

# D3.js

## 环境搭建与语法基础

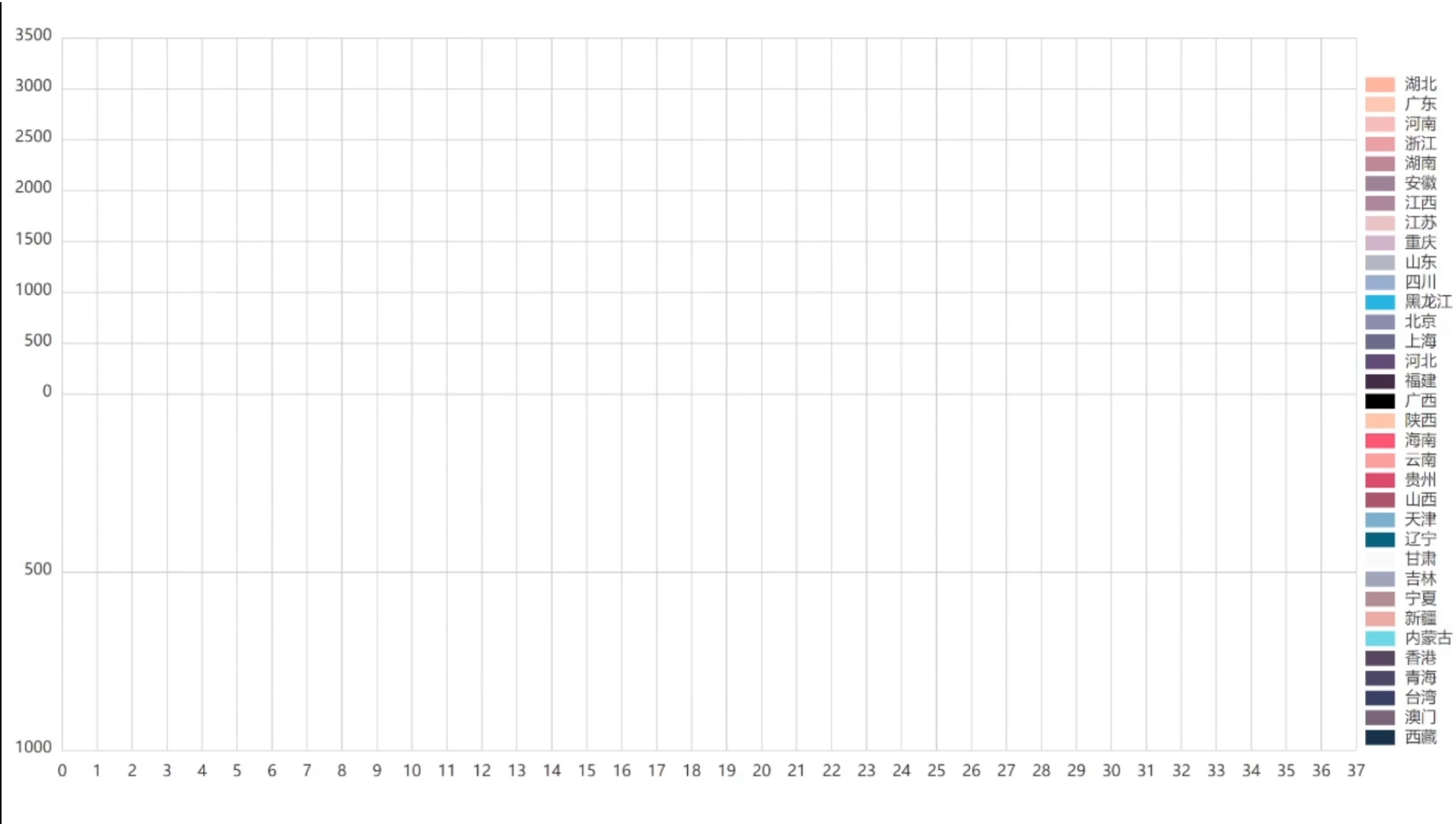
张松海 张少魁

清华大学 可视媒体研究中心

# Why D3.js?

- D3.js是现有最主流、社区规模最大、支持定制可视化效果的框架。
  - 有充足的官方文档、样例与社区支持。
  - 导航: [https://github.com/xswei/d3js\\_doc](https://github.com/xswei/d3js_doc)
  - 画廊: <https://observablehq.com/@d3/gallery>
  - 图元级别的‘定制’。
- Others:
  - Echarts: 不支持图元级别的定制, 但支持图表的混搭;
  - 本门课程不推荐使用: plotly
  - 本门课程禁用: matplotlib, seaborn

# 主题河流+柱状图



# 概览

- 本次D3.js课程中将会介绍：
- 语法基础。
- 可视化图表的绘制：
  - 可视化基本图表制作：柱状图、折线图、饼图、散点图、力导图等。
  - CSV、JSON、层级数据、网络数据等。
  - 定制化。
- 动画。
- 交互。
- D3.js中的其他常用接口。
- D3.js讲解面向所有专业的同学，尽可能不要求编程基础。

# 前言

- D3: **Data-Driven Documents**
  - 通过D3提供的接口来基于数据操控文档的各个图元。
- 先修条件与本节课的内容：
  - HTML与SVG: 超文本标记语言与其中的矢量图。
  - 配置一个Web开发环境, 有Web开发经验的同学完全可按照自己习惯的技术路线。
  - JavaScript简短介绍, 其余涉及到D3的语法会在之后编程的过程中讨论。
- D3.js的讲解形式：
  - 以接口为核心, 接口  $\approx$  D3.js提供的函数调用等。
  - 对于每个独立接口进行讨论: 输入、输出、用法等。
  - 通过编程实例(code)来讨论每个接口的应用。
- 张少魁 [zhangsk18@mails.tsinghua.edu.cn](mailto:zhangsk18@mails.tsinghua.edu.cn)

# 课程参考资源

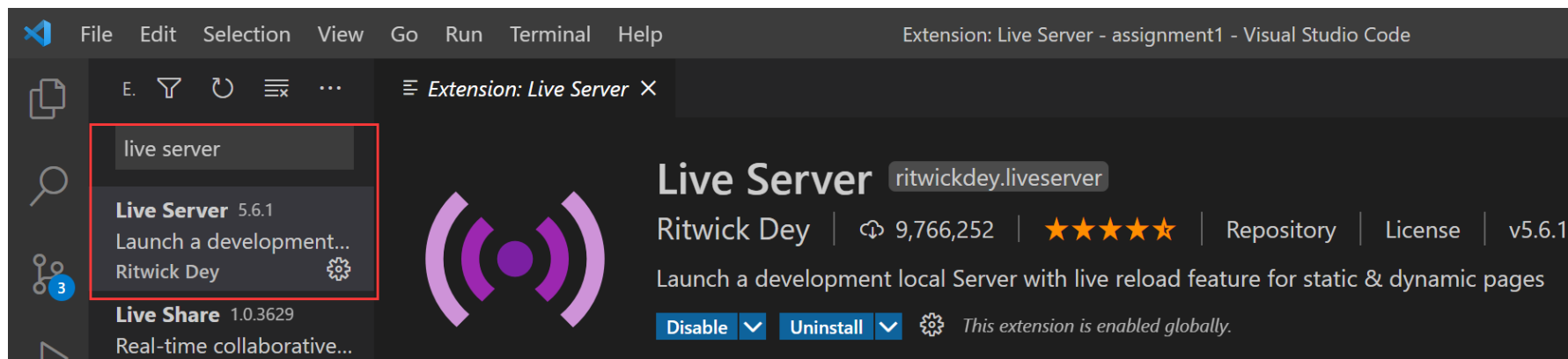
- 教学参考书：数据可视化（第二版），陈为 沈则潜 陶煜波 等，电子工业出版社，
- 教学主要参考资料：IEEE VIS会议论文和IEEE TVCG期刊论文集
- D3编程参考资料：
  - <https://d3js.org/> （官方网站，包括文档、样例）
  - <https://developer.mozilla.org/zh-CN/docs/Web/SVG/Attribute> （SVG属性表）
  - <https://github.com/d3/d3/wiki/Gallery> （官方样例的仓库）
  - <https://observablehq.com/@d3/gallery> （官方样例的仓库）
  - [https://github.com/xswei/d3js\\_doc](https://github.com/xswei/d3js_doc) （d3.js资源汇总，包括示例、书籍、API文档等）

# 配置Web环境

- 请大家解压网络学堂中的‘Class1.zip’;
- 解压后, 尝试运行‘demo.html’中的可视化效果;
- \* **运行需要配置一个简易的Web环境!**
- 若有Web开发经验, 根据自己习惯的框架运行即可。
- 若无Web开发经验, 参考接下来的配置指南。

# 配置Web环境

- D3.js基于JavaScript，常用于在Web前端操控HTML中的元素。
- 对于无Web开发经验的同学：
  - 下载VSCODE: <https://code.visualstudio.com/>
  - 安装Live Server:  
<https://marketplace.visualstudio.com/items?itemName=ritwickdey.LiveServer>



- 对于有Web开发经验的同学，可直接应用自己常用&熟悉的框架
  - 如Python Simple HTTP Server（不推荐）、Node.js Simple HTTP Server、Flask、Express等。

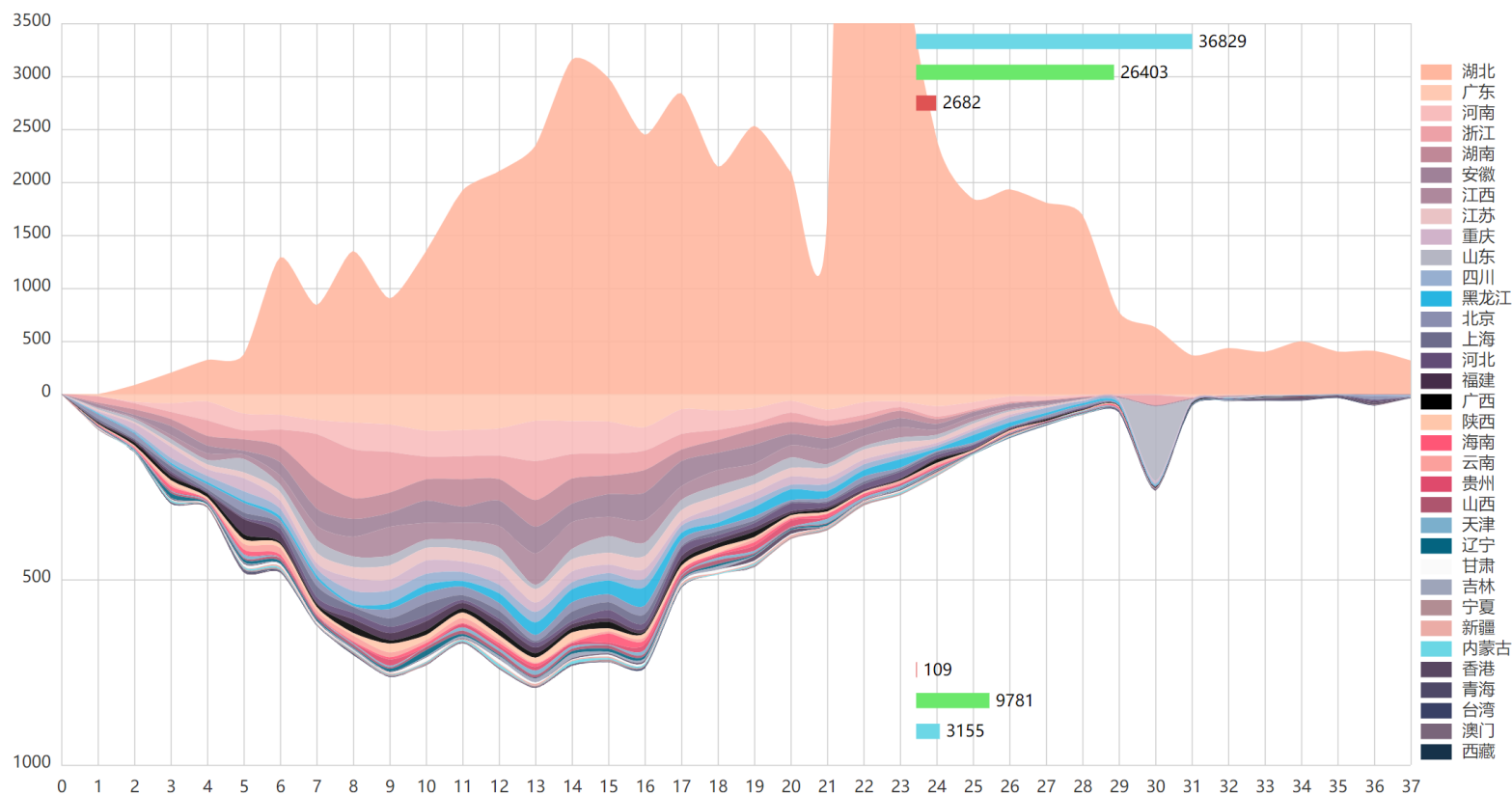


# 配置Web环境



# 配置Web环境

- 如果能成功打开如下demo，则已经完全搭建好Web环境。



# 配置Web环境

- 前端展示使用常用的浏览器：
  - 强烈推荐Chrome
  - 强烈不推荐IE
- 课程的代码讲解会使用VSCode，但文本编辑器不做任何限制：
  - Sublime Text (不免费), Atom等均可。
- **作业**检查不参考任何Web环境相关的内容，会只关注可视化效果。
  - 请勿使用过于复杂的Web后端，除非需要一些特殊的逻辑。

# HTML

- 超文本标记语言
- HyperText Markup Language
- HTML不是编程语言!
  - 由多个标签构成的标记语言
  - 通过浏览器来解析
- HTML用来描述我们常见的网页
- HTML包含大量元素（标签）
  - 元素与元素的类别不同，如矩形、直线、文本、圆……
  - 元素包含属性，如位置、大小、色调、文本风格……

```
<!DOCTYPE html>
<html>
  <head>
    <title>Data Visualization!</title>
  </head>
  <body>
    Hello World!
  </body>
</html>
```

# HTML - Tags

- 本课程只需要大家了解以下标签即可!
- `<html>`: 最外层的主标签, 每个HTML文件都需要有!
- `<head>`: 标题内容, 包含HTML的文件链接、标题等
- `<body>`: HTML的主体, 包括各种其中的各种元素
- `<title>`: 标题 (显示在浏览器的标签栏)
- `<script>`: JavaScript脚本(代码)或脚本的链接
  - **D3.js**的编程主要写在此标签中
- `<svg>`: 对于**D3(本讲解)**最为重要的标签, 主要操作的对象(画布)
- 所有标签都需要以`</tag>`结束

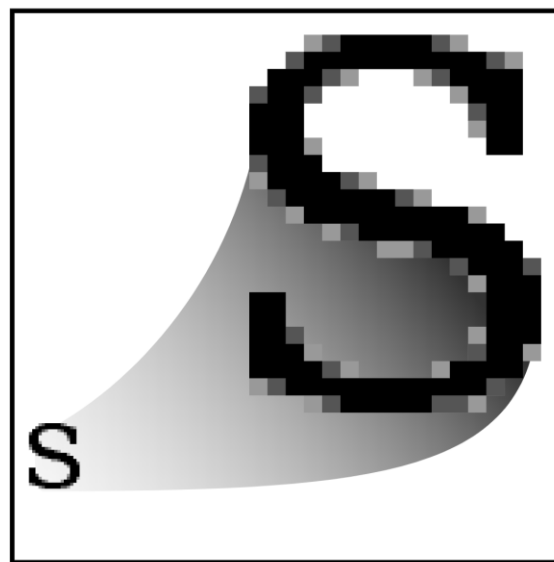
# HTML - 导入D3.js

- D3.js作为JavaScript的外库，必须先将其导入，如：
  - Python的import， C/C++的include、
  - Java的import、 node.js的require·····
- 通过Script标签导入
  - 直接通过互联网链接
    - <https://d3js.org/d3.v5.min.js>
  - 通过本地服务器链接（推荐）
    - ./d3.min.js
  - 通过unpkg链接
    - <https://unpkg.com/browse/d3@5.15.0/dist/d3.js>
  - 尽可能使用本地的d3.min.js库。

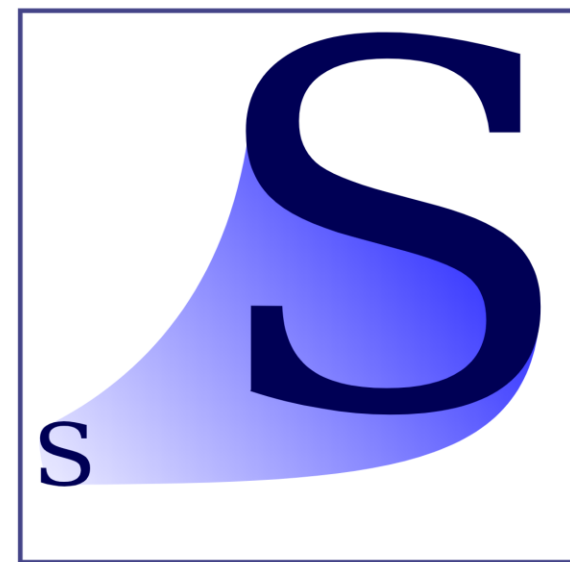
```
<!DOCTYPE html>
<html>
  <head>
    <title>Scatter</title>
    <link rel="stylesheet" href="/static/css/nCov.css">
    <script src="/static/js/d3.min.js"></script>
  </head>
  <body style="text-align: center">
```

# SVG – 可缩放矢量模型

- SVG  $\approx$  D3用来绘制的‘画布’。
  - 可缩放矢量图形（英語：Scalable Vector Graphics, SVG）
  - SVG是D3.js主要操作的对象
    - `const svg = d3.select('svg');`
    - D3.js获取svg对象
  - SVG同时也是一个容器，用于包含画在上面的各个图元。
  - SVG作为矢量图，不会随着图片的缩放而发生失真；
- <svg> 画布



**Raster**  
.jpeg .gif .png



**Vector**  
.svg



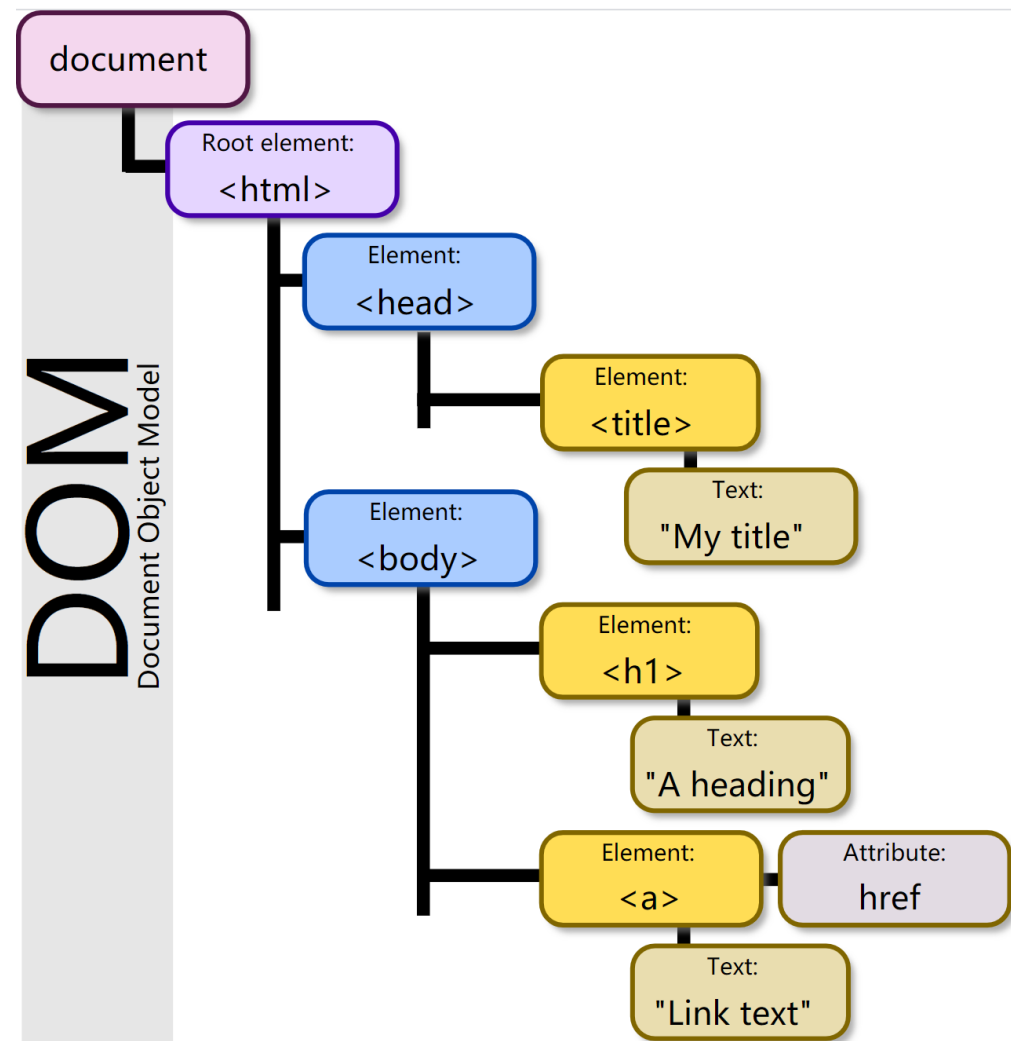
# 引入SVG

```
<body>
<svg height='200' width='200' style='display: block; margin: 0 auto;'>
  <g transform='translate(0,60)'>
    <rect width=100 height=100 fill='#EEEEEE' />
    <circle r=15 fill='#72bf67' cx=25 cy=30 />
    <circle r=15 fill='rgb(100, 149, 237)' cx=75 cy=30 />
    <g transform='translate(15,60) rotate(10)'>
      <path d="M0,0 A40,40 10 0,0 65,0" fill='none' stroke='gray' stroke-width=5 />
    </g>
  </g>
</svg>
</body>
```



# HTML – 文档对象模型

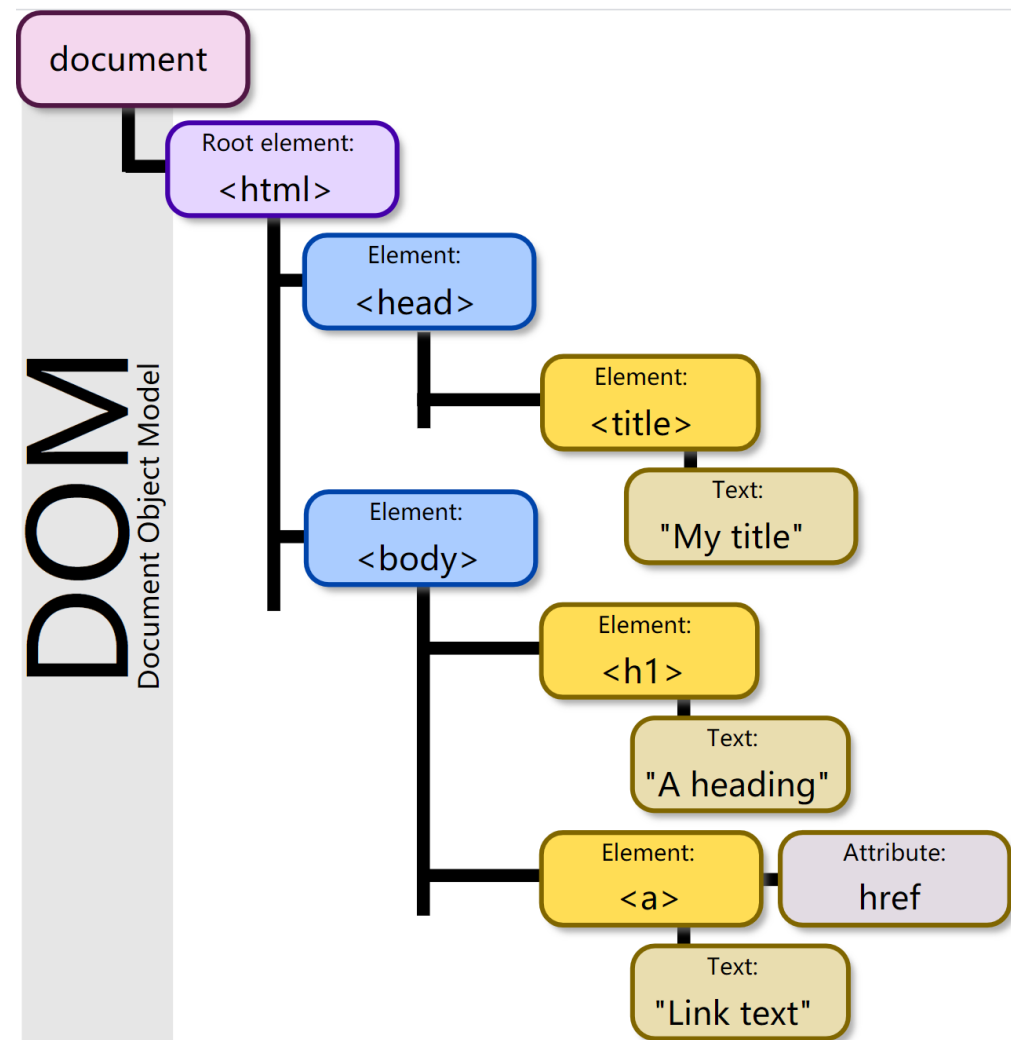
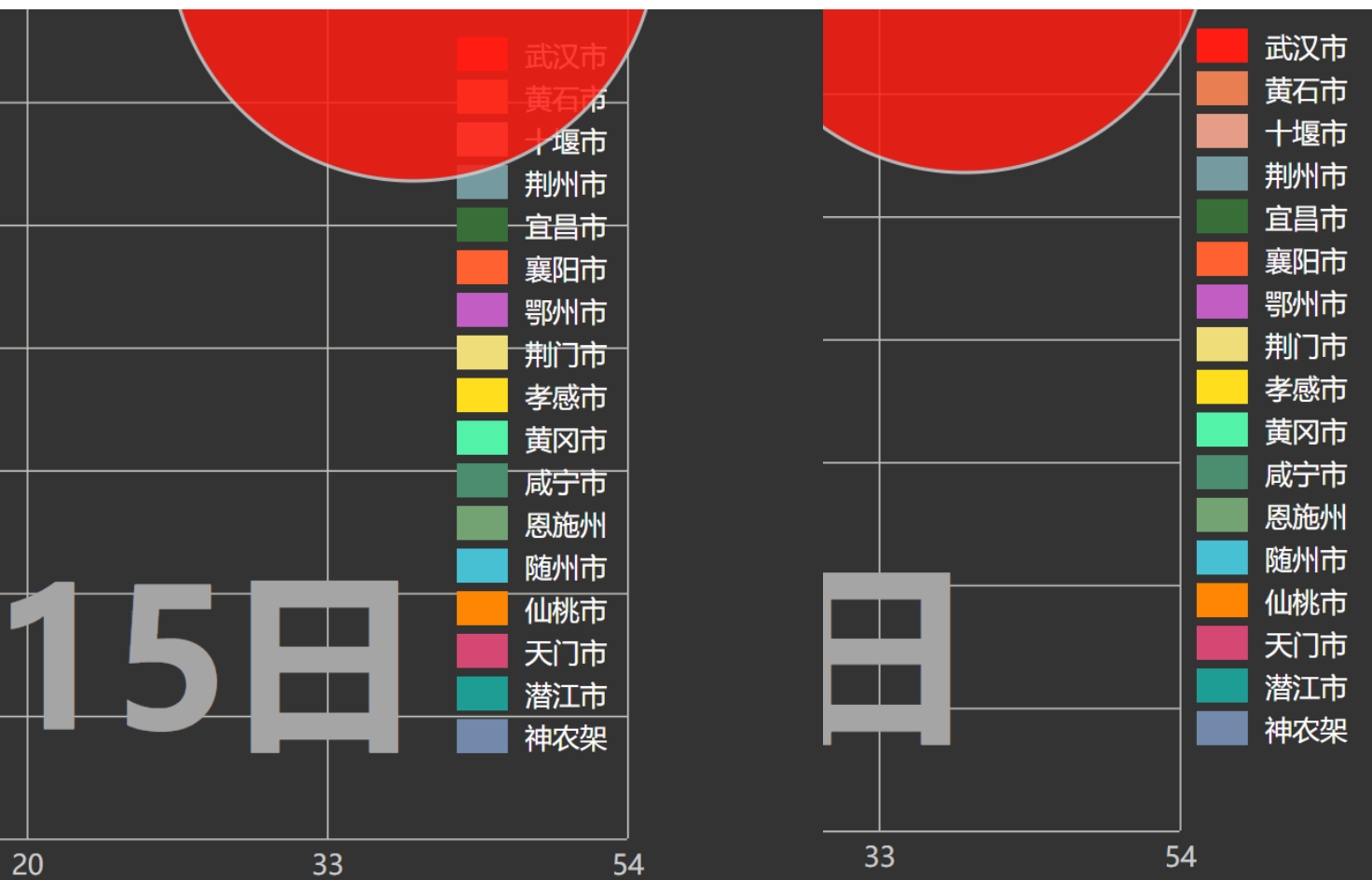
- HTML -> DOM
- DOM -> Document Object Model
- 对于根节点的操作会影响到子节点;
- 最常用的父节点<svg>中的<g>
  - Axis可封装成一个group
  - Legend (图例) 可封装成一个group
- Data-Driven **Document**
- **Document** Object Model



<https://en.wikipedia.org/wiki/File:DOM-model.svg>

# HTML – 文档对象模型

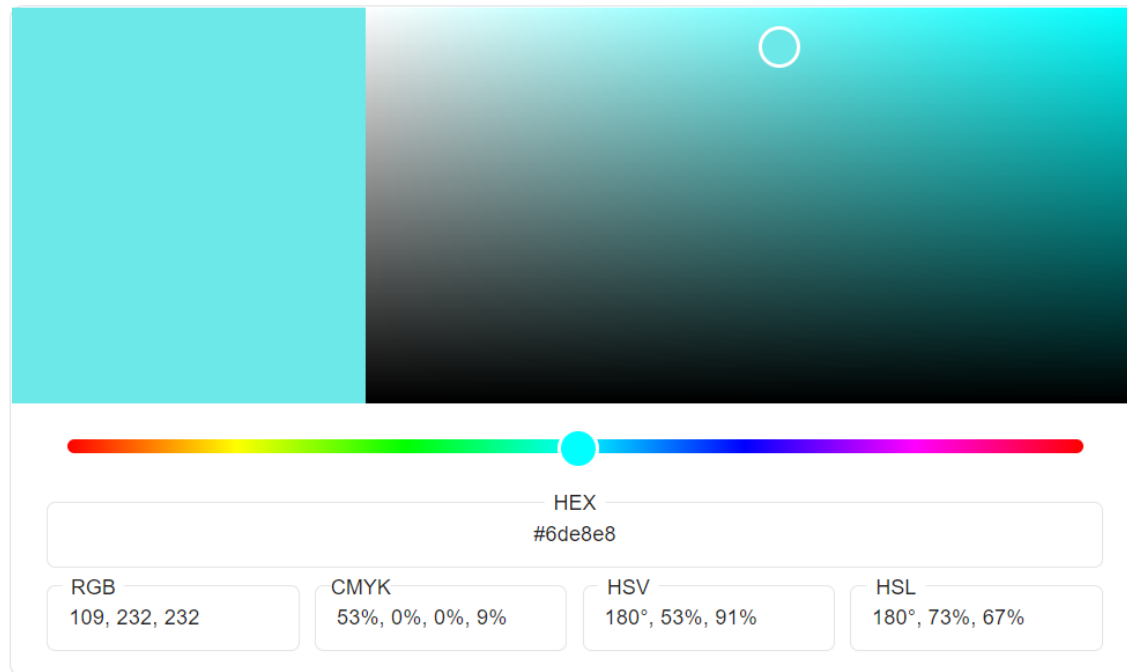
- 对于根节点的操作会影响到子节点;



<https://en.wikipedia.org/wiki/File:DOM-model.svg>

# HTML

- 代码示例:



```
<svg style='display: block; margin: 0 auto;'>
  <g transform='translate(0,60)'>
    <rect width=100 height=100 fill='#EEEEEE' />
    <circle r=15 fill='#72bf67' cx=25 cy=30 />
    <circle r=15 fill='rgb(100, 149, 237)' cx=75 cy=30 />
    <g transform='translate(15,60) rotate(10)'>
      <path d="M0,0 A40,40 10 0,0 65,0" fill='none' stroke='gray' stroke-width=5 />
    </g>
  </g>
</svg>
```

# JavaScript

- Web开发语言
  - 同时支持客户端和服务端，本课程主要集中在客户端
  - 前端通常由浏览器（如chrome）解析
- 解释型的编程语言
  - 不需要编译
  - E.g., C, C++需要经过编译才可执行
- JavaScript 是可插入 HTML 页面的编程代码
  - D3.js -> JavaScript
- JavaScript vs. Java
  - 两者没有直接关系！
- JavaScript vs. C(++)
  - 二者有很多类似，if条件语句、switch语句、while循环、do-while循环等

# JavaScript – 主要语法特性

- 变量声明不需指定类型 `int double function`
  - `let, var, const`
- 运算操作基本等同于C、C++、JAVA等语言
  - `+, -, *, /, %, ... ..`
- 函数定义同样不需要指定类型
  - `function abc(a){ return a + 5; }`
  - `let f = datum => datum.value;`
  - `const p = function(a, b) { return a + b; }`
  - `let myFunction = (a, b) => a + b`
  - `let f = (d, i) => { console.log(d); return d + i; }`

# JavaScript – 把函数作为参数

- 一个变量可以是一个函数
  - `const myFunction = function(a, b) { return a + b; }`
  - 类似与C/C++的函数指针
- 回调（Callback）：
  - JavaScript脚本中常见把函数作为变量输入
  - 用于实现异步编程
    - `setTimeout( function () {  
    console.log('hello world!')  
}, 1000);`
- 在D3中存在大量类似的调用：
  - 将函数作为参数给图元
  - 如：为每个数据点指定不同的颜色

# JavaScript – D3中的常用接口

- 模板字符串：
  - `let a = 10;`
  - `let myString = `abc-${a}`;` (myString最终为'abc-10')
- 数组 `a = [1, 2, 3]`
- 对象 `a = {name: 'Shao-Kui', age: 24.3, lab: 'cscg'}`
  - D3数据可视化中常见对象数组，如：
    - `a = [{name: 'Shao-Kui', age: 25.3, dept: 'cs'},`
    - `{name: 'Wen-Yang', age: 23, dept: 'cs'},`
    - `{name: 'Yuan', age: 29, dept: 'cs'}]`
- 数组的排序 `a.sort()`
  - 可加入回调函数来替代缺省的排序方案，如对日期排序
  - `a.sort(function(a,b){ return new Date(b.date) - new Date(a.date); })`
- 数组的查询 `a.find( d => d.name === 'Wen-Yang')`
- 把字符串转换成数值： `+( '3.14')`
- D3.js经常读取**CSV、JSON等**文件，会涉及大量的数组、对象的操作！

# D3语法基础概览

- 使用D3获取、修改、增加与删除节点（图元）。
- 数据的读取 – CSV。
- D3.js的数值计算。
- 比例尺：
  - 线性比例尺（Linear Scale）。
  - “条带”比例尺（Band Scale）。
- 坐标轴的绘制：
  - Margin。
- Data-Join基础。
- 基于D3的基础语法与Data-Join绘制柱状图。

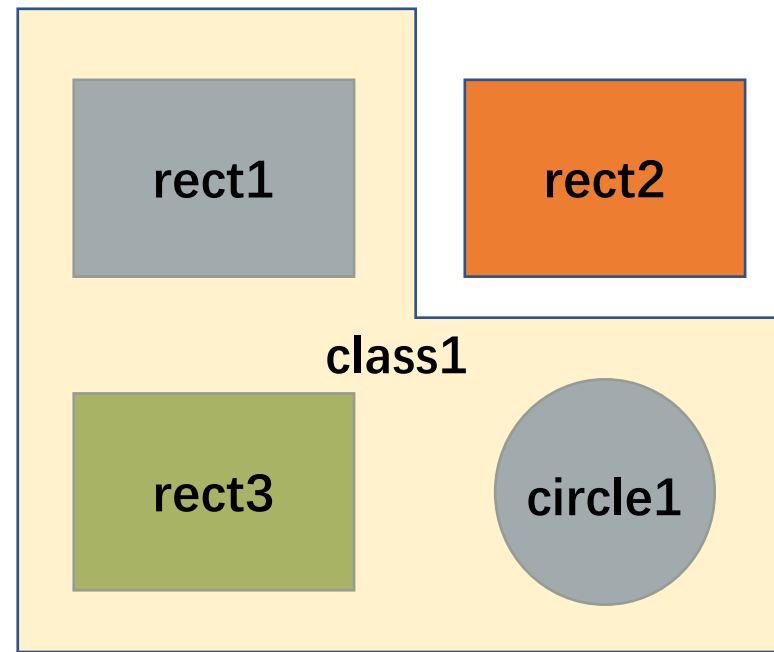


# 元素（标签）的标识

- 当我们在一个同学群体中（比如微信群）对某些同学发出通知时：
  - 请**学号为2020123456的同学**在东主楼集合；
  - 请**计算机系研一**的同学在东主楼集合；
- 在一个群体中，索引个体：
  - 通过唯一的标识索引到唯一的个体
  - 通过共同点索引到一批个体

# 元素（标签）的标识

- 操作元素首先需要知道元素的标识
  - 即要得到已有或已经创建的元素
- 元素的ID
  - 可以唯一找到元素的标识符
- 元素的Class
  - 人为赋予的“类别”可以标记元素的集合，其中的元素标签可以不相同！
- 元素的标签
  - HTML自带的标签名称，可以找到一批同类别的物体，如所有的“矩形”
  - 使用自带的标签往往难以直接索引到目标元素
  - `<rect>`, `<circle>`, `<svg>`等

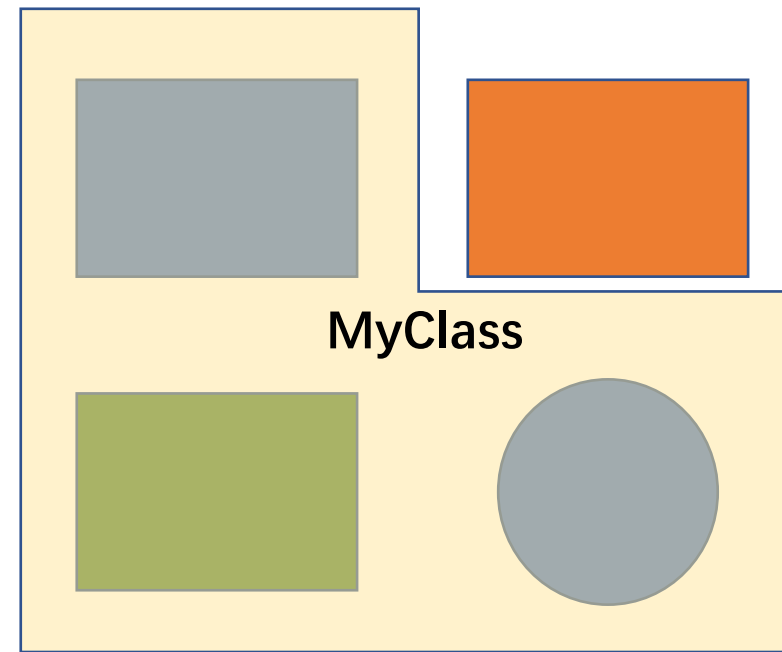


# 元素（标签）的标识

```
<g id='maingroup' transform='translate(100, 100)'>
  <circle id='circle1'
stroke='black' r='66' fill='#4B8E6F' cx='0'></circle>
  <rect id='rect1' class='class2'
stroke='black' height='200' width='66' fill='#ff8603' x='100' y='-100'></rect>
  <rect id='rect2' class='class1'
stroke='black' height='200' width='66' fill='#ffde1d' x='200' y='-100'></rect>
  <rect id='rect3' class='class1'
stroke='black' height='200' width='66' fill='#7289AB' x='300' y='-100'></rect>
  <text id='myText' class='class2'
stroke='yellow' font-size='2em' x='400' fill='#1e9d95'>Hey D3! </text>
</g>
```

# 使用D3查询SVG

- `d3.select(...)`
  - `d3.select('#rect1')`
  - 查询ID为'rect1'的元素
  - #表示后面的字符串是一个ID
  - 只找一个，若有重名也只返回第一个
- `d3.selectAll(...)`
  - `d3.selectAll('.class1')`
  - 查询所有class是'class1'的元素
  - `d3.selectAll('rect')`
  - 查询所有标签是'rect'的元素（rect为SVG中的矩形标签）
  - 有多少返回多少
  - 可配合Data-Join选取'不存在'的图元
- ID前加'#'， Class前加'.'， 标签名前不加符号。



# 使用D3查询SVG

- 基于层级的查询：
  - `d3.select('#maingroup rect')`
  - `d3.selectAll('.tick text')`
  - `d3.selectAll('#secondgroup rect')`
- 如： `'#secondgroup rect'`
  - 首先会找到id为secondgroup的标签
  - 进一步找到secondgroup的子标签中是rect的
  - 仍然是对rect做查询，只是结果通过父标签做了筛选

# 使用D3查询SVG

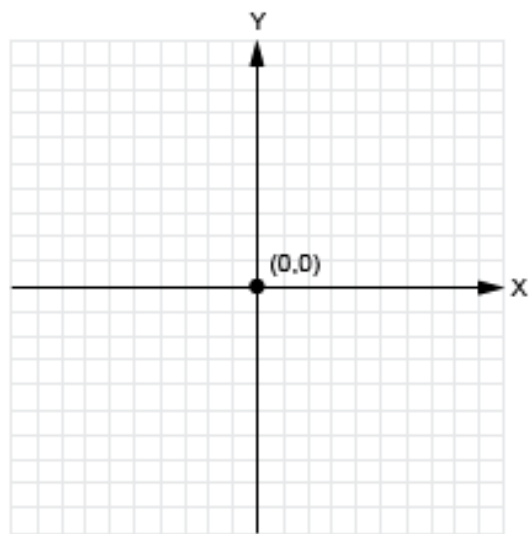
- `d3.select(...)`也可用于查询类别，如
    - `d3.select('.class1')`
    - 但只会返回找到的第一个元素
  - 因此对于class、标签名称的查询建议使用`d3.selectAll`
  - 对于特定某一个元素的查询建议使用`d3.select`
- 
- CSCG研一的同学如果只有一名，则同样可以唯一索引
  - ID与Class在编程上可自由决定如何使用

# 使用D3设置SVG中的属性

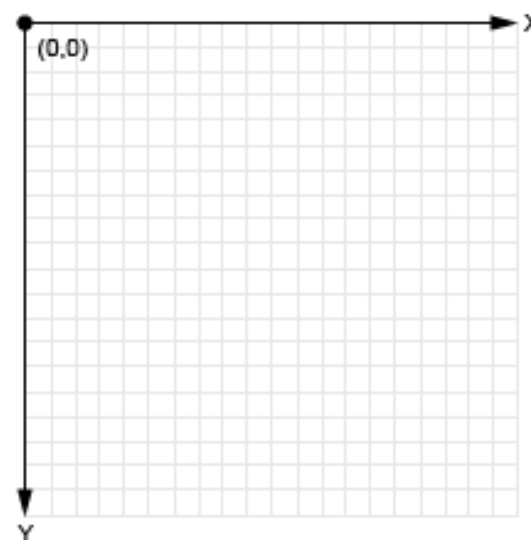
- 常见的属性
  - id, class (特殊的属性, 可以使用.attr设置)
  - x, y, cx, cy (注意屏幕的坐标系! 见下页)
  - fill, stroke
  - height, width, r (圆的半径)
  - transform -> translate, rotate, scale
- SVG的属性非常多, 且属性的取值 **范围&类型** 各不同
  - tip1: 尽可能记住一些常见的属性, 以提高编程速度
  - tip2: 遇到不认识or想要设置某个属性, 一定要查阅
    - <https://developer.mozilla.org/zh-CN/docs/Web/SVG/Attribute>
- `<rect id='rect3' class='class1'`
- `stroke='black' height='200' width='66' fill='#7289AB' x='300' y='-100'>`

# 使用D3设置SVG中的属性

- 屏幕空间的坐标系与常见坐标系不同
  - 左上方为原点
  - Y、X分别垂直向下、水平向右



1. Cartesian coordinate space



2. Canvas coordinate space



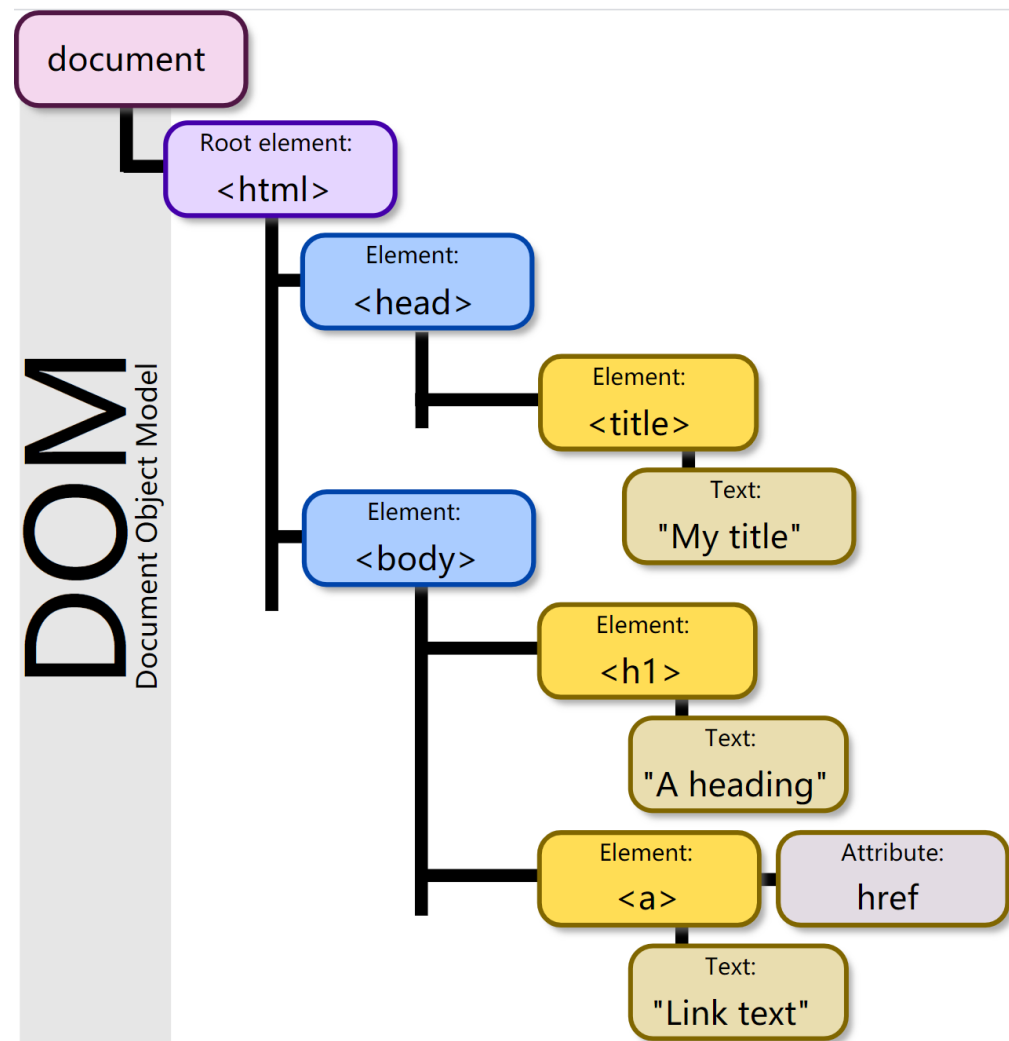
# element.attr(...)

- 设置元素的属性: element.attr('attr\_name', 'attr\_value')
  - 两个参数: 属性名、设置的值
  - rect1.attr('y', '100')
  - d3.select('#rect1').attr('y', '100')
- 获取元素的属性: element.attr('attr\_name')
  - 一个参数: 属性名
- 链式调用
  - selection.attr(...).attr(...).attr(...)
  - .attr(...)返回的是选择的图元本身

# 修改整组属性

- DOM
  - 父节点属性会影响子节点
  - 子节点属性会相对于父节点
- 下方代码可以直接移动组内所有元素
  - `d3.select('#maingroup')`
  - `.attr('transform', 'translate(200, 100)')`

```
<g id='maingroup' transform='translate(100, 100)'>
  <circle id='circle1'
    stroke='black' r='66' fill='#4B8E6F' cx='0'></circle>
  <rect id='rect1' class='class2'
    stroke='black' height='200' width='66' fill='#ff8603' x='100' y='-100'></rect>
  <rect id='rect2' class='class1'
    stroke='black' height='200' width='66' fill='#ffde1d' x='200' y='-100'></rect>
  <rect id='rect3' class='class1'
    stroke='black' height='200' width='66' fill='#7289AB' x='300' y='-100'></rect>
  <text id='myText' class='class2'
    stroke='yellow' font-size='2em' x='400' fill='#1e9d95'>Hey D3! </text>
</g>
```



# 使用D3 添加&删除 SVG元素

- `element.append(...)`
  - `const myRect = svg.append('rect');`
  - `const myRect = d3.select('#mainsvg').append('rect')`
  - `const myRect = d3.select('#mainsvg').append('rect').attr('x', '100')`
- D3的链式添加（调用）
  - `const myRect = d3.select('#mainsvg').append('g').attr('id', 'maingroup')`
  - `.append('rect').attr('fill', 'yellow')`
- `element.remove()`
  - 请小心使用
  - 会移除整个标签
- Tip:在debug的过程中可以考虑使用'opacity'属性hack出移除的效果
  - `element.attr('opacity', '0')`

# 操控SVG

- 代码调用示例:

```
<body>
  <svg width="1600" height="800" id="mainsvg" class="svgs" style='display: block; margin: 0 auto;'>
    <g id='maingroup' transform='translate(100, 100)'>
      <circle id='circle1'
        stroke='black' r='66' fill='#4B8E6F' cx='0'></circle>
      <rect id='rect1' class='class2'
        stroke='black' height='200' width='66' fill='#ff8603' x='100' y='-100'></rect>
      <rect id='rect2' class='class1'
        stroke='black' height='200' width='66' fill='#ffde1d' x='200' y='-100'></rect>
      <rect id='rect3' class='class1'
        stroke='black' height='200' width='66' fill='#7289AB' x='300' y='-100'></rect>
      <text id='myText' class='class2'
        stroke='yellow' font-size='2em' x='400' fill='#1e9d95'>Hey D3! </text>
    </g>
    <g id='secondgroup' transform='translate(550, 100)'>
      <rect id='rect4' class='class2'
        stroke='black' height='200' width='66' fill='#DD6B66' x='100' y='-100'></rect>
      <rect id='rect5' class='class1'
        stroke='black' height='200' width='66' fill='#759AA0' x='200' y='-100'></rect>
      <rect id='rect6' class='class1'
        stroke='black' height='200' width='66' fill='#E69D87' x='300' y='-100'></rect>
    </g>
  </svg>
  <script>
    d3.select('#rect3').attr('height', '300');
    d3.selectAll('.class1').attr('fill', 'steelblue');
    d3.selectAll('text').attr('y', '300').attr('font-size', '3em');
    d3.select('#maingroup').append('circle')
      .attr('cx', 200).attr('cy', 200).attr('r', '50').attr('fill', 'green');
    d3.select('#rect6').remove();
  </script>
</body>
```

# 数据的读取 – 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
2020/2/1,安徽,340,6,0,43,1,0,334,0.147260274
2020/2/2,安徽,408,8,0,68,2,0,400,0.203592814
```

```
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 – CSV数据

- d3.csv(...):
  - 读取目标路径下的某一个CSV文件。
  - e.g., d3.csv('static/data/hello.csv');
- d3.csv是一个JavaScript异步函数：
  - 不可以直接获得它的返回值，如：
  - let myData = d3.csv('static/data/hello.csv'); ✕
- d3.csv('path/to/data.csv').then( data => { // ‘数据读取后的代码逻辑’ } )
  - 要通过.then( data => { ... } )的方式来获得读取后的数据。
  - then(...)中的 ‘data => { ... }’ 是一个函数。
  - 此函数接受的输入(参数)，即data，为读取后的数据。
- JavaScript异步机制（下述不做要求）：
  - d3.csv作为异步函数，即便没有读取好数据，后面的代码也会继续执行。
  - d3.csv被调用后，其返回值是一个JavaScript的‘Promise’对象(object)。
  - Promise‘询问’：数据读取好了之后要做什么？‘做什么’即对应.then()中函数的内容。

# 数据的读取 – CSV数据

- 代码调用示例:

- 读取后的数据格式(接口)与原本的CSV结构不同。

```
d3.csv('platform_globalsale.csv').then(data => {  
  console.log(data);  
});
```

platform\_globalsale

Wii,926.71

NES,251.07

GB,255.45

DS,822.49

X360,979.96

PS3,957.84

PS2,1255.64

SNES,200.05


GBA,318.5

3DS,247.46

PS4,278.1

N64,218.88

PS,730.66

▼ Array(31) 

▶ 0: {platform: "Wii", globalsale: "926.71"}

▶ 1: {platform: "NES", globalsale: "251.07"}

▶ 2: {platform: "GB", globalsale: "255.45"}

▶ 3: {platform: "DS", globalsale: "822.49"}

▶ 4: {platform: "X360", globalsale: "979.96"}

▶ 5: {platform: "PS3", globalsale: "957.84"}

▶ 6: {platform: "PS2", globalsale: "1255.64"}

▶ 7: {platform: "SNES", globalsale: "200.05"}

▶ 8: {platform: "GBA", globalsale: "318.5"}

▶ 9: {platform: "3DS", globalsale: "247.46"}

▶ 10: {platform: "PS4", globalsale: "278.1"}

▶ 11: {platform: "N64", globalsale: "218.88"}

▶ 12: {platform: "PS", globalsale: "730.66"}

▶ 13: {platform: "XB", globalsale: "258.26"}

▶ 14: {platform: "PC", globalsale: "258.82"}

# D3.js的数值计算

- 数据可视化常涉及对数据的处理与计算：
  - 下述三个接口分别用于计算数组的最大值、最小值、[最小值, 最大值]。
- `d3.max(array)`
  - 返回数组中的最大值。
  - e.g., `d3.max([5,4,6,1,8,16,9]) // 16`
- `d3.min(array)`
  - 返回数组中的最小值。
  - `d3.min([5,4,6,1,8,16,9]) // 1`
- `d3.extent(array)`
  - 同时返回最小值与最大值, 以数组的形式, 即[最小值, 最大值]。
  - `d3.extent([5,4,6,1,8,16,9]) // [1, 16]`

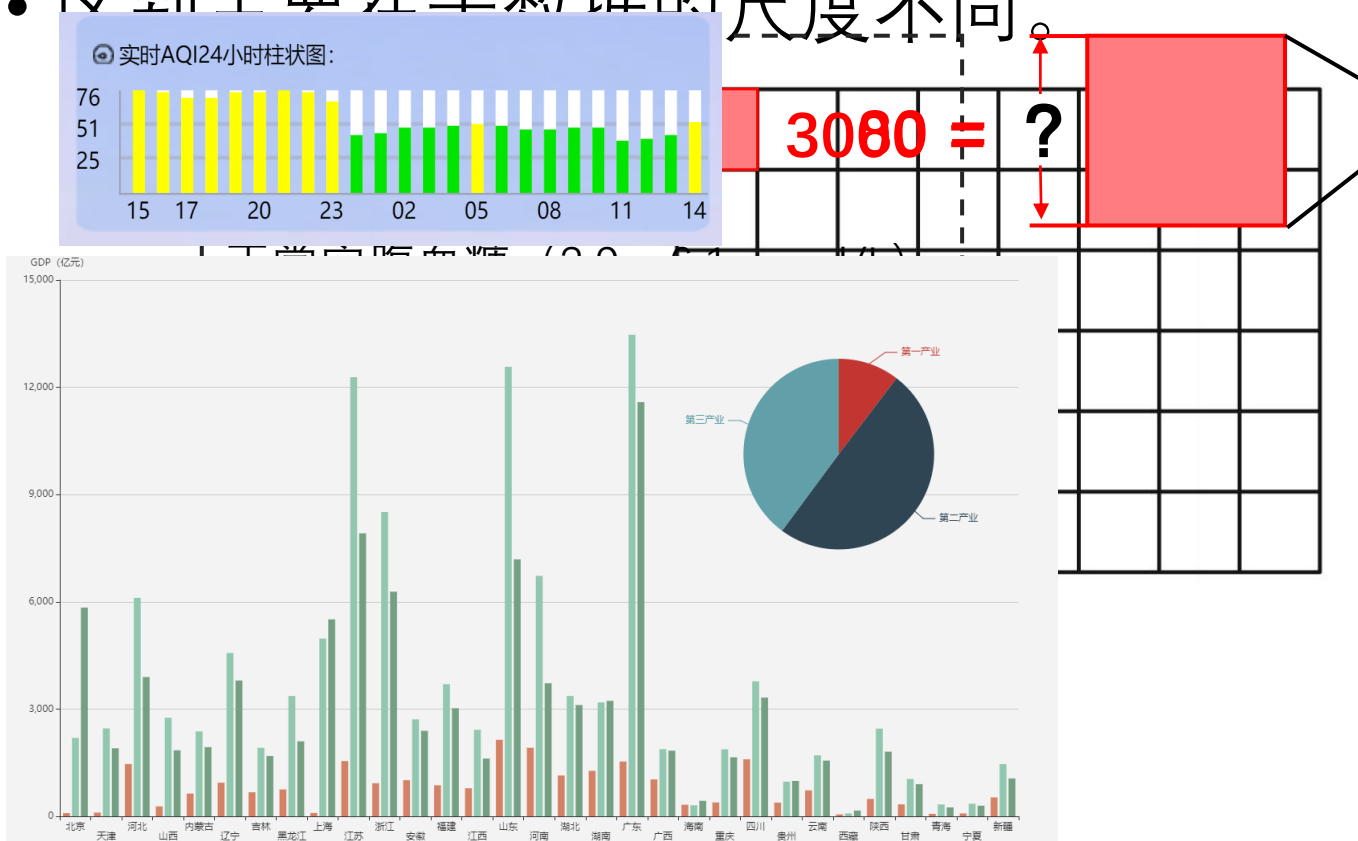


# D3.js的数值计算

- 数组中的内容可以是任意对象：
  - 每个对象可能包含多个属性。
  - 具体取哪个属性的最大值通过回调函数来提示d3.max、d3.min与d3.extent。
- e.g.,
  - `let a = [`  
`{name: 'Shao-Kui', age:25, height: 176}, {name:'Wen-Yang', age:24, height: 180},`  
`{name:'Liang Yuan', age: 29, height: 172}, {name:'Wei-Yu', age:23, height: 173}]`
    - `d3.max(a, d => d.age) // 29`
    - `d3.max(a, d => d.height) // 180`
    - `d3.extent(a, d => d.height) // [172, 180]`
    - `d3.min(a, d => d.age) // 23`

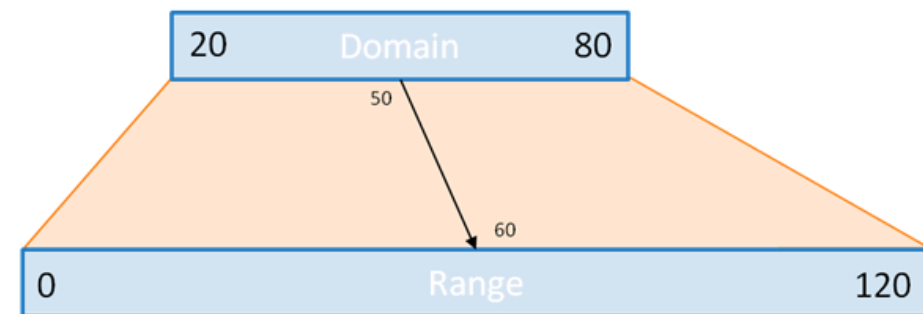
# 比例尺

- 比例尺用于把实际**数据空间**映射到**屏幕(画布)空间**，即两个空间的转化。
- 常用于映射数据and创建坐标轴。
- 区别主要在于数据的尺度不同。

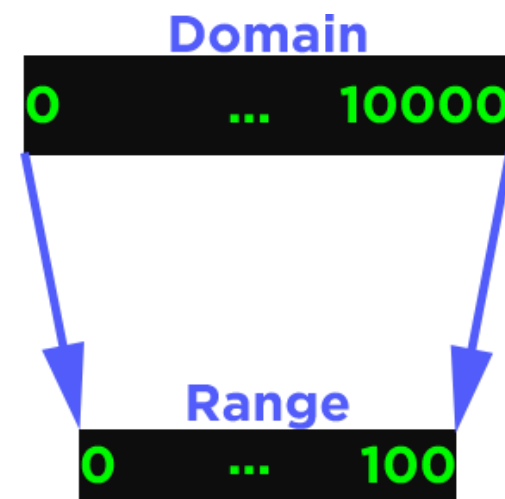


# Scale - Linear

- `d3.scaleLinear()`:
  - 定义一个线性比例尺，返回的是一个**函数**。
  - e.g., `let scale = d3.scaleLinear();` // scale为函数
- `scale.domain([min_d, max_d]).range([min, max])`:
  - 设置比例尺的**定义域与值域**。
  - 线性比例尺的定义域和值域都是连续的(Continuous)，需分别给出最大值与最小值。
  - e.g., `const scale = d3.scaleLinear().domain([20, 80]).range([0, 120]);`
- 比例尺本质上是一个**函数**:
  - `scale(20) // 0`
  - `scale(50) // 60`
- 常结合读取的数据与`d3.max`等接口连用:
  - `const xScale = d3.scaleLinear()  
 .domain([0, d3.max(data, d => d.value)])  
 .range([0, innerWidth]);`



(a)



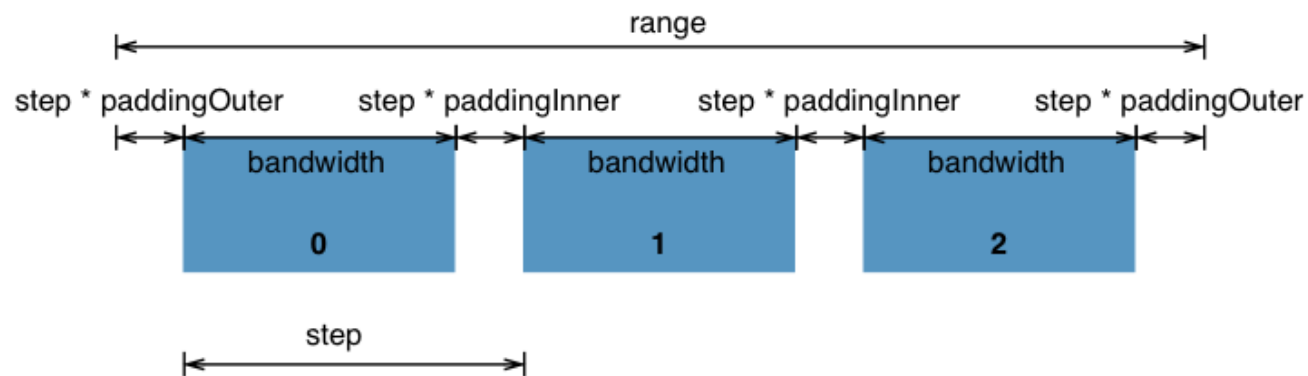
(b)

(a) <http://www.jeromecukier.net/wp-content/uploads/2011/08/d3scale1.png>

(b) [https://s3.amazonaws.com/dashingd3js/images/d3.js\\_scales\\_scale\\_domain\\_down\\_to\\_range\\_300x300.png](https://s3.amazonaws.com/dashingd3js/images/d3.js_scales_scale_domain_down_to_range_300x300.png)

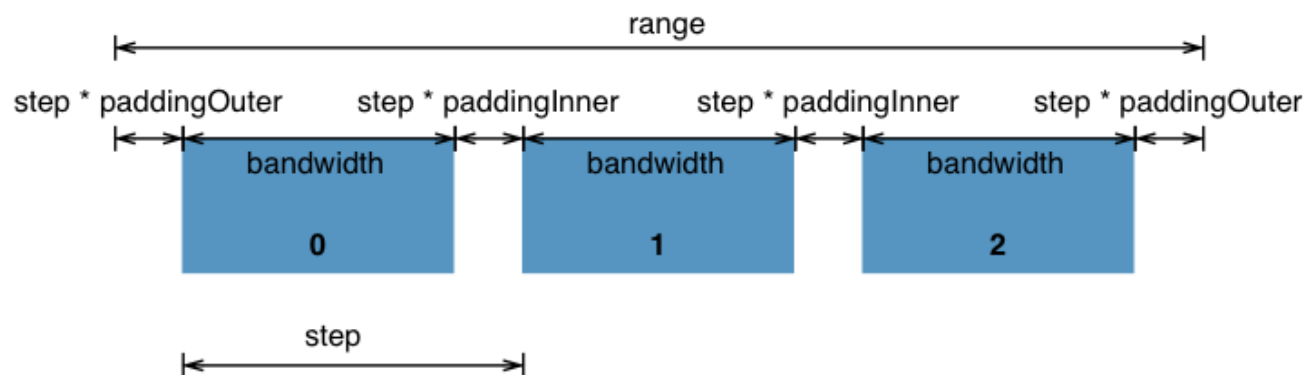
# Scale - Band

- `d3.scaleBand()`:
  - 定义一个‘条带’比例尺，返回的是一个函数。
  - e.g., `let scale = d3.scaleBand();`
- `scale.domain(array).range([min, max])`:
  - 设置比例尺的定义域与值域。
  - Band比例尺的定义域是离散的(Discrete)，值域是连续的。
  - e.g., `const scale = d3.scaleBand().domain(['a', 'b', 'c']).range([0, 120]);`
- 比例尺本质上是一个函数：
  - `scale('b') // 40`
  - `scale('c') // 80`



# Scale - Band

- 常结合JavaScript的array.map接口一起使用：
  - `let a = [{name: 'Shao-Kui', value:6}, {name:'Wen-Yang', value:6}, {name:'Yuan Liang', value:16}]`
  - `a.map(d => d.name) // ['Shao-Kui', 'Wen-Yang', 'Yuan Liang']`
  - `const yScale = d3.scaleBand()  
.domain(data.map(d => d.name))  
.range([0, innerHeight])`
- `scale.padding(0.1)`:
  - 设置条带的间距占各自区域的比重。
- `scale.bandwidth()`:
  - 返回条带的长度。



<https://raw.githubusercontent.com/d3/d3-scale/master/img/band.png>

# 比例尺

- 调用实例:

```
const data = [
  {name: 'Shao-Kui', value:6}, {name: 'Wen-Yang', value:6}, {name: 'Cai Yun', value:16},
  {name: 'Yuan-Chen', value:6}, {name: 'Rui-Long', value:10}, {name: 'Dong Xin', value:12},
  {name: 'He Yu', value:20}, {name: 'Xiang-Li', value:12}, {name: 'Tian-Xing', value:20},
  {name: 'Wei-Yu', value:15}, {name: 'Chen Zheng', value:14}, {name: 'Liang Yuan', value: 10},
  {name: 'Yu Peng', value:15}, {name: 'Li Jian', value:18}, {name: 'Wang Chen', value:16}
];

const xScale = d3.scaleLinear()
  .domain([0, d3.max(data, datum=>datum.value)])
  .range([0, innerWidth]);

const yScale = d3.scaleBand()
  .domain(data.map(datum => datum.name))
  .range([0, innerHeight])
  .padding(0.1);

console.log(xScale(12));
console.log(yScale('Wen-Yang'));
```

# 引入坐标轴

- 一个坐标轴为一个group (<g>)， 通常需要两个坐标轴。
- 坐标轴中包含：
  - 一个<path>用于横跨坐标轴的覆盖范围
  - 若干个刻度(.tick)
    - 每个刻度也是一个group
  - 每个刻度下属还会包含一个<line>和一个<text>
    - <line>用于展示坐标轴的轴线， 如左到右或上到下
    - <text>用于展示坐标轴的刻度值， 如实数、姓名、日期
  - (可选) 一个标签用以描述坐标轴
- 坐标轴的**定义**通常需要比例尺。



# 引入坐标轴

- 定义坐标轴（获得结果仍是**函数**）：

- `const yAxis = d3.axisLeft(yScale);`
- `const xAxis = d3.axisBottom(xScale);`
- `axisLeft`：左侧坐标轴。
- `axisBottom`：底侧坐标轴。
- 坐标轴的刻度对应比例尺的定义域。
- 坐标轴在画布的绘制对应比例尺的值域。
- 仅是对坐标轴的定义，还未绘制。

- 绘制坐标轴：

- `const yAxisGroup = g.append('g').call(yAxis);`
- `const xAxisGroup = g.append('g').call(xAxis);`
- 实际配置后会发现`<g>`中增添了与坐标轴相关的元素

- **任何坐标轴在初始化之后会默认放置在坐标原点，需要进一步的平移。**





# 关于 selection.call(...)

- (不做要求)
- 函数的输入为**另一个函数**。
- **另一个函数**以selection本身(即图元)作为输入。
- **另一个函数**中会根据函数体的内容修改selection对应的图元。
- 定义一个空白的<g>, D3会帮助我们定义好**另一个函数**, 我们通过.call(...)让<g>得以在**另一个函数**中修改。
  - `const yAxis = d3.axisLeft(yScale);`
  - `const yAxisGroup = g.append('g').call(yAxis);`

# 配置坐标轴

- 可以对坐标轴的风格进行修改：
  - 坐标轴本质上是图元的集合。
  - `d3.selectAll('.tick text').attr('font-size', '2em');`
  - `.tick`是D3对于坐标轴定义的统一class
- 坐标轴的标签加入不在D3-Axis接口的负责范围内：
  - 通过对坐标轴的`<g>`标签 `.append('text')`来实现
  - （左）纵轴坐标需要 `.attr('transform', 'rotate(-90)')` 来旋转
  - 纵轴坐标旋转后，`x / y` 会颠倒甚至取值范围相反
  - 回忆DOM：父节点的属性会影响子节点，而坐标轴默认的`'fill'`属性是`'none'`，因此请一定手动设置文字颜色 `.attr('fill', 'black')`



# 引入坐标轴 - Margin



- SVG对于D3.js是一个“画布”。
- SVG范围外的任何内容属于画布之外，浏览器将不予显示。
  - 然而坐标轴通常初始化在所在父节点的左上角。
- 定义Margin:
  - `const margin = {top: 60, right: 30, bottom: 60, left: 200}`
- 计算实际操作的 inner 长/宽
  - `const innerWidth = width - margin.left - margin.right;`
  - `const innerHeight = height - margin.top - margin.bottom;`
- 在SVG下额外定义一个组作为新的根节点
  - `const g = svg.append('g').attr('id', 'maingroup')`
  - `.attr('transform', `translate(${margin.left}, ${margin.top})`);`
- Tip: HTML确实在样式表中提供margin属性，然而设置其他图元的位置，仍需要计算innerWidth(Height)。

# 引入坐标轴

- 调用示例：
  - 比例尺可通过坐标轴可视化。

```
const yAxis = d3.axisLeft(yScale)//.tickSize(-innerWidth);
const xAxis = d3.axisBottom(xScale)//.tickSize(-innerHeight);

const yAxisGroup = g.append('g').call(yAxis)
.append('text') // -----
.text('Name')
.attr('font-size', '3em')
.attr('transform', 'rotate(-90)') // y-axis label needs an additional transform;
.attr('x', -innerHeight / 2)
.attr('y', -120)
.attr('fill', 'black')
const xAxisGroup = g.append('g').call(xAxis)
.attr('transform', `translate(${0}, ${innerHeight})`)
.append('text') // -----
.text('Value')
.attr('font-size', '3em')
.attr('x', innerWidth / 2)
.attr('y', 50)
.attr('fill', 'black');

d3.selectAll('.tick text').attr('font-size', '2em');

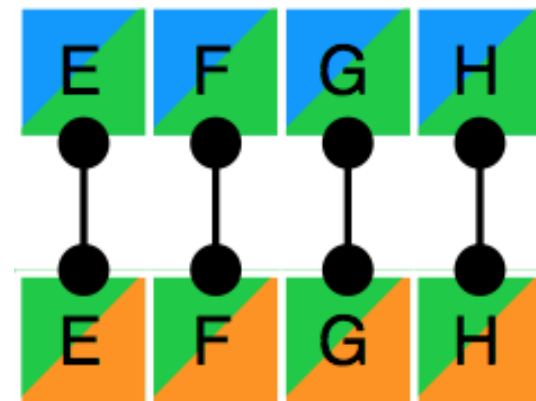
g.append('text').text('Members of CSCG').attr('font-size', '3em')
.attr('x', innerWidth / 2 - 200).attr('y', -10)
```

# 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；
  - 产品的销量绑定到矩形的长度；
  - 各类别商品的销售占比绑定到扇形的弧度。
- Why?
  - 以数据为中心(Data-Driven)的可视化操作：
    - 根据数据自动调整图元的属性。
    - **.attr(...)接口可基于图元自己绑定的数据自动调整属性值。**
  - 数据发生变化时可以自动对图元增删改查：
    - 不再需要手动添加、‘修改’、删除图元。
    - 根据数据的增加or删除or更新，自动补充or移除or更新图元。
- Data-Join并不是必要的操作，不使用Data-Join同样可以画出所有可视化作品。
- Data-Join只是让D3.js编程变得更高效率且语法更简洁。

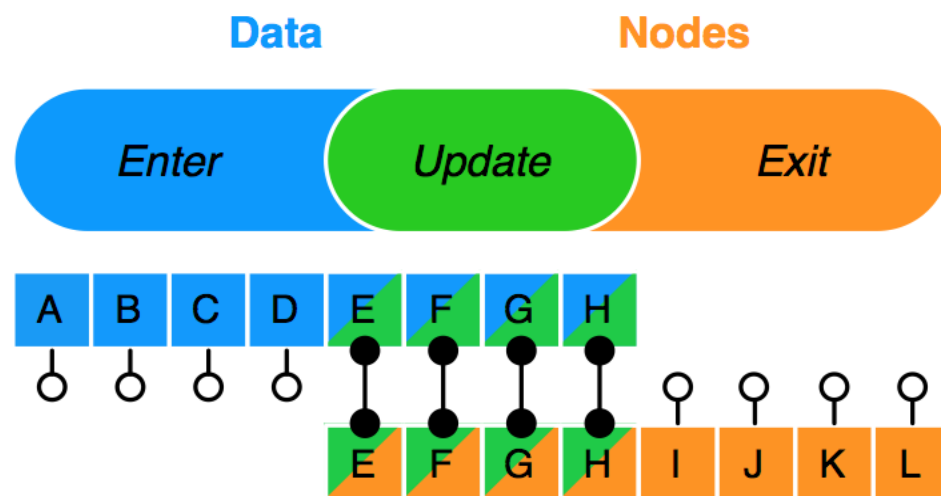
# Data-Join

- `d3.selectAll('.class').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, 32, 18]`;
  - e.g., `[{name: 'Sebastian', value:384}, {name:' Ciel', value:32},`  
`{name:'Wen-Yang', value:16}, {name:'Shao-Kui', value:19}]`;
- `.data(...)`只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。
- **默认的绑定按照双方的索引顺序：**
  - （Data的Key： 后续D3中会讨论。）
- **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。



# Data-Join

- 调用形式：
  - **d3.selectAll('.class').data(myData).join('图元').attr(d => ...).attr((d, i) => ...)**
  - .join(...)会根据数据的条目补全or删除图元。
- 若有新增的数据，则会自动增加对应图元。
- 若有修改的数据，则会自动更新对应图元。
- 若有删除的数据，则会自动移除对应图元。



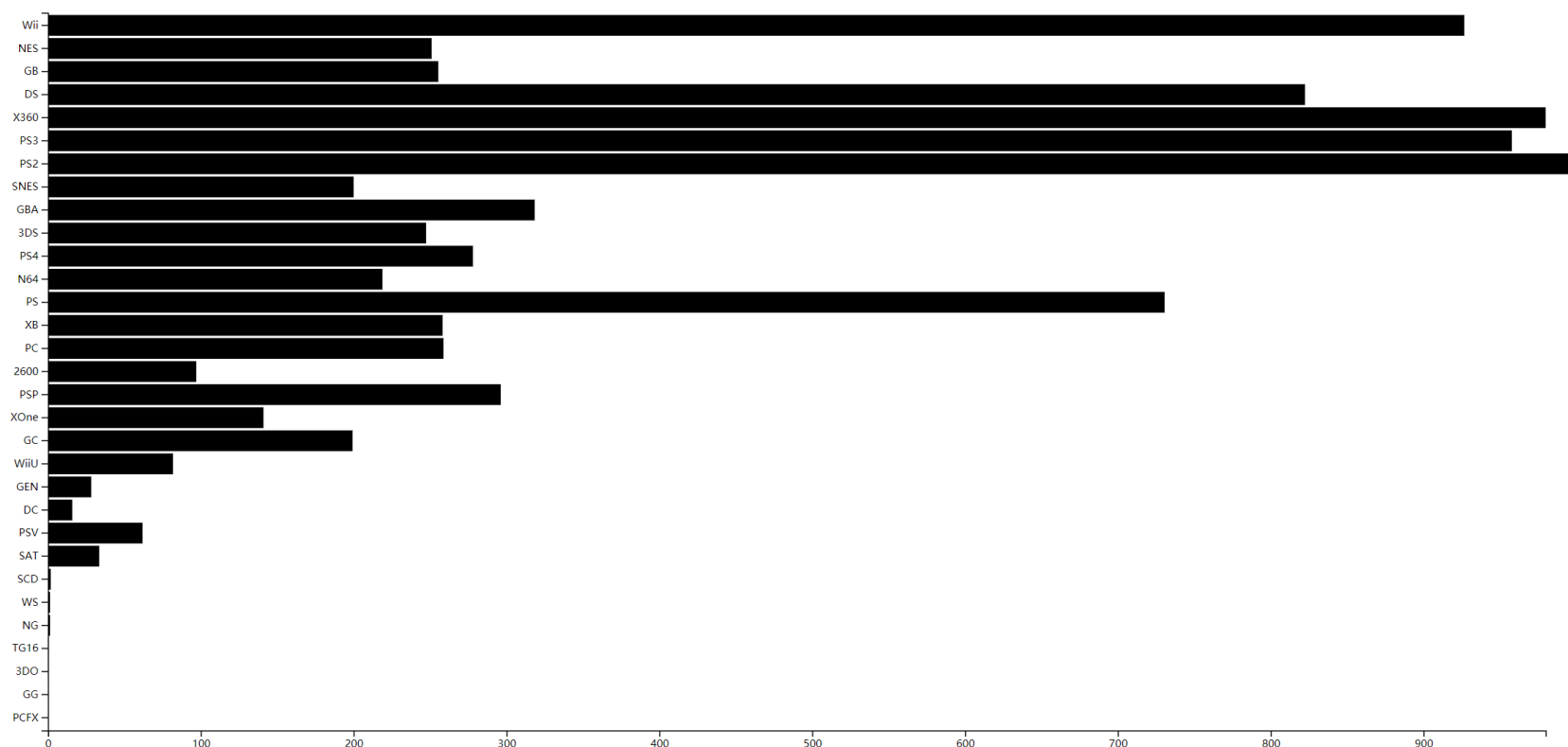
# Data-Join – 用函数设置图元属性

- `selection.attr('attributeName', 'value')`
  - 通过值设置属性。
- `selection.attr('attributeName', (d, i) => {...})`
  - 通过函数设置属性，函数的输入为绑定的数据，返回值为图元得到的属性值。
  - **d**为Data-Join中，`‘.data(array)’`绑定给每个图元的数据。
  - **i**为Data-Join中，`‘.data(array)’`绑定图元的顺序，即图元对应原本数组的第几个。
  - e.g., `d3.selectAll('rect').attr('width', (d, i) => 1000 * d.age );`
  - e.g., `d3.selectAll('circle').attr('cy', (d, i) => 200 * i + 30);`
  - 由于绑定数据的不同，故得到的结果也不同。
- 设置图元属性的函数遵循如下规则（顺序性）：
  - 函数可仅使用 `d => {...}`，即只有一个参数，但此时函数体无法使用索引。
  - 即使未使用到绑定的数据，如需使用索引，仍需要完整的写出 `(d, i) => {...}`。



# 基于D3的基础语法与Data-Join绘制柱状图

- 数据来源:
  - <https://www.kaggle.com/gregorut/videogamesales>



## Tip: 颜色 – ‘fill’属性

- PlanA: 人为定义一系列颜色组合
- PlanB: 使用D3提供的颜色组合（见下页）
- PlanC: 采样

```
var province_color_themeriver = {  
  "湖北": "#ffb79f",  
  "广东": "#ffc9b3",  
  "澳门": "#79657a",  
  "西藏": "#163249"  
}
```

```
// colors:  
const colorScale = d3.scaleOrdinal();  
colorScale.domain(platforms);  
const sp = d3.scalePoint().domain(platforms).range([0, 1]);  
colorScale.range(platforms.map(d => d3.interpolateSpectral(sp(d))));
```



# Tip: D3提供的各种色盘

- 定义一个离散数据到离散数据的映射
  - 如: 每个水果对应到某个颜色

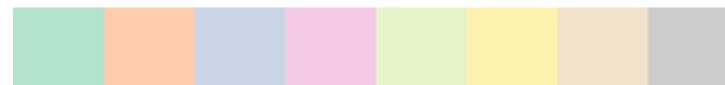
```
const color = d3.scaleOrdinal()  
.domain(naiveKeys)  
.range(d3.schemeSet3)
```

- D3.js的内嵌（自带）配色方案?
  - <https://github.com/d3/d3-scale-chromatic>

# d3.schemePastel1 <>



# d3.schemePastel2 <>



# d3.schemeSet1 <>



An array of nine categorical colors represented as RGB hexadecimal strings

# d3.schemeSet2 <>



An array of eight categorical colors represented as RGB hexadecimal strings

# d3.schemeSet3 <>

