1 way data binding vs 2 way data binding

Polymer dispone de un sistema de `binding` en una o dos direcciones.

Básicamente el binding de polymer nos permite ligar propiedades o subpropiedades de un polymer element a valores del localDOM.

Si la ligadura se produce de la propiedad o la subpropiedad hablamos de 1 way data binding. Si también incluye una ligadura entre localDOM y esa propiedad o subpropiedad hablamos de 2 way data binding.

Anotaciones de “binding”

En polymer existen 2 tipos de anoticaciones de binding para cada uno de los tipos mencionados.

En el caso de 1-way-binding:

```
[[property os subrproperty]]
```

En el caso de 2-way-binding:

```
{{property o subproperty}}
```

En este segunda caso además la propiedad debe estar adecuadamente configurada en nuestro objeto properties.

Es importante recordar que al igual que sucede en otros lugares de la librería se realizan conversiones camel case en los nombre de propiedades y subpropiedades también en el sistema de binding de modo que:

```
first-name=“{{managerName}}”
```

equivale a:

```
firstName = this.managerName
```

Binding a textContent y composiciones

En el local dom colocaremos:

```
<h2>Hello I am {{name}} {{lastName}}</h2>
```

y en nuestras propiedades tendremos:

```
  properties: {
      name:{
          type: String,
          value: "Mr.",
          reflectToAttribute: true
      },

    lastName:{
      type: String,
      value: "Element",
      reflectToAttribute: true
    },
  }
```

Binding a subpropiedades

En polymer podemos bindear también a subpropiedades

En el local dom tendremos:

```
<p>Mobile: {{phone.mobile}}</p>
<p>Work: {{phone.work}}</p>
```

Y en nuestras propiedades, por ejemplo:

```
phone:{
  type: Object,
  value: {
    mobile: "111-111-111",
```

```
    work: "222-222-222"
  },
},
```

Two way binding a elementos nativos

Es posible tomar directamente ligaduras a elementos nativos como input con la siguiente anotación: `{{propiedad::nombre-del-evento}}`

```
<h2>Hello I am {{name}} {{lastName}}</h2>

<input value="{{name::input}}">
<input value="{{lastName::input}}">
```

Expresiones en bindings

Se permiten sólo dos expresiones dentro de los bindings, la primera es:

```
{{!boolean}} : negación
```

Por otro lado existen los mencionados computed binding:

```
<input value="{{numberA::input}}">
<input value="{{numberB::input}}">
<p>Multiply: {{numberA}}x{{numberB}}={{multiply(numberA, numberB)}}</p>


...


  multiply: function(a, b){
    return a*b;
  },
```

Un computed binding nos permite ejecutar funciones cada vez que se producen cambios en una propiedad o subpropiedad.

Array bindings

Lo que parecería lógico utilizar como: `{{array[0]}}` o `{{array.0}}` no esta permitido.

Sin embargo los computed bindings nos pueden ayudar con los arrays, veamos un ejemplo:

```
<h3>Friends:</h3>
<p>[[arrayItem(friends.*, 0, 'name')]]</p>
<p>[[arrayItem(friends.*, 1, 'name')]]</p>
<p>[[arrayItem(friends.*, 2, 'name')]]</p>

...


    friends:{
      type: Array,
      value: [
        {name: "Mireia"},
        {name: "Lucy"},
        {name: "Edu"}
      ]
    }


...


  arrayItem: function(change, index, path) {
    // this.get(path, root) returns a value for a path
    // relative to a root object.
    return this.get(path, change.base[index]);
  },
```

Binding de atributos vs propiedades:

Attribute binding:
`attribute$="{{value}}" => el.setAttribute('selected', this.value)`

Property binding:
`property={{value}} => el.propery = this.value;`