

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

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
- 网站：luliu.me

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？
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。

2015.01.15. 注意：根据以下参考材料补充完善这部分内容

- <http://www.macwright.org/2012/11/02/css-for-maps.html>
- <http://www.developmentseed.org/blog/2011/feb/09/introducing-carto-css-map-styling-language/>
- <https://github.com/mapbox/carto/blob/master/README.md>

快速入门

译注：根据MapBox Studio文档中的QuickStart部分撰写

Mapbox Studio uses a language called CartoCSS to determine the look of a map. Colors, sizes, and shapes can all be manipulated by applying their specific CartoCSS parameters in the stylesheet panel to the right of the map. Read the CartoCSS manual for a more detailed introduction to the language.

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。

In this tutorial we'll create a custom style by writing CartoCSS for buildings, roads, and parks.

为了让读者快速的了解用CartoCSS能配出什么样的地图，如何配出这样的地图，本章以房屋、公园和道路的制图样式配置为例，简单展示一下CartoCSS的制图能力。

译注：以下例子为在Mapbox Studio中进行制图的实例，但并不局限于Mapbox Studio。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。

配置房屋样式（Styling buildings）

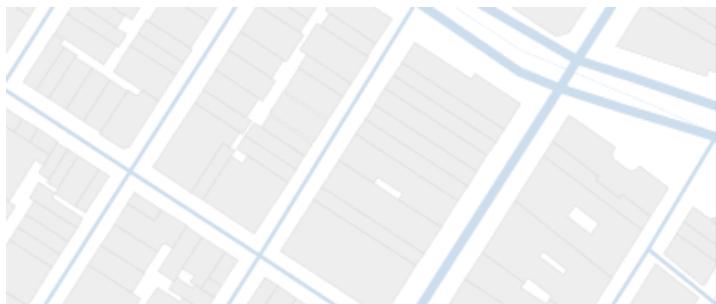
Add the following CartoCSS to your custom stylesheet and then click Save.

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: www.mapbox.com

- `#building` selects the building layer as the one that will be styled.
- `[zoom>=14]` restricts the styles to zoom level 14 or greater.
- `polygon-fill: #eee` fills the building polygons with a light grey color.
- `#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。

To add depth to our buildings at higher zoom levels let's add another set of rules that use the building symbolizer to render polygons as block-like shapes. Add the following CartoCSS to your custom stylesheet and then click Save.

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: www.mapbox.com

配置公园样式 (Styling parks)

Add the following CartoCSS to your custom stylesheet and then click Save.

这次我们试一下对公园这种类别的地块配置制图样式。

```
#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;  
}
```

通过使用以上CartoCSS脚本可以得到如下渲染效果:



图片来源: www.mapbox.com

下面逐一介绍其中用到的CartoCSS关键要素：

- `#landuse` selects features from the landuse layer.
- `[class='park']` restricts the landuse layer to features where the class attribute is park.
- `#poi_label` selects the poi_label layer for labelling parks.
- `[maki='park'][scalerank<=3][zoom>=15]` restricts the poi_label layer to prominent park labels and restricts their visibility to zoom level 15 or greater.
- `text-name: @name` sets the field that label contents will use for their text. It references the existing `@name` variable defined in the style tab.
- `text-face-name: @sans` sets the font to use for displaying labels. It references the existing `@sans` variable defined in the style tab.
- `text-wrap-width: 60` sets a maximum width for a single line of text.
- `text-halo-rasterizer: fast` uses an alternative optimized algorithm for drawing halos around text that improves rendering speed.
- `#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。

标注道路（Labelling roads）

Add the following CartoCSS to your custom stylesheet and then click Save.

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;

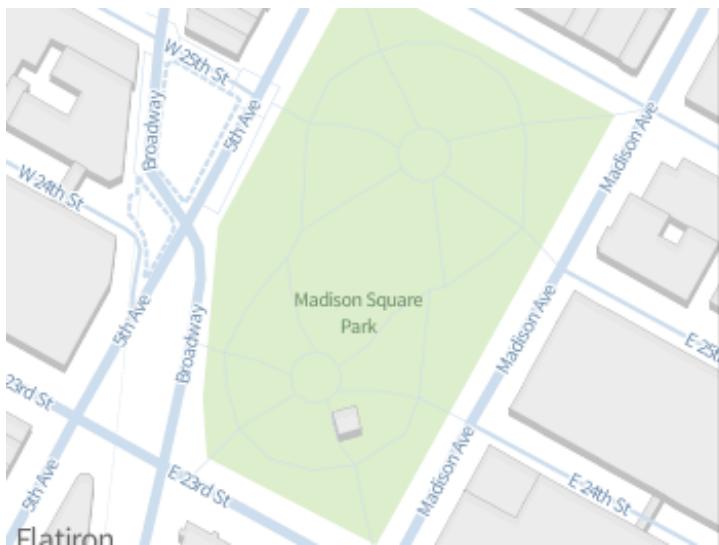
```

```

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: www.mapbox.com

其中涉及到的主要属性和参数的含义如下：

- `#road_label` selects features from the `road_label` layer.
- `[zoom>=13]` restricts the `road_label` layer to zoom level 13 or greater.
- `text-placement: line` sets labels to follow the orientation of lines rather than horizontally.
- `text-avoid-edges: true` forces labels to be placed away from tile edges to avoid being clipped.
- `#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，而不是规则的沿水平方向显示文字；
- `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。

基础用法

介绍CartoCSS中的基本概念与基础用法。

基本概念

译注：[原文地址](#)

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。

Mapnik中目前包含以下十种符号。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。

Multiple symbolizers can be applied to the same layer - some common combinations are line & polygons, point & text, line & markers, and line & line pattern.

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，以及线符号加线模式等。

A symbolizer is not present on the map unless it has a style defined, but once one of its style properties is added to the stylesheet default values will apply to the other properties for that symbolizer unless overridden. For example, the default line symbolizer color is black, so if you assign a line-width to a layer that line will be black unless you also assign a different color.

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，其它属性将全部被设置为默认值。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成为黑色。

多符号

A single layer is not limited to one of each symbolizer type. For example, multiple semi-transparent line symbolizers can be assigned to a polygon to achieve a soft glow or shadow effect. Multiple text symbolizers can be assigned to the same point with different offsets to label it with more than one field.

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标注在点要素的周围。

Normally when you assign a style to a layer, the style applies to a default symbolizer that is created. In the following example, the second rule overrides the first one because they both apply to the default symbolizer.

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;  
}
```

You can explicitly declare any number of new symbolizers for a layer that will be rendered in addition to styles they would otherwise conflict with. New symbolizers are defined using a double colon syntax inspired by pseudo-elements in CSS3:

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：

```
#layer {  
    /* styles for the default symbolizers */  
}  
  
#layer::newsymbol {  
    /* styles for a new symbolizer named 'newsymbol' */  
}
```

Note that newsymbol is not a special keyword but an arbitrary name chosen by the user. To help keep track of different symbolizers you can name additional symbolizers whatever makes sense for the situation. Some examples: #road::casing, #coastline::glow_inner, #building::shadow.

注意上面例子中的newsymbol不是关键字。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。

Returning to our previous example, declaring the second rule will add a blue glow on top of the red line instead of replacing it:

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](#)

Symbolizers are rendered in the order they are defined, so here the `::glow` (blue line) appears on top of the first style (red line).

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。

Named symbolizer styles can still be overridden by further styles that reference the same symbolizer name. In this example, the line color will be green, not green-on-yellow.

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](#)

需要补充以下部分 (来自[carto](#)项目的[README](#)) :

从属样式与实例 (Attachments and Instances)

In CSS, a certain object can only have one instance of a property. A <div> has a specific border width and color, rules that match better than others (#id instead of .class) override previous definitions. CartoCSS acts the same way normally for the sake of familiarity and organization, but Mapnik itself is more powerful.

Layers in Mapnik can have multiple [borders](#)(<http://trac.mapnik.org/wiki/LineSymbolizer>) and multiple copies of other attributes. This ability is useful in drawing line outlines, like in the case of road borders or 'glow' effects around coasts. CartoCSS makes this accessible by allowing attachments to styles:

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

Attachments are optional.

While attachments allow creating implicit “layers” with the same data, using **instances** allows you to create multiple symbolizers in the same style/layer:

```
#roads {  
    casing/line-width: 6;  
    casing/line-color: #333;  
    line-width: 4;  
    line-color: #666;  
}
```

This makes Mapnik first draw the line of color #333 with a width of 6, and then immediately afterwards, it draws the same line again with width 4 and color #666. Contrast that to attachments: Mapnik would first draw all casings before proceeding to the actual lines.

变量与表达式 (Variables & Expressions)

CartoCSS inherits from its basis in [less.js](http://lesscss.org/)(<http://lesscss.org/>) some new features in CSS. One can define variables in stylesheets, and use expressions to modify them.

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

嵌套样式 (Nested Styles)

CartoCSS also inherits nesting of rules from less.js.

```
/* Applies to all layers with .land class */  
.land {  
    line-color: #ccc;  
    line-width: 0.5;  
    polygon-fill: #eee;  
    /* Applies to #lakes.land */  
    #lakes {  
        polygon-fill: #000;  
    }  
}
```

This can be a convenient way to group style changes by zoom level:

```
[zoom > 1] {  
    /* Applies to all layers at zoom > 1 */  
    polygon-gamma: 0.3;  
  
    #world {  
        polygon-fill: #323;  
    }  
  
    #lakes {  
        polygon-fill: #144;  
    }  
}
```

字体 (FontSets)

By defining multiple fonts in a `text-face-name` definition, you create [FontSets](http://trac.mapnik.org/wiki/FontSet)(<http://trac.mapnik.org/wiki/FontSet>) in CartoCSS. These are useful for supporting multiple character sets and fallback fonts for distributed styles.

```
/* 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 (Filters)

CartoCSS supports a variety of filter styles:

Numeric comparisons:

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

General comparisons:

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

String comparisons:

```
/* a regular expression over name */
#world[name =~ "A.*"]
```

样式选择器 (Selectors)

译注：[原文地址](#)

CartoCSS styles are constructed by applying blocks of style rules to groups of objects. Style blocks are bounded by curly braces {} and contain style properties and values. Selectors are what allow you restrict these styles to specific layers or groups of objects within layers.

在CartoCSS中，地图样式通过一系列样式规则来表达。这些样式规则作用于地图上的各种要素对象，并且以模块化的形式进行组织。每一个样式块都由一对花括号{}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。（译注：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、文本和正则表达式三种类型。

图层标识 (By layer ID)

Select all of the objects from a single layer by the layer's ID. Separate multiple layer IDs with commas to select them for a single style.

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。对于多个图层使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 (By layer class)

You can also assign classes to layers to select multiple layers more simply. In Mapbox Studio (unlike TileMill) layer classes are only available for advanced usage.

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（译注：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.sample_class`和`layer_2.sample_class`两个新的标识，然后通过对`.sample_class`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 (Filter selectors)

You can modify selections with filters that reduce the number of objects a style applies to based on certain criteria. Filters let your style read into the various text and numeric properties attached to each object in a layer. For example, you might have all your roads in a single layer, but you could use filters to specify different line colors for different road classifications.

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（译注：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和分类的道路都包含在内，而利用过滤器，我们就可以为不同等级或类别的道路配置不同的样式。

Filters should be written inside square brackets after a layer selector or nested inside a larger style block.

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，或者还可以嵌套的写在一个更大的样式块中。

缩放级别过滤器 (Zoom level filters)

Restrict styles to certain zoom levels. This style will only apply when your map is zoomed all the way out to zoom level 0:

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。

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

You can specify ranges of zoom levels using two filters:

还可以定义缩放级别的范围：

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

Valid operators for zoom filters are = (equal to), > (greater than), < (less than), >= (greater than or equal to), <= (less than or equal to), != (not equal to).

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

You can nest filters to better organize your styles. For example, this style will draw red lines from zoom levels 4 through 10, but the lines will be thicker for zoom levels 8, 9, and 10.

过滤器可以以嵌套的方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级之间被绘制成红色，而在8级、9级和10级，线宽会更宽。

```
#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 (Numeric value comparison filters)

The same comparison operators available for the zoom filter can also be used for any numeric column in your data. For example, you might have a population field in a source full of city points. You could create a style that only labels cities with a population of more than 1 million.

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';  
}
```

You could also combine multiple numeric filters with zoom level filters to gradually bring in more populated cities as you zoom in.

而这种数值型过滤器可以和之前介绍的缩放级别过滤器结合，从而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';  
    }  
}
```

```
}
```

As with zoom levels, you can select data based on numeric ranges.

与缩放级别过滤器相同，数值型过滤器中也同样支持数值范围过滤：

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

文本型过滤器 (Text comparison filters)

You can also filter on columns that contain text. Filter on exact matches with the equals operator (=) or get the inverse results with the not-equal operator (!=). Unlike zoom and numeric values, text values must be quoted with either double or single quotes.

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（译注：这里隐去了Mapbox相关内容），在一个表示道路网的线要素图层中包含了一个名为class的文本类型属性字段，该字段取值为"motorway"，"main"或者"street"，用于表示每条道路的类型。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;
  }
}
```

To select everything that is not a motorway you could use the != ("not equal") operator in the filter:

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操作符：

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

正则表达式过滤器 (Regular expression filters)

Note: This is an advanced feature that may have negative performance implications.

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。

You can match text in filters based on a pattern using the regular expression operator (`=~`). This filter will match any text starting with ‘motorway’ (ie, both ‘motorway’ and ‘motorway_link’).

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motoway开头的要素记录，像motoway、motoway_link都会被匹配上。

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

The `.` represents ‘any character’, and the `*` means ‘any number of occurrences of the preceding expression’. So `.*` used in combination means ‘any number of any characters’.

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。

配置线样式 (Styling lines)

译注：[原文地址](#)

Line styles can be applied to both line and polygon layers. The simplest line styles have just a line-width (in pixels) and a line-color making a single solid line. The default values for these properties are 1 and black respectively if they are not specified.

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](#)

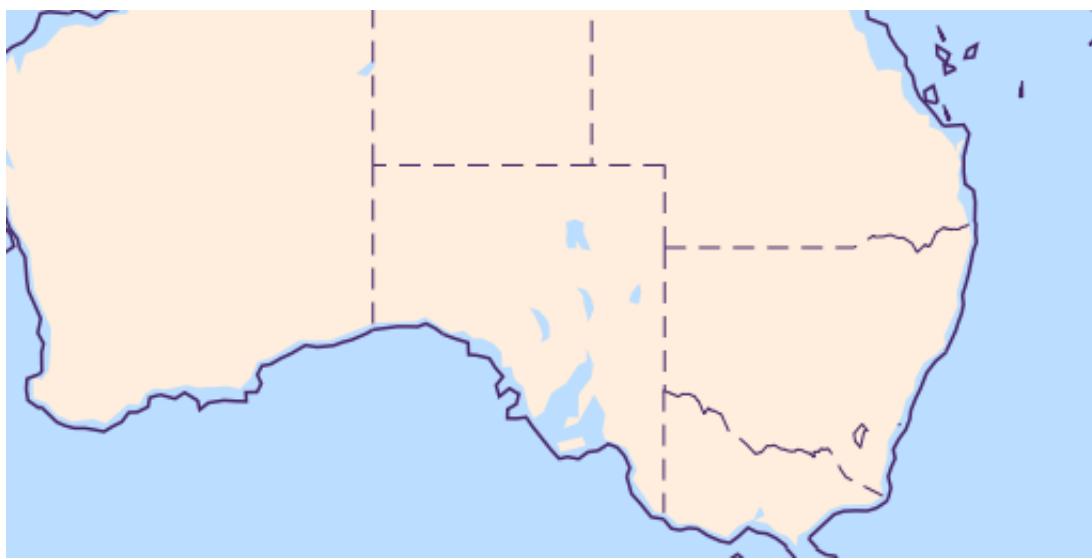
```
#admin[admin_level=2] {  
    line-width: 0.75;
```

```
    line-color: #426;  
}
```

虚线 (Dashed lines)

Simple dashed lines can be created with the `line-dasharray` property. The value of this property is a comma-separated list of pixel widths that will alternatively be applied to dashes and spaces. This style draws a line with 5 pixel dashes separated by 3 pixel spaces:

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，以像素为单位。下面例子中的虚线就是以5个像素的短线和3个像素的间隔进行绘制的。

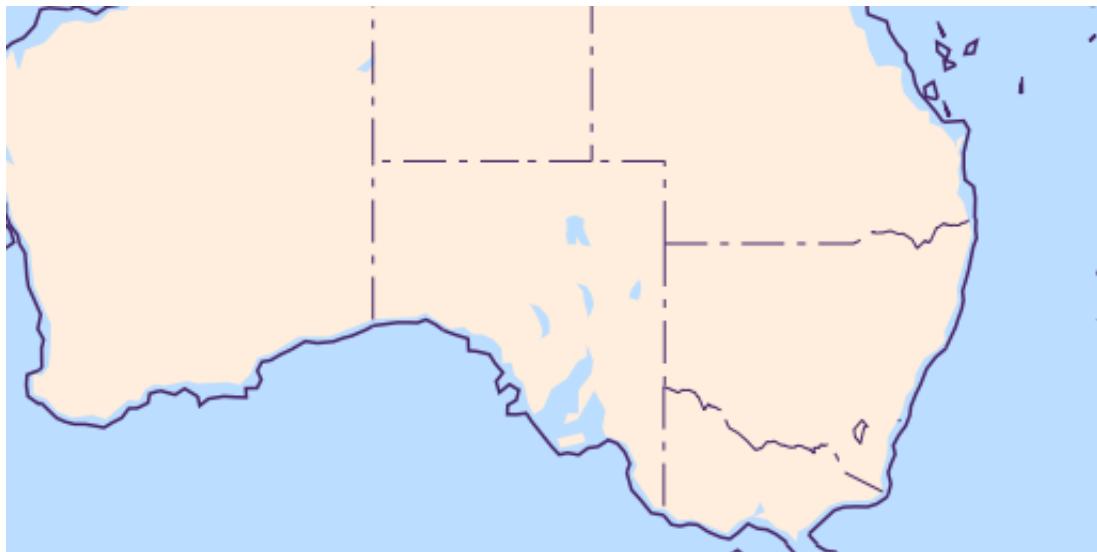


图片来源: [Mapbox](#)

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

You can make your dash patterns as complex as you want, with the limitation that the dasharray values must all be whole numbers.

虚线的样式还可以被配置成更复杂的模式，只要`line-dasharray`属性值列表中的元素都是整数就行。



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 (Caps & Joins)

With thicker line widths you'll notice long points at sharp angles and odd gaps on small polygons.

当把线宽设置成较大的值时，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;  
}
```

You can adjust the angles with the `line-join` property: `round` or `square` them off (the default is called `miter`). The gaps can be filled by setting `line-cap` to `round` or `square` (the default is called `butt`).

要解决这个问题，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;  
}
```

For dashed lines, `line-caps` are applied to each dash and their additional length is not included in the `dasharray` definition. Notice how the following style creates almost-solid lines despite the `dasharray` defining a gap of 4 pixels.

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](#)

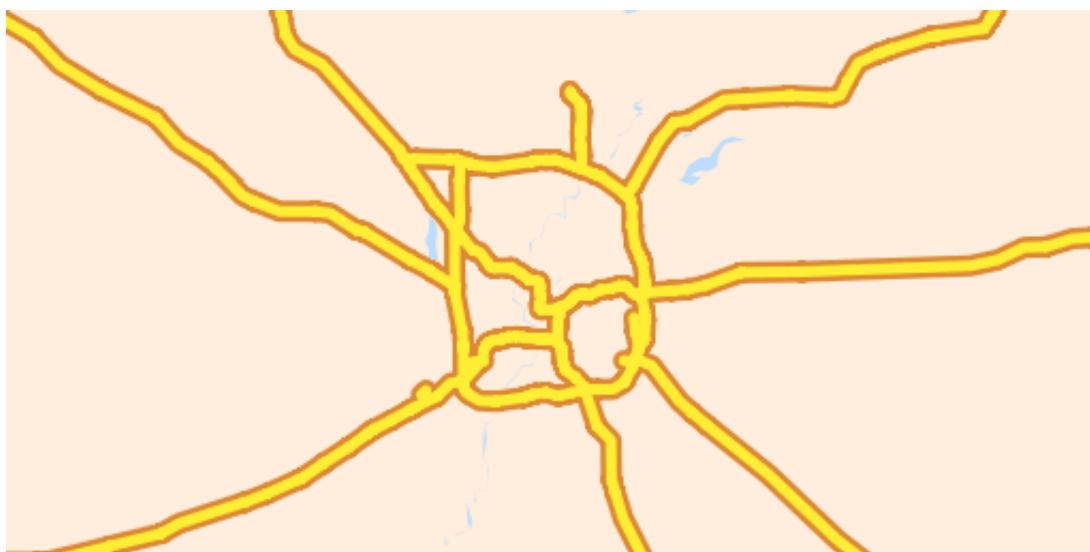
```
#layer {
  line-width: 4;
  line-cap: round;
  line-dasharray: 10, 4;
}
```

复合线样式 (Compound line styles)

道路 (Roads)

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），可以用多层相互压盖的方式来定义样式。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（译注：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。二者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;
  }
}
```

Dealing with multiple road classes, things get a little more complicated. You can either group your styles by class or group them by attachment. Here we've grouped by class (filtering on the `class` field).

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（译注：attachment在这里是指通过叠加覆盖方式组合而成的某类道路样式的各个从属样式，比如例子中的`::case`和`::fill`）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 (Railroads)

A common way of symbolizing railroad lines is with regular hatches on a thin line. This can be done with two line attachments - one thin and solid, the other thick and dashed. The dash should be short with wide spacing.

一种典型的铁路线样式是在一条细实线上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，还有一条短而粗的虚线。

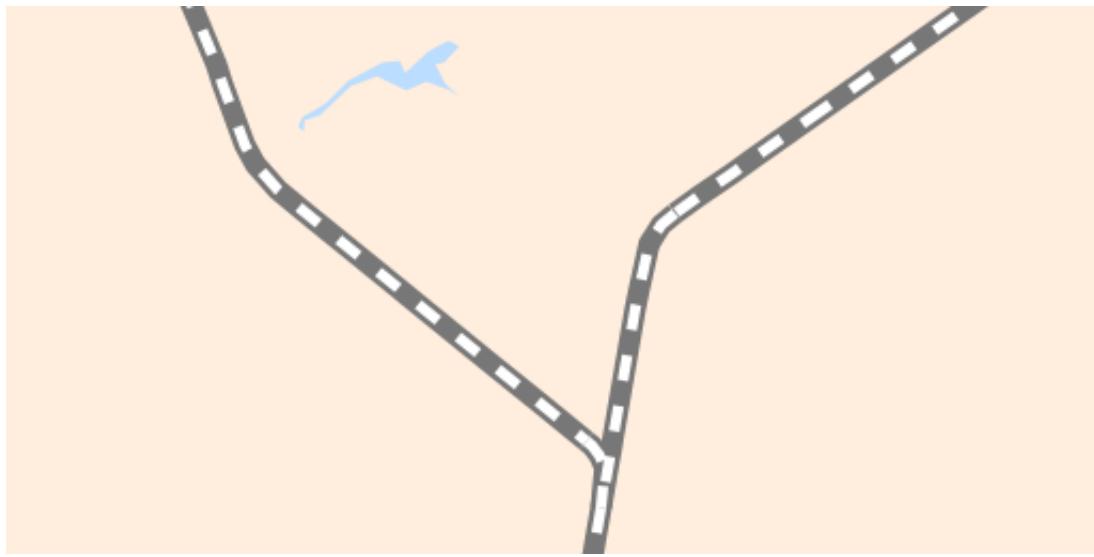


图片来源: [Mapbox](#)

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

Another common railroad line style is similar, but with a thin dash and a thick outline. Make sure you define the ::dash after the ::line so that it appears on top correctly.

另一种典型的铁路线样式与第一种类似，只是虚线部分中的短线更细更长，而实线部分更粗。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 (Tunnels)

A simple tunnel style can be created by modifying a regular road style and making the background line dashed.

一种简单的隧道样式可以通过把道路样式稍稍修改而成，具体的，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;  
    }  
}
```

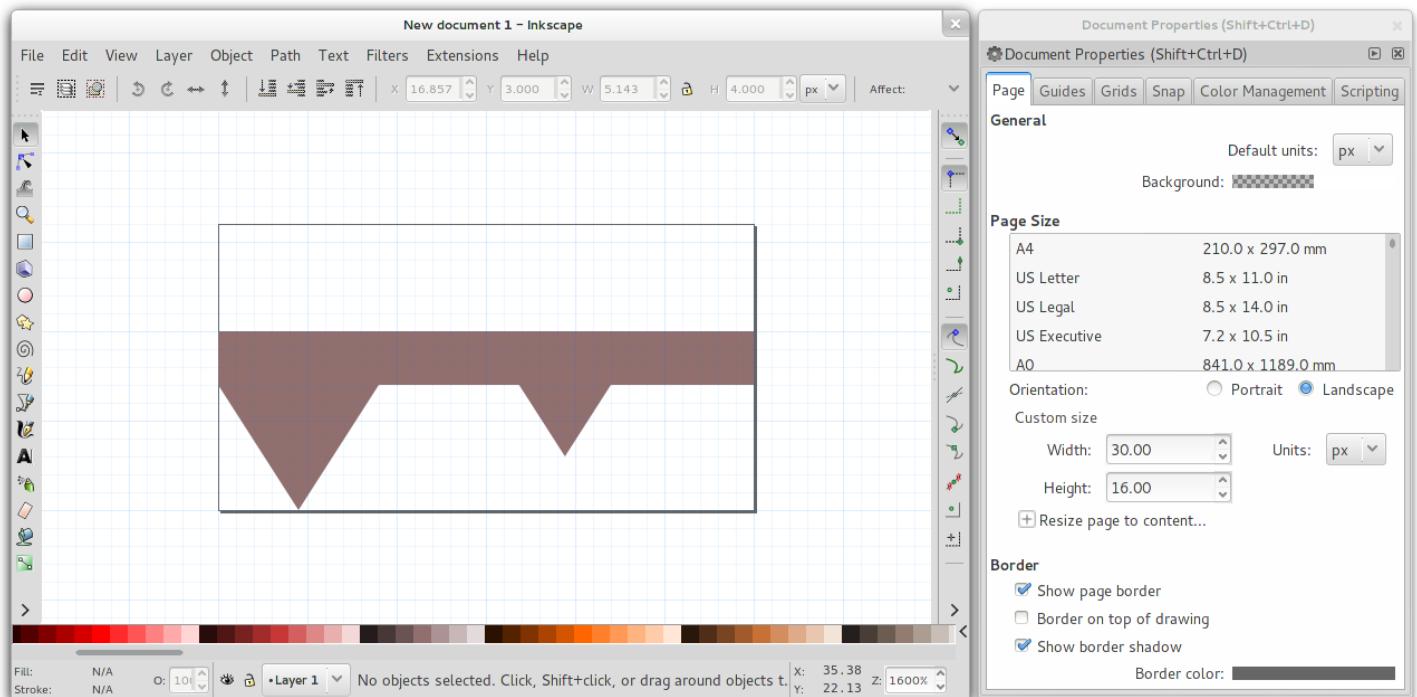
用图片实现线模式 (Line patterns with Images)

Certain types of line patterns are too complex to be easily achieved with regular compound line styles. TileMill allows you to use repeated images alongside or in place of your other line styles. As an example we'll make a pattern that we'll use to represent a cliff. To do this you'll need to work with external graphics software - we'll be using Inkscape in this example.

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（译注：这里用CartoCSS而非原文中的TileMill）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（译注：对这种“元图片”，这里称为线型图案）。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模式。

In Inkscape (or whatever you are using), create a new document. The size should be rather small - the height of the image will be the width of the line pattern and the width of the image will be repeated along the length of the line. Our example is 30x16 pixels.

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。



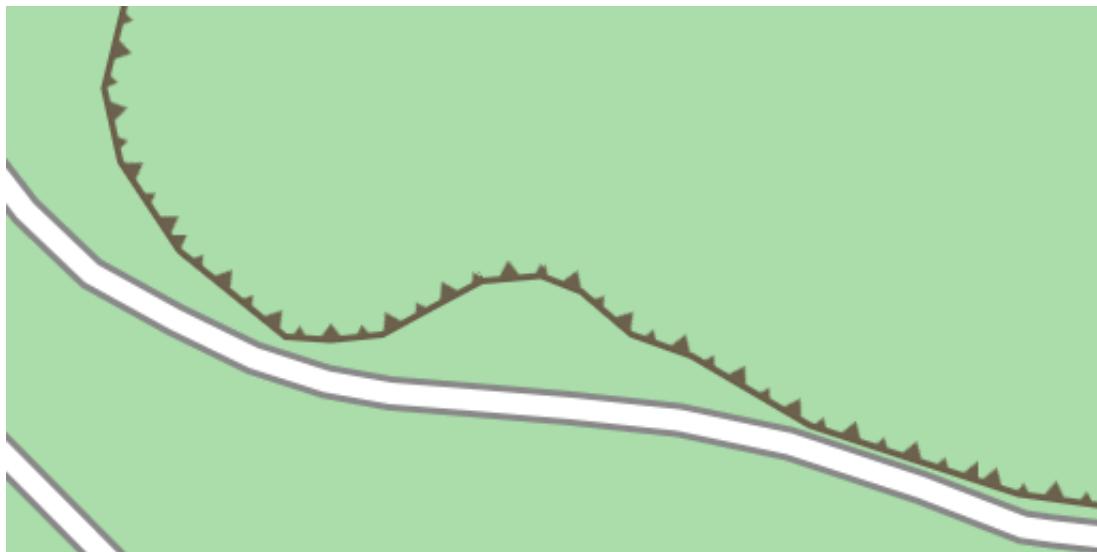
图片来源：[Mapbox](#)

Note how the centerline of the pattern is centered on the image (with empty space at the top) for correct alignment with the line data.

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。

To use the image from Inkscape, export it as a PNG file. Line patterns just need a single CartoCSS style to be added to your TileMill project:

将这个图片从Inkscape中导出成PNG文件。然后如果要在CartoCSS中使用它的话，只需增加一个包含`line-pattern-file`属性的样式块。



图片来源: [Mapbox](#)

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

For some types of patterns, such as the cliff in this example, the direction of the pattern is important. The bottom of line pattern images will be on the right side of lines. The left side of the image will be at the beginning of the line.

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的右侧；而图案图片的左侧会被置于线要素的起始位置。

配置面样式 (Styling polygons)

译注：[原文地址](#)

Polygons are areas that can be filled with a solid color or a pattern, and also given an outline.

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。

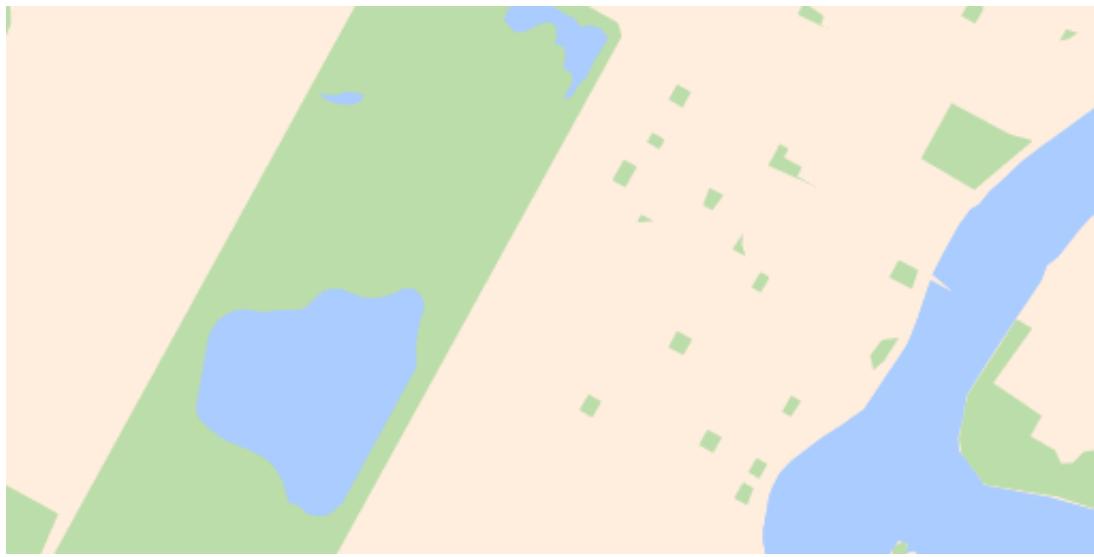
Tip: everything covered in the styling lines guide can also be applied to polygon layers.

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。

Basic Styling

The simplest polygon style is a solid color fill.

最简单的面样式是纯色填充。



图片来源: [Mapbox](#)

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

If you want to adjust the opacity of a polygon-fill you can use the polygon-opacity property. This is a number between 0 and 1 - 0 being fully transparent and 1 being fully opaque. With 50% opacity we can see overlapping shapes in the same layer add together to create more opaque areas.

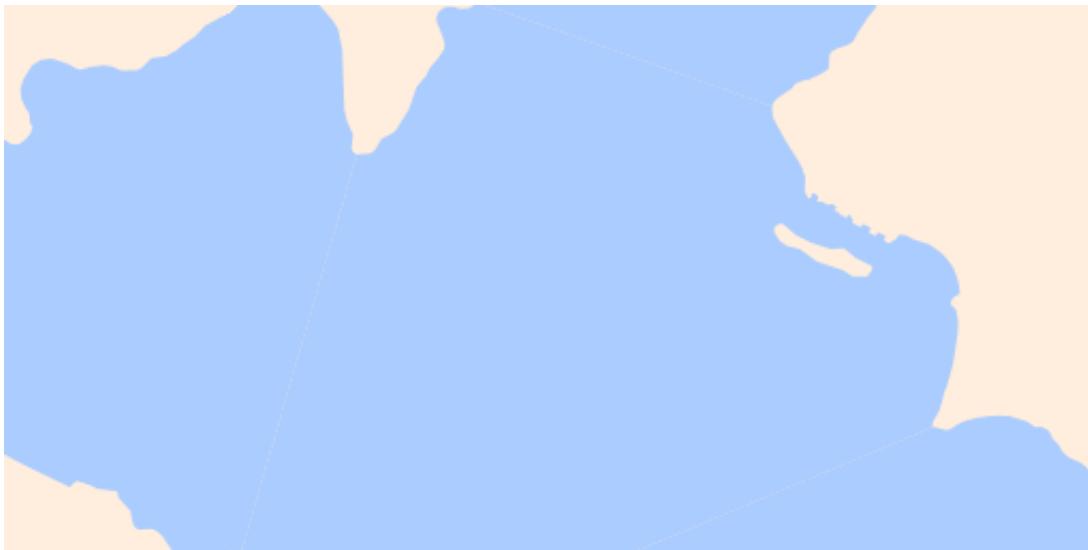
如果需要调整填充色的透明度，那么就用polygon-opacity属性。它的取值范围是从0到1，0表示全透明，1表示完全不透明。如果把它设置成50%透明，那么可以得到这样一种效果（如下图）：如果当前图层中有相互重叠的面要素，那么重叠的部分的透明度会显得比其它部分更低一些。



图片来源: [Mapbox](#)

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

用伽马属性调整缝隙（Gaps and Gamma）



图片来源: [Mapbox](#)

When you have a layer containing polygons that should fit together seamlessly, you might notice subtle gaps between them at certain scales. You can use the polygon-gamma style to help reduce this effect. Gamma takes a value between 0 and 1 - the default is 1, so try lowering it to hide the gaps. Be careful about setting it too low, though. You'll get jagged edges and possibly even stranger artifacts.

如果一个面图层中的所有面要素之间（译注：指在原始数据层次，各个面要素对应的多边形几何形状）都是无缝衔接的，那么在制图渲染后它们之间也理应完好的拼接在一起。然而在实际的制图效果中，却往往会在地图缩放到某些级别上看到这些原本无缝的多边形之间出现一些细微的缝隙（译注：如上图中水域部分靠图片上方的那一条灰白色细缝）。这种缝隙效果可以用一个名为的属性值来帮助消除。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。

（译注：是否有必要对为何会产生这种细缝，以及为什么会通过调整Gamma值消除这些细缝的原因和机理进行一下解释？如果有必要，那么放在基础用法这一章中是否合适？是否应该放到高级技巧中去？）



图片来源: [Mapbox](#)

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

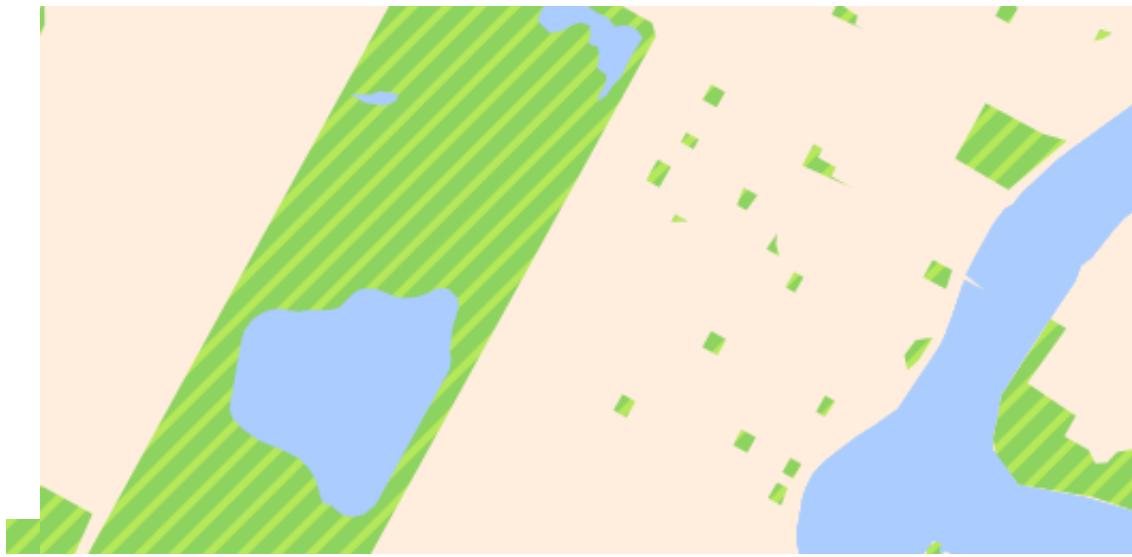
图案和纹理 (Patterns and Textures)

With CartoCSS, you can easily fill areas with textures and patterns by bringing in external images. You might create the patterns yourself in image editing software, or find ready-made images from resource websites such as [Subtle Patterns](#) or [Free Seamless Textures](#).

在CartoCSS中，使用外部图片（译注：主要指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");  
}
```

In order to have patterns work when your style is uploaded to Mapbox they will need to be stored in the style project folder (this is the folder ending in .tm2 that's created when you "Save As" a style project). If your project uses a lot of images you can create a subdirectory in the .tm2 folder and include that in the URL. For example, if you create a subdirectory named images:

(译注：这段文字为了去Mapbox化，完全使用了意译。)

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");  
}
```

Background patterns

If you want to add a pattern image to the background of the whole map, you can use the background-image property on the Map object.

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");  
}
```

Like all other properties on the Map object, background-image has a global effect - it cannot be filtered or changed depending on zoom level.

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（Combining patterns & fills）

Using transparency or compositing operations it is possible to get a lot of variety out of a single pattern image.

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（Ensuring seamlessness）

There are two types of pattern alignment: local (the default) and global (specified with polygon-pattern-alignment: global;).

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。

When a pattern style has local alignment, that means that the pattern will be aligned separately for each polygon it is applied to. The top-left corner of the pattern image will be aligned to the top-left corner of a polygon's bounding box.

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。

When a pattern style has global alignment, pattern images are aligned to the image tile instead of the individual geometries. Thus a repeated pattern will line up across all of the polygons it is applied to. With global alignment, pattern images should not be larger than the metatile (excluding the buffer), otherwise portions of the pattern will never be shown.

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。

Another important thing to keep in mind is with globally-aligned patterns is that the pixel dimensions of the image file must multiply evenly up to the width and height of the tile, 256×256 pixels. Your pattern width or height dimensions could be 16 or 32 or 128, but should not 20 or 100 or any other number you can't evenly divide 256 by. If you are using patterns from a resource website, you may need to resize them in an image editor to conform to this limitation.

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。

配置标注样式（Styling labels）

译注：[原文地址](#)

简单点标注 (Basic Point Labels)

In CartoCSS, labelling is handled by a variety of properties beginning with `text-`. For each text-related style there are two required properties: `text-name`, which specifies what text goes in the labels, and `text-face-name`, which specifies the typeface(s) will be used to draw the label. (You can see which typefaces are available in the font browser - click the 'A' icon on the left button bar.)

在CartoCSS中，文本标注的样式由一系列以`text-`开头的属性进行配置。在配置标注的属性中，有两个必填属性：`text-name`和`text-face-name`。前者用于指定显示在标注中的文本内容；后者用于指定显示文本所用的字体。

The `text-name` property can pull text from your layer's data fields. If your layer contains a column called `name_en`, a simple label style would look like this:

`text-name`属性可以从图层的属性数据字段中提取文本内容。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';  
}
```

The color and size of these labels will be the defaults - black and 10 pixels respectively. These can be adjusted with the `text-fill` and `text-size` properties.

上面例子中标注的颜色和尺寸都是默认值：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;  
}
```

To separate your text from the background, it is often useful to add an outline or halo around the text. You can control the color with `text-halo-fill` and the width of the halo (in pixels) is controlled with `text-halo-radius`. In the example below, we are using the `fadeout` color function to make the white halo 30% transparent.

为了能让文本标注在地图背景中更为醒目，通常会为文本文字增加边线或光晕。光晕的颜色用`text-halo-fill`属性进行配置，宽度用`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 (Text Along Lines)

You can also use CartoCSS to style labels that follow a line such as a road or a river. To do this we need to adjust the `text-placement` property. Its default is `point`; we'll change it to `line`. We've also added a simple style to visualize the line itself.

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。控制这种标注效果的属性是`text-placement`。它的默认值是`point`，如果要实现沿线标注，那么需要将其值设为`line`。在下面的例子中，除了标注以外，河流线本身也配置了简单的样式。



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;  
}
```

For rivers it is nice to have the label offset parallel to the line of the river. This can be easily done with the `text-dy` property to specify how large (in pixels) this offset should be. (`dy` refers to a displacement along the **y** axis.)

对于河流来说，让它的标注相对于河流线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。

We'll also adjust the `text-max-char-angle-delta` property. This allows us to specify the maximum line angle (in degrees) that the text should try to wrap around. The default is 22.5°; setting it lower will make the labels appear along straighter parts of the line.

还有一个`text-max-char-angle-delta`属性，它用来调整标注文本的最大弯折角度，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 (Adding custom text)

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，它还可以由多个属性字段组合而成，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;  
}
```

Other potential uses:

其它一些应用实例：

- Multilingual labels: [name] + ' (' + [name_en] + ')'
- Administrative units: [city] + ', ' + [province]
- Numeric units: [elevation] + 'm'
- Clever [unicode icons](#): '⚑ ' + [embassy_name] or '⚓ ' + [harbour_name]
- 多语言标注: [name] + ' (' + [name_en] + ')'
- 行政区划单位: [city] + ', ' + [province]
- 计量单位: [elevation] + 'm'
- 特殊的[unicode字符](#): '⚑ ' + [embassy_name] or '⚓ ' + [harbour_name]

You can also assign arbitrary text to labels that does not come from a data field. Due to a backwards-compatibility issue, you will need to quote such text twice for this to work correctly.

标注的内容还可以不从属性数据的字段中提取，而是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';  
}
```

```
}
```

If you need to include quotation marks in your custom quoted text, you will need to escape them with a backslash. For example, for the custom text **City's "Best" Coffee**:

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**，那么需要这样定义：

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

多行标注（Multi-line labels）

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位置处的第一个字，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;  
}
```

Note that text wrapping not yet supported with `text-placement: line`.

不过，需要注意多行文本不支持沿线标注的情况。

You may have a specific point where you want the line breaks to happen. You can use the code `n` to indicate a new line.

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，但译者认为这部分值得保留。)

You can use the `text-wrap-character` to cause labels to wrap on a character other than a space. With a properly constructed dataset this can give you better control over your labels.

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。

For example we could alter our compound label example to separate the two fields only with an underscore. Setting the wrap character to `_` (and also setting a very low wrap width to force wrapping) ensures that the two fields will always be written on their own lines.

例如我们可以将折行字符设置成下划线`_`，然后再把折行宽度`text-wrap-width`设得很小，这样就能够保证以`_`分隔的两个字段总是能分成两行显示。



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: '_';  
}
```

独立标注层（Layering Labels）

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如标注相互压盖等异常渲染效果。

For simple stylesheets you can control this by making sure your geometry styles and your text styles are in separate attachments:

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';
  }
}
```

However in many cases you'll need to create a label layer that is separate from the layer you use for line and polygon styling. As an example of this, you can look at the Open Streets DC project that comes with TileMill.

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。

The layers roads and roads-label reference the same data, but are separated for correct ordering. For more details on how object stacking works in TileMill, see the Symbol Drawing Order guide.

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。

符号的渲染顺序（Symbol drawing order）

译注：[原文地址](#)

Objects in Mapbox Studio are drawn using a [Painter's Algorithm](#), meaning everything is drawn in a specific order, and things that are drawn first might be covered by things that are drawn later.

通过CartoCSS配置好样式的地图将按照[Painter's算法](#)来绘制图中的所有对象。这些对象的渲染是按照特定顺序进行的，先绘制的对象可能会被后绘制的对象覆盖。

概述（Overview）

The order in which objects are drawn depends on the following conditions. See the sections that follow for more details.

地图要素依照以下条件所确定的顺序进行绘制。后面的小节中包含更多详细的解释。

1. Layers: “Higher” layers obscure “lower” ones.
2. Style attachments (eg, `::glow { ... }`) within a Stylesheet are applied from top to bottom.
3. Objects within an attachment are drawn in the order in which they are stored in the vector tile.
4. Multiple property instances on the same object (eg `a/line-color: blue; b/line-color: red;`) are drawn in the order they are defined.
5. 图层：位于更高层的图层会覆盖低层图层；
6. 同一样式块中的各个从属样式块（例如`::glow { ... }`）按照从上到下的顺序起作用；
7. 每个子符号所作用的要素对象按照它们在矢量瓦片中的存储顺序进行绘制；
8. 作用于同一对象上的多个样式属性按照属性定义的顺序起作用。

顺序vs.优先级 (Order vs. Priority)

For things like lines and areas, objects that are drawn first are less likely to be fully visible. Objects high in the stack might completely obscure other objects, thus you might associate these with a high ‘priority’ or ‘importance’.

对于线要素和面要素对象来说，越是先被绘制出来的，越有可能被后绘制的对象盖住，导致在最终的地图上只能看到这些对象的一部分甚至完全看不到。因为在绘制堆栈中的高位对象（译注：就是指那些按照顺序后绘制的对象）可能会在绘制时把其它对象完全覆盖，所以这些高位对象可以说天然的具有较高的“优先级”或“重要性”。

However for things like text, markers, and icons that have their `allow-overlap` properties set to false (the default) things work a bit differently. Objects that are drawn first are **more** likely to be visible; instead of letting things sit on top of each other, overlapping objects are simply skipped. Since such objects higher in the stack are less likely to be drawn, you might associate these with a low ‘priority’ or ‘importance’.

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。对于使用这些符号定义样式的对象，越是先被绘制出来的，越有可能在最后的地图上可见，因为如果后绘制的对象会对前面已经绘制的对象造成压盖，那么这些对象根本就不会被画出来，而是直接被忽略了。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，因而具有更低的“优先级”或“重要性”。

（译注：用一句简单的话说，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。这也是本节讨论的所谓顺序与优先级的关系问题。）

图层排序 (Layer Ordering)

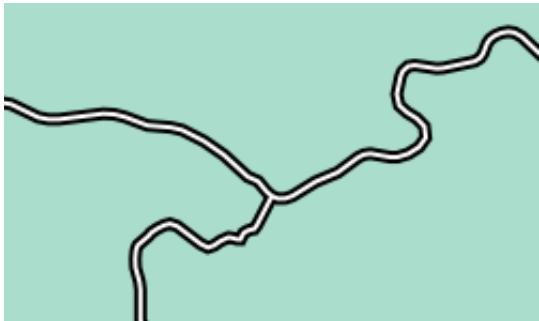
Layers are rendered in order starting at the bottom of the layers list moving up. If you look at the layers in the Mapbox Streets vector tile source you can see that the basic parts of the map (eg. landuse areas, water) are in layers at the bottom of the list. The things that shouldn’t be covered up by anything else (eg. labels, icons) are in layers at the top of the list.

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 (Attachment Ordering)

Within a layer, styles can be broken up into ‘attachments’ with the `::` syntax. Think of attachments like sub-layers.

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”。子符号可以被理解为子图层。



图片来源: [Mapbox](#)

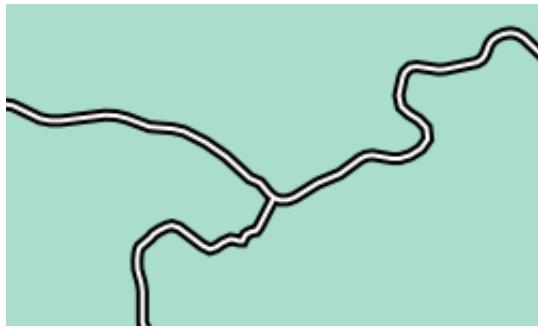
```
#layer {  
  ::outline {  
    line-width: 6;  
    line-color: black;  
  }  
  ::inline {  
    line-width: 2;  
    line-color: white;  
  }  
}
```

Attachments are drawn in the order they are first defined, so in the example above the `::outline` lines will be drawn below the `::inline` lines.

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

Note that all styles are nested inside attachments. If you don't explicitly define one, a default attachment still exists. Thus the following style produces the same result as the one above.

需要注意的是，实际上所有样式都是被定义在从属样式块之中的。如果不显式定义一个子符号，那么CartoCSS还是会为一系列样式属性生成一个默认的从属样式块。所以下面的样式代码能够产生与前面完全一样的渲染结果。（译注：注意第一句话中的nested不是“嵌套”的意思，而是单纯的“被包裹在...里面”的含义。具体而言，在下面的例子中，`line-width: 2;` `line-color: white;`两句话实际上也在一个从属样式块中，它是一个隐式定义的默认从属样式块，只是名字不叫`::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 (Data Ordering)

Within each attachment, the order that your data is stored/retrieved in is also significant. The ordering of objects in the Mapbox Streets vector tiles have been optimized for the most common rendering requirements.

在处理每个从属样式块时，其涉及到的每条要素记录的存储和检索后的顺序对于渲染结果也是很有影响的。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记录的存储顺序进行优化。

If you are creating a custom vector tile source this is something you will have to consider. When styling city labels, for example, it's good to ensure that the order of your data makes sense for label prioritization. For data coming from an SQL database you should ORDER BY a population column or some other prioritization field in the select statement.

如果用户要创建一个自定义的矢量数据源，那么就必须要考虑数据记录的顺序问题。就拿城市名称标注来说，让用于标注的数据按照某种优先级保持一个良好的顺序是很有好处的。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把它们按照某个特定的列或优先级规则排个序。

Data coming from files are read from the beginning of the file to the end and cannot be re-ordered on-the-fly by TileMill. You'll want to pre-process such files to make sure the ordering makes sense.

如果是从文件中读取数据，那么顺序通常只能是从文件的开头读到结尾，无法实时的调整数据记录的顺序。这种时候可以通过预处理来把数据记录的顺序调整得更合理。

You can do this from the terminal with ogr2ogr. This example rearranges all the objects in cities.shp based on the population field in descending order (highest population first).

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
```

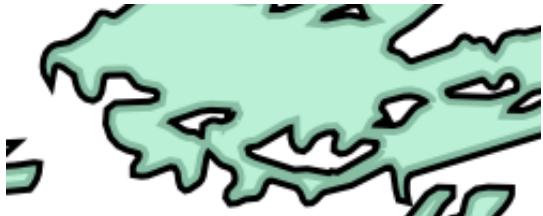
符号排序 (Symbolizer Ordering)

Each object in each attachment may have multiple *symbolizers* applied to it. That is, a polygon might have both a fill and an outline. In this case, the styles are drawn in the same order they are defined.

在每个从属样式块中，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，它既有用面符号修饰的填充样式，又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。

In this style, the outline will be drawn below the fill:

下面的例子中的绘制顺序是先画边界线，再在它上面画填充。



图片来源: [Mapbox](#)

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

In this style, the line is drawn on top of the fill:

而在下面的例子中则相反，是先画填充，再在它上面画边界线。

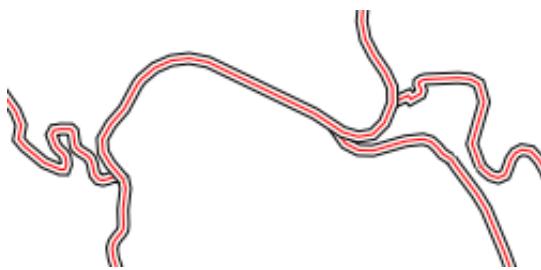


图片来源: [Mapbox](#)

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

It's also possible to create multiple symbols of the same type within an attachment using named *instances*. Like attachments, their names are arbitrary.

此外，还可以在一个从属样式块中利用具名_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。（译注：一个“多重样式”的翻译就已经够让人蛋疼了，这又来了个“实例”，我去。。。）



图片来源: [Mapbox](#)

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

Note that symbolizer ordering happens after all other types of ordering - so an outline might be on top of one polygon but beneath a neighboring polygon. If you want to ensure lines are always below fills, use separate attachments.

需要注意的是，符号排序发生在所有类型排序的最后，所以可能会出现一个多边形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（译注：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在填充样式之下，就需要使用独立的从属样式块来实现。

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。

关于合成操作（Compositing）

译注：[原文地址](#)

Compositing operations affect the way colors and textures of different elements and styles interact with each other.

合成操作会改变不同要素与样式的颜色、纹理之间相互作用的方式。

Without any compositing operations on a source it will just be painted directly over the destination – compositing operations allow us to change this. There are 33 compositing operations available in CartoCSS:

如果不在`_源_`上应用合成操作，那么它就会被直接覆盖绘制在`_目标_`之上。而合成操作允许我们改变这种绘制行为。CartoCSS一共支持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-rgb	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

The operations in the first two columns are color blending modes that provide a variety of ways to control the blending of the colors of objects and layers with each other. The operations in the last column are [Duff-Porter alpha blending modes](#). They provide a variety of ways to fill and mask objects and layers with each other.

左边两列是色彩混合模式，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。而最右边一列中的[Duff-Porter透明度混合模式](#)则提供的一系列关于填充和遮盖的不同方式。

If you are familiar with image editors such as the GIMP or PhotoShop you will recognize many of these as layer blending modes. They work much the same way in Mapbox Studio, but do not (necessarily) operate on the layer as a whole. There are two ways to invoke a composite operation - on an entire style attachment via the `comp-op` property, or on a particular symbolizer via a symbolizer-specific property:

如果你对图像编辑软件（比如GIMP、PhotoShop）比较熟悉，那么应该能感觉到上面这些模式很多都和图层混合模式很像。它们的确很相似，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`属性作用于一个从属样式块；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

There are times when you'll want to use the style-wide comp-op and times when you'll want to use the symbolizer-specific properties. It will depend on the results you want to achieve. With the symbolizer-specific approach, overlapping objects in the style will have their compositing operations applied to each other as well as the layers below. With the style-wide approach, the style will be rendered and flattened first.

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：如果采用方式一，那么地图会先按照所定义的样式进行渲染和平坦化（译注：不太确定flattened是个什么处理，达到了什么效果？）；而如果是方式二，那么不仅是样式中那些有重叠部分的对象之间，而且连当前图层下面的图层都会被应用合成操作。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;
}
```

Note: When we talk about the effects of composite operations, we need to talk about a *source* and a *destination*. The *source* is the style or symbolizer that the `comp-op` property is applied to, and the *destination* is the rest of the image that is drawn below that. There may also be more parts to the image that appear above the *source*; these are not affected by the `comp-op` and are drawn normally.

_注意：_我们在说合成操作的时候，需要明确_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。

色彩混合（Color Blending）

There are 22 color-blending compositing operations. This section will describe the ones that are most useful for cartographic design in Mapbox Studio. To illustrate the differences between them all, we'll show how each of them affect a few example layers and backgrounds.

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进行简单介绍。为了更明显的表现出不同模式之间的差异，我们将这些操作应用在两个示例图层和两幅背景图上。

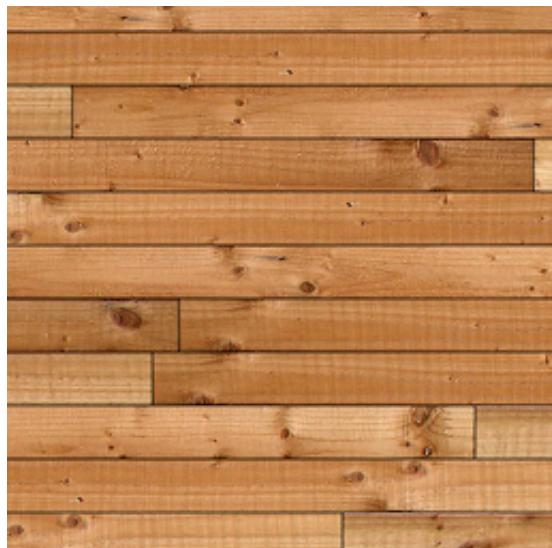
These are the layers the `comp-op` properties will be applied to:

下面是两个示例图层：



These are the backgrounds the comp-op layers will be overlaid on:

下面是两幅被comp-op图层叠加的背景图：



Here is what the result looks like with no comp-op property applied:

下面是在不定义comp-op属性时的渲染效果：



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`模式采用源图像中的颜色，并用这些颜色去夸大目标图像中的明暗度（译注：十分不确定这句话的意思）。`overlay`模式是几种可良好用于表现纹理的合成操作之一，可用于地形数据图层的样式配置。

Multiply



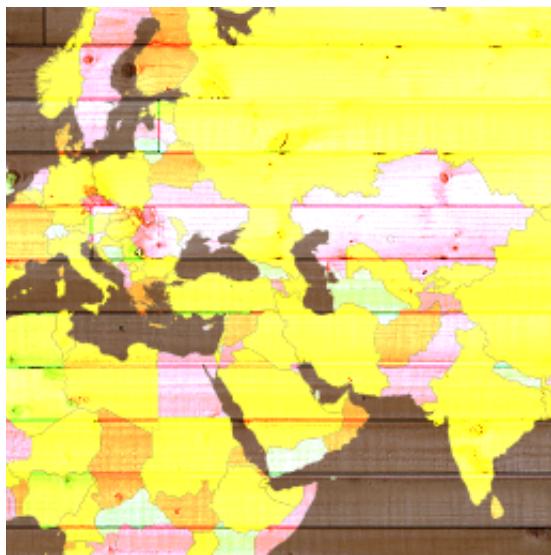
The `multiply` comp-op multiplies the color of the source and destination, usually resulting in a darkened image tinted to the color of the source. If either the source or destination is solid white, the other will appear unchanged. If either the source or destination is solid black, the result will also be solid black.

`multiply`合成操作将源与目标的颜色进行相乘运算。这种操作得到图像的颜色通常是基于源图像颜色的暗化效果。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。

One of the many uses for multiply is to simulate the way ink colors would blend with each other or with a textured surface. It can also be used for other kinds of texture effects.

`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



The `color-dodge` comp op brightens the colors of the destination based on the source. The lighter the source, the more intense the effect. You'll get nicer results when using this on dark to mid-tone colors, otherwise the colors can become too intense.

`color-dodge`操作以源图像为基础将目标的颜色亮化。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，需要在颜色较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

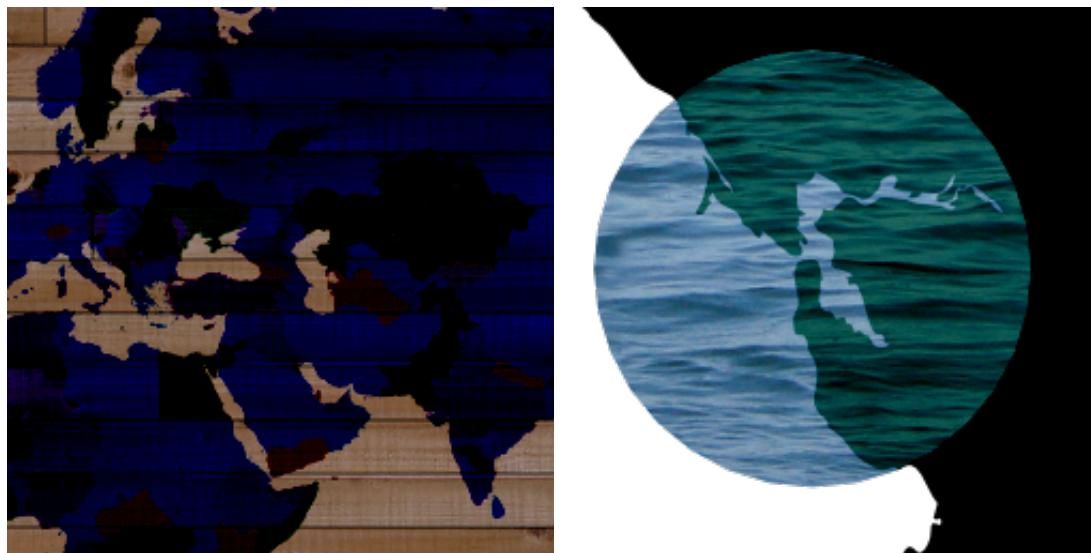


The `plus` comp-op adds the color of the source to the destination. For example, if your source color is dark red, this operation will add a small amount of red color to the destination causing it to brighten and also turn red. The lighter your source color, the lighter your result will be because a 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`操作将源的颜色与目标颜色相加。举个例子，如果源的颜色是深红（dark red），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？）。推荐在源图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（dark red），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。

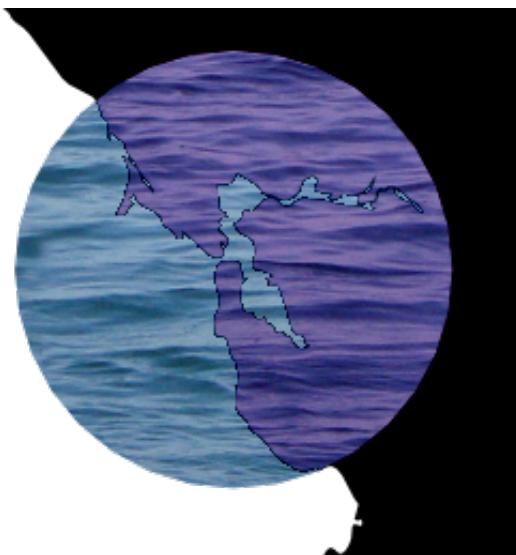
Soft-light



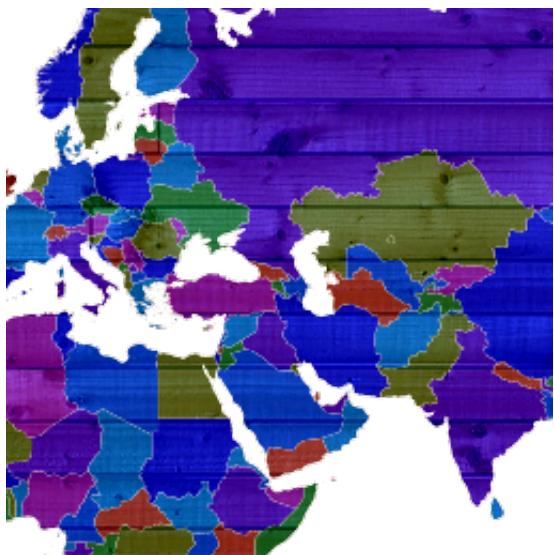
The **soft-light** comp-op works like a less intense version of the overlay mode. It is useful for applying texture effects or ghost images.

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。

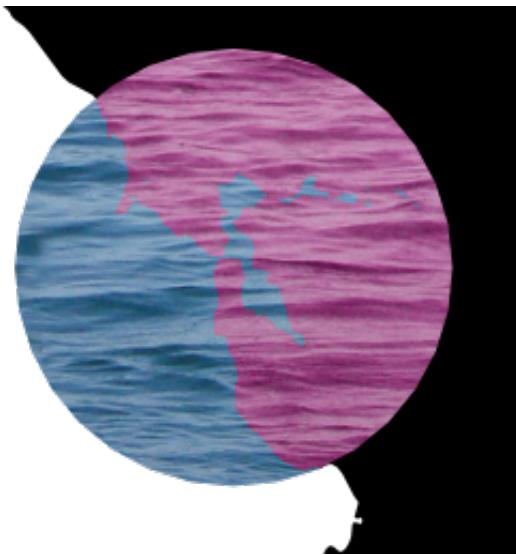
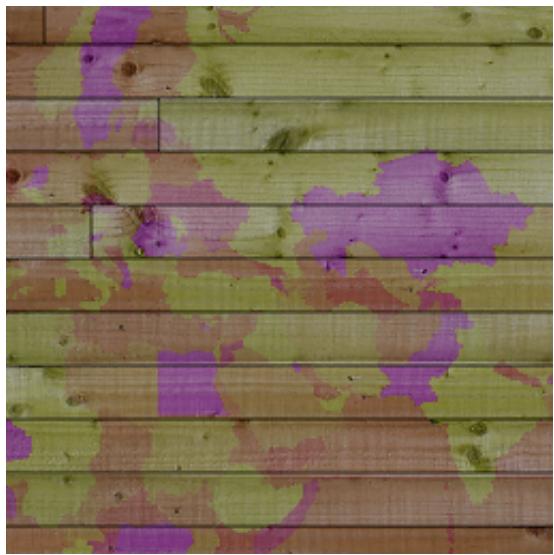
Grain-merge



Grain-extract



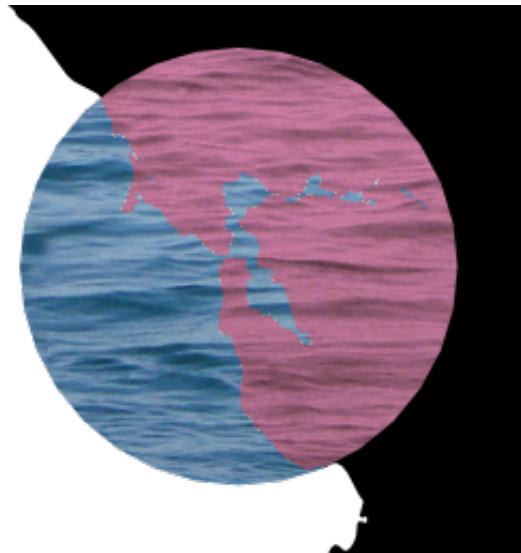
Hue



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）。

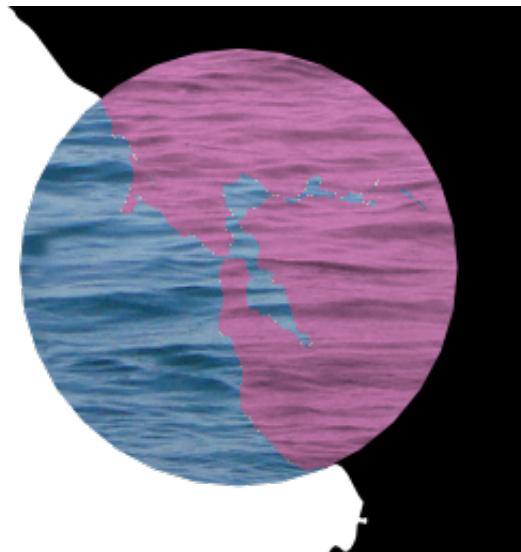
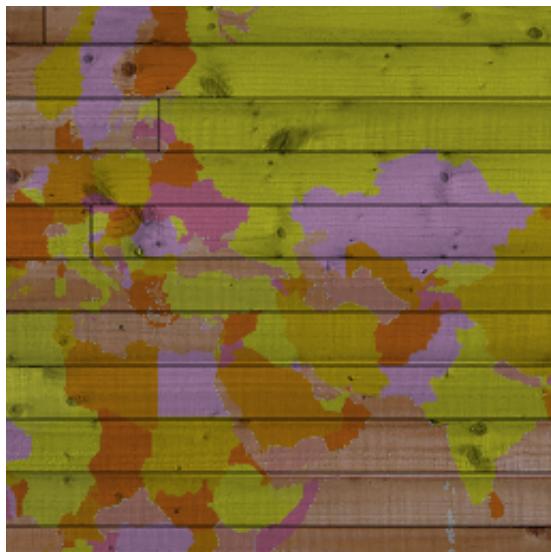
Saturation



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。

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，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](#)（有时需要翻墙）。

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HPC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。

高级技巧

介绍利用CartoCSS进行地图设计的一些高级技巧与方法。

高级地图设计

译注：[原文地址](#)

This guide will cover some of the more advanced techniques you can employ in TileMill to take your map to the next level. For demonstration, we will continue to work with the 2010 tornado data from the [preparing data with Google Docs guide](#).

这节中的内容包含了能够让你的地图质量提升一个档次的一些技巧。本节中使用的示例数据是2010年的龙卷风统计数据，关于数据准备可参考[“利用谷歌文档准备数据”](#)。

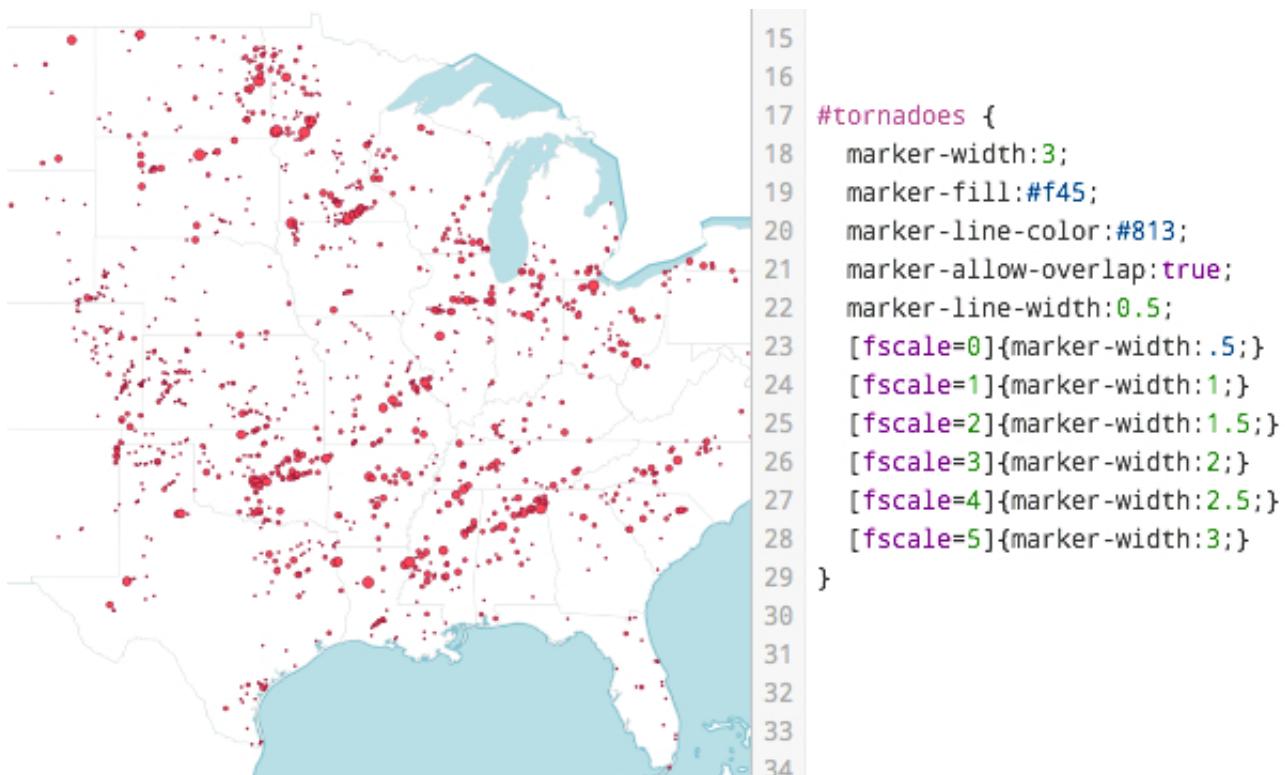
根据缩放级别调整样式（Styling for Zoom Levels）

Interactive maps give users the ability to change the scale by zooming in or out and we must design them accordingly. Assuming you have [imported your point data](#) and done some initial styling, it is time to start thinking about how your map will look at different **zoom levels**.

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。在地图设计的时候，需要针对不同缩放级别考虑不同的样式配置。我们这里假定有一组点数据已经准备好了，并且配置了一些初始样式。那么现在就考虑一下怎么才能让这张地图在不同的缩放级别具有不同的样式。

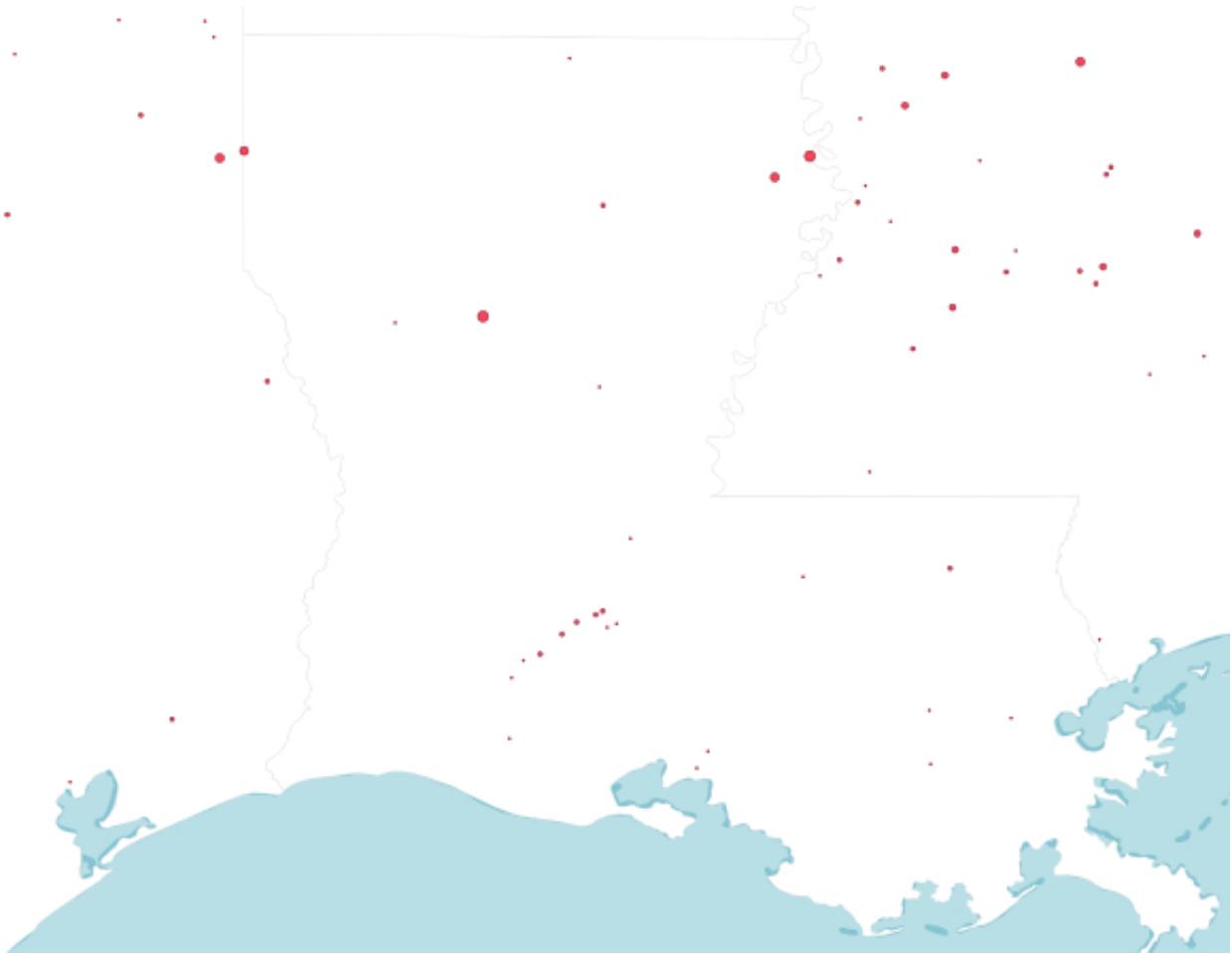
Here is the tornado map at **zoom level 4** after sizing the markers based on their intensity (f-scale).

下图是在地图缩放到4级时的龙卷风地图。其中的注记符号根据龙卷风的强度（f-scale）设置了各自的大小。



And here is the same map at **zoom level 7**. Notice that the dots did not scale with the rest of the maps and could be considered too small.

如果把上面的图放大到7级，那么就可以看到那些点不会随着地图放大而放大，因而在这个级别上看起来就太小了（如下图）。

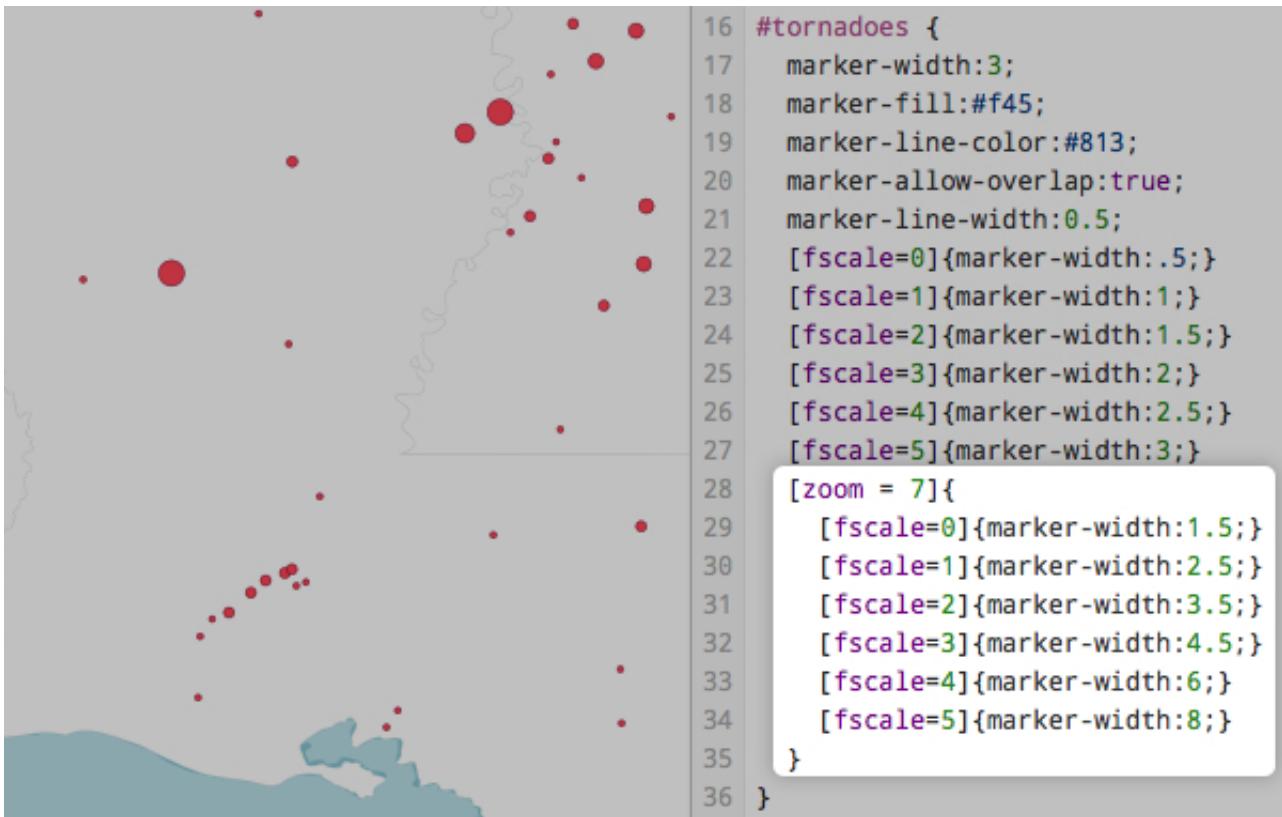


With some simple CartoCSS, you can solve this by **conditioning your styles based on the zoom level**.

只需要用一些简单的CartoCSS，就可以轻松实现让地图在不同缩放级别下展示出不同的样式。

The highlighted CartoCSS below is saying to TileMill, “when the zoom level is 7, apply the following style.” You can do this for as many levels as you wish, and include any kind of styling. This is useful for scaling back the number of dots, icons, and labels as you zoom out, and creating a greater level of detail as you zoom in.

下图中高亮部分的CartoCSS代码的意思就是说：“当缩放级别到达7级时，应用下面这段样式。”用户可以根据自己的需要添加任意数量这样的子样式块。有了这种能力，就可以实现随着地图不断放大也让点、图标和标注跟着一起适当放大，从而展示出更多细节的效果。



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

在条件表达式中可以使用这些比较运算符: =, !=, >, >=, <, <=, 还可以定义一个缩放级别范围, 就像这样:

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

Zoom level conditioning can also be used to limit the visibility of specific layers to certain zoom ranges, making it possible to “turn them on” or “turn them off” as you zoom in or out. This method is particularly useful for joining multiple geographic levels of data.

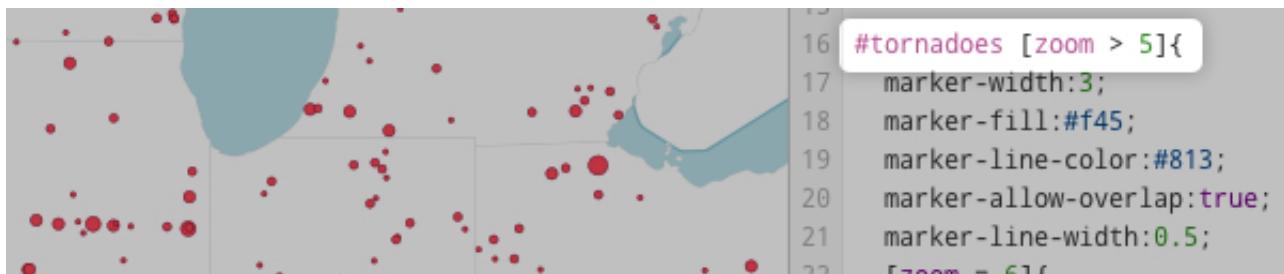
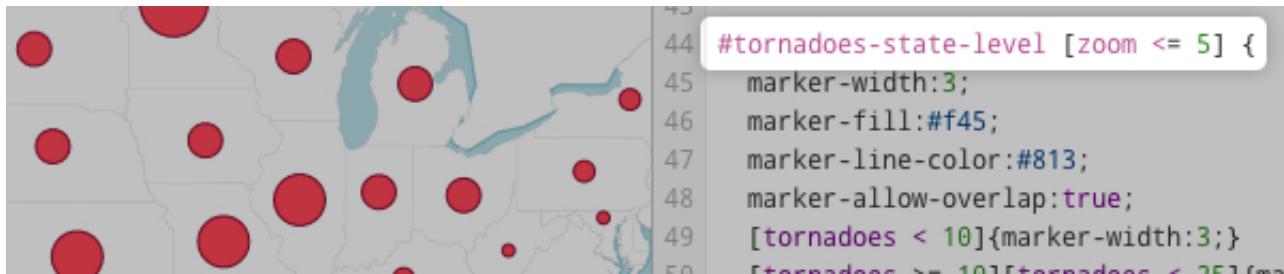
缩放级别条件过滤还可以用来控制某些图层在特定的缩放级别上的可见性, 从而使这些图层随着地图的放大或缩小被“打开”或“关闭”。这个方法对于包含多个地理级别数据的地图特别有用。

Take the tornado map for example. At the national level it is quite cluttered and hard to decipher individual points where there are clusters. At this zoom level, one option might be to display a scaled dot for each state that represents total number of tornadoes that occurred in that state. Then, as the user zooms in, the state-level layer goes away and the individual tornado points appear.

还是以龙卷风地图为例, 在国家级尺度上, 那些聚在一起的点会形成一些杂乱的点团而让人难以辨认那里究竟都有些什么。在这个级别, 可以为每个州按照各自龙卷风的总数只绘制一个点, 而且点的大小可以与龙卷风总数成一定比例。然后随着用户不断放大地图, 就不再显示这个国家级尺度的图层, 而是可以将每个独立的龙卷风点绘制出来。

With this particular data we were able to aggregate to the state level using a [pivot table](#) and then [geocoded](#) the state points. Add this newly created data to the project as a new layer. Then simply add zoom level conditions after the layer names in your CartoCSS code, and continue to style normally.

基于这个特定的数据集可以利用[数据透视表](#)聚合出州一级的点数据集，并且对这些点进行[地理编码](#)。下面要做的就是把这个新生成的数据集作为图层加入，然后就可以继续用缩放级别过滤器为其配置CartoCSS样式了。



You can also use this same syntax to single-out features on the map where a field in your data **meets a certain criteria**. For example, the code below will only show points on the #tornado layer for Oklahoma. "state" is the name of a field in the tornado data that contains state abbreviations.

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条件过滤器，从而挑选出一些特定的要素记录。下面的样式语句只显示#tornadoes图层中的俄克拉何马州的点数据。其中"state"是该图层中的一个包含了各州缩写字母的字段。

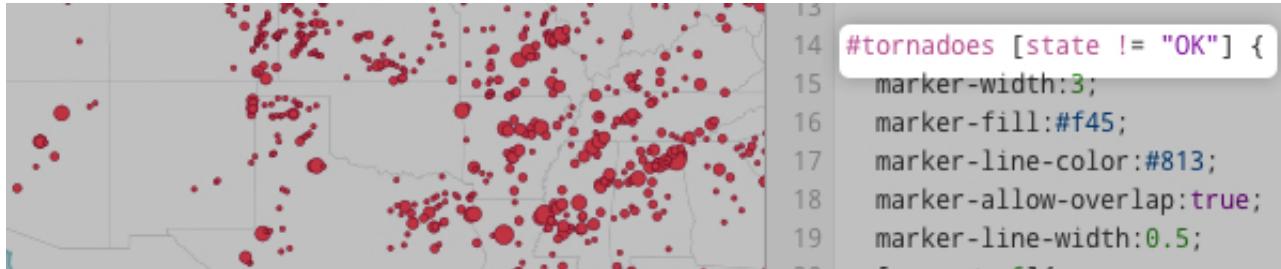
```
#tornadoes [state = "OK"] {
  ...styling...
}
```



The reverse is also possible. Show states that are not Oklahoma:

当然，还可以反向选择，例如显示除俄克拉何马以外其它所有州的数据：

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



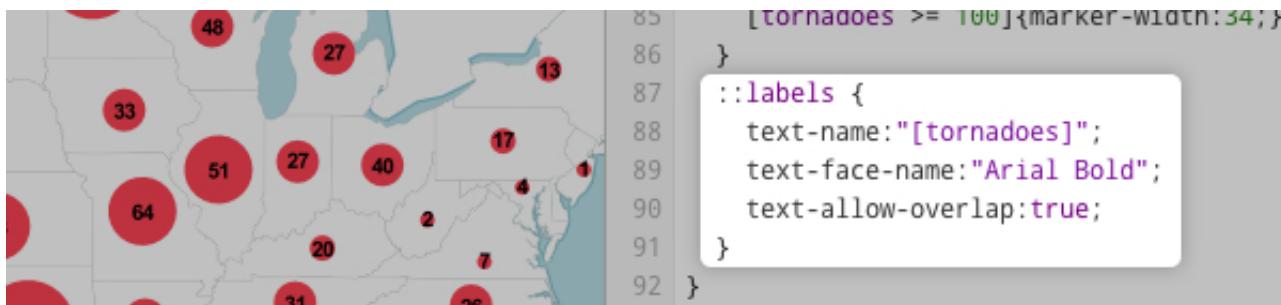
文本标注（Text Labels）

In order to deliver information on a map more immediately, sometimes it is useful to label your data with the actual number or feature that is being represented. This can be combined with the dot or just be a label on its own.

为了能够在地图上更快捷的展示信息，为所表达的内容添加数字或文字标注是一种有效的方式。标注可以标在对应的点旁边，也可以不把点绘制出来而只绘制标注。

For our tornado map, we have decided to display the total number of tornadoes inside the state-level dots. To do this we need to add just a few lines to the layer's CartoCSS:

在这张龙卷风地图上，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，只需要稍微增加几行CartoCSS代码即可：



1. ::label

This creates a new **symbolizer** for your layer. The name 'label' here is arbitrary, you can call it whatever you like. The position of the symbolizer in CartoCSS determines the order of its rendering. The first code in a CartoCSS layer is rendered first on the map and will be **below** anything that is rendered after it. Therefore, if you need a layer feature to be on **top**, like we do with the labels, it must come last in the code.

它的含义是创建一个从属样式块（译注：原文中是“新符号”，在TileMill的系列文档中，均使用“新符号”这个说法，而在新的Mapbox Studio文档中，都变成attachment，也就是“从属样式块”了，为了保持全文上下一致，我都翻成“从属样式块”）。这里的从属样式块名label可以随便起，而从属样式块在整个样式表中的位置则决

定了它被渲染的次序。在CartoCSS的样式块中，最先出现的代码会被首先渲染到地图上，因而会被绘制在后面其它渲染的要素的下面。因此，如果需要将某一层渲染到地图的最上层，那么就应该把它定义在样式代码的最后。

2. text-name

This denotes the field whose text will be displayed.

这是指明了哪个数据字段将被作为文本标记中显示的内容。

3. text-face-name

This sets the **font** for the text label. You can view a list of available system fonts by clicking the **font button (A)** on the lower left.

这是用于设置文本标注上文字的字体。

4. text-allow-overlap

This allows the text and the dots to be displayed together at the same location. By default this option is set to false, which prevents overlapping items.

这个设置是允许文本标注和圆点标记重叠显示。这个属性的默认值是false，也就是不允许重叠。

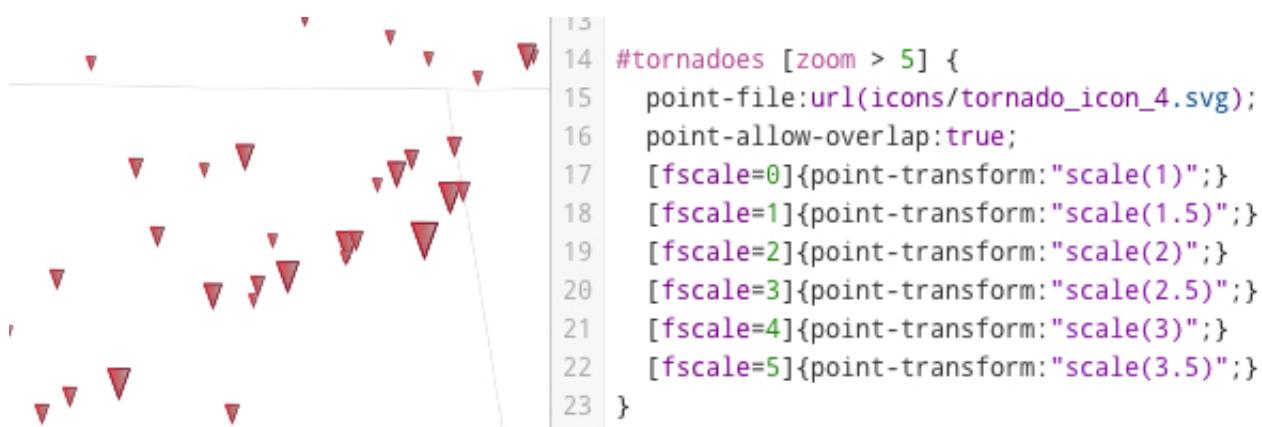
That is all you need to get started with **labels**. The same idea applies to placename labels as well. You can further style them with the text- style parameters, changing things like size, color, opacity, placement, and more.

以上这些用来配置一个基本的文本标注已经足够了。把它们用于地名标注也没有任何问题。用户可以通过text-系列属性进一步调整诸如字号、颜色、透明度、位置偏移等样式。

使用图像作为图标 (Images as Icons)

TileMill supports using **SVG (Scalable Vector Graphic)** images as markers on your map. It is possible that we could use a custom-made tornado icon in place of the circle markers. The first thing you need is the SVG file saved somewhere on your system, preferably in your project folder for the sake of organization (Documents/MapBox/project/project-name/). Then it's all in the CartoCSS.

在CartoCSS中可以使用SVG图像作为地图上的注记。在这个台风地图的例子中，我们可以使用自定义的图标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的SVG文件。为了组织方便，最好把它放在项目的目录中。



1. point-file

This designates the **path** to the SVG relative to your project folder. In this case the SVG is located in Documents/MapBox/project/2010-tornadoes/icons/.

这是指向SVG文件的路径

2. point-allow-overlap

Like other `-allow-overlap` parameters, this allows the images to be displayed even if they will be on top of each other.

就像其它`-allow-overlap`参数一样，这个属性允许表示点符号的图标重叠显示。

3. point-transform

This is the parameter used to **scale** and **move** the image, among other things. A value of "`scale(1)`" will display the image at its original size, while "`scale(0.5)`" and "`scale(2)`" will display it at 50% and 200% respectively. You can also use this property to move the image vertically and horizontally by using the property "`translate()`". For example, the value "`translate(20, -40)`" will move the image 20 pixels to the right and 40 pixels up. There are several other properties that you can employ with `point-transform`.

[Learn about them on W3.](#)

这个参数用来对注记图片进行拉伸与移位。使用"`scale(1)`"将按照原始比例绘制注记图片，而"`scale(0.5)`"和"`scale(2)`"则将分别按照原始图片50%与200%的比例来绘制。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，还有一些可用于点变换的属性，可以从[W3上参考更多详细内容](#)。

导出并合成（Exporting for Compositing）

When your map contains multiple levels of data as our tornado map does, it is sometimes wise to export each level separately. This has multiple benefits. Firstly, it compartmentalizes your map so that when updating you may not have to re-export the entire map. Secondly, it gives you greater flexibility when [compositing](#).

当一幅地图与这个龙卷风地图的例子一样包含了多个层次的数据时，把每个层次的数据都分别导出并作为独立的图层是值得推荐的做法。这样做有很多好处，首先，这样可以将地图划分成更多细粒度的部分，从而在地图更新的时候不必把整张图再重新导出一遍（译注：这里“导出”的含义有待斟酌，不知道是否理解正确。我认为是在预处理阶段将原始数据用过滤器拆分成若干个数据子集，而它这里的“导出”似乎未必是这个意思）。第二，这样可以为合成操作提供更多方便。

Thirdly, **interactivity** can only be active on one layer at a time. This means if we want a hover tooltip for the state-level dots and for the individual dots, we cannot export them together.

第三，地图的交互一次只能应用在一个图层上。这就是说，（在龙卷风地图的例子中）如果想要在州一级和原始的独立点数据集上都实现悬停工具条的效果，那么就不能把这些数据集一起导出。（译注：这点说的不清楚，需要梳理）

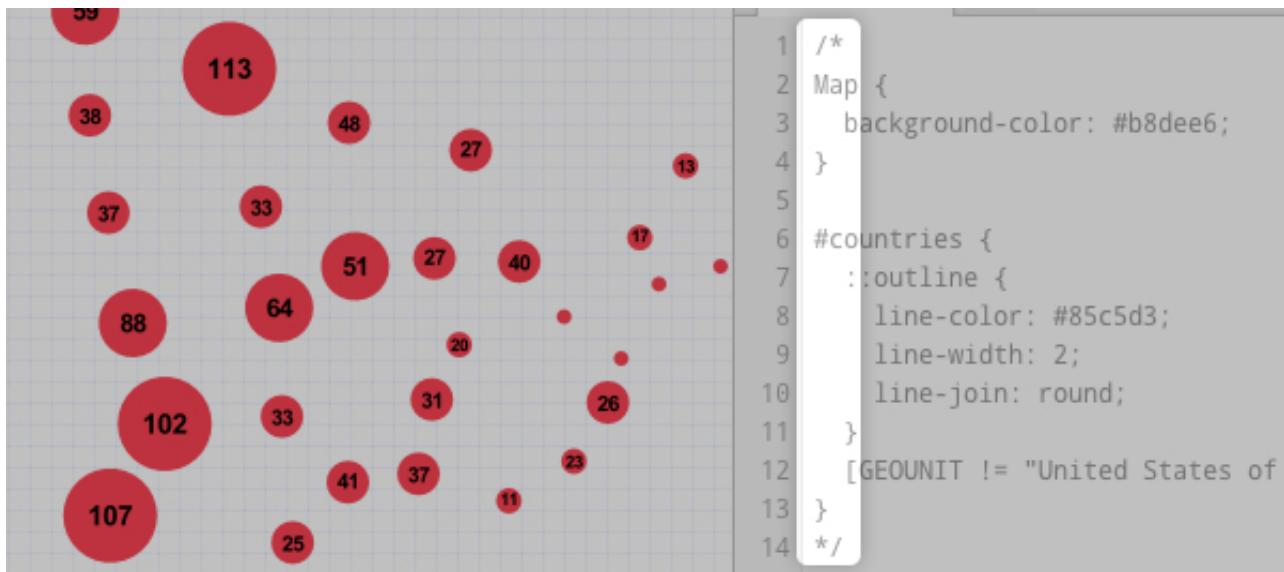
When exporting individual pieces of your project, a very helpful tool is the ability to **comment-out** specific CartoCSS code. Anything commented-out will remain in your code, but not render on the map. All this entails is placing `/*` before and `*/` after the code you want to comment-out. This is also a way to write comments into the code, hence the name.

当需要将项目中的某些独立部分导出的时候，有个技巧是对特定部分的CartoCSS代码灵活运用注释。被注释掉的部分对应的样式不会被渲染到地图上。CartoCSS中的注释也是使用`/*`和`*/`将被注释的部分代码包围。

（译注：最后一句话没有译出，因为感觉含义既有重复，又有不明确的地方）

We have plans to composite this tornado map with an existing world baselayer available from Mapbox, so the first thing to do will be to comment-out the default blue and white world base and the state borders.

在龙卷风地图的例子中，我们准备把龙卷风的分布图与一张现成的世界地图合成。为此，首先应该把蓝白色的世界底图和国界线样式代码注释掉。



Notice that the code between the `/*` and `*/` is now greyed-out, and the background of the map is **transparent**.

注意图中`/*`和`*/`之间的代码已经变成灰色的，而此时的地图背景已经变成全透明的了。

Now we can create a tooltip for the state-level dots, and export. On the export page, be sure to select only the zoom levels for this particular piece of the map.

现在我们可以为州一级的圆点符号创建悬浮提示效果并导出了。在导出配置页面上，要确定只选择导出那些针对这幅地图特定部分的缩放级别。（译注：关于导出，还是不太清楚具体的含义，所以翻译的不理想）



Next we create and switch the tooltip to the individual dots layer, and export. Change the zoom levels and filename accordingly.

接下来，我们为每个原始的龙卷风统计点创建悬浮提示效果并导出。在导出时，将缩放级别和文件名做相应的修改。



We now have two MBTiles with their own interactivity that we can [composite](#) together with a slick base map.

这样，我们就得到了两个都具有交互能力的MBTiles数据集，并且它们能够与一个平滑底图一起合成。

Here is the final map: <iframe width='600' height='400' frameBorder='0' src='https://a.tiles.mapbox.com/v3/mapbox.map-4qkj96dp.html#4/40/-98'></iframe>

最终的地图效果在这里: <iframe width='600' height='400' frameBorder='0' src='https://a.tiles.mapbox.com/v3/mapbox.map-4qkj96dp.html#4/40/-98'></iframe>

And the final project CartoCSS code for reference:

用于这幅例子地图的全部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;
}

```

色彩使用技巧

译注：[原文地址](#)

色彩理论与制图（Color Theory and Mapping）

It's important to realize how much your choice of color on a map can affect the story you are trying to tell with your data. As a brief introduction to some theories behind how to use colors on maps, here are three primary scheming designs for mapping data.

为地图选择的配色方案会在很大程度上影响你利用地图来讲述的关于数据的故事的精彩程度。本节内容是关于地图配色理论的简介，包括了三种用于展示地图数据的基本色彩设计方案。

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（Sources for Color-Picking）

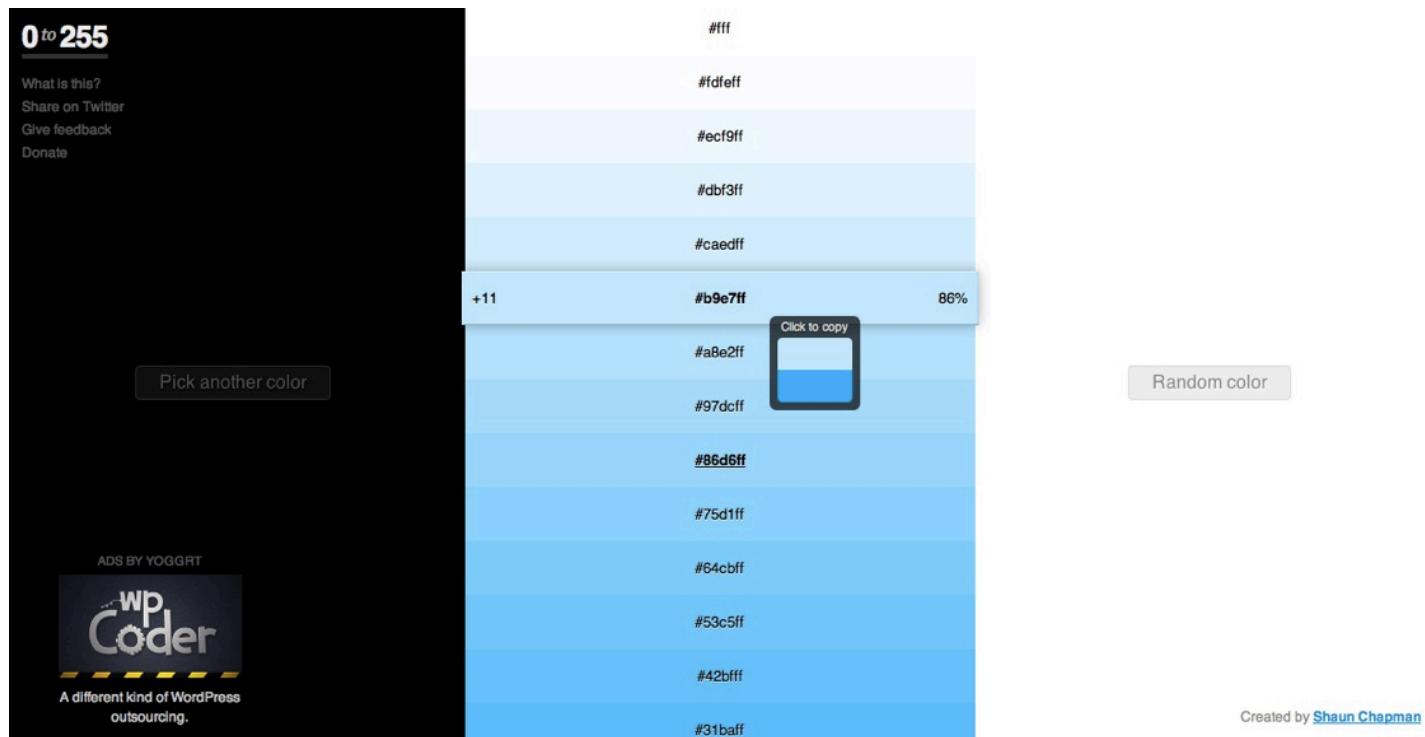
Fortunately a number of sites are available to explore color themes. Below are links to some of these with a brief synopsis of their main features that will help you more easily navigate the seas of storytelling via aesthetics. The section ends with two reviews of purchasable tools that you can use from your desktop.

幸运的是，有不少网站和工具可以帮助我们生成配色方案。这里简单介绍几个这样的网站以及其主要功能。它们可以帮助你更方便的在颜色的艺术海洋中找到最适合讲述自己故事的色彩。除了一些网站，最后还会介绍两个桌面应用，不过它们是付费软件。

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可以在同一色系中选取一组亮度不同的颜色，因此非常适合生成连续色方案。它为任意颜色都提供了30种选项，还可以直接拷贝颜色的十六进制值。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自动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）。

COLOURlovers

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

Sign Up Log In ▾ 

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.

 [Join the Community!](#)

- [ALL](#)
- [WEDDING](#)
- [HOME](#)
- [FASHION](#)
- [WEB](#)
- [PRINT](#)
- [CRAFT](#)
- [BUSINESS](#)

LATEST BLOG POSTS View More >



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



Mothers Day **TEMPLATE** **CONTEST**
 50 Comments



Contest: Mothers Day Floral Template Gets You ImageKind Bucks!
 5 Comments



24 Smart Logo Color Designs


PALETTES + PATTERNS + COLORS +



ColorSchemer • COLOURlovers

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

[DOWNLOAD »](#)



COLOURlovers

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。

Scish by riesjart

Created: 2011.04.16 at 04:34 AM
Last Edited: 2011.04.16 at 04:38 AM
Rated: 3.91 (6 votes)
Favorited by: 2 members
Downloaded: 3 times
Theme Link: <http://kuler.adobe.com/#themeID/1327095>

Comments: 1 Post on: 2011.05.12 at 10:55 AM by rickomoreira

News & Features

- Capture creative ideas anywhere! Use free Adobe Ideas mobile sketching app to create and use color themes on iPhone/iPad. [Learn more here](#)
- Kuler and Creative Suite: Access Kuler directly in your Creative Suite 4 & 5 applications. See tutorial on [Adobe TV](#)
- Try [Community Pulse](#) and explore Kuler colors in a new way
- Get inspired by [flickr](#) images: Create > From an Image
- Developers: Apply for your [Kuler API key!](#)

Welcome to Kuler

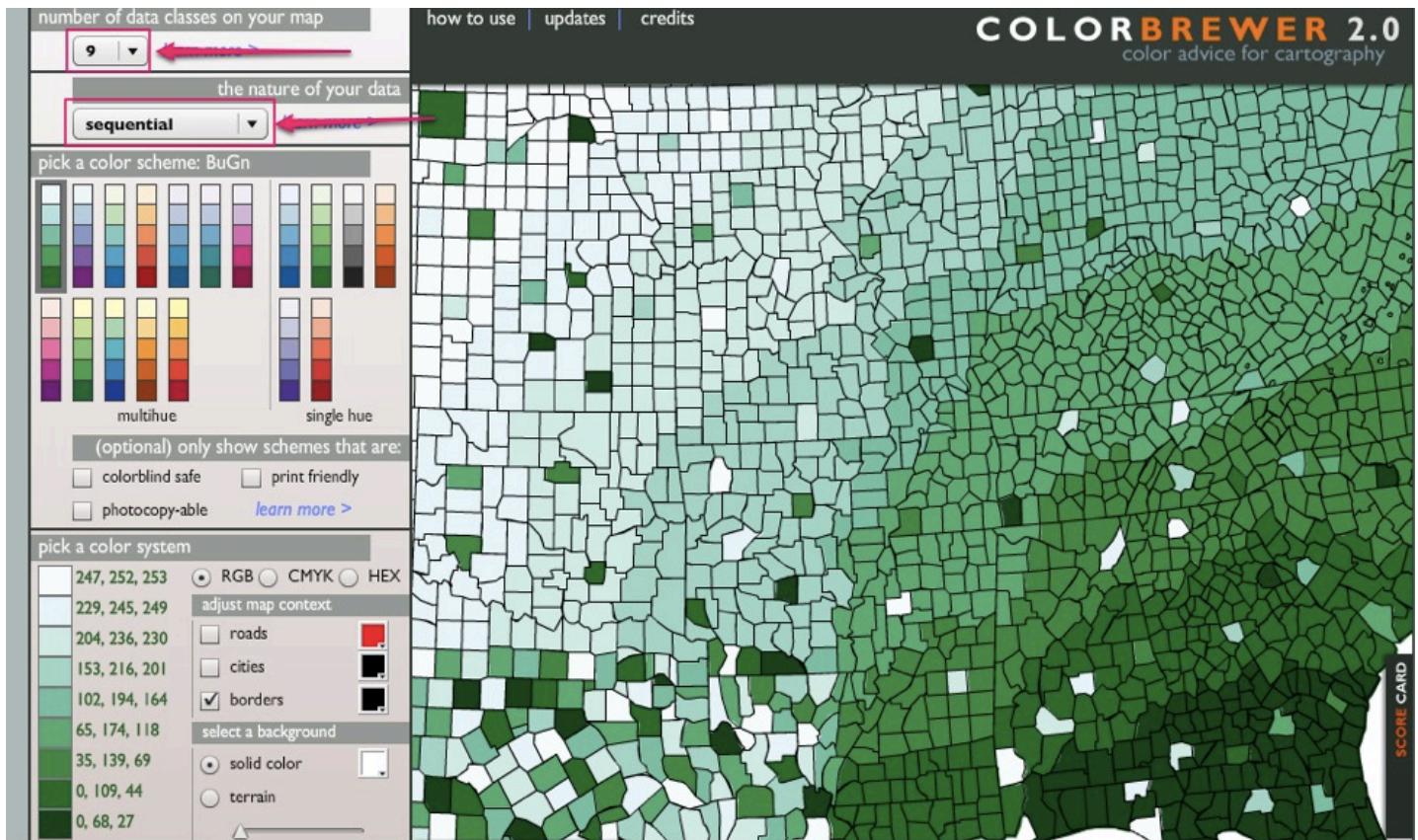
There are a lot of great ideas being posted by an active community of contributing users. Kuler also provides extensive links to things like forums, help pages, as well as several articles on the importance of color and color theory.

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。

高级标注方法

译注：[原文地址](#)

尝试在其它位置标注 (Trying multiple positions)

Recent versions of TileMill include two methods to choose for placing labels on points. The choice is made via the `text-placement-type` CartoCSS property. The default, original method is called `none`, and the newer, more advanced method is called `simple`.

CartoCSS通过`text-placement-type`属性支持两种在点符号上面放置标注的方法。其默认值是`none`，除此以外还有个高级方法——`simple`。

The `simple` method allows the designer to specify multiple potential positions on or around a central point, as well as multiple sizes of text to choose from. If the first attempt at placing a label is blocked by another label that has already been placed, it can look at this list to try the next position.

这个`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位置和字号。如果在默认位置渲染某个标注时出现了被其它标注遮挡的情况，那么就会从这些指定的候选位置中逐个尝试绘制。

A full CartoCSS example of the syntax looks like this:

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;
}
```

This will first attempt to place a label above the point, then below the point, then to the right, and so on with a text size of 16 until it finds a position that fits. If no labels fit at size 16, the positions will all be retried at a text size of 14, and then 12. If none of these fit the label will be skipped.

这段代码的作用是依次在原始点的上方（北方）、下方、右侧等进行尝试，如果位置合适，那么就用16号字绘制标注。如果这些位置都画不出来，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。如果都不行，那么这个标注就会被跳过不画了。

The `text-dx` and `text-dy` properties specify how far away (in pixels) the label should be placed from the point.

`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）。

改进标注位置的分布：随机法（Improved direction distribution: random approach）

It's great to try different placements for a label, but the previous example will always try the same position (North) first. This may not always be the best choice, even if the label happens to fit there. And it may be better for the overall map design to distribute the different placement positions better, rather than letting a single position dominate.

尽管在多个不同的位置尝试放置标注的想法不错，但它的问题在于每次都是从同一个位置开始尝试。这往往不是最好的选择（译注：从地图设计和美观的角度），即使标注可以被绘制在那个位置。将标注位置更合理的分布，而不是总绘制在同一个位置也会带来更好的整体地图设计效果。

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，就是只要将标注相对于原始点的偏移方向变成随机性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），它的值是随机生成的0或1。

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

You could then set up your CartoCSS to favor East placement for the 0s and West placement for the 1s.

然后就可以基于dir列的值，让每个点的标注在为0时向东、为1时向西偏移。

```
#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"; }  
}
```

改进标注位置的分布：邻居避让法（Improved direction distribution: avoiding nearby neighbors）

Using PostGIS its possible to come up with something smarter than random distribution to improve the look of simple label placement. One possibility is to calculate the direction of the nearest object of a certain type, and then try to avoid that. For example you could bias city lable placement away from the next nearest city, or county label placement away from the largest city in the county. These aren't perfect solutions, but can be a quick way to make your labels more correct in more cases.

利用PostGIS可以得到比随机方法更加美观合理的标注排布。一种方法是先找到当前标注点的特定类型的最近邻对象，看看它的标注偏移方向，然后在绘制标注时尽量避开最近的邻居。例如在标注城市名称时，可以让每个城市的名字标注都稍作偏移以避开其最近的其它城市，而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。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。

For labels on points-of-interest along a city block at high zoom level, the area most likely to have room for the label is away from the street. Placing labels here also keeps the street clear for its own labels and one-way arrows.

译注：第一句没看明白意思。前半句里的along a city block是个什么东东？想象不出来。后半句的意思应该是：最有可能放得下标注的地方（正好）远离道路。所以，将标注置于这里还可以保持道路自己的名字和通行方向等清晰可见。

So for each label we need to find the nearest city street and its direction relative to the point. Service streets, tracks, footways, and cycleways will be ignored for this logic, but you could adjust it to account for whatever you feel is appropriat. For a basic use case fine if our label sits on top of an alley or park path; the goal is to avoid the main city grid.

因而对于每个点标注，我们应该找到距离它最近的城市街道以及这条街道相对于原始点的方位。对于一些低等级道路（例如OpenStreetMap数据集中的service streets、tracks、footways和cycleways等），可以不用考虑，但也可以根据实际情况对标注的避让策略进行调整。通常情况下，点的标注被置于一条小巷或公园中的小路上是可以接受的，但城市的主干道不应该被点标注压盖。

Here are some of the spatial functions of PostGIS that will help determine this information:

以下是一些PostGIS中的空间操作函数，可以辅助实现上面的避让策略：

- `ST_Distance` will help us find the closest street to a POI
- `ST_ClosestPoint` will tell us the closest point along the closest street, and
- `ST_Azimuth` will help us determine the angle between the POI and the closest point.
- `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

We can put all these together as a user-defined PostgreSQL function:

所有这些都可以写在一个PostgreSQL函数中：

```
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;
```

This particular function assumes you are working with a standard OpenStreetMap rendering database generated by `osm2pgsql` (you can adjust it to be used with other schemas). The first two lines set up a function with a name, argument, and return value. `$$` starts the function. The result of the function, when given a point geometry as an argument, will be a number between 0 and 360 representing the angle between that point and the nearest street of any of the types defined in the `where` clause. (`ST_Azimuth()` returns a value in radians, but we convert that to degrees to make it easier to work with in CartoCSS.)

上面这个函数假设你已经通过`osm2pgsql`准备好了个标准的OpenStreetMap数据库（你可以对它进行修改以适应其它的数据库结构）。最前面两行定义了函数的名称`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）与该点之间的夹角以角度计算并返回，取值范围为0到360度。（`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。）

To use the above function to your database, copy its contents to a file (for example, `poi_ldir.sql` on your Desktop). Then run a command from the terminal to load it into your database:

要使用上面这个函数，只需要把它存入一个文本文件（例如以`poi_ldir.sql`文件保存在桌面上）并在终端中执行以下命令即可：

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

You can then use the function in your TileMill select statements. This selection will retrieve all amenity and shop points from the database, their names, and column named `ldir` that is the result of the `poi_ldir` function on the geometry for each point.

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
```

To use the `ldir` column in a stylesheet, set up a label style with the simple text placement type and nest some filters within that that adjust the `text-placements` parameter depending on the `ldir` value. This example will only try each label at one position:

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，每个标注就只需要一个在一个位置尝试放置了。

```
#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'; }
}
```

After integrating this style into a more complete OSM stylesheet you can see that most of the point labels are now avoiding the roads.

将上面这段样式应用在一个完整的OSM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。



使用图例

译注：[原文1](#), [原文2](#)

简单图例与浮动工具条

Tooltips and legends allow you to add interactivity, additional information, and context to your maps. Below we'll walk through how to add each to your map.

浮动工具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添加浮动工具和图例。

浮动工具条 (Tooltips)

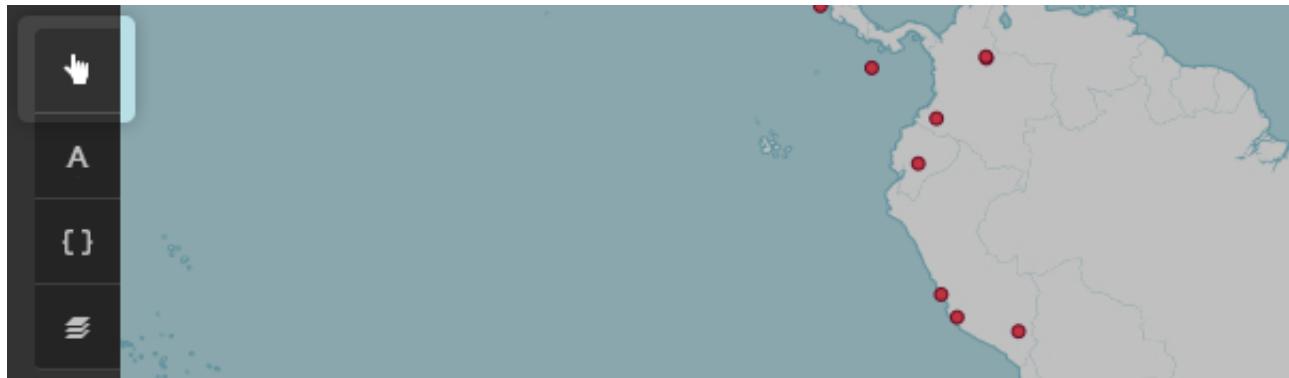
Tooltips allow you to make maps interactive with dynamic content that appears when a user hovers over or clicks on a map. They can contain HTML and are useful for revealing additional data, images, and other content.

浮动工具条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，因而使地图具有了交互能力。

Previously in the Importing data section of this guide, we created a map of earthquakes. Here we will add tooltips that reveal the magnitude, date, and time of each earthquake when users hovers over its point.

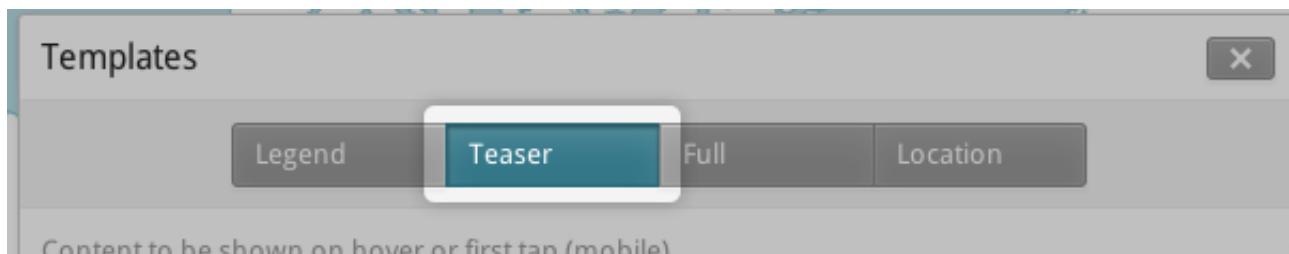
这里我们以一幅地震分布地图为例说明如何实现在鼠标悬浮于每个地震点时展示出震级和时间。

1. Open the Templates panel by clicking on the pointer button on the bottom left.
打开Templates面板

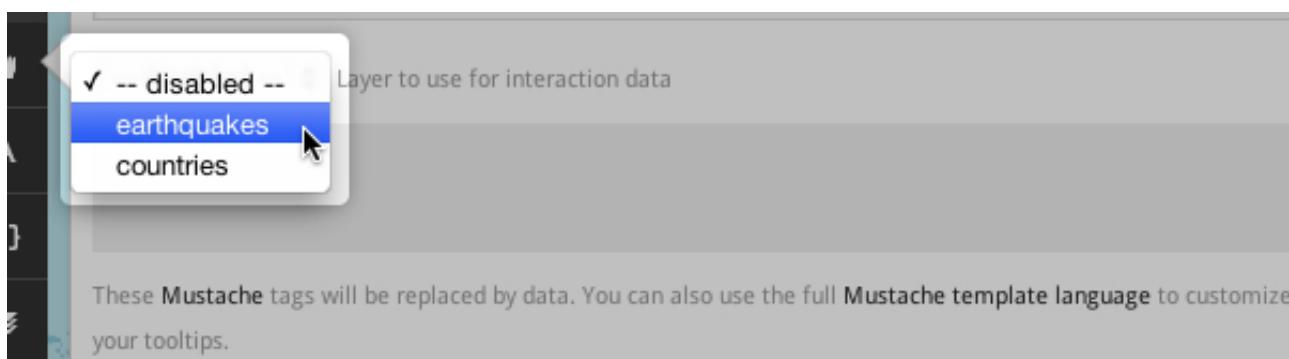


- Click on the “Teaser” tab. Teaser content appears when you hover over a feature and Full content appears when you click on a feature. You can use the Location field to define a URL to be loaded when a feature is clicked.

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。



- Select the “Earthquakes” layer to use it for interaction. TileMill only supports one interactive layer at a time.



- The data fields for the layer are displayed wrapped in curly Mustache tags. These tags will be replaced by data when you interact with the map. Locate the fields you want to use.

图层中需要显示的数据字段都需要被放在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。选择你想要显示的字段。

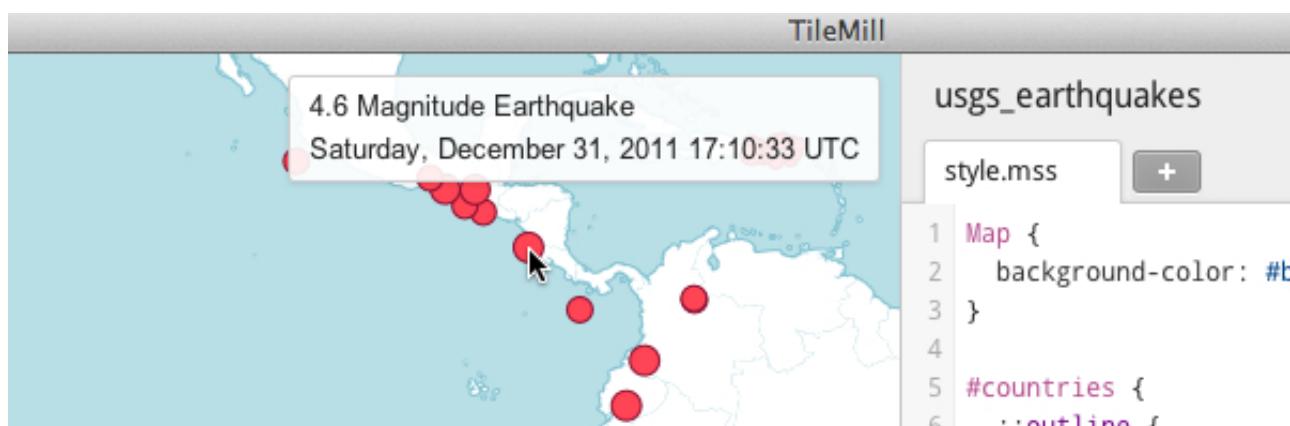
The screenshot shows the TileMill interface with the 'Templates' panel open. At the top, there's a dropdown menu labeled 'earthquakes' with a downward arrow, followed by the text 'Layer to use for interaction data'. Below this is a code editor containing Mustache template code:

```
{{{Src}}} {{{Eqid}}} {{{Version}}} {{{Datetime}}} {{{Lat}}} {{{Lon}}} {{{Magnitude}}} {{{Depth}}}  
{{{NST}}} {{{Region}}}
```

A note below the code says: 'These Mustache tags will be replaced by data. You can also use the full Mustache template language to customize your tooltips.'

5. Write your template using the Mustache tags. Paste the following code into the Teaser field and use the preview to make sure it looks good:
使用Mustache标签来构造你的工具条模板。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. Click “Save” to save your settings and refresh the map. Close the panel by clicking the close button (X) or by pressing the ESC key. Move your mouse over some points to see the tooltips.
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浮动工具条的展示效果。



图例 (Legends)

A legend is permanently on a map and is useful for displaying titles, descriptions, and keys for what is being mapped. It can be styled using HTML, or it can simply contain an image.

图例是始终展示在地图上，用于对地图中的标题、描述等关键信息进行说明的制图要素。图例可以使用HTML构建，也可以是一张简单的图片。

Let's add a legend that describes the theme of the map.

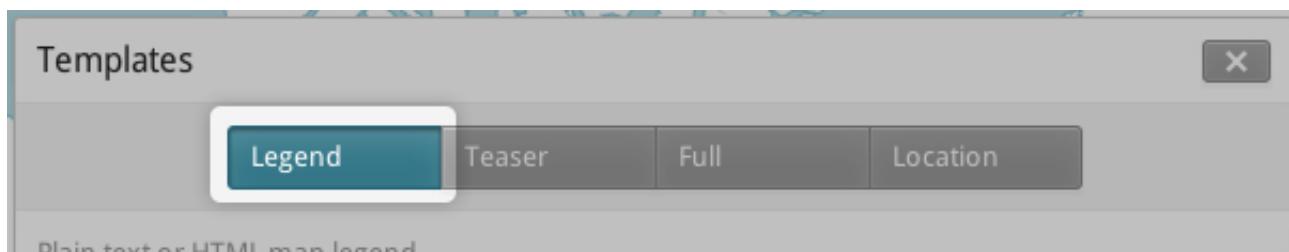
让我们看看如何来为地图增加一个描述其配色方案的图例。

1. Open the Templates panel by clicking on the pointer button in the bottom left.
打开“Templates”面板。



2. The Legend tab is open by default.

进入面板后默认打开的就是图例标签页。



3. Enter your legend text/html in the Legend field:

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. Click “Save” and close the panel. You will now see your legend in the bottom right corner of the map.

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。

Allowed HTML

合法的HTML

For security, unsafe HTML in tooltips and legends are sanitized and JavaScript code is removed. If you want to build sophisticated map interaction with JavaScript on your own website, you can write custom code using the [MapBox.js API](#).

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（Advanced Legends）

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.

要想实现一些不只是简单文本的高级图例，有几种方法可以做到。高级图例可以是图片、html/css或者是这些要素的组合。但无论哪种方法都是各有利弊。

嵌入图片（Insert an image）

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，还可以直接[在图例中使用base64编码方式嵌入](#)。

The advantage with images is that you have the ability to design every pixel and they can be as complex as you want. The drawback is that the image is static once it's in the map, and it may not be as easy to update, as you need the original file and software that can open it.

使用图片的好处在于你对它的设计可以控制到像素级，图片的复杂程度完全尽在设计者掌握。然而它的不足之处在于图片是静态的，一旦画到地图上就难以修改更新，因为通常需要图片的原始文件和专门的软件才能对其编辑修改（译注：例如Photoshop对应的ps文件，Illustrator对应的ai文件等）。

HTML/CSS

For simpler, table-like designs and those that feel more comfortable designing with code. This involves creating the layout and styling for the legend elements in the same way one would build a web page.

也可以简单点，利用代码把图例设计成表格形式。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。

The advantage with html/css is that you can quickly make edits to the legend directly in TileMill, and maintain the ability to manipulate the legend styling with css even after the map has been exported to MBTile format. However, you are limited to right angles and solid colors, and may have to write many lines of code to create a relatively simple design.

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，然后可以立即看到效果。即使是在将地图导出到其它外部格式（比如MBTile）之后，也可以对图例进行维护。但是，图例的设计将会被局限于使用直角和纯色，并可能为了实现一个相对简单的设计而不得不编写很多代码。

Another big advantage with html/css is that you can easily pass the source code from project to project and person to person. Below are a couple of **basic templates** for getting started, and how you can make them your own. This is not intended to be a tutorial on html or css. If you would like to learn more about these languages, check out the great guides at tizag.com.

使用html/css还有个巨大的优势，就是可以把图例代码随处拷贝和复用。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，那里有很好的学习材料。

Copy and paste the block of code directly into TileMill's legend field. Then follow this guide to tweak the template for your own purposes.

把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节内容进行适当的修改。

```


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%

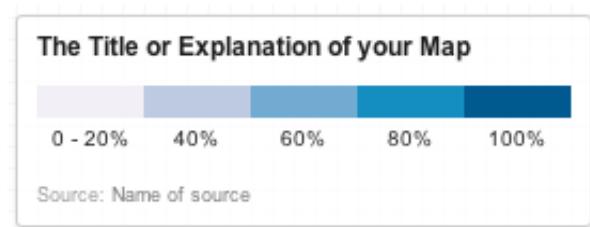


source link
source name


```

1. Horizontal Sequential

1. 水平排布的图例



```


The Title or Explanation of your Map



legend-labels



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



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


```

```

<style type='text/css'>
.my-legend .legend-title {
  text-align: left;
  margin-bottom: 8px;
  font-weight: bold;
  font-size: 90%;
}

```

```

}

.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. Vertical Qualitative

2. 垂直排布的非数值型图例



```
<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 (The legend class)

TileMill legends can be contained within an element with a custom class (e.g. `my-legend`). This is why you see it included in each selector in the above style sections. This class is attributed several default styles, including a `max-width` of 280 pixels and a `max-height` of 400 pixels. Under normal circumstances this should be plenty large enough. You'll know they're not if you see a scrollbar in your legend. In case you ever need to change these, here's how.

图例可以被包含在一个具有自定义类（比如`my-legend`）的要素中，就像上面两个例子中展示的那样。这个类具有多个默认样式，包括`max-width`是280像素、`max-height`是400像素。通常情况下，这个尺寸对于图例是足够的。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，那么需要按照下面的方法做。

Inside the `<style></style>` tags add a selector for `my-legend` and declare the new value(s). For values that are overriding previous declarations, you will likely need to add the `!important` tag. Say you want to increase the width to 300 pixels:

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声明，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;
}
```

理解元瓦片

译注：[原文地址](#)

TileMill displays maps in small seamless chunks referred to as tiles. Although displayed individually, TileMill generates groups of images at once in batches before separating them into the final tiles - this improves efficiency in various ways.

在现代Web制图中，地图是由一系列小块无缝组成的。这些小块被称为瓦片。尽管瓦片最后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处理瓦片，然后再对其拆分得到最终的瓦片。这种处理方式能够从多个方面改善制图效率。

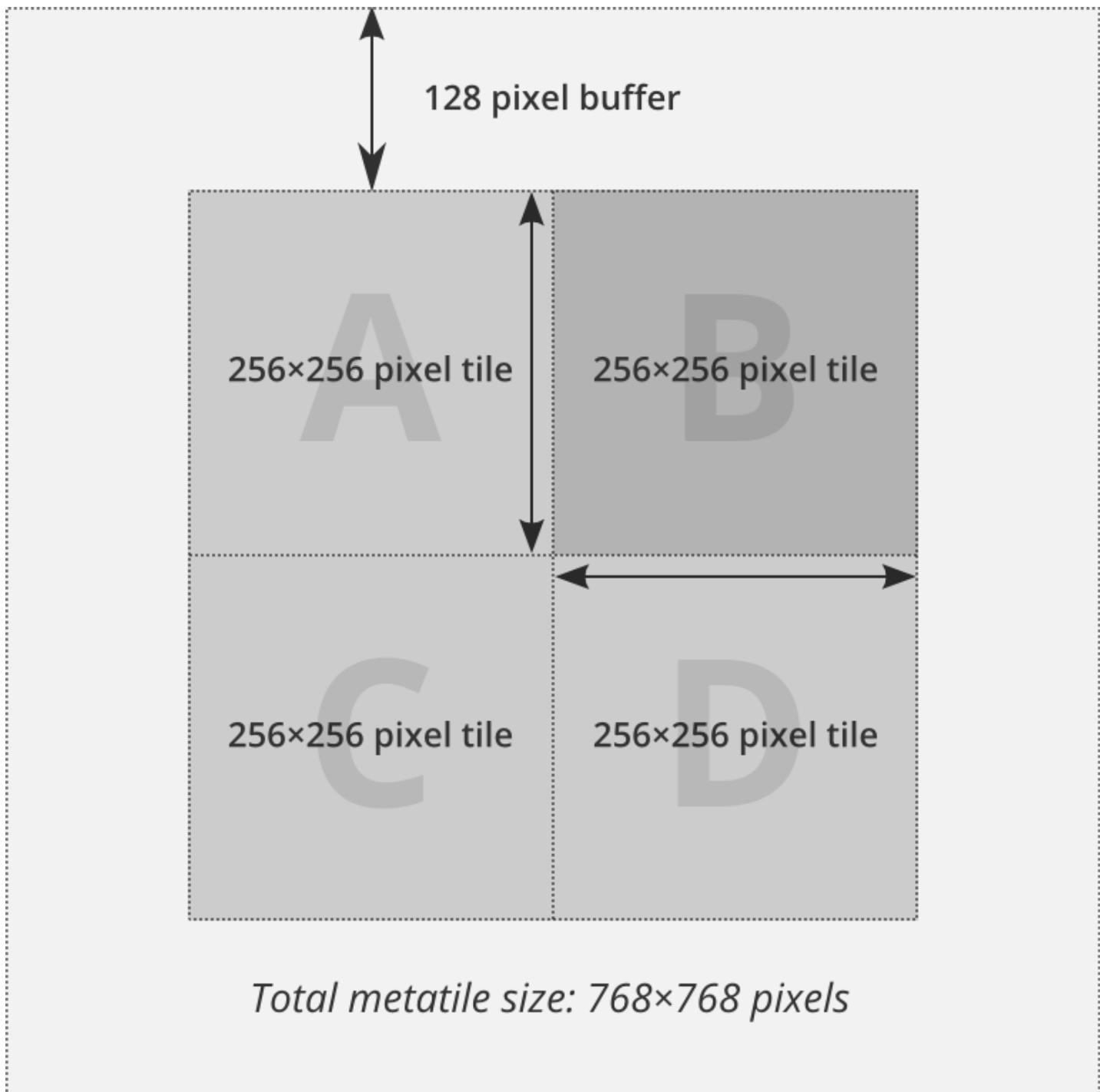
When designing maps in TileMill it is sometimes necessary to understand how metatiles work in order to create your style effectively and work around certain types of issues. Metatile settings play a particularly important role when working with labels, markers, and patterns in your stylesheet.

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，以及规避解决一些制图渲染中的特殊问题，了解一下元瓦片（**metatiles**）的工作机理是很有必要的。元瓦片在正确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（Structure of a metatile）

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 128 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像素宽的缓冲区组成。



The area within the buffer is drawn but never displayed - the purpose is to allow labels, markers, and other things that cross over the edge of a metatile to display correctly. Without a buffer you would notice many cut-off labels and icons along the seams between every other tile.

元瓦片的缓冲区部分也会被渲染引擎（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。这样做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（Adjusting metatile settings）

TileMill allows you to configure the number of tiles included in a metatile as well as the width of the buffer.

元瓦片中包含的瓦片数量和缓冲区大小都是可以配置的。

To adjust the number of tiles, open the project settings (wrench icon) and drag the MetaTile size slider to your desired size. The number represents how many tiles high and wide your metatile will be, thus the total number of tiles in each batch will be the square of this value.

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瓦片的个数。

Adjusting the width of the buffer is done in CartoCSS. Add a `buffer-size` property to your `Map` object. The value must be a whole number and represents pixel units. Example:

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;  
}
```

选择合理的缓冲区大小（Choosing your buffer size）

If you are noticing problems with your map such as cut off labels & icons you should try increasing your buffer size. A good starting point for choosing a buffer size is to make it about the width of your widest labels.

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。但是应该把宽度设成多少呢？建议先找到地图上最宽的标注，然后从设成它的宽度开始尝试。

选择合理的元瓦片尺寸（Choosing your metatile size）

For most projects it's reasonable to use the default metatile size (2). This means that tiles will be rendered in 512 px chunks and then broken down into 256 px tiles before being returned to the map view. When one or more adjacent tile requests hit the same metatile the renderer will pause momentarily to process the metatile a single time before slicing and then returning each individual 256 px tile to the map view.

对于大多数情况，默认的元瓦片尺寸（也就是2）都什么没问题。这意味着渲染引擎会将地图内容先渲染到长宽均为512像素的块上，然后再将其切分成4个长宽为256像素的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。

There is one reason why you might want to lower the size to 1, and two main reasons you may want to increase the metatile size above 2 to values like 8 or 16.

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（Going smaller）

There is only one size down from the default metatile size of 2; this effectively disables metatiling. Doing this can help the map view feel slightly more responsive during editing and light browsing because each individual tile will appear as fast as it can, alone, be rendered. If the current part of the map you are viewing has some tiles with lots of data and other tiles with less data, avoiding metatiling will ensure that the tiles with less data will load quicker than adjacent tiles with more data.

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的过程中响应更快，因为这时每个瓦片都是独立渲染的了。如果地图中某些瓦片对应的数据较多，而另一些瓦片上的数据较少，那么关闭元瓦片功能之后会使包含数据较少的瓦片比包含数据较多的瓦片更快绘制出来。

增大元瓦片尺寸（Going larger）

原因1：减少导出时间（Reason 1: reducing export time）

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性能，降低对导出任务的渲染时间。其原因是在穿行渲染大量瓦片时，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？）

There is no hard rule about how large your metatile should be for the best export performance. It will depend on how much data your project contains, how well it is spatially indexed, and how much memory your machine has.

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、有没有建空间索引、机器的内存有多大等许多因素都有关。

We recommend experimenting by setting up a reduced export job (just a few zoom levels or perhaps a more restricted area) and testing the export completion time as you gradually increase the metatile size to 4, 8, or 16.

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（Reason 2: reducing rendering problems at tile edges）

Larger metatiles mean that things like labels and markers are less likely to be rendered at a tile edge. It also means that labeling algorithms, like the one that can throw out duplicate names (if `text-min-distance` is set) can work over larger areas of the map and will be more successful at reducing duplicates for a given view.

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。此外，对于一些标注算法（比如在设置了`text-min-distance`属性时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。

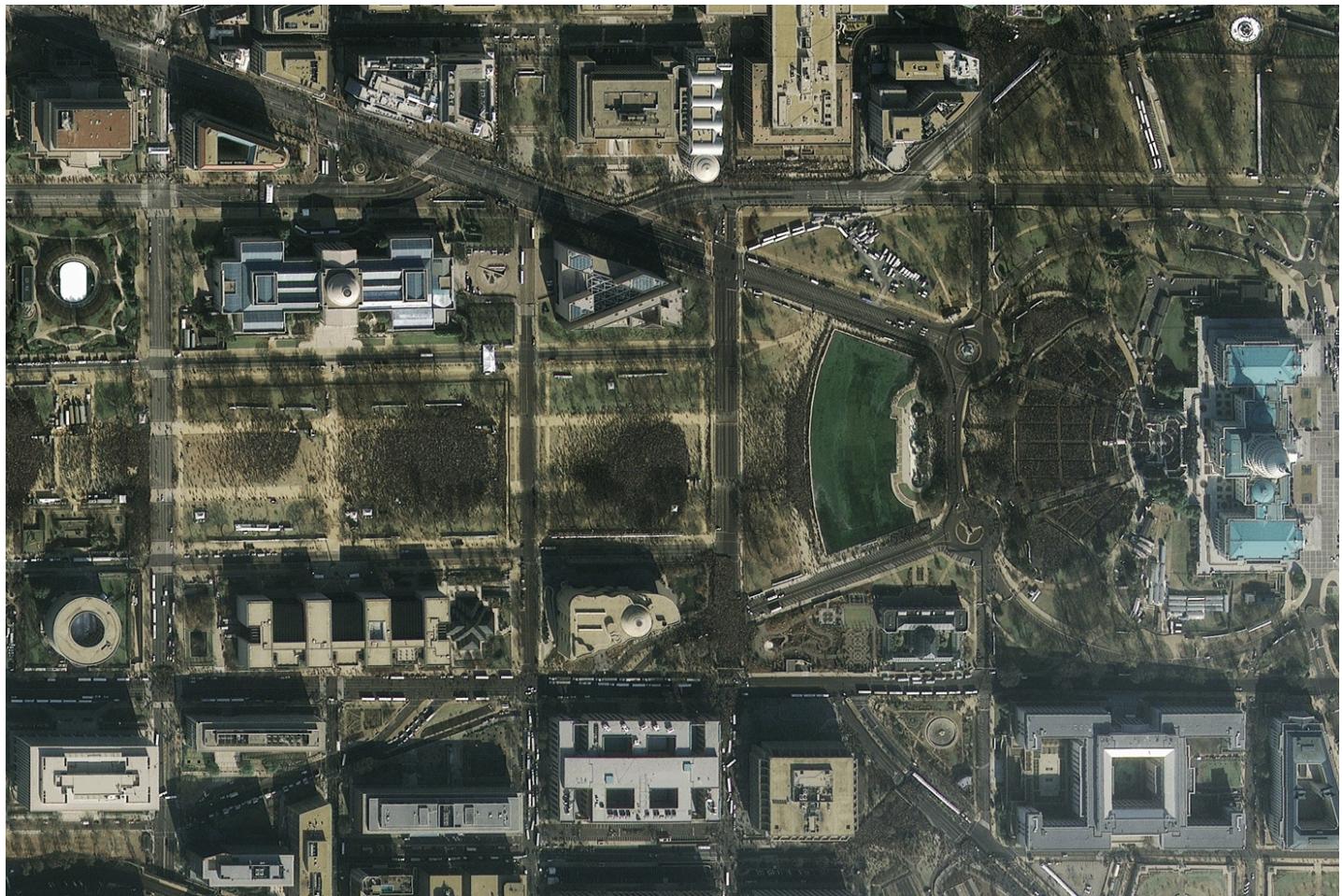
栅格制图技巧

译注：[原文1](#), [原文2](#), [原文3](#), [原文4](#), [原文5](#)

配准卫星影像 (**Georeferencing Satellite Images**)

The skies were clear on President Obama's first Inauguration and [GeoEye](#) was there to capture the incredible imagery from space.

奥观海总统首次就职典礼的那天，天空万里无云。那时[GeoEye](#)恰好在白宫上空，它从太空中捕捉到了那个精彩的瞬间：



GeoEye | 2009 Inauguration

Using [GDAL](#), [QGIS](#), and [MapBox Satellite](#), we were able to manually georeference the 2009 imagery and compare the ceremony attendance and changes to the city with our own [MapBox Satellite layer](#).

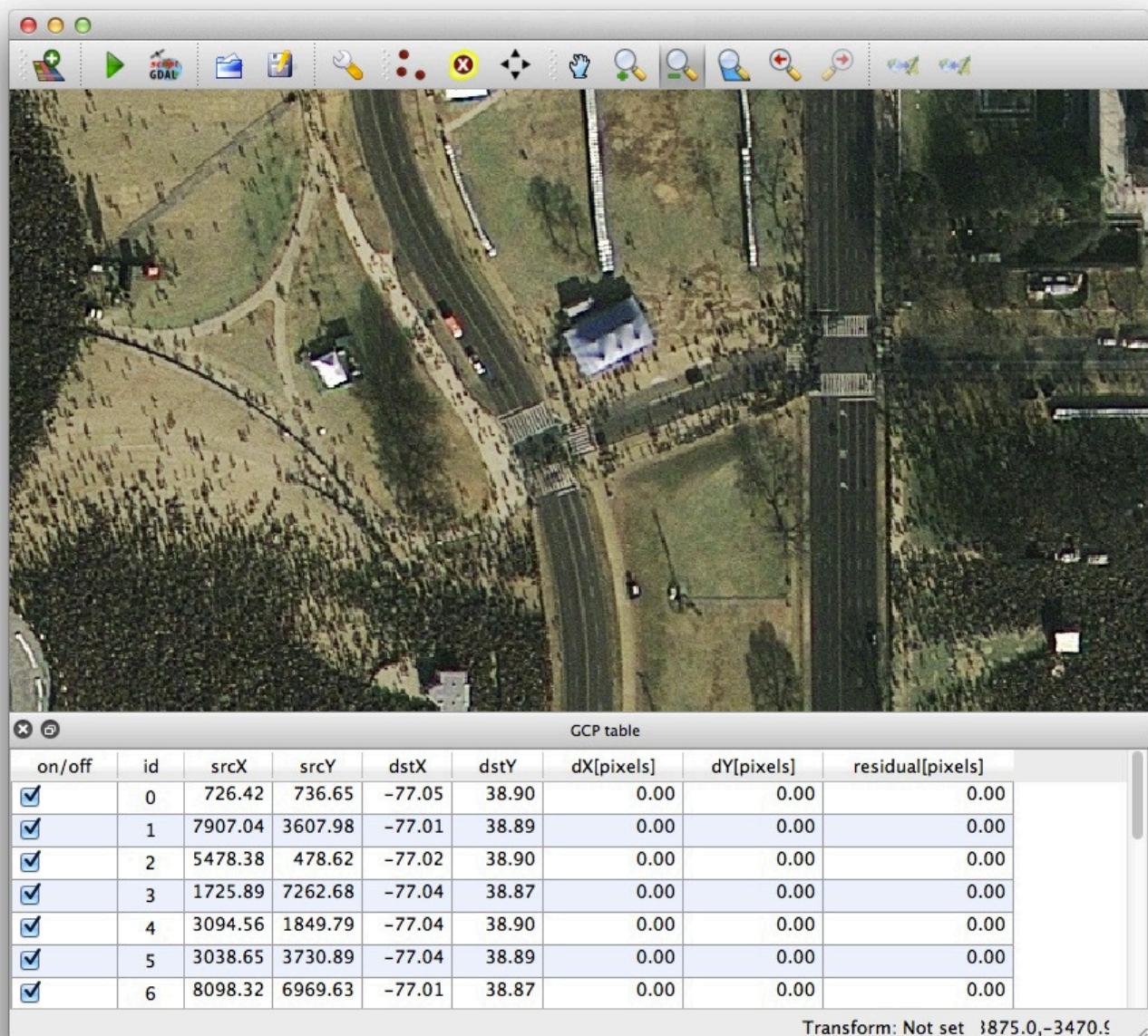
利用[GDAL](#), [QGIS](#)，还有[MapBox Satellite](#)等工具，我们可以对这幅2009年的卫星影像手工配准，然后通过与[MapBox Satellite layer](#)的对比来观察当时的就职典礼出席来宾，发现城市的变化。

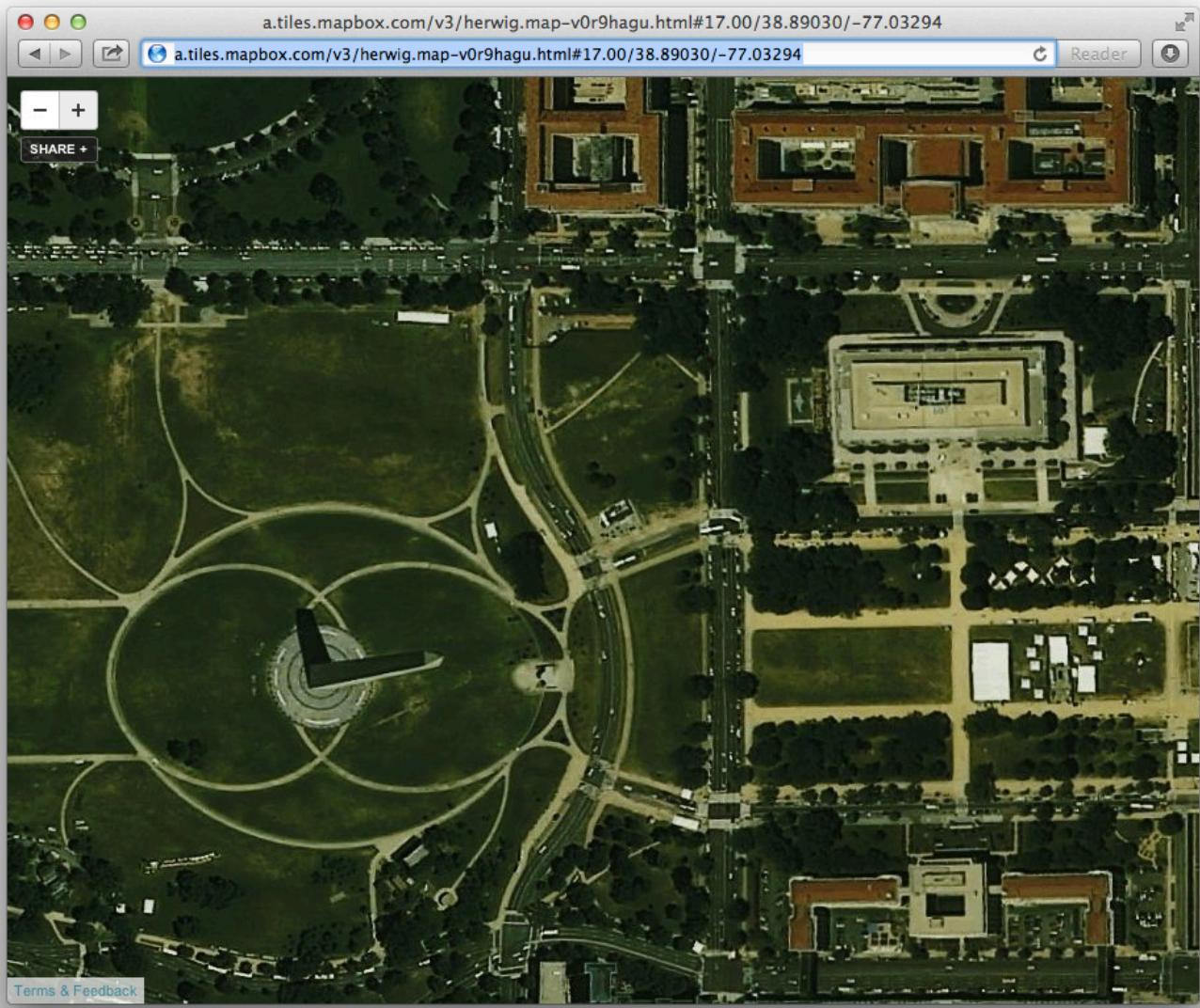
Below, we go over the steps of how to take a non-georeferenced JPEG image and turn it into a geospatial dataset ready for rendering in [TileMill](#) and uploading to MapBox hosting. We use three free, open-source software libraries: [Quantum GIS](#) (These instructions are based off of QGIS version 1.8.0 Lisboa), [GDAL](#), and [TileMill](#).

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（Manual Georeferencing）

1. Open up QGIS and activate the Georeferencer plugin from within the Plugins drop-down menu.
打开QGIS，从Plugins下拉菜单中激活Georeferencer插件。
2. Open raster dataset in Georeferencer window.
在Georeferencer窗口中打开栅格数据集。
3. Log in to your MapBox account and create a new map layer. To create a Satellite base layer, you'll need a [basic account or higher](#).
登录MapBox账户并创建一个新的图层。要使用MapBox的卫星影像图层，你至少需要一个[基本账户](#)。
4. In the Georeferencer window in QGIS, choose a recognizable location on the source image. Select the **Add a Point** tool (Command + A), and add a point on the source image over the location. Here we are using the corner of 15th Street NW and Madison Dr. NW, across from the Washington Monument.
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。
5. Search for the same location on the map shown in your custom MapBox Satellite layer, keeping the point of interest in the center of the screen. Here's a screenshot of the area on the source image.
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

Take a look at the url hash at the end of the MapBox url.

注意URL中最后的部分：

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

The first number after the # is the **zoom level**, the second is **latitude**, and the third is **longitude**.

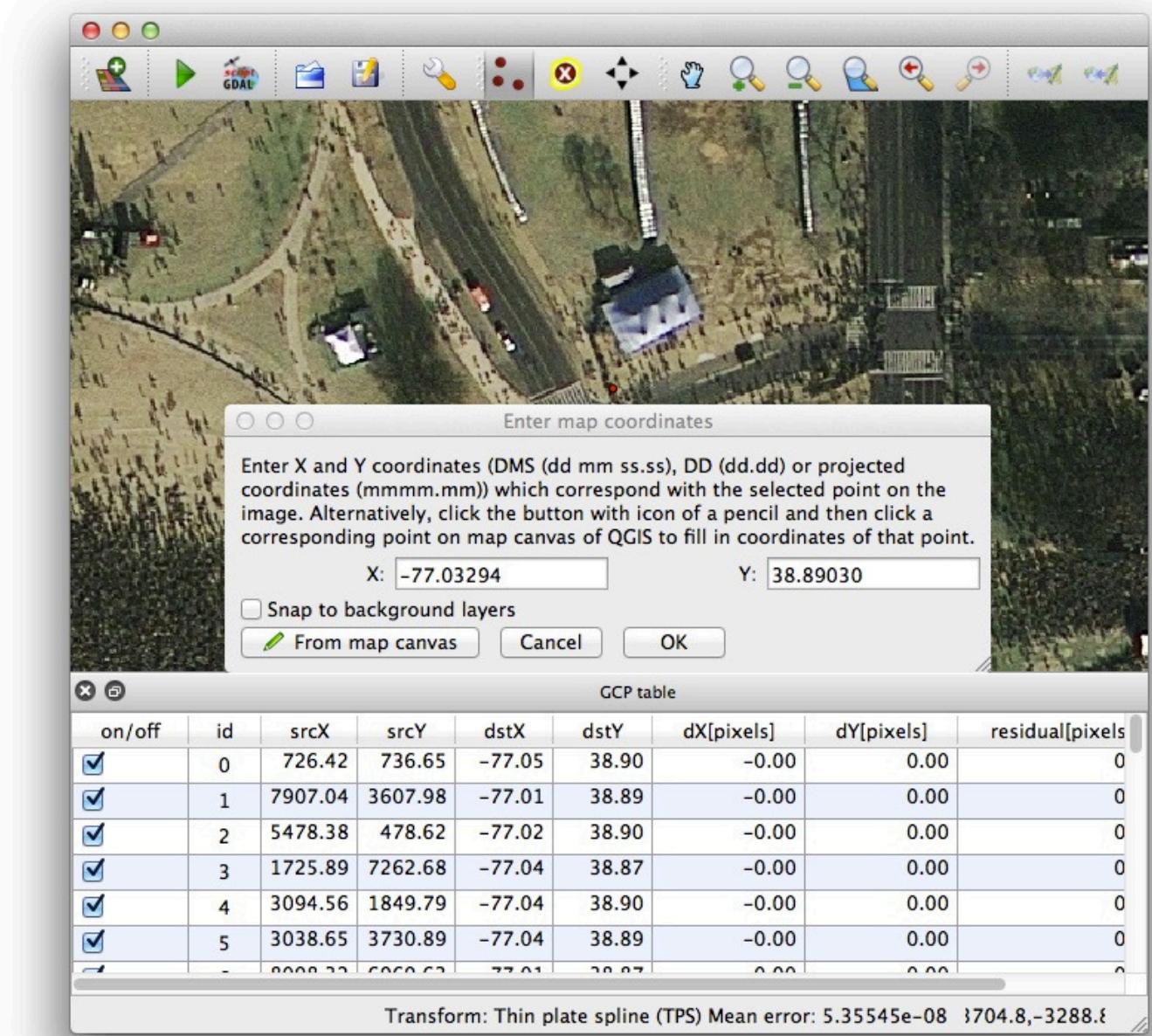
在#之后的第一个数字是缩放级别，第二个是纬度，第三个是经度。

Latitude: 38.89030

Longitude: -77.03294

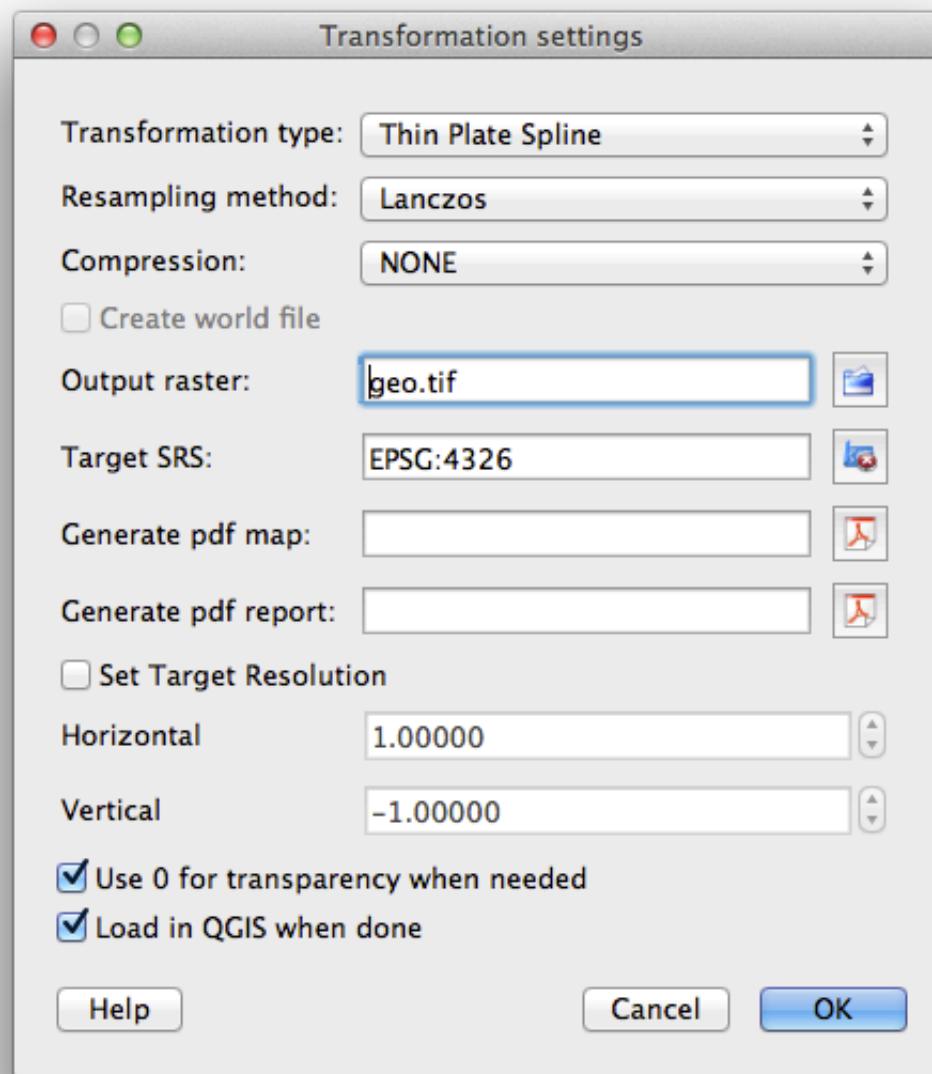
1. Back in QGIS, use the values obtained from the url hash in the “Enter Map Coordinates” dialog. The second number from the URL hash, **latitude**, is the **Y** value; the third number, **longitude**, from the hash is the **X** value.

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. Repeat steps 6–8 until you've added the desired number of control points to the image. A good rule of thumb here is to start with the four corners and work your way inward. To meet the project accuracy requirements, we added a total of 37 ground control points. Be sure to save your ground control points using the “Save GCP Points as” option in the georeferencer plugin. That way, you can reopen the project at a later date to modify points or add additional ones to improve spatial accuracy.
重复第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，逐渐向图片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. You can either perform the georeferencing within QGIS, or select the “Generate GDAL Script” from QGIS. We selected Thin Plate Spline transformation, Lanczos resampling, no compression, and generated a script to modify before running.
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，生成了一段可以修改的脚本。



Here's our final processing script, which incorporates the QGIS-generated ground control points, and my project-specific projection, resampling, and overview settings.

下面就是最终的处理脚本。它是基于QGIS生成的脚本，结合了我们项目中的地图投影、重采样和缩略图等特定配置的版本。

```
#!/bin/bash

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

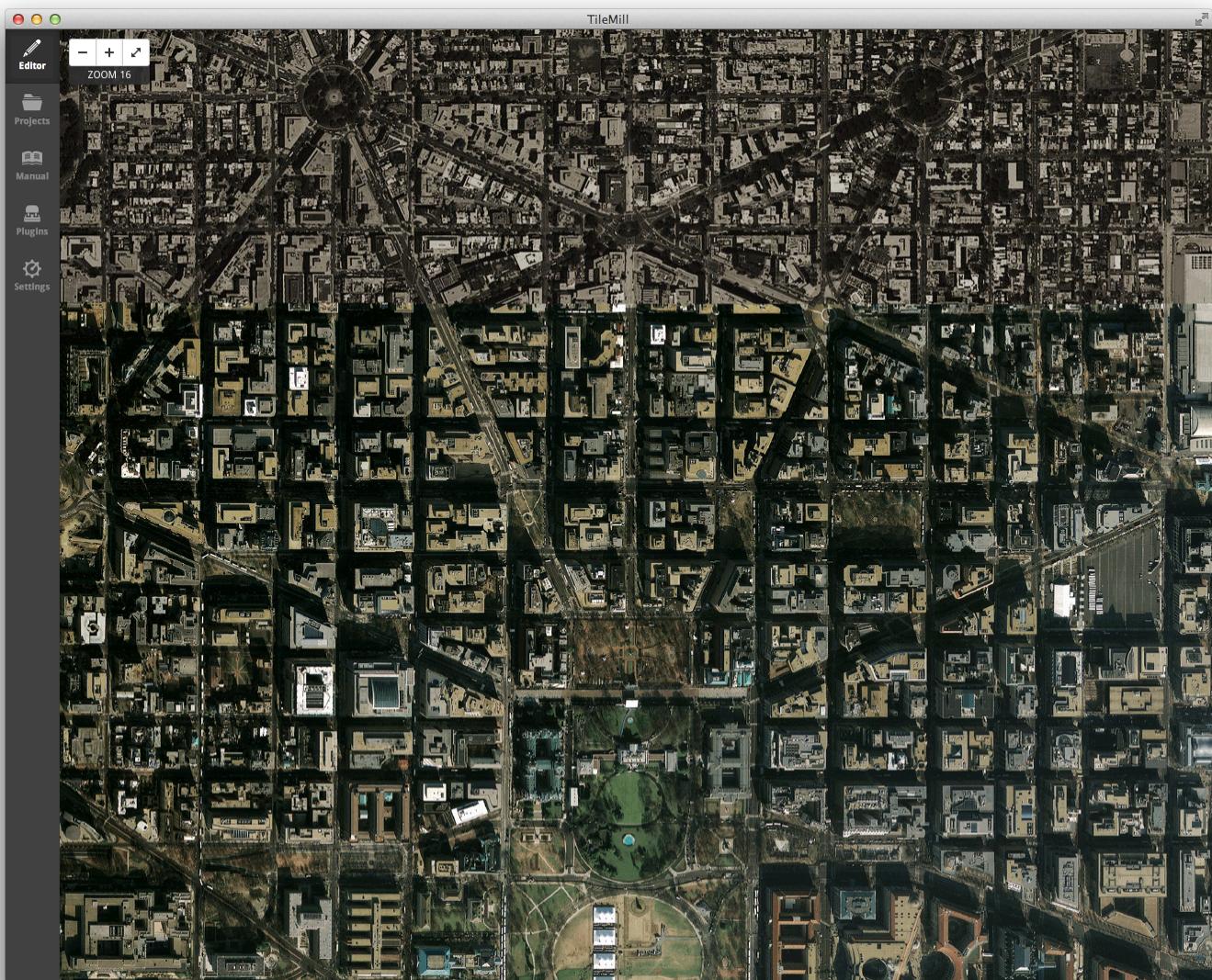
gdal_translate \
    -of GTiff \  
-
```

```
-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
```

The script produces a conventional GeoTIFF, which we can render in [TileMill](#), and upload to MapBox hosting.

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



We can check the spatial accuracy of the georeferencing against our [MapBox Satellite layer](#) using the Reference Layer Plugin from within TileMill.

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 (Colorizing Single-band Raster Data)

Single band raster data traditionally rendered as black and white in TileMill, but it's no longer so black and white.

传统上，单波段的栅格数据在TileMill中会被渲染成黑白的，但它也完全可以不被绘制成黑白分明的样子。



See our blog post on [processing DNB raster data from NASA and NOAA's Suomi NPP spacecraft](#) to create a nighttime lights map, showing lights visible from space at night. Thanks to raster-colorizer, we can now generate the same map with half as many lines of code, in a fraction of the time, by performing all of the false color steps from within TileMill, rather than a [combination of command line tools and virtual rasters \(VRT\)](#).

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地图，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。

To take advantage of the raster-colorizer functionality in TileMill, be sure to set band=1 in the Advanced input area of TileMill's "Add Layer" window.

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（Lights of the Night）

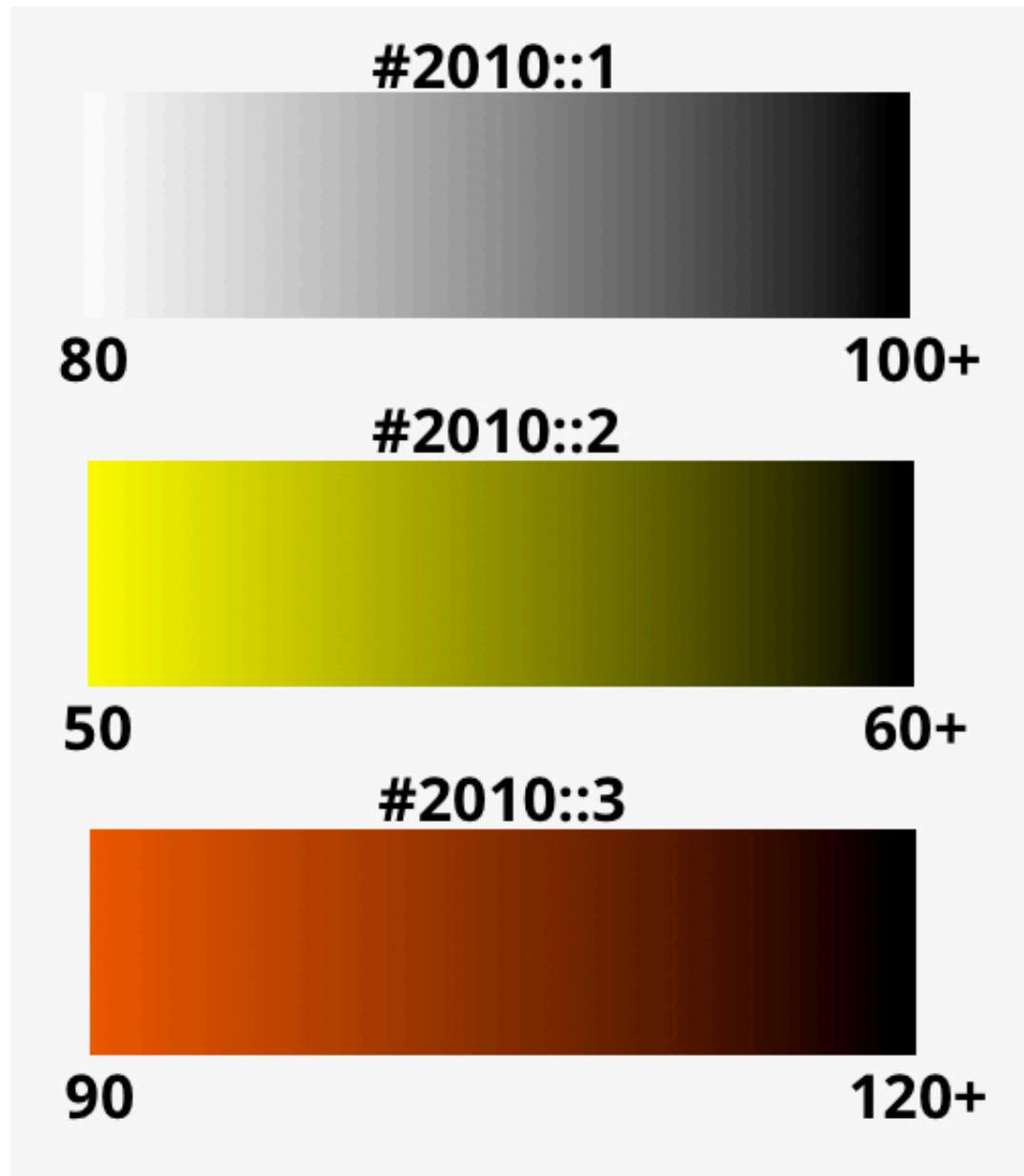
The source data is one band, but we can render it in TileMill as a three band raster using CartoCSS classes, resulting in three versions of the layer rendering on top of one another:

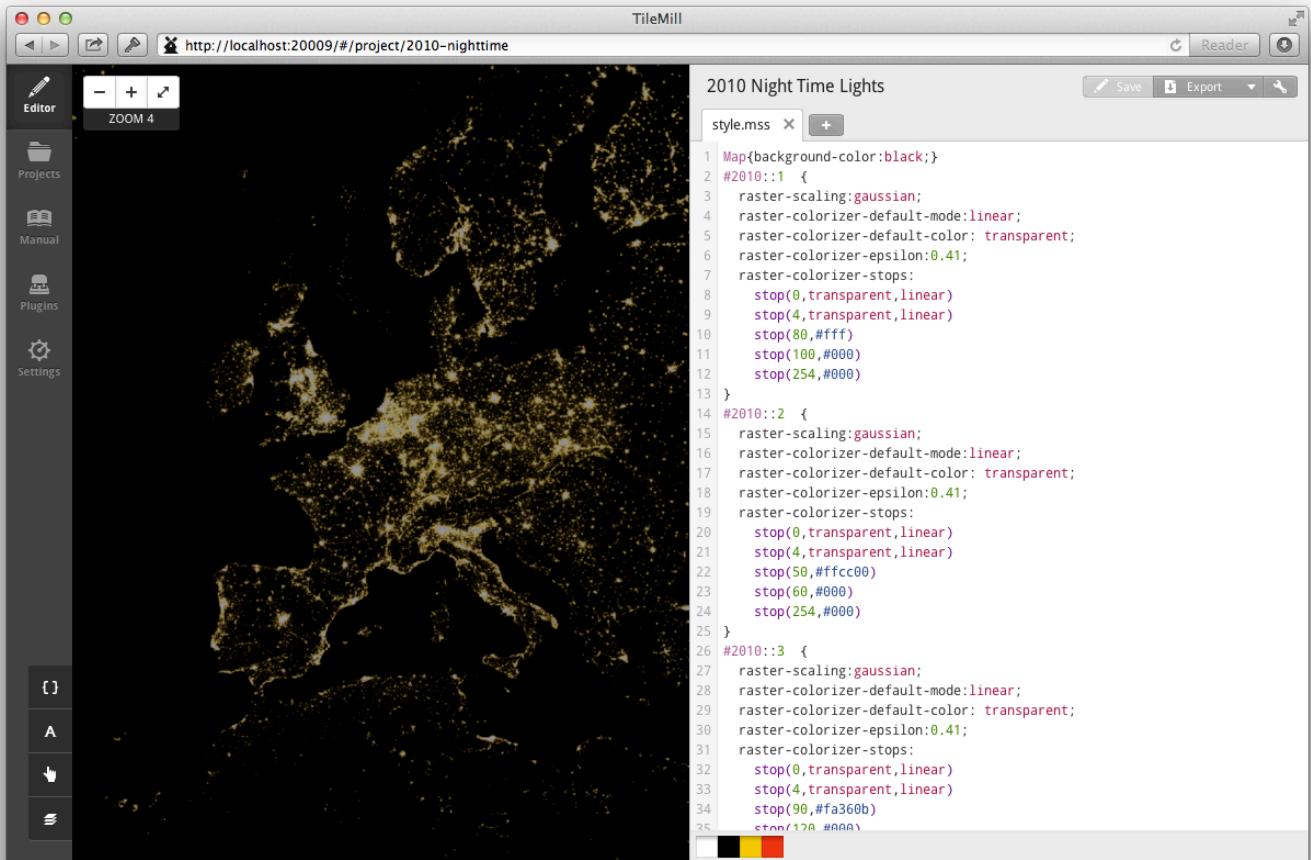
尽管原始数据只有一个波段，但我们可以利用CartoCSS的从属样式能力把它像一幅三波段数据那样去渲染，得到同一个图层的三个版本，相互叠加渲染：

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

Next, use the `raster-colorizer` functionality to color each class a different color to achieve the desired RGB finished product:

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 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（Finished Map）



全色影像的色彩校正（Color Correction of RGB Imagery）

Following a similar process to Single-band colorizing, we can perform color correction for 3-band natural-color RGB aerial or satellite imagery from within TileMill. Performing the color modifications from within TileMill is much easier and offers greater customization than my previous methods offered.

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。

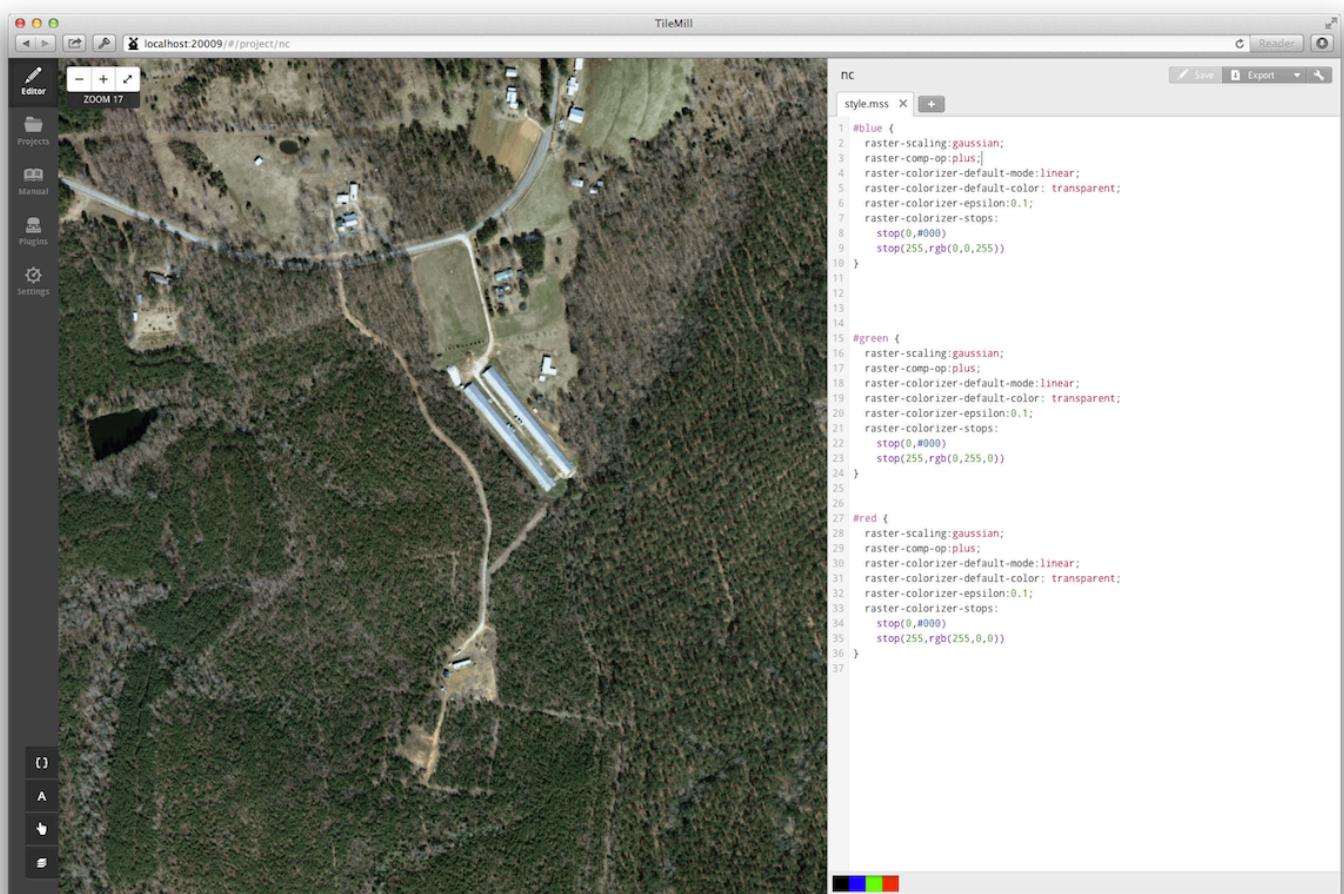
全色影像（RGB Imagery）

Normally, when you load an RGB image as a layer in TileMill the layer displays as natural color. To take advantage of the `raster-colorizer` functionality, we need to add the layer three times, including in the advanced option of band=1 for the red layer, band=2 for the green layer, and band=3 for the blue layer. The `band=` advanced option has TileMill load only the indicated band.

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。

To turn the three layers back into an RGB image, you'll want to use `raster-comp-op: plus;` and `raster-colorizer-default-mode: linear;` for each layer.

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))
}

```

With the layer rendering as an RGB image in TileMill, you can now apply color corrections to each band, simply by modifying the `raster-colorizer-stops`.

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。

A good starting place for color correcting in this manner is to adjust the min, max, and mean values. We found red and green bands looked best when we set the minimum to 20 and maximum to 200; for the blue band I set the minimum value to 40. For the red layer, pixels with values less than or equal to 20 are all registered as the darkest dark elements of the band, and all pixels with values greater than or equal to 200 are registered as the brightest red.

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。



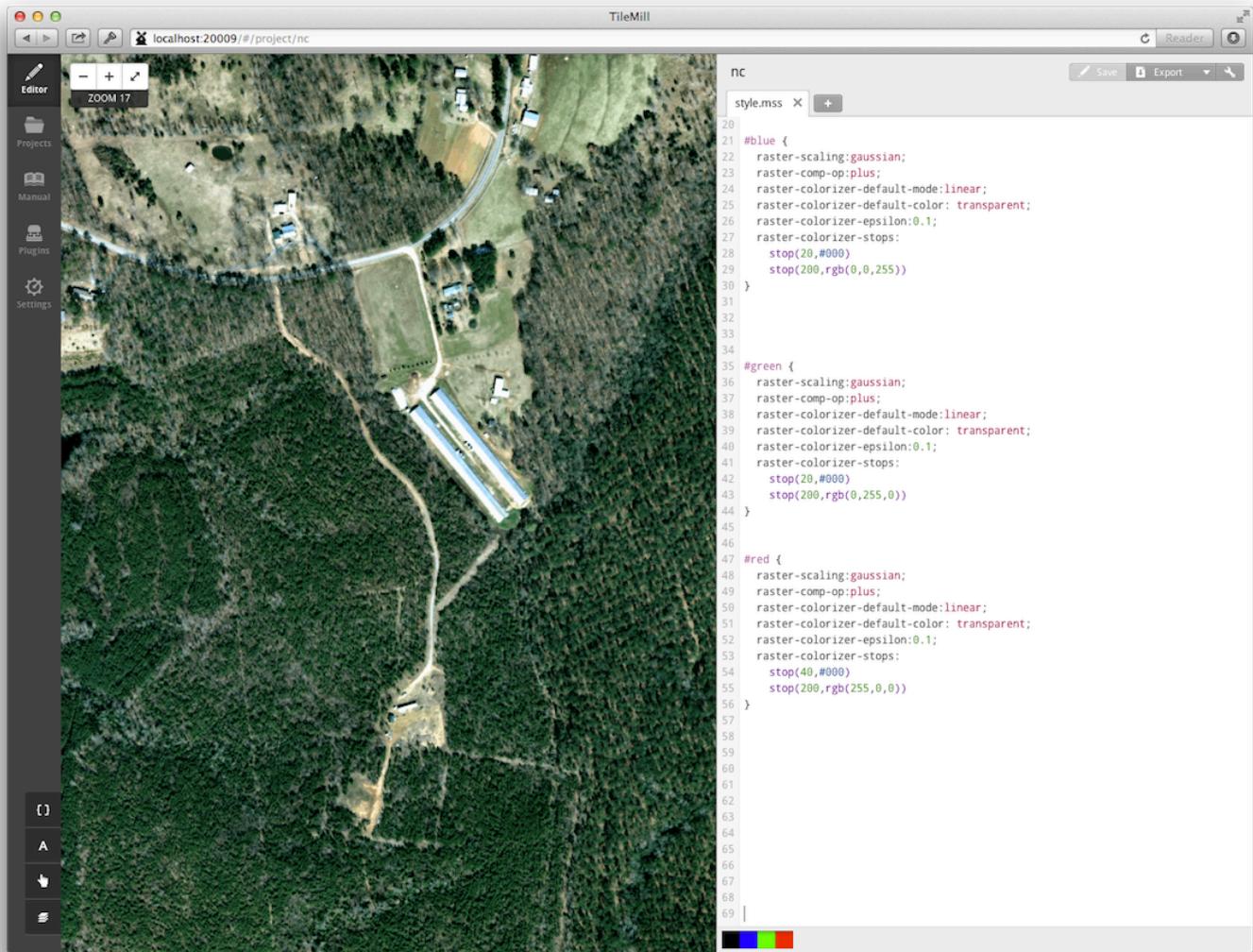
<= 20 200+



<= 20 200+



<= 40 200+



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（Discrete Raster Data）

离散栅格数据：土地覆盖（Discrete Raster Data: Land Cover）

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.

基于上下文的离散栅格数据的制图通常需要多个软件工具相互配合，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，具有特定值的像素会与某种颜色绑定。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（比如同一种植被覆盖的区域）合并使用一种颜色渲染等。

Here we will make a custom land cover map layer from a raster dataset. This guide uses [this one available from the Japan Aerospace Exploration Agency](#). Here is a [direct link to the zip file](#).

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文件的下载地址在[这里](#)。

The only pre-processing required is to reproject the dataset to Google Mercator projection, using an application like gdalwarp. All styling of the raster data can be accomplished from within TileMill using CartoCSS.

预处理阶段唯一需要做的就是将该数据用gdalwarp重投影成Google Mercator投影。后面的制图样式配置全部可以用CartoCSS制图工具完成。

预处理（Pre-processing）

After downloading and uncompressing the GeoTiff data, warp each image to the proper projection as we did to the [Natural Earth GeoTiff](#).

在下载并解压得到GeoTIFF数据之后，需要将其重投影，方法可以参考[这里](#)。

Warp all the images and move the reprojected ones to a directory called target (using Terminal):

下面这段bash脚本就是将所有位于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 (Build a Virtual Dataset)

Since TileMill natively supports [GDAL's Virtual Raster \(VRT\) format](#), we can take advantage of a VRT rather than creating a new GeoTIFF mosaic. [gdalbuildvrt](#) creates a single XML file from the source images that is read as a single mosaic image in TileMill. **Make sure you use absolute paths for the source images when you use gdalbuildvrt.**

很多CartoCSS制图工具（例如TileMill）原生支持[GDAL虚拟栅格格式](#)。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 (Importing and Styling in TileMill)

While in the TileMill “Add Layer” window, input band=1 in the Advanced input area. If you omit this step, the colorizer will not function properly.

在添加图层的时候，注意要加上band=1，否则着色器会工作不正常。

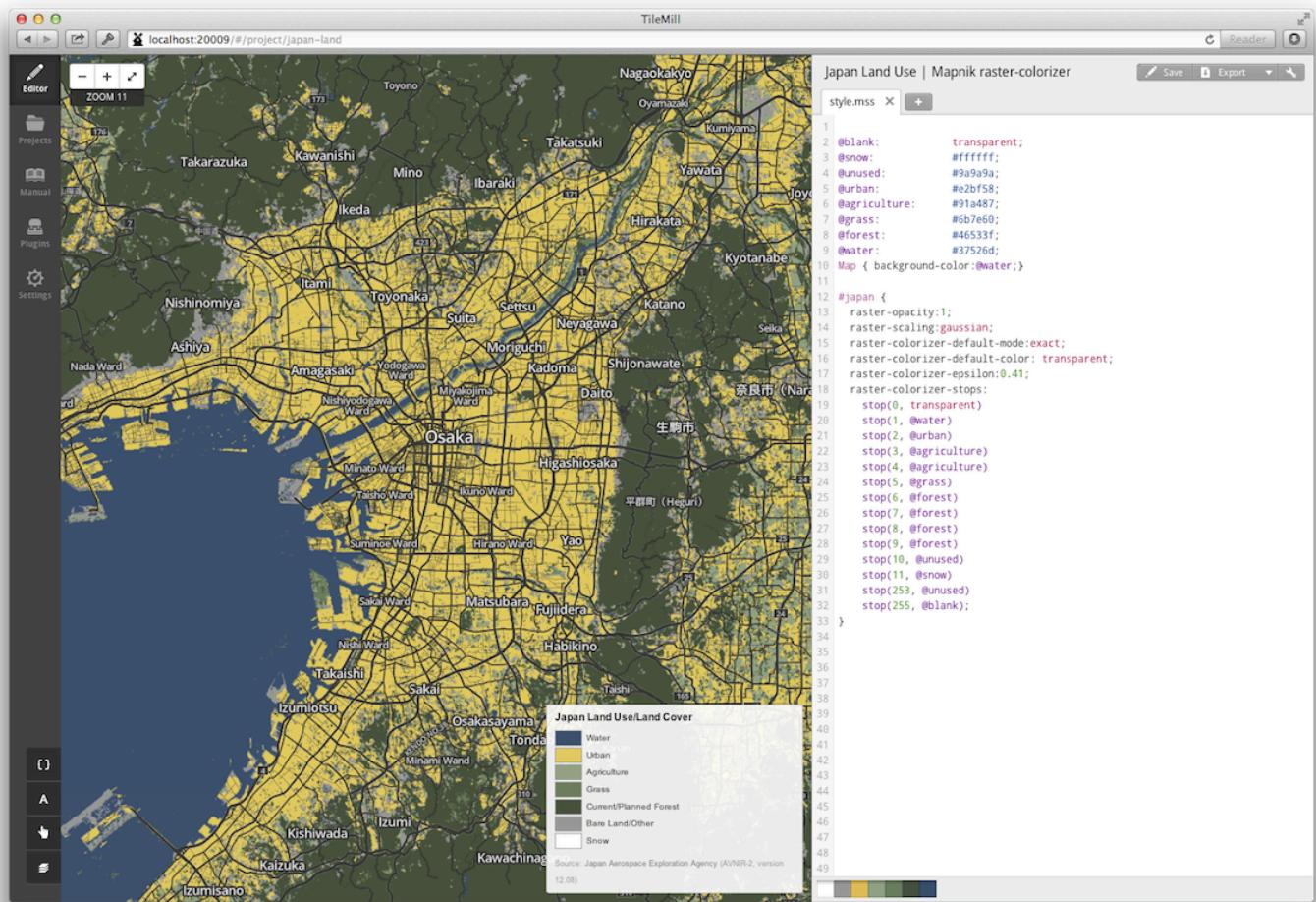
Since we're using a land cover GeoTIFF with specific pixel values mapping directly to land cover classifications we want to use `raster-colorizer-default-mode: exact` meaning stops will map to exact pixel values, and no other color values will be assigned through interpolation.

由于我们使用的土地覆盖GeoTIFF数据中的每个像素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。

Now all that remains is to translate the land use data key into CartoCSS style rules, using the `raster-colorizer stop` syntax:

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。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)
}

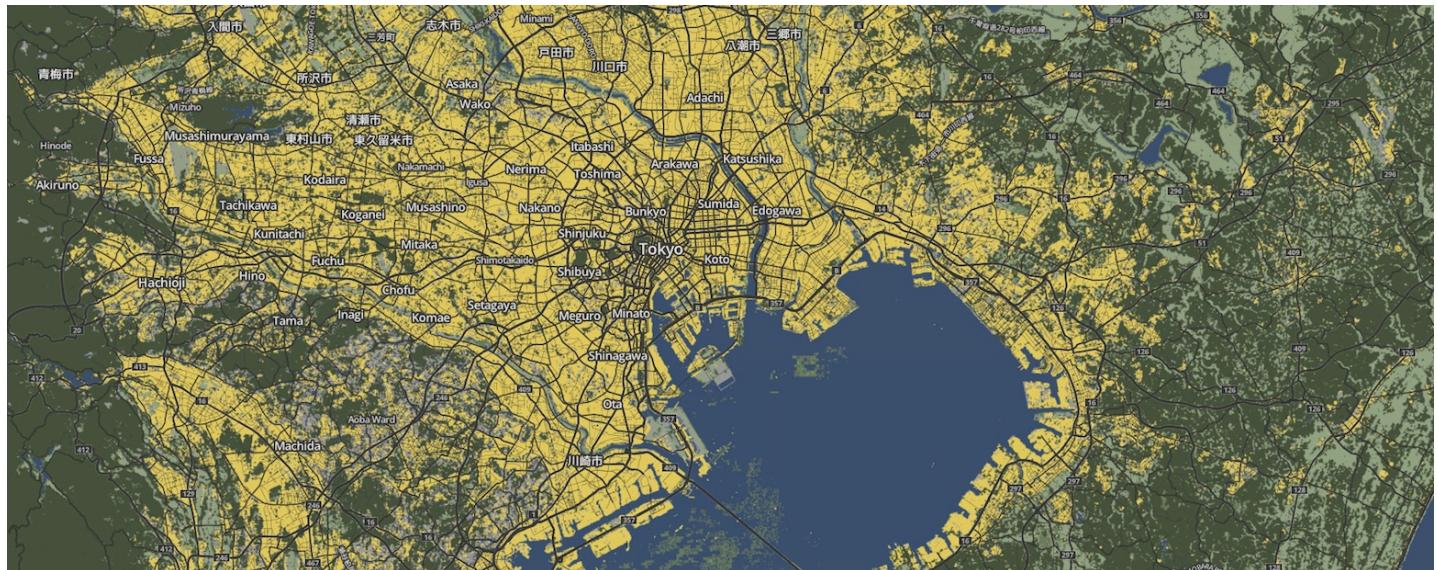
```

```

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（Finished Map）



地形数据制图（Working with terrain data）

To get good-looking terrain maps, we usually combine several types of visualizations generated by the [GDAL](#) DEM utilities.

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

DEM数据源（Digital Elevation Model sources）

There are many different data formats for storing and working with digital elevation models. In our examples we'll be working with geotiffs. Here are some examples of high quality free datasets that are available in geotiff format:

用于存储数字高程模型的数据格式有很多种。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

Data collected from NASA's [Shuttle Radar Topography Mission](#) is a high quality source of elevation data covering much of the globe. It is available free from NASA directly, however we recommend working with [CGIAR's cleaned up version](#), which is also free.

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的清理后版本](#)，它也同样是免费的。

ASTER

Aster is another global DEM datasource. It has better coverage of the earth's surface than SRTM, and is slightly higher-resolution, but contains more errors than CGIAR's clean SRTM set. Errors are usually in the form of spikes or pits, and can be significant.

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，而且分辨率也更高一些。但Aster数据中的误差比CGIAR清理后的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。

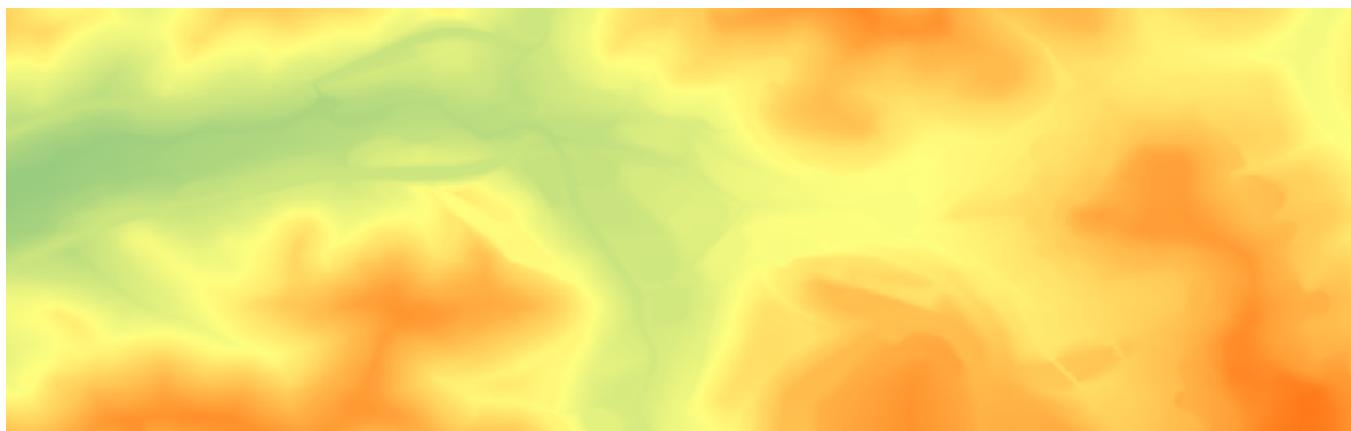
USGS NED

The US Geological Survey publishes a [National Elevation Dataset](#) for the United States. It high resolution and frequently updated from a variety of sources.

美国地质调查局也发布了一个覆盖美国的[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（[Types of Visualizations](#)）

彩色地形图（也叫分层设色）（[Color-relief \(or hypsometric tint\)](#)）



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）（**Hillshade (or shaded relief)**）

Hillshade visualization analyzes the digital elevation model to simulate a 3-dimensional terrain. The effect of sunlight on topography is not necessarily accurate but gives a good approximation of the terrain.

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（**Slope**）

Slope visualization assigns a color to a pixel based on the difference in elevation between it and the pixels around it. We use it to make steep hills stand out and enhance the look of our terrain.

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可视化效果。

使用**GDAL-DEM**实现地形可视化（**Visualizations with GDAL-DEM**）

对数据重投影（**Reprojecting the Data**）

Chances are your DEM datasource will not come in the Google Mercator projection we need - for example, SRTM comes as [WGS84](#) (EPSG:4326) and USGS NED comes as [NAD83](#) (EPSG:4269).

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。

For our example we'll be working with some NED data of the District of Columbia area. The following command will reproject the file to the proper projection - see [Reprojecting a GeoTIFF for TileMill](#) for more detailed information.

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
```

In the case of reprojecting elevation data, the `-r bilinear` option is important because other resampling methods tend to produce odd stripes or grids in the resulting image.

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奇怪的条纹或网格。

制作山体阴影图（**Creating hillshades**）

Run this command to generate the shaded relief image:

执行以下命令生成地形阴影图：

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

The `-co compress=lzw` option will compress the TIFF. If you're not concerned about disk space you can leave that part out. If you are using GDAL 1.8.0 or later you may also want to add the option `-compute_edges` in order to avoid a black pixel border around the edge of the image.

`-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，那么就可以不加压缩参数。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填充黑色像素值。

Our output looks like this:

输出结果如下图：



Note: If there is not a 1:1 relation between your vertical and horizontal units, you will want to use the `-s` (scale) option to indicate the difference. Since Google Mercator X & Y units are meters, and NED elevation is also stored in meters this wasn't necessary for our example. If your elevation data is stored in feet, you might do this instead (since there are approximately 3.28 feet in 1 meter):

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。但如果你的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
```

制作彩色地形图（Generating color-relief）

Before you can create a color-relief image, you need to decide what colors you want to assign to different elevations. This color configuration is created and stored in a specially formatted plain text file. Each line in the file should have four numbers - an elevation, and an RGB triplet of numbers from 0 to 255. To figure out the proper RGB values of a color, you can use the color selection tool of any image editor, or an online tool such as ColorPicker.com.

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，你可以使用任意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。

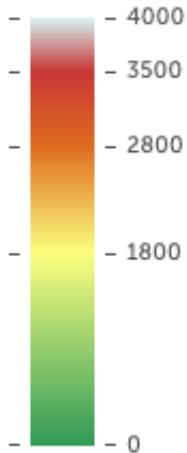
Here is our example, saved to a file called ramp.txt:

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
```

The above numbers define a gradient that will blend 5 colors over 4000 units of elevation. (Our units are meters, but we don't need to specify that in the file.) They will translate to a color-to-elevation relationship that looks like this:

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：



To apply this color ramp to the DEM, use the `gdaldem color-relief` command:

要在DEM数据上应用这个色带，需要使用`gdaldem color-relief`命令：

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

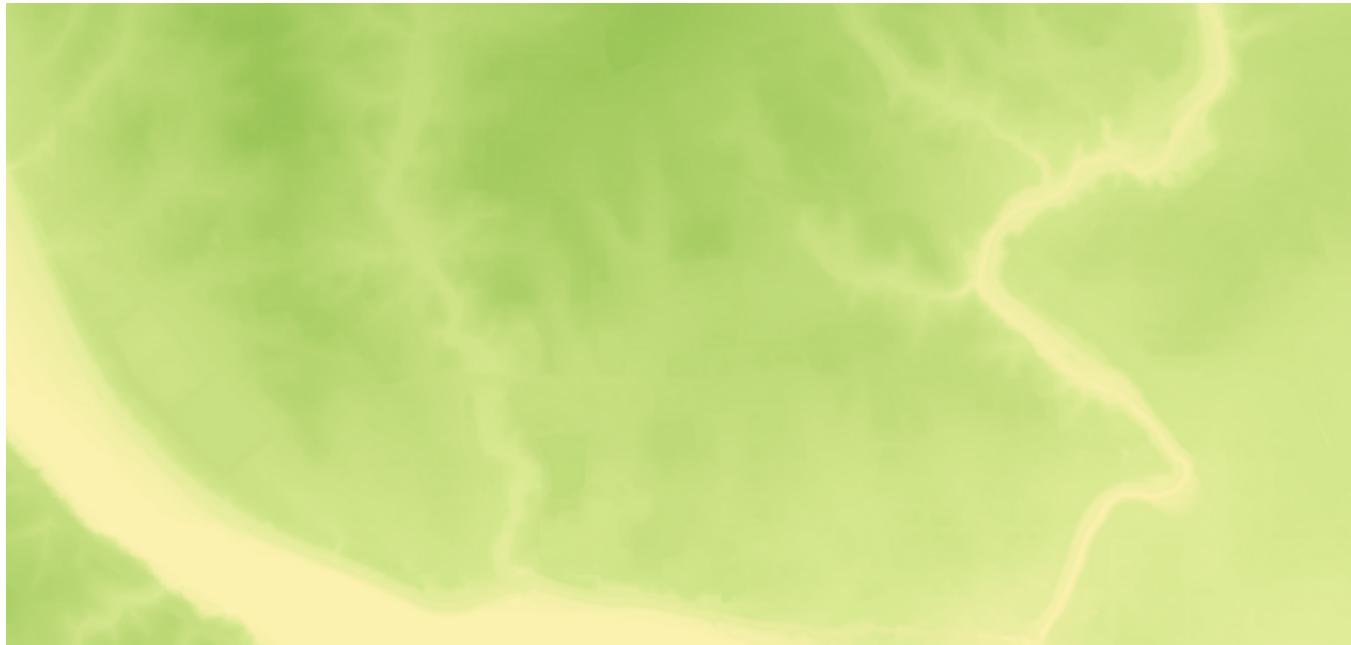
The area you are working with may not have such a wide range of elevations. You can check the minimum & maximum values of your DEM to adjust your ramp style accordingly. The `gdalinfo` command can tell us the range of values in the geotiff:

但在你选定要制图的目标区域中，未必会有这么大跨度的高程落差。你可以根据目标区域中地形的海拔最高与最低值来调整色彩配置。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
```

Among other things, this will tell you the minimum, maximum, and mean values of your elevation data to help you make decisions about how to choose your colors. With elevations tweaked for our area around DC, this is what we get:

抛开其它信息，这个命令可以告诉你的高程数据中的最大值、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（Generating slope shading）

With the gdaldem tools, generating a slope visualization is a two step process.

利用gdaldem工具，制作坡度可视化效果包含两个步骤。

First, we generate a slope tif where each pixel value is a degree, from 0 to 90, representing the slope of a given piece of land.

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
```

The same note about horizontal and vertical scale as hillshades applies for slope.

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。

The tif output by gdaldem slope has no color values assigned to pixels, but that can be achieved by feeding it through gdaldem color-relief with an appropriate ramp file.

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上色。

Create a file called ‘slope-ramp.txt’ containing these two lines:

新建一个名为slope-ramp.txt的文件，将以下两行拷进去并保存：

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

This file can then be referenced by the color-relief command to display white where there is a slope of 0° (ie, flat) and display black where there is a 90° cliff (with angles in-between being various shades of grey). The command to do this is:

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
```

Which gives us:

然后得到的结果如下图所示：



合并最终结果 (Combining the final result in TileMill)

To combine these geotiffs in TileMill, we'll make use of the ‘multiply’ image blending mode using the `raster-comp-op` property, which stands for “compositing operation”. Assuming you have added your three geotiffs as layers with appropriate IDs, the following style will work. You can adjust the opacity of the hillshade and slope layers to tweak the style.

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数据集加入了制图工具，得到三个图层，并分别赋予了合适的图层ID，那么就可以应用下面的样式配图了。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;
    // note: in TileMill 0.9.x and earlier this is called raster-mode
    raster-comp-op: multiply;
}

#hill-shade { raster-opacity: 0.6; }

#slope-shade { raster-opacity: 0.4; }

```

The end result is something like this, ready to be combined with your vector data:

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（Raster performance）

If your rasters are many MBs in size, then an important optimization is to build image pyramids, or overviews. This can be done with the gdaladdo tool or in QGIS. Building overviews does not impact the resolution of the original image, but it rather inserts new reduced resolution image references inside the original file that will be used in place of the full resolution data at low zoom levels. Any processing of the original image will likely drop any internal overviews previously built, so make sure to rebuild them if needed. You can use the gdalinfo tool to check if your tiff has overviews by ensuring the output has the ‘Overviews’ keyword reported for each band.

如果你的栅格数据有很多MB那么大（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来完成。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工具检查你的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。

实例：制作专题地图

译注：[原文地址](#)

This is a 10 minute walk through showing how to generate heat maps in QGIS and then display them in TileMill.

这是一个10分钟的实例教程，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。

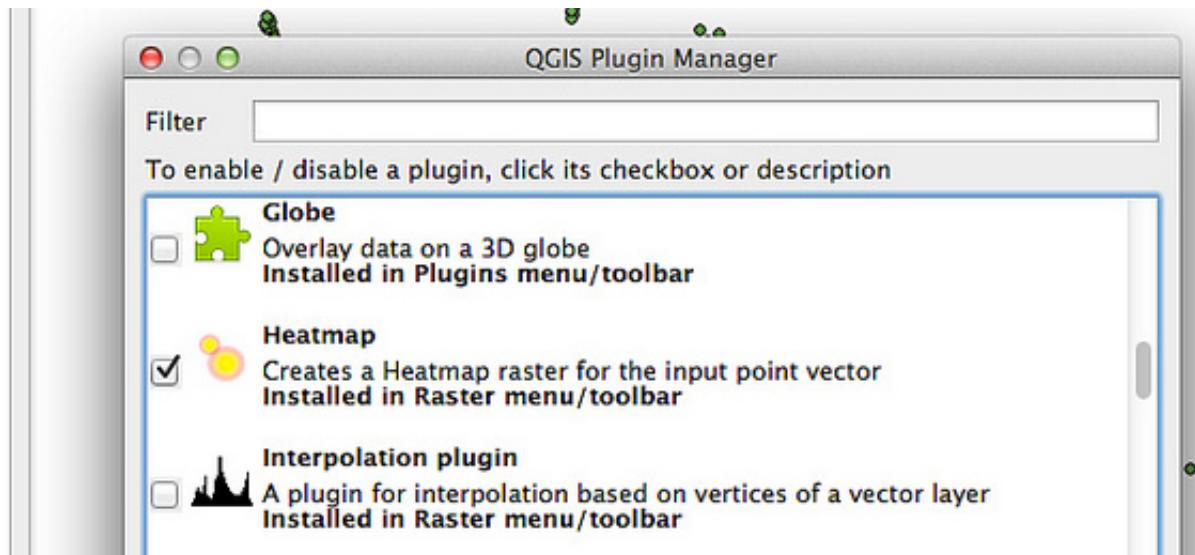
使用QGIS（Working with QGIS）

Make sure you are running at least QGIS >= 1.9. At the time of writing, this is the development release. Mac users can find it at [kyngchaos](#).

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。

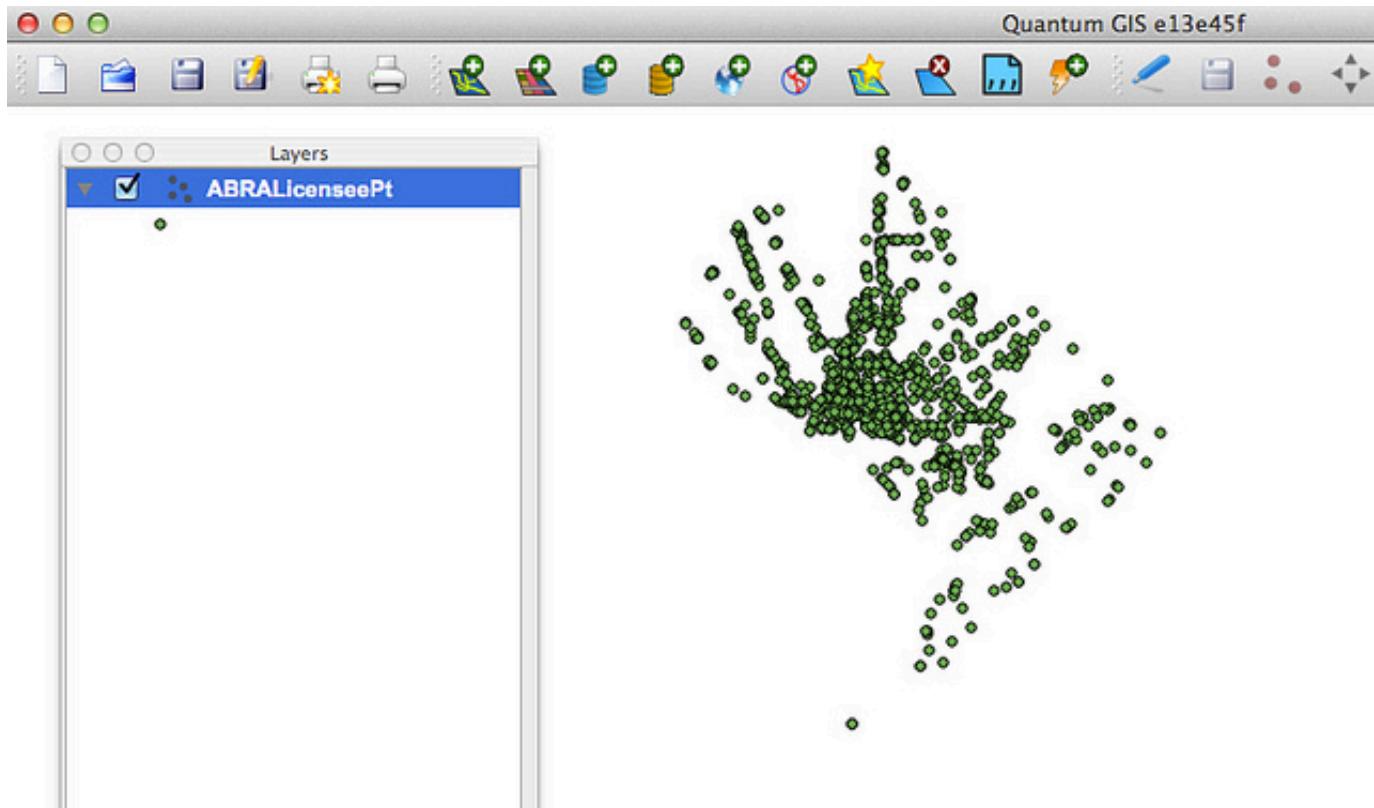
Next you'll need to enable the Heatmap Plugin. Open the pane below by selecting Plugins > Manage plugins. In the following pane, check the box for the Heatmap Plugin. Note: *This plugin is only available for QGIS 1.9 and above.* You may need to restart QGIS for these changes to take effect.

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。



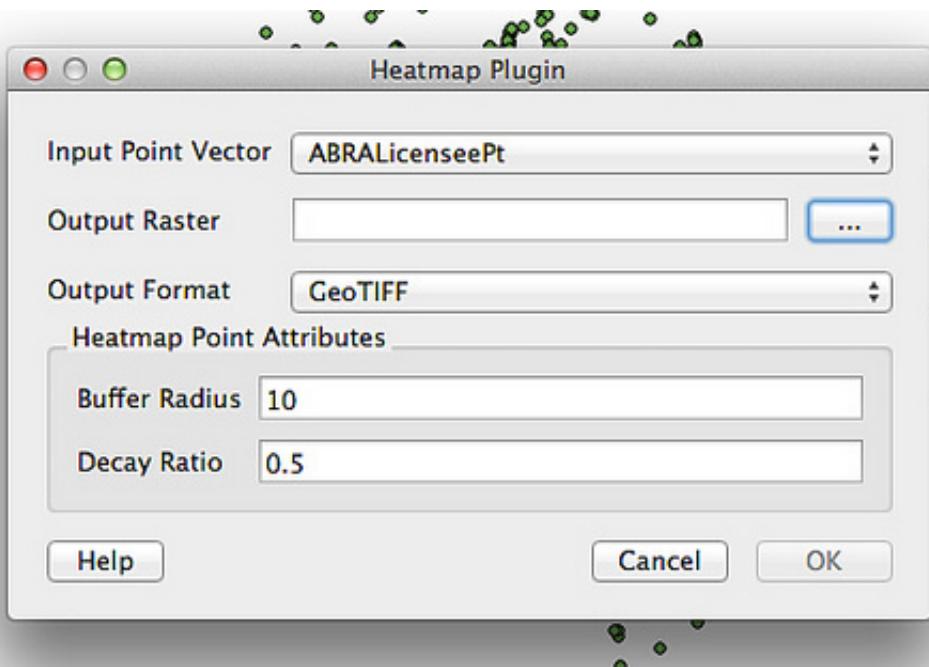
Now add some point data. Data below is from the Washington, DC Alcoholic Beverage Regulation Administration, made available by DC's open data portal [data.dc.gov](#). It can be downloaded [here](#).

现在需要加入一些点数据。下面这个数据集来自华盛顿特区酒精饮料管理局的开放数据门户[data.dc.gov](#)。下载地址在[这里](#)。



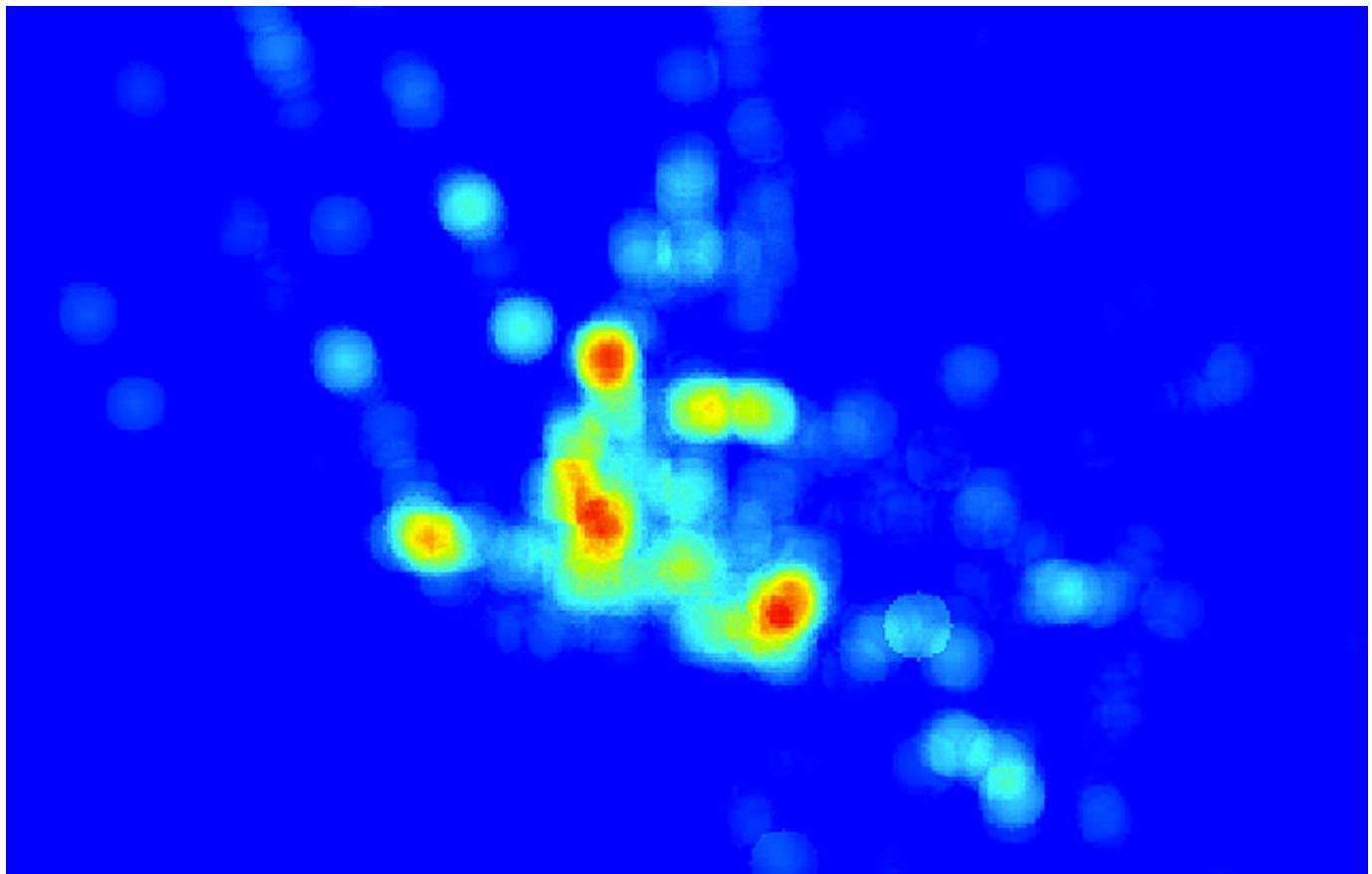
Now go to Raster > Heatmap > Heatmap. You should see a dialog box like this:

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



Specify an output filepath (with a .tif extension) and leave the default settings for now. Hit OK to generate your heat map. A big gray rectangle will be added to the project as a new layer. To see the different color values, open the properties pane for your new layer. To get a quick and dirty visualization of the different values we'll use one of the QGIS preset color schemes. Under the Style tab, set the Colormap drop down menu to 'Pseudocolor' and hit ok. You should see a map like this:

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，它是一张巨大的灰度矩形。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然后点击OK，然后可以得到一幅如下效果的地图：



Now that you have your map, play around with the colors and other parameters. Defaults are easy, but rarely expose data well. The Heatmap dialog has a great help section that painlessly explains what a ‘spatial buffer’ and ‘decay ratio’ are. See an [earlier post](#) on working with raster data to come up with your own custom color scheme. If you want to render your map to tiles with TileMill, the GeoTIFF default is one you don’t have to change.

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对话框的帮助中，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。如果要在TileMill中渲染地图并制作瓦片，那么请保持GeoTIFF格式不变。

To render in TileMill, open a new project and load your .tiff as a new layer. Get rid of the #countries layer, and change the map background so that your carto looks like this:

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;  
}
```

Your map should appear. Just hit upload in the [export](#) menu to start sharing your map online. For more info on working with raster data, read docs on [reprojecting GeoTIFFs](#) and [working with terrain data](#). If you want more inspiration, check out some of previous work like AJ's map of the [world population](#).

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[世界人口图](#)可以得到更多灵感。

用TileMill实现伪装热图（Faking it with TileMill）

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更加强化的和饱和的颜色值。

This example uses the original shapefile with just a few lines of CartoCSS:

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

Note: This code also omits any locations listed as 'Retailer B' to get rid of grocery stores to better show where bars and nightlife are in DC.

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。

This approach has the added benefit of retaining the exact locations of individual dots at higher zoom levels, as well as the possibility of feature-specific interactivity. For further styling tips, learn more about advanced map design.

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。

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 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` (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)

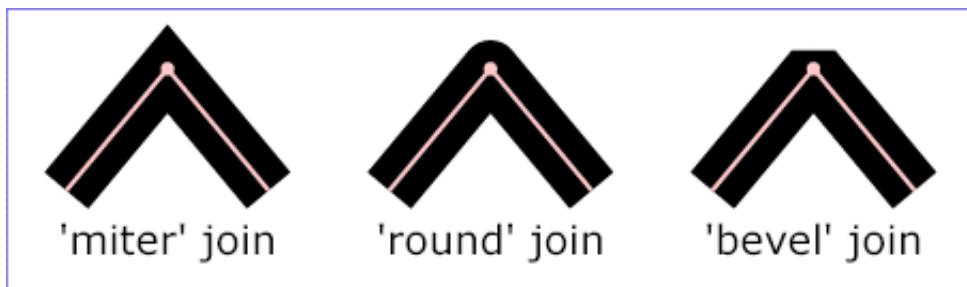
说明：设置线要素的透明度。

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。



图片来源：www.w3.org

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 γ 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 γ 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-Peucker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: 设置在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会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。

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);
}
```