April 8, 2012 / Mike Bostock

Nested Selections

D3's selections can be hierarchical, much like the elements and data they join. Consider a table:

```
<table>
  <thead>
    <tr><td>  A</td><td>  B</td><td>  C</td><td>  D</td></tr>
  </thead>
  <tbody>
    <tr><td>  0</td><td>  1</td><td>  2</td><td>  3</td></tr>
    <tr><td>  4</td><td>  5</td><td>  6</td><td>  7</td></tr>
    <tr><td>  8</td><td>  9</td><td> 10</td><td> 11</td></tr>
    <tr><td> 12</td><td> 13</td><td> 14</td><td> 15</td></tr>
  </tbody>
</table>
```

How would you select only the body cells? The selector `"td"` would match the `td` elements in the head as well as the body. To match only those elements *A* within some other elements *B*, use the descendant combinator, `"B A"`. For example:

```
var td = d3.selectAll("tbody td");
```

There are other combinators. For example, a comma results in a union: `"th, td"` matches both `th` *and* `td` elements.

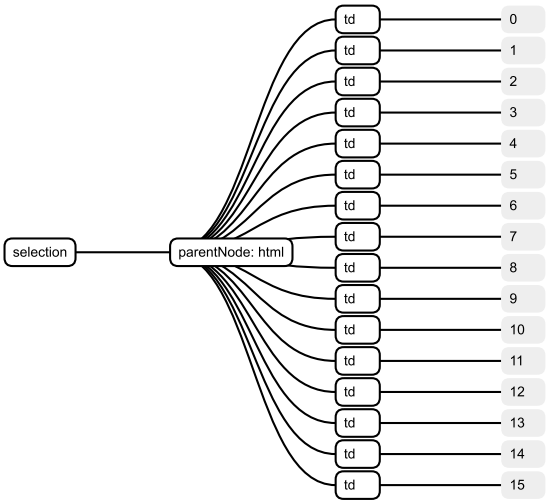
Alternatively, select the `tbody` element first, then select the `td` elements within:

```
var td = d3.select("tbody").selectAll("td");
```

This produces the same result because selectAll, for each element in the current selection, selects the matching descendants. This is convenient if you want to derive multiple selections from the same parent, such as splitting the even and odd rows of a table.

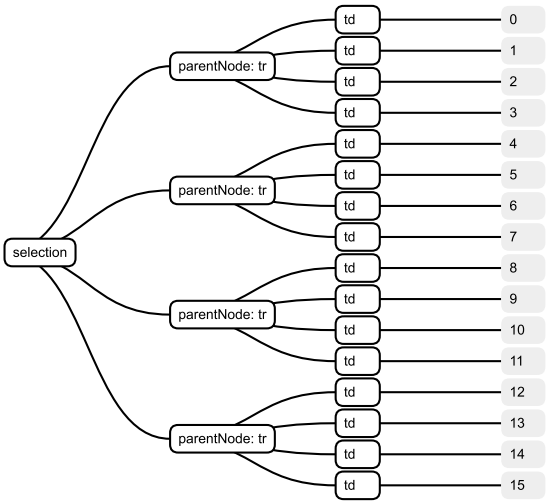
Nesting and Index

If you select the `td` elements using d3.selectAll, you get a flat selection, like so:




```
var td = d3.selectAll("tbody td");
```

Flat selections lack hierarchical structure: the table cells are merged into a single array, rather than grouped by parent row. This makes them more difficult to manipulate on a row- or column-basis. In contrast, D3's nested selections retain the hierarchy. To group by row, first select the `tr` elements, then select the `td` elements:



The concept of nested selections is one of the main differences between D3 and other DOM libraries, such as jQuery and Prototype.

```
var td = d3.selectAll("tbody tr").selectAll("td");
```

Now if you want to color the first column red, use the index `i`:

```
td.style("color", function(d, i) { return i ? null : "red"; });
```

This ternary expression returns null if `i` is a truthy value (non-zero), and "red" if is not. By returning null for the other columns, the color is inherited from the current stylesheet.

You can also access the row index (`j`) by adding a third argument to the function.

Nesting and Data

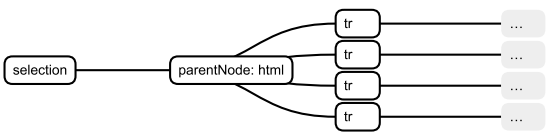
Hierarchical elements are often driven by similarly hierarchical data; nested selections are therefore convenient for binding data, too. To continue the example, you might represent the table's associated data as a matrix (an array of arrays):

```
var matrix = [
  [ 0,  1,  2,  3],
  [ 4,  5,  6,  7],
  [ 8,  9, 10, 11],
  [12, 13, 14, 15],
];
```

To join the numbers to the corresponding table cells, first join the outer array (`matrix`) to the rows, and then join the inner arrays (`matrix[0]`, `matrix[1]`, …) to the cells:

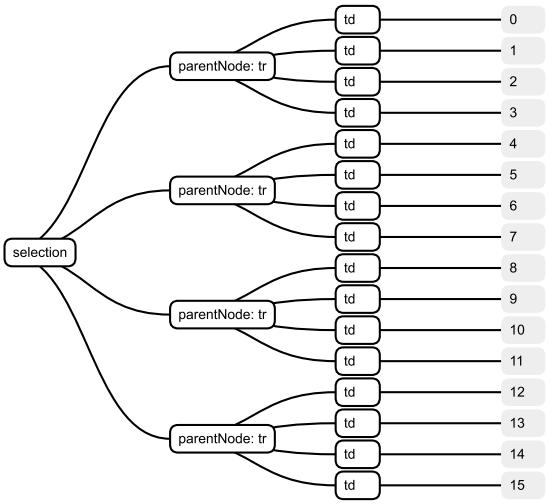
```
var td = d3.selectAll("tbody tr")
    .data(matrix)
  .selectAll("td")
    .data(function(d, i) { return d; }); // d is matrix[i]
```

Notice that the data method can either take an *array* (such as `matrix`) or a *function* that returns an array. Arrays are often used with flat selections, since flat selections only have one group, while nested selections typically require a function. The initial row selection is flat, since it was created with d3.selectAll:




```
var tr = d3.selectAll("tbody tr")
    .data(matrix);
```

The cell selection, in contrast, is nested:



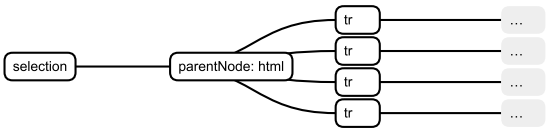

```
var td = tr.selectAll("td")
    .data(function(d) { return d; }); // matrix[i]
```

The data operator defines the array of data for each group. Here the data function is invoked once per row and successively passed each parent datum. Since the parent data is an array of arrays, the data function simply returns the inner array for each row of cells.

Nesting and the Parent Node

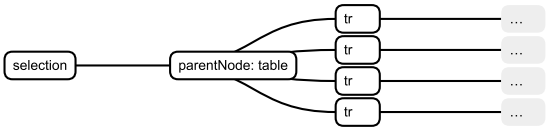
Nesting selections has another subtle yet critical side-effect: it sets the *parent node* for each group. The parent node is a hidden property on selections that determines where to append entering elements. For example, if you attempt a data-join on a top-level selection, you get an error:

See Thinking with Joins for a quick overview of D3's data-join concept.




```
d3.selectAll("table tr")
    .data(matrix)
  .enter().append("tr"); // error!
```

The error occurs because the default parent node is the document's root (`html`) element, and you can't add `tr` elements directly to the root. Instead, select a parent before performing the data join:




```
d3.select("table").selectAll("tr")
    .data(matrix)
  .enter().append("tr"); // success
```

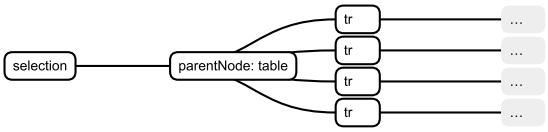
This approach extends to arbitrary levels of nested selection. Say you wanted to create the table from scratch, given the matrix of numbers. Start by selecting the body:


```
var body = d3.select("body");
```

Next append a table element to the body:

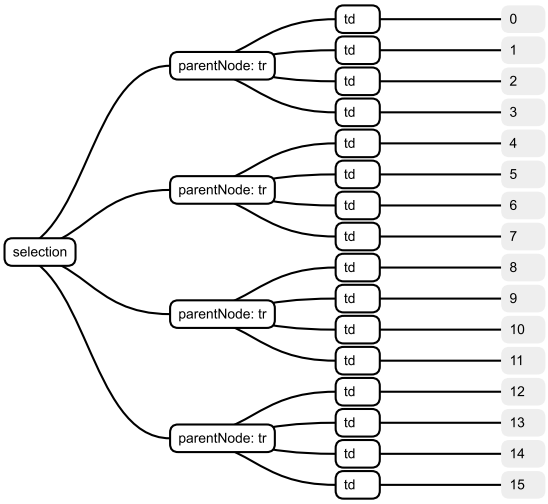

```
var table = body.append("table");
```

Now append entering rows to the table via data-join. Since selectAll is called on the selected table element, it establishes a new parent node:




```
var tr = table.selectAll("tr")
    .data(matrix)
  .enter().append("tr");
```

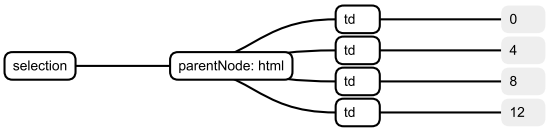
Finally, append entering cells to each row:




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

To Nest, or not to Nest

There is an important difference between select and selectAll: select *preserves* the existing grouping, whereas selectAll *creates* a new grouping. Calling select thus preserves the data, index and even the parent node of the original selection! For example, the following selection is flat, with the parent node still the document root:




```
var td = d3.selectAll("tbody tr").select("td");
```

The only way to obtain a nested selection, then, is to call selectAll on an existing selection; this is why a data-join typically follows a selectAll rather than select.

This tutorial employed tables as an example of hierarchical structure. This is but one contrived example—nested selections are surprisingly common! By making only a few code changes, nested selections can quickly turn any singular visualization into small multiples. Much like thinking with

joins, nested selections require a different mental model for creating and manipulating elements. But once mastered, they provide a concise way of creating data-driven documents.

April 8, 2012 / Mike Bostock