Pionpill's TikZ Notebook

Pionpill [1]

学习 TikZ 的笔记，正在学习中，有待更新
本文主要根据 TikZ 官方手册 [2] 学习

2021 年 12 月 16 日

[1]笔名：北岸，电子邮件: 673486387@qq.com，Github：https://github.com/Pionpill
[2]TikZ Github：https://github.com/pgf-tikz/pgf

目录

I 案例

II 语法

III 百科

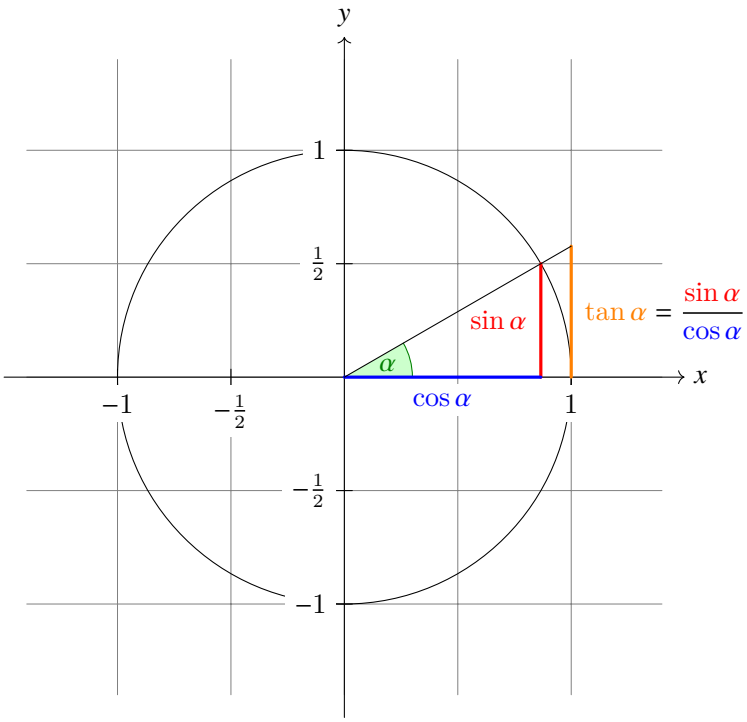
I 案例

次章节源于 TikZ 手册[1] 的学习

一、三角函数：A Picture for Karl's Students

1.1 效果预览

这个案例的最终效果如下



The angle α is 30° in the example ($\pi/6$ in radians). The sine of α, which is the height of the red line, is

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

By the Theorem of Pythagoras ...

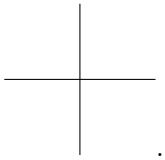
图 1.1.1

启用 tikz 环境：

```
\begin{tikzpicture}
    ......
\end{tikzpicture}
```

画坐标轴：

[1] TikZ Github 仓库：TikZGithub https://github.com/pgf-tikz/pgf



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

图 1.1.2 直线

两种 TikZ 画法：

1. 通过 TikZ 环境，如上所述
2. 行内 TikZ ，如 ——————

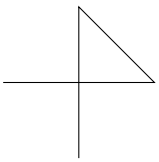
在不同的环境下 TikZ 有多种写法，这里不做说明

1.2 基本曲线

1.2.1 直线

直线通过 `\draw` 指令绘制：

```
\draw (x,y) -- (x,y) -- (x,y) ... ;
```



```
\tikz \draw (-1,0) -- (1,0) -- (0,1) -- (0,-1);
```

图 1.1.3 直线

1.2.2 曲线

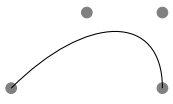
TikZ 的曲线原理在官方文档中没有详细解释，类似于贝塞尔曲线，我的理解如下：

- 和直线类似，曲线也由点组成，和分为起止点，过程点
- 起止点为曲线的起点和终点，必须经过
- 过程点比必须经过，曲线有趋近的趋势

通过 `\.. controls` 关键字控制曲线

```
Official: <start point> .. controls <1st control point> and <2nd control point> .. <end point>;
Simple: \draw (x,y) .. controls (x,y) and (x,y) .. (x,y);
```

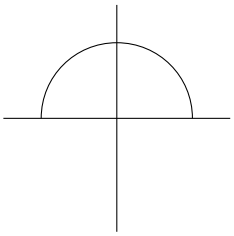
可以取消.. 这样会让第一个点使用两次



```
\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}
```

图 1.1.4 曲线

继续坐标系的例子，可以画出伪半圆，注意下面的例子中，(0,1) 点既作为终点又作为起点



```
\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}
```

图 1.1.5 带半圆曲线的坐标轴

1.2.3 椭圆

可以通过如下方式绘制椭圆

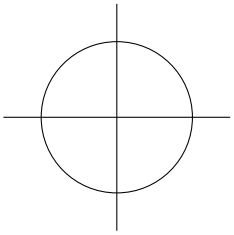
```
\draw <point> <category> [args];
```

其中：<point>: 圆心 <category>: 椭圆类型 [args]: 对应的参数例如下面绘制的圆和椭圆

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

图 1.1.6 椭圆

继续在坐标轴上加上圆



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

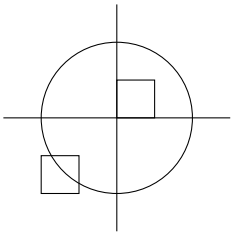
图 1.1.7 带单位圆的坐标轴

1.2.4 矩形

矩形的绘制方式和椭圆几乎一致，只需要将下面 category 写为 rectangle

```
\draw <point> <category> [args];
```

在上述坐标轴基础上继续绘制矩形



```
\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}
```

图 1.1.8 带矩形的单位圆

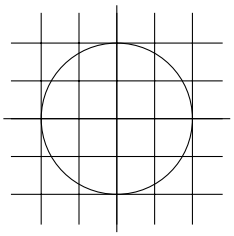
1.3 网格

1.3.1 网格的绘制

网格的绘制形式：

```
\draw[step=2pt] (0,0) grid (10pt,10pt);
```

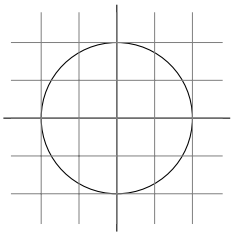
与曲线不同的是，第一个坐标点表示网格一个角，第二个坐标点则表示另一个不相邻的角，由此，我们可以在坐标轴基础上绘制网格背景了



```
\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}
```

图 1.1.9 带网格的坐标轴

网格的颜色与单位圆相同，不易区别，这里使用更多的参数来控制



```
\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,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
    \end{tikzpicture}
```

图 1.1.10 修改颜色的带网格的坐标轴

1.3.2 样式设置

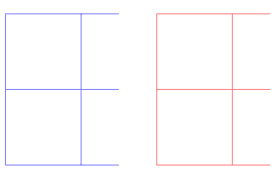
图像中的样式对于进场重复使用的样式，我们可以设置一个统一样式，如网格虚线，我们定义为辅助线 help lines。

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

全局样式除了在图像中定义样式供单个图像使用，还可以定义全局样式。全局样式允许嵌套

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

样式同时可以设置参数



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

图 1.1.11 带参数的样式

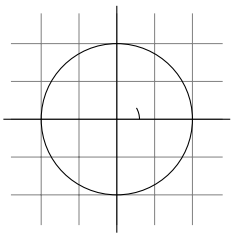
网格除了以上重要参数，还有虚线参数可以设置 dash pattern

1.4 弧度

绘制弧度的形式：

```
\draw (x,y) arc [args]
```

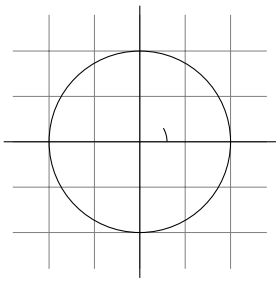
和网格类使，第一个坐标表示弧线起点，arc 为关键词，之后为样式参数，默认为逆时针旋转



```
\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}
```

图 1.1.12 带弧度的单位圆

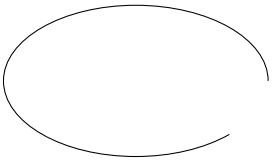
如果对图像大小不满意，可以使用 tikzpicture 的 scale 参数进行放大



```
\begin{tikzpicture}[scale=1.2]
    \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=3
        mm];
    \end{tikzpicture}
```

图 1.1.13 放大后的图像

除了圆弧，还能绘制椭圆弧



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

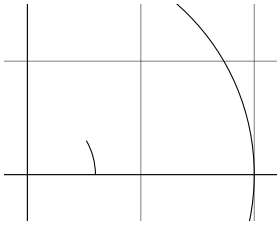
图 1.1.14 椭圆弧

1.5 截图

截图的用法与网格十分相识，可以指定截图方式，以及截图的对角来划定区域

```
\clip (x,y) graph (x,y)
```

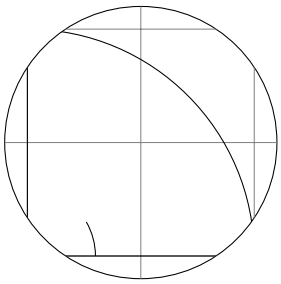
需要注意的是，clip 指令需要写在前面的位置，下面使用矩形截图：



```
\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=3
        mm];
\end{tikzpicture}
```

图 1.1.15 矩形截图

下面结合 draw 绘制圆形截图，技术细节不在此讨论



```
\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=3
        mm];
\end{tikzpicture}
```

图 1.1.16 结合 draw 绘制圆形截图

1.6 抛物线与三角函数

1.6.1 抛物线

抛物线同样使用 draw 命令绘制

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

图 1.1.17 抛物线

其他形式的抛物线

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

图 1.1.18 其他形式抛物线

1.6.2 三角函数

三角函数同样使用 draw 指令绘制

```
\draw[x=,y=] (x,y) sin (x,y)
```

其中，可选项 [x=,y=] 用于控制图像比例，第一个点为起点，随后指定三角函数类型，最后一个点为终点

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

图 1.1.19　部分三角函数图像

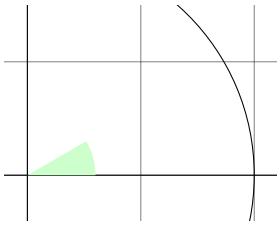
```
\tikz \draw [x=1.57ex,y=1ex] (0,0) sin(1,1) cos(2,0) sin(3,-1)
    cos(4,0) (0,1) cos(1,0) sin(2,-1) cos(3,0) sin(4,1);
```

图 1.1.20　三角函数图像

1.7 填充与边线

1.7.1 基本填充与边线

使用 fill 代替 draw 关键字可绘制填充图形



```
\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}
```

图 1.1.21　填充图形

边框线：useasboundingbox

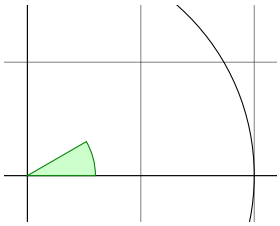
```
\useasboundingbox (low,high)
```

low 和 high 分别代表原边界向内，向外扩展，可使用 cycle 来指明曲线闭合

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

图 1.1.22　边框粗细与闭合

可以使用 filldraw 绘制有边界线和填充的图形



```
\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}
```

图 1.1.23 filldraw 效果

1.7.2 渐变

与填充类使，渐变也有两种画法 shade 与 shadedraw

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

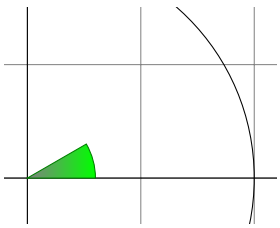
图 1.1.24 shade 效果

更详细的渐变设置

```
\begin{tikzpicture}[scale = 1]
    \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}
```

图 1.1.25 几种渐变样式

尝试绘制新的圆弧



```
\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}
```

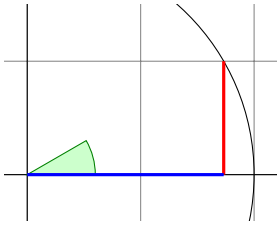
图 1.1.26 带渐变的圆弧

1.8 指定坐标

TikZ 指定坐标需要依靠数学计算，主要有两种方式

1. 平面直角坐标系：例如：(10pt,2cm) 代表 x 方向偏移 10pt, y 方向偏移 2cm
2. 极坐标系：例如：(30:1cm) 代表，逆时针选择 30°, 长度为 1cm

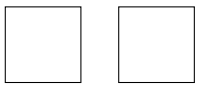
在确定坐标之后便可以依照新的点绘制图像，例如 +(0cm,1cm) 代表在坐标上方绘制 1cm 长线，++(2cm,0cm) 则再次绘制线段



```
\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}
```

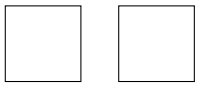
图 1.1.27 指定坐标绘制线段

从上述的例子可以发现，– 承担了指定线段类型的责任，若没有 – 则不会绘制线段，可以将这种用法用在定位点上，下面用几个例子详细说明 + 与 ++ 的作用



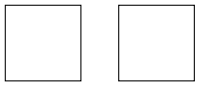
```
\begin{tikzpicture}[scale = 1]
    \def\rectanglepath{-- ++(1cm,0cm) -- ++(0cm,1cm) -- ++(-1cm
        ,0cm) -- cycle}
    \draw (0,0) \rectanglepath;
    \draw (1.5,0) \rectanglepath;
\end{tikzpicture}
```

图 1.1.28 ++ 绘制矩形



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

图 1.1.29 + 绘制矩形



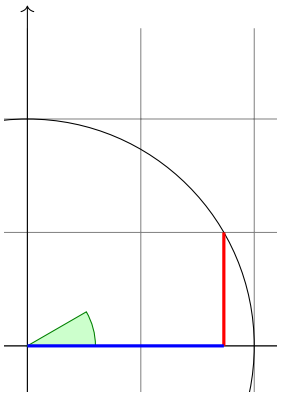
```
\tikz \draw (0,0) rectangle +(1,1) (1.5,0) rectangle +(1,1);
```

图 1.1.30 更快速的矩形绘制方式

1.9 线段与箭头

1.9.1 箭头样式

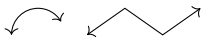
使用 -> 代替 – 绘制有箭头的坐标轴



```
\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);
\end{tikzpicture}
```

图 1.1.31 带箭头的坐标轴

可以在 draw 可选参数中使用箭头来指明箭头



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

图 1.1.32 箭头示例

使用不一样的箭头样式

```
\usetikzlibrary {arrows.meta}
\begin{tikzpicture}[scale = 1,>=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,10py);
\end{tikzpicture}
```

图 1.1.33 箭头样式

1.9.2 范围 scope

对于经常出现的同类样式，可以用 scope 设为统一样式

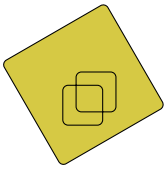
```
\begin{tikzpicture}[scale = 1,ultra thick]
    \draw (0,0) -- (0,1);
    \begin{scope}[thin]
        \draw (1,0) -- (1,1);
        \draw (2,0) -- (2,1);
    \end{scope}
    \draw (3,0) -- (3,1);
\end{tikzpicture}
```

图 1.1.34 scope 的作用

scope 还有一个重要作用，在 scope 内的操作只会影响内部范围

1.10 位移

可以使用 shift 关键字调整位置



```
\begin{tikzpicture}[scale = 1,rounded corners=2pt,x=15pt,y=15pt]
   \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}
```

图 1.1.35 位移示范

1.11 循环

1.11.1 数字循环

使用 foreach 指令，进行数字循环

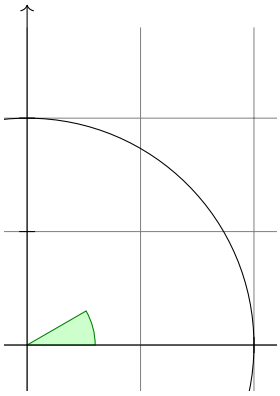
```
\foreach <variable> in <values> <commands>
```

$x = 1$，$x = 2$，$x = 3$，

```
\begin{tikzpicture}[scale = 1]
   \foreach \x in {1,2,3} {$x = \x$,}
\end{tikzpicture}
```

图 1.1.36 foreach 数字循环

同样，我们可以将其应用到绘图中



```
\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}
```

图 1.1.37 循环绘制刻度线

再进一步，我们可以设计两个变量，绘制二维矩阵

1,5	2,5	3,5	4,5	5,5
1,4	2,4	3,4	4,4	5,4
1,3	2,3	3,3	4,3	5,3
1,2	2,2	3,2	4,2	5,2
1,1	2,1	3,1	4,1	5,1

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

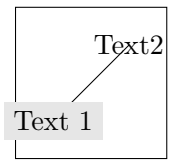
```
\begin{tikzpicture}[scale = 1]
   \foreach \x in {1,2,...,5,7,8,...,12}
      \foreach \y in {1,...,5} {
      \draw (\x,\y) +(-.5,-.5) rectangle ++(.5,.5);
      \draw (\x,\y) node{\x,\y};
      }
\end{tikzpicture}
```

图 1.1.38　二维矩阵

1.12 增加文字

1.12.1 node 关键字

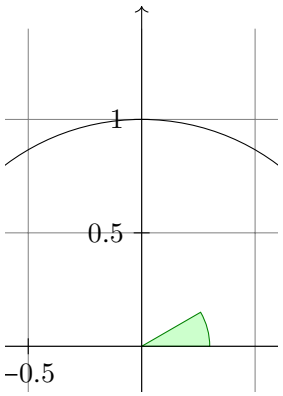
TikZ 可以通过 node 关键字指定节点并绘制区局，以达到添加文字的效果




```
\begin{tikzpicture}[scale = 1]
   \draw (0,0) rectangle (2,2);
   \draw (0.5,0.5) node [fill=grey!20] {Text 1} -- (1.5,1.5)
        node {Text2}
\end{tikzpicture}
```

图 1.1.39　node 关键字的作用

默认将以 node 为几何中心构建方形区域，但也可自定义文字位置

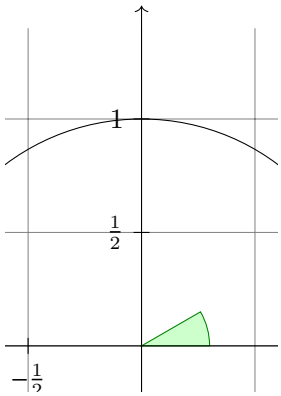



```
\begin{tikzpicture}[scale = 3]
   \clip (-0.6,-0.2) rectangle (0.6,1.51);
   \draw[step=.5cm,help lines] (-1.4,-1.4) grid (1.4,1.4);
   \filldraw[fill=green!20,draw=green!50!black] (0,0) -- (3mm,0mm)
        arc [start angle=0, end angle=30, radius=3mm] -- cycle;
   \draw[->] (-1.5,0) -- (1.5,0); \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}
```

图 1.1.40　绘制坐标系制度

如果需要使用分数形式的文本，可以使用如下形式

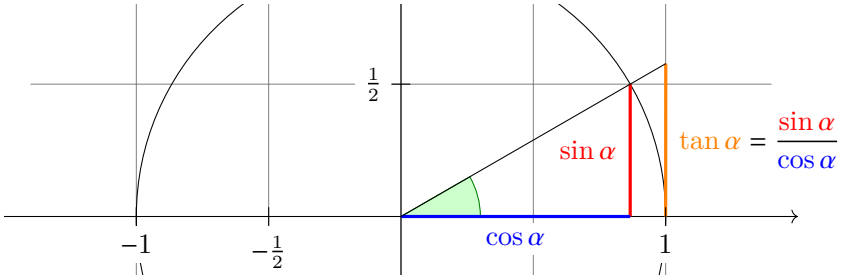
```
\x / \xtex
```




```
\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=east] {$\ytext$};
\end{tikzpicture}
```

图 1.1.41 绘制坐标系制度 (分数形式)

接着之前的直角坐标系，我们可以标出函数 (以下为官方代码，出现了部分前面未说明的内容)

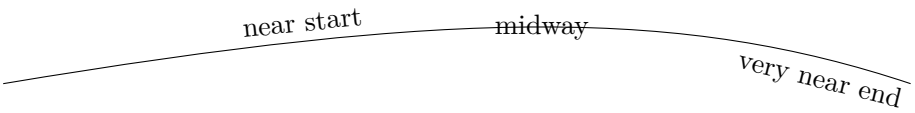



```
\begin{tikzpicture}[scale=3.5]
    \clip (-2,-0.22) rectangle (2,0.8);
    \draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
    \filldraw[fill=green!20,draw=green!50!black] (0,0) -- (3mm,0mm) arc [start angle=0, end angle=30,
         radius=3mm] -- cycle;
    \draw[->] (-1.5,0) -- (1.5,0) 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=east,fill=white] {$\ytext$};
\end{tikzpicture}
```

图 1.1.42 带文字的单位圆

1.12.2 线段文字

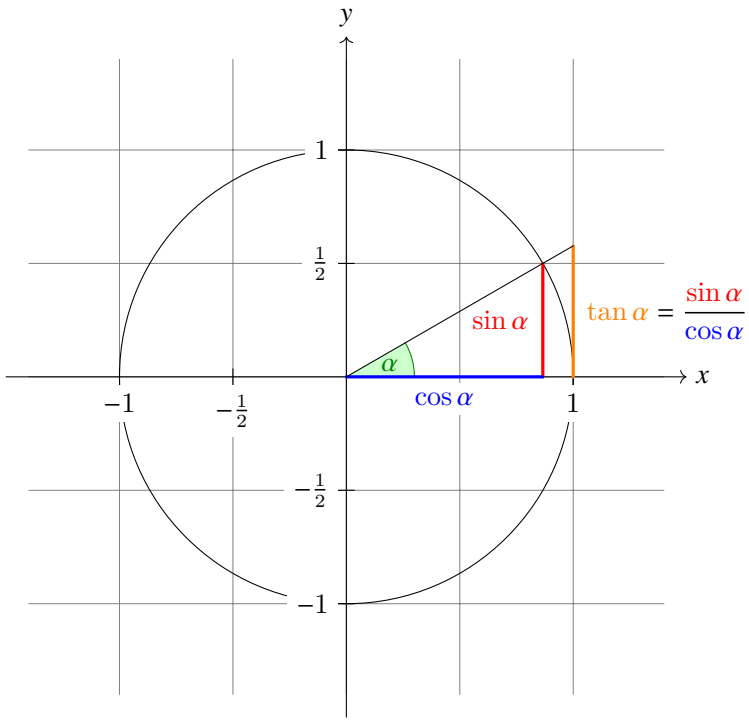
可以通过 sloped 关键字指定在线段的某一区域添加文字




```
\begin{tikzpicture}[scale = 1]
    \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}
```

图 1.1.43　sloped 控制段落文字位置

1.13 最终结果



The angle α is 30° in the example ($\pi/6$ in radians). The sine of α, which is the height of the red line, is

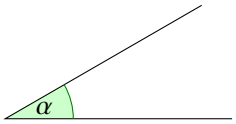
$$\sin\alpha = 1/2.$$

By the Theorem of Pythagoras ...

图 1.1.44

1.14 其他技巧

可以使用 coordinate 指定点位置并赋予别名，并通过 pic 来进行快速绘图，pic 这里只做演示，详细内容后续章节会讲解




```
\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}
```

图 1.1.45　使用 pic 快速绘制

二、流程图：Diagrams as Simple Graphs

2.1 效果预览

这个案例的最终效果如下

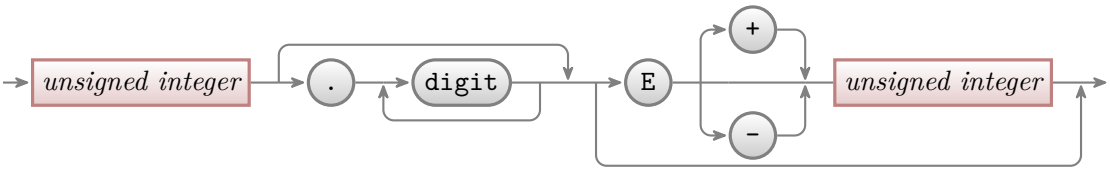


图 1.2.1　流程图效果预览

2.2 节点设置

2.2.1 节点样式

可以使用关键词 /.style 设置一个同一的样式


```
\begin{tikzpicture}[
      nonterminal/.style={
      rectangle,
      minimum size=6mm,
      very thick,
      draw=red!50!black!50,
      top color=white,
      bottom color=red!50!black!20,
      font=\itshape
   }]
   \node [nonterminal] {unsigned integer};
\end{tikzpicture}
```

图 1.2.2　基本节点样式

使用 rounded corner 进行圆角设置：


```
\begin{tikzpicture}[
   node distance = 5mm, terminal/.style = {
      rectangle,minimum size=6mm,rounded corners = 3mm,
      very thick, draw = black!50,
      top color = white, bottom color = black!20,
      font = \ttfamily
   }]
   \node (dot) [terminal] {.};
   \node (digit) [terminal,right=of dot] {digit};
   \node (E) [terminal,right=of digit] {E};
\end{tikzpicture}
```

图 1.2.3　节点样式：圆角

使用 shapes.misc 包的圆角设置:rounded rectangle


```
\usetikzlibrary {positioning}
\begin{tikzpicture}[node distance=5mm,
   terminal/.style={
      rounded rectangle,
      minimum size=6mm,
      very thick,draw=black!50,
      top color=white,bottom color=black!20,
      font=\ttfamily
   }]
   \node (dot) [terminal] {.};
   \node (digit) [terminal,right=of dot] {digit};
   \node (E) [terminal,right=of digit] {E};
\end{tikzpicture}
```

图 1.2.4 rounded 圆角节点

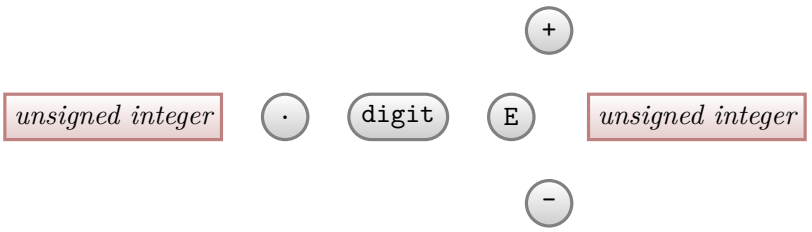
在上述的几个例子中，我们发现. 与其他字符并没有对齐，. 看起来更像 •。这是由于每个节点内容拥不同的高度与字母深度。可以使用 text height, text depth 来进行调整


```
\begin{tikzpicture}[
   node distance=5mm, text height=1.5ex, text depth = .25ex,
   terminal/.style={
      rounded rectangle,
      minimum size=6mm,
      very thick,draw=black!50,
      top color=white,bottom color=black!20,
      font=\ttfamily
   }]
   \node (dot) [terminal] {.};
   \node (digit) [terminal,right=of dot] {digit};
   \node (E) [terminal,right=of digit] {E};
\end{tikzpicture}
```

图 1.2.5 文字对齐

2.2.2 节点位置

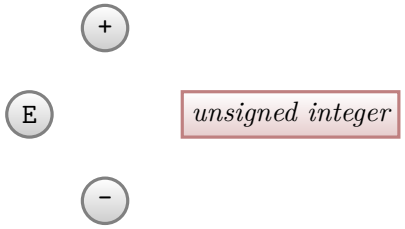
使用位置选项 position options 设置节点相对位置




```
\begin{tikzpicture}[node distance=5mm and 5mm]
    \node (ui1) [nonterminal] {unsigned integer};
    \node (dot) [terminal,right=of ui1] {.};
    \node (digit) [terminal,right=of dot] {digit};
    \node (E) [terminal,right=of digit] {E};
    \node (plus) [terminal,above right=of E] {+};
    \node (minus) [terminal,below right=of E] {-};
    \node (ui2) [nonterminal,below right=of plus] {unsigned integer};
\end{tikzpicture}
```

图 1.2.6 节点位置

精确控制偏移距离的方法之一：使用 xshift yshift




```
\begin{tikzpicture}[scale = 1,node distance = 5mm and 5mm]
    \node (E) [terminal] {E};
    \node (plus) [terminal,above right=of E,xshift=5mm] {+};
    \node (minus) [terminal,below right=of E,xshift=5mm] {-};
    \node (ui2) [nonterminal,below right=of plus,xshift=5mm] {
        unsigned integer};
\end{tikzpicture}
```

图 1.2.7 节点位置精确控制

2.2.3 节点连线

tikz 中的节点连接，尤其是非直线连接是一件非常复杂的事情。


```
\begin{tikzpicture}[node distance=5mm and 5mm]
    \node (dot) [terminal] {.};
    \node (digit) [terminal,right=of dot] {digit};
    \node (E) [terminal,right=of digit] {E};
    \path (dot) edge[->] (digit) % simple edges
        (digit) edge[->] (E);
    \draw [->]
        % start right of digit.east, that is, at the point that
            is the linear combination of digit.east and the
            vector (2mm,0pt). We use the ($ ... $) notation for
             computing linear combinations
        ($ (digit.east) + (2mm,0) $)
        % Now go down
        -- ++(0,-.5)
        % And back to the left of digit.west
        -| ($ (digit.west) - (2mm,0) $);
\end{tikzpicture}
```

图 1.2.8 节点连线

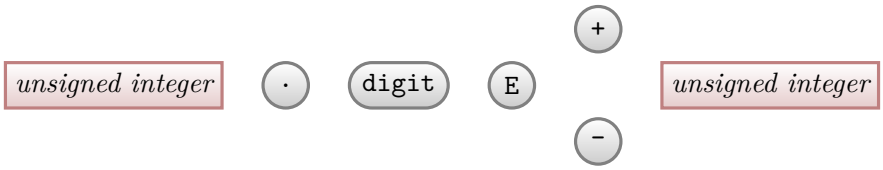
在此基础上，tikz 引入了 edge 参数来简化折线。


```
\begin{tikzpicture}[
    node distance=5mm and 5mm,skip loop/.style={to path={--
        ++(0,-.5) -| (\tikztotarget)}}]
    \node (dot) [terminal] {.};
    \node (digit) [terminal,right=of dot] {digit};
    \node (E) [terminal,right=of digit] {E};
    \path (dot) edge[->] (digit) % simple edges
        (digit) edge[->] (E)
        ($ (digit.east) + (2mm,0) $)
            edge[->,skip loop] ($ (digit.west) - (2mm,0) $);
\end{tikzpicture}
```

图 1.2.9 简化的节点连线

2.3 矩阵布局

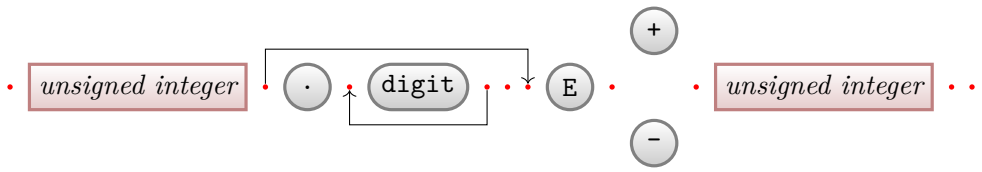
矩阵式布局与 latex 自身的数学公式布局十分相似




```
\begin{tikzpicture}
    \matrix[row sep=1mm,column sep=5mm] {
        % First row:
        & & & & \node [terminal] {+}; & \\
        % Second row:
        \node [nonterminal] {unsigned integer}; &
        \node [terminal] {.}; &
        \node [terminal] {digit}; &
        \node [terminal] {E}; &&
        \node [nonterminal] {unsigned integer}; \\
        % Third row:
        & & & & \node [terminal] {-}; & \\
    };
\end{tikzpicture}
```

图 1.2.10 矩阵布局

使用了矩阵布局后，连线将变得非常简单



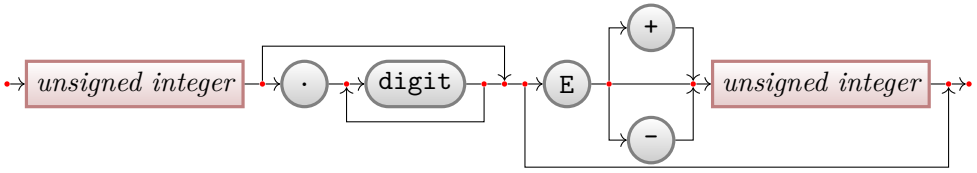

```
\begin{tikzpicture}
   [point/.style={circle,inner sep=0pt,minimum size=2pt,fill=red},skip loop/.style={to path
        ={-- ++(0,#1) -| (\tikztotarget)}}]
   \matrix[row sep=1mm,column sep=2mm] {
      % First row:
      & & & & & & & & & \node (plus) [terminal] {+};\\
      % Second row:
      \node (p1) [point] {}; & \node (ui1) [nonterminal] {unsigned integer}; &
      \node (p2) [point] {}; & \node (dot) [terminal] {.}; &
      \node (p3) [point] {}; & \node (digit) [terminal] {digit}; &
      \node (p4) [point] {}; & \node (p5) [point] {}; &
      \node (p6) [point] {}; & \node (e) [terminal] {E}; &
      \node (p7) [point] {}; & &
      \node (p8) [point] {}; & \node (ui2) [nonterminal] {unsigned integer}; &
      \node (p9) [point] {}; & \node (p10) [point] {};\\
      % Third row:
      & & & & & & & & & \node (minus)[terminal] {-};\\
   };
   \path (p4) edge [->,skip loop=-5mm] (p3)
         (p2) edge [->,skip loop=5mm] (p6);
\end{tikzpicture}
```

图 1.2.11 矩阵布局与连线

2.4 连线

2.4.1 连接节点

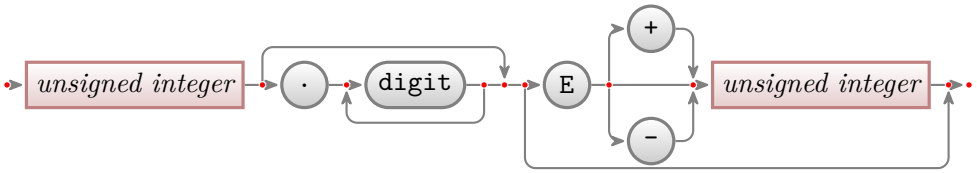
使用 graph 命令 (path graph 的简写) 绘制连线




```
\begin{tikzpicture}
    [point/.style={circle,inner sep=0pt,minimum size=2pt,fill=red},
        skip loop/.style={to path={-- ++(0,#1) -| (\tikztotarget)}},
        hv path/.style={to path={-| (\tikztotarget)}},
        vh path/.style={to path={|- (\tikztotarget)}}]
    \matrix[row sep=1mm,column sep=2mm] {
        ......
    };
    \graph {
        (p1) -> (ui1) -- (p2) -> (dot) -- (p3) -> (digit) -- (p4)
            -- (p5) -- (p6) -> (e) -- (p7) -- (p8) -> (ui2) -- (p9) -> (p10);
        (p4) -> [skip loop=-5mm] (p3);
        (p2) -> [skip loop=5mm] (p5);
        (p6) -> [skip loop=-11mm] (p9);
        (p7) -> [vh path] (plus) -> [hv path] (p8);
        (p7) -> [vh path] (minus) -> [hv path] (p8);
    };
\end{tikzpicture}
```

图 1.2.12　graph 连线

2.4.2 连线样式




```
\begin{tikzpicture}
    [>={Stealth[round]},thick,black!50,text=black,
        every new ->/.style={shorten >=1pt},
        graphs/every graph/.style={edges=rounded corners},
        point/.style={circle,inner sep=0pt,minimum size=2pt,fill=red},
        skip loop/.style={to path={-- ++(0,#1) -| (\tikztotarget)}},
        hv path/.style={to path={-| (\tikztotarget)}},
        vh path/.style={to path={|- (\tikztotarget)}}]
    \matrix[row sep=1mm,column sep=2mm] {
        ......
    };
    \graph [use existing nodes] {
        p1 -> ui1 -- p2 -> dot -- p3 -> digit -- p4 -- p5 -- p6 -> e -- p7 -- p8 -> ui2 -- p9
             -> p10;
        p4 ->[skip loop=-5mm] p3;
        p2 ->[skip loop=5mm] p5;
        p6 ->[skip loop=-11mm] p9;
        p7 ->[vh path] { plus, minus } -> [hv path] p8;
    };
\end{tikzpicture}
```

图 1.2.13　连线样式

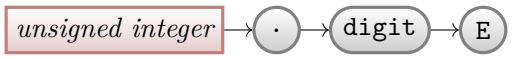
2.5 Graph 指令

使用 graph 节点可以快速绘制简单的流程图


```
\tikz \graph [grow right=2cm] {unsigned integer -> d -> digit -> E};
```

图 1.2.14　graph 指令绘图

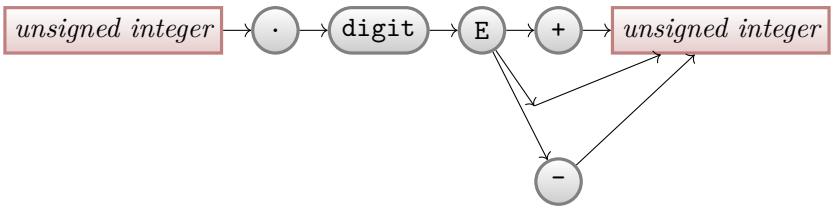
声明 grow right sep 指明让节点在前一节点右方生成




```
\tikz \graph [grow right sep] {
   unsigned integer[nonterminal] -> "."[terminal] -> digit[terminal] -> E[terminal]
};
```

图 1.2.15

创建分支




```
\tikz \graph [grow right sep] {
   unsigned integer [nonterminal] ->
   "." [terminal] ->
   digit [terminal] ->
   E [terminal] ->
   {
      "+" [terminal],
      "" [coordinate],
      "-" [terminal]
   } ->
   ui2/unsigned integer [nonterminal]
};
```

图 1.2.16

II 语法

一、Basic

1.1 TikZ 的导入与使用

导入 TikZ 包命令[1],TikZ 包并没有任何可选项 (options)：

```
\usepackage{tikz}
```

在 TikZ 被成功导入后，还可以使用它的进阶包：

```
\usetikzlibrary{<list of libraries>}
```

这将导入对应的文件[2]：tikzlibrary<library>.code.tex。这些进阶包将提供更多的样式，指令等。

1.2 创建 TikZ 图片

1.2.1 标准 TikZ 环境

标准的 TikZ 绘图环境如下：

```
\begin{tikzpicture}<animation spec>[<options>]
   <environment contents>
\end{tikzpicture}
```

所有的 TikZ 绘图指令都应该在此环境内给出，<animation options> 确保动画相关包被导入，[<options>] 则表示应用于全环境的一些修饰。

在绘图过程中，TikZ 会计算每一个坐标 (coordinate) 的位置以获得图片范围，通常这种以坐标确定边界的方式十分好用，但也可能获得糟糕的效果，比如线条过粗的时候，TikZ 不会考虑线条宽度；再比如绘制曲线时的控制点有时候离主要绘图区域过远会导致图片过大。可以使用 [use as bounding box] 选项来避免这些现象的发生。

TikZ 提供了精确控制图片位置的几种方式如下 (通过控制基准线控制图片位置):

- baseline = <dimension or coordinate or default> (默认: 0pt)

  该修饰可改变图片底部位置，值既可以是具体的长度，也可以是点。

```
\tikz            \draw(0,0)circle(.5ex);
\tikz[baseline=0pt] \draw(1,0)circle(.5ex);
```

图 2.1.1 baseline: 值为具体长度

[1]本文只说明在 Latex2e 环境下 TikZ 的使用

[2]这涉及到 tex 的语言处理，本人没有深入研究，非专业研究 tex 语言也不建议深究


```
\begin{tikzpicture}[baseline=(X.base)]
   \node [cross out,draw] (X) {world.};
\end{tikzpicture}
```

图 2.1.2 baseline: 值为坐标

- execute at begin/end picture = <code>
  顾名思义，可以提前/后续执行一些指令。


```
\begin{tikzpicture}[execute at end picture={
   \begin{pgfonlayer}{background}
   \path[fill=yellow,rounded corners]
   (current bounding box.south west) rectangle
   (current bounding box.north east);
   \end{pgfonlayer}
}]
\node at (0,0) {X};
\node at (2,1) {Y};
      \end{tikzpicture}
```

图 2.1.3 execute at begin/end picture

- every picture
  对所有指令起作用，经常使用 every picture/.style = ... 形式指定全局修饰

1.2.2 简化的 TikZ 指令

除了使用 LaTeX 下的标准 TikZ 环境，还可以使用如下简化指令绘图。

```
\tikz <animation spec> [<options>] {<path commands>}
```

1.2.3 图片背景

通常情况下，TikZ 绘制的图片是透明底色的，由于背景色不经常被使用，相关操作被移到了 background 包下。

1.3 使用范围 (scope) 构建图片

1.3.1 scope 环境

范围 (scope) 可以将一个图片划分成空间或逻辑上几个区域，通过 scope 可以对指定区域内容进行编辑。创建 scope 环境格式如下:

```
\begin{scope}<animations spec>[<options>]
   <environment contents>
\end{scope}
```

```
\begin{tikzpicture}[ultra thick]
   \begin{scope}[red]
      \draw (0mm,10mm) -- (10mm,10mm);
      \draw (0mm,8mm) -- (10mm,8mm);
   \end{scope}
   \draw (0mm,6mm) -- (10mm,6mm);
   \begin{scope}[green]
      \draw (0mm,4mm) -- (10mm,4mm);
      \draw (0mm,2mm) -- (10mm,2mm);
      \draw[blue] (0mm,0mm) -- (10mm,0mm);
   \end{scope}
\end{tikzpicture}
```

图 2.1.4 scope

scope 适用大部分的全局修饰，常用的几个如下：

- name = <scope name>
  用来指定范围名，方便后文定位。
- every scope
  指定所有范围的全局样式。
- execute at begin/end scope = <code>
  与上文提到的用法一致。

1.3.2 简化 scope

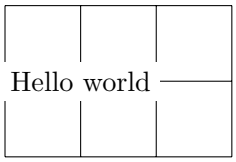
可以使用 scopes 包来简化 scope 环境的写法。这将用 区分 scope。

```
\begin{tikzpicture}[scale = 1]
   { [ultra thick]
      { [red]
         \draw (0mm,10mm) -- (10mm,10mm);
         \draw (0mm,8mm) -- (10mm,8mm);
      }
      \draw (0mm,6mm) -- (10mm,6mm);
   }{ [green]
      \draw (0mm,4mm) -- (10mm,4mm);
      \draw (0mm,2mm) -- (10mm,2mm);
      \draw[blue] (0mm,0mm) -- (10mm,0mm);
   }
\end{tikzpicture}
```

图 2.1.5 scopes 包的使用

某些特定的绘图指令需要在 scope 环境中生效，类似 tikz 指令，TikZ 提供了简化的 scoped 指令。

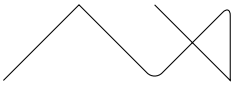
```
\scoped <animation spec> [<options>] <path command>
```




```
\begin{tikzpicture}
    \node [fill=white] at (1,1) {Hello world};
    \scoped [on background layer]
        \draw (0,0) grid (3,2);
\end{tikzpicture}
```

图 2.1.6 scpoed 简化范围语言

在 path 指令中，可以使用 {} 表示一个范围



```
\begin{tikzpicture}[scale = 1]
    \draw (0,0) -- (1,1)
        {[rounded corners] -- (2,0) -- (3,1)}
        -- (3,0) -- (2,1);
\end{tikzpicture}
```

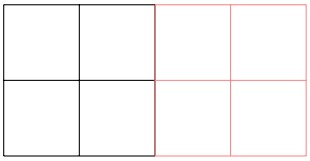
图 2.1.7 path 中的范围

1.4 绘制样式

1.4.1 定义与使用样式

TikZ 提供了 styles 可以方便用户定义一个范围的样式并保存复用。定义样式的格式如下：

```
<StyleName>/.style = {<contents>}
```

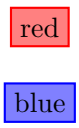


```
\begin{tikzpicture}[help lines/.style={red!50,very thin}]
    \draw            (0,0) grid +(2,2);
    \draw[help lines] (2,0) grid +(2,2);
\end{tikzpicture}
```

图 2.1.8 自定义样式

与 css 类似的，除了在环境中定义样式，还可以提前单独定义。在 \tikzset{} 中定义，可供全文复用。如果需要在原有的样式基础上修改，则可以使用 <StyleName>/.append style = {<contents>} 修改。

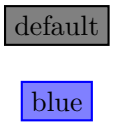
TikZ 支持参数化样式，可以使用 #num 的方式代替值，在使用时给出。




```
\begin{tikzpicture}[outline/.style={draw=#1,thick,fill=#1!50}]
    \node [outline=red] at (0,1) {red};
    \node [outline=blue] at (0,0) {blue};
\end{tikzpicture}
```

图 2.1.9 自定义参数化样式

在这基础上，TikZ 也提供了默认值。




```
\begin{tikzpicture}[outline/.style={draw=#1,thick,fill=#1!50},
   outline/.default=black]
   \node [outline] at (0,1) {default};
   \node [outline=blue] at (0,0) {blue};
\end{tikzpicture}
```

图 2.1.10 默认自定义参数化样式

二、Coordinate

2.1 概述

坐标 (coordinate) 通常以点的形式出现，在 TikZ 中使用 () 表示一个坐标，一般语法为

```
([<options>] <coordinate specification>)
```

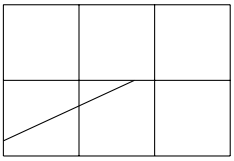
其中，coordinate specification 用于确定坐标的位置，可以使用平面直角坐标系或者极坐标系等方式确定。

主要有以下两种方式确定坐标系：

- 显式 (Explicitly)

  可以在坐标前显式指出坐标系名，随后跟上关键字 cs(coordinate system), 随后给出代表坐标位置的具体键值对。

  语法形式: (<coordinate system> cs: <list of key-value pairs specific to the coordinate system>)

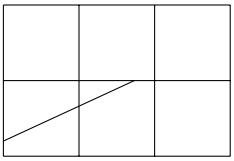


```
\begin{tikzpicture}[scale = 1]
   \draw (0,0) grid (3,2);
   \draw (canvas cs:x=0cm,y=2mm)
      -- (canvas polar cs:radius=2cm,angle=30);
\end{tikzpicture}
```

图 2.2.1 Coordinate:Explicitly

- 隐式 (Implicitly)

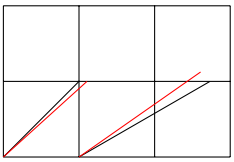
  精确说明坐标系的语法往往太过复杂，一般情况下我们更习惯与使用简化的语法，使用 (x,y) 表示平面直角坐标系点，使用 (radius,angle) 表示极坐标系。



```
\begin{tikzpicture}[scale = 1]
   \draw (0,0) grid (3,2);
   \draw (0cm,2mm) -- (30:2cm);
\end{tikzpicture}
```

图 2.2.2 Coordinate:Implicitly

可以通过 [options] 给出修饰：



```
\begin{tikzpicture}[scale = 1]
   \draw     (0,0) grid (3,2);
   \draw     (0,0) -- (1,1);
   \draw[red] (0,0) -- ([xshift=3pt] 1,1);
   \draw     (1,0) -- +(30:2cm);
   \draw[red] (1,0) -- +([shift=(135:5pt)] 30:2cm);
\end{tikzpicture}
```

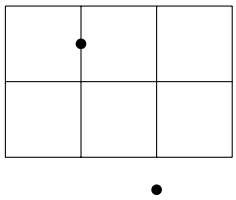
图 2.2.3 Coordinate: 修饰

2.2 坐标系

2.2.1 Canvas,XYZ,Ploar 坐标系

- **canvas 坐标系**

canvas 是最简单的坐标系，基本上与平面直角坐标系相同，其两个参数 x,y 分别表示偏移量 d_x, d_y，对应值均为 <dimension>。



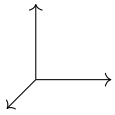
```
\begin{tikzpicture}
   \draw (0,0) grid (3,2);
   \fill (canvas cs:x=1cm,y=1.5cm) circle (2pt);
   \fill (canvas cs:x=2cm,y=-5mm+2pt) circle (2pt);
\end{tikzpicture}
```

图 2.2.4 Coordinate:canvas

上述显示给出了键值对，也可以隐式地只给出值。

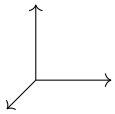
- **xyz**

  xyz 坐标系类似于空间直角坐标系，在 canvas 坐标系的基础上增加了 z 轴。



```
\begin{tikzpicture}[->]
   \draw (0,0) -- (xyz cs:x=1);
   \draw (0,0) -- (xyz cs:y=1);
   \draw (0,0) -- (xyz cs:z=1);
\end{tikzpicture}
```

图 2.2.5 Coordinate:xyz cs **显示写法**



```
\begin{tikzpicture}[->]
   \draw (0,0) -- (1,0);
   \draw (0,0) -- (0,1,0);
   \draw (0,0) -- (0,0,1);
\end{tikzpicture}
```

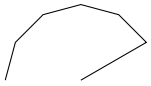
图 2.2.6 Coordinate:xyz cs **隐式写法**

值得说明的是 canvas 与 xyz 坐标系并没有非常明确的区分，有时候因为写法的不同，TikZ 常在这两种坐标系之间进行切换，这里仅提一下，没有必要深入了解，具体原理请参考官方手册。

此外，如果我们使用 (1,0) 表示 x 方向偏移 1cm, 但如果我们使用 (2+3cm,0) 的形式，默认单位则变成了 pt，其真实偏移量为 (2pt+3cm,0) 这适用于所有未显示指明单位的复合形式。

- **canvas polar**

  canvas polar 也即极坐标系，常用的参数有两个，angle = <degrees> 和 radius = <dimension>。其中 angle 值的范围为 [-360,720]。



```
\begin{tikzpicture}[scale = 1]
   \draw (0cm,0cm) -- (30:1cm) -- (60:1cm) -- (90:1cm)
         -- (120:1cm) -- (150:1cm) -- (180:1cm);
\end{tikzpicture}
```

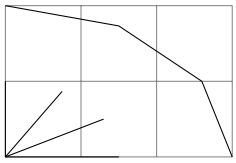
图 2.2.7 Coordinate:canvas polar

angle 除了给出具体的单位值，也可以使用方位 (如:north east) 来表示。

canvas polar 还有两个不常用的值 x/y radius, 当我们给出 radius 时，可以理解为做圆得到距离，而 x/y radius 则表示做椭圆。

- **xyz polar**

  与一般极坐标系不同的是，xyz polar 坐标系将值最终转译到 xy 坐标系上，即向极坐标转换为直角坐标，这样做的目的好处包括可以定义不同 x,y 的单位长度，或者对坐标系进行变换。其值与 canvas polar 坐标系一致。



```
\begin{tikzpicture}[x=1.5cm,y=1cm]
   \draw[help lines] (0cm,0cm) grid (3cm,2cm);

   \draw (0,0) -- (xyz polar cs:angle=0,radius=1);
   \draw (0,0) -- (xyz polar cs:angle=30,radius=1);
   \draw (0,0) -- (xyz polar cs:angle=60,radius=1);
   \draw (0,0) -- (xyz polar cs:angle=90,radius=1);

   \draw (xyz polar cs:angle=0,radius=2)
      -- (xyz polar cs:angle=30,radius=2)
      -- (xyz polar cs:angle=60,radius=2)
      -- (xyz polar cs:angle=90,radius=2);
\end{tikzpicture}
```

图 2.2.8 Coordinate:xyz polar-**单位长度**

```
\begin{tikzpicture}[scale = 1]
   \tikz[x={(0cm,1cm)},y={(-1cm,0cm)}]
   \draw (0,0) -- (30:1) -- (60:1) -- (90:1)
   -- (120:1) -- (150:1) -- (180:1);
\end{tikzpicture}
```

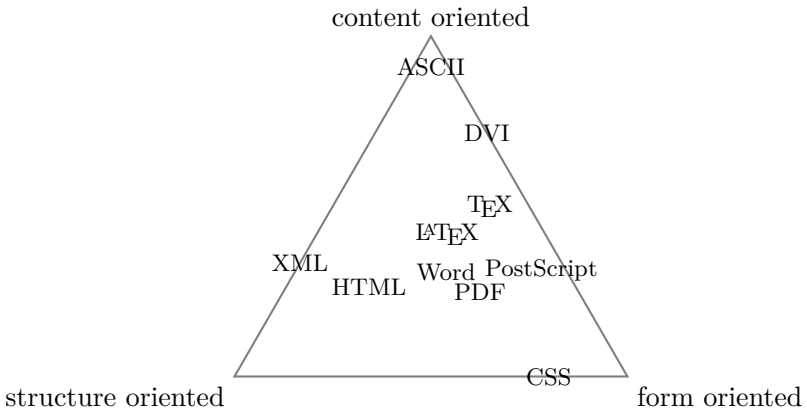
图 2.2.9 Coordinate:xyz polar-y **轴变换**

2.2.2 重心坐标系

重心坐标系 (barycentric cs) 通过向量 $v_1, v_2, \cdots, v_n$ 和数值 $\alpha_1, \alpha_2, \cdots, \alpha_n$ 来确定某个点位置，其计算函数如下：

$$\frac{\alpha_1 v_1 + \alpha_2 v_2 + \cdots + \alpha_n v_n}{\alpha_1 + \alpha_2 + \cdots + \alpha_n}$$

重心坐标系需要先指定重心位置，再通过键值对赋权值。




```
\begin{tikzpicture}
    \coordinate (content) at (90:3cm);
    \coordinate (structure) at (210:3cm);
    \coordinate (form) at (-30:3cm);

    \node [above] at (content) {content oriented};
    \node [below left] at (structure) {structure oriented};
    \node [below right] at (form) {form oriented};

    \draw [thick,gray] (content.south) -- (structure.north east) -- (form.north west) -- cycle
         ;

    \small
    \node at (barycentric cs:content=0.5,structure=0.1 ,form=1) {PostScript};
    \node at (barycentric cs:content=1 ,structure=0 ,form=0.4) {DVI};
    \node at (barycentric cs:content=0.5,structure=0.5 ,form=1) {PDF};
    \node at (barycentric cs:content=0 ,structure=0.25,form=1) {CSS};
    \node at (barycentric cs:content=0.5,structure=1 ,form=0) {XML};
    \node at (barycentric cs:content=0.5,structure=1 ,form=0.4) {HTML};
    \node at (barycentric cs:content=1 ,structure=0.2 ,form=0.8) {\TeX};
    \node at (barycentric cs:content=1 ,structure=0.6 ,form=0.8) {\LaTeX};
    \node at (barycentric cs:content=0.8,structure=0.8 ,form=1) {Word};
    \node at (barycentric cs:content=1 ,structure=0.05,form=0.05) {ASCII};
\end{tikzpicture}
```

图 2.2.10 Coordinate:barycentric cs

2.2.3 节点坐标系

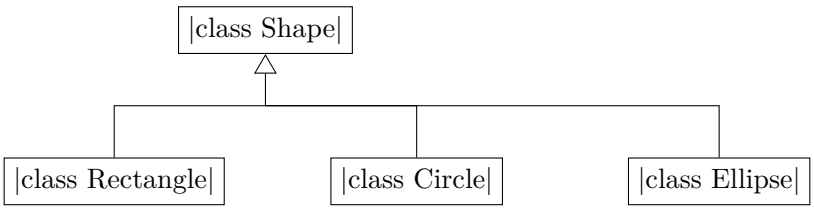
节点坐标系 (node cs) 主要用于获取不同节点名进行连接等操作。其主要修饰如下：

- name = <node name>

  指定节点名

- anchor = <anchor>

  节点的锚点，可理解为连线的初始/结束点。可以省略不写，TikZ 将会自动寻找合适的绘制路径。



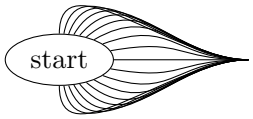

```
\begin{tikzpicture}
    \node (shape) at (0,2) [draw] {|class Shape|};
    \node (rect) at (-2,0) [draw] {|class Rectangle|};
    \node (circle) at (2,0) [draw] {|class Circle|};
    \node (ellipse) at (6,0) [draw] {|class Ellipse|};

    \draw (node cs:name=circle,anchor=north) |- (0,1);
    \draw (node cs:name=ellipse,anchor=north) |- (0,1);
    \draw [arrows = -{Triangle[open, angle=60:3mm]}]
          (node cs:name=rect,anchor=north)
          |- (0,1) -| (node cs:name=shape,anchor=south);
\end{tikzpicture}
```

图 2.2.11 Coordinate:node cs - anchor

- angle = <degree>

  除了通过锚点绘制线，也可以通过角度绘制。




```
\begin{tikzpicture}
    \node (start) [draw,shape=ellipse] {start};
    \foreach \angle in {-90, -80, ..., 90}
        \draw (node cs:name=start,angle=\angle)
        .. controls +(\angle:1cm) and +(-1,0) .. (2.5,0);
\end{tikzpicture}
```

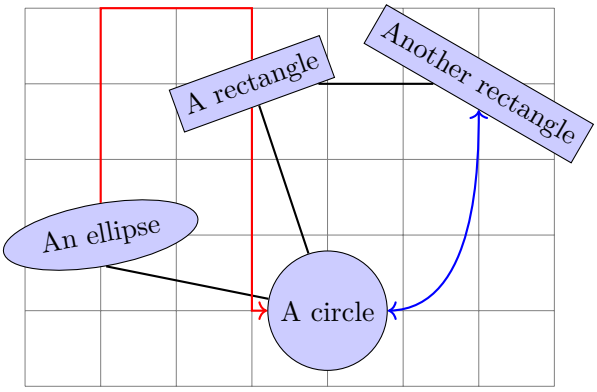
图 2.2.12 Coordinate:node cs - angle

在曲线 (包括直线) 绘制中，TikZ 使用 – 表示直线，|- 表示竖直和平行线，... 表示曲线，绘制时需要用到.. controls <contents> .. 语法。+(dx,dy) 表示偏移量。


```
\begin{tikzpicture}[scale = 1]
    \draw (0,0) node (x) [draw] {X}
          (2,0) node (y) {Y}
          (node cs:name=x) .. controls +(1,1) and +(-1,1) .. (
              node cs:name = y);
\end{tikzpicture}
```

图 2.2.13 Coordinate: **曲线绘制示例**

节点坐标系作为一种经常被使用到的坐标定位方式，在实际使用中经常使用简写方式，可以省略 name 等键。



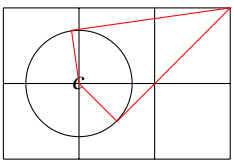

```
\begin{tikzpicture}[fill=blue!20]
    \draw[help lines] (-1,-2) grid (6,3);
    \path (0,0) node(a) [ellipse,rotate=10,draw,fill] {An ellipse}
        (3,-1) node(b) [circle,draw,fill] {A circle}
        (2,2) node(c) [rectangle,rotate=20,draw,fill] {A rectangle}
        (5,2) node(d) [rectangle,rotate=-30,draw,fill] {Another rectangle};
    \draw[thick] (a.south) -- (b) -- (c) -- (d);
    \draw[thick,red,->] (a) |- +(1,3) -| (c) |- (b);
    \draw[thick,blue,<->] (b) .. controls +(right:2cm) and +(down:1cm) .. (d);
\end{tikzpicture}
```

图 2.2.14 Coordinate:node cs 隐式写法

2.2.4 切线坐标系

切线坐标系 (tangent cs) 需要加载 calc 包。显而易见，它是用来画切线的。其主要键如下：

- node = <node>
  绘制切线的对象
- point = <point>
  发射切线的点
- solution = <number>
  如果有多种切线绘制方案，指定某一种



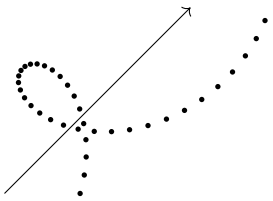
```
\begin{tikzpicture}
    \draw (0,0) grid (3,2);
    \coordinate (a) at (3,2);
    \node [circle,draw] (c) at (1,1) [minimum size=40pt] {$c$};
    \draw[red] (a) -- (tangent cs:node=c,point={(a)},solution
        =1) --
      (c.center) -- (tangent cs:node=c,point={(a)},solution=2)
          -- cycle;
\end{tikzpicture}
```

图 2.2.15 Coordinate:tangent cs

2.2.5 定义新的坐标系

TikZ 支持定义新的坐标系，语法需要了解 TeX 底层代码，这里仅给出示例。定义新坐标系的语法格式如下：

```
\tikzdeclarecoordinatesystem{<name>}{<code>}
```



```
\makeatletter
\define@key{cylindricalkeys}{angle}{\def\myangle{#1}}
\define@key{cylindricalkeys}{radius}{\def\myradius{#1}}
\define@key{cylindricalkeys}{z}{\def\myz{#1}}
\tikzdeclarecoordinatesystem{cylindrical}{
   \setkeys{cylindricalkeys}{#1}
   \pgfpointadd{\pgfpointxyz{0}{0}{\myz}}{\pgfpointpolarxy{\
        myangle}{\myradius}}
}
\begin{tikzpicture}[z=0.2pt]
   \draw [->] (0,0,0) -- (0,0,350);
   \foreach \num in {0,10,...,350}
      \fill (cylindrical cs:angle=\num,radius=1,z=\num) circle
           (1pt);
\end{tikzpicture}
```

图 2.2.16 Coordinate: 子弟你故意坐标系

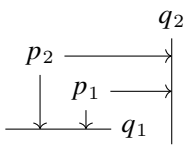
2.3 交叉点坐标

2.3.1 垂线交点

以下两个参数常用来绘制垂直于坐标轴的线段

- horizontal line through = (<coordinate>)
  经过某点且平行于 x 轴的直线
- vertical line through = (<coordinate>)
  经过某点且平行于 y 轴的直线

以上两种方式也可以用 -| 或 |- 的形式隐式代替。



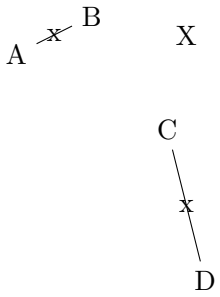

```
\begin{tikzpicture}
   \path (30:1cm) node(p1) {$p_1$} (75:1cm) node(p2) {$p_2$};

   \draw (-0.2,0) -- (1.2,0) node(xline)[right] {$q_1$};
   \draw (2,-0.2) -- (2,1.2) node(yline)[above] {$q_2$};

   \draw[->] (p1) -- (p1 |- xline);
   \draw[->] (p2) -- (p2 |- xline);
   \draw[->] (p1) -- (p1 -| yline);
   \draw[->] (p2) -- (p2 -| yline);
\end{tikzpicture}
```

图 2.2.17 Coordinate: 垂线交点

注意上述的点并没有使用 (), 如果比较复杂，则需要加上。




```
\begin{tikzpicture}
   \node (A) at (0,1) {A};
   \node (B) at (1,1.5) {B};
   \node (C) at (2,0) {C};
   \node (D) at (2.5,-2) {D};
   \draw (A) -- (B) node [midway] {x};
   \draw (C) -- (D) node [midway] {x};
   \node at ({$(A)!.5!(B)$} -| {$(C)!.5!(D)$}) {X};
\end{tikzpicture}
```

图 2.2.18 Coordinate: **垂线定位**

2.3.2 任意焦点的坐标

在获取任意交点坐标之前，需要导入交点包: intersections。这将帮助我们得出交点，但是由于 TeX 底层的精度 (相对专业软件) 并不高，交点仅适用于绘图作为参考。

需要获得交点的线段必须被命名，以便于后续引用。相关的键如下：

- name path = <name>

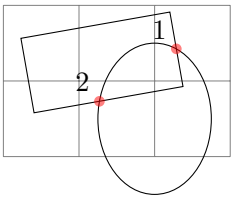
  路径名，在分号之前都有效。

- name path global = <name>

  全局路径名，在整个绘图环境中都有效。

- name intersections = <options>

  决定线段对象，只有值对应的曲线交点才会被计算。交点则会以 intersection-1 形式命名下去。




```
\begin{tikzpicture}[every node/.style={opacity=1, black, above
    left}]
   \draw [help lines] grid (3,2);
   \draw [name path=ellipse] (2,0.5) ellipse (0.75cm and 1cm);
   \draw [name path=rectangle, rotate=10] (0.5,0.5) rectangle
       +(2,1);
   \fill [red, opacity=0.5, name intersections={of=ellipse and
        rectangle}]
      (intersection-1) circle (2pt) node {1}
      (intersection-2) circle (2pt) node {2};
\end{tikzpicture}
```

图 2.2.19 Coordinate: **曲线交点**

- intersection/of = <name path 1> and <name path 2>

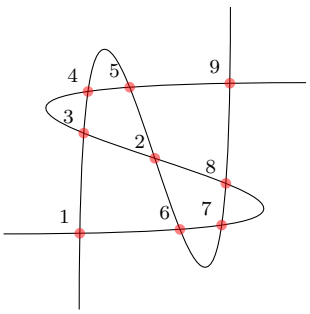
  用于指定计算交点的路径。

- intersection/name = <prefix>

  修改前缀 intersection 为对应的 prefix

- total = <macro>

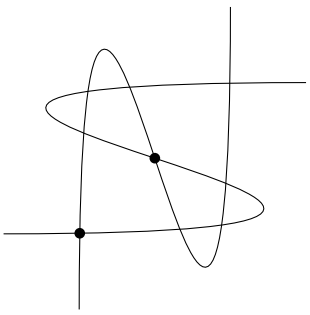
  交点的数量将被记录在 TeX 的 <macro> 中。




```
\begin{tikzpicture}
   \clip (-2,-2) rectangle (2,2);
   \draw [name path=curve 1] (-2,-1) .. controls (8,-1) and
        (-8,1) .. (2,1);
   \draw [name path=curve 2] (-1,-2) .. controls (-1,8) and
        (1,-8) .. (1,2);
   \fill [name intersections={of=curve 1 and curve 2, name=i,
        total=\t}]
      [red, opacity=0.5, every node/.style={above left, black,
           opacity=1}]
      \foreach \s in {1,...,\t}{(i-\s) circle (2pt) node {\
          footnotesize\s}};
\end{tikzpicture}
```

图 2.2.20 Coordinate: **曲线交点**

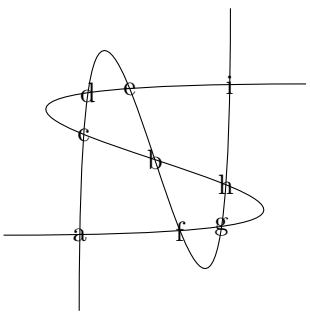
- by = <comma-separated list>
  这个键允许我们给一些特定的坐标命名，(这不影响 <prefix>-<number> 中的名称)。



```
\begin{tikzpicture}
   \clip (-2,-2) rectangle (2,2);
   \draw [name path=curve 1] (-2,-1) .. controls (8,-1) and
        (-8,1) .. (2,1);
   \draw [name path=curve 2] (-1,-2) .. controls (-1,8) and
        (1,-8) .. (1,2);
   \fill [name intersections={of=curve 1 and curve 2, by={a,b
        }}]
      (a) circle (2pt)
      (b) circle (2pt);
\end{tikzpicture}
```

图 2.2.21 Coordinate: **曲线交点命名**

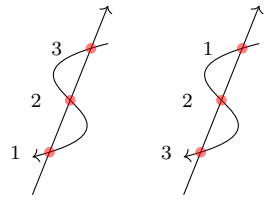
在命名过程中，可以使用类似 for each 中的省略。




```
\begin{tikzpicture}
   \clip (-2,-2) rectangle (2,2);
   \draw [name path=curve 1] (-2,-1) .. controls (8,-1) and
        (-8,1) .. (2,1);
   \draw [name path=curve 2] (-1,-2) .. controls (-1,8) and
        (1,-8) .. (1,2);
   \fill [name intersections={
      of=curve 1 and curve 2,
      by={[label=center:a],[label=center:...],[label=center:i
          ]}}];
\end{tikzpicture}
```

图 2.2.22 Coordinate: **曲线交点遍历命名**

- sort by=<path name>
  默认情形下，交点的顺序为算法计算的顺序，该键可以帮助我们按指定焦点名重新排序。



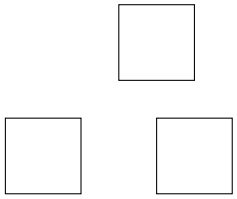

```
\begin{tikzpicture}
   \clip (-0.5,-0.75) rectangle (3.25,2.25);
   \foreach \pathname/\shift in {line/0cm, curve/2cm}{
      \tikzset{xshift=\shift}
      \draw [->, name path=curve] (1,1.5) .. controls (-1,1) and (2,0.5) ..
          (0,0);
      \draw [->, name path=line] (0,-.5) -- (1,2) ;
      \fill [name intersections={of=line and curve,sort by=\pathname, name=i}]
         [red, opacity=0.5, every node/.style={left=.25cm, black, opacity=1}]
         \foreach \s in {1,2,3}{(i-\s) circle (2pt) node {\footnotesize\s}};
      }
\end{tikzpicture}
```

图 2.2.23 Coordinate: 交点顺序

2.4 相对坐标

2.4.1 指定相对坐标

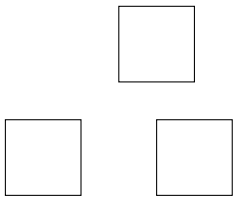
可以使用 ++ 来指定相对坐标 (位移), ++(dx,dy) 代表在当前坐标的基础上往 x 和 y 正方向偏移 x，y 距离，这个偏移后的坐标将替换原来的坐标 (类似于 C++ 传引用)。



```
\begin{tikzpicture}
   \draw (0,0) -- ++(1,0) -- ++(0,1) -- ++(-1,0) -- cycle;
   \draw (2,0) -- ++(1,0) -- ++(0,1) -- ++(-1,0) -- cycle;
   \draw (1.5,1.5) -- ++(1,0) -- ++(0,1) -- ++(-1,0) -- cycle;
\end{tikzpicture}
```

图 2.2.24 Coordinate:++

与 ++ 相似的还有 +，不过他不会替换原有的坐标 (C++ 传值)。



```
\begin{tikzpicture}
   \draw (0,0) -- +(1,0) -- +(1,1) -- +(0,1) -- cycle;
   \draw (2,0) -- +(1,0) -- +(1,1) -- +(0,1) -- cycle;
   \draw (1.5,1.5) -- +(1,0) -- +(1,1) -- +(0,1) -- cycle;
\end{tikzpicture}
```

图 2.2.25 Coordinate:+

2.4.2 旋转相对位移

有时候我们需要获得曲线的切线或者前进方向，这时可以用 turn 键。

```
\tikz \draw (0,0) -- (1,1) -- ([turn]-45:1cm) -- ([turn
    ]-30:1cm);
```

图 2.2.26 Coordinate:turn

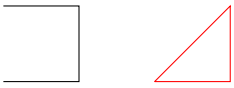
2.4.3 相对坐标与范围

当 scope 与相对偏移结合时，会发生什么？由于 scope 仅影响内部指令，坐标发生了偏移，但是路径没有发生变换；这可以帮助我们使用 scope 暂时改变坐标方位，但是又不影响原有的路径。(效果类似于 +)

一般情况下，TikZ 默认 scope 不会产生影响，即有没有 scope 没有区别。如果想要达到上述效果，需要使用 current point is local=<boolean> 键。

```
\begin{tikzpicture}
   \draw (0,0) -- ++(1,0) -- ++(0,1) -- ++(-1,0);
   \draw[red] (2,0) -- ++(1,0) { -- ++(0,1) } -- ++(-1,0);
\end{tikzpicture}
```

图 2.2.27 Coordinate:scope 与 ++



```
\begin{tikzpicture}
   \draw     (0,0) -- ++(1,0) -- ++(0,1) -- ++(-1,0);
   \draw[red] (2,0) -- ++(1,0)
             { [current point is local] -- ++(0,1) } --
                  ++(-1,0);
\end{tikzpicture}
```

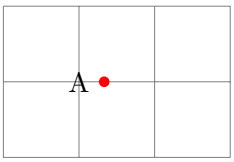
图 2.2.28 Coordinate:current point is local

2.5 坐标计算

此章需要使用到 calc 包。

calc 包允许我们对坐标进行一些操作 (计算，增减，测量...)

下面这个例子计算出 A 点的位置并进行了位置运算绘制出红点。




```
\begin{tikzpicture}
   \draw [help lines] (0,0) grid (3,2);
   \node (a) at (1,1) {A};
   \fill [red] ($(a) + 1/3*(1cm,0)$) circle (2pt);
\end{tikzpicture}
```

图 2.2.29 Coordinate:calc

2.5.1 基础语法

calc 包的通用语法格式如下：

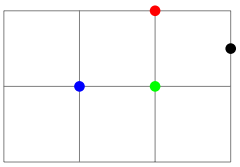
```
([<options>] $<coordinate computation>$)
```

上述语法借用了 TeX 的 $$ 表示数学计算。<coordinate computation> 需遵循如下结构：

```
<factor> * <coordinate> <modifiers>
```

2.5.2 语法：运算

TikZ 允许我们在 $$ 中嵌入数学运算公式 (对应 factors)，在写复杂公式时请使用 {} 区分长表达式。底层运算原理这里不做介绍，下面是几个示例：



```
\begin{tikzpicture}
   \draw [help lines] (0,0) grid (3,2);
   \fill [red] ($2*(1,1)$) circle (2pt);
   \fill [green] (${1+1}*(1,.5)$) circle (2pt);
   \fill [blue] ($cos(0)*sin(90)*(1,1)$) circle (2pt);
   \fill [black] (${3*(4-3)}*(1,0.5)$) circle (2pt);
\end{tikzpicture}
```

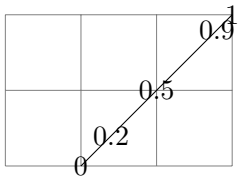
图 2.2.30　Coordinate:factors

2.5.3 语法：路径修饰

路径修饰[3](partway modifier) 语法格式如下：

```
<coordinate>!<number>!<angle>:<second coordinate>
```

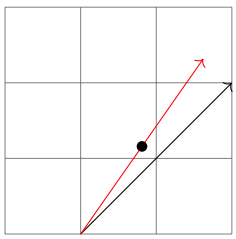
以 (1,2)!.75!(3,4) 为例，它表达的是计算 (1,2) 到 (3,4) 间线段距离 3/4 的坐标。




```
\begin{tikzpicture}
   \draw [help lines] (0,0) grid (3,2);
   \draw (1,0) -- (3,2);
   \foreach \i in {0,0.2,0.5,0.9,1}
      \node at ($(1,0)!\i!(3,2)$) {\i};
\end{tikzpicture}
```

图 2.2.31　Coordinate:partway modifier-number

接下来引入角度 (angle)，以 (1,2)!.5!60:(2,2) 为例，它表示在找到中点坐标后，以第一个点为原点旋转 60 度。

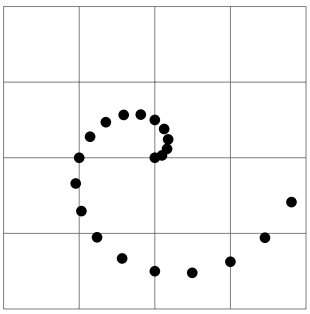


```
\begin{tikzpicture}
   \draw [help lines] (0,0) grid (3,3);
   \coordinate (a) at (1,0);
   \coordinate (b) at (3,2);
   \draw[->] (a) -- (b);
   \coordinate (c) at ($ (a)!1! 10:(b) $);
   \draw[->,red] (a) -- (c);
   \fill ($ (a)!.5! 10:(b) $) circle (2pt);
\end{tikzpicture}
```

图 2.2.32　Coordinate:partway modifier-angle

来点怪的：

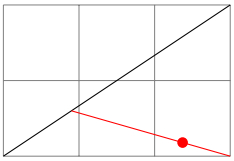
[3]这里是我根据其作用起的名字



```
\begin{tikzpicture}
   \draw [help lines] (0,0) grid (4,4);
   \foreach \i in {0,0.1,...,2}
      \fill ($(2,2) !\i! \i*180:(3,2)$) circle (2pt);
\end{tikzpicture}
```

图 2.2.33 Coordinate:partway modifier-angle

修饰可以嵌套，这样可以很好地辅助我们作图：



```
\begin{tikzpicture}
   \draw [help lines] (0,0) grid (3,2);
   \draw (0,0) -- (3,2);
   \draw[red] ($(0,0)!.3!(3,2)$) -- (3,0);
   \fill[red] ($(0,0)!.3!(3,2)!.7!(3,0)$) circle (2pt);
\end{tikzpicture}
```

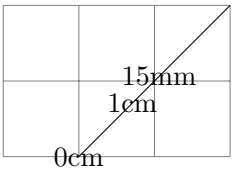
图 2.2.34 Coordinate:partway modifier-嵌套

2.5.4 语法：距离修饰

距离修饰 (distance modifier) 语法格式如下：

```
<coordinate>!<dimension>!<angle>:<second coordinate>
```

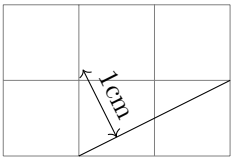
可以看出，这与路径修饰语法格式几乎一致，只是将路径修饰的按比例 (number) 改成了按距离 (dimension)。




```
\begin{tikzpicture}
   \draw [help lines] (0,0) grid (3,2);
   \draw (1,0) -- (3,2);
   \foreach \i in {0cm,1cm,15mm}
      \node at ($(1,0)!\i!(3,2)$) {\i};
\end{tikzpicture}
```

图 2.2.35 Coordinate:distance modifier

配合旋转修饰：




```
\begin{tikzpicture}
   \draw [help lines] (0,0) grid (3,2);
   \coordinate (a) at (1,0);
   \coordinate (b) at (3,1);
   \draw (a) -- (b);
   \coordinate (c) at ($ (a)!.25!(b) $);
   \coordinate (d) at ($ (c)!1cm!90:(b) $);
   \draw [<->] (c) -- (d) node [sloped,midway,above] {1cm};
\end{tikzpicture}
```

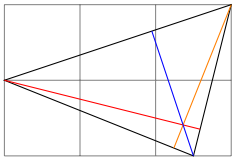
图 2.2.36 Coordinate:distance modifier

2.5.5 语法：投影修饰

投影修饰 (projection modifier) 语法格式如下：

```
<coordinate>!<projection coordinate>!<angle>:<second coordinate>
```

顾名思义，投影修饰是用来绘制投影 (垂线) 的。以 (1,2)!(0,5)!(3,4) 为例，从 (0,5) 点出发，做 (1,2) 与 (3，4) 构成的直线的垂线。



```
\begin{tikzpicture}
   \draw [help lines] (0,0) grid (3,2);
   \coordinate (a) at (0,1);
   \coordinate (b) at (3,2);
   \coordinate (c) at (2.5,0);
   \draw (a) -- (b) -- (c) -- cycle;
   \draw[red] (a) -- ($(b)!(a)!(c)$);
   \draw[orange] (b) -- ($(a)!(b)!(c)$);
   \draw[blue] (c) -- ($(a)!(c)!(b)$);
\end{tikzpicture}
```

图 2.2.37 Coordinate:projection modifier

三、Path

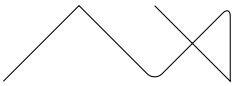
3.1 概述

路径 (path) 是指一系列直线/曲线的片段集合。它对应的指令为 \path。

```
\path <specification>
```

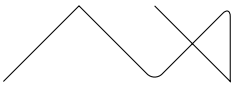
其中 <specification> 值一长串路径操作流，例如 (x,y) 坐标。在路径流中可以使用 [] 对路径进行修饰，用 {} 指定修饰范围。

\path 指令虽然绘制了路径，但它并不会显示在图像上，需要使用 [draw] 修饰，通常使用 \draw 或者 \fill 指令替代 \path 指令。



```
\draw (0,0) -- (1,1)
   [rounded corners] -- (2,0) -- (3,1)
   [sharp corners] -- (3,0) -- (2,1);
```

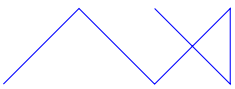
图 2.3.1 Path: 修饰



```
\draw (0,0) -- (1,1)
   {[rounded corners] -- (2,0) -- (3,1)}
   -- (3,0) -- (2,1);
```

图 2.3.2 Path: 范围

有些修饰无论在何处给出，都会影响整个路径。一种解决方法是写多个路径。



```
\draw (0,0) -- (1,1)
   [color=red] -- (2,0) -- (3,1)
   [color=blue] -- (3,0) -- (2,1);
```

图 2.3.3 Path: 全局修饰

值得注意的是，Node 并不是 Path 的一部分，虽然 node 可以写在路径中，但逻辑上应该先绘制路径，再在路径周围添加节点。

下面解释一些常用的修饰。

- name = <path name>

  为路径命名，方便复用与定位。

- every path

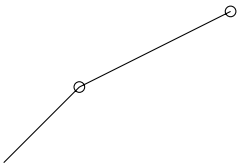
  用于指定所有路径样式

```
\begin{tikzpicture}[every path/.style = draw]
   \path (0,0) -- (1,0) -- (1,1) -- (0,1) -- cycle;
   \path (2,0) -- (2,1);
\end{tikzpicture}
```

图 2.3.4 Path:every path

- insert path = <path>

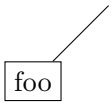
该修饰用于在路径上添加图形。它主要用于放置自定义图形。值得说明的是，最好不要将它用作节点。



```
\begin{tikzpicture}[c./style = {insert path = {circle[radius =
    2pt]}}]
   \draw (0,0) -- (1,1)[c] -- (3,2)[c];
\end{tikzpicture}
```

图 2.3.5 Path:insert path

- append after command = <path>
  在指令完成后执行。




```
\draw node [append after command={(foo)--(1,1)},draw] (foo){foo
    };
```

图 2.3.6 Path:append after command

- prefix after command = <path>
  与上一个修饰类似，不过它会在主路径绘制之后再绘制。

3.2 路径绘制

3.2.1 直线绘制

基本语法

直线绘制通过坐标点 (coordinate) 进行连线。其基本语法如下

```
\path ...<coordinate>...;
```

在一个命令中，直到; 结尾之前，TikZ 允许进行多个同类路径绘制。


```
\begin{tikzpicture}[scale = 1]
   \draw (0,0) -- (2,0);
   \draw (0,1) -- (2,1);
\end{tikzpicture}
```

图 2.3.7 Path: 直线绘制

路径闭合

在绘制的路径过程中，往往要考虑起始点与终点的连接问题，其中一种方式是通过 (current subpath start) 可获得起始点的坐标，其效果等同于将起始点坐标写在最后。

```
\begin{tikzpicture}[line width = 5pt]
   \draw (0,0) -- (1,0) -- (1,1) -- (0,1) -- (current subpath
       start);
\end{tikzpicture}
```

图 2.3.8 Path:current subpath start

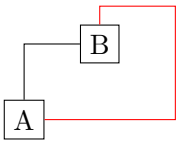
在这个过程中发现衔接处并没有达到理想的处理效果，更常用的 cycle 可以解决这种闭合问题。

```
\begin{tikzpicture}[line width = 5pt]
   \draw (0,0) -- (1,0) -- (1,1) -- (0,1) -- cycle;
\end{tikzpicture}
```

图 2.3.9 Path:cycle

垂线与水平线

如果需要绘制垂线或水平线，可以使用 | - 两个关键字，分别代表垂线与水平线。在两点间使用 |- 代替 – 意味着，先从第一个点出发，沿垂直方向到达第二个点的 y 坐标，再连接到第二个点，-| 同理。




```
\begin{tikzpicture}[scale = 1]
   \draw (0,0) node(a) [draw] {A} (1,1) node(b) [draw] {B};
   \draw (a.north) |- (b.west);
   \draw[color=red] (a.east) -| (2,1.5) -| (b.north);
\end{tikzpicture}
```

图 2.3.10 Path: 垂线与水平线

3.2.2 曲线与圆角

绘制曲线路径非常简单，在路径中加入 controls 语法即可。

```
\draw[line width=10pt] (0,0) .. controls (1,1) .. (4,0)
   .. controls (5,0) and (5,1) .. (4,1);
\draw[color=gray] (0,0) -- (1,1) -- (4,0) -- (5,0) -- (5,1) --
    (4,1);
```

图 2.3.11 Path: 曲线

如果需要加入圆角，则可以使用上文已经提及的 rounded corners 修饰, 同时也可以为其指定圆角值。

```
\draw [rounded corners = 5pt] (0,0) -- (1,1) -- (2,0) ..
    controls (3,1) .. (4,0);
```

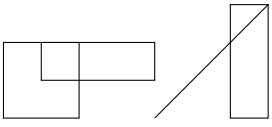
图 2.3.12 Path:rounded corner

3.3 常用图形绘制

3.3.1 矩形与网格

绘制矩形的语法如下, rectangle 关键字左边的坐标为矩形一顶点，右边的坐标为另一顶点。

```
\path ...rectangle<corner or cycle>...;
```



```
\draw (0,0) rectangle (1,1);
\draw (.5,1) rectangle (2,0.5) (3,0) rectangle (3.5,1.5) --
    (2,0);
```

图 2.3.13 Path:rectangle

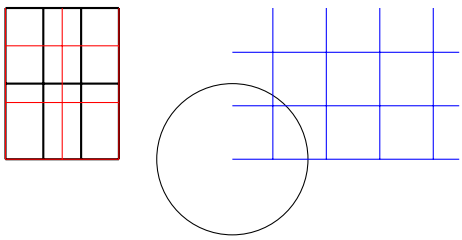
网格的绘制语法与矩形类似：

```
\path ... grid[<options>]<corner or cycle> ...;
```

网格的常用修饰如下：

- step = <number or dimension or coordinate> (默认:1cm)

  step 是网格最关键的修饰，决定了网格之间的间距。



```
\begin{tikzpicture}[scale = 1]
    \begin{scope}[x=.5cm]
        \draw[thick] (0,0) grid [step=1] (3,2);
        \draw[red] (0,0) grid [step=.75cm] (3,2);
    \end{scope}
    \begin{scope}
        \draw (3,0) circle [radius=1];
        \draw[blue] (3,0) grid [step=(45:1)] (6,2);
    \end{scope}
\end{tikzpicture}
```

图 2.3.14 Path:grid-step

- xstep/ystep = <dimension or number> (默认:1cm)

  用于控制 x 和 y 轴方向上的网格距离。

- help lines

  TikZ 提供的一种网格类型，有修饰：[line width=0.2pt,gray]

3.3.2 圆与椭圆

绘制圆的语法如下，circle 关键字左边的坐标即为圆心。

```
\path ... circle[<options>] ...;
```

圆的常用修饰如下：

- x radius = <value>

  指定水平方向半径，一般在定义椭圆时给出。

- y radius = <value>

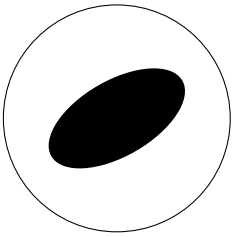
  指定竖直方向半径，一般在定义椭圆时给出。

- radius = <value>

  指定圆半径。

- at = <coordinate>

  指定圆心位置。



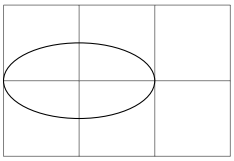
```
\begin{tikzpicture}[scale = 1]
    \draw (1,0) circle [radius=1.5];
    \fill (1,0) circle [x radius=1cm, y radius=5mm, rotate=30];
\end{tikzpicture}
```

图 2.3.15 Path:circle

椭圆语法如下：

```
\path ⋯ ellipse[hoptionsi] ⋯;
```

其参数和圆几乎一致，圆可以通过 xradius/yradius 绘制成椭圆，那为什么要单独再定义一个关键字椭圆呢？原因是在类似 every circle/.style 等全局定义时可以进行区分。



```
\begin{tikzpicture}[scale = 1]
   \draw [help lines] (0,0) grid (3,2);
   \draw (1,1) ellipse [x radius=1cm,y radius=.5cm];
\end{tikzpicture}
```

图 2.3.16　Path:ellipse

3.3.3 弧线绘制

绘制弧线的语法如下：

```
\path ... arc[<options>] ...;
```

弧线的常用修饰如下：

- start angle = <degrees>

  设置弧线开始角度

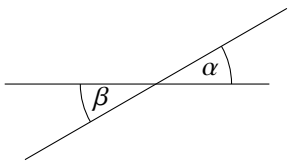
- end angle = <degrees>

  设置弧线结束角度

- delta angle = <degrees>

  设置弧线变化角度

弧线的绘制默认以正 x 轴方向开始，按逆时针方向旋转。arc 关键字前的坐标，用于指定弧线的初始点位置，注意不是圆弧对应的中心。




```
\begin{tikzpicture}[radius=1cm,delta angle=30]
   \draw (-1,0) -- +(3.5,0);
   \draw (1,0) ++(210:2cm) -- +(30:4cm);
   \draw (1,0) +(0:1cm) arc [start angle=0];
   \draw (1,0) +(180:1cm) arc [start angle=180];
   \path (1,0) ++(15:.75cm) node{$\alpha$};
   \path (1,0) ++(15:-.75cm) node{$\beta$};
\end{tikzpicture}
```

图 2.3.17　Path:arc

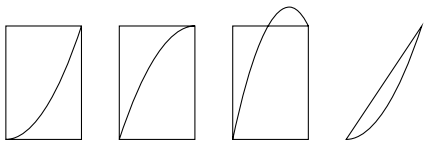
3.4 数学函数绘制

3.4.1 抛物线

绘制抛物线语法如下：

```
\path ...parabola[<options>]bend<bend coordinate><coordinate or cycle>...;
```

如果不使用 bend 关键字说明焦点坐标，则默认第一个点坐标为焦点坐标。



```
\begin{tikzpicture}[scale = 1]
   \draw (0,0) rectangle (1,1.5)
        (0,0) parabola (1,1.5);
   \draw[xshift=1.5cm] (0,0) rectangle (1,1.5)
        (0,0) parabola[bend at end] (1,1.5);
   \draw[xshift=3cm] (0,0) rectangle (1,1.5)
        (0,0) parabola bend (.75,1.75) (1,1.5);
   \draw[xshift=4.5cm] (1,1.5) --
        (0,0) parabola cycle;
\end{tikzpicture}
```

图 2.3.18 parabola

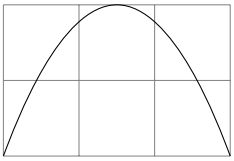
抛物线主要修饰如下：

- bend = <coordinate>

  指明焦点坐标，在 [] 中语法为 [bend at <coordinate>]，也可以写在 [] 外 bend <coordinate>。值得说明的是，如果指明的焦点与其他抛的点并不能构成抛物线，那么绘制的结果将不会是抛物线。

- bend pos = <fraction>

  用于指明焦点位置，需要配合 +<coordinate> 使用，以下面图形为例：



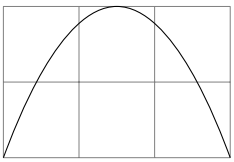
```
\begin{tikzpicture}[scale = 1]
   \draw[help lines] (0,0) grid (3,2);
   \draw (0,0) parabola[bend pos=0.5] bend +(0,2) +(3,0);
\end{tikzpicture}
```

图 2.3.19 Path:bend pos

bend pos = 0.5 指明了将曲线中点设置为焦点，紧随其后的 +(0,2) 和 +(3,0) 指明了该抛物线所占的矩形应为 2cm 高，3cm 宽。

- parabola height = <dimension>

  等效于：[bend pos=0.5,bend=+(0pt,<dimension>)].



```
\draw[help lines] (0,0) grid (3,2);
\draw (0,0) parabola[parabola height=2cm] +(3,0);
```

图 2.3.20 Path:parabola height

- bend at start/end

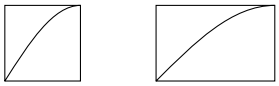
  将焦点放在起始点/终点。

3.4.2 三角函数

绘制三角函数的语法如下：

```
\path ... sin<coordinate or cycle> ...;
```

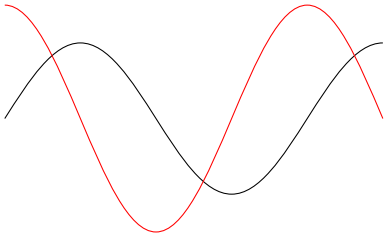
关键字 sin 左右边的坐标将作为区间 $[0,\pi/2]$ 的端点对应的正弦函数坐标，绘制曲线。



```
\draw (0,0) rectangle (1,1) (0,0) sin (1,1)
   (2,0) rectangle +(1.57,1) (2,0) sin +(1.57,1);
```

图 2.3.21 Path:sin

类似的，也有 cos 关键字，配合 cos 与 sin 关键字可以绘制出完整的三角函数图像。



```
\begin{tikzpicture}[scale = 1]
   \draw (0,0) sin (1,1) cos (2,0) sin (3,-1) cos (4,0) sin
        (5,1);
   \draw[color=red] (0,1.5) cos (1,0) sin (2,-1.5) cos (3,0)
        sin (4,1.5) cos (5,0);
\end{tikzpicture}
```

图 2.3.22 Path: 三角函数图像

3.5 高阶操作

3.5.1 To Path 操作

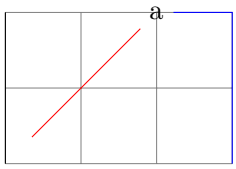
to 关键字的用法如下：

```
\path ... to[<options>]<nodes><coordinate or cycle> ...;
```

在默认情形下，to 与 – 的效果相同，下文所介绍的修饰部分在 – 关键字上也可以使用，但 to 关键字可以调用更多的修饰以达到更复杂的路径绘制效果。

基础用法

to 关键字的基础用法和 – 类似。

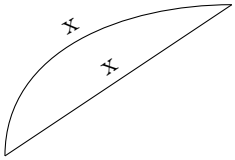



```
\begin{tikzpicture}[scale = 1]
   \draw[help lines] (0,0) grid (3,2);
   \node (a) at (2,2) {a};
   \draw (0,0) to (0,2);
   \draw[red] (10pt,10pt) to (a);
   \draw[blue] (3,0) -- (3,2) -- (a) -- cycle;
\end{tikzpicture}
```

图 2.3.23 Path:to

to path 上的 Nodes

在 to-path 路径上添加节点，与 – 上语法一致。其中 out 和 in 分别表示起点出发方向与终点进入方向，是 to 关键字独有的语法。



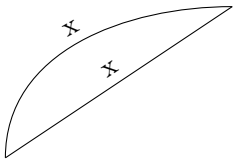

```
\begin{tikzpicture}
   \draw (0,0) to node [sloped,above] {x} (3,2);
   \draw (0,0) to[out=90,in=180] node [sloped,above] {x} (3,2);
\end{tikzpicture}
```

图 2.3.24 Path:to-node

由上例可以看出，添加 node 往往使得语句可读性下降，TikZ 还提供了将 node 写在 to 修饰中的语法，其几个常用修饰如下：

- edge node = <node specification>

  该修饰可用于将 node 节点写在 to 的修饰中。




```
\draw (0,0) to [edge node={node [sloped,above] {x}}] (3,2);
\draw (0,0) to [out=90,in=180,
   edge node={node [sloped,above] {x}}] (3,2);
```

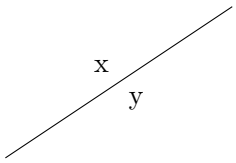
图 2.3.25 Path:edge node

- edge label = <text>

  上文的语法还是比较复杂，在控制要求没那么严格的情况下，可以使用 edge label 修饰，它相当于 edge node=node[auto]<text>.

- edge label' = <text>

  相当于：edge node=node[auto,swap]<text>.

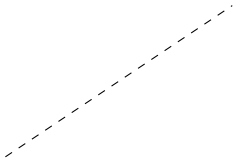



```
\draw (0,0) to [edge label=x, edge label'=y] (3,2);
```

图 2.3.26 Path:edge label

全局样式

TikZ 为 to 设置了 every to 修饰，这也是与 – 关键字的另一区分。



```
\begin{tikzpicture}[every to/.style={append after command={[
     draw,dashed]}}]
   \draw (0,0) to (3,2);
\end{tikzpicture}
```

图 2.3.27 Path:every to

to path = <path> 可以辅助我们做一些路径微操。

- to path 深入理解。

  每当 to 关键字起作用，就会创建一条对应的路径，在这个路径中会产生以下几个宏。

  – \tikztostart 起始点坐标

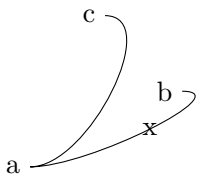
  – \tikztotarget 终点坐标

  – \tikztonodes 中间节点

  一个标准的包含 to 关键字的路径是由以下形式构成的

```
-- (\tikztotarget) \tikztonodes
```

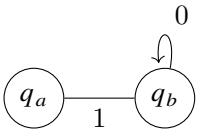
  我可以修改上述的 path 形式




```
\begin{tikzpicture}[to path={
   .. controls +(1,0) and +(1,0) .. (\tikztotarget) \
        tikztonodes}]
   \node (a) at (0,0) {a};
   \node (b) at (2,1) {b};
   \node (c) at (1,2) {c};
   \draw (a) to node {x} (b)
        (a) to (c);
\end{tikzpicture}
```

图 2.3.28 Path:to path

我们也可以自定义自己的 path 样式




```
\tikzset{
   my loop/.style={to path={
      .. controls +(80:1) and +(100:1) .. (\tikztotarget) \
           tikztonodes}},
   my state/.style={circle,draw}}

\begin{tikzpicture}[shorten >=2pt]
   \node [my state] (a) at (210:1) {$q_a$};
   \node [my state] (b) at (330:1) {$q_b$};
   \draw[->] (a) to node[below] {1} (b)
      to [my loop] node[above right] {0} (b);
\end{tikzpicture}
```

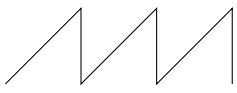
图 2.3.29 Path: 自定义 to path

- execute at begin to = <code>
  在 at 之前执行 <code>
- execute at end to = <code>
  在 at 之后执行 <code>

3.5.2 foreach 操作

在 path 中使用 foreach 的语法如下：

```
\path ...foreach<variables>[<options>] in {<path commands>} ...;
```



```
\draw (0,0) foreach \x in {1,...,3} { -- (\x,1) -- (\x,0) };
```

图 2.3.30 Path:foreach

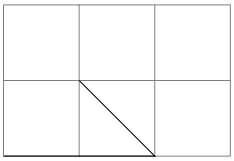
3.5.3 Let 操作

使用 let 关键字需要加载 calc 包，let 操作并不会拓展路径，主要起到辅助绘图作用，其语法如下。

```
\path ... let<assignment>,<assignment>,<assignment>... in ...;
```

在 <assignment> 中可以为坐标命名，以便整个语句中调用。具体的存储原理这里省略。

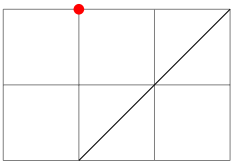
- \n <number register>
  这将存储一些数值，比如 \n1 = 1pt+2pt 再次调用 \n1 时，将传值：3pt。
- \p<point register> = <coordinate>
  与 n 类似，\p 用来保存坐标。



```
\begin{tikzpicture}[scale = 1]
   \draw [help lines] (0,0) grid (3,2);
   \draw let \p{foo} = (1,1), \p2 = (2,0) in
      (0,0) -- (\p2) -- (\p{foo});
\end{tikzpicture}
```

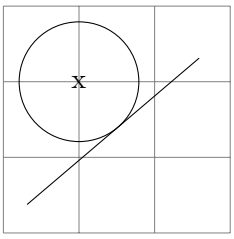
图 2.3.31 Path: 临时存储变量

- \x \y
  与上文 \p 类似，但是仅存储 x 轴和 y 轴坐标。



```
\begin{tikzpicture}
   \draw [help lines] (0,0) grid (3,2);
   \draw (1,0) coordinate (first point)
      -- (3,2) coordinate (second point);
   \fill[red] let \p1 = (first point),
                \p2 = (second point) in
      (\x1,\y2) circle [radius=2pt];
\end{tikzpicture}
```

图 2.3.32 Path:let 例子




```
\begin{tikzpicture}
   \draw [help lines] (0,0) grid (3,3);
   \coordinate (a) at (rnd,rnd);
   \coordinate (b) at (3-rnd,3-rnd);
   \draw (a) -- (b);
   \node (c) at (1,2) {x};
   \draw let \p1 = ($ (a)!(c)!(b) - (c) $),
      \n1 = {veclen(\x1,\y1)}
      in circle [at=(c), radius=\n1];
\end{tikzpicture}
```

图 2.3.33 Path:let 例子 2

3.5.4 路径存储

上述的 \p 仅能在一个语句内存储使用路径，如果需要在整个绘图环境中存储，则需要用到以下修饰。

- save path = <macro>
  存储路径
- use path = <macro>
  使用已存储的路径

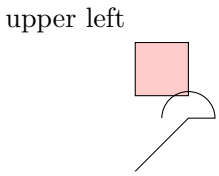
```
\begin{tikzpicture}
   \path[save path=\pathA,name path=A] (0,1) to [bend left]
        (1,0);
   \path[save path=\pathB,name path=B]
      (0,0) .. controls (.33,.1) and (.66,.9) .. (1,1);
   \fill[name intersections={of=A and B}] (intersection-1)
        circle (1pt);
   \draw[blue][use path=\pathA];
   \draw[red] [use path=\pathB];
\end{tikzpicture}
```

图 2.3.34 Path:use path

3.6 拓展用法

3.6.1 SVG 操作

svg 关键字可以让我们使用 svg 语法绘图，需要用到 svg.path 包。svg 在 TikZ 中的语法请查官方文档[4]，这里只给出示例。




```
\begin{tikzpicture}[scale = 1]
   \filldraw [fill=red!20] (0,1) svg[scale=2] {h 10 v 10 h
        -10}
      node [above left] {upper left} -- cycle;
   \draw svg {M 0 0 L 20 20 h 10 a 10 10 0 0 0 -20 0};
\end{tikzpicture}
```

图 2.3.35 Path:svg

3.6.2 Plot 操作

plot 关键字主要用于文件读取并绘制图像，语法请查官方文档。

[4]后续可能会更新

四、Path Style

4.1 样式概述

在 path 中添加修饰是十分自由的，在路径的任意地点添加的全局修饰都将起到相同的效果，同一段路径也允许有多个修饰，以下语句的效果相同。

```
\path [draw,fill] (0,0) circle (1cm);
\path [draw] [fill] (0,0) circle (1cm);
\path [fill] (0,0) circle (1cm) [draw];
\draw [fill] (0,0) circle (1cm);
\fill (0,0) [draw] circle (1cm);
\filldraw (0,0) circle (1cm);
```

下面指令都是基于 \path 指令衍生出来的常用指令：

- \draw

  等效于 \path[draw]；绘制边线。

- \fill

  等效于 \path[fill]；填充颜色。

- \filldraw

  等效于 \path[fill,draw]；绘制边线并填充颜色。

- \pattern

  等效于 \path[pattern]；填充图案。

- \shade

  等效于 \path[shade]；产生投影。

- \shadedraw

  等效于 \path[shadedraw]；绘制边线与投影。

- \clip

  等效于 \path[clip]；切片？

- \useasboundingbox

  等效于 \path[useasboundingbox]；绘制边框。

路径具有多种修饰，且部分修饰的值可能会重复，例如 color，有 fill = <color>，draw = <color> 等，如果不指定修饰名 (键) 则默认全部相关修饰均采用。


```
\begin{tikzpicture}[scale = 1]
   \path[yshift = 0cm, fill = red!20] (0,0) circle (1ex);
   \path[yshift = 2em, draw = red!20] (0,0) circle (1ex);
   \path[yshift = 4em, red!20,fill,draw] (0,0) circle (1ex);
\end{tikzpicture}
```

图 2.4.1 Path Style:color

4.2 线条样式

4.2.1 线条粗细

- line width = <dimension> (默认：0.4pt)
  控制线条粗细，该键可以省略。


```
\draw[line width = 5pt] (0,0) -- (3,0);
```

图 2.4.2 Path Style:line width

TikZ 为我们预设了几种常用的粗细，它们的值如下：

表 2.1 line width: 预设值

名称	粗细	名称	粗细	名称	粗细
ultra thin	0.1pt	very thin	0.2pt	thin	0.4pt
semithick	0.6pt	thick	0.8pt	very thick	1.2pt
ultra thick	1.6pt				

```
\draw [yshift = 0, ultra thin] (0,0) -- (3,0); % 0.1pt
\draw [yshift = -1em, very thin] (0,0) -- (3,0); % 0.2pt
\draw [yshift = -2em, thin]    (0,0) -- (3,0);  % 0.4pt
\draw [yshift = -3em, semithick] (0,0) -- (3,0); % 0.6pt
\draw [yshift = -4em, thick]   (0,0) -- (3,0);  % 0.8pt
\draw [yshift = -5em, very thick] (0,0) -- (3,0); % 1.2pt
\draw [yshift = -6em, ultra thick] (0,0) -- (3,0); % 1.6pt
```

图 2.4.3 Path Style: 线条粗细关键字

4.2.2 描边样式

- line cap = <type> (默认：butt)
  用于指明线条端点样式，有 round，rect，butt 三种样式：

```
\begin{tikzpicture}[scale = 1]
   \begin{scope}[line width=10pt]
      \draw[line cap=round] (0,1 ) -- +(3,0);
      \draw[line cap=butt] (0,.5) -- +(3,0);
      \draw[line cap=rect] (0,0 ) -- +(3,0);
   \end{scope}
   \draw[white,line width=1pt]
      (0,0 ) -- +(3,0) (0,.5) -- +(3,0) (0,1 ) -- +(3,0);
\end{tikzpicture}
```

图 2.4.4 Path Style:line cap

- line join = <type> (默认：miter)
  用于指定线条拐角处的样式，有 round，bevel，miter 三种。

```
\begin{tikzpicture}[line width=10pt]
   \draw[line join=round] (0,0) -- ++(.5,1) -- ++(.5,-1);
   \draw[line join=bevel] (1.25,0) -- ++(.5,1) -- ++(.5,-1);
   \draw[line join=miter] (2.5,0) -- ++(.5,1) -- ++(.5,-1);
   \useasboundingbox (0,1.5); % enlarge bounding box
\end{tikzpicture}
```

图 2.4.5 Path Style:line join

- miter limit = <factor> (默认：10)

  当夹角过小时，拐角处往往会显得过尖，用 miter limit 可以限制这一情况。

```
\begin{tikzpicture}[line width=5pt]
   \draw (0,0) -- ++(5,.5) -- ++(-5,.5);
   \draw[miter limit=25] (6,0) -- ++(5,.5) -- ++(-5,.5);
   \useasboundingbox (14,0); % make bounding box bigger
\end{tikzpicture}
```

图 2.4.6 Path Style:miter limit

4.2.3 虚线样式

- dash pattern = <dash pattern>

  设置虚线样式，值中有两个关键字 on 表示绘制，off 表示空缺。


```
\begin{tikzpicture}[dash pattern=on 2pt off 3pt on 4pt off 4pt]
   \draw (0pt,0pt) -- (3.5cm,0pt);
\end{tikzpicture}
```

图 2.4.7 Path Style:dash pattern

- dash phase = <dash phase>

  设置线条的起始位置。


```
\begin{tikzpicture}[dash pattern=on 20pt off 10pt]
   \draw[dash phase=0pt] (0pt,3pt) -- (3.5cm,3pt);
   \draw[dash phase=10pt] (0pt,0pt) -- (3.5cm,0pt);
\end{tikzpicture}
```

图 2.4.8 Path Style:dash phase

- dash = <dash pattern> phase <dash phase>

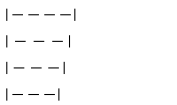
  可以将上面两个修饰合并为一个。

```
\begin{tikzpicture}
    \draw [dash=on 20pt off 10pt phase 0pt] (0pt,3pt) -- (3.5cm
        ,3pt);
    \draw [dash=on 20pt off 10pt phase 10pt] (0pt,0pt) -- (3.5
        cm,0pt);
\end{tikzpicture}
```

图 2.4.9 Path Style:dash

- dash expand off

  当路径长度不是一个 on 与 off 的整数倍时，往往会出现被截断的线段，使用该修饰可以强制使用完整的线段，将 off 增长以填充多余的区域。需要加载 decorations 包。



```
\begin{tikzpicture}[|-|, dash pattern=on 4pt off 2pt]
    \draw [dash expand off] (0pt,30pt) -- (26pt,30pt);
    \draw [dash expand off] (0pt,20pt) -- (24pt,20pt);
    \draw [dash expand off] (0pt,10pt) -- (22pt,10pt);
    \draw [dash expand off] (0pt, 0pt) -- (20pt, 0pt);
\end{tikzpicture}
```

图 2.4.10 Path Style:dash expand off

- 预设样式

  与 line width 类似的，TikZ 为我们提供了多个预设的虚线样式。

表 2.2 Path Style - 预设样式

名称	基础样式	密集样式	松散样式
点	dotted	densely dotted	loosely dotted
虚线	dashed	densely dashed	loosely dashed
点线	dash dot	densely dash dot	loosely dash dot
线点点	dash dot dot	densely dash dot dot	loosely dash dot dot

```
\draw [yshift = 0,solid]                (0,0) -- (3,0);
% 点
\draw [yshift = -2em,dotted]            (0,0) -- (3,0);
\draw [yshift = -3em,densely dotted]    (0,0) -- (3,0);
\draw [yshift = -4em,loosely dotted]    (0,0) -- (3,0);
% 虚线
\draw [yshift = -6em,dashed]            (0,0) -- (3,0);
\draw [yshift = -7em,densely dashed]    (0,0) -- (3,0);
\draw [yshift = -8em,loosely dashed]    (0,0) -- (3,0);
% 点线
\draw [yshift = -10em,dash dot]         (0,0) -- (3,0);
\draw [yshift = -11em,densely dash dot] (0,0) -- (3,0);
\draw [yshift = -12em,loosely dash dot] (0,0) -- (3,0);
% 线点点
\draw [yshift = -14em,dash dot dot]     (0,0) -- (3,0);
\draw [yshift = -15em,densely dash dot dot] (0,0) -- (3,0);
\draw [yshift = -16em,loosely dash dot dot] (0,0) -- (3,0);
```

图 2.4.11 Path Style: 预设样式

4.2.4 双线条

- double = <core color>

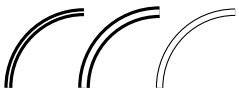
  该修饰下绘制的线条将拥有两条线，它的内部处理方式为：首先根据 double 修饰提供的颜色绘制一条路径 (默认为白色)，再在所绘制路径的两边绘制新的路径，颜色由 draw 修饰给出 (默认为黑色)。

```
\begin{tikzpicture}[scale = 1]
   \draw (0,0) -- (1,1);
   \draw[draw=white,double=red,very thick] (0,1) -- (1,0);
\end{tikzpicture}
```

图 2.4.12 Path Style:double

- double distance = <dimension> (默认：0.6pt)

  设置两条边线的距离。

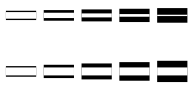


```
\begin{tikzpicture}[scale = 1]
   \draw[very thick,double] (0,0) arc (180:90:1cm);
   \draw[very thick,double distance=2pt] (1,0) arc (180:90:1cm);
   \draw[thin,double distance=2pt] (2,0) arc (180:90:1cm);
\end{tikzpicture}
```

图 2.4.13 Path Style:double distance

- double distance between line centers = <dimension>

  该修饰与 double distance 类似，但它的距离计算是指两边线的中点的距离



```
\begin{tikzpicture}
   \foreach \lw in {0.5,1,1.5,2,2.5}
      \draw[line width=\lw pt,double distance between line
           centers=3pt] (\lw,2em) -- ++(4mm,0);
   \foreach \lw in {0.5,1,1.5,2,2.5}
      \draw[line width=\lw pt,double distance = 3pt] (\lw,0)
           -- ++(4mm,0);
\end{tikzpicture}
```

图 2.4.14 Path Style:double distance between line centers

4.2.5 线条修饰

启用 decorations.pathmorphing 包，可以启动多种线条样式。


```
\begin{tikzpicture}
   \node [circular drop shadow={shadow scale=1.05},minimum
        size=3.13cm,decorate, decoration=zigzag,fill=blue!20,
        draw,thick,circle] {Hello!};
\end{tikzpicture}
```

图 2.4.15 Path Style:decorate

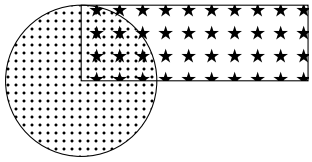
4.3 填充样式

4.3.1 填充图案

TikZ 为我们提供了很多内置的图案用来填充，值得说明的是，这些图案并不能进行大小或方向的修改，除非我们重新定义一种图案。

- pattern = <name>

  该修饰用来指定填充的图案。需要加载 patterns 包。



```
\begin{tikzpicture}[scale = 1]
   \draw[pattern=dots] (0,0) circle (1cm);
   \draw[pattern=fivepointed stars] (0,0) rectangle (3,1);
\end{tikzpicture}
```

图 2.4.16 Path Style:pattern

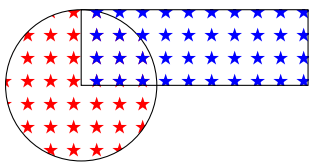
这里简单给出几个图案样式值如下表：

表 2.3 Path Style:patterns 部分样式

样式名	样式	样式名	样式
horizontal lines	横线	vertical lines	竖线
north east lines	斜线	north west lines	斜线
grid	网格	crosshatch	斜网格
dots	点	crosshatch dots	点
fivepointed stars	五角星	sixpointed stars	六芒星
bricks	砖型	checkerboard	棋盘型

- pattern color = <color>

  用于指定图案颜色



```
\begin{tikzpicture}[scale = 1]
   \draw[pattern color=red,pattern=fivepointed stars] (0,0)
        circle (1cm);
   \draw[pattern color=blue,pattern=fivepointed stars] (0,0)
        rectangle (3,1);
\end{tikzpicture}
```

图 2.4.17 Path Style:pattern color

4.3.2 填充渐变

- shade

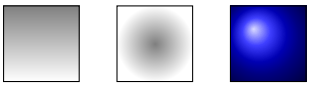
  基本的渐变填充，默认产生一个从上到下的灰色渐变效果。


```
\shade (0,0) circle (1ex);
```

图 2.4.18 Path Style:shade

- shading = <name>

  比 shade 提供了过多的渐变样式，总体上由 axis，radial，ball 三种渐变样式。



```
\shadedraw [shading = axis] (0,0) rectangle (1,1);
\shadedraw [shading = radial,xshift = 1.5cm] (0,0) rectangle
    (1,1);
\shadedraw [shading = ball,xshift = 3cm] (0,0) rectangle (1,1);
```

图 2.4.19 Path Style:shading

我们最常用到的是线性渐变，下面姐扫两个线性渐变中常用到的修饰。

- left/right color = <color>

  用来指定渐变色。

```
\shadedraw [left color=red,right color=blue] (0,0) rectangle
    (1,1);
```

图 2.4.20 Path Style:left/right color

- shading angle = <degrees>

  可以用来改变线性渐变的角度。

```
\shadedraw [shading=axis,shading angle=90] (0,0)
    rectangle (1,1);
```

图 2.4.21 Path Style:shading angle

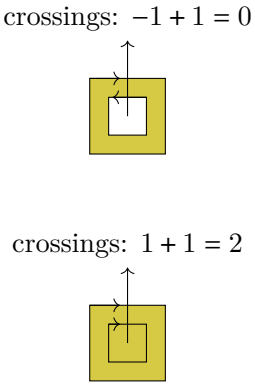
4.4 路径范围

4.4.1 基础概念

在讲述填充图案之前，我们需要了解 TikZ 的内部点计算方法。在默认情形下，我们总是对整个图案进行填充，但有时候我们需要对图案的交集等进行处理，这就需要了解到 TikZ 对内部点的判断两种算法。

- nonzero rule

  TikZ 采用如下计算方式来计算是否为内部点：从某点出发沿任何方向绘制一条射线，如果射线遇到路径，采用以下方式计数：如果路径方向为从左到右，计数加一，如果路径方向为从右到左，计数减一。最终计数若非 0 则为内部点，否则为外部点。可形象地将外部视为中空区域。

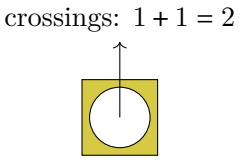



```
\begin{tikzpicture}
    \filldraw[fill=yellow!80!black]
    % Clockwise rectangle
    (0,0) -- (0,1) -- (1,1) -- (1,0) -- cycle
    % Counter-clockwise rectangle
    (0.25,0.25) -- (0.75,0.25) -- (0.75,0.75) -- (0.25,0.75) -- cycle;
    \draw[->] (0,1) -- (.4,1);
    \draw[->] (0.75,0.75) -- (0.3,.75);
    \draw[->] (0.5,0.5) -- +(0,1) node[above] {crossings: $-1+1 = 0$};
    \begin{scope}[yshift=-3cm]
        \filldraw[fill=yellow!80!black]
        % Clockwise rectangle
        (0,0) -- (0,1) -- (1,1) -- (1,0) -- cycle
        % Clockwise rectangle
        (0.25,0.25) -- (0.25,0.75) -- (0.75,0.75) -- (0.75,0.25) -- cycle;
        \draw[->] (0,1) -- (.4,1);
        \draw[->] (0.25,0.75) -- (0.4,.75);
        \draw[->] (0.5,0.5) -- +(0,1) node[above] {crossings: $1+1 = 2$};
    \end{scope}
\end{tikzpicture}
```

图 2.4.22 Path Style:nonzero rule

- even odd rule

  另一种内部点算法：同样绘制射线，不进行减法运算，始终累计遇到的路径数，如果计算为奇数，则为内部点；否则为外部点。



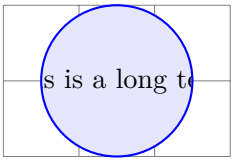

```
\begin{tikzpicture}
    \filldraw[fill=yellow!80!black,even odd rule]
        (0,0) rectangle (1,1) (0.5,0.5) circle (0.4cm);
    \draw[->] (0.5,0.5) -- +(0,1) [above] node{crossings: $1+1
        = 2$};
\end{tikzpicture}
```

图 2.4.23 Path Style:even odd rule

TikZ 还为我们提供了下面两个修饰对 path 路径创建的图形进行定位等操作。

- path picture = <code>

  如果调用了修饰，那么在任何通过 \path(及其衍生的 fill，shade 等) 指令创建的图形，都会以此为基础创建一个 scope 区域，并在此区域中执行 <code>。

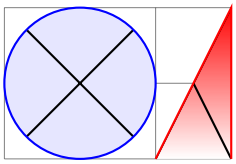



```
\begin{tikzpicture}
    \draw [help lines] (0,0) grid (3,2);
    \filldraw [fill=blue!10,draw=blue,thick] (1.5,1) circle (1)
        [path picture={
            \node at (path picture bounding box.center) {
                This is a long text.
            };}
    ];
\end{tikzpicture}
```

图 2.4.24 Path Style:path picture

- path picture bounding box

任何创建的路径都可认为被包括在一个矩形框内，通过调用 path picture bounding box 可以获得矩形框的位置。



```
\begin{tikzpicture}[cross/.style={path picture={
   \draw[black]
      (path picture bounding box.south east) --
      (path picture bounding box.north west)
      (path picture bounding box.south west) --
      (path picture bounding box.north east);
   }}]
   \draw [help lines] (0,0) grid (3,2);
   \filldraw [cross,fill=blue!10,draw=blue,thick] (1,1) circle
         (1);
   \path [cross,top color=red,draw=red,thick] (2,0) -- (3,2)
        -- (3,0);
\end{tikzpicture}
```

图 2.4.25　Path Style:path picture bounding box

4.4.2 边界框

在绘制图形的过程中，TikZ 会默认将所有点都包含在图片范围内，然而有时候我们会让某些点突破图片范围。

- use as bounding box

  在某个路径中启用该修饰后，图像的边界框将由该路径确定，随后的路径不再对边界框有影响。然而如果之前的路径确定了更大的边界框，该修饰并不会使得边界框变小 (可以通过 \pgfresetboundingbox 重置边界框)。

  此外，该修饰因为常被用来指定边界框大小，TikZ 也提供了指令形式 \useasboundingbox

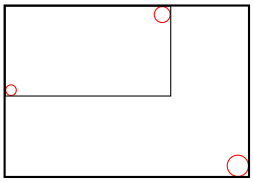
  Left of picture  right of picture.

```
Left of picture
   \begin{tikzpicture}
      \useasboundingbox (0,0) rectangle (3,1);
      \fill (.75,.25) circle (.5cm);
   \end{tikzpicture}
right of picture.
```

图 2.4.26　Path Style:useasboundingbox

- current bounding box

  边界框不仅可以被指定，也可以被调用：

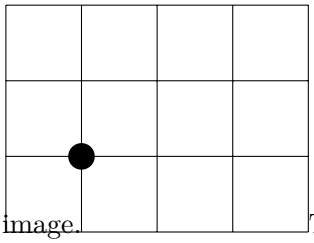


```
\begin{tikzpicture}
    \draw[red] (0,0) circle (2pt);
    \draw[red] (2,1) circle (3pt);
    \draw (current bounding box.south west) rectangle
        (current bounding box.north east);
    \draw[red] (3,-1) circle (4pt);
    \draw[thick] (current bounding box.south west) rectangle
        (current bounding box.north east);
\end{tikzpicture}
```

图 2.4.27 Path Style:current bounding box

- trim left = <dimension or coordinate> (默认：0pt)

  该修饰将传入的值设置为边界框的左边界，如果是坐标，则只使用 x 值。

Text before image.  Text after image.

```
Text before image.%
\begin{tikzpicture}[trim left]
\draw (-1,-1) grid (3,2);
\fill (0,0) circle (5pt);
\end{tikzpicture}%
Text after image.
```

图 2.4.28 Path Style:useasboundingbox

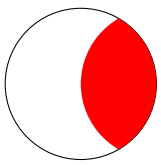
- trim right = <dimension or coordinate>

4.4.3 剪裁蒙版

剪裁路径是指将绘制的路径限定在一定的区域内，作用类似于 PS 的蒙版遮罩。

- clip

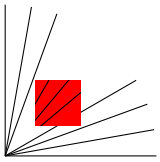
  该修饰对应的路径将作为蒙版，在整个 scope 中起作用。



```
\begin{tikzpicture}[scale = 1]
    \draw[clip] (0,0) circle (1cm);
    \fill[red] (1,0) circle (1cm);
\end{tikzpicture}
```

图 2.4.29 Path Style:clip

TikZ 提供了指令形式的 \clip。



```
\begin{tikzpicture}[scale = 2]
   \draw (0,0) -- ( 0:1cm);
   \draw (0,0) -- (10:1cm);
   \draw (0,0) -- (20:1cm);
   \draw (0,0) -- (30:1cm);
   \begin{scope}[fill=red]
      \clip[fill] (0.2,0.2) rectangle (0.5,0.5);
      \draw (0,0) -- (40:1cm);
      \draw (0,0) -- (50:1cm);
      \draw (0,0) -- (60:1cm);
   \end{scope}
   \draw (0,0) -- (70:1cm);
   \draw (0,0) -- (80:1cm);
   \draw (0,0) -- (90:1cm);
\end{tikzpicture}
```

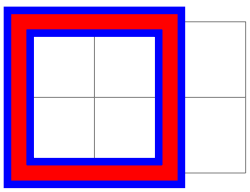
图 2.4.30 Path Style:\clip

4.5 复杂动作

通常 TikZ 绘制路径的顺序为: fill - draw - clip。然而有的时候我们需要对路径进行更为复杂的操作，例如多次填充图案等。

- preaction = <options>

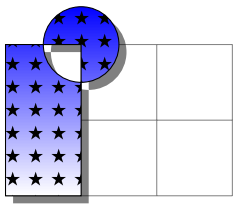
  当给 path 该修饰时，会创建一个对应的 scope。在这个 scope 中会首先强制执行 <options> 中的修饰，再执行其他修饰。



```
\begin{tikzpicture}[scale = 1]
   \draw[help lines] (0,0) grid (3,2);
   \draw [preaction={draw,line width=4mm,blue}]
      [line width=2mm,red] (0,0) rectangle (2,2);
\end{tikzpicture}
```

图 2.4.31 Path Style:preaction

同一个 path 可以拥有多个 preaction，以达到在同一个路径上多次绘制的效果。

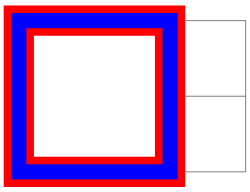


```
\begin{tikzpicture}
   \draw[help lines] (0,0) grid (3,2);
   \draw [pattern=fivepointed stars]
      [preaction={fill=black,opacity=.5, transform canvas={
           xshift=1mm,yshift=-1mm}}]
      [preaction={top color=blue,bottom color=white}]
         (0,0) rectangle (1,2)
         (1,2) circle (5mm);
\end{tikzpicture}
```

图 2.4.32 Path Style:multi-preaction

- postaction = <options>

  postaction 与 preaction 类似，只不过再其他修饰执行后再执行 postaction 中的修饰。



```
\begin{tikzpicture}
    \draw[help lines] (0,0) grid (3,2);
    \draw
        [postaction={draw,line width=2mm,blue}]
        [line width=4mm,red,fill=white] (0,0) rectangle (2,2);
\end{tikzpicture}
```

图 2.4.33 Path Style:postaction

五、Arrow

5.1 概述

TikZ 为我们提供了强大的 Arrow 处理功能，我们既可以使用预设的 Arrow 样式，自定义 Arrow，也可以自行修改预设的 Arrow。

TikZ 为我们预设的 Arrow 样式在 arrows.meta 包中。

绘制箭头大部分情况下需要满足两个基本条件：1. 有终止点 2. 不是闭合环路。除此之外遇到 clip，cycle 等修饰也将无法绘制箭头。

- arrows = <start arrow specification> - <end arrow specification>

  由于 arrows 修饰十分常用，在实际绘图中也可以省略该键，那么任何带 - 的修饰都会被认为是 arrows。

```
\begin{tikzpicture}
   \draw[->] (0,0) -- (1,0);
   \draw[>-Stealth] (0,0.3) -- (1,0.3);
\end{tikzpicture}
```

图 2.5.1 Arrow:arrows

上述例子十分简单，如果我们需要对箭头样式进行复杂的调整，可以使用 {} 添加多个修饰。


```
\draw[-{Stealth[red]}] (0,0) -- (1,0);
```

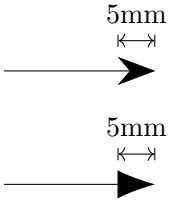
图 2.5.2 Arrow:arrows-{}

5.2 基本样式

5.2.1 箭头大小

- length = <dimension><line width factor><outer factor>

  这个参数控制着箭头沿发射方向的长度。




```
\begin{tikzpicture}[scale = 1]
   \begin{scope}[yshift = 0cm]
      \draw [-{Stealth[length=5mm]}] (0,0) -- (2,0);
      \draw [|<->|] (1.5,.4) -- node[above=1mm] {5mm} (2,.4);
   \end{scope}
   \begin{scope}[yshift = -1.5cm]
      \draw [-{Latex[length=5mm]}] (0,0) -- (2,0);
      \draw [|<->|] (1.5,.4) -- node[above=1mm] {5mm} (2,.4);
   \end{scope}
\end{tikzpicture}
```

图 2.5.3 Arrow:length

下面来深入了解一下 length 的三个参数，以 Latex arrow 为例，它的 length 对应的值为 length = 3pt 4.5 0.8。默认线宽为 0.4pt，最终计算的值为 $3\text{pt} + 4.5 \times 0.4\text{pt} = 4.8\text{pt}$。也即第二个参数 <line width factor> 若存在就与线宽进行乘法运算并累计入 length 的值。

上面计算中第三个参数并没有起作用，原因是 <outer factor> 仅在 path 存在 double 修饰时对 inner line width 和 line width 起作用。

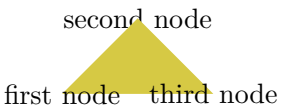
六、Node

6.1 节点基础

Node 是一个带有文字的简单图形 (矩形，圆形，点)。Node 不属于 path，通常在 path 已经绘制好后，或绘制之前产生。

Node 的最简单用法是在一些坐标点周围添加文字。与此同时，Node 也能添加图案以及复杂的颜色效果，甚至一些 Node 取消了文字。

添加 Node 并没有专用的 LaTeX 语法，通常用在 path 中用 node 指明。




```
\begin{tikzpicture}[scale = 1]
   \fill [fill=yellow!80!black]
         (0,0) node {first node}
      -- (1,1) node[behind path] {second node}
      -- (2,0) node {third node};
\end{tikzpicture}
```

图 2.6.1 Node 基本用法

6.1.1 Node 命令的语法

在 path 中添加 node 的完整语法如下：

```
\path ⋯ node <foreach statements> [<options>] (<name>) at(<coordinate>) :(animation attribute)={<
    options>} {<node contents>} ⋯;
```

另一种轻量化的写法

```
\node [<options>] (<name>) at(<coordinate>) :(animation);
```

主要用法

```
... node [options] {text} ...;
```

在 text 中添加 node 要显示的文字，[options] 指定样式，下面整理常用 [options]。

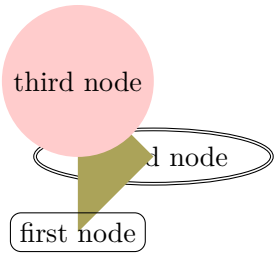
- 文字与颜色
  除了在 text 中指明颜色，在 [options] 也可以使用 [node contents=<text>] 指定文字，且在 [options] 中说明的颜色默认为文字颜色。


```
\begin{tikzpicture}[scale = 1]
   \path  (0,0)  node   [blue] {A}
          (1,0)  node   [red]  {B}
          (2,0)  node   [green,node contents=C]
          (3,0)  node   [node contents=D];
\end{tikzpicture}
```

图 2.6.2 Node 的文字与颜色

- 指定位置
  平面位置：[at=<coordinate>]：指定 node 的位置，当 node 在 path 中时，无效。
  图层位置：[behind path]：指定 node 的图层位置，效果见图2.6.1。默认样式为 [in front of path]

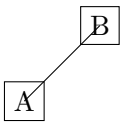
- 节点名
  节点名用于后续绘图指定节点，TikZ 允许至多两个节点名。
  节点名：[name=<name>]：节点的名称
  节点别名：[alias=<alias>]：节点别名
- 节点形状
  默认节点仅有文字，不具有形状，需要使用 [draw/fill] 命令指定对应形状。
  边框：[draw]：显示节点边线；
  底色：[fill=<color>]：显示节点底色
  边框形状：[shape = rectangle，circle，ellipse]：shape 可以省略，默认为矩形 rectangle ，更多形状可查阅官方资料。
  边框圆角：[rounded corners]
  边线数量：[double]




```
\begin{tikzpicture}[scale = 1]
   \fill[fill=yellow!60!black]
         (0,0) node [draw, rounded corners] {first node}
      -- (1,1) node [draw, double, behind path] {second node}
      -- (0,2) node [circle,fill=red!20] {third node};
\end{tikzpicture}
```

图 2.6.3 Node 形状

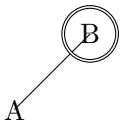
- 节点全局样式
  可以通过在 tikzpicture 环境开始处添加说明，给予全局节点样式。
  全部样式：[every node/.style = {}]




```
\begin{tikzpicture}[every node/.style={draw}]
   \draw (0,0) node {A} -- (1,1) node {B};
\end{tikzpicture}
```

图 2.6.4 Node 全部样式

指定样式：[every <shape> node/.style = {}]




```
\begin{tikzpicture}[every circle node/.style={draw,double}]
   \draw  (0,0)  node   {A}
      -- (1,1)  [circle] node {B};
\end{tikzpicture}
```

图 2.6.5 Node 指定样式

文字前后缀：[execute at begin/end node = {text}]

第一题：

```
\begin{tikzpicture}[execute at begin node = {第}, execute at
    end node = {题}]
  \node [execute at end node = {: }] {一};
\end{tikzpicture}
```

图 2.6.6

- 其他样式
  填充：[fill = <color>]：填充背景色
  缩放：[scale = <dimension>]：节点缩放
  边框粗细：[linewidth = <dimension>]：边框粗细

其他用法

- 节点动画
  通过:<animation attribute>=<options>, 可以设定动画[5]，下面只做简单举例


```
\begin{tikzpicture}[scale = 1]
  \node  :fill opacity={0s="1",2s="0",begin on=click}
         :rotate = {0s="0",2s="90",begin on=click}
         [fill = blue!20, draw = blue, ultra thick, circle
             ]
         {click me};
\end{tikzpicture}
```

图 2.6.7 Node 动画

- foreach
  foreach 语句仅允许紧跟在 node 之后出现。
  语法形式：foreach \x in {}
  在集合中可以使用... 表示省略类似的内容。

1 2 3

```
% 以下两句效果相同
\draw (0,0) node foreach \x in {1,2,3} at (\x,0) {\x};
\tikz \draw (0,0) node at (1,0) {1} node at (2,0) {2} node at
    (3,0) {3};
```

图 2.6.8 Node 一次迭代

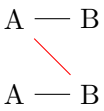
1,3 2,3 3,3 4,3
1,2 2,2 3,2 4,2
1,1 2,1 3,1 4,1

```
\node foreach \x in {1,...,4} foreach \y in {1,2,3} [draw] at
    (\x,\y) {\x,\y};
```

图 2.6.9 Node 两次迭代

[5]部分图形驱动并不支持动画，比如我的也不支持

- scope
  scope 用于限定范围，类似高级语言中的命名空间。
  和 LaTeX 中的环境十分相似，scope 需要 \begin 和 \end 来限定范围。
  在 \begin{scope}[name prefix = <text>] 限定范围名。
  使用时"name"+"node name" 即可。
  类似的，也可以使用 suffix。



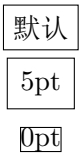

```
\begin{tikzpicture}[scale = 1]
   \begin{scope}[name prefix = top-]
      \node (A) at (0,1) {A};
      \node (B) at (1,1) {B};
      \draw (A) -- (B);
   \end{scope}
   \begin{scope}[name prefix = buttom-]
      \node (A) at (0,0) {A};
      \node (B) at (1,0) {B};
      \draw (A) -- (B);
   \end{scope}
   \draw [red] (top-A) -- (buttom-B);
\end{tikzpicture}
```

图 2.6.10 Node:sep

6.1.2 盒模型

盒模型[6]，这里指 Node 周围边距，底色等样式的控制。由于比一般的样式控制命令更多，而且重要，单独开一节做笔记。

- 边距 sep (默认: 0.3333em)
  总内边距：[inner sep = <dimension>]：边框与内部文字的边距



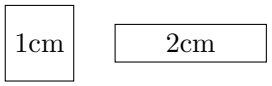

```
\begin{tikzpicture}[scale = 1]
   \draw  (0,0) node [inner sep = 0pt,draw] {0pt}
          (0,2em) node [inner sep = 5pt,draw] {5pt}
          (0,4em) node [draw] {默认};
\end{tikzpicture}
```

图 2.6.11 Node:inner sep

  左右/上下边距：[inner xsep/ysep = <dimension>]
  外边距：[outer sep = <dimension>]，外边距可能出现一些不准确的问题，可以在环境中加入 [outer sep = auto] 解决，与 inner sep 类使的，也可以指明 x/y 方向外边距。

- 最小高度/宽度
  最小高度/宽度用于限制节点与边线的最小距离
  最小距离：[minimum size]：最小高度与宽度
  最小高度：[minimum height], 最小宽度：[minimum width]

[6] 盒模型概念来自 css，与这里极其类使，但 TikZ 官方并没有指定这一系列样式的名称



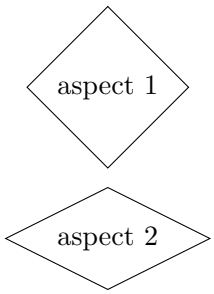

```
\begin{tikzpicture}[scale = 1]
   \draw  (0,0)  node   [minimum height = 1cm,draw] {1cm}
          (2,0)  node   [minimum width = 2cm,draw] {2cm};
\end{tikzpicture}
```

图 2.6.12 Node: 最小高度/宽度

6.1.3 边框形状

这里主要备注边框形状相关的内容。

- 横纵比
  横纵比：[shape aspect=<aspect ratio>]：外边框形状进行压缩。




```
\begin{tikzpicture}[scale = 1]
   \draw  (0,0)  node   [shape aspect=1,diamond,draw] {aspect
        1};
   \draw  (0,-2) node   [shape aspect=2,diamond,draw] {aspect
        2};
\end{tikzpicture}
```

图 2.6.13 Node:aspect

- 边框距 (默认: 1pt)
  TikZ 的边框距有两种计算方式，可以使用 [shape border uses incircle = <boolean>] 启动第二种边距计算，效果见下图：

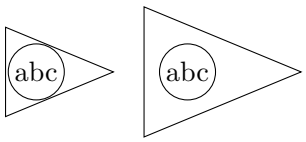
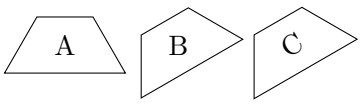


图 2.6.14 Node: 边距

- 旋转
  文字旋转：[rotate = <angle>]：文字和边框都将出现旋转。
  边框旋转：[rotate = <angle>]：仅边框旋转。




```
\begin{tikzpicture}[every node/.style={shape=trapezium, draw,
     shape border usincircle}]
   \node at (0,0) (A) {A};
   \node [shape border rotate=30] at (1.5,0) {B};
   \node [rotate=30] at (3,0) {C};
\end{tikzpicture}
```

图 2.6.15 Node: 旋转

- 边框粗细
  粗细：[linewidth = <dimension>]


```
\begin{tikzpicture}[scale = 1]
    \node [line width=2,draw] at (0,0) {A};
\end{tikzpicture}
```

图 2.6.16 Node: 边框粗细

6.2 节点样式

6.2.1 分割节点

在 node 文字中使用 \nodepart[<options>]{<part name>} 可以对节点进行切割；注意此时的 shape 形状后需加上 split，否则无效。同时需要声明 \use tikzlibrary shapes.multipart


```
\begin{tikzpicture}[scale = 1]
    \node [circle split,draw,double] {$q_1$ \nodepart{lower}
        $00$};
\end{tikzpicture}
```

图 2.6.17 Node: 分割节点

对于批量修改节点样式，可以使用类似 every lower node part/.style = color 的方法。

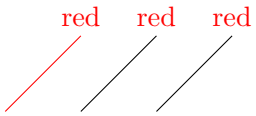

```
\begin{tikzpicture}[every lower node part/.style = red]
    \node [circle split,draw] {$q_1$ \nodepart{lower} $00$};
\end{tikzpicture}
```

图 2.6.18 Node: 分割节点样式

6.2.2 节点文字

文字本身包含颜色，字体，大小等样式，具体控制如下：

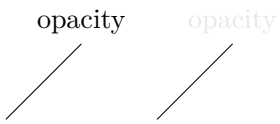
- 颜色
  文字颜色：[color = <color>]，这里的 color 可以省略，注意在节点中的颜色属性只影响节点文字，这与 \draw 不同




```
\begin{tikzpicture}[scale = 1]
    \draw[red]       (0,0) -- +(1,1) node[above] {red};
    \draw[text = red] (1,0) -- +(1,1) node[above] {red};
    \draw            (2,0) -- +(1,1) node[above,red] {red};
\end{tikzpicture}
```

图 2.6.19 Node: 文字颜色

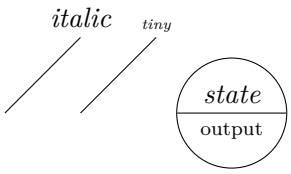
- 不透明度
  不透明度：[opacity = <value>]，注意这里是不透明度，1 表示完全不透明。




```
\begin{tikzpicture}[scale = 1]
   \draw[opacity = 1]      (0,0) -- +(1,1) node[above] {opacity
        };
   \draw            (2,0) -- +(1,1) node[above,opacity = 0.1]
        {opacity};
\end{tikzpicture}
```

图 2.6.20　Node: 文字透明度

- 文字字体
  文字字体：[node font = <font commands>]，这里的 node 可以省略。注意这里的 font 既可以指字体族，也可以控制字体大小。




```
\begin{tikzpicture}[every text node part/.style={font=\itshape
     },every lower node part/.style={font=\footnotesize}]
   \draw[node font=\itshape] (1,0) -- +(1,1) node[above] {
        italic};
   \draw[node font=\tiny] (2,0) -- +(1,1) node[above] {tiny};
   \node [circle split,draw] at (4,0) {state \nodepart{lower}
        output};
\end{tikzpicture}
```

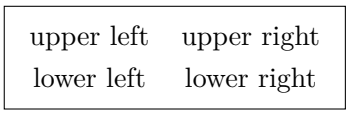
图 2.6.21　Node: 文字字体

- 文字高度与深度
  文字高度：[text height = <dimension>]
  文字深度：[text depth = <dimension>]
  这两个属性并不常用，一般情况下尽量用 inner sep 代替。

6.2.3 节点文本

文本格式包括文本框的长宽，对齐方式等。

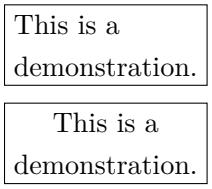
- 节点文本框
  与正文中的文本类似，节点中的文本也可以实现公式，换行，表格等功能。




```
\begin{tikzpicture}[scale = 1]
   \node [draw] {
      \begin{tabular}{cc}
         upper left & upper right\\
         lower left & lower right
      \end{tabular}
   };
\end{tikzpicture}
```

图 2.6.22　Node: 节点文本框

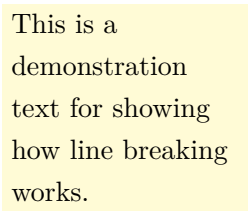
- 文本对齐
  对齐：[align = <alignment option>]，设置对其方式。




```
\begin{tikzpicture}[scale = 1]
    \node[draw,align=left]at (0,0) {This is a\\demonstration.};
    \node[draw,align=center] at (0,-1.3) {This is a\\
        demonstration.};
\end{tikzpicture}
```

图 2.6.23 Node: 文本对齐

文本对齐会自动分割长单词，可以使用 align = flush <alignment option> 来取消长单词。

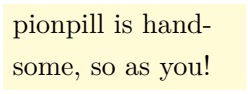



```
\begin{tikzpicture}[scale = 1]
    \node[fill=yellow!20,text width=3cm,align=flush left] {This
        is a demonstration text for showing how line breaking
        works.};
\end{tikzpicture}
```

图 2.6.24 Node: 分割长单词

alignment option 具体参数见属性百科：

- 文本宽度
  文本宽：[text width = <dimension>]：限定最大文本宽

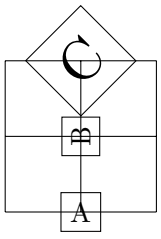



```
\begin{tikzpicture}[scale = 1]
    \draw (0,0) node [text width=8em, fill = yellow!20] {
        pionpill is handsome, so as you!};
\end{tikzpicture}
```

图 2.6.25 Node: 文本宽度

6.2.4 节点变换

常用的节点的变换 (transform) 修饰有缩放与旋转。




```
\begin{tikzpicture}[every node/.style = draw]
    \draw (0,0) grid (2,2);
    \draw (1,0) node {A};
    \draw (1,1) node [rotate = 90] {B};
    \draw (1,2) node [rotate = 45,scale = 2] {C};
\end{tikzpicture}
```

图 2.6.26 Node: 节点变换

6.2.5 节点复用

节点复用，即在使用过节点后，再使用节点。使用 also 关键字，即可启用之前定义过的节点。复用的节点无法添加文字。


```
\begin{tikzpicture}[scale = 1]
    \node [circle,draw] (a) {Hello};
    \node also [label=above:world] (a);
\end{tikzpicture}
```

图 2.6.27 Node:also

类似的，也可以使用 [late options = {}] 达到同样的效果。


```
\begin{tikzpicture}[scale = 1]
    \node [draw,circle] (a) {Hello};
    \path [late options = {name=a,label=above:world}];
\end{tikzpicture}
```

图 2.6.28 Node:late option

6.3 节点布局

6.3.1 定位

节点的位置往往由与之相关的坐标决定，默认以坐标为中心生成节点，同时也可在坐标周围生成节点。

- 节点锚点
  锚点：[anchor = <anchor name>]：锚点属性决定了节点将在坐标的哪个方向生成。
  这里 anchor 的意思是坐标位于节点的方向。

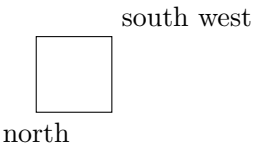
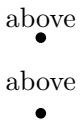


图 2.6.29 Node: **锚点**

- 偏移
  偏移：[<offset>]：无需属性名，效果与 anchor 类似，但是能更精确地控制偏移量。




```
\begin{tikzpicture}[scale = 1]
    \fill (0,0) circle (2pt) node [above] {above};
    \fill (0,-1) circle (2pt) node [above=5pt] {above};
\end{tikzpicture}
```

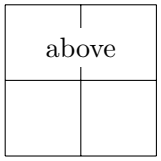
图 2.6.30 Node: **偏移**

6.3.2 高阶布局

除了简单的定位，还可以使用 positioning 包来进行更为高阶的定位操作。在启动了该包后，原本的 <dimension> 参数将被提升为 <specification>。<specification> 参数通常由两部分组成：<shifting part> + <of-part>

<shifting part> 为主要控制参数，通常有三种形式：

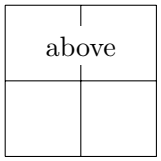
- <dimension> 形式
  这种形式在 <dimension> 参数的基础上，还可以使用数学式。




```
\begin{tikzpicture}[scale = 1]
   \draw (0,0) grid (2,2);
   \node at (1,1) [above = 2pt+3pt,fill=white] {above};
\end{tikzpicture}
```

图 2.6.31 Node: 高阶布局-数学形式

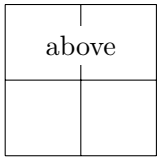
- <number> 形式
  这种形式可以不包含单位，其他和上面那个差别不大。




```
\begin{tikzpicture}[scale = 1]
   \draw (0,0) grid (2,2);
   \node at (1,1) [above = .2,fill=white] {above};
   % south border of the node is now 2mm above (1,1)
\end{tikzpicture}
```

图 2.6.32 Node: 高阶布局-数字形式

- and 组合形式
  可以通过 and 将多个偏移修饰组合，这里 and 的作用并不明显，下一节将详细说明。

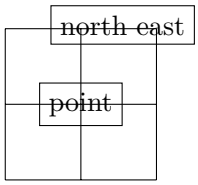



```
\begin{tikzpicture}[scale = 1]
   \draw (0,0) grid (2,2);
   \node at (1,1) [above = .2 and 2mm,fill=white] {above};
   % south border of the node is also 2mm above (1,1)
\end{tikzpicture}
```

图 2.6.33 Node: 高阶布局-组合形式

<of-part> 可以进行相对定位，注意点如下：

- 以坐标为参照的定位
  通过 of 关键字可以调用坐标点的方位进行相对定位。

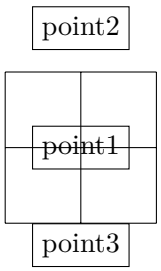



```
\begin{tikzpicture}[scale = 1]
   \draw (0,0) grid (2,2);
   \node (point) [draw] at (1,1) {point};
   \node [above = 5mm of point.north east,draw] {north east};
\end{tikzpicture}
```

图 2.6.34 Node:of-part 坐标定位

定位逻辑：首先找到 point 的 north east 位置，然后向 above 方向移动了 5mm。
不止于坐标，一般别名都可以作为定位对象。

- 距离测算
  默认状态下的偏移距离为边界之间的距离，而不是中心距。

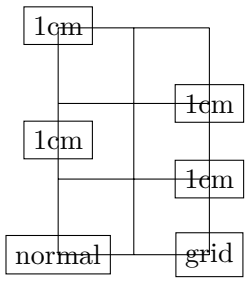



```
\begin{tikzpicture}[every node/.style = draw]
    \draw (0,0) grid (2,2);
    \node (point1) at (1,1) {point1};
    \node (point2) [above = 1cm of point1] {point2};
    \node (point3) [below = 1cm of point1.center] {point3};
\end{tikzpicture}
```

图 2.6.35 Node: 距离测算

发现没什么好办法获得中心距。

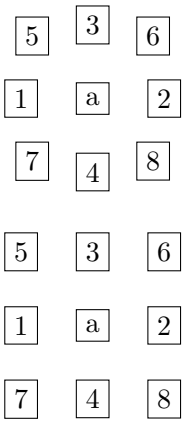
- 网格距
  网格距使用 on grid 将全部距离都定位在网格上，以获得中心距。




```
\begin{tikzpicture}[every node/.style = draw]
    \draw (0,0) grid (2,3);
    \node (a1) at (0,0) {normal};
    \node (a2) [above = 1 of a1] {1cm};
    \node (a3) [above = 1 of a2] {1cm};
    \begin{scope}[on grid]
        \node (b1) at (2,0) {grid};
        \node (b2) [above = 1 of b1] {1cm};
        \node (b3) [above = 1 of b2] {1cm};
    \end{scope}
\end{tikzpicture}
```

图 2.6.36 Node:on gird

- and 组合形式
  上文已说过 and 可以添加多个修饰，这里来看一下用不用 and 的区别。

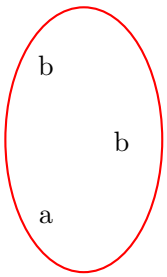



```
\begin{tikzpicture}[every node/.style = draw]
    \begin{scope}[node distance = 5mm]
        \node (a) at (2,3) {a};
        \node [left=of a] {1}; \node [right=of a] {2};
        \node [above=of a] {3}; \node [below=of a] {4};
        \node [above left=of a] {5}; \node [above right=of a] {6};
        \node [below left=of a] {7}; \node [below right=of a] {8};
    \end{scope}
    \begin{scope}[node distance = 5mm and 5mm]
        \node (a) at (2,0) {a};
        \node [left=of a] {1}; \node [right=of a] {2};
        \node [above=of a] {3}; \node [below=of a] {4};
        \node [above left=of a] {5}; \node [above right=of a] {6};
        \node [below left=of a] {7}; \node [below right=of a] {8};
    \end{scope}
\end{tikzpicture}
```

图 2.6.37 Node:and 详解

6.3.3 节点集

使用 fit 修饰可以将指定的节点圈起来。需要使用 fit 包




```
\begin{tikzpicture}
   \node (a) at (0,0) {a};
   \node (b) at (1,1) {b};
   \node (c) at (0,2) {b};
   \node[draw=red,inner sep=0pt,thick,ellipse,fit=(a) (b) (c)]
        {};
\end{tikzpicture}
```

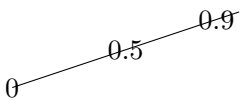
图 2.6.38 Node:fit

6.4 节点交互

6.4.1 曲线上的节点

节点交互指节点与其他 TikZ 图形之间的组合，例如在线段上某一位置插入节点。

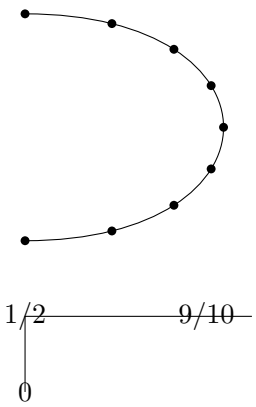
- 节点插入位置
  插入位置：[pos = <fraction>]，按线段长度比插入到对应的位置。




```
\begin{tikzpicture}[scale = 1]
   \draw (0,0) -- (3,1) node [pos = 0] {0} node [pos = 0.5]
        {0.5} node [pos = 0.9] {0.9};
\end{tikzpicture}
```

图 2.6.39 Node: 节点插入位置

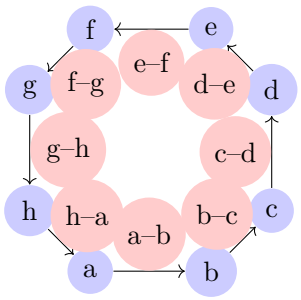
值得注意的是，pos 在曲线中代表的并不是长度比，pos 的具体计算方式比较复杂在这里不做说明。




```
\begin{tikzpicture}[scale = 1]
   \draw (0,0) .. controls +(right:3.5cm) and + (right:3.5cm)
        .. (0,3) node foreach \x in {0,0.125,...,1} [pos = \x,
        scale =3,radius = 1pt] {.};
   \draw (0,-2) |- (3,-1) node[pos=0]{0} node[pos=0.5]{1/2}
        node[pos=0.9]{9/10};
\end{tikzpicture}
```

图 2.6.40 Node:pos 位置

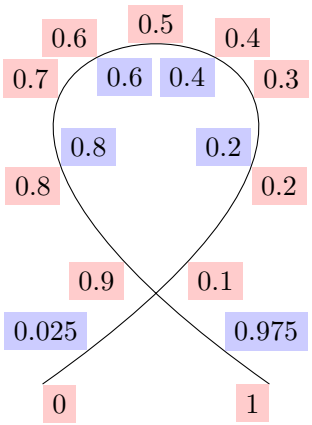
- 节点自动位置
  自动位置：[auto = <direction>] 这个修饰将节点自动生成在线段的某一方位，如果设置了 auto 的值，则为 auto 方向，如果没有设置，则为最近一个设置的方向，如果设置了 none，则禁用 auto。其具体的规则请查阅 TikZ 官方文档。




```
\begin{tikzpicture}[scale=.8,auto=left,every node/.style={
    circle,fill=blue!20}]
  \node (a) at (-1,-2) {a};
  \node (b) at ( 1,-2) {b};
  \node (c) at ( 2,-1) {c};
  \node (d) at ( 2, 1) {d};
  \node (e) at ( 1, 2) {e};
  \node (f) at (-1, 2) {f};
  \node (g) at (-2, 1) {g};
  \node (h) at (-2,-1) {h};
  \foreach \from/\to in {a/b,b/c,c/d,d/e,e/f,f/g,g/h,h/a}
  \draw [->] (\from) -- (\to)
  node[midway,fill=red!20] {\from--\to};
\end{tikzpicture}
```

图 2.6.41 Node:auto

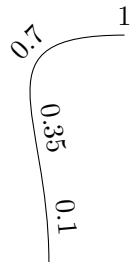
- 方位交换
  交换：[swap]：与原方向相反，常常配合 auto 使用。需要用的 automate 包。swap 可以用’ 来代替。




```
\begin{tikzpicture}[auto]
   \draw (0.5,0) .. controls (9,6) and (-5,6) .. (3.5,0) node foreach \pos in
        {0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1} [pos=\pos,swap,fill=red!20] {\
        pos} node foreach \pos in {0.025,0.2,0.4,0.6,0.8,0.975} [pos=\pos,fill=
        blue!20] {\pos};
\end{tikzpicture}
```

图 2.6.42 Node:swap

- 倾斜
  倾斜：[sloped]：倾斜可以让文本和曲线保持统一方向

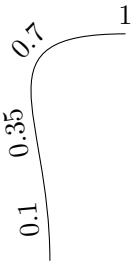



```
\begin{tikzpicture}[scale = 1]
   \draw (0,0) .. controls +(up:2cm) and + (left:2cm
        ) .. (1,3) node foreach \p in
        {0.1,0.35,0.7,1} [pos = \p,sloped,above] {\p
        };
\end{tikzpicture}
```

图 2.6.43 Node:sloped

- 颠倒

  颠倒：[allow upside down]：强制节点按指定方向生成，这可能对导致文字出现奇怪的朝向。

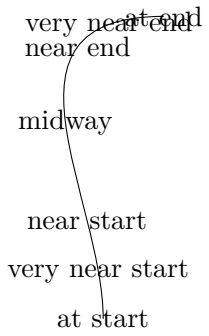



```
\begin{tikzpicture}[scale = 1]
   \draw (0,0) .. controls +(up:2cm) and + (left:2cm
        ) .. (1,3) node foreach \p in
        {0.1,0.35,0.7,1} [pos = \p,sloped,above,
        allow upside down] {\p};
\end{tikzpicture}
```

图 2.6.44 Node:sloped

- 预定义位置

  TikZ 预定义了几个 pos 对应的位置，可以直接使用其名称。




```
\begin{tikzpicture}[scale = 0.8]
   \draw (0,0) .. controls +(up:2cm) and +(left:3cm) .. (1,5)
      node[at end] {at end}
      node[very near end] {very near end}
      node[near end] {near end}
      node[midway] {midway}
      node[near start] {near start}
      node[very near start] {very near start}
      node[at start] {at start};
\end{tikzpicture}
```

图 2.6.45 Node: 预定义位置

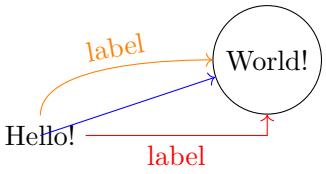
预定义位置具体位置如下表：

表 2.4 Node: 预定义位置

位置	对应 pos 值	位置	对应 pos 值
at start	0	very near start	0.125
near start	0.25	midway	0.5
near end	0.75	very near end	0.875
at end	1		

6.4.2 节点之间交互

节点之间除了默认的连线位置外，还可以使用 node.<anchor> 指定连接的位置。此外，与曲线类似，节点之间的连线也有多种样式。




```
\begin{tikzpicture}[scale = 1]
   \path (0,0) node (x) {Hello!}
        (3,1) node[circle,draw] (y) {World!};
   \draw [->,blue] (x.center) -- (y);
   \draw [->,red] (x) -| node[near start,below] {label} (y);
   \draw [->,orange] (x) .. controls +(up:1cm) and +(left:1cm)
         .. node[above,sloped] {label} (y);
\end{tikzpicture}
```

图 2.6.46 Node: 节点之间连线

6.4.3 节点图像

这里说明一些节点与图像之间的操作，部分操作只说明，但不写例子 (写例子将改变全局图像设置), 这些操作不是很常用，而且并不是所有的引擎都支持。

- 记录图像

```
\tikzset{every picture/.append style={remember picture}}
```

记录图像：[remember picture]：记录当前页所有图像位置等信息，这将在 aux 文件中增加对图像信息记录的数据，并不是所有引擎均支持，如果支持，需要编译两次。

- 覆盖

覆盖：[overlay = <boolean>]
启用此项，将可以使用其他图像中的节点，前提是对应的图像需要启用 remember picture ，因此也需要编译两次

```
\tikz[remember picture] \node [circle,fill=red!50] (n1) {};
\tikz[remember picture] \node [circle,fill=blue!50] (n2) {};
\tikz[remember picture,overlay] \draw[->] (n1)--(n2);
```

图 2.6.47 Node:overlay

- 当前页

TikZ 提供了一个特殊的节点 [current page]，这可以让我们在整个页面中进行节点操作。

```
\tikz[remember picture,overlay] \draw [line width=1mm,opacity = 0.25] (current page.center) circle (3cm);
```

图 2.6.48 Node:current page

6.5 节点拓展

节点拓展包含标签 (label)，大头针 (pin)，边缘 (edge)。他们都是基础写法的一种拓展，以不同的方式达到前文提到过的效果，可作为一种备选项。

6.5.1 节点标签

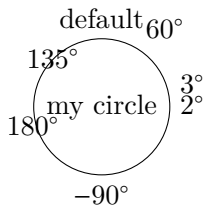
有时候我们需要在节点的周围添加一些文字信息，如果直接通过 添加文字会让节点变大，如果再画一个纯文本的节点，又会消耗许多时间，这个时候就可以使用标签功能。标签的许多修饰与节点一样，不做过多解释。

- 标签
  标签：[label = <text>]：给予文本标签，默认在上方生成


```
\begin{tikzpicture}[circle]
   \node [draw] (s) [label=$s$] {};
   \node [draw] (a) [right=of s] {} edge (s);
   \node [draw] (b) [right=of a] {} edge (a);
   \node [draw] (t) [right=of b, label=$t$] {} edge (b);
\end{tikzpicture}
```

图 2.6.49 Node:label

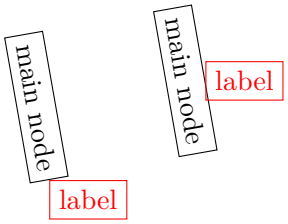
- 标签选项
  标签选项：label = {[<options>]<angle>:<text>}，区别于主节点 (main node)，这里将标签称为标签节点 (label node)，与节点类似的，标签节点也可以使用 every [] label/.style 指定默认样式，标签节点参数含义如下。
  - 位置：[position = <angle>] (默认：above)
    这里的 <angle> 与 <offset> 相同，但也可以设置具体的角度值。




```
\begin{tikzpicture}[scale = 1]
   \node [circle, draw,
      label=default,
      label=60:$60^\circ$,
      label=below:$-90^\circ$,
      label=3:$3^\circ$,
      label=2:$2^\circ$,
      label={[below]180:$180^\circ$},
      label={[centered]135:$135^\circ$}] {my circle};
\end{tikzpicture}
```

图 2.6.50 Node-Label:angle

  - 绝对位置：[absolute = <boolean>] (默认：true)
    若启用，采用全局坐标，否则，采用主节点的坐标。



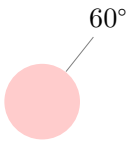

```
\begin{tikzpicture}[rotate=-80,every label/.style={draw,red}]
   \node [transform shape,rectangle,draw,label=right:label] at
         (0,0) {main node};
   \node [transform shape,rectangle,draw,label={[absolute]
        right:label}] at (0,2) {main node};
\end{tikzpicture}
```

图 2.6.51 Node-Label:absolute

6.5.2 大头针

大头阵 pin = [options]<angle>:<text>：pin 和 label 用法几乎一致，唯一的区别是 pin 会带上与主节点之间的连线。下面只对连线进行说明，其他参考前文。

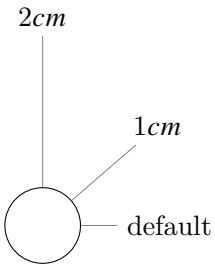
- 基本用法




```
\begin{tikzpicture}[scale = 1]
   \node [circle,fill=red!20,minimum size = 1cm,pin = 60:$60^\
        circ$] {};
\end{tikzpicture}
```

图 2.6.52 Node-Pin: 基础用法

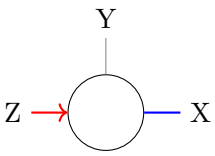
- 距离
  大头阵节点距：[pin distance = <distance>]




```
\begin{tikzpicture}[scale = 1]
   \node [circle,draw,minimum size = 1cm,
      pin = right:default,
      pin = {[pin distance = 1cm]above right:$1cm$},
      pin = {[pin distance = 2cm]above:$2cm$}] {};
\end{tikzpicture}
```

图 2.6.53 Node-Pin:distance

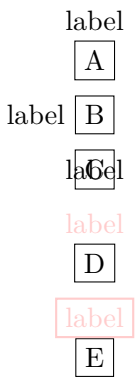
- 连线样式
  连线样式：[pin edge = <options>]




```
\begin{tikzpicture}[scale = 1]
   \node [circle,draw,minimum size = 1cm,
      pin={[pin edge={blue,thick}]right:X},
      pin={[pin edge={<-,red,thick}]left:Z},
      pin = above:Y] {};
\end{tikzpicture}
```

图 2.6.54

- 简化语法
  Label 和 Pin 的语法有点复杂，可以启用 quotes 包，简化语法。与原来的 [options]<angle>:<text> 对应，新的语法形式为”<text>” [options]。



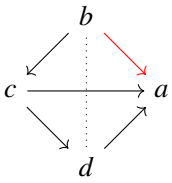

```
\begin{tikzpicture}[scale = 1]
   \matrix [row sep = 2mm] {
      \node [draw,"label"] {A}; \\
      \node [draw,"label" left] {B}; \\
      \node [draw,"label" centered] {C}; \\
      \node [draw,"label" color = red!20] {D}; \\
      \node [draw,"label" {red!20,draw,thick}] {E}; \\
   };
\end{tikzpicture}
```

图 2.6.55 Node-quotes: 基础用法

6.5.3 边缘

边缘 (edge) 是节点间连线的一种方案，与一般连线不同的是，边缘连线会在所有连线绘制完成后再绘制，同样可以使用 every edge/.style 控制全部边缘样式。

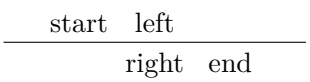
- 边缘基础用法
  边缘既可以在节点绘制过程中使用，也可以单独绘制。




```
\begin{tikzpicture}[scale = 1]
   \node (a) at (0:1) {$a$};
   \node (b) at (90:1) {$b$} edge [->] (a);
   \node (c) at (180:1) {$c$} edge [->] (a) edge [<-] (b);
   \node (d) at (270:1) {$d$} edge [->] (a) edge [dotted] (b)
        edge [<-] (c);
   % 这种写法效果一样
   \node foreach \name/\angle in {a/0,b/90,c/180,d/270}
      (\name) at (\angle:1) {$\name$};
   \path[->] (b) edge (a) edge (c) edge [-,dotted] (d)
          (c) edge (a) edge (d)
          (d) edge (a);
\end{tikzpicture}
```

图 2.6.56 Node-Edge: 基础用法

- 边缘在 quote 中的写法
  和其他拓展方法相同，边缘也可以利用 quote 包。




```
\draw (0,0) edge ["left", "right"', "start" near
    start, "end"' near end] (4,0);
```

图 2.6.57 Node:

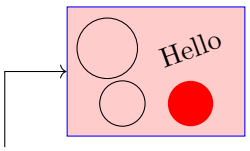
七、Matrix

7.1 矩阵基础

TikZ 中的矩阵类似于 LaTeX 中的表格，矩阵中各元素的控制则与 Node 十分相似，因为 Matrix 本质上就是 Node 的集合。

7.1.1 节点形式的矩阵

在 Node 中可以使用 matrix 选项，指定某个节点为矩阵。在内部可以使用 LaTeX 表格的语法。




```
\begin{tikzpicture}[scale = 1]
    \node [matrix,fill=red!20,draw=blue] (my matrix) at (2,1) {
        \draw (0,0) circle (4mm); & \node [rotate=20] {Hello};
            \\
        \draw (0.2,0) circle (3mm); & \fill[red] (0,0) circle (3
            mm); \\
    };
    \draw [->] (0,0) |- (my matrix.west);
\end{tikzpicture}
```

图 2.7.1 Matrix: 节点形式的矩阵

\every matrix 控制矩阵样式，这将同时控制矩阵内部图形的样式

\every outer matrix 仅控制矩阵框样式。

\matrix 是 \path node [matrix] 的简写。

尽管矩阵是由节点引申来的，但有些修饰无法使用：

- rotation 和 scale 对节点框无效。
- matrix 节点无法被拆分。
- 所有以 text- 开头的修饰均无效。
- matrix 在放置在路径上时，会出现许多错误。

7.1.2 矩阵位置

可以使用 matrix anchor = <anchor> 调整矩阵位置，注意 anchor = <anchor> 调整的是矩阵内单元格的对齐方式

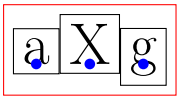
123 123
12 12
1 1

```
\begin{tikzpicture}[scale = 1]
    \matrix [matrix anchor=west] at (0,0) {
        \node {123}; \\ % still center anchor
        \node {12}; \\
        \node {1}; \\
    };
    \matrix [anchor=west] at (0,-2) {
        ...
    };
\end{tikzpicture}
```

图 2.7.2 Matrix: 矩阵位置

7.1.3 单元格

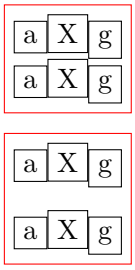
矩阵内部的每个元素均可视为一个单元格，使用 \\划分行，使用 & 划分列。矩阵每行/列的宽度与高度均取决于数值最大的单元格。




```
\begin{tikzpicture}[every node/.style = {draw = black,anchor =
    base,font = \huge}]
  \matrix [draw = red]{
    \node {a}; \fill [blue] (0,0) circle (2pt); &
    \node {X}; \fill [blue] (0,0) circle (2pt); &
    \node {g}; \fill [blue] (0,0) circle (2pt); \\
  };
\end{tikzpicture}
```

图 2.7.3 Matrix: 单元格

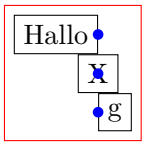
Matrix 中每行的高度与深度均由行本身决定，默认情况下，行之间间隙为 0, 可以使用 row sep 指定间隙。




```
\begin{tikzpicture}[every node/.style = {draw = black,anchor =
    base}]
  \matrix [draw = red]{
    \node {a}; & \node {X}; & \node {g}; \\
    \node {a}; & \node {X}; & \node {g}; \\
  };
  \matrix [draw = red,row sep = 3mm] at (0,-2){
    \node {a}; & \node {X}; & \node {g}; \\
    \node {a}; & \node {X}; & \node {g}; \\
  };
\end{tikzpicture}
```

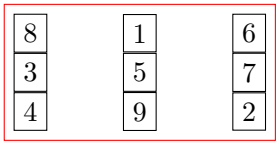
图 2.7.4 Matrix: 行间隙

矩阵的列与行一样，列宽取决于每列的最大宽度，默认列之间间隙为 0，可以通过 column sep 调整列宽。




```
\begin{tikzpicture}[every node/.style={draw}]
  \matrix [draw=red]{
    \node[left] {Hallo}; \fill[blue] (0,0) circle (2pt); \\
    \node {X};           \fill[blue] (0,0) circle (2pt); \\
    \node[right] {g};    \fill[blue] (0,0) circle (2pt); \\
  };
\end{tikzpicture}
```

图 2.7.5 Matrix: 列宽




```
\begin{tikzpicture}[every node/.style={draw}]
  \matrix [draw=red,column sep=1cm]{
    \node {8}; & \node{1}; & \node {6}; \\
    \node {3}; & \node{5}; & \node {7}; \\
    \node {4}; & \node{9}; & \node {2}; \\
  };
\end{tikzpicture}
```

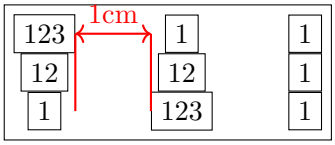
图 2.7.6 Matrix: 列间隙

7.2 矩阵样式

7.2.1 矩阵间距

控制矩阵间隔有两种方式：1. 通过 **row sep** 或 **column sep** 控制所有行列的宽度。2. 通过 \\或 & 精确控制某一行/列。

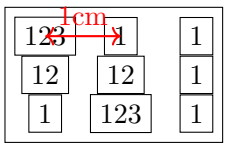
- 列间距
  列间距：[column sep = <spacing list>]：用来控制默认列间距，值可以是正数或负数。
  这里的 <spacing list> 类似于 <dimension>




```
\begin{tikzpicture}
   \matrix [draw,column sep = 1cm,nodes = draw]{
      \node(a) {123}; & \node (b) {1}; & \node {1}; \\
      \node {12};   & \node {12};    & \node {1}; \\
      \node(c) {1}; & \node (d) {123}; & \node {1}; \\
   };
   \draw [red,thick] (a.east) -- (a.east |- c)
                     (d.west) -- (d.west |- b);
   \draw [<->,red,thick] (a.east) -- (d.west |- b)
                     node [above,midway] {1cm};
\end{tikzpicture}
```

图 2.7.7 Matrix:column sep

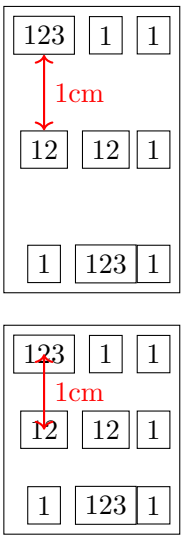
间距默认为单元格的边界，我们也可以指定从单元格原点开始计算。




```
\begin{tikzpicture}
   \matrix [draw,column sep={1cm,between origins},nodes=draw]{
      \node(a) {123}; & \node (b) {1}; & \node {1}; \\
      \node {12}; & \node {12}; & \node {1}; \\
      \node {1}; & \node {123}; & \node {1}; \\
   };
   \draw [<->,red,thick] (a.center) -- (b.center) node [above,
        midway] {1cm};
\end{tikzpicture}
```

图 2.7.8 Matrix:origin 的 column sep

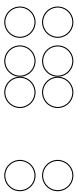
- 行间距
  行间距：[row sep = <spacing list>]：控制各行之间间距，用法和列间距相同




```
\begin{tikzpicture}[scale = 1]
    \matrix [draw,row sep=1cm,nodes=draw]{
        \node (a) {123}; & \node {1}; & \node {1}; \\
        \node (b) {12}; & \node {12}; & \node {1}; \\
        \node {1}; & \node {123}; & \node {1}; \\
    };
    \draw [<->,red,thick] (a.south) -- (b.north) node [right,
        midway] {1cm};
    \matrix [draw,row sep={1cm,between origins},nodes=draw] at
        (0,-10em){
        \node (a) {123}; & \node {1}; & \node {1}; \\
        \node (b) {12}; & \node {12}; & \node {1}; \\
        \node {1}; & \node {123}; & \node {1}; \\
    };
    \draw [<->,red,thick] (a.center) -- (b.center) node [right,
        midway] {1cm};
\end{tikzpicture}
```

图 2.7.9 Matrix:row sep

- 精确间距
  上述的控制方法只能设置整体的默认间距，精确到某行/列则需要在分隔符中设定距离。



```
\begin{tikzpicture}[scale = 1]
    \matrix [row sep=1mm]{
        \draw (0,0) circle (2mm); &[0.5cm,between origins] \draw
            (0,0) circle (2mm); \\
        \draw (0,0) circle (2mm); & \draw (0,0) circle (2mm);
            \\[-1mm]
        \draw (0,0) coordinate (a) circle (2mm); &
        \draw (0,0) circle (2mm); \\[1cm,between origins]
        \draw (0,0) coordinate (b) circle (2mm); &
        \draw (0,0) circle (2mm); \\
    };
\end{tikzpicture}
```

图 2.7.10 Matrix:

7.2.2 单元格样式

矩阵往往包含多个单元格，逐一控制往往不太现实，TikZ 提供了许多批量控制的修饰。

- every cell = <row><column>: 设置矩阵行列数量
- cells = <options>: 设置每个单元格的样式，等效于 every cell/.append style = <options>
- nodes = <options>: 设置每个节点的样式

8	1	6
3	5	7
4	9	2

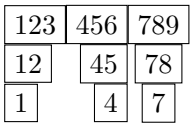
```
\begin{tikzpicture}[scale = 1]
    \matrix [nodes={fill=blue!20,minimum size=5mm}]{
        \node {8}; & \node{1}; & \node {6}; \\
        \node {3}; & \node{5}; & \node {7}; \\
        \node {4}; & \node{9}; & \node {2}; \\
    };
\end{tikzpicture}
```

图 2.7.11 Matrix: 节点样式

- column <number>: 设置 <number> 列的样式
- every odd column: 设置奇数列样式
- every even column: 设置偶数列样式
- row <number>: 设置 <number> 行的样式
- every odd row: 设置奇数行样式
- every even row: 设置偶数行样式
- row <number> column <number>: 设置对应行列下某个单元格的样式


```
\begin{tikzpicture} [row 1/.style={red}, column 2/.style={green
     !50!black}, row 3 column 3/.style={blue}]
   \matrix{
      \node {8}; & \node{1}; & \node {6}; \\
      \node {3}; & \node{5}; & \node {7}; \\
      \node {4}; & \node{9}; & \node {2}; \\
   };
\end{tikzpicture}
```

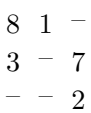
图 2.7.12 Matrix: 单元格样式




```
\begin{tikzpicture} [column 1/.style={anchor=base west}, column
      2/.style={anchor=base east}, column 3/.style={anchor=base
     }]
   \matrix[nodes = draw]{
      \node {123}; & \node{456}; & \node {789}; \\
      \node {12}; & \node{45}; & \node {78}; \\
      \node {1}; & \node{4}; & \node {7}; \\
   };
\end{tikzpicture}
```

图 2.7.13 Matrix: 单元格对齐

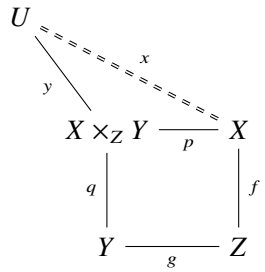
- execute at begin cell = <code>: 在每行非空的第一个单元格执行
- execute at end cell = <code>: 在每行非空的最后一个单元格执行
- execute at empty cell = <code>: 在每行空的单元格执行




```
\begin{tikzpicture}[matrix of nodes/.style={
   execute at begin cell=\node\bgroup,
   execute at end cell=\egroup;,%
   execute at empty cell=\node{--};%
   }]
   \matrix [matrix of nodes]{
      8 & 1 & \\
      3 & & 7 \\
      & & 2 \\
   };
\end{tikzpicture}
```

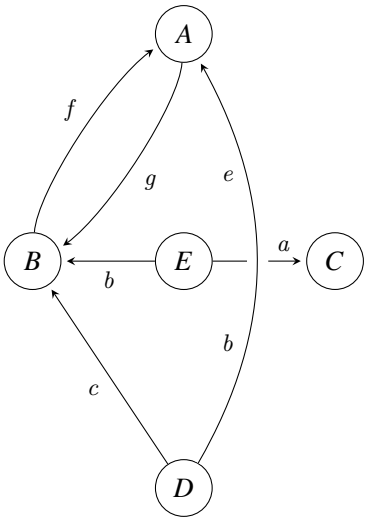
图 2.7.14

7.3 例子



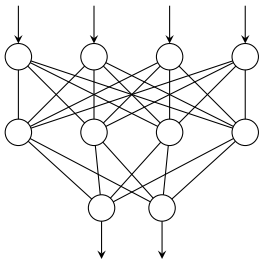

```
\begin{tikzpicture}[scale = 1]
   \matrix [matrix of math nodes,row sep=1cm] {
      |(U)| U &[2mm]              &[8mm] \\
             & |(XZY)| X \times_Z Y & |(X)| X \\
             & |(Y)| Y             & |(Z)| Z \\
   };
   \begin{scope}[every node/.style={midway,auto,font=\
        scriptsize}]
      \draw [double, dashed] (U) -- node {$x$} (X);
      \draw  (X) -- node {$p$} (X -| XZY.east)
             (X) -- node {$f$} (Z)
             -- node {$g$} (Y)
             -- node {$q$} (XZY)
             -- node {$y$} (U);
   \end{scope}
\end{tikzpicture}
```

图 2.7.15　Matrix: 例子 1




```
\begin{tikzpicture}[>=stealth,->,shorten >=2pt,looseness=.5,
     auto]
   \matrix [matrix of math nodes,column sep={2cm,between
        origins},
      row sep={3cm,between origins},nodes={circle, draw,
           minimum size=7.5mm}]
   {
             & |(A)| A &      \\
      |(B)| B & |(E)| E & |(C)| C \\
             & |(D)| D        \\
   };
   \begin{scope}[every node/.style={font=\small\itshape}]
      \draw (A) to [bend left] node [midway] {g} (B);
      \draw (B) to [bend left] node [midway] {f} (A);
      \draw (D) -- node [midway] {c} (B);
      \draw (E) -- node [midway] {b} (B);
      \draw (E) -- node [near end] {a} (C);
      \draw [-,line width=8pt,draw=white]
         (D) to [bend right, looseness=1] (A);
      \draw (D) to [bend right, looseness=1]
         node [near start] {b} node [near end] {e} (A);
   \end{scope}
\end{tikzpicture}
```

图 2.7.16　Matrix: 例子 2

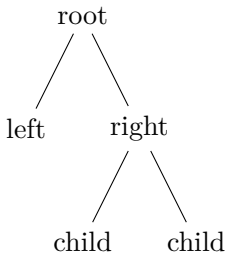


```
\begin{tikzpicture}
   \matrix (network) [matrix of nodes,%
      nodes in empty cells,
      nodes={outer sep=0pt,circle,minimum size=4pt,draw},
      column sep={1cm,between origins},
      row sep={1cm,between origins}]
   {
                     &              &                & \\
                     &              &                & \\
      |[draw=none]| & |[xshift=1mm]| & |[xshift=-1mm]| \\
   };
   \foreach \a in {1,...,4}{
      \draw (network-3-2) -- (network-2-\a);
      \draw (network-3-3) -- (network-2-\a);
      \draw [-stealth] ([yshift=5mm]network-1-\a.north) -- (
           network-1-\a);
      \foreach \b in {1,...,4}
         \draw (network-1-\a) -- (network-2-\b);
   }
   \draw [stealth-] ([yshift=-5mm]network-3-2.south) -- (
        network-3-2);
   \draw [stealth-] ([yshift=-5mm]network-3-3.south) -- (
        network-3-3);
\end{tikzpicture}
```

图 2.7.17 Matrix: 例子 3

八、Tree

一个基本的 Tree 如下图所示:




```
\begin{tikzpicture}[scale = 1]
   \node {root}
      child {node {left}}
      child {node {right}
         child {node {child}}
         child {node {child}}
   };
\end{tikzpicture}
```

图 2.8.1 TikZ 基础树

- child 关键字

  树的根节点下跟着 child 关键字即可构建树，其中 child 的语法如下:

```
\path ⋯ child [<options>] foreach <variables> in {<values>} {<child path>} ⋯;
```

其中，`foreach` 用于创建当前节点下的多个子结点。下面两种写法效果是相同的

```
% 没有样式的 foreach
node {root} child [red] foreach \name in {1,2} {node {\name}}
node {root} child [red] {node {1}} child[red] {node {2}}
% 带样式的 foreach
node {root} child[\pos] foreach \name/\pos in {1/left,2/right} {node[\pos] {\name}}
node {root} child[left] {node[left] {1}} child[right] {node[right] {2}}
```

图 2.8.2 child 节点的 foreach

III 百科

一、参数百科

参数[1]在这里指 TikZ 中一些可选项 (option) 对应的通用值。

1.1 通用参数

通用参数，即常见的参数，其参数名和值是大部分同类语言都具备的，只需理解参数意思就可明白如果写值。

1.1.1 常见通用参数意义

表 3.1 常见通用参数意义

参数	值	意义	举例
dimension	数字	一般为长度，可自定义单位	line width = <dimension>
angle	数字	旋转角度，单位为度	rotate = <angle>
scaling	数字	缩放比例	scale = <scaling >

1.2 TikZ 特有参数

特有参数，即参数和对应的值是 TikZ 所特定的，TikZ 专有这些值，参数的值往往需要查表得知。

1.2.1 对齐：alignment option

表 3.2 参数：alignment option

值	意义	值	意义
left	左对齐	flush left	左对齐 (禁止拆分单词)
right	右对齐	flush right	右对齐 (禁止拆分单词)
center	居中对齐	flush center	居中对齐 (禁止拆分单词)
justify	拉长	none	禁止之前的对齐参数

[1]这里借用了 css 的概念

1.2.2 锚点：anchor name

锚点主要用在定位与文本对齐上，锚点的值需要视具体的修饰选择。

表 3.3 锚点:anchor name

类型	值	意义	值	意义
对齐	center	文字中心对齐[1]	base	文字底部对齐[2]
	mid	文本中心对齐[3]		
位置	north	北	south	南
	east	东	west	西
	north-east	东北	south-east	东南
	north-west	西北	south-west	西南
组合	base west	采用 base 对齐的西方	base east	采用 base 对齐的东方
	mid west	采用 mid 对齐的西方	mid east	采用 mid 对齐的东方

[1] 单个文字的中心。
[2] 文字底边对齐，英文四线三格中第三条线对齐。
[3] 文本整体的中心。

1.2.3 偏移：offset

表 3.4 偏移:offset

值	意义	值	意义
above/below	上/下	left/right	左/右
above left/right	上左/右	below left/right	下左/右
centered	中心		

二、名称百科

2.1 坐标系名称

以下为坐标系对应前缀名：

表 3.5 坐标系前缀名

坐标系	前缀名	坐标系	前缀名
平面直角坐标系	canvas cs	空间坐标系	xyz cs
极坐标系	canvas polar	平面直角坐标系衍生	xyz polar
重心坐标系	barycentric cs	节点坐标系	node cs
切线坐标系	tangent cs		