Guía de buenas prácticas de TypeScript

Estas son las directrices de codificación para los colaboradores de TypeScript. Esta NO es una guía prescriptiva para la comunidad de TypeScript. Estas directrices están destinadas a los colaboradores de la base de código base del proyecto TypeScript. Hemos elegido muchos de ellos por la consistencia del equipo. Siéntase libre de adoptarlos para su propio equipo. DE NUEVO: Esta NO es una guía prescriptiva para la comunidad de TypeScript.

Nombres

1. Utilice PascalCase para los nombres de tipo.
2. No utilice como prefijo para los nombres de interfaz.`I`
3. Utilice PascalCase para los valores de enum.
4. Utilice camelCase para los nombres de las funciones.
5. Utilice camelCase para los nombres de propiedad y las variables locales.
6. No lo use como prefijo para propiedades privadas._
7. Use palabras enteras en los nombres cuando sea posible.

Componentes

1. 1 archivo por componente lógico (por ejemplo, analizador, escáner, emisor, verificador).
2. No agregue nuevos archivos. :)
3. los archivos con sufijo se generan automáticamente, no los edite a mano.`.generated.*`

Tipos

1. No exporte tipos/funciones a menos que necesite compartirlo entre varios componentes.
2. No introduzca nuevos tipos/valores en el espacio de nombres global.
3. Los tipos compartidos deben definirse en `.types.ts`
4. Dentro de un archivo, las definiciones de tipo deben ser lo primero.

`null` y `undefined`

1. Uso. No utilice null.`undefined`

Supuestos generales

1. Considere objetos como nodos, símbolos, etc. como inmutables fuera del componente que los creó. No los cambies.
2. Considere las matrices como inmutables de forma predeterminada después de la creación.

Clases

1. Por coherencia, no utilice clases en la canalización del compilador principal. En su lugar, utilice cierres de funciones.

Banderas

1. Más de 2 propiedades booleanas relacionadas en un tipo deben convertirse en una bandera.

Comentarios

1. Utilice comentarios de estilo JSDoc para funciones, interfaces, enums y clases.

Instrumentos de cuerda

1. Use comillas dobles para las cadenas.
2. Todas las cadenas visibles para el usuario deben estar localizadas (haga una entrada en diagnosticMessages.json).

Mensajes de diagnóstico

1. Use un punto al final de una oración.
2. Utilizar artículos indefinidos para entidades indefinidas.
3. Se deben nombrar entidades definidas (esto es para un nombre de variable, nombre de tipo, etc.).
4. Al establecer una regla, el tema debe estar en singular (por ejemplo, "Un módulo externo no puede ..." en lugar de "Los módulos externos no pueden...").
5. Usa el tiempo presente.

Códigos de mensajes de diagnóstico

Los diagnósticos se clasifican en rangos generales. Si agrega un nuevo mensaje de diagnóstico, utilice el primer número integral mayor que el último número utilizado en el intervalo apropiado.

- rango 1000 para mensajes sintácticos
- 2000 para mensajes semánticos
- 4000 para mensajes de emisión de declaración
- 5000 para mensajes de opciones del compilador
- 6000 para mensajes del compilador de línea de comandos
- 7000 para noImplicitAny mensajes

Construcciones generales

Por una variedad de razones, evitamos ciertas construcciones y usamos algunas de las nuestras. Entre ellos:

1. No utilice declaraciones; en su lugar, use , y . Tenga en cuenta su semántica ligeramente diferente.`for..ints.forEachts.forEachKeyts.forEachValue`
2. Trate de usar , , y en lugar de bucles cuando no sea muy inconveniente.`ts.forEachts.mapts.filter`

Estilo

1. Utilice funciones de flecha sobre expresiones de función anónimas.
2. Solo parámetros de función de flecha envolvente cuando sea necesario. Por ejemplo, es incorrecto pero lo siguiente es correcto: `(x) => x + x`
   - `x => x + x`
   - `(x,y) => x + y`
   - `<T>(x: T, y: T) => x === y`
3. Siempre rodee el bucle y los cuerpos condicionales con aparatos ortopédicos rizados. Las declaraciones en la misma línea pueden omitir llaves.
4. Los brackets rizados abiertos siempre van en la misma línea que lo que sea necesario.
5. Las construcciones entre paréntesis no deben tener espacios en blanco circundantes. Un solo espacio sigue comas, dos puntos y punto y coma en esas construcciones. Por ejemplo:
   - `for (var i = 0, n = str.length; i < 10; i++) { }`
   - `if (x < 10) { }`
   - `function f(x: number, y: string): void { }`
6. Utilice una sola declaración por instrucción de variable (es decir, use over).`var x = 1; var y = 2;var x = 1, y = 2;`
7. `else` va en una línea separada de la abrazadera rizada de cierre.
8. Utilice 4 espacios por sangría.

Fuente

Directrices de codificación · Microsoft/TypeScript Wiki (github.com)