D3.js – Path

张松海、张少魁、周文洋、蔡韵

数据可视化 – D3.js

清华大学 可视媒体研究中心

Previously…

- 函数式编程很重要、函数式编程很重要、函数式编程很重要
  - 重要的事情说三遍
  - 要习惯与把函数作为参数输入给另一个函数
- 以下函数的定义等价:
  - const myFunction = datum => datum.value
  - const myFunction = (datum) => { return datum.value }
  - const myFunction = function(datum){return datum.value}
  - const myFunction = function(d){return d.value}
  - const myFunction = d => d.value

Why Path?

- path元素是SVG基本形状中最强大的一个，它不仅能创建其他基本形状，还能创建更多其他形状。你可以用path元素绘制矩形（直角矩形或者圆角矩形）、圆形、椭圆、折线形、多边形，以及一些其他的形状，例如贝塞尔曲线、2次曲线等曲线。
- path元素的形状是通过属性d来定义的，属性d的值是一个“命令+参数”的序列（见下页）
- Path作为SVG提供的标签之一，是实现众多可视化方案的基础
- Path可以做什么？
  - **折线图**（即将到来）
  - 地图（下期预告）
  - 主题河流（待定）
  - ……

Why Path?

- 每一个命令都用一个关键字母来表示，比如，字母“M”表示的是“Move to”命令，当解析器读到这个命令时，它就知道你是打算移动到某个点。跟在命令字母后面的，是你需要移动到的那个点的x和y轴坐标。比如移动到(10,10)这个点的命令，应该写成“M 10 10”。这一段字符结束后，解析器就会去读下一段命令。每一个命令都有两种表示方式，一种是用大写字母，表示采用绝对定位。另一种是用小写字母，表示采用相对定位。
- 因为属性d采用的是用户坐标系统，所以不需标明单位。

Path

- 属性：
  - d
  - fill：填充颜色
  - stroke：描边颜色
  - stroke-width：描边宽度
  - transform="translate(x,y)": 加了描边后需要平移（x=stroke-width/2, y=stroke-width/2）

Path

- ‘d’属性：
  - M = moveto(M X,Y)： 将画笔移动到指定的坐标位置
  - L = lineto(L X,Y)： 画直线到指定的坐标位置
  - H = horizontal lineto(H X)：画水平线到指定的X坐标位置
  - V = vertical lineto(V Y)：画垂直线到指定的Y坐标位置
  - C = curveto(C X1,Y1,X2,Y2,ENDX,ENDY)：三次贝赛曲线
  - S = smooth curveto(S X2,Y2,ENDX,ENDY)：平滑曲率
  - Q = quadratic Belzier curve(Q X,Y,ENDX,ENDY)：二次贝赛曲线
  - T = smooth quadratic Belzier curveto(T ENDX,ENDY)：映射
  - A = elliptical Arc(A RX,RY,XROTATION,FLAG1,FLAG2,X,Y)：弧线
  - Z = closepath()：关闭路径
- 以上所有命令均允许小写字母。大写表示绝对定位，小写表示相对定位。

直线命令

- 顾名思义，直线命令就是在两个点之间画直线。首先是“Move to”命令，M，需要两个参数，分别是需要移动到点的x轴和y轴的坐标。在使用M命令移动画笔后，只会移动画笔，但不会在两点之间画线。所以M命令经常出现在路径的开始处，用来指明从何处开始画。
- M 移动到的点的x轴和y轴的坐标
  L 需要两个参数，分别是一个点的x轴和y轴坐标，L命令将会在当前位置和新位置（L前面画笔所在的点）之间画一条线段。
  H 绘制平行线
  V 绘制垂直线
  Z 从当前点画一条直线到路径的起点

直线命令示例

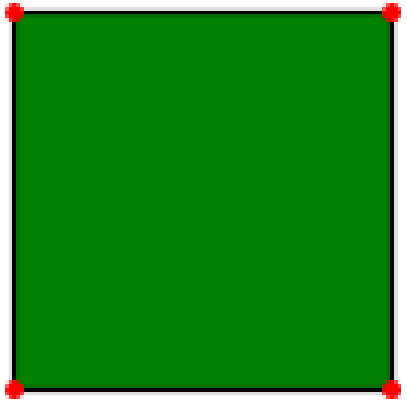
```
<svg width="100" height="100">
    <path d="M10 10 H 90 V 90 H 10 L 10 10" />
    <!-- Points -->
    <circle cx="10" cy="10" r="2" fill="red"/>
    <circle cx="90" cy="90" r="2" fill="red"/>
    <circle cx="90" cy="10" r="2" fill="red"/>
    <circle cx="10" cy="90" r="2" fill="red"/>
</svg>
```

Code: https://github.com/Shao-Kui/D3.js-Demos/blob/master/static/html-tutorial/hello-path.html
（之后的Path示例也均在此链接中）

直线命令示例

- 可以通过一个“闭合路径命令”Z来简化上面的path，简写形式：

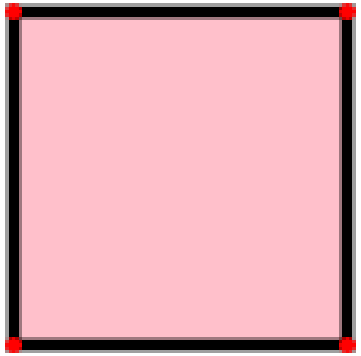
```
<svg width="100px" height="100px" fill="green">
    <path d="M10 10 H 90 V 90 H 10 Z" fill="green" stroke="black"/>
    <!-- Points -->
    <circle cx="10" cy="10" r="2" fill="red"/>
    <circle cx="90" cy="90" r="2" fill="red"/>
    <circle cx="90" cy="10" r="2" fill="red"/>
    <circle cx="10" cy="90" r="2" fill="red"/>
</svg>
```



直线命令示例

- 相对命令使用的是小写字母，它们的参数不是指定一个明确的坐标，而是表示相对于它前面的点需要移动多少距离。相对坐标形式：

```
<svg width="100px" height="100px">
    <path d="M10 10 h 80 v 80 h -80 Z" fill="pink" stroke="black" stroke-width="3"/>
    <!-- Points -->
    <circle cx="10" cy="10" r="2" fill="red"/>
    <circle cx="90" cy="90" r="2" fill="red"/>
    <circle cx="90" cy="10" r="2" fill="red"/>
    <circle cx="10" cy="90" r="2" fill="red"/>
</svg>
```



曲线命令

- 绘制平滑曲线的命令有三个，其中两个用来绘制贝塞尔曲线，另外一个用来绘制弧形或者说是圆的一部分。
- 在path元素里，只存在两种贝塞尔曲线：三次贝塞尔曲线C，和二次贝塞尔曲线Q。

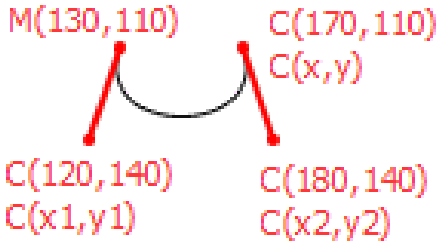
三次贝塞尔曲线

- 三次贝塞尔曲线需要定义一个点和两个控制点，所以用C命令创建三次贝塞尔曲线，需要设置三组坐标参数：C x1 y1, x2 y2, x y (or c dx1 dy1, dx2 dy2, dx dy)，最后一个坐标(x,y)表示的是曲线的终点，另外两个坐标是控制点，(x1,y1)是起点的控制点，(x2,y2)是终点的控制点。

三次贝塞尔曲线示例

- 原理分析：曲线沿着起点到第一控制点的方向伸出，逐渐弯曲，然后沿着第二控制点到终点的方向结束。

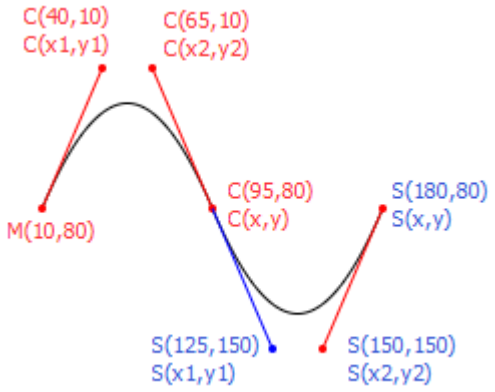
```svg
<!--三次贝塞尔曲线-->
<svg width="190px" height="160px">
    <path d="M130 110 C 120 140, 180 140, 170 110" stroke="black" fill="transparent"/>
    <circle cx="130" cy="110" r="2" fill="red"/>
    <circle cx="120" cy="140" r="2" fill="red"/>
    <line x1="130" y1="110" x2="120" y2="140" style="stroke:rgb(255,0,0);stroke-width:2"/>
    <circle cx="180" cy="140" r="2" fill="red"/>
    <circle cx="170" cy="110" r="2" fill="red"/>
    <line x1="180" y1="140" x2="170" y2="110" style="stroke:rgb(255,0,0);stroke-width:2"/>
</svg>
```



简写的贝塞尔曲线命令S

- 一个点某一侧的控制点是它另一侧的控制点的对称（以保持斜率不变）。可以使用一个简写的贝塞尔曲线命令S：S x2 y2, x y (or s dx2 dy2, dx dy)，S命令可以用来创建与之前那些曲线一样的贝塞尔曲线，但是，如果S命令跟在一个C命令或者另一个S命令的后面，它的第一个控制点，就会被假设成前一个控制点的对称点。如果S命令单独使用，前面没有C命令或者另一个S命令，那么它的两个控制点就会被假设为同一个点。

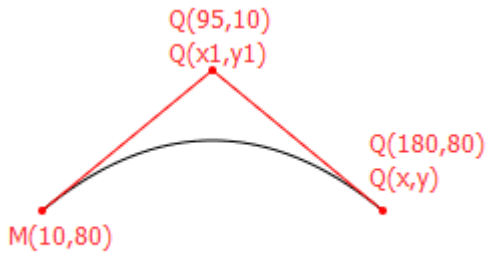
```
<!--三次贝塞尔曲线简写-->
<svg width="190px" height="160px">
    <path d="M10 80 C 40 10, 65 10, 95 80 S 150 150, 180 80" stroke="black" fill="transparent"/>
    <circle cx="10" cy="80" r="2" fill="red"/>
    <circle cx="40" cy="10" r="2" fill="red"/>
    <line x1="10" y1="80" x2="40" y2="10" style="stroke:rgb(255,0,0);stroke-width:1"/>

    <circle cx="65" cy="10" r="2" fill="red"/>
    <circle cx="95" cy="80" r="2" fill="red"/>
    <line x1="65" y1="10" x2="95" y2="80" style="stroke:rgb(255,0,0);stroke-width:1"/>

    <circle cx="125" cy="150" r="2" fill="blue"/>
    <circle cx="180" cy="80" r="2" fill="red"/>
    <circle cx="150" cy="150" r="2" fill="red"/>
    <line x1="95" y1="80" x2="125" y2="150" style="stroke:blue;stroke-width:1"/>
    <line x1="180" y1="80" x2="150" y2="150" style="stroke:rgb(255,0,0);stroke-width:1"/>
</svg>
```



二次贝塞尔曲线

- 二次贝塞尔曲线Q比三次贝塞尔曲线简单，只需要一个控制点，用来确定起点和终点的曲线斜率。需要两组参数，控制点和终点坐标。Q命令：Q x1 y1, x y (or q dx1 dy1, dx dy)

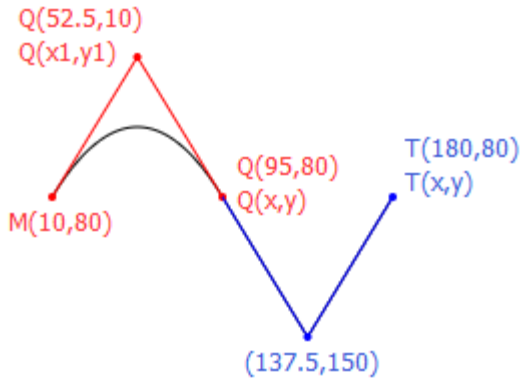
```
<!--二次贝塞尔曲线-->
<svg width="190px" height="160px">
    <path d="M10 80 Q 95 10 180 80" stroke="black" fill="transparent"/>
    <!--Points-->
    <circle cx="10" cy="80" r="2" fill="red"/>
    <circle cx="95" cy="10" r="2" fill="red"/>
    <circle cx="180" cy="80" r="2" fill="red"/>
    <line x1="10" y1="80" x2="95" y2="10" style="stroke:rgb(255,0,0);stroke-width:1"/>
    <line x1="95" y1="10" x2="180" y2="80" style="stroke:rgb(255,0,0);stroke-width:1"/>
</svg>
```



简写的贝塞尔曲线命令T

- 就像三次贝塞尔曲线有一个S命令，二次贝塞尔曲线有一个差不多的T命令，可以通过更简短的参数，延长二次贝塞尔曲线。T x y (or t dx dy)，快捷命令T会通过前一个控制点，推断出一个新的控制点。这意味着，在你的第一个控制点后面，可以只定义终点，就创建出一个相当复杂的曲线。需要注意的是，T命令前面必须是一个Q命令，或者是另一个T命令，才能达到这种效果。如果T单独使用，那么控制点就会被认为和终点是同一个点，所以画出来的将是一条直线。

```
<!--二次贝塞尔曲线简写-->
<svg width="190px" height="160px">
    <path d="M10 80 Q 52.5 10, 95 80 T  180 80" stroke="black" fill="transparent"/>

    <circle cx="10" cy="80" r="2" fill="red"/>
    <circle cx="52.5" cy="10" r="2" fill="red"/>
    <line x1="10" y1="80" x2="52.5" y2="10" style="stroke:rgb(255,0,0);stroke-width:1"/>

    <circle cx="95" cy="80" r="2" fill="red"/>
    <line x1="95" y1="80" x2="52.5" y2="10" style="stroke:rgb(255,0,0);stroke-width:1"/>

    <circle cx="180" cy="80" r="2" fill="blue"/>
    <circle cx="137.5" cy="150" r="2" fill="blue"/>
    <line x1="95" y1="80" x2="137.5" y2="150" style="stroke:rgb(0,0,255);stroke-width:1"/>
    <line x1="137.5" y1="150" x2="180" y2="80" style="stroke:rgb(0,0,255);stroke-width:1"/>
</svg>
```



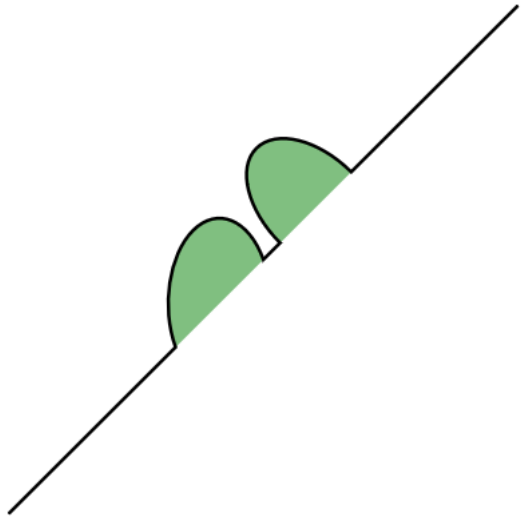
弧形

- A命令的参数:
  A rx ry x-axis-rotation large-arc-flag sweep-flag x y
  或者 a rx ry x-axis-rotation large-arc-flag sweep-flag dx dy
- 弧形命令A的前两个参数分别是x轴半径和y轴半径，弧形命令A的第三个参数表示弧形的旋转情况，large-arc-flag（角度大小） 和sweep-flag（弧线方向），large-arc-flag决定弧线是大于还是小于180度，0表示小角度弧，1表示大角度弧。sweep-flag表示弧线的方向，0表示从起点到终点沿逆时针画弧，1表示从起点到终点沿顺时针画弧。

弧形示例

- 如图例所示，画布上有一条对角线，中间有两个椭圆弧被对角线切开(x radius = 30, y radius = 50)。第一个椭圆弧的x-axis-rotation（x轴旋转角度）是0，所以弧形所在的椭圆是正置的（没有倾斜）。在第二个椭圆弧中，x-axis-rotation设置为-45，所以这是一个旋转了45度的椭圆，并以短轴为分割线，形成了两个对称的弧形。

```
<svg width="320px" height="320px">
    <path d="M10 315
             L 110 215
             A 30 50 0 0 1 162.55 162.45
             L 172.55 152.45
             A 30 50 -45 0 1 215.1 109.9
             L 315 10" stroke="black" fill="green" stroke-width="2" fill-opacity="0.5"/>
</svg>
```

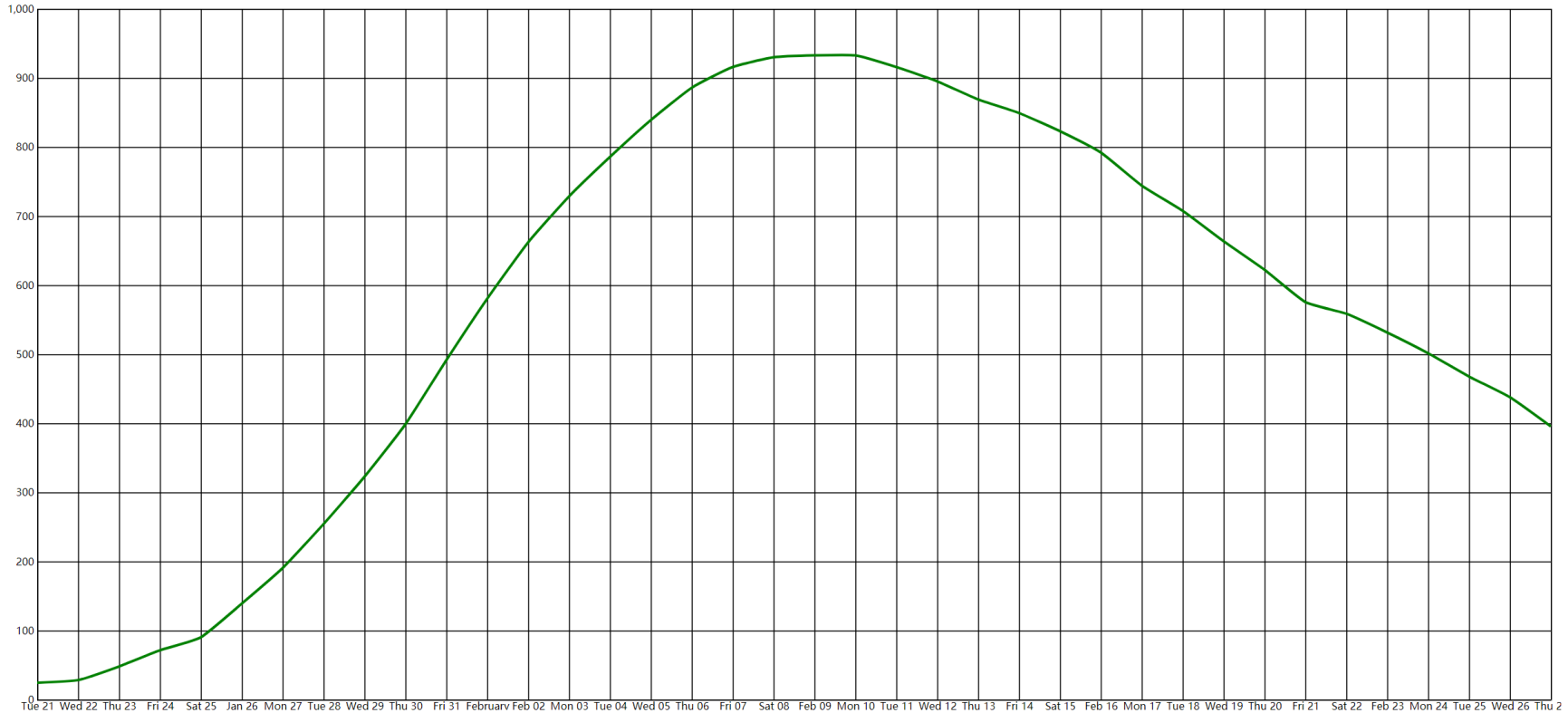


D3.js Path生成器

- d3.line(…).x(…).y(…)
  - 用于折线图
- d3.geoPath().projection()
  - 用于地图，下期预告
- d3.area()
  - 用于主题河流
- d3.arc(…).innerRadius(…).outerRadius(…)
  - 用于饼图
- d3.lineRadial().angle(…).radius(…)
  - 极坐标系版本的d3.line(…)
- D3-Shapes: https://github.com/d3/d3-shape/tree/v1.3.7

折线图

- Code: https://github.com/Shao-Kui/D3.js-Demos/blob/master/static/lineChart.html



折线图 – 数据预处理

- 根据每个省份重组数据
- 回忆散点（气泡）图的数据预处理

```
let provinces = {};
allkeys.forEach( key => {provinces[key] = []} );
data.forEach( d => { provinces[d['省份']].push(d) } )
allkeys.forEach( key => provinces[key].sort(function(a,b){
    return new Date(b.date) - new Date(a.date);
}));
```

折线图 – 坐标轴与比例尺（初始化）

- 回忆散点图（气泡图）的坐标轴与比例尺的设置
- .ticks(…): 设置需要多少个刻度
- 下方二者等价
  - d3.extent(data, xValue)
  - [d3.max(data, xValue), d3.min(data, xValue)]
- 日期比例尺

```
xScale = d3.scaleTime()
.domain(d3.extent(data, xValue))
.range([0, innerWidth])
.nice();
```

折线图 – 动画的主循环

- 回忆散点图（气泡图）的动画循环：

```
let c = 0;
let intervalId = setInterval(() => {
    if(c >= allkeys.length){
        clearInterval(intervalId);
    }else{
        let key = allkeys[c];
        render_update_alter(provinces[key]);
        c = c + 1;
    }
}, 2000);
```

折线图 - “d”属性

- 定义Path的d属性的生成方式:
- （定义如何根据数据绘制一条线：）

```
const line = d3.line()
.x(d => {return xScale(xValue(d))})
.y(d => {return yScale(yValue(d))})
//.curve(d3.curveBasis)
.curve(d3.curveCardinal.tension(0.5))
```

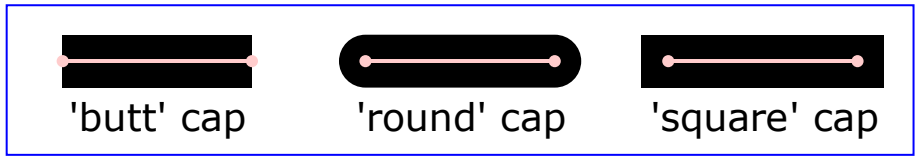
折线图 - .datum(…)

- 一条线为一个完整的“个体”
- .data(…)用于将一批图元与一批数据绑定
- .datum(…)用于给特定的一个图元绑定一个数据

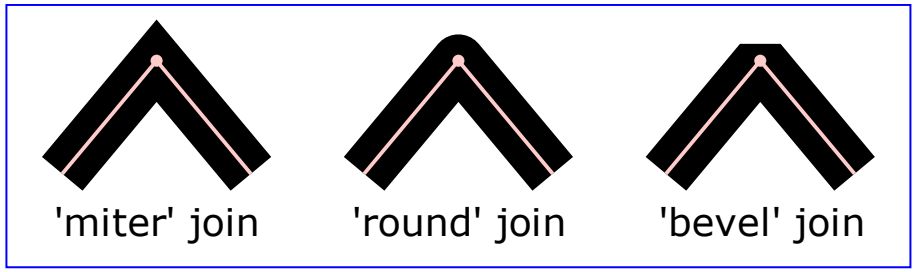
```
// See https://github.com/d3/d3-shape/blob/v1.3.7/README.md#curves
d3.select('#alterPath').datum(data)
.attr('class', 'datacurve')
.attr("fill", "none")
.attr("stroke", "green")
.attr("stroke-width", 2.5)
.transition().duration(2000).ease(d3.easeLinear)
.attr("d", line)
```

折线图 - Stroke

- stroke-linecap 属性可以让你在线条末端控制图形。你可以选择对接(butt)、方形(square)和圆形(round)



- stroke-linejoin 属性也是类似的，但是它控制的是两条线段之间的衔接。你可以在绘制折线时使用它。它有三个值，尖角(miter)、圆角(round)和斜角(bevel)。



折线图 - End

- 思考:
  - 第一条折线图（第一个动画）如何平滑出现?
  - 如何添加图例、时间戳?
  - 如何让坐标轴的刻度文字变大?
  - 答案均在这次与之前公开给大家的源代码和课上介绍中~
- 下次课:
  - 地图数据可视化
  - D3.js的交互