D3.js – 树与图

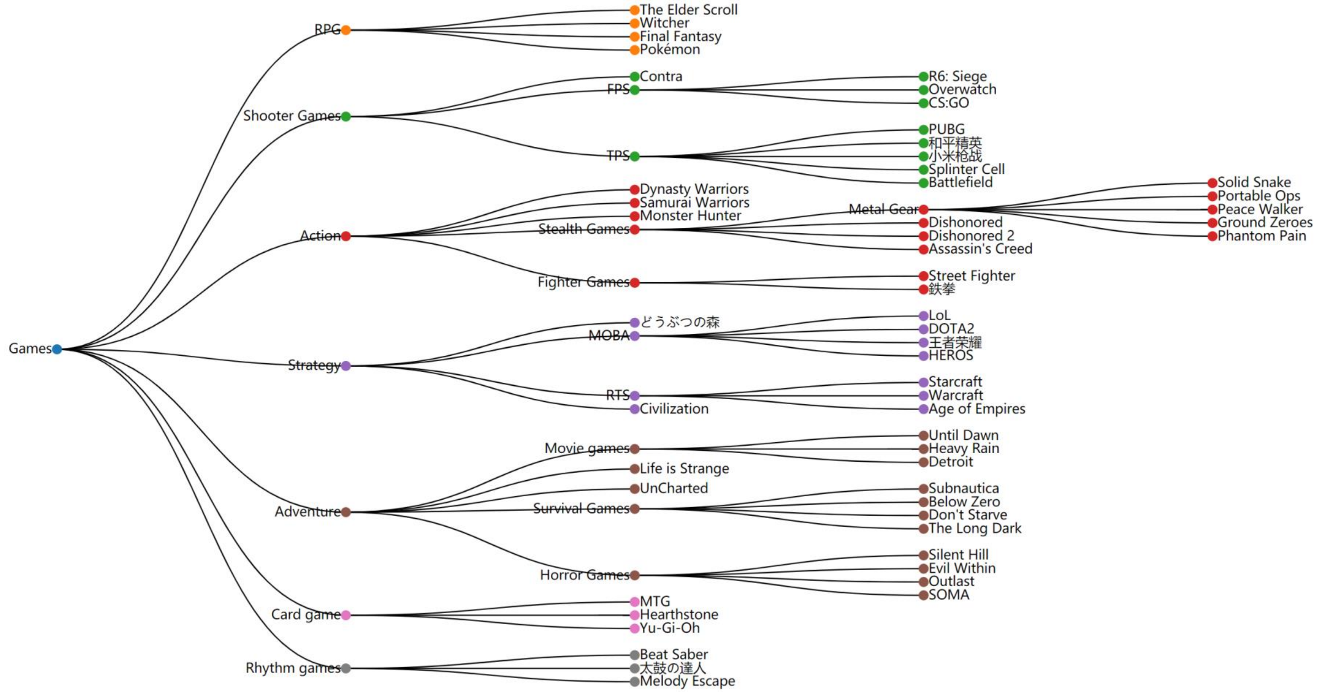
张松海、张少魁、周文洋、蔡韵

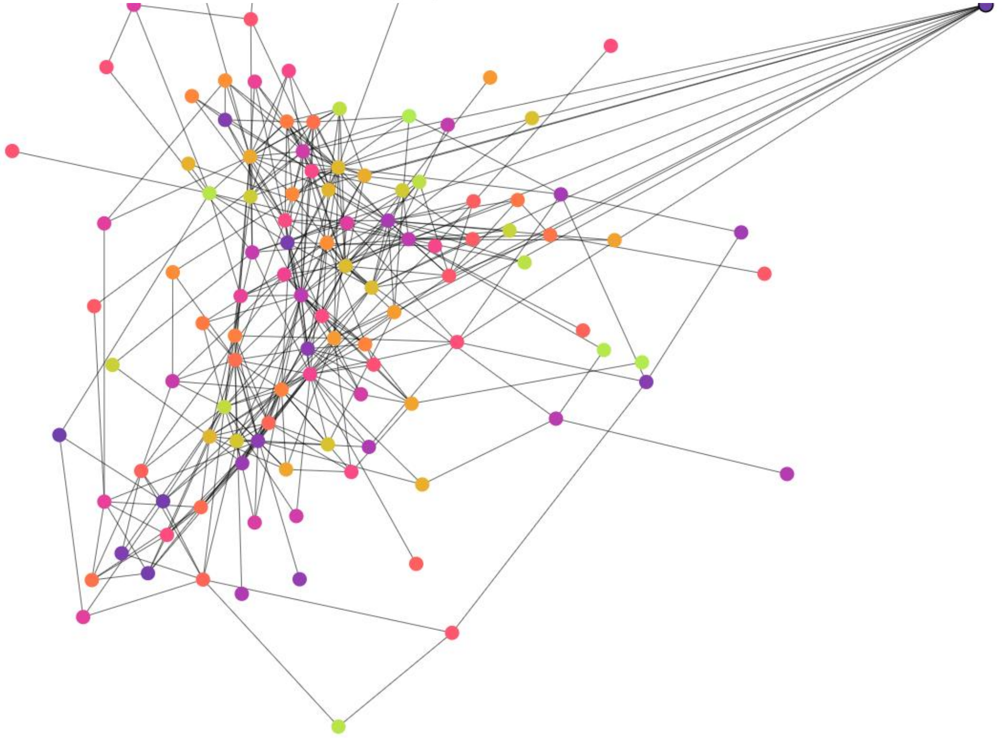
数据可视化 – D3.js

清华大学 可视媒体研究中心

Tree & Graph

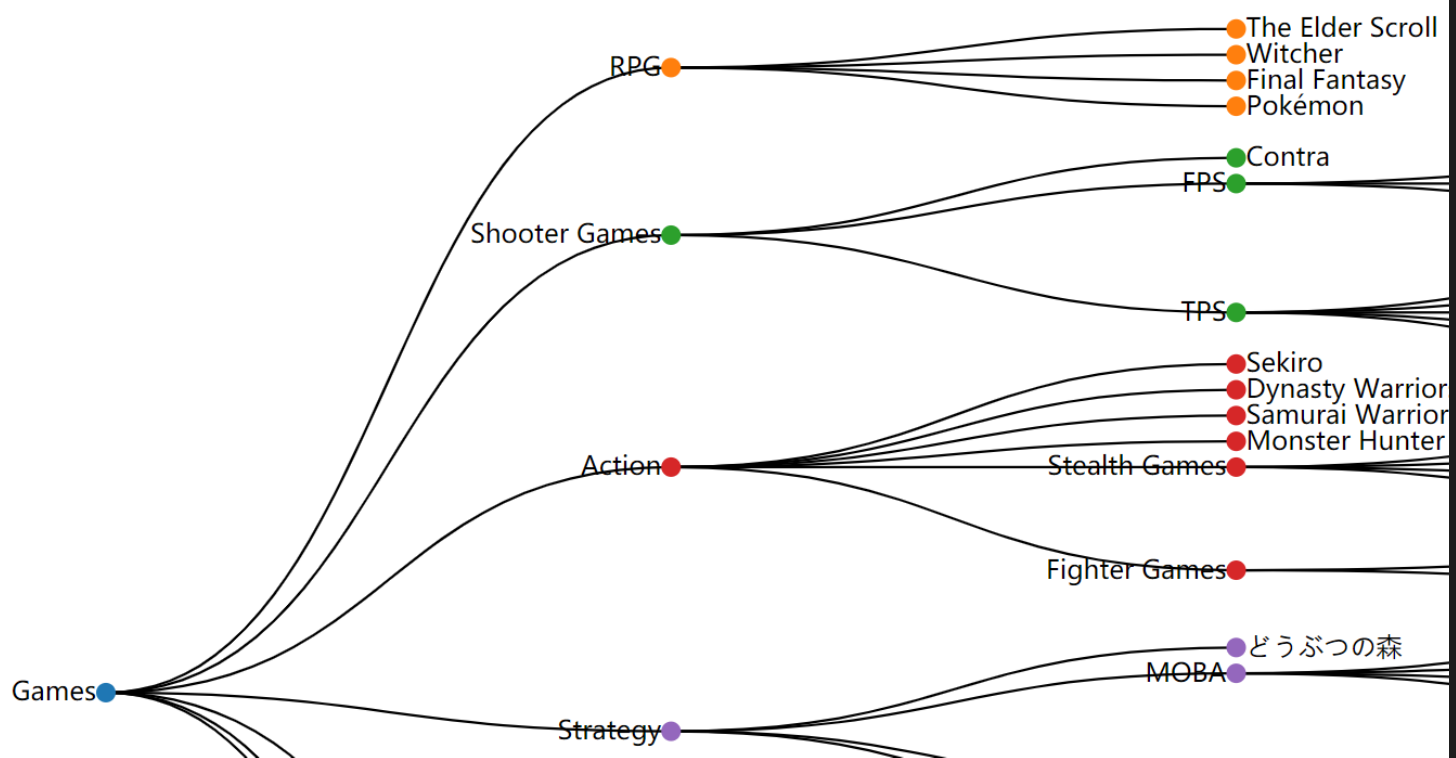
- A ‘tree’ is a ‘graph’ ☺
- 层级结构的可视化
    - 层级结构：树、Tree、Hierarchy
        - **d3.hierarchy**
    - ‘直接的’可视化方案
        - d3.tree
    - 更直观的可视化方案
        - d3.partition & **d3.arc** for ‘d’ of <path>
- 网络结构的可视化
    - 网络结构：图、Graph、Network
    - D3.js：**Force Simulation**





层级数据?

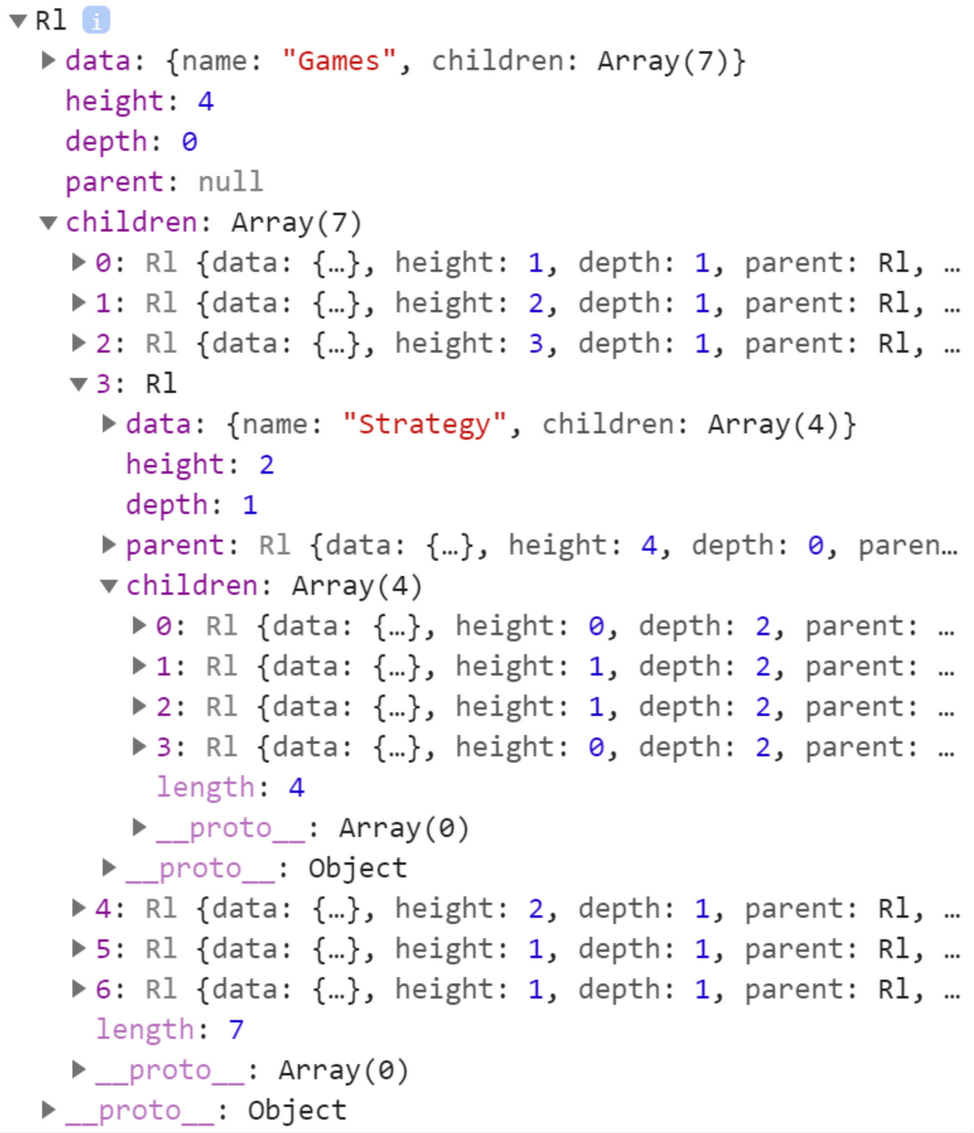
- 数据格式仍为Json
- 节点可以包含‘属性’




```json
"name": "Games",
"children": [
    {
        "name": "RPG",
        "children": [
            {
                "name": "The Elder Scroll",
                "popularity": 500
            },
            {
                "name": "Witcher",
                "popularity": 500
            },
            {
                "name": "Final Fantasy",
                "popularity": 500
            },
            {
                "name": "Pokémon",
                "popularity": 500
            }
        ]
    },
```

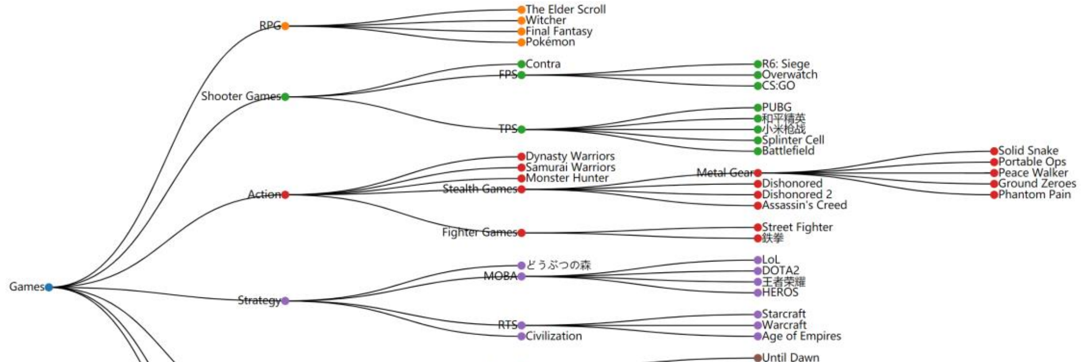
D3.js的层级数据预处理

- d3.hierarchy
- 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)，同时引入:
  - **height （* 不是逐层递减）**
  - **depth**
  - children (原始结构)
  - **parent**
  - data 原始数据的映射
- d3.hierarchy可作为一个‘**中间结果**’，**继续输入到更多D3.js提供的数据预处理接口中**



D3.js的层级数据预处理 cont.

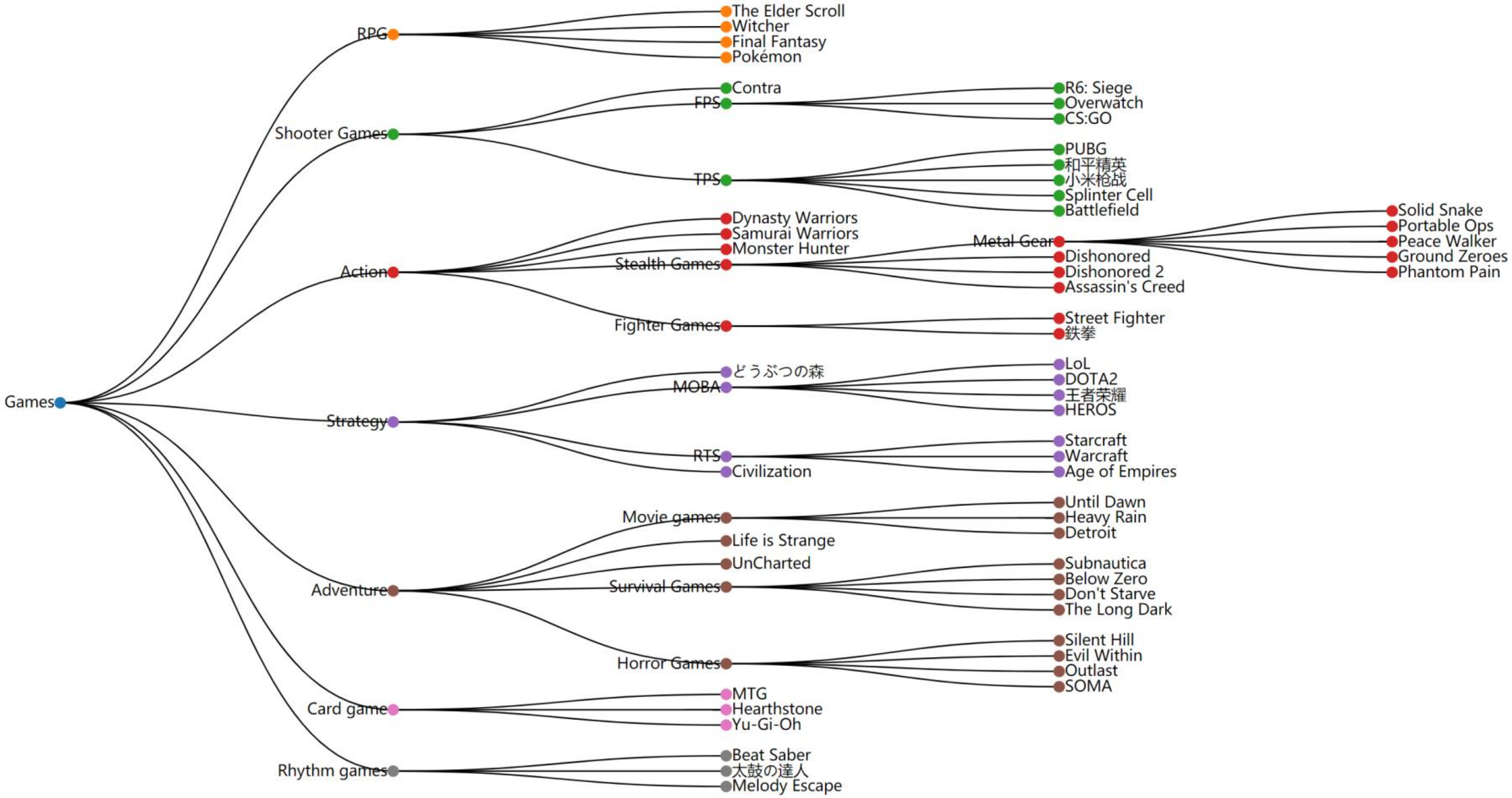
- 可以将处理好的数据'进一步'预处理?
  - d3.tree()
  - .size([innerHeight, innerWidth])
- ↑代码会:
  - 返回一个函数
  - 函数接受的参数为d3.hierarchy
  - 函数会根据设置的size将树形结构的每个节点映射到空间中'**合适**'的位置




```
▼Rl {data: {…}, height: 4, depth: 0, parent: null, children: Array(7), …}
  ▶data: {name: "Games", children: Array(7)}
   height: 4
   depth: 0
   parent: null
  ▼children: Array(7)
    ▶0: Rl {data: {…}, height: 1, depth: 1, parent: Rl, children: Array(4), …}
    ▶1: Rl {data: {…}, height: 2, depth: 1, parent: Rl, children: Array(3), …}
    ▶2: Rl {data: {…}, height: 3, depth: 1, parent: Rl, children: Array(6), …}
    ▼3: Rl
      ▶data: {name: "Strategy", children: Array(4)}
       height: 2
       depth: 1
      ▶parent: Rl {data: {…}, height: 4, depth: 0, parent: null, children: Array(7), …}
      ▶children: (4) [Rl, Rl, Rl, Rl]
       x: 447.7358490566038
       y: 347.5
      ▶__proto__: Object
    ▶4: Rl {data: {…}, height: 2, depth: 1, parent: Rl, children: Array(5), …}
    ▶5: Rl {data: {…}, height: 1, depth: 1, parent: Rl, children: Array(3), …}
    ▶6: Rl {data: {…}, height: 1, depth: 1, parent: Rl, children: Array(3), …}
     length: 7
    ▶__proto__: Array(0)
   x: 423.9622641509434
   y: 0
  ▶__proto__: Object
```

Tree

- code: https://github.com/Shao-Kui/D3.js-Demos/blob/master/static/tree.html



又是Path的'd'属性…

- 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
```

又是Path的'd'属性… cont.

- d3.linkHorizontal(…)
  - d3.linkHorizontal().x(d => d.y).y(d => d.x)
  - .x(…)与.y(…)分别表示如何在source与target中取横纵坐标值

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

添加Tree的节点

- Data-Join，以<text>为例：
  - Conditional operator：d.children ? -6 : 6
  - 即如果是子节点，则向右平移6个像素，反之向左平移6个像素
  - 注意：子节点的d.children为 undefined 从而被判 false
- root.descendants()：返回层级结构中的所有节点
  - 广度优先（层次优先）
  - 返回的内容本质上是对象的数组

```
g.selectAll('text').data(root.descendants()).join('text')
.attr("text-anchor", d => d.children ? "end" : "start")
// note that if d is a child, d.children is undefined which is actually false!
.attr('x', d => (d.children ? -6 : 6) + d.y)
.attr('y', d => d.x + 5)
.text(d => d.data.name);
```

Icicle

- 前面的Tree无法可视化节点的**数值**与**节点间数值比例**的关系
- Icicle：
- 层级结构的可视化
- 反应各个子节点数值占比
- code: https://github.com/Shao-Kui/D3.js-Demos/blob/master/static/icicle.html

Icicle

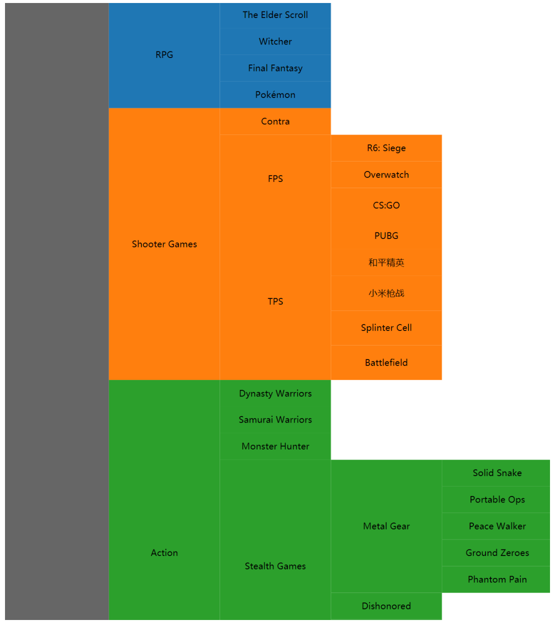
- d3.hierarchy仍作为‘中间结果’输入 d3.partition(…)
- d3.partition(…)
- d3.partition().size([height, width])
  - 返回一个函数
  - 返回函数接受的输入为d3.hierarchy
  - 自动将树形结构映射到‘**合适**’的区域

```
▼Rl {data: {…}, height: 4, depth: 0, parent: null, children: Array(7), …}
  ▶data: {name: "Games", children: Array(7)}
   height: 4
   depth: 0
   parent: null
  ▼children: Array(7)
    ▶0: Rl {data: {…}, height: 1, depth: 1, parent: Rl, children: Array(4), …}
    ▶1: Rl {data: {…}, height: 2, depth: 1, parent: Rl, children: Array(3), …}
    ▶2: Rl {data: {…}, height: 3, depth: 1, parent: Rl, children: Array(6), …}
    ▼3: Rl
      ▶data: {name: "Strategy", children: Array(4)}
       height: 2
       depth: 1
      ▶parent: Rl {data: {…}, height: 4, depth: 0, parent: null, children: Array(7), …}
      ▶children: (4) [Rl, Rl, Rl, Rl]
       value: 5910
       y0: 400
       y1: 800
       x0: 2715.5781385549635
       x1: 3814.227289626968
      ▶__proto__: Object
    ▶4: Rl {data: {…}, height: 2, depth: 1, parent: Rl, children: Array(5), …}
    ▶5: Rl {data: {…}, height: 1, depth: 1, parent: Rl, children: Array(3), …}
    ▶6: Rl {data: {…}, height: 1, depth: 1, parent: Rl, children: Array(3), …}
     length: 7
    ▶__proto__: Array(0)
   value: 32276
   y0: 0
   x0: 0
   x1: 6000
   y1: 400
  ▶__proto__: Object
```

Icicle cont.

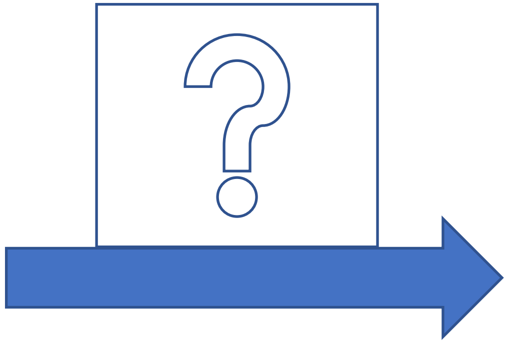
- d3.hierarchy仍作为‘中间结果’输入 d3.partition(…)并得到每个节点‘**合适**’的空间区域划分
  - 预处理后，画出主体部分仅需要一个↓Data-Join

```
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);
```



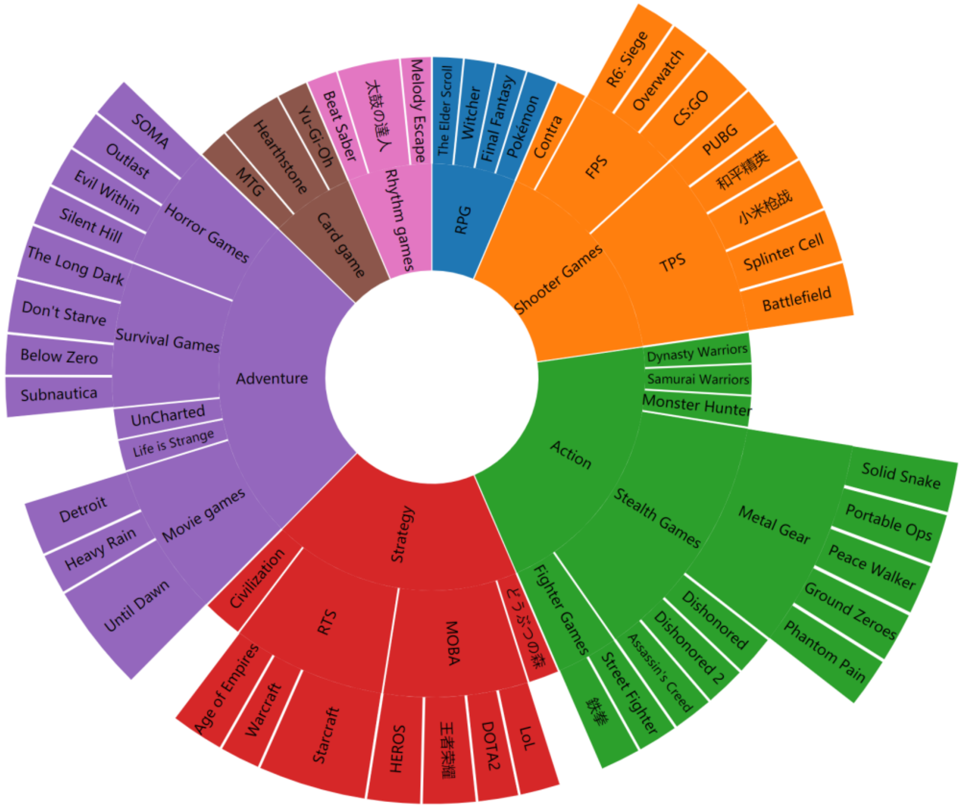
把Icicle图‘掰’成‘一圈’？…

- 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？…



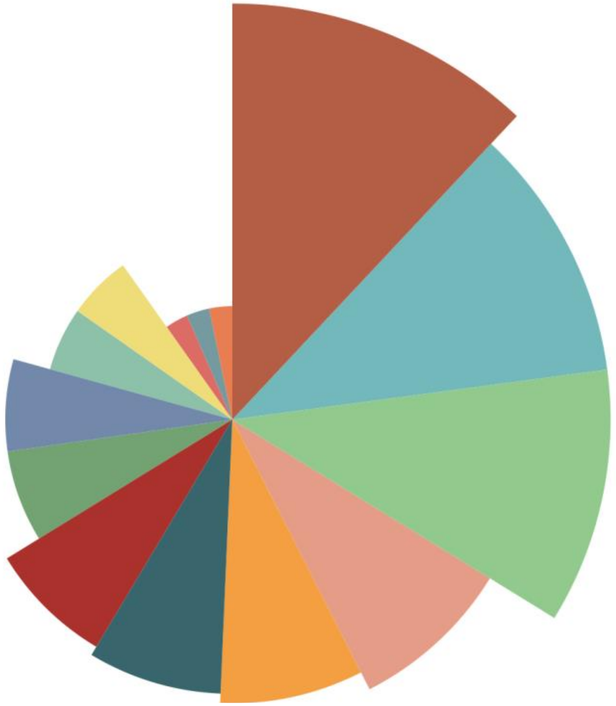
Sunburst

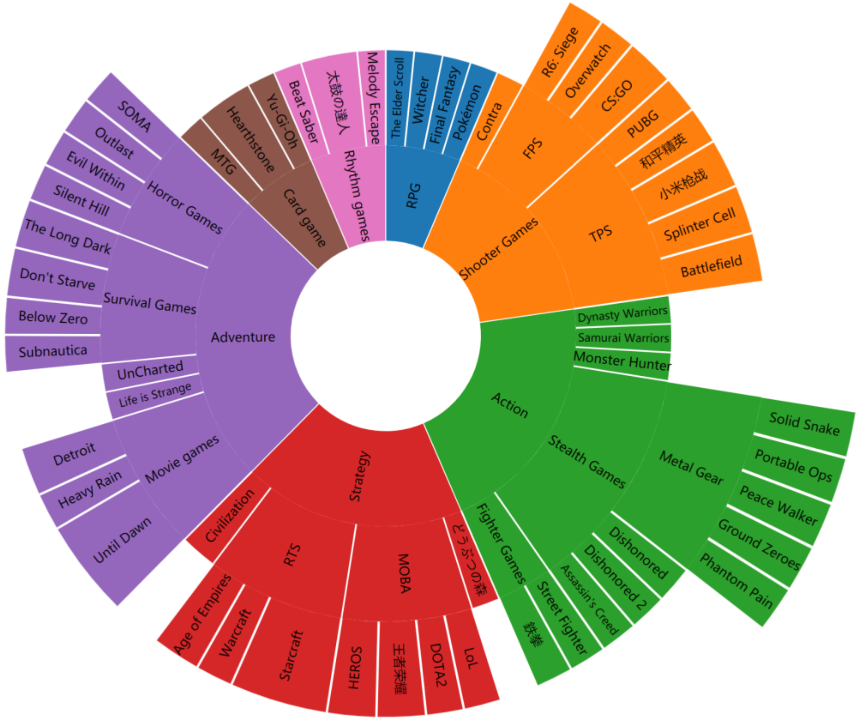
- code: https://github.com/Shao-Kui/D3.js-Demos/blob/master/static/Sunburst.html



又双叒叕是Path的'd'属性…

- d3.arc(…)
- 通过Path的d属性，绘制一个圆弧
- 常用于饼图的绘制
  - 配合d3.pie(…)
  - 饼图code: https://github.com/Shao-Kui/D3.js-Demos/blob/master/static/d3-tutorial/arc.html
- 可以设置：
  - 内半径（innerRadius）
  - 外半径（outerRadius）
  - padAngle
  - … … …





又双叒叕是Path的'd'属性… cont.

- d3.arc(…)返回给一个用于设置path的d属性的函数

```
const arc = d3.arc()
.startAngle(d => d.x0)
.endAngle(d => d.x1)
// pad distances equal to padAngle * padRadius;
// It's split into two parameters
// so that the pie generator doesn't need to concern itself with radius
.padAngle(d => Math.min((d.x1 - d.x0) / 2, 0.005))
//.padRadius(radius / 2)
.innerRadius(d => d.y0)
.outerRadius(d => d.y1)
```

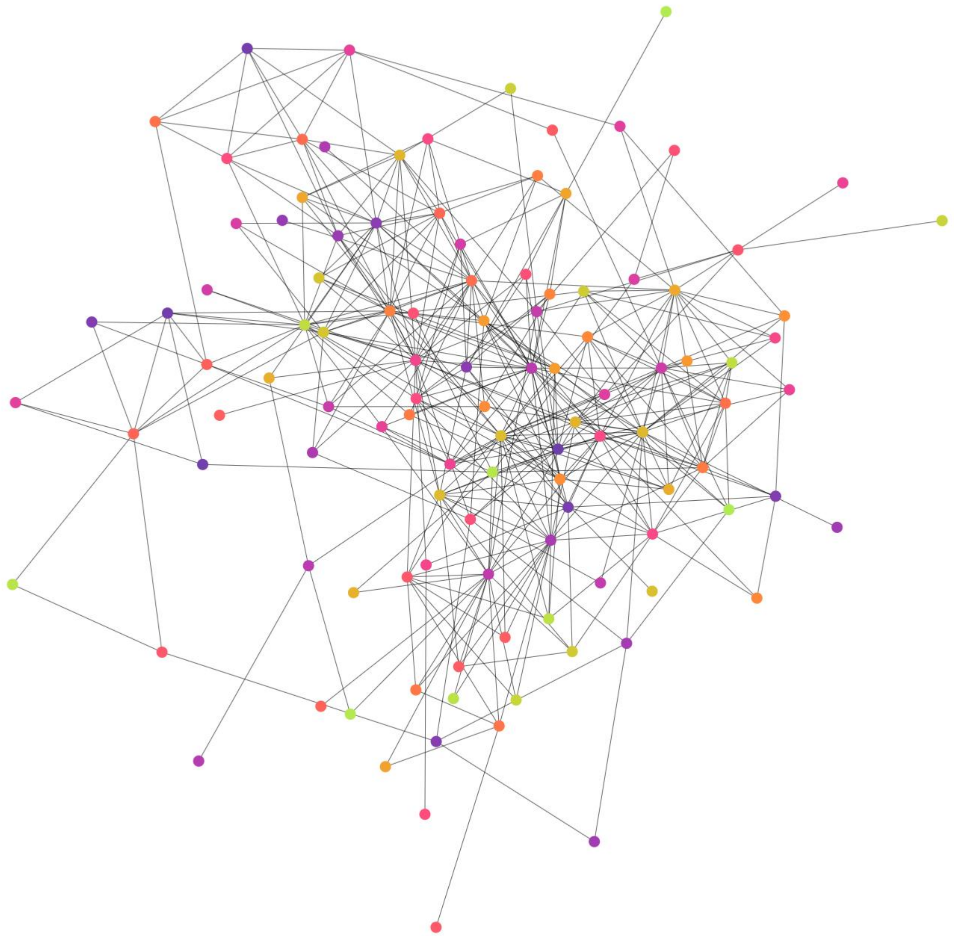
Sunburst cont.

- 得到d3.arc(…)的返回结果（函数）后，进行Data-Join

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

一般图的可视化

- code: https://github.com/Shao-Kui/D3.js-Demos/blob/master/static/graph.html



D3.js: Force


```
▶0: {index: 0, x: 738.3806527975547, y: 343.0306620165279, vy: -0.029376627447530432, vx: -0.1457
▶1: {index: 1, x: 825.4272261336539, y: 646.6813124206806, vy: -0.06945551447339872, vx: 0.021719
▶2: {index: 2, x: 690.0380469592366, y: 677.8974649507733, vy: -0.06087482519618888, vx: -0.04212
▶3: {index: 3, x: 420.41522887639513, y: 602.1439659029284, vy: -0.027755733036445917, vx: -0.123
▶4: {index: 4, x: 798.0128857957698, y: 315.8031506264l995, vy: -0.009943149939567702, vx: -0.143
▶5: {index: 5, x: 2862.020067133912, y: -48.6034103960l587, vy: -0.053645355558169884, vx: 0.2279
▶6: {index: 6, x: 629.5063624938902, y: 752.78717521969, vy: -0.024194706402747724, vx: -0.077017
```


- let nodes = [{}, {}, {}, {}, {}, {}]
- let simulation = d3.forceSimulation(nodes) 定义后会发生…
  - **补全**nodes中每个节点的数据结构
    - 包括index, x, y, vx, vy，后两者为速度
  - 开始**模拟**粒子运动
    - 粒子质量为1，单位时间加速度恒定
    - 不断地通过内部timer触发'tick'事件
  - 根据一系列的'**力**'来计算每个例子的加速度、速度、位置
    - '力'都是哪来的呢？
- 数据来源：http://networkrepository.com/socfb-Caltech36.php
  - Rossi R, Ahmed N. The network data repository with interactive graph analytics and visualization[C]//Twenty-Ninth AAAI Conference on Artificial Intelligence. 2015.

D3.js: Force 的 ‘force’

- d3.**forceManyBody**
  - 粒子之间两两的作用力
  - strength为正互相吸引，为负则互相排斥
- d3.**forceCenter**
  - 指向某一个中心的力，会尽可能让粒子向中心靠近或重合
- d3.**forceLink**
  - 粒子之间两两的作用力?
  - 让互相之间有链接的节点保持在某一个特定的距离
  - 是否有链接需要通过图的边集合给出

```
simulation = d3.forceSimulation(nodes)
.force('manyBody', d3.forceManyBody().strength(-30))
.force('center', d3.forceCenter(width / 2, height / 2))
.force("link", d3.forceLink(links).strength(0.1).distance(100))
```

d3.forceLink

- Link要通过一个数据格式给出， 即link的source与target
- 通过strength和distance两个函数分别设置期望的作用力与节点间距离

```
▼[0 … 99]
  ▶0: {source: {…}, target: {…}, index: 0}
  ▶1: {source: {…}, target: {…}, index: 1}
  ▶2: {source: {…}, target: {…}, index: 2}
  ▼3:
    ▶source: {index: 55, x: 1013.1089966584524, y: 398.58031002386207, vy: -0.00047989706956436275, vx: -0.0019270898010742773}
    ▶target: {index: 0, x: 896.3549810575581, y: 374.663789434326, vy: -0.0016396739857333313, vx: -0.0018180969751919926}
     index: 3
    ▶__proto__: Object
  ▶4: {source: {…}, target: {…}, index: 4}
  ▶5: {source: {…}, target: {…}, index: 5}
  ▶6: {source: {…}, target: {…}, index: 6}
  ▶7: {source: {…}, target: {…}, index: 7}
  ▶8: {source: {…}, target: {…}, index: 8}
  ▶9: {source: {…}, target: {…}, index: 9}
  ▶10: {source: {…}, target: {…}, index: 10}
  ▶11: {source: {…}, target: {…}, index: 11}
  ▶12: {source: {…}, target: {…}, index: 12}
  ▶13: {source: {…}, target: {…}, index: 13}
  ▶14: {source: {…}, target: {…}, index: 14}
```

Force: 时钟滴~tic~答~toc~

- forceSimulation会通过每次‘tick’来更新当前节点的状态
  - 状态包括位置、速度、加速度等
- 更新后的状态仅仅为‘状态’
  - 不会反映到任何图元
  - 仅仅对数据进行修改
- 人为设置每次tick要如何更新图元
  - simulation.on('tick', ticked);
- 在初始化每个图元后，只要为simulation配置了’tick’的回调，simulation会自动开始模拟。注意：simulation.stop()会停止timer的tick循环

```
function ticked() {
    lines
    .attr('x1', d => d.source.x)
    .attr('y1', d => d.source.y)
    .attr('x2', d => d.target.x)
    .attr('y2', d => d.target.y);
    circles
    .attr('cx', d => d.x)
    .attr('cy', d => d.y)
}
```

Tree & Graph: End

- 如何为Icicle与Sunburst添加文本？
  - Sunburst的文字添加核心在于文字的'transform':
  - rotate(${x - 90}) translate(${y},0) rotate(${x < 180? 0 : 180})
- 如何为Simulation添加图元的拖拽效果？
  - d3.drag()：注意需要手动暂定与继续模拟
- Weighted Graph？
  - code: https://github.com/Shao-Kui/D3.js-Demos/blob/master/static/force.html
  - 本质上根据link的权重设置forceLink的strength与distance
- d3.arc(…) + d3.pie(…) = ?
- 如何控制Force中的'加速度'？（动画速度）
  - e.g., simulation.alphaTarget(0.5)