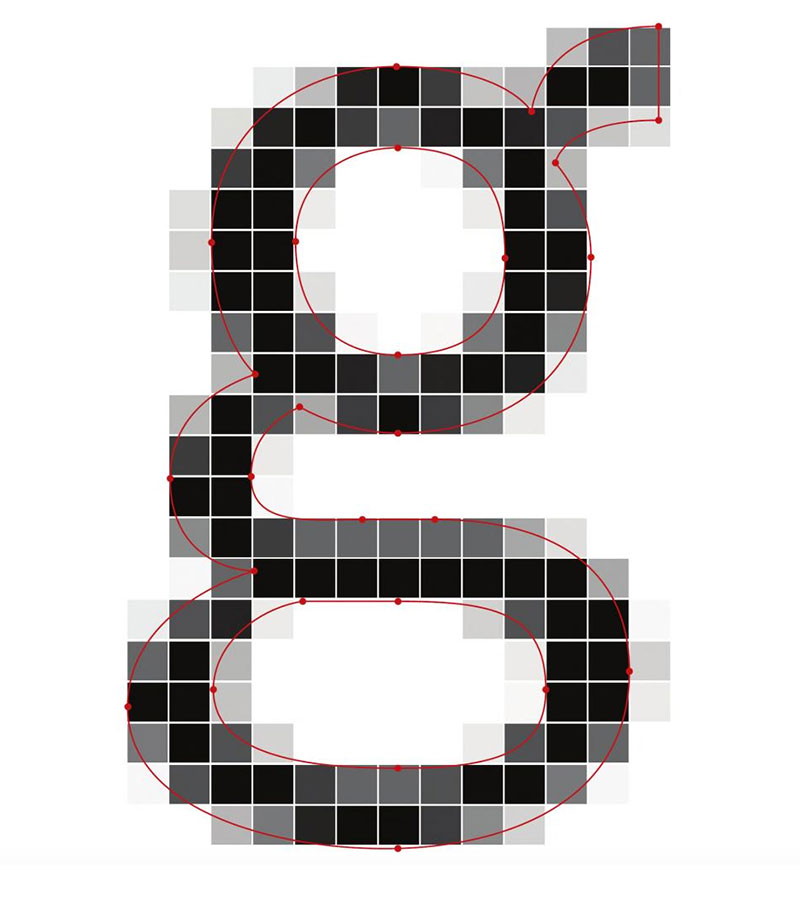
1. **网站排版简介**
2. 1第一个网站

1990年12月12日，万维网发明者Tim Berners-Lee在解决了URL、HTTP、HTML的基本概念后，发布了第一个网站。此外，他创造了浏览器并开发了软件来运行网络服务器。Tim Berners-Lee这个项目是World Wide Web ,目前这个网站还在运行。

1.2屏幕上的字体



The lowercase g in pixels with vector outlines. Design: Young Sun Compton, 2013.

90年代中期，Matthew Carter设计了Georgia和Verdana字体，这是两个在媒体上广泛应用的字体。受到微软公司委托，它们最开始被设计为位图格式（为了适应当时的分辨率），后来演化成矢量图。除了Georgia和Verdana，网络只能显示系统字体，比如Arial,Helvetica还有Times New Roman这些在所有电脑上都适用的字体。

1.3用图片来代替文本



The New Yorker’s old page using images as texts.

自从平面设计转型到互联网之后的90年代中期到00年代中期，设计师就想用更多的字型，而不是拘泥于操作系统。最简单的选择是，用图片来代替文字。用这种方法的好处之一是，设计师不用处理字体许可。设计师可以用他们电脑上有的任何字型，但缺点是每个字都必须用Photoshop之类的工具来单独切片。一个影响力比较大的用图片来代替文字的网站就是The New Yorker，跟它纸质版发行报一致，The New Yorker网站上的每个标题都是用的图片，为的就是用NY Vogue Goat作为他们的招牌字体。在2010年11月，出版物开始用Typekit作为习惯性字体之前，The New Yorker一直都有专员来负责切图工作。

1.4图像代替技术

用图片代替文字最大的一个问题是文本无法被搜索、选中或者翻译。为了避免这个问题，web从业者们提出了多种图像替代技术，来填补这个真空。2004年4月，作为web设计师和开发者的Shaun Inman开发了[Scalable Inman Flash Replacement](http://www.shauninman.com/archive/2004/04/23/ifr_revisited_and_revised) (sIFR)，用来将自定义字体嵌入小的flash视频。他还用JavaScript和css让文本变得可以被搜索。

sIFR解决切图问题的同时，也暴露了它的弱点，即依赖Flash。另外，使用sIFR要求一些web知识。2009年4月，系统工程师Simo Kinnunen发明了一项新技术：[Cufón](https://github.com/sorccu/cufon/wiki/About) (Scalable Vector Graphics, SVG)。[Cufón](https://github.com/sorccu/cufon/wiki/About)用JavaScript来扩展生成的字体。这个技术能被更简单地应用，也不会依赖flash。尽管许多图像替换技术在这些年被开发和优化，但它们都不是真正的网络排版。

1.5网络字体

网站排版并不是一个新概念。在1998年，Cascading Style Sheet(css) Working Group 提议支持@font-face规则使任何字型都可以在网页上展示出来。IE4最先实施它，但是失败了。这个提议没有盗版保护或许可协议。结果，@font-face被拖延了近十年。

2008年，@font-face东山再起，Apple Safari和Mozilla Firefox都在实施它。到了2009年5月，Jeffrey Veen引进了Typekit，一个字体服务站点，可以让设计师们在网站上使用高精度的字体，非常简单并且不用担心版权和浏览器兼容性问题。两年后，Adeobe收购了Typekit并且带来了更多经典字体，比如Garamond Pro,Minion Pro,还有Myriad Pro.

2010年，谷歌开启了自己的字体图书馆让世界各地的设计师免费使用。由于它API使用的简便性，谷歌成功地让网络字体更加易于获取。

现在@font-face规则已经被所有现代的浏览器支持（包括IE、火狐、谷歌、Safari、Opera）和手机浏览器（IOS Safari、Android、谷歌）。除了Typekit和Google Fonts外，还有很多字体库，包括Font Bureau,FontShop,Hoefler&Co.,还有Webtype,开始提供网络字体服务。在几年之类，网络字体设计已经横扫全世界。

随着响应式网站设计的发展，排版正在经历从未有过的新一轮的改革。不像纸质出版物，网站的灵活性让设计者们无法控制他们的作品。不管是通过智能手机，手写板，笔记本电脑或游戏机，他们都无法确定自己的作品会如何呈现在用户的设备上。为了适应市场上持续增长的设备数量，他们必须接受网站流动性并且放弃像素精准控制的想法。为未知的效果设计虽然会让人有些恐慌，但是也让网站设计充满了挑战和刺激。

1. **使用网络字体**

在研究排版细节之前，我们先来看一下在网页中使用字体的几种方法。用@font-face, 你可以自主地在自己的服务器上使用字体。这里是基本的css语法：

**@font-face {**

**font-family: 'Lora';**

**font-style: normal;**

**font-weight: 400;**

**src:**

**url('lora-regular.woff2') format('woff2'),**

**url('lora-regular.woff') format('woff'),**

**url('lora-regular.ttf')format('truetype');**

**}**

Font-family: 声明字体名称

Font-style: 指定样式（normal or italic）

Font-weight: 定义字体的粗细

Src: 通知浏览器字体的地址，有两个元素

Url: 提供字体文件的路径

Format: 支持特定的浏览器。比如，woff2格式只被Chrome和Opera支持。Woff和truetype被几乎所有浏览器支持。因为woff2文件大小少30%，所以被放在最前面。相反，truetype有最广泛的支持，所以放在最后。

**2.1 网络字体服务**

尽管你可以在服务器上自主使用字体，但使用网络字体如Google Fonts和Typekit有个优势就是，它们关注后台的所有工作。你无需担心字体最优化，浏览器兼容性，稳定的性能还有许可协议。下面是一个从Google下载字体的例子：

<linkhref='http://fonts.googleapis.com/css?family=Lora:400,700,400italic,700italic' rel='stylesheet' type='text/css'>

如果你用的是Typekit, 这是你需要的代码：

<script src="//use.typekit.net/XXXXXXX.js"></script>

<script>try{Typekit.load({ async: true });}catch(e){}</script>

**2.2 网络字体载入**

Link 元素是使用网络字体最简单的方法，但不是最理想的方法。因为字体样式在页头被引用，它们必须被加载完后才能显示出来。在很多浏览器中（如Chrome,Firefox,Opera,Safari），自定义字体加载之后，文本才会被渲染。结果就是，用户会经历不可见文本的闪烁（***FOIT：flash of invisible text***）.在IOS Safari中，用户会看到一个空白页多达30秒，取决于网速和网站自定义字体的大小。IE是唯一一个会先显示备用字体类型然后当字体加载完毕再切换到自定义字体的浏览器。这样做的结果是用户会经历显眼的文本无样式闪烁（***FOUT: flash of unstyled text***).

为了提供跨浏览器最好的体验，建议使用Web Font Loader异步实现字体，这个JavaScript方法只在需要的那个瞬间加载。用户第一次访问网站时可能会看到FOUT,但只要字体资源被下载和缓冲，他们的上网体验就不会被打断了。Web Font Loader是Google和Typekit联合开发的JavaScript库，但它能从其他服务商载入字体，包括Fonts.com和Fontdeck. 通过Web Font Loader异步载入Google fonts的代码如下:

<script type="text/JavaScript">

WebFontConfig = {

google: { families: [ 'Lora:400,700,400italic,700italic:latin' ] }

};

(function() {

var wf = document.createElement('script');

wf.src = ('https:' == document.location.protocol ? 'https' : 'http') +

'://ajax.googleapis.com/ajax/libs/webfont/1/webfont.js';

wf.type = 'text/javascript';

wf.async = 'true';

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

s.parentNode.insertBefore(wf, s);

})(); </script>

<noscript>

<link href='http://fonts.googleapis.com/css?family=Lora:400,700,400italic,700italic' rel='stylesheet' type='text/css'>

</noscript>

Script标签中的代码是用来实现异步，阻止页面渲染。Noscript标签中link元素是为了保证在JS被关闭或加载失败的情况下，仍然可以，自定义字体仍然是有效的。

因为异步被设为true（wf.async = ‘true’;），会发生FOUT.如果备用字体的字高跟自定义字体相差很大，FOUT会中断阅读的流畅性。为了减少FOUT闪烁的次数，你可以用CSS来为备用字体设置样式，来适应自定义字体的大小。Web Font Loader提供三个css类：.wf-loading（在自定义字体之前加载），.wf-active（在自定义字体之后加载），.wf-inactive（如果自定义字体加载失败）。以下是为备用字体设置样式的例子：

.wf-loading p {

/\* styles for fallback fonts \*/

}

.wf-active p {

/\* styles for custom fonts \*/

}

.wf-inactive p {

/\* use fallback fonts \*/

}

学习更多关于Web Font Loader，下载文档和资源代码<https://github.com/typekit/webfontloader>

**2.3字体监听**

虽然Web Font Loader解决了FOIT的问题，但当自定义字体换出系统字体时仍然会有闪烁，特别是当用户刷新网页时。此外，Web Font Loader的js库需在要文档头部被引用，增加了额外消耗。Font Face Observer，是由Typekit工程师Bram Stein开发的很新的delivery mechanism，用JS的scroll事件来加载和监听网络字体。跟Web Font Loader不同，Font Face Observer可以在文档底部被引用，减少了额外的消耗。网络性能专家Scott Jehl认为，目前Font Face Observer是最快的delivery mechanism. 下面是从Jehl那里整理的标记：

**<!DOCTYPE html>**

**<!--#if expr="$HTTP\_COOKIE=/fonts\-loaded\=true/" -->**

**<html lang="en" class="fonts-loaded">**

**<!--#else -->**

**<html lang="en">**

**<!--#endif -->**

**<head>**

**<meta charset="utf-8">**

**<title></title>**

**<meta name="viewport" content="width=device-width, initial-scale=1">**

**<style>**

**@font-face {**

**font-family: 'Lora';**

**src: url('lora-regular.woff2') format('woff2'),**

**url('lora-regular.woff') format('woff'),**

**url('lora-regular.ttf') format('truetype');**

**font-weight: 400;**

**font-style: normal;**

**}**

**@font-face {**

**font-family: 'Lora';**

**src: url('lora-italic.woff2') format('woff2'),**

**url('lora-italic.woff') format('woff'),**

**url('lora-italic.ttf') format('truetype');**

**font-weight: 400;**

**font-style: italic;**

**}**

**body {**

**font-family: serif;**

**}**

**.fonts-loaded body {**

**font-family: Lora, serif;**

**}**

**</style>**

**</head>**

**<body>**

**<p><!-- paragraph text --></p>**

**<script src="fontfaceobserver.js"></script>**

**<script>**

**(function( w ){**

**if( w.document.documentElement.className.indexOf( "fonts-loaded" ) > -1 ){**

**return;**

**}**

**var font1 = new w.FontFaceObserver( "Lora", {**

**weight: 400**

**});**

**var font2 = new w.FontFaceObserver( "Lora", {**

**weight: 400,**

**style: "italic"**

**});**

**w.Promise**

**.all([font1.check(), font2.check()])**

**.then(function(){**

**w.document.documentElement.className += " fonts-loaded";**

**});**

**}( this ));**

**</script>**

**</body>**

在上面的代码中，先判断自定义字体是否已被加载。初始化之后，字体被缓存在浏览器中；因此，你会想在用SSI（server side includes服务器端包含程序）之前，添加fonts-loaded类。

**<!--#if expr="$HTTP\_COOKIE=/fonts\-loaded\=true/" -->**

**<html lang="en" class="fonts-loaded">**

**<!--#else -->**

**<html lang="en">**

**<!--#endif -->**

这是SSI简介的链接：

<http://www.htmlgoodies.com/beyond/webmaster/article.php/3473341>

接下来，你用@font-face在css中来加载自定义字体。在页面中引用字体比在外部样式文档中引用更能提升加载速度。浏览器不必为外部文件创建一个额外的请求。

之后，为body和fonts-loaded声明系统后备字体，它们在自定义字体加载成功后会被换出。

**body {font-family: serif;}**

**.fonts-loaded body {font-family: Lora, serif;}**

然后在页面最后（</body>标签前），调用**Font Face Observer**.

**<script src="fontfaceobserver.js"></script>**

最后，下面的函数用**check()**方法来看字体什么时候加载完成，用**then()**方法来将fonts-loaded类放进HTML:

**w.Promise**

**.all([font1.check(), font2.check()])**

**.then(function(){**

**w.document.documentElement.className += " fonts-loaded";**

**});**

了解更多Jehl的方法，看他的博客文章：

<https://www.filamentgroup.com/lab/font-events.html>

这些代码只是为了让字体在你的页面上加载，这可能看起来是压倒性的，但是你只需要关注不同的选择。你选择的方法取决于你web开发的段位。用你觉得最顺手的那个。在接下来的章节中，我会用基础的delivery让例子简单。如果你想更深入地学习，去看Zach Leatherman的文章：

https://www.zachleat.com/web/comprehensive-webfonts/

1. **选择正文**

在大多数我设计过的网站中，我总是从正文文字开始。因为人

们用大多数时间来阅读内容---至少他们希望自己会这样---所以你会想为他们提供流畅的阅读体验。字型需要不唐突、合法、易读。

在为网页选择正文文字时，我会寻找具备以下特点的字体：generous x-height, open apertures, even spacing, clear termnals, 还有 distinguishable ascenders and decenders.

## 3.1慷慨的行高

行高是对小写字母x的垂直度量。行高往往比大写字母高度的一半要大一点，即大写字母底部到顶部的距离。行高越短，字母越小，会导致字体无法辨认。同样地，行高太高也会造成这个问题。比如说，一个很大的行高会导致字母n和h很难区分。理想的小写字母应该足够高来与它们的大写字母相和谐。



* 1. **Open Apertures**

Aperture是某些字母上的开口，比如c, e, 和s. Open apertures

有助于读者辨别字母。与此相反，closed apertures会影响可读性。比如，几何学的字型如Arial何Helvetica，字号小时，字母c总细小的缝隙会被误认为o。



## 3.3相等的字距

设计优秀的字型应该有相同的letter-spacing来创造稳定的阅读节奏。字母外的间距跟字母内的间距同等重要。字母之间不能太近或太远。相等的间距可以提升阅读进程。读者可以快速挑选出形状，如果间距平衡。



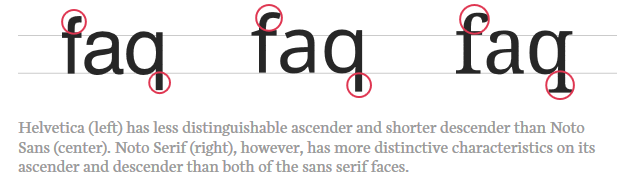
## 3.4Clear Terminals

Terminals有三种形态：ball（环形的），beak（尖形的），teardrop（球形的）。可以在字母a, c, f, j, r, y中看到它们。在很长的文本中，有清晰terminals的文本（清晰可辨的ball,beak,teardrop）更容易被认出；因此，它们比缺乏terminals的文本更容易阅读。



## 3.5可区别的Ascenders&Descenders

Ascenders指的是中间线之上延伸的笔划，descenders指的是中间线以下延伸的笔划。Ascenders和descenders越有辨识度，文字就越易读。大多数衬线字体往往是阅读的理想选择，因为它们在ascenders和desenders中明显的笔划。



## 3.6 Styles&Weights

当你浏览Google Fonts或Typekit之类的字体库时，缩小选择最快的方法之一是过滤掉少于四种样式的字体。作为正文字体的family类型，至少需要有regular, italic, bold, bold italic. 当你的字体文件中没有italic样式时，浏览器会尝试倾斜你的字型来给它一个仿造的italic. 当你的font family中没有bold样式时，浏览器会将你的字型double up使之加粗。没有真的italic和bold字体，结果会很辣眼睛。



## 3.7 Contrast

Contrast指的是一个字母笔划的粗细。高对比度的字体如Didot和Bodoni在字号大时表现很好——海报或时装广告——但是不适用于行文。在这类字体中，细笔划很难看见，而粗笔划又太突出。结果就是，粗细笔划的结合会打断阅读的流畅性。在另一端，几何字体如Arial和Helvetica也会引起相似的不好的阅读体验，因为它们的对比度太低。这些字体中固定的字母形状会让长时间阅读愚钝又无聊。为了阅读行文，应该选择中偏低对比度的字体。



## 3.8 Context

上面我们已经提及了选择正文字体的技术细节，但是了解你的字体将要使用的场景也是很重要的。你的字体需要跟设计配合得自然。把响应式设计考虑进去，你的字体在各种设备中表现得怎么样？它会在手机设备上看起来太宽吗？它会在大屏幕上看起来太窄吗？找到一种字体可以在不同的 场景和设备中平衡，来为你的读者保证一致的体验。

学习更多，阅读Tim Brown的

https://practice.typekit.com/lesson/selecting-typefaces-for-body-text/

**4. 设置浏览器字体**

一旦你缩小了正文字体的选择范围，你需要立刻在浏览器中测试它们。PS、AI还有InDesign之类的工具不会给你文字如何在网页上显示的真实感觉。此外，在字型的环境中设置字体才会有完美感。

## 4.1基础页面设置

下面是开始设置前的完整代码：

**<!DOCTYPE html>**

**<html dir="ltr" lang="en-US">**

**<head>**

**<meta charset="utf-8">**

**<title></title>**

**<meta name="viewport" content="width=device-width, initial-scale=1">**

**<link href='http://fonts.googleapis.com/css?family=Lora:400,700,400italic,700italic' rel='stylesheet' type='text/css'>**

**<style>**

**/\* CSS styles go here \*/**

**</style>**

**</head>**

**<body>**

**<p><!-- paragraph text --></p>**

**</body>**

**</html>**

我不会关注HTML文档的细节，但是我想指出跟排版有关的两个元素。第一个是viewport<meta>标签：

**<meta name="viewport" content="width=device-width, initial-scale=1">**

这个标签对响应式排版很关键。为了让你的文字在小屏幕上可读，你需要关注设备尺寸而不是关注视窗尺寸：**width=device-width, initial-scale=1.**更多细节，阅读Raymond Schwartz的<http://wdbp.net/documentation/viewport-meta-tag-and-at-viewport-css-rule/>

第二个是接入Google Fonts API的**link**元素：

**<link href='http://fonts.googleapis.com/css?family=Lora:400,700,400italic,700italic' rel='stylesheet' type='text/css'>**

为了测试，我让网络字体作为外部资源被加载，因为这会让标记脚本没那么复杂。我用Google Fonts在我所有的demos中因为这项服务是免费的而且你不用做任何特殊的设置来让字体在你本地机器上正常工作。你也可以在本地用Typekit，但你需要用MAMP或Python的SimplehttpServer组件来运行自己的服务器。对于已经预装在Mac os x中的Python,你可以用一行命令启动一个简单的web服务器：

**$python –m SimpleHTTPServer**

一旦你在kit settings中包含了“localhost”,你就可以从Typekit测试字体了，在你自己的本地web服务器上：<http://localhost:8000/>.

（https://helpx.adobe.com/typekit/using/typekit-develop-locally.html）

## 4.2段落

段落在一篇文章中占据了相当大的分量。当建立段落时，使用的是真正的内容。不像虚假的文字（lorem ipsum），真正的内容能让你更好地体会到字型会如何在你的网页上展现。有些字体在字体库中看起来很棒，但是它们不一定适合你的项目。

未完...

## 4.3Mobile First

当我已经载入了文字，便开始设置我的基础css:

**body {**

**font: 100%/1.6 "Lora", serif;**

**font-weight: 400;**

**margin: 0 auto;**

**padding: 2em;**

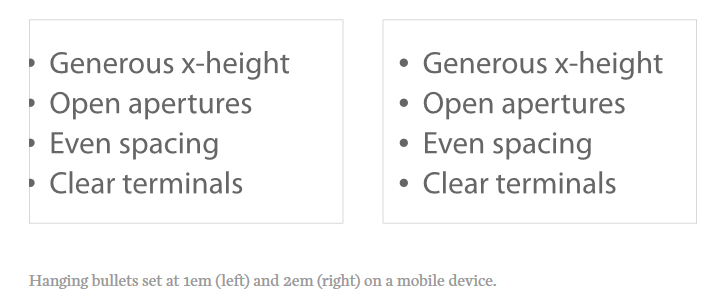
**text-align: left;**

**}**

我是mobile-frist推进的忠实粉丝，因为其中一个好处是你不必设置composition的宽度。将你的浏览器压缩到340px左右，你就可以看到文章在小屏幕上如何显示。我总是将body text的字号设为100%，即16px或1em。如果文字在小屏幕上看着太大或者太小，我可以调整段落元素（p）来获取最理想的阅读字号。

至于行高，一个合适的尺寸是1.4和1.6，但这取决于你字体的字高和文本行的长度。如果字高很大而且文本行很长，你会很想提高行距。相反，如果字高很小而且你的文本行很短，你就会想缩小行距。达到刚好的程度需要实验和用你最好的判断力。

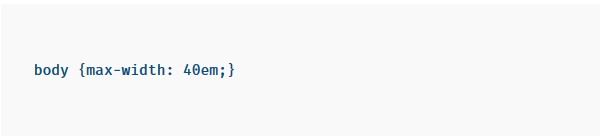
关于padding,我通常左右两侧都设置至少2em。因为我更喜欢用hanging bullet lists(<https://prowebtype.com/seeing-typographic-details/#hangingbullets>)。在小屏幕设备上，如果左内边距小于2em，hanging bullets将会接触到屏幕的边缘。



关于margin，我喜欢将左右外边距都设为auto，这样元素会在浏览器的中间显示，这在大屏幕中更有意义。Css中，font weight可以用关键字normal或bold来定义，但良好的做法是养成用数字来定义的的习惯，可设置的范围是100~900，这样能更精准地控制。一个典型的字型normal weight是400，bold weight 是700。

## 4.4Comfortable Measure fo Reading

设置正文时，为了可读性，往往一行45~75个字。过长的文本行会让浏览更困难；用户看到一行的最后时，可能不太方便定位到下一行的开头。挑战就在于找到最舒适的measure来提供流畅的预读体验。字号100%或者16px在小屏幕上很合适（宽段小于等于340px）。但在大屏幕上，你需要限制文本行的长度，用max-width设置段落或元素本身。下面是一个使用max-width在body上的例子：



## 4.5段落标识



**5. 选择标题**

## 5.1使用同样的字型

你可以用body字型的两倍或三倍来设置标题。对spacing, weight, alignment,和case（比如使用text-transform属性让标题全部为小写字母）, 甚至是color, 都进行尝试。重复使用同样的字体不仅让视觉和谐，而且减少了需要加载的字体数量，提高性能。如果你打算使用外文在网站上，那么你选择的字体需要包含特定语言子集，所以你的字体文件会非常大。一般来说，减少不必要的字体可以对网站加载速度有帮助。

## 5.2结合字型

如果你想要更与众不同的标题，你就需要找一个既能成为焦点又能补充正文的字型。大部分情况下，将两种衬线字体或两种非衬线字体组成一对并不是个好主意。有两种从根本上很相似但又有不同之处的字体看起来很笨拙。

推荐使用的其中一个方法是，保证字体是不同的类（e.g. sans, serif, slab），但是又相似的结构。比如，如果你使用FF Meta Serif Web Pro 作为正文字体，你可以用FF Meta Web Pro作为标题。另一个建议是使用同一个设计师的字体。你可以结合Egyptienne和Univers因为两种字体都是Adrian Frutiger设计的，它们有同样的结构。关于您正在阅读的这本书，正文是Adobe Caslon Pro，设计师是Carol Twombly，标题是Myriad Pro，设计师是Twombly和Robert Slimbach。请记住结合使用字体没有任何规则，全凭你的直觉来找到最佳平衡。

## 5.3为标题使用em

为标题写CSS样式时，比起px我更爱使用em. Em是font-size属性的一个计量单位。比如，如果我设置body字号为100%（或16px），那么1em等于16px，2em等于32px。对标题使用em的其中一个好处是，如果我需要改变正文字号，只需要改一个地方。比如，如果我将正文字号从原来的100%（16px）改为83.1%（13px），标题会等比例缩小。我理解有很多设计师选择px因为他们更容易计算出字号大小。转换px为em并不需要做数学，<http://pxtoem.com/>这是个很好的参照网站。如果你用SASS,下面是可以做转换的函数：

**@function calc-em($target-px, $context) {**

**@return ($target-px / $context) \* 1em;**

**}**

**h1 {font-size: calc-em(32px, 16px);} // Outputs 2em**

学习更多关于em，https://blog.typekit.com/2011/11/09/type-study-sizing-the-legible-letter/

## 5.4 Modular Scale

你至少需要六种规格的标题（h1~h6），不管你是否会使用它们。如果你设计了一个排版系统用于CMS(内容管理系统)，保险的做法就是将它们全部定义因为你不知道哪个规格的标题会被CMS的操作者选中放在WYSIWYG编辑框中（所见即所得）。你不会想要你的标题依赖于浏览器默认设置，无视你的层级。

为了选出合适的标题尺寸，我经常用em在css中，变改em值边刷新网页，来看效果。

**6. UI选择的字体**

**7. 看排版的细节**

**8. 练习排版**

**结语**