Lab 6: D3 Enter, Update & Exit (Pre Lab)

Wijaya, Mario edited this page 9 days ago · 2 revisions

Learning Objectives

After completing this lab you will be able to:

- Append new DOM elements with enter
- Update currently data-joined elements
- Remove old DOM elements with exit
- Write one function to update a chart

Prerequisites

- Update your *starter code repository* using one of the following methods:
  i. From the GitHub Desktop app click `Sync` on the top right
  ii. Open a command line prompt. Navigate to the repository directory, for example `cd ~\Development\CS4460-Spring2018\Labs` and run command `git pull`.
- You have **read Chapter 9** in D3 - Interactive Data Visualization for the Web by Scott Murray

Additional Reading

- Enter, Update, Exit by Christian Behrens
- Three Little Circles by Mike Bostock
- Understanding selectAll, data, enter, append sequence in D3.js
- Thinking with Joins by Mike Bostock
- Object Constancy by Mike Bostock
- General Update Pattern I by Mike Bostock
- General Update Pattern II by Mike Bostock
- Advanced D3: More on selections and data, scales, axis by A. Lex of U. of Utah

Refer to the files in `CS4460-Spring2018/06_lab/example` while reading through the pre-lab

D3 Enter, Update, Exit Pattern

By now you have learned how to load external data and how to map it to visual elements like a bar chart, scatter plot, and line chart. But very often you have to deal with a continuous data stream rather than a static CSV file. Dynamic data often requires more sophisticated user interfaces that allow users to interact with the data (e.g. filter, sort, navigate).

Instead of removing and redrawing visualizations each time new data arrives, update only affected components and focus on loading times and smooth transitions.

We will accomplish this by using the D3 update pattern (enter → update → exit).

"Updating data" means "joining data"

A data-join is followed by operations on the three virtual selections: enter, update and exit.

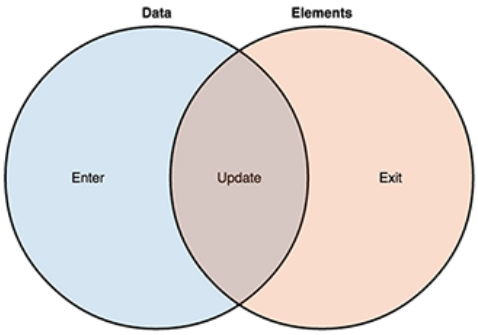
This means that we are merging new data with existing elements. In the merging process we have to consider:

- **Enter** - What happens to new data values without existing, associated DOM elements

Clone this wiki locally

`https://github.gatech.edu`

- **Update** - What happens to existing elements which have changed
- **Exit** - What happens to existing DOM elements which are not associated with data anymore



To take care of the update pattern you have to change the sequence of your D3 code a little bit. Instead of chaining everything together, some code snippets must be separated.

An Example

This week we will be working with letters. We'll be learning the ABCs of D3's update pattern with... wouldn't you know it the ABCs. We are going to create grouped circles with text for each letter.

First we will bind the data to .letter `<g>` elements: var svg = d3.select('svg'); var letter = svg.selectAll('.letter') .data(['A', 'B', 'C']);

The length of the dataset is 3 and we select all SVG elements with a classname of `letter` in the SVG. Remember from the last lab that `d3.select('svg')` is needed to specify the `parent` for this selection. That means, if there are 3 or more existing groups, the **enter selection** is empty, otherwise it contains placeholders for the missing elements.

Enter

The page is empty because we have not appended any groups yet. We can access the enter selection and append a new group for each placeholder with the following statement:

```
var letterEnter = letter.enter()
    .append('g')
    .attr('class', 'letter')
    .attr('transform', function(d,i) {
        return 'translate('+[i * 30 + 50, 50]+')';
    });
```

This will create spaced out `<g>` elements with the classname `letter` , but now we need to add the circles and text. This can be achieved by appending to `letterEnter` . Feel free to play with the position by changing the value for translate of [x,y] to different values.

```
letterEnter.append('circle')
    .attr('r', 10)
    .attr('fill', '#A9A9A9');

letterEnter.append('text')
    .attr('x', -5)
    .attr('y', 30)
    .attr('fill', '#A9A9A9');
    .text(function(d) {
        return d;
    });
```

(You might have noticed that we've actually already used this pattern multiple times in previous labs)

Update

Now with data-bound elements in the SVG canvas we can use the `update` selection to change them. In our drawing function we call:

```
var letterCircle = d3.select('svg').selectAll('.letter circle');
```

Which returns a selection of 3 circles - `.letter circle` is a css selector for all elements with a classname of `letter` and then selects any and all `circle` elements inside of the `.letter` element.

We can use this selection to update the already drawn circles:

```
letterCircle.attr('r', 15);
```

Exit

Often you want to remove elements from the SVG. If someone filters the dataset you may want to remove existing elements. In this case, you have to use the exit selection. `exit` contains the leftover elements for which there is no corresponding data anymore.

We call the drawing function again with new data:

```
var letter = svg.selectAll('.letter')
        .data(['A', 'B']);
```

All that's left to do, then, is to remove the exiting elements:

```
letter.exit().remove();
```

And now we have two letters:

Now if we want to add more data, we can put all use all three selections `enter`, `update`, `exit` in the same drawing function. This will update our circles based on any new data. Note that the `letterEnter.merge(letter)` creates the **update + enter selection** This is useful for changing styles and attributes on *new and old elements* at the same time.

```
function updateCircles(letters) {
    var letter = svg.selectAll('.letter')
        .data(letters);

    var letterEnter = letter.enter()
        .append('g')
        .attr('class', 'letter');

    letterEnter.merge(letter)
        .attr('transform', function(d,i) {
            return 'translate('+[i * 30 + 50, 50]+')';
        });

    letterEnter.append('circle')
        .attr('r', 15)
        .attr('fill', '#A9A9A9');

    letterEnter.append('text')
        .attr('x', -5)
        .attr('y', 30)
        .attr('fill', '#A9A9A9')
        .text(function(d) {
            return d;
        });

    letter.exit().remove();
}

updateCircles(['A', 'B', 'C']);
updateCircles(['A', 'B']);
updateCircles(['A', 'B', 'C', 'D', 'E', 'F']);
```

Putting everything together, consider the three possible outcomes that result from joining data to elements:

1. `enter` - incoming elements, entering the stage.
2. `update` - persistent elements, staying on stage.
3. `exit` - outgoing elements, exiting the stage.

By default, the data join happens by index: the first element is bound to the first datum, and so on. Thus, either the enter or exit selection will be empty, or both. If there are more data than elements, the extra data are in the enter selection. And if there are fewer data than elements, the extra elements are in the exit selection.

Key Function

Now that you understand update, enter, and exit selections, it's time to dig deeper into data joins.

The default join is by index order, meaning the first data value is bound to the first DOM element in the selection, the second value is bound to the second element, and so on.

You can control precisely which datum is bound to which element by specifying a key function in the `selection.data()` method. For example, by using the identity function `function(d){ return d;}` , you can rebind the circles to new data while ensuring that existing circles are rebound to the same value in the new data, if any.

Take our letter circles example. Say we already have 3 circles on the canvas for `['A', 'B', 'C']`. And we want to update the diagram with the letters `['B', 'C']`. If we use the default index order then the following items will be removed with the `exit` selection:

The index will be used again as the default key to match the new data to the actual circles. There are three circles on the webpage and two items in the new dataset. Therefore, the last circle will be removed and the other two circles will be bound to the new data.

This is the simplest method of joining data and often sufficient. However, when the data and the elements are not in the same order, joining by index is insufficient. In this case, you can specify a key function as the second argument (callback function).

This key function takes a data point as input and returns a corresponding key: a string, such as a name, that uniquely identifies the data point. The objects stay constantly bound to their original data because of this key function:

Here's an example of a key function for our letters:

```
var letter = svg.selectAll('.letter')
    .data(['A', 'B'], function(d){
        return d;
    });
```

The function is the second input for the `.data()` method. And as you might have guessed it is optional. So when should you use they key function?

Key functions are needed when using transitions. They can also be useful for improving performance independent of transitions.

Use a key function whenever you want to follow graphical elements through animation and interaction: **filtering (adding or removing elements)**, reordering (sorting), switching dimensions within multivariate data, etc. If you forget to specify a key function, the default join-by-index can be misleading! Assist your viewers by maintaining object constancy.

In the Lab 6 activity, you will need to use the key function to retain object constancy in the bar chart.

Interaction via Event Listeners

In Lab 6 Activity we are using a `change` event from the `select` element to update our chart. An event acts as a “trigger,” something that happens after page load to apply updates to our chart.

In JavaScript, events are happening all the time. Not exciting events, like huge parties, but really insignificant events like mouseover and click. Most of the time, these insignificant events go ignored. But if someone is listening, then the event will be heard, and can go down in posterity, or at least trigger some sort of DOM interaction.

An event listener is an anonymous function that listens for a specific event on a specific element or elements. In Lab 6 Activity, the `select` element has an attribute for an `onchange` listener. The listener listens for a change event. When that happens, the listener function is executed. You can put whatever code you want in between the brackets of the anonymous function (**Note: The code shown below are part of the Lab 6 activity code and will be posted in Lab 6 activity folder Wednesday, 3/21/18**):

```
<select class="custom-select" id="categorySelect" onchange="onCategoryChanged()">
      <option selected value="all-letters">All Letters</option>
      <option value="only-consonants">Only Consonants</option>
      <option value="only-vowels">Only Vowels</option>
</select>
```

When a user selects a new value, `onCategoryChanged` is executed in `main.js`. From there we can access the newly set value with:

```
// Global function called when select element is changed
function onCategoryChanged() {
    var select = d3.select('#categorySelect').node();
    // Get current value of select element
    var category = select.options[select.selectedIndex].value;
    // Update chart with the selected category of letters
    updateChart(category);
}
```

We'll cover more on interactivity in the following Labs.

Using other frameworks such as Bootstrap

A quick note on the Bootstrap framework that we are using to style the `select` input element in Lab 6 Activity (the files will be posted on Wednesday, 3/21/18).

Rather than coding from scratch, frameworks enable you to utilize ready made blocks of code to help you get started. They give you a solid foundation for what a typical web project requires and usually they are also flexible enough for customization.

We have chosen Bootstrap as an example open source HTML, JS and CSS framework. It is one of the most widely used frameworks, it is easy to understand and it provides a great documentation with many examples. The question whether a framework can be useful depends on the individual project and on the developer. Therefore, it is up to you to decide if you want to use it in your programming assignments.

Here is a summary of the main aspects of Bootstrap:

- Open source HTML, CSS, and JS framework
- Provides a base styling for common used HTML elements
- The grid system helps you to create multi-column and nested layouts, especially if your website should work on different devices
- Extensive list of pre-styled components (navigation, dropdown-menu, forms, tables, icons ...)
- Customizable: All CSS rules can be overridden by your own rules
- Compatible with the latest versions of all major browsers

In general, frameworks are very helpful when you need standard widgets or web page elements.

Home