



专栏 / web前端学习 / 文章详情



chenjsh36 RP 701 发布于 web前端学习

2016-08-01 发布

D3 源代码解析（二）

这是继上一篇D3源码解构文章后的对D3的研究笔记，笔者的能力有限，如有哪里理解错误，欢迎指正。

对集合的操作

关于d3.attr

一个可以处理很多情况的函数，当只传入一个参数时，如果是string，则返回该属性值，如果是对象，则遍历设置对象的键值对属性值，如果参数大于等于2，则是普通的设置样式：

```
var node = d3.select('body')

node.attr('class')
> 返回该属性值

node.attr('class', 'haha')
> 设置该属性值

node.attr({'class': 'haha', 'x': '10'})
> 设置该属性值
```

那么怎么做到一个函数处理多种情况，很明显是根据参数的数量来区别对待：

```
d3_selectionPrototype.attr = function(name, value) {
```



首页



问答



专栏



讲堂



更多

```

var node = this.node();
name = d3.ns.qualify(name);
return name.local ? node.getAttributeNS(name.space, name.local) : node.getAttribute(name.name);
}
for (value in name) this.each(d3_selection_attr(value, name[value]));
return this;
}
return this.each(d3_selection_attr(name, value));
};

```

关于getAttributeNS我们可以不用理会，对于web端，d3在设置和获取属性的时候用的都是getAttribute和setAttribute。

对于d3_selection_attr函数，它返回一个通用函数，该函数会对当前对象设置对应的属性值：

大概的思想：

```

function d3_selection_attr(name, value) {
  return function() {
    this.setAttribute(name, value);
  }
}

```

selection.classed

具体用法可以看文档介绍，大概的意思是如果有键值对或者对象传入，则根据value值来添加或删除name类，否则则检测是否含有该类，如果selection有多个，只检测第一个并返回该值

```

var line = d3.selectAll('line');
line.classed('a b c d', true)
>对所有节点设置class
line.classed({'a': true, 'b': false})
>分别添加和删除类

```

和attr一样，通过对参数长度和类型的区分，执行不同的方法

```

d3_selectionPrototype.classed = function(name, value) {
  if (arguments.length < 2) {
    if (typeof name === "string") {
      ...
    } else if (name !== undefined) {
      ...
    }
  }
}

```



首页



问答



专栏



讲堂



更多

```

        while (++i < n) if (!value.contains(name[i])) return false;
    } else {
        value = node.getAttribute("class");
        while (++i < n) if (!d3_selection_classedRe(name[i]).test(value)) return
    }
    return true;
}
for (value in name) this.each(d3_selection_classed(value, name[value]));
return this;
}
return this.each(d3_selection_classed(name, value));
};

```

这里考虑到传入的字符串可能含有多个类名，`d3_selection_classes`函数用来分割：

```
return (name + '').trim().split(/^\s+/)
```

这里涉及到一个小细节，先用`trim`过滤掉字符串两边的空白字符，然后用正则表达式去分割类名，正则表达式中的`s`匹配任何空白字符，包括空格、制表符、换页符等等。等价于`[fnrtv]`，而且还有一个`^`，它在这里应该是匹配第一个的意思，测试了一下，发现如果不加这个匹配的话，对于空白字符串不会返回长度为0的数组，而是会返回含有一个空字符串长度为一的数组，所以这应该是为了防止出现这种情况而做的匹配，不过原理还是不懂。对于正则的组合，暂时不理解加`^`就能防止该问题的原因。

关于匹配是否存在该类，为了防止匹配的时候发生类名为`'asdf'`，测试的类名为`'a'`，由于包含关系而被匹配成功，所以不能简单的使用`indexOf`的方法，而是要使用正则表达式去做匹配，由于类名要么在最开始，要么在中间两边有空格，要么在末尾，所以使用

```
new RegExp("(?:^|\\s+)" + d3.requote(name) + "(?:\\s+|$)", "g")
```

去做正则匹配

这里用到了`(?:pattern)`的方法，意思是匹配`pattern`但不获取匹配结果，也就是说这是一个非获取匹配，不进行存储供以后使用。这在使用`"或"`字符`(|)`来组合一个模式的各个部分是很有用。例如，`'industry(?:y|ies)`就是一个比`'industry|industries'`更简略的表达式。

d3_selectionPrototype.style



首页



问答



专栏



讲堂



更多

和attr结构类似的函数，特别在于如果传入的值是函数，则会分别对每个元素调用一次函数，并传入元素和元素的位置、优先级等

```
d3_selectionPrototype.style = function(name, value, priority) {  
  var n = arguments.length;  
  if (n < 3) {  
    if (typeof name !== "string") {  
      if (n < 2) value = "";  
      for (priority in name) this.each(d3_selection_style(priority, name[priority]))  
    }  
    return this;  
  }  
  if (n < 2) {  
    var node = this.node();  
    return d3_window(node).getComputedStyle(node, null).getPropertyValue(name);  
  }  
  priority = "";  
}  
return this.each(d3_selection_style(name, value, priority));  
};
```

关于样式的设置，d3用的是style.getProperty(name)和style.setProperty(name, x, priority)
样式的获取，用的是和jquery的实现方法，具体可以看看鑫大大的[文章](#)，

一般我们用的是window.getComputedStyle(elem, '伪类')还有IE自娱自乐的currentStyle，具体的细节就不说了。

两者的不同在于getPropertyValue只能获取设置在style中的属性，而window.getComputedStyle则会得到元素最终显示在页面上的综合样式，就算没有显示声明也可以拿到，这点是最重要的区别。

[selectionPrototype.property](#)、[selectionPrototype.text](#)

property 给元素设置额外的属性，例如：

```
node.property('bar', 'hahahaha')
```

```
node.property('bar') // hahahaha
```

text 设置元素的文本，是通过element.textContent来设置文本的，之前我们设置文本和html都是通过innerText和innerHTML去设置，那么这和textContent有什么区别吗？

实验



首页



问答



专栏



讲堂



更多

不是w3c标准，所以以前firefox并不支持innerText。

两者区别

- 转义上，textContent对传入的文本如果带有n等换行符，不会忽略，而innText会忽略并转义为空格
- .textContent会获取所有子节点的文本，而innerText不会理会隐藏节点的文本。

[selectionProperty.html](#)

这个没什么好讲的，封装了innerHTML的方法

[d3_selectionPrototype.append](#)

比较特别的是实现的代码：

```
d3_selectionPrototype.append = function(name) {  
  name = d3_selection_creator(name);  
  return this.select(function() {  
    return this.appendChild(name.apply(this, arguments));  
  });  
};
```

函数中返回一个函数的执行结果，该执行函数中又返回一个函数的执行结果，层层嵌套却又非常聪明的做法，我们从最里面的一层看，首先对当前的节点添加子元素，然后返回孩子节点元素，最后再通过select方法获取孩子元素。

```
d3_selectionPrototype_creator(name) {  
  function create() {  
    return document.createElement(name);  
  }  
  return typeof name == 'function' ? name : create;  
}
```

这是简易版本的creator，d3还要考虑到在xml中的情况，xml创建子节点调用的是document.createElementNS，d3是通过namespaceURI来判断页面类型的吧，不过在[MDN](#)上查询发现这个属性已经被列为废词，随时可能被废除的，查询了版本4，发现还是沿用了这个属性，这个比较危险吧。



首页



问答



专栏



讲堂



更多

d3_selectionPrototype.insert && d3_selectionPrototype.remove

insertBefore

同append类似，不过是封装了insertBefore的方法，注意需要用元素节点才能调用该方法，正确的调用方法是：

```
existNodeParents.insertBefore (newNode, existNodeToBeInsertBefore)
```

remove

很简单的实现：

```
function d3_selectionRemove() {
  var parent = this.parentNode;
  if (parent) parent.removeChild(this);
}
```

Data

关于d3_selectionPrototype.data函数

这个函数是D3经常使用到也是比较关键的函数，用它来进行数据的绑定、更新，具体解析可以参考上一篇文章[D3源代码解构](#)

这里涉及到一个特殊的属性**data**,如果不传入参数，**data**会返回所有算中集合元素的属性值（property），但是为什么是通过node._data_拿到的，通过搜索，终于找到了绑定该值得函数（一开始还以为是DOM的隐藏变量--）

```
d3_selectionPrototype.datum = function(value) {
  return arguments.length ? this.property("__data__", value) : this.property("__d",
};
```

如果传入参数，它会创建三个特殊的私有变量，分别是

- enter = d3_selection_enter([])
- update = d3_selection([])
- exit = d3_selection([])



首页



问答



专栏



讲堂



更多

上面提到的selectionPrototype所有的方法，而enter比较特殊，它单独使用一套原型方法，实现方法如下：

```
d3_selection_enterPrototype.call = d3_selectionPrototype.call;
d3_selection_enterPrototype.size = d3_selectionPrototype.size;
d3_selection_enterPrototype.select = function(selector) {
  var subgroups = [], subgroup, subnode, upgroup, group, node;
  for (var j = -1, m = this.length; ++j < m; ) {
    upgroup = (group = this[j]).update;
    subgroups.push(subgroup = []);
    subgroup.parentNode = group.parentNode;
    for (var i = -1, n = group.length; ++i < n; ) {
      if (node = group[i]) {
        subgroup.push(upgroup[i] = subnode = selector.call(group.parentNode, node));
        subnode.__data__ = node.__data__;
      } else {
        subgroup.push(null);
      }
    }
  }
  return d3_selection(subgroups);
};
d3_selection_enterPrototype.insert = function(name, before) {
  if (arguments.length < 2) before = d3_selection_enterInsertBefore(this);
  return d3_selectionPrototype.insert.call(this, name, before);
};
```

然后调用bind函数对传入的data和key（可选）进行数据绑定，我们知道d3会根据传入的数据和已有的元素进行一一对应，一开始以为是基于什么算法去对应，看代码实现就发现如果我们不传入key参数，其实就是简单的索引对应：

```
function bind(group, groupData) {
  var i, n = group.length, m = groupData.length, n0 = Math.min(n, m), updateNodes = [];
  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);
        }
      }
    }
  }
}
```



首页



问答



专栏



讲堂



更多

```

    }
    for (i = -1; ++i < m; ) {
      if (!(node = nodeByKeyValue.get(keyValue = key.call(groupData, nodeData
        enterNodes[i] = d3_selection_dataNode(nodeData);
      } else if (node !== true) {
        updateNodes[i] = node;
        node.__data__ = nodeData;
      }
      nodeByKeyValue.set(keyValue, true);

```

而当我们传入了key后，这个时候就不一样了，D3会根据我们传入的这个函数去将元素和数据做绑定和更新、退出，这个key函数会在三次循环中分别被调用，一次是检查是否有已经绑定了数据的元素，并初始化一个映射集合，第二次进行数据绑定元素，确定update和enter集合，第三次确定exit集合。

建议先看看官方文档，了解具体的用法在看代码会清晰很多。通俗的说，假设我们传入的数据有主键即唯一区分每个数据的属性，那么，我们便可以告诉data说用这个属性来区分，也就是：

```

selection.data(mydata, function(d, i) {
  return d.主键名称
})

```

关于d3_map集合可以参考[d3_map解析](#)

Animation & Interaction （动画和交互）

[d3_selectionPrototype.datum]()

这是上面讲到的一个函数datum，可惜在data中其实没有用到，我遍历了整个代码只有一处地方调用了这个函数，它和data类似用来获取或者设置元素的值，它是基于property上进行一层封装，但是和data不同的是它没有所谓的enter、exit集合返回，那么它有什么用呢？我们可以看看这篇[文章](#)

d3_selectionPrototype.filter

可以传入函数或者选择器字符串进行集合的过滤

D3的事件监听机制



首页



问答



专栏



讲堂



更多

看d3关于事件监听的实现，看到了关于JS事件的一个属性**relatedTarget**，关于JS的event对象之前接触的不多，突然看到关于这个属性，上网查找资料，才发现了这么冷门的属性：

relatedTarget 事件属性返回与事件的目标节点相关的节点。

对于 mouseover 事件来说，该属性是鼠标指针移到目标节点上时所离开的那个节点。

对于 mouseout 事件来说，该属性是离开目标时，鼠标指针进入的节点。

对于其他类型的事件来说，这个属性没有用。

怎么样，够冷门吧，只对两种事件生效

还有一个方法叫做**compareDocumentPosition**，比较两个节点，并返回描述它们在文档中位置的整数

1：没有关系，两个节点不属于同一个文档。

2：第一节点（P1）位于第二个节点后（P2）。

4：第一节点（P1）定位在第二节点（P2）前。

8：第一节点（P1）位于第二节点内（P2）。

16：第二节点（P2）位于第一节点内（P1）。

32：没有关系，或是两个节点是同一元素的两个属性。

注释：返回值可以是值的组合。例如，返回 20 意味着在 p2 在 p1 内部（16），并且 p1 在 p2 之前（4）。

知道了这两个属性，d3的一个函数就看懂了：

```
function d3_selection_onFilter(listener, argumentz) {
  var l = d3_selection_onListener(listener, argumentz);
  return function(e) {
    var target = this, related = e.relatedTarget;
    if (!related || related !== target && !(related.compareDocumentPosition(target).call(target, e));
  }
};
```

获取事件对应的对象和相关的对象，如果不存在相关的对象或者相关的对象不等于当前对象且相关对象不在当前对象之内，则执行监听函数。

```
function d3_selection_onListener(listener, argumentz) {
  return function(e) {
    var o = d3.event;
```



首页



问答



专栏



讲堂



更多

```

    try {
      listener.apply(this, argumentz);
    } finally {
      d3.event = o;
    }
  };
}

```

这个函数返回一个函数，返回的函数绑定了当前对象并执行。

```

var d3_selection_onFilters = d3.map({
  mouseenter: "mouseover",
 mouseleave: "mouseout"
});
if (d3_document) {
  d3_selection_onFilters.forEach(function(k) {
    if ("on" + k in d3_document) d3_selection_onFilters.remove(k);
  });
}

```

D3还做了一个事件映射，将mouseenter映射为mouseover，mouseleave映射为mouseout，然后判断环境中是否有这两个事件，如果有的话就取消这个映射。

以上三段代码都是为了处理执行环境中没有mouseenter和mousemove情况下如何利用mouseover和mouseleave去实现相同效果的问题。然后通过下面这个函数来判断：

```

this.removeEventListener(type, l, l.$);
delete this[name];
}
}

function onAdd() {
  var l = wrap(listener, d3_array(arguments));
  onRemove.call(this);
  this.addEventListener(type, this[name] = l, l.$ = capture);
  l._ = listener;

}

function removeAll() {
  var re = new RegExp("^__on(\\.[^.]*)" + d3.requote(type) + "$"), match;
  for (var name in this) {
    if (match = name.match(re)) {
      var l = this[name];

```



首页



问答



专栏



讲堂



更多

```

        }
    }
    console.log('d3_selection_on:', i, listener, i ? listener ? onAdd : onRemove
    return i ? listener ? onAdd : onRemove : listener ? d3_noop : removeAll;
}

```

现在再来看这个函数就可以看懂了，首先它判断传入的事件类型是否含有'!'，因为D3在实现事件绑定时，会清除同种事件类型之前绑定的监听函数，所以对于同一类型的事件，如果要绑定多个监听函数，那么就需要使用`click.foo*click.bar*`这种方式去进行区分，防止旧的事件被覆盖掉，查看`onAdd`函数就可以知道每次添加事件监听的时候，就会调用`onRemove`去清除该事件监听。

关于`capture`，默认是`false`，表示在冒泡阶段响应事件，如果设置为`true`，则是在捕获阶段响应事件，可以参考[这篇文章](#)，这是历史遗留原因，好像当初的浏览器响应事件的设置不是冒泡阶段，而是捕获阶段，后来为了兼容而给了这个参数。

好了，懂得了D3事件绑定的原理，那么实现这个函数就很容易，一样的根据参数的数量和类型做不同的处理就好了：

```

d3_selectionPrototype.on = function(type, listener, capture) {
    var n = arguments.length;
    if (n < 3) {
        if (typeof type !== "string") {
            if (n < 2) listener = false;
            for (capture in type) this.each(d3_selection_on(capture, type[capture]), lis
                return this;
        }
        if (n < 2) return (n = this.node().__on + type]) && n._;
        capture = false;
    }
    return this.each(d3_selection_on(type, listener, capture));
};

```

[d3.mouse]()

[MDN上关于svg的一些属性](#)

[一篇关于svg的讲解](#)

[关于svg坐标转换为屏幕坐标.aspx](#)

[手工使用五连体楼梯的实现](#)



首页



问答



专栏



讲堂



更多

- 【ownerSVGElement】，用来获取这个元素最近的svg祖先，没有的话就返回元素本身。
- 【svg.createSVGPoint】这个函数不在MDN中，看下MF的[介绍](#).aspx)，大概意思是初始化一个不在document文档内的坐标点
- 【getScreenCTM】

当我们获取网页上鼠标的坐标点的时候，可以很简单地调用e.clientX,或者e.pageY，但是svg有自己的一套坐标系，它可以自身旋转、平移，所以我们想知道按钮点击的位置相对于svg元素的位置时，需要考虑这些因素，从而使得获取鼠标在svg的位置时变得没那么容易，再加上各种浏览器的坑.....

这个时候就是线性代数就用上了（感谢线代老师！），忘的差不多的可以参考上面的几篇文章，svg自身已经提供了对应的矩阵运算，节省了我们的一些实现的代码。

再看看D3的代码，就知道原作者也是被坑过的：

```

var window = d3_window(container);
if (window.scrollX || window.scrollY) {
  svg = d3.select("body").append("svg").style({
    position: "absolute",
    top: 0,
    left: 0,
    margin: 0,
    padding: 0,
    border: "none"
  }, "important");
  var ctm = svg[0][0].getScreenCTM();
  d3_mouse_bug44083 = !(ctm.f || ctm.e);
  svg.remove();
}
if (d3_mouse_bug44083) point.x = e.pageX, point.y = e.pageY; else point.x =
point.y = e.clientY;
point = point.matrixTransform(container.getScreenCTM().inverse());
return [ point.x, point.y ];
}
var rect = container.getBoundingClientRect();
return [ e.clientX - rect.left - container.clientLeft, e.clientY - rect.top -
}
```

clientX是获取相对于浏览器屏幕的坐标，减去元素相对于屏幕的左边距，为了兼容IE等坑爹的默认开始位置为(2,2)，减去container的clientLeft，最终得到svg的鼠标位置，但真的是为了获取相对的位置么，需要再看看。



首页



问答



专栏



讲堂



更多

Behavior

[d3的touch、drag、touches]()

看不太懂这几个的实现，和自己没有怎么使用到这几个函数有关吧

[d3.zoom]()

zoom函数的实现，大概知道它通过绑定mouseWheel事件去记录了放缩的值、中心、放缩位置等。也是涉及到event的绑定，表示hin晕。

D3的颜色空间

具体可以参考前一篇[文章](#)

d3.xhr

D3对于ajax的实现，没有兼容IE6及6以下的`xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");`只考虑了`window.XMLHttpRequest`,因为老版本的IE压根就无法正常使用各种图形和动画。

D3的timer的实现有点厉害

当我们要用D3实现一个永久循环的动画的时候，就可以使用timer函数，向这个函数传入一个函数，timer函数会在每个动画帧中调用传入的函数直至该函数返回'true'，所以只要我们始终不返回true就好了。如果是这么简单当然就好实现了，但是如果多个timer怎么去控制呢？这个问题导致了实现的方法复杂了很多，直接上代码：

```
var d3_timer_queueHead, d3_timer_queueTail, d3_timer_interval, d3_timer_timeout,
    setTimeout(callback, 17);
};

d3.timer = function() {
    d3_timer.apply(this, arguments);
};

function d3_timer(callback, delay, then) {
    var n = arguments.length:
```



首页



问答



专栏



讲堂



更多

```

var time = then + delay, timer = {
  c: callback,
  t: time,
  n: null
};
if (d3_timer_queueTail) d3_timer_queueTail.n = timer; else d3_timer_queueHead
d3_timer_queueTail = timer;
if (!d3_timer_interval) {
  d3_timer_timeout = clearTimeout(d3_timer_timeout);
  d3_timer_interval = 1;
  d3_timer_frame(d3_timer_step);
}
return timer;

```

D3使用队列的方法实现，每次有新的timer进来，判断队列是否为空，如果为空，就将Head和队尾指向它，否则，将队尾和队尾的下一个指向它

```

if (d3_timer_queueTail) d3_timer_queueTail.n = timer; else d3_timer_queueHead =
d3_timer_queueTail = timer;

```

感谢C和C++，告诉我指针实现链表的概念！

然后开始执行回调函数。

```

if (!d3_timer_interval) {
  d3_timer_timeout = clearTimeout(d3_timer_timeout);
  d3_timer_interval = 1;
  d3_timer_frame(d3_timer_step);
}

```

timer_frame的实现是兼容了老版本的浏览器没有 `requestAnimationFrame` 而退而使用`setTimeout`去实现，如果不太清楚这个api的同学可以看看鑫旭的这篇文章或者上MDN查。

然后每个帧都会调用d3_timer_step这个函数，它调用了d3_timer_mark和d3_timer_sweep函数，循环遍历了一遍时间队列，然后获取最近的待执行的时间点，得到了delay时间差，当时间差大于24并且不为Infinity的时候，便重新设置时间器，让其在delay ms后执行，减少性能的消耗，若为Infinity，表示没有时间事件等待调用，停止了递归，否则，delay小于24ms，递归调用d3_timer_frame。

那么为什么为24ms呢？我们知道浏览器的最佳动画帧是60fps，算起来每一帧的间隔为 $1000/60 = 16.7\text{ms}$ ，



首页



问答



专栏



讲堂



更多

那么这个和24有什么关系呢？为什么要设定为24呢？我也不清楚...在github上面提交了[issues](#)，不知道会不会有人解答，好紧张。

关于timer的一些扩展：

[timer实现永久动画](#)

[点击预览](#)

[作者的实现](#)

早上提交的issue下午原作者就给了[回复](#)，不过作者的解释就尴尬了，大概的意思就是由于`setTimeout`的不稳定和不准确，存在一定的延迟，所以在设定这个值的时候也是拍脑袋设置的，值刚好在16.7到33.4之间，并回复说左右偏移都不会有什么影响就对了。

[d3关于number 的方法： `formatPrefix` 和 `round`]()

提供了将number转化为特定格式的字符串方法，基于正则表达做匹配，然后对应地做转化。这部分的实现比较琐碎，就没去仔细研究了，有兴趣的可以看看。

[d3.time]()

同样的，将d3.time初始化为一个空对象，并且将window.Date对象设置为私有变量：`d3_date = Date` 万物皆为我所用！

首先我们要了解Date的UTC函数，`UTC()` 方法可根据世界时返回 1970 年 1 月 1 日 到指定日期的毫秒数。然后来看这个函数：

```
function d3_date_utc() {
  this._ = new Date(arguments.length > 1 ? Date.UTC.apply(this, arguments) : arguments[0])
}
```

这个函数是一个构造函数，当我们`new d3_date_utc(xxx)`的时候，它会创建一个日期对象，并根据我们传入的参数数量去创建，如果我们传入的参数多余1个，那么很显然我们传入的是年月日这些参数，那么便调用`Date.UTC.apply`去返回时间戳，如果参数只有一个的话，那就直接返回咯，那么参数为0会怎么样？

我们可以实践下，相当于`new Date(undefined)`，返回的结果是 Invalid Date的Date对象。

为什么能肯定是Date对象呢，我们使用`instanceof Date`去测试，发现结果为true，那么当我们打印出来为什么为Invalid Date呢，很明显，它调用了`toString`方法或者`valueOf()`方法，经过测试是`toString`方法，`valueOf`方法返回的是NaN。

好了，扩展就到这里，继续看下去，



首页



问答



专栏



讲堂



更多

```
d3_date_utc.prototype = {
  getDate: function() {
    return this._.getUTCDate();
  },
  getDate: function() {
    return this._.getUTCDay();
  },
  ...
}
```

可以看到，D3封装了原始Date对象的一些方法，例如getDay和GetHours等，它不适用原生的Date.getDay等，而是使用getUTCDay去拿，那么这两者有什么不一样吗？

当你new一个Date对象的时候，返回的是本地的时间，注意，是你所在时区的时间哦，所以假设你现在的时
间是

Tue Jul 19 2016 14:44:19 GMT+0800 (中国标准时间)

那么当你使用getHours的时候，返回的时间是14，但是，当你使用getUTCHours的时候，它返回的是全球
的时间，什么叫全球？请参考[MDN](#)上关于这个函数的解释：

The **getUTCHours()

** method returns the hours in the specified date according to universal time.

它的意思是会参考0时区的时间来给你时间，由于我们所处的地方（中国）是在8时区，所以在0时区比我们
这里早8个小时，所以他们那边现在还是早晨8点正在洗脸刷牙吃早餐。

所以这个对象封装了Date对象的UTC方法，变成一个全球流的时间器，然后它的方法不再需要添加UTC这个名字就可以调用了，其实我们也可以做到。

接下来是几个函数的声明和定义：

```
function d3_time_interval(local, step, number) {
  function round(date) {}
  function ceil(date) {}
  function offset(date, k) {}
  function range(t0, t1, dt) {}
  function range_utc(t0, t1, dt) {}
  local.floor = local;
  local.round = round;
```



首页



问答



专栏



讲堂



更多

```
local.range = range;
var utc = local.utc = d3_time_interval_utc(local);
utc.floor = utc;
utc.round = d3_time_interval_utc(round);
utc.ceil = d3_time_interval_utc(ceil);
utc.offset = d3_time_interval_utc(offset);
utc.range = range_utc;
return local;
}
```

暂时不看这个函数里面的函数是做什么的，首先d3_time_interval这个函数接受三个参数，然后对传入的local参数，我们给了它五个方法，分别是我们在本地定义的五个方法，然后又给local定义个utc的属性，这个属性还额外拥有五个方法，最后返回了这个local对象，可以看出来这个函数是一个包装器，对传入的local对象进行包装，让它拥有固定的方法，接下来看下一个函数：

```
function d3_time_interval_utc(method) {
  return function(date, k) {
    try {
      d3_date = d3_date_utc;
      var utc = new d3_date_utc();
      utc._ = date;
      return method(utc, k)._;
    } finally {
      d3_date = Date;
    }
  };
}
```

一个返回函数的函数，这是在类库里面经常见到的用法，我经常被它给迷醉，能用的好能创造出很奇妙的作用。看代码我们仍然不知道具体是做什么的，不急，继续往下看

```
d3_time.year = d3_time_interval(function(date) {
  date = d3_time.day(date);
  date.setMonth(0, 1);
  return date;
}, function(date, offset) {
  date.setFullYear(date.getFullYear() + offset);
}, function(date) {
  return date.getFullYear();
});
```



首页



问答



专栏



讲堂



更多

我们知道d3_time就是d3.time对象，是一个空对象目前，这里开始给它添加属性了，并且调用了上面的d3_time_interval函数，向它传入了三个函数，d3没有注释就是惨，完全不知道传入的参数类型，这点以后写代码需要注意

```
function round(date) {
  // d0是初始化的date的本地日期，时间为默认的凌晨或者时区时间，d1是本地时间加了一个单位，而
  var d0 = local(date), d1 = offset(d0, 1);
  return date - d0 < d1 - date ? d0 : d1;
}
// 对传入的时间进行加一个单位
function ceil(date) {
  step(date = local(new Date(date - 1)), 1);
  return date;
}
// 对传入的时间做加减法
function offset(date, k) {
  step(date = new Date(+date), k);
  return date;
}
```

后面的一部分主要有针对传入的参数对时间进行不同的格式化等等

d3.geo

d3的图形化算法的实现，这一部分涉及到了几何、数据结构等方面的知识，大概三千多行的代码量，基本是各种符号和公式，没有注释的话看起来和天书没有区别，需要单独花时间来慢慢看了。

[d3.interpolate]()

接下来的是d3关于不同类型的插值的实现

首先是颜色：d3.interpolateRgb

```
d3.interpolateRgb = d3_interpolateRgb;
function d3_interpolateRgb(a, b) {
  a = d3.rgb(a);
  b = d3.rgb(b);
  var ar = a.r, ag = a.g, ab = a.b, br = b.r - ar, bg = b.g - ag, bb = b.b - ab;
  return function(t) {
```



首页



问答



专栏



讲堂



更多

```
};  
}
```

颜色的插值实现其实没有什么技巧，就是分别取rgb三个值做插值，然后再将三种颜色合并为一种颜色，以后可以自己实现一个颜色插值器了。

除了颜色，还有对对象的插值实现：

```
d3.interpolateObject = d3_interpolateObject;  
function d3_interpolateObject(a, b) {  
  var i = {}, c = {}, k;  
  for (k in a) {  
    if (k in b) {  
      i[k] = d3_interpolate(a[k], b[k]);  
    } else {  
      c[k] = a[k];  
    }  
  }  
  for (k in b) {  
    if (!(k in a)) {  
      c[k] = b[k];  
    }  
  }  
  return function(t) {  
    for (k in i) c[k] = i[k](t);  
    return c;  
  };  
}
```

遍历两个对象，用i存储两个对象都有的属性的值的插值，用c来存储两个对象各自独有的属性值，最后合并到c中，完事。

D3还实现了字符串的插值，不过不是对字符的插值，而是检测字符串的数字做插值，对传入的参数a和b，每次检测到a中的数字，便到b中找对应的数字然后做插值，如果a的数字找不到对应，就会被抛弃，a中的其他字符串都会被抛弃，只保留b中的字符串。

```
/[-+]?(:\d+\.\?\d*|\.\?\d+)(?:[eE][-+]?\d+)?/g
```



除了d3本身提供的这些插值器外，我们也可以[自定义插值器](#)

```
d3.interpolate = d3_interpolate;
function d3_interpolate(a, b) {
  var i = d3.interpolators.length, f;
  while (--i >= 0 && !(f = d3.interpolators[i])(a, b)) ;
  return f;
}
d3.interpolators = [ function(a, b) {
  var t = typeof b;
  return (t === "string" ? d3_rgb_names.has(b.toLowerCase()) || /^(#|rgb\(|hsl\()/.test(b) : true);
} ];
```

d3会自己循环遍历插值器队列，直到有插值器返回了对应的对象。

[d3.ease]()

d3.ease实现了多种动画函数，开发者可以根据自身的需要调用不同的动画效果，具体的示例可以参考这篇[文章](#)

d3.transform

d3只涉及到平面上的转化，transform包含四个属性：rotate、translate、scale、skew（斜交），transform也是一个变化，所以也可以作为插值器，[关于csstransform的文档](#)



赞 | 1

收藏 | 7

你可能感兴趣的

- [读Zepto源码之属性操作](#) 对角另一面 javascript zepto jquery 源码分析
- [JS高程读书笔记--第五章引用类型](#) 多喝热水早点睡 javascript
- [javascript基础之判断变量类型](#) funnycoderstar javascript
- [JS-RegEx函数](#) 普拉斯 javascript reaexp 正则表达式



首页



问答



专栏



讲堂



更多

- [JavaScript 数据类型转换](#) percy507 javascript
- [《前端竹节》（2）【正则表达式】](#) sept08 正则表达式 javascript
- [jQuery源码解析之Data](#) 荡漾 javascript jquery

评论

[默认排序](#) [时间排序](#)



文明社会，理性评论

[发布评论](#)

Copyright © 2011-2019 SegmentFault. 当前呈现版本 19.02.27

浙ICP备 15005796号-2 浙公网安备 33010602002000号 杭州堆栈科技有限公司版权所有

CDN 存储服务由 又拍云 赞助提供

[移动版](#) [桌面版](#)



首页



问答



专栏



讲堂



更多