**Documento de estilo de código: Angular**

Este artículo es una versión resumida de la guía de estilo del sitio web oficial de Angular, con algo de contenido eliminado / agregado. Además, el texto original explica los motivos de cada especificación y hay ejemplos. Haga clic para ver el texto original si es necesario: https://angular.io/guide/styleguide

**Responsabilidad única**

**Archivo único**

* Un archivo define algo, como un componente, un servicio, una tubería, una directiva
* No exceda las 400 líneas por archivo

**Función única**

* Definir una sola función.
* Una función no debe superar las 75 líneas.

**Convención de nomenclatura**

* El nombre del archivo es feature.type. \*\*, donde feature representa la característica y type representa el tipo.

1. .Module.ts para módulos
2. -Routing.module.ts para módulos de enrutamiento
3. .Component.ts | html | css para componentes
4. .Service.ts para servicio
5. .Pipe.ts para tuberías
6. Directiva con .directive.ts
7. Escriba .model.ts
8. Datos con .data.ts

* Use "-" para dividir palabras, como hero-list.component.ts
* Los nombres de los archivos de prueba de unidad siguen siendo los mismos que el objeto de prueba y terminan en .spec.ts
* El nombre del archivo de prueba de extremo a extremo sigue siendo el mismo que el objeto de prueba y termina con .e2e-spec.ts
* Los nombres de las clases se escriben en camello y se mantienen como el nombre del archivo

1. Módulo: por ejemplo, la clase definida por app.module.ts se llama AppModule.
2. Módulo de enrutamiento: como el nombre de clase definido por app-routing.module.ts Wei AppRoutingModule
3. Componentes: por ejemplo, la clase definida por hero-list.component.ts se llama HeroListComponent
4. Servicio: por ejemplo, la clase definida por logger.service.ts se llama LoggerService
5. Canalización: por ejemplo, la clase definida por address.pipe.ts se denomina AddressPipe
6. Directiva: por ejemplo, la clase definida por highlight.directive.ts se llama HighlightDirective
7. Excepción de tipo: dividido por módulo, un .model.ts define varios tipos
8. Datos: por ejemplo, la variable definida por address-book.data.ts se denomina addressBook

* El nombre del archivo de la entrada de inicio del script se especifica como main.ts, que no incluye ninguna lógica de negocios, pero recuerde tratar el error

***// main.ts***

1. **import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';**
2. **import { AppModule } from './app/app.module';**
3. **platformBrowserDynamic().bootstrapModule(AppModule)**
4. **.then(success => console.log(`Bootstrap success`))**
5. **.catch(err => console.error(err));**

* El selector de instrucciones se nombra usando una regla de joroba en minúsculas, como clickOutSide
* El selector de componentes se nombra en forma de un separador "-" que conecta letras minúsculas, como hero-list

**Especificación de código**

* Nomenclatura de clase
* Las constantes se definen con const y todas en mayúscula. Si hay varias palabras, use "\_" para conectarse, como HERO\_URL
* Bajo el entorno ES6, el uso de var para definir variables está prohibido
* Los nombres variables deben controlarse en 3 palabras, las palabras con abreviaturas comunes pueden abreviarse.
* El tipo de interfaz utiliza la regla de joroba en mayúsculas, no se recomienda prefijar con I, considere la clase en lugar de la interfaz
* Caso de camello en minúsculas para nombres de propiedades y métodos. No se recomienda que las propiedades y métodos privados tengan como prefijo "\_"
* Se recomienda utilizar una línea en blanco para distinguir la importación de bibliotecas de terceros de la importación del proyecto en sí.

**Estructura del proyecto**

* El criterio para una buena estructura de proyecto es ubicar, identificar rápidamente, mantenerlo lo más plano posible y no escribir código redundante / duplicado
* Ponga todo el código del proyecto en la carpeta src, cree carpetas para cada característica
* Las bibliotecas de terceros se colocan en una carpeta fuera de src
* Cada componente, servicio, canalización, instrucción es un solo archivo
* Cuando el componente tiene múltiples archivos auxiliares (htm, css, ts, spec.ts), se recomienda crear una carpeta separada

**Modulo**

* Cree el módulo raíz en src / app, se recomienda nombrarlo app.module.ts
* Cree un módulo para cada característica y mantenga consistentes los nombres de carpeta y módulo
* Se recomienda que el módulo compartido se denomine SharedModule y se coloque en app / shared / shared.module.ts
* Defina componentes, instrucciones y tuberías reutilizados en módulos compartidos, evite definir servicios en módulos compartidos
* Importe todos los módulos necesarios en el módulo compartido, como CommonModule y FormsModule, y exporte todos los módulos, componentes, instrucciones, tuberías reutilizados
* Considere colocar clases que se usan solo una vez en el módulo de características principales e importarlas solo en el módulo raíz. Se recomienda escribir guard.ts para garantizar
* Se recomienda que el módulo de funciones principales se denomine CoreModule y se coloque en app / core / core.module.ts
* Ponga servicios singleton en módulos de funciones principales, como ExceptionService y LoggerService
* Importe todos los módulos necesarios en el módulo de funciones principales, como CommonModule y FormsModule, exporte todos los componentes, servicios, etc. definidos.
* Coloque componentes que se usan solo una vez en el módulo de funciones principales y luego solo impórtelos en AppModule, como NavComponent y SpinnerComponent
* Los módulos de características independientes se pueden convertir en módulos de carga diferida, evitando los módulos de carga diferida importados en cualquier lugar, de lo contrario, los módulos se cargarán directamente

**Componente**

* Cuando la plantilla / estilo exceda las 3 líneas, escriba como archivo separado
* Eliminar archivos de estilo sin definición de estilo
* Definición de plantillas / estilos consistentes con la denominación de componentes
* Use rutas relativas al importar
* Use los decoradores @Input y @Output para modificar los datos de entrada y salida en lugar de usar los atributos de entradas y salidas en los metadatos
* No se recomienda cambiar el nombre de los datos de entrada y salida
* Defina según el orden de las variables, los constructores, las funciones del ciclo de vida y los métodos generales; los métodos generales se agrupan de acuerdo con los módulos funcionales de la página, y el método llamado se escribe más adelante; las variables y los métodos son primero públicos y privados;
* Escriba solo la lógica relacionada con la vista en el componente y ponga otra lógica empresarial en el servicio
* Ponga lógica empresarial reutilizable en servicios
* Los nombres de eventos no se recomiendan con el prefijo on
* Ponga la lógica de presentación en la clase del componente, no en la plantilla

**Instrucción**

* Cuando haya una lógica de visualización que no esté relacionada con la plantilla, use directivas de atributos, como la directiva [resaltar]
* Se recomienda utilizar @HostListener y @HostBinding en lugar de la propiedad de host de los metadatos.

**Servicio**

* Mantener reglas de uso para servicios de Singleton. Los servicios se utilizan para compartir datos y métodos en módulos de funciones o aplicaciones.
* Al igual que las funciones, un servicio tiene un solo propósito
* Inyecte el servicio en el componente / módulo de nivel más alto, de modo que el servicio singleton pueda compartirse entre subcomponentes y submódulos
* Use @Injectable para decorar clases de servicio en lugar de @Inject para decorar parámetros
* Operaciones relacionadas con datos, como solicitudes xhr, almacenamiento local y otros servicios abstractos.

**Ciclo de vida**

* Cuando se necesitan ganchos de ciclo de vida, implemente interfaces relevantes para prevenir errores de manera efectiva
* Herramientas auxiliares
* Para utilizar codelyzer Para mantener el código siguiendo las pautas, pero por supuesto puedes adaptar
* Use la herramienta de fragmentos de código de IDE para generar rápidamente fragmentos de código consistentes, como la instalación de fragmentos de código VS

**Fuente:** https://programmerclick.com/article/2245241791/