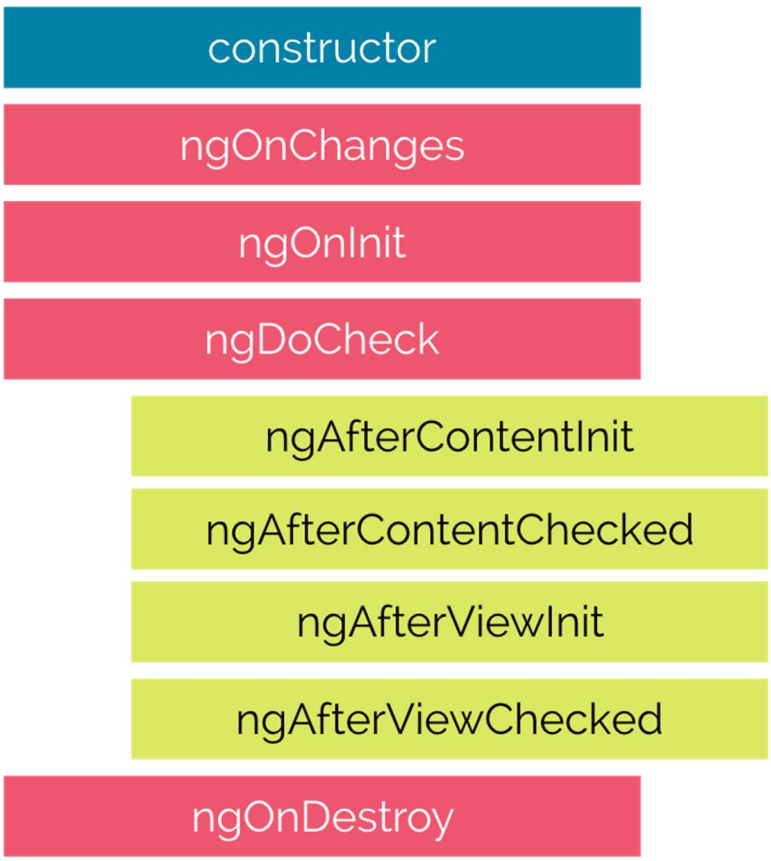
Angular: Componentes y sus ciclos de vida

En este articulo hablaremos sobre los ciclos de vida de los componentes en Angular. Ya que en angular los componentes son los principales pilares de la aplicación, es importante entender el ciclo de vida que estos tienen y la forma en la que se ejecuta ese ciclo de vida para así poder manipular los componentes en nuestra aplicación.

Lifecycle Method (Metodo de Ciclo de Vida)

En Angular, cada componente tiene un ciclo de vida, una cantidad de etapas diferentes que atraviesa. Hay 8 etapas diferentes en el ciclo de vida de los componentes. Cada etapa se denomina lifecycle hook event o en *'evento de enlace de ciclo de vida'*. Podemos utilizar estos eventos en diferentes fases de nuestra aplicación para obtener el control de los componentes. Como un componente es una clase de TypeScript, cada componente debe tener un método constructor.

El constructor de la clase de componente se ejecuta primero, antes de la ejecución de cualquier otro lifecycle hook. Si necesitamos inyectar dependencias en el componente, el constructor es el mejor lugar para hacerlo. Después de ejecutar el constructor, Angular ejecuta sus métodos de enganche de ciclo de vida en un orden específico.



Estas etapas están divididas principalmente en dos fases, una vinculada al componente en si y la otra vinculada a los hijos del componente.

Revisemos cada uno de los eventos:

- **ngOnChanges:** Este evento se ejecuta cada vez que se cambia un valor de un input control dentro de un componente. Se activa primero cuando se cambia el valor de una propiedad vinculada. Siempre recibe un change data map o mapa de datos de cambio, que contiene el valor actual y anterior de la propiedad vinculada envuelta en un SimpleChange

- **ngOnInit:** Se ejecuta una vez que Angular ha desplegado los data-bound properties(variables vinculadas a datos) o cuando el componente ha sido inicializado, una vez que ngOnChanges se haya ejecutado. Este evento es utilizado principalmente para inicializar la data en el componente.

- **ngDoCheck**: Se activa cada vez que se verifican las propiedades de entrada de un componente. Este método nos permite implementar nuestra propia lógica o algoritmo de detección de cambios personalizado para cualquier componente.

- **ngAfterContentInit**: Se ejecuta cuando Angular realiza cualquier muestra de contenido dentro de las vistas de componentes y justo después de ngDoCheck. Actuando una vez que todas las vinculaciones del componente deban verificarse por primera vez. Está vinculado con las inicializaciones del componente hijo.

- **ngAfterContentChecked**: Se ejecuta cada vez que el contenido del componente ha sido verificado por el mecanismo de detección de cambios de Angular; se llama después del método ngAfterContentInit. Este también se invoca en cada ejecución posterior de ngDoCheck y está relacionado principalmente con las inicializaciones del componente hijo.

- **ngAfterViewInit:** Se ejecuta cuando la vista del componente se ha inicializado por completo. Este método se inicializa después de que Angular ha inicializado la vista del componente y las vistas secundarias. Se llama después de ngAfterContentChecked. Solo se aplica a los componentes.

- **ngAfterViewChecked:** Se ejecuta después del método ngAfterViewInit y cada vez que la vista del componente verifique cambios. También se ejecuta cuando se ha modificado cualquier enlace de las directivas secundarias. Por lo tanto, es muy útil cuando el componente espera algún valor que proviene de sus componentes secundarios.

- **ngOnDestroy:** Este método se ejecutará justo antes de que Angular destruya los componentes. Es muy útil para darse de baja de los observables y desconectar los event handlers para evitar memory leaks o fugas de memoria.

Interfaces

Podemos definir los métodos directamente en la clase de componente, pero también podemos usar la ventaja de la interfaz, ya que cada uno de estos tiene una interfaz de TypeScript asociada. El nombre de esas interfaces es el mismo nombre que el método, simplemente sin el prefijo ng. Por ejemplo, ngOnInit tiene una interfaz llamada OnInit. Cada interfaz define solo un método. Es importante saber que el compilador de TypeScript basado en navegador no genera un error de compilación cuando no implementamos funciones de interfaz en nuestra clase. Pero, en el tiempo de compilación del código TypeScript, arrojará un error.