> *Wiki* ‣ *[[API--中文手册]]* ‣ *[[核心函数]]* ‣ ***选择器***

- 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.js:437278817，大数据可视化：436442115
- Github小组：VisualCrew
- demo: https://github.com/tianxuzhang/d3-api-demo/tree/master/core/selection

一个**选择**就是从当前文档中抽取的一组元素。D3使用[[CSS3|http://www.w3.org/TR/css3-selectors/]]来选择页面元素。例如，你可以使用的选择方式有标签 ("div")、类(".awesome")、唯一标识符("#foo")、属性("[color=red]")、或者包含("parent child")。选择器可以是交叉(".this.that" 表示逻辑与)的也可以是联合的(".this, .that" 表示逻辑或)。如果你的浏览器不支持选择器，你可以在D3前包含[[Sizzle|http://sizzlejs.com/]]来支持向后兼容。

当选择完元素之后你可以添加操作。这些操作可以设置或者获取属性、样式、特性、选择器和文本内容。属性值等可以被指定为常量或者函数；后者为每个元素求值。你也可以为数据选择器加入到data操作；这个数据就可以用做数据驱动的变换的操作符。另外，加上数据产生enter和exit子选择器，你就可以add和remove元素来响应数据中的改变。

一般来说，在D3中你不需要使用for循环后者递归来修改文档。因为你可以一次操作这个选择器而不是遍历各个元素。当然，你也可以手动地遍历各个元素，有一个each操作可以调用任意一个函数，选择器的结果是数组，所以元素可以访问(例如 `selection[0][0]`)。为了简便D3支持方法链：这个操作返回值是一个选择器。

选择元素

D3提供了两种高级方法来选择元素：select和selectAll。这些方法接收选择器字符串。前者只返回第一个匹配的元素，后者选择在文档遍历次序中所有匹配的元素。这个方法也可以接受节点，这可以用来和第三方库例如JQuery或者开发工具(`$0`)整合。

d3.**select**(*selector*)

选中与指定选择器字符串匹配的第一个元素，返回单元素选择结果。如果当前文档中没有匹配的元素则返回空的选择。如果有多个元素被选中，只有第一个匹配的元素（在文档遍历次序中）被选中。

d3.**select**(*node*)

选择指定的节点。如果你已经有一个节点的引用这将是很有用的。例如事件监听器中的 `d3.select(this)` 或者一个全局对象例如 `document.body` 。这个函数不会遍历DOM树。

d3.**selectAll**(*selector*)

选中匹配指定选择器的所有的元素。这些元素会按照文档的遍历顺序（从上到下）选择。如果当前文档中没有匹配的元素则返回空的选择。

d3.**selectAll**(*nodes*)

选择指定的元素数组。如果你已经有了一些元素的引用这将是很有用的，例如事件监听器中的 `d3.selectAll(this.childNodes)` ，或者一个全局对象例如 `document.links` 。节点的参数不用恰好是数组；任何可以转化为一个数组的伪数组（例一个 `NodeList` 或者 `arguments` ）都可以，这个函数不会遍历DOM树。

操作选择

选择是一组元素。D3绑定额外的方法到数组上。所以你可以在选中的元素上应用操作，例如为所有选中的元素设置属性值。一个细微的差别是选择结果是分组的，而不是一个一维数组。每一个选择都是元素数组中的一个数组。这保留了自选择的层次结果，大多数情况下你可以忽略这个细节，这就是为什么一个单一的元素选择看起来像 `[[node]]`

而不是 `[node]` 。想要了解更多关于嵌套选择的知识请参考嵌套选择。如果你想要学习选择器是怎么工作的你可以用浏览器的开发者工具控制台交互式地访问数据。你可以检查返回数组来查看哪些元素被选中了以及它们是如何分组的。你还可以再应用操作来选择元素并查看页面内容的改变。

内容

D3有一系列可以影响文档内容的操作。这些是你最常用来展示数据的。当操作用来设置文档内容的时候会返回当前选择，所以你就可以使用一个简单的声明将多个操作链接在一起。

selection.**attr**(*name*[, *value*])

如果指定了*value*参数，将为所有选中的元素通过指定的*name*为指定的*value*设置属性。如果*value*是一个常数，那么所有的元素都将设置为相同的属性值；如果*value*是一个函数，那么这个函数会为每个选中的元素（按顺序）计算。入参是当前数据元素 `d` 和当前索引 `i` ，以及代表当前DOM元素的 `this` 上下文。这个函数的返回值接下来用来设置每个元素的属性。null值将移除指定的属性。

如果*value*参数没有指定，就会返回为选择中第一个非空（null）元素所指定的属性值。一般来说，只有当你知道选择中恰好包含一个元素时才有用。

指定的*name*参数也会含有一个前缀，例如 `xlink:href` 是用来指定Xlink命名空间中*href*属性的。默认情况下，D3支持svg、xhtml、xlink、xml和 xmlns命名空间。可以添加d3.ns.prefix来注册其他的命名空间。

*name*也可以是一个*name*和*value*属性对象。

selection.**classed**(*name*[, *value*])

这个操作是用来设置*class*属性值得便捷程序。它能识别*class*属性是一个按照空格分隔的标记集合。这样它就能使用[[classList|https://developer.mozilla.org/en/DOM/element.classList]]（如果有的话）来方便地添加、移除和切换CSS类。

如果*value*属性被指定，不论是否指定类都会与选定元素相结合。如果*value*是一个常量且其值为真，那么所有的元素都会被分配指定的类（还没分配的话）。如果其值为假，那么就会移除选中元素的class（已经分配过）。如果*value*是一个函数，那么这个函数会为每个选中的元素（按顺序）计算。入参是当前数据元素 `d` 和当前索引 `i` ，以及代表当前DOM元素的 `this` 上下文。这个函数的返回值接下来用来分配或者不分配每个元素的class。

如果你想一次设置多个class可以使用一个对象，文字如同： `selection.classed({'foo': true, 'bar': false})` ，或者使用以空格分隔的class列表形如： `selection.classed('foo bar', true)` 。

如果*value*没有被指定，当且仅当选择中首个非空值有指定的class就会返回true。一般来说，只有当你知道选择中恰好包含一个元素时才有用。

selection.**style**(*name*[, *value*[, *priority*]])

如果*value*参数被指定，通过指定名称和指定的值为所有选中的元素设置CSS样式属性。如果*value*是一个常数，那么所有的元素都设置相同的样式值；否则，如果值是一个函数，则该函数为每个选定的元件（按顺序）计算，入参是当前数据元素 `d` 和当前索引 `i` ，以及代表当前DOM元素的 `this` 上下文。该函数的返回值被用来设置每个元素的样式属性。 null值将删除样式属性。可选参数*priority*也可以指定，无论是null空或字符串“important”（不带感叹号）。

如果你想一次设置多个样式属性，使用对象文本，如下所示：

```
selection.style({'stroke': 'black', 'stroke-width': 2})
```

如果未指定值，则返回在选择中的第一个非空元素指定样式属性的当前计算值。只有当你知道选择只包含一个元素时是很有用的。需要注意的是计算的值可能与先前设置的值不同，尤其是当样式属性使用了简写属性（如“font”样式，这是简写为"font-size"，"font-face","等）。

selection.**property**(*name*[, *value*])

一些HTML元素具有特殊的属性使用标准的属性或样式是不可寻址的。例如，表单文本（text）字段有一个 `value` 字符串属性，复选框（checkboxes）有一个 `checked` 布尔型属性。可以使用 `property` 操作符来获取或设置这些属性，，或任何其他基本元素的可寻址字段，例如 `className` 。

如果指定了*value*，就为所有选中的元素指定名称的属性设置指定的值（value）。如果值是一个常数，那么所有的元素被给予相同的属性值；如果*value*是一个函数，则该函数为每个选定的元素（按顺序）计算，入参是当前数据元素 `d` 和当前索引 `i` ，以及代表当前DOM元素的 `this` 上下文。该函数的返回值被用于设置每个元素的属性。空值将删除指定的属性。

如果你想一次设置多个属性，可以使用对象文本，如下所示： `.property({'checked':true,'disabled':false});` 。

如果未指定值，则返回在选择中第一个非空元素指定属性的值。只有当你知道选择只包含一个元素这通常是很有用的。

selection.**text**([*value*])

文本操作符是基于 `textContent` 属性；设置文本内容将取代任何现有的子元素。

如果指定了*value*时，设置所有选择元素的文本内容为指定的值。如果*value*是一个常数，那么所有的元素被赋予相同的文本内容；如果*value*是一个函数，则该函数被每个选定的元素（按顺序）计算，入参是当前数据元素 `d` 和当前索引 `i` ，以及代表当前DOM元素的 `this` 上下文。该函数的返回值被用于设置每个元素的文本内容。null值会清除内容。

如果未指定*value*，则返回在选择中第一个非空元素的文本内容。只有当你知道选择只包含一个元素时这通常是很有用的。

selection.**html**([*value*])

`html` 的操作基于[[innerHTML|http://dev.w3.org/html5/spec-LC/apis-in-html-documents.html#innerhtml]]属性；设置内部HTML内容将取代任何现有的子元素。此外，您可能更愿意使用数据驱动的方式`append`或`insert`操作创建HTML内容;该操作符的目的是，适用于你想用少量但有丰富格式的HTML。

如果指定了*value*，为所有选择的元素设置在内部的HTML内容为指定的值。如果*value*是一个常数，那么所有的元素被给予相同的内部HTML内容；如果*value*是一个函数，则该函数为每个选定的元素（按顺序）计算，入参是当前数据元素 `d` 和当前索引 `i` ，以及代表当前DOM元素的 `this` 上下文。该函数的返回值被用于设置每个单元的内部的HTML内容。null值会清除内容。

如果未指定*value*，则返回在选择中第一个非空元素的内部HTML内容。只有当你知道选择只包含一个元素时这通常是很有用的。 注：正如它的名字所暗示的，selection.html仅支持HTML元素。 SVG元素和其它非HTML元素不支持innerHTML属性，因此与selection.html不相容。请考虑使用XMLSerializer转换DOM树为文本。灵参见innersvg polyfill，它提供了一个垫片以支持SVG元素的innerHTML属性。

selection.**append**(*name*)

在当前选择的每个元素最后追加具有指定名称的新元素，返回包含追加元素的新选择。每个新的元素继承当前元素的数据（如果有的话）和select相同的方式使用子选择。

这个*name*可以被指定为一个常量字符串或一个函数，返回追加的DOM元素。当name被指定为一个字符串，它可能有以下形式的命名空间前缀"namespace:tag"。例如，"svg:text"将在svg命名空间创建"text"元素。默认情况下，D3支持svg，xhtml，xlink的，xml和xmlns命名空间。其他的命名空间可以通过添加到d3.ns.prefix注册。如果没有指定命名空间，那么命名空间会从封闭的元素继承；或者，如果该名称是已知的前缀之一，相应的命名空间将被使用（例如，"svg"表示"svg:svg"）。

selection.**insert**(*name*[, *before*])

在当前选择与指定before选择器匹配的每个元素之前插入具有指定*name*的新元素，返回包含插入的元素的一个新的选择。如果before选择器不匹配任何元素，那么新元素将用append追加为最后一个子元素。每一个新元素继承当前元素（如果有的话）的数据，子选择（subselections）和select以同样的方式。

这个*name*可以被指定为一个常量字符串或一个函数，返回追加的DOM元素。当*name*被指定为一个字符串，它可能有以下形式的命名空间前缀"namespace:tag"。例如，"svg:text"将在svg命名空间创建"text"元素。默认情况下，D3支持svg，xhtml，xlink的，xml和xmlns命名空间。其他的命名空间可以通过添加到d3.ns.prefix注册。如果没有指定命名空间，那么命名空间会从封闭的元素继承；或者，如果该名称是已知的前缀之一，相应的命名空间将被使用（例如，"svg"表示"svg:svg"）。

同样地，*before*选择器可以被指定为一个选择器字符串或一个函数，它返回一个DOM元素。例如，`insert("div", ":first-child")` 将在当前选择前面加上div子节点。对于enter选择器，*before*选择器在这种情况下也可以省略：输入的元素将被立即插入到更新选择紧随的兄弟元素前（如果有的话）。这使您可以插入DOM的元素与绑定的数据是一致的顺序。但是请注意，如果更新元素修改了顺序，selection.order可能仍然需要。

selection.**remove**()

删除从当前文档当前选择中的元素。返回"屏幕外（off-screen）"的当前选择（除去了相同的元素），从DOM分离。需要注意的是目前还没有一个专门的API来重新添加删除的元素到文档；然而，你可以通过一个函数来selection.append或selection.insert重新添加元素。

###数据

selection.**data**([*values*[, *key*]])

连接指定的一组数据的和当前选择。指定的*values*是一组数据值（例如，数字或对象）或一个函数返回一组值。如果没有指定*key*函数，则*values*的第一数据被分配到当前选择中第一元素，第二数据分配给当前选择的第二个元素，依此类推。当数据被分配给一个元素，它被存储在属性 `__data__` 中，从而使数据"沾粘"，从而使数据可以再选择。

`data` 操作的结果是*update*选择;这表示选择的DOM元素已成功绑定到指定的数据元素。*update*选择还包含对enter和exit的选择，对应于添加和删除数据节点。有关详细信息，请参阅简短的教程关于连接的思考

*key*函数可以被指定为控制数据是如何连接元素。这取代默认的by-index行为;*key*函数被新数据数组中的每个元素调用一次，并再次用于选择中的每个元素。在这两种情况下的*key*函数是通过传递数据d与索引i。当key 函数上被新的数据元素评价时，`this` 上下文是数据数组；当key 函数被现有选择评估时，`this` 上下文是相关的DOM元素。*key*函数，基于先前结合的数据返回一个用于连接数据和相关的元素的字符串。例如，如果每个数据都有一个唯一的字段 `name` ，该连接可以被指定为 `.data(data, function(d) { return d.name; })` 。如果指定了*key*函数，`data` 操作符也影响节点的索引；该索引被作为第二个参数 `i` 作为任何运算符函数的参数。然而，请注意，现有的DOM元素不自动重新排序；根据需要使用sort或order函数。有关*key*函数如何影响数据连接的更详细的示例，请参阅教程[[条形图2|http://bost.ocks.org/mike/bar/2/]]

这个*values*选择中的**每组数据**。因此，如果选择具有多个组（例如，一个d3.selectAll后跟一个selection.selectAll）），然后data应该被指定为一个函数，该函数返回一个数组（假设你对每个组想要不同的数据）。该函数将被传递的当前组数据（或 `undefined` ）和索引，组的 `this` 上下文。 例如，可以将一个二维数组和初始选择绑定，然后将包含的内部数组和每个子选择绑定。在这种情况下，*values*函数是标识函数：它被每个组中的子元素调用，被传递绑定到父元素的数据，并且返回这个数据数组。

```
var matrix = [
  [11975,  5871, 8916, 2868],
  [ 1951, 10048, 2060, 6171],
  [ 8010, 16145, 8090, 8045],
  [ 1013,   990,  940, 6907]
```

```
];

var tr = d3.select("body").append("table").selectAll("tr")
    .data(matrix)
  .enter().append("tr");

var td = tr.selectAll("td")
    .data(function(d) { return d; })
  .enter().append("td")
    .text(function(d) { return d; });
```

如果未指定*values*，则此方法返回选择中的第一组数据的数组。返回的数组的长度，将与第一组的长度匹配，并且在返回的数组中的每个数据的索引将匹配选择中相应的索引。如果选择的某些元素为null，或者如果他们有没有相关的数据，则数组中的相应元素将是 `undefined` 。

注意： `data` 方法不能用于清除先前结合数据；可以使用selection.datum代替。

selection.**enter()**

返回输入（enter）选择：当前选择中存在但是当前DOM元素中还不存在的每个数据元素的占位符节点。此方法只在由data运算符返回的更新选择中定义。此外，输入选择只定义了append（append），insert（insert），select（select）和call（call）操作符；您必须使用这些操作符在修改任何内容之前实例化输入元素。当你传递函数的参数给这些插入的元素的操作符时，index参数将反映新的位置，而不一定从零开始或者是连续的。 （输入选择也支持empty和size。）

举一个简单的例子，考虑现有的选择是空的情况下，我们希望创建新的节点来匹配我们的数据：

```
d3.select("body").selectAll("div")
    .data([4, 8, 15, 16, 23, 42])
  .enter().append("div")
    .text(function(d) { return d; });
```

假设body最初是空的，上面的代码将创建六个新的div元素，将它们按顺序追加到body，并指定其文本内容为相应的（强制转为字符串）号码：

```
<div>4</div>
<div>8</div>
<div>15</div>
<div>16</div>
<div>23</div>
<div>42</div>
```

另一种方式考虑进入的占位符的节点是，它们是指向父节点（在该示例中，就是文档的body）；然而，他们只支持追加和插入。 当你追加或插入时输入选择并入到**更新选择**。而不是对enter和update选择单独采用同样的操作符，现在你可以一次添加节点后，将其应用到更新选择。如果你发现自己移除整个选择的元素才重新插入其中大部分，用这个来替代。例如：

```
var update_sel = svg.selectAll("circle").data(data)
update_sel.attr(/* operate on old elements only */)
update_sel.enter().append("circle").attr(/* operate on new elements only */)
```

```
update_sel.attr(/* operate on old and new elements */)
update_sel.exit().remove() /* complete the enter-update-exit pattern */
```

selection.**exit()**

返回退出（exit）选择：找出在当前选择存在的DOM元素中没有新的数据元素时。此方法只被定义在data运算符返回的更新选择。exit选择定义了所有的正常操作符，但通常你主要使用的是remove；其他操作符存在的主要目的是让您可以根据需要定义一个退出的过渡。请注意，exit操作符只是返回一个exit选择引用，由你来删除新节点。 一个简单的例子，考虑更新在上面的例子中的enter操作符创建的六个DIV元素。在这里，我们把这些元素和一个含有一些新的，一些旧数据的新数组绑定：

```
var div = d3.select("body").selectAll("div")
    .data([1, 2, 4, 8, 16, 32], function(d) { return d; });
```

现在div--data操作符的结果--指的是更新的选择。因为我们指定的key函数使用标识函数，并且将新数据数组包含数字[4,8,16]，它也存在旧的数据数组中，这个更新选择包含3 个DIV元素。比方说，我们离开这些元素原样。我们可以实例化并使用enter选择添加新的元素[1,2,32]：

```
div.enter().append("div")
    .text(function(d) { return d; });
```

同样，我们可以删除退出的元素[15, 23, 42]： div.exit().remove();

```
现在，文档body如下：
<div>4</div>
<div>8</div>
<div>16</div>
<div>1</div>
<div>2</div>
<div>32</div>
```

注意，DOM元素现在是乱序。然而，选择索引i（操作函数的第二个参数），将正确地与新数据数组相匹配。例如，我们可以指定一个索引属性：

```
d3.selectAll("div").attr("index", function(d, i) { return i; });
```

结果是：

```
<div index="2">4</div>
<div index="3">8</div>
<div index="4">16</div>
<div index="0">1</div>
<div index="1">2</div>
<div index="5">32</div>
```

如果你想在文档遍历，以匹配选择数据顺序，可以使用sort或者order。

selection.**filter**(*selector*)

过滤选择，返回一个新的选择只包含其指定的*selector*是true的元素。*selector*可以被指定为一个函数或作为选择的字符串，如".foo"。和其他操作符一样，该函数被传递当前数据 `d` 和索引 `i` ，以及this上下文作为当前的DOM元素。过滤器应该只在选择与DOM元素绑定的时候。例如：从append或insert。只绑定的元素和数据的一个子集，可以在data参数中调用内置的数组过滤器filter。像内置函数一样，D3的过滤器*不会*在返回选择中保留原来的选择的索引；返回移除元素的副本。如果您想保留的索引，使用select代替。 例如，要选择奇数索引（相对于从零开始的索引）的每一个元素：

```
var odds = selection.select(function(d, i) { return i & 1 ? this : null; });
```

等价地，使用的过滤器函数：

```
var odds = selection.filter(function(d, i) { return i & 1; });
```

或过滤选择器（注意：nth-child伪类是一开始的索引，而不是从零开始的索引）：

```
var odds = selection.filter(":nth-child(even)");
```

因此，您可以使用选择或过滤器操作符应用到元素的一个子集上。

selection.**datum**([*value*])

获取或设置每个选定的元素绑定的数据。不像selection.data方法，这种方法不计算一个连接（并因此不计算enter和exit的选择）。此方法在selection.property之上实现：

```
d3.selection.prototype.datum = function(value) {
  return arguments.length < 1
      ? this.property("__data__")
      : this.property("__data__", value);
};
```

如果指定*value*，就为所有选中的元素设置元素的绑定数据为指定的值。如果*value*是一个常数，所有的元素被给予相同的数据；否则，如果*value*是一个函数，则该函数为每个选定的元素计算，被传递以前的数据 `d` 与当前索引 `i` ，使用 `this` 上下文作为当前的DOM元素。该函数之后被用来确定每个元素的数据。null值将删除绑定的数据。该操作数对索引没有影响。

如果未指定*value*，则返回在选择中绑定第一个非空的元素的数据。只有当你知道选择只包含一个元素这通常是很有用的。

注意：此方法是以前所谓的"map"。旧名已被弃用。

datum 方法用D3访问HTML5自定义数据属性非常有用。例如，给定下列元素：

```
<ul id="list">
  <li data-username="shawnbot">Shawn Allen</li>
  <li data-username="mbostock">Mike Bostock</li>
</ul>
```

你可以公开通过设置每个元素的数据作为内置的dataset属性自定义数据属性D3：

```
selection.datum(function() { return this.dataset; })
```

这可以用来，例如，按照用户名排序元素。示例

selection.**sort**([*comparator*])

根据指定的*comparator*函数对当前选择的元素排序。比较器函数默认为d3.ascending，通过传递两个数据元素*a*和*b*进行比较，并返回任一负的，正的，或零值。如果为负，则*a*应该在*b*之前；如果为正，则*a*应该为*b*后；否则a和b被认为是相等的顺序是任意的。需要注意的是，这种排序是不保证是稳定的；然而，它保证与浏览器内置的数组排序方法具有相同的行为。

selection.**order**()

重新插入元素到文档这样文档顺序选与择顺序就相匹配。这等同于调用sort()如果数据已经排序，但要快得多。

动画和交互

selection.**on**(*type*[, *listener*[, *capture*]])

在当前选择的每个元素，为指定的类型*type*，添加或删除事件监听器*listener* 。*type*是一个字符串事件类型的名称，如"点击"，"鼠标悬停"，或"提交"。基本上支持任何DOM事件。有关D3支持的事件类型的更多详细信息，请查看Mozilla或Stackoverflow。指定的listener 与其他操作符函数调用方式相同，被传递的当前数据 `d` 和索引 `i` 与 `this` 上下文作为当前的DOM元素。为了在侦听器内访问当前事件，使用全局函数d3.event。事件侦听器的返回值将被忽略。

如果所选择的元素相同类型的一个事件监听已经注册了，新的侦听加入之前的现有侦听被除去。为注册相同事件类型的多个监听器，该类型可以跟一个可选的命名空间，如"click.foo"和"click.bar"。 要删除一个监听器，传递null给*listener*，删除特定事件类型所有监听，传递null给listener，指定 `.type` 的类型，如： `selection.on(".foo", null)` 。

一个可选的捕获*capture*标志可以指定，对应于W3C的useCapture标志："开始捕获后，所有指定类型的事件将被分派到注册的EventListener在被分派到事件树任何EventTargets下，事件是沿着树向上冒泡而不会触发一个EventListener指定使用捕获"。

如果未指定监听器，指定类型（如果有）返回当前分配的监听器。

d3.**event**

存储当前的事件（如果有的话）。这个全局函数是在事件侦听器回调过程中使用on操作符注册的。当监听器在 `finally` 块被通知后当前事件被重置。这使侦听器函数与其他操作符函数具有相同的形式，传递当前数据 `d` 和索引 `i` 。 `d3.event` 对象是DOM事件，并实现了标准事件字段，像时间戳 `timeStamp` 和键代码 `keyCode` ，以及 `preventDefault()` 方法和的 `stopPropagation()` 方法。当然你可以使用原生事件的pageX and pageY，它往往更方便转变事件位置为接收该事件容器的局部坐标系。例如，如果你在网页的正常流程嵌入SVG，你可能想事件的位置是相对于SVG图像的左上角的。如果您的SVG包含转换，你也可能想知道事件的相对于那些变换的位置。标准鼠标指针使用d3.mouse运算符，d3.touches用在iOS多点触控事件上。

d3.**mouse**(*container*)

返回当前d3.event相对于指定的容器的x和y坐标，。该容器可以是一个HTML或SVG容器元素，如svg:g或svg:svg。该坐标返回为一个包含两个元素的数组[*x*, *y*]。

d3.**touch**(*container*[, *touches*], *identifier*)

返回指定标识符触摸当前d3.event相应的x和y坐标，相对于指定容器。如果未指定*touches*时，默认为当前事件的changedTouches。该容器可以是一个HTML或SVG容器元件，如一个的svg:g或svg:svg。坐标被返回为两元素数组的数组[[*x1*, *y1*], [*x2*, *y2*], …]。如果在指定的标识符*touches*没有接触，则返回null；这对于忽略touchmove那些只是touches移动了的事件是很有用的。

d3.**touches**(*container*[, *touches*])

返回与当前d3.event相关联的每个触摸*x*和*y*坐标，基于touches属性，相对于指定的容器中。该容器可以是一个HTML或SVG容器元素，如svg:g 或 svg:svg。坐标返回两个元素的数组的数组[[*x1*, *y1*], [*x2*, *y2*], ...]。如果指定了触摸，返回指定触摸的位置，如果没有指定*touches*时，则默认为当前事件的`touches`属性。

selection.**transition**([*name*])

开始为当前选择的过渡。转换的行为很像选择，除了操作符动画平滑的随着时间的推移，而不是瞬间完成。

selection.**interrupt**([*name*])

立即中断当前的过渡（如果有的话）。不会取消任何尚未启动的预定的转变。最好要取消原定过渡，简单地创建中断当前的过渡后，一个新的零延迟过渡：

```
selection
    .interrupt() // 取消当前过渡
    .transition(); //抢占任何预定的转换
```

子选择

顶层的select方法查询整个文档，一个选择的select和selectAll操作符限定查询每个选定元素的后代；我们称之为"部分选择"。例如，`d3.selectAll("p").select("b")`返回在每一个段落("p")元素中的第一个bold("b")元素。部分选择通过selectAll通过祖先分组元素。因此，`d3.selectAll("p").selectAll("b")`。通过段落分组而d3.selectAll("p b") 返回一个扁平的选择。部分选择通过select是相似的，但保留了团体和传播数据。分组起在数据连接中起重要的作用，和功能性操作符可依赖于其组内的当前元素的数值索引。

selection.**select**(*selector*)

对当前选中的每个元素，选中第一个匹配特定的选择器字符串*selector*的子代元素。如果当前元素没有元素匹配特定的选择器，当前索引处的元素在返回的选择中将是空值null。运算符（除了data）自动跳过null元素，从而保持现有选择的索引。如果当前元素具有相关联的数据，该数据由返回的子选择继承，并且自动绑定到新选定的元素。如果有多个元素匹配选择器，只有在文档遍历顺序中第一个匹配的元素会被选中。选择器也可以被指定为一个函数，返回一个元素，或者null（如果没有匹配的元素）。在这种情况下，指定的选择器以和其他操作函数同样的方式被调用，被传递的当前数据`d`和索引`i`，与`this`上下文作为当前的DOM元素。

selection.**selectAll**(*selector*)

对当前选择的每个元素，选取匹配指定选择器字符串的后代元素。返回的选择是通过在当前选择的祖先节点进行分组。如果没有元素和当前元素的指定选择器相匹配，返回的选择中当前索引处的组将是null。部分选取不从当前选择继承数据，但如果数据值指定为一个函数，这个函数会被调用，带有的祖先节点的数据d和组索引i来确定部分选定的数据绑定。

通过selectAll的分组也将影响后续进入的占位节点。因此，为追加进入的节点指定父节点，使用select后紧跟selectAll：

```
d3.select("body").selectAll("div")
```

您可以通过检查各组数据的parentNode属性查看每个组的父节点，例如selection[0].parentNode。

选择器也可以被指定为一个函数，返回元素的数组（或节点列表NodeList），或者空数组（如果没有匹配的元素）。在这种情况下，指定的选择器以和其他操作函数同样的方式被调用，被传递的当前数据`d`和索引`i`，与`this`上下文作为当前的DOM元素。

控制

对于高级用法，D3有一些额外的用户控制流操作符。

selection.**each**(*function*)

为当前选择的每个元素，调用指定的函数，传递当前数据 `d` 和索引 `i` ，与当前的DOM元素的 `this` 上下文。这个操作符几乎被所有的其他操作符内部使用，并可以用于调用任意代码为每个选定的元素。 `each` 操作符可以用来处理递归的选择，通过在回调函数内使用 `d3.select(this)` 。

selection.**call**(*function*[, *arguments*…])

调用指定的函数一次，通过在当前的选择以及任何可选参数。无论指定函数的返回值是什么，call操作符总是返回当前的选择。通过call调用函数与手动调用函数是完全一样的；但它可以更容易地使用方法链。例如，假设我们要在许多不同的地方以同样的方式设置一些属性。我们采取的代码，把它包在一个可重复使用的功能：

```
function foo(selection) {
  selection
      .attr("name1", "value1")
      .attr("name2", "value2");
}
```

现在我们可以这样写：

```
foo(d3.selectAll("div"));
```

或者等价的方式：

```
d3.selectAll("div").call(foo);
```

被调用函数的 `this` 上下文也是当前的选择。第一个参数是略显多余的，这我们可能在未来解决。 如果您使用的 `selection.call` 的对象，方法和需要 `this` 指向该对象创建之前调用绑定到对象的函数。

```
function Foo(text) {
    this.text = text;
}

Foo.prototype.setText = function(selection) {
    selection.text(this.text);
}

var bar = new Foo("Bar");

d3.selectAll("span").call(bar.setText.bind(bar));
// Or
d3.selectAll("span").call(Foo.prototype.setText.bind(bar));
```

selection.**empty**()

返回true如果当前选择为空；一个选择是空的，如果它不包含任何元素或null元素。

selection.**node**()

返回当前选择的第一个非空的元素。如果选择为空，则返回null。

selection.**size**()

Returns the total number of elements in the current selection.

扩展

d3.**selection**()

返回根的选择，相当于 `d3.select(document.documentElement)` 。此函数还可以用于检查一个对象是一个选择： `o instanceof d3.selection` 。您还可以添加新的方法到选择的原形中。例如，添加一个便捷的方法，用于设置复选框的 `checked` 属性，你可能这样写：

```
d3.selection.prototype.checked = function(value) {
  return arguments.length < 1
      ? this.property("checked")
      : this.property("checked", value);
};
```

本文参与	人员	组织	时间
翻译	大傻	VisualCrew小组	20141102
校对/排版	大傻	VisualCrew小组	2015-11-02T12:00:51Z
校对	liang42hao	VisualCrew小组	2016-03-04T16:36:18Z