PATTERNS IN POLYMER

BY CHRIS STROM

History

- 2013-11-16: Get started.
- 2013-12-15: Alpha edition, including 5 first draft chapters (Quick Intro/Refresher, Parent/Child Communication, Listening for Parent Events, Normalizing Events, Testing).
- 2014-01-15: Beta edition. Chapters on LiveReload with Grunt, Model Driven Views (MDV), working with external libraries (underscore.js is used) and configuration via JSON files. Many fixes.
- 2014-02-15: First edition. Added chapters on working with SVG, watching for changes, working with plain-old forms, and Angular.
- 2014-04-15: 1.1. Added chapters on internationalization and testing with "page objects." Dropped (possibly temporarily) chapters on theming and configuration — there was too much change in Polymer and Chrome to retain these as solid solutions in this edition.
- 2015-03-01: 1.2 (Beta). Major changes for latest Polymer libraries. Added new chapter on child communication with parent element.
- 2015-08-15: 1.5 (Beta). Major update for PolymerJS 1.0. Unfortunately had to (temporarily) drop efforts to keep Dart version updated.
- 2015-09-08: 1.5 (Final). Copy edits and minor updates since the beta release. Dropped the external CSS chapter (it may return in a future Polymer release).

Chapter 4. Passing Data from Child Polymer Elements

Code for this chapter resides in the book/code-dart/child_parent directory of the GitHub repository.

In the previous chapter, we discussed how Polymer elements might send information to elements that they contain. This begs the question of how best to send information in the other direction. How should child elements communicate information to Polymer elements in which they are contained?

The answer is almost always through events. On first reading of the Polymer documentation, this might seem an obvious answer. The Polymer library does a wonderful job of making it easy to fire events. And what better use of events than to send information to containing elements?

But in practice, it is super-easy to fall off the rails with communication. The best laid plans are quickly usurped when it seems natural to communicate between child and parent via the same mechanism we used to send information between parent and child.

And so, always remember the "Precipitation Pattern." Data rains down from parent to child via attribute binding. Data then evaporates back up in the form of events. And just as in real life, something is very wrong if this cycle is reversed.

Let's take a look at this in practice.

4.1. Setup

We will continue to use the `<x-parent>` and `<x-child>` Polymer elements from the previous chapter. The directory layout and code are identical to the way we left it at the end of that discussion.

4.2. Communication the Right Way

Recall from Chapter 3, *Passing Data to Child Polymer Elements*, that the `<x-child>` element accepts updates to its `count` property by exposing it as an attribute. In that chapter, the `<x-child>` element simply displayed that same value in its template.

Instead, consider a case that more closely follows real-life children: the parent says one thing, the child another. The `<x-parent>` will still send a count, but the `<x-child>` doubles that value:

```
@CustomTag('x-child')
class XChild extends PolymerElement {
  @published int count = 0;
  @observable int myCount = 0;

  XChild.created(): super.created();

  // Darn kids!!!
  countChanged(old, value) {
    myCount = 2 * count;
  }
}
```

For good measure, the `<x-child>` template should reflect the new `myCount` property:

```
<link rel="import" href="../../../packages/polymer/polymer.html">
<polymer-element name="x-child">
  <template>
    <div>
      <h2>Child</h2>
      <span>Count: {{count}}</span>
    </div>
  </template>
  <script type="application/dart" src="x_child.dart"></script>
```

```
</polymer-element>
```

The <x-child> sees what <x-parent> sent via the bound count attribute, but then completely disregards the parent by doubling that value. The <template> now shows the doubled value. So let's take it a step further and send this "sassy" answer back to <x-parent>.

As mentioned in the chapter introduction, we do so by firing an event:

```
  countChanged(old, value) {
    myCount = 2 * count;
    fire('x-child-answer', detail: myCount);
  }
```

Working back to the precipitation metaphor, information was already raining down via attribute binding between parent and child. We have now taken it a step further by evaporating data back up to the parent in the form of events. All that is left is to collect the "evaporating" events in the parent.

To accomplish that, we listen for events:

```
@CustomTag('x-parent')
class XParent extends PolymerElement {
  @observable int parent_count = 0;
  @observable int child_count = 0;

  XParent.created(): super.created() {
    new Timer.periodic(
      new Duration(seconds: 1),
      (_)=> parent_count++
    );
  }

  attached() {
    super.attached();
    on['x-child-answer'].
      listen((e){
        child_count = e.detail;
      });
```

```
  }
}
```

And reflect the sassy answer in the template:

```
  <template>
    <div>
      <h2>Parent</h2>
      <p>I say the count is: {{parent_count}}</p>
      <p>But my child insists the count is: {{child_count}}</p>
      <x-child count="{{parent_count}}"></x-child>
    </div>
  </template>
```

The circle is now complete. If we stick to this circle in our Polymer coding, we will find that things have a tendency to work out very nicely. Before moving on, let's take a quick look at what might go wrong if we break the circle.

4.3. Making Rain Fall Up

The first time child data is needed by a parent, it is extraordinarily tempting to use the same exact approach we used in the opposite direction. That is, the parent can push data down into the child via a bound attribute, so why can't the reverse work?

```
<!-- DO NOT TRY THIS! -->
<dom-module name="x-parent">
  <template>
     <!-- ... -->
     <x-child
       count="{{parent_count}}"
       answer="{{child_count}}"></x-child>
  </template>
</dom-module>
```

The main reason to avoid this approach is that it poorly encapsulates the functionality of the child element. Every element that wants to use

`<x-child>` now has to supply two arguments—one for input and one for output. If we find that we need to support additional input or output, the number of attributes quickly gets out of hand, making the element harder to use, harder to test, and harder to maintain.

Put another way, the best functions in object oriented programming accept a minimal number of arguments and return values. Functions and methods return values, they do not place values in supplied arguments.[1]

Return values from Polymer come in the form of events. This is not unique to Polymer either—updating native HTML element attributes never result in other attributes being updated. So if we tried this ourselves, we would not only be working against the grain in Polymer, but the web in general.

A more practical reason to avoid external communication via attributes is that it becomes *very* easy to produce infinite loops. If `<x-child>` updates its `answer` attribute and `<x-parent>` updates `parent_count` based on the answer, what happens next? The `<x-child>` element will see the update in its `count` attribute, update its `answer` and… boom. The browser crashes.

If our Polymer elements will never be used outside of a particular parent element, then *maybe* this approach could work. In general, there is little use in a non-reusable Polymer element.

Don't try to make it rain up.

4.4. Conclusion

Transferring information between Polymer elements and external Polymer elements (or even the containing page) is quite easy. Polymer's event system makes it easy for programmers to build elements that communicate well and that communicate in ways that the web expects. As long as we take care to avoid an easy trap, our elements—and other developers—will thank us.

[1] OK, low-level buffer programming might require a function to update supplied arguments, but Polymer ain't low-level buffer programming.

History

- 2013-11-16: Get started.
- 2013-12-15: Alpha edition, including 5 first draft chapters (Quick Intro/Refresher, Parent/Child Communication, Listening for Parent Events, Normalizing Events, Testing).
- 2014-01-15: Beta edition. Chapters on LiveReload with Grunt, Model Driven Views (MDV), working with external libraries (underscore.js is used) and configuration via JSON files. Many fixes.
- 2014-02-15: First edition. Added chapters on working with SVG, watching for changes, working with plain-old forms, and Angular.
- 2014-04-15: 1.1. Added chapters on internationalization and testing with "page objects." Dropped (possibly temporarily) chapters on theming and configuration — there was too much change in Polymer and Chrome to retain these as solid solutions in this edition.
- 2015-03-01: 1.2 (Beta). Major changes for latest Polymer libraries. Added new chapter on child communication with parent element.
- 2015-08-15: 1.5 (Beta). Major update for PolymerJS 1.0. Unfortunately had to (temporarily) drop efforts to keep Dart version updated.
- 2015-09-08: 1.5 (Final). Copy edits and minor updates since the beta release. Dropped the external CSS chapter (it may return in a future Polymer release).