"Estructuras de Control" en SASS o SCSS:

las Estructuras de Control en Sass, las cuales nos permiten generar código CSS de muchas formas. Los condicionales nos ayudan a crear CSS que dependa del resultado de una expresión, y los diferentes tipos de ciclos generan bloques de CSS repetidamente dependiendo de contadores, listas o a través del resultado de expresiones.

Sass nos permite emplear estructuras de control en CSS gracias a las directivas de que proporciona pudiendo realizar variaciones en el estilo basándonos en una condición. Básicamente, podemos realizar condiciones if y bucles en CSS. Las directivas de control de flujo no son complejas si ya estas habituado a la programación ya que el uso de bucles y condiciones son el día a día.

Las estructuras de control en CSS están más centrados en los mixins, que son como funciones. La finalidad es dotar de mayor flexibilidad a la hoja de estilos cuando queremos agregar nuevas reglas.

ciclo de vida de los componentes de Angular:

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.

**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.

Cuál es la lista de comandos "ng cli"

|  |  |  |
| --- | --- | --- |
| [add](https://angular.io/cli/add) |  | Agrega soporte para una biblioteca externa a su proyecto. |
| [analytics](https://angular.io/cli/analytics) |  | Configura la recopilación de métricas de uso de CLI angular. Ver <https://v8.angular.io/cli/usage-analytics-gathering> . |
| [build](https://angular.io/cli/build) | b | Compila una aplicación angular en un directorio de salida llamado dist / en la ruta de salida dada. Debe ejecutarse desde un directorio de espacio de trabajo. |
| [config](https://angular.io/cli/config) |  | Recupera o establece valores de configuración angular en el archivo angular.json para el espacio de trabajo. |
| [deploy](https://angular.io/cli/deploy) | d | Invoca el generador de despliegue para un proyecto específico o para el proyecto predeterminado en el espacio de trabajo. |
| [doc](https://angular.io/cli/doc) | d | Abre la documentación oficial de Angular (angular.io) en un navegador y busca una palabra clave determinada. |
| [e2e](https://angular.io/cli/e2e) | e | Crea y sirve una aplicación Angular, luego ejecuta pruebas de extremo a extremo utilizando Protractor. |
| [generate](https://angular.io/cli/generate) | g | Genera y / o modifica archivos basados ​​en un esquema. |
| [help](https://angular.io/cli/help) |  | Enumera los comandos disponibles y sus breves descripciones. |
| [lint](https://angular.io/cli/lint) | l | Ejecuta herramientas de linting en el código de la aplicación Angular en una carpeta de proyecto determinada. |
| [new](https://angular.io/cli/new) | n | Crea un nuevo espacio de trabajo y una aplicación angular inicial. |
| [run](https://angular.io/cli/run) |  | Ejecuta un objetivo de Architect con una configuración de generador personalizada opcional definida en su proyecto. |
| [serve](https://angular.io/cli/serve) | s | Crea y sirve su aplicación, reconstruyéndose en los cambios de archivos. |
| [test](https://angular.io/cli/test) | t | Ejecuta pruebas unitarias en un proyecto. |
| [update](https://angular.io/cli/update) |  | Actualiza su aplicación y sus dependencias. Ver <https://update.angular.io/> |
| [version](https://angular.io/cli/version) | v | Salidas versión CLI angular. |
| [xi18n](https://angular.io/cli/xi18n) |  | Extrae mensajes i18n del código fuente. |

Qué es RXJS

RxJs es el port a JavaScript de la librería Reactive Extensions que ha sido portada a numerosos lenguajes de programación y que toma lo mejor de fuentes tan diversas como la programación funcional, el patrón Observer o el patrón iterador.

Un vez estudiado el tema de los observables, quedo sorprendido con lo útil y versátil que es la librería Rx. Podemos hacer observables de lo que queramos y manejarlos a nuestro antojo. Un gran avance en Angular al incorporar los observables y un gran acierto haberlos estudiado a fondo.

La librería reactiva RxJS es un conjunto de bibliotecas para generar programas asíncronos y basados ​​en eventos que utilizan secuencias observables y operadores de consulta.

RxJS, nos permite representan flujos de datos asíncronos con Observables, consultar los flujos de datos asíncronos utilizando los operadores y parametrizar la concurrencia en los flujos de datos asíncronos usando Schedulers. En pocas palabras, RxJS = Observables + Operadores + Programadores(schedulers).