

TikZ & PGF Tutorials

Organized by Haocheng Xia

2022 年 3 月 10 日

目录

第一部分 教程和指南	1
第一章 教程: 给卡尔学生们的一幅图	3
1.1 问题描述	3
1.2 设置环境	4
1.2.1 在 L ^A T _E X 中设置环境	4
1.2.2 在普通 T _E X 中设置环境	5
1.2.3 在 ConT _E Xt 中设置环境	6
1.3 构建直线路径	7
1.4 构建曲线路径	7
1.5 构建圆形路径	8
1.6 构建矩形路径	9
1.7 构造网格路径	10
1.8 添加一点样式	10
1.9 画图选项	12
1.10 构造弧线路径	12
1.11 裁剪路径	14
1.12 构造抛物线和正弦路径	15
1.13 轮廓与填充	16
1.14 阴影	17
1.15 指定坐标	18
1.16 路径相交	20
1.17 添加箭头	21
1.18 作用域	23
1.19 变换	24

1.20 重复: For 循环	25
1.21 添加文本	27
1.22 再探角度	33
第二章 教程: 哈根的 Petri-Net	35
2.1 问题描述	36
2.2 设置环境	36
2.3 节点简介	36
2.4 使用 At 语法放置节点	36
2.5 节点大小	36
2.6 命名节点	36
2.7 使用相对位置放置节点	36
2.8 为节点添加标签	36
2.9 连接节点	36
2.10 添加波浪线和多行文本	36
2.11 使用图层: 矩形背景	36
2.12 完整代码	36

第一部分

教程和指南

This part is written by Till Tantau in English and translated by Haocheng Xia.

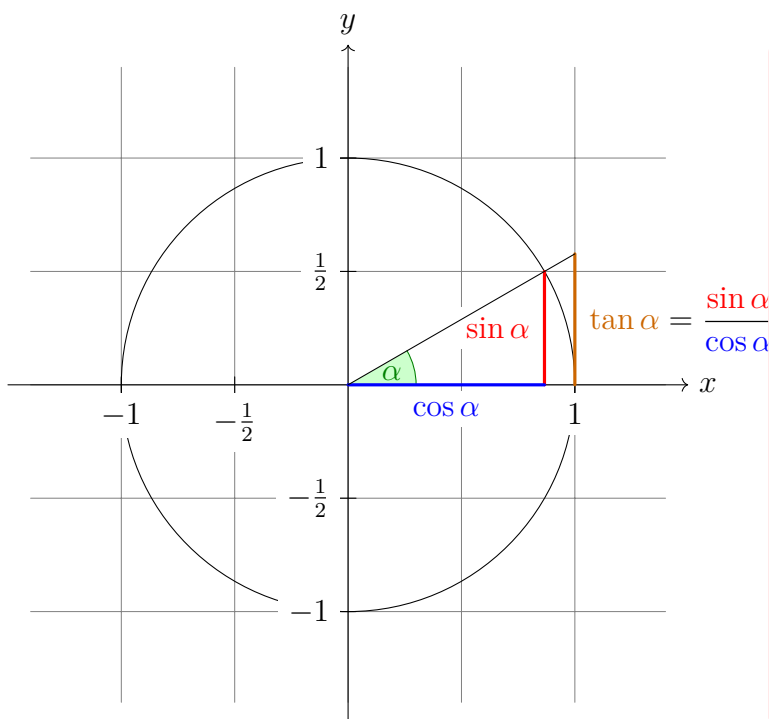
第一章 教程：给卡尔学生们的一幅图

本教程是为 TikZ 的新用户准备的. 它并没有详尽地介绍 TikZ 的所有功能, 只是介绍了那些你可能马上要用到的功能. 卡尔是一名高中数学和化学教师. 他曾经在他出的练习题和试卷中使用 L^AT_EX 的 `{picture}` 环境创建图形. 虽然结果是可以接受的, 但创建图形往往是一个漫长的过程. 而且, 往往会出现线条角度稍有偏差的问题. 而圆似乎也很画对. 当然, 他的学生并不关心线的角度是不是完全正确, 他们只是觉得卡尔的考试太难了. 但卡尔对此一直都不满意.

卡尔的儿子, 对这个结果更不满意 (毕竟他不用参加考试). 他告诉卡尔, 他有点想试试一个用于创建图形的新软件包. 但令人困惑的是, 这个软件包似乎有两个名字. 首先, 卡尔不得不下载并安装一个名为 `pgf` 的软件包. 然后, 事实证明, 在这个包里还有一个叫 TikZ 的包, 它应该代表 "TikZ ist kein Zeichenprogramm". 卡尔发现这一切都有点奇怪, TikZ 似乎没有完成他所要的事情. 然而, 使用 `gnu` 软件已经有一段时间了, 而且 "`gnu` 不是 `Unix`", 一切都似乎还有希望. 他的儿子向他保证, TikZ 的名字只是为了警告人们, TikZ 不是一个通过鼠标或平板来画图的程序. 相反, 它更像是一种 "图形语言".

1.1 问题描述

卡尔想为他的学生在下一份练习题上放一个图形. 他目前正在教他的学生正弦和余弦的知识. 他想得到的是像这样的东西 (理想情况下).



例子中的 α 角是 30° (弧度为 $\pi/6$). α 的正弦 ($\sin \alpha$), 表示为红线的高度,

$$\sin \alpha = 1/2.$$

通过毕达哥拉斯定理可得 $\cos^2 \alpha + \sin^2 \alpha = 1$. 因此蓝线的长度, 即 α 的余弦 ($\cos \alpha$), 为

$$\cos \alpha = \sqrt{1 - 1/4} = \frac{1}{2}\sqrt{3}.$$

而 $\tan \alpha$ 可以用橙线的高度表示

$$\tan \alpha = \frac{\sin \alpha}{\cos \alpha} = 1/\sqrt{3}.$$

1.2 设置环境

在 TikZ 中, 要画一幅图, 在开头, 你需要告诉 $\text{T}_\text{E}\text{X}$ 或 $\text{L}_\text{A}\text{T}_\text{E}\text{X}$ 你想开始绘图. 在 $\text{L}_\text{A}\text{T}_\text{E}\text{X}$ 中, 这是用环境 `{tikzpicture}` 来实现的, 而在普通 $\text{T}_\text{E}\text{X}$ 中, 你只需用 `\tikzpicture` 来开始绘图, `\endtikzpicture` 来结束绘图.

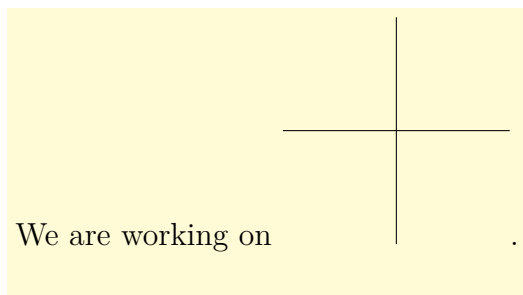
1.2.1 在 $\text{L}_\text{A}\text{T}_\text{E}\text{X}$ 中设置环境

卡尔是一个 $\text{L}_\text{A}\text{T}_\text{E}\text{X}$ 用户, 因此他的文件设置如下.

```
%\documentclass{article} % say
%\usepackage{tikz}
%\begin{document}
%We are working on
\begin{tikzpicture}
  \draw (-1.5,0) - (1.5,0);
  \draw (0,-1.5) - (0,1.5);
```

```
\end{tikzpicture}.\n%\\end{document}
```

当上述命令执行时, 即通过 `pdflatex` 或通过 `latex` 后的 `dvips` 运行. 所产生的结果将包含像这样的东西.



```
We are working on\n\\begin{tikzpicture}\n\\draw (-1.5,0) -- (1.5,0);\n\\draw (0,-1.5) -- (0,1.5);\n\\end{tikzpicture}.
```

诚然, 还不是很全面, 但我们确实建立了轴线. 也不完全是, 但我们已经画出了构成轴线的线条. 距离完整的图还很远, 卡尔突然感到有些沉重.

让我们更详细地看一下代码. 首先, 软件包 `tikz` 被加载. 这个包是基础的 `pgf` 系统所谓的"前端". `pgf` 作为基础层, 在本手册中也有描述, 它更加底层, 因此也更难使用. 而前端则通过提供一个更简单的语法使绘图变得更容易.

在环境中, 有两个 `draw` 命令. 它们的意思是: "在命令后面指定的路径, 直到分号为止, 应该被绘制. 第一个路径被指定为 $(-1.5, 0) - (1.5, 0)$, 意思是从位置 $(-1.5, 0)$ 的点到位置 $(1.5, 0)$ 的直线". 这里的位置是在一个特殊的坐标系中指定的, 它默认的单位是 1 厘米.

卡尔非常高兴地注意到, 环境自动保留了足够的空间来涵盖这幅图.

1.2.2 在普通 $\text{T}_{\text{E}}\text{X}$ 中设置环境

卡尔的妻子格尔达也是一名数学教师, 她不是 $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ 用户, 而是使用普通 $\text{T}_{\text{E}}\text{X}$, 因为因为她更喜欢用"老办法"做事. 她也可以使用 `TikZ`. 她需要用 `\input tikz.tex` 代替 `\usepackage tikz`, 用 `\tikzpicture` 代替 `\begin tikzpicture`, 用 `\end tikzpicture` 代替 `\end tikzpicture`.

因此, 她会用下面的命令:

```
%% Plain TeX file\n\\input tikz.tex\n\\baselineskip=12pt\n\\hsize=6.3truein
```



```
\vsize=8.7truein
We are working on
\tikzpicture
  \draw (-1.5,0) - (1.5,0);
  \draw (0,-1.5) - (0,1.5);
\endtikzpicture.
\bye
```

格尔达可以使用 `pdftex` 或 `tex` 与 `dvips` 一起对该文件进行排版。TikZ 会自动识别她正在使用的驱动程序。如果她想使用 `dvipdfm` 和 `tex`, 她需要修改 `pgf.cfg` 文件, 或者在她输入 `tikz.tex` 或 `pgfsys-dvipdfm.def` 之前, 在某处写下 `\def \pgfsysdriverpgfsys-dvipdfm.def`。输入 `tikz.tex` 或 `pgf.tex`。

1.2.3 在 ConTeXt 中设置环境

卡尔的叔叔汉斯使用 ConTeXt。和格尔达一样, 汉斯也可以使用 TikZ。他不用 `\usepackage{tikz}`, 而是用 `\usemodule[tikz]`。他不写 `\begin{tikzpicture}`, 而是写 `\starttikzpicture`, 不写 `\end{tikzpicture}`, 而是写 `\stoptikzpicture`。

他那版的例子如下:

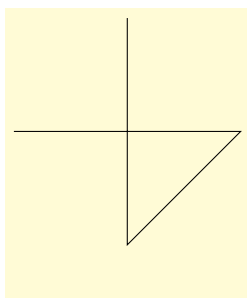
```
%% ConTeXt file
\usemodule[tikz]

\starttext
  We are working on
  \starttikzpicture
    \draw (-1.5,0) - (1.5,0);
    \draw (0,-1.5) - (0,1.5);
  \stoptikzpicture.
\stoptext
```

汉斯现在将以通常的方式使用 `texexec` 或 `context` 对这个文件进行排版。

1.3 构建直线路径

TikZ 中所有图片的基本构成部分是路径。路径是一系列的直线和曲线连接在一起（这不是全部，但让我们暂时忽略这些复杂的问题）。你在圆括号中指定起始位置的坐标，如 $(0,0)$ ，以此开始一条路径。接下来是一系列的"路径扩展操作"。最简单的是 `-`，我们已经用过了。它后面必须有另一个坐标，它将路径以直线方式延伸到这个新的位置。例如，如果我们要把两个轴的路径变成一个路径，会得到以下结果。



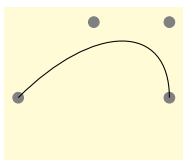
```
\tikz \draw (-1.5,0) - (1.5,0) - (0,-1.5) - (0,1.5);
```

卡尔对这里没有用 `{tikzpicture}` 环境感到有点困惑。取而代之的是小命令 `\tikz`。这个命令要么接受一个参数（以开头的大括号开始，如 `\tikz\draw (0,0) - (1.5,0)`，得到的是 `———`）或收集所有到下一个分号的内容，并将其放入一个 `{tikzpicture}` 环境中。作为一条经验法则，所有的 TikZ 图形绘制命令都必须作为 `\tikz` 的参数或是在 `{tikzpicture}` 环境内部出现。幸运的是，命令 `\draw` 只会在这个环境中定义，所以你几乎不可能在这一点上意外犯错。

1.4 构建曲线路径

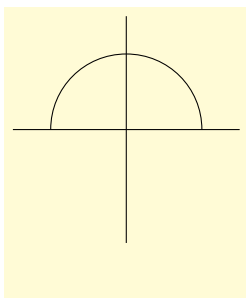
卡尔要做的下一件事是画圆。用直线显然是不行的。与之相反，我们需要一些方法来绘制曲线。为此，TikZ 提供了一种特殊的语法。我们需要一个或两个"控制点"。这背后的数学原理不是很简单，但是基本思路是这样的：假设你在点 x ，而第一个控制点是 y 。那么曲线将开始"向 x 处的 y 方向前进"，也就是说，曲线在 x 处的切线将会指向 y 。接下来，假设曲线的终点是 z ，第二个控制点是 w 。那么曲线会在 z 处结束，并且曲线在 z 处的切线会经过 w 。

下面是一个例子（为清晰起见，控制点已被添加）：



```
\begin{tikzpicture}
  \filldraw [gray] (0,0) circle [radius=2pt]
    (1,1) circle [radius=2pt]
    (2,1) circle [radius=2pt]
    (2,0) circle [radius=2pt];
  \draw (0,0) .. controls (1,1) and (2,1) .. (2,0);
\end{tikzpicture}
```

以"曲线"方式扩展路径的一般语法是`.. controls <first control point> and <second control point> .. <end point>`。你可以省略 `and <second control point>`, 这会使第一个控制点被使用两次。所以, 卡尔现在可以把第一个半圆添加到图片中。

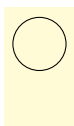


```
\begin{tikzpicture}
  \draw (-1.5,0) - (1.5,0);
  \draw (0,-1.5) - (0,1.5);
  \draw (-1,0) .. controls (-1,0.555)
    and (-0.555,1) .. (0,1)
    .. controls (0.555,1)
    and (1,0.555) .. (1,0);
\end{tikzpicture}
```

卡尔对这个结果很满意, 但发现以这种方式指定圆圈非常别扭。幸运的是, 有一个更简单的方法。

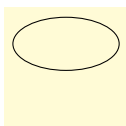
1.5 构建圆形路径

为了画一个圆, 可以使用路径构造操作 `circle`。这个操作后面的括号里可以指定半径, 如下面的例子: (注意, 前面的位置被用作圆心)



```
\tikz \draw (0,0) circle [radius=10pt];
```

你也可以用椭圆操作来构造一个椭圆。你可以指定两个半径, 而不是一个半径:

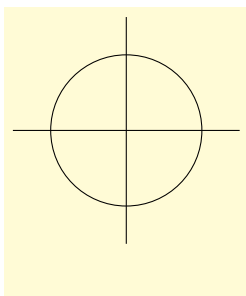


```
\tikz \draw (0,0) ellipse [x radius=20pt,
  y radius=10pt];
```

要画一个轴线不是水平和垂直的椭圆, 使其轴线指向一个任意的方向 (像 \circ 一样翻转的椭圆)。你可以使用接下来会讲的变换。对应的代码是 `\tikz \draw[rotate=30]`

(0,0) ellipse [x radius=6pt, y radius=3pt];.

因此, 回到卡尔的问题, 他可以写: `\draw (0,0) circle [radius=1cm];` 来画圆。

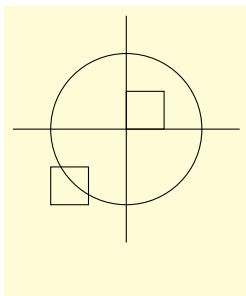


```
\begin{tikzpicture}
  \draw (-1.5,0) - (1.5,0);
  \draw (0,-1.5) - (0,1.5);
  \draw (0,0) circle [radius=1cm];
\end{tikzpicture}
```

在这一点上, 卡尔对圆如此之小感到有点慌张, 因为他想要的终稿比这要大得多。对此他很高兴地了解到, TikZ 有强大的转换选项, 将所有东西放大三倍是非常容易的。但是, 现在为了节省一些空间, 我们暂时让尺寸保持不变。

1.6 构建矩形路径

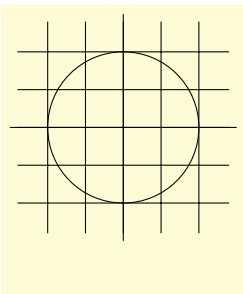
接下来我们想画的是背景中的网格。有几种方法可以生成它。例如, 人们可能会画很多的矩形。由于矩形非常常见, 所以有一个特殊的语法。要在当前路径上添加一个矩形, 请使用 `rectangle` 路径构造操作。这个操作后应该有另一个坐标, 并将在路径上附加一个矩形, 使前一个坐标和后一个坐标分别是矩形的两个对角。接下来, 让我们在图上添加两个矩形。

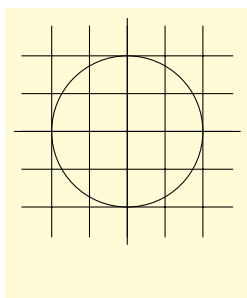


```
\begin{tikzpicture}
  \draw (-1.5,0) - (1.5,0);
  \draw (0,-1.5) - (0,1.5);
  \draw (0,0) circle [radius=1cm];
  \draw (0,0) rectangle (0.5,0.5);
  \draw (-0.5,-0.5) rectangle (-1,-1);
\end{tikzpicture}
```

虽然这在其他情况下可能很好用, 但这并不能真正解决卡尔的问题: 首先, 我们需要大量的矩形, 其次还存在不"封闭"的边。因此, 当知道有一个 `grid` 路径构建操作后, 卡尔准备使用 `\draw` 命令简单地画四条垂直线和四条水平线。

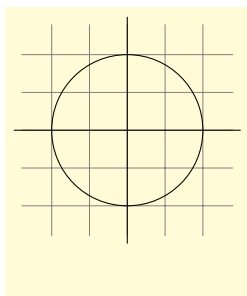
1.7 构造网格路径

`grid` 路径操作会为当前路径增加一个网格。它会在两个坐标（当前指定的点和 `grid` 后的点）指定的矩形之间添加直线来构成网格。例如，代码 `\tikz \draw[step=2pt] (0,0) grid (10pt,10pt);` 会生成 。注意 `\draw` 的可选参数可以用来指定网格宽度（当然也可以用 `xstep` 和 `ystep` 来独立定义网格宽度）。卡尔很快就会知道，有很多东西可以通过这些选项来改变。



```
\begin{tikzpicture}
  \draw (-1.5,0) - (1.5,0);
  \draw (0,-1.5) - (0,1.5);
  \draw (0,0) circle [radius=1cm];
  \draw[step=.5cm] (-1.4,-1.4) grid (1.4,1.4);
\end{tikzpicture}
```

再看一下所需的图片，卡尔注意到，如果网格能更多一些就好了。（他的儿子告诉他，如果网格没有被弱化，就会让人分心。）为了弱化网格。卡尔为绘制网格的 `\draw` 命令增加了两个选项。首先，他把网格的线变成灰色。其次，他把线的宽度变到很细。最后，他调换了命令的顺序以便先画出网格，其他图形的都在网格的上面。



```
\begin{tikzpicture}
  \draw[step=.5cm,gray,very thin] (-1.4,-1.4)
    grid (1.4,1.4);

  \draw (-1.5,0) - (1.5,0);
  \draw (0,-1.5) - (0,1.5);
  \draw (0,0) circle [radius=1cm];
\end{tikzpicture}
```

1.8 添加一点样式

卡尔其实也可以用辅助线 (`help lines`) 这种预定义样式，而不用自己指定 `gray`, `very thin` 这些选项。样式 (*style*) 就是一些预定义的用来控制具体图形的绘制方式的选项组。通过指定 `help lines` 等于下达“用我（或者其他人）之前设定的画辅助线的样式”。如果卡尔之后再遇到要绘制网格的情况，但那个时候，举个例子，线的颜色想要用 `blue!50` 而不是 `gray`，那他就可以在某个地方提供以下选项：

```
help lines/.style={color=blue!50,very thin}
```

这个"样式设置器"能使得在当前范围或环境中,辅助线选项具有与 `color=blue!50`, `very thin` 相同的效果。

使用样式使你的绘图代码更加灵活。并且你可以很容易地以一致的方式改变事物的外观。通常情况下,样式是在一张图片的开头定义的。不过有时你可能希望全局地定义一个样式,这样你的文档中的所有图片都可以使用这个样式。然后,你可以很容易地通过改变这一个样式来改变所有图片的外观。在这种情况下,你可以在文档的开头使用 `\tikzset` 命令,如

```
\tikzset{help lines/.style=very thin}
```

为了建立样式的层次结构,你可以让一个样式使用另一个样式。因此,为了定义一个基于 `grid` 样式的卡尔's `grid` 样式,可以像下面这样:

```
\tikzset{卡尔's grid/.style={help lines,color=blue!50}}  
...  
\draw[卡尔's grid] (0,0) grid (5,5);
```

样式通过参数化而变得更加强大。这意味着,像其他选项一样,样式也可以接收参数。例如,卡尔可以将他的网格参数化,使其默认为蓝色。但他也可以使用另一种颜色。

```
\begin{tikzpicture}  
  [卡尔's grid/.style = {help lines,color=#1!50},  
  卡尔's grid/.default=blue]  
  \draw[卡尔's grid] (0,0) grid (1.5,2);  
  \draw[卡尔's grid=red] (2,0) grid (3.5,2);  
\end{tikzpicture}
```

在本例中,卡尔's `grid` 样式的定义作为可选参数提供给 `{tikzpicture}` 环境。其他元素的其他样式跟在逗号后面。实际上由很多不同的样式,环境的可选参数可能很容易就会比实际内容还长。

1.9 画图选项

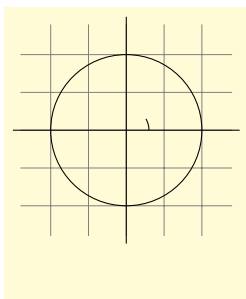
卡尔想知道还有什么其他选项会影响路径的绘制。他已经见识过 `color=<color>` 选项可以定制线的颜色。而选项 `draw=<color>` 的效果几乎一样，只是它不光能设置线的颜色，还能设置不同的填充色（卡尔在填充角对应的圆弧的时候会用到这个）。

他也见识过样式 `very thin` 可以生成很细的线。卡尔对此并不感到惊讶，他也不惊讶于发现 `thin` 会产生细线，`thick` 会产生粗线，`very thick` 会产生非常粗的线，`ultra thick` 会产生非常非常粗的线条和 `ultra thin` 会产生如此细的线条，以至于低分辨率的打印机和显示器将无法显示它们。他想知道什么选项能生成“正常”粗细的线条。事实证明，`thin` 是正确的选择，因为它给出的厚度与 \TeX 的 `\hrule` 命令相同。尽管如此，卡尔想知道在薄和厚之间是否有任何“中间”情况。答案是有的：`semithick`。

另一个对于画线来说有用的东西就是用线划线或点划线。与之相应的有 `dashed` 和 `dotted` 两种样式，分别生成----和.....。这两种选择也都同时存在于松散的和密集的版本，分别为 `loosely dashed`、`densely dashed`、`loosely dotted` 和 `densely dotted`。如果他真的需要，卡尔也可以用 `dash pattern` 选项定义更复杂的划线样式，但他的儿子坚持说，划线要非常小心地使用，而且大多数时候会分散读者的注意力。卡尔的儿子声称复杂的划线模式是很不好的。卡尔的学生也并不关心划线的样式。

1.10 构造弧线路径

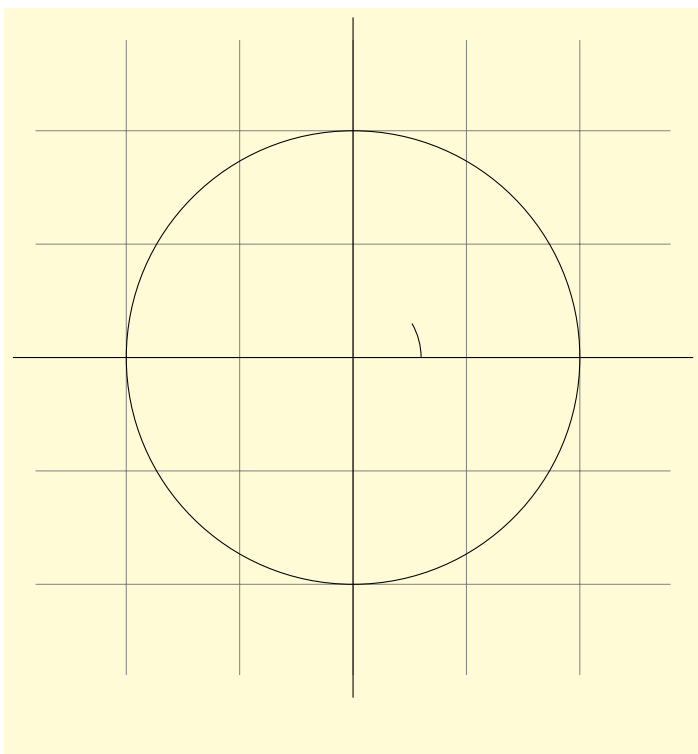
我们的下一个障碍是画出具体角度的弧线。为此，`arc` 路径的构建操作是有用的，它能够绘制圆或椭圆的一部分。这个 `arc` 操作后面是括号中的选项可以用来指定对应的圆弧。一个具体例子就是 `arc[start angle=10, end angle=80, radius=10pt]`。卡尔显然需要一个从 0° 到 30° 的弧线。半径应该比较小，大概是大圆半径的三分之一左右。当使用弧线路径构建操作时，指定的弧线将以当前位置为起点添加。所以，我们首先要“到达那里”。



```
\begin{tikzpicture}
  \draw[step=.5cm,gray,very thin] (-1.4,-1.4)
    grid (1.4,1.4);

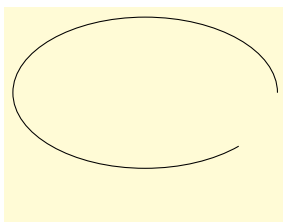
  \draw (-1.5,0) - (1.5,0);
  \draw (0,-1.5) - (0,1.5);
  \draw (0,0) circle [radius=1cm];
  \draw (3mm,0mm) arc [start angle=0, end angle=30,
    radius=3mm];
\end{tikzpicture}
```

卡尔认为现在的图真的有点小，除非他学会如何缩放，否则就不能继续画了。为此，他可以添加 `[scale=3]` 选项。他可以将这个选项添加到每个 `\draw` 命令中，但那样会很别扭。取代上面想法的是，他将其添加到整个环境中，从而使此选项适用于其中的所有内容。




```
\begin{tikzpicture}[scale=3]
  \draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
  \draw (-1.5,0) - (1.5,0);
  \draw (0,-1.5) - (0,1.5);
  \draw (0,0) circle [radius=1cm];
  \draw (3mm,0mm) arc [start angle=0, end angle=30, radius=3mm];
\end{tikzpicture}
```

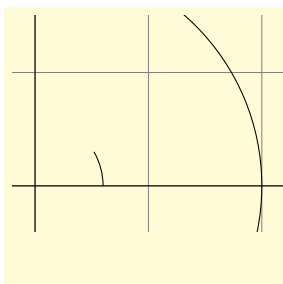
就像对圆的操作一样，你也可以指定“两个”半径以得到一个椭圆圆弧。



```
\tikz \draw (0,0)
arc [start angle=0, end angle=315,
x radius=1.75cm, y radius=1cm];
```

1.11 裁剪路径

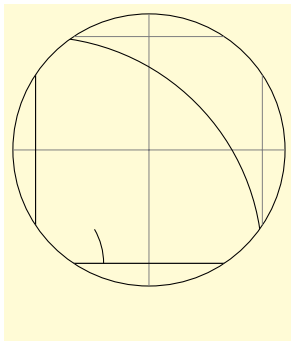
为了在本手册中节省空间，最好裁剪一下卡尔的图形，这样我们就可以专注于“有趣的”部分。在 TikZ 中裁剪非常简单。你可以使用 `\clip` 命令裁剪所有后续绘制的图形。它的工作方式类似于 `\draw`，只是它不绘制任何东西，但随后使用给定的路径裁剪所有东西。（下面是裁处一块矩形区域的例子，`rectangle` 前后分别是裁剪得到的矩形区域的两个对角）



```
\begin{tikzpicture}[scale=3]
  \clip (-0.1,-0.2) rectangle (1.1,0.75);
  \draw[step=.5cm,gray,very thin] (-1.4,-1.4)
    grid (1.4,1.4);
  \draw (-1.5,0) - (1.5,0);
  \draw (0,-1.5) - (0,1.5);
  \draw (0,0) circle [radius=1cm];
  \draw (3mm,0mm) arc [start angle=0, end angle=30,
    radius=3mm];
\end{tikzpicture}
```

你也可以同时做这两件事：绘制和裁剪路径。为此，使用 `\draw` 命令并添加 `clip` 选项。（这还不是全部：你也可以使用 `\clip` 命令并添加 `draw` 选项。好吧，这也不是全部：事实上，`\draw` 只是 `\path[draw]` 的缩写，`\clip` 是 `\path[clip]` 的缩写，

你也可以写成 `\path[draw,clip]`) 下面是一个例子:

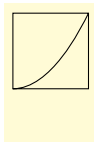


```
\begin{tikzpicture}[scale=3]
  \clip[draw] (0.5,0.5) circle (.6cm);
  \draw[step=.5cm,gray,very thin] (-1.4,-1.4)
                                         grid (1.4,1.4);

  \draw (-1.5,0) - (1.5,0);
  \draw (0,-1.5) - (0,1.5);
  \draw (0,0) circle [radius=1cm];
  \draw (3mm,0mm) arc [start angle=0, end angle=30,
                      radius=3mm];
\end{tikzpicture}
```

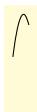
1.12 构造抛物线和正弦路径

虽然卡尔的图不需要它们,但他很高兴地得知,有 `parabola`(抛物线)、`sin`(正弦) 和 `cos`(余弦) 路径操作,可以将抛物线和正余弦曲线添加到当前的路径上。对于 `parabola` 操作,当前点和抛物线操作后给定的点一样位于抛物线上。考虑以下例子:



```
\tikz \draw (0,0) rectangle (1,1)
            (0,0) parabola (1,1);
```

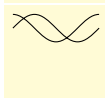
也可以在其他地方弯折:



```
\tikz \draw[x=1pt,y=1pt] (0,0) parabola bend (4,16)
                                         (6,12);
```

`sin`(正弦) 和 `cos`(余弦) 路径操作会添加位于 $[0, \pi/2]$ 之间的正弦或余弦曲线,并把当前位置作为起点,把之后给定的位置作为终点。这里是两个例子。

A sine \curvearrowright curve.

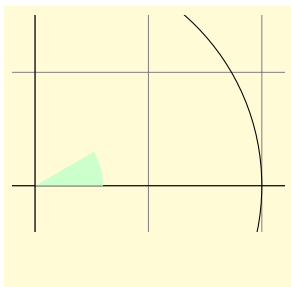


```
A sine \tikz \draw[x=1ex,y=1ex] (0,0) sin (1.57,1);
                                         curve.

\tikz \draw[x=1.57ex,y=1ex] (0,0) sin (1,1) cos (2,0)
sin (3,-1) cos (4,0) (0,1) cos (1,0) sin (2,-1) cos
(3,0) sin (4,1);
```

1.13 轮廓与填充

回到我们要绘制的图上，卡尔现在希望这个角被一种非常浅的绿色“填充”。为此，他使用 `\fill` 而不是 `\draw`。卡尔是这样做的：



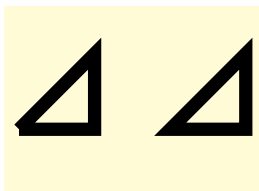
```
\begin{tikzpicture}[scale=3]
  \clip (-0.1,-0.2) rectangle (1.1,0.75);
  \draw[step=.5cm,gray,very thin] (-1.4,-1.4)
                                         grid (1.4,1.4);

  \draw (-1.5,0) - (1.5,0);
  \draw (0,-1.5) - (0,1.5);
  \draw (0,0) circle [radius=1cm];
  \fill[green!20!white] (0,0) - (3mm,0mm)
    arc [start angle=0, end angle=30, radius=3mm]
      - (0,0);
\end{tikzpicture}
```

`green!20!white` 表示 20% 的绿色和 80% 的白色混合在一起。这样的颜色表达是可能是因为 TikZ 使用了 Uwe Kern 的 `xcolor` 包，有关详细信息，请参阅该包关于颜色表达式的文档。如果卡尔没有在最后用 `- (0,0)` 来“关闭”路径，会发生什么？在这种情况下，路径是自动关闭的，所以这可以省略。不过事实上，像下面这样做会更好：

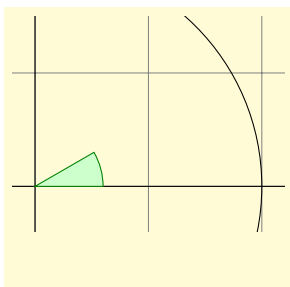
```
\fill[green!20!white] (0,0) - (3mm,0mm)
  arc [start angle=0, end angle=30, radius=3mm] - cycle;
```

`- cycle` 使得当前路径被关闭（实际上是当前路径的当前部分）第一点和最后一点顺利衔接。要理解这种差异，请参考以下示例



```
\begin{tikzpicture}[line width=5pt]
  \draw (0,0) - (1,0) - (1,1) - (0,0);
  \draw (2,0) - (3,0) - (3,1) - cycle;
  % make bounding box higher
  \useasboundingbox (0,1.5);
\end{tikzpicture}
```

你也可以使用 `\filldraw` 命令同时填充和绘制路径。这将首先绘制路径，然后填充它。这可能看起来不是很有用，但是你可以指定不同的颜色来用于填充和描边。这些参数被指定为可选参数，如下所示：

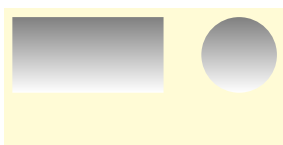


```
\begin{tikzpicture}[scale=3]
  \clip (-0.1,-0.2) rectangle (1.1,0.75);
  \draw[step=.5cm,gray,very thin] (-1.4,-1.4)
                                         grid (1.4,1.4);

  \draw (-1.5,0) - (1.5,0);
  \draw (0,-1.5) - (0,1.5);
  \draw (0,0) circle [radius=1cm];
  \filldraw[fill=green!20!white,
            draw=green!50!black]
            (0,0) - (3mm,0mm)
            arc [start angle=0, end angle=30, radius=3mm]
            - cycle;
\end{tikzpicture}
```

1.14 阴影

卡尔简要地考虑了通过阴影使角“更好看”的可能性。使用不同颜色之间的平滑过渡，而不是用统一的颜色填充区域。为此可以用 `\shade` 和 `\shadedraw` 来同时用于添加阴影和绘图：



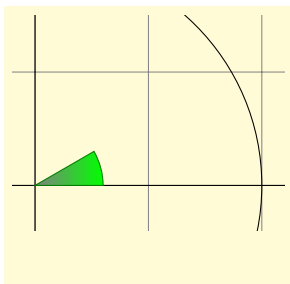
```
\tikz \shade (0,0) rectangle (2,1) (3,0.5)
        circle (.5cm);
```

默认的阴影是一个从灰色到白色的平滑过渡。要指定不同的颜色，你可以使用以下选项：



```
\begin{tikzpicture}[rounded corners,ultra thick]
  \shade[top color=yellow,bottom color=black] (0,0) rectangle +(2,1);
  \shade[left color=yellow,right color=black] (3,0) rectangle +(2,1);
  \shadedraw[inner color=yellow,outer color=black,draw=yellow] (6,0)
  rectangle +(2,1);
  \shade[ball color=green] (9,.5) circle (.5cm);
\end{tikzpicture}
```

对于卡尔来说，下面的样子可能比较合适：



```
\begin{tikzpicture}[scale=3]
  \clip (-0.1,-0.2) rectangle (1.1,0.75);
  \draw[step=.5cm,gray,very thin] (-1.4,-1.4)
                                         grid (1.4,1.4);

  \draw (-1.5,0) - (1.5,0);
  \draw (0,-1.5) - (0,1.5);
  \draw (0,0) circle [radius=1cm];
  \shadedraw[left color=gray,right color=green,
             draw=green!50!black]
    (0,0) - (3mm,0mm)
    arc [start angle=0, end angle=30, radius=3mm]
    - cycle;
\end{tikzpicture}
```

然而，他明智地判断，阴影通常只会分散注意力，而不会为图片添加任何有意义的东西。

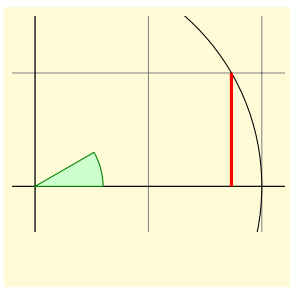
1.15 指定坐标

卡尔现在想要在图中添加正弦线和余弦线。他已经知道可以用 `color=` 这个选项来指定线的颜色。那么有什么好方法可以指定线的位置（坐标）呢？

有很多种不同的方式可以指定坐标。最简单的方式就是使用类似 `(10pt,2cm)` 这样的命令。这表示在 x 轴上 10pt 和 y 轴上 2cm 的位置。当然我们也可以不带单位，写成 `(1,2)`，这表示“一倍的 x 向量加上两倍的 y 向量”。默认情况下它们 (x 向量和 y 向量) 是在各自轴上 1cm 长度的向量。

为了指定极坐标上的点，可以用这样的记号 `(30:1cm)`，这表示在 30° 方向上 1cm 处。这显然对于获得在圆上 $(\cos 30^\circ, \sin 30^\circ)$ 的点很好用。

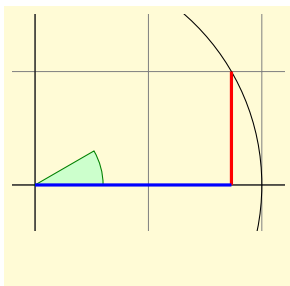
你可以在一个坐标前加一个或两个加 (+) 号，形如 `+(0cm,1cm)` 或 `++(2cm,0cm)`。这两种坐标表示对应不同的解释：第一种形式表示“从之前指定的位置向上 1cm ”，第二种形式表示“在之前指定的位置右侧 2cm ，使其成为新的指定位置”。例如，我们可以画如下的正弦线来帮助理解：



```
\begin{tikzpicture}[scale=3]
\clip (-0.1,-0.2) rectangle (1.1,0.75);
\draw[step=.5cm,gray,very thin] (-1.4,-1.4)
grid (1.4,1.4);
\draw (-1.5,0) - (1.5,0);
\draw (0,-1.5) - (0,1.5);
\draw (0,0) circle [radius=1cm];
\filldraw[fill=green!20,draw=green!50!black]
(0,0) - (3mm,0mm)
arc [start angle=0, end angle=30, radius=3mm]
- cycle;
\draw[red,very thick] (30:1cm) - +(0,-0.5);
\end{tikzpicture}
```

卡尔利用 $\sin 30^\circ = 1/2$ 这一条件。不过他不确定他的学生们知不知道这一点，所以如果能画出从 (30:1cm) 到 x 轴的垂线会很有帮助。这种情况可以用一个特殊的语法：卡尔可以写 (30:1cm |- 0,0)。一般来说，(<p> |- <q>) 表示经过 p 的竖线和经过 q 的横线交汇的位置。

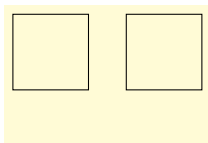
接下来，我们来画余弦线。一种方式是指定 (30:1cm |- 0,0) - (0,0)。另一种方式如下，我们从正弦结束的地方“继续”：



```
\begin{tikzpicture}[scale=3]
\clip (-0.1,-0.2) rectangle (1.1,0.75);
\draw[step=.5cm,gray,very thin] (-1.4,-1.4)
grid (1.4,1.4);
\draw (-1.5,0) - (1.5,0);
\draw (0,-1.5) - (0,1.5);
\draw (0,0) circle [radius=1cm];
\filldraw[fill=green!20,draw=green!50!black]
(0,0) - (3mm,0mm)
arc [start angle=0, end angle=30, radius=3mm]
- cycle;
\draw[red,very thick] (30:1cm) - +(0,-0.5);
\draw[blue,very thick] (30:1cm) ++(0,-0.5)
- (0,0);
\end{tikzpicture}
```

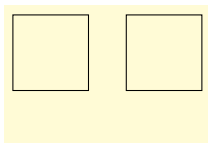
注意在 $(30:1\text{cm})$ 和 $++(0,-0.5)$ 之间没有两个短横 (-)。具体来说, 这个路径的解释是这样的: “首先, $(30:1\text{cm})$ 使得画笔移动到 $(\cos 30^\circ, 1/2)$ 的位置。然后, 我们就遇到了第二种指定坐标的形式, 所以我按照指定的偏移移动画笔但不绘制任何东西。新的点在原先点下方 $1/2$ 个单位处, 也就是 $(\cos 30^\circ, 0)$ 。最后, 我把画笔移动回原点, 并通过两个短横 (-) 画出一条直线。”

为了理解在 $+$ 和在 $++$ 之间的区别, 请看以下示例:



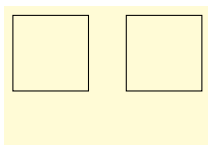
```
\begin{tikzpicture}
\def\rectanglepath{- ++(1cm,0cm) - ++(0cm,1cm)
- ++(-1cm,0cm) - cycle}
\draw (0,0) \rectanglepath;
\draw (1.5,0) \rectanglepath;
\end{tikzpicture}
```

相比之下, 当使用单个 $+$ 时, 用的坐标是不同的:



```
\begin{tikzpicture}
\def\rectanglepath{- +(1cm,0cm) - +(1cm,1cm)
- +(0cm,1cm) - cycle}
\draw (0,0) \rectanglepath;
\draw (1.5,0) \rectanglepath;
\end{tikzpicture}
```

当然, 所有这些都写得更加简洁, 更清楚, 就像下面这样 (用单个或两个 $+$ 都可以):



```
\tikz \draw (0,0) rectangle +(1,1)
(1.5,0) rectangle +(1,1);
```

1.16 路径相交

卡尔现在还剩下 $\tan \alpha$ 对应的线没有画, 不过这条线貌似很难用变换和极坐标表示出来。他可以做的第一件事 (也是最简单的一件事) 就是简单地使用坐标 $(1, \tan(30))$, 因为 TikZ 的数学引擎知道如何计算像 $\tan(30)$ 这样的东西。注意添加大括号, 否则, TikZ 的解释器会把第一个右括号当作坐标结束的标志 (通常, 当这些成分内部包含括号时, 你需要在坐标组件周围添加大括号)。

当然, 卡尔也可以使用一种更精细, 但也更“几何”的方法来计算橙色线的长度:

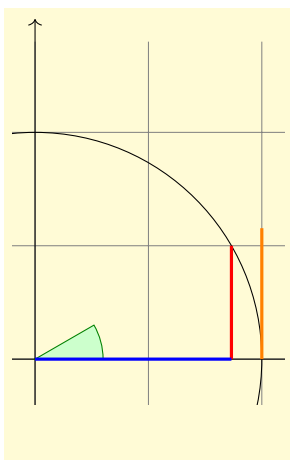
他可以将路径的交点指定为坐标。 $\tan \alpha$ 对应的直线以 $(1, 0)$ 为起点，向上直到和从原点穿过 $(30:1\text{cm})$ 的直线的相交。`intersections` 库提供了这样的计算。

```
\path [name path=upward line] (1,0) - (1,1);
\path [name path=sloped line] (0,0) - (30:1.5cm);
% a bit longer, so that there is an intersection
% (add '\usetikzlibrary{intersections}' after loading tikz in the preamble)
\draw [name intersections={of=upward line and sloped line, by=x}]
[very thick,orange] (1,0) - (x);
```

1.17 添加箭头

卡尔现在想在轴的末端加上小箭头。他注意到，在许多图表中，甚至在科学期刊上，这些箭头提示似乎都不见了，大概是因为生成程序无法生成它们。卡尔认为箭头应该在轴的末端。他的儿子也同意这一点。而他的学生实际上根本不关心箭头。

实际上添加箭头提示非常简单：卡尔在轴的绘制命令中添加了 `->` 选项：



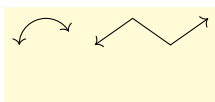

```

\usetikzlibrary {intersections}
\begin{tikzpicture}[scale=3]
\clip (-0.1,-0.2) rectangle (1.1,1.51);
\draw[step=.5cm,gray,very thin] (-1.4,-1.4)
grid (1.4,1.4);
\draw[->] (-1.5,0) - (1.5,0);
\draw[->] (0,-1.5) - (0,1.5);
\draw (0,0) circle [radius=1cm];
\filldraw[fill=green!20,draw=green!50!black] (0,0) - (3mm,0mm)
arc [start angle=0, end angle=30, radius=3mm] - cycle;
\draw[red,very thick] (30:1cm) - +(0,-0.5);
\draw[blue,very thick] (30:1cm) ++(0,-0.5) - (0,0);
\path [name path=upward line] (1,0)- (1,1);
\path [name path=sloped line] (0,0)- (30:1.5cm);
\draw [name intersections={of=upward line and sloped line, by=x}]
[very thick,orange] (1,0) - (x);
\end{tikzpicture}

```

如果卡尔使用 `<-` 而不是 `->` 选项, 箭头提示就会放在路径的开头。选项 `<->` 将箭头提示放在路径的两端。

只有部分类型的路径可以添加箭头提示。根据经验, 你只能在一条非闭合的“线”上添加箭头提示。例如, 你不能向矩形或圆形添加箭头。但是, 你可以给弧线路径和由几个线段构成的路径上添加箭头提示, 如下面的例子所示:



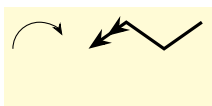
```

\begin{tikzpicture}
\draw [<->] (0,0) arc
[start angle=180, end angle=30, radius=10pt];
\draw [<->] (1,0) - (1.5cm,10pt) - (2cm,0pt)
- (2.5cm,10pt);
\end{tikzpicture}

```

卡尔对 TikZ 在结尾放置的箭头有了更详细的了解。箭头在放大后是这样的, \rightarrow 。这个形状似乎很熟悉, 实际上, 这正是 TeX 中标准箭头的结尾, 它用于 $f: A \rightarrow B$ 。

卡尔很喜欢这种箭头, 主要是因为它不像许多其他包提供的箭头那样粗。不过, 有时他可能也会需要使用其他种类的箭头。针对这种情况, 卡尔可以这样写 `>=<` 结尾箭头种类 `>`, 其中 `<` 结尾箭头种类 `>` 是一个特殊的箭头描述。例如, 如果卡尔写 `>=<Stealth>`, 那么等于他告诉 TikZ, 他想要“隐形战斗机一样”的箭头提示:



```
\usetikzlibrary {arrows.meta}
\begin{tikzpicture}[>=Stealth]
\draw [->] (0,0) arc
[start angle=180, end angle=30, radius=10pt];
\draw [«-,very thick] (1,0) - (1.5cm,10pt) - (2cm,0pt)
- (2.5cm,10pt);
\end{tikzpicture}
```

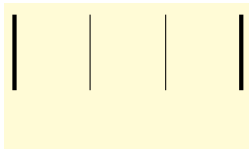
卡尔想知道这样一个箭头类型用夸张的军事名称是否真的有必要。当他的儿子告诉他微软的 PowerPoint 使用了相同的名字时, 他并没有真正平静下来。他决定让他的学生在某个时候讨论这个问题。

除了隐形, 卡尔还可以选择其他几种预定义的箭头提示, 参见 section 105。此外, 如果需要新的箭头类型, 他可以自己定义箭头类型。

Link to
第 105
节

1.18 作用域

卡尔已经看到有许多图形选项可以影响路径的渲染方式。通常, 他喜欢将某些选项应用于整个图形命令集。例如, 卡尔可能希望使用粗笔绘制三条路径, 但希望其他所有内容都“正常”绘制。如果卡尔希望为整个图片设置一个特定的图形选项, 他可以简单地将这个选项传递给 `\tikz` 命令或 `{tikzpicture}` 环境 (格尔达将这些选项传递给 `\tikzpicture`, 而 Hans 将它们传递给 `\starttikzpicture`)。然而, 如果卡尔想在本地的一些图形构成的组中应用图形选项, 他会把这些命令放在 `{scope}` 环境中 (格尔达使用 `\scope` 和 `\endscope`, 汉斯使用 `\startscope` 和 `\stopscope`)。该环境将图形选项作为可选参数, 这些选项适用于范围内的所有内容, 但不适用于范围外的任何内容。下面是一个例子:



```
\begin{tikzpicture}[ultra thick]
\draw (0,0) - (0,1);
\begin{scope}[thin]
\draw (1,0) - (1,1);
\draw (2,0) - (2,1);
\end{scope}
\draw (3,0) - (3,1);
\end{tikzpicture}
```

作用域还有另一个有趣的效果: 对裁剪区域的任何更改都是只作用于局部作用域

内。因此,如果你在作用域中的某个地方使用 `\clip`, `\clip` 命令的效果将在作用域中结束时结束。这是有意义的,因为我们没有其他方法来“扩大”裁剪区域。

怎么理解

卡尔注意到给具体的命令诸如 `\draw` 设置选项只作用于这条命令。这证明了实际的作用域机制会更加复杂一些。首先对于具体的命令诸如 `\draw` 的选项并非真正对条命令的选项,而是“路径选项”,可以出现在路径的任何位置。所以既可以写成这样 `\draw[thin] (0,0) - (1,0);`,也可以写成这样 `\draw (0,0) [thin] - (1,0);` 或这样 `\draw (0,0) - (1,0) [thin];`; 这些效果都是一样的。最后一种情况似乎看起来有点奇怪,因为 `thin` 都已经在画线“后面”了。然而,大多数图形选项作用于整条路径。如果你同时一条路径上添加了 `thin` 和 `thick` 两个选项,那么最后一个会起作用,覆盖前者。

当阅读上述内容时,卡尔注意到只有“大多数”图形选项适用于整个路径。实际上,所有转换选项并不适用于整个路径,而只适用于“路径上跟随它们的所有东西”。我们稍后将对此进行更详细的讨论。然而,在路径构建过程中给出的所有选项只适用于当前这一条路径。

1.19 变换

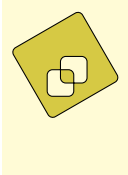
当你指定一个坐标,比如 `(1cm,1cm)`,会被放在页面的哪个位置呢? 为了解答这一点, `TikZ`、`TeX` 和 `pdf` 或 `PostScript` 都对指定的坐标应用某些转换,以确定页面上的最终位置。

`TikZ` 提供了许多选项,允许你转换 `TikZ` 的私有坐标系中的坐标。例如, `xshift` 选项允许你定量移动所有后续的点:

||

```
\tikz \draw (0,0) - (0,0.5) [xshift=2pt] (0,0) - (0,0.5);
```

需要注意的是,你可以“在路径中间”更改转换, `pdf` 或 `PostScript` 不支持这个特性。原因是 `TikZ` 会跟踪自己的变换矩阵。下面是一个更复杂的例子:



```
\begin{tikzpicture}
[even odd rule,rounded corners=2pt,x=10pt,y=10pt]
\filldraw[fill=yellow!80!black] (0,0) rectangle (1,1)
[xshift=5pt,yshift=5pt] (0,0) rectangle (1,1)
[rotate=30] (-1,-1) rectangle (2,2);
\end{tikzpicture}
```

用于平移的 `xshift` 和 `yshift` 是最有用变换。`shift` 用于把坐标系平移的给定的点, 比如 `shift={(1,0)}` 或 `shift={+(0,0)}` (大括号是必要的, 用来防止 `TEX` 把内部的逗号当成选项间隔的标识符), `rotate` 选项则是把坐标系以原点为中心旋转一定的角度 (还有一个 `rotate around` 可以指定旋转中心), `scale` 选项可以按一定的系数来缩放图片, `xscale` 和 `yscale` 是只在 `x` 轴或 `y` 轴方向上缩放 (`xscale=-1` 就是水平翻转), 而 `xslant` 和 `yslant` 用于倾斜。如果这些变换和我没有提到的那些变换不够充分, 那么 `cm` 选项允许您应用任意的变换矩阵。卡尔的学生, 顺便说一下, 不知道什么是变换矩阵。

1.20 重复: For 循环

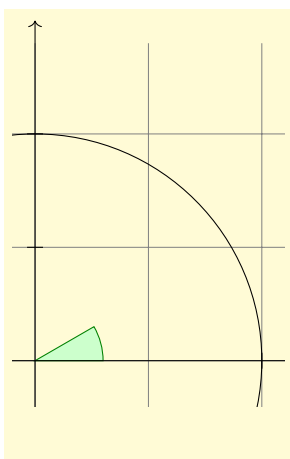
Karl 的下一个目标是在坐标轴 -1 , $-1/2$, $1/2$ 和 1 的位置上添加刻度。对于这一点, 最好使用某种“循环”, 特别是因为他希望在每个位置都做相同的事情。有不同的软件包可以实现这一点。`LATEX` 有用来做这件事的内置命令, `pstricks` 则有强大的 `\multido` 命令。这些都可以和 `TikZ` 配合使用, 所以如果你对这些命令很熟练, 完全可以在 `TikZ` 中使用他们。`TikZ` 也引入了一个新的命令, `\foreach`, 因为我记不住其他包对应命令的语法, 所以引入了它。`\foreach` 是在包 `pgffor` 中定义的, 可以独立于 `TikZ` 使用, 但是 `TikZ` 会自动包含它。

`\foreach` 的基本使用形式非常简便:

```
x = 1, x = 2, x = 3, \foreach \x in {1,2,3} {$x =\x$, }
```

一般语法是这样的 `\foreach <variable> in {<list of values>} <commands>`。在 `<commands>` 中, `<variable>` 将随着循环被赋以不同的值。如果 `<commands>` 不以大括号开头, 那么直到下一个分号的内容被用作 `<commands>`。

对于坐标轴上的刻度, 卡尔可以使用以下代码:

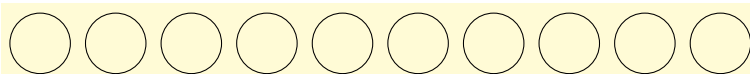


```
\begin{tikzpicture}[scale=3]
  \clip (-0.1,-0.2) rectangle (1.1,1.51);
  \draw[step=.5cm,gray,very thin] (-1.4,-1.4)
  grid (1.4,1.4);
  \filldraw[fill=green!20,draw=green!50!black]
  (0,0) - (3mm,0mm)
  arc [start angle=0, end angle=30, radius=3mm]
  - cycle;
  \draw[->] (-1.5,0) - (1.5,0);
  \draw[->] (0,-1.5) - (0,1.5);
  \draw (0,0) circle [radius=1cm];
  \foreach \x in {-1cm,-0.5cm,1cm}
  \draw (\x,-1pt) - (\x,1pt);
  \foreach \y in {-1cm,-0.5cm,0.5cm,1cm}
  \draw (-1pt,\y) - (1pt,\y);
\end{tikzpicture}
```

事实上, 加刻度的方法有很多种。例如, 卡尔可以把`\draw ...` 放在大括号内。他也可以像下面这样,

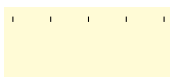
```
\foreach \x in {-1,-0.5,1}
\draw[xshift=\x cm] (0pt,-1pt) - (0pt,1pt);
```

卡尔很好奇在一个更复杂的情况下会发生什么, 比如说, 有 20 个刻度。在`\foreach` 集合中明确地提到所有这些数字似乎很麻烦。事实上, 可以在`\foreach` 语句中用... 迭代大量的值 (但是, 这必须是没有单位的实数), 如下例所示:



```
\tikz \foreach \x in {1,...,10}
\draw (\x,0) circle (0.4cm);
```

如果你在... 前面提供两个数字, 那么`\foreach` 语句就会用它们的差值作为迭代的步长:



```
\tikz \foreach \x in {-1,-0.5,...,1}
\draw (\x cm,-1pt) - (\x cm,1pt);
```

我们还可以嵌套循环来创建有趣的效果:

1,5	2,5	3,5	4,5	5,5	7,5	8,5	9,5	10,5	11,5	12,5
1,4	2,4	3,4	4,4	5,4	7,4	8,4	9,4	10,4	11,4	12,4
1,3	2,3	3,3	4,3	5,3	7,3	8,3	9,3	10,3	11,3	12,3
1,2	2,2	3,2	4,2	5,2	7,2	8,2	9,2	10,2	11,2	12,2
1,1	2,1	3,1	4,1	5,1	7,1	8,1	9,1	10,1	11,1	12,1

```

\begin{tikzpicture}
\foreach \x in {1,2,...,5,7,8,...,12}
\foreach \y in {1,...,5}
{
\draw (\x,\y) +(-.5,-.5) rectangle ++(.5,.5);
\draw (\x,\y) node{\x,\y};
}
\end{tikzpicture}

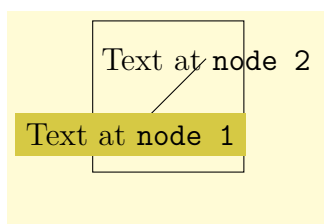
```

`\foreach` 语句还可以做更复杂的事情，不过上面这些只是为了给出基本的思路。

1.21 添加文本

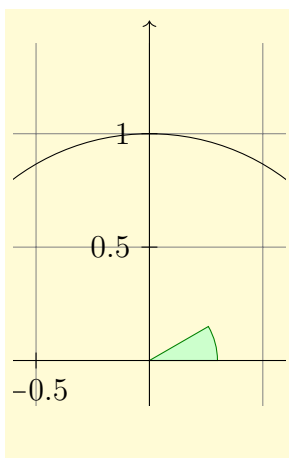
卡尔现在对这幅画很满意。然而，最重要的部分，即标签，仍然缺少！

TikZ 提供了一个易于使用且功能强大的系统，可以将文本以及更普遍的复杂形状添加到图片的特定位置。基本思路是这样的：当 TikZ 构造路径并在路径中间遇到关键字 `node` 时，它会读取节点声明 (*node specification*)。关键字 `node` 通常后面跟着一些选项，然后是大括号之间的一些文本。这些文本会被放进一个普通的 \TeX 盒子 (box, \TeX 中的基本处理单位) 中 (最常见的情况是节点声明直接跟在一个坐标后面，一字不差地使用盒子里的文本) 然后放在当前位置。



```
\begin{tikzpicture}
\draw (0,0) rectangle (2,2);
\draw (0.5,0.5) node [fill=yellow!80!black]
{Text at \verb!node 1!}
- (1.5,1.5) node {Text at \verb!node 2!};
\end{tikzpicture}
```

显然, 卡尔不仅希望将节点放置在最后指定的位置上, 还希望将节点放置在这些位置的左边或右边。为此, 您放入图片中的每个节点对象都配备了几个锚点 (*anchors*)。例如, 锚点 `north` 在形状的上端的中间, 锚点 `south` 则在底部, 锚点 `north east` 在右上角。当你给出选项 `anchor=north` 时, 文本作为一个节点对象, 它的 `north` 锚点 (在文本的上端) 就会挂靠到当前位置, 因此最终的结果就是, 文本位于当前位置之下。卡尔用这个画出了下面的刻度:

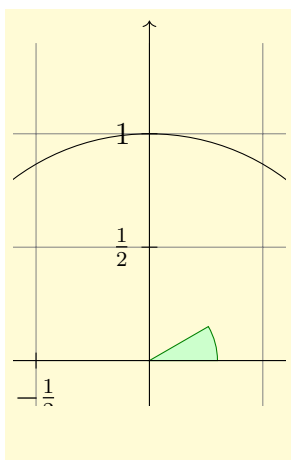


```
\begin{tikzpicture}[scale=3]
\clip (-0.6,-0.2) rectangle (0.6,1.51);
\draw[step=.5cm,help lines] (-1.4,-1.4) grid (1.4,1.4);
\filldraw[fill=green!20,draw=green!50!black] (0,0)
- (3mm,0mm)
arc [start angle=0, end angle=30, radius=3mm] - cycle;
\draw[->] (-1.5,0) - (1.5,0); \draw[->]
(0,-1.5) - (0,1.5);
\draw (0,0) circle [radius=1cm];
\foreach \x in {-1,-0.5,1}
\draw (\x cm,1pt) - (\x cm,-1pt)
node[anchor=north] {$\x$};
\foreach \y in {-1,-0.5,0.5,1}
\draw (1pt,\y cm) - (-1pt,\y cm)
node[anchor=east] {$\y$};
\end{tikzpicture}
```

这已经很不错了。使用这些锚, 卡尔现在可以添加大多数其他文本元素。然而, 卡尔认为, 虽然严格来说本意是为了将某物放置在给定点以下, 但为了做到这一点他必须使用锚点 `north`, 这是相当反直觉的。出于这个原因, 有一个选项叫做 `below`, 它的作用与 `anchor=north` 相同。类似地, `above right` 等同于 `anchor=south west`。此外, `below` 还可以接受一个可选的维度参数。如果给出了这个参数, 形状将以给定的量向下平移。因此, `below=1pt` 可以用于将文本标签放在某个点的下方 1pt 处。

卡尔对这些刻度不是很满意。他想要用 $1/2$ 或者 $\frac{1}{2}$ 来代替 0.5, 一部分出于卖弄 TeX 和 TikZ 强大的功能, 一部分出于表示像 $1/3$ 或 π 这样的数, 用“数学”刻度肯定比只有“数字”刻度更可取。另一方面, 他的学生相比 $\frac{1}{2}$ 更喜欢 0.5, 因为他们一般不太喜欢分数。

卡尔现在面临一个问题: 对于 `\foreach` 语句, 位置 `\x` 仍然应该是 0.5, 因为 TikZ 将不知道 `\frac{1}{2}` 应该被放在哪里。另一方面, 排版的文字应该确切地是 `\frac{1}{2}`, 为了解决这个问题, `\foreach` 提供了一种特殊的语法, 卡尔可以用斜杠分隔来指定两个或者更多的变量, 例如 `\x / \xtext`。其次, `\foreach` 在迭代元素的时候也应该按照 $\langle first \rangle / \langle second \rangle$ 。在每一轮迭代中, `\x` 会被设为 $\langle first \rangle$, 而 `\xtext` 会被设为 $\langle second \rangle$ 。如果没有指定 $\langle second \rangle$, 会再次使用 $\langle first \rangle$ 。所以, 下面是用来加刻度的新代码:

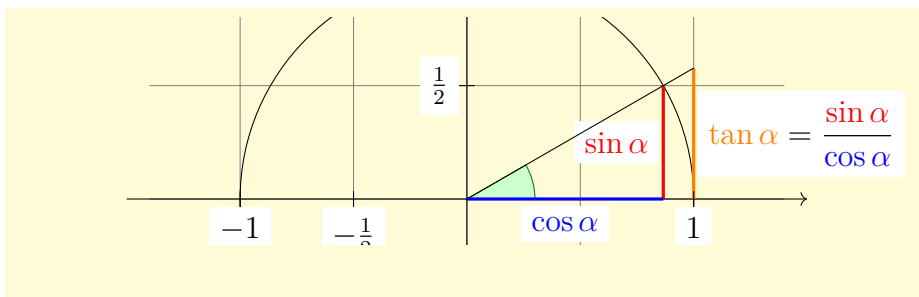


```
\begin{tikzpicture}[scale=3]
\clip (-0.6,-0.2) rectangle (0.6,1.51);
\draw[step=.5cm,help lines] (-1.4,-1.4)
grid (1.4,1.4);
\filldraw[fill=green!20,draw=green!50!black]
(0,0) -- (3mm,0mm)
arc [start angle=0, end angle=30, radius=3mm] -- cycle;
\draw[->] (-1.5,0) -- (1.5,0); \draw[->]
(0,-1.5) -- (0,1.5);
\draw (0,0) circle [radius=1cm];
\foreach \x/\xtext in {-1, -0.5/-\frac{1}{2}, 1}
\draw (\x cm,1pt) -- (\x cm,-1pt) node[anchor=north]
{\xtext};
\foreach \y/\ytext in
{-1, -0.5/-\frac{1}{2}, 0.5/\frac{1}{2}, 1}
\draw (1pt,\y cm) -- (-1pt,\y cm)
node[anchor=west] {\ytext};
\end{tikzpicture}
```

卡尔对结果非常满意, 但他的儿子指出, 这仍然不是完全令人满意的: 网格和圆圈干扰了数字, 降低了它们的易读性。卡尔对此并不十分关心 (他的学生甚至没有注意到), 但他的儿子坚持认为有一个简单的解决方案: 卡尔可以添加 `[fill=white]` 选项, 以将文本形状的背景填充为白色。

接下来卡尔想要做的是添加像 $\sin \alpha$ 这样的标签。为此, 他想要把标签放在“线

的中间”。为了做到这一点，卡尔可以在坐标之前的，-的后面直接指定标签 `node {\sin \alpha}`，而不是直接在直线的一个端点后面指定标签 `node {\sin \alpha}`。默认情况下，这将把标签放在线中间，但是 `pos=` 选项可以用来修改这一点。此外，可以使用 `near start` 和 `near end` 等选项来修改这个位置:

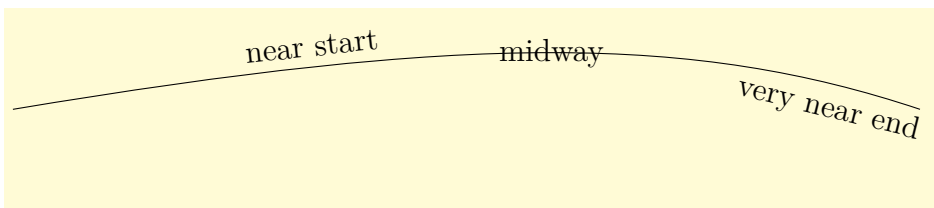


```

\usetikzlibrary {intersections}
\begin{tikzpicture}[scale=3]
  \clip (-2,-0.2) rectangle (2,0.8);
  \draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
  \filldraw[fill=green!20,draw=green!50!black] (0,0) - (3mm,0mm)
  arc [start angle=0, end angle=30, radius=3mm] - cycle;
  \draw[->] (-1.5,0) - (1.5,0) coordinate (x axis);
  \draw[->] (0,-1.5) - (0,1.5) coordinate (y axis);
  \draw (0,0) circle [radius=1cm];
  \draw[very thick,red]
  (30:1cm) - node[left=1pt,fill=white] {$\sin \alpha$} (30:1cm |- x axis);
  \draw[very thick,blue]
  (30:1cm |- x axis) - node[below=2pt,fill=white] {$\cos \alpha$} (0,0);
  \path [name path=upward line] (1,0) - (1,1);
  \path [name path=sloped line] (0,0) - (30:1.5cm);
  \draw [name intersections={of=upward line and sloped line, by=t}]
  [very thick,orange] (1,0) - node [right=1pt,fill=white]
  {$\displaystyle \tan \alpha \color{black} = \frac{\color{red}\sin \alpha}{\color{blue}\cos \alpha}$} (t);
  \draw (0,0) - (t);
  \foreach \x/\xtext in {-1, -0.5/-\frac{1}{2}, 1}
  \draw (\x cm,1pt) - (\x cm,-1pt) node[anchor=north,fill=white] {$\xtext$};
  \foreach \y/\ytext in {-1, -0.5/-\frac{1}{2}, 0.5/\frac{1}{2}, 1}
  \draw (1pt,\y cm) - (-1pt,\y cm) node[anchor=west,fill=white] {$\ytext$};
\end{tikzpicture}

```

你还可以在曲线上放置标签, 并通过添加倾斜选项, 使它们旋转, 以匹配直线的斜率。下面是一个例子:



```

\begin{tikzpicture}
\draw (0,0) .. controls (6,1) and (9,1) ..
node[near start,sloped,above] {near start}
node {midway}
node[very near end,sloped,below] {very near end} (12,0);
\end{tikzpicture}

```

仍需在图片右侧绘制解释性文字。这里的主要困难在于限制文本“标签”的宽度,因为它相当长,所以需要使用换行。幸运的是,卡尔可以使用 `text width=6cm` 的选项来获得想要的效果。下面是完整的代码:

```

\begin{tikzpicture}
[scale=3,line cap=round,
% Styles
axes/.style=,
important line/.style={very thick},
information text/.style={rounded corners,fill=red!10,inner sep=1ex}]
% Colors
\colorlet{anglecolor}{green!50!black}
\colorlet{sincolor}{red}
\colorlet{tancolor}{orange!80!black}
\colorlet{coscolor}{blue}
% The graphic
\draw[help lines,step=0.5cm] (-1.4,-1.4) grid (1.4,1.4);
\draw (0,0) circle [radius=1cm];
\begin{scope}[axes]
\draw[->] (-1.5,0) - (1.5,0) node[right] {$x$} coordinate(x axis);
\draw[->] (0,-1.5) - (0,1.5) node[above] {$y$} coordinate(y axis);
\foreach \x/\xtext in {-1, -.5/-\frac{1}{2}, 1}
\draw[xshift=\x cm] (0pt,1pt) - (0pt,-1pt) node[below,fill=white] {$\xtext$};
\foreach \y/\ytext in {-1, -.5/-\frac{1}{2}, .5/\frac{1}{2}, 1}
\draw[yshift=\y cm] (1pt,0pt) - (-1pt,0pt) node[left,fill=white] {$\ytext$};
\end{scope}
\filldraw[fill=green!20,draw=anglecolor] (0,0) - (3mm,0pt)
arc [start angle=0, end angle=30, radius=3mm];

```

```

\draw (15:2mm) node[anglecolor] {\alpha$};
\draw[important line,sincolor]
(30:1cm) - node[left=1pt,fill=white] {\sin \alpha$} (30:1cm |- x axis);
\draw[important line,coscolor]
(30:1cm |- x axis) - node[below=2pt,fill=white] {\cos \alpha$} (0,0);
\path [name path=upward line] (1,0) - (1,1);
\path [name path=sloped line] (0,0) - (30:1.5cm);
\draw [name intersections={of=upward line and sloped line, by=t}]
[very thick,orange] (1,0) - node [right=1pt,fill=white]
{\displaystyle \tan \alpha \color{black}=
\frac{{\color{red}\sin \alpha}}{{\color{blue}\cos \alpha}}$} (t);
\draw (0,0) - (t);
\draw[xshift=1.85cm]
node[right,text width=6cm,information text]
{
The {\color{anglecolor} angle \alpha$} is $30^\circ$ in the
example ($\pi/6$ in radians). The {\color{sincolor}sine of
\alpha$}, which is the height of the red line, is
\[
{\color{sincolor} \sin \alpha} = 1/2.
\]
By the Theorem of Pythagoras ...
};
\end{tikzpicture}

```

1.22 再探角度

卡尔估计他所创造的图片的某些部分的代码可能非常有用, 而且他可能希望在将来重用它们。一个自然的做法是创建 $\text{T}_\text{E}\text{X}$ 宏来存储他希望重用的代码。然而, TikZ 提供了另一种直接集成到其解析器中的方法: `pic`!

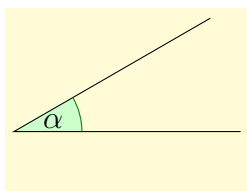
一个 “`pic`” 指的是 “不完全完整的图片”, 因此得名。其思想是, `pic` 只是一些代码, 然后你可以使用 `pic` 命令在不同的位置添加到图片中, 其语法和 `node` 命令几乎相同。主要的区别是, 与在花括号中指定一些文本不同的是, 此处是指定应该显示的预定义图片的名称。

定义新 `pics` 很简单, 参见第 18 节, 但现在我们只想使用这样一个预定义的

Link

`pic:angle` pic。顾名思义，它是一个由一个小楔形和一个弧和一些文本组成的小角度图（卡尔需要加载下面的例子的 `angle` 库和 `quotes`）。这个楔形的大小也会被自动计算。

`angle` pic 绘制线 BA 和 BC 之间的一个角， A, B 和 C 分别是三个坐标。在我们的例子中， B 是原点， A 是 x 轴上的某个点，而 C 是 30° 线上的某一点。



```
\usetikzlibrary {angles,quotes}
\begin{tikzpicture}[scale=3]
\coordinate (A) at (1,0);
\coordinate (B) at (0,0);
\coordinate (C) at (30:1cm);
\draw (A) - (B) - (C)
pic [draw=green!50!black, fill=green!20,
angle radius=9mm,
"$\alpha$"] {angle = A-B-C};
\end{tikzpicture}
```

让我们看看发生了什么。首先我们通过 `\coordinate` 命令声明了三个坐标。这使得我们可以用名字具体指定任意坐标。然后我们用 `\draw` 作为开头，但是引入了 `pic` 指令。这个指令用中括号接收很多选项参数并用花括号接收最重要的东西：我们声明要添加 `angle` pic，并且这个角在我们命名为 A, B 和 C 的三点间。请注意，我们希望在 `pic` 中显示的文本是在 `pic` 选项内的引号中指定的，而不是在花括号内。

要了解更多关于 `pics` 的信息，请参见第 18 节。

第二章 教程：哈根的 Petri-Net

2.1 问题描述

2.2 设置环境

2.3 节点简介

2.4 使用 At 语法放置节点

2.5 节点大小

2.6 命名节点

2.7 使用相对位置放置节点

2.8 为节点添加标签

2.9 连接节点

2.10 添加波浪线和多行文本

2.11 使用图层：矩形背景

2.12 完整代码