Sunday, November 2, 2014

注意点:

1. PPT修改成16:9 比比例布局, 一一般用用黑黑字, 重点用用红色色
2. 代码编辑器使用用白白底,字体24以上,网网网页页放大大显示示,调整工工具的字体也放大大
3. 讲课时,尽量不要用用老老师、同学什么的,用用第二二人人称、交流。
4. 举例什么的,尽量避开一一些政治、法律、热议问题等。
5. 所有写代码和命令行行的操作界面面要求全屏显示示,建议采用用白白底黑黑字,微软雅黑黑字体,字号不得
小小于24号,屏幕分辨率为1080*720或1366*768时,字号在24号或以上,分辨率为1920*1080 时字号设置为28号或以上

Sass and Compass In Action

《第一章：课程介绍》

大家好, 我是mater, 很高兴能跟大家一起分享一些Sass, Compass相关的知识和经验, 讲到Sass呢, 就一定绕不开一个命题, 那就是Sass，Compass和CSS这三者到底是个什么关系。

（翻页）曾经有位古人说过，如果你手里有一把锤子，所有东西看上去都会像钉子。还有一位今人说过，如果你有一个钉子，就会满大街找锤子。 虽然我这里放了一张老罗的照片， 但请注意，锤子不是老罗手机。 我要说的是无论是锤子还是钉子， 都是工具。 而Sass和Compass恰恰扮演的就是锤子和钉子的角色。 CSS则对应使用锤子和钉子最后完成的作品。 这个时候就一定会有同学有疑问： 反正最后都是直接生成CSS， 那我为什么不直接写CSS呢？ 这么问的话就图样图森破了， 有了做工具的人和工具，才有可能实现几十倍甚至成千上百倍效率的提升。那Sass和Compass这两个工具又有什么不同呢？ Compass是在Sass的基础上二次开发的工具，Sass如果是钉子，我们用手按很难把它深深地钉到墙里边，但有了Compass这个锤子就不一样了，锤子+钉子那就是天降神器啊。当然了， 用工具， 一定要用好工具。 用好工具的基础是对工具有足够的了解并加以练习。而这正是我们课程的目的之所在。

(翻页) 那么学完每次课程我们到底能掌握哪些知识呢？

1. 掌握如何使用Sass和Compass写出更优秀的CSS。
2. 痛殴CSS编写过程中，这么多年的拦路虎，比如说精灵图自动合图，属性的浏览器前缀处理。
3. 在项目的开发，生产周期内，使用的样式，图片，字体等内容，如何更好的组织起来。

(翻页)这个东西说起来这么好，那到底适不适合我们自己当前的所在项目或者对职业上有所帮助呢？

1. 如果你是一个重构同学，写了很多的CSS，却不知道如何自动化CSS的书写过程。

2. 如果你是一个前端JS工程师， 会在项目的开发，生产周期内使用样式，图片，字体等等，希望能够在整个项目周期内更好的组织这些内容。

满足以上两点，你就可以继续后边的内容了。 如果同时你还了解Yeoman，Grunt等前端自动化工具的知识那就再好不过了，当然不了解也没关系，不影响你学习Sass和Compass，只是会比较难形成一个Sass和Compass在现代web项目中是如何应用的一个整体概念。当然了，以上所有条件都基于你懂CSS，如果你对CSS一无所知，我建议你先学习CSS的相关知识。如果你想一步登天，那么你必将摔得体无完肤。

(翻页)

好, 这堂课我们就讨论到这里, 下一一节, 我将跟大家一一起搭建我们必须的本地环境。 谢谢大大家。

《第二章：准备工作 - 第一节： Sass简介及安装》

大家好， 我是mater。 这一节我们来正式介绍Sass以及本地环境的安装和命令行的使用。

上一节呢，我们已经说过， Sass和Compass是锤子和钉子， 是工具， 是我们用来书写CSS的工具。这当然是蛮草根的说法， 我们来看一看Sass的官网是怎么说的。

打开，balabala。。。。。。

对中国一知半解的老外常常会觉得我们很不谦逊，因为他们看到我们的钱上印着：中国人民很行。大街上随处可见的 中国很行， 北京很行， 商业很行， 农业很行。 殊不知他们自己才是吹牛不纳税的主。

Sass是这个世界上最成熟，稳定，强大，专业的CSS扩展语言。 啧啧， 很敢说啊。 当然了，说的有底气。

Sass引擎是用ruby写的， 当然你基本上不需要care这一点， 除非有一天你需要修改这个引擎的ruby代码。

拉到页面的底部， 有一个Contributor的介绍： 大家感兴趣可以一一点进去看看这些人的主页， 都是经验非常丰富的互联网从业者，就职于谷歌，LinkedIn等大公司。 当初翻译《Sass与Compass实战》这本书的时候跟其中一些人有过接触，都非常的nice。

接下来我们看一下Sass的工作流程（PPT）。

在开发阶段，我们书写Sass代码， Sass代码被Sass引擎编译成最终的CSS代码， 最后在部署阶段，我们部署的是生成的CSS代码。 从这个流程我们也可以看出： Sass并不是万金油，使用命令行也好，使用服务端继承框架也好，使用图形用户界面工具也好， 这里的关键环节是输出CSS。剩下的就跟你平时的做法无异了。也从另一个侧面反映了Sass的语言特性是帮我们更快地写出具有高可维护性的CSS代码，如果我们本身对CSS

一无所知，或者CSS技能不够好， 即使用了Sass， 也一样无法生成duang duang的CSS代码。相反如果你精通CSS，很快就能上手Sass。Sass专注的是怎样创建优雅的样式表，而不是其内容。

可能大家已经注意到了，在Sass的源码表示区域，有两种文件后缀，scss和sass。这是咋回事呢?

（PPT）这就要说到Sass的历史了。Sass支持两种语法， 最初的缩进式语法使用.sass作为文件扩展名， 这种语法对空格敏感， 所以选择器下面的属性要有缩进，并且不使用花括号， 每个属性之间通过换行来分隔而不是分号。 这是一种类ruby的语法， 这种语法对于我们写CSS的同学来说其实是比较陌生的。

所以Sass在3.0版本的时候就引入了Scss的语法，使用.scss作为文件扩展名，SCSS语法是CSS3的一个超集，到处都是我们熟悉的花括号和分号。 正因为此我们可以无缝的把老项目的CSS切换成Sass驱动的， 只需要把原来的CSS文件的后缀名从.css改成.scss即可。 这一改进对于Sass的推广普及起了至关重要的作用。

对于这两种语法，Sass会一直持续的支持。我还是那句：很多时候没有所谓的好与不好，选择合适的最重要。如果你的团队很多人使用Python或者Ruby，那么Sass的方式可能更合适，如果你的团队是一个地道，正统的前端团队，那么Scss的方式无疑更好。当然你也可以在一个项目中混合使用两种语法，但是注意不能在一个文件中使用两种语法。

接下来我们看一下Sass的安装。 因为Sass引擎是使用ruby编写的，所以我们首先需要安装ruby运行环境。

打开ruby的官网， 在Documentation页面下有一个installation(/ˈɪnstəˈleʃən/) 的链接， 里边有非常详细各个系统的安装说明。

我们这里挑mac osx， 和windows来演示， linux使用我们后边提到的rvm，其安装跟mac基本一致，即使不用rvm，使用linux自带的apt-get yum等包管理工具来安装也是非常容易的，安装页面上给出了对应的安装指令，我们就不赘述linux的安装了。

我们首先演示mac平台，等下再开虚拟机演示windows。

无论在哪个平台，我们首先使用ruby -v 来测试本机是否已经安装了ruby运行环境。

mac平台自带ruby运行环境。但一般系统自带的都是比较老的稳定版本，有时候我们也可能需要使用某一特定版本的ruby, 或者在不同的ruby版本间进行切换， 推荐大家安装rvm， 当然使用rvm就是用mac系统自带的ruby也是没有什么问题的， rvm只是有助于提高我们的控场能力，毕竟上帝视角令人陶醉吗。 rvm的官网是： https://rvm.io, 即 Ruby

Version Manager, 看介绍：RVM是一个让安装，管理，协同多个ruby环境变得更加简单的命令行工具。ruby环境意指ruby的解释程序和ruby程序包，我们把ruby程序包称为gem，gem跟node的npm 概念非常像。

使用主页上给的指引安装即可。

我们先说一下curl：

curl是利用URL语法在命令行方式下工作的开源文件传输工具，说直白点，这里你可以把它理解成一个下载工具，说一下这几个参数的意思 -s 是说不展示下载进度和错误信息，静默下载。 -S是配合-s使用的，意思是当下载失败的时候还是要个错误信息的，还有一个-f的参数，等下我们会看到，如果是http error，比如说404，403这种，这种情况curl会返回404，403这些信息，加了-f参数之后，取而代之的是统一返回一个22的错误码。-L 的意思是当访问的url地址返回一个302这样重定向header的时候， curl跟进到重定向后的地址进行下载。

l 管道，将下载到的内容传递给bash指令， 执行下载到的文件， -s 用来指定参数， 参数是 stable. 指明安装稳定版 RVM.

回过头来说gpg：

gpg大家可以简单把它理解成一个加密软件。这里跟大家科普一下它的历史，1991年的时候有个程序员哥们为了避开政府监视，开发了一个加密软件叫做PGP，（Pretty Good Privacy/ˈprɑɪvəsi/），然后这个软件就火了， 因为它是一个商业软件，得掏钱才能用，所以火了以后，自由软件基金会决定开发一个PGP的替代品，就是GunPG，简称GPG。

mac 本身没有自带gpg程序，需要我们额外安装， mac上的软件管理推荐大家使用homebrew，很像ubuntu下的apt-get， fedora下的yum。自动帮我们下载安装相关软件并处理其依赖。

进入homebrew的主页： http://brew.sh/

因为mac已经自带了ruby， 所以可以直接copy页面上给出的安装指令 使用curl下载homebrew的安装脚本， 然后使用ruby执行这个脚本， -e 用来指定待执行脚本。

安装好了以后，就可以通过homebrew来管理安装mac软件了， brew install 后边跟相关程序的名字即可。

这里推荐一个homebrew的图形化辅助软件，叫做cakebrew， 我们进到cakebrew的官网看一眼。 https://www.cakebrew.com/

点击下载， 是一个标准的mac安装程序，一键式傻瓜安装，不赘述。 安装好了以后， 打开。 它会列出所有我们通过homebrew已经安装的程序， 其中有更新的， 以及现在通过homebrew可安装的所有程序， 这里解释一下leaves为什么比Installed多，举例说明： 这里我们搜索安装 gnupg， 安装新版本 gnupg2， 点击选中 有一项是dependencies， brew会在安装gnupg的时候帮我们把dependencies中的依赖程序一并安装了，而这些是不算做leaves的， gnupg才算是一个leave。 好， 双击， 按提示进行安装。

安装好了一下， 按照rvm的官网指示执行gpg， 这句的意思是： 指定我们的密钥服务器为kyes.gnupg.net hkp: 是密钥服务器的网络协议。 然后把指纹 409B** 对应的公钥下载下来。 而这个公钥是用来在安装rvm的时候，rvm对下载下来的程序解密用的，只有解密成功才能正常安装。 以此来保证没有下载到被人篡改过的rvm程序。 只能说老外对于安全的重视意识比我们好的不是一点半点。

好， 按照指令执行完， rvm就安装好了。

说一下如何使用rvm进行ruby的安装。 rvm的指令非常简单，好记：

rvm list known 列出所有的已知的ruby版本。

rvm list 列出你本机通过rvm安装的ruby环境， 大家可以看到我本地安装了ruby 2.2.0 版本号前边的符号啥意思， 下边通过注释说明了： =* 表示当前使用的ruby版本，以及默认使用的ruby版本。 单独一个*表示默认使用的ruby版本。 => 表示当前正在使用的ruby版本。所以这里的意思是默认使用*** 当前正在使用的是***

我们再安装一个2.0.0 版本， rvm install 2.0.0 只需要在后边跟你想要安装的ruby的版本号，只要能唯一标识这个版本即可。

现在我们有两个版本了， 如何在两个版本间切换呢？ 直接使用use命令即可， rvm use 后边跟上我们要使用的版本的版本号。

如何指定系统默认使用哪个ruby版本呢？ rvm use —default 2.0.0 后边跟上我们要默认使用的版本的版本号。

如果我想切回去使用系统预装的ruby呢？ rvm use system 即可。

再list 看一下状态， 发现通过rvm安装的这两个ruby版本前边都没有 = 号了。 我们使用较新的2.2版本， 这样方便我们窥探新特性。 rvm use 2.2.0

好， mac上的ruby环境安装就说完了，接下来我们切到windows，看看如何安装ruby。

windows上安装ruby则相对简单的多， 我们直接在ruby官网上， 点击RubyInstaller 跳转到rubyinstaller官网， 下载其安装程序, 根据我们的机器选择合适的版本， 因为我虚拟机是安装的32位windows，所以这里我选择 Ruby2.1.5， 64位机器请选择 x64. 下载好了以后直接点击安装， *** 因为我们要在命令行中使用ruby，所以把ruby可运行文件加到我们的System Path中， Install, Finsh即可。

这时候打开我们的命令行， 输入 ruby -v 正确打印出ruby的版本说明安装成功。

好， 回到mac环境， 剩下的在命令行内的操作在各个系统下就都完全一致了， 所以无须担心。

前边我们已经提到gem相当于ruby的包管理工具， 所以安装sass， compass这些都是使用gem安装， 安装ruby环境的同时gem也就安装好了， 这就像我们安装node的同时，npm也就安装好了是一样的。

但是正式安装之前我们先来修改一下我们的ruby软件包的source， 因为国内网络的原因，你懂得， 导致 rubygems.org 存放在 Amazon S3 上面的资源文件间歇性连接失败。所以我们换用国内淘宝提供的ruby sources。

gem sources --remove https://rubygems.org/ 首先移除掉 rubygems.org

gem sources -a https://ruby.taobao.org/ 添加淘宝的源

gem sources -l -l list 当前所有的源， 看看是否前两步操作成功。

成功以后， 我们通过gem install 来安装sass和compass，

gem install sass

sass -v 现在Sass最新的版本是3.4

既然提到了gem，我们就顺便多说两句， 一段时间以后我们可能需要更新所有的ruby程序包， 执行 gem update 即可。

如果我们需要安装某一特定版本的ruby程序， 比如说sass 3.3 则在后边跟 gem install sass —version=3.3

然后我们执行 gem list 列出我们所有本地安装的ruby程序包， 可以看到sass现在有两个版本。

如果想删除Sass 3.3， gem uninstall sass —version 即可， 想删除sass 就是 gem uninstall sass.

好， 常用的gem命令就这些， 接下来就让我们进入Sass和Compass的世界。

以前课程的老规矩，我们首先创建一个项目目录： 这里我们叫做 learn-sass-cli 进入**

生成一个 main.scss 文件， * {margin:0; padding:0}， 保存

我们在当前目录执行sass命令，把它编译成css文件， sass 后边跟sass文件的名字 然后是你要输出的css文件的名字。

看一下目录下的所有文件， main.css 是scss文件经过编译之后生成的css文件， .map则是用来映射scss和css文件每条规则的文件， 方便我们在chrome中调试样式。 .sass-

cache 目录中存放的是Sass编译过程中生成的临时文件， 这些临时文件可以让编译过程更迅速。我们不需要管它。

其实真实的项目开发中， 我们很少直接使用sass命令， 一般都会使用Compass。

下一节我们就来一起聊一下Compass的安装及使用。

《第二章：准备工作 - 第二节： Compass简介及安装》

大家好， 我是mater。 这一节我们来正式介绍Compass及其配置，使用。

我们首先打开Compass的官网： http://compass-style.org/ 一对比就发现Compass明显比Sass低调，还记得Sass官网怎么介绍自己的吗？Sass是这个世界上最成熟，稳定，强大，专业的CSS扩展语言。 而Compass只用一句： Compass是一个开源的CSS书写框架来描述自己。 对于我们这种标题党完全没有吸引力吗。 所以我们只能继续往下看： 这里给出了为啥大家喜欢Compass的几大理由： 因为初中的时候父母对我的期望就是读完初中，回家种田，娶妻生子一辈子。自己也抱着英语可能我这辈子都用不到的愚蠢想法，就没好好学， 后来上了高中，虽然恶补上了读写， 音标却成了永远的痛， 所以还望大家海涵发音， 会意就好。

1. Experience cleaner markup without presentational(/ˌprezən’tei-ʃənəl/) classes.

   体验如何不使用过多的表意类名的修饰来保持我们标签的简洁性。

2. It’s chock full of the web’s best reusable patterns.

   它本身包含了很多最佳的web上的可重用范式。

3. It makes creating sprites a breeze.

   它让创建雪碧图变得如此轻而易举。

4. Compass mixins make CSS3 easy.

   Compass 的mixins 让CSS3变得如此容易。 大家可以把mixins理解成我们常说的函数

5. Create beautiful typographic rhythms.

   创建美丽的段落文本布局， 老外喜欢称之为段落韵律

6. Download and create extensions with ease.

这句话的意思是： Compass周边生态丰富， 有很多扩展， 无论是你自己创建， 还是下载使用别人的。

然后介绍了一下Compass的构成， Compass是在Sass的基础之上二次开发的一套CSS框架。

好，接下来我们就来安装一下Compass， 安装方式非常简单， 跟sass一样，属于标准的gem程序包， 使用gem install命令安装即可。

gem install compass

compass -v 现在Compass的最新版本是 1.0.3

compass用起来就比Sass要复杂一些了， 它有一整套项目流程化的东西。

老规矩，我们首先进入我们的workspace

compass可以直接用来搭建我们的前端项目的样式部分， 当然这其实并不是大家现在常用的方法， 所以在后边我也会介绍如何将compass跟现有的前端集成化解决方案整合

直接在命令行下执行

compass create learn-compass-init 后边跟我们的项目名字。

好， 稍等片刻我们的项目就创建好了， 它会告诉你创建了那些个目录， 以及文件。我们不急着直接看目录结构，先看看下边这一段话说的啥。

大家都知道国内的IT圈其实是比较浮躁的， 所以很少有人会去耐心的看这么一大段话到底是在BB啥， 而我的态度是这样的： 第一次遇见这种的时候， 还是要读一读， 而之后再遇到便可以直接忽视了。 因为老外虽然啰嗦， 但是这种信息他也一定是再三权衡应该说些啥的。

恭喜！ compass 项目创建成功。 - 先给你吃颗定心丸

你可以在项目的sass子目录添加和编辑sass样式文件。 开始介绍生成的目录结构了， 从这句话我们知道有个sass子目录， 里边用来存放sass文件。

以一个下划线开头命名的Sass文件叫做局部文件， 主要用来被其他的sass样式文件引入， 其在编译的时候不会被单独编译成CSS文件。 告诉我们sass文件如何来命名。

可以通过修改生成的config.rb配置文件来配置我们的项目。 指明了生成的config.rb文件的作用。

在sass样式发生改变的时候，必须把它们编译成css才能真实生效。有下边两种方式：

1. 按需编译：

使用命令 compass compile [] 后边跟项目路径， 这里用[]括起来， 说明这个参数是一个可选参数， 可以直接在项目目录下运行 compass compile 就是编译当前项目了。

2. 监听文件变化并自动编译：

使用命令 compass compile [] 后边跟项目路径， 一样的， 项目路径是一个可选参数， 在项目目录下直接运行 compass compile 就是监听当前项目

这是告诉我们compass的编译方法。

更多的学习资源：

compass的官网， sass的官网， 以及compass的google group

这是告诉我们持续学习compass的方式。

按照以下方式将compass自动生成的样式文件引入到我们的html文件， 样式就算用起来了。

读完了这个长篇大论，我们基本上已经知道compass怎么用了，初次读来还是受益良多的。

好， 看一下真实的生成情况：

在我们的工程目录生成了一个我们的项目名命名的项目目录。 进入这个目录

有一个 config.rb 配置文件， sass源文件存放目录， css样式编译输出目录 stylessheets .

进入sass目录看一眼： 。。

里边已经预先帮我们生成了三个sass文件， 分别是ie.scss print.scss screen.scss

我们分别打开看一眼

。。。。

可以看到文件里边根本除了一句告诉我们怎么用这个css文件的注释就没有内容了，screen里边这句话我们稍后再讲是用来干嘛的。

这样一来我们就会意其这么设计的初衷了， 只是提供一个框架希望我们尽可能做完美，（鼠标勾选media处）， 为打印写一套样式， 为屏幕，投影显示写一套样式， 使用IE注释，为IE hack一套样式。 只能说真的不适合中国国情啊。 我从业这么多年了， 没有专门为打印样式写过CSS， 倒是为盲人用户做过无障碍化，还记得唯一一次给马化腾写信，还是因为盲人用户把产品无障碍化支持的不好的问题捅到了他那，然后俺充当救火队员才有这样的机会， 呵呵扯远了， 当然了这也跟业务场景有关， ToC的业务真的很少有这样的场景。

我们再进到stylesheets 目录， 可以看到每一个sass文件都有一个对应生成的css文件。

config.rb 文件我们打开看一眼， 可以看到配置方式非常简单， 注释跟bash一样， 使用# 放在行首注释一行， 各项参数配置基本上就是在写等式， 比如说配置我们的 sass路径，css路径 。。。。

至于其他的配置项， 我们会在sass语法讲解过程中穿插介绍， 大家不用着急。

好， 这节课我们就到这里， 谢谢大家。 下一节我们就来聊聊sass的书写。

《第二章：准备工作 - 第三节：Sass语法介绍-基础篇》

大家好， 我是mater。 相信大家通过前两节的学习都已经在本地搭建起了Sass和Compass的环境，这一节我们就来看看如何高效的书写Sass。

之前跟一个朋友聊到Sass，他说Sass的两种书写语法， 就像酒跟愁的关系，算是一对孪生兄弟， 相辅相成。我也不知道他是咋联想到这的，莫不是他当时正处于 “抽刀断水水更流，举杯消愁愁更愁” 的阶段？ 我个人却觉得他俩的关系跟上帝和凯撒的关系相似：正所谓 凯撒的归凯撒,上帝的归上帝。强烈建议在一个项目中保证只使用一种Sass的语法风格，考虑到学习本课程的多是地道的前端同学，在我们整个课程过程中我会始终使用scss的语法， 那是不是偏爱sass语法格式的同学就要失落了呢， 当然不是。 因为我们要讲的第一件事就是如何在两种语法格式间进行转化。

老规矩：

main.scss

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

```
sass-convert style.sass style.scss
```

当然也可以反向转换，一样的指令，只不过两个文件名的位置互调一下，我就不演示了。

哪里不会转哪里， so easy. 除了书写方式的不同， 两者大部分东西都是互通的，在我们讲解基本语法的过程中， 你随时可以转换看看sass如何书写， 学一赠一， 你赚到了。

一般学习语法会是一个比较漫长的过程，但这里我们只讲最核心的，注意， 是只讲最核心的，所以篇幅不会很长。 其他的一是比较少遇到，比较少应用。 二是语法这个东西，即使遇到了，在我们所讲这些知识的基础上，大家通过查询sass的官网也完全可以自己解决问题。 最最重要的， 语法 你就是学的再好， 不实战不应用也是白搭。

首先通过Compass新建一个我们的测试项目 learn-sass-syntax

为了让我们的文件能够实时编译，我们使用之前学到的 compass watch 指令， 监听当前目录下sass文件的变化。

(修改font-family 为 “Braggadocio")

正式开始之前呢，我们首先假定一下我们的项目背景。 我们要构建一个项目的索引介绍页： http://webdemo.myscript.com/#/home 大概就长这个样子。 我们先看一下， 页面分了八个板块，每个版本有一段自我介绍的文字和图片。 为了重点突出我们Sass和Compass的特性， 我们不会从头到尾的构建这个页面， 会大讲特讲sass和compass特性在其中如何应用， 其是怎样让我们书写CSS的过程更加酣畅淋漓的。

接下来我们打开screen.scss文件，来进行书写。

项目新建的文件均是使用utf-8编码的， 这已经是业内惯例了， 因为大部分程序在解析文件的时候， 如果没有明确指定编码， 默认都会使用utf-8. Sass可以使用跟CSS一样的@charset “UTF-8” 指令指定文件编码， 但是我希望大家永远不要用到它， 为什么要增加这不必要的麻烦呢， 是吧。

我们要讲的第一个sass语法特性是： 变量， 我相信来听我们课程的同学一定对这个词不陌生， 在编程过程中， 我们使用变量来存储计算结果或者表示抽象的值概念。

相信大家已经注意到页面的title和主体部分使用了两种不同的字体。 title部分使用了一种毛笔字体， 叫做Braggadocio， 而主体body部分使用了Arial 字体。 这种值我们就可以轻易的把它抽象成变量， 方便我们以后的变更，修改。 随时应对产品经理和设计师的“无理取闹” 注意： 无理取闹加引号。

body： Arial　　Arial,Verdana,Helvetica,sans-serif

一般我们把变量放在行首。 ***

```
$headline-ff: Braggadocio,Arial,Verdana,Helvetica,sans-serif;
$main-sec-ff: Arial,Verdana,Helvetica,sans-serif;

.headline {
  font-family: $headline-ff;
}

.main-sec {
  font-family: $main-sec-ff;
}
```

而对于这种常见变量， 我们喜欢抽离到一个专门的变量文件， 接下来我们了解如何使用Sass的import 引入文件能力。

首先我们新建一个文件，起个名字叫做 variables.scss， 因为这个文件只需要存储变量，所以我们不需要它编译出对应的css文件， 我们给文件的名字前边加一个下划线， 这样就是声明它为一个局部文件。

把变量声明移到这个文件去。 然后如何在screen.scss 文件中把variables.scss包含进来了呢， 非常的简单， 就是使用import指令。

```
@import “variables";
```

这个时候我们也给screen.scss文件起个名字， 我们称之为宿主文件。

但是大家不要以为这个是CSS中的原生的@import指令。 CSS中的@import指令也是用来引入额外的样式文件，但是它有两大弊端： 1. 比如放在代码的最前边， 否则它将会不起作用，2. 我们假设是A引入B， 只有当浏览器把A下载下来， 解析渲染过程中 读到A中@import B的指令的时候才会去下载B，然后浏览器处于一个阻塞的过程， 大大延长了我们的页面渲染时间， 对性能不利。 所以我们通常是不建议使用CSS原生的@import指令的。

我们之前说了， Sass是一个CSS3的语法超集， 所以Sass是一定支持 @import 指令的，但是Sass却重新定义了@import指令的功能。 我们把这个称之为Sass中的 control directives.

不同的地方在于， Sass在编译阶段将 被引入文件合并输出到相应的CSS文件， 并且@import指令可以用在文档的任何地方。 如果被引入文件里边包含的是变量或者后边我们要讲到的mixin，函数， 则意味着在宿主文件中我们可以随意使用这些变量和函数。

默认呢， Sass的import指令相对于我们当前的宿主文件寻路， 但有同学可能会发问了，说screen.scss文件中默认的这句 @import “compass/reset” 按照这种说法， 我们的sass目录下应该有一个compass目录啊， 这个到底是干啥的呢？ 刚才我们提到的只是sass的默认行为， sass提供了一个 load_paths的选项可以用来添加额外的寻路目录， compass通过配置这个选项， 将compass的目录加入到了寻路路径中。 而reset则是compass提供的一个重置模块， 我们都知道HTML标签在浏览器里边有默认的样式， 例如p标签会有一个上下边距。 不同浏览器的默认样式之间也会有差别，例如ul默认带有缩进的样式，在IE下，它的缩进是通过margin实现的，而Firefox下，它的缩进是由padding实现的。在切换页面的时候，浏览器的默认样式往往会给我们带来麻烦，影响开发效率。所以解决的方法就是一开始就将浏览器的默认样式全部去掉，更准确说就是通过重新定义标签样式“覆盖”浏览器的CSS默认属性。而这就是CSS reset 干的事情。

我们再稍微延展一下： CSS reset 干的其实比较激进， 不管默认样式属性有用没用， 上来就全部重置归0。 这有点像吃大锅饭，看起来是众生平等了， 实则呢导致效率的降低，资源的浪费。 当我们需要默认样式的时候还得再贱兮兮的加回去。

所以就有了比较温和的改良一派， 我们称之为 Normalize, 其更加注重通用的方案， 重置掉该重置的样式，保留有用的 user agent 样式，同时进行一些 bug 的修复，统一跨浏览器的默认样式差异，比如说统一不同浏览器间的p标签的上下边距大小， 而不是把p标签的上下边距大小重置为0. 这点是 reset 所缺乏的。

后边介绍到Compass的时候， 我们再跟大家介绍如何使用compass的插件机制，引入normalize替换掉reset。

好， 回到我们的@import指令：

那有同学可能就要抬杠了， 说我就是想用css的@import， 你这样给我覆盖了， 我还怎么用啊。 你先别急： 一切都有既定规则：

1. 当@import 后边跟的文件名是以.css结尾的时候
2. 当@import后边跟的是 http:// 开头的字符串的时候
3. 当@import后边跟的是一个url() 函数的时候。 url("bluish.css")
4. 当@import后边带有media queries的时候。 @import “variables” projection, tv;

只要满足以上的任何一种， sass会认为你是想使用css原生的@import，而不再做任何的狗拿耗子-多管闲事的无用功。

另外大家应该也注意到了， 我们并没有写完整 _variables.scss, 这也是基于两个sass的既定股则：

1. 当没有文件后缀名的时候， sass会尝试为这个文件名添加.scss或者.sass的后缀， 就看寻路路径中有哪一个。
2. 同一目录下，局部文件和非局部文件不能重名， 也就是说我们已经有一个 _variables.scss 文件了， 就不能再当前目录再建一个 variables.scss 文件了。 基于这个前提， 在引入局部文件的时候， 我们不需要把前边的 _ 也带上。

现在我们有两个 import指令， 完全可以把它们合并为一个。

@import "variables", “compass/reset";

无论是我跟人合作还是我自己带新人， 我都会反复强调一点： 好的代码习惯应该是注释占整个篇幅的三分之二。这样维护人员对项目进行后期的维护的时候，才能很好的进行维护或者说是升级。

所以我们开始给我们的sass文件添加注释。

还是那句sass是css3语法的超集， 所以css的注释语法， sass一定是支持的， 正如screen.scss 文件默认生成的文件注释一样。 支持 /* */ 这种块注释语法。 同时sass还支持 // 双斜线的这种单行注释语法。 比如说： ***

```
// 主标题样式
.headline {
  font-family: $headline-ff;
}

/* 页面主体内容样式 */
.main-sec {
  font-family: $main-sec-ff;
}
```

那这两种注释语法有什么不一样的地方呢？ 我们看一眼生成的样式文件就知道了。 我们发现以 双斜线方式注释的内容在输出css文件的时候被抹掉了， 而以 /* */ 方式注释的内容在输出css文件的时候则被保留了。

同时我们发现还有自动生成的注释信息， 这个注释信息可以大大方便我们的css样式调试， 稍后我们再介绍， 同时会介绍如何关掉这个调试信息。

我个人呢有个习惯， 就是我会在sass的宿主文件， 通过一个块注释的方式， 说明每个被引入文件的作用。

```
/**
 * CONTENTS
 *
 * SETTINGS
 * variables.....................变量集中存储文件
 *
```

```
 * TOOLS
 *
 * COMPONENTS
 * reset.....compass内置浏览器重置样式
 *
 * BUSINESS
 *
 * BASE
 * screen.scss.....针对当前站点主页的样式修饰
 */
```

一般呢会分成几个大块， 设置类， 工具类， 组件类， 业务逻辑类， 和基础类， 大家可以根据自己的业务需要呢， 来决定如何分类。

（在页面上指引说明）接下来我们假设我们每一个板块上的主介绍文字， 它的css class name也是 headline, 这时候， 我们就需要使用

```
.main-sec .headline{
  font-family: $main-sec-ff;
}
```

来进行加权修饰， 这样的加权修饰多了以后就会变得不那么清晰。 sass提供了一种很方便的嵌套语法。

我们可以把 .headline 移到 .main-sec 里边。好，看一眼生成的css文件。 虽然最后输出的还是 .main-sec .headline的形式， 在sass中看上去结构却变得更加清晰易维护了。

输出规则非常的简单， 在parents class name 后边跟一个空格， 然后再跟上 child class name。 但有些情况下，就会出问题， 比如说我们想修饰一个超链接的hover样式，

```
a {
  :hover {
```

```
    color: blue;
  }
}
```

看一眼生成的css文件， 我们可以看到 a跟:hover之间有一个空格， 而我们都知道 a :hover 就相当于 a *:hover 我们修饰的不再是a标签超链接的hover效果，而是a里边所有元素的hover效果， 这是不符合我们预期的。

Sass通过引入一个父选择器 & 来解决这个问题。

```
a {
  &:hover {
    color: blue;
  }
}
```

& 用来显式的将父类引入，并按照你书写的位置， 格式， 最终输出， 我们再看一眼生成的css， 就已经符合预期了。

（把 a 删掉）

（在页面上指示）

我们还注意到在每个版块的介绍区域， title 和 detail的字体大小是不一样的， 所以我们还需要加一个字体大小的限制。

```
.main-sec {
  font-family: $main-sec-ff;

  .headline{
    font-family: $main-sec-ff;
    font-size: 16px;
  }
```

```
  .details {
    font-size: 12px;
  }
}
```

我们都知道font-fami ly 和 font-size 可以通过font缩写的方式进行归纳， 其实sass除了提供类之间的嵌套， 还提供了属性的嵌套能力。 我们可以把这段改写成：

```
.headline{
    font: {
      family: $main-sec-ff;
      size: 16px;
    }
  }
```

我们看一下生成的样式文件。 完全符合预期， 只是注意 属性后边的冒号不要漏掉。

好， 这一节我们就介绍到这里， 下一节我们来看看变量和sass directives的高级特性。

《第二章：准备工作 - 第三节：Sass语法介绍-进阶篇》

大家好， 我是mater。 上一节我们粗略介绍了Sas的语法以及应该在何种场景下使用相关的特性， 这一节我们来进一步看看sass的高级语法及其应用场景。

上一节我们讲了变量， 便随着变量出现的就一定是变量的操作。 变量操作分为两种， 一种是直接操作变量，也就是我们常说的变量表达式。 另一种是通过函数，但是Sass呢又很奇怪，它根据函数功能细分领域的不同， 把函数分为了两种不同的类型， 第一种是跟代码块无关的函数，多是它自己的内置函数，在Sass术语表中我们就称之为functions，第二种是可重用的代码块， 这有点像C语言的宏(macro/'mækro/)，但就是可重用的代码块，Sass又细分了两种概念， 使用时以复制拷贝的方式存在的，在Sass术语表中我们称之为mixin，通过@include的方式来调用， 还有一种在使用时以组合声明的方式存在的，我们通过@extend的方式来调用。下面我们就来详细聊一聊这几种方式的异同。

Sass的变量表达式支持基本的加减乘除以及模除操作，数值类型支持>= <= > < 比较，所有变量类型支持 == != 判断，支持（）修改运算符优先级， 不过应用场景真的是太窄了， 我唯一一次用到是在写一个Compass合图过程中自动适配手机retina屏的插件的时候，如果只是简单使用Sass，用到的概率真的极少。

我在学习Sass的时候看到老外的示例程序都是：

```
p {
  height: (500px / 2);
}
```

从中我们可以看出两点， 1： 老外的数学令人捉急。 2： sass中的数值计算是可以带单位的， 也就意味着你不能混用两种不同的单位。

同样很少使用的还有sass的functions， 我们知道css2中的色彩模式只有RGB和十六进制的模式，但是这两种模式都无法主观感受，

当我们第一眼看到一种颜色的时候，脑海中想的一定是： “这是什么颜色？深浅如何？明暗如何”而不是，多少红加多少绿加多少蓝能够混合出这个颜色。

（http://cdc.tencent.com/?p=3760 使用这个站点页面演示）所以CSS3才新增了HSL颜色模式，也就是色调hue(/hju/), 饱和度saturation(/'sætʃə'reʃən/), 以及亮度(lightness), 色调是一个360度的颜色盘， 饱和度和亮度分别用百分比来表示。

我这里再强调一次， sass的语法是CSS3语法的超级，所以sass一定是支持hsl语法的。

通过色盘我们可以知道 300度的时候，是洋红，240度的时候是蓝色，那如果我们想要紫色的话， 取个中间值即可，写成：

```
p {
  height: (500px / 2);
  color: hsl(270, 100%, 50%);
}
```

但如果是普通的CSS代码就存在着一个问题， IE6, 7, 8是不支持hsl属性的，我们的这段CSS样式将在IE6，7，8下失效， 我们看一眼生成的代码， 我们发现最后输出的是一个十六进制表示的颜色值。 Sass通过内部定义hsl这样一个函数， 替换掉CSS3默认的hsl功能，完美帮我们解决了浏览器兼容的问题。

当然这只是Sass无数函数中的一个，在Sass的官网有一个Functions页面， 汇总了Sass中所有的functions，很多，但是用到的几率却很低， 大家可以把这个地址记下来， 有时间呢， 大致浏览一下， 有个大概的印象， 等你真的需要为Sass贡献插件的时候，再回过头来看也不迟。sass也允许我们通过@function指令来声明函数，但一般我们极少使用，你知道有这么个能力就好。

Sass中的Mixin通过@mixin指令来定义。 后边跟mixin的名字和参数列表。 然后是这个mixin的主体内容。

我们重新回顾一下我们要构建的站点主页， 8个介绍版块的结构其实非常的相似， 自适应效果的实现也非常的容易，将我们的main-sec 留出一个左右的margin， 然后每一个版块div占宽50%， float：left即可。 那我们就可以把这一共同的结构体抽离成一个mixin。 一般我们要么把我们的mixin放在页面的顶部， import之后，要么单独的抽离出一个文件。这里呢，我们新建一个mixin存放文件， 我们就叫它mixin的scss， 老规矩， 局部文件以下划线开头。 在注释中补全， 以及引入。

我们接下来来构建一下这个mixin， 给他起个通用点的名字， 叫做 col-6， 为什么要叫6，相信用过bootstrap的同学就一定不陌生了，在bootstrap的格式系统中， 整个容器区域被

分为了12等分， 所以6刚好就是一半， 这个呢也实在是因为bootstrap太火了。 设定width占50%， float：left；

```
@mixin col-6 {
  width: 50%;
  float: left;
}
```

这样一个最简单的mixin就写好了， 当然， 随着我们后边讲到的知识越来越多， 我们还会不停的扩充这个mixin， 觉得简单的同学先不用着急。

调用也非常的简单， 通过@include指令来实现。我们给每一个sec起名 webdemo-sec, 写出来就是 。。。。

好，我们看一眼生成的代码， 我们看到在mixin中定义的样式被引入到了我们当前的CSS规则中。

那一单使用了mixin， 规则中还能不能包含其他的规则呢， 答案是肯定的， 我们看在我们hover每个版块的时候，背景色会变成浅灰， 我们把这个样式修饰加入进去。

使用我们上一节学过的父选择器， 。。。 经验丰富了之后， 有些常用色值你是能记住的， 比如这种背景灰一般是 #f5f5f5。

我们在看一眼生成的代码， 发现hover样式修饰也已经顺利生成了。 同时一个规则中还可以@include多个mixin， 这里我就不演示了。

这时候可能有同学要问了，说我为什么还要额外写个类名呢， 我直接用col-6不就行了吗？ 就是说直接给我们的每一个sec加一个col-6的类， 这当然是没问题的， 但我们已经注意到我们的mixin在不调用的情况下并不会生成任何css内容， 调用的时候也只是把mixin内的内容输出到调用位置， 那应该怎么做呢？我们只需要对col-6 mixin微调即可， 。。。。。。

```
@mixin col-6 {
  .col-6 {
    width: 50%;
    float: left;
  }
}
```

然后我们把对col-6 mixin的调用提到最外层， 注释掉我们的webdemo-sec规则， 好， 我们看一眼生成的代码。 .col-6 类的修饰成功输出到了我们的css文件中。

虽然轻而易举的我们就做到了， 我极其不建议大家这么做。我们先把文件恢复一下。 为什么？ 因为CSS书写的指导规则中其实就有一条就是： 我们的类名一定要有语义化的作用， 而不是视觉化的表现。 举个例子来说， 错误提示消息， 为了强调其视觉效果我们往往用红色字体， 一些传统的做法为了复用往往样式的正交性这种做法，简单来说就是搞个

```
.red {
    color: red;
}
```

然后给这段错误消息加一个red的类修饰。 正确的做法我们应该是给它一个 error-text的类名修饰， 然后在css中修饰error-text这个类名的字体颜色为红色。我这么说相信大家都能听懂吧。 原因呢也没啥好争的， 1是我们开发的是一个站点， 不过多么炫丽， 也不管你加多少特效， 我们的最终目的是提供有意义的信息，而不是单纯的实现UI， 当然了纯体验类的游戏除外。2是这样做可以保证网站有持续的可维护性。3.别忘了IE6是不支持多个class的， 这在很大程度上限制了样式正交性的做法。虽然这条原因现在看来是这么的苍白无力。

sass通过mixin很好的解决了样式复用和语义化的问题， 所以说工具一定是为解决问题而生的。

我们如何让我们的mixin变得更加可配置一些呢？ 把mixin跟函数对应起来，答案就不言而喻了， 那就是参数。 参数的定义方式跟函数很像， 一对左右大括号， 里边是参数的名字。

我们把宽度变成一个可配置参数。

对参数的引用就相当于引用一个变量。

```
@mixin col($width) {
```

```
  width: $width;
  float: left;
}
```

然后在引用的时候传入参数。

```
.webdemo-sec {
  @include col(50%);

  &:hover {
    background-color: #f5f5f5;
  }
}
```

好， 我们看一眼生成的代码， 效果是一样的。 只不过变得更加灵活， 我们可以传入一个25% 来控制一行显示四个板块。

但你说我就是懒， 我即想要灵活性又不想传50%， 我想让它默认50%， 需要的调整的时候再传参可不可以呢？ mixin呢支持默认参数值的使用。

```
@mixin col($width: 50%) {
  width: $width;
  float: left;
}
```

我们在调用的时候不传参数看看编译出什么效果。 依然是50%的width。

现在让我们回到刚才聊的样式正交性和语义化的问题。还拿error来举例，如果我们要把我们的错误消息分级， 除了error，我们还想强调更严重的error，我们给这种错误起名serious-error, 那serious-error 既要有error的样式， 同时还得有自己特有的样式。

我们新建一个html文件，来设计一下我们的标签。

html:5

然后我们来构建一下我们的错误提示消息。

```
<div class="error">
  Oh no! 你今天还没有上慕课网学习。
</div>
<div class="serious-error">
  Oh no! 你已经一周没有上慕课网学习了。
</div>
```

设计一下我们的CSS，

```
.error {
  color: #f00;
}

.serious-error {
  border: 1px #f00;
}
```

.serous-error 我们除了想让它有红色的字体颜色， 还想让它有一个红色的边框。 这时候我们传统的做法要么是在html标签里边添加一个error的class， 要么是再为serious-error的类加上这样一条规则， 前一种做法意味着无论何时我们在使用serious-error 的时候都必须记得同步使用error， 这样的事情干多了一定会出现维护上的问题， 从而导致诡异bug的出现， 并且这其实也不符合标签的语义化规则。 第二种做法则让css出现了冗余， 那天我们要修改error信息的字体颜色了， 是不是既要修改error类下的规则， 又要修改serious-error类下的规则呢?

Sass的extend指令可以帮我们完成这件事情，明确指定一个选择器去继承另一个选择器的样式， 用起来非常的简单， 。。。。。

```
.serious-error {
  @extend .error;
  border: 1px #f00;
}
```

好， 我们看一眼生成的代码， 我们发现最后输出的代码， .error 和 .serious-error 共用了color: red 这一样式修饰， serious-error 还多了一个border的样式修饰。这样我们就只需要在我们的html标签中写一个 serious-error的类即可。

@extend的工作原理也非常的简单， 就是把 .serious-error这个继承者的选择器， 插入到被继承者 .error的选择器出现的地方。 沿着这个思路， 如果我们有一个

```
.error.instrusion {
  background-image: url("/image/hacked.png");
}
```

入侵检测的错误提示消息样式修饰， 这个时候.serious-error 继承 .error 会发生什么呢?

我们看一眼生成的代码:

我们发现 serious-error也成功被插入到了 .error.instrusion的位置， 即也继承了入侵检测的样式修饰。

当然 extend 也支持多继承， 你可以extend多个选择器，跟我们mixin的多个include一样，同样也不演示， 留待大家自己实验。 同时还希望大家自己实验一下如果是连续继承会是一个什么情况， 比如说B继承了A， 同时C又继承了B。

最后我们再聊两个extend的知识点。

1. extend不可以继承一个选择器序列， 比如说

```
.A .B {
  color: #000;
}

.C {
  @extend .A .B;
}
```

我们看一眼生成文件， sass把错误消息输出在了生成文件的头部， 告诉我们不能继承嵌套选择器。

2. 将来如果我们要写一个Sass的插件， 可能会提供很多供别人继承的选择器， 但是我们并不希望这些选择器被输出到CSS文件中， 因为别人也可能用不着继承这个选择器， 会有冗余样式生成。 这个时候可以选择使用%来构建一个仅用来继承的选择器。

先看一眼生成的文件， .error的样式也被输出到了CSS中，

我们修改一下 .error 改成 %error, 继承的时候使用 %error， 再看一眼生成的文件呢？ 我们发现只有serious-error的样式修饰， 而没有error的样式修饰。

好， 函数我们就说这么多， 下一节我们来聊聊Sass中剩余的指令以及逻辑控制，数组，map的处理。

《第二章：准备工作 - 第四节：Sass语法介绍-高级篇》

大家好， 我是mater。通过前两节的学习，我们掌握了Sass的变量，函数以及初级指令等， 这一节我们则向Sass的最高峰发起进攻， 最后扫尾一下我们常用到的Sass高级特性。

移动互联网的浪潮滚滚而来， 连猪站在风口上都已经飞上天了。作为搞技术的我们，一定要跟上时代的步伐， 所以就有了响应式布局设计的概念， 目的是为移动设备提供更好的体验， 并且整合从桌面到手机的各种屏幕尺寸和分辨率。

体现在我们的站点上就是当我们把窗口大小缩小到一定程度的时候， 整个页面就变成了一个介绍版本占一行的布局了。 这实现起来也不是什么高精尖的技术了， 主要用到的就是css的media queries。

sass中的@media跟CSS不一样的地方就在于sass中的media query可以内嵌在css规则中， 最后在生成CSS的时候，media query才会被提到样式的最高层级。 这样做的一大好处就是避免了我们重复书写选择器或者打乱样式表的流程。

我们来重构一下我们的 col-6 mixin, 还是按照boostrap的惯例， 屏幕尺寸分了这么几个临界值： （使用bootstrap的页面演示）

。。。。

而我们这里只需要适配 Small devices Tablets 两边的情况即可。 所以我们加一个min-width:768px 的media查询。 同时按照惯例， 修改一下col 的名字为 col-sm-

```
@mixin col-6 {
  @media (min-width: 768px) {
    width: 50%;
    float: left;
  }
}
```

```
@mixin col($width: 50%) {
  @media (min-width: 768px) {
    width: $width;
    float: left;
  }
}
```

这句的意思就是 当屏幕尺寸大于768px的时候， col-6 占宽50%， 小于768px的时候， 因为我们没有加任何修饰， div的默认行为就是占宽100%。 我们看一眼最终生成的css。

可以看到media query被提到了样式表的最外层。

虽然Sass提供了非常棒的嵌套能力， 但CSS最佳实践告诉我们，浏览器在解析一个css选择器的时候是按照从右往左的顺序解析的， 举例来说

```
.main-sec {
  font-family: $main-sec-ff;

  .headline {
    font: {
      family: $main-sec-ff;
      size: 16px;
    }
  }

  .detail {
    font-size: 12px;
  }
}
```

浏览器会首先找到页面中所有的类名为headline的元素， 然后再去一层层遍历headline元素的父级元素中有没有类名为main-sec的的，直到找到或者遍历到html一级，这样往往导致样式渲染的低效。 另外通过嵌套， 还有一些副作用， 比如说增加了样式修饰的权重，制造了这种样式位置的依赖。 那最佳实践的做法应该是怎么样的呢?

我们会给页面中响应的元素起名 main-sec-headline 和 main-sec-detail, 然后加以修饰。

```
.main-sec-headline {
  font: {
      family: $main-sec-ff;
      size: 16px;
    }
}

.main-sec-detail {
  font-size: 12px;
}
```

那这样写多了以后又不如嵌套清晰易维护， 那有没有啥好的办法呢？我们可以在嵌套的时候使用sass的at-root指令， 明确指明被嵌套内容输出到样式表的顶层。

```
@at-root {
   .main-sec-headline {
      font: {
        family: $main-sec-ff;
        size: 16px;
      }
    }
```

```
  .main-sec-detail {
    font-size: 12px;
  }
}
```

好， 我们看一眼生成的代码， 发现main-sec-headline, main-sec-detail 虽然是嵌套书写的， 但是最后生成的时候却不包含嵌套。

假设我们要把我们写的col这个mixin贡献出去给别人用， 那就很有必要加上参数校验了对不对， 不符合我们要求的参数则应该给一个错误消息。接下来我们再扩展一下我们的col mixin。

为了响应式布局， 我们只支持传入百分比的值。所以字符串的值肯定不行， 首先校验这个。 Sass中有@if的操作符用来进行条件判断。

我们使用sass内置的type-of函数来校验width是否为数值类型。

如果不是数值类型， 使用Sass的 @error directives 来输出一条错误消息

$width 必须是一个数值类型, 你输入的width是：#{$width}.

在选择器，属性名或者字符串中我们如果想要引用sass变量， 需要使用 sass的interpolation/ɪn,tɚ˞pə'leʃən/ 也就是#{} 里边跟变量的名字。

好, 确认是一个数值了， 我们使用Sass的内置函数unitless()来判断一下当前的这个数值后边是否跟有单位，像30px， 50%这种在sass里边都是数值类型， 30px的单位值是px，50%的单位值是%。只有纯数字才是没有单位的。 所以unitless(50%) 返回的值是false，因为50% 是带单位的。 所以我们要想命中百分值就得对这个判断条件取反。 sass中boolean操作符使用英文 and or 以及not来表示与 或， 非。 如果是没有带单位， 那我们也利用@else分支处理一下。

这里写的有点绕哈， 我主要是想演示 not 这个非操作符。

没有单位，我们利用warn directives输出一条警告信息， 然后把$width值利用sass的内置函数 percentage 转换为百分比。

else if的语法也很简单， 在else 后边再跟一个if就可以了。

如果已经确定有单位了， 那我们就利用sass的内置函数unit来判断一下单位是否为%，

如果不是再输出一条error信息。

好，这样基本上就写完了， 我们来简单测试一下 。。。

abcd 30px 30 50%

写的有点绕， 主要还是为了演示sass相关的特性， 大家可以根据自己的实验加以优化。

剩下的还有 @each 用来遍历map， @for 和 @while 用来循环，这几个我是真的没有在实战中用到过， 所以这里就不耽误大家的时间了， 感兴趣的同学可以到sass的documentation页面上看一眼， 非常简单， 一看便知。

最后我们来说一说sass的四种输出格式。

我们打开config.rb 文件， 可以看到有一项 output_style, 这项就用来控制最后CSS的输出样式。 我们一个一个来看一下长什么样子。

compass默认的输出格式就是expanded， 所以expanded就是我们现在看到的这个样子，样式一个个都是展开的，跟我们手动书写CSS的习惯一样。

nested呢 可以很好地反映CSS样式修饰的html的结构， 因为它是根据我们的嵌套在输出的时候对应缩进的，嵌套的越深， 缩进的就越多。 我们可以看到。

compact呢， 将所有的属性汇总到一行。 这种模式下大家关注更多的是选择器之间的关系， 而不是选择器内的属性。

compressed, 顾名思义， 将样式表压缩以占用最少的空间， 这是一种对于我们来说不可读的输出格式。上线前的css一般我们会采用compressed输出格式。

好， Sass的特性我们就说完了， 引用后会无期中的一句经典台词： “经历了一万多公里的长路，我们不会辜负每一份信任和付出。” 当我们在下节讲到Compass是如何有效解决我们的各种痛点问题的， 你一定会感谢现在掌握的sass的基本功。 谢谢大家。

补补补补补补补《第二章：Sass和Compass必备技能之Sass课程总结》

我觉得有些同学从上一节到这里，一下子发现是课程总结， 可能要在心底里边骂人了。 “经历了一万多公里的长路” 最后发现到头了还没看到Compass。 您听我细细道来，为了照顾已经有Sass基础的同学呢， 我将Compass的相关知识另起了一个课程《Sass和Compass必备技能之Compass》，是无论如何不能给已经摩拳擦掌，跃跃欲试按耐不住要学习Compass的同学当头泼一盆冷水的。 再来一针鸡血哈： CSS至今已经发展了将近20个年头，暮气沉沉，廉颇老矣？且看Compass是如何让CSS重焕青春的吧。

接下来引用《教父》中的一句经典台词： “我准备向你们提出一个你们不可能拒绝的条件”

如果我没有让你失望， 敬请移步： Sass和Compass必备技能之Compass。即使失望了，也一定要了解一下Compass， 真的会惊艳到你。

谢谢大家。

补补补补补补补《第三章：Sass和Compass必备技能之Compass课程介绍》

大家好, 我是mater, 很高兴能跟大家一起分享一些Sass, Compass相关的知识和经验。

CSS至今已经发展了将近20个年头，暮气沉沉，廉颇老矣？通过本课程我们将彻底了解Compass是如何让CSS重焕青春的。

Compass课程介绍这门课有一个前置课程叫做： 《Sass和Compass必备技能之Sass课程介绍》， 如果没有Sass基础的同学呢， 建议首先学习前置课程。 如果你本身已经具备了Sass的相关技能， 则可以直接上手本课程， 但我仍然会建议你过一遍前置课程， 一定会带给你不一样的视角和思考角度， 这样在后边的学习过程中， 听到我穿插说， 前边我们已经介绍过某某某， 正如之前所说XXX， 你也不会感觉有所疏漏， 最终有助于形成一个系统化的认识。

“人们走进喧闹中去，是为了忘却孤寂”， 接下来让我们走进Compass的世界， 忘却CSS的痛。

《第三章：实战应用 - 第一节：Compass核心模块概述&Reset模块》

大家好， 我是mater。在前边呢，我曾经把Compass与Sass比作锤子和钉子，显得略俗哈。刚好最近有部电视剧特别的火，据说还被习大大推荐了，叫《平凡的世界》，里边有一句经典台词是：“当我的巴特农神庙建立起来的时候，我从这遥远的地方也能感受到他的辉煌。”套用过来，在我看来，Compass就是Sass的巴特农神庙， 有了Compass，Sass才迎来了它真正的辉煌。今天开始， 我们就来深入剖析Compass了。

Compass在Sass的基础上构建了一整套强大的工具，其主要分了这几大模块（PPT）

reset 模块是我们最早接触的， 正如我们之前所述， 其是一个浏览器样式重置模块， 用来减少不同浏览器间的差异性。 当时还提到我个人更推崇normalize的做法， 如果你已经不记得当时我们的对比分析了， 可能就需要回顾一下之前的课程了， 因为我们等下就会讲如何利用compass的插件机制，引入normalize替换掉reset。

layout 模块用来提供页面布局的控制能力。比如说将一个容器内的子元素横向拉伸占满，纵向拉伸占满 或者 整个拉伸占满 这种能力。 使用 @import “compass/layout” 来引入。

这两个模块之所以需要单独拿出来， 是因为Compass的核心模块只有这两个模块是需要明确指定引入的。 怎么讲：

我们只需要 @import “compass” 就默认包含了其他的五大模块， 却不包含reset和layout模块。

剩下的4个功能模块分别是：

CSS3 模块， 主要用来提供跨浏览器的CSS3能力。

Helpers模块， 内含一系列的函数， 跟Sass的函数列表很像， 比较少用到。 但功能确实丰富强大。

Typography模块， 主要用来修饰我们的文本样式，垂直韵律。

Utilities模块， 你可以认为没办法放到其他模块的内容， 都可以放到这个模块。 但还是见文知意： 辅助工具类的模块， 跟Helpers不同的是， Helpers内是函数， Utilities内多是mixin。

在这6个功能模块之外还有一个模块：

Browser Support 模块。

用来配置compass默认支持哪些浏览器， 对于特定浏览器默认支持到哪个版本。可以认为Browser Support 模块中的配置一旦修改， 将影响其余6个模块的输出。因为其要针对不同的浏览器做不同的适配。

接下来我们一一讲过， reset我们已经见识过了， 来看看怎么用normalize 来替换掉它，原理呢都是一样的， 都是引入一段提前写好的CSS样式，只是reset是compass内置的，而normalize需要我们额外多做一点。

有两种做法， 一是我们到normalize.css的官网： http://necolas.github.io/normalize.css/ 直接把这个文件download下来放到我们的项目目录， 然后通过@import 把这个文件引入进来。 我们可以看到官网上除了直接下载还给出了前端的包管理工具npm, component, bower 的下载方式，大家可自行选择。

另外一种方式，就是使用compass的normalize插件， 最早我们介绍compass的时候就已经说过， compass的周边生态非常的丰富， 有很多扩展插件， 所以normalize肯定早就有人做了，插件的名字就叫做compass-normalize.

本地安装normalize插件的过程， 其实就是安装一个标准的gem程序包的过程，

我们使用 gem install compass-normalize 来进行安装。

然后要做的就是引入插件， 只需要修改我们的config.rb 文件.

第一行就告诉我们应该怎么引入compass插件了， 使用require指令。 我们这里把compass-normalize引进来，

require ‘compass-normalize'

使用之前呢， 扩展一下， 跟大家说一下这里 compass/import-once/activate ， 这个插件是干嘛的呢?

Sass提供的@import指令，它的工作方式其实有点粗暴， 同一个文件， 比如说 reset

我不小心在在一个文件或者多个有引入关系的文件里边引入了多次， 这里为了掩饰方便， 我们就在screen.scss 的最后再引入一次，Sass在解析的时候， 只要遇到一次@import指令， 它不管你之前是不是已经引入过了， 都再引入一遍。 在大多数时候这其实会带来不必要的CSS样式冗余和性能问题， 而import-once插件就是用来解决这个问题的， 在我们的config.rb文件中直接require import-once/active 将其引入并启用即可， 这样即使我们不小心多次@import了同一个文件， compass也只会插入一次被引入文件。

但使用了import-once，万一真的遇到一个文件必须被引入两次的情况怎么办呢？我们可以通过在被引入文件名的后边加一个！的方式告知Compass这里需要重复引入。

@import "something";

@import "something!"; // this will be imported again.

我们来试一下。(在页面尾部添加 @import “compass/reset” 编译查看 加叹号， 编译查看)

讲到！了就再扩展一下， 我们之前讲sass的输出格式，如果把config.rb的output_style改为compressed，最后输出的css则是经过压缩的，注释是一定会被去掉的。 但是我们看到网上很多开源项目要求如果你的项目中用到了它， 你是需要在注释中声明的， 或者说你自己想要在注释中声明你的项目权益。 怎么做才能不被压缩掉呢？ 有一个约定俗成的规矩， 那就是： 在注释的最前边加一个叹号。

我们来试一下（ 先执行编译看效果， 然后加叹号， 执行编译看效果 ）, 我们可以看到我们的注释内容被保留了下来， 直接把Copyright 这些信息写在注释里就好。

好， 插件引进来了， 接下来就是在我们的screen.scss文件中加以使用。 使用方法跟reset一样， 直接使用@import “normalize” 引入即可， 我们把reset替换掉看一下效果。

balabala

我们可以看到 normalize.css 文件已经被引入进来了， 相信细心的同学已经注意到了，normalize 文件使用了我们刚才提到的方法在注释中声明自己的copyright信息，使用MIT开源协议。

这两种normalize引入的方式比较起来呢， 我一定是推荐大家使用第二种的， 为什么呢?

范范地整体引入一个reset，或者normalize并不符合我们的极客精神，因为随着业务需求的不同， 我们可能并不需要reset或者normalize 浏览器间所有的差异， 可能只需要统一links的表现即可， 也可能只需要统一tables的表现即可。而这个时候， 插件的这种方式给我们提供了更大的灵活性。

normalize 本身包含八个模块，

base: 用来统一html和body标签的字体， 文字大小调整，边距等

html5：统一html5中新增的元素， 比如说article, aside, main, nav, section 等的展现形式。

links： 统一a标签的样式修饰，去掉hover和active时候的边线

typography: 统一 b, strong, h1, sub, sup 等段落文本的样式修饰

embeds： 统一 img， svg 等标签的样式修饰， 比如说统一img标签的border为0宽

gruops： 统一 figure, h4, pre, code 等标签的样式修饰

forms: 统一form相关的button, input, textarea等标签的样式修饰， 比如说margin统一为0.

tables: 统一table相关的table, td, th 等标签的样式，比如说消除padding。

引入也非常的简单， 通过 子路径的方式引入，

@import "normalize/base";

/* 分割线2---------- */

@import "normalize/html5";

/* 分割线3---------- */

@import “normalize/links";

但是在任何一个这样的引入使用之前， 需要先引入一下 @import “normalize-version";

我个人认为这是一个败笔， 不知道normalize compass插件的作者为啥这么设计。既然设计成了采用子路径的方式引入子模块， 干嘛非得要求用之前先 @import “normalize-version” 一下呢？ 把这个整合进每一个子模块不就ok了吗， 直接只写一个子模块的引入， 多方便省事啊。或者你采用Compass reset的设计思路吗。我们来看一下reset的设计思路。

reset把对样式的重置封装成了一个又一个的mixin， 我们通过@import “compass/reset/utilities” 就能引入这些mixin的集合。

而 @import “compass/reset” 干的事情呢， 其实就是引入 compass/reset/utilities 然后调用其中的 global-reset mixin 。

我们可以试验一下， 看看二者生成的代码是否一模一样。

balabala

补充一个额外的知识点， 如果我们调用的mixin，无需传递参数， 我们可以省略 mixin名字后边的这个括号。

balabala 省略看效果。

reset一共包含12个核心mixin， 其中有几个是跟normalize的模块划分能对的上的。 这个大家不用记笔记， （http://compass-style.org/reference/compass/reset/utilities/#mixin-reset-font） 因为这些模块在Compass Reset Utilities 的介绍页面有着非常清楚的描述。

reset-body 对应 normalize的 base

reset-html5 对应 normalize 的 html5

reset-list-style 对应 normalize 的 links

reset-table reset-table-cell 对应normalize的 tables

等等我就不一一细说了， 大家可以点击 code reference 页面的view source按钮， 直接看这个mixin的实现代码， 比如说

我们随便点开一个

reset-box-model 这个mixin就是用于选择器内的， 用于将当前选择器的 margin, padding, border 全部设为0， 我们来测试一下

```
.test-reset-box-model {
    …… balabala
}
```

说一下 nested-reset, 用于只重置我们页面上的某个选择器下的所有元素。 怎么讲， 比如说我们只想充值 标签 .reset-sec 下的标签样式， 那我们就可以用

```
.reset-sec {
  …… balabala
}
```

再说一下 reset-display , 刚我们吐槽了normalize，现在来吐槽一下reset，真心不知道为啥会有这么二的设定。 我们都知道span元素默认display的值是inline，我们为了一行只显示一个span元素， 我们用类 text-line-block 修饰span元素， 然后设了一个 display: block .

```
.text-line-block {
  display: block;
}
```

而reset-display 的作用就是遍历所有默认display:inline 和 默认display:block 的标签， 然后为每一个标签 组合 .text-line-block这一选择器， 然后按照标签原本的display 值再输出一遍， 这样就覆盖即reset了我们对 span .text-line-block 这一元素的display值的修改。

我们直接看代码就明白了。

@include reset-display(‘.text-line-block’);

balaba

大家看明白了吗?

还可以在后边跟一个 important 修饰， 强行覆盖样式， 默认不传这个值为false。

sass中的boolean值就用 true 和 false。 我们看一眼生成的结果。

好吧， 我只能用电影《灰姑娘》里边的一句台词来形容这种做法了：“正常并不等于正确”。 真心没有想到这个东东的使用场景。大家可以帮我一起想想， 你想到了告诉我一声。可以直接在课程下边回复或者到微博上@我， 我的微博账号也是materliu。

好， 关于Compass几大核心模块和reset的模块就介绍这么多， 下一节我们来聊一下Layout模块。 谢谢大家。

《第三章：实战应用 - 第二节：Layout模块》

大家好， 我是mater。如果说reset模块是Compass用起来最简单的模块， 那么Layout模块则是Compass中使用率最低的一个模块。

layout 模块用来提供页面布局的控制能力。 需要我们使用 @import “compass/layout” 来显式引入。

layout模块内部又分了3大模块，分别是

Grid Background 模块 @import “compass/layout/grid-background"

Sticky Footer 模块 @import “compass/layout/sticky-footer"

和 Stretching 模块 @import “compass/layout/stretching"

我们先说Stretching， 用来提供将一个子元素拉伸填满整个父容器的能力， 其做法其实也非常的简单， 我们看一眼就知道了。

```
.stretch-full {
  @include stretch();
}
```

对相应元素绝对定位， 然后设定其top，right，bottom，left的值，默认为0， 也就是完全填充， 占满整个父元素。

stretch后边允许跟参数， 最长4个， 分别是 top， right， bottom， left的值， 参数顺序跟我们写简写margin， padding属性时的顺序一样， 上右下左。我们试一下。

```
.stretch-full {
  @include stretch(5px, 5px, 5px, 5px); 传参的时候不要漏掉px单位
```

}

好， 我们看一眼生成的代码，top， right， bottom， left的值就被设成了我们传入的参数值大小。

有同学可能会说， 我就记不住这个参数顺序， 那怎么办呢？ sass提供了我们一种叫做keyword arguments的方式， 在传参的时候你可以明确指定某个参数对应的值是啥。

```
.stretch-full {
  @include stretch($offset-top: 3px, $offset-bottom: 4px, $offset-left: 5px, $offset-right: 6px);
}
```

我们明确指定上下左右分别是3，4，5， 6 ，px 而这个顺序是打乱的。 我们看一下生成的结果对不对，

ok， 是没问题的。

这个模块还有其他的两个mixin， http://compass-style.org/reference/compass/layout/stretching/

分别是 stretch-x stretch-y, 我们用脚后跟想也知道这两个mixin是干嘛的了， 只在水平方向 或者 只在垂直方向 延展。 其实也就是只修饰 left, right 或者 只修饰 top, bottom。 就不多说了。

Sticky Footer 模块， 提供页面中的页脚部分始终处于最底部的能力， 当主体内容的高不足以占据整个浏览器窗口的高度时， footer要位于的浏览器窗口的最底部， 当主体内容的高超出整个浏览器窗口的高度时， footer要位于页面的最底部。

使用起来有点鸡肋： 它要求我们页面必须符合这样的一个html结构 （别忘了在html文件中引入css）

http://compass-style.org/reference/compass/layout/sticky_footer/

有一个root主体部分， 有一个footer部分， root主体部分还要有一个占位元素。

```
<div id="root">
  abcd
  <div id="root_footer"></div>
</div>
<div id="footer">
  Footer content goes here.
</div>
```

（别忘了在html文件中引入css）

引入一下我们的css文件。

然后在我们的sass文件的样式表根部， 调用 sticky-footer 这个mixin，

```
@include sticky-footer(30px);
```

还必须要给footer指定一个固定的高。

我们来看一下效果

```
@include sticky-footer(30px, "#my-root", "#my-root-footer", "#my-footer")
```

当然在符合这样一个html结构的基础上，选择器不一定非得是root, root_footer, footer, 可以自定义， 通过@include sticky-footer(30px);的参数指定， 第二个参数传入 root结构的选择器，第三个参数传入 占位结构的选择器， 第4个参数传入 footer结构的选择器。 这里我就不演示了， 大家课下自行实验。

Grid Background 模块， 使用CSS3的gradients， 为一个元素添加定宽，定高或者自适应宽高的格式背景。

我们直接看效果大家就明白了， 在刚才代码的基础上， 我们给root选择器添加一个 grid-background()

使用CSS3 gradients中的线性渐变生成这样一张格式图片， 然后把它设为相应元素的background-image。

还可以通过变量配置， 修改纵列的颜色：

$grid-background-column-color: rgba(255, 0, 0, .25); 我们改一个半透明的红色。

好， 我们看一下效果。

通过变量配置，修改mixin的行为， 是Compass的常用方式， 后边我们还会遇到很多次。

http://compass-style.org/reference/compass/layout/grid_background/ 相关的配置选项和可用的mixin在Compass Grid Backgrounds的介绍页面上都有详细的描述。

同样的， 我没有在实际项目中应用过这个模块， 经常做页面大段文本布局比如说电子书在线阅读的同学可能会用到， 比如说测试一下水平或者垂直方向上的布局是否协调。 所以建议有相关需求的同学， 仔细浏览这个页面上的所有mixin和配置项。没需求的同学记住Compass有这么个能力就好， 用到了再查，我们就不深讲这个东西了。

好， 关于Layout模块就介绍这么多， 下一节我们来聊一下CSS3模块&Browser Support模块。 谢谢大家。

《第三章：实战应用 - 第三节：CSS3模块&Browser Support模块》

大家好， 我是mater。如果说Layout模块是Compass中使用率最低的一个模块，那么CSS3模块肯定是主动使用率最高的一个模块。为什么加了主动两个字， 因为我们在用CSS3模块的时候往往要调整Browser Support 模块的配置， 即使我们不主动去调整，CSS3也引入了Browser Support模块，所以Browser Support模块算是CSS3模块的必要不充分条件。

CSS3 模块， 主要用来提供跨浏览器的CSS3能力。

打开我们要构建的项目的索引介绍页： http://webdemo.myscript.com/#/home

box-shadow: 1px 1px 3px 2px #CFCECF;

可以看到我们每一个版本介绍边缘都有一个阴影效果， 这个实现起来也非常的简单，

只需要给我们的webdemo-sec添加一个CSS3的box-shadow 修饰即可。

```
.webdemo-sec {
  @include col-sm(50%);

  box-shadow: 1px 1px 3px 2px #cfcecf;  横向偏1px，纵向偏1px， 模糊半径设个3px，阴影展开半径设一个2px， shadow颜色设成 #cfcecf 就差不多了。

  &:hover {
    background-color: #f5f5f5;
  }
}
```

但是问题来了， 我们都知道各个浏览器厂商对CSS3标准的支持情况不同， 要想让这个CSS3属性可以在各个浏览器里边都能正常地被呈现， 我们需要为这一属性添加浏览器前缀。

```
box-shadow: 1px 1px 3px 2px #cfcecf;
-webkit-box-shadow: 1px 1px 3px 2px #cfcecf;
-moz-box-shadow: 1px 1px 3px 2px #cfcecf;
```

这其实是所有CSS3属性在使用的时候一个非常令人头疼的问题， 还好我们有Compass，Compass为我们封装了几乎所有的CSS3新属性，我们只需要调用Compass封装的同名mixin即可。我们来看一下

我们首先引入CSS3模块。

balabala

@include box-shadow(1px 1px 3px 2px #cfcecf); 参数跟我们使用CSS3原生的box-shadow是一样的

好， 我们看一眼生成的代码。 可以看到Compass自动帮我们把该加的浏览器前缀给加上了。

可能有同学要说了， 我的页面是一个hybrid架构的内嵌页面， 只跑在webkit窗口里边，我不需要自动帮我生成适配firefox的CSS代码， 这时候就需要用到Browser Support模块了，我们用来它配置compass默认支持哪些浏览器， 对于特定浏览器默认支持到哪个版本。可以认为Browser Support 模块中的配置一旦修改， 将影响其余6个模块的输出。其中自然有CSS3模块。

这里我们引入Browser Support, @import “compass/support” 其实不写也没关系， 因为layout和css3模块都已经import了 Browser Support模块， 我们import了layout和CSS3相当于间接引入了Browser Support。 但重复写了也没副作用， 为了演示清晰， 我们写上。

Compass 内置了一个 browsers() 函数， 用于返回一个浏览器 name的list， 我们可以使用Sass的debug指令，打一条debug日志看一眼。

@debug browsers();

我们可以看到有 android, android-chrome, chrome, firefox, 等等

除了debug 的方式， compass也提供了一个console可以直接交互：

我们在命令行直接执行 compass interactive ， 就进入了这样一个console， 然后可以调用browsers

browsers() 注意不要加分号

如果是要查看某一浏览器都有哪些版本， 可以使用 browser-versions() 函数， 参数传浏览器的name

browser-versions(chrome)

balaba 我们可以看到chrome有这么多版本现在仍然是被考虑的。

ctrl c 退出console。

browser support 模块的所有配置项和函数， mixin 在 http://compass-style.org/reference/compass/support/ 模块的介绍页都有详细的介绍， 我这里只演示用的比较多的， 想了解更多， 大家下来看文档。

$supported-browsers: chrome; 声明我们想支持哪些浏览器， 值为一个list， Sass对List变量的定义很灵活， 用逗号 或者 空格 分隔的一组值就叫一个list， 所以我们这里既可以写成 chrome firefox 又可以写成 chrome, firefox 效果是一样的。只有一个chrome它则认为是只有一个元素的list。 我们这里只写一个chrome， 看一下生成效果。

。。。balabala

我们发现Compass只对应生成了box-shadow对chrome浏览器的适配。

现在如果有哪个项目经理跑过来告诉你说我们的项目要支持到IE6， 我觉得这都不是道德堕落，败坏可以形容的， 这简直就是犯罪。 开个玩笑， Compass默认支持到了IE5.5，而我对我的团队的要求是支持到IE8就已经很不错了， 这可以通过配置：

$browser-minimum-versions: ("ie": "8", "chrome": "18"); 来实现， 值为一个map数值类型。很像js里边的object， 由键值对组成， 用一个大括号括起来。

因为并不是所有的浏览器版本都是一个数值， 所以注意这里要传一个字符串。

如果不在这里边指出的话， 默认支持所有的 browser-versions 返回的版本。

好， Browser Support模块我们就说这两个配置项， 因为用的确实不多， 回到CSS3模块， 我们再大致浏览一下compass都支持了哪些CSS3特性：

Animation: CSS3动画相关的特性， 包括 keyframes， animation等属性。 这些我就不一一演示了， 原理跟box-shadow 都是一样的， 大家首次用到哪个属性的时候就过来读一遍。

Appearance: CSS3的appearance属性， 也是css3的新规范中定义的一个新属性。

比如说我们可以修饰div， 让它看上去像一个button。

```
div {
  appearance: button;
}
```

现在还没有一个主流浏览器直接支持appearance属性，chrome， safari支持添加-webkit前缀， firefox支持添加 -moz前缀。 所以我们可以看到 这个输出结果跟 box-shadow 不同的地方在于， 生成结果里边它没有一个 不加前缀的 appearance 属性了。

```
div {
  @include appearance("button");
```

}

Background Clip, Origin, Size： CSS3新增的background 相关的属性， 用来规定背景的绘制区域， 背景图像的定位原点， 背景图像的尺寸等。

Border Radius: 边框圆角属性

Box Sizing： 用来修改盒模型宽高的度量方式

CSS Regions： 控制内容布局的新方式， http://www.qianduan.net/css-regions.html/comment-page-2#pageTop 可以控制文字这样显示。

CSS3 Pie： 呢不是一个浏览器属性， 所以不太一样， 因为IE浏览器的升级由于众所周知的原因不像chrome， firefox 那么容易和快速， 所以很多CSS3的属性IE是没办法支持的， 外国的一些大牛就想了一些hack的方式让IE支持这些CSS3属性， 而pie就是其中的一种。 Compass包含了一个pie的扩展，但需要额外安装，在pie的介绍页都有详细说明，用到的也不多， 因为对于IE国内通常的做法都是降级处理。

Columns： CSS3的多列布局属性。

Filter： 大家不要以为这个是说IE的滤镜，完全两个东西， 这个是CSS3 的filter，主要是用在图片上实现一些特效。

Flexbox： 布局， 现在移动端web开发用的比较多， 但是关于这个属性的草案实在是有点多， 这个时候就更有必要用Compass了。

Font Face： 不依赖于用户计算机上安装好的字体， 指定下载某一种字体。

Hyphenation：/ˌhaifə’neiʃən/ 如何断字换行。

Images： CSS3 新增了/ˈlɪnɪə-/ liner-gradient, /ˈredɪəl/radial-gradient 这种生成渐变图形的方式， Images用于需要使用这两种方式充当图片的场景， 覆盖background，background-image， content等属性。

Inline Block: 实现跨浏览器的 display: inline-block 能力。 这个就不是添加一个 -webkit -moz 属性前缀这么简单的事情了， 而是要兼容IE6, 7. 我们看一下Compass是如何实现的。

```
.text-inline-block {
  @include inline-block();
}
```

Compass使用了IE独有的 * hasLayout 特性来模拟inline-block能力。 如果有同学不知道hasLayout可以看我课后的wiki， 里边有相关的学习资料。

Opacity： 透明属性, 同样是为了兼容低版本IE， 使用IE的filter来实现透明效果。 我们来看一下。

```
.opacity {
  @include opacity(.3);
}
```

Selection： 使用CSS3的 ::selection 伪元素， 定义被选中文本的颜色和背景色。

Shared Utilities: 如果哪天你想贡献CSS3模块的相关Compass插件了， 你可能会用到这个模块， 这是一个工具类模块。

Text Shadow： 文字阴影属性

Transform：Transition： 变换，动画属性。

User Interface：包括 user-select 属性， 限制某一区域是否允许鼠标拖拽选择。 input-placeholder：input元素在无填写状态下提示语的样式。

好， 说完了， 为什么要全部过一遍， 因为我相信其中有些CSS3的能力， 大家甚至都没有听说过， 有必要过一下， 留个印象。

关于CSS3模块&Browser Support模块我们就介绍这么多， 下一节来聊一下Helpers模块&Typography模块。 谢谢大家。

《第三章：实战应用 - 第四节：Typography模块 一》

大家好， 我是mater。这一节我们来看一下Typography模块。 王家卫的《一代宗师》里边有一句： “人活一世， 有的人成了里子， 有的人成了面子。” 如果从表意上划分一下，Typography模块一定属于面子， 因为它主要用来修饰我们的文本样式，垂直韵律。而我们后边要介绍的Helpers模块则属于里子。

Typography 内又划分了4个模块， 分别是

： Links 链接修饰模块

： Lists 列表修饰模块

： Text 文本修饰模块

： Vertical Rhythm 垂直韵律修饰模块

因为Typography模块的内容比较多， 所以我们分两个小节， 每个小节讲两个子模块。 这一节我们先来看 Links 和 Lists。

既可以针对不同的模块分别引入Typography， 比如说可以单独引入 links,

@import “compass/typography/links"

也可以笼统的全部引进来， @import “compass/typography” 咱们为了演示， 一个个的来引入。

我们对超链接最常做的修改就是： 1. 正常态下去掉下划线， 在hover或者focus的时候才显示下划线 2. 修改不同状态下超链接的颜色。

1. 我们可以通过 hover-link() 这个mixin实现，

```
a {
  @include hover-link();
}
```

balabala

2. 我们可以通过 link-colors(#00c, #0cc, #c0c, #ccc, #cc0) mixin来实现

```
a {
  // normal hover active  visited focus  后4个参数非必填， inherit
  @include link-colors(#00c, #0cc, #c0c, #ccc, #cc0)
}
```

balabala

有时候我们可能仅仅是想抹平超链接的样式， 让它跟它所位于其中的文本的样式一样，就可以通过 unstyled-link mixin 来实现

```
a {
  @include unstyled-link();
}
```

balabala

使用Lists模块， 需要首先引入 @import "compass/typography/lists"

(使用bootstrap的演示页面说明list的各种情况)

file:///Users/mater/Library/Application%20Support/Dash/DocSets/Bootstrap_3/Bootstrap%203.docset/Contents/Resources/Documents/getbootstrap.com/css/index.html

我们一般使用ul标签构造无序list， 使用ol标签构造有序list， 不加任何修饰的ul，ol就长长这个样子， 但有时候我们需要去掉前边的点号， 变成这个样子，可以使用Compass封装的no-bullets mixin 来进行 list-style: css属性的修饰。

```
.list-unstyled {
  @include no-bullets();
}
```

balabala

也可以使用 no-bullet mixin 去掉单个li元素前边的list-style,

```
.list-unstyled-li {
  @include no-bullet();
}
```

可以使用mixin pretty-bullets() 很方便的把 list-style 设置成一张我们自定义的图片， 但是我真没见到过哪个互联网站点这么干的， 所以用到的概率应该非常低， 感兴趣的同学自己看。

有时候我们需要list以这样横向的方式来布局， 最简单的方式就是将li元素的display值设为inline，Compass封装了相关的 inline-list mixin,

```
.list-inline {
  @include inline-list;
}
```

balabala

但inline存在一个问题， 举个例子， 我们看bootstrap 上边的这个导航条， 就是一个横向的list， 因为inline元素没有高度， 所以就无法撑起这个导航条， 鼠标hover的这个阴影效果就不能直接在li元素上实现了， 这时候我们倾向于选择让li元素保持原有的display值为block, 然后float 浮动li元素。

可以使用 horizontal-list mixin来实现。

```
.list-horizontal {
  @include horizontal-list（）;
}
```

balabala

horizontal-list 支持两个参数， 第一个是 padding的值， 第二个是 浮动的方向， 我们来试一下。 我们把padding设为0.

```
.list-horizontal {
  @include horizontal-list(0, right);
}
```

balabala

但是相信被IE虐惯了同学，刚才看到这个地方的 first-child, last-child，就已经开始在惊呼了： 你丫在逗我吗， last-child 连IE8都不支持好不好， 更别提IE6， IE7了。

如果我们的li元素不需要padding的话， 这个问题就好解决了， 我们也可以把 padding的值设为false， 来去掉li元素之间的padding。 当然了， 我更希望的是大家都不用兼容IE6，7，8.

```
.list-horizontal {
  @include horizontal-list(false);
}
```

balabala

除了float的方式， 我们还可以通过设置li 的display值为 inline-block 来实现同样的目的，Compass也封装了相关的mixin： inline-block-list;.list-inline-block {

```
  @include inline-block-list;
}
```

balabala

inline-block-list 同样支持padding参数，

```
.list-inline-block {
  @include inline-block-list(7px);
}
```

balabala

可以看到 inline-block-list 和 horizontal-list 的 container element的样式修饰一模一样，
其实 horizontal-list 和 inline-block-list 还分别封装了对应功能的只用于container和 list内部元素的mixin，

horizontal-list-container

和

horizontal-list-item

inline-block-list-container

和

inline-block-list-item($padding)

用来满足我们的特殊化需求，

而通过source 我们可以看到 inline-block-list-container 调用的就是：

horizontal-list-container;

一般不会单独用到这些mixin， 所以我们就不演示了。

好， Links 和 Lists 子模块就说这么多， 下节我们来看看 Text 和 Vertical Rhythm 子模块。 谢谢大家。

《第三章：实战应用 - 第五节：Typography模块 二》

大家好， 我是mater。这一节我们来看一下Typography的Text和Vertical Rhythm 子模块。

文本操作常常是针对那些过长的文本， 强制换行或者不换行， 超长使用省略号显示等等， 使用Text模块， 需要首先引入 @import “compass/typography/text";

如果是一个很长的url地址或者连续的长文本， 默认是单行显示不换行的， 如果其长度超出了父容器的宽度， 就会破坏我们的布局， 所以我们需要使用 word-wrap 属性强制其换行。

我们可以使用 force-wrap mixin来实现这个目的：

```
.text-force-wrap {
  @include force-wrap;
}
```

balabala

word-wrap: 默认值为normal，设为break-word 用来强行把连续的长文本在达到父容器边界的位置掰断。

white-space 的pre-wrap和 pre-line是CSS2.1中新增的， 所以这里做了一个降级处理， 设为pre-line 合并空白符序列， 但是保留换行符， 该换行的地方换行。

如果一串文本， 不想它在任何地方换行， 只需要把 white-space 属性设为 nowrap 即可。 但是Compass呢就像一个贴心的小姑娘， 担心我们记不住white-space属性在写的时候中间是不是应该有个中划线， 所以也帮我们封装了一个mixin nowrap,

```
.text-nowrap {
  @include nowrap;
}
```

balabala

不换行的时候， 文本超出了容器的宽怎么办呢？ 我们常用的一个方法是以省略号的形式显示， 然后hover的时候展示全部文本。(用 find.qq.com 来举例， 提前登陆QQ账号)

我们可以使用 ellipsis mixin 来实现这个目的：

```
.text-ellipsis {
  @include ellipsis;
}
```

balabala

但是firefox 的低版本并不能很好的支持 text-overflow 属性， 我们可以使用firefox私有的 -moz-bind 属性来进行hack， 使用XBL（XML Binding Language）绑定一个DOM元素，这需要对应的XML文件， Compass都已经封装好了， 我们只需要在命令行调用：

```
compass install compass/ellipsis
```

这个文件就会被输出到我们的 stylesheets 目录， 然后通过变量配置的方式：

$use-mozilla-ellipsis-binding: true; 开启对firefox ellipsis 能力的支持即可。

balabala

（打开QQ查找，找服务页面）

有时候我们会有用图片替换文本内容的需求， 比如说这里的查找按钮， 因为背景是一个渐变色效果， 所以CSS实现起来就比较麻烦， 我们通常的做法就是用一张背景图片。 同时为了能让读屏软件读到相关的文本信息， 标签内的文本我们要保留有效信息， 比如说这里标签内还是要写查找， 但如果我们的背景图片出图的时候上边已经带有查找两个字了， 则需要使用 text-indent 属性， 将这文本偏移隐藏。

隐藏文本可以使用 hide-text mixin。

```
.text-hide {
  @include hide-text;
}
```

balabala 单词的首字母大写（贴心小tips）中文，呵呵

hide-text 支持参数来修改 text-align 的方向， left还是right， 像阿拉伯语这种从右到左布局的语言才有可能用得到修改这里。

当然了隐藏文字还有另外一种方式， 就是把它的字体设置为0， 颜色透明， 不要字体阴影。

我们通过 squish-text mixin来实现这个效果。

```
.text-hide {
  @include squish-text;
}
```

balabala

只隐藏文字还不行， 还得把背景图片加上去， 可以使用mixin replace-text 轻松实现这个目的。

比如说我们的项目介绍主页 左上角的返回公司主页的超链接， 我们假设这张图片的地址是： http://www.webdemo.myscript.com/images/home.png

那么就可以写成；

```
.home-link {
  @include replace-text('http://www.webdemo.myscript.com/images/home.png');
}
```

balabala

当然我们这里也可以直接传一个本地图片的路径，关于如何方便快捷的使用本地图片，我们在下一节会有重点介绍， 所以这里我们先使用一个url地址表示。

默认呢， background position 就是50% 50% ， 这样背景图片刚好居中显示。 可以通过 replace-text 的第二个和第三个参数分别调整 background position x，y的值。

但有一个问题， 如果home-link 容器的大小比图片大， 那么就会有背景色漏出来， 也不符合我们的预期， 我们希望容器大小刚好跟图片一样大。

如果我们引用的是本地图片这个问题就非常好解决了， 可以使用 mixin replace-text-with-dimensions, 它会自动帮我们计算本地图片的宽高， 用得到的宽高值去设置容器元素的宽高。 正如刚才所说，关于本地图片的路径如何书写， 我们下节会重点介绍。 这里先卖个关子， 在我们学完下一节内容之后， 大家课下回过头来来试验一下这个mixin。

我们在用CSS格式布局的时候， 更多的是关注垂直列之间内容的空隙， 却往往忽略了行与行之间内容的留白。相信大家一定知道五线谱长什么样子， 把页面类比过来， 网格布局的线很像节拍线， 主体文本内容沿着节拍线形成一个韵律， 这种效果我们就称之为垂直韵律。

使用之前， 需要 @import "compass/typography/vertical_rhythm"

大家还记得 Layout模块有一个 Grid Background 的子模块吗?

讲到的时候我们说因为没有在实际项目中应用过这个模块所以不展开， 垂直韵律也一样，经常做页面大段文本布局比如说电子书在线阅读的同学可能会用到， 用来测试一下水平或者垂直方向上的布局是否协调。实际用到的比较少，所以我们这里也不太深入。

垂直韵律的原理就是： 需要让所有文本元素的高是我们的基准高的整数倍。所以

实现垂直韵律我们首先需要设定我们的行高即主体文本之间的行间距和基本的字体大小。

一般我们常用的是字体大小16px, 这是浏览器默认的字体大小， line-height 24px，根据经验，可读文本的一般规则是将行高设置为字体大小的1.4 ~1.5倍， 这里我们为了给文本之间留出更多的间隙，采用1.5倍， 16 * 1.5 就是24px。

通过配置变量， $base-font-size 和 base-line-height设定， 16px 和 24px 其实也分别是这两个配置变量的默认值。

$base-font-size: 16px;

$base-line-height: 24px;

在垂直韵律模块内，封装了类似于Grid Background的mixin，debug-vertical-alignment 可以生成辅助我们调试的背景线， 加在body上。

```
body {
  @include debug-vertical-alignment;
}
```

看效果之前呢， 让我们先应用一下之前学到的知识， 把reset给应用进来， 抹平不必要的margin， padding。 好， 看一下效果。

balabala

这时候我们在我们的html文件里边随便塞些内容。

(提前把内容拷贝好)

可以看到这些内容并没有沿着我们的基线排布。为了修复这个问题， 需要让标题内容和其他元素的高是我们的基线24px高的整数倍。

首先调用 @include establish-baseline mixin 初始化html标签的字体和行高。

balabala

em相对单位不用我多说了吧，1em是当前font-size的一倍，而1.5em就是当前font-size的1.5倍， 如果是用在font-size属性 上， 就是相对于从父元素继承来的font-size大小的比率。

然后基于此指定我们各个h标签和p标签的字体。

h1 {

font-size: 3em; /* 48 ÷ 16 = 3 */

// 这时候要想让h1的高为基线24px高的整数倍，我们可以取1倍， 那line-height就是24px但我们的字体大小都已经48px, line-height总不能比字体大小还小， 所以我们取2倍， 就是48px，但是line-height 和 font-size同样大小， 在多行的时候就很难阅读了， 所以我们取3倍， 就是72px 相对于h1的font-size 48px 表示出来就是： line-height: 1.5em;

}

h2 {

font-size: 2.25em; /* 36 ÷ 16 = 2.25 */

依次类推下去， h2的line-height 我们可以用 48px, 相对于h2的字体大小36px， 就是1.333333333 em

}

h3 {

font-size: 1.5em; /* 24 ÷ 16 = 1.5 */

h3的line-height 我们可以用 48px, 相对于h3的字体大小24px， 就是 2 em

}

h4 {

font-size: 1.3125em; /* 21 ÷ 16 = 1.3125 */

h4的line-height 我们可以用 24px, 相对于h4的字体大小21px略微大那么一点点，但担心太过于紧凑了，所以我们取48px， 相对于h4的字体大小21px 就是 2.28571em,

}

h5 {

font-size: 1.125em; /* 18 ÷ 16 = 1.125 */

h5的line-height 我们可以用 24px, 相对于h5的字体大小18px， 就是 1.3333333333333333em;

}

h6 {

font-size: 1em; /* 16 ÷ 16 = 1 */

h6的line-height 我们可以用 24px, 相对于h6的字体大小16px， 就是 1.5em;

}

p {

font-size: 1em; /* 16 ÷ 16 = 1 */

p的line-height 我们可以用 24px, 相对于p的字体大小16px， 就是 1.5em;

同时， 我们再给p设一个上下各24px的margin， 就是1.5em。

}

好，我们看一下效果。

balabala

但是这样计算的过程太痛苦了， 尤其是对于老外这种100以内的加减法都必须用计算器的。 所以Compass帮我们封装了mixin adjust-font-size-to

```
h1 {
  font-size: 3em; /* 48 ÷ 16 = 3 */
  line-height: 1em;

  @include adjust-font-size-to(48px);
}
```

我既不需要计算font-size的倍率了， 又不需要计算 line-height的倍率了， 只告诉compass， 我想让h1的字体变成多大。这里我们指定变为48px。Compass就会自动帮我计算出字体应有的值和line-height应有的值。

balabala

当然 我们依然可以传入em为单位的字体大小值。 其他的都是一样的， 我们改一下， 看一下效果。

回过头来说p元素， 为了便于阅读， 我们设了一个1.5em的margin， 这其实也有了一个计算过程。

我们这里的原则就是计算通通的不要，

有mixin leader 和 trailer 来帮我们做这样的计算， leader用来计算margin-top 应有的值，trailer 用来计算margin-bottom 应有的值。

默认值是一倍于基线的高， 我们可以通过参数指定几倍。

```
p {
```

```
  @include adjust-font-size-to(16px);


  @include leader();
  @include leader(2);
}
```

balabala

好， 垂直韵律还有一些其他的辅助mixin， 感兴趣的同学呢，自行了解， 我们就不多说了， 这节就到这里， 下一节我们一起看看Compass Helpers 模块。

《第三章：实战应用 - 第六节：Helpers模块》

大家好， 我是mater。这一节我们来看一下Helpers模块。

http://compass-style.org/reference/compass/helpers/ Compass的Helpers页面列出了所有Compass封装的函数，跟Sass的函数列表很像， 感兴趣的同学可以一一浏览一下， 这里我们不会全讲，大多数是用不到的，甚至有你完全搞不懂可以拿来干嘛的函数。

比如说Constants 模块里的这个函数： opposite-position 对上下左右中取反方向， 看下边的示例， balabala 这样的接口设计出来就完全是浪费脑细胞吗。

（提前把迅雷的下载协议地址复制进剪切板）

（http://webdemo.myscript.com/#/home）依然是继续实现我们项目介绍主页的相关功能。 可以看到我们每一个功能板块都有一张对应的介绍图片。 8张图片，如果不合并图片，不使用CDN cache的话每次打开页面就会有额外的8条HTTP请求，为了减少链接数，我们可以把图片转成Base64编码的形式放在CSS中，随CSS一起返回，这种方式我们称之为Data URI。这样做麻烦的一点在于我们一张张去获取每张图片的Base64编码。Base64大家应该都已经了解了： Base64是网络上最常见的用于传输8Bit字节代码的编码方式之一，可用于在HTTP环境下传递较长的标识信息。我们经常用的迅雷下载， 它的下载协议地址： thunder://QUFodHRwOi8vd3d3LmJhaWR1LmNvbS9pbWcvc3NsbTFfbG9nby5naWZaWg== 就是base64编码的。一堆连续的字母数字，最后1~2个“=”的代码往往就是base64. 最近迅雷宣布1.3亿卖掉了旗下的迅雷看看， 哎， 怎么说呢？ 迅雷这两年一直都处于一种“瞎忙”且“累死”的节奏，诸多努力都化为乌有。现在卖掉视频这一还算有想象空间的业务， 个人感觉再难翻身了。有机会也希望能跟大家聊聊对互联网的一些感慨。

好，看一眼我们的项目结构： 我们新增了一个images文件夹，里边放了我们的8张功能介绍图片， 文件夹名字的声明是要符合我们 config.rb 里边的配置的， 我们这里配置的图片目录名就是 images。使用Compass的相关图片操作的Helpers, Compass对图片的寻路就是根据我们这个images_dir 的配置来的。

（演示：）

我们传统的做法是找个小工具， 计算出图片的base值， 然后把它填写进原来应该填写图片url地址的地方。

```
.analyzer-logo {
  background-image: url('}
```

这种方式繁琐且不直观， 所以Compass就封装了一个Helpers， inline-image , 之前我们已经讲过functions直接用来调用， 所以这里直接写成：

```
.analyzer-logo {
  background-image: inline-image('analyzer.png');
}
```

参数就是相对于我们设定的图片目录 寻路的图片文件名。

我们看一眼生成的代码： Compass帮我们计算出相关图片的base64值，并在生成的css文件中进行相关的替换。 简单方便快捷。

但是Data URI的方式呢，我们之前做过实验： 比直接使用图片大概要多消耗50%左右的CPU资源，多使用4倍的内存。 并且其不支持IE6，7， 所以用的场景也不是特别多， 但也绝不是毫无用武之地。大家选择合适的场景使用。

我们后边会讲如何利用Compass自动合图， 那个是我大力推荐的方式。引用霸王别姬中的一句《仗剑在手，胜券在握》， 相信你遇见的时候也一定会爱上其强大的功能。

另一个跟 images_dir 配置项相关的helper 函数是 image-url，如果按照CSS的写法， 我们引用analyzer.png这张图片肯定是写成：

```
.analyzer-logo {
  background-image: url('../images/analyzer.png');
}
```

在写的过程中，我们很容易一不留心就把这个路径给拼错了。 即使路径写的没问题， 还存在一个图片更新的问题， 如果图片和CSS都是用CDN cache加速的话， 我们要想更新这张图片就需要在后边加一个时间戳或者版本号， 老外喜欢把这串东西称之为 cache-buster, 缓存克星。

```
.analyzer-logo {
```

```
  background-image: url('../images/analyzer.png?t=20150401001');
}
```

图片每更新一次，就修改一次这里的版本号或者时间戳，cache-buster。 麻烦且易出错。

image-url 很好的解决了这个问题，

```
.analyzer-logo {
  background-image: image-url('analyzer.png');
}
```

首先是引用路径， 我们只需要关心图片相对于我们配置的 images_dir 的路径， 所以这里我们可以直接写成 analyzer.png。

好， 我们看一眼生成的代码:

我们发现在图片路径的最后跟了一串数字， 这个cache-buster是Compass根据图片的修改时间算出来的， 这也就是说只要我们的图片发生改变， 后边跟的cache-buster就会相应变化，解决了缓存问题。

相信细心的同学肯定注意到了一个问题， 那就是这里自动生成的url是一个绝对地址， 一个 / 跟在最前边，表示相对于url根路径寻路。 这个是由另一个配置项决定的。

http_path = “/"

Compass helpers functions 在生成url路径的时候，默认都是采用绝对路径的方式，做法就是将当前的config.rb 文件所在的位置认为成是根路径，得到其他资源文件相对于根路径的路径，我们这里analyzer.png 文件的相对于config.rb 文件的路径就是： images/analyzer.png 然后再把 http_path 加在这个相对路径的前边。

如果我们需要直接使用完整的url路径，这里就很方便了。

举个例子：慕课网网站用到的图片资源都是放在域名： img.mukewang.com 上的， 那我们就可以直接把http_path 改成 http://img.mukewang.com/compass-demo, 我们看一眼生成的结果：

url地址变成了完整的，加上我们域名以及额外path的地址。

如果我们的图片， css， html等都不是放在一个域名下的， 也很方便解决， Compass提供了更细的http_path的声明， 我们可以声明：

http_stylesheets_path

http_images_path

http_generated_images_path

http_javascripts_path

http_fonts_path

等等， 针对不同的文件类型，配置不同的http_path, 绝对足以满足我们的个性化需求。

当然了， 我们这里还是要使用相对路径的， 非常容易， 还是修改一个配置的问题，config.rb 文件中的 relative_assets 配置项就是用来控制使用绝对路径还是相对路径的，只要我们把这个配置项设为true， Compass就会使用相对路径了。

我们来试一下， balabala。

image-url() helper function 还有两个额外的参数， 用来控制是否返回前边的url，即只返回url路径 和 是否生成cache-buster, 或生成什么样的cache-buster, 一般都是用不到的，大家课下感兴趣自行了解。

跟image-url 功能类似的，还有stylesheet-url 和 font-url 两个 helper, 用来帮我们管理指向 css 目录和 font 目录的资源文件路径，参数用法都一致，就不演示了。

接下来我们快速过一下Compass的Helper list,

color-stops()

我们在使用CSS3的渐变属性生成图片的时候，有时候为了打造更丰富的渐变效果，除了声明渐变线上的起始点和终止点色值，还要声明一些中间点的色值， 这些点我们称之为color-stop points, 这种点往往是一个色值+当前点的位置表示， 当我们省略点的位置表示的时候， color-stops helper function 会自动帮我们计算输出这个点应有的位置值， 其实也就是它前一个点和后一个点的中间位置， 但是浏览器在渲染省略点的位置的color-stop points的时候， 其实也是这么干的， 自动计算中间值， 所以说， 这个helper function 用处不大。

Colors 里边的函数用来量化调整一个色值，用到的极少极少。

Constants 我们已经说过了， 完全摸不到头脑。

Cross Browsers 如果我们要为Compass贡献跨浏览器的CSS新特性插件，常会用到这里边的helpers 函数， 用到了再来查。

Display Helpers:

只有一个helper function, elements-of-type 参数值为 block, inline inline-block 等display的取值， 返回结果为默认display属性为对应参数的html标签，还记得我们在讲Reset模块的时候吐槽的 reset-display mixin吗？(点看相关页面的source展示) 其内部就是调用的 elements-of-type 这个 helpers.

这里还调用了我们马上要讲 append-selector() 这个helper。 等下讲到再说， 注意这里的写法， # {} 。

Environment Helpers, 获取当前编译的一些环境信息， 我们只讲其中一个。

compass-env()

返回当前compass的编译环境， 函数的执行结果只有两种可能， 一个是development, 一个是 production。 在这两种环境下， compass的有些行为是会不一样的， 比如说之前我们一直觉得有点烦的（在css中勾选）这种便于调试的注释信息， 在我们不需要调试的时候，其实是可以通过配置 config.rb 文件中的 line_comments 来关掉的。

我们来试一下。

balabala

一下子， 清爽。

我们先恢复一下。

当我们不设置这个值的时候， 使用的其实就是当前Compass编译环境下的默认配置项。默认我们是在 development 环境下的， 我们来验证一下， 在screen.scss中加一条调试语句:

@debug compass-env();

我们编译一下看

演示。。。balabala

我们可以在命令行中强制指定使用production环境。

compass compile -e production --force

-e 用来指定环境 force 让Compass覆盖重写已有的文件。

看一眼生成的文件， balabala

我们也可以在config.rb 文件中通过配置项指定Compass的编译环境，

set the Compass compile environment

environment = :production 参数就是 enviroment, 值为冒号 production 或者 development

好， 我们编译一下。

balabala

Font Files： 作用跟我们前边讲的 image-url 很像， 只不过这个是针对字体文件的， 基于我们config.rb 文件中fonts_dir 配置项， 配置的字体文件目录， 帮我们解决文件寻路以及cache-buster的问题， 额外根据字体文件帮我们生成 format的值。

好， 我们来试一下， 在fonts目录中添加一些字体文件， 我们这里添加的是常用的字符画的字体文件， 为了达到兼容所有浏览器的目的， 我们常常需要嵌入五种格式的字体文件， 真心不容易啊。

我们先来看一下font-files 的输出，参数就是一个字体文件组成的list。还是使用Sass的debug指令

@debug font-files("FontAwesome.otf", "fontawesome-webfont.eot", "fontawesome-webfont.svg", "fontawesome-webfont.ttf", “fontawesome-webfont.woff");

好， 编译一下

可以看到每一个字体文件都对应输出了相应的url和format。

font-files helper 绝大多数时候都是用于我们前边讲的CSS3模块的 font-face mixin,

@include font-face("FontAwesome", font-files("FontAwesome.otf", "fontawesome-webfont.eot", "fontawesome-webfont.svg", "fontawesome-webfont.ttf", "fontawesome-webfont.woff"));

看一眼输出结果， 如果是原生CSS， 我们要写这么多东西， 为了便于浏览器识别，format还不能错。

关于font-face的知识呢， 我就不展开了，如果有同学想了解更多， 可参见课后wiki。

Image Dimensions 包含两个helper, image-width 和 image-height, 用来计算图片文件的宽高， 合图组件会用到这两个能力。

Inline Data 一开始就说了， 将文件转化为Base64编码的格式， 内嵌在我们的css文件中。

Math：一些数学计算能力的Helper， 比如说pi的值，sin() cos() 计算一个值的正弦，余弦值， 还记得正弦函数吗？ sin30deg=1/2, sin45deg=二分之根号2， sin60deg= 二分之根号3. 我们一般也用不到。

Selectors： 一个选择器嵌套组合函数， 一个选择器叠加组合函数。 工作原理是一样的，所以讲一个就够了， 我们讲append-selector

append-selector() 一共有两个参数， 作用就是将第二个参数叠加组合进第一个参数。 很明显哈不是用在我们的规则属性里的，是用在选择器位置的。很多新人常犯的一个错误就是这样写：

```
append-selector("p, div, span", ".bar") {
  color: #000;
}
```

我们试着编译一下。 balabala

（注意演示）前边课程学的好的同学在我重点提醒注意 reset-display mixin 这里的写法的时候， 可能已经发现了端倪，在选择器，属性名或者字符串中我们如果想要引用sass变量或者函数， 需要使用 sass的interpolation/ɪn,tɚ·pə'leʃən/

#{} 里边跟变量或者函数的名字。 所以正确的写法应该是：

```
#{append-selector("p, div, span", ".bar")} {
  color: #000;
}
```

好， 我们编译看一下， 可以看到编译通过， .bar 被叠加组合加入了前边的选择器。

Sprites, 合图相关的helper, 一般我们不会直接用到， 因为Compass基于这里边的helper为我们封装了相关的mixin， 这些mixin被划分在了Compass的 utilities 模块中， 我们下节会讲， 但是正如我一直说的， 这些helper在我们为Compass贡献插件的时候作用就大了，

需要我们有一个清晰的掌握， 所以讲我们的retina合图插件的时候， 我们也会回过头来细讲这里知识， 现在先跳过。

URLs： 用于解决我们对于资源文件寻路和cache-buster的痛苦问题， image-url 我们已经说了， 其他的都大同小异， 有对CSS文件的， 有对字体文件的， 有对合图生成的图片的， 大家用到的时候过来看一眼。

好， Helpers模块我们就说完了， 所谓： 欲练此功，必现 “修炼好内功哈”， 你是不是想歪了， Helpers呢， 就有点像内功， 虽然用的少， 大家也不可忽视， 需打好基础。

下一节让我们来看一下最后一个模块， Utilities .

《第三章：实战应用 - 第六节：Utilities模块一》

大家好， 我是mater。这一节我们来看一下Utilities模块。

见文知意： 辅助工具类的模块， 跟Helpers不同的是， Helpers内是函数， Utilities内多是mixin。那到底哪些应该归类于Utilities模块呢？ 你可以认为没办法放到其他模块的内容，都可以放到这个模块。脑海里边突然蹦出一句美国作家梭罗在《瓦尔登湖》里边的一句名言： 可怜呐！人类最终却沦为了自己工具的工具。 我们一定要对这块掌握透彻， 让工具彻底的为我们所用。

Utilities模块内部又分了5个子模块， 分别是：

Color：

颜色相关的工具集合

Print：

打印控制的工具集合。

Tables

Table样式相关的工具集合， 比如table border的控制， 隔行色值差异控制等。

我们下一节重点介绍这三个模块。

General：

通用的一般类的工具集合， 比如说跨浏览器的float， 清除浮动等等。

内容略多， 我们单独放一节来讲。

Sprites

精灵图合图相关的工具集合。

使用Compass的重中之重， 我们放到Compass介绍的最后一节来讲。

既可以笼统的将全部Utilities模块引进来@import “compass/utilities”

又可以针对不同的模块分别引入， 比如说可以单独引入 Color子模块,

@import “compass/utilities/color"

咱们为了演示， 一个个的来引入。

Color相关的功能，咱们直接来看： brightness funcs, 用来计算一个值的亮度， 还记得我们之前聊过的用HSL表示色值的方法吗？ brightness 就是用来计算出一个色值的L。 函数的参数就是一个色值， 返回值是一个 0% 到 100%的数值。

balabala

@debug brightness(#000);

@debug brightness(#ccc);

@debug brightness(#fff);

constrasted mixin 根据我们传入的背景色的色值， 自动帮我们生成background-color 属性， 同时在预设的默认深色值和默认浅色值中选一个跟我们的背景色对比度大的设为我们的color属性， 为的是让文字可以在当前背景下更好的凸现出来。

我一直觉得Color相关的, 不管是Sass， 还是Compass的特性用处都不大，但 东野圭吾的《嫌疑犯X的献身》里边有一句：

这个世上没有无用的齿轮，也只有齿轮本身能决定自己的用途。

又常常刺激着我去辩证的想问题，也许是我还没发现吧。所以希望大家能够善于发掘，并且呢， 如果你能找到好的应用场景， 分享给我。 课后的wiki里边也给了大家两个玩Color function 的网站。

接下来看打印模块。

首先将Print模块单独引入进来， @import “compass/utilities/print"

Print 跟我们之前介绍的模块很大不同的一点是， 我们必须在两个文件中协同使用它。 别忘了我们还有一个专为打印准备的css文件， print.css, print.css 也需要引入 print模块。

@import "compass/utilities/print";

@include print-utilities();

然后在其内调用 print-utilities mixin， 回到 screen.scss 文件， 同样也在其内调用 print-utilities mixin，但要传一个额外的media参数， 指明是screen， 不传的话， 默认为print。

编译看一下，

screen.scss

对于那些只想让它在打印的时候可见的元素， 我们给它赋个类名叫做 print-only 即可。 在screen上， 其实不可见的。

看 print.css ，还记得我们print.css 文件的引入方式吗？ 注释里边有写哈。

<link href="/stylesheets/print.css" media="print" rel="stylesheet" type="text/css" />

将其引入进html。

只有打印的时候， print.css 才会被应用。

对于那些不想打印的内容， 我们给它们赋一个类名： no-print 建议大家采用 中划线的这种命名方式。 对于那些类名为print-only 的元素， print.css 根据其html标签对应的display 值设置其display 值， 来保证一个一致性。

因为没有相关的业务， 所以我没有在实际项目中应用过print模块， 但我觉得， 适配print样式， 绝不是这么简单的一件事， 最有可能的做法是screen下一套html结构， 在打印的时候， 在页面中开一个iframe， 这个iframe才是要打印的目标页面， 其内部的html结构会靠近screen下的结构， 但肯定得为打印去做调整。 尤其是页面上有 select， ratio 这种控件的时候， 这种控件直接打印出来是没有意义的。 如果采用iframe的做法， 对应地方的结构就可以用有意义的文本值替换。 大家下去也想一下， 希望你能把你的想法告诉我。我的微博，微信都是materliu， 大家也可以直接在课程后回复。

接下来我们看Tables模块。

其是Table样式相关的工具集合。

我们首先将其单独引入： @import “compass/utilities/tables”

Table Borders 子子模块， 用来给table添加border， 其中有两个mixin， 分别是：

outer-table-borders 和 inner-table-borders, 见文知意， 一个是用来修饰table外侧的边框， 一个是用来修饰table里边单元格之间的border。

Table Scaffolding 子子模块， Table 脚手架。 对table进行单元格文本的对齐， 以及 padding 的初始化。

Table Stripping 子子模块， 对奇偶行进行不同的颜色修饰。

我们一个示例就把这些mixin全部贯穿下来了。

在html中构建我们的table结构。

```
<table class="goods-price" cellspacing='0'>
  <thead>
  <tr class='odd'>
    <th>水果品类</th>
    <th>橘子</th>
    <th>苹果</th>
    <th>鸭梨</th>
    <th>香蕉</th>
    <th>打包</th>
  </tr>
  </thead>
```

```
<tbody>
<tr class='even'>
  <th>单价</th>
  <td class='numeric'>1</td>
  <td class='numeric'>2</td>
  <td class='numeric'>3</td>
  <td class='numeric'>4</td>
  <td class='numeric'>10</td>
</tr>
<tr class='odd'>
  <th>十个</th>
  <td class='numeric'>10</td>
  <td class='numeric'>20</td>
  <td class='numeric'>30</td>
  <td class='numeric'>40</td>
  <td class='numeric'>100</td>
</tr>
</tbody>
<tfoot>
<tr class='even'>
  <th>总额</th>
  <td class='numeric'>11</td>
  <td class='numeric'>22</td>
  <td class='numeric'>33</td>
  <td class='numeric'>44</td>
  <td class='numeric'>110</td>
</tr>
</tfoot>
```

</table>

首先给table起个类名， goods-price, 商品价目单。 加一个 cellspacing=“0” 的属性， 单元格之间我们不需要间隙。

有头，有尾，有身体。 分别是 thead , tfoot 和 tbody. 奇数行和偶数行之间， 我们使用类名 odd, even 来进行区分， 相关mixin中就是用类名odd和even来做修饰的。

横向第一行和纵向第一列是我们的table header部分， 因为我们都是用的th标签。这点很关键， 不要忽略了纵向第一列。

对于填充内容为数值的单元格， 我们用类名 numeric 来修饰， 这个类名不能随便取， 这是相关mixin中使用的类名， 所以这里必须一致。

好， 我们在浏览器里边打开看一眼。 非常的粗糙哈。

接下来，我们就一个个的把这些mixin用上去。

首先给我们的表格添加一个外边框。

```
.goods-price {
  @include outer-table-borders();
}
```

outer-table-borders mixin支持两个参数， 第一个参数是border的宽， 第二个参数是border的颜色， 这里我们用默认的 2px 宽， 黑色边框。

看一眼效果。

我们发现 thead, tfoot ，以及table最外框均被圈了起来。生成的相关CSS， 我就不带大家看了， 大家实践的时候， 自己看一下生成的CSS， 是怎么实现相关功能的， 记住下去一定要亲自动手实践。

内部单元格之间还没有边框， 我们给内部单元格也加上边框。

```
.goods-price {
  @include outer-table-borders();
  @include inner-table-borders(1px);
}
```

参数格式跟outer-table-borders mixin 一样， 这里我们让内部单元格之间的边框窄一点，取个 1px，

刚好利用这个示例给大家演示一下局部变量的用法。

我们通过变量定义table单元格的一个基准底色值， 给它一个浅紫色 #7a98c6

```
$table-color: #7a98c6;
.goods-price {
  @include outer-table-borders();
  @include inner-table-borders(1px);
}
```

但其实这里定义的这个变量， 只可能我们table内部用， 怎么把它变为一个更易于管理的局部变量呢， 只需要把它挪到 规则大括号里边即可。 这样只有括号内才可以引用这个变量。

```
.goods-price {
```

```
  $table-color: #7a98c6;

  @include outer-table-borders();
  @include inner-table-borders(1px);
}
```

我们利用Sass的颜色函数， darken， 把 $table-color 的亮度降低 40%， 然后把这个色值赋给我们的内部边框， 以便让边框突出， 也算是给颜色函数找个应用的场景。

```
.goods-price {
  $table-color: #7a98c6;

  @include outer-table-borders();
  @include inner-table-borders(1px, darken($table-color, 40%));
}
```

好， 我们看一眼效果。

内部单元格之间现在也区分开了， 但还有几个很突出的问题， 1是单元格太过拥挤， 2是单元格内容都是左对齐的， 我们希望th标题内的文本内容居中对齐，普通文本左对齐，数值呢右对齐。3是表格内容不够突出。

mixin table-scaffolding 就负责干这样的事情。

```
.goods-price {
  $table-color: #7a98c6;

  @include outer-table-borders();
  @include inner-table-borders(1px, darken($table-color, 40%));
```

```
  @include table-scaffolding();
}
```

我们刷新看一下， 一下子就清爽了许多。 我们看一眼生成的代码:

可以看到 table-scaffolding 默认对th单元格居中显示， 字体加粗， 所有单元格加了一个2px的padding， 但是对类名为 numeric 的单元格， 让其右对齐。 这也是为什么我们的table html结构里边要额外加 numeric 类名的原因。

引用高晓松的一句名言： 人生不止眼前的苟且， 还有诗和远方。 虽然表格已经很清爽了， 但是我们不能止步， 还得追求更好。

那怎么样才能让它变得更好呢， 要想更好地突出内容， 无非就是奇偶行不同色， 纵行隔行不同色。

```
@include alternating-rows-and-columns($table-color, adjust-hue($table-color, -120deg), #222222);
.goods-price {
  $table-color: #7a98c6;

  @include outer-table-borders();
  @include inner-table-borders(1px, darken($table-color, 40%));

  @include table-scaffolding();

  @include alternating-rows-and-columns($table-color, adjust-hue($table-color, -120deg), #222222);
}
```

我们使用 alternating-rows-andcolumns mixin来实现这个目的， 它支持5个参数， 前三个属于必填参数， 分别是 偶数行的颜色， 这里我们就用我们的table-color. 然后是奇数行的颜色， 我们再给color function 找个应用场景， 用 adjust-hue function 对 $table-color, 的色相值， 在色相360度圆盘上逆时针转动120度， 应该是一个偏绿的颜色。 第三个参数是 间隔纵列的颜色差值， 什么叫颜色差值， 为了突出相邻两列的差异， 相邻的两列每隔一列， 我们会在其原来颜色的基础上去减一个颜色差值， 这样相邻两列看上去就有区别了， 这个差值不宜过大， 这里我们用 #222222， 第四个参数是header部分的颜色值， 注意这里的header说的不是thead标签， 而是 th标签， 可以留空， 默认就是白色， 第五个参数是footer部分的颜色值， 留空的话默认也是白色。

我们看一下效果。

现在看上去就有层次的多了。 这里tfoot里边的单元格之所以是这个这个颜色， 是因为， 我们给他赋了一个 even的类名。 至于生成的详细css代码， 同样也是留给大家课下实践的时候阅读。

好， Color， Print， 和Tables 子模块我们就说这么多， 谢谢大家。

第三章：实战应用 - 第六节：Utilities模块二》

大家好， 我是mater。这一节我们来看一下Utilities模块的General子模块。

首先单独引入： @import “compass/utilities/general"

相信大家肯定不需要我讲解为啥需要清除浮动了， 这算是学CSS的基本功吧。当一个子元素float之后， 如果它的高度超出了其父元素的高度， 其父元素的高度并不会相应的进行调整， 解决这个问题， 我们就称之为清除浮动， 有些人也称之为闭合浮动。

清除浮动有很多种方式， 最简单的方法就是将父容器的overflow 设为hidden，直接调用

clearfix 就是采用的这种方式。

```
.clearfix {
  @include clearfix;
}
```

balaba。

zoom属性， 是为了兼容IE6。

但这种做法存在一个问题， 就是没办法利用margin 负值等构造float子元素悬挂在父元素外边的效果， 因为一旦超出父容器的边界， 就被 overflow: hidden 了。

如果有这种需求， 我们常常会采用伪类的方式来清除浮动， pie-clearfix mixin 就是这么干的。

```
.pie-clearfix {
  @include pie-clearfix;
}
```

balabala

但低版本的浏览器，display: table 这个属性可能会不支持， 所以就有了稍微复杂一点的方法， legacy-pie-clearfix mixing.

.legacy-pie-clearfix {

  @include legacy-pie-clearfix;

}

balabala

浮动界还有另一个非常经典的bug： 我们称之为 Doubled Float-Margin Bug, IE兼容的少的同学可能就不知道了， 在IE6下边， 子元素相对于父容器float的时候， 挨着父容器边界的子元素如果有margin的值， 比如说 float:left, margin-left: 100px, 渲染出来这个margin会被拉大到200px， 别问为什么会这样？ 因为这是IE6. 特别赞的是 Steve Clason发现了一种fix方法， 特别的有意思， 就是给这个float的子元素添加一个 display:inline 的属性，w3c 的规范定义float的元素， 会自动变为block的元素， 不管你的display值是啥， 所以加了这个display:inline 并不会改掉子元素的display特性， 却神奇地fix了IE6的这个bug。

所以现在如果我们要写兼容IE6的float布局，就得写成

演示

pull-left {

    balabala

}

不兼容IE6的话， 则只需要写

balabala

略麻烦哈， 所以Compass帮我们封装了 float mixin， 它会根据我们在Browser Support中的配置， 来决定是否添加上 display:inline hack方式。

.pull-left {

```
  @include float(left);
}
```

balabala， 为什么没有我说的 display:inline 呢?

之前介绍Browser Support模块时我们只说了两个配置项， 这里呢就必须多说一点了。

$supported-browsers; 用来声明我们想支持哪些浏览器， 默认值值为一个很丰富的list，我们不去修改这个配置项。

$browser-minimum-versions: ("ie": "6"); 声明Compass支持的浏览器的最小版本。值为一个map数值类型。很像js里边的object， 由键值对组成， 用一个大括号括起来。

如果不在这里边指出最小版本的话，Compass就会采取另外一种策略来决定一个浏览器是否应该支持。

大家都知道有一个著名的查看一个CSS属性或者属性value在某个浏览器的某个版本上是否被支持的网站叫 caniuse.com，同时它上边也会统计各个浏览器各个版本的某个CSS属性的使用率。

Compass在安装或者更新的时候， 会同时放一份当时caniuse的各个浏览器版本各个CSS属性使用率的数据在本地， 以后除非Compass重新安装或者更新， 否则本地这份数据都不会随着时间变化而更新了。

在Compass这里呢， CSS属性被分为了两类， 一类是即使不兼容，不去hack， 在不支持它的浏览器上渲染也不会出大的问题，无非就是不够美观的属性，比如说 border-radius, box-shadow 这种属性， 仅是一个锦上添花的作用， 就算没有圆角， 没有阴影， 我们的布局也不至于彻底乱掉。

另一类是如果不兼容， 不去hack， 在不支持它的浏览器上渲染会出大问题， 可能导致整个布局都乱掉的属性， 比如说box-sizing, clearfixing 相关的属性， 直接影响布局， 必须慎重对待。

这两类属性分别对应Browser Support模块中的另外两个配置变量：

$graceful-usage-threshold：对应我们说的第一类属性， 默认值为0.1，这是一个百分比的值， 意思是 如果我们支持的浏览器，在其某一版本上， 第一类属性的使用率达到 千分之一，就是1000个人里边有1个人在这个版本的浏览器上用到了第一类属性， 那这个第一类属性在Compass编译输出的时候就得考虑兼容这个浏览器版本。 低于千分之一， 则可以忽略不管。

$critical-usage-threshold 对应我们说的第二类属性， 默认值为0.01， 同样也是百分比的值， 意思是 如果我们支持的浏览器，在其某一版本上，第二类属性的使用率达到 万分之一， 那这个第二类属性在Compass编译输出的时候就该考虑兼容这个浏览器版本了。 低于万分之一， 则可以忽略不管。

大家想想，我们这里的float应该归为哪一类属性， 很明显是影响布局的第二类对吧， 应该采用 $critical-usage-threshold 阀值， 那第二类属性在IE6上的使用率如何呢？ 我们可以通过调用函数 omitted-usage 看到数据，

@debug omitted-usage("ie", “6");

我们可以看到返回的值为 0.0093 这个值是小于 0.01 的， 所以， Compass认为这里不需要为IE6 兼容hack。

但是， 我们都清楚， IE在国内的市场占有率官方数据现在还高达 1.91%， 我猜真实数据要在5%以上， 因为我已经很久没有兼容IE6了， 只是猜测哈。所以有时候还是得兼容。怎么做呢？

我们只需要声明 ie的最小支持版本为IE6即可。 这个时候Compass就不会根据属性使用率去判断IE6是否需要兼容支持了， 而是乖乖的按照我们browser-minimum-versions的声明去考虑兼容输出。

$browser-minimum-versions: ("ie": “6”);

$browser-minimum-versions 的默认值， 个个浏览器后边跟的都是一个null值。

类似于 ("chrome": null, "firefox": null, "ie": null, "safari": null, "opera": null)

我们再看一眼， balabala

这块我当时故意没展开， 就是因为太容易把大家听迷糊了， 但这里碰到了， 不得不讲了。

我们也可以省略这里的参数， 直接调用 float-left, float-right mixin.

```
.pull-left {
  @include float-left();
}
```

接下来看 General字模块的 Hacks 子子模块。 其出发点是做对特定浏览器hack方案的mixin集合，但其实我们都知道，无非就是hack ie低版本吗。

之前碰到IE的 hasLayout 特性，我们都一语带过了，首先要知道hasLayout是IE独有的属性。 在IE中， 触发一个element的hasLayout属性， 可以让其自己承担自己的布局， 大小，位置， 以及其子元素的布局，而不是依赖于父级元素，容器。有些Element元素，本身其hasLayout就是被设为true的， 比如说table， img等元素， 但有些则不是。 没有为全部元素打开这个属性， 微软给出的理由是： 基于性能和复杂度的考虑。

这样就会导致很多时候布局错乱的问题， 解决的方案就是通过设置能开启hasLayout属性的CSS属性， 包括 display: inline-block

height: (any value except auto)

float: (left or right)

position: absolute

width: (any value except auto)

zoom: (any value except normal) 等

我们最常用的方式就是 利用zoom属性， 设置 zoom:1 这样即激活了hasLayout属性， 又不会对原来的CSS样式产生影响。

这里的hack也主要是针对hasLayout来的，即针对IE6， IE7， 因为在IE8以后的版本中，hasLayout属性已经被废弃。

我们通过调用 has-layout mixin 来激活响应元素的hasLayout 属性，

```
.need-has-layout {
  @include has-layout();
}
```

在CSS属性前加*号， 是因为只有IE6和IE7才能识别星号， 其他浏览器则选择忽略， 避免这个属性在非IE6， IE7的浏览器下被处理。

has-layout 支持参数指定是通过哪种方式来激活元素的hasLayout, 默认参数就是我们使用的zoom， 还有block的方式。

```
.need-has-layout {
  @include has-layout(zoom);
}
```

```
.need-has-layout {
  @include has-layout(block);
}
```

这种方式也是激活相应元素的hasLayout属性的方式， 但是没啥意义， 用 zoom的方式就挺好， 简单直观。

（网页说明）也可以通过mixin has-layout-zoom 和 has-layout-block 直接调用， 避免传递参数， 不演式了， 大家下去自己尝试一下。

bang-hack mixin官方已经不建议用了， 建议用 underscore-hack 来兼容IE6的一些属性，我这里只是举个例子哈， 一个元素的display属性，我们把它设为block， 但是希望hack其在IE6下是inline， 应该怎么做呢?

```
.underscore-hack-display {
  @include underscore-hack(display, block , inline);
}
```

balabala

是因为只有IE6能够识别带下划线的CSS属性。

看minimums 子子模块， 我们知道 min-height, min-width 这两个属性， 在IE6下也是不支持的， hack方法也是使用 下划线属性， compass帮我们封装了 min-height, min-width mixin， 看一下效果。

```
.test-min-height {
  @include min-height(10px);
}
```

balabala

Reset 模块我们是最早介绍的， 不再赘述。

Tag Cloud：标签云： 一般来说长这个样子。 http://media.nngroup.com/media/editor/2012/11/18/wordle-word-cloud-applications.png

Compass封装了一个简单的，实现出来是这个样子， http://www.webdots.be/sites/default/files/field/image/tagcloud.gif 略朴素哈。

对 标签云 容器 调用 tag-cloud mixin， 需要指定基准 字体大小。一般tag要大一些， 我们这里设定基准字体

大小为24px。

```
.tag-cloud-container {
  @include tag-cloud(24px);
}
```

balabala　extra large extra extra large

把 body @include debug-vertical-alignment 删掉。

通过生成的CSS代码， 我们能想到对应的html 结构应该是:

```
<div class="tag-cloud-container">
    <span class="s">imooc</span>
    <span class="l">imooc</span>
    <span class="xs">imooc</span>
    <span>imooc</span>
    <span class="xl">imooc</span>
    <span class="xxs">imooc</span>
    <span class="xxl">imooc</span>
  </div>
```

balbalabala 要想更丰富多彩， balabala

好， General 子模块我们就说这么多， 下一节我们来学习重中之重的Compass合图。

第三章：实战应用 - 第六节：Utilities模块三》

大家好， 我是mater。这一节我们来看一下Utilities模块的最后一个子模块。重中之重，大家一定要仔细听哈。

Sprites

精灵图合图相关的工具集合。 可以这么说，Sprites本身在Web开发中就是一个最佳实践， 但在Compass之前， 合图一直都是一个令前端同学无比头疼的问题，想想当设计人员将sprites中的一个图标改了哪怕一个像素，只要影响到了其他的图标，开发人员就会吐血的修改所有被影响到的样式代码。

而现在一切都将变得如此简单， 我们只需要把需要合图的图片放在一个目录中， 在Sass文件中将其引入， 然后就可以轻松随意地使用合图了， 任由这些图片再怎么改， 我们都不需要修改我们的代码结构了。美中不足的是现在合图只支持png图片， 但我觉得这其实不算啥问题了。

我习惯的做法是把图片，icon相关的css文件， 单独放一个文件。

我们新建一个 _icons.scss, 将其引入进来。注意文件名以下划线开头。

然后在icons.scss 中将sprites模块单独引入进来： @import “compass/utilities/sprites"

将我们已有的八张logo图片放到 images目录下的 logo 子目录， 以便将来和其他图片区分开。

在Sass中通过import将我们的图片引入进来。

@import “logo/*.png";

import 后边跟我们的图片路径， 这里使用*号通配符， 引入所有的logo目录下的png图片。

现在我们只需要调用 mixin， all-logo-sprites 即可。

@include all-logo-sprites;

这个mixin是从哪来的呢？ all - sprites 中间是我们的目录名， 我们等下说。

编译看一眼，

首先是我们的images目录下多了一个 logo-**.png 的图片， 这个图片就是Compass帮我们生成的合图， 后边跟的这串字符是Compass为合图文件添加的cache-buster, 只要我们的图片不变，合图配置项不变， 这串字符就不会发生变化， 能够有效利用CDN缓存和便于我们更新图片。

然后我们再看一眼生成的css文件， Compass 根据每张图片的名字， 自动帮我们生成了对应的类名，

```
logo-sprite, .logo-analyzer, .logo-equation, .logo-music, .logo-search, .logo-shape, .logo-superimposed, .logo-translate, .logo-write {
  background-image: url('../images/logo-s1ecd90d7e9.png');
  background-repeat: no-repeat;
}

.logo-analyzer {
  background-position: 0 0;
}
```

同时自动计算好了每个类应对应的图片的位置， 帮我们生成相关的图片引用CSS。

回过头来我们看我们调用的mixin， all-logo-sprites， 这个mixin是哪来的呢？

我们上边这句 @import “logo/*.png”; import不是一个普通的sass或者compass模块， 我们import的是一系列图片文件， 这种做法在Compass中被称为 Magic Imports, Compass会根据这句import， 帮我们生成一个sass样式文件， 只不过这个文件默认是不写到硬盘上的， 同时这个sass样式文件会被当做一个普通的文件在我们import的位置被import进来。 这就是它为啥被称之为 Magic imports的原因。

诺兰有部非常著名的片子叫做《致命魔术》，里边有句非常著名的话是： 每个令人惊叹的魔术背后都有一只冤死的小鸟。

接下来我们就为这只冤死的小鸟申一下冤。

我们可以通过指令

compass sprite “images/logo/*.png"

将这只没机会被写入硬盘的小鸟输出出来。

可以看到我们的sass目录下多**

看一眼这个文件。

里边有N多的变量配置项， 以及定义的mixin。 最后一个被定义的mixin就是我们刚刚调用的all-logo-sprites mixin。

这里边的变量和mixin， 默认情况下是根据我们的图片所在文件夹的名字来命名的， 比如说这里就是

all - sprites 中间是我们的目录名

logo- sprites 最前边是我们的目录名

如果我们的目录结构是一个嵌套结构， 比如说我们有多套不同的主题：

@import "themes/logo/*.png";

Compass在生成这些配置变量和mixin的时候， 也只取这个路径里边的最后一个目录名，所以这里还是 all-logo-sprites ， logo-sprites 等等

看一下变量， 变量的定义方式跟我们常用的方法不一样， 在值的后边跟了一个 !default, 这是可配置变量常用的方式， 我们前边也已经用过配置变量， 给变量赋值的时候加一

个！ default， 就是告诉Compass， 如果在这句的前边， 这个变量已经有值了， 拿我们的$logo-sprite-dimensions 举例，

$logo-sprite-dimensions: true;

@import "themes/logo/*.png";

@include all-logo-sprites;

我们在import之前， 将 $logo-sprite-dimensions: 设为true， 那这里的 $logo-sprite-dimensions : false !default; 这个false值就不会被设上去了。 如果这个变量没有被提前配置设定， 那则使用这里的赋值 false。

$logo-sprite-dimensions 用来控制输出CSS的时候， 是否根据图片大小， 对我们的相应类名CSS属性添加宽高。 这里设为true， 我们看一下。

balabala

合图用起来也非常的简单， 只需要给对应元素设定生成的对应的类名即可。比如说 logo-analyzer, logo-music

balabala

我们现在使用图片需要用对应的类名才行， 比如说 logo-analyzer, 那不叫这个名字的元素如果想用这张图片怎么办呢?

比如说类名为 .main-logo 的元素， 也要用这张图片。

```
.main-logo {
    @include logo-sprite("analyzer");
}
```

则可以通过调用 logo-sprites mixin 实现我们的目的， 参数传我们要使用的图片的图片名。

balabala

这些mixin和变量就不一一说了，

所有的配置项在 http://compass-style.org/help/tutorials/spriting/customization-options/ compass的站点上都有详细说明， 感兴趣的同学可以下来自己浏览一遍。

对于btn类的背景图片，我们希望其在鼠标hover， active等的时候使用不同的背景图片。

拿analyzer.png 这张图片来举例， 比如说它hover状态下是图片， analyzer_hover.png, 我们这里简单的把图片拷贝，复制一下。 active状态下是图片， analyzer_active.png,

balbala

只要我们按照这个模式_hover _active 命名我们的图片文件， Compass在帮我们合图的时候就会自动生成相应的 hover, active 的伪类修饰的样式， 也可以用中划线， 我们看一眼。

balbabala

main-log 引用的地方也一样。

同时其也支持CSS3中新增的target伪类这么搞， 这里不演示了， 大家下去自行实验。

如果不想按照这种方式来命名我们的hover， active图片也没问题， 使用

$disable-magic-sprite-selectors:true; 将 这个特性关闭即可。

balabala

analyzeractive, analyzerhover

我们注意到合成的图默认是纵向排布的， 可以是通过修改 配置变量 $icon-layout 来修改合图的布局方式：

$logo-layout: horizontal;

balabala

$logo-layout: diagonal;

balabala

$logo-layout: smart;

balabala

好， 利用 utilities 合图大概就说这么多， 主要是mixin和变量配置， 其内部主要还是依赖于我们的 Sprites Helper 里边的helper function。

balabala。

所谓授人以鱼不如授人以渔， 不用我说大家都知道是哪个🐟吧。 希望大家课下能自主学习一下 sprite helper。

我利用 sprite helper 封装了一个 Compass插件， 在输出合图的时候， 自动适配高清retina屏， 这里跟大家演示一下怎么用， 源代码会放在课后的课件中。

把我们的插件代码 _retina-sprites.scss 拷贝到sass目录， 在_icons.scss 中将其引入进来。

@import "retina-sprites";

在images目录下新增一个 retina-version 目录， 用来放我们需要适配高清屏的图片。

在其下再分别创建两个目录， sprites 和 sprites-retina， 一个放1倍图片， 一个放2倍的高清图片。

分别放两个示例图片进去，一个小日历图标， 一个文件图标， balabala 。

利用 sprite-map helper function 分别生成非高清， 和 高清的 sprite map， 并赋值给 $sprites 和 $sprites2x 变量。 sprite-map的作用就是根据参数指定的图片目录，生成合图， 并返回合成后的图片路径。

$sprites: sprite-map("retina-version/sprites/*.png");

$sprites2x: sprite-map(“retina-version/sprites-retina/*.png");

在我们的插件中会用到这两个变量， 拿到合成后的图片路径。 然后利用其它的 sprite helper 进行相关布局。

通过调用插件中封装 retina-sprite mixin来进行最终使用。

```
.ico-file {
    @include retina-sprite(file);
}

.ico-calendar {
    @include retina-sprite(calendar);
}
```

参数就是我们的 图片文件名。

balabala

hover balabala

好， 关于Compass合图呢， 我们就说这么多， 是不是感觉世界一下子美好了许多呢? 谢谢大家。

第四章：课程总结

大家好, 我是mater, 很高兴跟大家一起分享了一些Sass, Compass相关的知识和经验, 相信通过我们系列课程《Sass和Compass必备技能》大家已经清楚认识到了Sass，Compass和CSS这三者之间的关系。

我之前就说过用工具， 一定要用好工具。 用好工具的基础是对工具有足够的了解并加以练习。而这正是我们课程的目的之所在。

相信随着课程的结束， 大家也已经掌握了一下技能：

1。如何使用Sass和Compass写出更优秀的CSS。

2. 痛殴CSS编写过程中，这么多年的拦路虎，比如说精灵图自动合图，属性的浏览器前缀处理。

3. 在项目的开发，生产周期内，使用的样式，图片，字体等内容，如何更好的组织起来。

但是我们课程的焦点主要是在Sass和Compass上了，

Yeoman，Grunt等前端自动化工具是怎么和Sass，Compass结合的我们却没有涉及， 不过大家不用担心， 我们会有后续课程来专门介绍， Sass，Compass是怎么与现代Web项目集成解决方案相结合的， 通过真实的实战来掌握现代web项目的开发流程。 敬请期待哦。 谢谢大家。