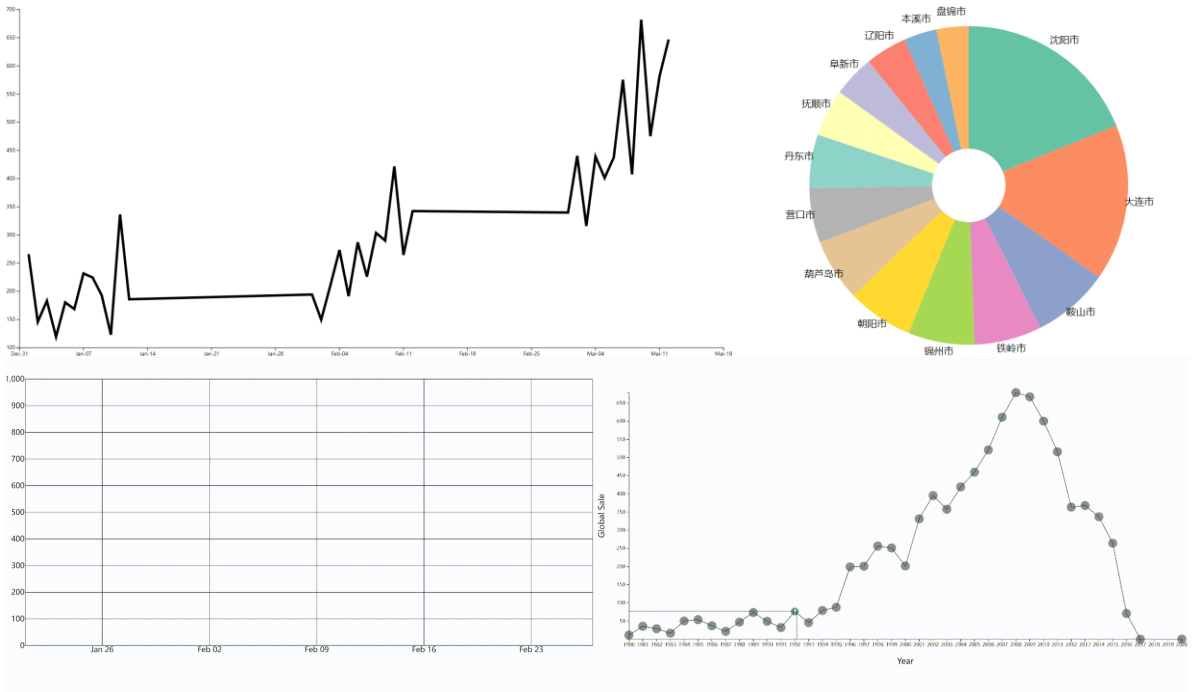
D3.JS - PATH

张松海 张少魁
清华大学计算机系
2022

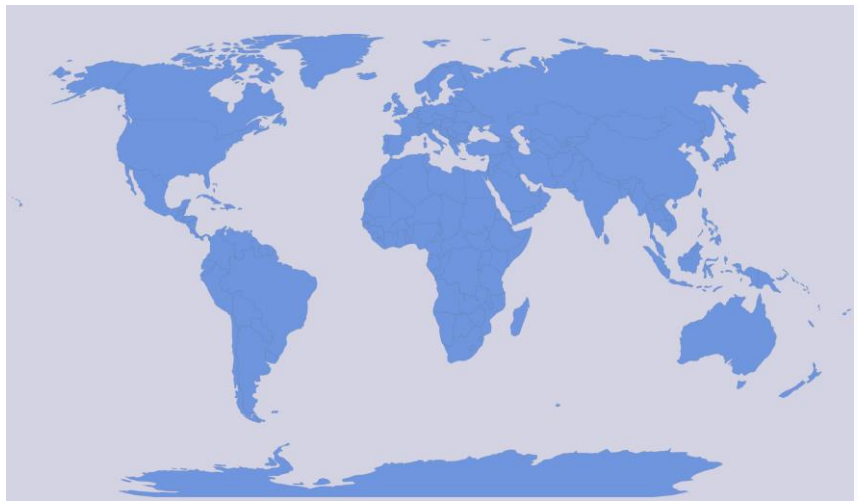
概览

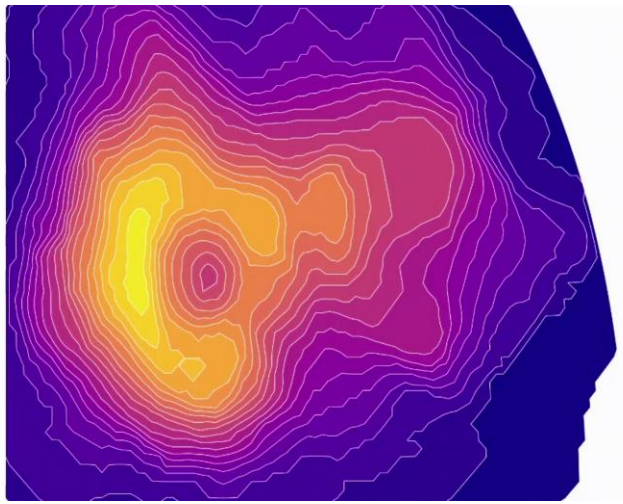
- 图元Path：
  - Path的常见属性。
  - ‘d’属性-‘笔画’的设置规则。
- D3.js的‘d’属性接口：
  - **d3.line()**
  - **d3.arc()**
- 基于d3.line()绘制折线图：
  - 时间数据的处理与时间比例尺。
  - 单一图元的Data-Join，selection.datum(…)。
- 基于d3.arc()绘制饼图：
  - 离散比例尺与D3.js的配色方案。
  - d3.pie()。



WHY PATH?

- <path>是SVG中非常强大的一个图元，可被用来绘制多种形状。
- 可用<path>绘制矩形（直角矩形、圆角矩形）、圆形、椭圆、弧、折线、多边形、贝塞尔曲线、基于数据的不规则轮廓等。
- Path是众多可视化方案使用的基础图元：
  - 折线图
  - 饼图
  - 地图
  - 等值线
  - 主题河流
  - …… ……



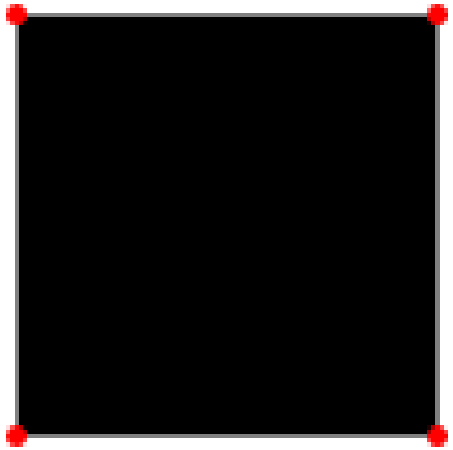


<PATH>

- 属性：
  - d：<path>勾勒的方式，即‘笔画’。
  - fill： 填充颜色 。
  - stroke：描边颜色 。
  - stroke-width：描边宽度 。
  - transform：变换。
- <path>的‘笔画’通过属性‘d’来勾勒，其值为“命令+参数”的序列：
  - 每一个命令都由一个关键字母给出，后可能接命令的参数，如坐标。
  - e.g., M 10 10 // 将画笔移动到(10, 10)。
  - e.g., H 90 // 将画笔水平勾勒到(90, 10)。
  - e.g., V 90 // 将画笔竖直勾勒到(90, 90)。
  - e.g., H 10 // 将画笔水平勾勒到(10, 90)。
  - e.g., L 10 10 // 将画笔勾勒到(10, 10)。

<PATH> - 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>
```

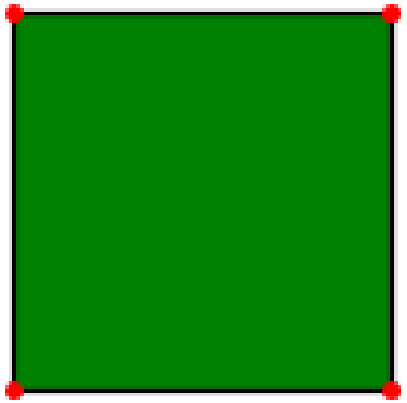


Code: hello-path.html（之后的Path示例也均在此代码中）

<PATH> - 直线命令示例

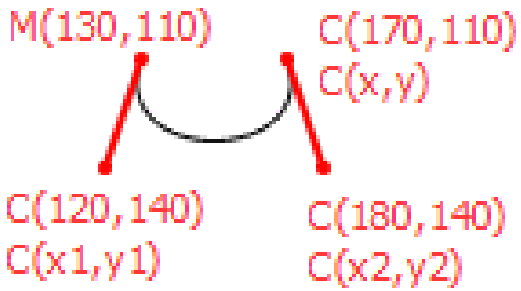
- 可以通过一个“闭合路径命令”Z来简化上面的path，简写形式：
  - Z：路径闭合。

```
<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>
```



<PATH> - 曲线命令示例

- 三次贝塞尔曲线需要定义一个起点、一个终点和两个控制点，用C命令创建三次贝塞尔曲线，需要设置四组坐标参数：
  - 起点基于‘M’命令，如M 130 110。
  - C x1 y1, x2 y2, x y：绘制从起点开始的曲线。
  - (x,y)：曲线的终点
  - (x1,y1)：起点的控制点。
  - (x2,y2)：终点的控制点。

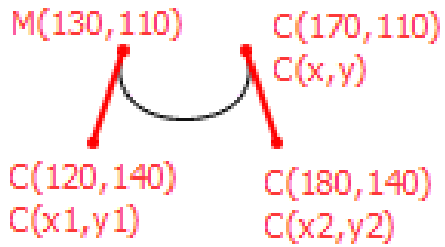


$$B(t) = P_0(1-t)^3 + 3P_1 t(1-t)^2 + 3P_2 t^2(1-t) + P_3 t^3, t \in [0,1]$$

<PATH> - 曲线命令示例

- 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。

```
<!--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>
```



<PATH> - 更多的‘D’属性

- ‘d’属性的可用命令：
  - 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
- 更详细的‘d’属性讲义已上传至网络学堂，made by 周文洋。

D3.js的‘d’属性接口

- ‘d’属性的原始语法可以拼出所有形状，但语法本身繁琐且复杂。
  - D3.js提供了常见形状到‘d’属性的转化。
- d3.line(...).x(...).y(...)
  - 用于将多个点依次连线，如折线图。
- d3.arc(...).innerRadius(...).outerRadius(...)
  - 弧，用于饼图。
- d3.geoPath().projection()
  - 用于地理、地形数据。
- d3.area()
  - 区域的‘d’属性，如主题河流。
- D3-Shapes: https://github.com/d3/d3-shape/tree/v1.3.7

D3.LINE()

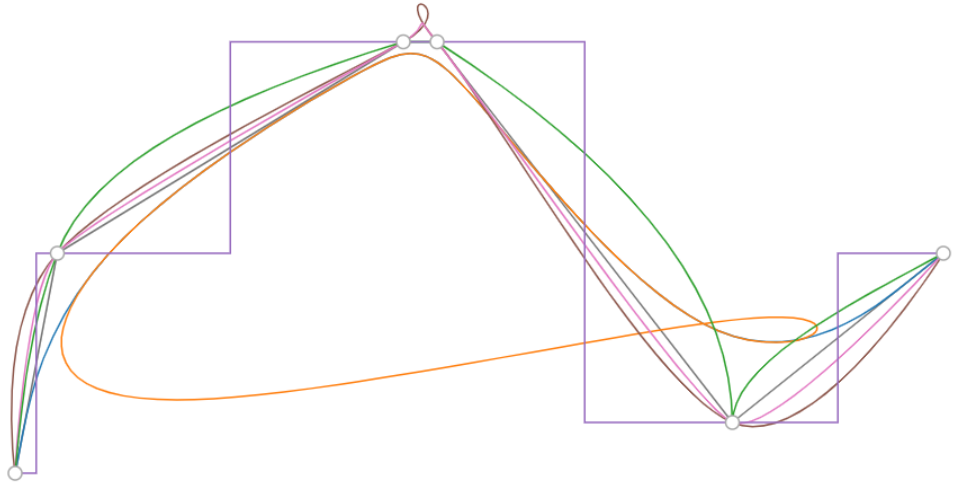
- d3.line()：用于将若干个点连成一条线，如折线图。
- D3.js中对于‘d’属性设置的接口都返回函数。
  - 输入通常是图元绑定的数据。
  - 设置图元‘d’属性时，直接把这个函数输入.attr(...)接口。
  - e.g., selection.attr(‘d’, **path**);
  - 故，返回函数的参数直接接受图元绑定的数据。
- **const path = d3.line().x(...).y(...)**
  - 定义一个函数，输入为**数组**、输出为‘d’属性。
  - 折线上每个点对应**数组**中每个元素，每个点的横纵坐标通过.x(...)与.y(...)，基于元素属性设置。
  - path.x(...)，横坐标要如何取值，如.x(d => d.age * 100)。
  - path.y(...)，纵坐标要如何取值，如.y(d => d.height * 3+50)。
  - e.g., const pathLine = d3.line().x(d => d.x).y(d => d.y);
  - e.g., pathLine([{'x': 100, 'y': 100},{'x': 200, 'y': 300},{'x': 300, 'y': 50},{'x': 400, 'y': 600}])
  - // 输出 ‘M100,100L200,300L300,50L400,600’

D3.LINE()

- 调用实例：

```
const myArray = [
    {'x': 100, 'y': 100},{'x': 200, 'y': 300},
    {'x': 300, 'y': 50},{'x': 400, 'y': 600}
];
const pathLine = d3.line().x(d => d.x).y(d => d.y);
svg.append('path').attr('stroke', 'black').attr('fill', 'none')
.attr('d', pathLine(myArray));
```

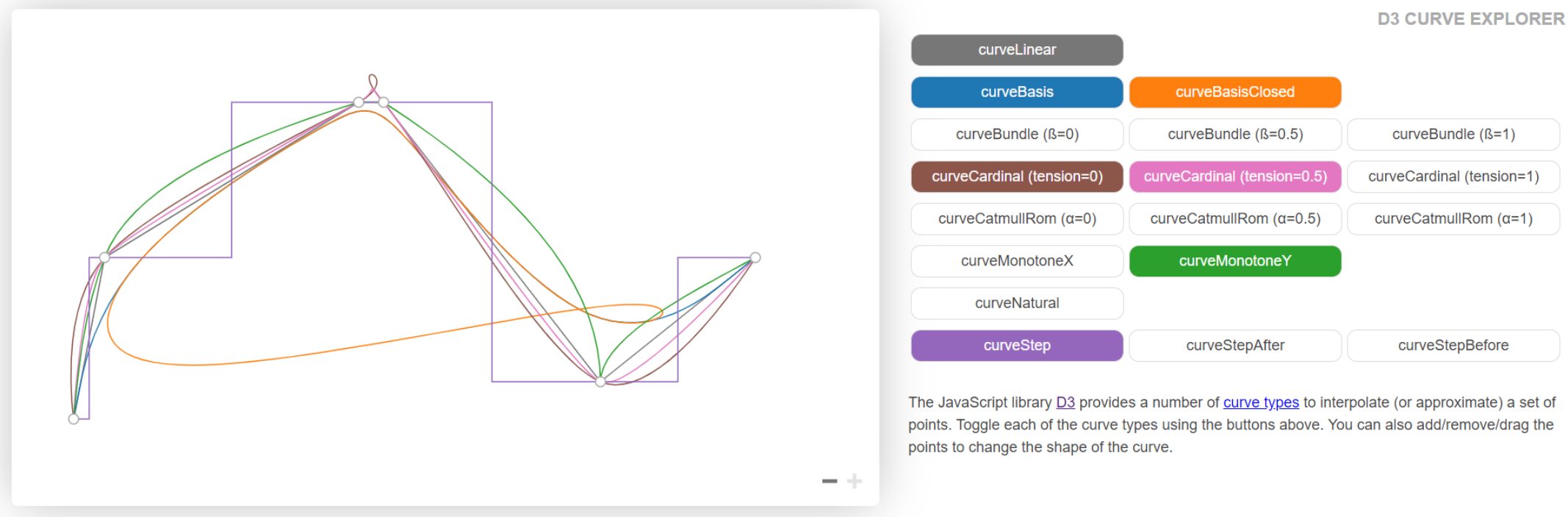
D3.line() – 连线的插值



- 要如何把若干端点连成一条线？
  - e.g., 端点间直接连线段。
  - e.g., 端点间可连一条平滑的曲线。
- **d3.line().x(...).y(...).curve(d3Curve)**
  - .curve(d3.curveCardinal.tension(0.5)); //平滑拟合，必过端点。
  - .curve(d3.curveBasis); //平滑拟合，不保证过端点。
  - .curve(d3.curveStep); //只走水平、竖直的直线，呈阶梯形状。
  - .curve(d3.curveLinear); //默认，端点间连直线。
- D3.js Curves：
  - https://github.com/d3/d3-shape/blob/v2.0.0/README.md#curves
- D3.js Curve Explorer：
  - http://bl.ocks.org/d3indepth/b6d4845973089bc1012dec1674d3aff8

D3.LINE() – 连线的插值

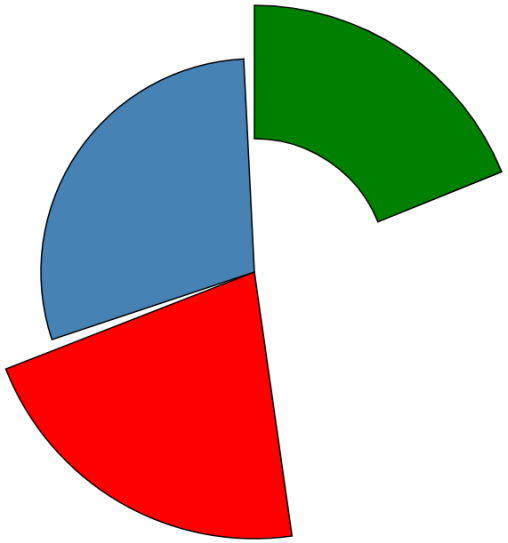
- 编程实例：
  - http://bl.ocks.org/d3indepth/b6d4845973089bc1012dec1674d3aff8



D3.ARC()

- d3.arc()：根据提供的角度勾勒圆弧，如饼图。
- **const path = d3.arc().innerRadius(100).outerRadius(200)**
  - 定义一个函数，用于将两个角度连成圆弧。
  - innerRadius(…)圆弧的内半径，如100个像素path.innerRadius(100)。
  - outerRadius(…)圆弧的外半径，如200个像素path.outerRadius(200)。
- 返回的函数接受的输入为一个对象，包括起始角度和终止角度：
  - {'startAngle': 3.0, 'endAngle': 4.34}
  - 注：’startAngle’与’endAngle’是必须的两个字段，要遵循d3.arc()的规则。
- path({‘startAngle’: 0, ‘endAngle’: 1.186}) // 输出下述结果

  如：M1.2246467991473532e-14,-200A200,200,0,0,1,185.34379999977125,-75.15101996410158L92.67189999988562,-37.575509982050 79A100,100,0,0,0,6.123233995736766e-15,-100Z
- d3.arc().innerRadius(…).outerRadius(…).**padAngle(0.1)**：
  - 设置圆弧两侧的预留间隔为0.1倍的圆弧角度。



D3.ARC()

- 编程实例：

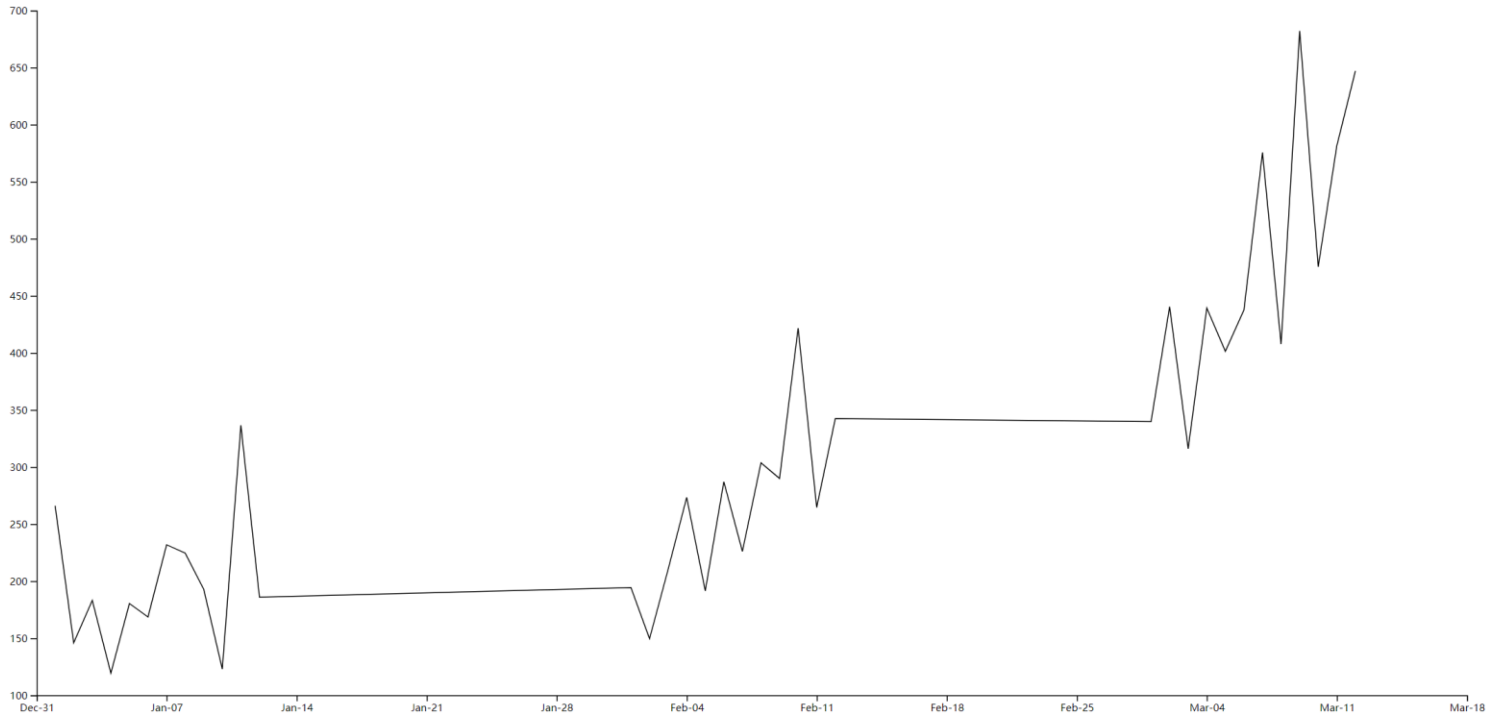
```
const part1 = {'startAngle': 0, 'endAngle': 1.1855848768577961};
const pathArc1 = d3.arc().innerRadius(100).outerRadius(200);
svg.append('path').attr('stroke', 'black').attr('fill', 'green').attr('transform', 'translate(800, 400)')
.attr('d', pathArc1(part1));

const part2 = {'startAngle': 3.0, 'endAngle': 4.34};
const pathArc2 = d3.arc().innerRadius(0).outerRadius(200);
svg.append('path').attr('stroke', 'black').attr('fill', 'red').attr('transform', 'translate(800, 400)')
.attr('d', pathArc2(part2));

const part3 = {'startAngle': 4.34, 'endAngle': 6.283185307179586};
const pathArc3 = d3.arc().innerRadius(0).outerRadius(160).padAngle(0.1);
svg.append('path').attr('stroke', 'black').attr('fill', 'steelblue').attr('transform', 'translate(800, 400)')
.attr('d', pathArc3(part3));
```

基于D3.line()绘制折线图

- 任务：商品销量变化随时间的趋势可视化。
- 数据来源：
  - https://www.kaggle.com/redwankarimsony/shampoo-saled-dataset
- D3.js接口扩展：
  - 时间数据的处理。
  - 时间比例尺。
  - 单一图元的Data-Join。



时间数据的处理与时间比例尺

- JavaScript提供处理日期的对象：
  - 输入为日期的字符串，输出为日期**对象**。
  - let myDate = new Date("2011-03-09");
- 在D3.js中，‘最大’与‘最小’并不是最准确的词会：
  - 任何一种数据类型存在‘顺序’，数值、字符串、日期等。
  - D3.js的接口，如**d3.min([...])**、**d3.max([...])** 、**d3.extent([...])**，只是参考顺序返回相应的端点值，即‘第一个’和‘最后一个’。
- 并非数值之间可以‘比较大小’：
  - e.g., new Date(“2011-03-09”) > new Date(“2018-06-06”); // false
  - e.g., 'abc' > '123' // true
- d3.min([...])、d3.max([...]) 、d3.extent([...])可以接受日期对象。

时间数据的处理与时间比例尺

- 将csv数据读取后，对于每一条数据：
  - 把字符串形式的日期转换成JavaScript的日期对象。
  - 把销量转换成数值。

```
d3.csv('shampoo_sales.csv').then(data => {
    data.forEach(d => {
        d.day = new Date(d.day);
        d.sale = +(d.sale);
    })
```

```
day,sale
1-01,266.0
1-02,145.9
1-03,183.1
1-04,119.3
1-05,180.3
1-06,168.5
1-07,231.8
1-08,224.5
1-09,192.8
1-10,122.9
1-11,336.5
1-12,185.9
2-01,194.3
2-02,149.5
2-03,210.1
2-04,273.3
2-05,191.4
2-06,287.0
2-07,226.0
2-08,303.6
```

时间数据的处理与时间比例尺

- const xScale = d3.**scaleTime()**;
  - 定义时间比例尺。
- xScale.domain(d3.extent(data.map(d => d.day))).range([0, 1600]);
  - 设置时间比例尺的定义域与值域。
  - 由于JavaScript的Date对象本身支持‘顺序’、’>’与’<’，因此可按照数值型变量基于d3.extent接口计算两个端点值。
  - 从语法、调用的角度，与线性比例尺并无区别。
- Tip：scaleTime和scaleLinear的对外接口本质上区别不大，主要在于scaleTime内部针对日期的插值。

时间数据的处理与时间比例尺

- 日期与时间的格式化方式与纯数值有区别。
- const timeFormat = d3.timeFormat('%b-%d')
  - 返回一个函数，函数用于将输入的日期整理成‘月-日’的格式。
- const xAxis = d3.axisBottom(xScale).tickFormat(timeFormat);
  - 对一个坐标轴统一应用某一个格式。
- 不同的格式占位符：
  - https://github.com/d3/d3-scale/blob/v3.2.2/README.md#time_tickFormat
  - %Y - for year boundaries, such as 2011.
  - %B - for month boundaries, such as February.
  - %b %d - for week boundaries, such as Feb 06.
  - %a %d - for day boundaries, such as Mon 07.
  - %I %p - for hour boundaries, such as 01 AM.
  - %I:%M - for minute boundaries, such as 01:23.

时间数据的处理与时间比例尺

- 编程实例：

```
d3.csv('shampoo_sales.csv').then(data => {
    data.forEach(d => {
        d.day = new Date(d.day);
        d.sale = +(d.sale);
    })
    xScale.domain(d3.extent(data.map(d => d.day))).range([0, innerWidth]).nice();
    yScale.domain(d3.extent(data.map(d => d.sale))).range([innerHeight, 0]).nice();
    const timeFormat = d3.timeFormat('%b-%d')
    const xAxis = d3.axisBottom(xScale).ticks(15).tickFormat(timeFormat);
    const yAxis = d3.axisLeft(yScale);
```

SELECTION.DATUM()

- selection.datum()
  - datum：英文中数据(data)的单数型。
  - 对**单一图元绑定单一数据**。
- 一条线为一个完整的“个体”：
  - d3.line(...)接受一个完整的数组作为数据。
  - .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。
  - .datum(...)由于只针对一个图元，故不涉及图元的补充与删除。
- 直接把数据(一个数组)作为单一图元绑定给一个<path>即可。

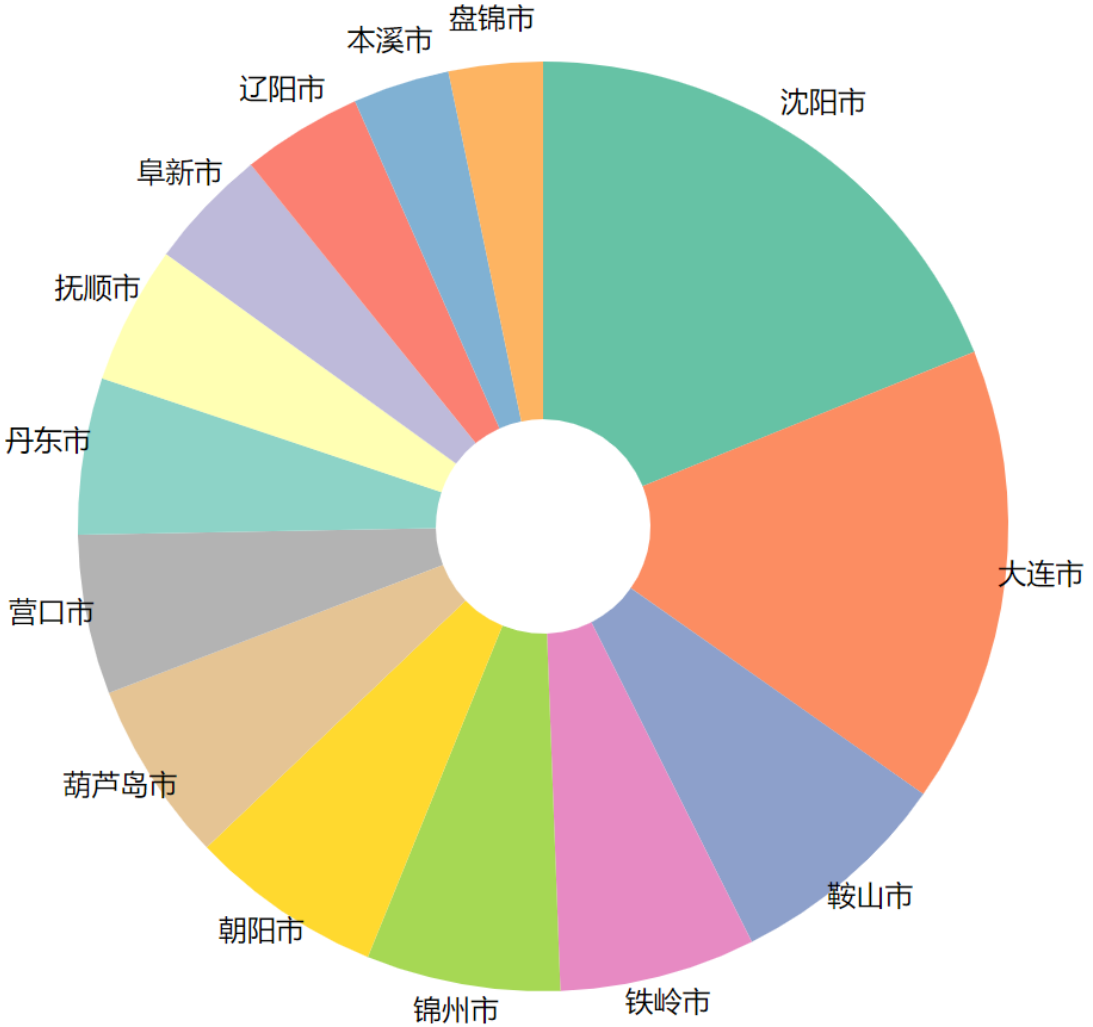
基于D3.LINE()绘制折线图

- 注意：图元已经绑定了数据，line作为一个函数接受的输入也是绑定的数据。
- 注意：不需要‘selectAll’。
- 编程实例：

```
const line = d3.line().x(d => xScale(d.day)).y(d => yScale(d.sale));
// line.curve(d3.curveCardinal.tension(0.5));
// line.curve(d3.curveBasis);
// line.curve(d3.curveStep);
// line.curve(d3.curveNatural);
line.curve(d3.curveLinear);
mainGroup.append('path').datum(data).attr('fill', 'none')
.attr('d', line).attr('stroke', 'black');
```

基于D3.ARC()绘制饼图

- 任务：辽宁省各城市人口占比可视化。
- 数据来源：
  - http://www.ttpaihang.com/news/daynews/2019/19021532835.htm
- D3.js接口扩展：
  - d3.pie()：由‘数值’到‘比例’的转化。
  - scaleOrdinal：离散到离散的比例尺。



D3.PIE()

- 原始数据是没有比例信息的：
  - 需要预计算每个城市人口占总人口的比例。
  - 需要把比例映射到[0, 2Pi]的弧度区间。
  - 可以自行计算，但无论是整理比例还是数据结构都很繁琐。
- const pie = d3.pie().value(...);
  - 返回一个函数，用于将输入关于值的数组转换成关于比例的数组。
  - d3.pie()是一个数据预处理 或 数据转换的接口。
  - e.g., const pie = d3.pie().value(d => d.population);
  - e.g., const arcData = pie(data);

```
city,population
沈阳市,829.4
大连市,698.75
鞍山市,344.0
铁岭市,299.8
锦州市,296.4
朝阳市,295
葫芦岛市,277.0
营口市,243.8
丹东市,239.5
抚顺市,210.7
阜新市,186.2
辽阳市,183.7
本溪市,147.63
盘锦市,143.65
```

D3.PIE()

- 数组转换前后对比：

```
▶ 0: {city: "沈阳市", population: "829.4"}
▶ 1: {city: "大连市", population: "698.75"}
▶ 2: {city: "鞍山市", population: "344.0 "}
▶ 3: {city: "铁岭市", population: "299.8"}
▶ 4: {city: "锦州市", population: "296.4"}
▶ 5: {city: "朝阳市", population: "295"}
▶ 6: {city: "葫芦岛市", population: "277.0"}
▶ 7: {city: "营口市", population: "243.8"}
▶ 8: {city: "丹东市", population: "239.5"}
▶ 9: {city: "抚顺市", population: "210.7"}
```

```
▶ 0: {data: {…}, index: 0, value: 829.4, startAngle: 0, endAngle: 1.1855848768577961, …}
▶ 1: {data: {…}, index: 1, value: 698.75, startAngle: 1.1855848768577961, endAngle: 2.184412261357899, …}
▶ 2: {data: {…}, index: 2, value: 344, startAngle: 2.184412261357899, endAngle: 2.6761426660348726, …}
▶ 3: {data: {…}, index: 3, value: 299.8, startAngle: 2.6761426660348726, endAngle: 3.104691431506258, …}
▶ 4: {data: {…}, index: 4, value: 296.4, startAngle: 3.104691431506258, endAngle: 3.5283800708849062, …}
▶ 5: {data: {…}, index: 5, value: 295, startAngle: 3.5283800708849062, endAngle: 3.950067481872427, …}
▶ 6: {data: {…}, index: 6, value: 277, startAngle: 3.950067481872427, endAngle: 4.346024813545455, …}
▶ 7: {data: {…}, index: 7, value: 243.8, startAngle: 4.346024813545455, endAngle: 4.694524443371752, …}
▶ 8: {data: {…}, index: 8, value: 239.5, startAngle: 4.694524443371752, endAngle: 5.036877443139587, …}
▶ 9: {data: {…}, index: 9, value: 210.7, startAngle: 5.036877443139587, endAngle: 5.338062316004233, …}
▶ 10: {data: {…}, index: 10, value: 186.2, startAngle: 5.338062316004233, endAngle: 5.604225692024153, …}
▶ 11: {data: {…}, index: 11, value: 183.7, startAngle: 5.604225692024153, endAngle: 5.8668154459170605, …}
▶ 12: {data: {…}, index: 12, value: 147.63, startAngle: 5.8668154459170605, endAngle: 6.077844979761426, …}
▶ 13: {data: {…}, index: 13, value: 143.65, startAngle: 6.077844979761426, endAngle: 6.283185307179586, …}
```

D3.PIE()

- d3.pie()会自动帮助映射到原始数据，保证Data-Join的输入数据仍保留原始的全部内容：

```
▼ (14) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}]
  ▼ 0:
    ▶ data: {city: "沈阳市", population: "829.4"}
      endAngle: 1.1855848768577961
      index: 0
      padAngle: 0
      startAngle: 0
      value: 829.4
    ▶ __proto__: Object
  ▶ 1: {data: {…}, index: 1, value: 698.75, startAngle: 1.1855848768577961, endAngle: 2.184412261357899, …}
  ▶ 2: {data: {…}, index: 2, value: 344, startAngle: 2.184412261357899, endAngle: 2.6761426660348726, …}
  ▶ 3: {data: {…}, index: 3, value: 299.8, startAngle: 2.6761426660348726, endAngle: 3.104691431506258, …}
```

D3.PIE()

- 编程实例：

```
const pie = d3.pie().value(d => d.population);
console.log(data);
const arcData = pie(data);
console.log(arcData);
```

基于D3.ARC()绘制饼图

- 注意：
  - 一个圆弧对应一个<path>图元。
  - 对于每个<path>，只需要起始角度和终止角度。
  - 使用.data()而不是.datum()。
- 编程实例：

```
const path = d3.arc().innerRadius(60).outerRadius(260);
svg.selectAll('path').data(arcData).join('path')
.attr('d', path)
.attr('transform', `translate(${width / 2}, ${height / 2})`)
.attr('fill', d => color(d.data.city));
```

离散比例尺与D3.js的配色方案

- **let s = d3.scaleOrdinal().domain(...).range(...):**
  - .domain()接受一个离散的数组，如.domain(['Shao-Kui', 'Wen-Yang', 'Yuan-Chen'])。
  - .range()接受一个离散的数组，如.range(['red', 'green', 'yellow'])。
  - 返回一个函数，s('Shao-Kui') // 'red'.
- D3.js的配色方案：
  - d3.schemeSet1、d3.schemeSet2、d3.schemeSet3。
  - 上述这些方案不是函数，本身就是数组。
  - e.g., console.log(d3.schemeSet1) // ["#e41a1c", "#377eb8", "#4daf4a", "#984ea3", "#ff7f00", "#ffff33", "#a65628", "#f781bf", "#999999"]

离散比例尺与D3.js的配色方案

- array1.concat(array2)：
  - JavaScript的接口，用于拼接两个数组。
  - d3.schemeSet2内的配色数量不够，故拼接一个d3.schemeSet3。
- 编程实例：

```
const color = d3.scaleOrdinal()
.domain(data.map(d => d.city))
.range(d3.schemeSet2.concat(d3.schemeSet3));
```

基于D3.ARC()绘制饼图

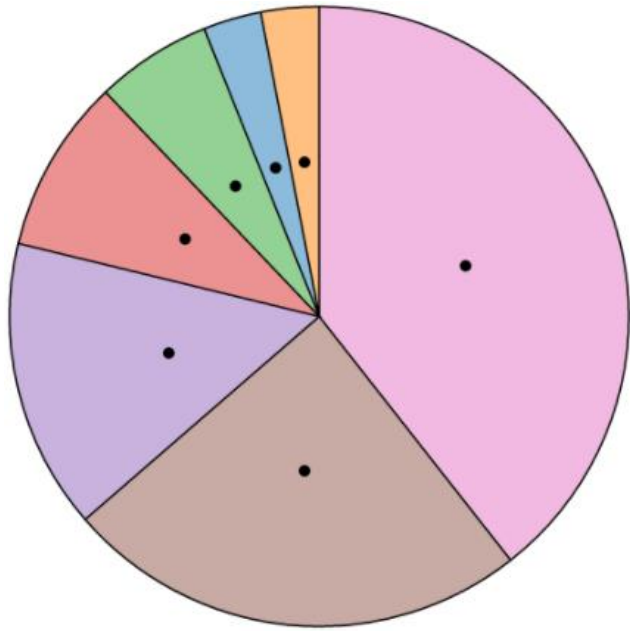
- 编程实例：

```
d3.csv('liaoning.csv').then(data => {
    const pie = d3.pie().value(d => d.population);
    const arcData = pie(data);
    console.log(arcData);
    const path = d3.arc().innerRadius(60).outerRadius(260);
    const color = d3.scaleOrdinal()
    .domain(data.map(d => d.city))
    .range(d3.schemeSet2.concat(d3.schemeSet3));

    svg.selectAll('path').data(arcData).join('path')
    .attr('d', path)
    .attr('transform', `translate(${width / 2}, ${height / 2})`)
    .attr('fill', d => color(d.data.city));
```

Tip：如何为饼图添加文字标签?

- const arcOuter = d3.arc().innerRadius(280).outerRadius(280);
  - 定义一个完全‘贴’在外围的arc函数。
- arcOuter.centroid({‘startAngle’: 0, ‘endAngle’: 1.186}):
  - 返回输入圆弧的中心点坐标。
  - 如右下方图片所示。
  - 根据圆弧内外径的定义，中心点也会随之向内/外。
- 代码示例：



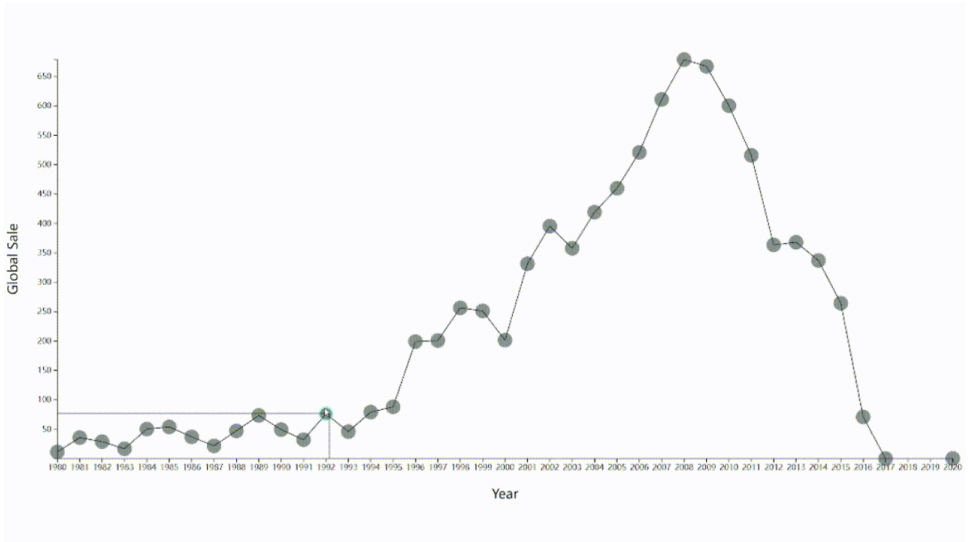
```
const arcOuter = d3.arc().innerRadius(280).outerRadius(280);
svg.append('g').attr('transform', `translate(${width / 2}, ${height / 2})`)
.selectAll('text').data(arcData).join('text')
.attr('transform', d => `translate(${arcOuter.centroid(d)})`)
.attr('text-anchor', 'middle')
.text(d => d.data.city);
```

Tip：如何为饼图添加文字标签?

- 不做要求

```
arcGroup.selectAll('polyline').data(pie(data)).join('polyline')
.attr('opacity', d => isGoodAngle(d)? 1:0)
.attr('stroke', 'black').attr('stroke-width', '2px').attr('fill', 'none')
.attr('points', d => {
    let pos = outerArcGenerator.centroid(d);
    pos[0] = 130 * (midAngle(d) < Math.PI ? 1 : -1);
    return [arcGenerator.centroid(d), outerArcGenerator.centroid(d), pos]
}).transition().duration(1000).attrTween("stroke-dasharray", tweenDash);
```

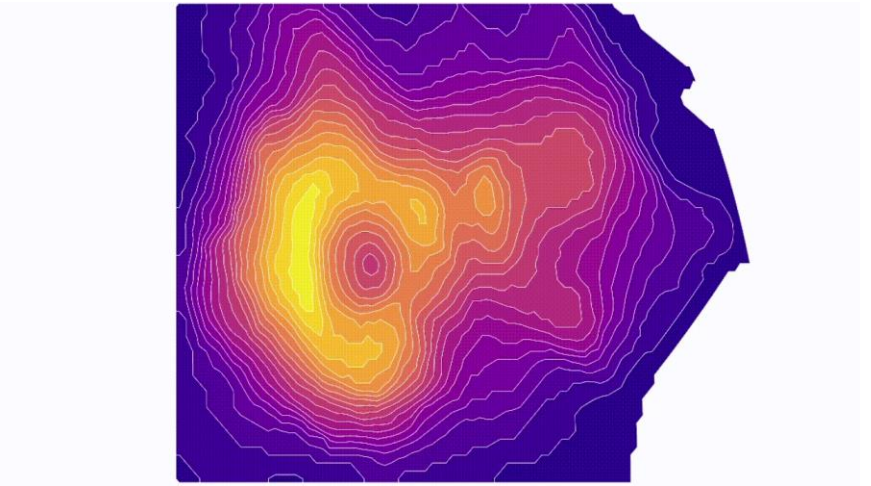
```
const tweenDash = function(){
    let l = this.getTotalLength(),
        i = d3.interpolateString("0," + l, l + "," + l);
    return function (t) { return i(t); };
}
```



Tip：D3.js提供对‘d’属性的插值

- D3.js提供对‘d’属性的插值：
  - 可对‘d’属性应用.transition()接口。
  - e.g., pathupdate.merge(pathenter).transition().duration(2000).attr("d", line)
  - 注意：谨慎使用，插值过程（行为）在一些情况并不会符合预期…

插值成功 😀



插值失败 😭

为什么会这样？

Tip：D3.js提供对‘d’属性的插值

- 这个做要求！至少请勿想当然的对’d’属性插值！
- 当D3-Transition模块遇到需要插值的属性时：
  - 如果是数值，直接插值。
  - 如果是颜色，使用InterpolateRGB。
  - 如果是文本，会使用InterpolateString。
- InterpolateString做了什么：
  - 找到文本中嵌入的所有数字，对这些数字进行插值。
  - 因此，’d’属性只有在前后‘格式’一直的情况下才可以照常插值。
  - M100,100L200,300L300,50 -> M100,100L666,233L123,55 😁
  - M100,100L200,300L300,50 -> M130 110 C 120 140, 180 140, 170 110 😫
- https://github.com/d3/d3-transition/blob/v3.0.0/README.md#modifying-elements