页面重构中的设计模式

@Ghostzhang 2014.7.01

目录

写在前面

首先，本文不适合初学者，如果你不知道HTML、CSS的实现原理，没有掌握常用的布局实现方法，更不知道为什么要做模块化，那么以下的内容并不适合你。

如果你已经有相当的页面重构工作经验，参与过项目的合作开发，对模块化有所思考，那么本文将帮你更深入的理解页面重构中的模块化。

关于设计模式

"设计模式"这个词也许你已经听过了很多次，知道的同学不需要我再重复，不知道的同学，我也没把握在这短短的时间里跟你明白，以我的理解归纳了一下：

- 设计模式是被发现而不是被发明的。
- 设计模式是在解决特定问题时，所独立开发出类似的技术来解决这些问题，它们的共通性即为模式。
- 设计模式是用来解决一类相关问题的通用技术，而不是解决问题的特解。
- 设计模式是一种解决问题的思路，而不是解决方案本身。

简单的说，我们想通过设计模式来**解决问题（某种或某类问题的解决方案）**。于是，随之而来

的问题就是：我们要解决什么样的问题？或者说，我们遇到了什么问题？

我们一直追求这样的代码：

简洁：

HTML：最少的、语义化的标签，实现所需要的结构

CSS：高复用、低冗余的定义

灵活：

HTML、CSS：最少改动，最多效果

可维护：

HTML、CSS：最快的速度定位到问题，最少的改动解决问题

于是，问题的根源就是**“如何实现HTML、和CSS代码的简洁与灵活”**，如何在HTML、CSS中实现脚本编程语言中的：“封装”、“组件”、“可移植”等等特性。

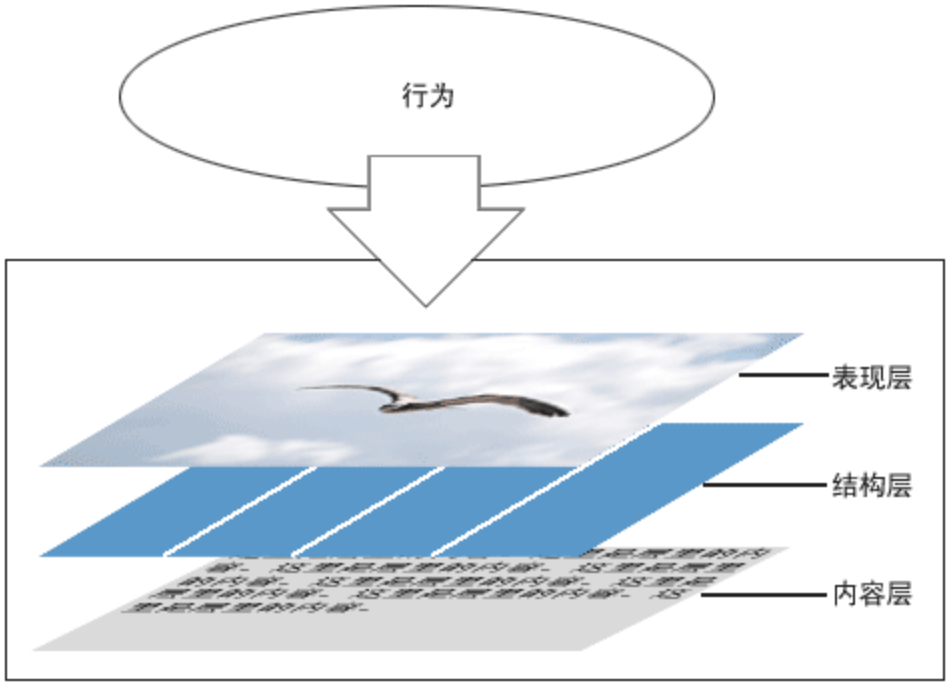
在实际的工作中，可能还需要考虑以下的情况：

- 开发效率
- 沟通成本
- 发布风险
- 容错度
- 链接数
- 打开速度（首屏）

在开始主要的内容之前，我们还需要一个思想来做为“基础”，那就是WEB标准中的“**内容、结构、表现、行为分离**”的思想，非常重要，是后继内容的基础原则。

分离——内容、结构、表现、行为

先来看一张图：



图片来源：http://www.w3cn.org/article/tips/2004/43.html
这张图应该大家都非常熟悉了，很形像的表现了内容、结构和表现三者的关系。

内容

指网页中的文本、内容图片、音频、视频等需要传达信息给用户的部分。

结构

标明内容的类型，划分不同内容的范围；

为表现上的需要提供必要的结构支持。

表现

网页的视觉呈现，辅助内容更好的传达。

行为

网页与用户的交互行为。

分离的思想，能让站点在开发、维护上获得很高的效率和便利。对于了解编程语言的同学来说，面向对象编程中的思想，也是体现着"分离"的思想。而在页面重构中，结构主要由HTML实现、表现由CSS支持。

关于模块化思维

模块化的内容已经说了很长一段时间了，从2008年开始就不断的在讨论、思考、验证、完善。个人感觉已经算是比较成熟的一个解决方案了，也得到了不少同学的认可，有意识无意识的在使用。包括现在这篇文章，也是在对模块化的思考过程中，逐渐清晰起来的。

页面重构中的模块化核心思想：**将HTML和CSS通过一定的规则进行分类、组合，以达到特定HTML、CSS在特定范围内最大程度的复用。**

有三个关键词： **规则** 、 **特定范围** 、 **最大程度的复用** 。

规则

编写模块时需要遵循的规范，如命名的规范、样式定义的规范。

特定范围

模块可使用的范围。与样式的作用域有关，大部分模块的使用范围仅仅是某一个栏目或站点。

最大程度的复用

做最少的修改即可重复使用。很多同学都把"复用"理解成不用修改的直接使用，但在网页制作中，由于实际的项目环境，基本是不可能做到"一个模块走天下"的。不同的栏目会有不同的需求，大家应该都多少有所体会，我就不多讲了。

为什么这里要提起模块化呢？因为上面我们已经说，做这些是要来解决问题的，而做好模块化，就能解决掉上面提出的大部分的问题：

- 提高代码重用率
- 提高开发效率
- 减少沟通成本
- 降低耦合
- 降低发布风险
- 减少Bug定位时间和Fix成本
- 提高页面容错
- 更好的实现快速迭代
- 更好的支持灰度发布

还有很重要的一点，模块化可以较好的实现成本的平衡。

页面重构中的模块化

HTML、CSS也有模块化？

如果你够细心，你会发现现有的理论与我们正在讨论的内容最大的区别在于，程序语言间有很强的共性，都是以一种理论为基础的，既使一个功能的实现使用了不同的程序语言。几乎所有的编程语言都会有"类"、"对象"、"变量"等概念，实现上都是"定义变量，赋值"、"如果符合条件就执行这个代码，不然就执行那个代码"，都有差不多的流程控制"条件判断"、"循环语句"等等。而HTML和CSS并算不上是编程语言，HTML是超文本标记语言，是用于描述网页文档的一种标记语言，而CSS是一种用来表现HTML或XML等文件式样的计算机语言。两者都没有编程序语言所必须具备的流程控制能力，甚至不能定义变量。这也是为什么页面重构在大多数程序员同学前面显得不太好理解吧。

在很长一段时间里，我都是把HTML和CSS放在一起去考虑的，因为在重构里HTML和CSS就是不可分开的关系，但它们之间有着不同的特性，侧重的点有所不同。HTML强调的是语义化、结构合理性，CSS则更多的偏重于灵活性。它们间甚至可以说在某种程度上是矛盾的。在相当长一段时间里，都在考虑怎样处理好HTML和CSS间的这种矛盾，试图去寻找解释。包括从程序实现上的"模块化"、"面向对象编程"去找解释，虽然有大部分概念能对应上，但还是有一部分概念需要更多的理论支持。

在学习"设计模式"这个概念时，发现自己其实一直在给自己画圈，总想着要把HTML、CSS给套到开发语言中的模块化里，而实际上CSS和HTML并不是编程语言。

再次应用分离的思想，在重构中的模块化中，一个效果的实现，需要由几个部分组成——HTML、CSS、脚本（如：Javascript），因为脚本属于程序的范畴，这里不做讨论。剩下的两个部分，我们分别来看看。

HTML模块化

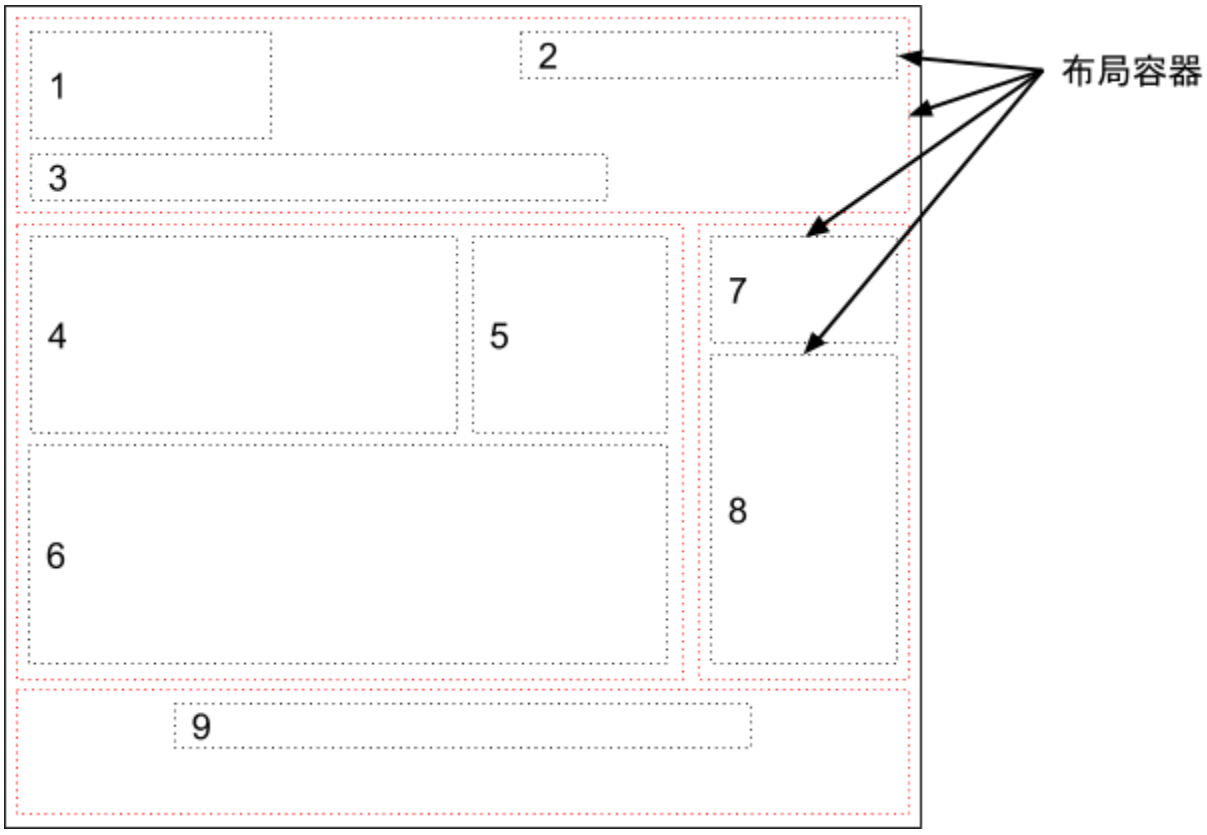
因为HTML对应着结构部分，所以这里也可以称为"结构的模块化"。很多时候同个站点中部分内容的实现是可以重复使用的，像站点的头部、底部、导航、翻页、面包屑等，虽然在不同的页面中出现，但它们的HTML结构基本是一致的（根据统一性的原则），可以更多的复用。

模块类型

是的，模块也有不同的类型，用于不同的场景，一般可以分为以下两种：

布局容器

用于页面布局的结构，一般由div标签组成，如常见的二栏布局、三栏布局等。



布局容器就像是一栋房子的墙壁，它把基本布局规划了出来，有多少个房间，房间的大小，房间与房间之间的位置等等。想像一下，如果把网页看成是一个房子，墙壁就是一个个的框架，把整个房子分成N个空间，当然了，也并不是只能用墙来做为空间的分隔，同样的，在网页中除了能看到的分隔，也有看不到的分隔，像一些辅助性的框架结构。

如果转换成HTML，就像这样：

```
<!-- 头部 [[ -->
<div>
   <div>1</div>
   <div>2</div>
   <div>3</div>
</div>
<!-- 头部 ]] -->
<!-- 内容 [[ -->
```

```
<div>
  <div>
    <div>4</div>
    <div>5</div>
    <div>6</div>
  </div>
  <div>
    <div>7</div>
    <div>8</div>
  </div>
</div>
<!-- 内容 ]] -->
<!-- 底部 [[ -->
<div>
  <div>9</div>
</div>
<!-- 底部 ]] -->
```

可是都说好的代码是可以『自解释』的，即不需要注释也可以看懂，那么我们可不可以给这些*div*加个标记呢：

```
<!-- 头部 [[ -->
<div class="header">
  <div>1</div>
  <div>2</div>
  <div>3</div>
</div>
<!-- 头部 ]] -->
<!-- 内容 [[ -->
<div class="content">
  <div>
    <div>4</div>
    <div>5</div>
    <div>6</div>
  </div>
  <div>
    <div>7</div>
    <div>8</div>
  </div>
</div>
<!-- 内容 ]] -->
<!-- 底部 [[ -->
<div class="footer">
  <div>9</div>
</div>
<!-- 底部 ]] -->
```

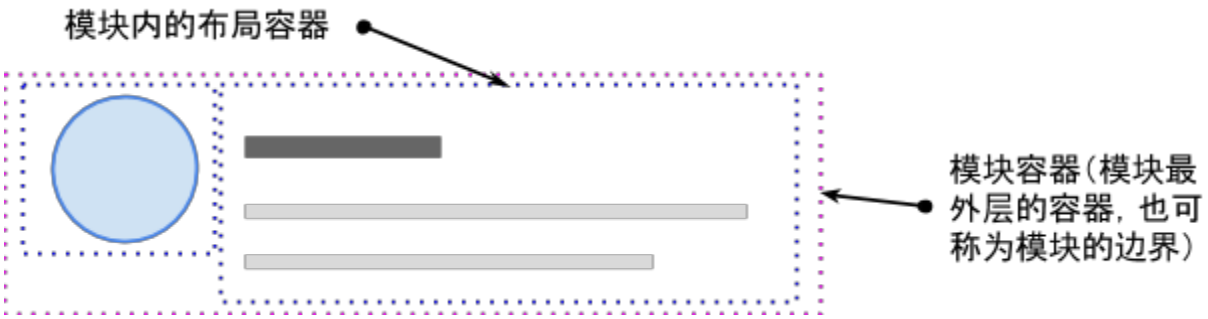
当然你如果用HTML5的标签来写的话也可以：

```
<!-- 头部 [[ -->
<header>
  <div>1</div>
  <div>2</div>
  <div>3</div>
</header>
```

```
<!-- 头部 ]] -->
<!-- 内容 [[ -->
<div class="content">
   <div>
      <div>4</div>
      <div>5</div>
      <div>6</div>
   </div>
   <div>
      <div>7</div>
      <div>8</div>
   </div>
</div>
<!-- 内容 ]] -->
<!-- 底部 [[ -->
<footer>
   <div>9</div>
</footer>
<!-- 底部 ]] -->
```

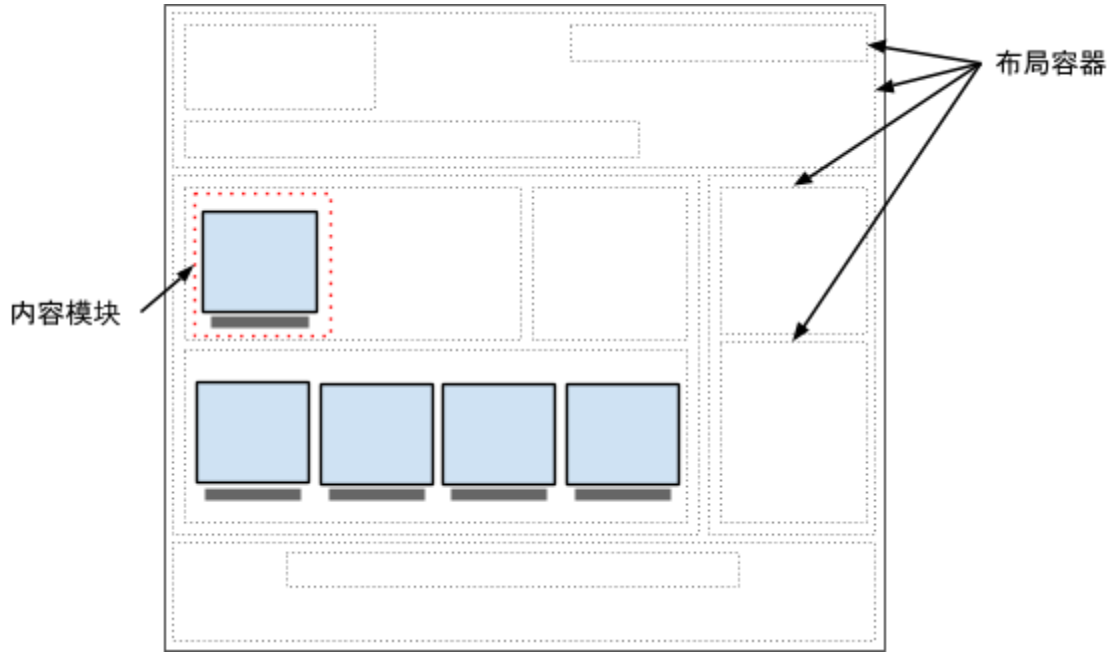
这样看起来就更清晰些了，当然用HTML5标签的话还是会有问题，根据模块化的实现原则（下面会讲到），标签的样式定义是要避免的，特别是在外层的容器上。因此，最好还是加上个*class*。

布局容器也分为两种：
用于页面的布局容器和用于模块内的布局（包括模块容器），像上面的图表示的就是用于页面布局的容器，而下面的图则是模块的布局容器



内容模块

有了框架，就可以往里面加内容了，内容模块是用于实际内容显示的模块，可以看成是具体的家具，像是床、柜子。



内容模块原型

```
<div class="基类 扩展类 实例类">
   <div>...</div>
</div>
```

其中，"扩展类"是非必须的，如

```
<div class="mod-tab-1 new-tab-name">
   <div>...</div>
</div>
```

*mod-tab-1*是基类，*new-tab-name*是实例类，也许你会问怎么看出*new-tab-name*就是实例类的名称呢？也可能是扩展类啊。这里需要用到第一个关键字——"规则"，关于命名的规则，在后面的"解决样式定义的冲突"里会讲到。

CSS模块化

由于CSS语言自身的特点，让模块化的实现起来更加的困难。很高的自由度，基本上除了最基础的格式之外，其它的内容都是想怎么写就怎么写：

- 不会因为重复定义而出错

  ```
  p{font-size:14px;}
  p{font-size:12px;}
  ```

- 不关心大小写的问题（实际应用中由于XHTML是关注大小写的，所以CSS就不能不关注大小写了）

  ```
  p{font-size:12px;} 等于 P{font-size:12px;}
  ```

- 可以不在乎定义的前后位置

  ```
  <body>
          <p class="demo-a"></p>
          <p class="demo-b"></p>
  </body>

  .demo-a{color:#000000;}
  .demo-b{color:#0000FF;}
  等于
  .demo-b{color:#0000FF;}
  .demo-a{color:#000000;}
  ```

这么自由，那CSS模块化应该从何入手呢？是的，需要有能达成共识的理论基础，或者规范。由此我们可以引出一个新的问题——如何得到这个规范？一个理论的形成，总是需要借鉴现在的理论的，这里我们可以借鉴OOP的理论：

类OOP方式

因为面向对象编程的关系，在相当一段时间里，CSS的设计都想像着用变量之类的方式去实现，甚至出现了像less、Sass等用类似编程的方法去写CSS的方案，的确可以让编写时的冗余代码大量的减少，不过最终生成的CSS还是一样的臃肿，还要为此去学习一门新的语言（虽然相对简单），对于CSS这种本身已经很简单的语言来说，实在不能说利大于弊。

最有名的实践，应该要算OOCSS了，通过使用更多的类名来实现一个样式定义更高的复用性，其实这也算得上是某种程度上的内联样式，并不能发挥出CSS的灵活性。相比起国外网站的风格，国内的站点风格更多样化，修改更频繁。当然它也有适用的场景，像需要做各种换肤的站点，Qzone是个例子，Qzone皮肤的实现就是使用了样式类拼装的方式。

但是OOP的一些设计上的原则，还是有很强的借鉴意义的，像"开闭原则"、"里氏替换原则"、"依赖倒置原则"、"接口分离原则"等。当然也需要重新的解读：

基本原则

1. **开闭原则**

   扩展性是开放的；更改性是封闭的

“开放的扩展性”，就是在基类的基础上，可以做的扩展是灵活的。封闭性的原则是指对模块内的修改，应该是封闭在模块内的，不影响到模块之外的内容。通常我们使用包含选择符的方式来实现。

```
.mod-name .class-1{...}
.mod-name .class-2{...}
```

2. 里氏替换原则

父类可出现的地方，子类可出现；反之不一定

一个能够反映这个原则的例子是圆和椭圆，圆是椭圆的一个特殊子类。因此任何出现椭圆的地方，圆均可以出现。但反过来就可能行不通。这里的父类与子类，主要是指基类与扩展类之间的关系。

3. 依赖倒置原则

接口：扩展类只负责模块效果的补充；实例类不影响扩展类与基类间的依赖关系

结构化设计：高层抽象模块依赖底层抽象模块

这里主要是指基类、扩展类与实例类间的关系，它们是共同作用于一个HTML结构的。扩展类依赖于基类，基类或基类加护展类组成一个完整的效果，实例类是做为模块在当前页面的补充定义，帮助模块实现最终的效果。

4. 接口分离原则

各接口负责不同的效果

一个样式定义比如.css{}，与HTML结构 之间要产生关联，是通过标签的class属性进行的，结果就是 ，这个class属性就是HTML与CSS之间的接口。因此，使用内联的样式定义，就与这个原则相违背了。同样的，一个HTML文件使用外链的方式引用CSS文件，也是适合接口分离的原则。

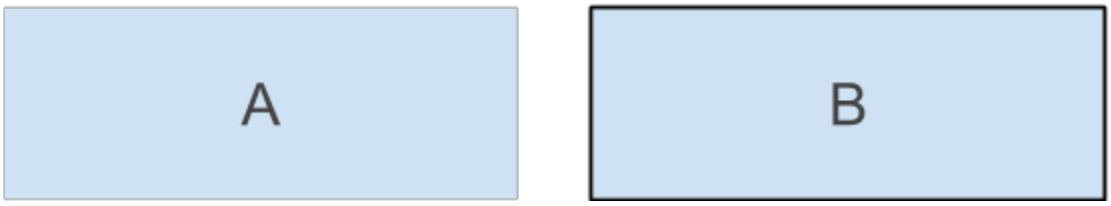
继承

扩展类与基类的实现，就是建立在样式的继承性的基础上。这里所说的“继承”与样式属性中的继承有一点点不同。众所周知的，背景（background）的属性是不会继承的，但在模块中，属性在基类中的背景，是会继承到扩展类与实例类中的。

```
.mod-demo{width:200px;height:100px;background-color:#CFE2F3;}
.new-demo-btest{border:1px solid #000000;}
```

```
<div class="mod-demo new-demo-atest">
</div>
```

```
<div class="mod-demo new-demo-btest">
</div>
```



如上图中的B继承了.mod-demo定义的宽高与背景，同时还定义自己的一个边框。

误区

对分离思想的误区

不少同学会有这种误区，以为我们强调分离的思想，是为了让所有的需求都只通过修改样式就能满足，其实是不正确的。做好分离，是为了将其中某一块内容的修改所带来的影响变得更可控。在实现工作中，一个需求的实现，往往要修改的不只是其中的某一块。

上面我们分别了解了HTML和CSS的模块化，实际的应用中，还是需要将两者结合起来，毕竟只有其中一样，是实现不了我们所要的丰富效果的。

面向"效果"的模块化设计

很多同学一听这个叫法，会有种不太容易接受的感觉，为什么要叫"面向效果"呢？与"面向对象"又有什么不同呢？

跟面向功能开发的程序语言不同，HTML和CSS并不是用于编程的语言，它们的出现，更多的为了让网页更"美观"，是更倾向于"效果"的语言，而这个"效果"，很大程度受网页的设计风格影响，这也就注定了不能简单的像面向功能开发的编程语言那样，做出一个可以被广泛使用的功能类或库，因为这样往往是满足不了需求的。像JQuery的样式库，虽然也是可以换风格，但它的限制性也是很明显的，其它的样式库也存在着这样的问题。

到目前为止，我们可以把稍微复杂的实现分为两种方式："换样式"和"换结构"。

换样式

HTML不变，通过变更CSS的定义到实现，一般的实现方法是通过改变页面外链的样式文件来实现换肤。如“禅意花园”。

换结构

把样式定义细分，通过在HTML中组合不同的样式类来实现所要的效果。

与面向“对象”的方式不同，面向“效果”的设计，更关注模块的最终效果，即模块的使用场景。结合前面所讲的模块化与类OOP的内容，我们更深入一点来看下：

模块的类型

根据模块的使用范围，大概可以划分为以下两类：

1. 内容模块：我们通常所说的模块
2. 页面元素：一般通用于所有页面，独立存在。

也可分为：

1. 不可嵌套：页面元素、不可嵌套的内容模块
2. 可嵌套：内容模块

框架，顾名思义就是用于布局的结构，用于承载内容。页面框架主要用于划分页面上模块的区

域、位置；模块框架则更多的在于规划模块内的页面元素。

基类

基础模块定义，一般我们会把经常用的模块定义为基类，以减少代码量。

扩展类

当一个基类在不同的位置使用时，有可能会因具体环境而需要有**大部分展现上**的或**小部分结构上**的调整，如果通过复制新建模块的方式，会出现大部分的代码冗余，这时可以通过定义一个扩展类来解决。

实例类

用于模块在具体页面中的细节修正，如位置、颜色等只在当前页面的展现。

模块样式的定义

一般情况下，模块的定义使用后代选择器来完成，尽量控制在三层或以下，可以使用以下的规则：


```
<div class="mod-news new-index-local-news">
        <div class="news-title">
<h3 class="title">本地新闻</h3>
<p>更新时间 <span class="time">2012-04-10</span></p>
</div>
        <div class="news-lists">
                <ol>
                <li><span class="title">新闻内容</span><span
class="time">2012-04-10</span></li>
                <li><span class="title">新闻内容</span><span
class="time">2012-04-10</span></li>
                <li><span class="title">新闻内容</span><span
class="time">2012-04-10</span></li>
                <li><span class="title">新闻内容</span><span
class="time">2012-04-10</span></li>
</ol>
</div>
</div>

/* 基类 */
.mod-news{}
.mod-news .news-title{}
.mod-news .news-title .title{}
.mod-news .news-lists li{}
```

```
.mod-news .news-lists .title{}
.mod-news .news-lists .time{}

/* 实例类 */
.new-index-local-news .news-title{}
.new-index-local-news .news-title .title{}
.new-index-local-news .news-lists .title{}
.new-index-local-news .news-lists .time{}
```

除了模块名和作用对象之外，中间的用于控制作用范围的选择器可以尽可能的少，只要能与其它的定义区分开即可，并不一定非得把所有的层级都写进去。这样可以更好的减少权值的大小，有利于后继的重定义。

TIPS：

并不是所有的模块都必需很健壮。

虽然我们都希望我们写出来的模块十分的健壮，可以在从任何的地方使用，但事实上并非如此，健壮的代码意味着更多的冗余，这对于网站整体来说并不是最有利的。

解决样式定义的冲突

看完上面的内容，你可能已经跃跃欲试了，不过在此之前，还有些细节的东西需要进一步的了解下：

样式的作用域

作用域是模块化最核心的部分，也是第二个关键字——"特定范围"，是指样式定义所能影响到的范围。掌握了样式的作用域，可以说就掌握了模块化，它可以由以下的方式体现：

样式的优先级

决定一个定义是否最终能够生效，有以下几条规则：

1. 不同的样式定义有不同的权值，具体的值不得而知，但优先顺序大体如下：**important > 内联 > ID > 类 > 标签 | 伪类 | 属性选择 > 伪对象 > 通配符 > 继承**
2. 只有同类的定义才需要比较大小，否则遵循权值的优先级
3. 同类定义的比较中，如ID与ID比较，权值相同时，遵循"就近原则"，即最接近标签的定义优先，因此，样式定义所在的位置也是控制作用域所必需掌握的，优先级为：**内联 > 内嵌 > 外部**；同个文件内的优先级：后定义的高于先定义的。

了解优先级的目的在于，更准确的知道这个定义是否生效，如果没生效，原因是什么。当然，我们现在有很高科技的开发者工作可以很快的查看到定义是否生效，用于分析问题是很方便的，但如果我们没有有效的控制方法，即便你找到了错误，修复也会遇到问题，也许改出新的问题也不定。

换句话说，我们要以最小的权值实现对模块的扩展和作用，减少样式的污染。

样式的作用域与标签的关系

样式的作用范围，与HTML的标签是很大的关系，除了标签的权值之外，还能影响到作用的范围。同样的一个样式定义，在不同的标签位置时，它所影响的范围是不同的。

```
.demo{color:#FF0000;}
```

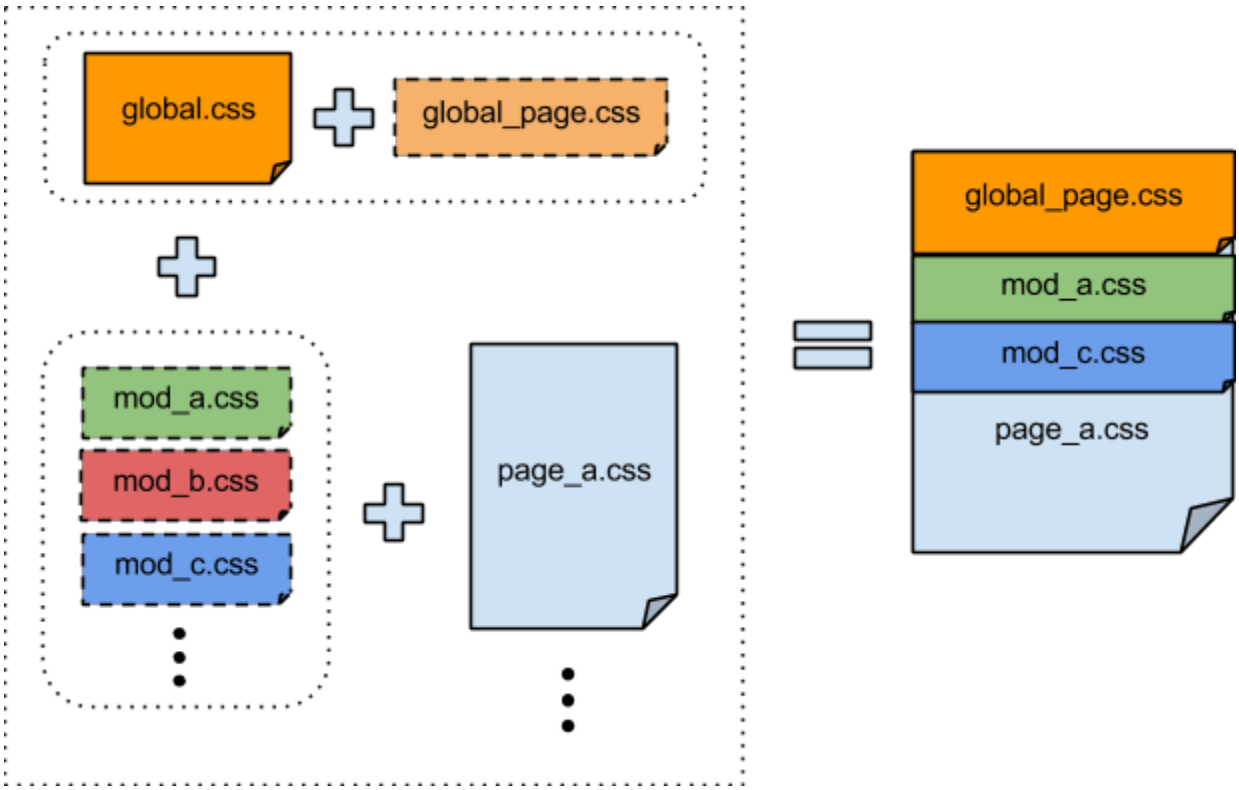
```
<p class="demo">
作用在这里
</p>
```

```
<div class="demo">
   <p>作用在这里</p>
   <p>还有这里</p>
</div>
```

我们知道如果把样式的选择符以一层层的标签写出来，是可以最精确的定位到样式的作用节点的，但这样代码会很长，也会丢掉CSS的灵活性，在实践中发现，一般情况下，三级的后代选择器已经足以选择到所要的节点，这也是模块样式的定义中为什么选择三层的原因。

样式的作用域与在文件中的位置的关系

根据"就近原则"，当权值相等时，我们可以用位置来控制样式定义的优先级，如：



“global”的范围一般只限定在一个站点中，即不同站点的“global”基础并不一定存在通用性，同样名称的基类，在不同的站点中实现上可能是不一样的，因为规范一般都只对于同一个站点有效，如果要多个站点间也达到通用，这就相当于是跨站点的规范了。

由于同文件内的优先级是后定义的高于先定义的，于是要关注各部分在一个文件中的具体位置，像公共的定义要放在模块与页面的定义之前。

TIPS：

可以在每个样式文件中划分出具体的位置，不同的定义对号入座。

样式定义在文件中的位置分布：

```
/* == S 全局定义 == */
html{}
/* == E 全局定义 == */
/* == S 通用模块 按需 == */
.mod-xxx-1{}
.mod-xxx-1 .a{}
.mod-xxx-1 .b{}
/* == E 通用模块 按需 == */
/* == S 页面级扩展模块 == */
.exp-xxx-1-1{}
.exp-xxx-1-1 .a{}
/* == E 页面级扩展模块 == */
/* == S 页面定义 == */
.new-xxx-1-1-news{}
.new-xxx-1-1-news .a{}
/* == E 页面定义 == */
```

不可入侵的区域

不可嵌套模块的内部区域为不可入侵的区域，在模块内的定义，必须严格限制在模块内部，同时也要注意避免被模块外可继承的样式声明

样式命名规则

内容模块的命名

基类命名

前缀(mod)-模块名-模块编号

前缀：基类的模块标识。如“mod”。

模块名：模块名称。

编号：模块的编号，用于区分同类模块。当只有一个版本时不需要写模块编号，即基类命名中模块编号一定是大于“1”的。

例：

```
.mod-tab{}
.mod-tab-2{}
```

扩展类命名

属于栏目级的公共模块定义。

前缀(expand|exp)-无前缀基类名-基类模块编号-扩展类编号

前缀：扩展类模块标识，如“exp”。

例：

```
.exp-tab-1{}/*扩展类1，基于基类tab*/
.exp-tab-2{}/*扩展类2，基于基类tab*/
.exp-tab-2-2{}/*扩展类2，基于基类tab-2*/
```

实例类命名

页面级模块定义，与ID类似，有在一个页面中唯一的特点。

前缀(new)-无前缀基类名-基类模块编号[-扩展类编号]-名称

前缀：实例类标识。如“new”。

作用：定义类的作用，用于对类型的补充。

状态：定义类的状态，用于对类型的补充。

位置：定义类所使用的位置，如首页、导航等等，不排除使用左、右这样的词，但应尽量避免。

例：

```
.new-news-page-form{}/**/
.new-index-login{}/**/
```

用于模块内部的命名

模块框架命名

前缀(lay)-[位置|区域]-名称

例：

```
<div class="lay-local-name">
        <div class="box-name-a">
                <div class="mod-modname-1 exp-modname-1-1 new-modname-1-1-newname">
                        <div class="g-modname-1"></div>
                </div>
        </div>
        <div class="box-name-b">
                <div class="mod-modname-1 new-modname-1-newname">
                        <div class="g-modname-1"></div>
                </div>
        </div>
```

```
</div>
```

其它命名

状态类

功能名-状态

状态类一般以表示状态的单词来表示，只用于模块内部，即定义时必须限制在模块内部。一般与实例类放在一起。

例：

```
.new-index-login .nonce{}
.new-index-login.hover{}
```

元素类

前缀(类型名)-名称

前缀：元素类型名称。如“icon”。

例：

```
.icon-ok{}
.icon-error{}
```

尽可能的使用不带状态的名词

TIPS:

同一个单词可以被多处使用，但当可能出现嵌套时，应该为它加上前缀。

当一个定义可能被继承或影响到其它同类标签的时候，应该选择使用“class”。

CSS样式属性义类型

CSS样式的属性义有很多，像手册上就分为“字体”、“文本”、“背景”、“定位”等等17种，在实际工作中，并不需要分得这么的细，常用的属性大致可以分为以下两类：

表现属性

与颜色、字体、图片相关的属性，如：

```
color:#FF00FF;
font-size:12px;
background-color:#FF00FF;
```

```
background:#FF00FF url(demo.img) top left;
```
这类属性一般用于皮肤的定义，

布局属性

与位置、大小相关的属性，如：

```
width:100px;
position:relative;
top:10px;
```

样式注释

由于样式命名上已经做到了模块的自解释，所以注释这部分已经没有太多的内容需要写上了。
Demo

```
/** mode **
@author:作者名
@note:模块说明文本 \n
模块注释内所有内容都以关键字开始，换行结束 \n
如需换行:可添加 "\n" \n 也可以连着写，但如果换行书写时，必需作为结尾。不能有空行。
@demo:\ref{说明文本:URL}
@html:
<!-- HTML结构 -->
<div class="mod-tab-1">
        <div class="tab-title">...</div>
        <div class="tab-cont">...</div>
</div>
@endhtml
*/
.mod-tab-1{}
.mod-tab-1 .tab-title{}
.mod-tab-1 .tab-cont .title{}
.mod-tab-1 .tab-cont .cont{}
.mod-tab-1 .tab-cont .op{}
/* @{
@note:模块内区域注释
@demo:URL地址
@html:
<!-- HTML结构 -->
<div class="mod-tab-img-1">
        <div class="tab-img-lists">...</div>
</div>
@endhtml
*/
.mod-tab-img-1{}
.mod-tab-img-1 .tab-img-lists{}
/* @} */
/* @end **/
```

说明

- 以"**/** mode **"标记模块的开始
- 从"**/** mod **"到第一个"***/**"作为模块相关信息的说明，包含关键字
- 关键字以" **@**"开头，"**:**"后开始到一行结束的内容为相关的值，即：@关键字:值

- 以"**/* @end **/**"标记模块的结束
- 模块注释内不可嵌套
- 模块内部小区域注释可包含在"**/* @{**"与"**/* @} */**"之间。

相关关键字

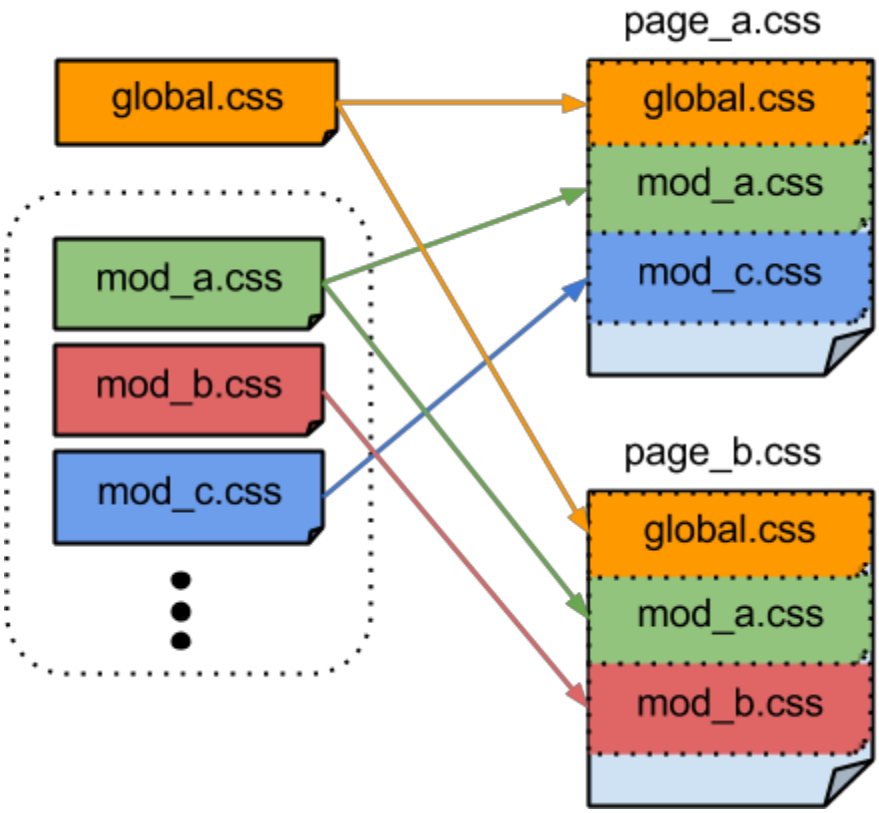
@name	标明模块的名称
@author	标明模块的作者
@version	标明该模块的版本
@note	功能说明
@relating	标明该关联的模块
@dependent	标明该所依赖的模块
@type	模块的类型分为三类：公用、基类和扩展类
@demo	指向一个外部的示例地址
@html	样式相关的HTML结构
@endhtml	*@html*的结束标记

文本中的关键字

\ref{说明文本:URL}	添加一个链接
\n	添加一个换行

模块管理的问题

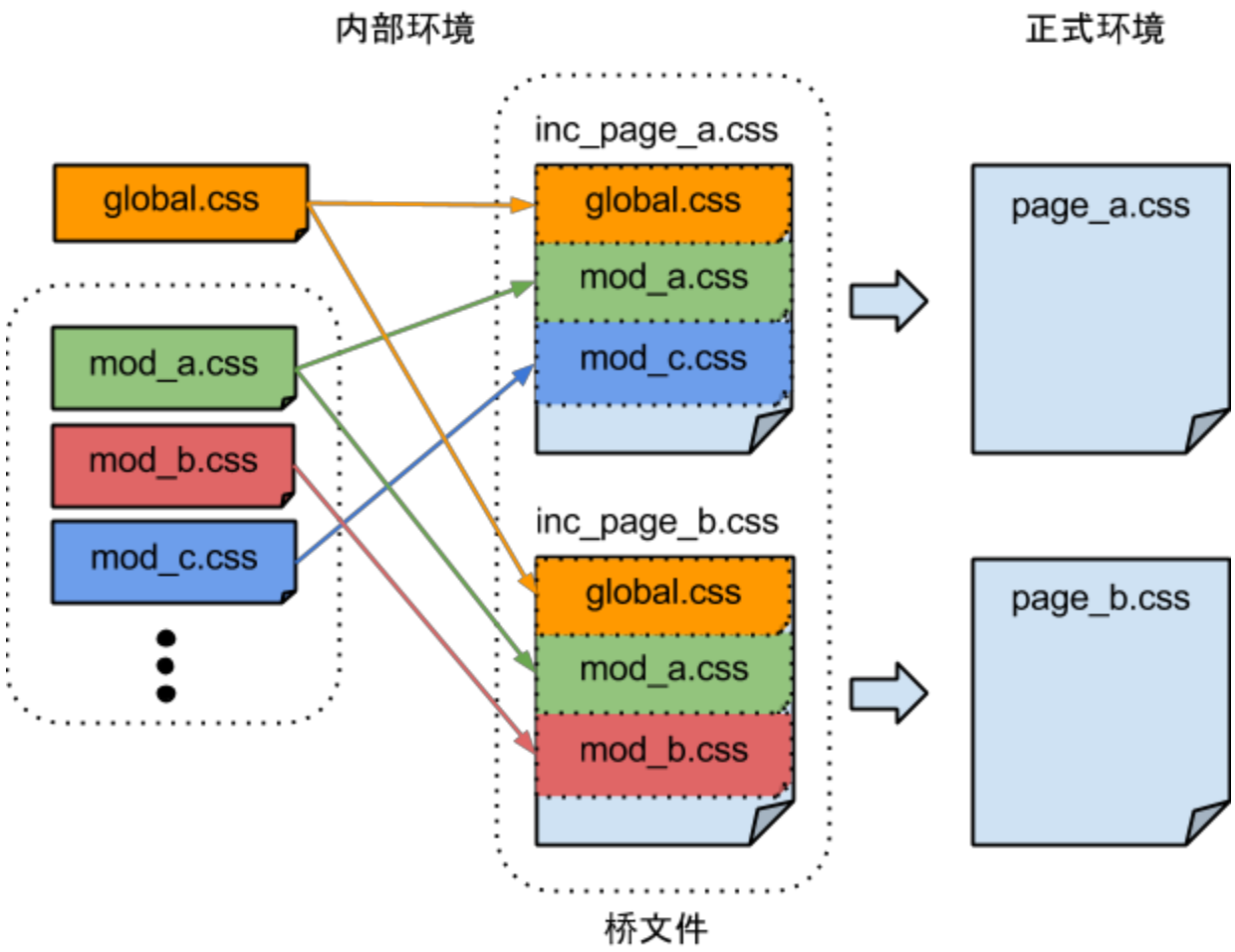
当你的模块越来越多之后，模块的管理问题就会随之而来。公共文件越来越大，一些使用不上的模块也因为放在公共文件中而被不断的重复加载，造成带宽的浪费。聪明的你一定想到了，把模块拆出来，按需加载：



如上图，每一个模块分别拆成独立的文件，然后通过引用外部文件的方式链接到当前页面的样式文件里，从而实现按需加载。然而，这样会引发另一个问题，链接数的增加。一个页面有多少个模块，就会增加多少个链接，显然不是我们所希望的。

我们都知道，解决链接数过多的方法很简单，就是把它们合并就行了。那么，我们同时需要解决合并后所可能带来的问题——怎么更新的问题。

使用“桥文件”解决文件更新问题



至于如何把“桥文件”转为合并后的文件，你可以手工操作，也可以借助自动化工具。

至此，大部分的理论的讲完了，剩下的就是通过实践去巩固这部分的知识了。

@Ghostzhang

CSSForest.org