

# D3.js

## 动态可视化效果

张松海 张少魁

清华大学 可视媒体研究中心

# 概览

- 动画：
  - d3.transition与时长控制。
  - ease。
  - 动画的同步。
  - 自定义插值（不做要求）。
- 数字的格式化。
- 基于动画绘制动态散点图。
  - 数据的清洗与预处理。

# 动画

- D3-transition
  - 引用D3.js文档: 'Selection-like' interface。
  - D3的selections后调用transition, 将后续的.attr(…)加以动画效果。
- **selections.transition().duration(ms).attr(type, value)**
  - ms是毫秒。
  - e.g., d3.select("#my\_rect").transition().duration(2000).attr("width", "400");
  - 即用两秒钟(2000毫秒)的时间把选择图元的宽变成400。
- 同d3.selections, 支持链式调用:
  - d3.select('#my\_rect').transition().duration(1000).attr(…).attr(…).attr(…)
- 只有transition()后面的.attr(…)会有动画效果。
  - d3.select('#my\_rect').**attr(…)**.transition().duration(1000).**attr(…)**

# 动画

- 调用实例:

```
<body>
  <svg width="960" height="400" id="mainsvg"
    class="svgs" style='display: block; margin: 0 auto;'>
    <rect id="my_rect"
      x="10" y="200" width="200" height="30"
      stroke="black" fill="#69b3a2" stroke-width="1"
    />
  </svg>
  <script>
    d3.select("#my_rect")
      .transition().duration(2000)
      .attr("width", "400");
  </script>
</body>
```

# 动画

- ‘ease’, 动画过渡的方式。
- 通过链式调用的方式作用在transition对象后：
  - `d3.select("#my_rect").transition().duration(2000).ease(d3.easeLinear).attr(...)`
- D3提供多种Ease函数供选择：
  - `d3.easeCubic` 默认, 加速->减速
  - `d3.easeLinear` 线性
  - `d3.easeElastic` 弹射
  - ... .. (见下页)

# 动画 - Ease



easeElastic

easeBounce

easeLinear

easeSin

easeQuad

easeCubic

easePoly

easeCircle

easeExp

easeBack

# 动画 – 同步

- ‘显示’定义Transition对象：
  - `let transition = d3.transition().duration(ms)`
- 并让其与多个图元共享，即所有绑定的图元会同步播放动画：
  - `rects.transition(transition).attr(...).attr(...)`
  - `circles.transition(transition).attr(...).attr(...)`
- 等待一个transition全部相关的图元执行完毕：
  - `await transition.end()`
- JavaScript异步：
  - `async`: 将函数转换成异步函数，即把返回值包装成一个Promise对象。
  - `await`: 等待异步函数执行结束。

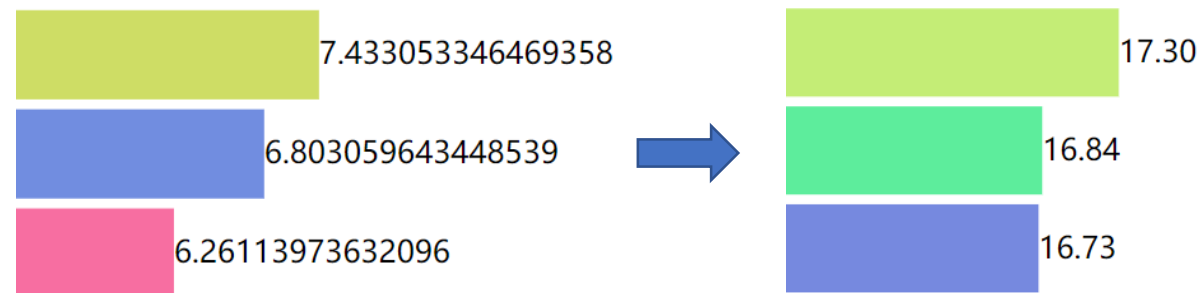
# 动画 - 同步

- 编程实例:

```
(async () => {  
  while(true){  
    data.forEach(d => {  
      d.value = Math.random() * 20;  
    });  
    let transition = d3.transition().duration(1000);  
  
    rects.transition(transition)  
      .attr('width', d => xScale(d.value))  
      .attr('fill', () => d3.interpolateRainbow(Math.random()));  
  
    texts.transition(transition)  
      .attr('x', d => xScale(d.value))  
      .text(d => formatPercent(d.value));  
  
    await transition.end();  
  }  
})();
```



# 数字的格式化



- 把输入的数字按照要求的精度、长度、有效数字等输出：
  - `d3.format('specifier')`: 返回一个函数。
- 典型的formater, '.'后的数字表示精度:
  - `d3.format('.2f')(666.666)` // 小数点后保留两位, 666.67
  - `d3.format('.2r')(2467)` // 只保留两位有效数字 2500
  - `d3.format('.3s')(2366.666)` // 只保留三位有效数字且加以后缀 2.37k
- 坐标轴的数字格式化:
  - `axis.tickFormat(d3.format(...))`: 根据formater来设置坐标轴上的数字格式,
  - 等价于把数字格式化的函数作为参数输入给坐标轴。
- 格式化'specifier':
  - 完整规则: <https://github.com/d3/d3-format/blob/v2.0.0/README.md#format>
  - 大量实例: <https://observablehq.com/@d3/d3-format>
  - 根据需求查询文档即可。

# 数字的格式化

- 调用实例:

```
var formatPercent = d3.format(".2f");  
console.log(435.1241241234);  
console.log(formatPercent(435.1241241234)); // 435.12  
console.log(66.818213412314);  
console.log(formatPercent(66.813213412314)); // 66.81
```

# 动画 – 自定义插值

- (不做要求)
- D3.js并不提供所有属性的插值方式：
  - ease仅仅是0 ~ 1的过程, easeLinear会让0-1平缓过渡、easeCubic会在0时加速在接近1时减速。
  - ‘0 ~ 1’所控制的是插值函数  $f(t)\{ \cdots \}$ 。
  - 如, 文本没有默认的插值。
  - tween(...), attrTween(...), styleTween(...)
- d3.interpolate(a, b):
  - 返回一个函数, 函数的输入从0到1, 会映射到原始的[a, b]。
  - `let f = d3.interpolate(233, 666); f(0.3) // 362.9`

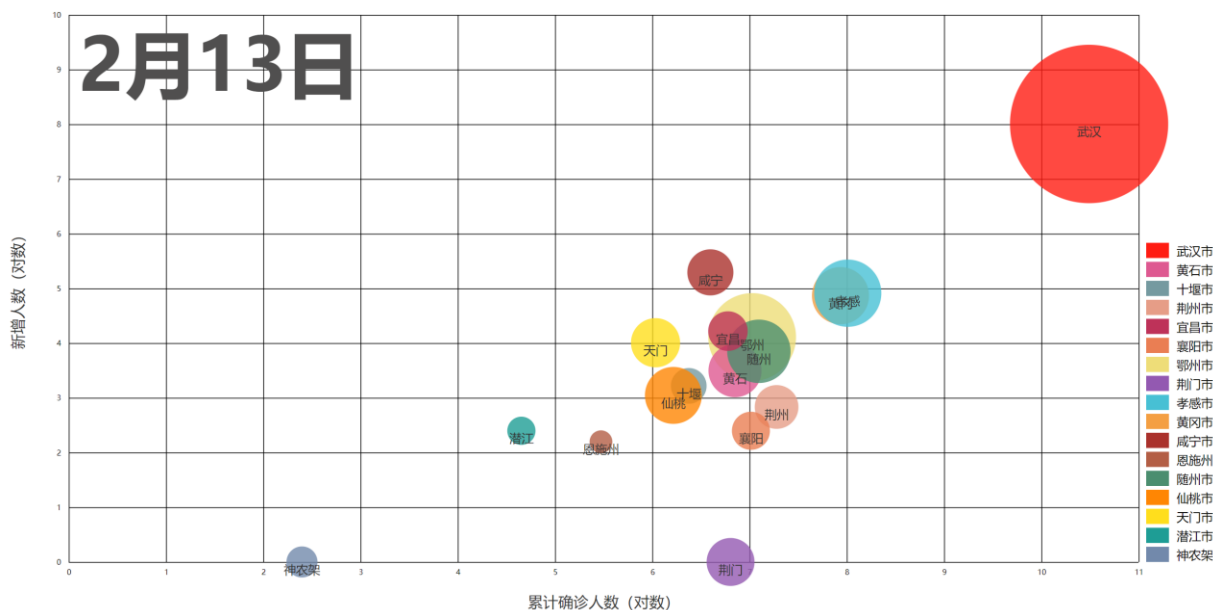
# 动画 – 自定义插值

- 以.tween为例，其输入的为一个返回函数的函数：
  - 返回的函数：接受的输入为0-1的值，此0-1值由D3在transition的过程中调控，受easeLinear、easeElastic等的影响，根据输入的0-1的某一值，返回相应的图元属性。
  - 输入的函数：用于生成返回的函数，如定义函数的前后端点值、插值方式等。
- 调用实例：

```
.tween("text", function(d) {  
    var i = d3.interpolate(this.textContent, d.value);  
    return function(t) {  
        this.textContent = formatPercent(i(t));  
    };  
});
```

# 实现动态散点图

- 编程实例:
- 数据的读取、清洗与预处理;
- 初始化比例尺、坐标轴与图例;
- 主循环;
- 图元添加与属性的修改、属性的更新与动画;



# 数据的读取、清洗与预处理;

- 湖北疫情数据:
  - hubei\_day14.csv
- Array.from(new Set(data.map( datum => datum['日期'] )));
- Array.from(arrayLike):从一个类似数组或可迭代对象创建一个新的数组。
- new Set(array)根据数组创建集合,会自动删去其中的重复元素。
- \* csv(...).then(...)中的回调函数已经是异步函数。
- 编程实例: →

```
d3.csv('hubei_day14.csv').then(async function(data){
  data = data.filter(d => {return keyValue(d) !== '总计'});
  data.forEach(datum => {
    // pre-process the data;
    datum['确诊人数'] = +(datum['确诊人数']);
    datum['治愈人数'] = +(datum['治愈人数']);
    datum['死亡人数'] = +(datum['死亡人数']);
    datum['新增确诊'] = +(datum['新增确诊']);
    if(datum['新增确诊'] < 0){
      datum['新增确诊'] = 0;
    }
    datum['感染率'] = datum['确诊人数'] / 1000;
  });

  // remove duplicated items;
  alldates = Array.from(new Set(data.map( datum => datum['日期'])));

  // make sure dates are listed according to real time order;
  alldates = alldates.sort(function(a,b){
    return new Date(b.date) - new Date(a.date);
  });

  // re-arrange the data sequentially;
  sequential = [];
  alldates.forEach(datum => {
    sequential.push([]);
  });
  data.forEach(datum => {
    sequential[alldates.indexOf(datum['日期'])].push(datum);
  });
});
```

# 初始化比例尺、坐标轴与图例

- 比例尺、坐标轴、图例等通常定义并渲染一次后不再改变：
- 将可视化代码分成初始化、更新两个部分：
  - 初始化：定义比例尺、定义maingroup、渲染坐标轴；
  - 更新：在主循环中不断改变数据并做Data-Join与Transition。
- \* 仍需要定义margin。
- 编程实例：→

```
const renderinit = function(data, seq){  
  xScale = d3.scaleLinear()  
    .domain(d3.extent(data, xValue)) // "extent"  
    .range([0, innerWidth])  
    .nice();  
  
  yScale = d3.scaleLinear()  
    .domain(d3.extent(data, yValue).reverse())  
    .range([0, innerHeight])  
    .nice();  
  
  maxX = xScale(d3.max(data, xValue));  
  maxY = yScale(d3.max(data, yValue));
```

```
let yAxisGroup = g.append('g').call(yAxis)  
  .attr('id', 'yaxis');  
yAxisGroup.append('text')  
  .attr('font-size', '2em')  
  .attr('transform', `rotate(-90)`)  
  .attr('x', -innerHeight / 2)  
  .attr('y', -60)  
  .attr('fill', '#333333')  
  .text(yAxisLabel)  
  .attr('text-anchor', 'middle')
```

# 主循环

- 代码示例:

```
renderinit(data, sequential[0]);  
  
for(let i = 0; i < sequential.length; i++){  
    await renderupdate(sequential[i]);  
}
```



# 图元添加、属性的修改与更新、动画

```
const renderupdate = async function(seq){
  const g = d3.select('#maingroup');

  d3.select('#date_text').text(seq[0]['日期']);

  let transition = d3.transition().duration(aduration).ease(d3.easeLinear);

  let circleupdates = g.selectAll('circle').data(seq).join('circle')
    .attr('fill', d => color[keyValue(d)] )
    .attr('opacity', .8)
    .transition(transition)
    .attr('cy', d => yScale(yValue(d)))
    .attr('cx', d => xScale(xValue(d)))
    .attr('r', c => rValue(c));
    ...
  await transition.end();
};
```

```
let textupdates = g.selectAll('.province')
  .attr("class", "province_text")
  .attr("dy", "1em")
  .style("text-anchor", "middle")
  .attr("fill", "#333333")
  .text( d => keyValue(d))
  .transition(transition)
  .attr('x', d => xScale(xValue(d)))
  .attr('y', d => yScale(yValue(d)));
```

# 图元的起始状态？

- 若按照上述方法实现，存在的问题：
  - 图元从**创建到第一次绑定数据**的transition不合理。
- PlanA：对第一次的数据绑定单独做data(...).join(...)，把图元创建时的动画与更新的动画分开。
- PlanB：利用D3.js自带的接口，分别设置Data-Join的行为，包括enter-图元创建时的行为、update-图元更新的行为（即data(...)本身）、exit-图元移除时的行为。