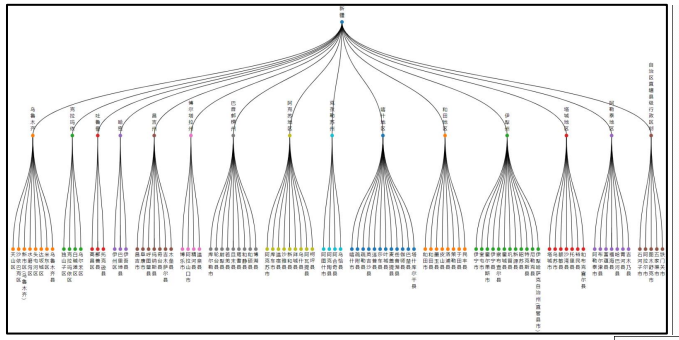
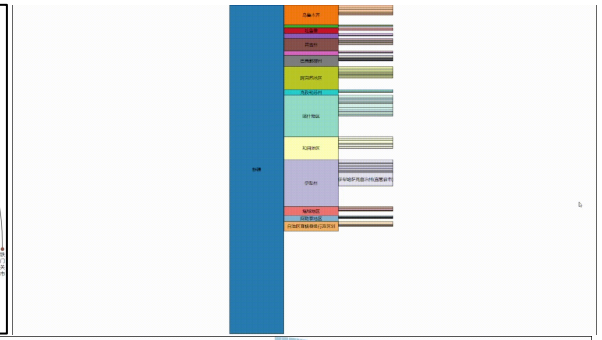
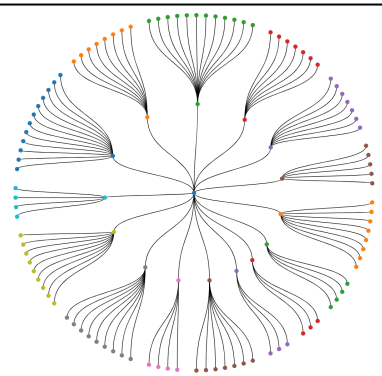
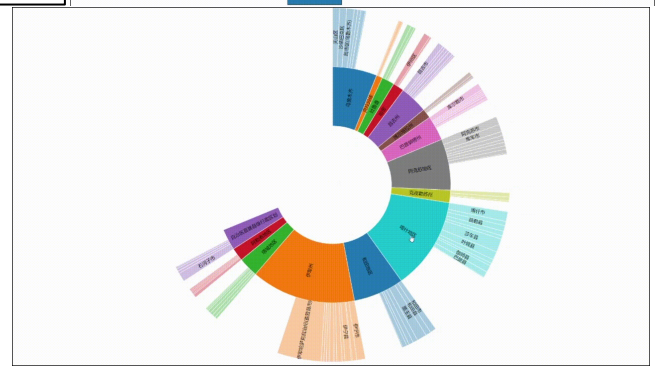
D3.JS – 层次数据可视化

张松海 张少魁
清华大学计算机系

2022

概览

- D3.js的层级数据可视化：
  - 层级数据的数据结构。
  - **d3.hierarchy**：层级数据的处理与预计算。
  - **d3.tree & d3.partition**：层级数据的划分与映射。
- 基于‘d3.tree’实现树状图：
  - d3.linkVertical() & d3.linkHorizontal()。
- 基于‘d3.partition’实现冰锥图与日晕图。

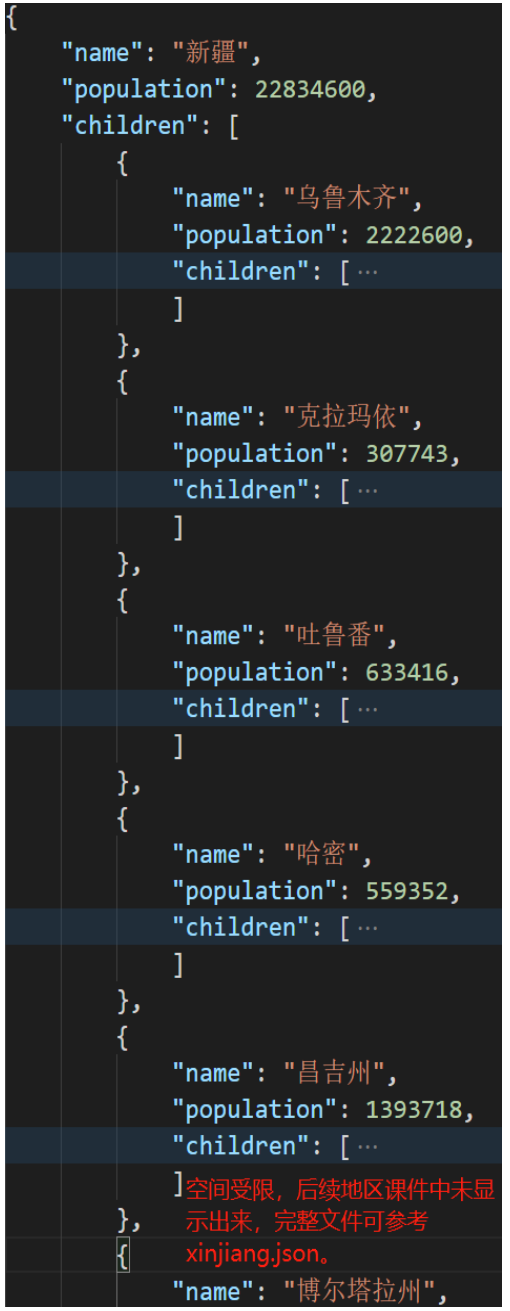





层级数据的数据结构

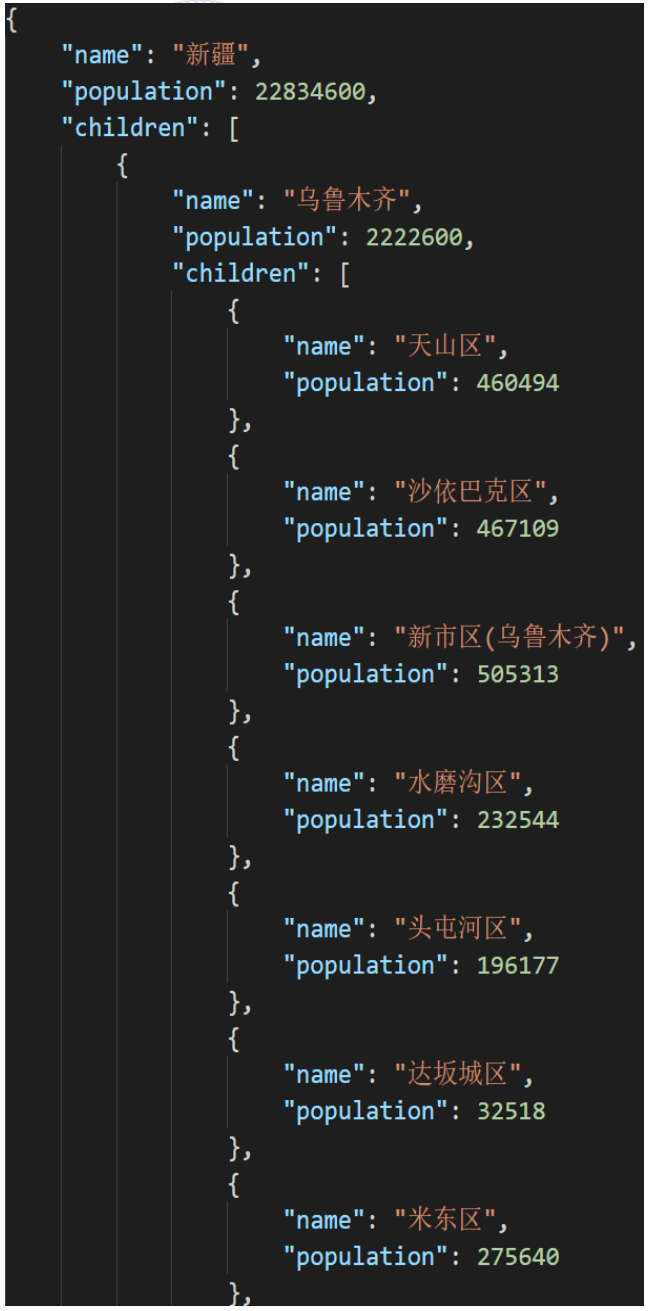
- D3.js接受的一种层级数据结构：
  - 层次结构没有一个非常‘通用’的形式。
  - 右侧仅仅是D3.js接受的形式，故需把自己的数据整理成这种形式。
- 一个节点的数据格式：
  - 属性1：属性值1，
  - 属性2：属性值2，
  - 属性3：属性值3，
  - …… ……
  - ‘**children**’：
    - 节点：{属性1：属性值1，属性2：属性值2，**children**：[…]}，
    - 节点：{属性1：属性值1，属性2：属性值2，**children**：[…]}，
    - …… ……

层级数据的数据结构

- 对于每个节点，可包含节点的若干属性，如名称、人口等：
  - "name": "新疆",
  - "population": 22834600,
  - ‘**children**’：
    - 节点：{"name": "乌鲁木齐", "population": 2222600，**children**：[...]},
    - 节点：{"name": "克拉玛依", "population": 307743，**children**：[...]},
    - … … …



空间受限，后续地区课件中未显示出来，完整文件可参考xinjiang.json。

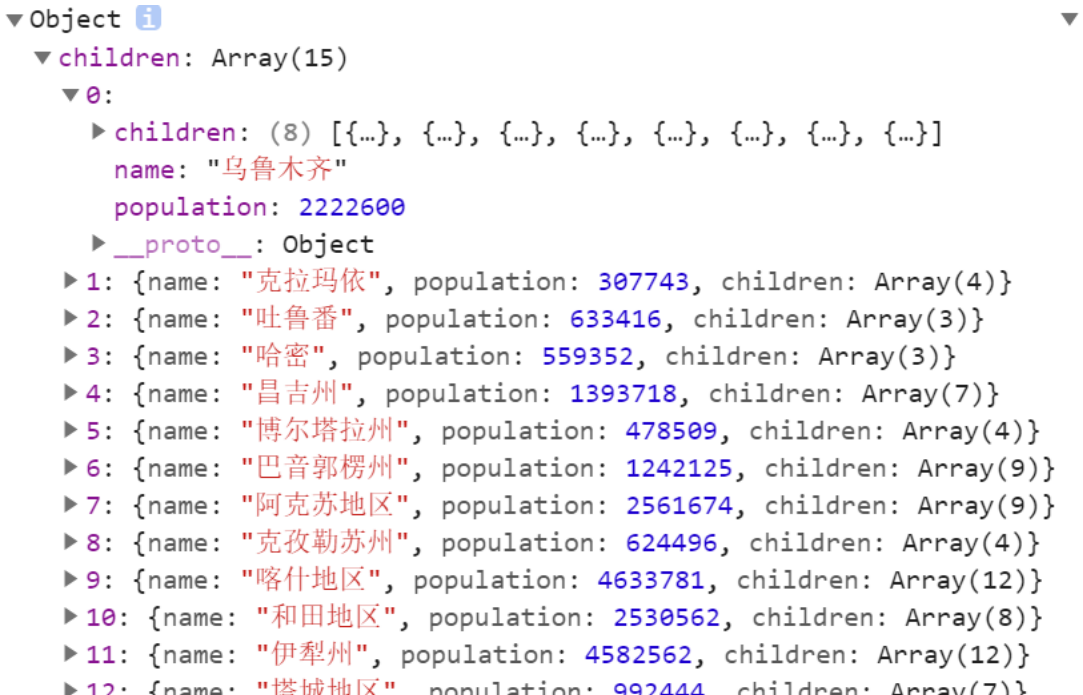


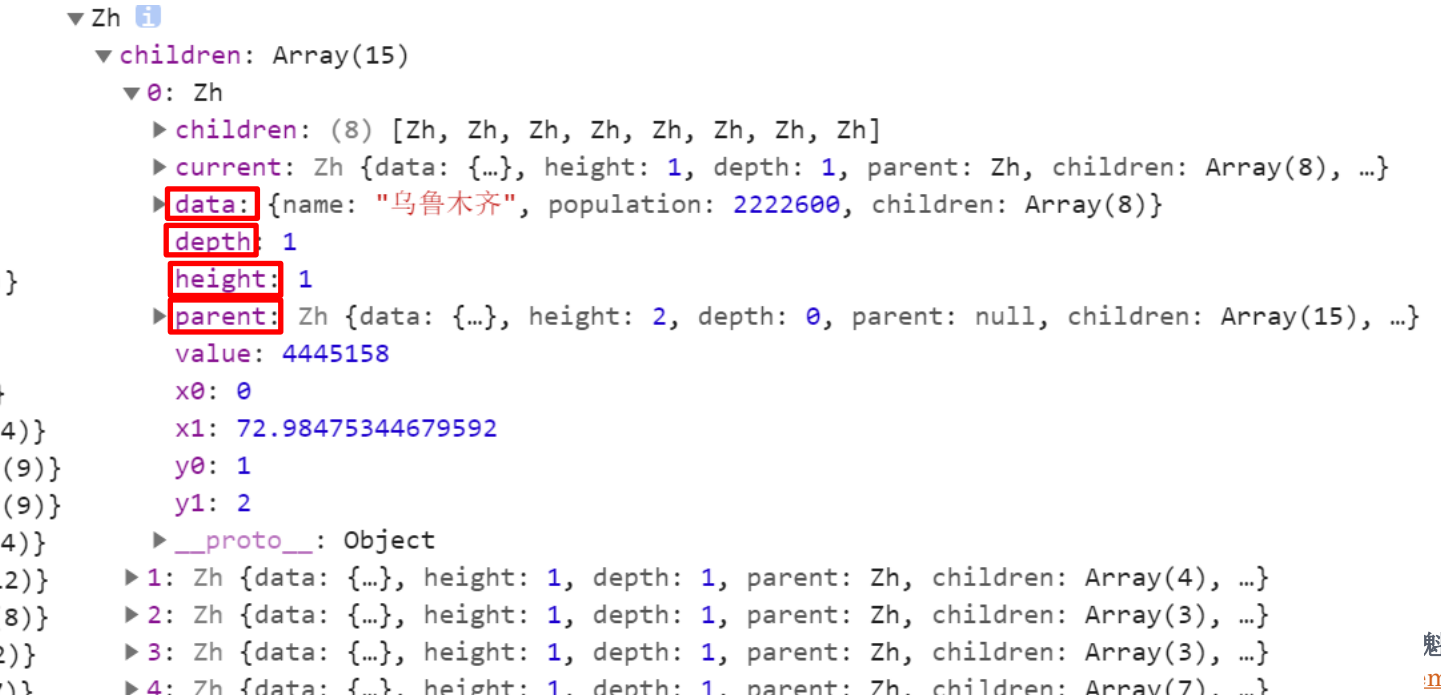
- **let root = d3.hierarchy(data).sum(...).sort(...):**
  - 参数**data**为上一页中的‘.json’层级数据。
  - 将输入数据进一步引入更多层级相关属性，保持数据的原始结构，并将输入层级数据转换成D3中的hierarcy对象(result instanceof d3.hierarchy)，引入:
    - **height**：所在节点的高度，**depth**：所在节点的深度；
    - **children**：原本数据的格式被保留，**parent**：到父节点的映射；
    - **data**：到原始数据的映射，**value**：节点的参考值。
- .sum(...):
  - 节点的取值，父节点的取值等于子节点的取值之和。
  - 非必须，因原数据本身可能已经带有求和信息，但建议每次都要调用，.sum(...)决定可视化参考的值，即**节点.value**。
  - e.g., d3.hierarchy(data).sum(d => d.population) // 父节点的值等于子节点的‘人口’之和。
- .sort(...):
  - 对每个父节点下设的子节点们进行排序。
  - e.g., d3.hierarchy(data). sort((a, b) => b.population - a.population) // 同一父节点的所有子节点按照人口数排序。

D3.HIERARCHY

- 注意：d3.hierarchy自己就是函数。
  - 编程实例与转换前后对比：

```
let hiedata = d3.hierarchy(data)
.sum(d => d.population).sort((a, b) => b.population - a.population);
```



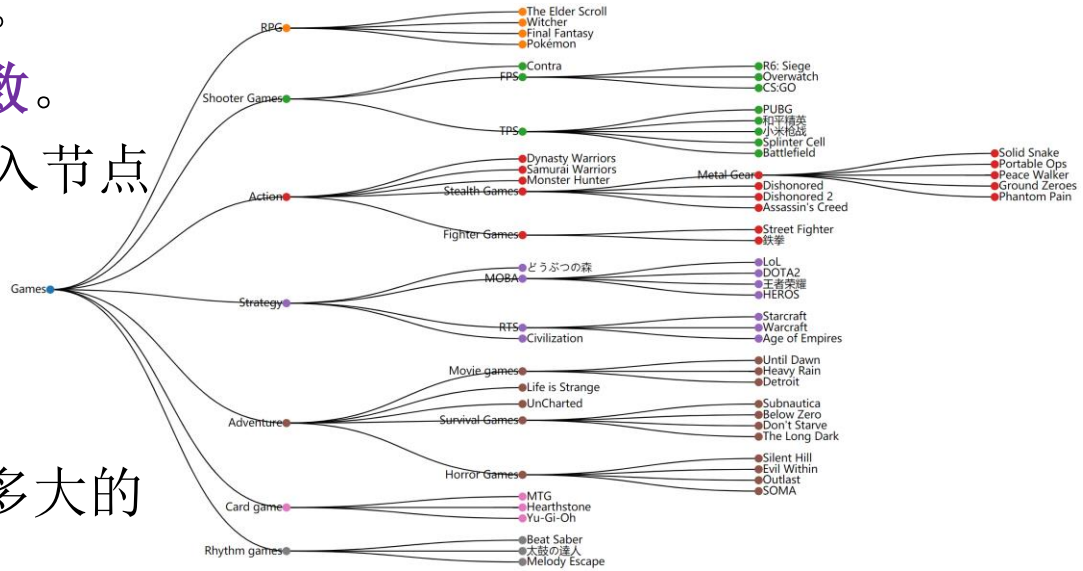


D3.HIERARCHY

- d3.hierarchy返回的数据带有若干接口：
  - let root = d3.hierarchy(data);
- root.descendants()：
  - 把层级结构‘拍平’，得到一个包含所有节点的数组。
  - 返回层级结构中的所有节点，包括根节点自己。
  - 广度优先。
  - 主要用于节点相关的Data-Join，如添加代表节点的图元（矩形、圆）等。
- root.links()：
  - 返回层级结构中的所有链接，以‘source’和‘target’的形式。
  - 主要用于连线相关的Data-Join，如添加节点间的连线。
  - 如：树型图中需要用到此接口返回所有节点间的链接。

D3.TREE

- d3.tree()：
  - 层级数据本身是抽象的结构，并不包含任何位置信息。
  - 定义一个函数，把d3.hierarchy(...)返回的数据在画布上划分，**得到每个节点，相对于层级结构，在画布中的位置**。
  - e.g., let tree = d3.tree(); // 定义一个树形图的映射函数。
  - e.g., let root = tree(d3.hierarchy(data)); // 进一步引入节点位置。
  - 函数**接受的输入是d3.hierarchy处理后的数据**
- tree.size([width, height])：
  - 定义划分树形区域的比例尺映射，即树形结构要画到多大的画布上。
  - 回忆地图数据可视化中的projection.fitSize()。
  - e.g., d3.tree().size([innerWidth, innerHeight])(d3.hierarchy(data));



D3.TREE

- 调用实例：

```
d3.json('xinjiang.json').then(data => {
    root = d3.hierarchy(data);
    // alternatively, we can set size of each node;
    // root = d3.tree().nodeSize([30, width / (root.height + 1)])(root);
    root = d3.tree().size([innerWidth, innerHeight])(root);
    render(root);
});
```

```
▼ Zh {data: {…}, height: 2, depth: 0, parent: null, children: Array(15), …}
  ▼ children: Array(15)
    ▼ 0: Zh
      ▶ children: (8) [Zh, Zh, Zh, Zh, Zh, Zh, Zh, Zh]
      ▶ data: {name: "乌鲁木齐", population: 2222600, children: Array(8)}
        depth: 1
        height: 1
      ▶ parent: Zh {data: {…}, height: 2, depth: 0, parent: null, children: Array(15), …}
        x: 84.61538461538461
        y: 375
      ▶ __proto__: Object
```

D3.PARTITION

- d3.partition()：
  - 定义一个函数，把d3.hierarchy(...)返回的数据在画布上划分，**得到每个节点的位置和所占区域，所占区域的比例随层级的值**。
  - e.g., let partition = d3.partition() // 定义一个层级区域的映射函数。
  - e.g., let root = partition(d3.hierarchy(data)); 进一步引入节点的位置与区域占比。
  - 注意1：**区域由四个值确定，但并不一定矩形**
  - 注意2：使用此接口前必须调用‘root.sum’
- partition.size([width, height])：
  - 定义划分树形区域的比例尺映射。
  - e.g., d3.partition().size([innerWidth, innerHeight])(d3.hierarchy(data));

D3.PARTITION

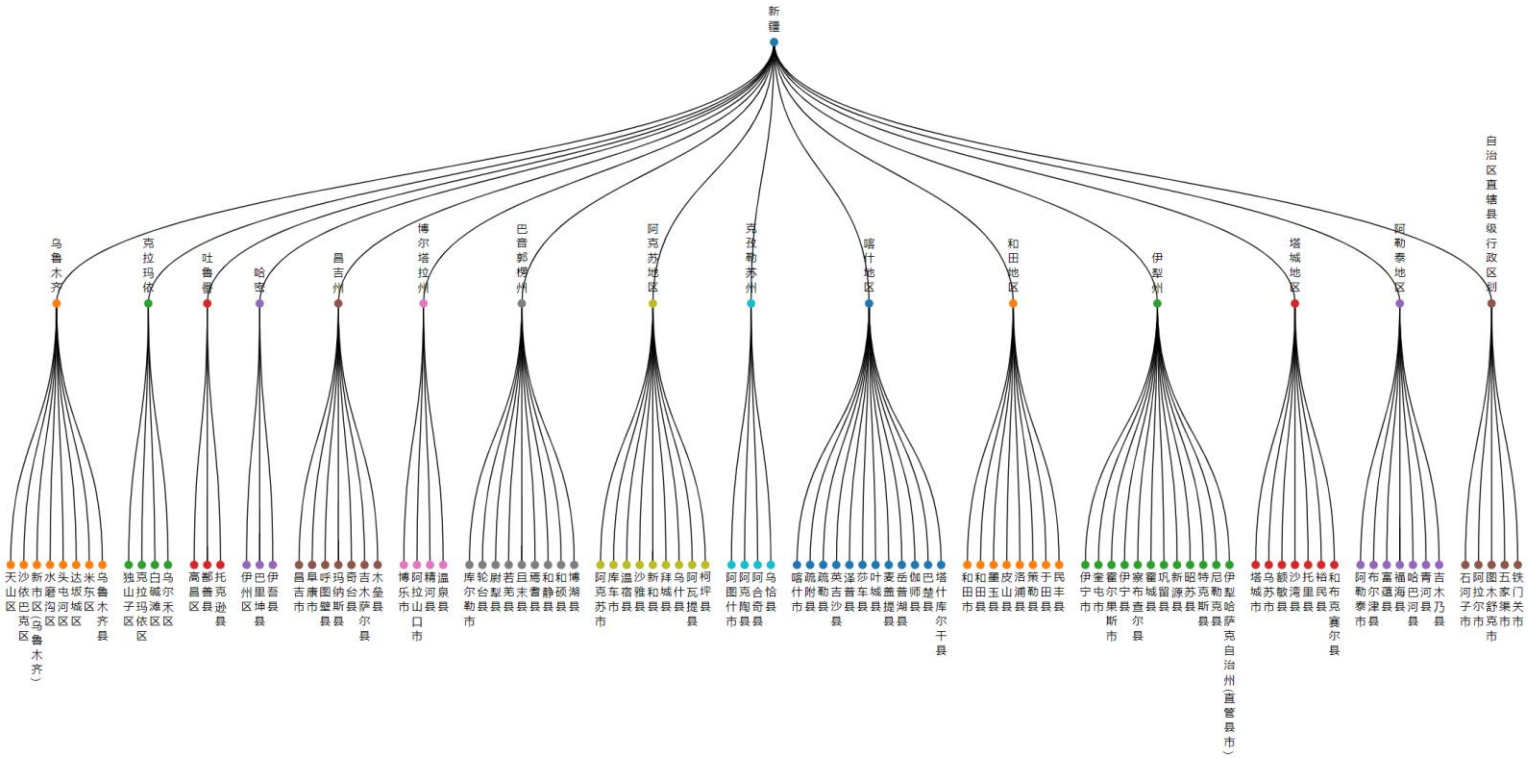
- 编程实例：

```
d3.json('xinjiang.json').then(data => {
    let hiedata = d3.hierarchy(data)
    .sum(d => d.population).sort((a, b) => b.population - a.population);
    root = d3.partition().size([height, width])(hiedata);
    render(root);
});
```

```
▼children: Array(15)
  ▼0: Zh
    ▶children: (8) [Zh, Zh, Zh, Zh, Zh, Zh, Zh, Zh]
    ▶data: {name: "乌鲁木齐", population: 2222600, children: Array(8)}
     depth: 1
     height: 1
    ▶parent: Zh {data: {…}, height: 2, depth: 0, parent: null, children: Array(15), …}
     value: 4445158
     x0: 0
     x1: 182.4618836169898
     y0: 333.3333333333333
     y1: 666.6666666666666
```

基于‘D3.TREE’实现树状图

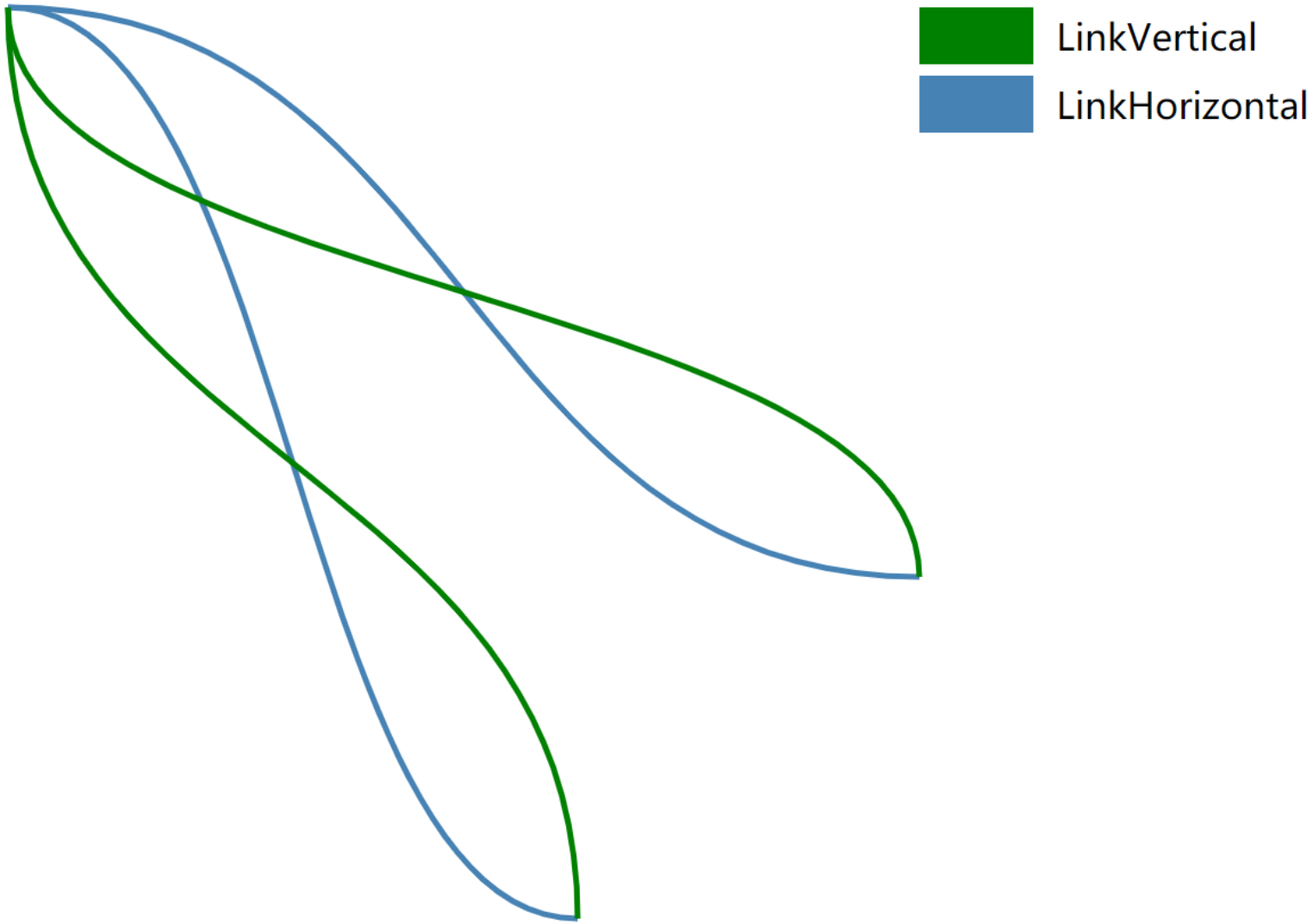
- 数据来源：
  - 中经网，后续层次可视化均使用此数据，https://njk.cei.cn/jsps/ShowTJKByAreaList?ProductID=779a6e57-e8e7-457b-a9de-6caef2764b35#。
- 需要用到的图元：
  - 圆(Data-Join)。
  - 文本(Data-Join)。
  - 连接线(d3.linkVertical)。



D3.linkVertical & D3.linkHorizontal

- d3.linkHorizontal().x(…).y(…):
  - 返回一个函数，输入为两个节点，**输出为<path>的‘d’属性**。
  - e.g., let path = d3.linkHorizontal();
  - 输入数据，必须包含‘source’属性作为起点，‘target’属性作为终点。
  - e.g., const data1 = {'source': {'x': 100, 'y': 100}, 'target': {'x': 600, 'y': 900}}
  - 通过**path.x(…).y(…)**来指定起点与终点的取值。
  - e.g., path.x(d => d.x).y(d => d.y);
  - e.g., path(data1); // M100,100C350,100,350,900,600,900
- d3.linkVertical():
  - 调用方式与d3.linkHorizontal完全相同，区别在于<path>的‘d’属性的笔顺。

D3.LINKVERTICAL & D3.LINKHORIZONTAL



D3.linkVertical & D3.linkHorizontal

- 调用实例：

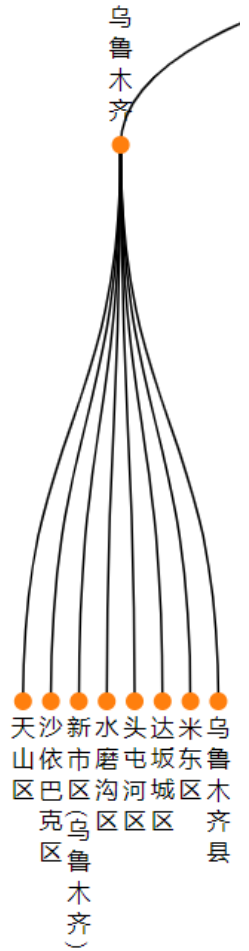
```
const height = +svg.attr('height');
const data1 = {'source': {'x': 100, 'y': 100}, 'target': {'x': 600, 'y': 900}};
const data2 = {'source': {'x': 100, 'y': 100}, 'target': {'x': 900, 'y': 600}};
const pathH = d3.linkHorizontal().x(d => d.x).y(d => d.y);
const pathV = d3.linkVertical().x(d => d.x).y(d => d.y);
svg.append('path').attr('d', pathH(data1)).attr('stroke', 'steelblue');
svg.append('path').attr('d', pathV(data1)).attr('stroke', 'green');
svg.append('path').attr('d', pathH(data2)).attr('stroke', 'steelblue');
svg.append('path').attr('d', pathV(data2)).attr('stroke', 'green');
```

D3.linkVertical & d3.linkHorizontal

- let root = d3.hierarchy(data);
- root.links()
  - 返回树形结构中存在的所有‘链接’，链接(们)以如下形式给出：

```
> root.links()
<- (72) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
   {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
   {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
   {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}]
   ▶ 0: {source: Rl, target: Rl}
   ▶ 1: {source: Rl, target: Rl}
   ▶ 2: {source: Rl, target: Rl}
   ▼ 3:
     ▶ source: Rl {data: {…}, height: 4, depth: 0, parent: null, children: Array(7), …}
     ▶ target: Rl {data: {…}, height: 2, depth: 1, parent: Rl, children: Array(4), …}
     ▶ __proto__: Object
```

  - 故，返回的一系列链接，直接对应d3.linkHorizontal&d3.linkVertical。



基于‘D3.TREE’实现树状图

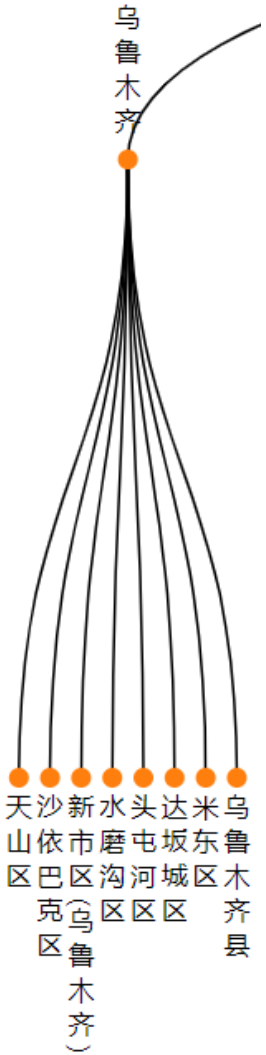
- 调用实例 – 为属性图添加节点间的连线：

```
g.selectAll("path")
.data(root.links())
.join("path")
.attr("fill", "none")
.attr("stroke", "black")
.attr("stroke-width", 1.5)
.attr("d", d3.linkVertical().x(d => d.x).y(d => d.y));
```

```
g.selectAll('circle').data(root.descendants()).join('circle')
.attr("stroke-width", 3)
.attr("fill", fill)
.attr('cx', d => d.x)
.attr('cy', d => d.y)
.attr("r", 6);
```

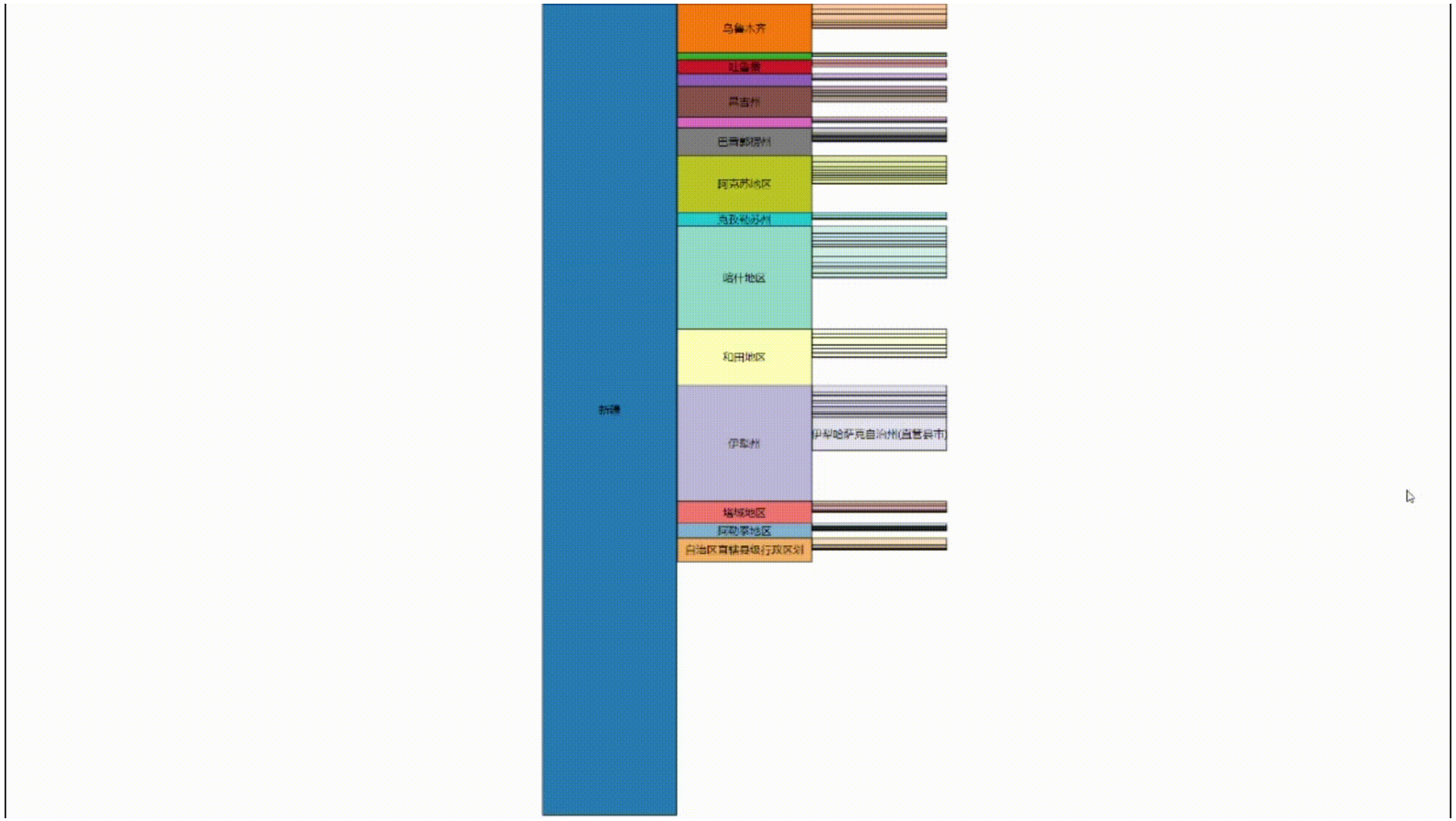
TIP：文本的属性

- .attr("text-anchor", **d => d.children ? "end" : "start"**)
  - 若是根节点，返回‘end’，否则返回‘start’。
  - 变量 ? a : b：JavaScript中的三元表达式，根据变量的真假，返回a OR b。
  - 变量为‘undefined’时，JavaScript默认结果为‘false’。
  - 调用结果：根节点的文本行文向外、非根节点的文本行文向内。
- .attr('writing-mode', 'vertical-rl')
  - 设置文本的行文方向为垂直。
  - 调用结果：文本竖直行文。



基于‘D3.PARTITION’实现冰锥图

- 数据来源：新疆人口数据（同）。



基于‘D3.PARTITION’实现冰锥图

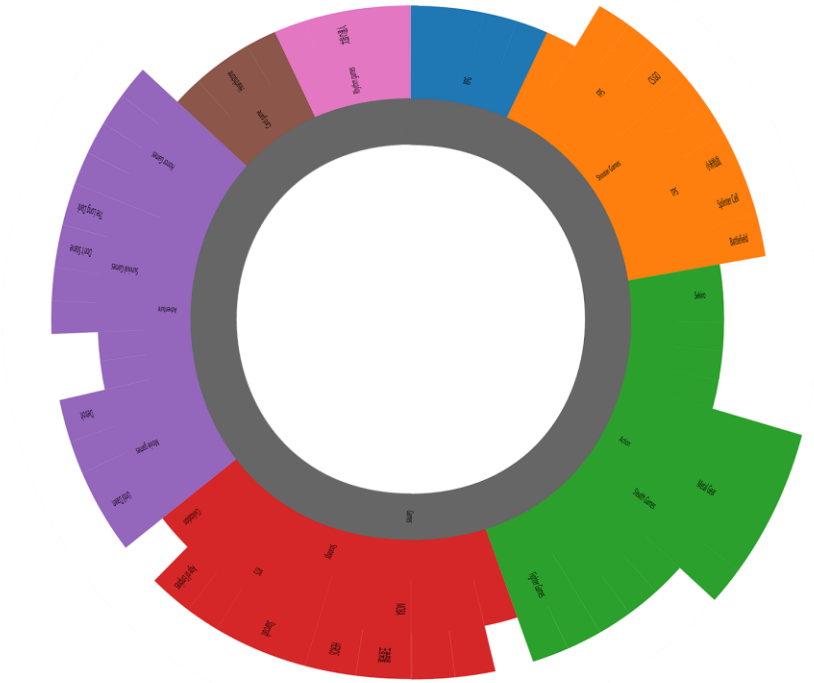
- d3.partition直接返回各个节点所占的区域，故直接根据每个节点Data-Join即可。
- 编程实例：

```
g.selectAll('.datarect')
.data(root.descendants())
.join('rect')
.attr('class', 'datarect')
.attr('x', d => d.y0)
.attr('y', d => d.x0)
.attr('height', d => d.x1 - d.x0)
.attr('width', d => d.y1 - d.y0)
.attr("fill", fill);
```

```
d3.json('xinjiang.json').then(data => {
    let hiedata = d3.hierarchy(data)
    .sum(d => d.population).sort((a, b) => b.population - a.population);
    root = d3.partition().size([height, width])(hiedata);
    render(root);
});
```

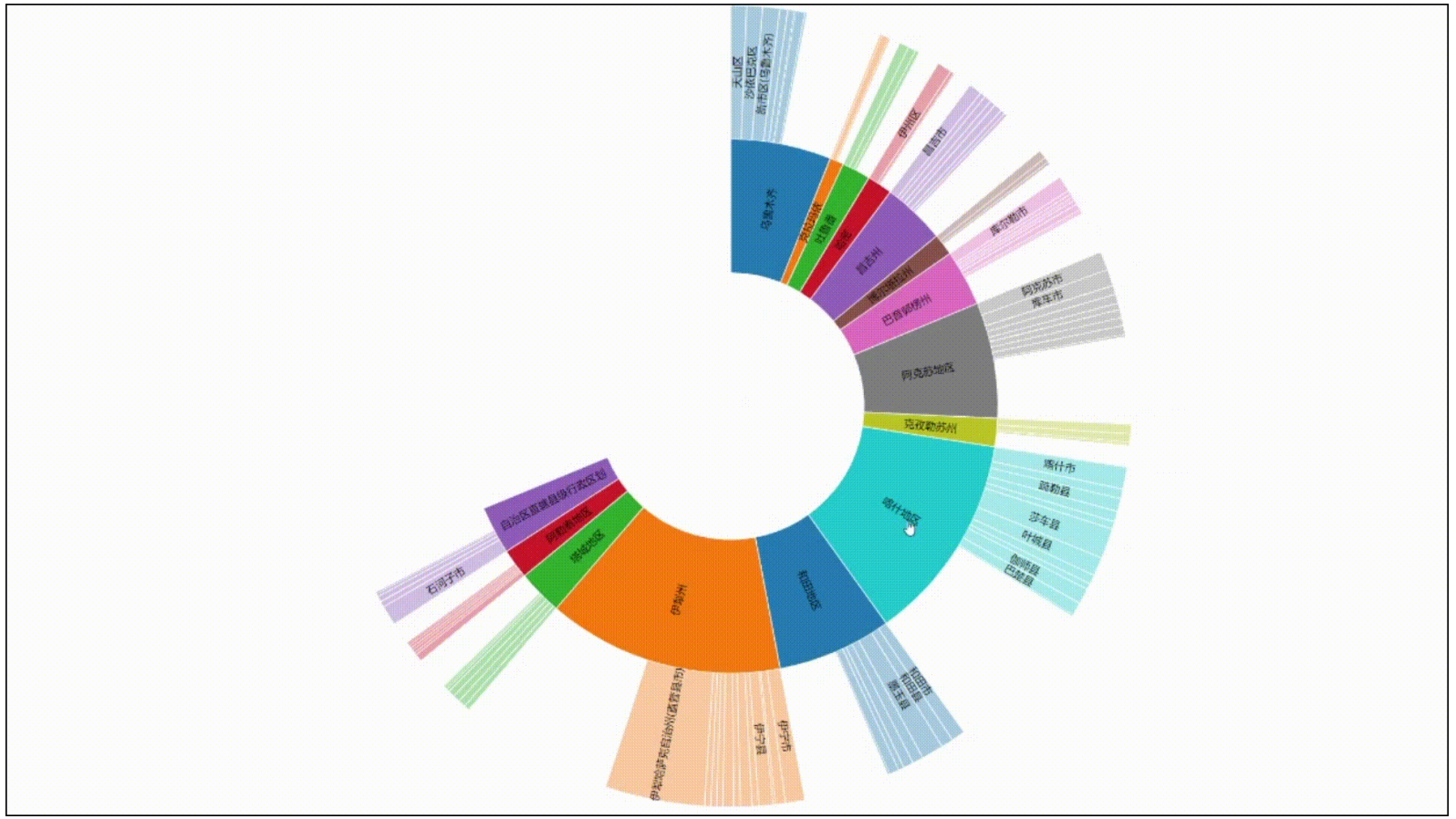
把Icicle图‘掰’成‘一圈’？...

- Icicle的问题？
  - 前后‘断开’了，头尾之间的比例比较直观性略差...
  - 能否把头尾相连？...



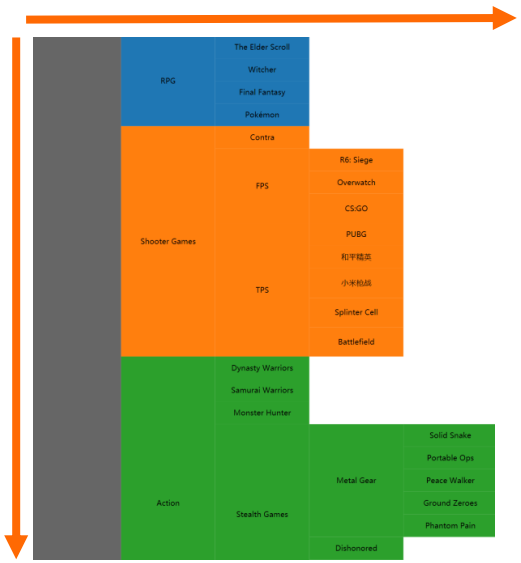
基于‘D3.PARTITION’实现日晕图

- 数据来源：新疆人口数据（同）。



基于‘D3.PARTITION’实现日晕图

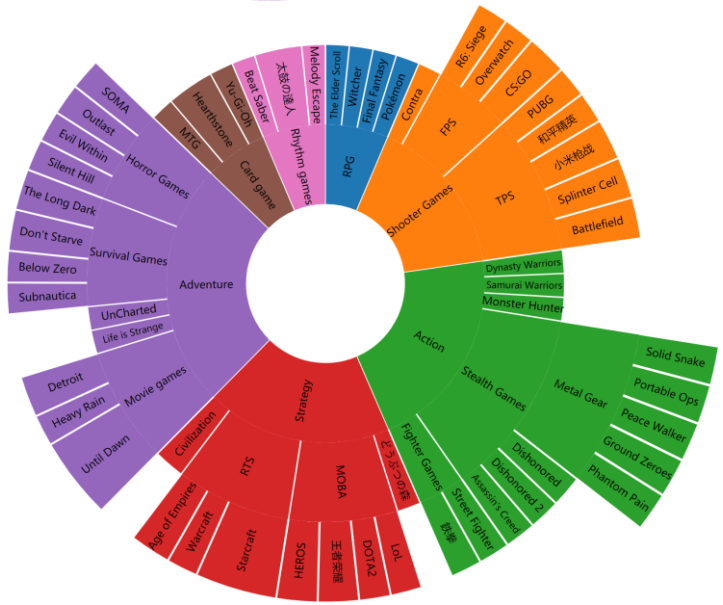
- 冰锥图的区域划分：
  - 纵向：画布的高度。
  - 横向：画布的宽度。
- 日晕图的区域划分：
  - ‘纵向’：圆周，$[0, 2\pi]$。
  - ‘横向’：半径，从圆心到最外侧。
- 编程实例：

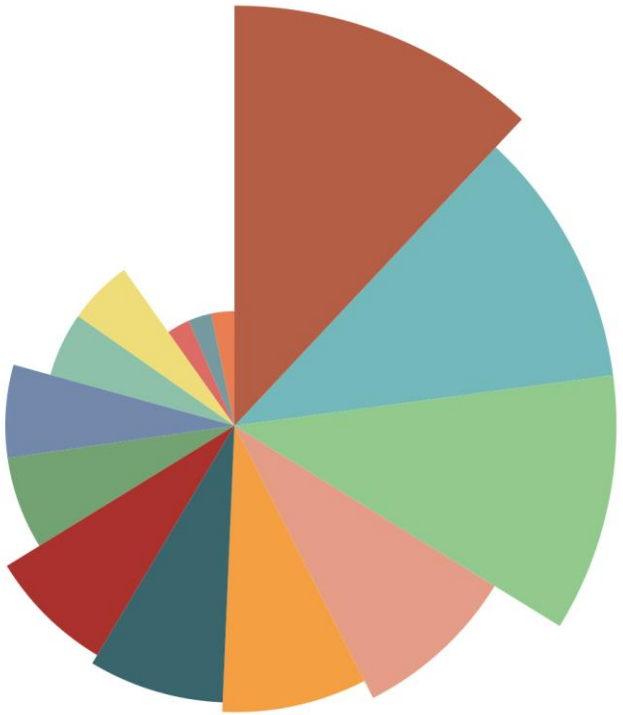



```
d3.json('xinjiang.json').then(data => {
    root = d3.partition().size([2 * Math.PI, 500])
        (d3.hierarchy(data).sum(d => d.population)
            .sort((a, b) => b.population - a.population));
    render(root);
});
```

基于‘D3.PARTITION’实现日晕图

- 图元：矩形 → 弧。
- d3.arc(...)可以根据绑定的数据调节**起始角度、终止角度、内半径与外半径：**
  - e.g., let arc = d3.arc()
  - .startAngle(d => d.x0); // 起始角度。
  - .endAngle(d => d.x1); // 终止角度。
  - .innerRadius(d => d.y0); // 内半径。
  - .outerRadius(d => d.y1); // 外半径。
- 其中，变量d是每个弧绑定的数据。
- Tip：玫瑰图的绘制同理。基于数据，不仅调节起始、终止角度，并且调节半径。





基于‘D3.PARTITION’实现日晕图

- 编程实例：

```
const arc = d3.arc()
.startAngle(d => d.x0)
.endAngle(d => d.x1)
.innerRadius(d => d.y0)
.outerRadius(d => d.y1)
```

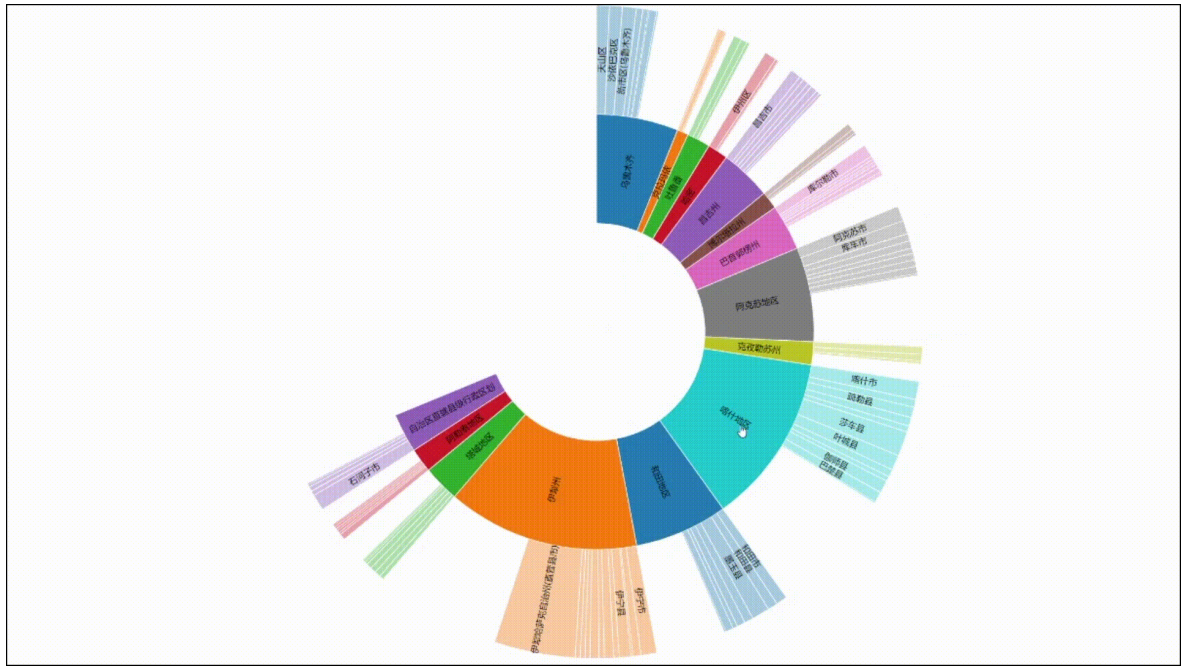
```
g.selectAll('.datapath')
.data(root.descendants().filter(d => d.depth !== 0))
.join('path')
.attr('class', 'datapath')
.attr("fill", fill)
.attr("d", arc)
```

Tip：冰锥图与日晕图的文本如何添加?

- PlanA：
  - 不显示过长的文本或小区域的文本，e.g.,

```
.data(root.descendants()
.filter(d => d.depth && (d.x1 - d.x0) > Math.PI / 65 && d.data.name.length
< 15));
```

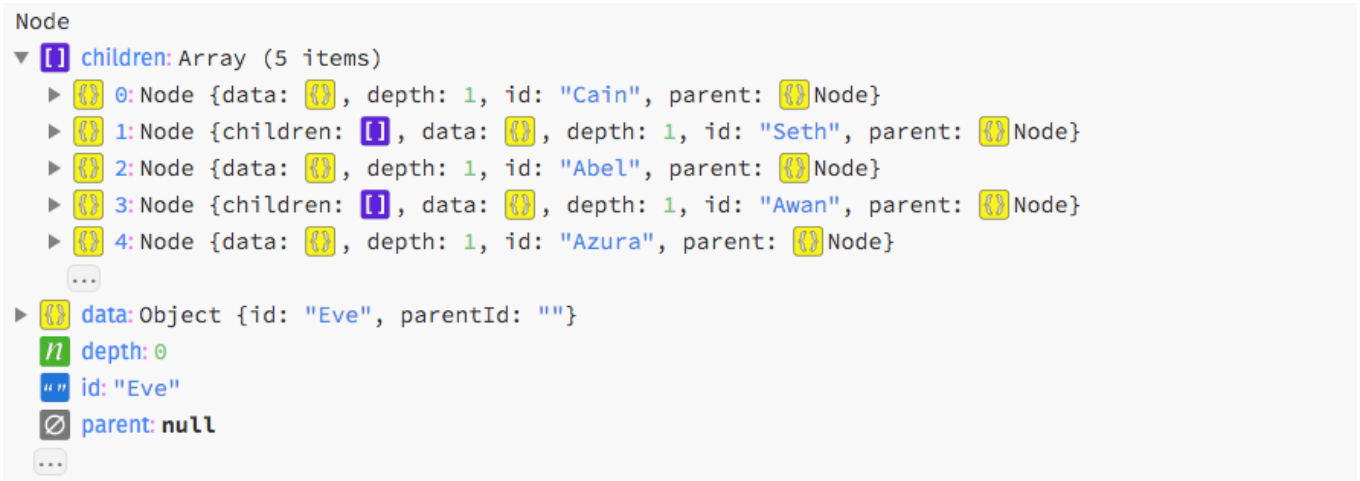
  - 或可根据区域大小，调整文本的大小。
- PlanB：交互与动画。
- PlanC：使用Echarts。



TIP：如何整理得到‘D3.HIERARCHY’的输入?

- d3.stratify：
  - https://github.com/d3/d3-hierarchy/blob/v2.0.0/README.md#stratify

```
[
  {"name": "Eve",   "parent": ""},
  {"name": "Cain",  "parent": "Eve"},
  {"name": "Seth",  "parent": "Eve"},
  {"name": "Enos",  "parent": "Seth"},
  {"name": "Noam",  "parent": "Seth"},
  {"name": "Abel",  "parent": "Eve"},
  {"name": "Awan",  "parent": "Eve"},
  {"name": "Enoch", "parent": "Awan"},
  {"name": "Azura", "parent": "Eve"}
]
```



Tip：树状图的径向布局？

- 同日晕图。