Wiki ‣ ***Selections (Выборки)***

Выборка это массив документов, полученных из текущего документа. D3 использует [[CSS3|http://www.w3.org/TR/css3-selectors/]] для выборки элементов. Например, вы можете делать выборку по тегу ("div"), классу (".awesome"), уникальному итендефикатору ("#foo"), аттрибуту ("[color=red]"), или вложенности ("parent child"). Выборки можно пересекать (".this.that" работает как логическое "*И*") или объединять (".this, .that" работает как логическое "И"). Если ваш браузер не поддерживает выборки по умолчанию, вы можете подключить [[Sizzle|http://sizzlejs.com/]] перед D3 для обратной совместимости.

После выборки элементов, вы можете применять к ним **операторы** чтобы делать с ними разные штуки. С помощью операторов можно получать или устанавливать значения аттрибутов, стилей, свойств, HTML и текстового контента. Значения аттрибутов и всего остального можно задавать константой или функцией; последняя вычисляется для каждлого элемента.

Вам совсем не нужно будет использовать `for` для циклов и рекурсивных функция для модификации документа с D3. Это связано с тем, что вы можете работать с целыми выборками, а не с каждым элементом в отдельности. Однако, вы все равно можете делать цикл по элементам, если хотите.

Выборка элементов

D3 предоставляет два высокоуровневых метода для выборки элементов: select и selectAll. Эти методы принимают селектор; первая выбирает только первый найденный элемент, а вторая выбирает *все* подходящие элементы из документа. Эти методы могут также принимать узлы, что полезно для интеграции со сторонними библиотеками, такими как JQuery.

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

Выбирает первый подходящий элемент по селектору. Если ни один из элементов в текущем документе не соответствует указанному селектору, возвращает пустую выборку. Если несколько элементов соответствуют селектору, только первый соответствующий элемент (в порядке обхода документа) будет выбран. # d3.**select**(*node*)

Выбирает указанный узел. Это полезно, если у вас уже есть ссылку на узел, например, `d3.select(this)` внутри обработчика событий, или глобально в `document.body`.

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

Выбирает все элементы, которые соответствуют указанному селектору. Элементы будут выбраны в документе порядке обхода (сверху-вниз). Если ни один из элементов в текущем документе не соответствует указанному селектору, возвращает пустую выборку.

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

Выбирает указанный массив элементов. Это полезно, если у вас уже есть ссылка на узлы, такие как `d3.selectAll (this.childNodes)` внутри обработчика событий, или глобальные, такие как `document.links`. Аргумент узла не должен быть массивом; любой псевдо-массив, который может быть приведен в массив (например, `NodeList` или аргументов) будет работать.

Операции над выборками

D3 добавляет дополнительные методы в массив, так что можно применить операторы к выбранным элементам, такие, как установка атрибута всем выбранным элементам.

Если вы хотите узнать, как выборки работают, попробуйте выбрать элементы интерактивно, используя консоль разработчика браузера. Вы можете проверить возвращенный массив чтобы увидеть, какие были выбраны элементы, и как они группируются. Вы также можете затем применить операторы к выбранным элементам и увидеть, как меняется содержание страницы.

Контент

D3 имеет целый ряд операторов, которые влияют на содержимое документа.

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

Если *value* указано, устанавливает атрибут с указанным именем в заданное значение для всех выбранных элементов. Если *value* является константой, то все элементы получат это значение атрибута; в противном случае, если *value* является функцией, то функция вычисляется для каждого выбранного элемента (по порядку). Возвращаемое значение функции затем используется для установки атрибута каждого элемента. Нулевое значение удалит указанный атрибут.

Если *value* не указано, возвращает значение указанного атрибута для первого ненулевого элемента в выборке. Как правило, это полезно, только если вы знаете, что выборка содержит ровно один элемент.

Указанное *name* может иметь префикс пространства имен, такой как `xLink:href`, указывающее атрибут "href" в пространстве имен xlink. По умолчанию, D3 поддерживает SVG, XHTML, XLink, XML, и Xmlns пространства имен.

name может быть объектом с *name* и *value* аттрибутами.

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

This operator is a convenience routine for setting the "class" attribute; it understands that the "class" attribute is a set of tokens separated by spaces. Under the hood, it will use the [[classList|https://developer.mozilla.org/en/DOM/element.classList]] if available, for convenient adding, removing and toggling of CSS classes.

If *value* is specified, sets whether or not the specified class is associated with the selected elements. If *value* is a constant and truthy, then all elements are assigned the specified class, if not already assigned; if falsey, then the class is removed from all selected elements, if assigned. If *value* is a function, then the function is evaluated for each selected element (in order), being passed the current datum `d` and the current index `i`, with the `this` context as the current DOM element. The function's return value is then used to assign or unassign the specified class on each element.

If you want to set several classes at once, use an object literal like so: `selection.classed({'foo': true, 'bar': false})`.

If *value* is not specified, returns true if and only if the first non-null element in this selection has the specified class. This is generally useful only if you know the selection contains exactly one element.

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

If *value* is specified, sets the CSS style property with the specified name to the specified value on all selected elements. If *value* is a constant, then all elements are given the same style value; otherwise, if *value* is a function, then the function is evaluated for each selected element (in order), being passed the current datum `d` and the current index `i`, with the `this` context as the current DOM element. The function's return value is then used to set each element's style property. A null value will remove the style property. An optional *priority* may also be specified, either as null or the string "important" (without the exclamation point).

If you want to set several style properties at once, use an object literal like so: `selection.style({'stroke': 'black', 'stroke-width': 2})`

If *value* is not specified, returns the current *computed* value of the specified style property for the first non-null element in the selection. This is generally useful only if you know the selection contains exactly one element. Note that the computed value may be *different* than the value that was previously set, particularly if the style property was set using a shorthand property (such as the "font" style, which is shorthand for "font-size", "font-face", etc.).

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

Some HTML elements have special properties that are not addressable using standard attributes or styles. For example, form text fields have a `value` string property, and checkboxes have a `checked` boolean property. You can use the `property` operator to get or set these properties, or any other addressable field on the underlying element, such as `className`.

If *value* is specified, sets the property with the specified name to the specified value on all selected elements. If *value* is a constant, then all elements are given the same property value; otherwise, if *value* is a function, then the function is evaluated for each selected element (in order), being passed the current datum `d` and the current index `i`, with the `this` context as the current DOM element. The function's return value is then used to set each element's property. A null value will delete the specified attribute.

If you want to set several properties at once, use an object literal like so: `selection.properties({'foo': 'bar', 'baz': 'qux'})`.

If *value* is not specified, returns the value of the specified property for the first non-null element in the selection. This is generally useful only if you know the selection contains exactly one element.

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

The `text` operator is based on the [[textContent|http://www.w3.org/TR/DOM-Level-3-Core/core.html#Node3-textContent]] property; setting the text content will replace any existing child elements.

If *value* is specified, sets the text content to the specified value on all selected elements. If *value* is a constant, then all elements are given the same text content; otherwise, if *value* is a function, then the function is evaluated for each selected element (in order), being passed the current datum `d` and the current index `i`, with the `this` context as the current DOM element. The function's return value is then used to set each element's text content. A null value will clear the content.

If *value* is not specified, returns the text content for the first non-null element in the selection. This is generally useful only if you know the selection contains exactly one element.

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

The `html` operator is based on the [[innerHTML|http://www.w3.org/TR/html5/apis-in-html-documents.html#innerhtml]] property; setting the inner HTML content will replace any existing child elements. Also, you may prefer to use the `append` or `insert` operators to create HTML content in a data-driven way; this operator is intended for when you want a little bit of HTML, say for rich formatting.

If *value* is specified, sets the inner HTML content to the specified value on all selected elements. If *value* is a constant, then all elements are given the same inner HTML content; otherwise, if *value* is a function, then the function is evaluated for each selected element (in order), being passed the current datum `d` and the current index `i`, with the `this` context as the current DOM element. The function's return value is then used to set each element's inner HTML content. A null value will clear the content.

If *value* is not specified, returns the inner HTML content for the first non-null element in the selection. This is generally useful only if you know the selection contains exactly one element.

Note: as its name suggests, selection.html is only supported on HTML elements. SVG elements and other non-HTML elements do not support the innerHTML property, and thus are incompatible with selection.html. Consider using XMLSerializer to convert a DOM subtree to text. See also the innersvg polyfill, which provides a shim to support the innerHTML property on SVG elements.

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

Appends a new element with the specified *name* as the last child of each element in the current selection, returning a new selection containing the appended elements. Each new element inherits the data of the current elements, if any, in the same manner as select for subselections.

The *name* may be specified either as a constant string or as a function that returns the DOM element to append. When the *name* is specified as a string, it may have a namespace prefix of the form "namespace:tag". For example, "svg:text" will create a "text" element in the SVG namespace. By default, D3 supports svg, xhtml, xlink, xml and xmlns namespaces. Additional namespaces can be registered by adding to d3.ns.prefix. If no namespace is specified, then the namespace will be inherited from the enclosing element; or, if the name is one of the known prefixes, the corresponding namespace will be used (for example, "svg" implies "svg:svg").

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

Inserts a new element with the specified *name* before the element matching the specified *before* selector, for each element in the current selection, returning a new selection containing the inserted elements. If the before selector does not match any elements, then the new element will be the last child as with append. Each new element inherits the data of the current elements (if any), in the same manner as select for subselections.

The *name* may be specified either as a constant string or as a function that returns the DOM element to append. When the *name* is specified as a string, it may have a namespace prefix of the form "namespace:tag". For example, "svg:text" will create a "text" element in the SVG namespace. By default, D3 supports svg, xhtml, xlink, xml and xmlns namespaces. Additional namespaces can be registered by adding to d3.ns.prefix. If no namespace is specified, then the namespace will be inherited from the enclosing element; or, if the name is one of the known prefixes, the corresponding namespace will be used (for example, "svg" implies "svg:svg").

Likewise, the *before* selector may be specified as a selector string or a function which returns a DOM element. For instance, `insert("div", ":first-child")` will prepend child div nodes to the current selection. For enter selections, the *before* selector may be omitted, in which case entering elements will be inserted immediately before the next following sibling in the update selection, if any. This allows you to insert elements into the DOM in an order consistent with bound data. Note, however, the slower selection.order may still be required if updating elements change order.

selection.**remove**()

Удаляет выбранные элементы из текущего документа. Возвращает текущее выделение (те элементы, которые были удалены), которые сейчас вне DOM. Обратите внимание, что в настоящее время нет специального API для добавления удаленных элементов обратно в документ; однако можно передать функцию `selection.append` или `selection.insert` для повторного добавления элементов.

Data

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

Joins the specified array of data with the current selection. The specified *values* is an array of data values, such as an array of numbers or objects, or a function that returns an array of values. If a *key* function is not specified, then the first datum in the specified array is assigned to the first element in the current selection, the second datum to the second selected element, and so on. When data is assigned to an element, it is stored in the property `__data__`, thus making the data "sticky" so that the data is available on re-selection.

The *values* array specifies the data **for each group** in the selection. Thus, if the selection has multiple groups (such as a d3.selectAll followed by a selection.selectAll), then *data* should be specified as a function that returns an array (assuming that you want different data for each group). For example, you may bind a two-dimensional array to an initial selection, and then bind the contained inner arrays to each subselection. The *values* function in this case is the identity function: it is invoked for each group of child elements, being passed the data bound to the parent element, and returns this array of data.

```
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; });
```

To control how data is joined to elements, a *key* function may be specified. This replaces the default by-index behavior; the key function is invoked once for each element in the new data array, and once again for each existing element in the selection. In both cases the key function is passed the datum `d` and the index `i`. When the key function is evaluated on new data elements, the `this` context is the data array; when the key function is evaluated on the existing selection, the `this` context is the associated DOM element. The key function returns a string which is used to join a datum with its corresponding element, based on the previously-bound data. For example, if each datum has a unique field `name`, the join might be specified as:

```
.data(data, function(d) { return d.name; })
```

For a more detailed example of how the key function affects the data join, see the tutorial [[A Bar Chart, Part 2|http://mbostock.github.com/d3/tutorial/bar-2.html]].

The result of the `data` operator is the *update* selection; this represents the selected DOM elements that were successfully bound to the specified data elements. The *update* selection also contains a reference to the enter and exit selections, for adding and removing nodes in correspondence with data. For example, if the default by-index key is used, and the existing selection contains fewer elements than the specified data, then the *enter* selection will contain placeholders for the new data. On the other hand, if the existing contains more elements than the data, then the *exit* selection will contain the extra elements. And, if the existing selection exactly matches the data, then both the enter and exit selections will be empty, with all nodes in the update selection. For more details, see the short tutorial Thinking With Joins.

If a key function is specified, the `data` operator also affects the index of nodes; this index is passed as the second argument `i` to any operator function values. However, note that existing DOM elements are not automatically reordered; use sort or order as needed.

If *values* is not specified, then this method returns the array of data for the first group in the selection. The length of the returned array will match the length of the first group, and the index of each datum in the returned array will match the corresponding index in the selection. If some of the elements in the selection are null, or if they have no associated data, then the corresponding element in the array will be undefined.

Note: the `data` method cannot be used to clear previously-bound data; use selection.datum instead.

selection.**enter()**

Returns the entering selection: placeholder nodes for each data element for which no corresponding existing DOM element was found in the current selection. This method is only defined on a selection returned by the data operator. In addition, the entering selection only defines append, insert, select and call operators; you must use these operators to instantiate the entering nodes before modifying any content. (Enter selections also support empty to check if they are empty.) Note that the *enter* operator merely returns a reference to the entering selection, and it is up to you to add the new nodes.

As a simple example, consider the case where the existing selection is empty, and we wish to create new nodes to match our data:

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

Assuming that the body is initially empty, the above code will create six new DIV elements, append them to the body in order, and assign their text content as the associated (string-coerced) number:

```
<div>4</div>
<div>8</div>
```

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

Another way to think about the entering placeholder nodes is that they are pointers to the parent node (in this example, the document body); however, they only support append and insert.

The enter selection **merges into the update selection** when you append or insert. This approach reduces code duplication between enter and update. Rather than applying operators to both the enter and update selection separately, you can now apply them to the update selection after entering the nodes. In the rare case that you want to run operators only on the updating nodes, you can run them on the update selection before entering new nodes.

selection.**exit()**

Returns the exiting selection: existing DOM elements in the current selection for which no new data element was found. This method is only defined on a selection returned by the data operator. The exiting selection defines all the normal operators, though typically the main one you'll want to use is remove; the other operators exist primarily so you can define an exiting transition as desired. Note that the *exit* operator merely returns a reference to the exiting selection, and it is up to you to remove the new nodes.

As a simple example, consider updating the six DIV elements created in the above example for the enter operator. Here we bind those elements to a new array of data with some new and some old:

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

Now `div` —the result of the data operator—refers to the updating selection. Since we specified a key function using the identity function, and the new data array contains the numbers [4, 8, 16] which also exist in the old data array, this updating selection contains three DIV elements. Let's say we leave those elements as-is. We can instantiate and add the new elements [1, 2, 32] using the entering selection:

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

Likewise, we can remove the exiting elements [15, 23, 42]:

```
div.exit().remove();
```

Now the document body looks like this:

```
<div>4</div>
<div>8</div>
<div>16</div>
<div>1</div>
<div>2</div>
<div>32</div>
```

Note that the DOM elements are now out-of-order. However, the selection index `i` (the second argument to operator functions), will correctly match the new data array. For example, we could assign an index attribute:

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

This would result in:

```
<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>
```

If you want the document traversal order to match the selection data order, you can use sort or order.

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

Filters the selection, returning a new selection that contains only the elements for which the specified *selector* is true. The *selector* may be specified either as a function or as a selector string, such as ".foo". As with other operators, the function is passed the current datum `d` and index `i`, with the `this` context as the current DOM element. Like the built-in array [[filter|https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Array/Filter]] method, the returned selection *does not* preserve the index of the original selection; it returns a copy with elements removed. If you want to preserve the index, use select instead. For example, to select every odd element (relative to the zero-based index):

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

Equivalently, using a filter function:

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

Or a filter selector (noting that the :nth-child pseudo-class is a one-based index rather than a zero-based index):

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

Thus, you can use either select or filter to apply operators to a subset of elements.

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

Gets or sets the bound data for each selected element. Unlike the selection.data method, this method does not compute a join (and thus does not compute enter and exit selections). This method is implemented on top of selection.property:

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

If *value* is specified, sets the element's bound data to the specified value on all selected elements. If *value* is a constant, all elements are given the same data; otherwise, if *value* is a function, then the function is evaluated for each selected element, being passed the previous datum `d` and the current index `i`, with the `this` context as the current DOM element. The function is then used to set each element's data. A null value will delete the bound data. This operator has no effect on the index.

If *value* is not specified, returns the bound datum for the first non-null element in the selection. This is generally useful only if you know the selection contains exactly one element.

Note: this method was previously called "map". The old name is deprecated.

The `datum` method is useful for accessing HTML5 custom data attributes with D3. For example, given the following elements:

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

You can expose the custom data attributes to D3 by setting each element's data as the built-in dataset property:

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

This can then be used, for example, to sort elements by username.

selection.**sort**(*comparator*)

Sorts the elements in the current selection according to the specified comparator function. The comparator function is passed two data elements *a* and *b* to compare, returning either a negative, positive, or zero value. If negative, then *a* should be before *b*; if positive, then *a* should be after *b*; otherwise, *a* and *b* are considered equal and the order is arbitrary. Note that the sort is not guaranteed to be stable; however, it is guaranteed to have the same behavior as your browser's built-in [[sort|https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Array/sort]] method on arrays.

selection.**order**()

Re-inserts elements into the document such that the document order matches the selection order. This is equivalent to calling sort() if the data is already sorted, but much faster.

Animation & Interaction

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

Adds or removes an event *listener* to each element in the current selection, for the specified *type*. The *type* is a string event type name, such as "click", "mouseover", or "submit". The specified *listener* is invoked in the same manner as other operator functions, being passed the current datum `d` and index `i`, with the `this` context as the current DOM element. To access the current event within a listener, use the global d3.event. The return value of the event listener is ignored.

If an event listener was already registered for the same type on the selected element, the existing listener is removed before the new listener is added. To register multiple listeners for the same event type, the type may be followed by an optional namespace, such as "click.foo" and "click.bar".

To remove a listener, pass null as the *listener*. To remove all listeners for a particular event type, pass null as the *listener*, and `.type` as the *type*, e.g. `selection.on(".foo", null)`.

An optional *capture* flag may be specified, which corresponds to the W3C useCapture flag: "After initiating capture, all events of the specified type will be dispatched to the registered EventListener before being dispatched to any EventTargets beneath them in the tree. Events which are bubbling upward through the tree will not trigger an EventListener designated to use capture."

If *listener* is not specified, returns the currently-assigned listener for the specified *type*, if any.

d3.**event**

Stores the current event, if any. This global is during an event listener callback registered with the on operator. The current event is reset after the listener is notified in a finally block. This allows the listener function to have the same form as other operator functions, being passed the current datum `d` and index `i`.

The `d3.event` object is a [[DOM event|https://developer.mozilla.org/en-US/docs/DOM/event]] and implements the standard event fields like `timeStamp` and `keyCode` as well as methods like `preventDefault()` and `stopPropagation()`. While you can use the native event's [[pageX|https://developer.mozilla.org/en/DOM/event.pageX]] and [[pageY|https://developer.mozilla.org/en/DOM/event.pageY]], it is often more convenient to transform the event position to the local coordinate system of the container that received the event. For example, if you embed an SVG in the normal flow of your page, you may want the event position relative to the top-left corner of the SVG image. If your SVG contains transforms, you might also want to know the position of the event relative to those transforms. Use the d3.mouse operator for the standard mouse pointer, and use d3.touches for multitouch events on iOS.

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

Returns the *x* and *y* coordinates of the current d3.event, relative to the specified *container*. The container may be an HTML or SVG container element, such as an [[svg:g|http://www.w3.org/TR/SVG/struct.html#Groups]] or [[svg:svg|http://www.w3.org/TR/SVG/struct.html#SVGElement]]. The coordinates are returned as a two-element array [*x*, *y*].

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

Returns the *x* and *y* coordinates of the touch with the specified identifier associated with the current d3.event, relative to the specified *container*. If *touches* is not specified, defaults to the current event's [[changedTouches|http://developer.apple.com/library/safari/documentation/UserExperience/Reference/TouchEventClassReference/TouchEvent/TouchEvent.html#//apple_ref/java
The container may be an HTML or SVG container element, such as an svg:g or svg:svg. The coordinates are returned as an array of two-element arrays [[*x1*, *y1*], [*x2*, *y2*], …]. If there is no touch with the specified identifier in *touches*, returns null; this can be useful for ignoring touchmove events where the only some touches have moved.

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

Returns the *x* and *y* coordinates of each touch associated with the current d3.event, based on the [[touches|http://developer.apple.com/library/safari/documentation/UserExperience/Reference/TouchEventClassReference/TouchEvent/TouchEvent.html#//apple_ref/javascript/in
attribute, relative to the specified *container*. The container may be an HTML or SVG container element, such as an svg:g or svg:svg. The coordinates are returned as an array of two-element arrays [[*x1*, *y1*], [*x2*, *y2*], …]. If *touches* is specified, returns the positions of the specified touches; if *touches* is not specified, it defaults to the `touches` property on the current event.

selection.**transition**()

Starts a [[transition|Transitions]] for the current selection. Transitions behave much like selections, except operators animate smoothly over time rather than applying instantaneously.

selection.**interrupt**()

Immediately interrupts the current transition, if any. Does not cancel any scheduled transitions that have not yet started. To cancel scheduled transitions as well, simply create a new zero-delay transition after interrupting the current transition:

```
selection
    .interrupt() // cancel the current transition
    .transition(); // preempt any scheduled transitions
```

Subselections

Whereas the top-level select methods query the entire document, a selection's select and selectAll operators restrict queries to descendants of each selected element; we call this "subselection". For example, `d3.selectAll("p").select("b")` returns the first bold ("b") elements in every paragraph ("p") element. Subselecting via selectAll groups elements by ancestor. Thus, `d3.selectAll("p").selectAll("b")` groups by paragraph, while `d3.selectAll("p b")` returns a flat selection. Subselecting via select is similar, but preserves groups and propagates data. Grouping plays an important role in the data join, and functional operators may depend on the numeric index of the current element within its group.

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

For each element in the current selection, selects the first descendant element that matches the specified *selector* string. If no element matches the specified selector for the current element, the element at the current index will be null in the returned selection; operators (with the exception of data) automatically skip null elements, thereby preserving the index of the existing selection. If the current element has associated data, this data is inherited by the returned subselection, and automatically bound to the newly selected elements. If multiple elements match the selector, only the first matching element in document traversal order will be selected.

The *selector* may also be specified as a function that returns an element, or null if there is no matching element. In this case, the specified *selector* is invoked in the same manner as other operator functions, being passed the current datum `d` and index `i`, with the `this` context as the current DOM element.

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

For each element in the current selection, selects descendant elements that match the specified *selector* string. The returned selection is grouped by the ancestor node in the current selection. If no element matches the specified selector for the current element, the group at the current index will be empty in the returned selection. The subselection does not inherit data from the current selection; however, if the data value is specified as a function, this function will be called with the data `d` of the ancestor node and the group index `i` to determine the data bindings for the subselection.

Grouping by selectAll also affects subsequent entering placeholder nodes. Thus, to specify the parent node when appending entering nodes, use select followed by selectAll:

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

You can see the parent node of each group by inspecting the `parentNode` property of each group array, such as `selection[0].parentNode`.

The *selector* may also be specified as a function that returns an array of elements (or a NodeList), or the empty array if there are no matching elements. In this case, the specified *selector* is invoked in the same manner as other operator functions, being passed the current datum `d` and index `i`, with the `this` context as the current DOM element.

Control

For advanced usage, D3 has a few additional operators for custom control flow.

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

Invokes the specified *function* for each element in the current selection, passing in the current datum `d` and index `i`, with the `this` context of the current DOM element. This operator is used internally by nearly every other operator, and can be used to invoke arbitrary code for each selected element. The each operator can be used to process selections recursively, by using `d3.select(this)` within the callback function.

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

Invokes the specified *function* once, passing in the current selection along with any optional *arguments*. The call operator always returns the current selection, regardless of the return value of the specified function. The call operator is identical to invoking a function by hand; but it makes it easier to use method chaining. For example, say we want to set a number of attributes the same way in a number of different places. So we take the code and wrap it in a reusable function:

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

Now, we can say this:

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

Or equivalently:

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

The `this` context of the called function is also the current selection. This is slightly redundant with the first argument, which we might fix in the future.

selection.**empty**()

Returns true if the current selection is empty; a selection is empty if it contains no elements or only null elements.

selection.**node**()

Returns the first non-null element in the current selection. If the selection is empty, returns null.

selection.**size**()

Returns the total number of elements in the current selection.

Extension

d3.**selection**()

Returns the root selection, equivalent to `d3.select(document.documentElement)`. This function can also be used to check if an object is a selection: `o instanceof d3.selection`. You can also add new methods to the selection prototype. For example, to add a convenience method for setting the "checked" property of checkboxes, you might say:

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