
chenjsh36 RP 701 发布于 web前端学习
2016-07-06 发布

D3 源代码解构

> D3是一个数据可视化的javascript库，相对于highchart和echarts专注图表可视化的库，D3更适合做大数据处理的可视化，它只提供基础的可视化功能，灵活而丰富的接口让我们能开发出各式各样的图表。

D3代码版本：“3.5.17”

D3的代码骨架比较简洁，相比jquery来说更适合阅读，你可以很舒服地自上而下的看下去而不用看到一个新的函数发现声明在千里之外，然后在代码中跳来跳去。

内部代码流水线

- 基本的数学计算：最小最大、均值中值方差、偏分值.....
- 各种集合类型： map、set、nest.....
- 集合的操作、方法： text、html、append、insert、remove
- d3的dragging
- 图形操作
- .....

自执行匿名函数

首先是典型的自执行匿名函数，对外提供接口，隐藏实现方式，实现私有变量等等功能。

```
!function() {
   // code here
}()
```

这里用到的是感叹号，其实和使用括号是一样的作用，就是将函数声明变成函数表达式，以便于函数的自执行调用，你可以试试

```
function() {
  console.log('no console')
}()
```

这是因为JS禁止函数声明和函数调用混用，而括号、逻辑运算符（+、-、&&、||）、逗号、new等都可以将函数声明变成函数表达式，然后便可以自执行。有人做过调查关于这些转化的方法哪个更快，可以查看这篇博客，大概new是最慢的，相比使用括号是基本最快，感叹号反而性能一般，所以其实用哪个都没什么区别，当然如果你想省敲一个符号也是可以用感叹号的。

对外暴露私有变量d3

对于d3，采用的是创建私有变量对象，然后对它进行扩展，最后对外暴露

```
var d3 = {
  version: '3.5.17'
};

// code here
//...

if (typeof define === 'function' && defind.amd)
  this.d3 = d3, define(d3);
else if (typeof module == 'object' && module.exports)
  module.exports = d3;
else
  this.d3 = d3;
```

第一种为异步模块加载模式，第二种为同步模块加载或者是ecma6的import机制，第三种则是将d3设置为全局变量，因为匿名自执行函数中，函数的环境就是全局的，所以this == window。

创建公用方法

d3的方法是属于d3对象的属性：

```
d3_xhr( url, mimeType, response, callback) {
  // code
}
d3.json = function(url, callback) {
  return d3_xhr(url, 'application/json', d3_json, callback);
};
function d3_json(request) {
  return JSON.parse(request.responseText);
}
```

不太好的是d3没有在命名上区分哪些是私有函数，哪些是公用函数，不过对于通过创建对象来对外暴露接口的对象来说，应该也不用去区分吧。

提取一些常用的原生函数

```
var d3_arraySlice = [].slice, d3_array = function(list) {
  return d3_arraySlice.call(list);
};
var d3_document = this.document;
```

提取slice方法，使用它来生成数组的副本，slice不会对原生数组做切割，而是会返回数组的复制品，但是要注意是浅复制，对于数组中的对象、数组，是单纯的引用，所以对原数组中的对象或数组的更改还是会影响到复制品。

部分代码实现阅读

一段用来测试d3_array的函数，但什么情况下会重写d3_array函数呢?

【line15】

```
if (d3_document) {
  var test = d3_array(d3_document.documentElement.childNodes);
  console.log(test);
  try {
    d3_array(d3_document.documentElement.childNodes)[0].nodeType;
  } catch (e) {
    console.log('catch error:', e);
    d3_array = function(list) {
      var i = list.length, array = new Array(i);
      while (i--) array[i] = list[i];
      return array;
    };
  }
}
```

由前面我们可以知道d3_array可以用来获取传入数组的副本，通过try来测试document的子节点的第一个子元素，一般就是header这个元素，我们通过查询w3c可以知道nodeType为1，表示html element，感觉应该是测试是否是浏览器环境，如果不是的话，就换成自己写的函数的意思吗？还是为了兼容一些少数的浏览器呢？

设置对象属性的兼容?

【line 30】

```
if (d3_document) {
  try {
    d3_document.createElement("DIV").style.setProperty("opacity", 0, "");
  } catch (error) {
    var d3_element_prototype = this.Element.prototype, d3_element_setAttribute = d3
    d3_element_prototype.setAttribute = function(name, value) {
      d3_element_setAttribute.call(this, name, value + "");
    };
    d3_element_prototype.setAttributeNS = function(space, local, value) {
      d3_element_setAttributeNS.call(this, space, local, value + "");
    };
    d3_style_prototype.setProperty = function(name, value, priority) {
      d3_style_setProperty.call(this, name, value + "", priority);
    };
  }
}
```

数组最小值函数

【line 53】

```
d3.min = function(array, f) {
  var i = -1, n = array.length, a, b;
  if (arguments.length === 1) {
    while (++i < n) if ((b = array[i]) != null && b >= b) {
      a = b;
      break;
    }
    while (++i < n) if ((b = array[i]) != null && a > b) a = b;
  } else {
    while (++i < n) if ((b = f.call(array, array[i], i)) != null && b >= b) {
      a = b;
      break;
    }
    while (++i < n) if ((b = f.call(array, array[i], i)) != null && a > b) a = b;
  }
  return a;
};
```

首先获取第一个可比较的元素，测试了下，发现对于**b >= b**，无论b是数字、字符串、数组甚至是对象都是可以比较的，那么什么情况下 b>=b == false呢，对于NaN来说，无论和哪个数字比较，都是false的，但是对于Infinity却返回真，是个点。所以应该是为了排除NaN这种有问题的数字。

d3的洗牌方法

```
d3.shuffle = function(array, i0, i1) {
  if ((m = arguments.length) < 3) {
    i1 = array.length;
    if (m < 2) i0 = 0;
  }
  var m = i1 - i0, t, i;
  while (m) {
    i = Math.random() * m-- | 0;
    t = array[m + i0], array[m + i0] = array[i + i0], array[i + i0] = t;
    console.log(i, m);
  }
  return array;
};
```

d3使用的洗牌算法,关于Fisher-Yates shuffle的文章可以参考一下，它的演变思路简单而优雅：

正常的思路是

- 每次从原数组中随机选择一个元素，判断是否已经被选取，是的话删除并放入新的数组中，不是的话重新选择。
- 缺点：越到后面重复选择的概率越大，放入新数组的时间越长。

优化

- 为了防止重复，每次随机选择第m张卡牌，m为待洗牌组从原始长度n逐步递减的值
- 缺点：每次都要重新获取剩余数组中的卡牌的紧凑数组，实际的效率为n2

再次优化

- 就地随机洗牌，使用数组的后一部分作为存储新的洗牌后的地方，前一部分为洗牌前的地方，从而将效率提升为n。

d3.map 关于内置对象

【line 291】

```
function d3_class(ctor, properties) {
  for (var key in properties) {
    Object.defineProperty(ctor.prototype, key, {
      value: properties[key],
      enumerable: false
    });
  }
}
d3.map = function(object, f) {
  var map = new d3_Map();
  if (object instanceof d3_Map) {
    object.forEach(function(key, value) {
      map.set(key, value);
    });
  } else if (Array.isArray(object)) {
    var i = -1, n = object.length, o;
    if (arguments.length === 1) while (++i < n) map.set(i, object[i]); else whi
```

```
    }
    return map;
  };
```

关于enumerable

在这里，使用d3_Map来作为对象的构造函数，d3_class来封装类，这里调用了Object.defineProperty来设置属性和值，这里有一个enumerable: false的属性，它将该属性的可枚举性设置为false，使得该属性在一般的遍历中（for...in...)等中无法被获取，但是还是可以通过obj.key直接获取到，如果需要获取对象自身的所有属性，不管enumerable的值，可以使用 Object.getOwnPropertyNames 方法。

为什么要设置这个属性呢？我们可以看到对d3_Map构造对象时，引入了一些原生内置的方法，其中有一个叫做empty的方法用来判断后来设置的属性是否为空，我们来看看这个函数的实现：

```
  function d3_map_empty() {
    for (var key in this._) return false;
    return true;
  }
```

看完之后再结合上面提到的enumerable设置为false的属性在for循环中会被忽略，这样的话就不用再写额外地条件去判断是否为内置属性，很棒的实现方式。

数据绑定函数data

还记得D3独特的将数据和图形领域联系起来的方式吗？ `进入(enter)--更新(update)--退出(exit)` 模式。

【line 832】

```
d3.selectAll('div')
  .data(dataSet)
  .enter()
  .append('div')
  ;
d3.selectAll('div')
  .data(data)
  .style('width', function(d) {
```

```
  ;
d3.selectAll('div')
  .data(newDataSet)
  .exit()
  .remove()
  ;
```

这里涉及到了三个函数，data、enter、exit，每次进行操作前我们需要先调用data对数据进行绑定，然后再调用enter或者exit对图形领域进行操作，那么内部实现原理是怎么样的呢，看完下面这段代码就恍然大悟了：

```
  d3_selectionPrototype.data = function(value, key) {
    var i = -1, n = this.length, group, node;
    if (!arguments.length) {
      value = new Array(n = (group = this[0]).length);
      while (++i < n) {
        if (node = group[i]) {
          value[i] = node.__data__;
        }
      }
      return value;
    }
    function bind(group, groupData) {
      var i, n = group.length, m = groupData.length, n0 = Math.min(n, m), updateN
      if (key) {
        var nodeByKeyValue = new d3_Map(), keyValues = new Array(n), keyValue;
        for (i = -1; ++i < n; ) {
          if (node = group[i]) {
            if (nodeByKeyValue.has(keyValue = key.call(node, node.__data__, i))) 
              exitNodes[i] = node;
            } else {
              nodeByKeyValue.set(keyValue, node);
            }
            keyValues[i] = keyValue;
```

数据绑定函数data最终返回了变量update，这个变量update一开始为一个空集合，它拥有d3的集合操作方法，然后data函数通过调用bind函数对传入的参数进行逐项绑定，获得update集合作为本身，以及enter集合和exit集合，最后在update上绑定了函数enter和exit，使得用户在调用data后，可以再次调用enter和exit去获取另外两个集合。

关于后期debug的足迹

d3也会有bug的时候，这个时候需要对bug进行修复，然后再更新，为了方便下次找到修改的bug，在代码里面对其进行命名，是很好的做法：

【1167】

```
var d3_mouse_bug44083 = this.navigator && /WebKit/.test(this.navigator.userAgent)
? -1 : 0;
```

D3的颜色空间

D3支持五种颜色表示方式，除了我们常常接触了rgb、hsl外，还有lab、hcl、cubehelix，它们之间都可以转化为rgb，内部的实现方式值得参考：

【line 1582】

```
function d3_hsl_rgb(h, s, l) {
  var m1, m2;
  h = isNaN(h) ? 0 : (h %= 360) < 0 ? h + 360 : h;
  s = isNaN(s) ? 0 : s < 0 ? 0 : s > 1 ? 1 : s;
  l = l < 0 ? 0 : l > 1 ? 1 : l;
  m2 = l <= .5 ? l * (1 + s) : l + s - l * s;
  m1 = 2 * l - m2;
  function v(h) {
    if (h > 360) h -= 360; else if (h < 0) h += 360;
    if (h < 60) return m1 + (m2 - m1) * h / 60;
    if (h < 180) return m2;
    if (h < 240) return m1 + (m2 - m1) * (240 - h) / 60;
    return m1;
  }
  function vv(h) {
    return Math.round(v(h) * 255);
  }
  return new d3_rgb(vv(h + 120), vv(h), vv(h - 120));
}
```

关于csv、dsv、tsv存储方式

> 看代码的好处之一是能看到很多平时不会用到的接口，然后会主动去了解是干什么的。

csv格式

在文本数据处理和传输过程中，我们常常遇到把多个字段通过分隔符连接在一起的需求，如采用著名的CSV格式(comma-separated values)。CSV文件的每一行是一条记录（record），每一行的各个字段通过逗号','分隔。

dsv格式

由于逗号和双引号这两个特殊字符的存在，我们不能简单地通过字符串的split操作对CSV文件进行解析，而必须进行CSV语法分析。虽然我们可以通过库的形式进行封装，或者直接采用现成的库，但毕竟各种平台下库的丰富程度差异很大，这些库和split、join这样的简单字符串操作相比也更加复杂。为此，我们在CSV格式的基础上设计了一种DSV (double separated values)格式。DSV格式的主要设计目的就是为了简化CSV语法，生成和解析只需要replace, join, split这3个基本的字符串操作，而不需要进行语法分析。

DSV的语法非常简单，只包括以下两点：

- 通过双竖线'||'作为字段分隔符
- 把字段值中的'|'替换为'_|'进行转义

tsv格式

TSV 是Tab-separated values的缩写，即制表符分隔值。

查询网上关于这三种格式的定义是如上所示，不过d3的实现不太一样，dsv是可以定义为任何一种分隔符，但是分隔符只能为长度为1的字符，csv是以半角符逗号作为分割符，tsv则是以斜杠作为分隔符。

d3.geo

【line 2854】

geo是d3的图形处理实现，应该算是核心代码了，不过到了4.0版本被分割成依赖，并且不再有d3.geo.path了，而是改用d3.geoPath的方式去引用。

总结

版本3的d3九千多行代码，版本4的d4则进行了依赖分割，如果全部依赖引入的话不压缩就要过16000行了，如果想整体去看骨架的话，版本3是比较清晰的，版本4则适合深入研究每一部分的实现，因为依赖都分割得很清晰了，并且相互独立开。

初步了解整个d3的骨架后，接下来可以深入到代码函数实现中去研究其中奥妙。

你可能感兴趣的

- **electron+vue制作桌面应用--自定义标题栏** 王恩智 css html node.js html5 javascript
- **Angular 4.x Reactive Forms** semlinker angular.js angular2
- **Bootstrap 之 Metronic 模板的学习之路 - （7）GULP 前端自动化工具** ingood metronic gulp
- **Linux虚拟网络设备之veth** wuyangchun 网络 linux
- **Java9的新特性** codecraft java
- **从命令式到响应式 （二）** sxlwar javascript rxjs angular5
- **Java多线程进阶（二）—— juc-locks锁框架：接口** Ressmix 多线程 java
- **Java多线程进阶（一）——JUC并发包概述** Ressmix 多线程 java

评论