- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3数据可视化205076374，大数据可视化436442115

当你使用D3处理数据可视化时，通常会倾向于做大量的**数组操作(array manipulation)**。那是因为数组是D3的标准的数据呈现形式。数组处理的一些常见形式包括：取数组的一个连续片段（子集），使用判定函数过滤数组，使用变换函数映射数组为一组平行的值。在看到D3框架提供的一系列方法处理数组时，你应当很熟悉强大的JavaScript内置的数组的方法.

JavaScript包含修改数组的**赋值方法(mutator methods)**:

- array.pop -删除数组最后一位元素。
- array.push - 往数组的末尾新增一个或多个元素。
- array.reverse - 把数组元素的逆转顺序。
- array.shift - 删除数组第一位元素。
- array.sort - 给数组排序。
- array.splice - 给数组添加或者删除元素。
- array.unshift - 往数组的第一位新增一个或多个元素。

还有一些数组的**存取方法(accessor methods)**，返回数组的一些描述:

- array.concat - 合并数组或合并数组的值。
- array.join - 合并数组所有元素拼接成字符串。
- array.slice - 提取数组的一个选择。
- array.indexOf - 定位到数组第一个值。
- array.lastIndexOf - 定位到数组内最后一个值。

最后，对数组中的元素使用用函数的**迭代方法(iteration methods)**:

- array.filter - 由满足特定条件的元素创建一个新的数组。
- array.forEach - 为数组中每一个元素调用一个函数。
- array.every - See if every element in the array satisfies a predicate.
- array.map - Create a new array with the result of calling a function on every element in the array.
- array.some - See if at least one element in the array satisfies a predicate.
- array.reduce - Apply a function to reduce the array to a single value (from left-to-right).
- array.reduceRight - Apply a function to reduce the array to a single value (from right-to-left).

排序 （Ordering）

d3.**ascending**(*a*, *b*)

如果 `a < b` 返回-1， `a > b` 返回1， `a = b` 返回0。 这是固有的比较器方法，也可用于关联内置数组排序的方法来给元素升序排序:

```
function ascending(a, b) {
  return a < b ? -1 : a > b ? 1 : a >= b ? 0 : NaN;
}
```

注意，如果没有给数组的内置排序方法没有指定比较器函数，那默认的排序是字典排序(按字母顺序排序)，而非自然排列！所以当以数组的数字来排序时会导致bug。

d3.**descending**(*a*, *b*)

如果 `a > b` 返回-1， `a < b` 返回1， `a = b` 返回0。 这是固有的比较器方法，也可用于关联内置数组排序的方法来给元素降序排序：

```
function descending(a, b) {
  return b < a ? -1 : b > a ? 1 : b >= a ? 0 : NaN;
}
```

注意，如果没有给数组的内置排序方法没有指定比较器函数，那默认的排序是字典排序(按字母顺序排序)，而非自然排列！所以当以数组的数字来排序时会导致bug。

d3.**min**(*array*[, *accessor*])

返回给定数组（array）中自然排序最小的值。如果数组为空，返回undefined。如果指定了accessor 参数，等同与在计算最小值之前调用了array.map(accessor)方法。不同于内置的Math.min，这个方法会忽略未定义的值；这对比例尺([[d3.scale|比例尺]])定义域计算很有用处，当只考虑数据的定义区域。另外，元素的比较用的是自然排序而不是数字排序。例如，["20","3"]的最小值是20，然而[20,3]的最小值是3。

d3.**max**(*array*[, *accessor*])

返回给定数组（array）中自然排序最大的值。如果数组为空，返回undefined。如果指定了accessor 参数，等同与在计算最大值之前调用了array.map(accessor)方法。而并非内置的Math.max，这个方法会忽略未定义的值；这对当只需要定义数据的区域的比例尺定义域计算很有用处。另外，元素的比较用的是自然排序而不是数字排序。例如，["20","3"]的最大值是3，然而[20,3]的最大值是20。

d3.**extent**(*array*[, *accessor*])

返回给定数组（array）自然排序的最小值和最大值，等同于同时调用d3.min和d3.max.

d3.**sum**(*array*[, *accessor*])

返回给定数组（array）的和。如果数组为空，返回 0。可选参数accessor函数 被指定，等同于在计算和之前调用 *array.map(accessor)* 。此方法忽略无效值(如 `NaN` 和 `undefined`)；当只考虑明确定义的值时，这个方法可用于计算数据的和。

d3.**mean**(*array*[, *accessor*])

返回给定数组（array）的平均数。如果数组为空，返回 `undefined` 。可选参数accessor函数 被指定，等同在计算平均数之前调用*array.map(accessor)* 。此方法忽略无效值(如 `NaN` 和 `undefined`)，当只考虑明确定义的值时这个方法计算数据和是很有用的。

d3.**median**(*array*[, *accessor*])

返回给定数组（array）以R-7算法得出的中位数。如果数组为空，返回 undefined。可选参数accessor 被指定，等同在计算中位数之前调用*array.map(accessor)* 。此方法忽略无效值(如 `NaN` 和 `undefined`)，当只考虑明确定义的值时这个方法计算数据和是很有用的。

d3.**quantile**(*numbers*, *p*)

返回给定数组*numbers*的*p*分位数，其中p 是一个0到1范围的数。例如，中位数可以由p = 0.5计算，第一个四分位数是p = 0.25，第三个四分位数是p = 0.75。这个特别实现了R-7算法，这是R编程语言和Excel默认的方式。这个方法需要数组numbers包含数字且数字升序顺序排列，例如使用 d3.ascending排序。

```
var a = [0, 1, 3];
d3.quantile(a, 0); // return 0
d3.quantile(a, 0.5); // return 1
```

```
d3.quantile(a, 1); // return 3
d3.quantile(a, 0.25); // return 0.5
d3.quantile(a, 0.75); // return 2
d3.quantile(a, 0.1); // return 0.19999999999999996
```

d3.**variance**(*array*[, *accessor*])

返回给定数组（array）的无偏总体方差（[unbiased estimator of the population variance](http://mathworld.wolfram.com/SampleVariance.html "http://mathworld.wolfram.com/SampleVariance.html")）。如果数组的长度小于2，返回`undefined`。可选参数accessor 被指定，等同在计算中位数之前调用*array.map(accessor)* 。此方法忽略无效值(如 `NaN` 和 `undefined`)。

d3.**deviation**(*array*[, *accessor*])

返回给定数组（array）的标准差，即方差（bias-corrected variance）的平方根。如果数组的长度小于2，返回 `undefined` 。可选参数accessor 被指定，等同在计算中位数之前调用*array.map(accessor)* 。此方法忽略无效值(如 `NaN` 和 `undefined`)。

d3.**bisectLeft**(*array*, *x*[, *lo*[, *hi*]])

定位数组 *array* 中的 *x* 的插入点，以保持已有序列。参数 *lo* 和 *hi* 用来指定数组的子集；默认情况下整个数组都被使用。如果 *x* 在 *array* 中已存在，插入点在所有元素之前（左侧）。返回值适合用作拼接（splice）*已经排序的*数组*array* 的第一个参数。返回的插入点*i*把*array* 分为两个区：数组中所有*array*.slice(lo, i)中 `v < x` 的v在左边，数组中所有 *array*.slice(i, hi)中 `v >= x` 的v在右边。

d3.**bisect**(*array*, *x*[, *lo*[, *hi*]])
d3.**bisectRight**(*array*, *x*[, *lo*[, *hi*]])

和bisectLeft类似，但返回插入点来自于数组*array*中任意实体*x*之后（右侧）。返回的插入点*i*把*array* 分为两个区：数组中所有*array*.slice(lo, i)中 `v <= x` 的v在左边，数组中所有 *array*.slice(i, hi)中 `v > x` 的v在右边。

d3.**bisector**(*accessor*)
d3.**bisector**(*comparator*)

使用指定参数*accessor*或者*comparator* 函数返回一个二等分线。返回的对象有 `left` 和 `right` 属性，分别类似于bisectLeft和bisectRight方法。这个方法能用于二等分对象数组而不适用于原始的简单数组。例如下列对象的数组:

```
var data = [
  {date: new Date(2011,  1, 1), value: 0.5},
  {date: new Date(2011,  2, 1), value: 0.6},
  {date: new Date(2011,  3, 1), value: 0.7},
  {date: new Date(2011,  4, 1), value: 0.8}
];
```

一个合适的二等分函数可定义为：

```
var bisect = d3.bisector(function(d) { return d.date; }).right;
```

然后调用 `bisect(data, new Date(2011, 1, 2))` ，返回索引。 如果你想使用不同于自然排序的方法对值进行排序，那么可以使用比较器（comparator）而不是访问器（accessor），例如降序排序而不是升序排序的时候。

```
var bisect = d3.bisector(function(a, b) { return a.date - b.date; }).right;
```

d3.**shuffle**(*array*[, *lo*[, *hi*]])

使用Fisher–Yates shuffle来把传入参数*array*随机排序.

关联数组 （Associative Arrays）

关联数组(字典)和数组类似，由以名称作为键的字段和方法组成。 它包含标量数据，可用索引值来单独选择这些数据，和数组不同的是， 关联数组的索引值不是非负的整数而是任意的标量。这些标量称为Keys，可以在以后用于检索数组中的数值. JavaScript 中另一种常见数据类型就是关联数组，或者简单说就是具有一系列命名属性的对象。在Java中简称映射(键值对) map，而在Python中称为字典dictionary。JavaScript为关联数组中键（属性名称）的迭代提供一个标准机制：那就是 for...in loop。然而，注意迭代的次序是未定义的。D3提供了一些将关联数组转化为索引数组的方法。

d3.**keys**(*object*)

返回一个包含指定对象(关联数组) 属性名称的数组。返回数组的顺序未定义。

d3.**values**(*object*)

返回一个包含指定对象(关联数组) 属性值的数组。返回数组的顺序未定义。

d3.**entries**(*object*)

返回一个包含对象(*object*)(一个关联数组)中名称以及值(键和值， key and value)的数组 (array)。每一个实体都是有键值对的对象，例如 `{key: "foo", value: 42}` 。返回数组的顺序未定义。

```
d3.entries({foo: 42,  bar: true}); // returns [{key: "foo", value: 42}]
```

映射（Maps）

当你尝试在JavaScript中用空对象作为map，当内部属性名称(键)作键时，会导致意外的行为(unexpected behavior)。比如，当你设置 `object["__proto__"] = 42` 时，最终不会达到你理想中的结果。又如你尝试查询给定key是否在map中定义了；"hasOwnProperty" in object 返回true，因为空对象(从对象原型)继承了hasOwnProperty方法。为避免这些问题，ES6提出了简单映射和集合(simple maps and sets) 理论。直到现代浏览器支持了这些集合，你可以使用[d3.map]替代.

注意：不同于建议的 ES6 map ，d3.map的key仍然强制使用字符串，而不是严格的相等。

d3.**map**([*object*][, *key*])

构建一个新的map，如果指定参数*object*，复制参数*object*对象内所有枚举属性到map中。参数对象可能是数组. 可以使用一个键*key*函数来计算数组里每个数值的键. 如下:

```
var m = d3.map([{name: "foo"}, {name: "bar"}], function(d) { return d.name; });
m.get("foo"); // {"name": "foo"}
m.get("bar"); // {"name": "bar"}
m.get("baz"); // undefined
```

参见 d3.nest。

map.**has**(*key*)

当且仅当map有指定key的实体时返回true。注意：该值可能是 `null` 或 `undefined` 。

map.**get**(*key*)

返回参数*key*的值。如果map中没有参数*key*相同元素，返回 `undefined` 。

map.**set**(*key*, *value*)

指定*key*的*value* ； 返回新的*value*。如果map之前同样的*key* 有一个实体了，那么旧实体被新值替代。

map.**remove**(*key*)

若map有指定*key*的实体，删除此实体并返回 `true` 。否则，此方法不做任何操作，返回 `false` 。

map.**keys**()

返回在这个map所有的键的数组。返回键的集合顺序是随机的。

map.**values**()

返回在这个map所有的值的数组，返回值的集合顺序是随机的。

map.**entries**()

返回一个map内所有键-值对象的数组。返回元素的集合顺序是随机的。任何元素的键必须是字符串类型，但值可为任何类型。

map.**forEach**(*function*)

给map中每个元素调用一个指定函数*function*，传递元素的键和值作为两个参数。*function*的使用的 `this` 指针将指向这个map。返回 `undefined` 。迭代的顺序是随机的。

map.**empty**()

返回 `true` 当且仅当map中没有元素。

map.**size**()

返回map中元素的个数

集合（Sets）

d3.**set**([*array*])

新建一个集合，如果指定了*array* ，添加*array* 的字符串值到返回集合中。

set.**has**(*value*)

当且仅当集合中具有指定参数*value* 字符串相同的实体，返回 `true` 。

set.**add**(*value*)

添加指定参数*value* 字符串到集合中.

set.**remove**(*value*)

如果集合中含有指定参数*value* 字符串相同元素，返回 `true` 并删除元素。否则，这个方法不做任何操作，并返回 `false` 。

set.**values**()

返回一个由集合中所有字符串类型值组成的数组。数组中的值的顺序为随机的。可作为集合中唯一值的简便计算方法(去重)。例如:

```
d3.set(["foo", "bar", "foo", "baz"]).values(); // "foo", "bar", "baz"
```

set.**forEach**(*function*)

给集合中每个元素调用一个指定*function*，传递元素的值作为参数。*function*的使用的 `this` 指针将指向这个map。返回 `undefined` ，迭代的顺序是随机的。

set.**empty**()

当且仅当集合中没有值，返回 `true` 。

set.**size**()

返回集合中值的个数。

数组运算符 (Array Operators)

d3.**merge**(*arrays*)

合并指定参数*arrays* 为一个数组，此方法和数组内置方法 `concat` 类似；唯一不同是当你要处理二维数组时，`d3.merge(arrays)` 方法更方便。

```
d3.merge([ [1], [2, 3] ]); // returns [1, 2, 3]
```

d3.**range**([*start*,]*stop*[, *step*])

生成一个包含算数级数的数组，类似于Python的内置函数range。这个方法常用来遍历一个数字序列或者整型数值。例如数组中的索引。不同于Python版本，这个参数不必是整形。尽管如果它们是浮点精度类型时这个结果更加可预测。如果省略*step*，默认值是1。如果省略*start*参数，默认值就是0。结果中不包含*stop*值。完整的形式是返回一个数字数组 `[start,start+step,start+2 *step,…]` 。如果*step*是正的，则最后一个元素是小于 `stop` 的 `start+i*step` 中的最大数值；如果*step*是负的，最后一个元素是大于 `stop` 的 `start + i*step` 中的最小数值。如果返回的数组将包含值无限多数字，就会抛出一个错误，而不是造成无限循环。

d3.**permute**(*array*, *indexes*)

使用指定的*indexes*数组返回指定数组的转置。返回数组包含indexes数组中索引对应的元素，按顺序。例如，`permute(["a", "b", "c"], [1, 2, 0])` 返回 `["b", "c", "a"]` 。indexes数组的长度和array中的元素长度不一样是可以接受的，并且允许indexes数组重复或者省略。 这个方法可以用来按固定顺序提取对象中的值到一个数组中。 （在JavaScript中indexes数组是和 `.length` 有特殊关系的简单属性）。按顺序提取带键的值可以用来生成嵌套选择中的数据数组。例如，我们可以用表格形式展示上述的一些明尼苏达州大麦产量数据：

```
var cols = ["site", "variety", "yield"];
thead.selectAll('th').data(cols)
    .enter().append('th').text(function (d) { return d.toUpperCase(); });
tbody.selectAll('tr').data(yields)
    .enter().append('tr').selectAll('td').data(function (row) { return
d3.permute(row, cols); })
        .enter().append('td').text(function (d) { return d; });
```

d3.**zip**(*arrays*...)

返回的数组的数组，其中，第i个数组包含来自每个*arrays*参数的第i个元素。返回的数组长度被截断为*arrays*的最短的数组的长度。如果*arrays*只包含一个数组，则返回的数组是包含一个元素的数组。不带任何参数，则返回的数组是空的。

```
d3.zip([1, 2], [3, 4]); // returns [[1, 3], [2, 4]]
```

d3.**transpose**(*matrix*)

等价于`d3.zip.apply(null, matrix)`；使用zip操作符作为二维矩阵变换(matrix transpose)。

d3.**pairs**(*array*)

对指定参数array中元素的每个相邻对，返回元组(元素i和元素i-1)的新数组。例如：

```
d3.pairs([1, 2, 3, 4]); // returns [[1, 2], [2, 3], [3, 4]]
```

如果指定参数array 中少于两个元素，则返回一个空数组。

嵌套 (Nest)

嵌套允许数组中的元素被组织为分层树型结构；类似SQL语句里面的GROUP BY方法，但不能多级别分组，而且输出的结果是树而不是一般的表。树的层级由key方法指定。树的叶节点可以按值来排序，而内部节点可以按键来排序。可选参数汇总（rollup）函数可以使用加法函数瓦解每个叶节点的元素. nest 操作符(d3.nest返回的对象)是可以重复使用的，不保留任何嵌套数据的引用。

例如，思考下面1931-2年间明尼苏达州（美国州名）麦田地皮的表格数据结构：

```
var yields = [{yield: 27.00, variety: "Manchuria", year: 1931, site: "University
Farm"},
              {yield: 48.87, variety: "Manchuria", year: 1931, site: "Waseca"},
              {yield: 27.43, variety: "Manchuria", year: 1931, site: "Morris"}, ...]
```

为了方便查看，可以嵌套元素首先按year然后按variety，如下：

```
var nest = d3.nest()
    .key(function(d) { return d.year; })
    .key(function(d) { return d.variety; })
    .entries(yields);
```

返回的嵌套数组中。每个元素的外部数组是键-值对，列出每个不同键的值:

```
[{key: 1931, values: [
    {key: "Manchuria", values: [
        {yield: 27.00, variety: "Manchuria", year: 1931, site: "University Farm"},
        {yield: 48.87, variety: "Manchuria", year: 1931, site: "Waseca"},
        {yield: 27.43, variety: "Manchuria", year: 1931, site: "Morris"}, ...]},
    {key: "Glabron", values: [
        {yield: 43.07, variety: "Glabron", year: 1931, site: "University Farm"},
        {yield: 55.20, variety: "Glabron", year: 1931, site: "Waseca"}, ...]},
```

```
...]},
 {key: 1932, values: ...}]
```

嵌套的形式可以在SVG或HTML很容易迭代和生成层次结构。

有关 `d3.nest` 详见:

- Phoebe Bright's D3 Nest Tutorial and examples
- Shan Carter's Mister Nester

d3.**nest**()

创建一个新操作符。keys集合初始为空。如果map或entries 操作符在任何键函数被注册之前被调用，这个嵌套操作符通常返回输入数组。例如http://bl.ocks.org/phoebebright/raw/3176159/

nest.**key**(*function*)

注册一个新的键函数*function*。键函数将被输入数组中的每个元素调用，并且必须返回一个用于分配元素到它的组内的字符串标识符。通常，这个函数被实现为一个简单的访问器，如上面例子中year和variety的访问器。 输入的数组的引导（index）并*没有*传递给*function*。每当一个key 被注册，它被放到一个内部键数组的末端，和由此产生的map或实体将有一个额外的层级。当前没有一个方法可以删除或查询注册的键。最近注册的键在后续的方法中被当作当前键。

nest.**sortKeys**(*comparator*)

使用指定的参数*comparator*来给当前键值排序，等效于d3.descending。如果没有指定comparator 参数键的排序则返回undefined。注意：只影响实体操作符的结果；map操作符返回键的顺序永远是undefined，无论什么比较器。

```
var nest = d3.nest()
    .key(function(d) { return d.year; })
    .sortKeys(d3.ascending)
    .entries(yields);
```

nest.**sortValues**(*comparator*)

使用指定的*comparator*参数给叶子元素排序，等效于d3.descending。这相当于在应用nest操作符前给输入的数组排序；然而，当每一组更小时它通常是更有效的。如果没有指定值的比较器，元素则按输入数组中显示的顺序排列。通常用于map和实体（entries）的操作符。

nest.**rollup**(*function*)

为每组中的叶子元素指定汇总函数（rollup ）*function*。汇总函数的返回值会覆盖叶子值数组。不论是由map操作符返回的关联数组，还是实体操作符返回的每个实体的值属性。

nest.**map**(*array*[, *mapType*])

对指定的数组使用nest操作符，返回一个关联数组。返回的关联数组*array*中每个实体对应由第一个key函数返回的不同的key值。实体值决定于注册的key函数的数量：如果有一个额外的key，值就是另外一个嵌套的关联数组；否则，值就是过滤自含有指定key值得输入数组*array*的元素数组。

如果指定了*mapType*，指定的函数就会用来构造一个map而不是返回一个简单的JavaScript对象。推荐使用d3.map 实现这个目的，例如：

```
var yieldsByYearAndVariety = d3.nest()
    .key(function(d) { return d.year; })
```

```
    .key(function(d) { return d.variety; })
    .map(yields, d3.map);
```

使用d3.map而不是一个对象提供便捷（例如：返回的map含有 keys 和 values 函数），防止不寻常的键名与JavaScript内置的属性冲突，例如 `__proto__` 。

nest.**entries**(*array*)

为指定的*array*参数应用nest操作符，返回一个键值对数组。从概念上讲，这和对 map 返回的关联数组应用 d3.entries 操作符类似，但是这个是用于每一层而不是只有第一层（最外层）。返回数组中的每个实体对应于第一个key函数返回的不同的key值。实体值取决于注册的key函数的数量：如果有一个关联的key，值就是另外一个嵌套实体数组；否则，值就是含有指定key值得输入数组*array*过滤得到的元素数组。嵌套的案例：
http://bl.ocks.org/phoebebright/raw/3176159/

[1] accessor function, 亦为getter, callback, 访问器, 常译作'回调（函数）'。术语翻译 (计算机软件/编程) // Howard Liang 注 Nov 27, 2015

- Carry on 、2014-3-29翻译
- 咕噜2014-11-18翻译，并校对之前的翻译。