

# D3.js – Data-Join

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

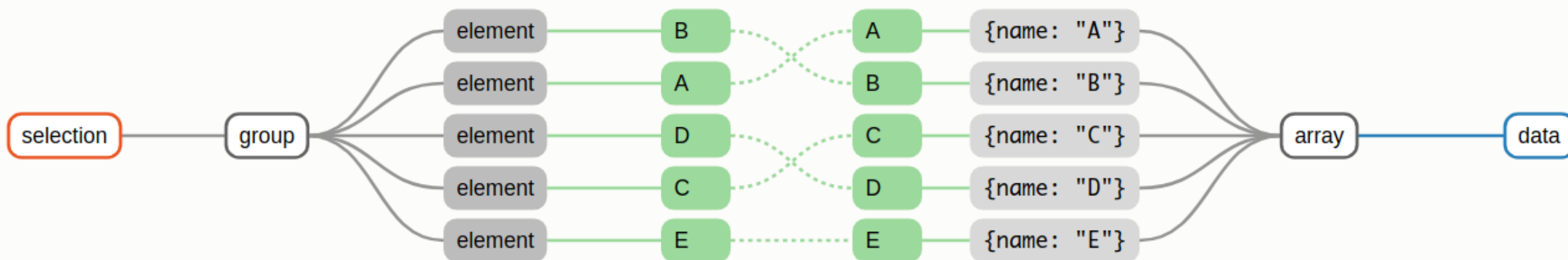
# 用函数设置图元属性

- 回忆使用.attr设置图元属性
- selection.attr('attributeName', 'value')
  - 支持直接通过值来设置属性
  - 支持通过函数来设置属性
- selection.attr('attributeName', (d, i) => {...})
  - d为绑定给图元的数据（即将到来）
  - i为图元的索引，是一个整数，如d3.selectAll('rect')中的第几个矩形
  - 函数也可以仅使用 d => {...}，但此时函数体无法使用索引
  - 即使不使用绑定的数据（如没有绑定数据），如需使用索引，仍需要完整的写出 (d, i) => {...}

# 数据与图元的绑定

- `.data( dataArray )`
- `dataArray`在保证是一个数组的前提下可以是任何形式
  - e.g., `[0, 2, 5, 6, 233, 666, 384, 32, 18]`
  - e.g., `[{name: 'Sebastian', value:384}, {name:' Ciel', value:32},`  
`{name:'Cai Yun', value:16}]`
- 先考虑数据和图元数目相同的情况：
  - `dataArray`是一个数组，其中的每‘条’数据会与一个图元绑定（反之亦然）
- 绑定给每个图元的数据将对应 `.attr(, (d, i) => {...})` 中的 **d**
- **默认的绑定按照双方的索引顺序**
- 不调用`.data(...)`，则图元不会与任何数据绑定！
- **数据的更新只需要重新绑定另一个 `dataArray` 即可**

# Key<sup>^</sup>



<https://user-gold-cdn.xitu.io/2018/6/29/1644982de2098431?imageslim>

# Key

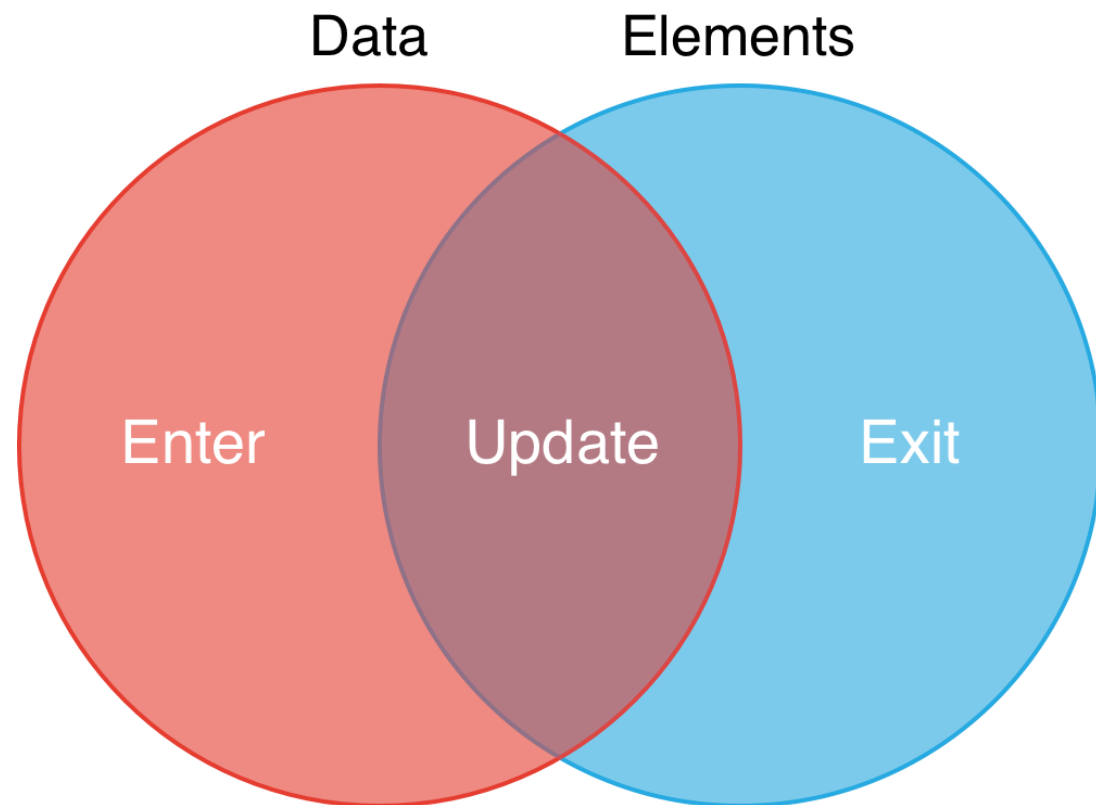
- `.data(data, keyFunction)`
  - `keyFunction`的返回值通常是一个字符串 (string)
  - `keyFunction`的定义根据数据, 比如 `keyFunction = d => d.name`
- 在绑定数据给图元时:
  - `keyFunction`为每条**输入绑定的数据**执行一次
  - `keyFunction`为每个**包含数据的图元**执行一次
- 如果图元之前没有绑定过任何数据, 则`keyFunction`会报错!
  - 第一次绑定时根据索引即可
  - 实际的可视化任务, 图元都是根据数据的‘条’数动态添加 (enter)、删除 (exit), 只需要在添加时指定好DOM的ID即可
- code: <https://github.com/Shao-Kui/D3.js-Demos/blob/master/static/d3-tutorial/datajoin.html>

# Data-Join

- 本质上是将数据与图元进行绑定
  - 每个国家的人数绑定到矩形的长度
  - 疫情感染的人数比例绑定到圆的半径
- 以数据为中心的可视化操作
  - 根据数据的每个属性自动调整绑定图元的属性
- 不再需要手动添加、‘修改’、删除图元
  - 会根据Data-Join的绑定自动推断
- 如果图元的数目不等于数据的条目？
  - 根据数据条目的数量选定相应数量的图元

# Enter Update Exit

- D3.js绑定数据的三个‘状态’
- Update
  - 图元和数据条目相同，之前的介绍均为单纯的update
- Enter
  - 数据的条目多于图元甚至没有图元，常用于第一次绑定数据
- Exit
  - 数据的条目少于图元甚至没有数据，常用于结束可视化



# Enter

- 有数据没图元
- D3.js会自动‘搞清楚’哪些数据是新增的
- 根据新增的数据生成相应的图元
- 生成图元的占位，占位的内容需要编程者自行添加（append）
- `const p =`  
`maingroup.selectAll('.class').data(data).enter().append('').attr(...)`
- （不做要求）
  - enter本质上生成指向父节点的指针，而append操作相当于在父节点后添加指针数量的图元并将其与多出的数据绑定



# 使用Data-Join (enter) 来实现柱状图


- 通过enter状态来实现之前的柱状图
- code: <https://github.com/Shao-Kui/D3.js-Demos/blob/master/static/d3-tutorial/datajoin2.html>

```
g.selectAll('.dataRect').data(data).enter().append('rect')  
.attr('class', 'dataRect')  
.attr('width', d => xScale(d.value))  
.attr('height', yScale.bandwidth())  
.attr('y', d => yScale(d.name))  
.attr('fill', 'green').attr('opacity', 0.8)
```

# Update

- 有图元有数据
- `const p =  
 maingroup.selectAll('.datacurve').data(data).attr(...).attr(...)`
- Update作为实际可视化任务最常用的状态，经常被单独封装成一个函数
- `updateSelection.merge( enterSelection ).attr(...).attr(...)`
  - 将两个selection合并到一起操作
  - `enterSelection`需要至少`append(...)`图元

# 让数据动起来

- Update经常与D3.js的动画一起使用
- `.transition().duration()` 
- `d3.selectAll('rect').data(data2, d => d.name)`
- `.transition().duration(3000).attr('width', d => xScale(d.value))`
- `.duration(...)`中为毫秒，即3000表示3秒钟
- `.transition(...)`经过调用后，后续的链式调用会变成数值上的渐变，渐变的时间由`.duration(...)`设定
- （本节课不做要求）插值的方式由`.ease(...)`设定

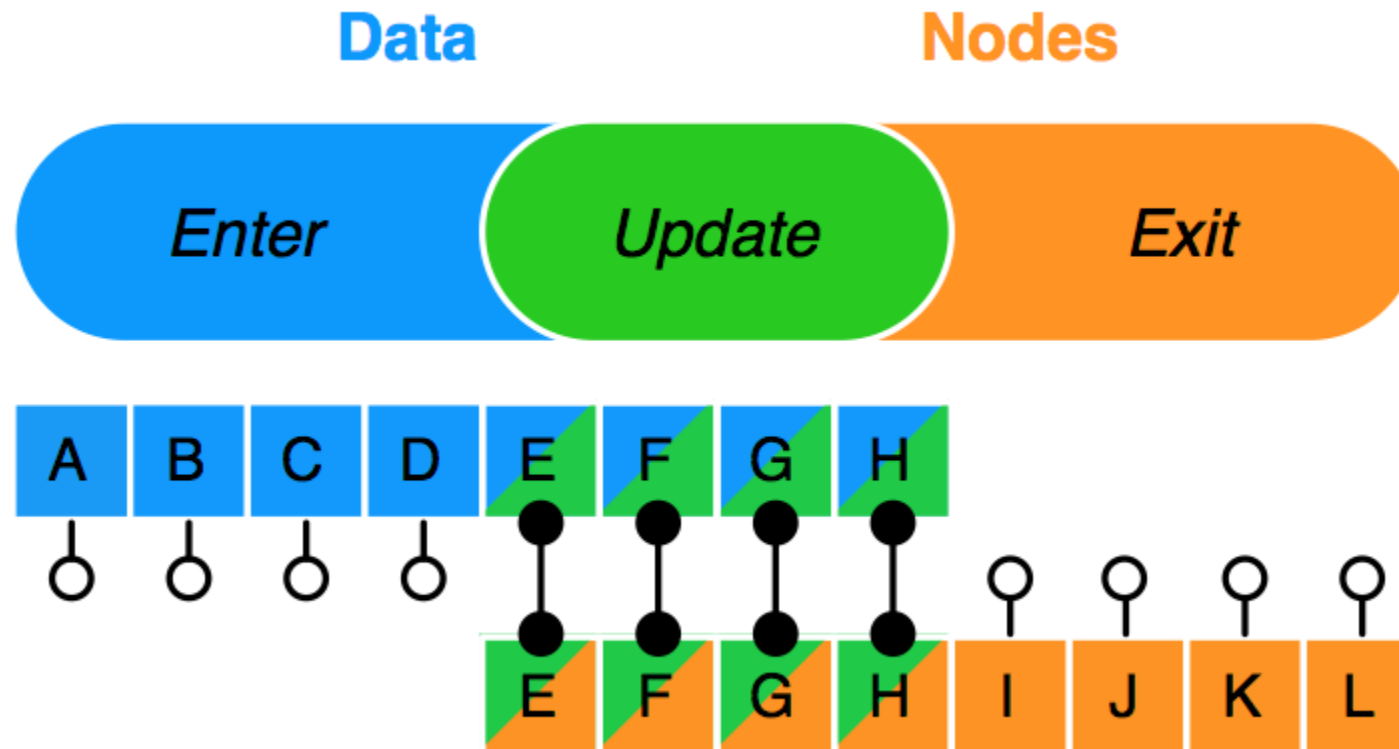
# Exit

- 有图元没数据  
D3.js会自动‘搞清楚’哪些图元是不绑定数据的
- 引用官方文档: existing DOM elements in the selection for which no new datum was found
- `const p = maingroup.selectAll('.class').data(data).exit().remove()`

# Data-Join的简洁形式

- `.data(...).join('...')`
- 默认enter和update的执行形式相同
- 默认exit是删除 (remove) 节点
- 默认data-join形式简洁但不灵活
  - 必须需要设置enter数据的初始图元属性, update会每次重新设置初始值, 从而导致动画出现‘奇怪’的效果
  - 仍支持‘定制’:
  - `.join(
    - enter => enter.append("text").attr("fill", "green").text(d => d),
    - update => (), exit => ())`

# Data-Join



<http://cs.wellesley.edu/~mashups/pages/am5d3p1.html>

# 常见的CSV数据

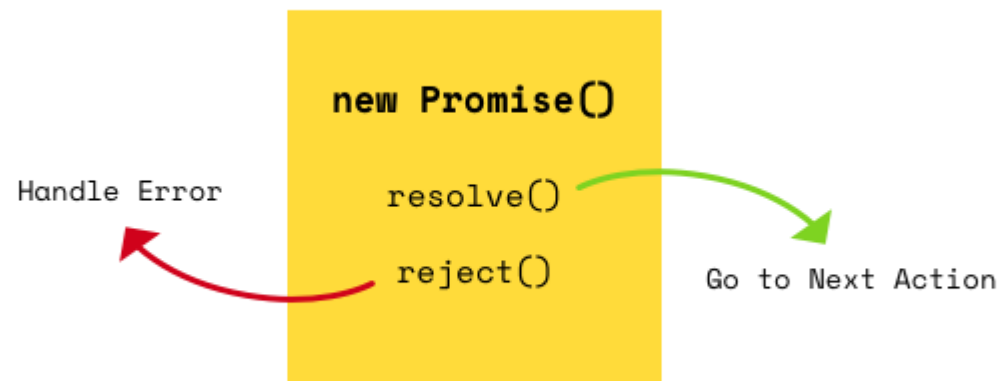
- 第一行为属性列表
- 后续每行对应一‘条’数据
- CSV本质上是纯文本，区别于EXCEL的格式

```
日期,省份,确诊人数,治愈人数,死亡人数,新增确诊,新增治愈,新增死亡,现有确诊,扩散曲线
2020/1/21,安徽,1,0,0,0,0,0,1,0
2020/1/22,安徽,9,0,0,8,0,0,9,8
2020/1/23,安徽,15,0,0,6,0,0,15,0.666666667
2020/1/24,安徽,39,0,0,24,0,0,39,1.6
2020/1/25,安徽,60,0,0,21,0,0,60,0.538461538
2020/1/26,安徽,70,0,0,10,0,0,70,0.166666667
2020/1/27,安徽,106,0,0,36,0,0,106,0.514285714
2020/1/28,安徽,152,2,0,46,2,0,150,0.433962264
2020/1/29,安徽,200,3,0,48,1,0,197,0.32
2020/1/30,安徽,237,3,0,37,0,0,234,0.187817259
2020/1/31,安徽,297,5,0,60,2,0,292,0.256410256
```

```
name,value
Shao-Kui,141
Bro-Yuan,135
Rui-Long,326
Xin,266
Xu-Qiang,210
Shi-Sheng,999
Jia-Ju,999
Xiang-Li,195
Yu,166
Yuan-Chen,143
Godness-Lan,130
Wei-Yu,130
Yun,130
Zheng, 366
Zu-Ming, 636
Jia-Hui, 663
Kai-Xiang, 666
```

# 数据的读取

- `d3.csv('path/to/data.csv').then( data => {…} )`
- `.csv`函数的返回值是一个JS的'Promise'对象
  - 'Promise'对象用于执行异步操作
- `.then(...)`的参数为一个函数，参数为`.csv(...)`的返回值
  - （不做要求）实际上在JavaScript异步中是一个resolve
- `d3.csv(...)`会正常向服务器请求数据，在请求并处理好之后，将结果扔给`.then(...)`中的回调函数



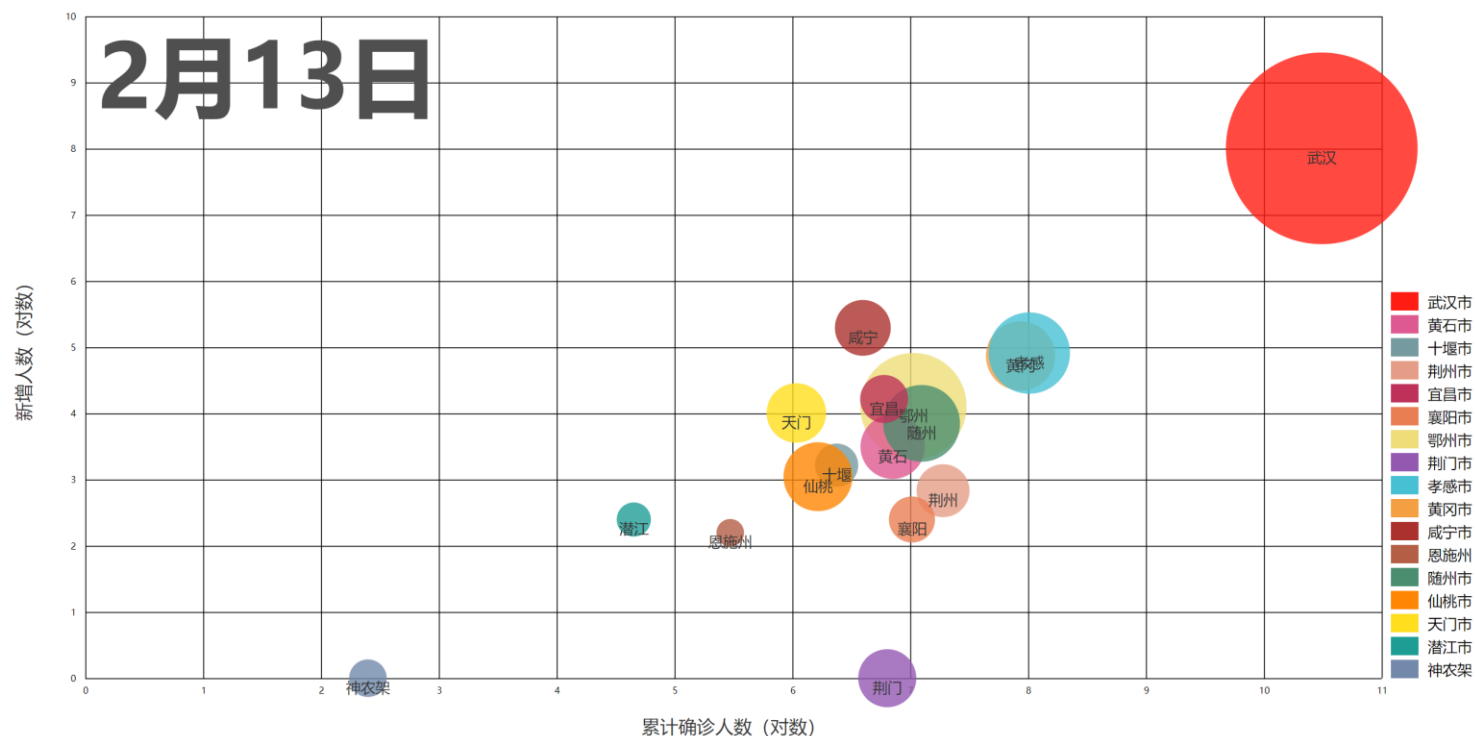


# 数据预处理

- 读取来的数据大多数都是文本（string）形式
  - （已经第三次加深大家印象了） 要记得使用 `+(...)` 及时转换数值数据
- 对于日期数据：
  - `d3.timeParse("%Y年%m月%d日")('2020年2月6日')`
- 对于日期数据的排序
  - `.sort(function(a,b){`
  - `return new Date(b) - new Date(a);`
  - `})`
- Map: `array.map( item => item['日期'] )`
- 数据预处理非常重要，是可视化（D3）编程之前必须经过的步骤

# D3实现动态散点图

- Code: <https://github.com/Shao-Kui/D3.js-Demos/blob/master/static/d3-tutorial/scatter.html>
- Code: <https://github.com/Shao-Kui/D3.js-Demos/blob/master/static/d3-tutorial/scatter-simple.html>
  - 简化的动态散点图，简化掉的部分思路类同，请大家参考未删减版思考！



# 数据与数据的预处理

- 湖北疫情数据: ./static/data/hubeinxt.csv

```
日期,省份,确诊人数,治愈人数,死亡人数,新增确诊,新增治愈,新增死亡,现有确诊,扩散曲线
2020/1/21,安徽,1,0,0,0,0,0,1,0
2020/1/22,安徽,9,0,0,8,0,0,9,8
2020/1/23,安徽,15,0,0,6,0,0,15,0.666666667
2020/1/24,安徽,39,0,0,24,0,0,39,1.6
2020/1/25,安徽,60,0,0,21,0,0,60,0.538461538
```

```
// 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(a) - new Date(b);
});
```

# 数据与数据的预处理 cont.

- alldates =
- 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
- 数据预处理的方法有很多，甚至有很多相关的库，欢迎大家分享更多更简洁、更高效的方式

```
// 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(a) - new Date(b);
});
```

# 复习：比例尺、坐标轴

- 实际动态可视化中，比例尺、坐标轴通常是定义（或渲染）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
  - \*Design Decision
- 仍需要定义margin

```
const margin = {top: 100, right: 120, bottom: 100, left: 120};
const innerWidth = width - margin.left - margin.right;
const innerHeight = height - margin.top - margin.bottom;
```

```
const renderinit = function(data){
  // Linear Scale: Data Space -> Screen Space
  xScale = d3.scaleLinear()
    .domain(d3.extent(data, xValue)) // "extent" returns the min and max values
    .range([0, innerWidth])
    .nice();

  // Introducing y-Scale;
  yScale = d3.scaleLinear()
    .domain(d3.extent(data, yValue).reverse()) // reverse the y-axis
    .range([0, innerHeight])
    .nice();

  // The reason of using group is that nothing is rendered at the origin (0,0)
  const g = svg.append('g')
    .attr('transform', `translate(${margin.left}, ${margin.top})`)
    .attr('id', 'maingroup');

  // Adding axes;
  const yAxis = d3.axisLeft(yScale)
    .tickSize(-innerWidth)
    // .tickFormat(d3.format('.2s'))
    .tickPadding(10); // .tickPadding is used to add padding between ticks
```

# 根据日期重新排布数据

- 我们需要根据日期动态更新数据
- `array.indexOf(item)`: 获取某个元素在数组中的位置
- `array.push(item)`: 在数组末尾加入某个元素

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

# 更新的主循环

- setInterval(myfunction, aduration): 每间隔aduration的时间，调用一次myfunction
- clearInterval( intervalId ): 根据setInterval返回的ID来结束掉某一个循环调用

```
// set the animation interval;
let c = 0;
intervalId = setInterval(function(){
    if(c >= alldates.length){
        console.log('time to close this animation');
        clearInterval(intervalId);
    }else{
        renderupdate(sequential[c]);
        c = c + 1;
    }
}, aduration);
```

# Data-Join: 使用Circle

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

  circleupdates = g.selectAll('circle').data(seq, d => d[keyHint]);

  circleenter = circleupdates.enter().append('circle')
    .attr('cy', (datum) => { return yScale(yValue(datum)) })
    .attr('cx', (datum) => { return xScale(xValue(datum)) })
    .attr('r', 10)
    .attr('fill', function(d,i) { return color[d[keyHint]] })
    .attr('opacity', .8);

  circleupdates.merge(circleenter)
    .transition().ease(d3.easeLinear).duration(aduration)
    .attr('cy', (datum) => { return yScale(yValue(datum)) })
    .attr('cx', (datum) => { return xScale(xValue(datum)) });
};
```



# Data-Join: 使用Circle cont.

- xValue 与 yValue :

- 数据通常包含多个属性，而可视化会经历人为选取、调整属性的过程，在更新（渲染）阶段调整选取的属性会造成代码的耦合，且增加选取、调整属性的时间
- 可以将属性选取本身封装成函数
  - `const xValue = d => { return Math.log(d['确诊人数'] + 1) };`
  - `const yValue = d => { return Math.log(d['新增确诊'] + 1) };`
  - `const xValue = d => d['确诊人数'];`
  - `const yValue = d => d['新增确诊'];`
  - ... ..

# D3实现更好的动态散点图?

- 右侧的图例?
- 让文字随着气泡一起移动?
- 左上角的时间?
- 让气泡的半径与数据绑定?

