Tipos de unión e intersección en TypeScript

2 minutos

TypeScript proporciona opciones más avanzadas para declarar tipos. Los tipos Union e Intersection ayudan a controlar situaciones en las que un tipo se compone de dos o más tipos posibles. Los tipos literales permiten restringir los valores asignados a un tipo a una lista delimitada de opciones.

Tipos de unión

Un tipo de **unión** describe un valor que puede ser uno de entre varios tipos. Esta flexibilidad puede ser útil cuando no tenga controlado un valor (por ejemplo, los valores de una biblioteca, una API o una entrada de usuario).

El tipo `any` también puede aceptar tipos diferentes, así que ¿por qué querría usar un tipo de unión? Los tipos de unión restringen la asignación de valores a los tipos especificados en la unión, mientras que el tipo `any` no tiene restricciones. Otro motivo es la compatibilidad con IntelliSense.

Un tipo de unión usa la barra vertical o pleca (`|`) para separar cada tipo. En el ejemplo siguiente, `multiType` puede ser un valor `number` o `boolean`:

TypeScript

```
let multiType: number | boolean;
multiType = 20;         //* Valid
multiType = true;       //* Valid
multiType = "twenty";   //* Invalid
```

Con las restricciones de tipos, puede trabajar fácilmente con una variable de un tipo de unión. En este ejemplo, la función `add` acepta dos valores que pueden ser `number` o `string`. Si ambos valores son tipos numéricos, los agrega. Si ambos son tipos de cadena, los concatena. De lo contrario, genera un error.

TypeScript

```
function add(x: number | string, y: number | string) {
    if (typeof x === 'number' && typeof y === 'number') {
        return x + y;
    }
    if (typeof x === 'string' && typeof y === 'string') {
        return x.concat(y);
    }
    throw new Error('Parameters must be numbers or strings');
}
console.log(add('one', 'two'));  //* Returns "onetwo"
console.log(add(1, 2));          //* Returns 3
console.log(add('one', 2));      //* Returns error
```

Tipos de intersección

Los tipos de intersección están estrechamente relacionados con los tipos de unión, pero se usan de manera muy diferente. Un tipo de intersección combina dos o más tipos para crear uno que tiene **todas las propiedades** de los tipos existentes. Una intersección permite agregar tipos existentes de forma conjunta para obtener un tipo único que tenga todas las características que necesita.

Un tipo de intersección usa el símbolo de y comercial (`&`) para separar cada tipo.

Los tipos de intersección se usan con mayor frecuencia con las interfaces. En el ejemplo siguiente se definen dos interfaces, `Employee` y `Manager`, y luego se crea un tipo de intersección llamado `ManagementEmployee` que combina las propiedades en ambas interfaces.

TypeScript

```
interface Employee {
  employeeID: number;
  age: number;
}
interface Manager {
  stockPlan: boolean;
}
type ManagementEmployee = Employee & Manager;
let newManager: ManagementEmployee = {
    employeeID: 12345,
    age: 34,
    stockPlan: true
};
```

Puede obtener más información sobre las interfaces en el módulo Implementación de interfaces en TypeScript.

Tipos literales

Un literal es un subtipo más concreto de un tipo colectivo. Esto significa que `"Hello World"` es un elemento `string`, pero un elemento `string` no es `"Hello World"` dentro del sistema de tipos.

Hay tres conjuntos de tipos literales disponibles en TypeScript: `string`, `number` y `boolean`. Mediante el uso de tipos literales, puede especificar un valor exacto que debe tener una cadena, un número o un valor booleano (por ejemplo, "yes", "no" o "maybe").

¿Qué es la restricción literal?

Cuando se declara una variable mediante `var` o `let` en TypeScript, se indica al compilador que existe la posibilidad de que esta variable cambie su contenido. Al declarar una variable con let se escribe la variable (por ejemplo, como un elemento `string`), lo que permite un número infinito de valores posibles.

Por el contrario, al usar `const` para declarar una variable, informará a TypeScript de que este objeto nunca cambiará. Al declarar con tipos `const`, la escribe en el valor (por ejemplo, "Hola mundo").

El proceso de pasar de un número infinito de casos posibles a uno finito más pequeño se denomina restricción.

Definición de tipos literales

Los tipos literales se escriben como objetos, matrices, funciones o literales de tipo constructor, y se usan para crear tipos a partir de otros.

La mejor manera de mostrar el uso de los tipos literales es con un ejemplo. Esta definición de tipo crea un tipo literal denominado `testResult`, que puede contener uno de estos tres valores `string`:

TypeScript

```
type testResult = "pass" | "fail" | "incomplete";
let myResult: testResult;
myResult = "incomplete";    //* Valid
myResult = "pass";          //* Valid
myResult = "failure";       //* Invalid
```

Al establecer el valor de la variable `myResult`, `"incomplete"` y `"pass"` son entradas válidas, a diferencia de `"failure"`, que no lo es porque no es uno de los elementos de la definición de tipo `testResult`.

TypeScript también tiene tipos literales numéricos, que actúan igual que los literales de cadena anteriores. Por ejemplo:

TypeScript

```
type dice = 1 | 2 | 3 | 4 | 5 | 6;
let diceRoll: dice;
diceRoll = 1;    //* Valid
diceRoll = 2;    //* Valid
diceRoll = 7;    //* Invalid
```

También puede usar valores `boolean` al definir tipos literales o cualquier combinación de tipos.

Siguiente unidad: Tipos de colección en TypeScript

Continuar >

¿Qué tal lo estamos haciendo? ☆ ☆ ☆ ☆ ☆