page detail, info and etc...etc
 */
```

> 有特殊作用的规则一定要有注释说明
> 应用了高级技巧的地方一定要注释说明

15.hack规范

- 尽可能的减少对Hack的使用和依赖，如果在项目中对Hack的使用太多太复杂，对项目的维护将是一个巨大的挑战；
- 使用其它的解决方案代替Hack思路；
- 如果非Hack不可，选择稳定且常用并易于理解的。

```
.test {
   color: #000;       /* For all */
   color: #111\9;     /* For all IE */
   color: #222\0;     /* For IE8 and later, Opera without Webkit */
   color: #333\9\0;   /* For IE8 and later */
   color: #444\0/;    /* For IE8 and later */
   *color: #666;      /* For IE7 and earlier */
   _color: #777;      /* For IE6 and earlier */
}
```

- 严谨且长期的项目，针对IE可以使用条件注释作为预留Hack或者在当前使用

IE条件注释语法：

```
<!--[if <keywords>? IE <version>?]>
<link rel="stylesheet" href="*.css" />
<![endif]-->
```

语法说明：

```
<keywords>
if条件共包含6种选择方式：是否、大于、大于或等于、小于、小于或等于、非指定版本
是否：指定是否IE或IE某个版本。关键字：空
大于：选择大于指定版本的IE版本。关键字：gt(greater than)
大于或等于：选择大于或等于指定版本的IE版本。关键字：gte(greater than or equal)
```

```
小于:选择小于指定版本的IE版本。关键字:lt(less than)
小于或等于:选择小于或等于指定版本的IE版本。关键字:lte(less than or equal)
非指定版本:选择除指定版本外的所有IE版本。关键字:!
```

```
<version>
目前的常用IE版本为6.0及以上,推荐酌情忽略低版本,把精力花在为使用高级浏览器的用户提供更好的体验上,另从IE10开始已无此特性
```

16.避免低效率选择器

- **避免类型选择器**

不允许:

```
div#doc { sRules; }
li.first { sRules; }
```

应该:

```
#doc { sRules; }
.first { sRules; }
```

> CSS选择器是由右到左进行解析的,所以 div#doc 本身并不会比 #doc 更快

- **避免多id选择器**

不允许:

```
#xxx #yyy { sRules; }
```

应该:

```
#yyy { sRules; }
```

17.属性缩写与分拆

- **无继承关系时,使用缩写**

不推荐:

```
body {
    margin-top: 10px;
    margin-right: 10px;
    margin-bottom: 10px;
    margin-left: 10px;
}
```

推荐:

```
body {
    margin: 10px;
}
```

- **存在继承关系时，使用分拆方式**

不推荐：

```
.m-detail {
    font: bold 12px/1.5 arial, sans-serif;
}
.m-detail .info {
    font: normal 14px/1.5 arial, sans-serif;
}
```

要避免错误的覆盖：

```
.m-detail .info {
    font: 14px sans;
}
```

> 如果你只是想改字号和字体，然后写成了上面这样，这是错误的写法，因为 `font` 复合属性里的其他属性将会被重置为 user agent 的默认值，比如 `font-weight` 就会被重置为 `normal`。

推荐：

```
.m-detail {
    font: bold 12px/1.5 arial, sans-serif;
}
.m-detail .info {
    font-weight: normal;
    font-size: 14px;
}
```

> 在存在继承关系的情况下，只将需要变更的属性重定义，不进行缩写，避免不需要的重写的属性被覆盖定义

- **根据规则条数选择缩写和拆分**

不推荐：

```
.m-detail {
    border-width: 1px;
    border-style: solid;
    border-color: #000 #000 #f00;
}
```

推荐：

```
.m-detail {
    border: 1px solid #000;
    border-bottom-color: #f00;
}
```

18.模块化

- 每个模块必须是一个独立的样式文件，文件名与模块名一致；
- 模块样式的选择器必须以模块名开头以作范围约定；

假定有一个模块如前文 HTML模块化，那么 `m-detail.scss` 的写法大致如下：

```
.m-detail {
    background: #fff;
    color: #333;
    &-hd {
        padding: 5px 10px;
        background: #eee;
        .title {
            background: #eee;
        }
    }
    &-bd {
        padding: 10px;
        .info {
            font-size: 14px;
            text-indent: 2em;
        }
    }
    &-ft {
        text-align: center;
        .more {
            color: blue;
        }
    }
}
```

编译之后代码如下：

```
.m-detail {
    background: #fff;
    color: #333;
}
.m-detail-hd {
    padding: 5px 10px;
    background: #eee;
}
.m-detail-hd .title {
    background: #eee;
}
.m-detail-bd {
    padding: 10px;
}
.m-detail-bd .info {
    font-size: 14px;
    text-indent: 2em;
}
.m-detail-ft {
    text-align: center;
}
.m-detail-ft .more {
    color: blue;
}
```

> 任何超过3级的选择器，需要思考是否必要，是否有无歧义的，能唯一命中的更简短的写法

图像约定

1.图像压缩

所有图片必须经过一定的压缩和优化才能发布

2.背景图

- 使用PNG格式而不是GIF格式，因为PNG格式色彩更丰富，还能提供更好的压缩比；
- 在需要兼容IE6的项目中，尽可能选择PNG8，而不是使用PNG24+滤镜。

3.前景图

- 内容图片建议使用JPG，可以拥有更好地显示效果；
- 装饰性图片使用PNG。

4.Sprite

- CSS Sprite是一种将数个图片合成为一张大图的技术（既可以是背景图也可以是前景图），然后通过偏移来进行图像位置选取；
- CSS Sprite可以减少http请求。

结语

坚持一致性的原则。

一个团队的代码风格如果统一了，首先可以培养良好的协同和编码习惯，其次可以减少无谓的思考，再次可以提升代码质量和可维护性。

统一的代码风格，团队内部阅读或编辑代码，将会变得非常轻松，因为所有组员都处在一致思维环境中。