Objeto listeners

Polymer tiene un poderoso sistema de eventos que hereda de HTML y se completa de forma inteligente para dar todas las capacidades de interacción orientada a eventos a nuestros componentes.

Veamos como:

```
Polymer({
  is: 'my-first-element',

  properties: {
      name:{
          type: String,
          value: "element",
          reflectToAttribute: true,
          readOnly: false,
          notify: false,
          observer: "_observeName"
      }
  },

  listeners: {
    'click': '_genericClick',
    'name.click': '_nameClick'
  },

  _genericClick: function(e){
    alert('_genericClick');
  },

  _nameClick: function(e){
    e.stopPropagation();
    alert('_nameClick');
  },

  ...
```

el objeto `listeners` nos permite situar listeners de eventos en todo el elemento, por ejemplo responder al click en cualquier parte del elemento con la función `_genericClick`

```
'click': '_genericClick',
```

o responder solo en la parte con `id="name"` con

```
'name.click': '_nameClick'
```

```
name.click : _nameClick
```

Daos cuenta que el método `_nameClick` incluye un

```
e.stopPropagation();
```

Ya que en caso contrario el evento se propagaría hacia abajo haciendo saltar también el `_genericClick` que monitoriza el componente entero.

Listeners declarativos on-event

Además podemos colocar listeners declarativos directamente en nuestro localDOM del siguiente modo:

```
<p id="byeP" on-tap="_handleTapAndLaunchEvent">Bye {{name}}</p>
```

De esta manera podemos llamar a la función `_handleTapAndLaunchEvent` sin necesidad de un objeto listeners.

A tener en cuenta:

1. Se pueden usar todos los eventos HTML
2. Todos los eventos deben escribirse en minusculas
3. on-tap es mejor que on-click ya que también cubre los navegadores móviles.
4. el uso de event.stopPropagation() es importante para evitar efectos indeseados y controlar hasta donde pueden provocarse nuestros eventos.

Listeners imperativos

A través de funciones JS podemos declarar listeners de un tercer modo:

```
this.listen(this.$.myButton, 'tap', 'onTap');
this.unlisten(this.$.myButton, 'tap', 'onTap');
```

los métodos `listen` y `unlisten` nos permiten tres parámetros, elemento al que escuchamos, tipo de evento y finalmente el método de callback que debe ejecutarse.

Eventos personalizados

A parte de los eventos típicos Polymer nos permite lanzar nuestros propios eventos personalizados de la siguiente manera:

```
this.fire('my-awesome-event',{foo:"bar"});
```

Siendo como anteriormente `this` el elemento, fire admite dos parámetros, el nombre del evento y un objeto a serializar con los detalles del mismo.

Eventos de cambio de propiedad y `notify`

Cuando notify es true en una propiedad de polymer ese elemento lanzará eventos de cambio de propiedad.

```
properties: {
     name:{
         type: String,
         value: "element",
         reflectToAttribute: true,
         readOnly: false,
         notify: false,
         observer: "_observeName"
     },


   myTest:{ //my-test-changed
     type: String,
     value: "default",
     notify:true
   }
 },
```

Cuando modifiquemos esa propiedad vía cualquiera de las posibles opciones vistas en anteriores y próximas lecciones nuestro elemento lanzará un evento que en este caso se llamará: `my-test-changed` fijaos como polymer cambia el nombre de la propiedad internamente de myTest a my-test de modo que el evento a escuchar es my-test-changed.

Una forma de ver que notify esta funcionando es haciendo un:

```
mielemento.addEventListener('my-test-changed', function(event){console.log(event)});
```

Eventos táctiles

Es importante conocer los eventos táctiles y utilizarlos siempre que sea posible en especial tap frente a click por el motivo expuesto anteriormente los eventos táctiles soportados en polymer son:

1. down
2. up
3. track
4. tap