TRABAJO DIRIGIDO LP

TypeScript

Cuatrimestre de primavera, curso 2021/22

Hash 50178

Índice

Propósito del lenguaje

Typescript es un lenguaje diseñado y mantenido por Microsoft, de código abierto y programación libre. Se desarrolló sobre Javascript lo que significa que extiende la sintaxis ya existente de este lenguaje y, por tanto, cualquier código existente que funcione en Javascript, también lo hará en Typescript. En resumidas cuentas, se podría decir que Typescript es idéntico a su antecesor pero añadiendo tipos estáticos, que pueden ser usados ahí cuando hagan falta.

El principal problema que tiene Javascript, es que no fue pensado para crear sistemas grandes y complejos como los que tenemos hoy en día, y es en este contexto en el que aparece Typescript. El principal "problema" del predecesor es la libertad que ofrece para los tipos dinámicos, a pesar de que en usos a pequeña escala esto es cómodo y rápido tanto para programar y diseñar como para interpretar y entender el código. El problema viene, como ya hemos comentado, cuando se quiere hacer un proyecto a gran escala, en estos contextos la libertad de tipado puede ser un arma de doble filo y complicar la programación y la búsqueda de posibles *bugs* y errores. En otras palabras, podríamos decir que Typescript añade tipados estáticos a Javascript y, además, también añade objetos basados en clase.

Gracias a que Typescript es un añadido de características a Javascript, este lenguaje puede utilizarse también tanto en *frontend* como en *backend* utilizando frameworks como *Node.js* y *Deno*. Según encuestas, más del 60% de los programadores de Javascript utilizan Typescript como lenguaje de programación, y el 22% desearían aprender Typescript.

Paradigmas de programación

Los paradigmas de programación es el método utilizado para clasificar los distintos lenguajes de programación en función de sus características. Dentro de los paradigmas más comunes que existen, tenemos de tres tipos, los imperativos, los declarativos y los funcionales, de estos paradigmas nos centraremos en el imperativo. Un paradigma imperativo se caracteriza por constituirse por una sucesión de instrucciones en las que el programador da órdenes concretas, dicho de otra forma, el programador describe paso a paso todo lo que hará el programa. Typescript se encuentra dentro de los lenguajes de programación que cuentan con un paradigma imperativo, concretamente dentro de aquellos que tienen una programación estructurada. Es decir, Typescript es un lenguaje de programación en el que hay que definir con detalle que queremos que haga nuestro programa y cómo queremos que lo haga, además, al ser estructurado el flujo de control se define mediante bucles anidados, subrutinas y condicionales.

Además de esto, Typescript también se encuentra dentro de los lenguajes de programación con una estructura orientada a objetos. La programación orientada a objetos se basa en la representación de los objetos del problema a resolver como elementos con sus características y funciones. Esto trae consigo múltiples mejoras como la distinción de los distintos componentes del programa, simplificar el tratamiento de estos, añadir futuras mejoras o la búsqueda de errores entre otros. Sabiendo todo esto, se puede definir Typescript como un lenguaje de programación multiparadigma ya que es un lenguaje totalmente imperativo, orientado a objetos y además es estructurado.

Sistema de ejecución

Respecto a los sistemas de ejecución, actualmente existen dos predominantes, los sistemas de ejecución compilados y los interpretados. La principal diferencia entre ambos es la forma en la que interactúan con el computador antes de ejecutarse. Por un lado, los sistemas con una ejecución compilada, necesitan de un paso previo que convierte el código fuente en un archivo binario que puede ser leído por el ordenador a la hora de ejecutarse. Por otro lado, los sistemas interpretados, no requieren de esta conversión a un archivo binario, ya que a medida que se van procesando, el propio ordenador los va convirtiendo a binario y ejecutando en el momento.

Cuando hablamos del sistema de ejecución de Typescript, nos referimos a un sistema de ejecución interpretado en el que, como hemos explicado, el código a ejecutar se "traduce" en código que puede interpretar una máquina virtual y será esta máquina quien ejecutará el programa en cuestión. Pero en ningún momento se genera un archivo binario, sino que se traduce el código a un equivalente JavaScript y este, que es interpretado, se ejecuta a través de, por ejemplo, un navegador web que recibe el código JavaScript en su forma de texto original y ejecuta el script a partir de ahí.

Analizando en mayor profundidad el funcionamiento de la compilación y ejecución de Typescript nos encontramos con lo siguiente. El compilador, de Typescript, llamado *tsc*, no compila los programas de en un ejecutable binario, cómo podría ser el caso de *C++*, sino que *tsc* "traduce" el fichero Typescript en uno Javascript totalmente funcional y equivalente. Una vez compilado, dicho programa ya puede ser ejecutado en cualquier máquina que pueda ejecutar Javascript, como podría ser una página web o un servidor equipado con *Node.js*.

Sistema de tipos

Visto desde una percepción general, existen dos sistemas de tipos, los tipados estáticos o fuertemente tipados y los tipados dinámicos o débilmente tipados. En los lenguajes estáticos, el tipo se asocia con una variable o expresión y se comprueba en tiempo de compilación, por otro lado, con los lenguajes dinámicos, el tipo se asocia a un valor y este se comprueba en tiempo real. En los tipados fuertes, el propio lenguaje de programación impone restricciones para evitar la mezcla de valores de diferentes tipos. Los lenguajes de programación con tipados estáticos, proveen entornos con mayor control sobre el código y que ofrecen una gran robustez y estructura.

Centrándonos en Typescript, este se trata de un lenguaje fuertemente tipado, es decir se trata de un lenguaje de programación estático. Aún así Typescript va un poco más allá, debido a que está implementado sobre Javascript, el cual es un lenguaje de programación dinámico, Typescript se encarga de representar estáticamente sus tipos dinámicos. Esto permite que los programadores puedan definir variables y funciones tipadas sin perder la esencia de Javascript. Typescript cuenta con los siguientes tipos básicos:

- String
- Number
- Boolean
- Array
- Tuple
- Enum
- Any

- Void
- Never

Principales aplicaciones

Como hemos ido viendo a lo largo del trabajo, Typescript está pensado para que destaque en aquellos aspectos donde Javascript más flaqueaba. Javascript fue ideado para utilizarlo en pequeños proyectos o en entornos no muy complejos, el problema venía cuando grandes empresas, por ejemplo, necesitan usar Javascript a gran escala. En estas situaciones, los errores y *bugs* eran más frecuentes debido a los problemas de tipado y era más complicado organizar el código y facilitar su escalabilidad. Es en estos casos en los que Typescript, el superconjunto de Javascript, hace aparición y se convierte en una herramienta de gran utilidad y, en algunos casos, prácticamente indispensable.

Debido a que Typescript funciona en cualquier contexto en el que lo haga Javascript, los usos que tiene este lenguaje son los mismos que tiene su antecesor Javascript. Se puede utilizar en páginas web, ejecutándose desde el navegador o también en cualquier aplicación desarrollada a partir de *Node.js*. De igual forma, Typescript es utilizado actualmente tanto por programadores encargados del desarrollo de *frontend*, como por aquellos que se encargan del *backend* de un proyecto. Además, a las empresas les gusta utilizar Typescript ya que proporciona todo lo bueno que tiene Javascript, permite una mayor escalabilidad y facilita la lectura y comprensión del código a la hora de analizarlo.

Historia

Typescript, como ya hemos visto, nació con la intención de mejorar y acabar de perfilar los fallos que existían en el lenguaje de Javascript, el más destacable, la mala portabilidad de Javascript en proyectos a grandes escalas. Después de que Microsoft estuviera desarrollando este lenguaje de programación durante 2 años, Typescript, en la versión 0.8, fue publicado el 1 de octubre de 2012, aproximadamente hace una década. Pese al buen recibimiento e integración que tiene Typescript en la actualidad, al principio fue criticado ya que no existía un IDE para todos los sistemas operativos capaz de trabajar con este nuevo lenguaje.

En 2014 se publicó la versión 1.0 del lenguaje el cual ya contaba con soporte para genéricos entre otras distintas mejoras. Fue en julio de 2014, cuando Microsoft anunció un nuevo compilador para Typescript que aseguraba una velocidad de hasta 5 veces superior, por otro lado, el código fuente fue movido de *CodePlex* a *GitHub*. Con el paso del tiempo, y la salida de nuevas versiones, se han ido añadiendo mejoras como la posibilidad de prevenir que las variables tomen por valor *null* o la posibilidad de tener tuplas. Pese a que ahora Typescript se encuentra en la versión 4.6, no se han añadido nuevas mejoras rompedoras al lenguaje.

Actualmente Typescript se encuentra dentro de los lenguajes de programación más utilizados alrededor del mundo. Y es que el crecimiento que tuvo Typescript a principios de década fue exponencial y superó con creces las expectativas que existían respecto a este lenguaje, que era relativamente nuevo. La limpieza, legibilidad, escalabilidad son algunos de los factores que hacen que Typescript se haya convertido en un lenguaje tan popular y con tanta gente que lo utiliza o quiere aprenderlo.

Características particulares

Más allá de las propiedades que hemos ido a lo largo del trabajo, como que Typescript es un lenguaje de programación compilado o que está orientado a los objetos, vamos a comentar aquellas características de Typescript que más gustan a los desarrolladores que trabajan con este lenguaje de programación. El principal de todos, es que se trata de un lenguaje fuertemente apoyado por grandes compañías como Microsoft o Google, por otro lado, gracias a que está basado y desarrollado a partir de Javascript, podemos utilizar la plataforma *Node.js* para desarrollar aplicaciones y proyectos en el lado de los servidores.

Además, está visto que Typescript es uno de los mejores lenguajes actuales en lo que a documentación y legibilidad se refiere y esto es gracias a *Types*, este aspecto puede llegar a ser de vital importancia en proyectos a grandes escalas. Typescript permite que el compilador encuentre errores antes de que estos se lleguen a producir en tiempo de ejecución, lo que también puede llegar a ser crucial a la hora de ahorrar tiempo a los desarrolladores y programadores.

Typescript vs Javascript

Tal y como hemos visto a lo largo y hemos ido explicando, Typescript es al fin y al cabo lo que se conoce como un *superset* de Javascript, es decir, está totalmente basado y diseñado a partir de Javascript. Parece evidente entonces, que ambos lenguajes comparten muchas similitudes en cuanto a sintaxis y diseño a la hora de programar. Aún así, Typescript fue diseñado con particularidades que Javascript no posee, ya que se creó para mejorar y reforzar en aquellos aspectos donde más flaqueaba. Y es en estas diferencias entre los dos lenguajes donde aparecen las ventajas y desventajas de Typescript respecto a Javascript y otros lenguajes de programación.

A pesar de que, al fin y al cabo, programar es cuestión de comodidades y gustos personales, Typescript posee cualidades y características que los demás no. A continuación, comentaremos algunos aspectos populares que son dichos sobre Typescript sobre sus puntos más flojos y sus puntos fuertes como lenguaje de programación.

Principales desventajas de Typescript:

- La curva de aprendizaje es mayor que la de Javascript
- Sistema de tipos algo complejo respecto al que encontramos en Javascript
- Es necesario que se compile y por tanto no es directamente soportado por navegadores
- Falsa sensación de seguridad debido a la muestra de errores antes de que estos ocurran

Principales ventajas de Typescript:

- Es más rico textualmente que Javascript
- Permite y facilita la creación de código estandarizado
- Facilita el crecimiento de proyectos a grandes escalas
- Es uno de los mejores métodos de documentación que existen actualmente
- La escritura de código orientado a objetos es sencilla y esclarecedora

Ejemplos

El primer ejemplo que vemos, el de la Figura 1, nos muestra un segmento de código bastante simplista. Se llama a una función para imprimir por pantalla el resultado de la multiplicación entre los números 4 y 5. Podemos ver como se declara un array constante al principio y luego tenemos la creación de una función que, dependiendo de la operación que se quiera realizar hara una u otra. En el caso de que la supuesta operación demanda no exista se devolverá por consola ya que dicha operación es inexistente en el sistema.

Adicionalmente también se lanzará un aviso cuando se vaya a realizar una división entre 0. Como podemos ver, este lenguaje tiene una sintaxis muy parecida a la que hemos visto en otros lenguajes de programación como C o Java. Además podemos observar como en la cabecera de la función, se instancia de qué tipo serán las variables *a*, *b* y *op*.

```
const operations = ['add', 'substract', 'multiply', 'divide']

const calculator = (a: number, b: number, op: string) => {
      if(!operations.includes(op)) console.log('This operation is not defined')
      if(op == 'add') return a + b
      if(op == 'subtract') return a - b
      if(op == 'multiply') return a * b
      if(op == 'divide') {
            if(b == 0) return 'cant divide by 0!'
            return a / b
      }
}

console.log(calculator(5, 4, 'multiply'))
```

Figura 1: Ejemplo de Typescript con llamadas a funciones

En el segundo ejemplo, el que vemos en la Figura 2, podemos ver como funciona en Typescript el uso de clases, subclases, herencia y la creación de objetos. En el código de la Figura 2, hay dos clases declaradas, cada una con sus atributos y métodos pertinentes. Por un lado, tenemos la clase *Person*, que contiene el nombre, el apellido y la edad de una persona instanciada, además, tiene un método que imprime por pantalla el nombre y la edad de la persona en cuestión. Por otro lado, está la clase *Employee*, que hereda los atributos de la clase *Person* y añade un nuevo atributo y método.

Para declarar un empleado, además de añadir los atributos de dicha clase, tenemos que especificar su nombre, apellido y edad, porque recordemos que esta clase extiende de la primera. Después de instanciar a un empleado, y como podemos ver en el código, este objeto podrá utilizar tanto las funciones de la clase *Employee* como las funciones de la clase *Person*. A través de este pequeño ejemplo, podemos hacernos una idea de cómo funcionan las clases, los objetos y la herencia en Typescript. Es fácil darse cuenta que el funcionamiento de este lenguaje respecto a la orientación a objeto es, a simple vista, muy similar al que podemos encontrarnos en otros lenguajes de programación como podría ser *C++*.

```
class Person {
      firstName: string;
      lastName: string;
      age: number;

      constructor(public firstName: string, public lastName: string,
                  public age: number) {}

    sayHello() {
            console.log("My name is", this.firstName, this.lastName);
      }
}

class Employee extends Person {
      department: string;

      constructor(public firstName: string, public lastName: string,
                  public age: number, public department: string) {
          super(firstName,lastName,age);
      }

      sayDepartment() {
          console.log("I work in", this.department);
      }
}

const empl = new Employee("John","Smith",24,"Finance");
empl.sayHello();    //My name is John Smith
empl.sayDepartment(); //I work in Finance
```

Figura 2: Ejemplo de Typescript con clases, subclases y instanciación de objetos

Visión personal y crítica

Como conclusión y resumen a lo visto en este trabajo, podemos afirmar que Typescript es un lenguaje de programación que nació con la idea de mejorar un lenguaje ya existente pero que rápidamente se ha hecho hueco como uno de los más utilizados actualmente. Tal y como hemos ido viendo, Typescript es una herramienta muy potente en cuanto a legibilidad, limpieza, orden y claridad a la hora de programar y no solo eso, sino que además también constituye códigos fácilmente escalables y pensados para poder ser diseñados en grandes proyectos. Sabiendo esto, y viendo que grandes empresas como Microsoft o Google apoyan el proyecto en el que se ha convertido Typescript, no es difícil creer que este lenguaje sería creciendo y seguirá siendo utilizado en un futuro a la hora de diseñar tanto proyectos en *frontend* como en *backend*.

Respecto a mi opinión, Typescript es un lenguaje que no conocía a profundidad y con el que nunca he llegado a trabajar, aún así, para mi sorpresa me ha llamado la atención para bien. Mi experiencia programando se ha visto influenciada sobre todo por lenguajes de tipado fuerte, como puede ser *C++*, y que siguen los mismos paradigmas que sigue Typescript, estructurado, orientado a objetos, imperativo… Es por estas razones que no descarto en un futuro llegar a aprender Typescript e incluso algún día llegar a trabajar con este lenguaje de manera profesional. Debido a la facilidad que aporta com la escalabilidad y lo útil que es en sectores como el *frontend* o el *backend*, creo que es una herramienta de mucha utilidad que estaría bien tener a disposición. Además, la similitud a otros lenguajes que ya conozco, haría la curva de aprendizaje menos acentuada y cómoda.

Estudio bibliográfico

Descripción de las fuentes de información

Para la búsqueda de información durante la realización de este trabajo, he hecho uso únicamente de fuentes de internet, como pueden ser páginas webs o vídeos. Al ser el tema en cuestión un lenguaje de programación, me ha parecido que investigar a través de un buscador era una forma mucho más rápida, cómoda, eficiente y probablemente igual de completa que si hubiera ido a una biblioteca a buscar libros de Typescript. Además, hacer la búsqueda de esta forma, me ha permitido encontrar información concreta más rápido y la he podido contrastar de diversas fuentes. Otro problema con el que me he encontrado al leer algún libro sobre lenguajes de programación, es la extensa y compleja explicación que se da a veces para conceptos que deberían ser rápidos y sencillos de explicar.

La principal fuente de trabajo que utilicé al principio para coger una pincelada general de Typescript fue Wikipedia, ya que proveé información clara, concisa y organizada. Después de eso, a medida que fui entrando en temas más concretos sobre Typescript o necesitaba saber detalles más específicos, fui visitando otras páginas webs que contenían explicaciones más detalladas y extensas. Además de páginas webs también he visto algunos vídeos para acabar de aclarar algunos conceptos y funcionamientos sobre el lenguaje de programación. Ya que a veces, ver una representación visual o un trozo de código, puede ser más esclarecedor que una explicación detallada.

Evaluación de la calidad de la información

En esta sección, vamos a analizar que tan buenas son las fuentes de información consultadas y si podemos o no fiarnos de según que páginas webs o fuentes. Tal y como he comentado en el apartado anterior, mi punto de partida fue Wikipedia la cual, para llevarte una idea general sobre el lenguaje de programación, como está estructurado y para qué está pensando, es una buena herramienta. Aún así, a la que uno quiere profundizar un poco más en algún tema, Wikipedia como fuente de información se queda corta. Aquí es donde entran las otras fuentes consultadas. El siguiente paso fue consultar páginas webs, blogs y documentos más especializados y extensos sobre Typescript. Mediante estas fuentes más especializadas, y pudiendo contrastar varias de ellas, uno consigue tener una visión mucho más clara de aquello que estaba buscando.

Tras haber visitado bastantes fuentes más de fiar y haberlas contrastado, lo siguiente era empezar a mirar algún código explícito de Typescript. Lo primero que hice para ver de primera mano cómo se estructuraba y como era la sintaxis del lenguaje, fue ver varios vídeos de programadores utilizando Typescript, para ver y entender cómo funcionaba y qué reglas de sintaxis tiene. Por último, utilicé la documentación oficial de Typescript para acabar de entender cómo funcionaban algunos conceptos como puede ser la herencia entre clases. Bajo mi punto de vista, leer la documentación después de haber visto algo de código fue una decisión muy acertada, ya que si de primera me hubiera optado por leerme esa fuente, probablemente no la habría podido entender tan bien como lo hice al final.

Bibliografía

1. https://jpetit.jutge.org/lp/02-introduccio.html
2. The starting point for learning TypeScript
3. TypeScript - Wikipedia
4. ¿Qué es TypeScript, y por qué utilizarlo?.
5. Why does TypeScript exist?
6. What is TypeScript and why would I use it in place of JavaScript? - Stack Overflow.
7. Why You Should Use TypeScript
8. ¿Qué son los paradigmas de programación?
9. Unidad 6. El lenguaje de programación Typescript
10. Understanding TypeScript's “Compilation Process” & TypeScript Compiler
11. What is TypeScript? Strongly typed JavaScript | InfoWorld
12. https://www.youtube.com/watch?v=EUlM3wx546o
13. ¿Qué es TypeScript y para qué sirve?
14. TypeScript, el JavaScript de última generación - Blog de arsys.es.
15. Uso de TypeScript en las aplicaciones modernas
16. TypeScript: Más que el lenguaje para Angular
17. Tipos de datos · Introducción a TypeScript
18. TypeScript: Static or Dynamic? The war is over. | by Vlad Balin | ITNEXT
19. 10 Características que me gustan de TypeScript - Arquitectura Java
20. Los lenguajes de programación más usados [2022]
21. TypeScript contra JavaScript: ¿cuál deberías utilizar? | campusMVP.es
22. Ventajas y desventajas de TypeScript sobre JavaScript – Acervo Lima