Programación con genéricos en Angular

Gustavo García, 4 de mayo de 2023.

# Qué es

La programación con genéricos en Angular es una característica importante que permite definir componentes y servicios que sean reutilizables y flexibles. Los genéricos en TypeScript proporcionan una manera de definir tipos de datos y funciones que se pueden usar con diferentes tipos de datos.

Una parte importante de la ingeniería de software consiste en crear componentes que no solo tengan API coherentes y bien definidas, sino que también sean reutilizables. Los componentes que son capaces de trabajar con los datos de hoy y los datos del mañana le brindarán las capacidades más flexibles para construir grandes sistemas de software.

Una de las principales herramientas para crear componentes reutilizables son los genéricos, es decir, poder crear un componente que pueda funcionar con una variedad de tipos en lugar de uno solo. Esto permite a los usuarios consumir estos componentes y utilizar sus propios tipos.

# Hola mundo en genérico

Para empezar, hagamos el "hola mundo" de los genéricos: la función de identidad. La función de identidad es una función que devolverá todo lo que se le pase. Puede pensar en esto de manera similar al comando echo.

Sin genéricos, tendríamos que darle a la función de identidad un tipo específico:

function identity(arg: number): number {

  return arg;

}

O bien, podríamos describir la función de identidad usando cualquier tipo:

function identity(arg: any): any {

  return arg;

}

Si bien el uso de any es ciertamente genérico en el sentido de que hará que la función acepte todos y cada uno de los tipos para el tipo de argumento, en realidad estamos perdiendo la información sobre cuál era ese tipo cuando la función regresa. Si pasamos un número, la única información que tenemos es que se puede devolver cualquier tipo.

En cambio, necesitamos una forma de capturar el tipo de argumento de tal manera que también podamos usarlo para indicar lo que se devuelve. Aquí, usaremos una variable de tipo, un tipo especial de variable que funciona con tipos en lugar de valores:

function identity<Type>(arg: Type): Type {

  return arg;

}

Ahora hemos agregado una variable de tipo Type a la función de identidad. Type nos permite capturar el tipo que proporciona el usuario (por ejemplo, number), para que podamos usar esa información más adelante. Aquí, usamos Type nuevamente como el tipo de devolución. Podemos ver que se usa el mismo tipo para el argumento y el tipo de retorno. Esto nos permite traficar ese tipo de información en un lado de la función y fuera del otro.

Decimos que esta versión de la función de identidad es genérica, ya que funciona en una variedad de tipos. Pero a diferencia de any es preciso (es decir, no pierde ninguna información) sobre el tipo del argumento y el tipo de retorno.

Una vez que hemos escrito la función de identidad genérica, podemos llamarla de dos maneras. La primera forma es pasar todos los argumentos, incluido el argumento de tipo, a la función:

let output = identity<string>("myString");

Aquí establecemos explícitamente que Type sea string como uno de los argumentos de la llamada a la función, denotada con el <> alrededor de los argumentos en lugar de (). En este caso, lo que va entre los dos <> es un argumento de tipo, y string es una variable de tipo. Esto le avisa a la función que le vamos a pasar un argumento de tipo string, pero ahora esa argumento es la cadena "myString" que va paréntesis, somo se ve en el ejemplo.

La segunda forma es quizás la más común. Aquí usamos la inferencia de argumento de tipo, es decir, queremos que el compilador establezca el valor de Tipo para nosotros automáticamente en función del tipo de argumento que pasamos:

let output = identity("myString");

Tenga en cuenta que no tuvimos que pasar explícitamente el tipo entre paréntesis angulares (<>); el compilador simplemente miró el valor "myString" y estableció Type al tipo apropiado, que en este caso es string. Si bien la inferencia de argumentos de tipo puede ser una herramienta útil para mantener el código más corto y más legible, es posible que deba pasar explícitamente los argumentos de tipo como hicimos en el ejemplo anterior cuando el compilador no puede inferir el tipo, como puede suceder en ejemplos más complejos.

# Para qué se usa

En Angular, los genéricos se pueden utilizar para definir los tipos de datos de los parámetros de entrada y salida de los componentes y servicios. Esto permite crear componentes y servicios que sean reutilizables para diferentes tipos de datos.

Por ejemplo, para definir un servicio que pueda manejar diferentes tipos de datos, se puede utilizar un genérico como se muestra a continuación:

import { Injectable } from '@angular/core';

import { HttpClient } from '@angular/common/http';

import { Observable } from 'rxjs';

@Injectable()

export class DataService<T> {

  constructor(private http: HttpClient) {}

  getData(url: string): Observable<T> {

    return this.http.get<T>(url);

  }

}

En este ejemplo, se ha definido un servicio llamado DataService que utiliza un genérico T para definir el tipo de datos que se manejarán. La función getData del servicio utiliza el genérico T para definir el tipo de datos que se devolverán en la respuesta.

Para utilizar este servicio, se debe especificar el tipo de datos que se manejarán al momento de crear una instancia del servicio. Por ejemplo, si se quiere obtener datos de tipo User, se podría hacer lo siguiente:

import { Component } from '@angular/core';

import { DataService } from './data.service';

interface User {

  id: number;

  name: string;

  email: string;

}

@Component({

  selector: 'app-root',

  template: `

    <ul>

      <li \*ngFor="let user of users$ | async">{{ user.name }}</li>

    </ul>

  `,

})

export class AppComponent {

  users$: Observable<User[]>;

  constructor(private dataService: DataService<User>) {

    this.users$ = this.dataService.getData('https://api.example.com/users');

  }

}

En este ejemplo, se ha importado el servicio DataService y se ha definido una interfaz User para especificar el tipo de datos que se manejarán. Luego, en el constructor del componente, se ha creado una instancia del servicio DataService para manejar datos de tipo User. Finalmente, se ha utilizado la función getData del servicio para obtener los datos de la API y se ha asignado a la variable users$ para mostrarlos en la plantilla del componente.

En resumen, la programación con genéricos en Angular permite crear componentes y servicios que sean reutilizables y flexibles para diferentes tipos de datos. Los genéricos en TypeScript proporcionan una manera poderosa de definir tipos de datos y funciones que se puedan usar con diferentes tipos de datos, lo cual se puede aprovechar en Angular para crear aplicaciones más escalables y fáciles de mantener.

# Fuentes

Tomé de las fuentes citadas lo que me pareció más apropiado, lo edité y lo compaginé. Está todo muy cambiado respecto del original.

* ChatGPT
* <https://www.typescriptlang.org/docs/handbook/2/generics.html>