Static Visualizations (II)

Riccardo Pucella

October 5, 2015

Last time

- Basic visualization architecture
  - useful to keep in mind
- Simple example:
  - going from data to charts (views)
- First steps in D3

Today:

- D3 selections and data binding
- View layout

D3 selections

D3 enables DOM manipulations via element selection:

```
d3.select("#somerect")
  .attr("width",100);
```

Selecting multiple elements on the page:

```
d3.selectAll("rect");
```

Selecting multiple children of an element:

```
var elt = d3.select("#some_element");
elt.selectAll("rect");
```

D3 selections

D3 enables DOM manipulations via element selection:

```
d3.select(“#somerect”)
    .attr(“width”,100);
```

Selecting multiple elements on the page:

```
d3.selectAll(“rect”);
```

Selecting multiple children of an element:

```
var elt = d3.select(“#some_element”);
elt.selectAll(“rect”);
```

D3 selections

D3 enables DOM manipulations via element selection:

```
d3.select(“#somerect”)
  .attr(“width”,100);
```

Selecting multiple elements on the page:

```
d3.selectAll(“rect”);
```

Selecting multiple children of an element:

```
var elt = d3.select(“#some_element”);
elt.selectAll(“rect”);
```

Selections: updating

Operations distribute to all elements in a selection

```
d3.selectAll(“rect”)
    .attr(“width”,100)
    .style(“fill”,”violet”);
```

Selections: appending

Operations distribute to all elements in a selection, *including appending a new element*

```
d3.selectAll(“g”)
    .append(“rect”)
    .attr(“x”,10)
    .attr(“y”,20)
    .attr(“height”,100)
    .attr(“width”,100)
    .style(“fill”,”violet”);
```

Selections: appending

Operations distribute to all elements in a selection, *including appending a new element*

```
d3.selectAll(“g”)
    .append(“rect”)
    .attr(“x”,10)
    .attr(“y”,20)
    .attr(“height”,100)
    .attr(“width”,100)
    .style(“fill”,”violet”)
```

Returns a new selection with the added elements

Selections: appending

Operations distribute to all elements in a selection, *including appending a new element*

```
d3.selectAll(“g”)
    .append(“rect”)
    .attr(“x”,10)
    .attr(“y”,20)
    .attr(“height”,100)
    .attr(“width”,100)
    .style(“fill”,”viole
```

These attribute updates apply to the new selection

Selections: transitioning

Updating elements in a selection can be spread out over time

```
d3.selectAll(“rect”)
    .transition()
    .duration(3000)
    .attr(“width”,100)
    .style(“fill”,”violet”);
```

Selections: transitioning

Updating elements in a selection can be spread out over time

```
d3.selectAll("rect")
    .transition()
    .duration(3000)
    .attr("width",100)
    .style("fill","violet'
```

Returns a new *transition* selection

Selections: transitioning

Updating elements in a selection can be spread out over time

```
d3.selectAll(“rect”)
    .transition()
    .duration(3000)
    .attr(“width”,100)
    .style(“fill”,”violet
```

Set that transition to take 3000ms

Selections: transitioning

Updating elements in a selection can be spread out over time

```
d3.selectAll(“rect”)
     .transition()
     .duration(3000)
     .attr(“width”,100)
     .style(“fill”,”violet”
```

These attribute updates are (continuously?) transitioned over 3000ms

Selections: non-constant updating

Updating elements in a selection forces each element to be updated to the same value

```
d3.selectAll(“rect”)
  .style(“fill”,”violet”);
```

How do we update different elements differently? Associate data with the selection:

```
d3.selectAll(“rect”)
 .data([“red”,”green”,”blue”])
 .style(“fill”,function(d) { return d;});
```

Selections: non-constant updating

Updating eleme
element to be u

```
d3.selectAll(
   .style("fil
```

Basically injects the data in the elements so that attribute updates can look them up

How do we upd
differently? Associate with the selection:

```
d3.selectAll("rect")
  .data(["red","green","blue"])
  .style("fill",function(d) { return d;});
```

Selections: extra data points

What happens when you have more data than elements in the selection?

- extra data points are ignored
- but we can do something with them!

```
d3.selectAll(“rect”)
  .data([“red”,”green”,”blue”])
  .enter().append(“rect”)
  // … create rect …
d3.selectAll(“rect”)
  .style(“fill”,function(d) { return d;});
```

Sele

Wha
than

- ex
- bu

Returns a new *enter* selection corresponding to the extra data points and to which you can append elements

```
d3.selectAll(“rect”)
  .data([“red”,”green”,”blue”])
  .enter().append(“rect”)
  // … create rect …
d3.selectAll(“rect”)
  .style(“fill”,function(d) { return d;});
```

Example

Adapting the Social Media 2014 example from last week.

Laying out views

In a step towards interactive views, let’s see how to put multiple views on a page.

Really a special case of HTML layouts.

- libraries
- tables
- CSS layout
- by-hand layout in SVGs

Approach 1: libraries

Lots of libraries out there to help you layout your web page.

- e.g., Bootstrap

Key aspect: handle responsive design

- adapt layout to the screen format
- mobile vs tablet vs desktop

If you need to do anything professional, that’s the way to go.

Approach 2: tables

Historical workhorse of page layout.

Great if you need a regular grid.

We can put an SVG in each cell.

Really painful to work with.

Approach 3: CSS layout

CSS now the way to go to do layout in HTML, moving away from tables.

- Use `<div>` to describe groups
- Use `float` style, `position` style, CSS3 `flexbox`, etc to lay out elements

Require some understanding of how browsers flow HTML.

- more flexible than tables.

Approach 4: Use a large SVG

If views are SVG-based, can construct those views within a larger SVG element, by hand.

```
function makeView (svg,x,y,width,height) {
  // build view in svg at (x,y)
  // of size (width,height)
}
```