目录

第 1 章　D3.js

1.1　简介

D3.js（D3 或 Data-Driven Documents）是一个使用动态图形，基于数据操作文档的，进行数据可视化的 JavaScript 程序库。D3 帮助您通过使用 HTML、SVG 和 CSS 使数据栩栩如生，产生交互式的数据展示效果——分层条形图、动画树状图、力导向图、等高线、散点图……。且 D3 提供了现代浏览器的全部功能，无需将束缚在特定框架中，可以与 Vue、React 等结合使用，提供强大的可视化组件和数据驱动的 DOM 操作方法。目前最新版本的 D3 已经更新到了 7.0 版本（截止到 2021 年 7 月）。

D3 是一个开源项目，其源码托管于 GitHub，地址为https://github.com/d3/d3，官网地址为https://d3js.org/。另外，官方的 Wiki 手册和推荐资源可在https://github.com/d3/d3/wiki中找到。

D3.js 有这样一些特点：

(1) **使用 Web 标准：**D3 是一个非常强大的可视化工具，用于创建交互式数据可视化。它利用现代网络标准：SVG、HTML 和 CSS 来创建数据可视化。

(2) **数据驱动：**D3 是数据驱动的。它可以使用静态数据或从远程服务器以不同格式（如数组、对象、CSV、JSON、XML 等）获取数据来创建不同类型的图表。

(3) **DOM 操作：**D3 允许您根据数据操作文档对象模型 (DOM)。

(4) **数据驱动元素：**它使您的数据能够动态生成元素并将样式应用于元素，表格、图形等都支持。

(5) **动态属性：**D3 可以灵活地为其大部分功能提供动态属性。属性可以指定为数据的函数。这意味着您的数据可以驱动您的样式和属性。

(6) **可视化类型：**对于 D3，尽管没有标准的可视化格式，但它允许你自由发挥，创建从 HTML 表格到饼图、图形、条形图到地理空间地图等任何内容。

(7) **自定义可视化效果：**由于 D3 使用 Web 标准，因此您可以完全控制可视化功能。

(8) **交互和动画：**D3 通过 `duration()`、`delay()` 和`ease()` 等函数为动画提供了很好的支持，能快速响应用户交互的需要。

如图 1-1 、图 1-2 、图 1-3 、图 1-4 中所示，这些都是用 D3.js 所绘制出的交互式

数据可视化图表。

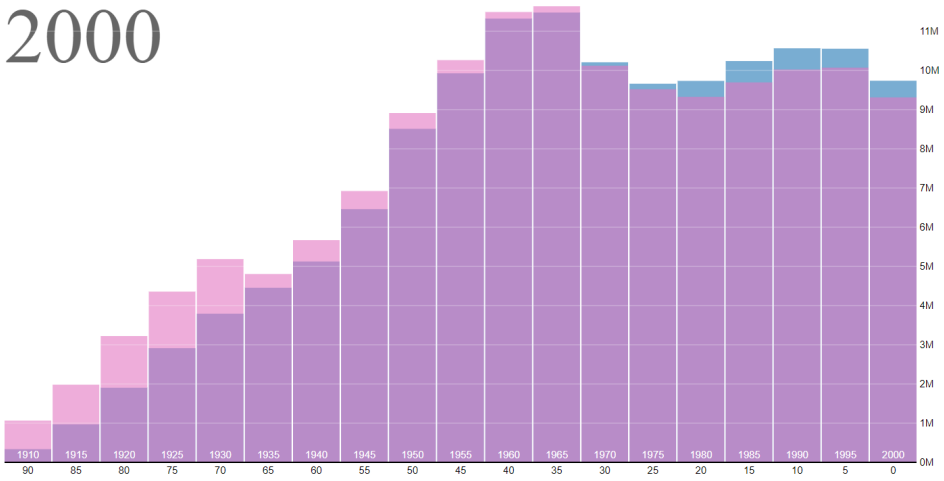

图 1-1　示例条形图

1.2　安装

D3 作为一个 JavaScript 函数库，其实并不是标题中所说的“安装”，更准确地说是“导入”。它只有单文件，在 HTML 中引用即可。有两种方法：

方法一：从官网处下载 D3.js 的压缩包文件并解压。

当前官网可下载到最新 7.0.0 版本的 D3，链接为https://registry.npmjs.org/d3/-/d3-7.0.0.tgz。

解压后，在 HTML 中导入相关的 js 文件即可使用 D3。（`package/dist` 文件夹下的 `d3.js` 或 `d3.min.js`，其中含 min 的文件为压缩后版本）

方法二：直接通过网络上的 D3 地址，引用链接到 HTML 中。

```
1 <script src="https://d3js.org/d3.v7.min.js"></script>
```

注意应用方法二时，需要保证使用 D3 时网络畅通。

注：本章节中所使用到的 D3 版本为 7.0.0 版本。

1.3　预备知识和工具

D3 是作为 JavaScript 语言编写的程序库，常应用并展示于网页浏览器之中，故离不开制作网页相关的技术栈，主要包括了这样一些预备知识：

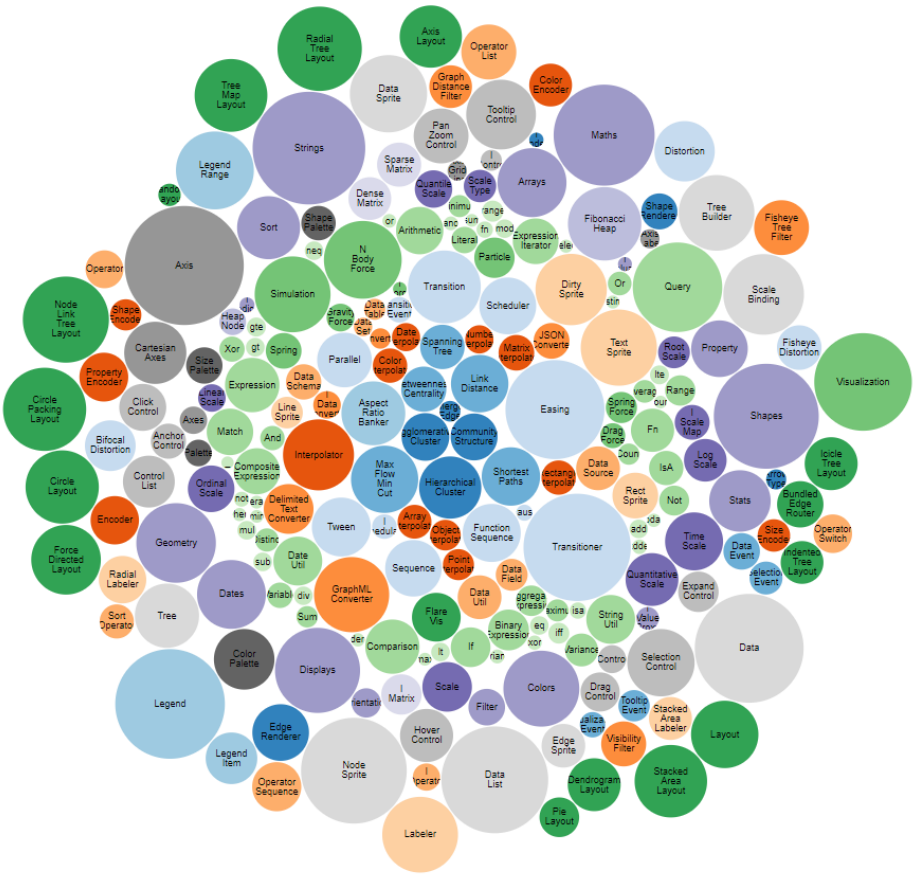
Radial Tree Layout
Axis Layout
Operator List
Tree Map Layout
Data Sprite
Graph Distance Filter
Color Encoder
Strings
Tooltip Control
Maths
Pan Zoom Control
Distortion
Legend Range
Sparse Matrix
Arrays
Dense Matrix
Quantile Scale
Tree Builder
Fisheye Tree Filter
Sort
Fibonacci Heap
Axis
Arithmetic
Expression Iterator
Operator
N Body Force
Particle
Simulation
Transition
Scheduler
Dirty Sprite
Query
Scale Binding
Node Link Tree Layout
Cartesian Axes
Text Sprite
Property
Fisheye Distortion
Visualization
Property Encoder
Expression
Parallel
Spanning Tree
Link Distance
Easing
Shapes
Circle Packing Layout
Click Control
Aspect Ratio Banker
Bifocal Distortion
Anchor Control
Match
Interpolator
Community Structure
Circle Layout
Control List
Max Flow Min Cut
Hierarchical Cluster
Shortest Paths
Data Source
Log Scale
Stats
Encoder
Ordinal Scale
Delimited Text Converter
Tween
Function Sequence
Transitioner
Time Scale
Force Directed Layout
Geometry
Sequence
Data Event
Radial Labeler
Dates
GraphML Converter
Data Field
Quantitative Scale
Expand Control
Tree
Flare Vis
String Util
Data
Color Palette
Displays
Comparison
Selection Control
Legend
Matrix
Scale
Colors
Drag Control
Filter
Stacked Area Labeler
Layout
Edge Renderer
Hover Control
Edge Sprite
Visibility Filter
Legend Item
Node Sprite
Data List
Dendrogram Layout
Stacked Area Layout
Operator Sequence
Labeler
File Layout

图 1-2　示例气泡图

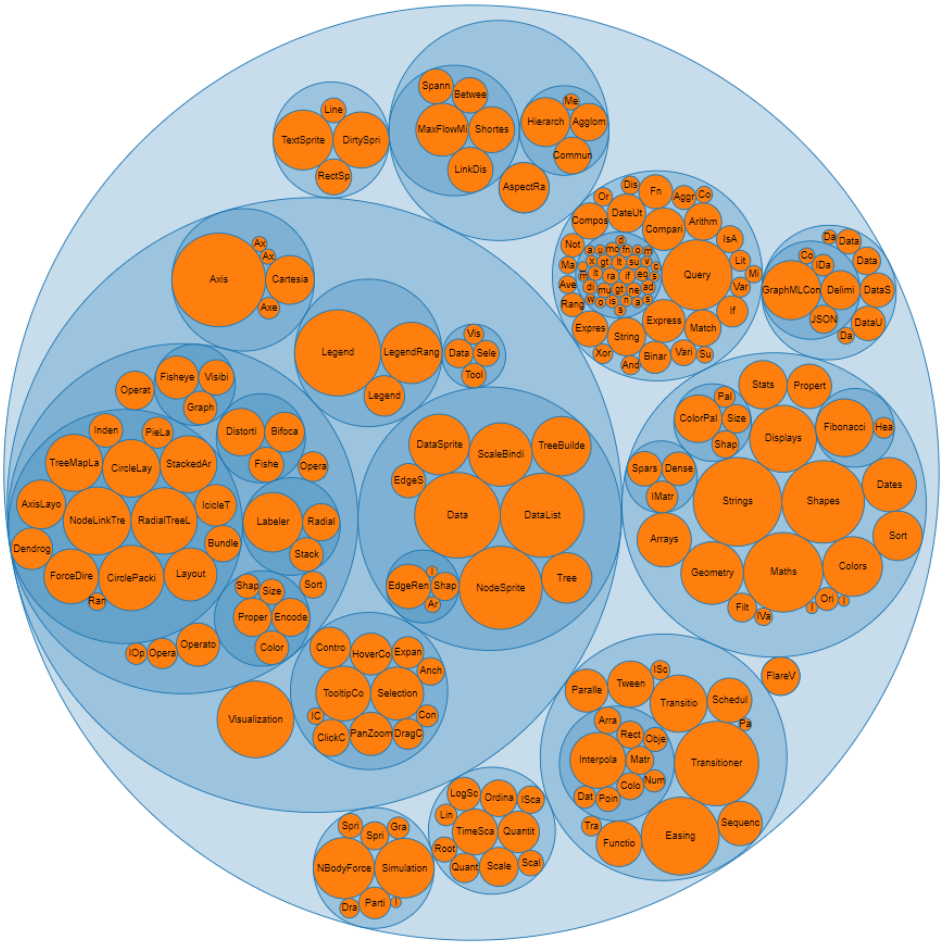

图 1-3　示例圈层图

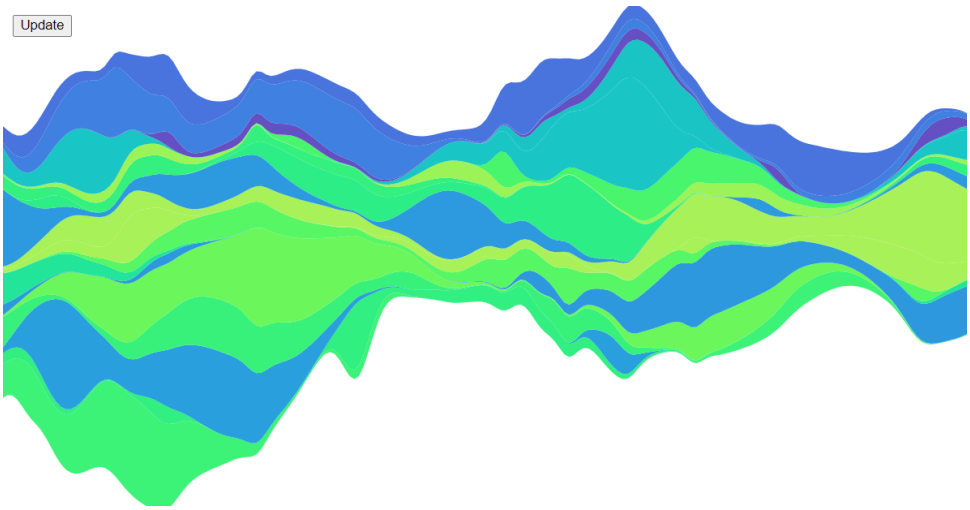

图 1-4　示例流图

(1) HTML：超文本标记语言，用于设定网页的内容

(2) CSS：层叠样式表，用于设定网页的样式

(3) JavaScript：一种直译式脚本语言，用于设定网页的行为

(4) DOM：文档对象模型，用于修改文档的内容和结构

(5) SVG：可缩放矢量图形，用于绘制可视化的图形

本章节对 D3 的应用不需要对这以上几个技术有很深的了解，大概知道其作用即可。我们会通过 D3 的案例简单地综合运用，重点则关注于 D3 的使用和操作上。

D3 可以认为是一套含有可视化代码片段模板的程序库，操作 D3 的数据可视化应用还要通过编程写代码来实现。通常以制作网页的形式来呈现出 D3 可视化的数据。在此推荐使用几个制作网页时会用到的 D3 编程相关工具。

(1) 编辑器：Visual Studio Code（VS Code），微软出品的非常强大且流行的编辑器，在前端开发者中广泛使用。配合插件，开发体验非常好。

(2) 浏览器：Chrome、Firefox、Edge 等主流浏览器均可。

1.4 HTML 模板和导入 D3

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>D3.js</title>
  </head>
  <body>
    <p>Hello, World! --1</p>
    <p>Hello, World! --2</p>
  </body>
</html>
```

这便是最基础的一个 HTML 文件，可将其命名并保存为`index.html`，在浏览器中打开，可知会打印出两行 “Hello, World!”。但现在还没有引入 D3，我们要怎么样引入 D3，并让 D3 对该 HTML 发挥作用呢?

这里,我将之前下载安装好的`d3.js`或`d3.min.js`存放在新建的`js`文件夹下,而`index.html`与该文件夹同级。即文件目录情况为:

```
|-  index.html
|-  js/
```

```
        d3.js
        d3.min.js
```

将文件安置在相对目录后可从 VS Code 中打开项目进行编辑。如下为引入了 D3 的 HTML 文件`index.js`，只需通过添加一行`<script src="xxx">`来引入即可，`src=`可指定 D3.js 的存放位置。如我在这里使用的是`d3.min.js`, 通过相对目录`./js/d3.min.js`引入。

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>D3.js</title>

    <!-- 引入 D3 -->
    <script src="./js/d3.min.js" charset="utf-8"></script>

  </head>
  <body>
    <p>Hello World! --1</p>
    <p>Hello World! --2</p>
  </body>
</html>

<script>
    var e = d3.select("body").selectAll("p");
    e.style("color","blue").style("font-size","72px");
</script>
```

如我使用 VS Code 打开该项目，并安装了 “Preview on Web Server” 插件，便可实时地在侧边栏中看到该 HTML 的展示效果，非常方便。如图 1-5 中所示。通常，在编写 Web 应用时会利用 `console.log()` 等方法进行调试，则可选择在浏览器中打开，进入到浏览器的开发者调试模式（通常通过 F12 进入），查看 console 输出、HTML 元素信息等。

以上导入了 D3 对 p 标签文本进行操作后的效果，两行 “Hello, World!” 变为了蓝色，且字号变大了许多。这是通过用 D3 的 script 脚本来控制的，而这些 script 我在以上代码片段中写在了该 html 文件的末尾被`<script>...</script>`标签（HTML 中标签总是成对出现）包围起来的部分，也可以单独保存为一个 js 文件，通过类似于 D3 导入的方式引用。

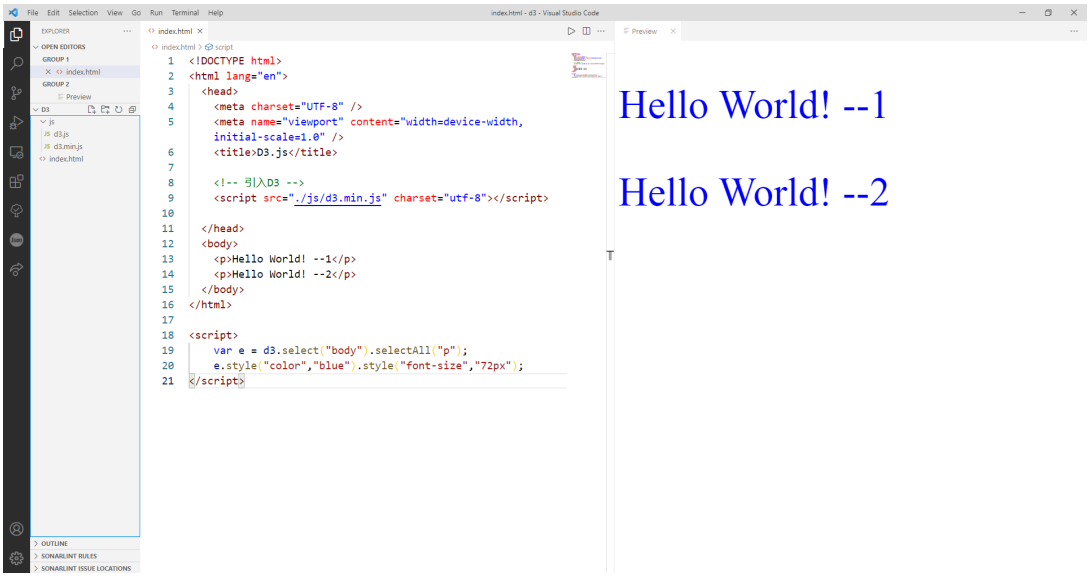

图 1-5　在 VS Code 中打开引入了 D3 操作的项目

1.5　元素选择和数据绑定

我们再观察上述<script>中的 D3 代码：

```
var e = d3.select("body").selectAll("p");
// 选择所有 <p> 标签的网页元素
e.style("color","blue").style("font-size","72px");
// 将颜色样式改为 blue，用链式语法继续将文本大小修改为 72px
```

1.5.1　元素选择

D3 可以非常简洁地操作 HTML 中的 DOM 元素，我们通过d3.select()（选择第一个找到的元素）或 d3.selectAll()（选择所有找到的元素）选择元素后返回了对象，这就是选择集。我们可以根据元素的不同特性来选择出想要的对象，根据属性值、class、ID 等都可以进行选择。

而多次连续调用的.style()等函数被称为链式语法，和 JQuery 中的语法颇为类似。此处我们调用.style()改变了元素的样式，而 D3 还可以提供设置属性（.attr()）、添加（.append()）、更改文本内容（.text()）等方法，能满足用户大部分的需求。

1.5.2 数据绑定

D3 可以将数据绑定到 DOM 上去（DOM 能将 HTML 文档表达为树结构，数据绑定与 DOM 绑定即是让 HTML 标签与数据进行绑定）。比如，让的段落元素 p 标签与字符串变量 “Hello” 绑定，绑定后，当需要依靠该数据操作元素时，会更为方便。

D3 中有两个函数可以绑定数据：

(1) `datum()`：绑定一个数据到选择集上。

(2) `data()`:绑定一个数组到选择集上,数组的各项值分别与选择集的各元素绑定。(更常用)

举一个例子，当前有三个段落元素如下：

```
<body>
    <p> 张三 </p>
    <p> 李四 </p>
    <p> 王五 </p>
</body>
```

方式一：使用`datum()`绑定。假设有一个字符串 “China”，可将其分别与三个段落 p 元素绑定：

```
var str = "China";
var body = d3.select("body");
var p = body.selectAll("p");
p.datum(str);
p.text(function(d, i){
    return i +": "+ d;
});
```

绑定数据后，使用此数据来修改三个段落元素的内容，其结果为：

```
0: China
1: China
2: China
```

在上面的代码中，用到了一个匿名函数`function(d, i)`。当选择集需要使用被绑定的数据时，常需要这么使用。其包含两个参数，其中 d 代表数据，即与某元素绑定的数据，而 i 代表索引，代表数据的索引号，从 0 开始。

方式一：使用data()绑定。有一个数组var arr = ["a", "b", "c"];，接下来要分别将数组的各元素绑定到三个段落元素上。

绑定后，其对应关系应为张三-a，李四-b，王五-c。我们调用data()函数绑定数据，并替换三个段落元素的字符串为被绑定的字符串，代码如下：

```
var arr = ["a", "b", "c"];
var body = d3.select("body");
var p = body.selectAll("p");
p.data(arr).text(function (d, i) {
  return d;
});
```

此处也用到了一个无名函数 function(d, i)，其对应的情况为 i=0, 1, 2 时，d 分别为 a, b, c。

此时，三个段落 p 元素与数组 arr 的三个字符串是一一对应的。因此，在函数 function(d, i) 直接 return d 即可。

1.6 插入和删除元素

1.6.1 插入元素

插入元素涉及的函数有两个，分别是：

(1) append()：在选择集末尾插入元素

(2) insert()：在选择集前面插入元素

假设有和前文一样的三个段落 p 元素：张三、李四、王五，其中给李四用 id 加了标签id="label"。

```
<body>
    <p> 张三 </p>
    <p id="label"> 李四 </p>
    <p> 王五 </p>
</body>
```

```
var body = d3.select("body");
body.append("p").text(" 赵六")
// 在 body 的末尾 append p 元素
```

```
body.insert("p","#label").text("insert here")
// 找到 id 为 label 的标签 p, insert 插入元素
```

最终结果为:

```
张三
insert here
李四
王五
赵六
```

与预想的一致，即append()在末尾插入，insert()在元素前插入。

1.6.2 删除元素

删除一个元素时，对于选择的元素，使用 remove() 函数即可，例如:

```
var p = body.select("#label");
p.remove();
```

于是，便删除了指定 id 的段落元素。

1.7 enter() 和 exit() 方法

使用 D3 中的enter()和exit()对象选择方法，可以为传入的数据（通常为数组形式）创建新节点，以及删除不再需要的传出节点。

当数据绑定到选择集上后，数据数组中的每个元素都与选择中的相应节点配对。如果节点数少于数据的长度（即该数组的长度），则额外的数据元素可通过enter()选择，附加进节点。如果节点数多于数据的长度，通过exit()选择，多余的节点会被删除。下面通过代码来说明。

假设当前有 2 个段落标签，其中的内容分别是 a 和 b。

```
<body>
    <p>a</p>
    <p>b</p>
</body>
```

我们考察以下三种情况：不使用enter()和exit()，使用enter()，和使用exit()来观察其它们两个的作用。

```
// Case 1: Update
var p =  d3.select("body")
  .selectAll("p")
  .data([1, 2, 3, 4])
    .text(function(d) { return d; });
// 结果为 1 2
// 即 a 和 b 被替换成了数组的中的前两个元素

// Case 2: Enter
p.enter().append("p")
    .text(function(d) { return d; });
// 结果为 1 2 3 4
// 在 Case 1 的基础上继续操作
// 数据长度大于节点数，通过 enter()，3 和 4 成为了附加节点

// Case 3: Exit
d3.select("body")
  .selectAll("p")
  .data([8])
  .exit().remove()
    .text(function(d) { return d; });
// 结果为 1
// 在 Case 2 的基础上继续操作
// 数据长度小于节点数，通过 exit()，删除了多余节点，留下了 1
// 注：如果先.text，再.exit().remove()，则会因为先赋值了而结果为 8
```

通过分别处理这三种情况，我们可以观察和分析了解了这二者的作用。可以认识到各个操作具体作用于了哪些节点。这在绘制图形时可以得到应用——如对于条形图，我们可能先使用了旧的尺度来初始化了输入的条的个数，当遇到新的输入导致数据长度与初始化时的条数不一致时，就可以通过这二者来进行更新。

1.8 绘制 SVG 图形

前面我们所处理对象都是 HTML 的文字，没有涉及图形的制作。若要进行绘图，首要需要的是一块绘图的“画布”。在 HTML 5 中，提供了两种强有力的“画布”：SVG 和 Canvas。其中，D3 对 SVG 的支持非常好，提供了众多的 SVG 图形的生成器。本节中，我们将用 SVG 绘制简单的条形图为例来了解 D3 是如何操作 SVG 图形的。

1.8.1 什么是 SVG

SVG，指可缩放矢量图形（Scalable Vector Graphics），是用于描述二维矢量图形的一种图形格式，是由万维网联盟制定的开放标准。SVG 使用 XML 格式来定义图形，除了 IE8 之前的版本外，绝大部分浏览器都支持 SVG，可将 SVG 文本直接嵌入 HTML 中显示。

SVG 有如下特点：

(1) SVG 绘制的是矢量图，因此对图像进行放大不会失真。

(2) 基于 XML，可以为每个元素添加 JavaScript 事件处理器。

(3) 每个图形均视为对象，更改对象的属性，图形也会改变。

1.8.2 添加画布

使用 D3 在 body 元素中添加 SVG 画布的代码如下：

```
var width = 300;  // 画布的宽度
var height = 300;   // 画布的高度
var svg = d3.select("body") // 选择文档中的 body 元素
    .append("svg") // 添加一个 SVG 元素
    .attr("width", width) // 设定宽度
    .attr("height", height); // 设定高度
```

有了画布，接下来就可以在画布上作图了。

1.8.3 绘制矩形

在 SVG 中，矩形的元素标签是 rect，在 HTML 中可作为标签使用。

```
<svg>
    <rect></rect>
    <rect></rect>
</svg>
```

上面的 rect 里没有矩形的属性。矩形的属性，常用的有四个：

(1) x：矩形左上角的 x 坐标

(2) y：矩形左上角的 y 坐标

(3) width：矩形的宽度

(4) height：矩形的高度

要注意的是，在 SVG 中，x 轴的正方向是水平向右，y 轴的正方向是垂直向下的。

现在，我们给出一组数据（截止到 2021 年 7 月 31 日的东京奥运会金牌榜前五名的金牌数）对此进行可视化。数据如下：

```
var arr = [21, 17, 16, 11, 10];  // 数据（表示矩形的宽度）
```

我们直接将数值的大小作为矩形的宽度，然后添加以下代码：

```
var rectHeight = 25; // 每个矩形所占的像素高度（包括空白）
svg.selectAll("rect")
    .data(arr)
    .enter()
    .append("rect")
    .attr("x",20)
    .attr("y",function(d,i){
        return i * rectHeight;
    }) // 为各元素的属性赋值
    .attr("width",function(d){
        return d*10;
        // 为了显示效果对数值进行了缩放
        // 更好的方法是通过设置比例尺来优化
    })
    .attr("height",rectHeight-5)
    .attr("fill","gold"); // 设置填充色为金色
```

其中便应用到了enter()方法，它使得在有数据，而没有足够图形元素的情况下，补充足够的元素。

最终结果如图 1-6 所示。

1.8.4 使用比例尺

在先前的代码注释中，提到了比例尺的概念。比例尺中有线性比例尺（连续）、序数比例尺（离散）等多种类型之分。其中线性比例尺能使数值从一个连续的区间（定义域 domain）映射到另一个区间（值域 range），来解决条形图宽度的问题。

仍然对于金牌榜的数据，我们应用以下代码来使用线性比例尺：

图 1-6　金牌榜条形图

```
var arr = [21, 17, 16, 11, 10];
var max = d3.max(arr);
var linear = d3.scaleLinear()
        .domain([0, max]) // 定义域
        .range([0, 300]); // 值域
```

其中d3.scaleLinear() 的返回值，可被当做函数来使用。因此，有如这样的用法: linear(0.9)去调用该比例尺,于是先前代码中的return d*10;可替换成return linear(d);

1.9　坐标轴

1.9.1　类型简介

坐标轴，是可视化图表中经常出现的一种图形，由轴线、刻度和标签组成，可以分为水平的 x 轴和垂直方向上的 y 轴。D3 支持了以下四种绘制坐标轴的函数，使用起来很方便。

(1) d3.axisTop(): 创建顶部坐标轴

(2) d3.axisRight(): 创建垂直居右坐标轴

(3) d3.axisBottom(): 创建底部坐标轴

(4) d3.axisLeft(): 创建垂直居左坐标轴

1.9.2 x 轴坐标轴

```
<script>
  // 将 SVG 画布的宽高定义成变量
  var width = 400,
    height = 100;

  // 需要刻画的数据
  var data = [10, 15, 20, 25, 30];

  // 添加 SVG 画布
  var svg = d3
    .select("body")
    .append("svg")
    .attr("width", width)
    .attr("height", height);

  // 创建线性比例尺
  // 设置其宽高、定义域值域
  // 从定义域到值域：10->0, 30->300
  var scale = d3
    .scaleLinear()
    .domain([d3.min(data), d3.max(data)])
    .range([0, width - 100]);

  // 创建横向底部的 x 轴，并向 x 轴添加比例尺
  var x_axis = d3.axisBottom().scale(scale);

  // 创建“组”并向其中插入 x 轴坐标
  svg.append("g").call(x_axis);
</script>
```

运行这个例子，可以观察到已作出了一个 x 轴坐标轴。如图 1-7 为其渲染出的形状。

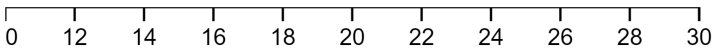

图 1-7 D3 中绘制 x 轴坐标轴

对应生成的 HTML 代码如下：

```
<svg width="400" height="100">
  <g fill="none" font-size="10" font-family="sans-serif" text-anchor="middle">
    <path class="domain" stroke="currentColor" d="M0,6V0H300V6"></path>
    <g class="tick" opacity="1" transform="translate(0,0)">
```

```
      <line stroke="currentColor" y2="6"></line>
      <text fill="currentColor" y="9" dy="0.71em">10</text>
    </g>
    <g class="tick" opacity="1" transform="translate(30,0)">
      <line stroke="currentColor" y2="6"></line>
      <text fill="currentColor" y="9" dy="0.71em">12</text>
    </g>
    ...
  </g>
</svg>
```

1.9.3 y 轴坐标轴

类似地，我们也可以创建垂直方向上的轴，代码如下：

```
<script>
  var width = 400,
    height = 400;

  var data = [10, 15, 20, 25, 30];
  var svg = d3
    .select("body")
    .append("svg")
    .attr("width", width)
    .attr("height", height);

  var scale = d3
    .scaleLinear()
    .domain([d3.min(data), d3.max(data)])
    .range([height / 2, 0]);

  var y_axis = d3.axisLeft().scale(scale);

  svg.append("g").attr("transform", "translate(50, 10)").call(y_axis);
  // translate transform 操作调整了坐标轴在 SVG 图中的位置
</script>
```

运行后的效果如图 1-8 所示。

1.9.4 同时包含 x 轴和 y 轴坐标轴

现在我们可以把 x 轴和 y 轴并在一张图里，代码如下，效果见图 1-9 所示。

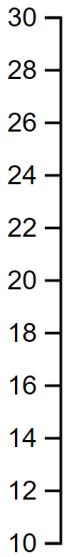

图 1-8　D3 中绘制 y 轴坐标轴

```
<script>
  var width = 400,
    height = 400;
  var data = [10, 15, 20, 25, 30];

  var svg = d3
    .select("body")
    .append("svg")
    .attr("width", width)
    .attr("height", height);

  var xscale = d3
    .scaleLinear()
    .domain([0, d3.max(data)])
    .range([0, width - 100]);

  var yscale = d3
    .scaleLinear()
    .domain([0, d3.max(data)])
    .range([height / 2, 0]);

  var x_axis = d3.axisBottom().scale(xscale);

  var y_axis = d3.axisLeft().scale(yscale);

  svg.append("g").attr("transform", "translate(50, 10)").call(y_axis);

  var xAxisTranslate = height / 2 + 10;

```

```
  svg
    .append("g")
    .attr("transform", "translate(50, " + xAxisTranslate + ")")
    .call(x_axis);
</script>
```

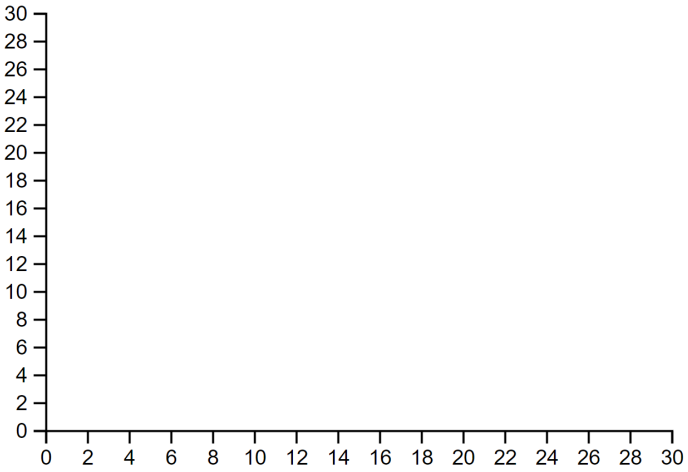

图 1-9　D3 中同时绘制出 x 轴和 y 轴坐标轴

1.10　条形图

此前，我们已经学会了如何用 SVG 绘制简单的矩形、创建比例尺和坐标轴等。现在，我们可以综合运用起来画出一个带有完整信息的条形图。

我们模拟 X 公司在 2011 年至 2016 年期间的股票价格作为数据集，并为它作出条形图实现数据可视化。数据保存在与 HTML 文件同级目录下，命名为“data.csv”，内容如下：

```
year,value
2011,45
2012,47
2013,52
2014,70
2015,75
2016,78
```

接下来，我们将用这份数据创建垂直方向的条形图。

1.10.1 建立画布并定义比例尺

在 HTML 标签中的 body 部分建立好 SVG 画布空间：

```
<body>
    <svg width="600" height="500"></svg>
</body>
```

在<script>脚本中，为 SVG 画布定义宽高等，并分别为 x 轴和 y 轴创建比例尺，设定了比例尺的值域。

```
<script>
  var svg = d3.select("svg"),
    margin = 200, // 通过 margin 外边距调整位置
    width = svg.attr("width") - margin,
    height = svg.attr("height") - margin;

  // scaleBand() 序数比例尺常用于离散值，如年份
  // padding 用于调整条之间的距离
  var xScale = d3.scaleBand().range([0, width]).padding(0.4),
    yScale = d3.scaleLinear().range([height, 0]);

  // 创建“组”元素，调整了图表在 SVG 中的位置
  var g = svg
    .append("g")
    .attr("transform", "translate(" + 100 + "," + 100 + ")");
</script>
```

1.10.2 加载数据并创建坐标轴

在上述<script>的代码中，继续添加以下这些部分：

```
d3.csv("data.csv").then(function (data) {
  xScale.domain(
    data.map(function (d) {
      return d.year;
    })
  );
  yScale.domain([
    0,
    d3.max(data, function (d) {
      return d.value;
```

```
    }),
  ]);

  g.append("g")
    .attr("transform", "translate(0," + height + ")")
    .call(d3.axisBottom(xScale));

  g.append("g")
    .call(
      d3
        .axisLeft(yScale)
        .tickFormat(function (d) {
          return "$" + d;
        })
        .ticks(10)
    )
});
```

我们来一点点拆解这里新增的代码：

```
d3.csv("data.csv").then(function(data) {
  // ...
});
```

这一步使用了d3.csv()方法加载了数据集data.csv。然后，我们可以为 x 轴和 y 轴上的比例尺继续添加定义域的范围（在上一步中已经给出了值域）。

```
// 使用 data.map() 映射离散的年份值给 x 比例尺
xScale.domain(
data.map(function (d) {
  return d.year;
})
);
// 对于 y 轴，使用 d3.max() 将定义域设为 [0, max]
yScale.domain([
0,
d3.max(data, function (d) {
  return d.value;
}),
]);
```

通过创建组（g）元素，将 x 轴添进组元素中，然后我们用transform属性来调整其在 SVG 画布中的位置居底部，并调用了d3.axisBottom(xScale)插入了 x 轴坐标轴。

```
g.append("g")
 .attr("transform", "translate(0," + height + ")")
 .call(d3.axisBottom(xScale));
```

同样地，我们调用axisLeft()在组元素中创建了 y 轴坐标轴。由于 y 轴上的元素是股价，我们可以格式化添加$的前缀，同时使用了ticks()方法来指定 y 轴大致有多少个区间。

```
.tickFormat(function (d) {
  return "$" + d;
})
.ticks(10)
```

截止到这里，如图 1-10 所示，我们已经作出了与数据相适应的坐标系。

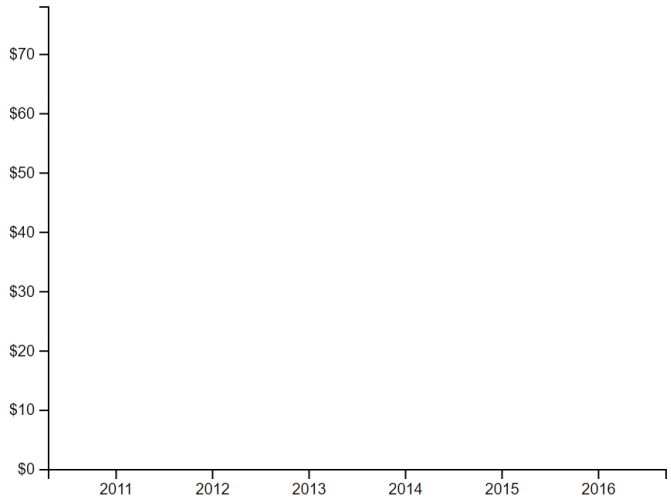

图 1-10　x、y 轴坐标系建立完成

1.10.3　条形绘制

然后，我们可以在轴上根据数据创建条形。见如下代码（添加在.csv(){}函数内部），我在此通过注释方式来讲解。

```
g.selectAll(".bar") // 选择所有 class 为 bar 的元素
  .data(data)
  .enter()
```

```
  .append("rect") // 使用 enter() 绑定数据
  .attr("class", "bar") // 添加 class 属性
  .attr("x", function (d) {
    return xScale(d.year);
  }) // x 轴坐标为年份
  .attr("y", function (d) {
    return yScale(d.value);
  }) // y 轴坐标为股票价格
  .attr("width", xScale.bandwidth())
  // 和 x 轴下的 scaleBand() 对应
  .attr("height", function (d) {
    return height - yScale(d.value);
  })
  .attr("fill", "grey");
  // 使用灰色填充，也可以根据.bar 写 CSS 样式来改变颜色
```

此时的效果如图 1-11 所示。

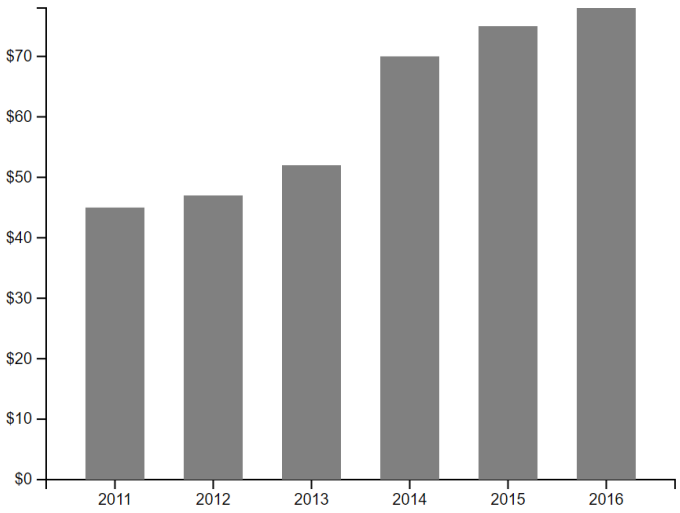

图 1-11　添加了条形

1.10.4　添加标签

我们还需要为图表添加标签，如标题、坐标轴上的单位等。使用以下代码添加条形图位于正上方的标题：

```
svg
  .append("text")
  .attr("transform", "translate(100,0)")
```

```
  .attr("x", 50)
  .attr("y", 50)
  .attr("font-size", "24px")
  .text("X 公司股票价格");
```

找到此前为 x 轴和 y 轴分别使用axisBottom()和axisLeft()方法创建坐标轴的代码段，修改为：

```
g.append("g")
  .attr("transform", "translate(0," + height + ")")
  .call(d3.axisBottom(xScale))
  .append("text")
  .attr("y", height - 250)
  .attr("x", width - 100)
  .attr("text-anchor", "end")
  .attr("stroke", "black")
  .text(" 年份");

g.append("g")
  .call(
    d3
      .axisLeft(yScale)
      .tickFormat(function (d) {
        return "$" + d;
      })
      .ticks(10)
  )
  .append("text")
  .attr("transform", "rotate(-90)")
  .attr("y", 6)
  .attr("dy", "-5.1em")
  .attr("text-anchor", "end")
  .attr("stroke", "black")
  .text(" 股价");
```

至此，条形图绘制完成，效果如图 1-12 所示。

以下附上完整代码：

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>D3.js</title>
```

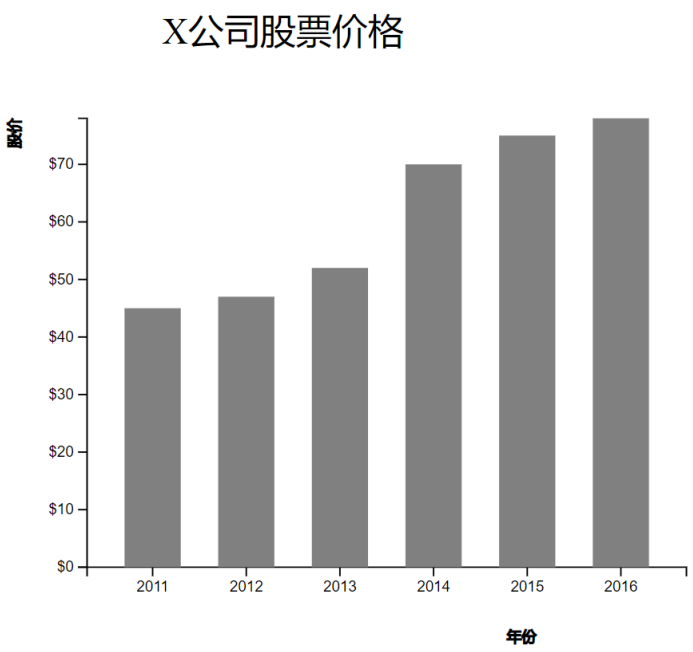

图 1-12　完整条形图

```
    <!-- 引入 D3 -->
    <script src="./js/d3.min.js" charset="utf-8"></script>
  </head>

  <body>
    <svg width="600" height="500"></svg>
  </body>
</html>

<script>
  var svg = d3.select("svg"),
    margin = 200,
    width = svg.attr("width") - margin,
    height = svg.attr("height") - margin;

  var xScale = d3.scaleBand().range([0, width]).padding(0.4),
    yScale = d3.scaleLinear().range([height, 0]);

  var g = svg
    .append("g")
    .attr("transform", "translate(" + 100 + "," + 100 + ")");

  d3.csv("data.csv").then(function (data) {
    xScale.domain(
      data.map(function (d) {
        return d.year;
      })
```

```
34      );
35      yScale.domain([
36        0,
37        d3.max(data, function (d) {
38          return d.value;
39        }),
40      ]);
41
42      g.append("g")
43        .attr("transform", "translate(0," + height + ")")
44        .call(d3.axisBottom(xScale))
45        .append("text")
46        .attr("y", height - 250)
47        .attr("x", width - 100)
48        .attr("text-anchor", "end")
49        .attr("stroke", "black")
50        .text(" 年份");
51
52      g.append("g")
53        .call(
54          d3
55            .axisLeft(yScale)
56            .tickFormat(function (d) {
57              return "$" + d;
58            })
59            .ticks(10)
60        )
61        .append("text")
62        .attr("transform", "rotate(-90)")
63        .attr("y", 6)
64        .attr("dy", "-5.1em")
65        .attr("text-anchor", "end")
66        .attr("stroke", "black")
67        .text(" 股价");
68
69      g.selectAll(".bar")
70        .data(data)
71        .enter()
72        .append("rect")
73        .attr("class", "bar")
74        .attr("x", function (d) {
75          return xScale(d.year);
76        })
77        .attr("y", function (d) {
78          return yScale(d.value);
79        })
80        .attr("width", xScale.bandwidth())
81        .attr("height", function (d) {
```

```
82          return height - yScale(d.value);
83        })
84        .attr("fill", "grey");
85    });
86
87    svg
88      .append("text")
89      .attr("transform", "translate(100,0)")
90      .attr("x", 50)
91      .attr("y", 50)
92      .attr("font-size", "24px")
93      .text("X 公司股票价格");
94  </script>
```

1.11 饼图

在这一节中，我们将学习如何使用 D3 来绘制饼图做数据可视化。我们会用到以下几个方法:

(1) SVG 路径：使用预定义的命令创建 SVG 路径

(2) d3.scaleOrdinal()：创建序数比例尺

(3) d3.pie()：饼图生成器

(4) d3.arc()：弧生成器

1.11.1 SVG 路径

路径元素用于在 SVG 上创建路径。我们可以使用命令在 SVG 中绘制出路径。

```
1  <body>
2      <svg height="210" width="400">
3          <path d="M150 0 L75 200 L225 200 Z" />
4      </svg>
5  </body>
```

如上述代码定义了一条从起点 (150, 0) 开始，经过 (75, 200), (225, 200) 的路径，并在起点处汇合。见图 1-13 所示。

1.11.2 d3.scaleOrdinal()

我们在此前也学习过了比例尺的相关概念,此处我们会新使用到序数比例尺d3.scaleOrdinal()。

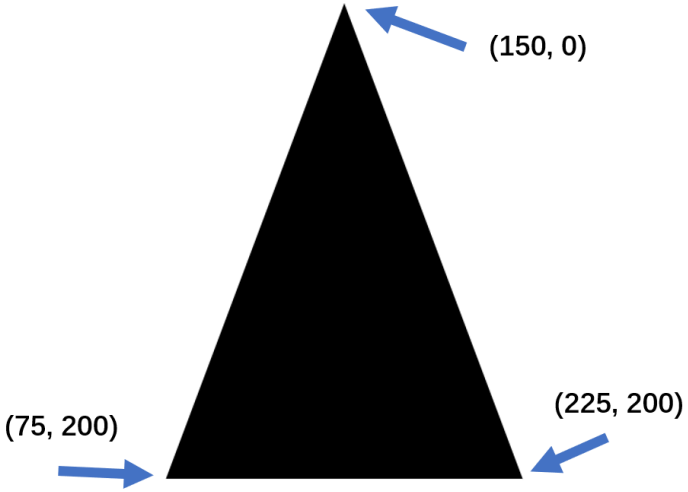

图 1-13　SVG 路径

```
<script>
    var color = d3.scaleOrdinal(['#4daf4a','#377eb8','#ff7f00','#984ea3','#e41a1c']);
    console.log(color(0)) // #4daf4a
    console.log(color(1)) // #377eb8
    console.log(color(2)) // #ff7f00
    console.log(color(3)) // #984ea3
    console.log(color(4)) // #e41a1c
    console.log(color(5)) // #4daf4a, 循环序数
</script>
```

在这段代码中，我们定义了 5 种颜色，并进行了枚举遍历。当遍历到第 6 个值时，超出了颜色总数量会回到起点，即循环序数。

1.11.3　d3.pie()

`d3.pie()`函数根据给定的数据，生成在 SVG 中的饼图对象（楔形）。对每一个楔形计算了初始角度和结束角度，而这便能被用于创建 SVG 中楔形的实际路径。

```
<script>
    var data = [2, 4, 8, 10];
    var pie = d3.pie()
    console.log(pie(data))
</script>
```

打开浏览器中的 Console 调试，可以看到 log 输出：

```
[
    {
        "data": 2,
        "index": 3,
        "value": 2,
        "startAngle": 5.759586531581287,
        "endAngle": 6.283185307179586,
        "padAngle": 0
    },
    {...}, {...}, {...}
]
```

1.11.4 d3.arc()

d3.arc()函数生成弧，具体楔形的路径。弧需要一个内径和外径。如果内径为 0，则结果将是饼图，否则结果将是环形图。我们需要用到这些生成的弧线提供给我们的 SVG 路径元素。

以下这段代码生成了一个简单的饼图，如图 1-14 所示。

图 1-14 简单饼图

<body>标签内的部分：

```
<body>
  <svg width="300" height="200"> </svg>
</body>
```

<script>脚本部分：

```
<script>
  var data = [2, 4, 8, 10];

  // 定义宽 高 半径变量
  var svg = d3.select("svg"),
      width = svg.attr("width"),
      height = svg.attr("height"),
      radius = Math.min(width, height) / 2, // 保证不会超出 SVG 画布边界
      g = svg.append("g").attr("transform", "translate(" + width / 2 + "," + height / 2 + ")");
      // 添加组元素

  // 对颜色使用序数比例尺
  var color = d3.scaleOrdinal(['#4daf4a','#377eb8','#ff7f00','#984ea3','#e41a1c']);

  // 生成饼
  var pie = d3.pie();

  // 生成弧，设置内径和外径
  var arc = d3.arc()
              .innerRadius(0)
              .outerRadius(radius);

  // 生成组
  var arcs = g.selectAll("arc")
              .data(pie(data))
              .enter()
              .append("g")
              .attr("class", "arc")

  // 绘制路径，枚举序数填充颜色
  arcs.append("path")
      .attr("fill", function(d, i) {
          return color(i);
      })
      .attr("d", arc);
</script>
```

1.11.5 饼图案例：浏览器市场份额

接下来，我们将以绘制桌面端浏览器市场份额饼图的实际案例来演示一个完整包含了标签等信息的饼图。

创建一个browser_share.csv的 CSV 文件，内容为：

```
browser,percent
Chrome,68.4
Safari,9.41
Firefox,8.03
Edge,6.36
Opera,2.5
其他,5.3
```

完整代码如下，绘制出的效果如图 1-15 所示。

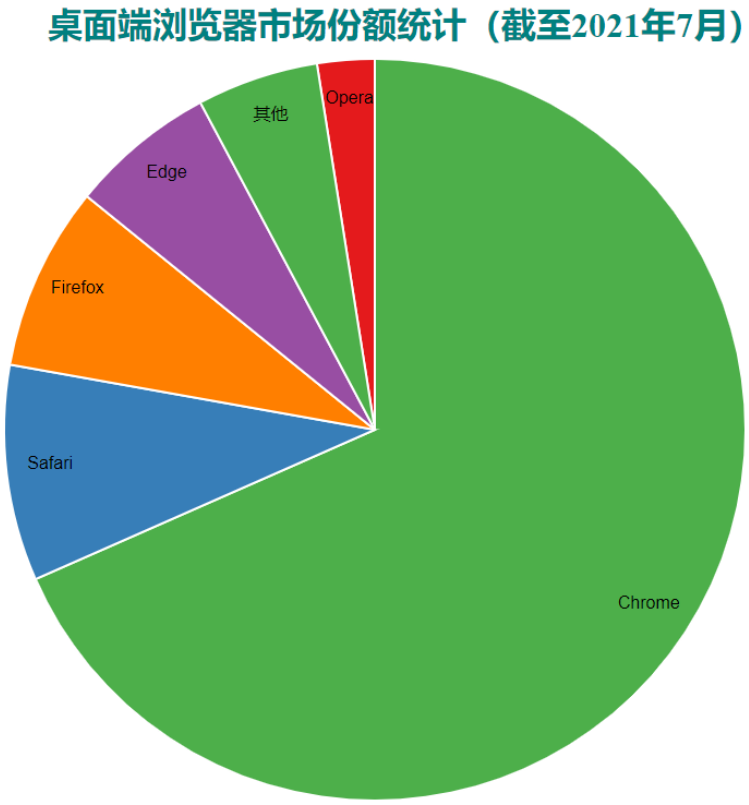

图 1-15　桌面端浏览器市场份额统计（截至 2021 年 7 月）

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>D3.js</title>

    <!-- 引入 D3 -->
    <script src="./js/d3.min.js" charset="utf-8"></script>
  </head>
```

```
11    <body>
12      <svg width="500" height="400"></svg>
13    </body>
14  </html>
15
16  <script>
17    var svg = d3.select("svg"),
18      width = svg.attr("width"),
19      height = svg.attr("height"),
20      radius = Math.min(width, height) / 2;
21
22    var g = svg
23      .append("g")
24      .attr("transform", "translate(" + width / 2 + "," + height / 2 + ")");
25
26    var color = d3.scaleOrdinal([
27      "#4daf4a",
28      "#377eb8",
29      "#ff7f00",
30      "#984ea3",
31      "#e41a1c",
32    ]);
33
34    // 创建匿名函数返回数据中百分比的值
35    var pie = d3.pie().value(function (d) {
36      return d.percent;
37    });
38
39    // 定义内径和外径
40    var arc = d3
41      .arc()
42      .outerRadius(radius - 30)
43      .innerRadius(90); // 0 时为饼图，非 0 为环形图
44
45    // 定义标签所在位置
46    var label = d3
47      .arc()
48      .outerRadius(radius)
49      .innerRadius(radius - 100);
50
51    // 读取 CSV 文件
52    d3.csv("browser_share.csv").then(function (data) {
53      // 为每个 data 创建组元素
54      var arcs = g
55        .selectAll(".arc")
56        .data(pie(data))
57        .enter()
58        .append("g")
```

```
      .attr("class", "arc");

    // 将路径元素添加进组中
    // 使用序数比例尺填充颜色
    arcs
      .append("path")
      .attr("d", arc)
      .attr("fill", function (d) {
        return color(d.data.browser);
      });

    // 在每个楔形中填写标签为浏览器名
    arcs
      .append("text")
      .attr("transform", function (d) {
        return "translate(" + label.centroid(d) + ")";
      })
      .text(function (d) {
        return d.data.browser;
      });
  });

  // 添加图表标题
  svg
    .append("g")
    .attr("transform", "translate(" + (width / 2 - 150) + "," + 20 + ")")
    .append("text")
    .text(" 桌面端浏览器市场份额统计（截至 2021 年 7 月）")
    .attr("class", "title");
</script>

<style>
  /* 设置 CSS 样式 */
  .arc text {
    font: 8px sans-serif;
    text-anchor: middle;
  }

  .arc path {
    stroke: #fff;
  }

  .title {
    fill: teal;
    font-weight: bold;
  }
</style>
```

若在为arc设置内径时，不为 0，如设置成 90 则可得到环形图，如图 1-16 所示。

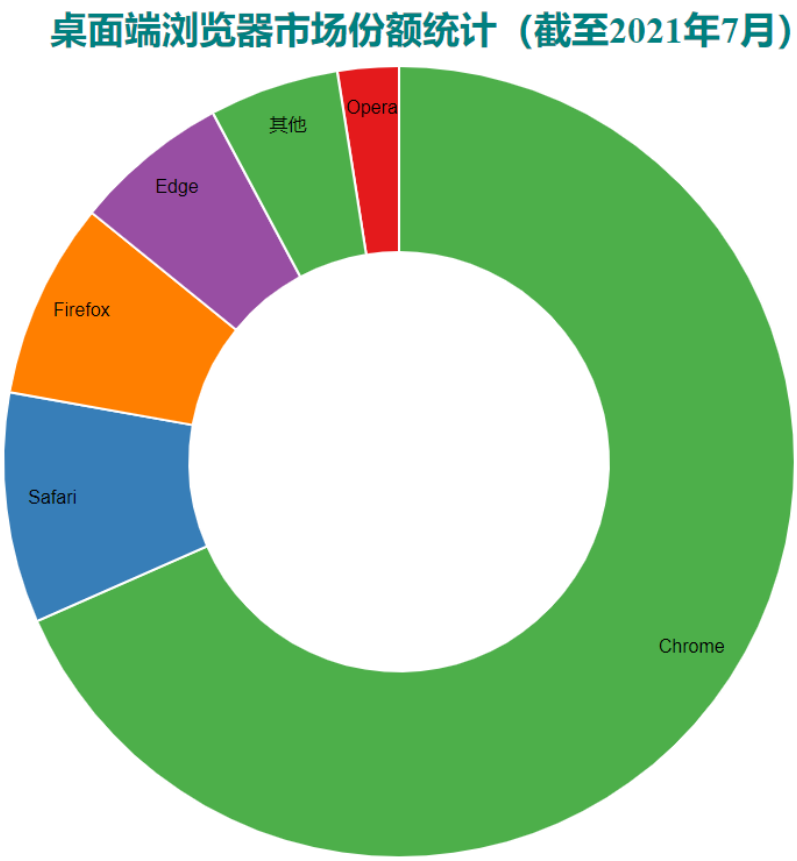

图 1-16　环形图

1.12　动态交互

D3 支持制作动态的图表。有时候，图表的变化需要缓慢地发生，以便于让用户看清楚变化的过程。此外，用户还可能会对图表中的部分元素进行点击，而图表可会对不同的事件作出反应。D3 提供了这些能提升交互式的用户体验的方法和操作。

1.12.1　什么是动态效果

前面几节中制作的图表是直接显示出现，且绘制完成后不再发生变化的，这是静态的图表。

动态的图表，是指图表在某一时间段会发生某种变化，可能是形状、颜色、位置等，用户能够看到变化的过程。

例如，有一个圆，圆心坐标为 (100, 100)。现在我们希望圆的 x 坐标从 100 移到 300，并且移动过程在 2 秒的时间内发生。

这种时候就需要用到动态效果，在 D3 里我们称之为过渡（transition）。

1.12.2 实现动态的方法

D3 提供了 4 个方法用于实现图形的过渡：从状态 A 变为状态 B。

1. transition()

`transition()` 用于启动过渡效果。其前后是图形变化前后的状态（形状、位置、颜色等）。例如这段代码片段：

```
.attr("fill", "red") // 初始颜色为红色
.transition() // 启动过渡
.attr("fill", "green") // 终止颜色为绿色
```

D3 会自动对两种颜色（如上例中的红色和绿色）之间的颜色值（RGB 值）进行插值计算，得到过渡用的颜色值。我们可以观察到颜色变化时的动态渐变效果。

2. duration()

`duration()` 用来指定过渡的持续时间，单位为毫秒。

```
.attr("fill", "red") // 初始颜色为红色
.transition() // 启动过渡
.duration(1000) // 设置过渡的持续时间为 1 秒
.attr("fill", "green") // 终止颜色为绿色
```

3. delay()

`delay()` 指定延迟的时间，表示一定时间后才开始转变，单位同样为毫秒。此函数可以对整体指定延迟，也可以对个别指定延迟。

如对整体指定延时：

```
.transition()
.duration(1000)
.delay(500)
```

图形整体会在延迟 500 毫秒后发生变化，变化的时长为 1000 毫秒。因此，过渡的总时长为 1500 毫秒。

也能对一个个的图形（假设已经绑定了数据）分别指定设置延时：

```
.transition()
.duration(1000)
.delay(funtion(d,i){
    return 200*i;
})
```

假设有 10 个元素，则第 1 个元素延时 0 毫秒，第 2 个元素延时 200 毫秒，第 3 个元素延时 400 毫秒，以此类推。

4. ease()

ease() 指定过渡的缓动函数。常用的有：

(1) d3.easeLinear：普通的线性变化

(2) d3.easeCircle：慢慢地到达变换的最终状态

(3) d3.easeElastic：带有弹跳的到达最终状态

(4) d3.easeBounce：在最终状态处弹跳几次

调用方式形如 .ease(d3.easeLinear)。

1.12.3 什么是交互

交互，指的是用户输入了某种指令，程序接受到指令之后必须做出某种响应。对可视化图表来说，交互能使图表更加生动，能表现更多内容。例如，拖动图表中某些图形、鼠标滑到图形上出现提示框、用触屏放大或缩小图形等等。

用户用于交互的工具一般有三种：鼠标、键盘、触屏。

1.12.4 如何添加交互

对某一元素添加交互操作十分简单，代码如下：

```
var circle = svg.append("circle");

circle.on("click", function(){
```

```
    // 在此处添加交互内容
});
```

这段代码在 SVG 中添加了一个圆，然后通过 `on()` 添加了一个监听器。在 D3 中，每一个选择集都有 `on()` 函数，用于添加事件监听器。

其中，`on()` 的第一个参数是监听的事件，第二个参数是监听到事件后响应的内容，第二个参数是一个函数。

对于鼠标，常用的事件有：

(1) click：鼠标单击某元素时，相当于 mousedown 和 mouseup 组合在一起。

(2) mouseover：光标放在某元素上。

(3) mouseout：光标从某元素上移出来时。

(4) mousemove：鼠标被移动的时候。

(5) mousedown：鼠标按钮被按下。

(6) mouseup：鼠标按钮被松开。

(7) dblclick：鼠标双击。

键盘常用的事件有三个：

(1) keydown：当用户按下任意键时触发，按住不放会重复触发此事件。该事件不会区分字母的大小写，例如“A”和“a”被视为一致。

(2) keypress：当用户按下字符键（大小写字母、数字、加号、等号、回车等）时触发，按住不放会重复触发此事件。该事件区分字母的大小写。

(3) keyup：当用户释放键时触发，不区分字母的大小写。

触屏常用的事件有三个：

(1) touchstart：当触摸点被放在触摸屏上时。

(2) touchmove：当触摸点在触摸屏上移动时。

(3) touchend：当触摸点从触摸屏上拿开时。