Estudi de TypeScript

Llenguatges de Programació
Treball Dirigit

Hash: 8355

Facultat d'Informàtica de Barcelona
Curs 22-23 Q2

Índex

1 Introducció

1.1 Història

TypeScript és un llenguatge de programació creat per Microsoft i llançat el 2012, per tant la seva història no remunta molt enrere.

La idea de crear-lo va sorgir a causa de la creixent complexitat de les aplicacions web modernes[17]. Els desenvolupadors de Microsoft van veure que les aplicacions web estaven creixent en mida i complexitat a un ritme vertiginós, i que els llenguatges de programació existents com *JavaScript* no oferien les eines necessàries per a gestionar aquesta complexitat.

Així, van començar a treballar en un llenguatge de programació que combinés les avantatges de *JavaScript* amb un tipat estàtic opcional i característiques de programació orientada a objectes. El resultat va ser *TypeScript*, que va ser presentat per primera vegada al *TechEd Europe 2012*.

Un fet que ha impulsat molt el seu creixement és el la seva integració en llibreries populars, tals com *Angular 2* en 2014, *React* i *Vue* en 2017 i *Next.js* i *NestJS* en 2020[12]. Això, i la seva compatibilitat amb codi *JavaScript*, com s'explicarà més endavant, va facilitar molt l'aprenentatge i adopció del llenguatge per part de la comunitat.

Avui dia, *TypeScript* ha estat adoptat per moltes empreses i desenvolupadors de tot el món. Els seus avantatges l'han convertit en una opció popular per a aplicacions a gran escala i projectes de codi obert.

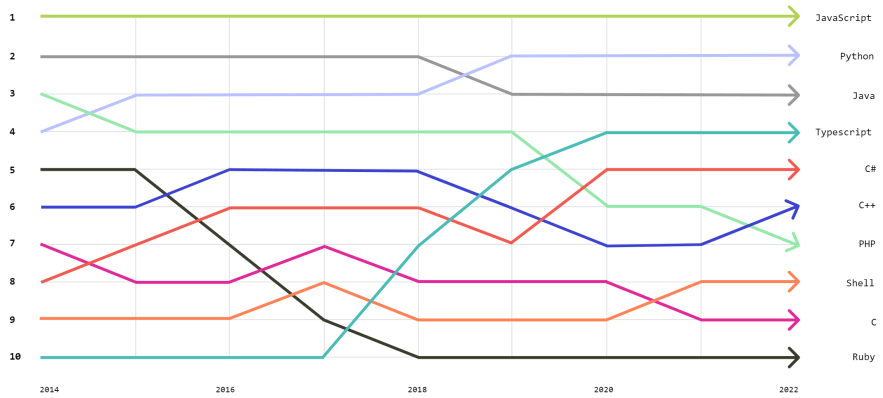

Figura 1: Llenguatges de programació més populars.[6]

1.2 Influencies i llenguatges similars

El disseny de *TypeScript* s'ha inspirat en varios llenguatges de programació.

En primer lloc, és un superset de *JavaScript*, amb el que son llenguatges

molt similars. *TypeScript* no pretén substituir *JavaScript*, sinó complementar-lo, especialment en aquells contextos on s'exposen més els seus punts febles[2].

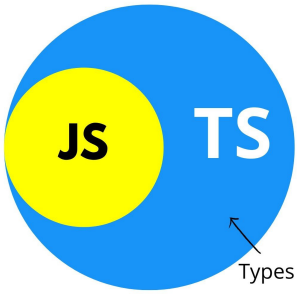

Figura 2: *TypeScript* es un superset de *JavaScript*[3]

D'altra banda, s'influencia de *Java* i *C#* en quant l'orientació a objectes, i *Haskell* en quant al tipat estàtic i les seves característiques de programació funcional[8]. Un llenguatge molt similar es també *ActionScript*, el qual va sortir el 1997 i té una idea molt similar amb els tipus, però el seu enfoc era ser un llenguatge per desenvolupar contingut interactiu multimèdia, no de propòsit general [15].

També és interessant comentar que el desenvolupament de *TypeScript* va ser liderat per Anders Hejlsberg, creador de *C#* i *Turbo Pascal*. Això ajuda a comprendre perquè s'assemblen tant els llenguatges [4].

Per aquestes raons, els llenguatges més similars a *TypeScript* son justament *JavaScript*, *Java*, *C#*, *Haskell* i *ActionScript*.

2 Usos

2.1 Propòsit del llenguatge

Com s'ha comentat anteriorment, *TypeScript* és una extensió de *JavaScript* que afegeix diferents característiques per ajudar als desenvolupadors[11]. Afegint tipus a les variables, funcions i altres elements en el seu codi, *TypeScript* pot ajudar a prevenir errors comuns en temps d'execució, com ara la referència a variables no definides o la crida a mètodes inexistents. A més, amb característiques com ara classes i mòduls permet escriure codi més organitzat i modular.

Per tant, el propòsit principal del llenguatge és millorar la qualitat del codi *JavaScript* i fer que el desenvolupament sigui més segur i eficient.

2.2 Principals aplicacions

Al ser *TypeScript* un superset de *JavaScript*, tenen les mateixes aplicacions.

Les principals aplicacions de *TypeScript* són[9]:

1. Desenvolupament web
2. Aplicacions web
3. Servidors web
4. Aplicacions mòbils
5. Desenvolupament de jocs
6. Presentacions i diapositives

Tot i això, en certs contextos *TypeScript* destaca sobre *JavaScript* i se sol utilitzar amb més freqüencia. Això succeeix, per exemple, aplicacions on la complexitat del codi és alta, o en projectes grans on la mantenibilitat i la seguretat són importants[11].

Com a curiositat, *TypeScript* es el llenguatge estàndard per programar extensions de *VS Code*.

3 Característiques del llenguatge

3.1 Paradigmes

Un paradigma de programació és una manera sistemàtica d'abordar el disseny d'un programa, que defineix un conjunt de regles i metodologies per resoldre problemes computacionals. Cada paradigma té la seva pròpia forma de descriure l'arquitectura, la lògica i les operacions del programar[10]. Es podria dir que representa una filosofia sobre com escriure el codi, en lloc d'un conjunt específic de regles o instruccions.

TypeScript és un llenguatge de programació multi-paradigma, i per tant suporta diversos paradigmes de programació, incloent[5][8]:

1. **Programació imperativa:** Ofereix control sobre el flux d'execució del programa mitjançant seqüències d'instruccions. Proporciona totes les estructures necessàries com són: bucles, condicionals, instruccions d'assignació i funcions.

   Exemple:

```
const conjunt = [1, 2, 3, 4, 5];
let suma = 0;

for (let i = 0; i < conjunt.length; i++) {
    suma += conjunt[i];
}

console.log(suma); // Resultat: 15
```

2. **Programació orientada a objectes[2]:** També permet la programació orientada a objectes, que és un paradigma de programació que es basa en l'ús d'objectes que interactuen entre si per resoldre problemes. Proporciona classes i interfícies per definir objectes, i suporta herència, encapsulament i polimorfisme.

   Exemple:

```
class Persona {
  constructor(private nom: string, private edat:
     number) {}

  public dirHola(): void {
    console.log('Hola, em dic ${this.nom} i tinc
       ${this.edat} anys.');
  }
}

// Exemple d'us
const persona = new Persona("Jordi", 24);
persona.dirHola(); // Output: "Hola, el meu nom
   es Jordi i tinc 24 anys."
```

3. **Programació funcional[2]:** Un element declaratiu que permet és la programació funcional, que és un paradigma de programació basat en el tractament de les funcions com a valors, i en l'ús de funcions pures per realitzar transformacions de dades. Proporciona funcions d'ordre superior, tipus de funció i funcions anònimes per permetre la programació funcional.

   Exemple:

```
const numeros = [1, 2, 3, 4, 5];

const nombresDuplicats = numeros.map((num) => num
   * 2);
console.log(nombresDuplicats); // Resultat: [2,
   4, 6, 8, 10]

const sumaDeNombres = numeros.reduce((acc, num)
   => acc + num, 0);
console.log(sumaDeNombres); // Resultat: 15
```

4. **Programació asincrònica:[1]** Finalment, també permet la programació asincrònica. Aquest que és un paradigma de programació que es basa en l'ús de funcions que s'executen en segon pla per resoldre tasques mentre el programa principal continua l'execució. Proporciona promeses i *async/await* per permetre la programació asincrònica.

Exemple:

```
function wait(ms: number): Promise<void> {
    return new Promise((resolve) => {
        setTimeout(() => {
            resolve();
        }, ms);
    });
}

async function execucioAsincrona(): Promise<void>
   {
    console.log("Abans d'esperar...");
    await wait(2000);
        console.log("Despres d'esperar...");
    }

execucioAsincrona();
//Resultat: "Abans d'esperar..."
//(2 segons)
//"Despres d'esperar..."
```

3.2 Sistema de tipus

El sistema de tipus de *TypeScript* és la seva principal diferencia en respecte a *JavaScript*, i l'atracció principal del llenguatge de programació.

TypeScript té un tipat estàtic i fort[3].

- **Estàtic**, ja que els tipus es comproven durant el temps de compilació en lloc de fer-ho en temps d'execució, proporcionant una detecció d'errors més ràpida i reduint la possibilitat d'errors de tipus durant l'execució.
- **Fort**, ja que els tipus no es converteixen implícitament, i si s'intenta operar amb diferents tipus, *TypeScript* emetrà un error.

Tot i això, els tipus son opcionals. Això no vol dir que la variable no tingui tipus, sinó que s'infereix, com s'explicarà més endavant. Aquest fet dona flexibilitat al llenguatge, en per exemple una etapa inicial de desenvolupament, on no cal explicitar els tipus[3].

Un exemple del sistema de tipus és que si declarem una variable com a *number*, no es pot assignar un valor de tipus *string* a aquesta variable, ja que no són compatibles.

```
let numero: number = 10;
numero = "hola"; // Aixo donaria un error de tipus en
   temps de compilacio
```

3.3 Sistema d'execució

El sistema d'execució de *TypeScript* implica la transpilació del codi en codi *JavaScript*, ja que *TypeScript* no s'executa directament en els navegadors o en els servidors[14]. El procés de transpilació permet que el codi pugui ser executat en qualsevol plataforma que suporti *JavaScript*.

Per exemple, si es té el següent codi *TypeScript*:

```
function saludar(nom: string): void {
    console.log('Hola, ${nom}!');
}
```

Després de compilar aquest codi utilitzant el compilador de *TypeScript*, el resultat seria el següent codi *JavaScript*:

```
function saludar(nom) {
    console.log('Hola, ${nom}!');
}
```

El codi *JavaScript* resultant és equivalent al codi *TypeScript* original i es pot executar en qualsevol plataforma que suporti *JavaScript*[2].

Trobo necessari comentar el sistema d'execució de *JavaScript* per completar la secció, degut a que hem d'executar aquest per veure el resultat del codi *TypeScript*. *JavaScript* sempre havia sigut un llenguatge interpretat, és a dir, no es compila, sinó que un interpret va llegint el codi i executant-lo línia per línia[14]. Això es el que porta a molts errors, ja que no es poden detectar fins executar el codi. Actualment, però, existeix la possibilitat de compilar *JavaScript Just In Time*, es a dir, es va compilant a mida que es va executant per tal d'optimitzar parts repetides del codi.

3.4 Altres característiques

A continuació comentaré altres característiques de *TypeScript* que he trobat interessants.

3.4.1 Tipus d'unió[2]

Els tipus d'unió permeten definir un tipus que pot ser una combinació de diversos altres tipus. Això significa que un valor que compleixi qualsevol dels tipus especificats en la unió serà considerat vàlid.

Per exemple, si volem definir una variable que pot ser un *number* o una *string*, ho podem fer-ho mitjançant una unió de tipus de la següent manera[2]:

```
let id: number | string;
id = 123; // valor valid
id = "abc"; // valor valid
id = true; // valor no valid
```

En aquest exemple, hem definit la variable *id* amb una unió de tipus *number* | *string*, que significa que pot ser qualsevol dels dos valors. Els valors *123* i *"abc"* són considerats vàlids per al tipus d'Unió i poden ser assignats a la variable *id*. No obstant, el valor *true* no és vàlid per al tipus d'unió i generaria un error de tipus.

Les unions de tipus són molt útils per definir variables que poden tenir diversos tipus possibles, i són una eina fonamental per a la programació en *TypeScript*.

3.4.2 Inferència de tipus [2]

TypeScript té inferència de tipus, el que significa que el llenguatge pot deduir els tipus de les variables on no s'ha especificat de manera explícita. Té dues maneres de fer aquesta inferència:

1. El "*best common type*" s'utilitza quan hi ha diferents expressions i s'ha de calcular el tipus comú que les uneix. Per exemple, en el codi[2]:

```
1        let x = [0, 1, null];
```

   TypeScript ha d'escollir un tipus comú que sigui compatible amb els tipus *number* i *null*. En aquest cas, el tipus d'*x* serà *(number | null)[]*.

2. D'altra banda, el "*contextual typing*" és quan el tipus d'una expressió s'infereix a partir del context en què s'utilitza. Per exemple, en el següent codi[2]:

```
1 window.onmousedown = function (mouseEvent) {
2     console.log(mouseEvent.button);
3     console.log(mouseEvent.kangaroo);
4 };
```

   TypeScript infereix que el tipus del paràmetre de la funció és *MouseEvent*, ja que *window* ja té una propietat *onmousedown* declarada amb aquest tipus, i ens dona l'error "*Property 'kangaroo' does not exist on type 'MouseEvent'*".

3.4.3 Decoradors [2]

Un altre característica interessant de *TypeScript* son els decoradors. Aquests estan a diversos llenguatges de programació que suporten els paradigmes de programació funcional i programació orientada a objectes.

Els decoradors són funcions que es poden adjuntar a diferents elements del codi per modificar-ne el comportament. Són útils ja que funcionen com una capa "per sobre", i a més ajuden a documentar el codi.

Per exemple, donat el cas que tenim una aplicació web que es connecta a una *API* per obtenir dades de diferents usuaris. Volem assegurar-nos que només els

usuaris autenticats poden accedir a aquesta informació. Per això, podem crear un decorador "autenticat" i adjuntar-lo a les funcions que accedeixen a l'*API*.

```
function autenticat(/*...*/) {
  // Retorna si l'usuari ha estat autenticat o no
}

class API {
  @autenticat
  obtenirDadesUsuari(id: string) {
    // crida a l'API per obtenir les dades de
       l'usuari amb l'id especificat
  }
}
```

En aquest exemple, el decorador "autenticat" comprova si l'usuari està autenticat abans de permetre l'accés a l'*API*. Així, qualsevol funció que es defineixi dins la classe "*API*" i que estigui decorada amb "*@autenticat*" només serà accessible si l'usuari està autenticat.

4 Crítica i visió personal

Personalment, crec que *TypeScript* es un bon llenguatge de programació que tapa molts forats de *JavaScript*, i és definitivament una evolució sobre aquest. Qualsevol persona que ha tocat una mica de *JavaScript* sap com de frustrant és debugar el codi. *JavaScript* és un bon llenguatge per tenir moltes aplicacions i proporcionar flexibilitat, però en molts casos les seves mancances representant una barrera a la productivitat, tant a curt plaç, pels problemes de debugar, com a llarg, ja que el codi tendeix a ser poc documentat i obscur. Trobo que això *TypeScript* ho soluciona molt bé, amb el tipatge fort i estàtic, ajudant a prevenir errors i produint un codi més robust, segur i documentat. A més, les funcions i característiques addicionals que ofereix, com les interfícies i els tipus de unions, permeten als desenvolupadors crear codi més modular i fàcilment mantenible.

No obstant això, en la meva recerca he trobat crítiques del llenguatge. Les més repetides d'aquestes són:

- Pot ser excessivament verbós i difícil de llegir per als desenvolupadors que estan acostumats a altres llenguatges de programació més concisos.
- El tipatge pot augmentar la complexitat del codi, ja que els desenvolupadors han de pensar més en la definició i validació de tipus.
- La validació estàtica de tipus pot portar a una falsa sensació de seguretat, ja que no cobreix tots els errors possibles.

En general, crec que *TypeScript* és una eina potent per als desenvolupadors, però com a amb qualsevol tecnologia, depèn de com s'utilitzi i de la situació. En

projectes grans i complexos, crec que els avantatges superen les desavantatges, però en projectes més petits i senzills, potser no és necessari.

5 Estudi bibliogràfic

A continuació s'analitzaran les diverses fonts usades, mitjançant el seu número en la taula de referències que apareix més a sota.

En primer lloc, per realitzar la introducció i comprendre el llenguatge he accedit diverses fonts, per tal de veure diferents punts de vistes. S'ha utilitzat Wikipedia [8] i diverses revistes científiques [4], [12], [15], [17], per obtenir context, però al ser fonts poc contrastades tenen una qualitat mitjana, i menys pes. La figura [6] s'ha obtingut d'estadístiques oficials, i per tant és una font de molta qualitat. Aquesta secció s'ha complementat contrastat amb la documentació oficial [2] i un article acadèmic, *Understanding TypeScript* [13], les dues fonts sent de qualitat molt alta, al ser la documentació una font oficial i l'article acadèmic una font altament contrastada i revisada.

Per a realitzar l'anàlisi dels usos, he utilitzat el context adquirit amb les fonts prèviament mencionades, i he afegit informació explícita mitjançant dues revistes científiques, [9] i [11]. La qualitat d'aquestes es, un altre cop, mitjana, al ser la seva validació limitada.

Finalment, la part més extensa d'aquest document es l'anàlisi de les característiques del llenguatge, i és també on més fonts s'han utilitzat. La font principal per aquesta secció ha estat la documentació oficial [2], d'on s'han extret molts exemples també, i és una font de qualitat molt alta. S'han utilitzat diferents revistes científiques [1], [5], [10], [14], i Wikipedia [8], per obtenir definicions i entendre característiques. Aquestes fonts tenen una qualitat mitjana per ser poc contrastades. Un altre font usada es GeeksForGeeks [3], coneguda pàgina de tutorials d'informàtica amb informació ben contrastada i de qualitat alta. Finalment, s'ha usat el llibre *Concrete Types for TypeScript* [16], una font de qualitat molt alta, per contrastar tota la informació. Els exemples de codi no citats són exemples propis, provats al *Playground* proporcionat a la web oficial de *TypeScript* [7].

En conclusió, el fet que *TypeScript* és un llenguatge nou i actual ha facilitat molt la recerca en aquest treball, al haver-hi moltes fonts recents. Les fonts utilitzades son diverses i de qualitat. Totes aquelles fonts de qualitat mitjana han sigut recolzades per fonts qualitat oficials o contrastades, de qualitat alta, el que assegura una informació bona.

Referències

[1] Asynchronous Programming in TypeScript — blog.fildon.me. https://blog.fildon.me/asynchronous-programming-in-typescript. [Accessed 01-May-2023].

[2] Documentation - TypeScript for JavaScript Programmers — typescriptlang.org. https://www.typescriptlang.org/docs/. [Accessed 01-May-2023].

[3] How Typescript Is Optionally Statically Typed Language ? - GeeksforGeeks — geeksforgeeks.org. https://www.geeksforgeeks.org/how-typescript-is-optionally-statically-typed-language/. [Accessed 01-May-2023].

[4] Microsoft TypeScript: Can the father of C# save us from the tyranny of JavaScript? — zdnet.com. https://www.zdnet.com/article/microsoft-typescript-can-the-father-of-c-save-us-from-the-tyranny-of-javascript/. [Accessed 01-May-2023].

[5] Paradigms In JavaScript — dev.to. https://dev.to/alamin$_y usuf/paradigms-in-javascript-1m31. [Accessed 01-May-2023].

[6] The top programming languages — octoverse.github.com. https://octoverse.github.com/2022/top-programming-languages. [Accessed 01-May-2023].

[7] TS Playground - An online editor for exploring TypeScript and JavaScript — typescriptlang.org. https://www.typescriptlang.org/play. [Accessed 01-May-2023].

[8] TypeScript - Wikipedia — en.wikipedia.org. https://en.wikipedia.org/wiki/TypeScript. [Accessed 01-May-2023].

[9] What are the uses of JavaScript - javatpoint — javatpoint.com. https://www.javatpoint.com/what-are-the-uses-of-javascript. [Accessed 01-May-2023].

[10] What Is A Programming Paradigm? Data Defined - Indicative — indicative.com. https://www.indicative.com/resource/programming-paradigm/. [Accessed 01-May-2023].

[11] Why does TypeScript exist? — typescriptlang.org. https://www.typescriptlang.org/why-create-typescript. [Accessed 01-May-2023].

[12] Dr. Derek Austin. A Brief History of TypeScript: From Origin to Modern Adoption — medium.com. https://medium.com/totally-typescript/a-brief-history-of-typescript-from-origin-to-modern-adoption-791368ec4b91. [Accessed 01-May-2023].

[13] Gavin Bierman, Martín Abadi, and Mads Torgersen. Understanding typescript. In Richard Jones, editor, *ECOOP 2014 – Object-Oriented Programming*, pages 257–281, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[14] Uday Hiwarale. Understanding TypeScript's "Compilation Process" the anatomy of "tsconfig.json" file — medium.com. https://medium.com/jspoint/typescript-compilation-the-typescript-compiler-4cb15f7244bc. [Accessed 01-May-2023].

[15] Gary Nelson. How ActionScript foreshadowed TypeScript — javascript.plainenglish.io. https://javascript.plainenglish.io/how-actionscript-foreshadowed-typescript-149cdb764de9. [Accessed 01-May-2023].

[16] Gregor Richards, Francesco Zappa Nardelli, and Jan Vitek. Concrete Types for TypeScript. In John Tang Boyland, editor, *29th European Conference on Object-Oriented Programming (ECOOP 2015)*, volume 37 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 76–100, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[17] IDG News Service staff. Microsoft augments JavaScript for large-scale development — infoworld.com. https://www.infoworld.com/article/2614863/microsoft-augments-javascript-for-large-scale-development.html. [Accessed 01-May-2023].