

# CartoCSS中文指南

## 前言

这是一份根据[Mapbox Studio的在线帮助文档](#)、[TileMill的帮助文档](#)以及CartoCSS语言的开源解释器项目[carto](#)的文档（这也是Mapbox官方的CartoCSS Reference链接指向的文档）翻译、整理并适当补充和修订之后的中文版CartoCSS指南与语言参考手册。翻译整理的目的是为了方便越来越多的使用CartoCSS进行制图的中文用户了解这种制图样式语言中各种属性的含义及用法。读这份文档需要具有一些的地理信息系统（GIS）、地图制图学（Cartography）方面的背景知识。由于CartoCSS是为[Mapnik](#)而设计的，而且其脚本最终会被解译为Mapnik样式并进行制图渲染，所以如果知道并了解Mapnik的工作原理，那么会对CartoCSS的语法要素及工作机理有更深刻的理解，但这不是使用CartoCSS进行制图所必需的。关于背景知识，可以参考[Mapnik项目中的相关文档](#)。

看了前面一段，你可能已经被其中出现的除CartoCSS以外的其它几个名词搞晕了。我在这里简单解释一下它们都是什么，以及它们之间是什么关系。

- **Mapnik**是一个开源的地图渲染引擎，用C++语言开发，有Python和node.js接口。它的开发可以追溯到2005年，但直到2008年的0.5版本发布之后才真正展现出它的强大——渲染质量高，而且速度很快。Mapnik有一套基于XML的地图样式描述方法，但在描述较复杂地图样式的时候其XML样式也会变得很长很复杂，难以让人类直接阅读和修改，而这恰恰就是后来CartoCSS这种高级地图样式描述语言出现的一个重要原因。
- **Mapbox**是一个专注于数字化制图服务的公司，成立于2010年。它是目前能紧紧把互联网、云计算、移动计算和传统的地图制图设计结合起来而且结合的最好的公司之一。目前，它已经网罗了该领域中全世界能数的过来的众多牛人，这其中包括Mapnik的作者[Dane Springmeyer](#)，MBTiles的设计者[Tom MacWright](#)和[Justin Miller](#)，其中Tom MacWright也是CartoCSS的设计者，还有OSRM的作者[Dennis Luxen](#)等等，不一而足。目前，Mapbox的发展突飞猛进，产品线逐渐拉长，业务范围向企业级私有空间数据基础设施服务等方向推进。这里要强调Mapbox的一个很重要的特点，就是它维护着大量的开源项目。注意Mapbox的官方网站在中国大陆地区是被GFW禁止访问的，所以要了解更多关于它的信息请自备梯子。
- **TileMill**和**Mapbox Studio**都是Mapbox维护的开源项目，都是跨平台的制图客户端软件，都是用node.js开发的，只是后者是最近发布的，有取代前者的意思，但目前这两个软件都可以使用。在TileMill和Mapbox Studio中进行制图需要使用CartoCSS语言，制好的地图可以导出成MBTiles格式的瓦片数据集，或直接连接Mapbox账号上传到用户自己的账户空间中。
- **carto**也是Mapbox维护的一个开源项目，它是将CartoCSS脚本解析成Mapnik XML地图样式的解释器，是用node.js开发的。CartoCSS的语言参考手册就在carto项目的wiki中。

我会尽力保证这份文档与官方英文文档同步。github仓库中保存了中文版文档的markdown格式版本，以及通过[Ulysses](#)和[Marked 2](#)生成的pdf与html版本。html的在线版本可以直接从[这里](#)查看，但阅读体验并不好，所以建议阅读[gitbook上的在线版本](#)。

关于翻译的更多信息请参考[这篇文章](#)，或关注[我的博客](#)了解最新的翻译整理进展。

## 翻译整理计划

《指南》最初只是一系列有关CartoCSS的文档的中文翻译。但因为这些文档之间本身并没有很强的逻辑关系，所以我还是把它们重新进行了组织，形成了目前的结构。但是这样的组织的结果就是在各个章节之间需要加入一些承上启下的衔接内容，以及在必要的时候需要对原文进行必要的补充和内容调整。基于这些考虑，翻译整理工作会按照以下步骤进行。

1. 原文整理。将有必要列入书中的英文原文进行整理，以markdown格式放入对应章节。
2. 翻译。以段为单位进行翻译。翻译的过程中保留原文，每段译文写在原文下面，翻译的过程尽量忠于原文，但鼓励意译。如果觉得原文阐述不清或有错误，需要修正的时候，请用“译注”标出，并尽可能提交issue给官方仓库，或在邮件列表中进行讨论。
3. 校对。将译文进行整理，梳理语句、段落、章节，修订格式与错别字。
4. 增补。在章节过渡、承上启下等位置和需要其它说明解释的地方补充文字。
5. 整体审校。

因为都是利用业余时间在做这件事情，所以这些步骤暂时不设deadline，但大致的计划是在2015年4月1日前完成第一版。

## 致谢

感谢以下用户参与本书翻译：

- [Anran Yang](#)

特别感谢[Anran Yang](#)对本书进行封面设计。

## 源码仓库

本书内容以markdown格式保存在github上，仓库地址为：

[www.github.com/tumluliu/carto\\_zh-cn](http://www.github.com/tumluliu/carto_zh-cn)

## 问题与反馈

如果对翻译有任何问题或建议请到项目的仓库中[提交issue](#)，或者直接与该项目的负责人[Lu Liu](#)联系：

- 邮箱：[nudtlliu@gmail.com](mailto:nudtlliu@gmail.com)
- 网站：[luliu.me](http://luliu.me)

# 1. 概述

CartoCSS是一种语法类似CSS（Cascading Style Sheets，层叠样式表，一种对网页进行设计的样式语言）的制图样式描述语言。如果熟悉CSS的话，那么CartoCSS这种对地图进行样式设计的语言也会看起来不陌生，尽管二者所包含的要素、属性等内容和含义完全不同。

译注：把关于符号和多符号的内容挪到基础用法部分中去了，准备在这里讲一下下面这三个内容，恐怕对吸引读者更有作用。

1. 为什么要用一种脚本语言来进行地图样式设计？这是否符合目前主流的设计（in general sense）工具潮流？其

背后是不是隐藏着什么规律，无论是工业设计领域的，还是计算机软件设计领域的，还是制图设计和地理信息科学领域的一种一般性的规律？

2. 如果第一点的结论是“使用脚本语言进行地图制图设计是符合潮流与规律的先进方法”，那么该领域内是否还有其它制图脚本语言？它们的现在的状况如何？

其它制图脚本语言：MapCSS，SLD/GeoServer CSS Module，Cartagen GSS，MapServer's Mapfile，Cascadenik，etc.

3. 如果第二点中列出的其它制图脚本语言的状况不佳，那么为什么CartoCSS又有什么亮点呢？是哪些特点使得它值得我们关注？

如果上面三点阐述清楚了，那么接下来可以再介绍一下CartoCSS的诞生和发展过程。这里面就不得不提到Mapnik，这个尺度就不太好把握了。主要还是侧重于发展历程吧，即从Mapnik到Cascadenik，再到CartoCSS的发展过程。

最后，再介绍一下用CartoCSS制图与目前其它主流（比如ArcGIS，或者真正的地图出版社）制图方法及工具链的对比。这个也蛮有挑战的。

## 参考文献

1. Tom MacWright, [CSS for Maps](#), 2012.11.2
2. Konstantin Käfer, [Introducing Carto: A CSS-like Map Styling Language](#), 2011.2.9
3. [CartoCSS readme](#), 2014.7.3
4. [CartoCSS on OpenStreetMap wiki](#), 2014.10.1

# 2. 快速入门

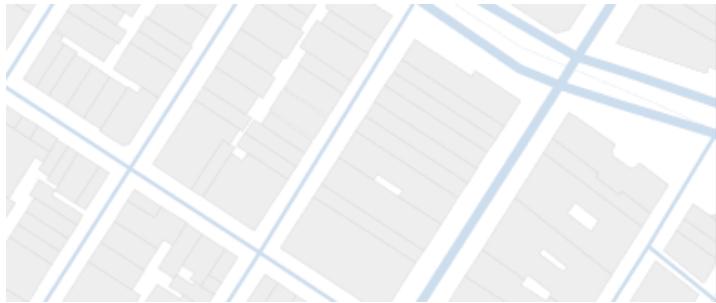
CartoCSS可以对地图中各种要素样式的细节进行控制。包括颜色、大小、形状等，都可以通过设置CartoCSS的各种属性参数来实现制图样式的配制。为了能让读者快速了解用CartoCSS能制出什么样的地图，以及如何制出这样的地图，我们在本章中以房屋、公园和道路的制图样式配置为例，初步展示一下CartoCSS的制图能力。需要说明的是，本章中的制图示例用到了Mapbox Studio。但这些制图任务并不是仅能够在Mapbox Studio中才能完成。而且这些例子需要有对应的数据准备。我们先假定所需要的数据已经准备好。

## 2.1 配置房屋样式

为了对一个表示建筑物轮廓的面要素矢量数据集进行样式配置，可使用以下CartoCSS脚本：

```
#building[zoom>=14] {  
  polygon-fill:#eee;  
  line-width:0.5;  
  line-color:#ddd;  
}
```

制图渲染效果如下图：



```
2  
3 #building[zoom>=14] {  
4   polygon-fill: #eee;  
5   line-width: 0.5;  
6   line-color: #ddd;  
7 }  
8  
9
```

图片来源: [Mapbox](#)

- `#building`标识了需要进行样式配置的图层。
- `[zoom>=14]`指定了该样式只有在地图缩放级别大于等于14级（缩放级别数字越大，则视点距离观察区域越近，反之越远。1级的时候通常为世界地图视图）的时候才起作用。
- `polygon-fill: #eee`设定了房屋多边形的填充色为浅灰。

我们还可以让房屋在更高的缩放级别上展示出具有一定高度的效果，类似于一个个积木块。为此，还需要添加这么一段代码：

```
#building[zoom>=16] {  
  building-fill:#eee;  
  building-fill-opacity:0.9;  
  building-height:4;  
}
```

修改后的制图渲染效果如下图：



```
2  
3 #building[zoom>=14] {  
4   polygon-fill: #eee;  
5   line-width: 0.5;  
6   line-color: #ddd;  
7 }  
8  
9 #building[zoom>=16] {  
10  building-fill: #eee;  
11  building-fill-opacity: 0.9;  
12  building-height: 4;  
13 }
```

图片来源: [Mapbox](#)

## 2.2 配置公园样式

这次我们试一下对公园这种类别的地块配置样式。

```

#landuse[class='park'] {
    polygon-fill:#dec;
}

#poi_label[maki='park'][scalerank<=3][zoom>=15] {
    text-name:@name;
    text-face-name:@sans;
    text-size:10;
    text-wrap-width: 60;
    text-fill:#686;
    text-halo-fill:#fff;
    text-halo-radius:1;
    text-halo-rasterizer:fast;
}

```

应用以上CartoCSS脚本可以得到如下渲染效果：



```

14 #landuse[class='park'] {
15     polygon-fill:#dec;
16 }
18
19 #poi_label[maki='park'][scalerank<=3][zoom>=15] {
20     text-name:@name;
21     text-face-name:@sans;
22     text-size:10;
23     text-wrap-width: 60;
24     text-fill:#686;
25 }

```

图片来源：[Mapbox](#)

下面逐一介绍其中用到的CartoCSS关键要素：

- `#landuse`标识了公园地块数据所在的图层，即`landuse`层。
- `[class='park']`修饰符利用条件过滤器限定了该样式的作用范围，即`class`属性值为`park`的那些面要素。
- `#poi_label`标识了用于对公园进行文字标注的图层，即`poi_label`层。
- `[maki='park'][scalerank<=3][zoom>=15]`修饰符利用了一组条件过滤器限定了该样式在`poi_label`层中的作用范围，以及满足这些条件的标注只有在缩放级别大于15级的时候才可见。
- `text-name: @name`用于设置用于标注公园的数据字段，以一个变量`@name`的形式给出，`@name`可以在之前先定义好，例如`@name: '[name_en]';`。
- `text-face-name: @sans`用于设置文本标注的字体。与`text-name`一样，这里也使用了变量，即`@sans`。
- `text-wrap-width: 60`设置了文本标注中每一行的最大长度。
- `text-halo-rasterizer: fast`指定使用一种经过优化的快速绘制方法来渲染标注文字的光晕。

## 2.3 标注道路

前面两个例子中都是对矢量数据中的面要素类型进行样式配置。在这个例子中，我们以道路为例，看看如何配置线要素标注的样式。

```
#road_label[zoom>=13] {  
    text-name:@name;  
    text-face-name:@sans;  
    text-size:10;  
    text-placement:line;  
    text-avoid-edges:true;  
    text-fill:#68a;  
    text-halo-fill:#fff;  
    text-halo-radius:1;  
    text-halo-rasterizer:fast;  
}
```

以上CartoCSS脚本将产生如下渲染效果：



```
26  
27 #road_label[zoom>=13] {  
28     text-name:@name;  
29     text-face-name:@sans;  
30     text-size:10;  
31     text-placement:line;  
32     text-avoid-edges:true;  
33     text-fill:●#68a;  
34     text-halo-fill:○#fff;  
35     text-halo-radius:1;  
36     text-halo-rasterizer:fast;  
37 }  
38
```

图片来源：Mapbox

其中涉及到的主要属性和参数的含义如下：

- `#road_label`标识了道路标注对应的图层为`road_label`层。
- `[zoom>=13]`限定了只有在缩放级别大于13级的时候才显示道路标注。
- `text-placement: line`设置标注的放置方式为沿着线的走向显示文字，而不是通常的水平显示。
- `text-avoid-edges: true`强制所有标注要避开绘制区域（例如瓦片）的边缘，防止出现被切断的情况。

## 参考文献

1. Mapbox, [Mapbox Studio Style Quickstart](#)

# 3. 基础用法

介绍CartoCSS中的基本概念与基础用法。

## 3.1 基本概念

### 符号

CartoCSS所基于的地图渲染引擎Mapnik提供了一组基本样式，基于这些基本样式可以配制出复杂的地图样式。这些基本样式被称为符号，每种符号都包含一系列属性。

目前Mapnik支持以下十种符号。每种符号都可以用于对某一种或几种类型的空间数据进行样式配置：

1. 线符号（可用于线要素和面要素）
2. 面符号（可用于面要素）
3. 点符号（可用于点要素）
4. 文本符号（可用于点要素、线要素和面要素）
5. 盾标符号（可用于点要素和线要素）
6. 线图案（可用于线要素和面要素）
7. 面图案（可用于面要素）
8. 栅格符号（可用于栅格数据）
9. 注记符号（可用于点要素、线要素和面要素）
10. 建筑物符号（译注：通常用于面要素）

需要注意的是，尽管面符号可以用于定制线要素的样式，但往往会出现不可预期的不理想结果，因此不推荐使用。

对同一个图层可以同时应用多种符号来定制样式。这种用法我们称之为多符号。常用的多符号组合包括：线符号加面符号，点符号、文本符号、线符号加注记符号，以及线符号加线图案等。

一种符号只有在明确定义了它的样式之后才能被绘制在地图上。在每种符号的诸多属性中，除了显式赋值的属性以外，其它属性将全部被赋予默认值。例如，线符号中颜色属性的默认值为黑色，所以如果用户只显式设置了线宽，那么图层中的线要素就将以用户设置的宽度被绘制为黑色线条。

### 多符号

对一个图层来说，它的样式可以不局限于仅使用单一的某种符号来定制。举例来说，为了在多边形的边界上获得一种柔和的光晕或阴影效果，可以定义多个半透明线符号，共同发挥作用达到渲染效果。再举一例，对点要素，可以通过定义多个文本符号将若干个属性字段以不同偏移的形式散布标注在点要素的周围。

通常，如果对一个图层定义了一种样式，那么这种样式就会应用于一种默认的符号。在下面的例子中，后一个样式规则就会将前一个覆盖，因为二者都应用了相同的默认符号，即线符号。

```
#layer {  
    line-color: #C00;  
    line-width: 1;  
}
```

```
#layer {  
    line-color: #0AF;  
    line-opacity: 0.5;  
    line-width: 2;  
}
```

用户可以通过显式声明的方式为一个图层增加任意数量的新符号。由这些新符号所定义的样式之间只要不互相冲突，那么它们都将被用于渲染该图层。为图层定义新符号使用双冒号`::`语法，与CSS3中的伪元素定义类似。这里所谓“新符号”也被称为“从属样式”（下文详细介绍）：

```
#layer {  
    /* 定义默认符号的样式 */  
}  
  
#layer::newsymbol {  
    /* 定义名为'newsymbol'的新符号的样式 */  
}
```

注意上面例子中的`newsymbol`并不是CartoCSS关键字。用户可以为新符号自定义名字，但是为了便于理解，最好取一些有意义的名字，例如：`#road::casing`, `#coastline::glow_inner`, `#building::shadow`。

在上一个例子中，我们可以通过再声明一个新符号来实现一个蓝色光晕效果。而正是通过增加了这个新符号的声明，使得蓝色光晕能够被叠加渲染在之前的红色轮廓线之上，而不是覆盖了前面红色线样式（如图）。

```
#layer {  
    line-color: #C00;  
    line-width: 1;  
}  
  
#layer::glow {  
    line-color: #0AF;  
    line-opacity: 0.5;  
    line-width: 4;  
}
```



图片来源: [Mapbox](#)

在对所定义的符号进行渲染的时候，是按照其在样式脚本中出现的顺序进行的。所以上面例子中的新符号`::glow`（蓝色光晕线）会被绘制在之前定义的红色轮廓线之上。

具名的新符号样式也同样会有同名覆盖问题，即后定义的新符号会覆盖之前先定义的同名符号的样式设置。在下面的例子中，线的颜色最终将被渲染为绿色（RGB值为#3F6），而不是半透明黄色上叠加一层绿色效果（如图）。

```
.border::highlight {  
  line-color: #FF0;  
  line-opacity: 0.5;  
}  
  
.border::highlight {  
  line-color: #3F6;  
}
```



图片来源: [Mapbox](#)

## 从属样式与实例

在CSS中，对象的属性只能有一个明确定义的值。比如一个

，它具有明确定义的边界宽度和颜色。CSS中对象属性的定义满足更精确匹配原则（例如#id的匹配度比.class要高），匹配度较高的属性定义会覆盖匹配度较低的。为了尽量与CSS保持相似，CartoCSS也秉承了相同的规则。但实际上，作为底层渲染引擎的Mapnik具有更强大的能力，支持更丰富的特性。

Mapnik中的图层可以有很多个**边界**等属性。这种能力在绘制带有边框的线条（例如道路线上的边框，海岸线上的光晕效果等）时非常有用。在CartoCSS中，这种能力通过定义**从属样式**（attachments）来实现。当然，从属样式不是必需的。

```
#world {  
    line-color: #fff;  
    line-width: 3;  
}  
  
#world::outline {  
    line-color: #000;  
    line-width: 6;  
}
```

从属样式实际上是从同一个数据集上创建出了隐式图层。与从属样式对应的是**实例**。实例的作用是在同一个图层样式块中创建多个符号。

```
#roads {  
    casing/line-width: 6;  
    casing/line-color: #333;
```

```
  line-width: 4;  
  line-color: #666;  
}
```

上面这种实例定义在Mapnik中的对应行为是先将道路线按照颜色#333宽度6进行绘制，然后立即用颜色#666和宽度4把同一条道路线再绘一次。而如果是从属样式，Mapnik就会先用边框的样式将整个道路图层绘制一遍，然后再按照实际线样式绘制道路。

## 变量与表达式

CartoCSS的设计受到了less.js的很多启发，支持CSS中的一些新特性。用户可以在样式表中定义变量，还可以用表达式修改变量。

```
@mybackground: #2B4D2D;  
  
Map {  
  background-color: @mybackground  
}  
  
#world {  
  polygon-fill: @mybackground + #222;  
  line-color: darken(@mybackground, 10%);  
}
```

## 嵌套样式

CartoCSS也像less.js一样支持嵌套规则。

```
/* 以下样式块将应用在所有class为.land的图层上 */  
.land {  
  line-color: #ccc;  
  line-width: 0.5;  
  polygon-fill: #eee;  
  /* 应用在#lakes.land图层上 */  
  #lakes {  
    polygon-fill: #000;  
  }  
}
```

这可以方便的将样式根据缩放级别进行分组：

```
[zoom > 1] {  
  /* 以下样式块在zoom大于1时将应用在所有图层上 */  
  polygon-gamma: 0.3;  
  #world {
```

```
    polygon-fill: #323;
}
#lakes {
    polygon-fill: #144;
}
}
```

## 字体集

通过text-face-name属性可以对地图中使用的字体集进行设置，这一设置在底层对应的是Mapnik中的FontSet。它的作用是通过定义多个字体来保证当地图中包含多种语言的文字时（比如有英文和中文），如果用一种字体渲染不出来某些字符，那么就会尝试用其它字体进行渲染。

```
/* CartoCSS样式 */
#world {
    text-name: "[NAME]";
    text-size: 11;
    text-face-name: "Georgia Regular", "Arial Italic";
}
```

```
/* 编译后的Mapnik XML样式 */
<FontSet name="fontset-0">
    <Font face-name="Georgia Regular"/>
    <Font face-name="Arial Italic"/>
</FontSet>
<Style name="world-text">
    <Rule>
        <TextSymbolizer fontset-name="fontset-0"
            size="11"
            name="[NAME]"/>
    </Rule>
</Style>
```

## 过滤器

CartoCSS支持多种过滤器类型，其中有数值类型的过滤器：

```
#world[population > 100]
#world[population < 100]
#world[population >= 100]
#world[population <= 100]
```

有常规类型的过滤器（等于，不等于运算）：

```
#world[population = 100]
#world[population != 100]
```

还有字符串类型的过滤器：

```
/* 在name字段上应用正则表达式进行匹配 */
#world[name =~ "A.*"]
```

## 参考文献

1. Mapbox, [CartoCSS](#)
2. carto Project Readme, [carto README.md](#)

## 3.2 样式选择器

在CartoCSS中，地图样式通过一系列描述样式的规则来表达。这些样式规则被模块化的组织成样式块，作用于地图上的各种要素对象。每个样式块都由一对大括号{}包围，其中包含了若干条用于描述样式的属性和值。样式选择器的作用就是指明某个样式块是作用于哪个图层，或者进一步限定这些样式在特定图层中的作用范围是哪些要素对象。因此从本质上说，样式选择器其实就是描述了样式块的作用域。

样式选择器可以有三种不同的形式：图层标识、图层类别，以及过滤器。其中过滤器还可以分为缩放级别、数值、文本和正则表达式等三种具体类型。

### 图层标识

通过图层的唯一标识（ID）来将样式块的作用范围限定在某一个或几个图层上。对于多个图层共用同一样式块的情况，应该将多个图层标识以逗号隔开。

```
#layer_name {
    // 样式描述
}
#layer_1,
#layer_2 {
    // 这里的样式将被应用于layer_1和layer_2两个图层
}
```

### 图层类别

当需要对多个图层定义样式时，除了可以采用之前提到的将图层标识全部列出的方法以外，还可以使用图层类别来实现，也就是在这些图层标识的后面都加上一个类别名后缀，还以上面的两个图层为例，可以为它们增加一个同一个类别得到layer\_1.roads和layer\_2.roads两个新的标识，然后通过对.roads进行样式定义来实现与之前同样的效果。

```
.roads {
    // 这里定义的样式会被应用到所有
    // 类别后缀为'roads'的图层上
```

```
}
```

## 过滤器

除了用图层的标识或类别来限定样式块的作用范围以外，还可以进一步用基于条件表达式的过滤器在图层内部筛选出需要应用样式的那些要素对象。这些过滤器使样式块与图层内要素对象的各种文本或数值类型的属性数据建立起关联，从而实现了条件样式。过滤器在制图过程中非常实用，举个例子：一个内容为道路网的线要素图层中通常是将各种不同等级的道路都包含在内，而利用过滤器，我们就可以为不同等级的道路配置不同的样式。

过滤器需要被置于一对中括号[]中，跟在图层选择器（标识或类别）的后面，或者是嵌套的写在一个更大的样式块中。

### 缩放级别过滤器

将样式的作用范围限制在特定的地图缩放级别上。在下面的例子中，样式块中定义的样式只在地图缩放到0级时发挥作用。0级是观察点距离地球表面目标区域最远的级别，这个级别看到的通常就是世界地图视图。随着缩放级别变大，观察点也逐渐向地球表面“推进”，位于当前视图中的部分地图比例尺不断变大，分辨率逐渐升高，就如同将相机镜头推向长焦端，因此这个过程的英文是“zoom in”，而相反的过程为“zoom out”。

```
#layer[zoom=0] { /* style */ }
```

还可以定义缩放级别的范围：

```
#layer[zoom>=4][zoom<=10] { /* style */ }
```

缩放级别过滤器支持6种关系运算符：=（等于，注意不是双等号）、>（大于）、<（小于）、>=（大于等于）、<=（小于等于）和!=（不等于）。

过滤器可以以嵌套的形式出现在样式块内部。例如，在下面这段CartoCSS代码中，线要素会在4级到10级之间被绘制成2个像素宽的红色线条，而在8级、9级和10级，线宽会被重新修正为3、4和5个像素。

```
#layer[zoom>=4][zoom<=10] {
  line-color: red;
  line-width: 2;
  [zoom=8] { line-width: 3; }
  [zoom=9] { line-width: 4; }
  [zoom=10] { line-width: 5; }
}
```

### 数值型过滤器

关系运算符还可以用于对图层中的数值型属性字段进行过滤。举个例子，在一个表示城市信息的点要素图层中，有一个用于记录每个城市人口数据的字段，那么我们就可以在该字段上应用数值型过滤器，实现“只有人口在一百万以上的城市才被标注在地图上”的效果：

```
#cities[population>1000000] {
```

```
    text-name: [name];
    text-face-name: 'Open Sans Regular';
}
```

这种数值型过滤器还可以和之前介绍的缩放级别过滤器结合使用。还以城市和人口的数据为例，将两种过滤器结合使用可以实现在不同的缩放级别上显示不同人口规模的城市。

```
#cities {
  [zoom>=4][population>1000000],
  [zoom>=5][population>500000],
  [zoom>=6][population>100000] {
    text-name: [name];
    text-face-name: 'Open Sans Regular';
  }
}
```

与缩放级别过滤器相同，数值型过滤器中也同样支持范围过滤：

```
#cities[population>100000][population<2000000] { /* styles */ }
```

## 文本型过滤器

对于文本类型的属性字段，同样也可以应用过滤器。通过使用等号运算符`=`，可以实现精确匹配，或者通过不等号运算符`!=`来得到相反的结果。与前两种过滤器不同的是，文本型过滤器关系表达式中的文本值必须要有双引号或单引号。

As an example, look at the roads layer in Mapbox Streets (the default vector tile source in Mapbox Studio). It contains a field called `class`, and each value for this field is one of just a few options such as “motorway”, “main”, and “street”. This makes it a good column to filter on for styling.

举个例子，在一个表示道路网的线要素图层中包含了一个名为`class`的文本型字段，该字段取值为“motorway”，“main”或者“street”，用于表示每条道路的类别。那么在制图过程中，用户就可以在该字段上应用文本型过滤器，从而实现对不同类型的道路使用不同的样式进行渲染：

```
#roads {
  [class='motorway'] {
    line-width: 4;
  }
  [class='main'] {
    line-width: 2;
  }
  [class='street'] {
    line-width: 1;
  }
}
```

如果要对所有不是motorway的道路进行样式配置，那么还可以使用不等号：

```
#roads[class!='motorway'] { /* style */ }
```

## 正则表达式过滤器

注意：正则表达式过滤器作为一种高级过滤功能，可能会对制图渲染性能带来负面影响。

用户还可以通过基于模式匹配的正则表达式进行过滤，正则表达式过滤器的运算符是=~。在下面的例子中，正则表达式过滤器会匹配所有class字段中以motorway开头的要素记录，像motorway、motorway\_link都会被匹配上。

```
#roads[class=~'motorway.*'] { /* style */ }
```

在上面的例子中，.表示任意字符，\*表示出现任意多次，因而.\*合在一起就表示由任意字符组成的任意长度的字符串。

## 参考文献

1. Mapbox, [Mapbox Studio Styling Selectors](#)

## 3.3 配置线样式

线样式既可以用于矢量线要素图层，也可以用于矢量面要素图层的样式配置。最简单的线样式可以只包含线宽（以像素为单位）和颜色。这两个属性的默认值分别为1和黑色。下面的例子中展示了对面要素边界进行样式配置的效果。



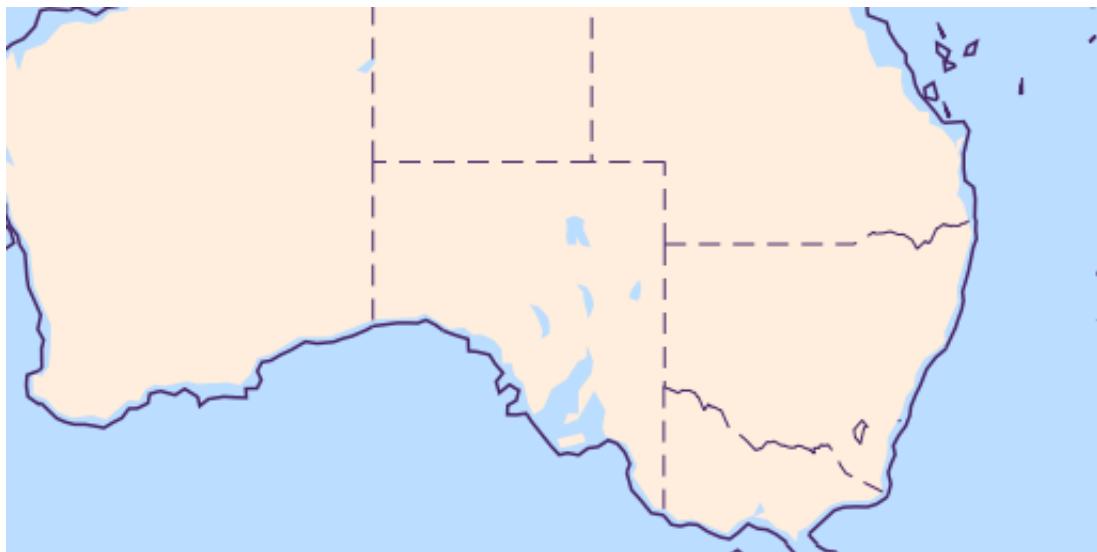
图片来源: [Mapbox](#)

```
#admin[admin_level=2] {  
  line-width: 0.75;  
  line-color: #426;
```

```
}
```

## 虚线

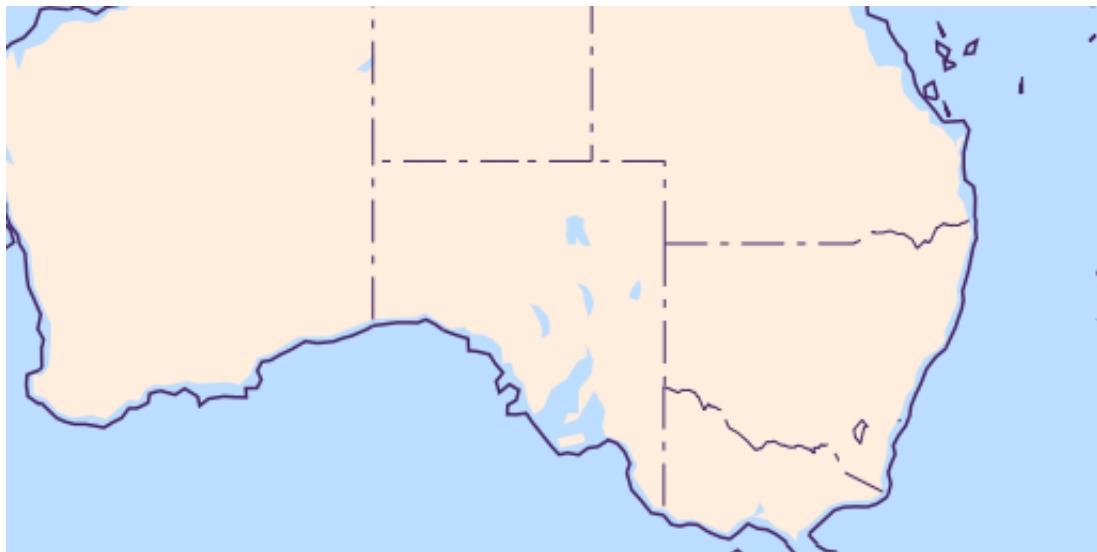
简单的虚线可以通过使用`line-dasharray`属性实现。这个属性的值是一个数值列表，列表中的元素以逗号分隔，列表中的元素交替表示虚线中短线和间隔的长度，均以像素为单位。下面例子中的虚线就是以5像素长的短线和3像素长的间隔进行绘制的。



图片来源: [Mapbox](#)

```
#admin[admin_level>=3] {  
  line-width: 0.5;  
  line-color: #426;  
  line-dasharray: 5,3;  
}
```

还可以将虚线配置成更复杂的样子，但要求`line-dasharray`属性值列表中的元素必须都得是整数。



图片来源: [Mapbox](#)

```
#admin[admin_level>=3] {  
  line-width: 0.5;  
  line-color: #426;  
  line-dasharray: 10,3,2,3;  
}
```

## 端点与交汇点

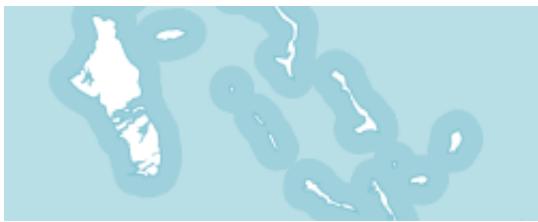
当把线宽设得比较大时，会出现一些不正常的绘制效果，例如在比较尖锐的拐角处会出现一些伸展得很长的拐点（译注：long points是不是这个意思？），还有在很小的多边形上会出现一些奇怪的缝隙，等等（如下图）。



图片来源: [Mapbox](#)

```
#countries::bigoutline {  
  line-color: #9ed1dc;  
  line-width: 20;  
}
```

要解决前一个问题，可以通过将line-join属性的值调整为round或square（其默认值为miter）。而对于第二个问题，可以通过将line-cap属性设置为round或square（默认值为butt）来填充不必要的缝隙。



图片来源: [Mapbox](#)

```
#countries::bigoutline {  
  line-color: #9ed1dc;  
  line-width: 20;  
  line-join: round;  
  line-cap: round;  
}
```

对于虚线，line-cap会被应用于所有的短线，但多出来的那部分“线头”却不会被算在line-dasharray中定义的短线长度中。在下面这个例子中，尽管在line-dasharray中定义了4个像素的短线间隔，但由于使用了圆头短线，所以这些间隔几乎都被填满了，所以整体看起来已经很像是一条实线了（译注：其实更像是—串香肠）。



图片来源: [Mapbox](#)

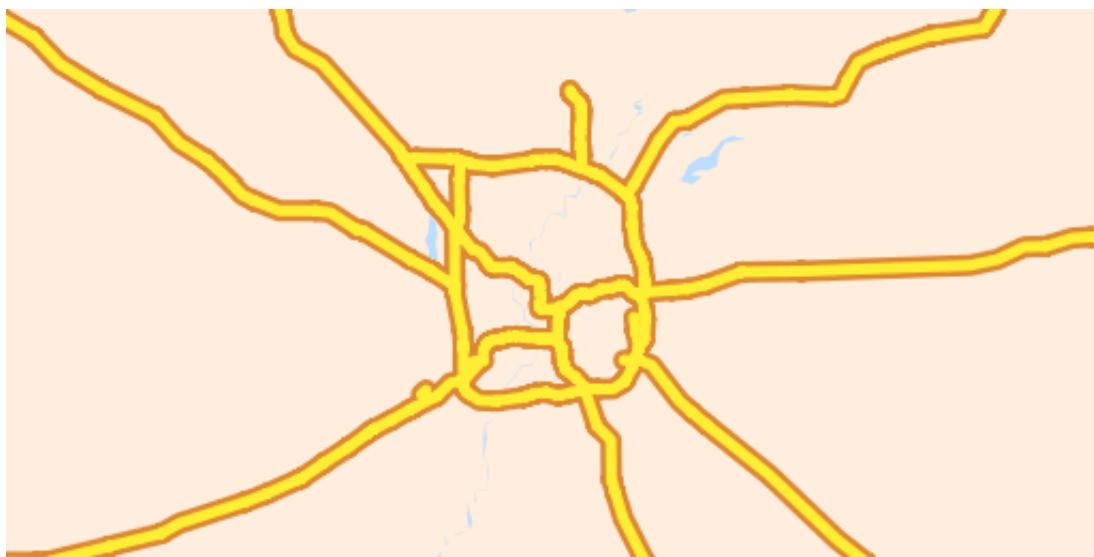
```
#layer {  
  line-width: 4;  
  line-cap: round;  
  line-dasharray: 10, 4;  
}
```

## 复合线样式

### 道路

For certain types of line styles you will need to style and overlap multiple line styles. For example, a road with casing:

对一些特定类型的线（如道路网），可以利用多个从属样式相互压盖来实现较为复杂的样式。例如，要实现一个带有边框效果的道路，可以用下面的方式实现。其中`::case`部分定义了道路的边框，实际上是位于下层，颜色为`#d83`的一条宽线；而`::fill`部分则定义了一条覆盖在宽线上面的一条颜色为`#fe3`的窄线。这两个从属样式共同实现了一条带有边框的道路。



图片来源: [Mapbox](#)

```
#road[class='motorway'] {  
    ::case {  
        line-width: 5;  
        line-color: #d83;  
    }  
    ::fill {  
        line-width: 2.5;  
        line-color: #fe3;  
    }  
}
```

对于不同的级别或类型的道路，样式定义也会相应复杂一些。用户可以按照道路的级别或组成样式的从属样式（即例子中的`::case`和`::fill`部分）对样式进行分组。下面的例子中的样式是按照道路级别进行分组（基于`class`字段设置过滤器）。



图片来源: [Mapbox](#)

```
#road {
  [class='motorway'] {
    ::case {
      line-width: 5;
      line-color: #d83;
    }
    ::fill {
      line-width: 2.5;
      line-color: #fe3;
    }
  }
  [class='main'] {
    ::case {
      line-width: 4.5;
      line-color: #ca8;
    }
    ::fill {
      line-width: 2;
      line-color: #ffa;
    }
  }
}
```

## 铁路

一种典型的铁路线样式是在一条细实线上等间隔的画上一系列垂直于细线的小短线。这种效果可以通过用两个相互叠加的从属线样式实现：一条细实线，还有一条短而宽且间隔较长的虚线。



图片来源: [Mapbox](#)

```
#road[class='major_rail'] {  
  ::line, ::hatch { line-color: #777; }  
  ::line { line-width: 1; }  
  ::hatch {  
    line-width: 4;  
    line-dasharray: 1, 24;  
  }  
}
```

另一种典型的铁路线样式与第一种类似，只是要把上层的虚线部分中的短线稍微调细一些和长一些，而把下层的实线部分调得更粗一些以形成铁路线的边框。配置这种样式的时候需要注意，要把`::dash`部分写在`::line`部分的后面，从而保证虚线样式能够盖在实线部分的上面。

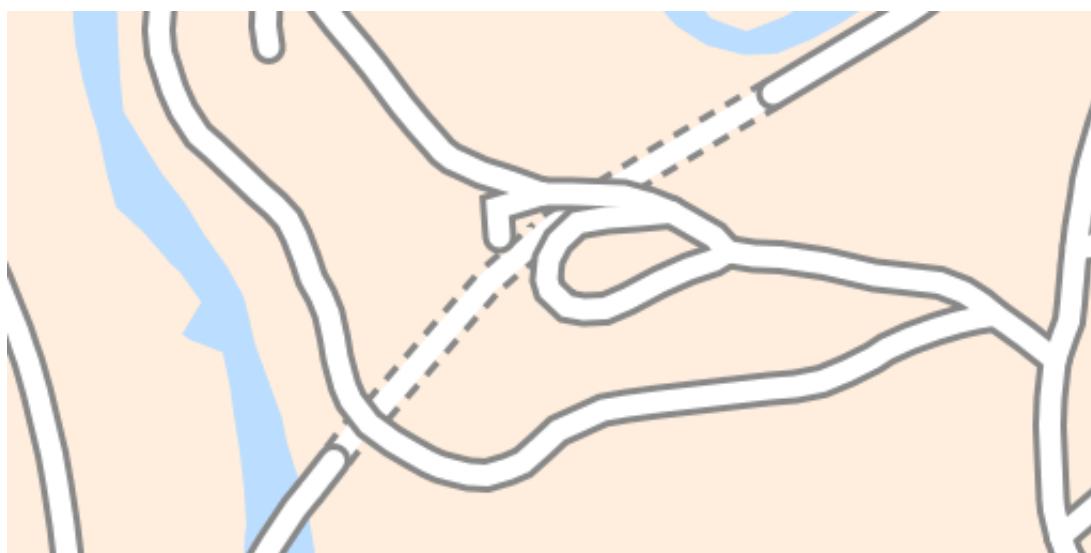


图片来源: [Mapbox](#)

```
#road[class='major_rail'] {  
    ::line {  
        line-width: 5;  
        line-color: #777;  
    }  
    ::dash {  
        line-color: #fff;  
        line-width: 2.5;  
        line-dasharray: 6, 6;  
    }  
}
```

## 隧道

一种简单的隧道样式可以通过把道路样式稍稍修改而成。具体来说，就是把作为背景的底层线条由实线改为虚线，即可得到下图中的渲染效果。



图片来源: [Mapbox](#)

```
#road,  
#bridge {  
    ::case {  
        line-width: 8;  
        line-color:#888;  
    }  
    ::fill {  
        line-width: 5;  
        line-color:#fff;  
    }  
}  
  
#tunnel {
```

```

::case {
  line-width: 8;
  line-color:#888;
  line-dasharray: 4, 3;
}

::fill {
  line-width: 5;
  line-color:#fff;
}

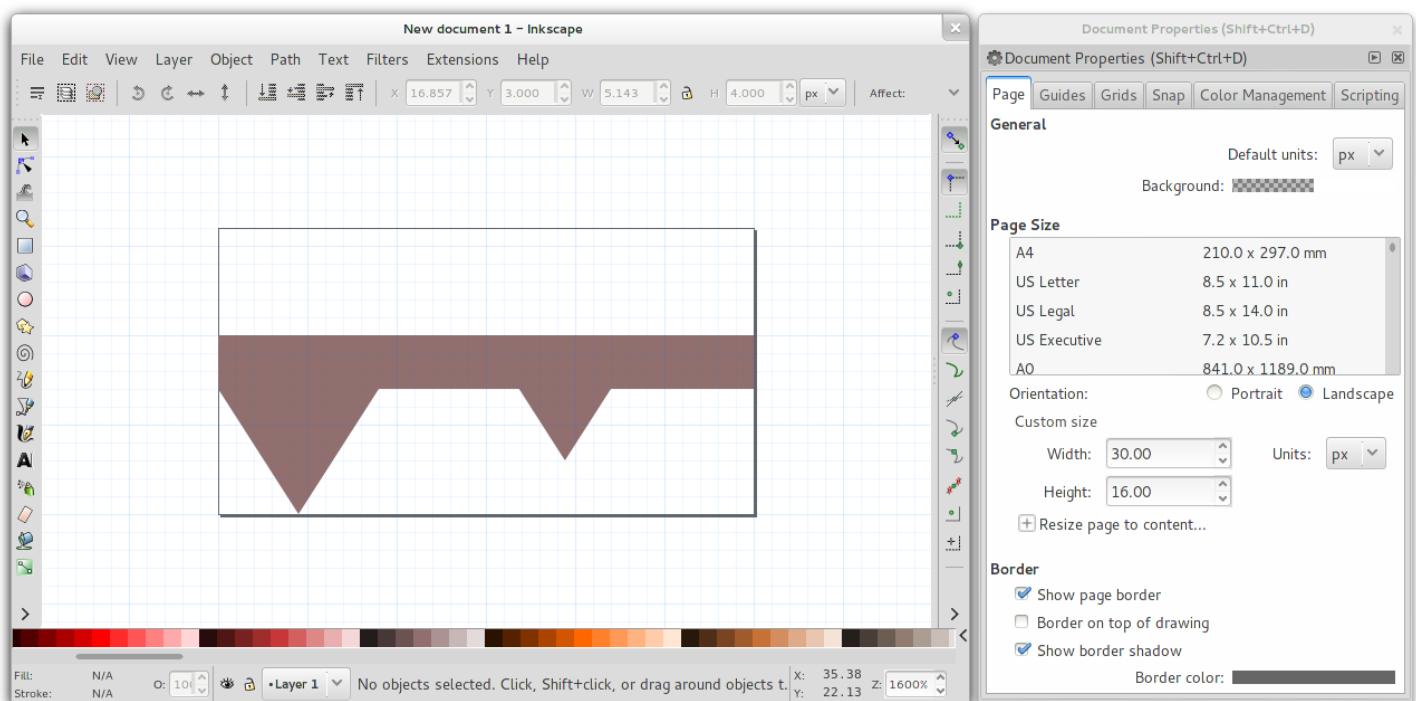
}

```

## 用图片作为线型图案

有些线型是非常复杂的，复杂到难以通过常规的复合线样式来实现。CartoCSS允许用户对线要素以沿线重复出现同一张图片的方式来绘制。在下面的例子中，一张用Inkscape绘制出的图片被用于表示悬崖，并且就以它作为线要素样式中的基本图案。

在Inkscape（或者其它矢量图形编辑软件）中，创建一个新的矢量图形，尺寸不能太大，其高度相当于线宽，宽度就是它沿线重复出现的单位。下面例子中的悬崖图形尺寸为30x16像素。



图片来源: [Mapbox](#)

请注意这个图形在设计过程中的对齐方式和上下留白的宽度，只有这样才能保证它画在线上可以对齐。将这个图片从Inkscape中导出成PNG文件，然后在CartoCSS中通过增加一个包含line-pattern-file属性的样式块来使用它。



图片来源: [Mapbox](#)

```
#barrier_line[class='cliff'] {  
    line-pattern-file: url(cliff.png);  
}
```

对于某些图案，比如上面例子中的悬崖，它的方向非常重要。CartoCSS中的约定是：图案图片的底部会被置于线要素（沿线方向）的右侧；而图案图片的左侧会与线要素的起始位置对准。

## 参考文献

1. Mapbox, [Mapbox Studio Styling Lines](#)

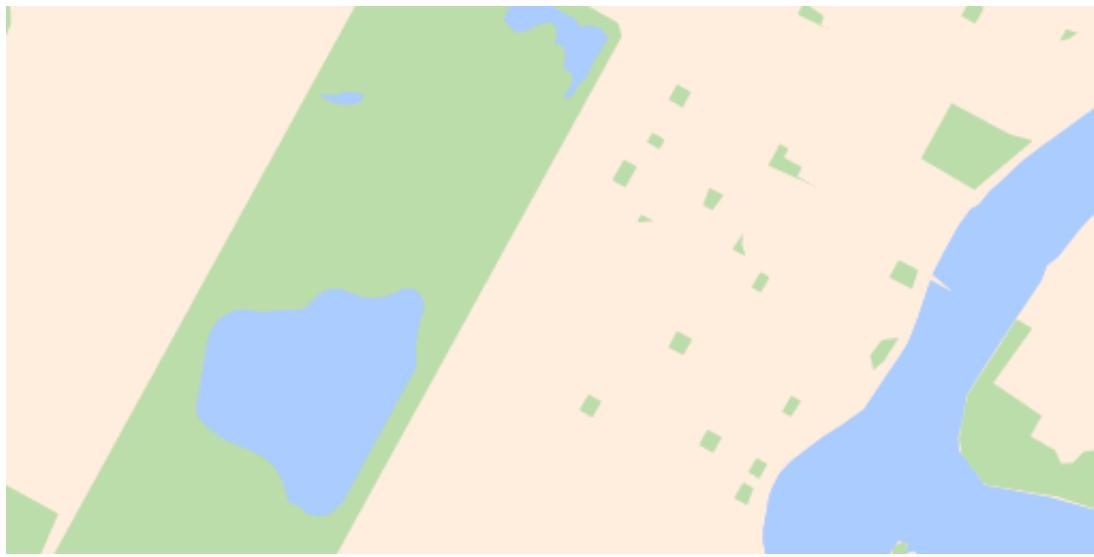
## 3.4 配置面样式

面要素由内部区域和边界组成。其中，内部区域可以用纯色或图案进行填充。

小贴士：用于配置线要素的方法也都同样适用于面要素图层。

### 基本样式配置

最简单的面样式就是纯色填充。



图片来源: [Mapbox](#)

```
#landuse[class='park'] {  
  polygon-fill: #bda;  
}  
}
```

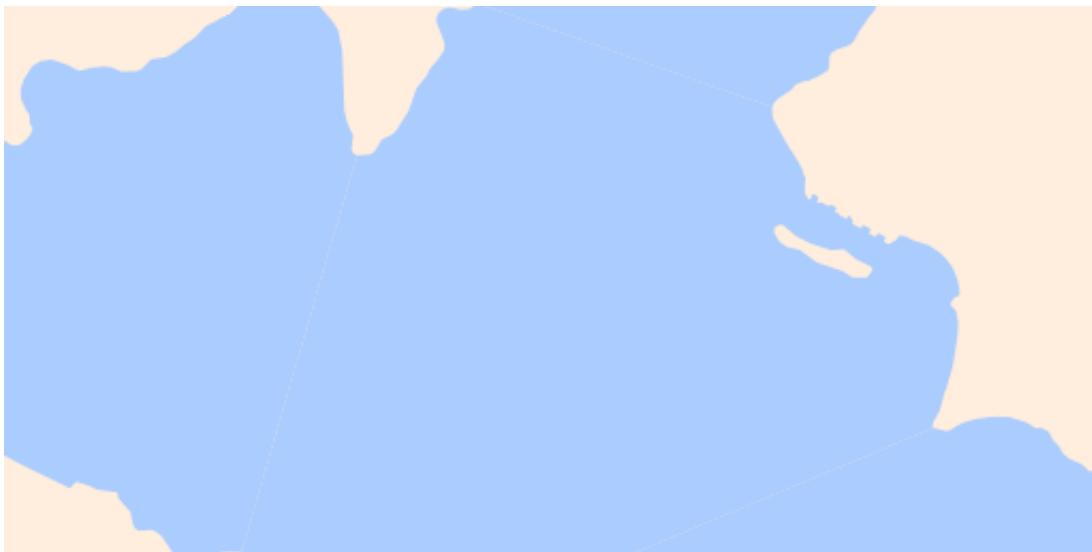
使用`polygon-opacity`属性可以调整填充色的透明度。它的取值范围是从0到1，0表示全透明，1表示完全不透明。如果把它设置成50%透明，那么可以得到这样一种效果（如下图）：如果当前图层中有相互重叠的面要素，那么重叠的部分的透明度会显得比其它部分更低一些。



图片来源: [Mapbox](#)

```
#landuse[class='park'] {  
  polygon-fill: #bda;  
  polygon-opacity: 0.5;  
}  
}
```

## 用伽马属性调整缝隙



图片来源: [Mapbox](#)

如果一个面图层中所有面要素对应的几何多边形之间都是无缝衔接的，那么在制图渲染后它们也理应完好的拼接在一起。然而在实际的制图效果中，却往往会在地图放大到某些级别时观察到这些原本无缝的多边形之间出现一些细微的缝隙（如上图中水域部分靠图片上方的那一条灰白色细缝）。要消除这种不该有的缝隙效果，可以借助于`polygon-gamma`属性。该属性的取值范围是0到1，默认值为1。可以尝试调小这个值来消除缝隙，但也要注意如果调的太小可能导致出现参差不齐的边缘甚至一些更加怪异的效果。



图片来源: [Mapbox](#)

```
#water {  
  polygon-fill: #acf;  
  polygon-gamma: 0.5;  
}
```

## 图案和纹理

在CartoCSS中，使用外部图片（例如来自文件系统或互联网的图片资源文件）作为图案和纹理对面要素进行填充是很容易实现的。用户可以自己用图像编辑软件制作这些图片，或者从图片资源网站（如[Subtle Patterns](#)、[Free Seamless Textures](#)等）上去获取现成的图片资源。

You can add a pattern style from any local file or web URL using the `polygon-pattern-file` style. Here is a simple diagonal stripe pattern you can use to try out - you can reference it from CartoCSS as in the snippet below.

用户可以从本地文件系统或互联网上添加图案样式资源，然后把资源文件的地址（文件路径或资源链接）赋给`polygon-pattern-file`属性。下面的例子是一个简单的斜纹图案，在CartoCSS中可以引用这个图片作为填充图案。



图片来源: [Mapbox](#)

```
#landuse[class='park'] {  
  polygon-pattern-file: url("pattern-stripe.png");  
}
```

为了让这些图案生效，这些外部的图片文件必须要保证自己是能够被CartoCSS解析器找到。如果图片文件来自本地文件系统，而且在CartoCSS脚本中是以相对路径的形式进行引用，那么就需要在与脚本文件相同的目录中建立对应的子目录。在图片资源文件较多的时候，这是一种值得推荐的做法，比如都放在`images`子目录中，然后通过相对路径引用这些图片资源：

```
#landuse[class='park'] {  
  polygon-pattern-file: url("images/pattern-stripe.png");  
}
```

## 背景图案

如果要在整个地图上应用某种背景图案，那么可以利用Map对象的background-image属性实现。

```
Map {  
  background-image: url("pattern.png");  
}
```

与其它Map对象上的属性一样，background-image的效果也是全局性的。这意味着它的渲染效果无法利用缩放级别过滤器进行调整。

## 图案与填充的结合

通过对透明度与合成操作的灵活运用，可以从一种图案衍生出千变万化的填充效果。

## 如何保证无缝

关于填充图案的对齐方式，主要有两种：一种是局部（默认方式）模式，另一种是全局模式。对齐方式通过polygon-pattern-alignment属性进行配置。

如果采用本地对齐模式，那么图案图片将与每个面要素对象分别对齐。图案图像的左上角将与面要素外包框的左上角对齐。而如果采用全局对齐模式，图案图片将会与地图瓦片对齐，而非每个面要素对象。因而填充图案会在全部面要素上排队一样反复出现。在全局对齐模式下，填充图案的尺寸不能超过元瓦片（除缓冲区以外的部分），否则图案的一些部分会无法显示。

在全局对齐模式下，还有一点需要特别注意：图案的尺寸必须要能够整除地图瓦片的尺寸。比如说地图瓦片的尺寸是256x256，那么图案的尺寸可以是16, 32或者128，但不能是20, 100或者其它不能整除256的数字。如果是从一些图片资源网站获取的图案，那么请根据这个约束条件用图像编辑软件对图案文件的尺寸进行修正。

## 参考文献

1. Mapbox, [Mapbox Studio Styling Polygons](#)

## 3.5 配置标注样式

### 简单点标注

在CartoCSS中，文本标注的样式由一系列以text-开头的属性来配置。在这些属性中，text-name和text-face-name是必需的。前者用于指定显示在标注中的文本内容；而后者用于指定显示文本所用的字体。text-name属性可以将图层中的属性数据字段提取出来作为标注的文本内容。比如一个图层中有个属性字段叫name\_en，那么一个简单的文本标注就可以用下面的方式进行样式定义：



图片来源: [Mapbox](#)

```
#place_label {  
    text-name: [name_en];  
    text-face-name: 'Open Sans Condensed Bold';  
}
```

上面例子中标注的颜色和字号都是默认值: 黑色, 10像素。这两个值可以用text-fill和text-size属性进行调整。



图片来源: [Mapbox](#)

```
#place_label {  
    text-name: [name_en];  
    text-face-name: 'Open Sans Condensed Bold';  
    text-fill: #036;
```

```
    text-size: 20;  
}  
}
```

为了能让文本标注在地图背景中更为醒目，通常会为标注上的文字增加一圈边线或光晕。光晕的颜色和宽度分别用`text-halo-fill`和`text-halo-radius`属性进行配置。在下面的例子中，我们利用了`fadeout`颜色变换函数得到一个具有30%透明度的白色光晕。



图片来源: [Mapbox](#)

```
#place_label {  
    text-name: [name_en];  
    text-face-name: 'Open Sans Condensed Bold';  
    text-fill: #036;  
    text-size: 20;  
    text-halo-fill: fadeout(white, 30%);  
    text-halo-radius: 2.5;  
}
```

## 沿线标注

CartoCSS支持沿线标注，例如沿着道路或河流的走向来绘制文本标注。要实现沿线标注需要将`text-placement`属性值设置为`line`（它的默认值是`point`）。在下面的例子中，除了标注样式以外，还有一些简单的河流线样式。



图片来源: [Mapbox](#)

```
#waterway_label {  
    text-name: [name_en];  
    text-face-name: 'Open Sans Condensed Bold';  
    text-fill: #036;  
    text-size: 20;  
    text-placement: line;  
}
```

对于河流来说，让它的标注相对于河流流向稍作平移会比较美观。这种效果可通过设置text-dy属性来实现。它的值以像素为单位，用于指定将标注沿y轴方向的偏移量。

还有一个text-max-char-angle-delta属性，它用来控制标注文本的最大转弯角度，其默认值是22.5°。如果把它的值调小，那么标注就会被绘制在线要素上更为平直的部分，避开尖锐拐角。



图片来源: Mapbox

```
#waterway_label {  
    text-name: [name_en];  
    text-face-name: 'Open Sans Condensed Bold';  
    text-fill: #036;  
    text-size: 20;  
    text-placement: line;  
    text-dy: 12;  
    text-max-char-angle-delta: 15;  
}
```

## 添加自定义文字

Labels aren't limited to pulling text from just one field. You can combine data from many fields as well as arbitrary text to construct your `text-name`. For example you could include a point's type in parentheses.

用于标注的文字（也就是`text-name`属性的值），不仅可以从空间数据的某一个属性字段中获取，还可以由多个属性字段组合而成，甚至还可以是用户自定义的任意文本。例如，可以在标注文字的后面添加一个写在括号中的兴趣点类型：



```
#poi_label {  
    text-name: [name_en] + ' (' + [type] + ')';  
    text-face-name: 'Open Sans Condensed Bold';  
    text-size: 16;  
}
```

其它一些应用实例:

- 多语言标注: `[name] + '(' + [name_en] + ')'`
- 多级行政区划: `[city] + ', ' + [province]`

- 计量单位: [elevation] + 'm'
- 特殊的unicode字符: '🚩' + [embassy\_name] or '⚓' + [harbour\_name]

标注的内容还可以不从属性数据的字段中提取，而是使用任意用户自定义的文本。但由于向后兼容的原因，这样的自定义文本必须要套上两层引号：

```
#poi_label[maki='park'] {
  text-name: "'Park'";
  text-face-name: 'Open Sans Regular';
}
```

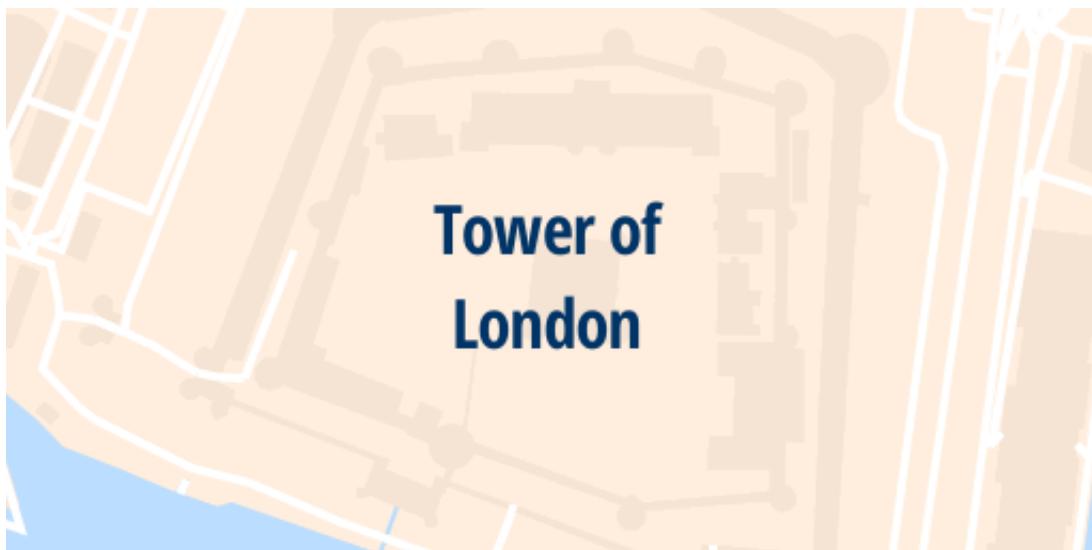
如果标注文本中包含引号，那么需要使用反斜杠来转义。例如，如果要标注文字**City's “Best” Coffee**，那么要这样写：

```
text-name: "'City\'s \"Best\" Coffee'";
```

## 多行标注

You can wrap long labels onto multiple lines with the `text-wrap-width` property which specifies at what pixel width labels should start wrapping. By default the first word that crosses the wrap-width threshold will not wrap - to change this you can set `text-wrap-before` to `true`.

对于包含较长文本的标注，可以通过设置`text-wrap-width`属性来将其折行显示。这个属性用于指定标注文本从什么位置（以像素为单位）开始折行。对于正好位于设定的折行宽度处的第一个字，默认是不会被折到下一行显示。如果要修改这个默认行为，可以将`text-wrap-before`属性设置为`true`。



图片来源: [Mapbox](#)

```
#poi_label {
  text-name: [name];
  text-face-name: 'Open Sans Condensed Bold';
```

```
text-size: 16;
text-wrap-width: 150;
text-wrap-before: true;
}
```

不过，需要注意多行文本暂不支持沿线标注的情况。

用户还可以自行指定一个特定的折行点，比如\n。

```
#poi_label {
  text-name: [name] + '\n' + [type];
  text-face-name: 'Open Sans Condensed Bold';
  text-size: 16;
}
```

(译注：以下部分为原TileMill文档中的内容，但在新的Mapbox Studio文档中已被删除，但本书编译者认为这部分值得保留。)

用户还可以利用text-wrap-character属性来指定一个折行字符（默认是空格）。这个特性在对一些具有特定结构的数据集进行文本标注时特别有用。例如在之前的多字段组合标注例子中，我们可以用下划线\_作为多个字段的连接符，然后将折行字符就设成下划线\_，之后再把折行宽度text-wrap-width设得很小，这样就能够保证不同的字段总是能分别显示在独立的行中。



图片来源：[Mapbox](#)

```
#cities {
  text-name: [NAME] + '_' + [ADM1NAME];
  text-face-name: 'Droid Sans Regular';
  text-size: 20;
  text-wrap-width: 1;
  text-wrap-character: '_';
}
```

## 将标注层独立出来

If you are applying label styles to layers that also have line or polygon styles you might notice some unexpected overlapping where the labels aren't necessarily on top.

如果在一个已经配置了线样式或面样式的图层上也同时配置标注样式，那么往往会出现一些标注和几何要素相互压盖等异常的渲染效果，无法保证标注能出现在最上层。

对于这个问题，在地图样式比较简单的时候，可以通过将地理要素的样式与标注样式分别定义在不同的从属样式中来解决：

```
#layer {
  ::shape {
    polygon-fill: #ace;
    line-color: #68a;
  }
  ::label {
    text-name: [name];
    text-face-name: 'Arial Regular';
  }
}
```

而对于样式比较复杂的大多数情况，则最好是为标注专门创建一个独立于地理要素的图层，然后分别定义它们的样式。（可参考TileMill中的示例制图项目Open Streets DC）

这时，专门用于标注的图层与其对应的地理要素图层实际指向的是同一个地理数据集，但是要把标注图层置于地理要素图层之上，以保证标注不会被地理要素压盖。关于图层和地图对象的绘制顺序问题，请参考下一节“符号的渲染顺序”。

## 参考文献

1. Mapbox, [Mapbox Studio Styling Labels](#)

## 3.6 符号的渲染顺序

通过CartoCSS配置好样式的地图将按照Painter's算法来绘制地图中的各种对象。这些对象的渲染遵循特定的顺序，先绘制的对象可能会被后绘制的对象覆盖。

### 概述

概括的说，地图上的对象要依照以下四个条件所确定的顺序进行绘制：

1. 图层的绘制顺序：按照从低层到高层的顺序进行绘制，高层图层覆盖低层图层。
2. 从属样式的绘制顺序：按照从属样式（例如::glow { ... }）在同一样式块中出现的顺序自顶向下进行绘制。
3. 从属样式所作用的要素对象的绘制顺序：按照这些对象在原始数据集中的存储顺序进行绘制。
4. 同一对象上定义的多个样式实例的绘制顺序：按照样式实例定义的顺序进行绘制。

接下来我们来详细解释这些规则。

### 顺序vs.优先级

对于线要素和面要素对象来说，越是先被绘制出来的，越有可能在最终的地图上只能露出一部分甚至完全被遮住。因为在绘制堆栈中的高位对象，也就是那些按顺序应该后画的对象很可能会在绘制时把之前先画出来的对象部分或完全覆盖。所以我们说对于这一类要素，处于绘制堆栈中的高位对象具有较高的“优先级”或“重要性”。

而对于像文本标注、注记和图标等`allow-overlap`属性默认为`false`的符号，情况就有所不同了。对于用这些符号定义样式的对象，越是先被画出来的，就越有可能在最后的地图上可见。其原因是如果后画的对象会遮挡前面已经画出来的对象，那么这些对象就直接被忽略了，根本不会被画出来。所以说在这种情况下绘制堆栈中的高位对象可能会在绘制过程中反而被忽略掉，因而具有较低的“优先级”或“重要性”。

总而言之，就是对于线、面要素而言，绘制顺序比较靠后的对象具有更高的优先级，也就是更可能在最终的地图上可见；而对于文本、注记和图标等要素，却是完全相反，即绘制顺序比较靠后的对象具有更低的优先级，更可能在最终的地图上不可见。这就是本节讨论的所谓顺序与优先级的关系问题。

## 图层排序

一幅地图中所包含的若干个图层是按照自底向上的顺序逐层绘制的。在一个典型的地图数据集中，像土地利用、水系这样的基础数据通常位于图层列表的底部。而那些在地图上不允许被遮盖的要素（如标注、图标等）则通常位于列表的顶部。

## 从属样式排序

在一个图层内部，样式可以进一步通过使用`::`而被拆分为多个从属样式块。从属样式也可以被理解为子图层。



图片来源: [Mapbox](#)

```
#layer {  
  ::outline {  
    line-width: 6;  
    line-color: black;  
  }  
  ::inline {  
    line-width: 2;  
    line-color: white;  
  }  
}
```

从属样式块将按照其被首次定义的顺序来绘制。因此在上面的例子中，`::outline`部分定义的线样式将先于`::inline`部分被画出来。

需要注意的是，实际上所有样式定义都被包含在某一个从属样式块之中。如果不显式定义一个从属样式块，那么CartoCSS还是会为这些“自由”样式定义生成一个默认的匿名从属样式块。在下面的例子中，`line-width: 2;` `line-color: white;`这两句样式定义实际上也属于一个从属样式块，一个隐式定义的默认、匿名从属样式块。与前面的不同之处仅仅在于其名字不叫`::inline`而已，所以最后渲染出来的效果和上面的完全一样。



图片来源: [Mapbox](#)

```
#layer {
  ::outline {
    line-width: 6;
    line-color: black;
  }
  line-width: 2;
  line-color: white;
}
```

## 数据排序

在处理每个从属样式块时，其涉及到的每条要素数据在数据集（无论是在外存还是内存）中的排序对于渲染结果也有影响。在某些矢量数据集中，会为了保证制图渲染效果而专门对要素数据的存储顺序进行优化。

如果用于制图的矢量数据集是用户自己创建的、完全可控的，那么就应该要考虑其中要素数据记录的顺序问题。就拿城市名称标注来说，把用于标注的数据按照某种优先级排个序会对后面的制图有好处。如果这些数据是从关系数据库中查出来的，那么最好在`select`语句中用`ORDER BY`把它们按照某个列（或其它优先级规则）排一下序。

而如果数据来自磁盘文件，那么通常就只能是按照从头到尾的顺序去读，无法实时的调整数据记录的顺序。这种时候可以先把数据文件预处理一下，将其中要素数据记录的顺序调整好之后再把它作为图层加载并制图。

这种调整顺序的预处理可以用`ogr2ogr`在命令行完成。下面的命令就实现了把`cities.shp`文件中的所有要素记录按照`population`字段的值从大到小的顺序重新排列。

```
ogr2ogr -sql \
'select * from cities order by population desc' \
```

```
cities_ordered.shp cities.shp
```

## 符号排序

在每个从属样式中，可以为每个要素对象应用多种符号进行修饰。就拿面要素来说，它既可以有用面符号修饰的填充样式，又可以有用线符号修饰的边界线样式。在这种情况下，多个样式符号将按照其定义的顺序进行绘制。根据这样的原则，下面例子中的绘制顺序就是先边界线，后填充。



图片来源: [Mapbox](#)

```
#layer {
  line-width: 6;
  polygon-fill: #aec;
  polygon-opacity: 0.8;
}
```

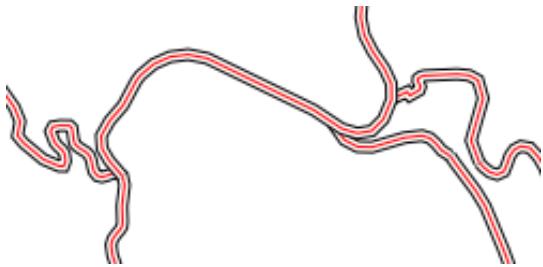
而在下面的例子中则正相反，是先画填充，再在它上面画边界线。



图片来源: [Mapbox](#)

```
#layer {
  polygon-fill: #aec;
  polygon-opacity: 0.8;
  line-width: 6;
}
```

此外，还可以在一个从属样式块中利用具名的实例实现同一个样式属性的重定义。与从属样式的命名规则一样，实例也可以取任意的名字。实例的概念我们在3.1节中已经介绍过了。



图片来源: [Mapbox](#)

```
#layer {  
    bottomline/line-width: 6;  
    middleline/line-width: 4;  
    middleline/line-color: white;  
    topline/line-color: red;  
}
```

需要注意的是，各种排序也是有优先级和顺序的，而符号排序就是所有类型排序中优先级最低的，所以可能会出现一个边界的边界线绘制在最上层，但却被相邻的多边形盖在下面的情况，这是因为数据排序的优先级高于符号排序。如果想要确保边界线总是被画在填充的下面，就需要另外定义一个从属样式来实现。

## 图层数据源排序 (Layer Source Ordering)

The order in which your layer sources are specified also influences rendering order: data from sources are rendered in order. So if you click “Change source” under “Layers” and you see `you.id123`, `mapbox.mapbox-terrain-v1`, the layers from `mapbox.mapbox-terrain-v1` will render last, over the layers from `you.id123`. To ensure that your own data renders last, use `mapbox.mapbox-terrain-v1`, `you.id123`.

图层的数据源的排序也会影响渲染顺序，它们会按照指定的顺序进行绘制。比如图层数据源列表为`you.id123`, `background-terrain`, 那么从`background-terrain`数据中派生出的那些图层将被后绘制，盖在那些`you.id123`数据派生出的图层上面。为了保证用户自己的数据最后再绘制，需要把用户自己的数据集尽量放在图层数据源列表的尾部，例如`background-terrain`, `you.id123`。

## 参考文献

1. Mapbox, [Mapbox Studio Symbol Drawing Order](#)

## 3.7 关于合成操作

合成 (Compositing) 操作是CartoCSS中颇具特色的功能。当然，CartoCSS能支持合成的基础自然是因为作为其底层地图渲染引擎的Mapnik提供了这个强大的能力。Mapnik在从0.7.x版本开始提供了对栅格数据的合成功能，而从2.1.x版本则开始能够支持矢量数据的合成操作，而且是同时能够支持要素级和图层级的合成。

合成这个概念是来自图像处理领域，它还有个名字，叫Photomontage，中文是“蒙太奇照片”，指的是将两幅或更多幅照片通过剪辑、合成等处理生成一幅新的看起来很和谐的图片，有点像电影后期制作中的特效。在很多图像处理软件（例如Adobe Photoshop、Paint.NET、GIMP等）中都直接内置了对图层的合成功能。

CartoCSS中的合成操作则是针对地图上的各种制图对象，在对它们进行渲染成最终图片的过程中通过改变不同要素、样式、图层的颜色、纹理之间相互作用的方式（例如在像素级进行各种算术或逻辑运算）来达到一些特殊的视觉效果，让地图上也可以有“特技”的感觉。

既然是叫“合成”，那么自然至少要涉及到两个参与“合成”的成员。在CartoCSS中，这两个成员分别被称为源与目标。我们在讨论合成操作的时候，必须明确源与目标。所谓源，就是应用了comp-op属性的样式或符号，而所谓目标，则是其它那些绘制在它下面的图像。在源的上面当然还可能有其它要绘制的部分，但它们都不会受到当前源中comp-op的影响，该怎么画就还怎么画。在默认情况下，也就是不在源上显式指定任何合成操作时，那么它就会被直接覆盖绘制在目标之上。而通过使用各种合成操作，我们就改变这种默认绘制行为。

目前，在CartoCSS中一共有33种合成操作：

CartoCSS支持的合成操作列表		
plus	difference	src
minus	exclusion	dst
multiply	contrast	src-over
screen	invert	dst-over
overlay	invert-rbg	src-in
darken	grain-merge	dst-in
lighten	grain-extract	src-out
color-dodge	hue	dst-out
color-burn	saturation	src-atop
hard-light	color	dst-atop
soft-light	value	xor

上面的列表中左边两列是22种色彩混合。它为对象与对象、图层与图层之间在颜色上如何融合提供了一系列不同方式。而最右边一列中列出的11种Duff-Porter透明度混合则提供了关于填充和遮盖的不同方式。

如果你对图像编辑软件（例如GIMP、Adobe Photoshop）比较熟悉，那么应该能感觉到上面这些模式很多都和这些软件中的图层混色很像。没错，就是很像，但不同的是，CartoCSS中的合成操作却可以不必作用于整个图层。具体而言，CartoCSS的合成操作有两种使用方式：一是通过comp-op属性作用于一整块从属样式；二是通过以下这些针对特定符号的合成操作属性作用于某种符号：

- line-comp-op
- line-pattern-comp-op
- marker-comp-op
- point-comp-op
- polygon-comp-op
- polygon-pattern-comp-op
- raster-comp-op
- shield-comp-op
- text-comp-op

这两种方式都很常用，具体使用哪种要看用户自己想要得到哪种渲染效果。它们的区别在于：如果采用第一种方式，那么样式块对应的内容会被完全渲染成图，然后再和目标进行合成；而如果采用第二种方式，那么不仅是样式块中那些有重叠部分的对象之间，而且在当前图层与其下方的图层之间都将应用合成操作。两种方式的差异见下面这个例子。



// 方式一: style-wide

```
#countries {  
    line-color: #345;  
    line-width: 4;  
    polygon-fill: #fff;  
    comp-op: overlay;  
}
```



// 方式二: symbolizer-specific

```
#countries {  
    line-color: #345;  
    line-width: 4;  
    line-comp-op: overlay;  
    polygon-fill: #fff;  
    polygon-comp-op: overlay;
```

}

## 色彩混合

色彩混合合成操作一共有22种。这里对其中那些在制图设计中最常用的17种操作进行简单介绍。为了更明显的表现出不同合成操作之间的差异，我们将这些操作应用在两个示例图层和两幅背景图上。

下面是两个应用comp-op的示例图层（也就是源）：



下面是两幅被comp-op图层叠加的背景图（也就是目标）：



下面是在不定义comp-op属性时的渲染效果：



(\_译注：在下面的这些操作的翻译中，首先需要统一的就是这个source和destination的译法。在很多情况下，把它们译成“源”和“目标”会感觉很怪异拗口，但译成“源层”、“源图像”又都不准确。叫“源对象”？“合成源”？略感苦闷。。\_)

## Overlay



The **overlay** comp-op combines the colors from the source image, and also uses them to exaggerate the brightness or darkness of the destination. Overlay is one of a few composite operations that works well for texturing, including using it for terrain data layers.

**overlay**操作取源图像中颜色与目标合并，并用源图像的颜色去增强目标图像中的明暗度（译注：把控不好exaggerate的翻译）。**overlay**是几种适用于表现图像纹理特征的合成操作之一。此外，它还可以用于地形数据图层的样式配置。

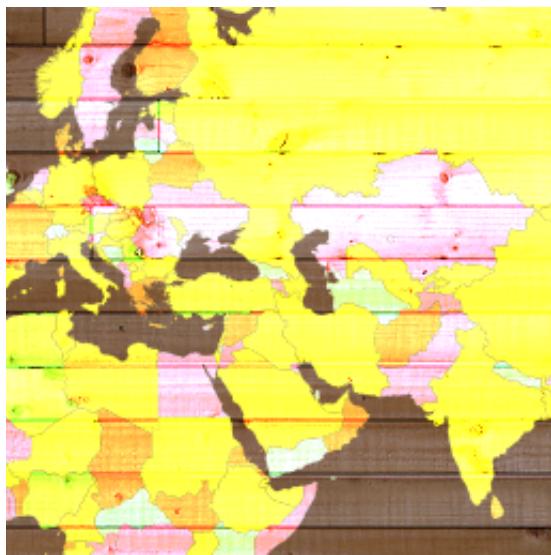
## Multiply



`multiply`操作将源与目标的颜色进行相乘运算。应用`multiply`通常能得到一幅颜色比源图像更暗的图像。如果源（或目标）是纯白色的，那么合成的效果就是保持目标（或源）的颜色不变。如果源或目标是纯黑色的，那么合成的效果就是纯黑色图像。

`multiply`操作的众多应用之一是模拟不同颜色的墨水混合或墨水与带有纹理的表面混合的效果。此外，它还可以用于其它纹理效果。

### Color-dodge



`color-dodge`操作以源层为基础提高目标颜色的亮度。源图像的颜色越亮，合成后的效果越刺眼。要想得到比较美观的效果，就要在偏暗的颜色上应用该操作，否则就会得到过于刺眼的合成效果。

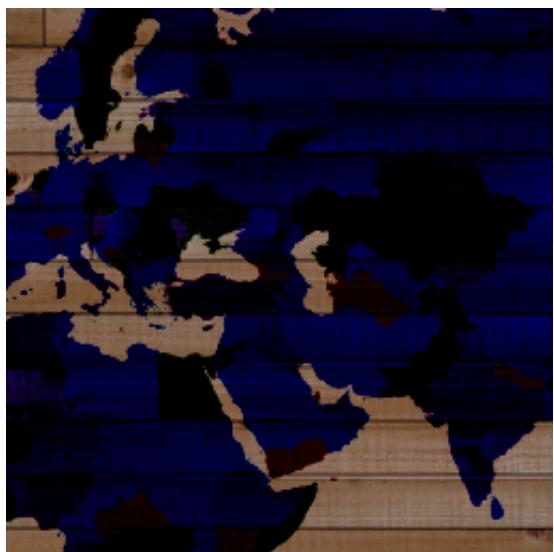
### Plus (译注：求帮忙校对翻译)



The **plus** comp-op adds the color of the source to the destination. For example, if your source color is dark red, this operation will add a small amount of red color to the destination causing it to brighten and also turn red. The lighter your source color, the lighter your result will be because a lot of color will be added. A completely black source will not affect the destination at all because no color will be added. Using this mode on darker source layers is recommended.

**plus**操作将源层的颜色与目标颜色相加。举个例子，如果源的颜色是深红，那么**plus**操作就会为目标增加少量的红色，使其亮度增加而且颜色偏红。源的颜色越浅，得到结果的颜色也会越浅，因为会有很多颜色被添加（译注：神马意思？？）。纯黑色的源层不会对目标产生任何影响，因为没有颜色会被叠加（译注：这里的“颜色”指的是RGB中的颜色分量吗？黑色是#000，所以“no color”；白色是#FFF，所以有“a lot of color”，是这个意思？）。推荐在源层颜色较深的时候使用这个合成操作。

## Minus



The **minus** comp-op subtracts the color of the source from the destination. For example, if your source color is a dark red, this operation will remove a small amount of red color from the destination causing it to darken and turn slightly green/blue. The lighter your source color, the darker your result will be because a lot of color will be subtracted. A completely black source will not affect the destination at all because no color will be removed. Using this mode on darker source layers is recommended.

**minus**操作从目标层的颜色中减去源层的颜色。举个例子，如果源层颜色是深红，那么这个操作会从目标层中减少一部分红色，使其亮度变暗而且颜色偏绿/蓝。源层颜色越浅，得到的结果颜色越深，因为会有更多的颜色被减掉。纯黑色的源不会对目标产生任何影响，因为没有任何颜色分量被减掉。推荐在源层颜色较深的

时候使用这个合成操作。

In the bathymetry example above there are more polygons overlapping each other. The subtraction is run for each overlapping piece, causing areas with a lot of overlap to darken more and shift more to the green spectrum.

在上面海水背景叠加的例子中，有更多相互叠加的多边形。减操作会被作用于每个叠加的部分，从而引起重叠很多的区域亮度更暗，颜色更偏绿色。

### Screen



The screen comp-op will paint white pixels from the source over the destination, but black pixels will have no affect. This operation can be useful when applied to textures or raster layers.

screen操作是把源层中的白色像素点画到目标层上，而黑色像素点则不会有效果。这个操作适用于纹理或栅格图层。

### Darken



The darken comp-op compares the individual red, green, and blue components of the source and destination and takes the lower of each. This operation can be useful when applied to textures or raster layers.

`darken`操作会对源层与目标层颜色中的R、G、B分量分别比较，然后取较小的值作为合成结果的颜色分量。这个操作适用于纹理或栅格图层。

### Lighten



The `lighten` comp-op compares the individual red, green, and blue components of the source and destination and takes the higher of each.

`lighten`操作会对源层与目标层颜色中的R、G、B分量分别比较，然后取较大的值作为合成结果的颜色分量。

### Color-burn



The `color-burn` comp op darkens the colors of the destination based on the source. The darker the source, the more intense the effect.

`color-burn`操作基于源层颜色对目标层颜色进行加深。源层的颜色越深，合成结果就会越强烈。（译注：intense这里该译成什么？）

### Hard-light



The **hard-light** comp-op will use light parts of the source to lighten the destination, and dark parts of the source to darken the destination. Mid-tones will have less effect

hard-light操作会用源层中较明亮的部分去调亮目标层的亮度，用源层中较暗的部分去调暗目标层的亮度，而那些中间色调的颜色则不会有太多影响。

### **Soft-light**



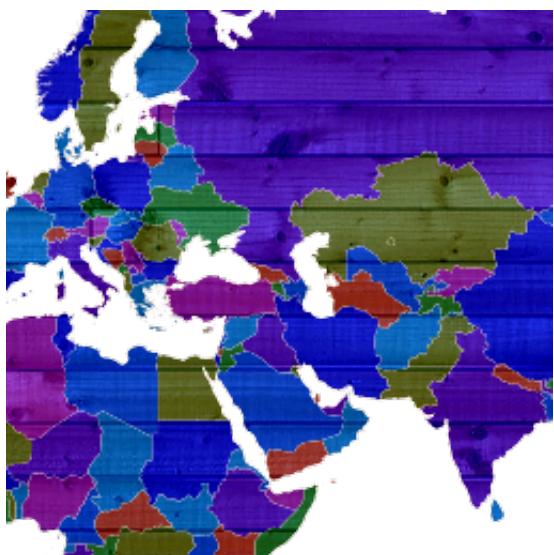
The **soft-light** comp-op works like a less intense version of the overlay mode. It is useful for applying texture effects or ghost images.

soft-light操作可以看作是overlay操作的一个弱化版本。它可用于产生纹理效果或虚影图像。

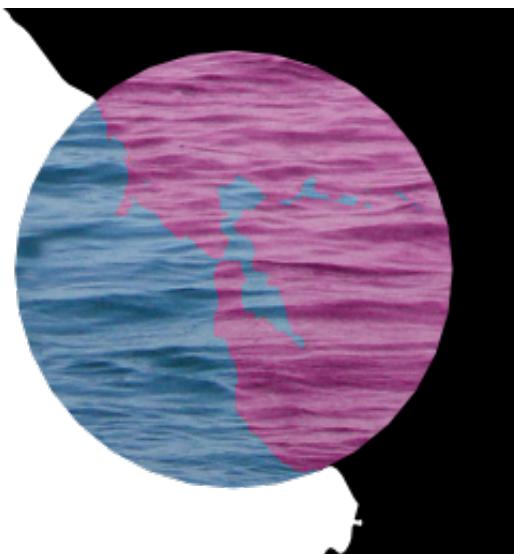
### **Grain-merge**



**Grain-extract**



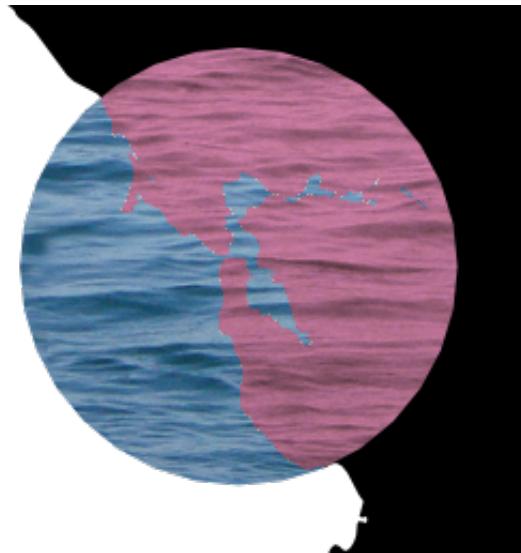
**Hue**



The hue comp-op applies the hue of the source pixels to the destination pixels, keeping the destination saturation and value.

`hue`操作将源层中每个像素的色调应用于目标层的对应像素中，而保持饱和度与明度不变（译注：即在HSV颜色模型中，应用H，而保持S和V不变）。

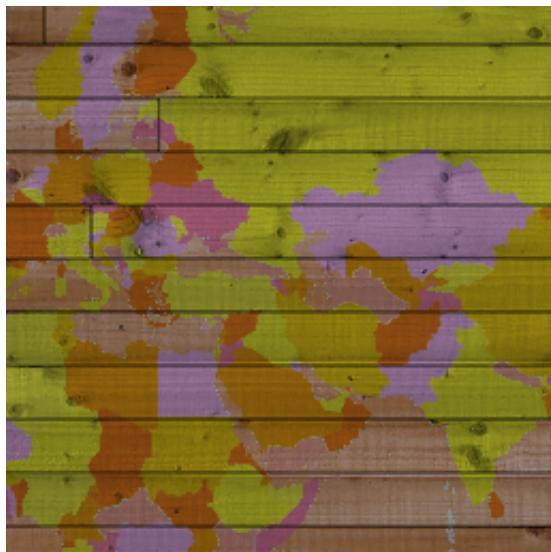
## Saturation



The `saturation` comp-op applies the saturation of the source pixels to the destination pixels, keeping the destination hue and value.

`saturation`操作将源层中每个像素的饱和度应用于目标层的对应像素中，而保持色调与明度不变（译注：即在HSV颜色模型中，应用S，而保持H和V不变）。

## Color



The `color` comp-op applies the saturation of the source pixels to the destination pixels, keeping the destination hue and value.

`color`操作将源层中每个像素的饱和度应用于目标层的对应像素中，而保持色调与明度不变（译注：即在HSV颜色模型中，应用S，而保持H和V不变。可是可是，这个和上边的`saturation`操作的解释怎么完全一样？）。

## Value



The `value` comp-op applies the value of the source pixels to the destination pixels, keeping the destination hue and saturation.

`value`操作将源层中每个像素的明度应用于目标层的对应像素中，而保持色调与饱和度不变（译注：即在HSV颜色模型中，应用V，而保持H和S不变）。

## 透明度混合（Alpha Blending）

There are 11 alpha blending compositing operations. Rather than altering the colors of a layer, these operations use the shapes of a layer to show or hide the rest of the image in different ways.

透明度混合合成操作一共有11种。不同于颜色混合操作是对层的颜色进行修改，透明度混合主要利用层与层之间的形状关系来显示或隐藏渲染图像的某些部分，且能够支持很多种不同的显示或隐藏方式。

Some of these modes will be more useful when applied to the whole style with the `comp-op` property, rather than with a symbolizer-specific property such as `polygon-comp-op`. All of the examples below were created with `comp-op`; there would be fewer differences between some of them had `polygon-comp-op` been used.

在这些透明度混合操作中，有一些更适用于通过`comp-op`属性应用于整个样式块，而非用于特定符号的合成属性（如`polygon-comp-op`）。本小节的例子全部都是通过`comp-op`属性实现的。它们其中有一些如果应用`polygon-comp-op`属性的话，效果会有些许不同。

The `src` and `dst` composite operations show only the source and destination layers, respectively. Neither are of much use in Mapbox Studio (where you can just as easily hide the layers). The `src-over` comp-op is another one you won't be using (**typo**) much. It draws the source and destination normally, the same as not applying a comp-op at all. The rest of the alpha blending compositing operations may be useful for cartography, however.

`src`和`dst`操作的含义是分别只显示源层和目标层。在实际制图中，这两种操作很少被用到（因为可以通过打开或关闭是否隐藏图层的开关来实现一样的效果）。`src-over`也是一个基本不会被用到的操作，因为它的含义就是按顺序正常绘制源层和目标层，和不使用`comp-op`属性效果完全一样。剩下的其它8种透明度混合操作在制图中就比较有用了，下面来一一介绍。

### Dst-over



The `dst-over` comp-op will draw the source beneath everything else. If your destination forms a solid background, this will effectively hide the source.

`dst-over` 操作会将源层绘制在最底层。如果目标层是不透明的背景，那么这个操作的效果就是把源层隐藏起来。

#### Src-in



The `src-in` comp-op will only draw parts of the source if they intersect with parts of the destination. The colors of the destination will not be drawn, only alpha channel (the shapes). If your destination forms a solid background, this operation will effectively be the same as `src`, since all parts of the source will intersect with the destination.

`src-in` 操作的效果是只绘制出源层中与目标层相交叠的部分。目标层中除了Alpha通道以外的颜色值都不会被绘制。如果目标层只是一个单纯的不透明背景，那么这个操作的效果就和`src`一样，因为源层中的所有部分都会和目标层相交叠。

#### Dst-in



The `dst-in` comp-op will only draw parts of the destination that intersect with parts of the sources. The colors of the source will not be drawn, only the alpha channel (the shapes). If your source is completely solid, this operation will effectively be the same as `dst`, since all parts of the destination will intersect with the source.

`dst-in`操作是只绘制目标层中与源层交叠的被遮罩。源层中除了Alpha通道以外的颜色值都不会被绘制。如果源层完全不透明，那么这个操作就和`dst`效果一样，因为目标层的所有部分都会和源层相交叠。

#### Src-out



The `src-out` comp-op will only draw parts of the source that do not intersect parts of the destination. The colors of the destination will not be drawn, only alpha channel (the shapes). If your destination forms a solid background, this operation will completely hide both the source and the destination, since all parts of the source intersect the destination.

`src-out`操作的效果是只绘制源层中那些与目标层不相交的部分。目标层中除了Alpha通道以外的颜色值都不会被绘制。如果目标层只是一个单纯的不透明背景，那么这个操作的效果就是将源层与目标层都隐藏掉，因为源层的所有部分都与目标层相交。

#### Dst-out



The `dst-out` comp-op will only draw parts of the destination that do not intersect parts of the source. The colors of the source will not be drawn, only alpha channel (the shapes). If your source is completely solid, this operation will completely hide both the source and the destination, since all parts of the source intersect the destination.

`dst-out`操作的效果是只绘制目标层中那些与源层不相交的部分。源层中除了Alpha通道以外的颜色值都不会被绘制。如果源层完全不透明，那么这个操作的效果就是将源层与目标层都隐藏掉，因为源层的所有部分都与目标层相交。

### Src-atop



(译注：第一张图看起来有点问题，怎么地理范围都变化了？)

The `src-atop` comp-op will only draw the source where it intersects with the destination. It will also draw the entire destination. If your destination forms a solid background, the result will be the same as `src-over` (or no comp-op at all).

`src-atop`操作的效果是只绘制源层中与目标层相交叠的部分。但它也会把整个目标层都画出来。如果目标层只是个不透明的背景，那么效果就和`src-over`（或者不应用合成操作）一样。

### Dst-atop



The `dst-atop` comp-op will only draw the destination on top of the source, but only where the two intersect. All parts of the source will be drawn, but below the destination. If your destination forms a solid background, no part of the source will be visible.

`dst-atop`操作的效果是只将目标层中与源层相交的部分绘制在源层的上面，而源层的所有部分都会被绘制在目标层的下面。如果目标层是一个不透明背景，那么目标层就会完全不可见。

## Xor



The `xor` comp-op means ‘exclusive or’. It will only draw parts of the source and destination that do not overlap each other. If either your source or your destination forms a solid layer, neither will be drawn because there are no non-overlapping parts.

`xor`，也就是“异或”操作的效果是只绘制源层与目标层不相交的部分（译注：相交的部分似乎是作全透明处理了，待考证）。如果源层或目标层中有一个是完全不透明的，那么结果将是两个层都不会被画出来，因为二者没有不相交的部分。

## 参考文献

1. Mapbox, [Mapbox Studio Compositing Reference](#)
2. Mapbox, [TileMill Compositing Operations](#)
3. Mapnik, [Compositing](#)

4. Ron Brickmann, *The Art and Science of Digital Compositing, Second Edition: Techniques for Visual Effects, Animation and Motion Graphics*, Morgan Kaufmann, 2008 (这本书被称为是数字合成领域中的圣经。[豆瓣链接](#), [Amazon链接](#))
5. 布林克曼 (著), 谢毓湘等 (译), 数字合成的科学与艺术, 清华大学出版社, 2011 (这是上面这本的中译版, 但不清楚它的翻译质量。另外, 它还有中文引进图书的通病——纸张和印刷质量都比原版差很多, 这对于一本讲述色彩、视觉效果的书来说是致命伤。[豆瓣链接](#), [亚马逊链接](#))

## 3.8 支持CartoCSS的软件和系统

### Mapbox Studio

Mapbox Studio的前身是TileMill, 为Mapbox公司的开源地图制图软件, 其制图所使用的脚本语言就是CartoCSS。它既有基于Web的在线应用, 也有支持各种操作系统平台的客户端软件, 使用JavaScript语言, 基于node.js开发。围绕Mapbox Studio, 还有一系列与CartoCSS相关的开源软件。更多信息请参考Mapbox Studio的[官方介绍](#) (需要翻墙), 以及Mapbox在github上的[开源项目](#)。

### CartoDB

CartoDB (需要翻墙) 是一个以管理地理空间数据为主要目标, 同时也可以将所管理的地理空间数据进行分析、制图渲染并发布的在线系统。从某种意义上说, CartoDB更像是一个GIS, 而且是一个没有桌面客户端的在线WebGIS。与Mapbox一样, CartoDB也维护着一系列[开源项目](#), 而且CartoDB的在线应用本身就是开源的。

更酷的是, CartoDB通过其开源项目[torque](#) (不是那个高性能计算资源管理软件[torque](#)) 实现了对动态过程的可交互式展示, 而且动态过程的样式可以通过CartoCSS进行配置。关于[torque](#)的更多信息, 请参考其[API文档](#)。

### higis

higis是一个利用高性能计算 (High Performance Computing, HPC) 平台实现对地理空间数据的高效管理、分析处理与制图可视化的地理信息系统。它的核心理念是利用HPC提供的大规模混合并行计算环境 (多机、多核、众核) 来提升地理空间信息管理、分析与可视化的效率。关于higis的更多特点, 可参考发表于Geocomputation 2013会议上的文章[HiGIS: When GIS Meets HPC](#)。从技术体系上来说, higis受到了Mapbox与CartoDB中众多开源项目的启发, 但也针对HPC环境进行了修改与定制。其在线制图功能采用CartoCSS对地图进行样式配置。

目前, higis在互联网上暂无可供展示其功能与能力的演示系统。但可以通过一些部署了higis的应用单位来一窥其样貌。这些单位有: 中国国土资源部地质调查局某应用, 上海交通大学OpenSAR处理平台, 湖南省国土资源厅某应用。

## 4. 高级技巧

本章将介绍利用CartoCSS进行地图制图设计的一些高级技巧与方法。

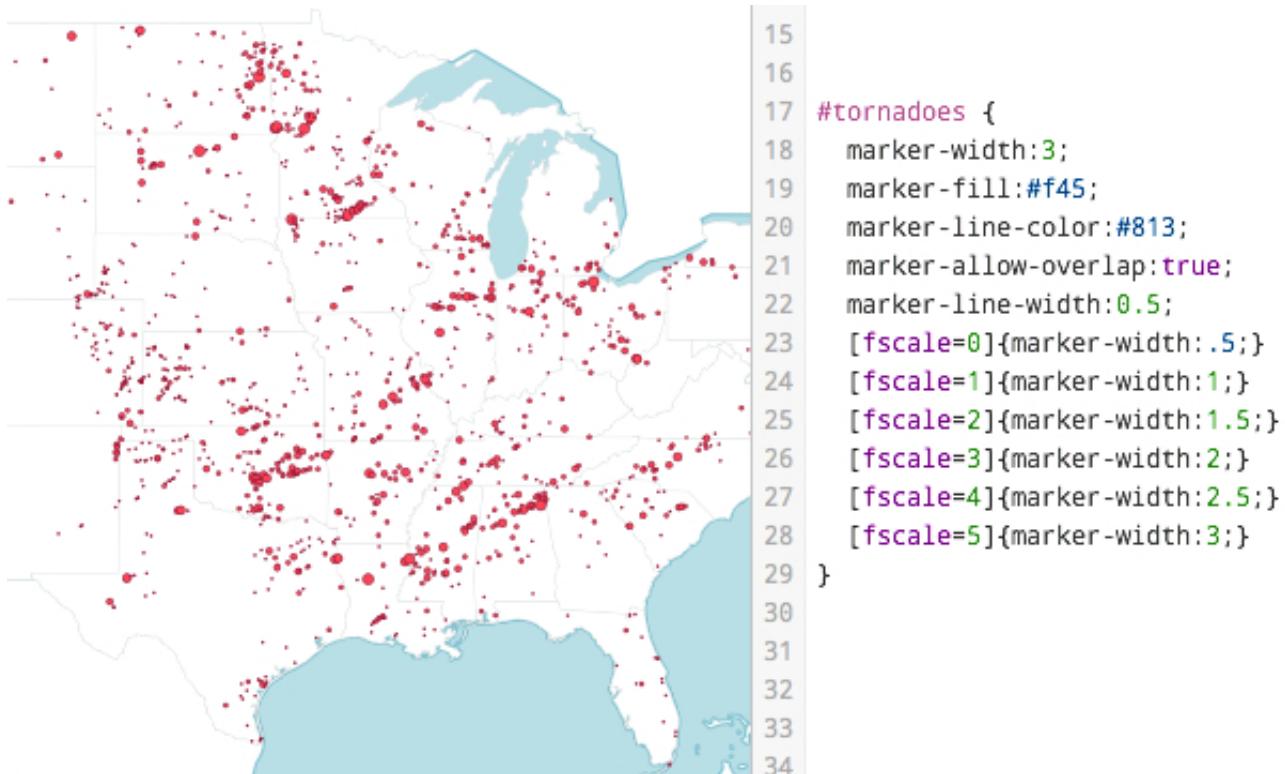
### 4.1 高级地图设计

在本节中，我们会谈到一些运用CartoCSS制图的高级技巧，它们可以让你的地图看起来更加高端大气。本节示例使用的是美国2010年的龙卷风统计数据。关于制图所需数据的准备，有很多途径，比如[利用谷歌文档](#)来做。

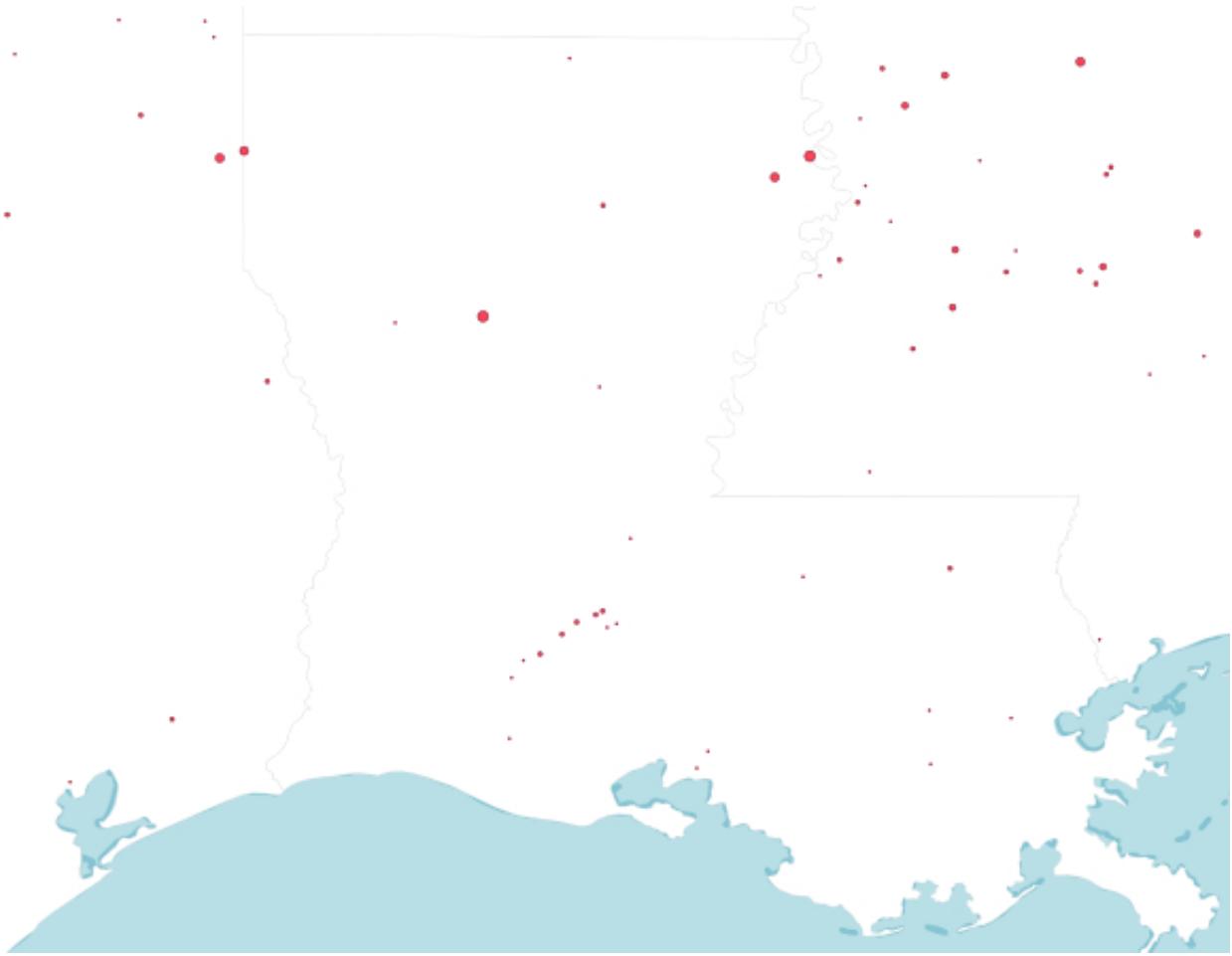
## 不同的级别，不同的样式

交互式地图通过缩放功能提供给用户改变对地图观察尺度的能力。不同的缩放级别对应着地图的不同比例尺或分辨率。在缩放级数较小时，我们观察者距离观测区域较远，视口中的地理范围较大，对应的地图比例尺较小，分辨率也较低；而当缩放级数较大时则是相反的效果。这种能力是目前数字化地图，特别是基于Web的在线地图服务能够提供的最基本的一种交互能力。缩放级别通常是一组离散的数值，这意味着地图的缩放被限制在一组（目前通常最多为20个左右）的固定级别上，而不是连续的“无级变化”。

在地图设计的时候，可以对同一幅地图在其不同的缩放级别上考虑配置不同的样式。我们这里假定制图所需要的[龙卷风点数据已经准备好了](#)，并且配置了一些初始样式。那么现在就来考虑一下怎样才能让这张地图在不同的级别具有不同的样式。下图是缩放到4级时的龙卷风分布图。其中的龙卷风注记符号根据风力（f-scale属性）大小设置了符号的尺寸。

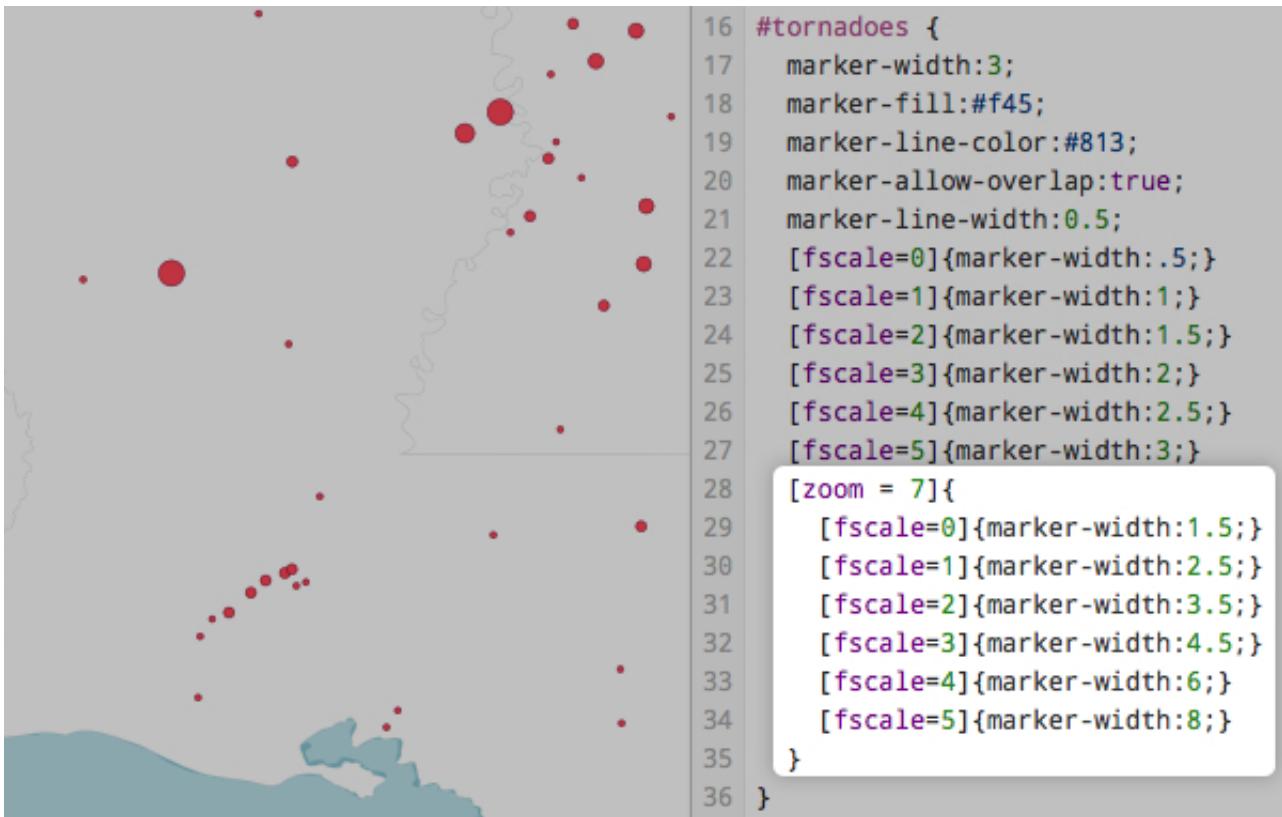


如果把这张地图放大到7级，那么就可以看到那些点符号并不会随着地图放大而变大，因而在这个级别上看起来就显得太小了（如下图）。



那么我们此时请出CartoCSS。只需寥寥几行代码，就可以轻松实现让地图在不同的级别下展示出不同的样式。

下图中高亮部分的CartoCSS代码的含义是说：“当缩放级别到达7级时，应用下面这段样式。”你可以根据自己的需要添加任意数量这样的样式块。这样一来，就可以让地图上的点、图标和标注随着地图放大而适当变大，从而美观的展示出更多细节。



The following symbols are allowed in conditional statements: =, !=, >, >=, <, <= You can also group by zoom ranges by setting a beginning and an end, like this:

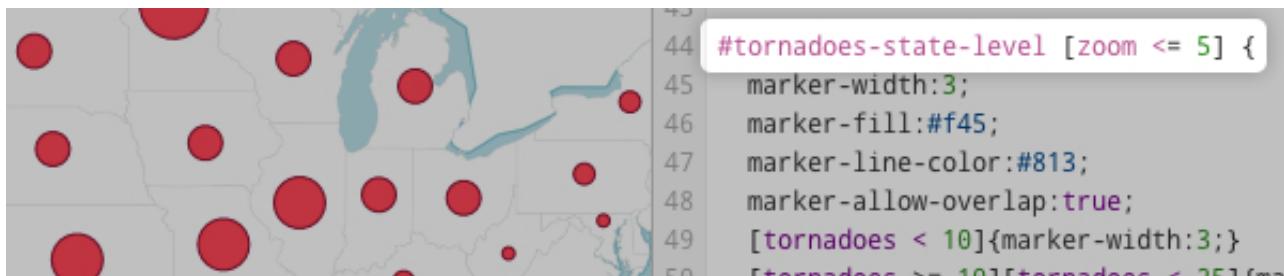
在缩放级别过滤器的条件表达式中可以使用比较运算符: =, !=, >, >=, <, <=, 还可以定义一个缩放级别范围, 就像这样:

```
[zoom >= 4][zoom <=8] {
  ...styling...
}
```

利用缩放级别过滤器还可以控制某些图层在特定级别上的可见性, 从而使这些图层随着地图的放大或缩小被“打开”或“关闭”。这个方法对于包含多个地理等级数据(比如多个行政区划等级)的地图的样式配置很有用。

我们还以龙卷风分布图为例。在国家级尺度上, 视口部分的地图比例尺很小, 因此会出现一些聚在一起的点, 它们形成一些杂乱的点团而让人难以辨认那里究竟都些什么。在这个等级上, 可以将每个州按照各自龙卷风的总数统计出来然后只用一个点来表达, 并且该点的尺寸与龙卷风总数正相关。而随着用户不断放大地图, 从适当的级别开始就不再显示这个国家级尺度的图层, 而是可以将原始的独立龙卷风点分别绘制出来。

具体的实现方法是, 利用[数据透视表](#)从原始数据集中聚合出州一级的龙卷风统计数据, 然后对这个新的数据集[地理编码](#), 赋予其地理位置信息。接下来要做的就是把这个新的点数据集作为图层加入地图, 然后就可以继续用缩放级别过滤器为其配置CartoCSS样式了。



用户还可以使用与缩放级别过滤器相同的语法来从数据集中基于某些字段设置其它的过滤条件，从而筛选出一些满足特定条件的要素记录。下面的样式语句只显示#tornadoes图层中俄克拉何马州的数据。其中的"state"是该图层属性数据中的一个字段，它包含了各州的缩写。

```

#tornadoes [state = "OK"] {
  ...styling...
}

```

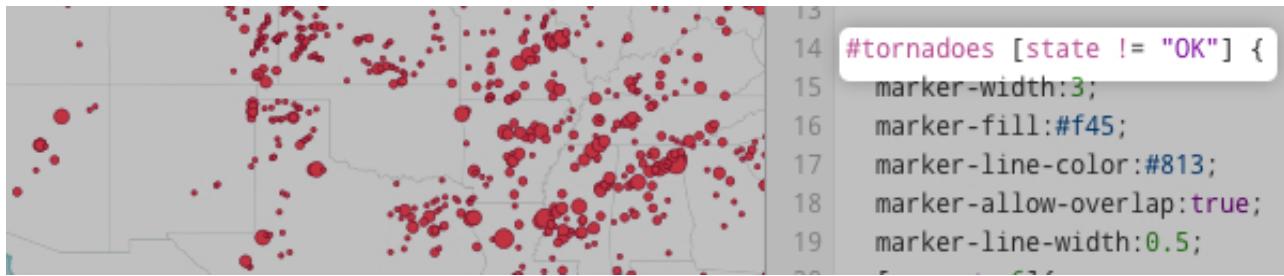


当然，还可以反向选择，例如显示除俄克拉何马以外其它各州的数据：

```

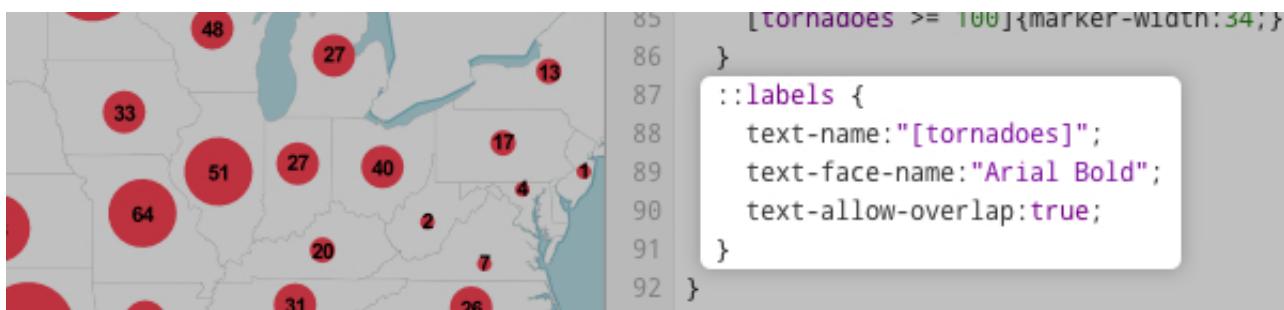
#tornadoes [state != "OK"] {
  ...styling...
}

```



## 文本标注

为了能够在地图上更直观的展示信息，我们可以为想要在地图上表达的内容加上数字或文字形式的标注。标注的样式非常丰富（详见4.3节），对于本例中的点标注来说，既可以将标注文本填在其对应的圆点当中，也可以不画圆点而只绘制标注。在这张龙卷风分布图上，我们是将发生龙卷风的总次数标记在每个州一级的圆点里面。为了实现这种效果，需要再加几行CartoCSS代码：



### 1. ::label

从前面第三章的基本概念介绍中我们已经了解了这个用双冒号::开头的标记定义了一个从属样式块(attachment)，这里的从属样式名label是任意的，并不是关键字。而从属样式在整个样式表中的定义位置则决定了它被绘制的次序。在CartoCSS的样式块中，遵循“先定义，先绘制”的原则。而先绘制的要素会被后绘制的要素压在下面。因此，如果想把某一层绘制到地图的最上层，那么就应该把它的样式定义代码写在最后。

### 2. text-name

指明作为文本标注内容的属性数据字段。

### 3. text-face-name

设置标注中文字的字体。

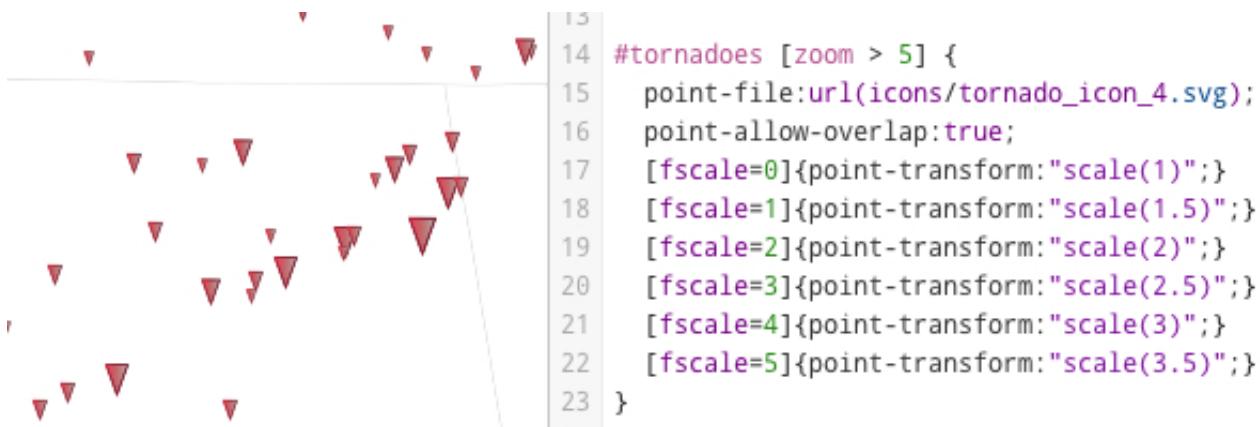
### 4. text-allow-overlap

这个属性的作用是允许文本标注和圆点标记重叠显示。注意它的默认值是false，也就是不允许重叠，我们这里把它改成了true。

用上面这些样式属性来配置基本的文本标注已经足够了。用同样的方法还可以配置地名标注。当然，如果需要，那么你还可以通过text-系列属性进一步调整标注的其它样式，例如字号、颜色、透明度、位置偏移等。

## 使用自定义图标

CartoCSS支持将外部的SVG格式图片作为地图上的注记图标。在本节的例子中，我们可以使用自定义的SVG图片来代替圆点形状注记。当然首先，我们需要先要有一个保存在系统中能够被制图项目引用到的SVG文件。为了方便管理，最好把它就放在项目的目录中。



### 1. point-file

明确SVG文件的路径。

### 2. point-allow-overlap

跟其它-allow-overlap参数一样，这个属性的作用也是允许点符号图标重叠显示。

### 3. point-transform

这个参数用来对注记图标进行拉伸与移位。使用"scale(1)"将按照原始比例绘制图标，而"scale(0.5)"和"scale(2)"则将分别按照原始图片尺寸的50%与200%来绘制。除了拉伸，还可以用"translate()"来移位，例如，可以用"translate(20, -40)"将图片向右平移20像素、向上平移40像素。除此以外，point-transform的其它属性可参考[W3上的相关文档](#)。

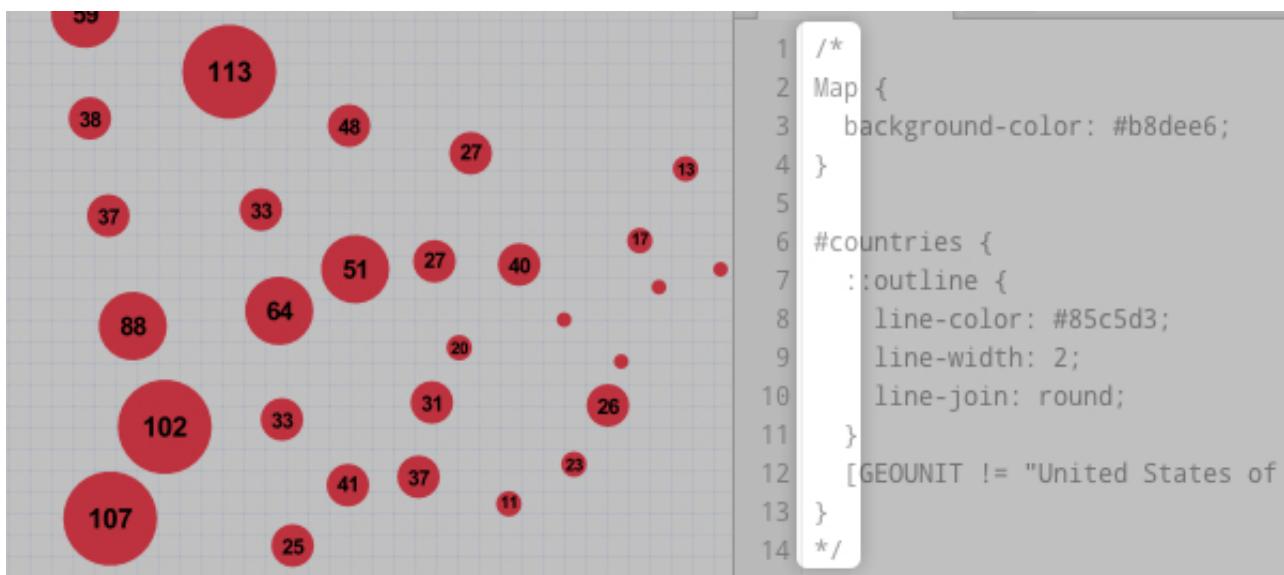
## 导出与合成

在完成制图样式的配置后，最后可以将地图以瓦片数据集（MBTiles）的形式导出。导出的瓦片数据集中包含了按照所配置的样式渲染后的地图以及图例、浮动工具等内容。导出的瓦片数据集是一种可移植的外存文件格式，可以被直接用于很多瓦片地图服务。需要强调的是，导出到瓦片数据集这个功能本身并不是CartoCSS的一部分，但在实际的制图工作流程中是重要的一环。试想一下，当你用CartoCSS设计好了一幅地图，然后怎么才能让其他人看呢？总不能每次都打开制图工具吧。所以，最自然的做法就是把制好的图保存成一种可移植、可发布的格式，然后拷贝或上传到一个支持瓦片地图服务的服务器上，这样才能让别人能够在线观赏你制作的精美地图。因此，导出这个功能通常是支持CartoCSS的制图工具提供的一项基本功能。

如果你的地图包含多个等级的数据，就像本例中的龙卷风分布图一样，那么我们建议把不同等级的数据分别单独导出。这样做好处有三点：首先，它将地图划分成多个可以独立更新的部分，因而在其中某个部分发生变化（比如更新了样式）的时候无需将整张图全部重新导出；第二，按照不同等级拆分开的独立数据集可以更加灵活的进行合成操作（参见3.7节）；第三，地图的交互效果（参见4.4节）一次只能应用在一个图层上，还以龙卷风分布图为例，比如说我想在州和原始点两个级别上都提供浮动工具条的交互效果，那么就不能把这两个级别的瓦片数据集打包导出，而必须要分开才行。

当需要将制图项目中的某一部分单独导出的时候，有个技巧是对其它部分的CartoCSS代码灵活运用注释。被注释掉的样式不会被渲染。CartoCSS中的注释也是使用/\*和\*/将被注释的部分代码包围。

前面既然谈到了“灵活的合成操作”，那么我们现在就来把本节的示例中的龙卷风分布图与一张全球底图合成。为此，我们先把原来项目中的蓝白色全球底图与国界线样式代码注释掉。注意图中/\*和\*/之间的代码已经变成灰色的，而此时的地图背景变成了全透明。



现在我们为州一级的圆点符号创建浮动工具条（参见4.4节），然后将其导出。在配置导出参数时，要根据实际需要和地图本身的特点（比如地理覆盖范围、最高分辨率等），想好要导出地图中的哪些级别。如果选择的级别范围不合理，那么可能会造成导出的数据量过大（对那些本来没有那么大地理覆盖范围的图选择了较多的小比例尺级别）等问题。



接下来，我们为原始的龙卷风统计点创建浮动工具条，然后将其导出。在配置导出参数时，将缩放级别范围和导出的数据文件名做相应修改。



这样，我们就得到了两个都具有交互能力（浮动工具条）的瓦片数据集，而且还可以将它们合成（参见3.7节），并叠加到一幅在线全球底图上。最终的地图效果可从[这里](#)看到。用于这幅地图的完整CartoCSS代码如下：

```

Map {
  background-color: #b8dee6;
}

```

```

#countries {
  ::outline {
    line-color: #85c5d3;
    line-width: 2;
    line-join: round;
  }
  [GEOUNIT != "United States of America"]{polygon-fill: #fff;}
}

/*Individual tornado points*/
#tornadoes [zoom > 5]{
  marker-width:6;
  marker-fill:#f45;
  marker-line-color:#813;
  marker-allow-overlap:true;
  marker-line-width:0.5;
  [zoom = 6]{
    [fscale=0]{marker-width:2.5;}
    [fscale=1]{marker-width:4;}
    [fscale=2]{marker-width:5.5;}
    [fscale=3]{marker-width:7;}
    [fscale=4]{marker-width:9;}
    [fscale=5]{marker-width:12;}
  }
  [zoom = 7]{
    [fscale=0]{marker-width:4;}
    [fscale=1]{marker-width:6;}
    [fscale=2]{marker-width:8;}
    [fscale=3]{marker-width:11;}
    [fscale=4]{marker-width:14;}
    [fscale=5]{marker-width:18;}
  }
  [zoom = 8]{
    [fscale=0]{marker-width:6;}
    [fscale=1]{marker-width:9;}
    [fscale=2]{marker-width:12;}
    [fscale=3]{marker-width:16;}
    [fscale=4]{marker-width:22;}
    [fscale=5]{marker-width:30;}
  }
}

/*State-level dots and labels*/
#tornadoes-state-level [zoom <= 5] {
  marker-width:6;
  marker-fill:#f45;
}

```

```
marker-line-color:#813;
marker-line-opacity:0;
marker-allow-overlap:true;

[zoom = 3]{
    [tornadoes < 10]{marker-width:6;}
    [tornadoes >= 10][tornadoes < 25]{marker-width:10;}
    [tornadoes >= 25][tornadoes < 50]{marker-width:16;}
    [tornadoes >= 50][tornadoes < 100]{marker-width:24;}
    [tornadoes >= 100]{marker-width:16;}
}

[zoom = 4]{
    [tornadoes < 10]{marker-width:7;}
    [tornadoes >= 10][tornadoes < 25]{marker-width:12;}
    [tornadoes >= 25][tornadoes < 50]{marker-width:20;}
    [tornadoes >= 50][tornadoes < 100]{marker-width:32;}
    [tornadoes >= 100]{marker-width:44;}
}

[zoom = 5]{
    [tornadoes < 10]{marker-width:10;}
    [tornadoes >= 10][tornadoes < 25]{marker-width:18;}
    [tornadoes >= 25][tornadoes < 50]{marker-width:28;}
    [tornadoes >= 50][tornadoes < 100]{marker-width:44;}
    [tornadoes >= 100]{marker-width:68;}
}

::labels {
    text-name:"[tornadoes]";
    text-face-name:"Arial Bold";
    text-allow-overlap:true;
}

[zoom = 3]{
    [tornadoes < 25]{text-opacity:0;}
}

[zoom = 4]{
    [tornadoes < 10]{text-opacity:0;}
    [tornadoes >= 10][tornadoes < 25]{text-size:8;}
    [tornadoes >= 25][tornadoes < 50]{text-size:10;}
    [tornadoes >= 50][tornadoes < 100]{text-size:11.5;}
    [tornadoes >= 100]{text-size:13;}
}

[zoom = 5]{
    [tornadoes < 10]{text-size:8;}
    [tornadoes >= 10][tornadoes < 25]{text-size:10;}
    [tornadoes >= 25][tornadoes < 50]{text-size:11.5;}
    [tornadoes >= 50][tornadoes < 100]{text-size:13;}
    [tornadoes >= 100]{text-size:16;}
}
```

```
}

/* State borders */
#states {
  line-color:#ccc;
  line-width:0.5;
  polygon-opacity:1;
  polygon-fill:#fff;
}
```

## 参考文献

1. Mapbox, [Advanced Map Design](#)

## 4.2 地图配色技巧

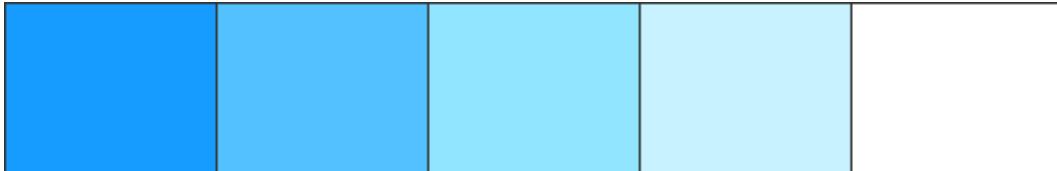
### 色彩理论与制图

一个恰当的配色方案对于一幅地图来说至关重要。作为地图设计师，我们实际上是在通过设计一幅地图来讲述一个故事。而地图的配色将会在很大程度上影响这个故事的精彩程度。本节将简单介绍一些关于如何在地图上运用色彩的理论。我们先来看三种用于制图的基本配色方案。

#### 连续色方案（译注：还是“渐变色方案”？）（**Sequential Schemes**）

Sequential schemes order data from high to low, accenting the highest as a dark shade and the lowest as a light shade (or vice versa). Sequential schemes are best if you are mapping quantitative data and do not want to focus on one particular range within your data.

连续色方案先将地图数据依次排序，然后对高位数据用深色、低位数据用浅色表示（或者反过来）。这种配色方案特别适用于地图数据是数值类型而且不需要对数据中的某个区间特别强调的情况。



#### 发散色方案（**Diverging Schemes**）

Divergent schemes are best at highlighting a particular middle range of quantitative data. Pick two saturated contrasting colors for the extremes of the data, and the middle ranges blend into a lighter mix of the two. This is particularly great for accenting the mean of your data and exposing locations that significantly ‘diverge’ from the norm.

发散色方案特别适用于突出强调数据中的某个中间区间。要构造这种方案，需要先选择两种饱和度对比色作为色带的两端，然后基于这两个端点颜色的混色构造色带的中间部分。这种配色方案适合用来强调显示数据中的均值，以及那些不符合正常分布的毛刺数据（译注：这两个半句的意思不矛盾吗？）



## 定性配色方案 (Qualitative Schemes)

If you are working with qualitative data, such as ethnicity or religion, you want to pick a series of ‘unrelated’ colors. The trick is to pick a really nice color theme so your map looks great. You can also accent particular aspects of your data by your choice of color. For example, one strong dark color among a group of lighter colors will ‘pop’ out of the map, highlighting that particular facet of your data against all others.

如果处理的数据是定性数据，比如种族、宗教等这种属性数据，那么用一组互不相关的颜色会比较合适。遇到这种情况时，选一组漂亮的颜色可以让地图看起来很赞。另外，还可以通过选取特定的颜色来凸显数据中你想要强调的内容。举个例子，在一组较浅的颜色中，使用深黑色标记的部分会在地图上很乍眼，从而可以这部分特定的数据更加引人注意。



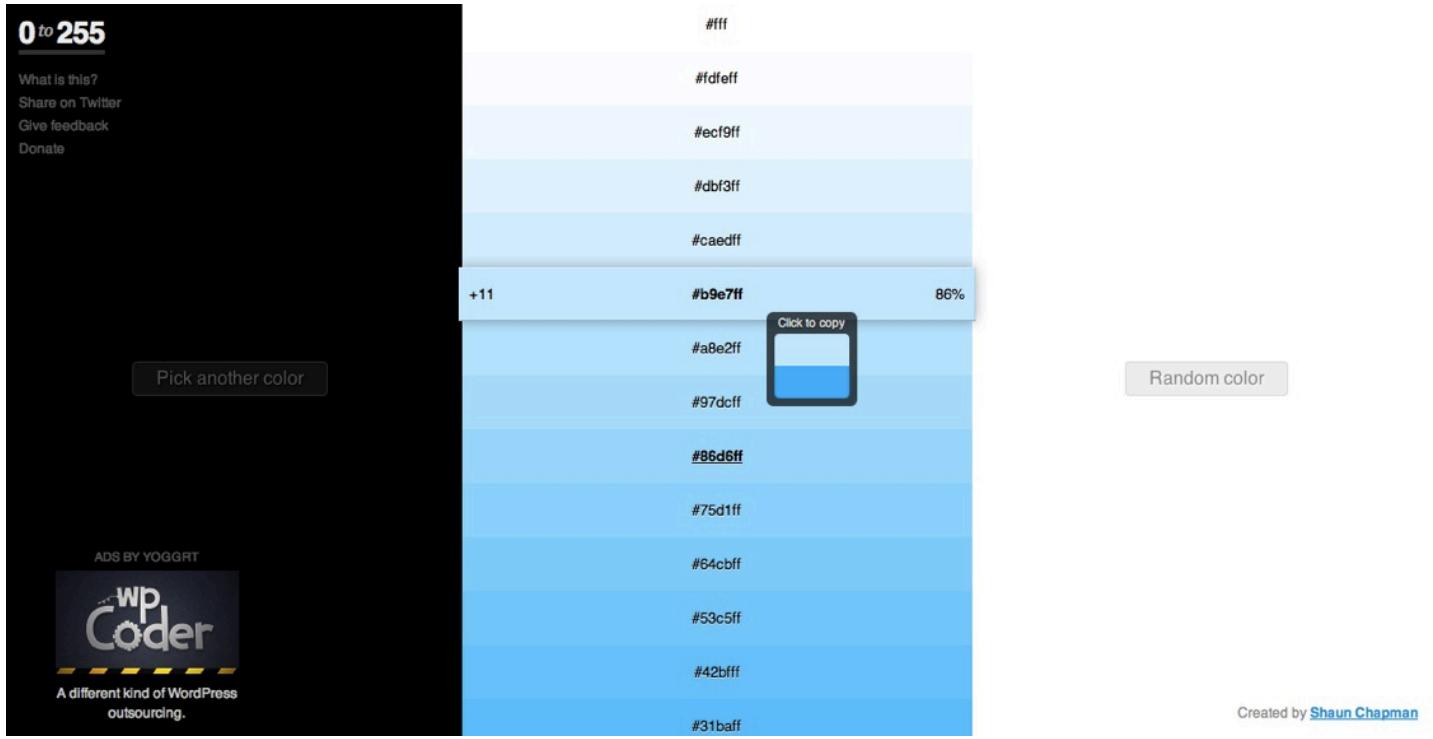
## 常用的取色工具

有不少工具应用可以辅助我们选择颜色或设计配色方案。这里简单介绍几个典型的工具应用及其主要功能。要想让地图具有更强的表现力，那么利用这些辅助工具可以助你一臂之力。它们可以帮助你设计出恰当的配色方案，让你的地图讲出的故事更加精彩。这些应用中有在线版的，也有桌面版的，最后两个是付费应用。

### 0\_255

0\_255 is a great site for picking between different shades of one color, so it's ideal for sequential schemes. It gives you thirty options for any given color, allowing you to instantly copy the color's hex code. There is a large grid of colors to select from for an initial color, and then 0\_255 visualizes a range of shades based on that color. If the grid does not have the color you want, you can pick your own color by pasting in the hex code of your chosen color.

0\_255是一个生成渐变色带的在线工具。它可以为同一色系生成一组亮度不同的颜色，因此非常适用于连续色方案。它为任意一种颜色都提供了从全黑（#000）到全白（#FFF）共32种亮度渐变的不同颜色。对于生成的每种颜色，都可以直接拷贝其十六进制的颜色值。用户可以先从一个很大的颜色网格中选取一种初始色，然后0\_255就可以基于这种颜色生成一条亮度渐变的色带。如果在颜色网格中没有用户想要的颜色，那么也可以通过把自定义的十六进制颜色值粘贴进去来选取自己的颜色。



## Color Scheme Designer

Color Scheme Designer allows you to select a color from a color wheel and presents several options for automatically generating colors complementary to your initial choice. Specifically, you can choose between:

Color Scheme Designer 通过让用户在一个色盘上选取颜色和配置来生成所选颜色的补色（译注：有专业术语吗？）。具体而言，用户可以选择以下五种生成补色的模式：

- Complements
- Triads
- Tetrad
- Analogic
- Accented Analogic

Once you've chosen your color and scheme, you'll have a color table that gives you several variations of your colors and their corresponding hex codes.

在用户选定了初始色和模式之后，就可以得到一个包含了几种变化的颜色表，其中的每个颜色值都有十六进制的编码。



## Colour Lovers

Colour Lovers is great if you are looking for a design theme for mapping qualitative data. Active users contribute palettes to the site, and these palettes are searchable, browsable, and ready to be used on your project. Colour Lovers users also post patterns if you need some spatial inspiration as well.

Colour Lovers 是一个为定性数据生成配色方案的出色应用。有很多活跃的用户为其贡献色板。在网站上可以对这些色板进行搜索和浏览，它们可以直接被用于你的项目中。如果你需要一些空间上的灵感，那么还可以从 Colour Lovers 上找到其他用户发布和分享的样式（译注：指那些可以用于填充多边形或背景的样式）。

The COLOURlovers Family is Getting Bigger... Awesome Investors &amp; We're Hiring!

X

[Browse](#) [Create](#) [Search](#) [Community](#) [Channels](#) [Trends](#) [Tools](#) [Store](#)

## Share Your Color Ideas & Inspiration.

COLOURlovers is a creative community where people from around the world create and share **colors**, **palettes** and **patterns**, discuss the latest **trends** and explore colorful **articles**... All in the spirit of love.

[!\[\]\(7817a33ddd47907eb851844d5c033e61\_img.jpg\) Join the Community!](#)
[ALL](#) [WEDDING](#) [HOME](#) [FASHION](#) [WEB](#) [PRINT](#) [CRAFT](#) [BUSINESS](#)

### LATEST BLOG POSTS



The COLOURlovers Family is Getting Bigger... Awesome Investors & We're Hiring!

 20 Comments



Contest: Mothers Day Floral Template Gets You ImageKind Bucks!

 50 Comments



24 Smart Logo Color Designs

 5 Comments

[View More >](#)

## ColorSchemer • COLOURlovers



- LiveSchemes
- CMYK support
- PC or Mac
- Much More...

[DOWNLOAD »](#)



## Kuler

Kuler is a service similar to Colour Lovers offered by Adobe. A community of designers submit their own themes, which are available for RGB values and hex codes. The design of the site is a little less intuitive than Colour Lovers, but still worth checking out.

Kuler是一个Adobe公司旗下的与Colour Lovers功能相似的网站。上面聚集的一些设计师会发布一些自己的方案，其中的颜色值由RGB或十六进制表示。Kuler网站的设计不如Colour Lovers直观，但仍然值得收藏。

The screenshot shows the Kuler website interface. At the top, there's a navigation bar with the Kuler logo, 'Register', and 'Sign In'. Below the header is a large color palette consisting of four vertical bars: dark red, maroon, slate blue, and forest green. Underneath this palette, there's a search bar and a sidebar with links like 'Create', 'Themes - Last 30 days', 'Newest', 'Most Popular', 'Highest rated' (with an RSS icon), 'Random', 'Community', 'Pulse' (beta), and 'Links'. To the right of the sidebar, a specific theme titled 'Scish' by 'riesjart' is displayed. The theme has a rating of 5 stars. Below the title are five color swatches with names: 'MINT CHOCOLATE ...', 'Black Swan', 'philosophy', 'Mexican Spice', 'veritas', 'Wild Plains', and 'orange brownie'. Each swatch has a 5-star rating. To the right of the theme details is a 'News & Features' section with various links and information about Kuler's integration with Adobe Creative Suite and its mobile app.

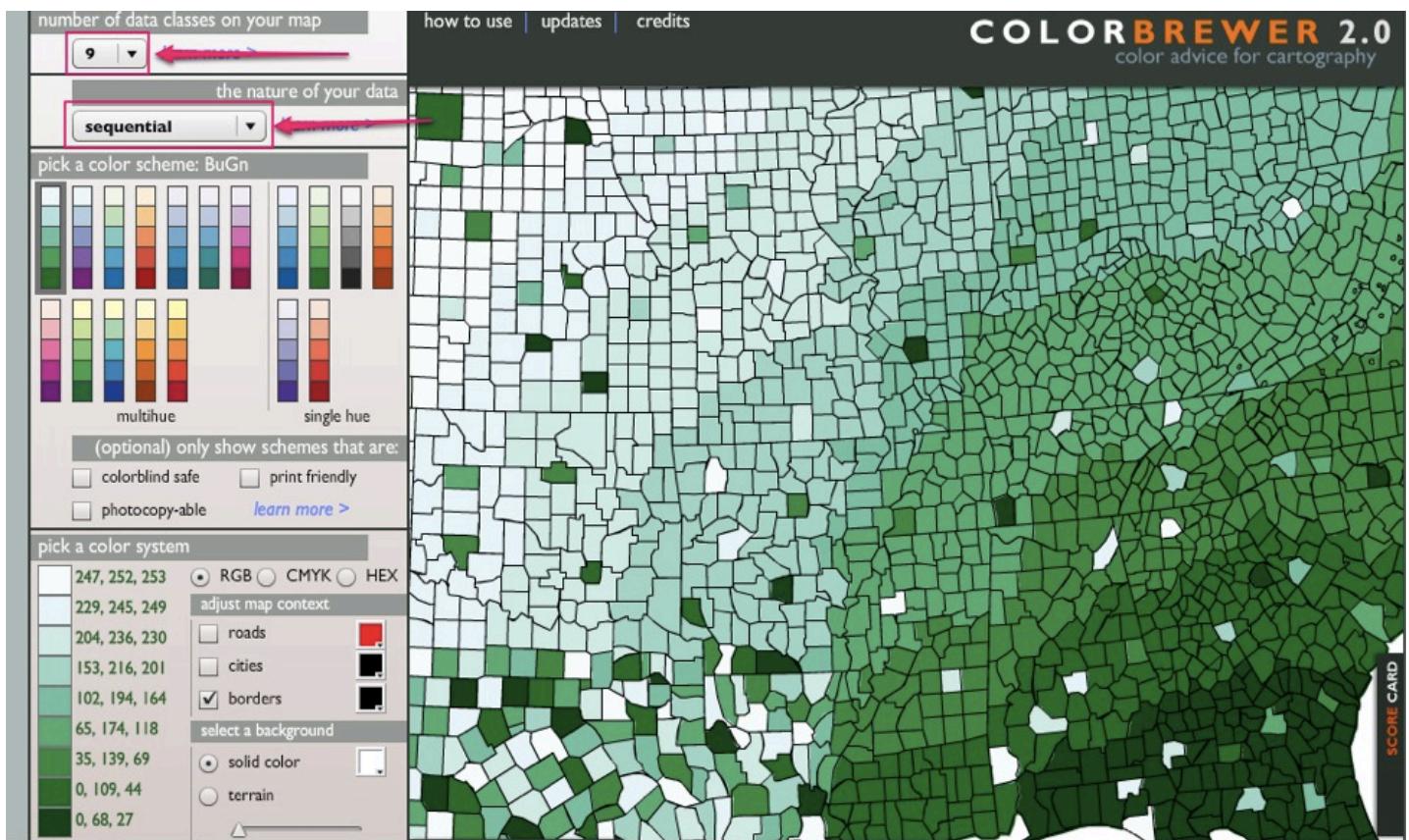
There are a lot of great ideas being posted by an active community of contributing users. Kuler also provides extensive links to things like forums, help pages, as well as several articles on the importance of color and color theory.

Kuler上面有很多很棒的想法。它们都是由一大群活跃的用户发布的。除此以外，Kuler还提供了很多有用的外部资源链接，包括论坛、在线帮助，以及多篇关于颜色和色彩理论重要性的文章。

## Colorbrewer

Colorbrewer is great for contrasting the three kinds of color models mentioned in the first section of this article. It allows you to test out different color themes based on whether you want sequential, diverging, or qualitative schemes and to vary the number of color classes you want in your pallet (up to 12). It also provides some useful ‘further reading’ articles on these theories and other cartographic design ideas.

Colorbrewer是个用来对比本节之前提到的三类色彩方案的好工具。它可以对不同颜色方案进行对比和测试，包括连续色、发散色和定性配色等方案，支持最多12种选定颜色。此外，它还提供了一些关于色彩理论和制图设计思想方面的扩展阅读文章，值得一读。



## Pochade

Pochade is a color picking program that allows you to determine the RGB value and hex code of any color you see on your screen. It also provides a few different ways of manipulating your color, such as changing its HSB, RGB, or CMYK values. This program is available for download for \$9.99.

Pochade是一个取色工具，能够获取屏幕上任意位置的颜色值（RGB值或十六进制值）。它支持通过HSB、RGB或CMYK等多种模型来修改颜色。Pochade是付费软件，价格9.99美元。



## Pochade Select any color off your screen



- ✓ Easily select any color from your screen
- ✓ Save your collection of colors
- ✓ Several ways to pick the color you want
- ✓ Get the color value the way you want

[Download](#)
[Purchase \\$9.99](#)


### Requirements

- ✓ Intel Processor
- ✓ OS X 10.6+

## ColorSchemer Studio

ColorSchemer Studio provides three main features: (1) a color class generator (up to 254 - many more than Color Brewer) based on two colors of your choice, (2) a ‘PhotoSchemer’, which allows you to upload a photo and determine up to ten different colors on your chosen photo and (3) integration with colourlovers, including a color browser and the ability to load colors into the first tool to manipulate on your own. ColorSchemer is a powerful tool for your desktop, available for the slightly higher price of \$49.99.

ColorSchemer Studio主要有三个功能：(1) 基于两种预选颜色的色系生成器（支持最多254种颜色，远远多于Colorbrewer）；(2) “PhotoSchemer”可以从一张现有的图片中提取色系；(3) 与Colour Lovers集成，包括一个颜色浏览器和向第一个工具（译注：first tool指的是什么？）中加载颜色的能力。ColorSchemer是个强大的桌面应用，但价格稍贵，要49.99美元。

## 参考文献

1. Mapbox, [Tips for using color in maps](#)

## 4.3 高级标注方法

### 试探标注位置

在默认情况下，如果对一个点符号进行文本标注，那么标注将被绘制在原始几何点的位置处。这在大部分情况下没什么问题，但有的时候我们会希望从地图总体设计的角度去考虑标注的摆放问题，比如在标注比较密集的地方让它们向周围散一散，让POI的标注尽量避开主干道路网，等等。在这种时候，我们希望允许标注可以不必精确绘制在原始点的位置，而是可以稍有偏移。这种需求在CartoCSS中是可以实现的。在CartoCSS

中，标注的位置可以通过text-placement-type属性来配置，它目前能够支持两种标注方案：一是none（该属性的默认值），效果是将标注标在原始点处；二是simple，别看它名字叫“simple”，实际上却是个高级标注方法。

simple方法允许设计师在原始点的周围为标注另外指定几个候选的摆放位置和字号大小。如果在默认位置渲染标注时会与现有的其它标注冲突，那么就从这些指定的其它候选位置中逐个尝试绘制，直到可以把标注绘制出来为止。而如果在所有候选位置都无法绘制，那就不画它了。

下面是一个较为完整的例子：

```
#labels {
    text-name: "[name]";
    text-face-name: "OpenSans Regular";
    text-placement-type: simple;
    text-placements: "N,S,E,W,NE,SE,NW,SW,16,14,12";
    text-dy: 3;
    text-dx: 3;
}
```

这段代码的意思是依次在原始点的上方（北方）、下方、右侧、左侧等八个位置尝试以16号字绘制标注。如果这些位置都不行，那么就换成14号字再试一遍，如果还不行，就再换成12号字试。如果上面这些尝试都画不出来，那么就跳过不画这个标注了。

text-dx和text-dy属性指定了标注位置相对于原始点位置的偏移量（以像素为单位）。

## 改进标注的排布：随机法

尽管前面这个在多个位置尝试绘制标注的想法不错，但它的问题在于每次的试探位置序列都是完全一样的，例如在上面的例子中就是N,S,E,W,NE,SE,NW,SW 这个顺序。从地图整体美感的角度来看，这通常不是最好的方案，即使标注都能找到自己在地图上的“容身之所”。那么为了能从地图总体设计的角度来将所有的标注合理排布，我们需要考虑另外的试探绘制方法，比如不总是按照同样的位置序列来尝试画标注。

Something as simple as randomly assigning a direction bias can help even out the look of the labels. For example, you could create a PostGIS query that creates a column called dir which is randomly assigned a value of either 0 or 1.

有个很简单的改进方法，就是只要将试探绘制的位置序列由固定的改为随机的便可以让标注的分布变得美观许多。具体而言，你可以利用一条PostGIS的SQL语句为原始点数据增加一个名为dir的新列（译注：注意这不是说要去修改原始数据，而是通过SQL语句生成一个临时的属性列），它的值是随机生成的0或1。

```
(select *, floor(random()*2) as dir from city_points) as data
```

然后就可以基于dir的值，让每个点的标注试探位置序列分别在dir等于0时为E,NE,SE,W,NW,SW；等于1时为W,NW,SW,E,NE,SE。

```
#labels {
    text-name: "[name]";
    text-face-name: "OpenSans Regular";
    text-placement-type: simple;
```

```
text-placements: "E,NE,SE,W,NW,SW";
[dir=1] { text-placements: "W,NW,SW,E,NE,SE"; }
}
```

## 改进标注的排布：邻居避让法

前面我们稍稍利用了一下PostGIS和SQL，就得到了一种让标注排布更加合理的方案。其实那只是PostGIS和SQL强大能力的冰山一角。现在就让我们再深入一点，利用它们实现一种比随机法更加美观合理的标注排布方案——邻居避让法。在这种方法中，先找到距离当前标注点最近的邻居对象，看看它的标注是向哪偏移。然后在绘制标注时尽量避开这个最近邻居的标注。例如，在为城市标注名称时，可以让每个城市的名称都稍作偏移以避开离它最近的另一个城市，而在标注地区名称时则尽量让其避开该地区中最大城市的名字，以防止出现标注冲突导致的无法绘制。这些方法和实践未必是最佳方案，但对于大多数情况来说可以让你的标注排布更加合理。

这里我们讨论一个应用邻居避让法的典型场景。对于那些正好位于[城市街区](#)边缘附近的兴趣点，它们的名称完全可以尽量标注在街区覆盖的区域中，而避开其临近的城区街道。将这些标注置于街区区域还可以保持街道名称和通行方向等道路标注信息清晰可见。那么如何达到这种效果呢？思路并不复杂。对于每个点标注，找到距离它最近的城市街道以及这条街道相对于它的方位。在搜索最近邻街道的时候可以忽略一些低等级道路（例如OpenStreetMap数据集中的service streets、tracks、footways和cycleways等），但也可以根据实际情况对避让策略进行调整。在大部分时候，将标注置于一条小巷或公园小路上是可以接受的，但城市主干道不应该被其附近的点标注压盖。

那么又如何利用PostGIS和SQL来具体实现呢？在PostGIS中，有一系列空间操作函数，可以帮助我们实现上面的避让策略：

- `ST_Distance`函数可以帮助我们找到距离一个兴趣点最近的道路
- `ST_ClosestPoint`函数则可以找到在最近的道路上的最近的几何形点
- `ST_Azimuth`函数可以帮助我们计算从当前兴趣点到其最近形点的方位夹角

利用这些函数，我们可以写一个PostgreSQL函数。但需要特别说明的是，这个函数假设你已经通过[osm2pgsql](#)准备好了个标准的OpenStreetMap数据库，其中的属性和值都是针对OpenStreetMap数据结构的。你当然可以对它进行修改以适应其它的数据库结构。

```
create or replace function poi_ldir(geometry)
  returns double precision as
$$
select degrees(st_azimuth(st_closestpoint(way, $1), $1)) as angle
from planet_osm_line
where way && st_expand($1, 100)
  and highway in ('motorway', 'trunk', 'primary', 'secondary', 'tertiary',
                  'unclassified', 'residential', 'living_street', 'pedestrian')
order by st_distance(way, $1) asc
limit 1
$$
language 'sql'
stable;
```

函数最前面的两行定义了名称poi\_ldir、参数和返回值。函数体从\$\$符号处开始。调用poi\_ldir时需要传入一个几何点要素作为参数，而后将距离这个几何点最近的道路（道路类型由where子句确定）与该点之间夹角的角度算出并返回，结果的取值范围为0到360度。（注：ST\_Azimuth()函数本来返回的是弧度，但为了在CartoCSS中方便使用，我们将其转成了角度值。）

如何让这个函数在数据库中可用呢？很简单，只要把它先存入一个文本文件（例如以poi\_ldir.sql文件保存在桌面上）然后在终端中执行以下命令（译注：当然这里假定你使用的是Mac OS或Linux等\*nix类型的系统，如果是MS Windows则需要调整文件路径），那么这个函数就被创建在你的数据库your\_database\_name中了。当然，通过一些PostgreSQL的图形化客户端（例如pgAdmin、phppgsql等）可以利用菜单项中的创建函数功能来实现。

```
psql -f ~/Desktop/poi_ldir.sql -d <your_database_name>
```

然后在制图过程中就可以使用这个函数了。以下这个查询语句将数据库中所有的设施和商店点要素取出来，结果中包括了每个要素的名称和名为ldir的属性列。ldir属性列就是通过poi\_ldir函数计算出来的结果。

```
( select way, name, poi_ldir(way) as ldir
  from planet_osm_point
  where amenity is not null or shop is not null
) as pois
```

接下来，在CartoCSS样式表中怎么使用ldir属性列呢？在将text-placement-type属性设置为simple之后，内嵌一组基于ldir值的过滤器以调整text-placements属性的值。在下面的样式表例子中，每个标注只需在一个候选位置尝试绘制。

```
#poi [zoom > 15] {
  text-name: '[name]';
  text-face-name: @sans_medium;
  text-size: 12;
  text-fill: #222;
  text-wrap-width: 60;
  text-wrap-before: true;
  text-halo-radius: 2;
  text-halo-fill: #fff;
  text-min-distance: 2;
  text-placement-type: simple;
  text-dx: 5;
  text-dy: 5;
  text-placements: 'N';
  [ldir >= 45][ldir < 135] { text-placements: 'E'; }
  [ldir >= 135][ldir < 225] { text-placements: 'S'; }
  [ldir >= 225][ldir < 315] { text-placements: 'W'; }
}
```

将上面这段样式应用在一段完整的OpenStreetMap数据样式表中之后，可以看到其中绝大部分的点标注都避开了道路网。



## 参考文献

1. Mapbox, [Advanced Label Placement](#)

## 4.4 使用图例

### 简单图例与气泡悬浮框

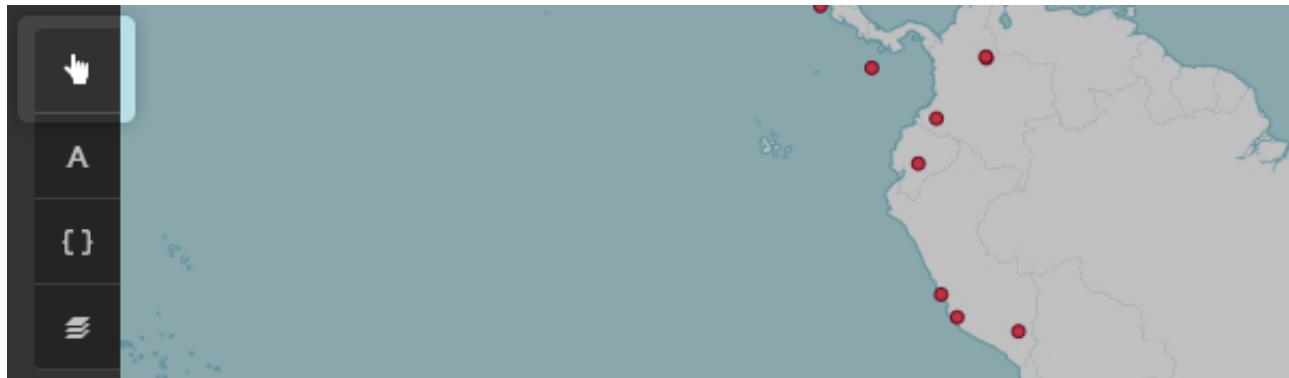
图例是地图不可或缺的重要组成部分。CartoCSS气泡悬浮框和图例可以为地图增加交互效果、附加信息和上下文信息。我们在这一节中就来讨论一下如何为地图添加气泡悬浮框和图例。

#### 气泡悬浮框

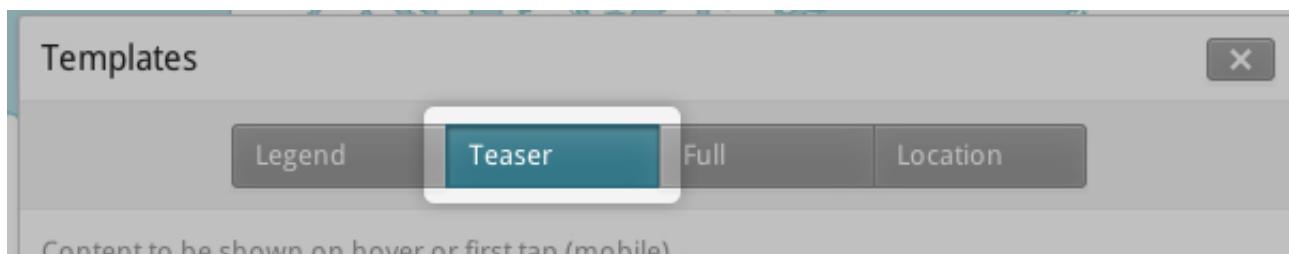
气泡悬浮框是当用户的鼠标悬浮或点击地图上的某个要素时展示出来的动态内容。由于在其中可以包含HTML，所以它在展示其它属性数据、图片等更多详细信息时非常有用。

这里我们以一幅地震分布地图为例说明如何实现在鼠标悬浮于每个地震点时展示出震级和地震发生时间。

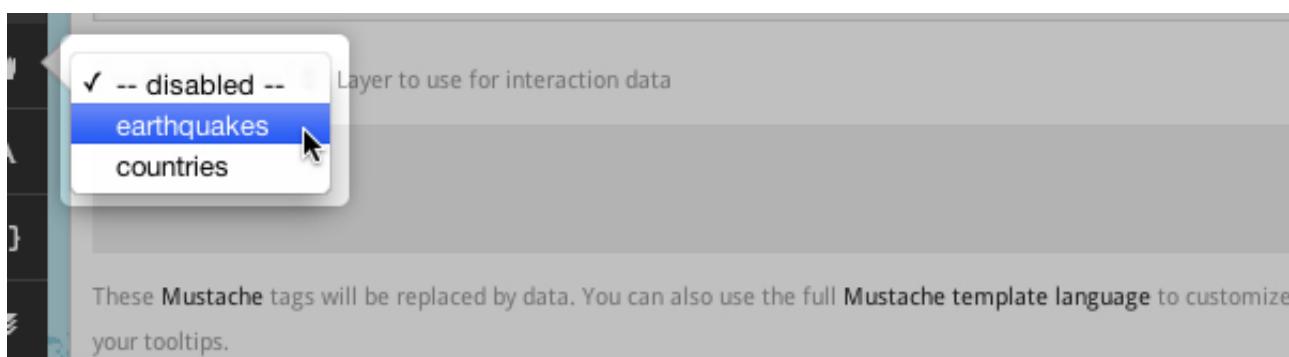
1. 打开Templates面板



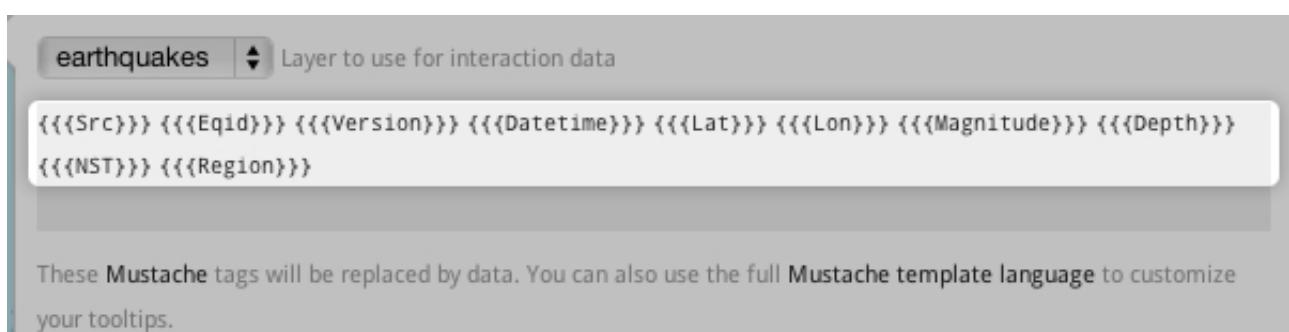
2. 选择“Teaser”标签页。鼠标悬浮在目标对象上时展示的是小样（译注：Teaser原意是正式的电影或广告播放之前，放出的预览内容，但相对于正式的“预告片”，也就是trailer，又没有那么内容丰富，所以这里译作“小样”），而点击目标对象时则展示出完整信息。可以通过为“Location”字段赋予一个URL值来定义要素对象在单击时需要加载的页面。



3. 选择“Earthquakes”图层用于本次交互。注意一次只能选择一个用于交互的图层。



4. 图层中需要显示的数据字段都要放在Mustache括号标签中。这些标签在地图交互时会被替换为对应的值。把你想要显示的字段填入花括号标签。



5. 使用Mustache标签来构造你的悬浮框模板。可以把下面的代码拷贝到小样文本框中，然后用预览功能看看效果。

```
{{{Magnitude}}} Magnitude Earthquake<br/>{{{DateTime}}} ![]  
(/tilemill/assets/pages/tooltips-4.png)
```

6. 点击“Save”保存并刷新地图。点击(X)按钮关闭Templates面板（或者可以直接按ESC键）。在地图上试试用鼠标悬浮在某个地震点上看看气泡悬浮框的展示效果。



## 图例

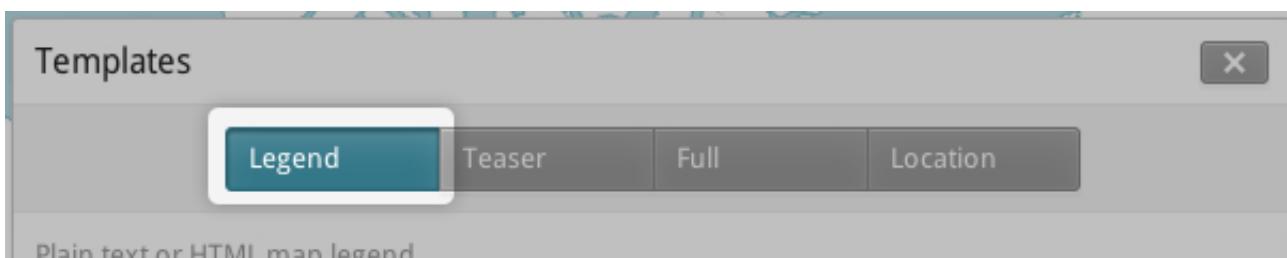
图例是始终展示在地图上，用于对地图的标题、描述、符号的含义等关键信息进行说明的制图要素。图例可以是一段HTML，也可以是一张简单的图片。

现在就让我们来为地图增加一个描述其专题含义的图例。

1. 打开“Templates”面板。



2. 进入面板后默认打开的就是图例标签页。



3. 在图例文本框中输入以下text/html文本：

```
<strong>Magnitude 2.5+ Earthquakes (Past 7 Days)</strong><br/>Circle size  
indicates magnitude of earthquake. 
```

4. 点击“Save”保存并关闭面板。这时图例应该就已经出现在地图的右下角了。

## HTML的合法性问题

安全起见，HTML格式的工具条和图例中的不安全代码和所有JavaScript代码都会被移除。如果你真的想利用JavaScript构建一些具有高级交互能力的地图，那么可以试试[MapBox.js API](#)。

## 高级图例

When designing a legend for TileMill that requires more than plain text, there are a few paths you can take. An image, html/css, or a combination. Both have their advantages and disadvantages.

之前我们介绍了图例与气泡悬浮框的基本用法。那能否实现一些不只是简单文本的高级图例呢？可以，而且有好几种方法可以实现。高级图例可以是图片、html/css或者是这些要素的组合，这些方法各有利弊。

### 嵌入图片

For complex graphics and those that feel more comfortable designing in a graphics editor. This involves creating a PNG or JPG and either serving it on the web and linking to it, or [base64-encoding it directly into the legend](#).

对于比较复杂的图形，最好是在专业的图形图像编辑中进行设计（译注：这句话的原文是没有谓语的，不是个完整的句子，所以这里是推测出来的意思）。图形图像可以是PNG或者JPG格式，既可以保存在Web服务器上也可以通过外链的方式引用，或者可以直接[在图例中使用base64编码方式嵌入](#)。

使用图片的好处在于你对它的设计可以控制到像素级，其复杂程度完全尽在设计者掌握。而图片的不足之处在于：它是静态的，一旦画到地图上就难以修改更新，要修改的话必须要找到图片的原始文件和专门的软件工具才能对其编辑修改（译注：例如Photoshop对应的ps文件，Illustrator对应的ai文件等）。

### HTML/CSS

相比起来，利用代码把图例设计成表格形式就简单许多。这其中包括设计图例中要素的布局和样式，过程和设计网页很类似。

使用html/css的好处是你可以直接在支持CartoCSS的软件（比如TileMill）中编辑图例代码，然后就能立即看到效果。即使是在将地图导出到其它外部格式（比如MBTiles）之后，也可以对图例进行维护。但是，图例的样式设计会受到很多局限（比如使用直角和纯色），并且常常是为了实现一个简单的设计效果而不得不写很多代码。

使用html/css写图例的另一个巨大优势是可以把图例代码在设计师之间和项目之间拷贝复用。下面有一些简单的图例模板帮你入门。但请注意这并不是html和css的入门指南。如果需要深入学习html和css，请参考[tizag.com](#)，那里有更多更好的学习材料。

下面的代码可以直接拷贝到CartoCSS制图软件（比如TileMill）中。然后根据你自己的需要和本指南进行适当的修改。

```


title



colors



- <li><span style='background-color: #F1EEF6;'></span>0 - 20%
- <li><span style='background-color: #BDC9E1;'></span>40%
- <li><span style='background-color: #74A9CF;'></span>60%
- <li><span style='background-color: #2B8CBE;'></span>80%
- <li><span style='background-color: #045A8D;'></span>100%



labels



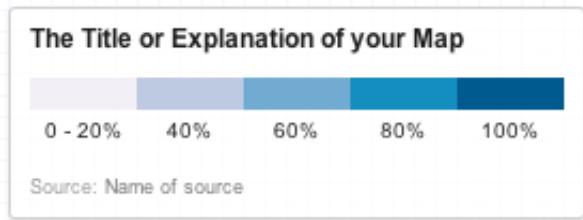
- 0 - 20%
- 40%
- 60%
- 80%
- 100%



Source: <a href="#link to source">Name of source</a>


```

## 1. 水平排布的图例



```


The Title or Explanation of your Map



colors



- <li><span style='background-color: #F1EEF6;'></span>0 - 20%
- <li><span style='background-color: #BDC9E1;'></span>40%
- <li><span style='background-color: #74A9CF;'></span>60%
- <li><span style='background-color: #2B8CBE;'></span>80%
- <li><span style='background-color: #045A8D;'></span>100%



labels



- 0 - 20%
- 40%
- 60%
- 80%
- 100%



Source: <a href="#link to source">Name of source</a>


```

```

.my-legend .legend-scale ul {
  margin: 0;
  padding: 0;
  float: left;
  list-style: none;
}

.my-legend .legend-scale ul li {
  display: block;
  float: left;
  width: 50px;
  margin-bottom: 6px;
  text-align: center;
  font-size: 80%;
  list-style: none;
}

.my-legend ul.legend-labels li span {
  display: block;
  float: left;
  height: 15px;
  width: 50px;
}

.my-legend .legend-source {
  font-size: 70%;
  color: #999;
  clear: both;
}

.my-legend a {
  color: #777;
}


```

</style>

## 2. 垂直排布的定性型图例



```

<div class='my-legend'>
  <div class='legend-title'>The Title or Explanation of your Map</div>

```

```
<div class='legend-scale'>
  <ul class='legend-labels'>
    <li><span style='background:#8DD3C7;'></span>One</li>
    <li><span style='background:#FFFFB3;'></span>Two</li>
    <li><span style='background:#BEBADA;'></span>Three</li>
    <li><span style='background:#FB8072;'></span>Four</li>
    <li><span style='background:#80B1D3;'></span>etc</li>
  </ul>
</div>
<div class='legend-source'>Source: <a href="#link to source">Name of source</a></div>
</div>

<style type='text/css'>
.my-legend .legend-title {
  text-align: left;
  margin-bottom: 5px;
  font-weight: bold;
  font-size: 90%;
}
.my-legend .legend-scale ul {
  margin: 0;
  margin-bottom: 5px;
  padding: 0;
  float: left;
  list-style: none;
}
.my-legend .legend-scale ul li {
  font-size: 80%;
  list-style: none;
  margin-left: 0;
  line-height: 18px;
  margin-bottom: 2px;
}
.my-legend ul.legend-labels li span {
  display: block;
  float: left;
  height: 16px;
  width: 30px;
  margin-right: 5px;
  margin-left: 0;
  border: 1px solid #999;
}
.my-legend .legend-source {
  font-size: 70%;
  color: #999;
  clear: both;
```

```
}

.my-legend a {
  color: #777;
}

</style>
```

## 图例类别

图例可以被包含在一个具有自定义类别的要素中，就像上面两个例子中的my-legend那样。这个类别具有多个默认样式，包括max-width是280像素、max-height是400像素等。通常情况下，这个尺寸对于图例来说已经足够大了。但如果在实际显示时出现了滚动条，那么就说明这个尺寸不够大了。假设你需要修改这些默认属性，那么需要按下面的方法做：

在<style></style>标记中为my-legend增加一个选择器并在其中声明你要重新定义的新值。对于那些需要被覆盖重载的属性，最好在后面加上!important标签。比如说把最大宽度增加到300像素：

```
.my-legend {
  max-width: 300px !important;
}
```

## 参考文献

1. Mapbox, [Adding tooltips and legends](#)
2. Mapbox, [Advanced legends](#)

## 4.5 理解元瓦片

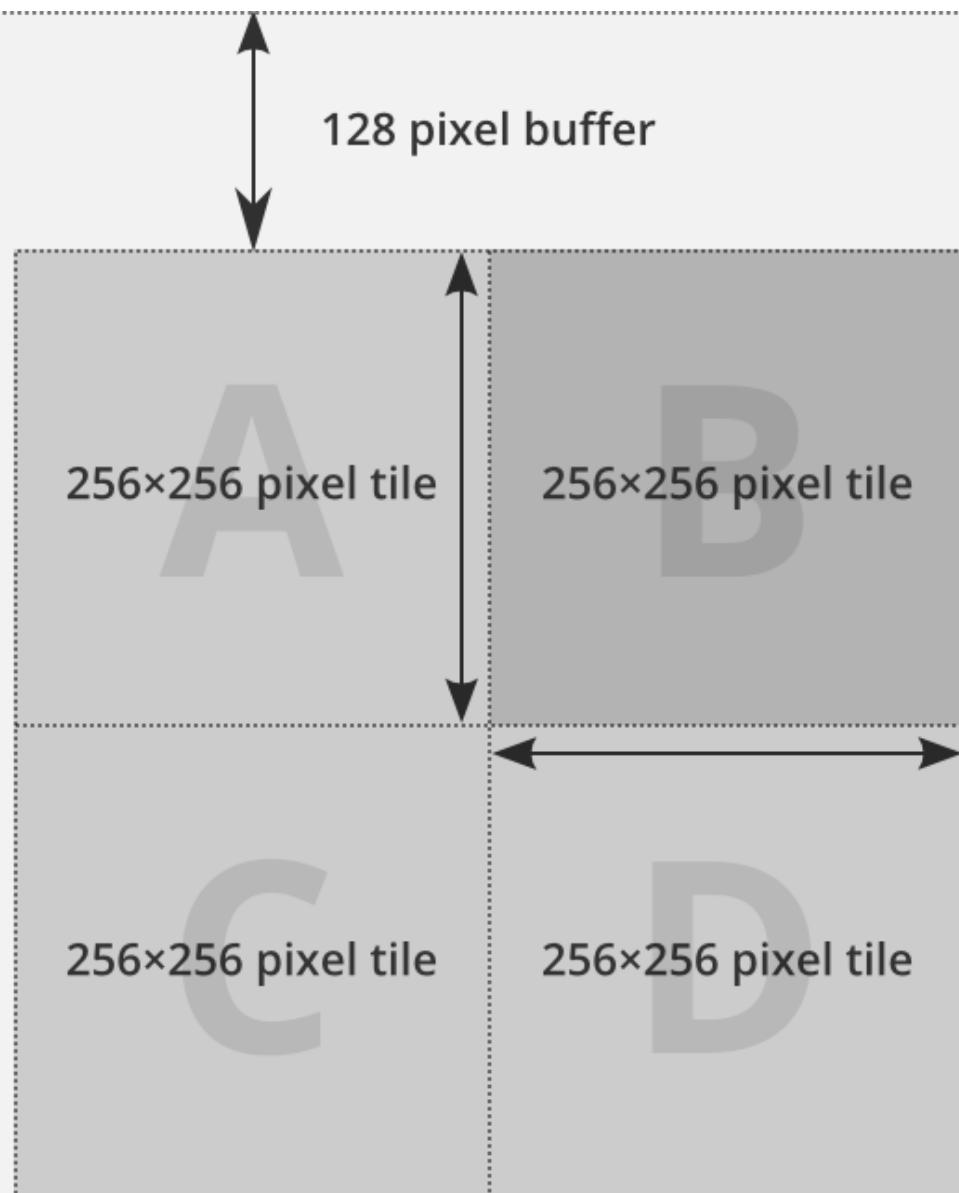
在现代Web制图应用中，一幅地图通常是由一系列具有相同尺寸的小块无缝拼合而成的。在专业术语中，这些小块被称为瓦片（tile）。尽管瓦片最终是以独立图片的形式出现，但支持CartoCSS的Web制图软件及其底层的渲染引擎会先以组的方式批量渲染瓦片，然后再对其切分得到最终的瓦片。这种处理方式能够从多个方面改善制图效率。

在使用支持CartoCSS的工具进行制图时，为了能够更高效的配置地图样式，并且合理规避一些制图渲染中的特殊问题，了解元瓦片（metatiles）的工作机理是很有必要的。元瓦片的配置在绘制地图样式中的标注、注记和图案等方面扮演着重要的角色。

## 元瓦片的结构

There are two main parts to a metatile: the tiles and the buffer. By default a metatile in TileMill consists of 4 tiles (arranged 2 wide and 2 high) and a buffer of 256 pixels around the tiles.

元瓦片主要包括两个组成部分：瓦片和缓冲区。一个元瓦片在默认情况下由4个（2行2列）个瓦片和一个环绕在这4个瓦片周围，256像素宽（译注：图中缓冲区条带的宽度不是128么？）的缓冲区组成。



*Total metatile size: 768x768 pixels*

元瓦片的缓冲区部分也会被制图渲染引擎（译注：也就是Mapnik）绘制，但这部分在最终的地图上不会被显示出来。在瓦片周围多渲染一个缓冲区的目的是为了保证标注、注记等地图要素在跨越元瓦片边界的时候仍然能够被正确绘制。如果没有这个缓冲区，那么就会在每个瓦片的边缘处出现大量被切割的标注、图标等。

## 调整元瓦片配置

元瓦片中包含的瓦片数量和缓冲区大小都是可以配置的。以TileMill为例，调整元瓦片中瓦片数量的方法是打开项目设置，然后拖动元瓦片大小滑块到合适的位置。滑块数值的平方代表了元瓦片中包含瓦片的个数。而缓冲区大小的调整则需要在CartoCSS中完成。在Map对象中增加一个`buffer-size`属性，它的值（必须是整数）就是缓冲区的像素宽度：

```
Map {
```

```
background-color: white;  
buffer-size: 256;  
}
```

## 合理设置缓冲区大小

如果你发现在地图上出现了标注和图标被切割的情况，那么就应该考虑调整增大缓冲区了。但是应该把缓冲区宽度设成多少呢？建议找到地图上最宽的标注，然后就先从它的宽度开始尝试。

## 合理设置元瓦片尺寸

对于大多数情况，使用默认的元瓦片尺寸（也就是2）就可以了。这意味着渲染引擎会将地图内容先渲染到长宽均为512像素的“大”瓦片上，然后再将其切分成4个长宽为256像素的正常瓦片并返回给地图显示前端。当一个或者更多的相邻瓦片请求正好命中同一个元瓦片时，渲染引擎会在切片之前暂停很短的时间以处理元瓦片，然后再将每个独立的256像素瓦片返回给地图显示前端。

但在一些特殊的情况下需要调整元瓦片的尺寸：有一种可能的原因让你想要将元瓦片的尺寸减小到1；而有两种可能的原因让你想把这个值增大到8或16。下面分别介绍这两种情况。

### 减小元瓦片尺寸

要把元瓦片尺寸从默认值2调小，那么只可能是调成1，也就是关闭元瓦片功能。这样做可以让地图在编辑制图样式过程中的响应轻微加快，因为这时每个瓦片都是独立渲染的了。如果当前地图视图范围内的某些瓦片包含的数据较多，而另一些瓦片包含的数据较少，那么关闭元瓦片功能之后会使包含数据较少的瓦片比相邻的包含数据较多的瓦片更快绘制出来。

### 增大元瓦片尺寸

#### 原因1：减少导出时间

While disabling metatiling can give a more responsive feel to the map UI, the opposite is true when exporting to MBTiles. Increasing the metatile size can significantly increase overall performance and decrease the overall time it takes to render an entire export job. This is because rendering many tiles in sequence using larger metatiles means doing less overall work.

如果说关闭元瓦片功能可以让制图过程感觉响应速度更快，那么反过来（增大元瓦片尺寸）则可以让导出地图到MBTiles的过程变得更快。增大元瓦片的尺寸可以显著提高地图整体的渲染效率，缩短在执行导出任务时的渲染时间。其原因是在串行渲染大量瓦片时，如果使用尺寸更大的元瓦片，那么就意味着需要更少的整体工作量。（译注：这里的潜台词是：渲染4个小瓦片的时间要大于渲染1个大瓦片的时间。但事实是这样吗？原因何在？极端情况下，把整张地图全画在一个超级大瓦片上会是最快的吗？）

然而到底使用多大的元瓦片才能获得最佳的导出性能呢？这的确是没有个硬性准则的。这和地图中包含多少数据量、有没有建好空间索引、执行导出任务的机器内存有多大等许多因素都有关。

我们的建议是先取地图的一小部分（只选几个缩放级别，或者框一小块区域）做做测试，看看把元瓦片的尺寸分别设成4、8或16时导出性能会有什么变化。然后根据实验结果选取最佳的元瓦片大小。

#### 原因2：减少瓦片边缘的切割问题

更大的元瓦片意味着标注、注记等要素被绘制在瓦片边缘的概率会降低。另外，对于一些标注算法（比如通过设置text-min-distance属性控制重复绘制标注的间距），更大的元瓦片可以让它有更大的工作空间，从而得到更好的标注绘制效果。

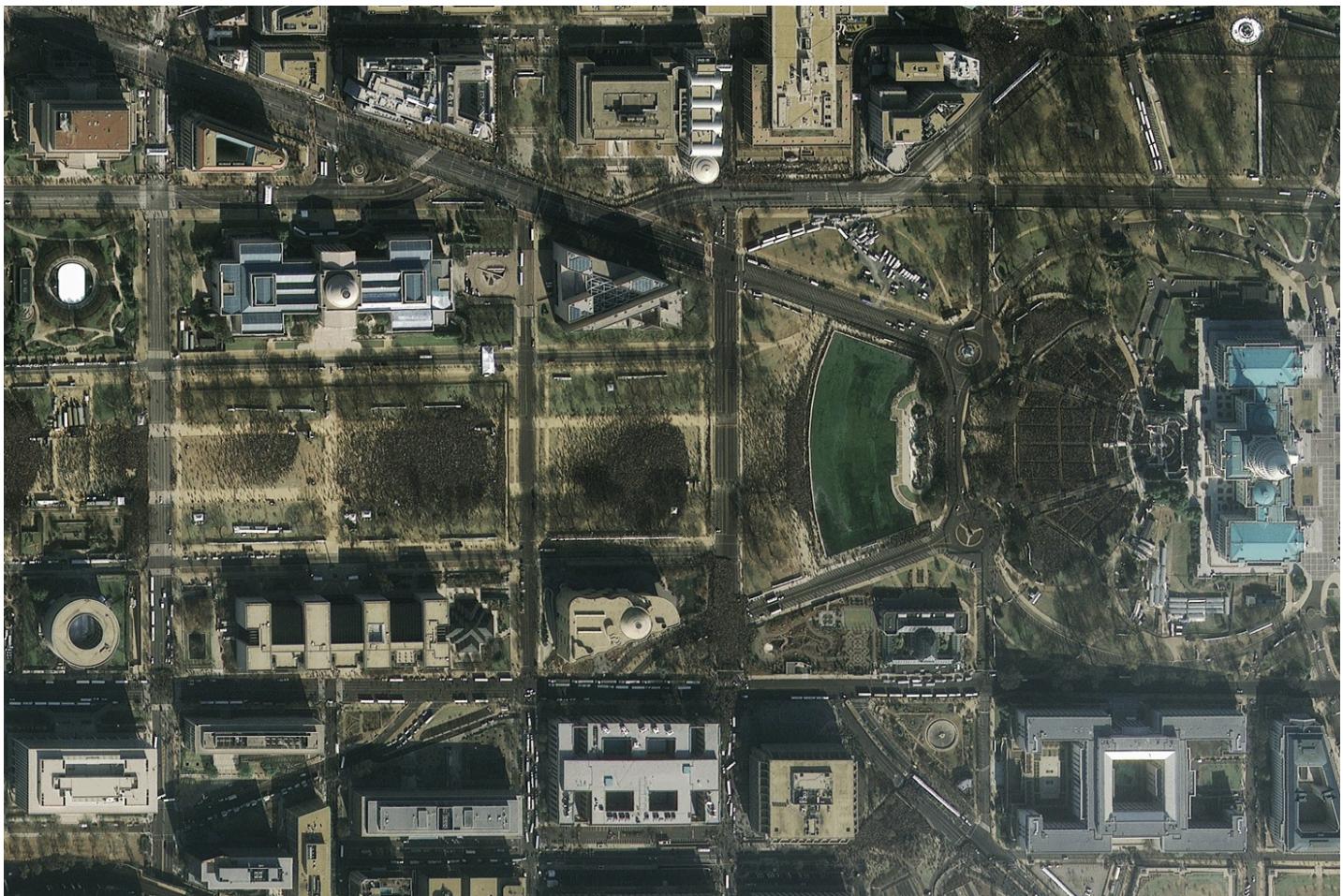
## 参考文献

1. Mapbox, [Understanding Metatiles](#)

## 4.6 棚格制图技巧

### 配准卫星影像

奥观海总统首次就职典礼当天，华盛顿的天空万里无云。而GeoEye卫星恰好在那时飞过白宫上空，它从太空中捕捉到了那个精彩的瞬间：



GeoEye | 2009 Inauguration

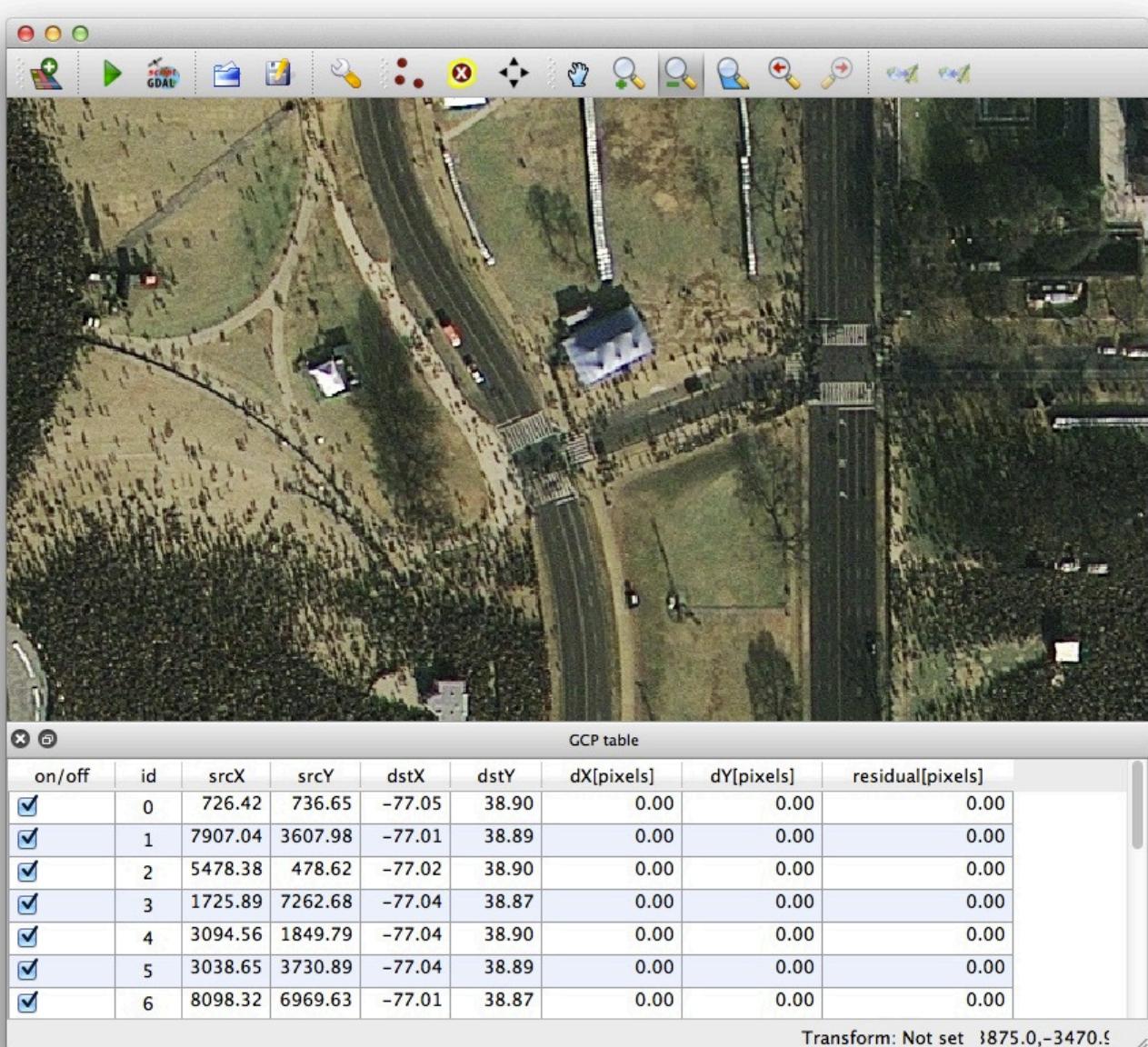
利用[GDAL](#), [QGIS](#), 还有[MapBox Satellite](#)等工具，我们可以对这幅2009年的卫星影像手工配准，然后通过与[MapBox Satellite layer](#)的对比来观察当时的就职典礼出席来宾，发现城市的变化。

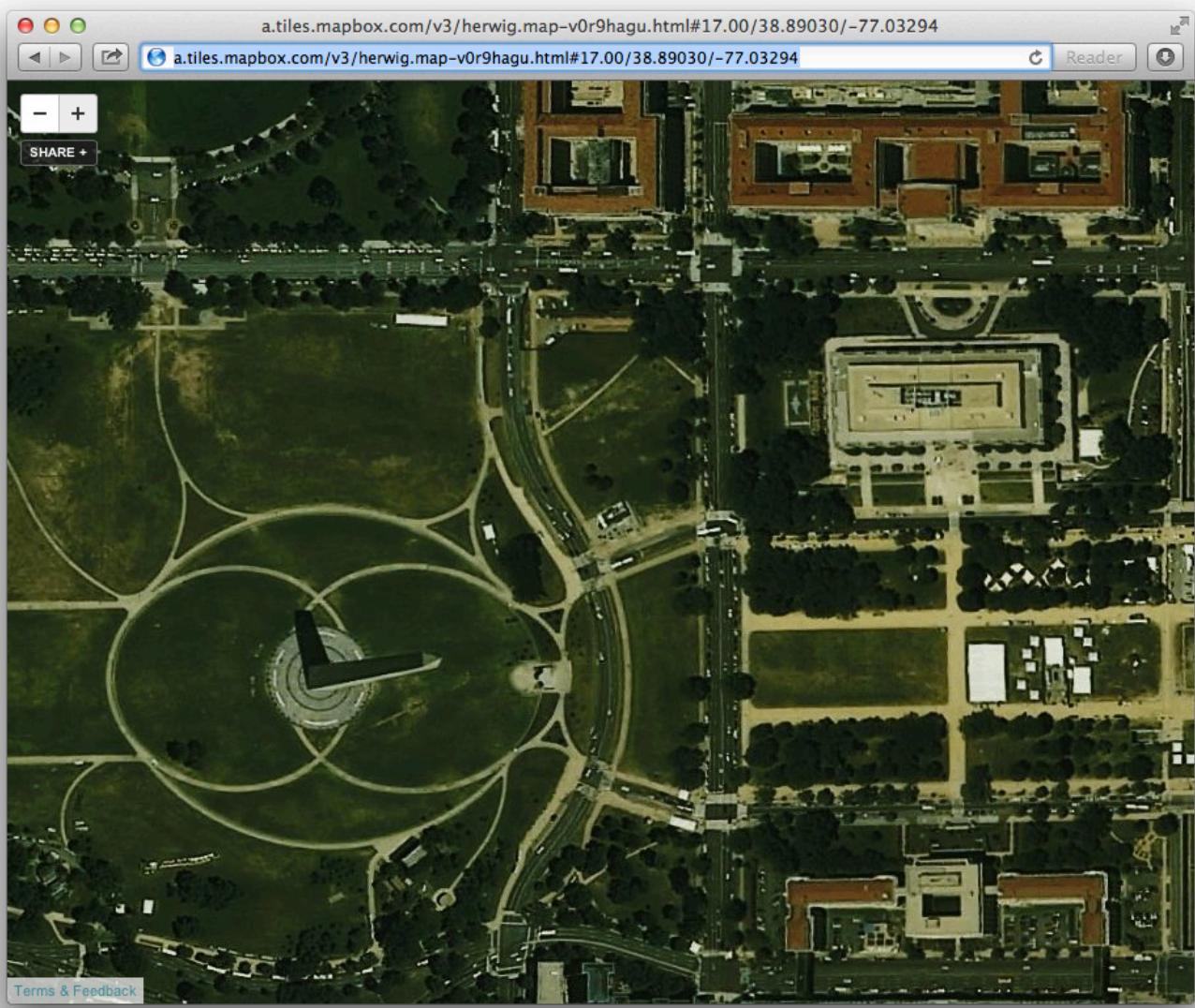
下面我们就来介绍如何将一幅缺少空间参考信息的普通JPEG图片转成一个可以在制图工具中渲染成图的地理空间数据集。整个过程中需要用到三个免费、开源的软件工具：[Quantum GIS](#)（这里使用的QGIS版本为1.8），[GDAL](#)，还有[TileMill](#)。

### 手工配准

1. 打开QGIS，从Plugins下拉菜单中激活Georeferencer插件。
2. 在Georeferencer窗口中打开栅格数据集。
3. 登录MapBox账户并创建一个新的图层。注意，如果要使用MapBox的卫星影像图层，你至少需要一个[基本账户](#)。

- 在QGIS的Georeferencer窗口中，从源图片中找一个明显的标志性位置。用**Add a Point**工具（Mac下的快捷键为Cmd+A）在刚才的位置上放一个点。这里我们选取的位置是15th Street NW和Madison Dr. NW的交叉口，在华盛顿纪念碑对面。
- 在你的MapBox卫星影像图层上找到和上一步中完全相同的位置，并且拖动地图使该点位于屏幕中心，如下图所示。





### MapBox Satellite

注意URL中最后的部分：

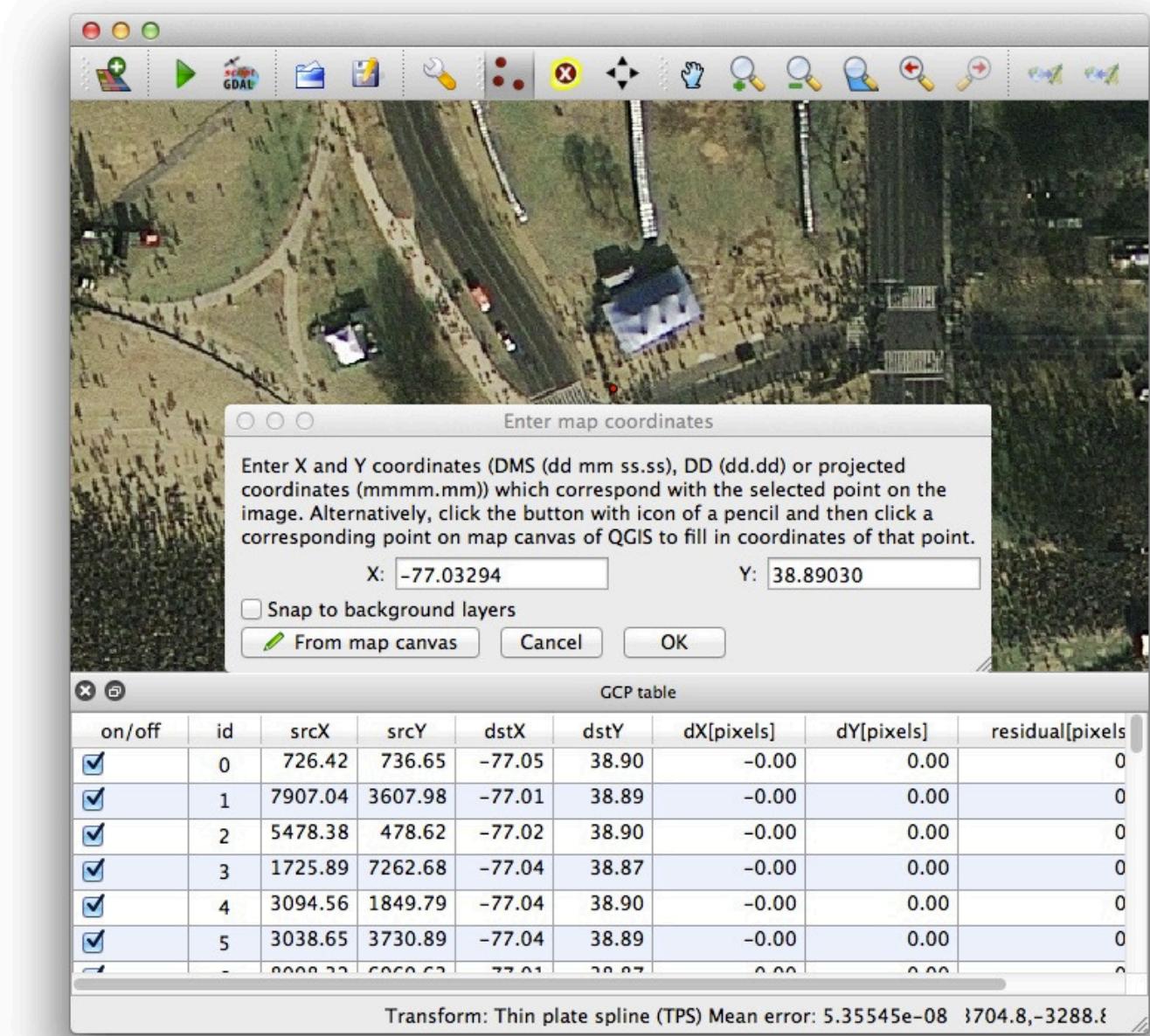
```
#17.00/38.89030/-77.03294
```

在#之后的第一个数字是缩放级别，第二个是纬度，第三个是经度。

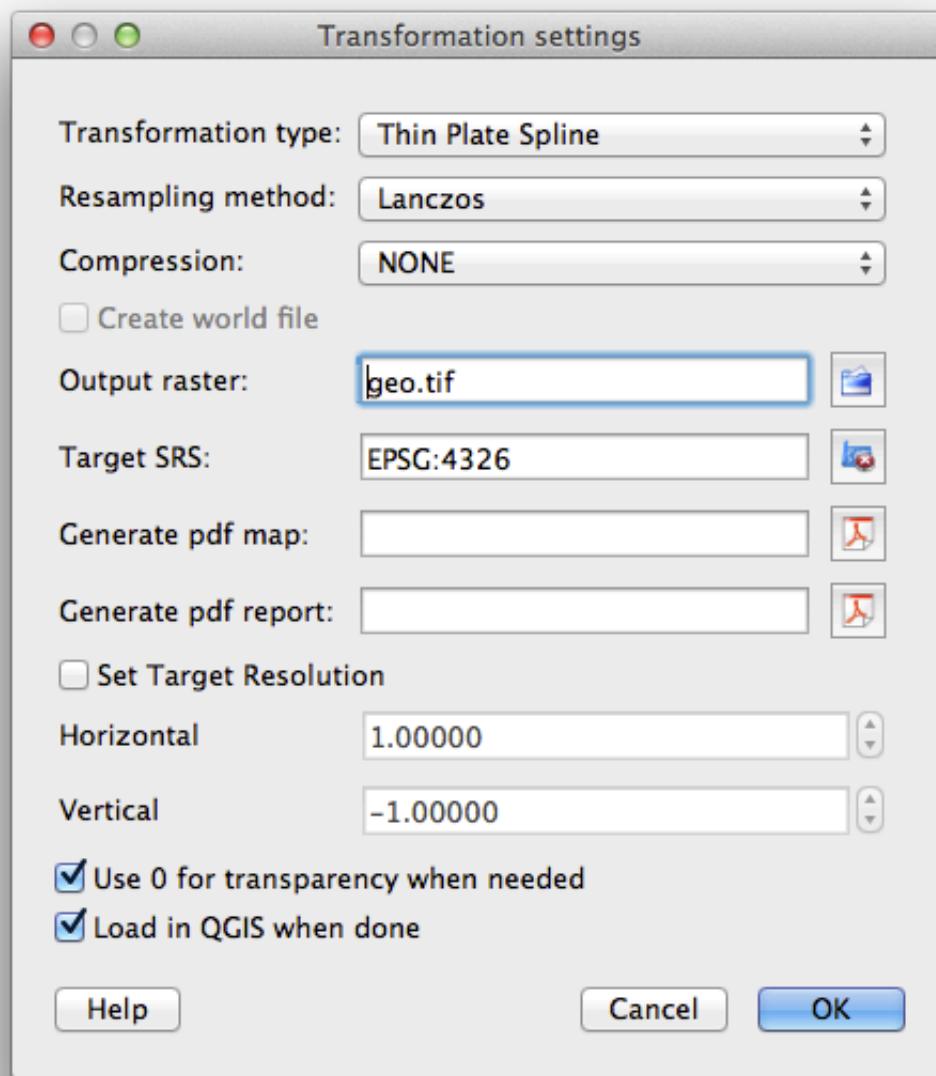
纬度: 38.89030

经度: -77.03294

1. 再回到QGIS中，在“Enter Map Coordinates”对话框中输入从URL中获得的经纬度座标。注意纬度对应的是Y值，经度对应的是X值。



1. 重复进行第6到第8步的操作，直到已经在原始图片上增加了足够多的控制点。推荐的做法是按照从图片的四个角开始，逐渐向中心推进的方式选取控制点。为了达到项目的精度需求，我们一共选取了37个控制点。然后请务必用Georeferencer插件中的**“Save GCP Points as”**功能保存这些控制点，这样你才可以在以后重新打开这些控制点，修改它们或添加新的控制点来进一步提高空间精确度。
2. 配准操作既可以直接在QGIS中做，也可以通过选择**“Generate GDAL Script”**来生成GDAL脚本。我们这里选择了Thin Plate Spline变换、Lanczos重采样、无压缩选项之后，生成了一段可以进一步修改的脚本。



下面就是结合了QGIS生成的地面控制点、我们项目中的地图投影、重采样和缩略图等特定配置的最终处理脚本。

```
#!/bin/bash

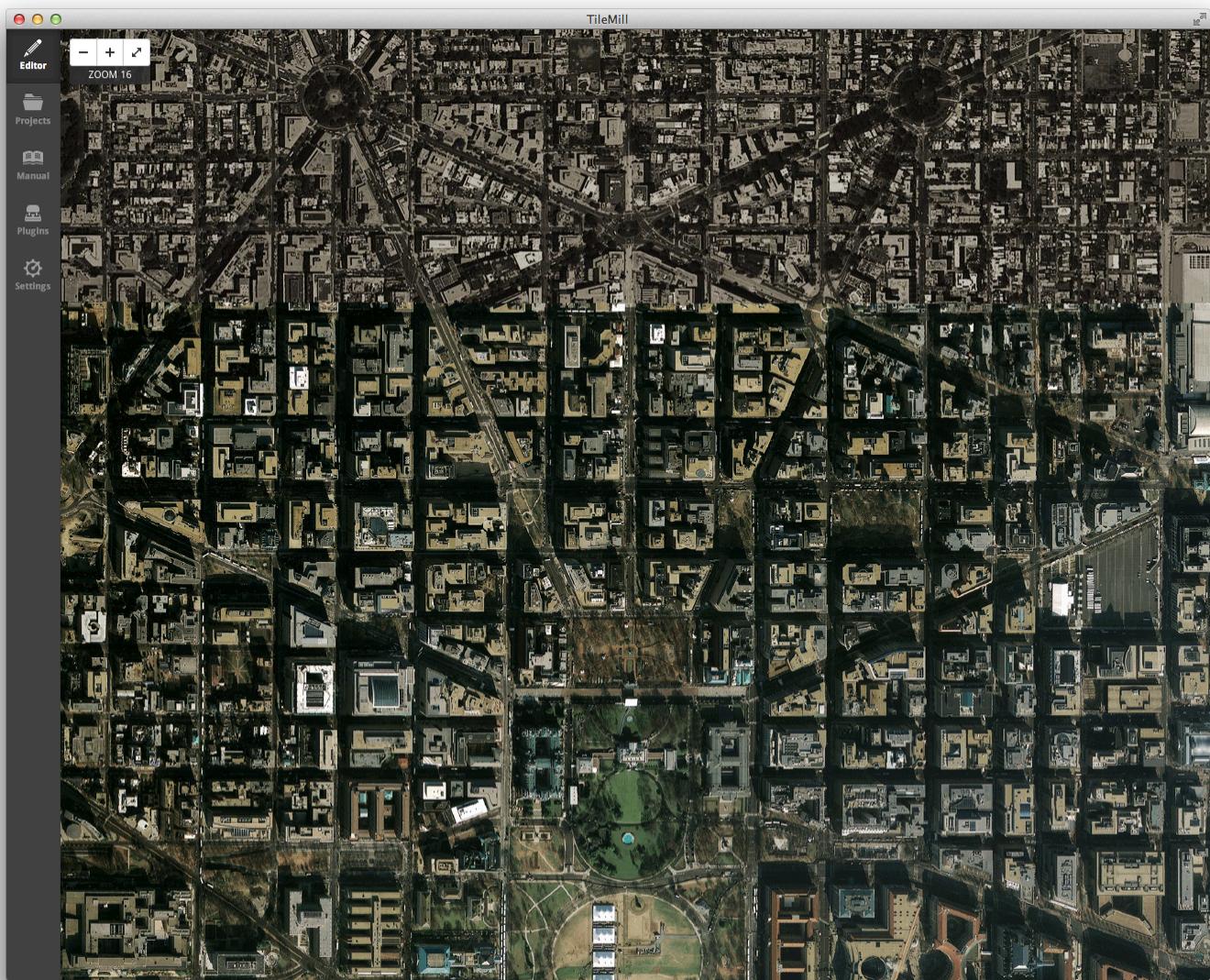
ADDO="2 4 8 16 32 64 128 256 512 1024 2048 4096 8192"

gdal_translate \
-of GTiff \
-a_nodata "0 0 0" \
-a_srs EPSG:4326 \
-gcp 726.415 736.655 -77.0501 38.9025 \
-gcp 7907.04 3607.98 -77.0091 38.8898 \
-gcp 5478.38 478.625 -77.0231 38.9037 \
```

```
-gcp 1725.89 7262.68 -77.0443 38.8731 \
-gcp 3094.56 1849.79 -77.0365 38.8975 \
-gcp 3038.65 3730.89 -77.0367 38.889 \
-gcp 8098.32 6969.63 -77.0076 38.8744 \
-gcp 6988.43 324.384 -77.0141 38.9044 \
-gcp 8066.53 1871.22 -77.0079 38.8974 \
-gcp 735.208 3692.09 -77.0501 38.8893 \
-gcp 166.054 6045.52 -77.0533 38.8786 \
-gcp 7344.64 7467.87 -77.012 38.8722 \
-gcp 4911.28 5353.86 -77.026 38.8817 \
-gcp 4621.91 6858.44 -77.0276 38.8749 \
-gcp 2528.87 5761.23 -77.0396 38.8798 \
-gcp 5429.01 3224.15 -77.0229 38.8913 \
-gcp 5433.9 3222.92 -77.0229 38.8913 \
-gcp 3698.56 3448.02 -77.0329 38.8903 \
-gcp 3623.56 3631.47 -77.0334 38.8894 \
-gcp 3009.31 3514.59 -77.0369 38.89 \
-gcp 3283.65 3610.7 -77.0353 38.8896 \
-gcp 2927.7 4022.01 -77.0373 38.8877 \
-gcp 3892.68 3804.94 -77.0318 38.8887 \
-gcp 3058.53 5374.32 -77.0366 38.8816 \
-gcp 4093.72 5910.69 -77.0306 38.8792 \
-gcp 7320.98 3316.66 -77.012 38.8909 \
-gcp 7738.56 3826.98 -77.0097 38.8887 \
-gcp 7739.92 3295.92 -77.0097 38.891 \
-gcp 7755.51 3482.34 -77.0097 38.8902 \
-gcp 7296.21 3723.98 -77.0122 38.889 \
-gcp 6804.33 3235.89 -77.015 38.8912 \
-gcp 6801.08 3869.32 -77.015 38.8884 \
-gcp 319.63 7391.94 -77.0524 38.8725 \
-gcp 300.338 3897.82 -77.0525 38.8883 \
-gcp 265.529 3509.08 -77.0527 38.89 \
-gcp 1038.55 3611.29 -77.0482 38.8895 \
-gcp 1039.54 3708.54 -77.0482 38.8891 \
Inauguration.jpg \
Inauguration_4326.tif
gdalwarp \
    -r lanczos \
    -rcs \
    -t_srs EPSG:3857 \
    -wm 1000 \
    -srcnodata "0 0 0" \
    -dstnodata "0 0 0" \
    -dstalpha \
    -co COMPRESS=LZW \
    -co TILED=YES \
```

```
Inauguration_4326.tif \
Inauguration_3857.tif
gdaladdo \
-r gauss \
--config COMPRESS_OVERVIEW LZW \
Inauguration_3857.tif \
$ADDO
rm Inauguration_4326.tif
```

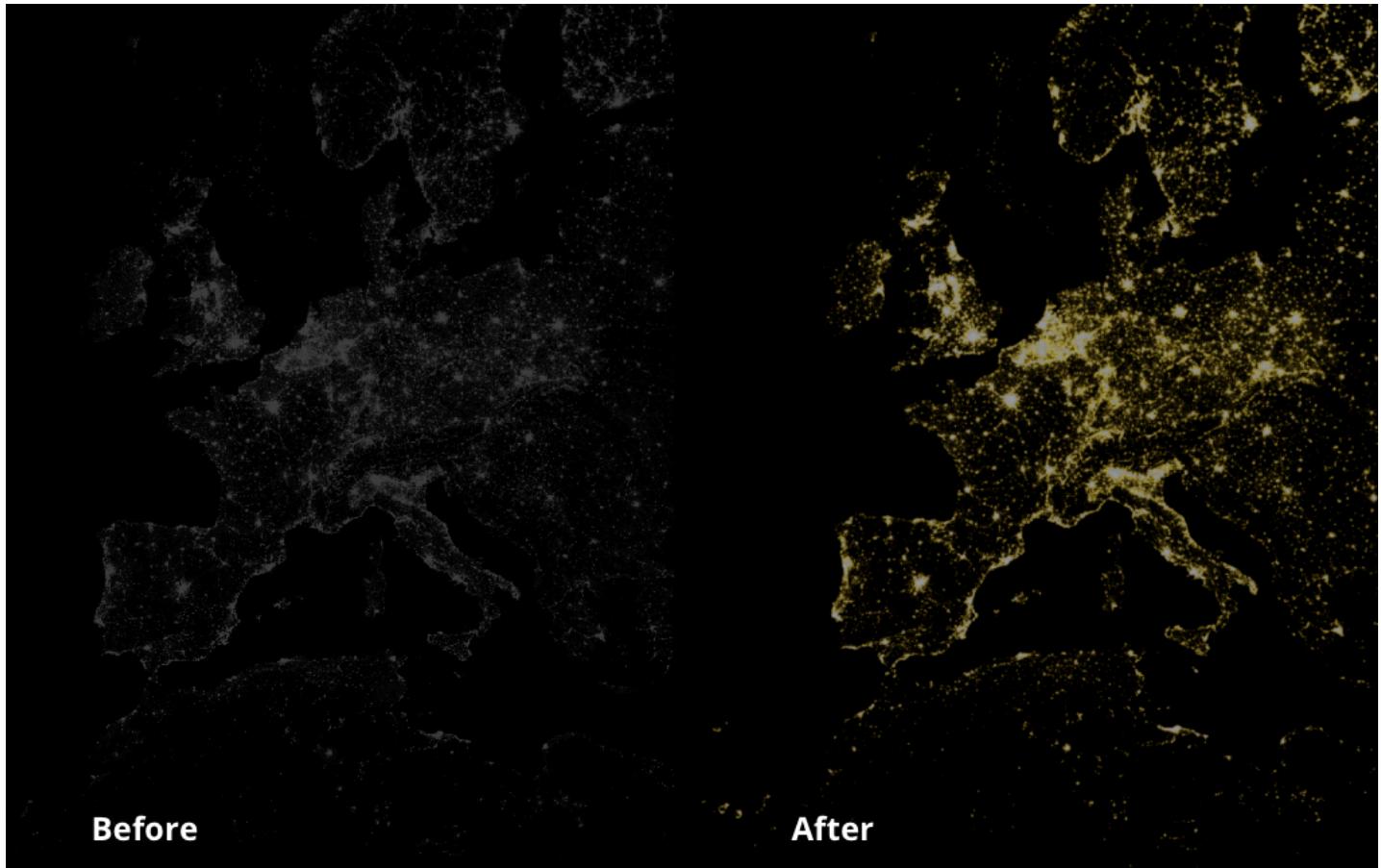
执行上面这段脚本会生成一个可以在TileMill中渲染的标准GeoTIFF。



我们可以在TileMill中利用Reference Layer插件将这幅配准后图像与[MapBox Satellite layer](#)对比来检查其空间精度。

## 为单波段栅格数据着色

传统上，单波段的栅格数据在TileMill中会被渲染成灰度的（译注：或者通俗的说，就是黑白的），但它也完全可以不这样渲染。



MapBox的博文[processing DNB raster data from NASA and NOAA's Suomi NPP spacecraft](#)介绍了如何制作一幅夜间灯光分布图，让我们领略到从太空中看地球上夜晚灯光分布的美妙效果。而现在在TileMill中，我们只需一半的代码，用更短的时间就可以制出同样效果的地图。这要归功于强大的raster-colorizer。有了它，我们就不用再费劲的用命令行工具加虚拟栅格（VRT）的方法了。

要充分利用强大的raster-colorizer，请先确认在图层的高级设置中加入了band=1。（译注：其实我觉得这很不自然，在使用HiGIS的制图前端过程中，大家就经常会忘记加这个东西。这更像是个神秘的trick，应该在今后的设计中修正）

## 暗夜的灯光

尽管原始数据只有一个波段，但我们可以利用CartoCSS的从属样式能力把它像一幅三波段数据那样去渲染，得到同一个图层的三个版本，相互叠加渲染：

```
#2010::1 #2010::2 #2010::3
```

下一步，利用raster-colorizer对每个从属样式分别定义不同的渲染色带，从而最终得到合成的RGB效果：

**#2010::1**



**80**

**100+**

**#2010::2**



**50**

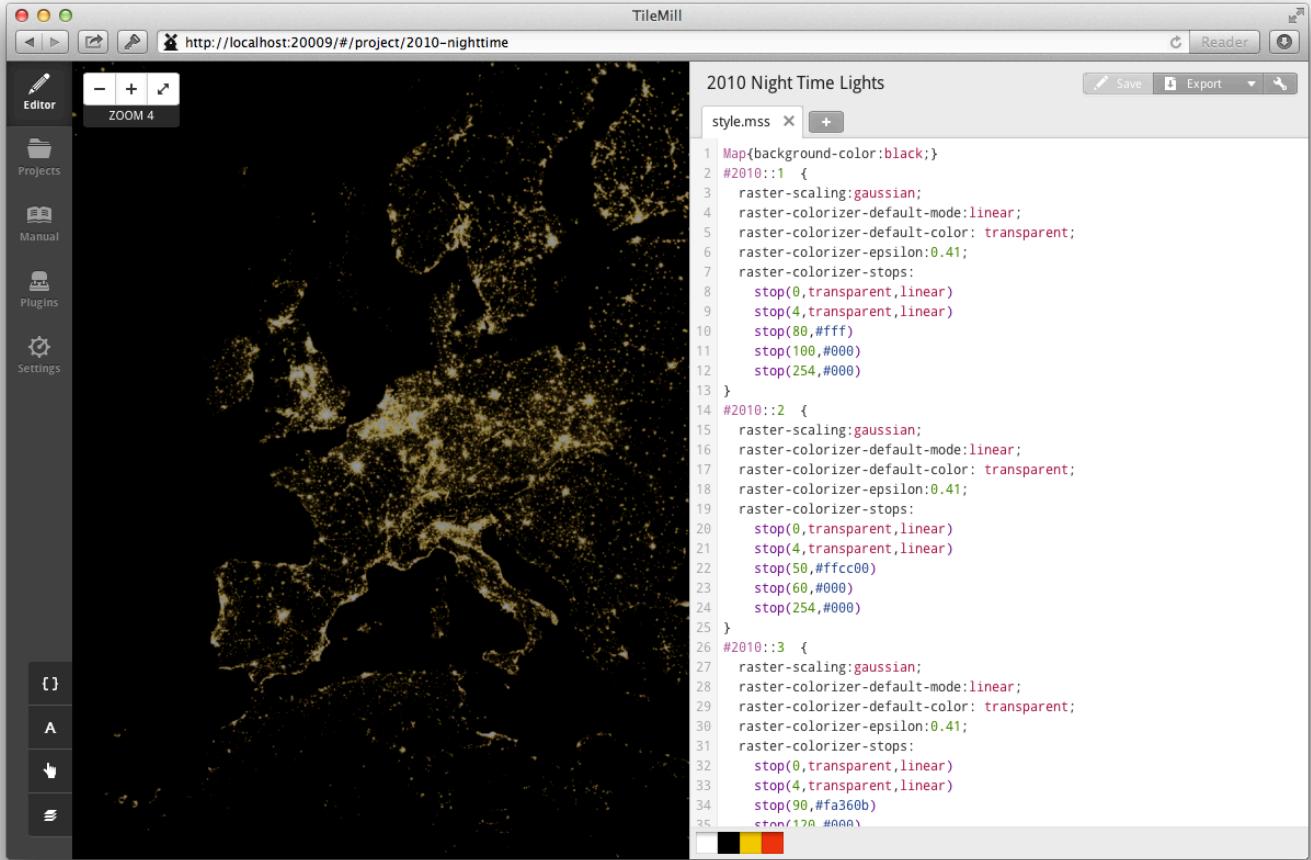
**60+**

**#2010::3**



**90**

**120+**



## 2010 NightTime Lights

```
#2010::1  {
  raster-scaling:gaussian;
  raster-colorizer-default-mode:linear;
  raster-colorizer-default-color: transparent;
  raster-colorizer-epsilon:0.41;
  raster-colorizer-stops:
    stop(0,transparent,linear)
    stop(80,#fff)
    stop(100,#000)
}

#2010::2  {
  raster-scaling:gaussian;
  raster-colorizer-default-mode:linear;
  raster-colorizer-default-color: transparent;
  raster-colorizer-epsilon:0.41;
  raster-colorizer-stops:
    stop(0,transparent,linear)
    stop(50,#ffcc00)
    stop(60,#000)
}

#2010::3  {
  raster-scaling:gaussian;
```

```

raster-colorizer-default-mode:linear;
raster-colorizer-default-color: transparent;
raster-colorizer-epsilon:0.41;
raster-colorizer-stops:
  stop(0,transparent,linear)
  stop(90,#fa360b)
  stop(120,#000)
}

```

最终效果图



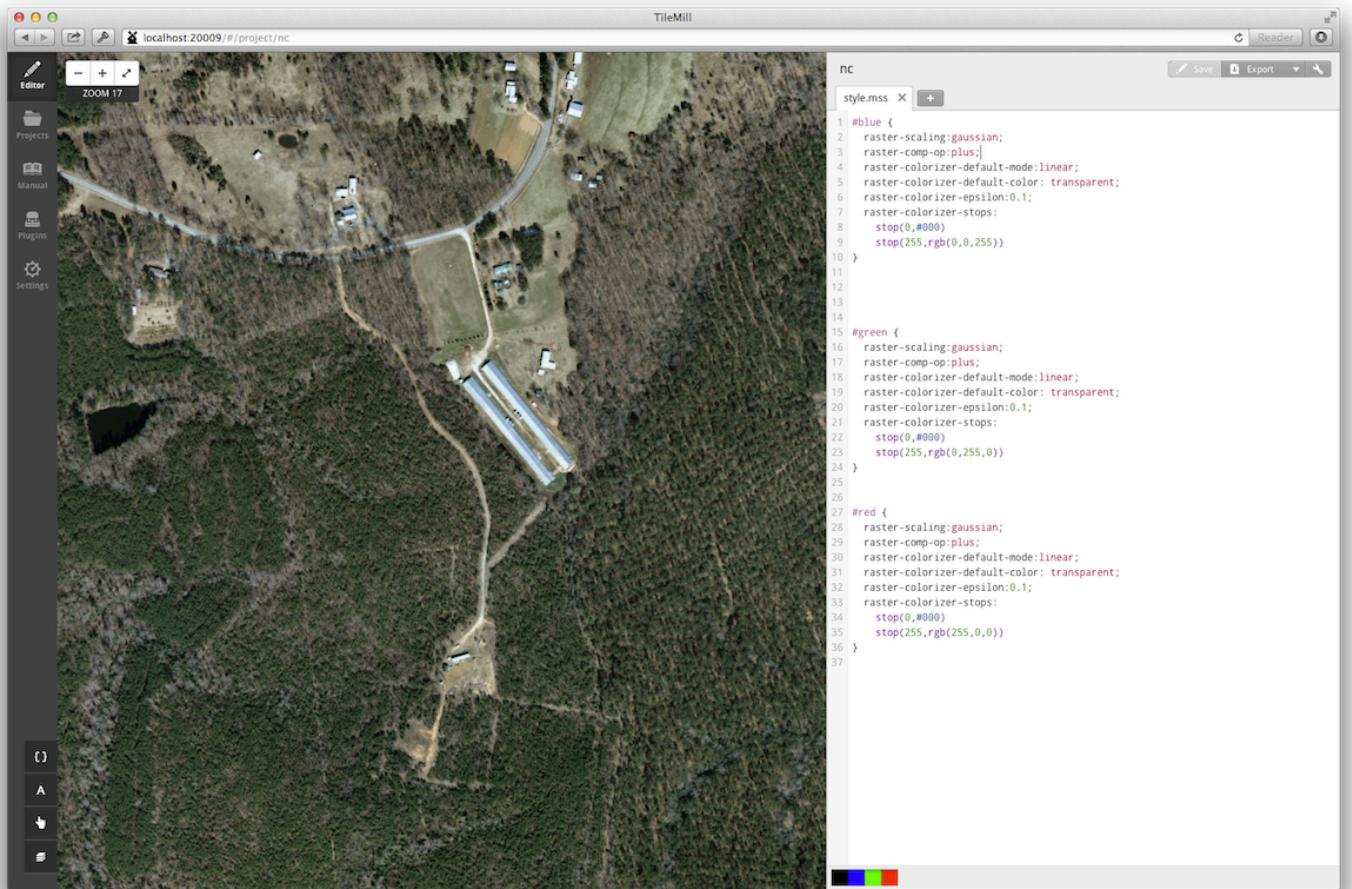
## 全色影像的色彩校正

与前面介绍的单波段栅格数据着色过程类似，我们可以对三波段可见光航空或卫星影像进行色彩校正。利用CartoCSS和相关工具（例如TileMill）可以使这项工作大为简化，而且可定制性更强。

### 全色影像

正常情况下，一幅全色影像在加载到CartoCSS制图工具中之后会以可见光自然色显示。但为了能充分利用raster-colorizer，我们需要对这同一幅全色影像加载三次，而且三次对应的红、绿、蓝图层要分别在高级设置中加上band=1、band=2和band=3选项。这里band=的含义是告诉制图工具只加载指定波段的数据。

为了能让这三个图层叠加后仍能按照全色影像正常显示，还需要为每个图层定义raster-comp-op: plus；和raster-colorizer-default-mode: linear；属性。



## 色彩校正前

```
#red {
    raster-scaling:gaussian;
    raster-comp-op:plus;
    raster-colorizer-default-mode:linear;
    raster-colorizer-default-color: transparent;
    raster-colorizer-epsilon:0.1;
    raster-colorizer-stops:
        stop(0,#000)
        stop(255,rgb(255,0,0))
}

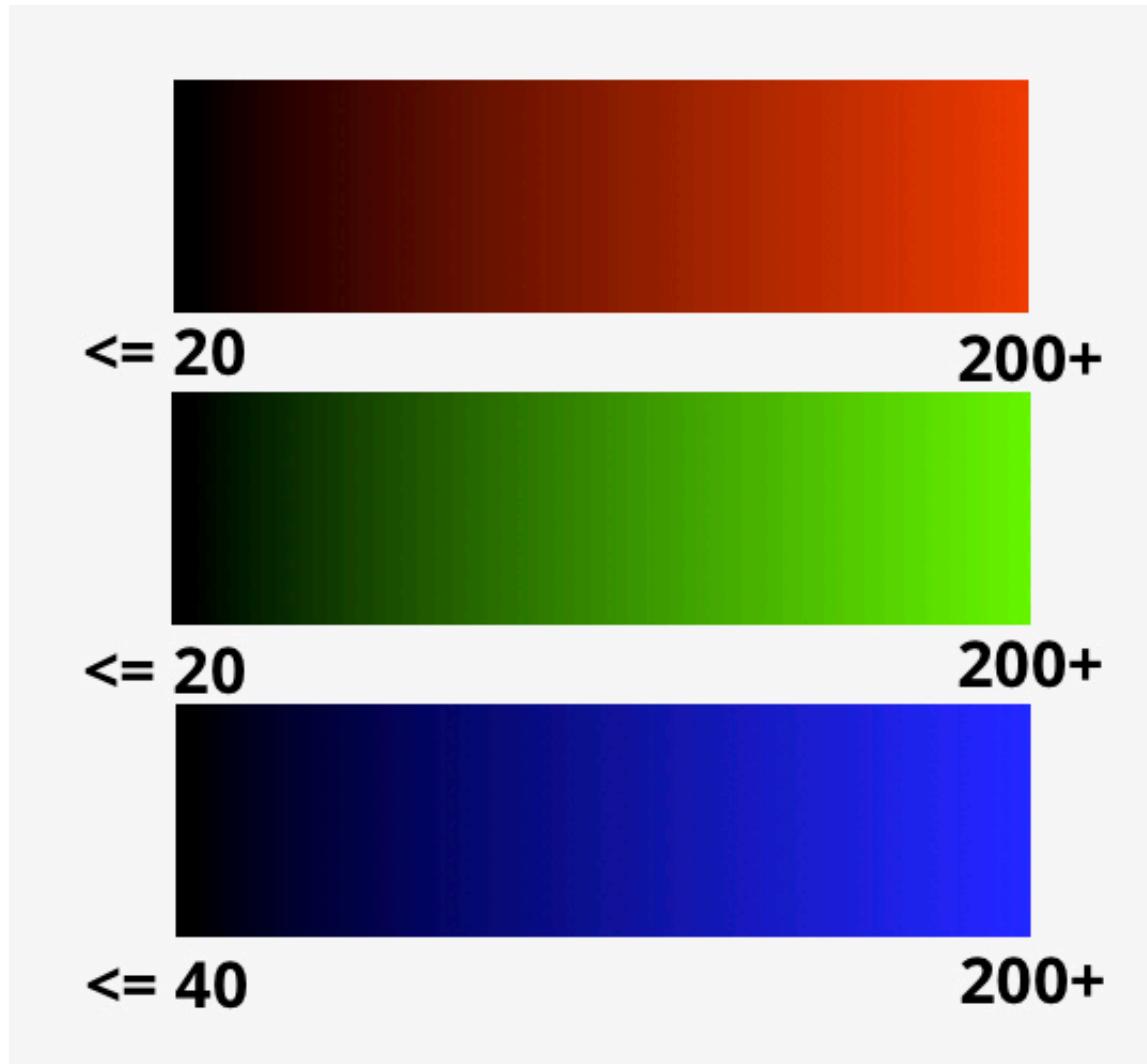
#green {
    raster-scaling:gaussian;
    raster-comp-op:plus;
    raster-colorizer-default-mode:linear;
    raster-colorizer-default-color: transparent;
    raster-colorizer-epsilon:0.1;
    raster-colorizer-stops:
        stop(0,#000)
        stop(255,rgb(0,255,0))
}

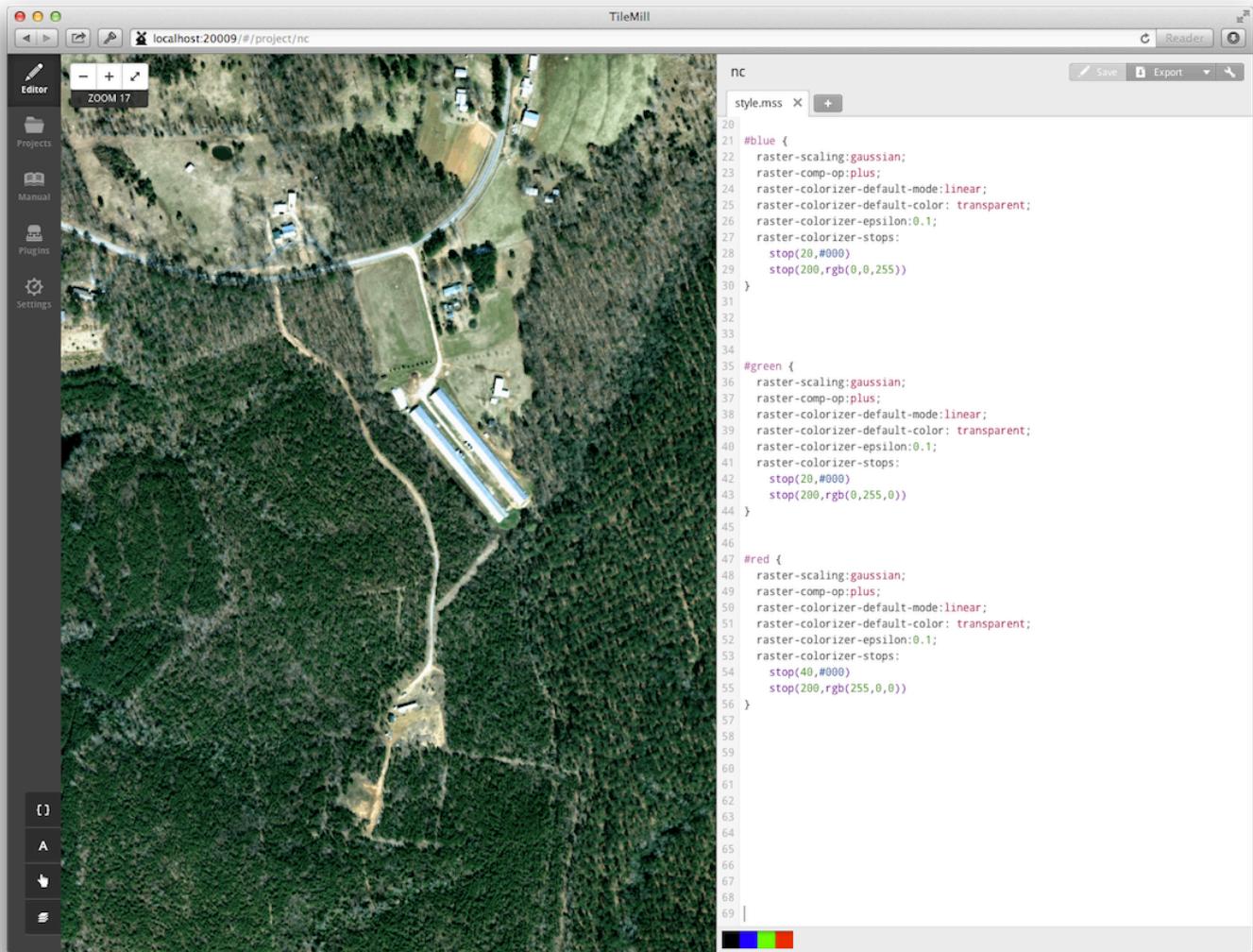
#blue {
    raster-scaling:gaussian;
```

```
raster-comp-op:plus;  
raster-colorizer-default-mode:linear;  
raster-colorizer-default-color: transparent;  
raster-colorizer-epsilon:0.1;  
raster-colorizer-stops:  
  stop(0,#000)  
  stop(255,rgb(0,0,255))  
}
```

现在就可以利用raster-colorizer-stops对每个波段对应的图层进行色彩校正了。

色彩校正可以从调整最大、最小和均值开始。我们发现红色和绿色波段在将最小值设为20、最大值设为200，蓝色波段的最小值设为40时具有最好的视觉效果。对于红色波段图层，所有像素值小于等于20的都会被置为最暗的深色，而所有大于等于200的像素都会被置为最亮的红色。





## 色彩校正后

```
#blue {
  raster-scaling: gaussian;
  raster-comp-op: plus;
  raster-colorizer-default-mode: linear;
  raster-colorizer-default-color: transparent;
  raster-colorizer-epsilon: 0.1;
  raster-colorizer-stops:
    stop(20, #000)
    stop(200, rgb(0, 0, 255))
}

#green {
  raster-scaling: gaussian;
  raster-comp-op: plus;
  raster-colorizer-default-mode: linear;
  raster-colorizer-default-color: transparent;
  raster-colorizer-epsilon: 0.1;
  raster-colorizer-stops:
    stop(20, #000)
    stop(200, rgb(0, 255, 0))
```

```

}

#red {
  raster-scaling: gaussian;
  raster-comp-op: plus;
  raster-colorizer-default-mode: linear;
  raster-colorizer-default-color: transparent;
  raster-colorizer-epsilon: 0.1;
  raster-colorizer-stops:
    stop(40, #000)
    stop(200, rgb(255, 0, 0))
}

```

## 离散栅格数据

### 离散栅格数据：土地覆盖

Contextually styling a discrete raster data set – a task once completed over several steps across different applications – can be completed within TileMill, our open source design studio. When you contextually style raster data, you bind a color value to particular pixel values, which is great for highlighting urban areas using a bright color, making no-data pixels appear transparent, and grouping similar categories, like types of tree cover, into larger categories and making them all green.

基于上下文的（译注：还是译成“考虑应用场景的”？）离散栅格数据的制图通常需要多个软件工具相互配合，并且经过多个步骤才能完成。而使用CartoCSS制图工具，这项工作则可以一站式搞定。在进行基于上下文的栅格制图时，具有特定值的像素将被赋予某种颜色。这在利用特别的颜色突出显示某些部分的场合下非常有用，例如用亮色表示城镇区域、让no-data值全部透明、将相同类别的地块合并使用一种颜色渲染（比如将同一种植被覆盖的区域全部使用绿色渲染）等。

这里我们就来基于栅格数据制作一幅土地覆盖图。原始数据来源于[日本空间局](#)，zip压缩包的下载地址在[这里](#)。

唯一需要在预处理阶段做的就是将该数据用gdalwarp重投影成Google Mercator投影。接下来的样式配置工作全部可以在CartoCSS制图工具中完成。

### 预处理

在下载并解压得到GeoTIFF数据之后，需要将其重投影，具体方法可以参考[Natural Earth GeoTiff](#)，我们在这里不赘述。

在终端中运行下面这段shell脚本可以将所有位于target目录中的影像数据重投影：

```

ls *.tif > abc
mkdir target
while read line
do
  file=$(echo $line | awk -F. '{ print $1 }')
  gdalwarp -t_srs EPSG:3857 $line target/$file.tif
done < abc

```

## 构建虚拟数据集

很多CartoCSS制图工具（例如TileMill）原生支持**GDAL虚拟栅格数据格式**。因此我们就可以充分利用这一特性，而不需要将所有的影像重新拼接成一个新的GeoTIFF。使用**gdalbuildvrt**工具可以基于原始影像数据集生成一个XML文件，然后它就可以被CartoCSS制图工具识别成一个拼接好的影像了。请注意在用**gdalbuildvrt**处理原始影像时要使用绝对路径。

```
$ gdalbuildvrt mosaic.vrt /absolute/path/to/input/tiffs/*.tif
```

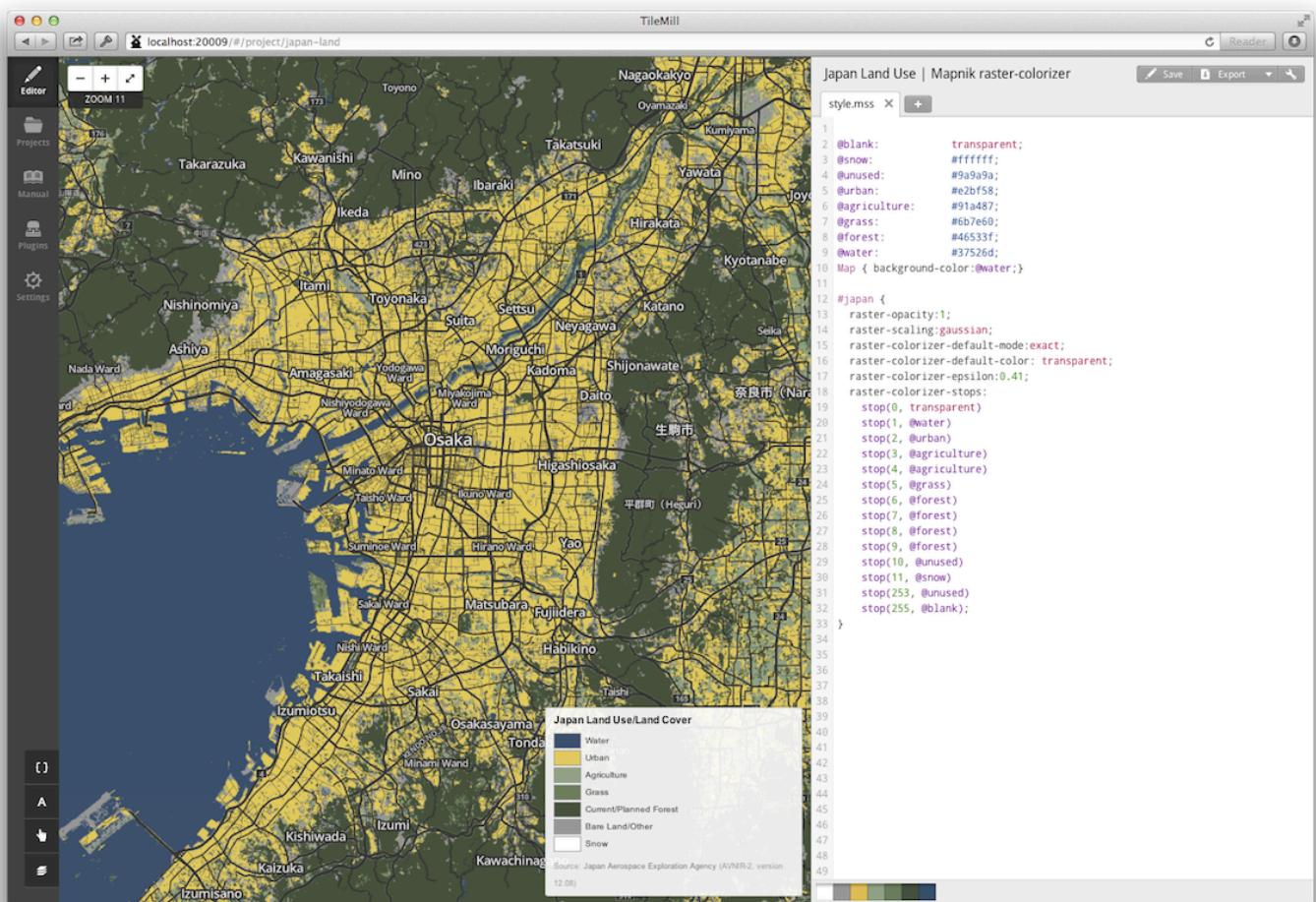
## 在CartoCSS制图工具中配置样式

首先需要注意的一点是，在添加图层的时候要在高级设置中加上band=1，否则着色器会工作不正常。

由于我们使用的土地覆盖GeoTIFF数据中的每个像素的值都直接对应某种地块分类，所以需要使用**raster-colorizer-default-mode: exact**属性来指明每个颜色值都与特定的像素值一一对应，而在相邻的颜色值之间不需要插值填充。

剩下要做的就是把土地利用数据中的各种像素值与不同的颜色对应起来。这在**raster-colorizer**中应该采用以下语法：

```
stop( + pixel value + , + color to assign + )
```

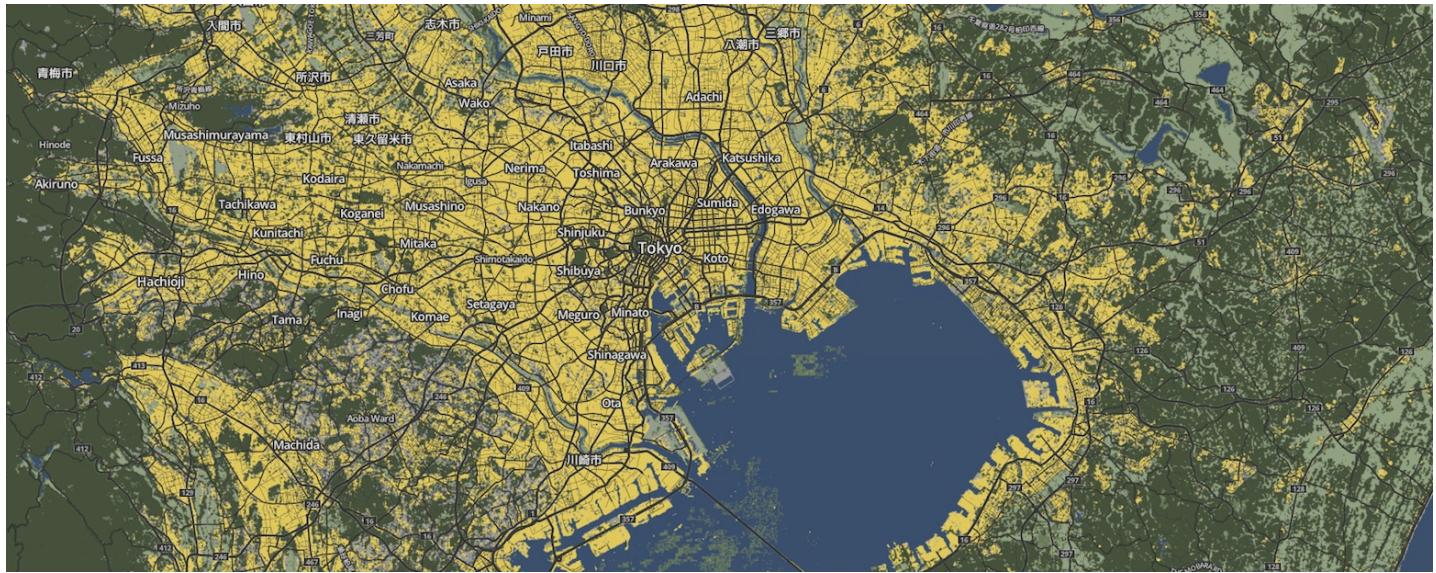


```
@blank: transparent;
@snow: #ffffff;
@unused: #9a9a9a;
@urban: #e2bf58;
@agriculture: #91a487;
@grass: #6b7e60;
@forest: #46533f;
@water: #37526d;

Map { background-color:@water; }

#japan {
  raster-opacity:1;
  raster-scaling:gaussian;
  raster-colorizer-default-mode:exact;
  raster-colorizer-default-color: transparent;
  raster-colorizer-epsilon:0.41;
  raster-colorizer-stops:
    stop(0, transparent)
    stop(1, @water)
    stop(2, @urban)
    stop(3, @agriculture)
    stop(4, @agriculture)
    stop(5, @grass)
    stop(6, @forest)
    stop(7, @forest)
    stop(8, @forest)
    stop(9, @forest)
    stop(10, @unused)
    stop(11, @snow)
    stop(253, @unused)
    stop(255, @blank);
}
```

最终效果图



## 地形数据制图

为了得到赏心悦目的地形图，我们通常需要利用[GDAL](#)中的DEM工具生成若干种不同类型的可视化效果，然后再将它们合理组合。

### 获取数字高程模型（DEM）

用于存储DEM数据的格式有很多种。在我们的例子中将使用GeoTIFF。下面列出几个提供高质量免费GeoTIFF格式地形数据的数据源：

#### SRTM

数据来源于NASA的[Shuttle Radar Topography Mission](#)。它是一个高程数据的高质量数据源，数据覆盖了地球表面的绝大多数地区。尽管SRTM数据可以从NASA直接免费下载，但我们还是推荐使用[CGIAR的净化版](#)，它也同样是免费的。

#### ASTER

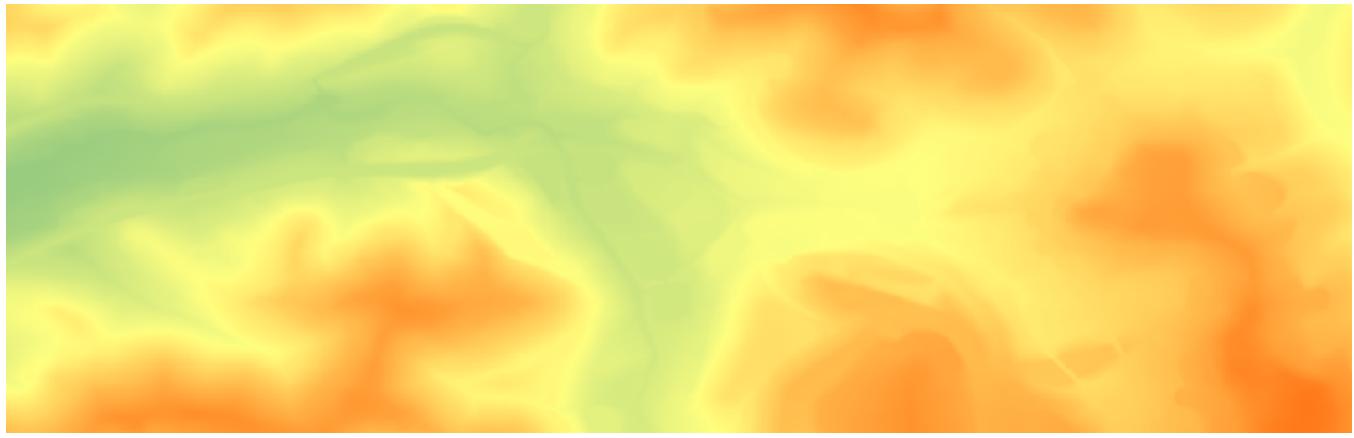
Aster是另一个全球DEM数据源。与SRTM相比，Aster对地球表面的覆盖率更高，而且分辨率也稍高一些。但Aster数据中的误差比CGIAR的净化版SRTM数据多。这些误差通常是一些尖峰或凹坑，而且还比较明显。

#### USGS NED

美国地质调查局也发布了一个覆盖全美的[国家高程数据集（National Elevation Dataset）](#)。它的分辨率高，而且从多个来源频繁更新。

### 地形数据可视化的类型

#### 彩色地形图（也叫分层设色）



Color relief assigns a color to a pixel based on the elevation of that pixel. The result is not intended to be physically accurate, and even if natural-looking colors are chosen the results should not be interpreted as representative of actual landcover. Low-lying areas are often given assigned green and yellow shades, with higher elevations blending to shades of grey, white, and/or red.

彩色地形图是根据每个像素的高程值赋予其一个颜色。其效果并不追求物理上的精确，而且即使是选择了自然光效果的颜色，也不应该将其与地块类型联系起来（译注：就是说渲染成绿色的地区不是指的那里就是森林绿地）。地势低洼的地区通常使用绿色和黄色来渲染，而在海拔较高的地区则倾向于使用灰色、白色和/或红色来渲染。

#### 山体阴影图（也叫地形阴影图）

山体阴影图是将数字高程模型进行分析并模拟出三维地形的一种可视化效果图。阳光照射山体产生阴影的效果不要求精确，但提供了对真实地形可视化效果的一种良好近似。

#### 坡度图

坡度图基于每个像素与其周围像素的高程差来为该像素赋予一个颜色。使用坡度图可以让陡峭的山体在地图上更加明显，从而增强地形起伏的视觉效果。

#### 使用gdaldem实现地形可视化

##### 对数据重投影

原始的DEM数据通常都不是Google Mercator投影。例如SRTM的参考系是[WGS84](#) (EPSG:4326)，而USGS NED是[NAD83](#) (EPSG:4269)，所以它们在使用之前都需要重投影。

例如我们需要用哥伦比亚大区的NED数据来作图，那么需要使用以下命令对其重投影（参见前面关于栅格数据重投影的介绍）。

```
gdalwarp -s_srs EPSG:4269 -t_srs EPSG:3785 -r bilinear dc.tif dc-3785.tif
```

请注意，在对高程数据重投影的时候，`-r bilinear`参数非常重要。因为其它的重采样方法会导致结果图像中产生怪异的条纹或网格。

##### 制作山体阴影图

执行以下命令可以从原始地形数据中生成山体阴影数据：

```
gdaldem hillshade -co compress=lzw dc-3785.tif dc-hillshade-3785.tif
```

-co compress=lzw参数将对TIFF数据进行压缩。如果你的磁盘空间足够大，那么也可以不加这个压缩参数。如果你使用的是1.8.0以上版本的GDAL（译注：翻译本文档时GDAL的最新版本为1.11），那么还可以加上`-compute_edges`参数以防止地形数据的周围被黑色像素填充。

输出结果如下图：



注意：如果数据的水平和垂直方向的度量单位比例关系不是1:1，那么需要加上`-s`（即`scale`，比例）参数来指定其差别。由于在Google Mercator投影中X和Y方向都是以米为单位，而NED数据也是以米为单位的，所以不需要加这个参数。但如果高程数据是以英尺为单位存储的，那么就需要加这个参数了（因为它们的换算关系大约是1米等于3.28英尺）：

```
gdaldem hillshade -s 3.28 -co compress=lzw dc-3785.tif dc-hillshade-3785.tif
```

更多详细信息请参见[gdaldem相关文档](#)。

制作彩色地形图

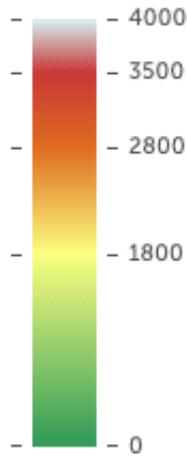
在制作彩色地形图之前，你需要先想好对于不同的海拔高度应该赋予什么颜色。这个色彩配置将被存储在一个具有特定格式的文本文件中。文件中的每一行都是四个数字：先是高程值，然后是RGB的三个分量（取值范围都是0到255）。为了得到合理的RGB颜色，你可以借助某一个图像编辑工具中的调色板，或者利用像[ColorPicker.com](#)这样的在线工具。

我们这里给出一个例子，保存在名为ramp.txt的文件中：

```
0 46 154 88
1800 251 255 128
2800 224 108 31
3500 200 55 55
```

```
4000 215 244 244
```

上面的色彩配置定义了一组包含5个颜色的色阶，涵盖了高度落差为4000个度量单位的区间（我们这个数据集中是以米为单位的，但在这个文件中不需要明确具体的单位）。这个色彩配置会被解译成一个颜色-高程对应关系的色带：



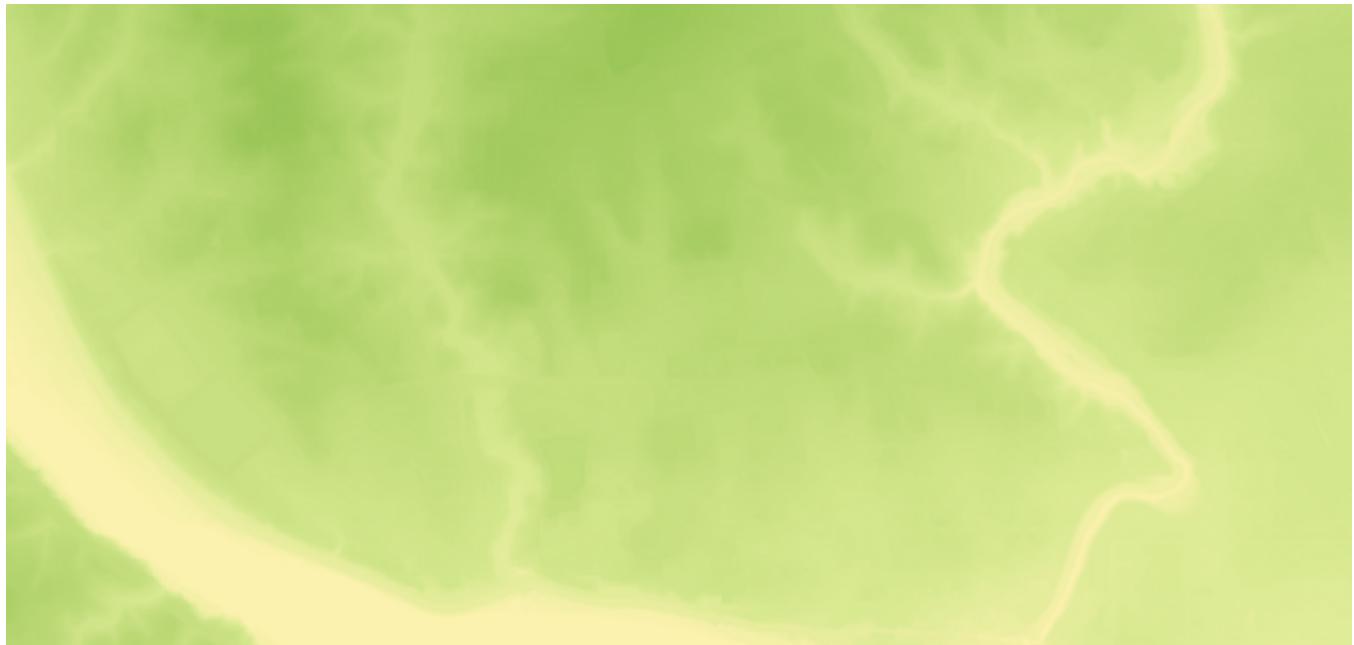
要在DEM数据上应用这个色带，可以使用gdaldem color-relief命令：

```
gdaldem color-relief input-dem.tif ramp.txt output-color-relief.tif
```

但在你选定要制图的目标区域中，未必会有这么大跨度的海拔高度落差。你可以根据目标区域中的最高与最低海拔值来调整色彩配置。利用gdalinfo命令可以帮助你看到GeoTIFF中的高程值的统计信息：

```
gdalinfo -stats your_file.tif
```

抛开其它信息，这个命令可以告诉你的高程数据中海拔高度的最大值、最小值和均值，从而辅助你设计色彩配置表。在调整了色彩配置后，华盛顿特区周边地区的彩色地形图如下：



## 制作坡度阴影图

利用gdaldem工具，通过一个两步过程就能够生成坡度可视化图。

首先，我们生成一个坡度tif，其中每个像素为一个0到90之间的角度值，表示对应区域的坡度。

```
gdaldem slope dc-3785.tif dc-slope-3785.tif
```

这里也需要考虑水平和垂直方向的度量单位比例，和制作山体阴影图时一样。

gdaldem slope生成的tif是没有为其中的像素值赋予颜色的，但可以通过gdaldem color-relief工具和色彩配置文件为其着色。

创建一个名为slope-ramp.txt的文件，包含以下两行内容：

```
0 255 255 255  
90 0 0 0
```

然后在调用color-relief命令时应用该色彩配置文件，即可得到坡度为0°（即完全平坦）的地方被渲染为白色，90°直上直下的峭壁处被渲染成黑色，而中间的其它值则被渲染成各级灰度。使用如下命令实现该效果：

```
gdaldem color-relief -co compress=lzw dc-slope-3785.tif slope-ramp.txt dc-slopeshade-3785.tif
```

然后得到的结果如下图所示：



## 合并最终结果

为了将最终结果合并，我们需要利用raster-comp-op合成操作中的multiply混色模式。假设你已经将前面制作完成的三个GeoTIFF数据集加入了CartoCSS制图工具，得到三个图层，并分别为彩色地形图、坡度阴影图和山体阴影图赋予了color-relief、slope-shade和hill-shade作为图层ID，那么就可以应用下面的样式制图了。你可以根据需要调整山体阴影和坡度图层的透明度以达到最佳效果。

```
#color-relief,  
#slope-shade,  
#hill-shade {  
    raster-scaling: bilinear;  
    raster-comp-op: multiply;  
}  
  
#hill-shade { raster-opacity: 0.6; }  
  
#slope-shade { raster-opacity: 0.4; }
```

最终的结果如下图所示，它还可以再与其它矢量图层进一步叠加。



## 关于性能

如果你的栅格数据有很多MB（译注：这应该不算大吧，几个GB的可能还算），那么一个重要的优化手段就是构建影像金字塔，或缩略图。这可以利用gdaladdo工具或在QGIS中完成。构建缩略图不会影响原始影像的分辨率，它只是在原始文件中增加一系列分辨率递减的影像，从而在缩放级别较低时不必加载数据量较大的原始影像。然而如果对原始影像进行了编辑，那么之前构建的金字塔和缩略图都将会失效，需要重新构建。你可以利用gdalinfo工具来查看你的tif文件中是否已经包含了金字塔/缩略图。如果在输出的关于每个波段的信息中有Overviews关键字，那么就说明已经构建了金字塔/缩略图。

## 参考文献

1. Mapbox, [Georeferencing Satellite Images](#)
2. Mapbox, [Colorizing Single-band Raster Data](#)
3. Mapbox, [Color Correction of RGB Imagery](#)
4. Mapbox, [Discrete Raster Data](#)
5. Mapbox, [Working with Terrain Data](#)

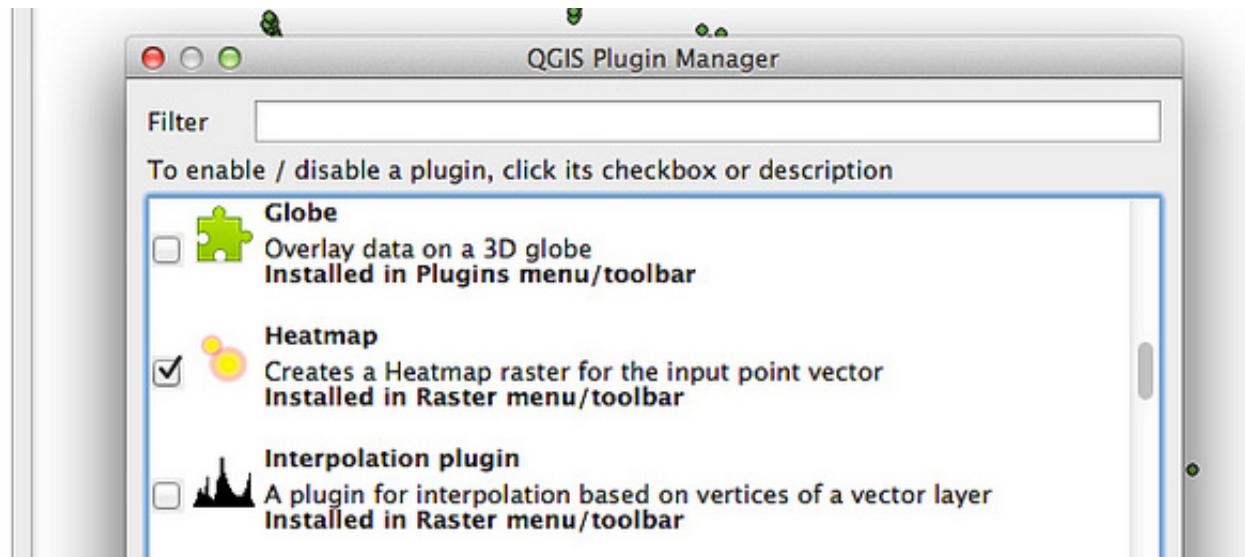
## 4.7 实例：制作专题地图

在本章的最后，我们看一个10分钟的制图实例教程，它将演示如何利用QGIS生成热图，以及如何在TileMill中将其制图可视化。

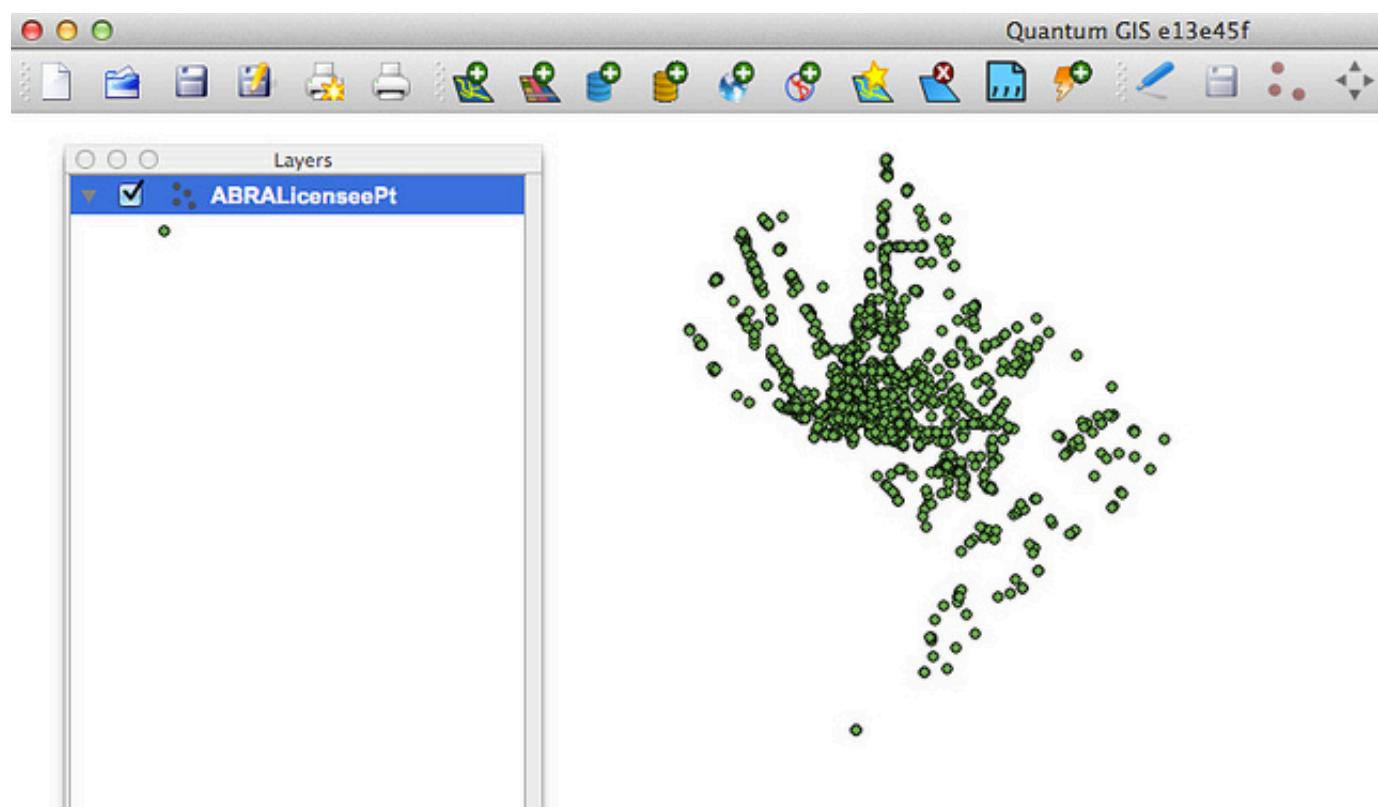
## 使用QGIS

在开始工作之前，请确认你使用的QGIS版本至少是1.9，这是撰写本文档时的最新开发版本（译注：在整理译稿时的最新稳定版本是2.8）。Mac用户可以从[kyngchaos](#)下载到最新版（译注：也可以通过Homebrew安装）。

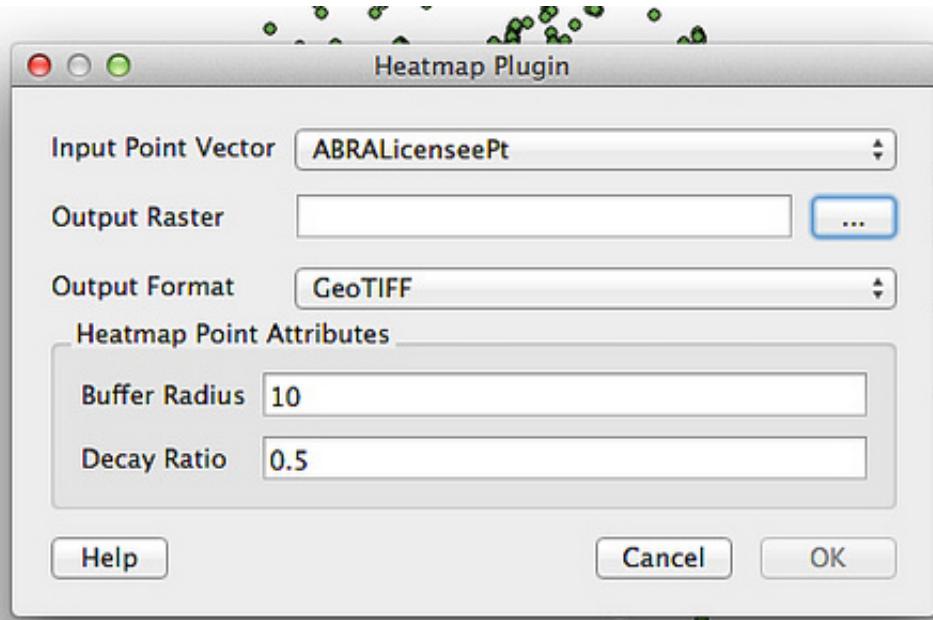
接下来你需要激活QGIS中的Heatmap插件。方法是从Plugins > Manage plugins菜单中打开如下面板，然后在Heatmap插件前面的复选框打上勾。注意：这个插件只有在1.9以上版本的QGIS中才可以使用。重启QGIS使该设置生效。



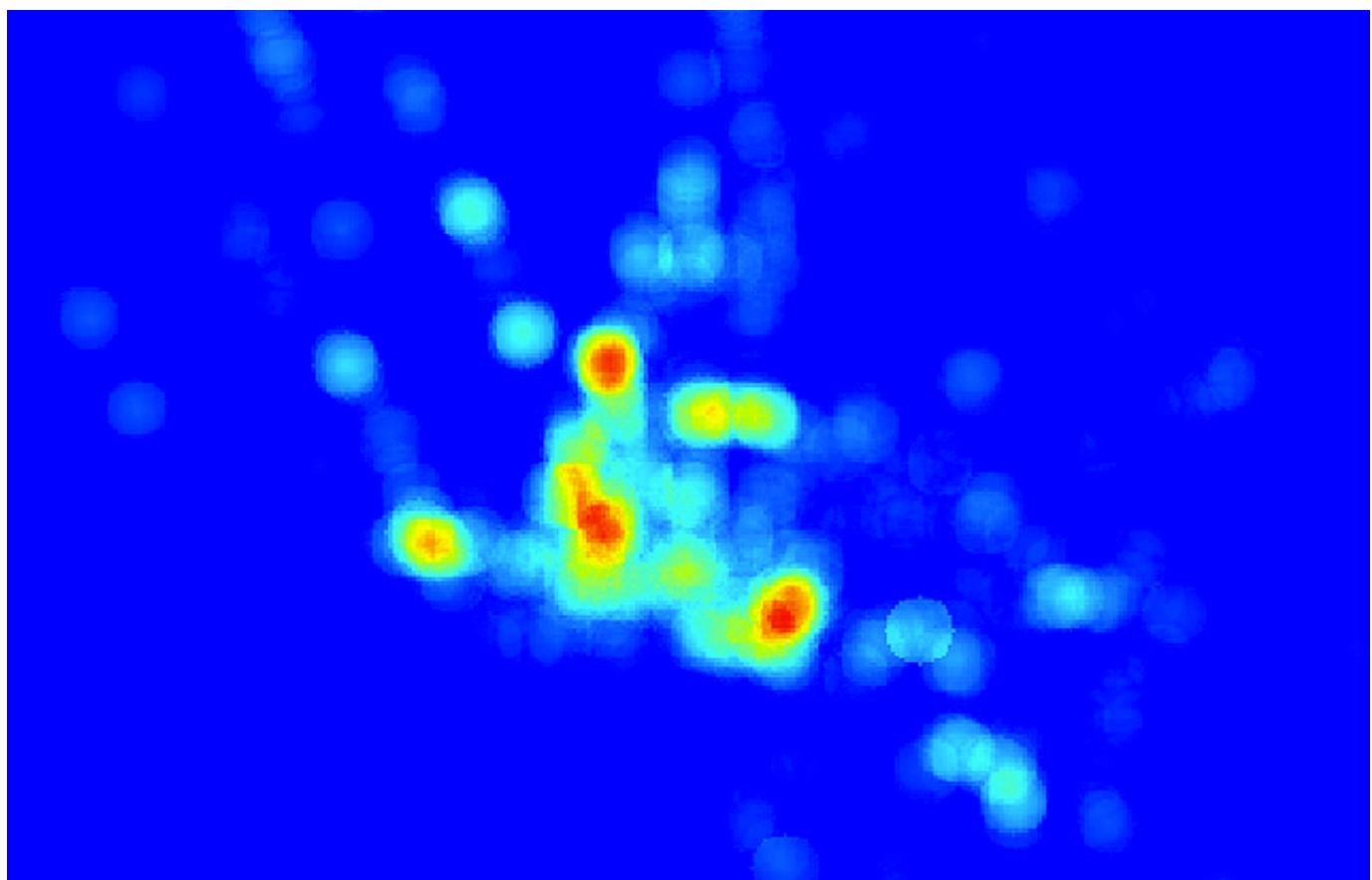
现在需要加入一些点数据。下面这个数据集由华盛顿特区酒精饮料管理局提供，发布于华盛顿特区开放数据门户网站[data.dc.gov](http://data.dc.gov)。数据的下载地址在[这里](#)。



依次进入菜单Raster > Heatmap > Heatmap，然后可以看到如下对话框：



指定输出文件的路径（带.tif后缀），暂时保持其它的设置为默认值，然后点击OK按钮即可生成热图。此时项目中会多出一个新的图层，它是一个巨大的灰度矩形。打开这个新图层的属性面板可以看到它不同的颜色值。为了快速得到一个粗糙的可视化效果，我们使用QGIS中内置的一个配色模板。在Style标签页的Colormap下拉框中选中Pseudocolor并点击OK，就可以得到一幅如下效果的地图：



既然已经得到了这幅地图，那么我们就可以再试试调整一下它的颜色和其它参数。默认的配置简单是简单，但却很难将数据表现得赏心悦目。在Heatmap对话框的帮助下，有关于“空间缓冲区”和“衰变率”的解释。而在前一节“栅格制图技巧”中，也讲述了如何使用自定义的配色方案来渲染栅格数据。为了能够在接下来的步骤中使用TileMill制图并制作瓦片，需要保证输出文件的格式为GeoTIFF。

在TileMill中新建一个项目，把之前QGIS中保存的GeoTIFF文件作为一个图层加载进来。删掉创建项目时默认加入的#countries层，修改地图背景，得到如下CartoCSS代码：

```
#heatmap {  
    raster-opacity: 1;  
    raster-scaling: bilinear;  
}
```

此时地图应该已经出现了。关于栅格数据样式的配置请参考前节内容。参考AJ设计的[世界人口分布图](#)可以得到更多灵感。

## 用TileMill实现伪热图

TileMill won't generate rasterized heatmaps like the QGIS plugin can, but you can approximate the effect with a few CartoCSS tricks to take advantage of aggregated opacity: low opacity of individual points means that overlap in dense areas has a stronger, more saturated color value.

TileMill不会像QGIS插件一样生成栅格化的热图。但是利用一些CartoCSS技巧，可以实现近似的效果。这些技巧包括聚合透明度：低透明度的点意味着在密集区域相互叠加会得到强度（译注：stronger是指什么？）和饱和度都更高的颜色值。

对原始矢量数据shapefile使用如下几行CartoCSS即可得到所需的效果：

```
#abralicenseept [DESCRIPTION != 'Retailer B'] {  
    marker-width:4;  
    marker-fill:#ef0;  
    marker-opacity:.45;  
    marker-line-opacity:0;  
    marker-allow-overlap:true;  
}
```

注意：这段代码忽略了所有'Retailer B'，也就是去掉了那些杂货店点。这样可以更好的反映出华盛顿特区中酒吧的位置，以及夜生活的分布情况。

这种方法的好处在于能够在地图放大到较高级别时保留和展示每个点的精确信息，而且还可以支持要素级的交互。更多关于样式配置方面的内容，可以参见本章中“高级地图设计”一节。

## 参考文献

1. Mapbox, [Designing Heat Maps](#)

# 语言参考

CartoCSS提供了一系列用于定义地图样式的属性。以下列表中包含了这些属性的含义和所有可取的值。

## 所有符号的公共属性

### **image-filters functions**

默认值: `none` (不使用图像过滤器)

说明: 以函数形式提供的一组图像过滤器。图像过滤器会作用于处于活动状态的画布。如果设置了多个图像过滤器, 那么每增加一个过滤器都会触发创建一个新的画布, 当这个新的画布被渲染完成后, 再通过合成的方式与主画布合并。如果要直接在主画布上应用图像过滤器, 那么需要使用`direct-image-filters`属性。

---

### **direct-image-filters functions**

默认值: `none` (不使用图像过滤器)

说明: 作用于主画布上的图像过滤器 (参见`image-filters`)

---

### **comp-op keyword**

取值范围: `clear src dst src-over dst-over src-in dst-in src-out dst-out src-atop dst-atop xor plus minus multiply screen overlay darken lighten color-dodge color-burn hard-light soft-light difference exclusion contrast invert invert-rgb grain-merge grain-extract hue saturation color value`

默认值: `src-over` (把当前图层覆盖在其它图层之上)

说明: 合成操作。该属性用于定义当前图层与其相邻图层如何合成。关于合成操作, 请参见本书基础用法一章中关于合成操作一节的内容。

---

### **opacity float**

取值范围: 0到1

默认值: `1` (不透明)

说明: 设置透明度。为样式设置alpha值 (首先, 创建一个独立的缓冲区, 在这个缓冲区中为所有要素应用alpha实现透明化, 然后再把这个独立缓冲区合成到主缓冲区中)

---

## 地图 (`map`) 的属性

与其它十种符号不同, 本节列出的是用于配置地图整体样式的属性。

### **background-color color**

默认值: none (透明色)

说明: 设置地图的背景颜色。

---

### **background-image uri**

默认值: (透明色)

说明: 设置一张以平铺形式置于最底层的背景图片。

---

### **background-image-comp-op keyword**

取值范围: clear src dst src-over dst-over src-in dst-in src-out dst-out dst-atop dst-atop xor plus minus multiply screen overlay darken lighten color-dodge color-burn hard-light soft-light difference exclusion contrast invert invert-rgb grain-merge grain-extract hue saturation color value

默认值: src-over (背景图片被置于所设置的背景色上一层)

说明: 设置背景图片与背景颜色之间的合成操作方式。

---

### **background-image-opacity float**

默认值: 1 (背景图片的透明度保持不变)

说明: 设置背景图片的透明度。

---

### **srs string**

默认值: +proj=longlat +ellps=WGS84 +datum=WGS84 +no\_defs (这是EPSG:4326空间参考系的proj4表达形式。地图中所有图层的数据都会采用这同一种参考系来绘制。如果地图中的某一图层没有显式声明自己所使用的参考系,那么这个图层将被认为与地图的参考系相同,并且在绘制的时候不会对这个图层中包含的数据进行座标变换。)

说明: 设置地图的空间参考系 (以proj4字符串表达)。

\*\*\*

### **buffer-size float**

默认值: 0 (无缓冲区)

说明: 在地图周围增加一圈额外的绘制区域 (以像素数表达)。这个属性的设置是为了保证那些出现在地图边界附近的文本标注不至于在渲染时被截断。注意这个属性不应该与avoid-edges同时使用。

---

### **base string**

默认值: (工作路径默认为空。此时在样式定义中所有通过相对路径的方式引用的外部资源文件都会以应用程序所在的路径为父目录去寻址。)

说明：如果map是从内存中加载的，那么所有以相对路径方式引用的外部资源则均为相对于由该base属性定义的路径。如果map是从文件系统中加载的，并且这个base属性没有被显式设置，那么base的值就是样式文件所在的目录。

---

### **font-directory uri**

默认值： none (不注册专门用于当前map的字体)

说明：指定专门用于当前map的字体目录（自动加载的默认字体除外）

---

## 线符号（line）的属性

### **line-color color**

默认值： rgba(0,0,0,1) (完全不透明的黑色)

说明：设置线要素的线条颜色。

---

### **line-width float**

默认值： 1

说明：设置线要素的线条宽度，单位为像素。

---

### **line-opacity float**

默认值： 1 (不透明)

说明：设置线要素的透明度。

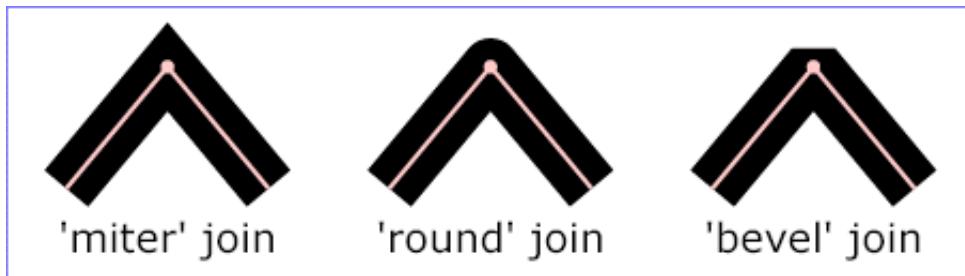
---

### **line-join keyword**

取值范围： miter round bevel

默认值： miter

说明：设置线要素之间在交汇点处如何绘制。三种绘制方法的比较如下图所示。



### **line-cap keyword**

取值范围: butt round square

默认值: butt

说明: 设置线要素的端点形状。

---

### **line-gamma float**

取值范围: 0到1

默认值: 1 (完全抗锯齿)

说明: 设置绘制线要素时的抗锯齿级别。

---

### **line-gamma-method keyword**

取值范围: power linear none threshold multiply

默认值: power (使用 $\text{pow}(x, \gamma)$ 来计算像素 $\gamma$ 值。与linear相比, 应用power值绘制出来的线与面要素更加平滑。而其它的取值通常只是用来关闭抗锯齿。)

说明: 设置抗锯齿的具体算法, 控制绘制质量。在底层的Mapnik渲染引擎中, 这个方法和 $\gamma$ 值(默认为1)结合使用。其代码位于AGG中, 地址在[这里](#)。

---

### **line-dasharray numbers**

默认值: none (实线 (无虚线效果))

说明: 通过设置 $[a, b]$ 的值设置虚线样式。其中 $a$ 为虚线段的长度,  $b$ 为虚线段间隔的长度。此外, 还可以设置更多的值, 从而获得更加复杂的绘制效果。

---

### **line-miterlimit float**

默认值: 4 (当 $\theta$ 角小于29度时, 自动将线要素交汇样式从miter改为bevel)

说明: 设置线端的切角长度与线宽的比例上限。当线要素交汇处出现尖锐的锐角时, 由于切角与线宽比例失调会导致错误的绘制结果。设置了该属性则会在出现上述情况时自动将线要素交汇样式从miter改为bevel。一般情况下, 这个属性不需要显式设置, 但有时可以通过设定一个较大的值可以避免出现参差不齐的不良绘制效果。

---

### **line-clip boolean**

默认值: true (几何要素会根据地图的地理范围进行切割)

说明：为了提高绘制效率，可以先将矢量要素中所有超出地图边界的部分切掉，再进行绘制。但在某些情况下，为了防止出现绘制错误，也可以通过将该值设为`false`而不采用这个策略。

---

### **line-simplify float**

默认值： 0 (不对几何要素进行简化)

说明：如果要对几何要素按照地图综合的方法进行简化，那么通过该属性来设定阈值。

---

### **line-simplify-algorithm keyword**

取值范围：`radial-distance` `zhao-saalfeld` `visvalingam-whyatt`

默认值： `radial-distance` (不使用`radial-distance`算法进行简化)

说明：设置对线要素进行综合的简化算法。

---

### **line-smooth float**

取值范围：0到1

默认值： 0 (不对拐点进行平滑)

说明：对线的拐点进行平滑处理。0表示不进行平滑，1表示完全平滑。如果取值大于1，会导致绘制的几何要素扭曲变形。

---

### **line-offset float**

默认值： 0 (无偏移)

说明：将线要素相对于其原有位置向左（沿着线的走向）或向右偏移一定量的像素绘制。正值表示左偏，负值表示右偏。

---

### **line-rasterizer keyword**

取值范围：`full` `fast`

默认值： `full`

说明：设置线渲染方式，可以通过牺牲部分精确度以换取绘制速度。

---

### **line-geometry-transform functions**

默认值： `none` (不对几何要素进行变换)

说明：为几何要素定义变换函数。

---

## **line-comp-op keyword**

取值范围: clear src dst src-over dst-over src-in dst-in src-out dst-out src-atop dst-atop xor plus minus multiply screen overlay darken lighten color-dodge color-burn hard-light soft-light difference exclusion contrast invert invert-rgb grain-merge grain-extract hue saturation color value

默认值: src-over (将当前符号置于其它符号的上一层)

说明: 这也是一个合成操作。它定义了当前的符号应该如何与其相邻图层进行合成。

---

## 面符号 (polygon) 的属性

### **polygon-fill color**

默认值: rgba(128,128,128,1) (完全不透明的灰色)

说明: 设置面要素的填充色

---

### **polygon-opacity float**

默认值: 1 (不透明)

说明: 面要素的透明度 (0为完全透明, 1为完全不透明)

---

### **polygon-gamma float**

取值范围: 0到1

默认值: 1 (完全抗锯齿)

说明: 设置面要素边缘的抗锯齿级别, 影响绘制效果和速度。抗锯齿级别越高 (最高为1), 绘制效果越好, 但绘制速度最慢; 反之, 绘制效果最差, 但绘制速度最快。注意这里所说的绘制速度的快慢只是理论上的, 实际效果与软硬件环境密切相关。

---

### **polygon-gamma-method keyword**

取值范围: power linear none threshold multiply

默认值: power (使用 $pow(x, gamma)$ 来计算像素 $gamma$ 值。与linear相比, 应用power值绘制出来的线与面要素更加平滑。而其它的取值通常只是用来关闭抗锯齿。)

说明: 设置抗锯齿的具体算法, 控制绘制质量。在底层的Mapnik渲染引擎中, 这个方法和 $gamma$ 值 (默认为1) 结合使用。其代码位于AGG中, 地址在[这里](#)。

---

### **polygon-clip boolean**

默认值: true (几何要素会根据地图的地理范围进行切割)

说明：为了提高绘制效率，可以先将矢量要素中所有超出地图边界的部分切掉，再进行绘制。但在某些情况下，为了防止出现绘制错误，也可以通过将该值设为`false`而不采用这个策略。

---

### **polygon-simplify float**

默认值： 0 (不对面要素的边线进行简化)

说明：如果要对面要素的边线按照地图综合的方法进行简化，那么通过该属性来设定阈值。（参见地图综合中的线简化算法，如Douglas-Peuker算法）

---

### **polygon-simplify-algorithm keyword**

取值范围：radial-distance zhao-saalfeld visvalingam-whyatt

默认值： radial-distance (不使用radial-distance 算法进行简化)

(译注：这里奇不奇怪？赋了radial-distance这个默认值，却不用它简化？)

说明：设置对面要素的边线进行综合的简化算法。

---

### **polygon-smooth float**

取值范围：0到1

默认值： 0 (不对拐点进行平滑)

说明：对边线的拐点进行平滑处理。0表示不进行平滑，1表示完全平滑。如果取值大于1，会导致绘制的几何要素扭曲变形。

---

### **polygon-geometry-transform functions**

默认值： none (不对几何要素进行变换)

说明：为几何要素定义变换函数。

---

### **polygon-comp-op keyword**

取值范围：clear src dst src-over dst-over src-in dst-in src-out dst-out src-atop dst-atop xor plus minus multiply screen overlay darken lighten color-dodge color-burn hard-light soft-light difference exclusion contrast invert invert-rgb grain-merge grain-extract hue saturation color value

默认值： src-over (将当前符号置于其它符号的上一层)

说明：这也是一个合成操作。它定义了当前的符号应该如何与其相邻图层进行合成。

---

## **点符号（point）的属性**

## **point-file uri**

默认值: none

说明: 设置用于绘制点符号的图像文件。

---

## **point-allow-overlap boolean**

默认值: false (不允许点符号相互压盖)

说明: 设置是否显式相互压盖的点符号。

---

## **point-ignore-placement boolean**

默认值: false (不在冲突检测器的缓存中存储几何形状的外包框)

说明: 设置是否允许在与当前要素重叠的位置放置其它要素。

---

## **point-opacity float**

取值范围: 0到1

默认值: 1 (完全不透明)

说明: 设置点符号的透明度。

---

## **point-placement keyword**

取值范围: centroid interior

默认值: centroid

说明: 设置点符号的放置方式。

---

## **point-transform functions**

默认值: (无变换)

说明: 设置SVG图形的变换方法。

---

## **point-comp-op keyword**

取值范围: clear src dst src-over dst-over src-in dst-in src-out dst-out src-atop dst-atop xor plus minus multiply screen overlay darken lighten color-dodge color-burn hard-light soft-light difference exclusion contrast invert invert-rgb grain-merge grain-extract hue saturation color value

默认值: src-over (将当前符号置于其它符号的上一层)

说明：这也是一个合成操作。它定义了当前的符号应该如何与其相邻图层进行合成。

---

## 文本符号（text）的属性

### **text-name expression**

默认值：

说明：设置文本符号上显示的文字。可以通过用中括号括起来的字段名来指定要使用的数据字段，例如 [column\_name]。

---

### **text-face-name string**

默认值： undefined

说明：设置文本符号所使用的字体。

---

### **text-size float**

默认值： 10

说明：设置文本符号中文字的字号，以像素为单位。

---

### **text-ratio unsigned**

默认值： 0

说明：设置折行后的文本所占比例。

---

### **text-wrap-width unsigned**

默认值： 0

说明：设置文本块在多长的时候进行折行，以字符为单位。

---

### **text-wrap-before boolean**

默认值： false

说明：控制文本文字的折行动作。如果该值为false，那么每一行文本都会比wrap-width属性设定的值略长。

---

### **text-wrap-character string**

默认值：

说明：使用设置的字符而非空格作为文本标注的折行字符。

---

#### **text-spacing `unsigned`**

默认值： `undefined`

说明：设置沿线绘制文本符号时每两个文本符号之间的间距。

---

#### **text-character-spacing `float`**

默认值： `0`

说明：设置文本文字的字间距，以像素为单位。

---

#### **text-line-spacing `unsigned`**

默认值： `0`

说明：设置文本文字的行间距。

---

#### **text-label-position-tolerance `unsigned`**

默认值： `0`

说明：设置文本标注相对于其理想位置的偏移量，以像素为单位（目前仅适用于线要素）

---

#### **text-max-char-angle-delta `float`**

默认值： `22.5`

说明：设置文本文字的最大折转角，以十进制角度为单位。这个值会在绘制时被换算成弧度，例如默认的22.5度会按照 $22.5/\text{math.pi}*180.0$ 公式被换算成0.3925弧度。这个值越大，则被绘制在尖锐转角处的文本符号会越少。

---

#### **text-fill `color`**

默认值： `#000000 (黑色)`

说明：设置文本文字的颜色。

---

#### **text-opacity `float`**

取值范围： 0到1

默认值： `1 (不透明)`

说明：设置文本文字的透明度，取值范围为0到1。

---

### **text-halo-fill color**

默认值： #FFFFFF (白色)

说明：设置文本文字边缘的光晕颜色。

---

### **text-halo-radius float**

默认值： 0 (无光晕)

说明：设置文本文字边缘的光晕大小，以像素为单位。

---

### **text-halo-rasterizer keyword**

取值范围： full fast

默认值： full

说明：设置用于渲染文字光晕的方法，速度优先还是质量优先。

---

### **text-dx float**

默认值： 0

说明：设置文本文字的水平偏移量，以像素为单位。正值表示向右偏移。

---

### **text-dy float**

默认值： 0

说明：设置文本文字的垂直偏移量，以像素为单位。正值表示向下偏移。

---

### **text-vertical-alignment keyword**

取值范围： top middle bottom auto

默认值： auto (自动，但受到dy值的影响。当dy>0时，取bottom；而当dy<0时，取top)

说明：设置文本符号相对于点要素座标的位置。

---

### **text-avoid-edges boolean**

默认值： false

说明：设置是否避免将文本标注置于绘制区域（通常为瓦片）的边缘处。

---

## **text-min-distance float**

默认值: undefined

说明: 设置文本符号之间的最小间距。

---

## **text-min-padding float**

默认值: undefined

说明: 设置文本符号在元瓦片中的最小边距。

---

## **text-min-path-length float**

默认值: 0 (无论路线长度是多少, 都要绘制文本符号)

说明: 如果设置了该值, 那么只有在当路线长度大于该值的时候才绘制文本符号。

---

## **text-allow-overlap boolean**

默认值: false (不允许文本符号相互压盖)

说明: 设置是否显式相互压盖的文本符号。

---

## **text-orientation expression**

默认值: undefined

说明: 设置文本旋转。

---

## **text-placement keyword**

取值范围: point line vertex interior

默认值: point

说明: 设置文本符号在对应几何要素上的放置方式。

---

## **text-placement-type keyword**

取值范围: dummy simple

默认值: dummy

说明: 设置文本符号之间相互避让的算法。simple表示使用由text-placements属性指定的基本算法。而dummy则表示不使用该特性。

---

## **text-placements string**

默认值：

说明：如果placement-type属性被设置为simple，那么就会依据该属性的值（即形如“POSITIONS, [SIZES]”的字符串）执行文本符号相互避让算法。例如：text-placements: "E,NE,SE,W,NW,SW";

---

## **text-transform keyword**

取值范围：none uppercase lowercase capitalize

默认值：none

说明：设置是否对文本字符进行大小写转换。

---

## **text-horizontal-alignment keyword**

取值范围：left middle right auto

默认值：auto

说明：设置文本文字的水平对齐方式。

---

## **text-align keyword**

取值范围：left right center auto

默认值：auto (默认的自动方式是居中对齐，但如果已经设置了placement-type属性，那么就会依据text-placements属性的值来对文字进行靠左或靠右对齐)

说明：设置文本文字的对齐方式。

---

## **text-clip boolean**

默认值：true (几何要素会根据地图的地理范围进行切割)

说明：为了提高绘制效率，可以先将矢量要素中所有超出地图边界的部分切掉，再进行绘制。但在某些情况下，为了防止出现绘制错误，也可以通过将该值设为false而不采用这个策略。

---

## **text-comp-op keyword**

取值范围：clear src dst src-over dst-over src-in dst-in src-out dst-out src-atop dst-atop xor plus minus multiply screen overlay darken lighten color-dodge color-burn hard-light soft-light difference exclusion contrast invert invert-rgb grain-merge grain-extract hue saturation color value

默认值：src-over (将当前符号置于其它符号的上一层)

说明：这也是一个合成操作。它定义了当前的符号应该如何与其相邻图层进行合成。

---

## 盾标符号 (**shield**) 的属性

### **shield-name expression**

默认值: `undefined`

说明: 设置盾标上显示的标注文字。可以通过用中括号括起来的字段名来指定要使用的数据字段, 例如 `[column_name]`。

---

### **shield-file uri**

默认值: `none`

说明: 设置显示在盾标文本后面的背景图片。

---

### **shield-face-name string**

默认值:

说明: 设置盾标上标注文字的字体与样式。

---

### **shield-unlock-image boolean**

默认值: `false` (盾标文字将被置于盾标背景图片的中心位置)

说明: 设置盾标文字与背景图片之间的位置关系。如果不想把盾标文本绘制在背景图的中心, 那么就应该将该属性值设置为`true`。

---

### **shield-size float**

默认值: `undefined`

说明: 设置盾标文字的大小, 以像素为单位。

---

### **shield-fill color**

默认值: `undefined`

说明: 设置盾标文字的颜色。

---

### **shield-placement keyword**

取值范围: `point line vertex interior`

默认值: `point`

说明：设置盾标的放置方式。`point`方式是将盾标置于点要素的位置，`line`方式是将盾标在线要素上沿线绘制多次，`vertex`方式是将盾标置于多边形的顶点位置，而`interior`方式则是将盾标置于面要素的内部。

---

### **shield-avoid-edges boolean**

默认值： `false`

说明：设置是否避免在地图或瓦片的边缘处绘制盾标。

---

### **shield-allow-overlap boolean**

默认值： `false` (不允许盾标与其它现有地图要素重叠)

说明：该属性用于设置在盾标与地图上其它符号出现压盖时，是否显示盾标。

---

### **shield-min-distance float**

默认值： 0

说明：设置相邻两个盾标符号（可以是相同的盾标，也可以是不同的）之间的最小距离。

---

### **shield-spacing float**

默认值： 0

说明：设置同一线要素上多次绘制的盾标之间的间隔。

---

### **shield-min-padding float**

默认值： 0

说明：设置盾标在瓦片上绘制时的最小边距。

---

### **shield-wrap-width unsigned**

默认值： 0

说明：设置盾标文本多长的时候需要折行。

---

### **shield-wrap-before boolean**

默认值： `false`

说明：控制盾标文本的折行动作。如果该值为`false`，那么每一行文本都会比`wrap-width`属性设定的值略长。

---

## **shield-wrap-character string**

默认值：

说明：设置盾标文本的折行字符。

---

## **shield-halo-fill color**

默认值： #FFFFFF (白色)

说明：设置盾标文本的光晕颜色。

---

## **shield-halo-radius float**

默认值： 0 (盾标文本无光晕效果)

说明：设置盾标文本光晕的大小，单位为像素。

---

## **shield-character-spacing unsigned**

默认值： 0

说明：设置盾标文字的字间距。该属性目前仅适用于点要素上的盾标。

---

## **shield-line-spacing unsigned**

默认值： undefined

说明：设置盾标文本中的行距，以像素为单位。

---

## **shield-text-dx float**

默认值： 0

说明：设置盾标文本的水平偏移量，以像素为单位。正值表示向右偏移。

---

## **shield-text-dy float**

默认值： 0

说明：设置盾标文本的垂直偏移量，以像素为单位。正值表示向下偏移。

---

## **shield-dx float**

默认值： 0

说明：设置盾标本身的水平偏移量，以像素为单位。正值表示向右偏移。

---

### **shield-dy float**

默认值： 0

说明：设置盾标本身的垂直偏移量，以像素为单位。正值表示向下偏移。

---

### **shield-opacity float**

默认值： 1

说明：设置盾标背景图片的透明度。

---

### **shield-text-opacity float**

默认值： 1

说明：设置盾标文本的透明度。

---

### **shield-horizontal-alignment keyword**

取值范围：left middle right auto

默认值：auto

说明：设置盾标相对于其中心点的水平对齐方式。

---

### **shield-vertical-alignment keyword**

取值范围：top middle bottom auto

默认值：middle

说明：设置盾标相对于其中心点的垂直对齐方式。

---

### **shield-placement-type keyword**

取值范围：dummy simple

默认值：dummy

说明：设置盾标之间相互避让的算法。`simple`表示使用由`shield-placements`属性指定的基本算法。而`dummy`则表示不使用该特性。

---

### **shield-placements string**

默认值：

说明：如果shield-placement-type属性被设置为simple，那么就会依据该属性的值（即形如“POSITIONS, [SIZES]”的字符串）执行盾标相互避让算法。例如：`shield-placements: "E,NE,SE,W,NW,SW";`

---

### **shield-text-transform keyword**

取值范围：none uppercase lowercase capitalize

默认值：none

说明：设置盾标文字的大小写方式。

---

### **shield-justify-alignment keyword**

取值范围：left center right auto

默认值：auto

说明：设置盾标文本的对齐方式。

---

### **shield-transform functions**

默认值：(无变换)

说明：设置SVG的变换函数。

---

### **shield-clip boolean**

默认值：true (几何要素会根据地图的地理范围进行切割)

说明：为了提高绘制效率，可以先将矢量要素中所有超出地图边界的部分切掉，再进行绘制。但在某些情况下，为了防止出现绘制错误，也可以通过将该值设为false而不采用这个策略。

---

### **shield-comp-op keyword**

取值范围：clear src dst src-over dst-over src-in dst-in src-out dst-out src-atop dst-atop xor plus minus multiply screen overlay darken lighten color-dodge color-burn hard-light soft-light difference exclusion contrast invert invert-rgb grain-merge grain-extract hue saturation color value

默认值：src-over (将当前符号置于其它符号的上一层)

说明：这也是一个合成操作。它定义了当前的符号应该如何与其相邻图层进行合成。

---

## **线图案 (line-pattern) 的属性**

### **line-pattern-file uri**

默认值: `none`

说明: 设置沿线重复绘制的图像文件。

---

#### **line-pattern-clip boolean**

默认值: `true` (*几何要素会根据地图的地理范围进行切割*)

说明: 为了提高绘制效率, 可以先将矢量要素中所有超出地图边界的部分切掉, 再进行绘制。但在某些情况下, 为了防止出现绘制错误, 也可以通过将该值设为`false`而不采用这个策略。

---

#### **line-pattern-simplify float**

默认值: `0` (*不对几何要素进行简化*)

说明: 如果要对几何要素按照地图综合的方法进行简化, 那么通过该属性来设定阈值。

---

#### **line-pattern-simplify-algorithm keyword**

取值范围: `radial-distance zhao-saalfeld visvalingam-whyatt`

默认值: `radial-distance` (*不使用radial-distance算法进行简化*)

说明: 设置对线要素进行综合的简化算法。

---

#### **line-pattern-smooth float**

取值范围: 0到1

默认值: `0` (*不进行平滑处理*)

说明: 对线的拐点进行平滑处理。0表示不进行平滑, 1表示完全平滑。如果取值大于1, 会导致绘制的几何要素扭曲变形。

---

#### **line-pattern-offset float**

默认值: `0` (*无偏移*)

说明: 将线要素相对于其原有位置向左 (沿着线的走向) 或向右偏移一定量的像素绘制。正值表示左偏, 负值表示右偏。

---

#### **line-pattern-geometry-transform functions**

默认值: `none` (*不对几何要素进行变换*)

说明: 为几何要素定义变换函数。

---

### **line-pattern-comp-op keyword**

取值范围: clear src dst src-over dst-over src-in dst-in src-out dst-out src-atop dst-atop xor plus minus multiply screen overlay darken lighten color-dodge color-burn hard-light soft-light difference exclusion contrast invert invert-rgb grain-merge grain-extract hue saturation color value

默认值: src-over (将当前符号置于其它符号的上一层)

说明: 这也是一个合成操作。它定义了当前的符号应该如何与其相邻图层进行合成。

---

## 面图案 (polygon-pattern) 的属性

### **polygon-pattern-file uri**

默认值: none

说明: 设置用于平铺填充面要素的图像文件。

---

### **polygon-pattern-alignment keyword**

取值范围: local global

默认值: local

说明: 设置填充时的对齐方式, local指在当前图层中对齐, global指在整个地图中对齐。

---

### **polygon-pattern-gamma float**

取值范围: 0到1

默认值: 1 (完全抗锯齿)

说明: 设置面要素边缘的抗锯齿级别, 影响绘制效果和速度。

---

### **polygon-pattern-opacity float**

默认值: 1 (保持填充图片的透明度不变)

说明: 设置填充图案的透明度。

---

### **polygon-pattern-clip boolean**

默认值: true (几何要素会根据地图的地理范围进行切割)

说明: 为了提高绘制效率, 可以先将矢量要素中所有超出地图边界的部分切掉, 再进行绘制。但在某些情况下, 为了防止出现绘制错误, 也可以通过将该值设为false而不采用这个策略。

---

## **polygon-pattern-simplify float**

默认值： 0 (不对面要素的边线进行简化)

说明：如果要对面要素的边线按照地图综合的方法进行简化，那么通过该属性来设定阈值。

---

## **polygon-pattern-simplify-algorithm keyword**

取值范围：radial-distance zhao-saalfeld visvalingam-whyatt

默认值：radial-distance (不使用*radial-distance*算法进行简化)

说明：设置对面要素的边线进行综合的简化算法。

---

## **polygon-pattern-smooth float**

取值范围：0到1

默认值：0 (不对拐点进行平滑)

说明：对线的拐点进行平滑处理。0表示不进行平滑，1表示完全平滑。如果取值大于1，会导致绘制的几何要素扭曲变形。

---

## **polygon-pattern-geometry-transform functions**

默认值：none (不对几何要素进行变换)

说明：为几何要素定义变换函数。

---

## **polygon-pattern-comp-op keyword**

取值范围：clear src dst src-over dst-over src-in dst-in src-out dst-out src-atop dst-atop xor plus minus multiply screen overlay darken lighten color-dodge color-burn hard-light soft-light difference exclusion contrast invert invert-rgb grain-merge grain-extract hue saturation color value

默认值：src-over (将当前符号置于其它符号的上一层)

说明：这也是一个合成操作。它定义了当前的符号应该如何与其相邻图层进行合成。

---

# 栅格符号 (**raster**) 的属性

## **raster-opacity float**

默认值：1 (不透明)

说明：设置栅格符号的透明度。

---

## **raster-filter-factor float**

默认值： -1 (允许数据源自行选用缩小图像尺寸的方法)

说明：用于栅格或GDAL数据源（译注：这个是Mapnik概念），对图像尺寸进行预先缩小。将该值调高可以得到更好的缩略图效果（译注：怎么才叫“好”？），但相应的处理时间也会变长。

---

## **raster-scaling keyword**

取值范围：near fast bilinear bilinear8 bicubic spline16 spline36 hanning hamming hermite kaiser quadric catrom gaussian bessel mitchell sinc lanczos blackman

默认值：near

说明：设置对栅格数据进行重采样的算法。bilinear可以在速度和质量方面得到不错的平衡，而lanczos则能够得到最高的绘制质量。

---

## **raster-mesh-size unsigned**

默认值：16 (取原始图像分辨率的1/16作为栅格的重投影网格大小)

说明：在对原始图像进行重投影时，是先将图像切分成若干网格，对网格中的小图像片分别重投影。如果设定该值使得网格的尺寸变大（译注：即把该属性的值调小），那么重投影的速度会加快，但可能会导致图像变形。

---

## **raster-comp-op keyword**

取值范围：clear src dst src-over dst-over src-in dst-in src-out dst-out src-atop dst-atop xor plus minus multiply screen overlay darken lighten color-dodge color-burn hard-light soft-light difference exclusion contrast invert invert-rgb grain-merge grain-extract hue saturation color value

默认值：src-over (将当前符号置于其它符号的上一层)

说明：这也是一个合成操作。它定义了当前的符号应该如何与其相邻图层进行合成。

---

## **raster-colorizer-default-mode keyword**

discrete``linear``exact

默认值：undefined

TODO

---

## **raster-colorizer-default-color color**

默认值：undefined

TODO

---

## **raster-colorizer-epsilon float**

默认值: undefined

TODO

---

## **raster-colorizer-stops tags**

默认值: undefined

TODO

---

# 注记符号（**markers**）的属性

## **marker-file uri**

默认值: (一个椭圆或正圆形符号)

说明: 设置绘制注记符号所使用的SVG文件。如果没有指定具体的文件，则默认使用一个椭圆形符号对每个位置的注记进行渲染。

---

## **marker-opacity float**

默认值: 1 (边缘和填充均不透明)

说明: 设置注记符号整体的透明度。如果设置了该属性，则会覆盖在marker-fill-opacity和marker-line-opacity属性中设置的透明度。

---

## **marker-fill-opacity float**

默认值: 1 (注记符号填充部分为不透明)

说明: 设置注记符号填充部分的透明度。

---

## **marker-line-color color**

默认值: black

说明: 设置注记符号边线的颜色。

---

## **marker-line-width float**

默认值: undefined

说明: 设置注记符号边线的宽度，以像素为单位。但如果该值设置过大导致注记本身被过粗的边线覆盖。

---

## **marker-line-opacity float**

默认值： 1 (注记符号边线完全不透明)

说明：设置注记符号边线的透明度。

---

## **marker-placement keyword**

取值范围： point line interior

默认值： point (注记符号被置于几何要素的重心（形心）位置)

说明：设置注记在几何要素上的放置方式，可以位于点要素上，或者在面要素的中心位置，还可以沿着线要素反复出现（通过设置marker-placement:line实现）。如果取值interior，那么可以确保注记符号被绘制在多边形的内部。

---

## **marker-multi-policy keyword**

取值范围： each whole largest

默认值： each (如果一个要素包含了多个几何形状，而且放置方式是point或interior，那么注记符号就会在每个几何形状处都会被绘制一次)

说明：该属性是为包含多个几何形状的地理（multi-geometries）要素准备的，对沿线放置的注记符号不起作用。其默认值为each，也就是每个几何形状上都会被绘制一个注记；whole表示注记将被绘制在所有几何形状组合后的重心位置；而largest表示注记将被绘制在最大（依据最小包围框的面积）的那个几何形状上（这也同样是文本标注在多几何要素上绘制的默认方法）。

---

## **marker-type keyword**

取值范围： arrow ellipse

默认值： ellipse

说明：设置默认的注记符号类型。如果没有指定用于渲染注记的SVG文件，那么内置的渲染引擎可以提供两种选择：箭头或椭圆。

---

## **marker-width expression**

默认值： 10

说明：设置注记符号的宽度。这个属性只适用于两种内置的默认注记样式。

---

## **marker-height expression**

默认值： 10

说明：设置注记符号的高度。这个属性只适用于两种内置的默认注记样式。

---

## **marker-fill color**

默认值: blue

说明: 设置注记的填充色。

---

## **marker-allow-overlap boolean**

默认值: false (不允许注记符号相互压盖 (被压盖的注记将不被显式))

说明: 设置被压盖的注记符号是否被显式在地图上。

---

## **marker-ignore-placement boolean**

默认值: false (不在冲突检测器的缓存中存储几何形状的外包围)

说明: 设置是否允许在与当前要素重叠的位置放置其它要素。

---

## **marker-spacing float**

默认值: 100

说明: 设置重复绘制的注记之间的间距, 单位为像素。如果设定的间距小于注记符号本身的尺寸, 或者大于线要素的长度, 那么注记就绘制不出来。

---

## **marker-max-error float**

默认值: 0.2

说明: 设置实际的注记位置与marker-spacing属性值之间的最大误差。如果将该属性值调高, 那么渲染引擎就会尝试处理与其它注记符号之间的位置冲突。

---

## **marker-transform functions**

默认值: (无变换)

说明: 设置SVG图形的变换方法。

---

## **marker-clip boolean**

默认值: true (几何要素会根据地图的地理范围进行切割)

说明: 为了提高绘制效率, 可以先将矢量要素中所有超出地图边界的部分切掉, 再进行绘制。但在某些情况下, 为了防止出现绘制错误, 也可以通过将该值设为false而不采用这个策略。

---

## **marker-smooth float**

取值范围：0到1

默认值：0 (不对拐点进行平滑)

说明：对线的拐点进行平滑处理。0表示不进行平滑，1表示完全平滑。如果取值大于1，会导致绘制的几何要素扭曲变形。

---

### **marker-geometry-transform functions**

默认值：none (不对几何要素进行变换)

说明：为几何要素定义变换函数。

---

### **marker-comp-op keyword**

取值范围：clear src dst src-over dst-over src-in dst-in src-out dst-out src-atop dst-atop xor plus minus multiply screen overlay darken lighten color-dodge color-burn hard-light soft-light difference exclusion contrast invert invert-rgb grain-merge grain-extract hue saturation color value

默认值：src-over (将当前符号置于其它符号的上一层)

说明：这也是一个合成操作。它定义了当前的符号应该如何与其相邻图层进行合成。

---

## **建筑物符号 (building) 的属性**

### **building-fill color**

默认值：#FFFFFF (白色)

说明：设置建筑物的外墙填充色

---

### **building-fill-opacity float**

默认值：1

说明：设置建筑物整体的透明度，包括建筑物所有的面。

---

### **building-height expression**

默认值：0

说明：设置建筑物的高度，以像素为单位。

---

## **调试模式下的属性**

### **debug-mode string**

默认值: collision

说明: 设置调试模式渲染

---

## 关于取值类型的说明

这里列出了CartoCSS中所有属性的取值类型及其说明。

### 颜色型 (Color)

CartoCSS可以使用一系列不同的方法表示颜色: HTML风格的16进制值, RGB值, RGBA值, HSL值或HSLA值都可以, 还可以使用HTML预定义颜色名, 像yellow、blue等。

```
#line {
    line-color: #ff0;
    line-color: #ffff00;
    line-color: rgb(255, 255, 0);
    line-color: rgba(255, 255, 0, 1);
    line-color: hsl(100, 50%, 50%);
    line-color: hsla(100, 50%, 50%, 1);
    line-color: yellow;
}
```

这里特别需要强调的是对HSL值的支持, 这其实是比RGB值更易用的颜色表达方式 (参见[这里](#))。CartoCSS中还支持几种颜色函数, 这是从LESS中借用的概念 (参见[这里](#)), 例子如下:

```
// 把色调亮或调暗
lighten(#ace, 10%);
darken(#ace, 10%);

// 饱和度调高或调低
saturate(#550000, 10%);
desaturate(#00ff00, 10%);

// 提高或降低颜色的透明度
fadein(#fafafa, 10%);
fadeout(#fefefe, 14%);

// 按照一定角度旋转色盘
spin(#ff00ff, 10%);

// 将两种颜色混合
mix(#fff, #000, 50%);
```

以上这些函数的参数可以是颜色值, 也可以是颜色名, 还可以是其它颜色函数。

## 浮点型 (Float)

浮点数是数值类型的时髦说法。在CartoCSS中，这指的就是一个数值，没有单位，但其实所有的单位都是像素。

```
#line {  
    line-width: 2;  
}
```

还可以对数值类型做简单运算：

```
#line {  
    line-width: 4 / 2; // 除  
    line-width: 4 + 2; // 加  
    line-width: 4 - 2; // 减  
    line-width: 4 * 2; // 乘  
    line-width: 4 % 2; // 取余  
}
```

## 统一资源描述符型 (URI)

URI是URL的一种时髦说法（译注：这实在不敢苟同，在http协议和REST架构中，URI和URL都有明确的定义，它们是不同的）。当一个属性的值类型是URI时，用户可以像在HTML中使用url('place.png')一样的表示方法。URL地址上的引号不是必需的，但最好加上。URI可以指向本地文件系统，也可以是互联网上资源的链接地址。

```
#markers {  
    marker-file: url('marker.png');  
}
```

## 字符串型 (String)

字符串也就是文本类型。在CartoCSS中，字符串应该有引号包围。字符串可以是任意文本，但在text-name和shield-name属性中，可以使用中括号包围的数据字段名来表示。例如：

```
#labels {  
    text-name: "[MY_FIELD]";  
}
```

## 布尔型 (Boolean)

布尔类型即是或否，取值为true或false。

```
#markers {  
    marker-allow-overlap:true;
```

```
}
```

## 表达式型 (Expressions)

表达式是一种语句，它可以将数据字段、数值以及其它类型灵活的组合起来。前面提到的 "[FIELD]" 形式包含了表达式。实际的表达式可以不用加引号就执行加、减、乘、除、连接等CartoCSS语法支持的操作。

```
#buildings {
  building-height: [HEIGHT_FIELD] * 10;
}
```

## 数列型 (Numbers)

数列型是逗号分隔的一组有序数值。数列类型在用于配置虚线样式时，其中的数字交替表示的是实线段长度、间隔长度和实线段长度。

```
#disputedboundary {
  line-dasharray: 1, 4, 2;
}
```

## 百分数型 (Percentages)

在CartoCSS中，百分号%表示值/100。它可用于表示比例的属性，例如透明度。

注意，百分数不能用于定义宽度、高度等属性。这一点与CSS不同，因为在CartoCSS中没有CSS中层次化的页面要素和页宽。它们在这里只是除以100以后的值。

```
#world {
  // 这种表达方式与...
  polygon-opacity: 50%;

  // ...这种方式效果一样
  polygon-opacity: 0.5;
}
```

## 函数型 (Functions)

这种类型可以包含一组逗号分隔的函数。例如，各种变换都是用functions作为值类型，而且这些函数还可以串接起来。

```
#point {
  point-transform: scale(2, 2);
}
```