Data Binding

HTML attribute vs. DOM property

The distinction between an HTML attribute and a DOM property is crucial to understanding how Angular binding works.

Attributes are defined by HTML. Properties are defined by the DOM (Document Object Model).

- A few HTML attributes have 1:1 mapping to properties. `id` is one example.
- Some HTML attributes don't have corresponding properties. `colspan` is one example.
- Some DOM properties don't have corresponding attributes. `textContent` is one example.
- Many HTML attributes appear to map to properties ... but not in the way you might think!

That last category is confusing until you grasp this general rule:

Attributes *initialize* DOM properties and then they are done. Property values can change; attribute values can't.

For example, when the browser renders `<input type="text" value="Bob">`, it creates a corresponding DOM node with a `value` property *initialized* to "Bob".

When the user enters "Sally" into the input box, the DOM element `value` *property* becomes "Sally". But the HTML `value` *attribute* remains unchanged as you discover if you ask the input element about that attribute: `input.getAttribute('value')` returns "Bob".

The HTML attribute `value` specifies the *initial* value; the DOM `value` property is the *current* value.

The `disabled` attribute is another peculiar example. A button's `disabled` *property* is `false` by default so the button is enabled. When you add the `disabled` *attribute*, its presence alone initializes the button's `disabled` *property* to `true` so the button is disabled.

Adding and removing the `disabled` *attribute* disables and enables the button. The value of the *attribute* is irrelevant, which is why you cannot enable a button by writing `<button disabled="false">Still Disabled</button>`.

Setting the button's `disabled` *property* (say, with an Angular binding) disables or enables the button. The value of the *property* matters.

The HTML attribute and the DOM property are not the same thing, even when they have the same name.

This fact bears repeating: **Template binding works with *properties* and *events*, not *attributes*.**

Uno de los principales valores de Angular es que nos abstrae de la lógica pull/push asociada a insertar y actualizar valores en el HTML y convertir las respuestas de usuario (inputs, clicks, etc) en acciones concretas. Escribir toda esa lógica a mano (lo que típicamente se hacía con JQuery) es tedioso y propenso a errores, y Angular 2 lo resuelve por nosotros gracias al ***Data Binding***.

Simplifiquemos el template anterior para centrarnos en el *data binding*:

```
<div>{{todo.subject}}</div>

<todo-detail [todo]="selectedTodo"></todo-detail>

<div (click)="selectTodo(todo)"></div>
```

Angular 2 dispone de 4 formas de data binding:

- **Interpolación:** (Hacia el DOM)
  Al hacer `{{todo.subject}}`, Angular se encarga de insertar el valor de esa propiedad del componente entre las etiquetas `<div>` donde lo hemos definido. Es decir, evalúa *todo.subject* e introduce su resultado en el DOM.

- ***Property binding:*** (Hacia el DOM)
  Al hacer `[todo]="selectedTodo"`, Angular está pasando el **objeto** *selectedTodo* del Componente padre a la propiedad *todo* del Componente hijo, en este caso de *TodoDetailComponent*. Recordemos de la ***sección Componentes*** que para esto el componente *TodoDetailComponent* habrá definido una propiedad *todo* con el decorador **@Input()**.

- ***Event binding:*** (Desde el DOM)
  Al hacer `(click)="selectTodo(todo)"`, le indicamos a Angular que cuando se produzca un evento *click* sobre esa etiqueta `<div>`, llame al método *selectTodo* del Componente, pasando como atributo el objeto *todo* presente en ese contexto. (Aunque hemos simplificado el ejemplo, esto venía de un bucle que itera el *array todos* obteniendo la variable *todo*).

  Angular mapea los eventos típicos de cualquier elemento del DOM para que los podamos utilizar como *event binding*.

- ***Two-way binding***: (Desde/Hacia el DOM)
  Un caso importante que no hemos visto con los ejemplos anteriores es el *binding bi-direccional*, que combina *event binding* y *property binding*, como podemos ver en el siguiente ejemplo:

  ```
  <input [(ngModel)]="todo.subject">
  ```

  En este caso, el valor de la propiedad fluye a la caja de *input* como en el caso *property binding*, pero los cambios del usuario también fluyen de vuelta al componente, actualizando el valor de dicha propiedad.

Pero para que funcione el [(ngModel)], además de usarlo hay que importar el módulo "FormsModule", pues [(ngModel)] es una directiva que pertenece a este Módulo:

app/app.module.ts

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';
import { AppComponent }   from './app.component';

@NgModule({
  imports:      [ BrowserModule, FormsModule ], //< added FormsModule here
  declarations: [ AppComponent ],
  bootstrap:    [ AppComponent ]
})

export class AppModule { }
```

Angular procesa los *data binding* una vez por cada ciclo de eventos JavaScript, desde la raíz de la aplicación siguiendo el arbol de componentes en orden de profundidad.

Los siguientes gráficos de la documentación de Angular 2 ilustran la importancia del *data-binding* para la comunicación entre componentes, así como componente-*template.*