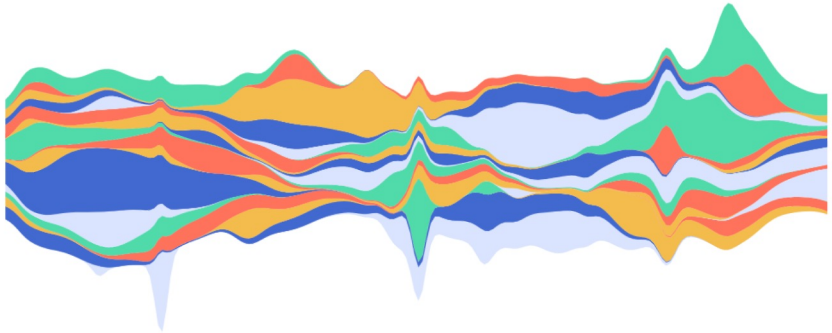
by Observable

Introduction to D3

Session 4

Observable notebook

YouTube video



These slides are provided with the course “Introduction to D3” by Paul Buffa – Head of Product Education at Observable

Agenda

1. The General Update Pattern
2. Enter, Exit, Update with Join
3. Transitions

General Update Pattern

The General Update Pattern

The general update pattern refers to a D3 technique for handling entering, updating and removing elements based on your data.

The General Update Pattern

The general update pattern refers to a D3 technique for handling entering, updating and removing elements based on your data. *It's technically a deprecated pattern, with selection.join() being the recommended approach.*

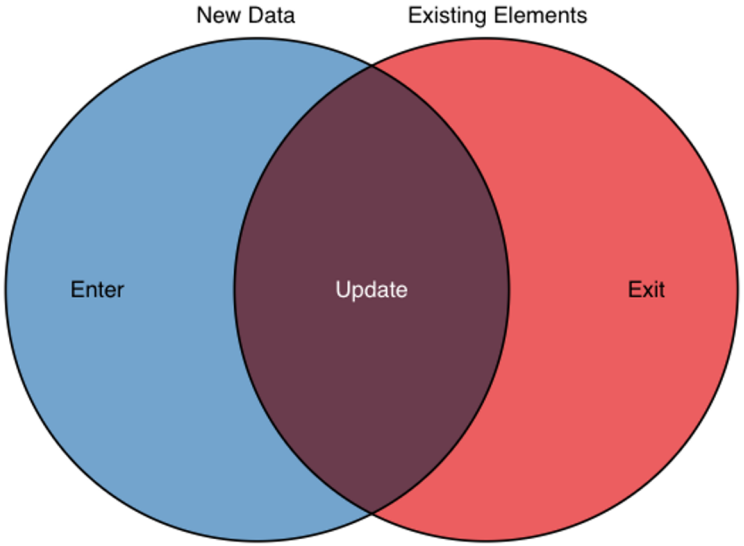
The General Update Pattern

We’re covering the general update pattern because:

- It’s useful for reading D3 examples that still follow this paradigm.
- It’s helpful for understanding how selection.join() works and what it makes more concise.

The data join

In session 2, we talked about the data join, and handling for each state. The general update pattern is an explicit way of doing this.



The three selections computed by the *selection*.data.

General Update Pattern

Here is code for a “visualization” showing the general update pattern. Let’s break it down.

```
{
  const svg = d3.create("svg")
      .attr("width", width)
      .attr("height", 33)
      .attr("viewBox", `0 -20 ${width} 33`);

  while (true) {
    const t = svg.transition()
        .duration(750);

    const textUpdate = svg.selectAll("text")
      .data(randomLetters(), d => d)
        .attr("fill", "black")
        .attr("y", 0)
      .call(update => update.transition(t)
        .attr("x", (d, i) => i * 16));

    const textEnter = textUpdate.enter().append("text")
        .attr("fill", "green")
        .attr("x", (d, i) => i * 16)
        .attr("y", -30)
        .text(d => d)
      .call(enter => enter.transition(t)
        .attr("y", 0));

    const textExit = textUpdate.exit()
        .attr("fill", "brown")
      .call(exit => exit.transition(t)
        .attr("y", 30)
        .remove());

    yield svg.node();
    await Promises.tick(2500);
  }
}
```

General Update Pattern

The beginning of the general update pattern is to make a selection and compare to the data, like in in our Venn diagram.

Source: General Update Pattern

```
const textUpdate = svg.selectAll("text")
  .data(randomLetters(), d => d)
    .attr("fill", "black")
    .attr("y", 0)
  .call(update => update.transition(t)
    .attr("x", (d, i) => i * 16));
```

General Update Pattern

Note the 2nd argument for the data method. That is for assigning an ID to your datapoint, which here is a letter from the alphabet.

Source: General Update Pattern

```
const textUpdate = svg.selectAll("text")
  .data(randomLetters(), d => d)
    .attr("fill", "black")
    .attr("y", 0)
  .call(update => update.transition(t)
    .attr("x", (d, i) => i * 16));
```

General Update Pattern

Then we update those that have a 1:1 match. Selection + data will simply update existing elements with new data.

Source: General Update Pattern

```
const textUpdate = svg.selectAll("text")
  .data(randomLetters(), d => d)
    .attr("fill", "black")
    .attr("y", 0)
  .call(update => update.transition(t)
    .attr("x", (d, i) => i * 16));
```

General Update Pattern

Next we enter elements that don't currently exist. It uses the same selection as the update from before. Think of enter as a filter.

Source: General Update Pattern

```
const textEnter = textUpdate.enter().append("text")
    .attr("fill", "green")
    .attr("x", (d, i) => i * 16)
    .attr("y", -30)
    .text(d => d)
  .call(enter => enter.transition(t)
    .attr("y", 0));
```

General Update Pattern

Lastly, we exit the elements we need to remove.

```
const textExit = textUpdate.exit()
    .attr("fill", "brown")
  .call(exit => exit.transition(t)
    .attr("y", 30)
    .remove());
```

Source: General Update Pattern

Activity 1

Update with Join

Selection.join

Updating your visualization with the join method is the more modern approach with D3. It can be extremely concise if you don’t care to do any special treatment of each state, but still allows for that level of customization.

Selection.join

This code outputs the same visualization as we went through earlier, but uses the selection.join method.

```
{
  const svg = d3.create("svg")
      .attr("width", width)
      .attr("height", 33)
      .attr("viewBox", `0 -20 ${width} 33`);

  while (true) {
    const t = svg.transition()
        .duration(750);

    svg.selectAll("text")
      .data(randomLetters(), d => d)
      .join(
        enter => enter.append("text")
            .attr("fill", "green")
            .attr("x", (d, i) => i * 16)
            .attr("y", -30)
            .text(d => d)
          .call(enter => enter.transition(t)
            .attr("y", 0)),
        update => update
            .attr("fill", "black")
            .attr("y", 0)
          .call(update => update.transition(t)
            .attr("x", (d, i) => i * 16)),
        exit => exit
            .attr("fill", "brown")
          .call(exit => exit.transition(t)
            .attr("y", 30)
            .remove())
      );

    yield svg.node();
    await Promises.tick(2500);
  }
}
```

Selection.join

Like before, we start with the selection and compare to the data, like in in our Venn diagram.

Source: selection.join

```
svg.selectAll("text")
  .data(randomLetters(), d => d)
  .join(
    enter => enter.append("text")
        .attr("fill", "green")
        .attr("x", (d, i) => i * 16)
        .attr("y", -30)
        .text(d => d)
      .call(enter => enter.transition(t)
        .attr("y", 0)),
    update => update
        .attr("fill", "black")
        .attr("y", 0)
      .call(update => update.transition(t)
        .attr("x", (d, i) => i * 16)),
    exit => exit
        .attr("fill", "brown")
      .call(exit => exit.transition(t)
        .attr("y", 30)
        .remove())
  );
```

Selection.join

Now you'll see some of the differences. For handling the different states, we use logic within the join method.

Source: selection.join

```
svg.selectAll("text")
  .data(randomLetters(), d => d)
  .join(
    enter => enter.append("text")
        .attr("fill", "green")
        .attr("x", (d, i) => i * 16)
        .attr("y", -30)
        .text(d => d)
      .call(enter => enter.transition(t)
        .attr("y", 0)),
    update => update
        .attr("fill", "black")
        .attr("y", 0)
      .call(update => update.transition(t)
        .attr("x", (d, i) => i * 16)),
    exit => exit
        .attr("fill", "brown")
      .call(exit => exit.transition(t)
        .attr("y", 30)
        .remove())
  );
```

Activity 2

Transitions

Transitions

Perhaps one of the most fun parts of working with D3 is the ability to add transitions, i.e. animations, to your visualization. To transition an element between two coordinates, you simply use the transition method then specify the new coordinates and the transition method will handle the interpolation for you.

Transitions

Here is a very basic example of a circle on an SVG.

Source: [selection.join]

```
{
  const width = 500;
  const height = 200;
  const r = 50;

  const svg = d3.create("svg")
    .attr("width", width)
    .attr("height", height);

  const circle = svg.append("circle")
    .attr("r", r)
    .attr("cx", r)
    .attr("cy", height / 2)

  return svg.node()
}
```

Transitions

By adding the transition method and providing a new x coordinate, we can move the circle to the right.

Source: selection.join

```
{
  const width = 500;
  const height = 200;
  const r = 50;

  const svg = d3.create("svg")
    .attr("width", width)
    .attr("height", height)

  const circle = svg.append("circle")
    .attr("r", r)
    .attr("cx", r)
    .attr("cy", height / 2)
    .transition()
    .attr("cx", width - r)

  return svg.node()
}
```

Transitions

We can chain transitions so it moves after the first transition is completed.

Source: [selection.join]

```
{
  const width = 500;
  const height = 200;
  const r = 50;

  const svg = d3.create("svg")
    .attr("width", width)
    .attr("height", height)

  const circle = svg.append("circle")
    .attr("r", r)
    .attr("cx", r)
    .attr("cy", height / 2)
    .transition()
    .attr("cx", width - r)
    .transition()
    .attr("cx", r)

  return svg.node()
}
```

Transitions

We can add optional settings for controlling the duration of the transition.

```
{
  const width = 500;
  const height = 200;
  const r = 50;

  const svg = d3.create("svg")
    .attr("width", width)
    .attr("height", height)

  const circle = svg.append("circle")
    .attr("r", r)
    .attr("cx", r)
    .attr("cy", height / 2)
    .transition().duration(2000)
    .attr("cx", width - r)
    .transition()
    .attr("cx", r)

  return svg.node()
}
```

Source: [selection.join]

Activity 3, 4

Session 4 is a wrap!

Observable