

Manual de TypeScript



Enrique Fernandez Guerra
Jaime Peña Tresancos
Miguel Angel Alvarez



desarrolloweb.com/manuales/manual-typescript.html

Introducción: Manual de TypeScript

En este manual explicamos TypeScript, un lenguaje de programación que compila a Javascript.

TypeScript es lo que se conoce como un superset de Javascript, es Javascript con nuevas utilidades que lo convierten en un lenguaje más completo. Entre otras muchas cosas, lo más particular de TypeScript es que incluye tipos en el lenguaje Javascript, para convertirlo en un lenguaje fuertemente tipado o de tipado estático.

En el Manual de TypeScript explicamos cómo usar TypeScript como lenguaje en cualquier proyecto y cuáles son las novedades que presenta en relación a Javascript, haciendo énfasis en las ventajas que nos ofrece el aplicar el superset.

Para aprovechar el Manual de TypeScript deberías conocer Javascript previamente, a un nivel medio por lo menos. De momento no tenemos como objetivo en hacer este manual un recorrido exhaustivo a todo lo que ofrece TypeScript, sino centrarnos más bien en las principales utilidades y en los temas que necesitamos conocer para poder usarlo en diversos entornos.

Uno de los entornos donde podrás sacarle partido es para resolver tus dudas en la programación con Angular, ya que Angular 2 (y las versiones sucesivas) usan TypeScript como lenguaje.

Encuentras este manual online en:

<http://desarrolloweb.com/manuales/manual-typescript.html>

Autores del manual

Las siguientes personas han participado como autores escribiendo artículos de este manual.

Miguel Angel Alvarez

Miguel es fundador de DesarrolloWeb.com y la plataforma de formación online EscuelaIT. Comenzó en el mundo del desarrollo web en el año 1997, transformando su hobby en su trabajo.



Jaime Peña Tresancos

Escritor. Colaborador habitual de revistas de tecnología y experto en nuevas tecnologías y programas Microsoft



Enrique Fernandez Guerra

Desarrollador web Frontend, especializado en Javascript y TypeScript. Trabaja actualmente en PlainConcepts. Colaborador de DesarrolloWeb es impulsor de los #jsIO



Introducción a TypeScript

Qué es TypeScript y qué necesitamos para empezar a usar este lenguaje de programación que compila a Javascript.

TypeScript es un lenguaje de programación de alto nivel que implementa muchos de los mecanismos más habituales de la programación orientada a objetos, pudiendo extraer grandes beneficios que serán especialmente deseables en aplicaciones grandes, capaces de escalar correctamente durante todo su tiempo de mantenimiento.

La característica fundamental de TypeScript es que compila en Javascript nativo, por lo que se puede usar en todo proyecto donde se esté usando Javascript. Dicho con otras palabras, cuando se usa TypeScript en algún momento se realiza su compilación, convirtiendo su código a Javascript común. El navegador, o cualquier otra plataforma donde se ejecuta Javascript, nunca llegará a enterarse que el código original estaba escrito en TypeScript, porque lo único que llegará a ejecutar es el Javascript resultante de la compilación.

En resumen, TypeScript es lo que se conoce como un "superset" de Javascript, aportando herramientas avanzadas para la programación que traen grandes beneficios a los proyectos.



Qué es un Superset

Es un lenguaje escrito encima de otro lenguaje o mejor dicho, que compila a otro lenguaje. En el caso de TypeScript es un lenguaje que compila a JavaScript, pero que incluye muchas facilidades y ventajas.

Los lenguajes como JavaScript basados en un estándar evolucionan muchas veces más lento que las necesidades de los desarrolladores. Entonces surgen empresas o comunidades que deciden expandir un lenguaje, aportando todas las herramientas que se echan en falta para poder desarrollar con las mejores condiciones.

Los superset compilan en el lenguaje estándar, por lo que el desarrollador programa en aquel lenguaje expandido, pero luego su código es "transpilado" para transformarlo en el lenguaje estándar, capaz de ser entendido en todas las plataformas.

Nota: En este artículo mencionamos que TypeScript compila a Javascript, pero quizás es más específico decir que "transpila" a Javascript. Transpilar es un término relativamente nuevo, viene del inglés "transpiler", fruto de la unión de las palabras "translate" y "compiler". Realmente es más correcto decir que TypeScript (u otros superset) transpilan, puesto que lo cierto es que no hay una compilación, sino es una traducción de un código fuente en otro código fuente en un lenguaje diferente.

Existen dos superset populares para Javascript. Por un lado tenemos CoffeeScript y por otro lado TypeScript. La diferencia principal es que mientras que CoffeeScript te aleja del lenguaje, con TypeScript escribes en un lenguaje que es muy similar al propio Javascript. Por ese motivo los programadores interesados en usar un superset se han volcado principalmente a TypeScript.

De todos modos, existen personas a favor y en contra de los superset, por varios motivos. Pero en este caso usar TypeScript no tiene una desventaja particular, ya que si queremos aprovechar las ventajas de Javascript tendremos que decantarnos por el uso de ES6, lo que nos obligará a usar un [transpilador como Babel](#). Al final, desarrollar en TypeScript o en el propio JavaScript con ES6 hace muy poca diferencia en la práctica, puesto que las dos alternativas te obligarán a usar un transpilador. En resumen, en el caso de usar ES6 necesitas de Babel, u otro transpilador, y en el caso programar en TypeScript usarás el propio transpilador de TypeScript (sigue leyendo para mayores aclaraciones).

Por qué TypeScript

En concreto TypeScript nos ofrece muchas de las utilidades que se necesitan en JavaScript para poder convertirlo en un lenguaje escalable, a la altura de las necesidades más exigentes. TypeScript nos ofrece muchas de las cosas que los desarrolladores de lenguajes más tradicionales vienen usando en su día a día.

Una de las diferencias fundamentales es que TypeScript es verdaderamente orientado a objetos, trayendo herramientas como la herencia, sobrecarga, etc. En resumen tiene cosas que suenan a lenguajes como Java, C++, C#, etc. Otro ejemplo clave es el del tipado estático. En el caso de TypeScript este tipado estático es opcional, pero obviamente su uso es muy recomendado y es una de las principales utilidades que nos van a facilitar mucho el trabajo y la depuración de los programas.

Nota: TypeScript se parece a C#. Es un hecho. El motivo es que el propio creador del lenguaje, Anders Hejlsberg, es ingeniero de Microsoft y actualmente arquitecto jefe de C#. El propio TypeScript nace con el objetivo de aportar a Javascript las herramientas y ventajas en el desarrollo con las que están acostumbrados desarrolladores de lenguajes más potentes como C# o Java.

Cuando trabajamos con TypeScript tenemos a nuestra disposición todas las herramientas de Javascript ES6 (EcmaScript 6) y muchas de las que encontraremos en la futura especificación ES7. Por lo tanto, usar TypeScript te asegura que podrás adelantarte al futuro y empezar a usar las posibilidades de estos estándares en el presente.

Además, TypeScript es la opción más natural de entrar en Angular 2, la nueva versión del más popular de los frameworks JavaScript. Si trabajas o vas a trabajar con Angular 2 apreciarás dominar TypeScript, por varios motivos que explicaremos. Entre ellos porque la mayor parte de la documentación la encontrarás con

este lenguaje y porque te ofrece las mejores herramientas para combinar con el estilo de desarrollo promovido por Angular 2.

Qué necesitamos para usar TypeScript

Básicamente necesitamos descargar dos programas. El primero es NodeJS, no porque necesitemos desarrollar con Node, sino porque el compilador de TypeScript está desarrollado en NodeJS.

Desde el sitio de NodeJS encontramos las opciones para la instalación en nuestro sistema operativo, por medio de un típico instalador con su asistente (siguiente, siguiente...) Más información en el Manual de NodeJS.

Luego necesitarás el TSC (Command-line TypeScript Compiler), la herramienta que nos permite compilar un archivo TypeScript a Javascript nativo. Este software es el que está realizado con NodeJS y su instalación se realiza vía npm con el siguiente comando:

```
npm install -g typescript
```

Cómo crear y compilar un archivo TypeScript

El archivo TypeScript lo creas con cualquier editor de texto para programadores, como habitualmente haces con cualquier otro lenguaje. Solo que generalmente usarás la extensión ".ts".

Algo muy importante: Cualquier código Javascript compila en TypeScript. Esto quiere decir que el código que tenemos en Javascript lo podemos usar también perfectamente dentro de Javascript. Por tanto, para trabajar con TS podemos usar perfectamente los conocimientos que tenemos en Javascript, agregando diversas ayudas que nos ofrece TypeScript.

Este sería un posible código TypeScript donde hemos colocado básicamente unos cuantas declaraciones de tipos en variables. Además de una función que se ha declarado devuelve "void" (nada).

```
var a:number = 9;
a += 4;

function mostrar(b:string) :void{
    console.log(b);
}
mostrar('hola');
```

Para compilarlo usarás el mencionado TSC y lanzarás un sencillo comando que convertirá el código a Javascript nativo.

```
tsc ejemplo.ts
```

La herramienta TSC incluye además "watchers" que permiten vigilar cambios en los archivos TS, compilando su código a JS en el instante sin que tengamos que intervenir. Además es habitual que el

compilador, se use en conjunto con otras herramientas como Gulp para mejorar aún más el flujo de desarrollo.

Existe asimismo un archivo de configuración llamado "tsconfig.json" en el que podemos indicar todas las opciones que queramos para el compilador (por ejemplo el estándar EcmaScript al que deseamos compilar el código, el tipo de reporte de errores que se desea, las rutas donde colocar los archivos compilados, etc).

Editores y TypeScript

Una de las cosas que debemos de conseguir cuando vamos a programar con TypeScript es un editor que lo soporte. El motivo es que los editores son capaces de compilar el código TypeScript a la vez que se está escribiendo, informando de errores en el código en el instante de su creación, lo que ahorra mucho tiempo y facilita el flujo de desarrollo.

Te informará de asuntos como:

- Una variable con tipado estático a la que se le intenta cargar un dato de otro tipo
- Una función que devuelve un valor de un tipo distinto al que debería
- Una función a la que no se le están pasando los valores de tipos correctos, o los objetos de clases correctas
- Un objeto sobre el que se intentan invocar métodos privados
- Y la lista no para de crecer...

Además, al tipar las variables seremos capaces de obtener muchas más ayudas "intellisense", como métodos invocables sobre cadenas o números, métodos que contiene un objeto, etc.

Existen editores que ya incorporan las ayudas para la programación en TypeScript, como Visual Studio o Visual Studio Code (Este último un editor ligero y gratuito para cualquier uso). Otros editores como Vim, Atom, Sublime Text o Brackets necesitarán la instalación de plugins especiales para poder aportar todas las ayudas de TypeScript.

TypeScript Playground

Un sitio interesante para probar TypeScript sin necesidad de instalar nada, o para revisar rápidamente cómo un código TS compilaría a Javascript, es el [TypeScript Playground](#).

Encontrarás dos cajas de texto, en la primera escribes código TypeScript y en tiempo real verás cómo este código compila a Javascript en la caja de la derecha. Además encontrarás ayudas en la caja de la izquierda, donde escribes TS, indicando en qué partes de tu código se encuentran problemas y sus motivos.

Esta es una herramienta muy útil tanto para aprender TypeScript como para probar puntualmente un código que os recomendamos experimentar ya mismo. Tiene un desplegable muy útil porque ya te muestra varios ejemplos de código ya escritos, si es que quieres ver cómo funciona el compilador sin tener ni idea de cómo se escribe en TS.

Vídeo de presentación de TypeScript

Para acabar os ofrecemos a continuación el vídeo de presentación de TypeScript, que grabamos con motivo del inicio del [Curso de TypeScript en EscuelaIT](#). En esta clase online tuvimos a nuestros profesores del

curso, que nos explicaron para qué sirve TypeScript y nos realizaron unas cuantas demostraciones de su uso, capaces de aclarar dudas y facilitar los primeros pasos.

Para ver este vídeo es necesario visitar el artículo original en:

<http://desarrolloweb.com/articulos/introduccion-a-typescript.html>

Este artículo es obra de *Enrique Fernández Guerra*

Fue publicado por primera vez en 02/06/2016

Disponible online en <http://desarrolloweb.com/articulos/introduccion-a-typescript.html>

Microsoft TypeScript

Primer programa con Visual Studio 2012.

TypeScript es una extensión del lenguaje JavaScript desarrollada por el prestigioso creador/programador Anders Hejlsberg, quien ya posee un destacado currículum de lenguajes de programación, entre ellos, los tan importantes Turbo Pascal, Delphi y C#.

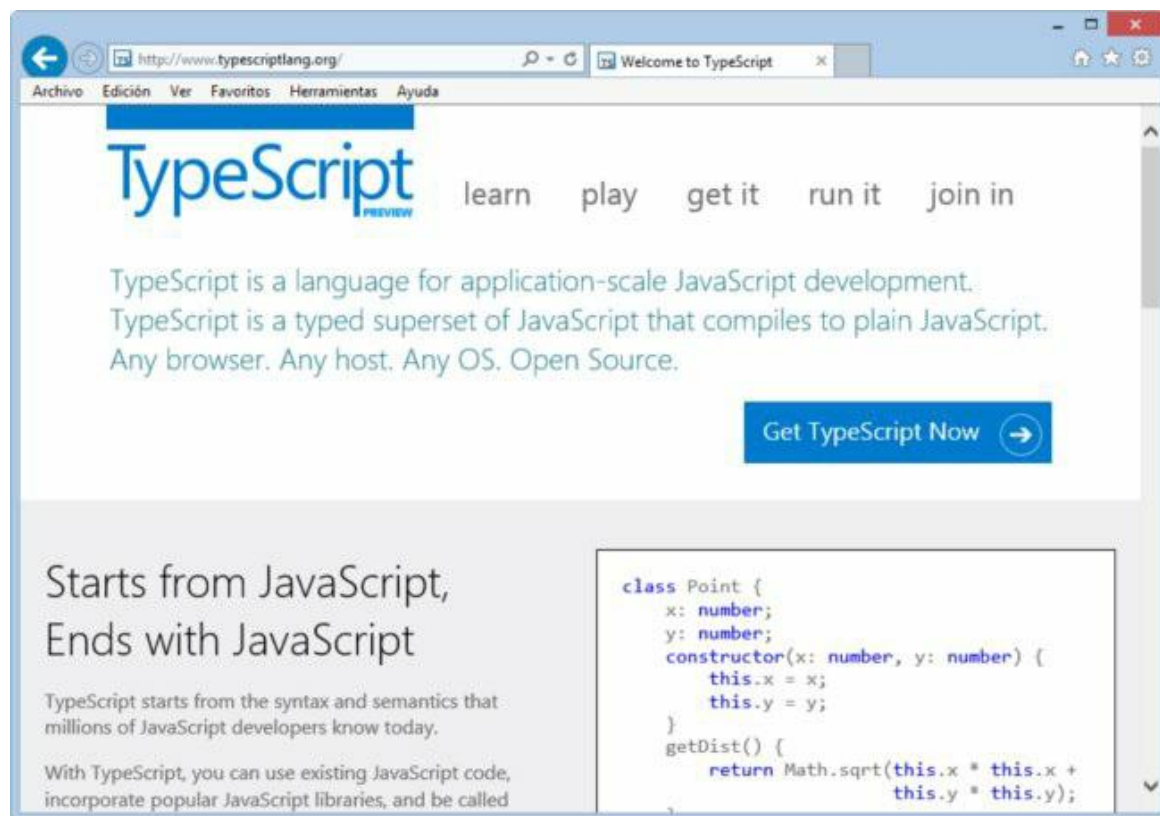
En el presente artículo trataremos lo esencial del nuevo lenguaje de programación y desarrollaremos un ejemplo, basándonos en el entorno de programación de Visual Studio 2012.



Fuentes de información oficiales

Para más información acerca del lenguaje puede acudir, inicialmente, a las siguientes fuentes oficiales en Internet:

- www.typescriptlang.org
- typescript.codeplex.com (lugar de descarga)
- go.microsoft.com (especificación en PDF)
- www.ecmascript.org
- www.ecma-international.org



Lo más notable del lenguaje

De la propia especificación del lenguaje –que se puede obtener de la dirección URL que hemos dado más arriba–, se desprende y así resaltamos lo siguiente:

1. Se trata de un superconjunto de JavaScript
2. Cada programa TypeScript tiene su correspondiente en JavaScript y viceversa
3. Opcionalmente, puede proporcionar un mapeado de las fuentes que permita su depuración
4. Incluye ya propuestas dadas para ECMAScript en su versión 6 –la última aprobada es la 5.1–, como son las clases y los módulos –véase más abajo–
5. Incluye ya la inferencia de tipo en la declaración de variables y funciones. Se trataría de una de las columnas vertebrales del nuevo lenguaje –nótese su propio nombre–
6. Define declaraciones ambiente que son visibles en el ámbito del archivo TS, pero no se propagan en el archivo JS correspondiente, para así poder asimilarse a declaraciones explícitas en navegadores o en librerías específicas, como por ejemplo JQuery
7. El lenguaje en sí es débilmente tipado –tipificado, si se prefiere– y se permiten declaraciones en tiempo de ejecución, de modo que no se atiene a una rigidez de declaraciones tales como en C# o Java
8. Un módulo es un objeto totalmente anónimo, todo lo que se encuentre dentro de él estará oculto al mundo exterior, a menos que se haga visible de manera explícita. Dentro se pueden definir variables, funciones, objetos, clases e interfaces. En el correspondiente archivo JS se traducirán en componentes de un objeto encapsulado
9. Una interfaz es un conjunto de declaraciones de tipo, sin implementaciones de contenido; por ejemplo declaraciones de simplemente el tipo de variables y funciones –pero sólo eso, el tipo–. Son de validez exclusiva en tiempo de compilación y su utilidad principal estriba en la documentación y validación del tipado –o tipificación– correcto(a) de las declaraciones de variables, funciones, ...

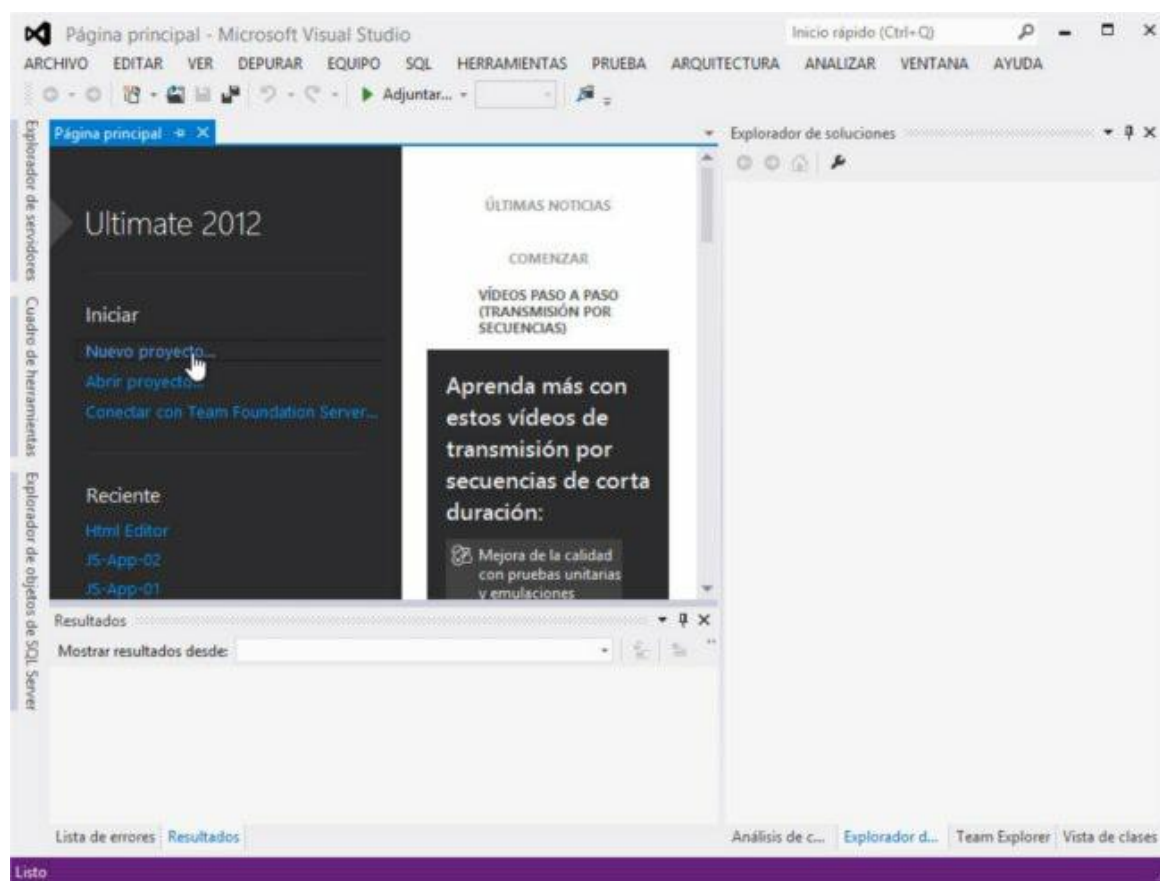
10. El soporte de clases abarca bastante de lo que cabría esperar –dentro de su simplicidad, que no es momento de tratar-, herencia, accesibilidad pública y privada y sobrecarga, básicamente

Un ejemplo basándonos en Visual Studio 2012

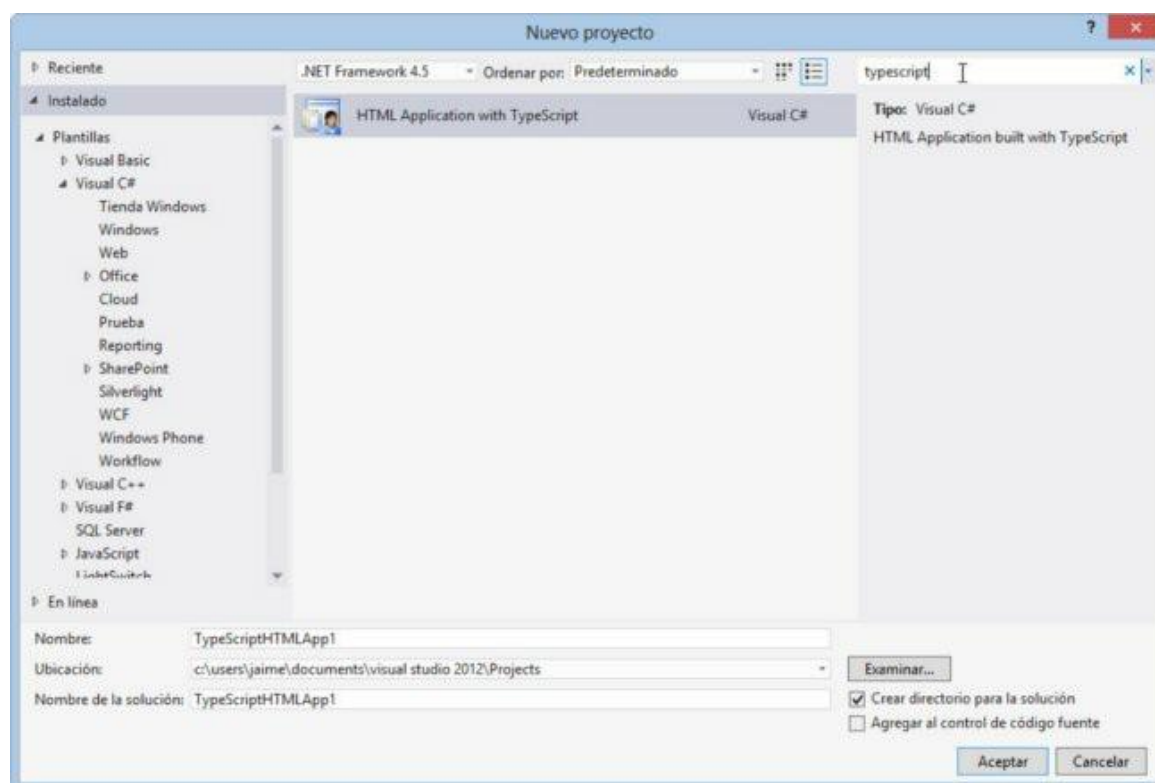
Para hacer desarrollos con TypeScript no necesitamos más que el compilador en línea de comandos y un editor de textos, como podría ser el modesto Bloc de notas; lógicamente, para un mejor y más cómodo trabajo nos alegrará saber que podemos, ya en el momento presente, apoyarnos en Visual Studio, en sus versiones 2012, tanto las profesionales como las gratuitas –Express-. Nosotros pasaremos a implementar un ejemplo con éste entorno visual. En primer lugar hemos de instalar el complemento –plug-in- de desarrollo de TypeScript para Visual Studio 2012 en sus diversas versiones. Se trata de una tarea sencilla en la que primeramente deberemos ir al sitio de Internet:

<http://www.microsoft.com/en-us/download/details.aspx?id=34790>

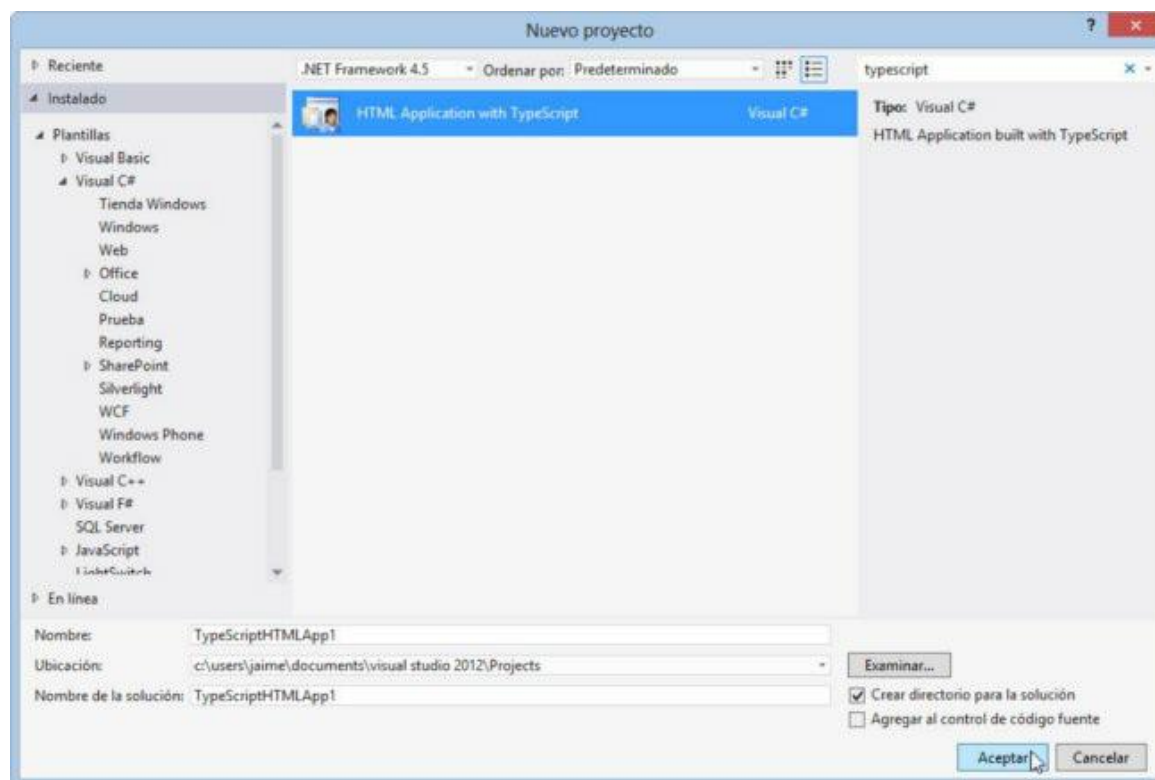
Una vez instalado, desde el entorno de desarrollo de Visual Studio 2012 crearemos un nuevo proyecto como es costumbre.



Pero en vez de seleccionar uno predeterminado, en el área de búsqueda escribimos Typescript.

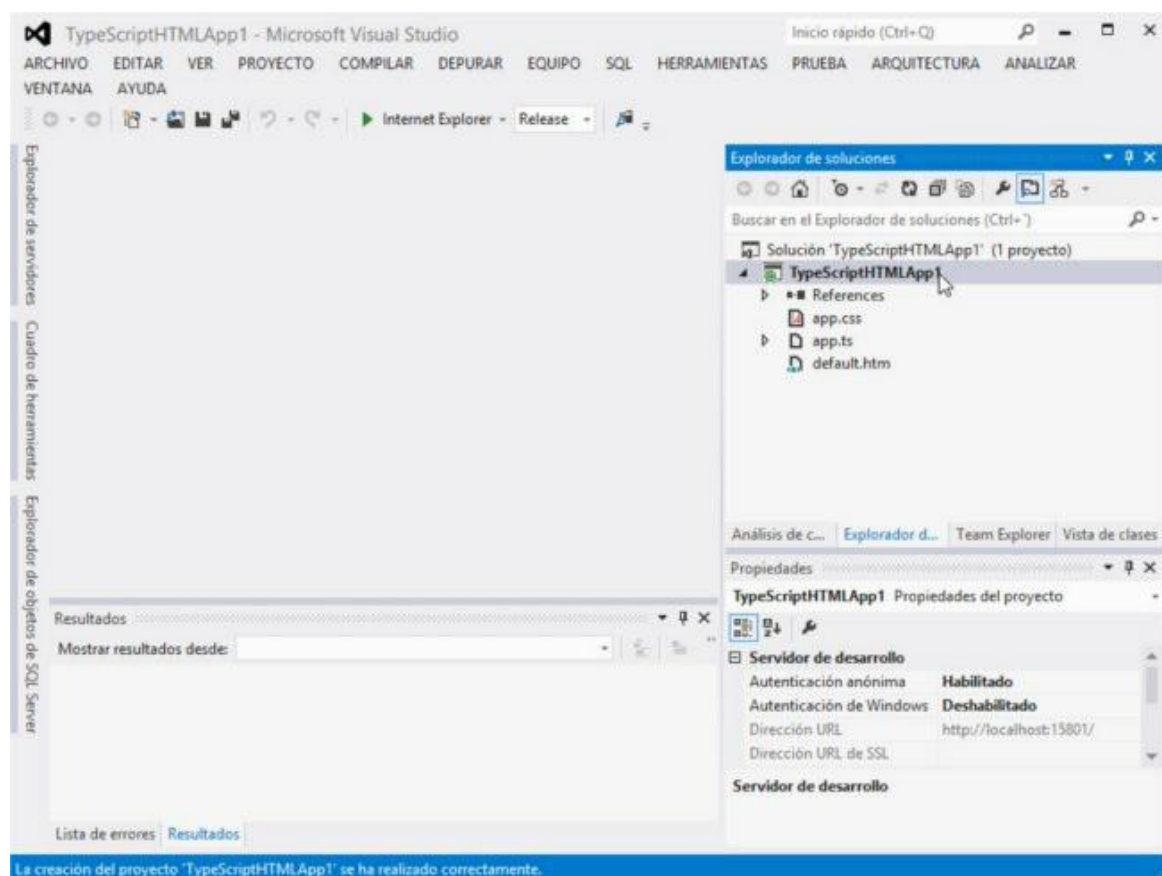


En el área de nuevos proyectos seleccionamos HTML Application with TypeScript –una aplicación HTML estándar, sin más, basada en un documento HTML simple y JavaScript, codificado con TypeScript- le damos el Nombre deseado y hacemos clic en el botón Aceptar.

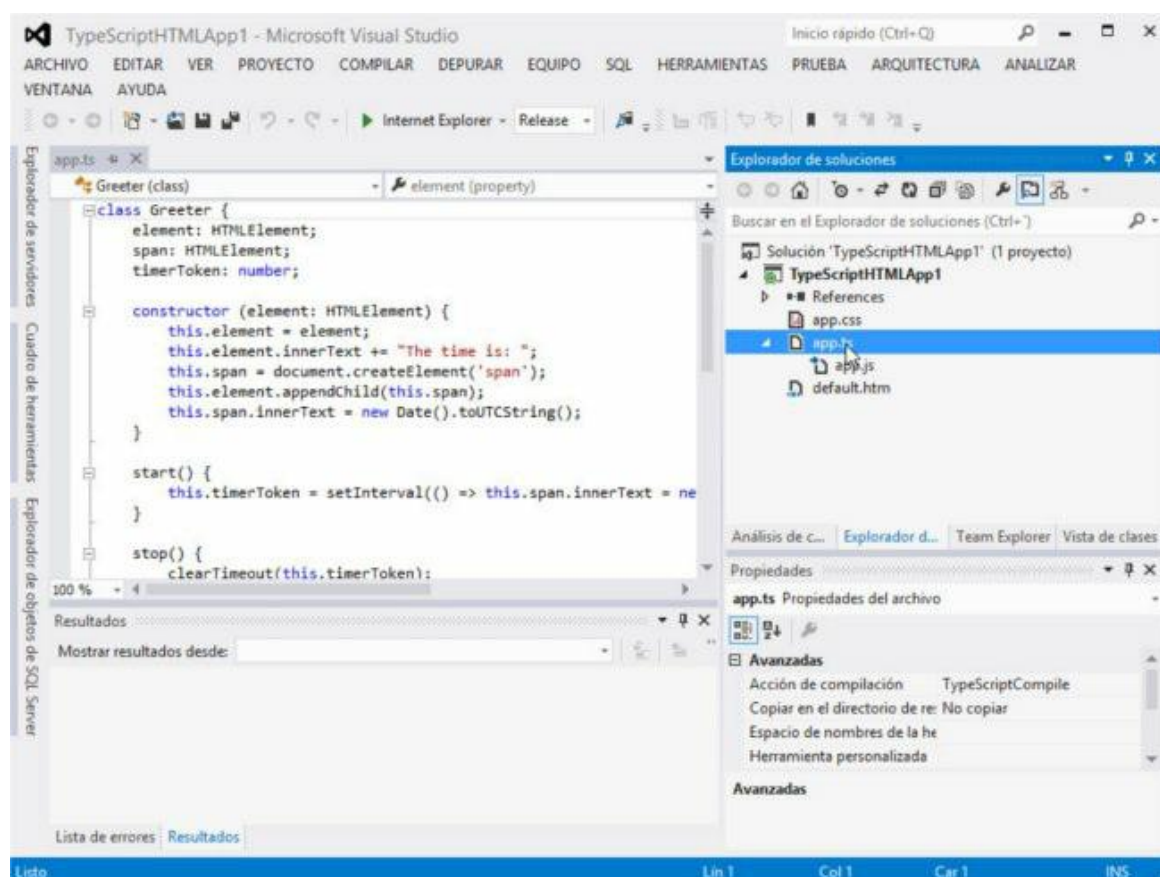


Ya de vuelta en el entorno de trabajo de Visual Studio 2012, nos encontramos que se han creado cuatro archivos que comentaremos seguidamente:

- apps.ts: el archivo TypeScript de trabajo
- apps.js: el equivalente archivo JavaScript, sobre el que no deberíamos trabajar
- apps.css: la hoja de estilos de la aplicación
- default.html: el documento HTML principal de la aplicación



Hacemos doble clic sobre apps.ts para editarlo. A la vez se abre el árbol que nos deja ver el archivo apps.js equivalente de la solución –también podríamos trabajar sobre él, pero romperíamos la secuenciación de compilación del proceso; no deberemos hacerlo–.



Modificaremos el archivo `apps.ts` según nuestros intereses. En nuestro ejemplo será para adecuarlo a nuestro idioma. Inicialmente codifica una secuencia de comandos para mostrar la fecha actual en formato inglés UTC.

Seleccionamos todo su contenido y los sustituimos por el listado siguiente:

```
class Greeter {
  element: HTMLElement;
  span: HTMLElement;
  timerToken: number;

  constructor (element: HTMLElement) {
    this.element = element;
    this.element.innerText += "Ahora estamos a: ";
    this.span = document.createElement('span');
    this.element.appendChild(this.span);
    this.span.innerText = new Date().toLocaleDateString();
  }

  start() {
    this.timerToken = setInterval(() => this.span.innerText = new Date().toLocaleDateString(), 500);
  }

  stop() {
    clearTimeout(this.timerToken);
  }
}
```



```
}

window.onload = () => {
  var el = document.getElementById('content');
  var greeter = new Greeter(el);
  greeter.start();
};
```

De igual manera, en el archivo default.html seleccionamos su contenido y lo sustituimos por el listado de más abajo:

```
<!DOCTYPE html>

<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta charset="utf-8" />
  <title>TypeScript HTML Aplicación</title>
  <link rel="stylesheet" href="app.css" type="text/css" />
  <script src="app.js"></script>
</head>
<body>
  <h1>TypeScript HTML Aplicación</h1>

  <div id="content"/>
</body>
</html>
```

Por último, en el archivo app.css podremos hacer nuestras modificaciones pertinentes, por ejemplo, fijar un color de fondo del documento:

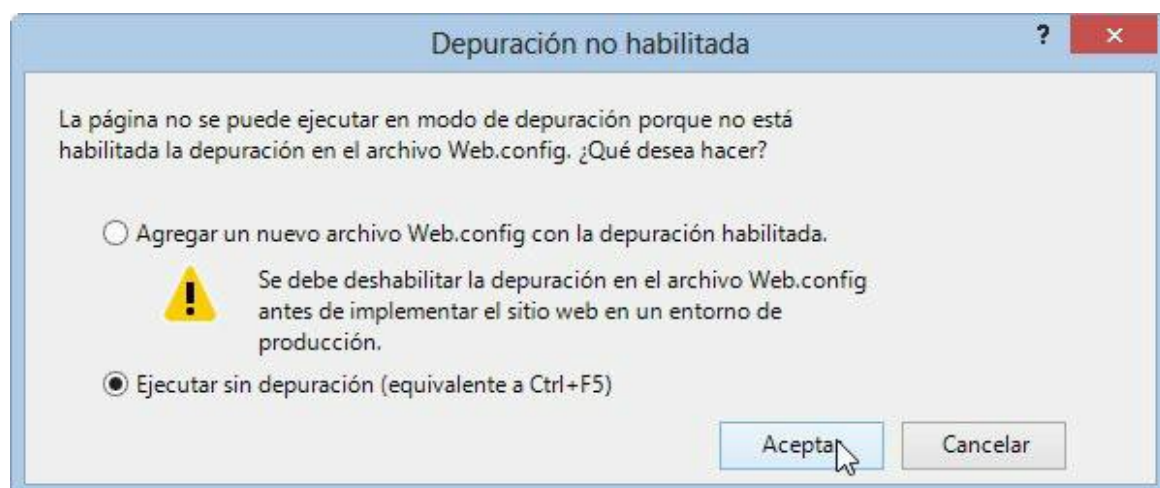
```
body
{
  font-family: 'Segoe UI', sans-serif;
  background-color: yellow;
}

span {
  font-style: italic;
}
```

Finalmente, hacemos clic sobre el botón de Internet Explorer —o del navegador que tengamos instalado por defecto— en la barra de herramientas, o bien pulsamos la tecla F5.

En la caja de diálogo que se nos abre, "Depuración no habilitada", podremos seleccionar:

- Agregar un nuevo archivo Web.config y realizar una depuración
- Ejecutar sin depuración, tal como si hubiéramos pulsado la combinación de teclas Ctrl+F5
- Tomada la decisión, terminaremos haciendo clic en "Aceptar" y se nos mostrará nuestro resultado en el navegador de Internet por defecto que hayamos seleccionado.



Este artículo es obra de *Jaime Peña Tresancos*
Fue publicado por primera vez en 23/11/2012
Disponible online en <http://desarrolloweb.com/articulos/microsoft-typescript.html>

Configurar un proyecto para usar TypeScript

Cómo configurar un proyecto a desarrollar en el que quieras usar TypeScript como lenguaje, mediante el watch que permita compilar automáticamente los archivos a Javascript.

TypeScript es un superset de Javascript, un lenguaje que añade un montón de posibilidades a Javascript, que seguramente muchos desarrolladores agradecerán. Como ya hemos hablado de TypeScript en diversos artículos anteriormente, no vamos a ahondar en los conceptos, sino que iremos directamente a la práctica, explicando qué hacer si quieres comenzar a usar TypeScript. No obstante, si quieres saber más sobre este lenguaje, te recomendamos la lectura del artículo [Introducción a TypeScript](#).

Supongamos que deseas comenzar un proyecto y quieres usar TypeScript como lenguaje, para beneficiarte de sus ventajas, o simplemente para practicar y ponerte al día. Entonces tendrás que realizar una serie de pasos esenciales, que te permitan compilar el código de tu aplicación a Javascript de toda la vida. Esta es una tarea fundamental, porque los navegadores (o NodeJS) no entienden TypeScript y necesitas proveer código Javascript estándar.

Además, cada vez que se cambia el código de un programa, no quieres estar lanzando comandos por consola para compilarlo a Javascript, sino que quieres que esa traducción se realice de manera automática. Para ello necesitas un "watcher", que no es más que un vigilante que detecta cambios en el código fuente, desatando los procesos de compilación de los archivos para pasar a Javascript. Todo esto es lo que te vamos a enseñar en este artículo.



Nota: el proceso de compilación o traducción de TypeScript a Javascript se conoce habitualmente como transpilación, que es una jerga habitual de los desarrolladores, a mitad de camino entre traducción y compilación.

Archivo tsconfig.json

El archivo `tsconfig.json` es el que indica en un proyecto que se está trabajando con TypeScript. Lo colocas en la raíz de carpetas del proyecto y en él situamos un JSON con todas las configuraciones de trabajo para el transpilador de TypeScript.

El archivo `tsconfig.json` puede tener decenas de configuraciones, útiles para cada tipo de desarrollador o tipo de proyecto, pero una versión muy elemental podría ser la que nos encontramos aquí:

```
{
  "compilerOptions": {
    "target": "es5",
    "outDir": "dist",
    "rootDir": "src"
  }
}
```

Básicamente le estamos diciendo:

- **target:** indicamos que queremos que compile a código Javascript escrito con el estándar ES5, que es el que todos los navegadores comprenden, incluso los Internet Explorer más antiguos.
- **outDir:** indicamos el directorio donde se van a colocar los archivos Javascript, ".js", una vez transpilados.
- **rootDir:** indica dónde están los archivos fuente, con el código TypeScript, ".ts", que debe ser traducido.

Lanzar el compilador TypeScript con el watcher

Como hemos dicho, necesitamos el watcher para evitarnos el trabajo manual de lanzar la compilación de los archivos fuente, cada vez que éstos se modifican. Esta tarea la puede realizar el compilador de TypeScript, sin necesidad de ninguna herramienta adicional.

Obviamente, necesitamos tener instalado el compilador en nuestro sistema y aunque ya explicamos esto previamente en la Introducción a TypeScript, se hace vía npm con el comando:

```
npm install -g typescript
```

Luego nos tenemos que situar con el terminal en el directorio raíz del proyecto, donde está el `tsconfig.json`, y lanzamos el comando que activa el watcher.

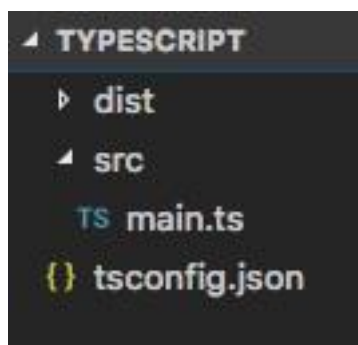
```
tsc -w
```

Nota: Si en este momento no teníamos ningún archivo a traducir, nos mostrará un error con algo como "No inputs were found..."

Ahora no tenemos más que editar los archivos TypeScript y ver cómo se compilan desde la consola. Es importante insistir en que los archivos TypeScript deben estar en el directorio configurado en "rootDir" y que se crearán los archivos Javascript transpilados en el directorio "outDir". Tal como lo tengas configurado en el tsconfig.json.

Resumen del proceso de trabajo con TypeScript

Ahora veremos un ejemplo del proceso, dado el código de tsconfig.json que hemos indicado al principio de este artículo. Partimos de un esquema de carpetas y de directorios como el que sigue:



En nuestro archivo main.js podríamos haber colocado un código como el que sigue:

```
function potencia(base: number, potencia: number) {  
    return Math.pow(base, potencia);  
}  
  
console.log(potencia(2,3));
```

Entonces lanzamos el comando de consola "tsc -w", con lo que obtendremos una salida como la siguiente:

```
mcMiguel-2:typescript midesweb$ tsc -w  
20:10:37 - Compilation complete. Watching for file changes.
```

Ahora puedes probar a realizar algún cambio en el archivo main.ts y verás que el watcher se activa inmediatamente, indicando que ha detectado cambios en los ficheros "File change detected. Starting incremental compilation..."

Hasta aquí todo fue correcto, pero ahora vamos a provocar un error en el código TypeScript, para que veamos cómo el compilador nos advierte. Simplemente vamos a enviarle una cadena a la función potencia(), cuando lo que esa función esperaba era dos números.

Para comenzar, si tu editor soporta TypeScript, como es el caso de Visual Studio Code, ya te advertirá con una marca roja.

```
console.log(potencia("pepe", 3));
```

Nota: si no usas VSCode, que ya viene con todo lo necesario para ofrecerte ayudas en tiempo de programación con TypeScript, seguramente encontrarás un plugin que haga algo similar.

Pero además el compilador, por consola, también se quejará de los cambios cuando grabes el archivo.

```
20:14:19 - File change detected. Starting incremental compilation...  
  
src/main.ts(5,22): error TS2345: Argument of type '"pepe"' is not assignable to parameter of type 'number'.  
20:14:19 - Compilation complete. Watching for file changes.
```

De momento eso es todo, esperamos que esta ayuda te sirva para experimentar con TypeScript y compilar tu propio código de una manera cómoda y automatizada.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en 10/10/2017
Disponible online en <http://desarrolloweb.com/articulos/configurar-proyecto-typescript.html>

Definición de interfaces TypeScript

Qué es una interfaz (interface) y cómo se definen en TypeScript. Ejemplos donde una interfaz ofrece una ayuda a la hora de desarrollar.

Continuamos nuestra [serie de artículos dedicados a TypeScript](#), el superset de Javascript que implementa el tipado estático en este lenguaje (entre otras cosas), para hablar de interfaces.

TypeScript ofrece la posibilidad de trabajar con interfaces, que son un tipo de construcción utilizada en la [Programación Orientada a Objetos](#). Las interfaces nos pueden dar algo más de trabajo a la hora de tirar líneas de código, pero nos ayudan mucho durante todo el tiempo que estamos desarrollando un programa.

En este artículo explicaremos de manera resumida qué es una interfaz ("interface" en inglés), aprenderás a definir una interfaz y podremos ver cómo una interfaz nos facilita la visualización rápida de errores cuando estamos escribiendo código.



Qué es una interfaz

No queremos perder la oportunidad de aclarar el concepto de interfaz, aunque lo cierto es que éste escapa un poco del objetivo del presente artículo.

Técnicamente, las interfaces son un mecanismo de la [programación orientada a objetos](#) que trata de suplir la carencia de herencia múltiple. La mayoría de los lenguajes que implementan la orientación a objetos no ofrecen la posibilidad de definir una clase que extienda varias clases a la vez y sin embargo a veces es deseable. Ahí es donde entran las interfaces.

Una clase puede extender otra clase, heredando sus propiedades y métodos y declarar que implementa cualquier número de interfaces. La diferencia de las clases que extiendes con respecto a las interfaces es que las interfaces no contienen implementación de sus métodos, por lo que la clase que implementa una interfaz debe escribir el código de todos los métodos que contiene. Por este motivo, se dice que las interfaces son como un contrato, en el que se especifica las cosas que debe contener una clase para que pueda implementar una interfaz o cumplir el contrato declarado por esa interfaz.

Ese sería el concepto de manera genérica. Luego cada lenguaje puede tener ligeras diferencias a la hora de aplicar interfaces. Por ejemplo, en TypeScript una interfaz puede definir propiedades, mientras que en otros lenguajes las interfaces sólo definen métodos.

Declarar una interfaz en TypeScript

Las interfaces en TypeScript se declaran de manera bastante similar a la de las clases, indicando la lista de propiedades y métodos que contendrán. Solo hay un detalle fundamental, que las propiedades no pueden tener valores y los métodos no pueden tener código para su implementación.

Aquí podemos ver el código de una interface llamada "sumergibleInterface".

```
interface sumergibleInterface {  
    tiempoMaxBajoElAgua: number;  
    profundidadMaxima: number;  
  
    repelerAgua(): void;  
}
```

Como ves, solo hemos indicado los tipos y los métodos, pero no hemos indicado sus valores. Hemos usado además el tipado de TypeScript, ya que es básico para aprovecharnos de las características del lenguaje.

Implementar una interfaz

A nivel de programación tradicional, lo más típico que puedes hacer con una interfaz es implementarla en una clase.

Una vez definida tu interfaz, podrás implementarla en todas las clases que desees mediante la palabra "implements" en la cabecera de la clase. Para nuestro ejemplo, una vez definida la interfaz sumergibleInterface, todas las clases que vayan a tener objetos que se puedan sumergir, tendrán que implementarla.

```
class relojSumergible implements sumergibleInterface {  
    tiempoMaxBajoElAgua = 1;  
    profundidadMaxima = 10;  
    repelerAgua() {  
        console.log('El agua me resbala');  
    }  
}
```

La interfaz provoca que sea necesario declarar todas las propiedades e implementar todos los métodos a la hora de definir la clase. En resumen, es como un contrato.

Si no se cumple el contrato de la interfaz, entonces nuestro editor se quejará (si está preparado para mostrar los errores de código TypeScript), o el compilador nos lo advertirá.


```
class relojSumergible implements sumergibleInterface {
  tiempo: number;
  profundidad: number;
}

[ts] Class 'relojSumergible' incorrectly implements interface 'sumergibleInterface'.
      Property 'repelerAgua' is missing in type 'relojSumergible'.
class relojSumergible
```

La interfaz como un nuevo tipo

Pero además, TypeScript nos ofrece una aplicación adicional de las interfaces: la creación de un nuevo tipo que podemos usar a lo largo de nuestro código.

```
interface citaCalendario {
  fechaHora: Date;
  titulo: string;
  lugar: string;
}

let cita1: citaCalendario;
```

Como puedes ver, a la hora de crear variables, puedo decir que su tipo es una interfaz declarada. Quizás en un primer momento no le encuentres mucha utilidad, pero realmente es interesante por todas las ayudas que te ofrece el editor, o el compilador de TypeScript, a la hora de programar.

Por ejemplo, esto es lo que pasaría si asignas un valor a la variable "cita1" que no cumpla el contrato de la interfaz, por asignarle cualquier otra cosa que no tenga este tipo.

```
interface citaCalendario {
  fechaHora: Date;
  titulo: string;
  lugar: string;
}

let cita1: citaCalendario;

[ts] Type '"Valor"' is not assignable to type 'citaCalendario'.
let cita1: citaCalendario
cita1 = 'Valor';
```

No solo el editor nos advierte, también el compilador de TypeScript.

```
main.ts(21,1): error TS2322: Type '"Valor"' is not assignable to type 'citaCalendario'.
```

Para que nadie se queje, necesitarás asignar un valor a la variable "cita1" que concuerde con la interfaz, es decir, cumplir el contrato establecido en la declaración del tipo.

```
cita1 = {  
  fechaHora: new Date(Date.now()),  
  titulo: 'Programar en TypeScript',  
  lugar: 'Oficina de DesarrolloWeb.com'  
}
```

Ahora sí tenemos todas las propiedades con los tipos declarados, por lo que el editor ya no nos marcará esta asignación como un error y el compilador podrá hacer su trabajo de transpilación a Javascript.

Conclusión

Como has podido ver, las interfaces no son complicadas de definir. En el mundo de la programación orientada a objetos se usan bastante, pero lo que queríamos mostrar en este artículo es que se pueden usar en TypeScript para producir ayudas en tiempo de desarrollo.

Nos obligan a escribir un poco más de código pero nos ofrecen tipos valiosos, que seremos capaces de usar en nuestras variables. A partir de entonces, cualquier pequeño error a la hora de escribir el código, ya sea porque nos olvidemos de algo, o hagamos un tipeo incorrecto y escribamos mal una propiedad, el editor o el compilador nos lo advertirán, evitando que nos tengamos que volver locos para encontrar errores, que a veces son difíciles de detectar.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en 28/10/2017
Disponible online en <http://desarrolloweb.com/articulos/definicion-interfaces-typescript.html>

Generics en TypeScript

Qué son los genéricos en TypeScript, conocidos habitualmente como generics, una utilidad de TypeScript muy relacionada con su sistema de tipado.

En este artículo vamos a abordar el concepto de generics en el lenguaje TypeScript y ver algunos ejemplos sencillos. Es algo que no existe en Javascript y que por tanto tenemos a nuestra disposición solamente cuando desarrollamos en TypeScript. Los genéricos son una utilidad que disponemos en el lenguaje en tiempo de desarrollo, o de compilación, como las interfaces. Es decir, una vez compilado el código, ya en Javascript, no existe tal construcción y por lo tanto no nos podrán ayudar en tiempo de ejecución.

Los genéricos son construcciones ya disponibles en lenguajes como C# o Java, por lo que posiblemente ya sepas a lo que nos estamos refiriendo. Tengas o no idea, esperamos ayudarte aclarando el concepto, para ver un poco de código para aclarar su sintaxis y uso.



Nota: Recuerda que tienes más sobre TypeScript, aclarando cosas más básicas en el [Manual de TypeScript](#).

Qué son los genéricos

Podemos entender los genéricos como una especie de "plantilla" de código, mediante la cual podemos aplicar un tipo de datos determinado a varios puntos de nuestro código. Sirven para aprovechar código, sin tener que duplicarlo por causa de cambios de tipo y evitando la necesidad de usar el tipo "any".

Así dichas las afirmaciones anteriores, quizás no te digan demasiado. En cambio, la manera más sencilla de entender los genéricos es mediante un ejemplo.

Imagina que tienes una función que muestra un valor numérico y luego lo devuelve. Más tarde te piden hacer esa misma funcionalidad, pero con un string. Para conseguirlo tenemos dos alternativas:

a.- Crear dos funciones con declaraciones de tipos distintas

```
function display(valor: number): number {  
    console.log(valor);  
    return valor;  
}  
  
function displayString(valor: string): string {  
    console.log(valor);  
    return valor;  
}
```

El problema de esta alternativa es obvio, pues hemos repetido prácticamente el mismo código dos veces. Incluso aunque en una clase se aplicase sobrecarga, el código sería prácticamente el mismo.

b.- Usar una única función en la que recibimos y devolvemos el tipo any

```
function display(valor: any): any {  
    console.log(valor);  
    return valor;  
}
```

Ahora el problema es que "any" no nos ayuda para nada en TypeScript. No nos advierte si nos equivocamos asignando otros tipos de datos y el editor deja de ayudarnos con el intellisense.

La solución pasa por aplicar los genéricos

Mediante genéricos podemos indicar que aquello que se recibe tiene un tipo maleable. Es decir, puede admitir diversos tipos, pero sin embargo, dentro de la función podemos referirnos al tipo que se está admitiendo, para que el compilador y el editor nos ayuden en donde puedan.

Sintaxis de los generic de TypeScript

Los genéricos se indican entre "menores y mayores que", como si fueran etiquetas HTML, asignando un alias al tipo variable que se esté recibiendo. Luego podemos usar ese alias para definir el tipo.

```
function display<elTipo>(valor: elTipo): elTipo {  
    console.log(valor);  
    return valor;  
}
```

Aquí estamos definiendo el genérico "elTipo". Esto nos permite que el tipo de parámetro se indique con el alias, así como el tipo de valor devuelto. En la práctica hará que el tipo del valor devuelto se pueda deducir del tipo del valor recibido.

Gracias a esta situación nos conseguimos librar de la declaración de tipo "any", que no ayudaba para nada, y

sustituirla por un generic. Aunque es verdad que no ayudan igual que si fuera un tipo escrito a fuego, sí representan algunas ayudas interesantes como podemos ver en la siguiente imagen:

```
function display<tipo>(valor: tipo): tipo {
  console.log(valor);
  return valor;
}
let cadena = 'test de generics';
let dato: string
let dato = display(cadena);
```

Como ves, aquí el editor es capaz de inferir el tipo de la variable "dato", ya que lo que devuelve la función display() es del mismo tipo del valor que recibió.

Por ejemplo, aquí el editor nos muestra un error por intentar asignar en una variable un dato de otro tipo.

```
function display<tipo>(valor: tipo): tipo {
  console.log(valor);
  return valor;
}
let cadena = 'test de generics';
let dato: number;
[ts] Type 'string' is not assignable to type 'number'.
let dato: number
dato = display(cadena);
```

Un genérico puede ser cualquier tipo, incluso una clase o un array

Cuando usamos un genérico podemos entregar cualquier tipo, incluso una clase de programación orientada a objetos.

```
class Animal {
}

function cuidar<T>(algo: T): T {
  return algo;
}

let algo = cuidar(new Animal());
```

La variable "algo" en la última línea de código, quedaría inferida como de tipo Animal, ya que el parámetro enviado era un objeto de esa clase.

```
function display<tipo>(valor: tipo): tipo {  
  console.log(valor);  
  return valor;  
}  
let cadena = 'test de generics';  
  
let dato: string  
let dato = display(cadena);
```

Existen diversas utilidades en generics que nos ayudan a definir el tipo con un rango más acotado.

Otro ejemplo interesante es el siguiente, en el que definimos que el parámetro será un array de elementos del tipo genérico.

```
function display<T>(valor: T[]): T[] {  
  console.log(valor.length);  
  return valor;  
}
```

Como se ha declarado el parámetro como un array de elementos "T", es seguro que dentro de nuestra función podamos acceder a la propiedad "length" de ese parámetro.

No existe traducción a Javascript de un genérico

Los generics no tienen traducción a código Javascript, ya que Javascript no admite tipos. Como habíamos dicho, los genéricos solo te ayudan en tiempo de desarrollo.

Para observar este detalle, es interesante ver la traducción que el compilador de TypeScript hace de una función donde se usan genéricos. Por ejemplo:

```
function display<T>(valor: T): T {  
  console.log(valor);  
  return valor;  
}
```

Se traduce a Javascript por este otro código:

```
function display(valor) {  
  console.log(valor);  
  return valor;  
}
```

Dos genéricos en una cabecera de función

En la cabecera de una función también podemos definir dos tipos genéricos, separando por comas, tal como se puede ver en el siguiente código.

```
function prueba<T, K extends keyof T>(obj: T, key: K) {  
    return obj[key];  
}
```

Aquí hemos indicado que tendremos dos parámetros. El primero es genérico y podría ser de cualquier tipo, mientras que el segundo deberá ser un valor que se encuentre como llave (key) del primer parámetro. Por tanto, el primer parámetro tendría que ser un objeto y el segundo parámetro uno de las cadenas que sirven como llaves dentro de ese objeto.

```
let profesional = {  
    nombre: 'Miguel A A',  
    empresa: 'DesarrolloWeb.com'  
}  
  
console.log(prueba(profesional, 'empresa')); // Perfecta invocación  
console.log(prueba(profesional, 'propiedadInexistente')); // invocación incorrecta, porque la propiedad no pertenece al objeto
```

La última línea dará un error en compilación, dado que no existe la "propiedadInexistente" como propiedad del objeto recibido como primer parámetro.

```
let profesional = {  
    nombre: 'Miguel A A',  
    empresa: 'DesarrolloWeb.com'  
}  
  
console.log(prueba(profesional, 'empresa')); // Perfecta invocación  
console.log(prueba(profesional, 'propiedadInexistente')); // invocación incorrecta, porque la propiedad no pertenece al objeto
```

Clases genéricas

El conjunto de recursos disponibles con genéricos aumenta cuando los usamos con clases. Así podemos tener una clase en la que declaremos el uso de un tipo genérico.

```
class Generica<T> {  
    hazAlgo(x: T, y: T): T {  
        return x;  
    }  
}
```


Como puedes ver, ese genérico lo puedes usar a lo largo del código de la clase, por ejemplo, el método `hazAlgo()` recibe dos genéricos y luego devuelve otro genérico.

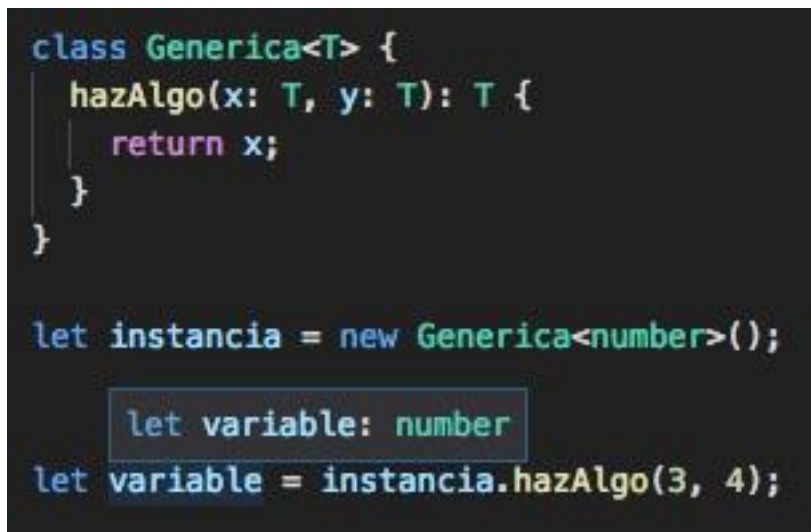
En el momento de instanciar un objeto de esta clase, podemos indicar el tipo concreto de datos que se aplicará a ese genérico.

```
let instancia = new Generica<number>();
```

Ahora, usando los métodos de la clase se podrá inferir el tipo de datos que reciben los métodos con parámetros genéricos o el tipo de datos genérico devuelto.

```
let variable = instancia.hazAlgo(3, 4);
```

Aquí, "variable" será deducida como de tipo "number", tal como puedes apreciar en la siguiente imagen.



```
class Generica<T> {  
  hazAlgo(x: T, y: T): T {  
    return x;  
  }  
}  
  
let instancia = new Generica<number>();  
  
let variable: number  
let variable = instancia.hazAlgo(3, 4);
```

Solo que esto nos plantea un problema y es que en el método `hazAlgo()`, tal como está definido ahí, es imposible saber de qué tipo son los parámetros en la implementación del método `hazAlgo()`, por lo que intentar hacer operaciones con ellos puede derivar en errores en tiempo de compilación.

Así que se puede tomar una alternativa de declarar los métodos con sus tipos genéricos y realizar la implementación más adelante. Es más o menos lo que puedes apreciar en el siguiente código.

```
class Generica<T> {  
  suma: (x: T, y: T) => T;  
}  
  
let instancia = new Generica<number>();  
instancia.suma = function(x, y) {  
  return x + y;  
}  
  
let instanciaString = new Generica<string>();
```

```
instanciaString.suma = function(x, y) {  
    return x + y;  
}
```

Luego podríamos usar estos métodos para producir salida, enviando parámetros de los tipos adecuados, porque si no el compilador se quejará.

```
let variable = instancia.suma(3, 4);  
console.log(variable);  
  
let variableString = instanciaString.suma("Hola", " DesarrolloWeb.com");  
console.log(variableString);
```

Nota: Quizás un código más claro saldría haciendo a la class Generica como abstracta, porque así puede tener el método suma declarado como abstracto. Luego hacer clases derivadas que implementen ese método abstracto, para poder instanciarse objetos, ya indicando los tipos concretos. Quizás el problema con las abstractas te puede llegar porque no podrías instanciar objetos si fuera necesario, mientras que en este código usando generics sí podrías.

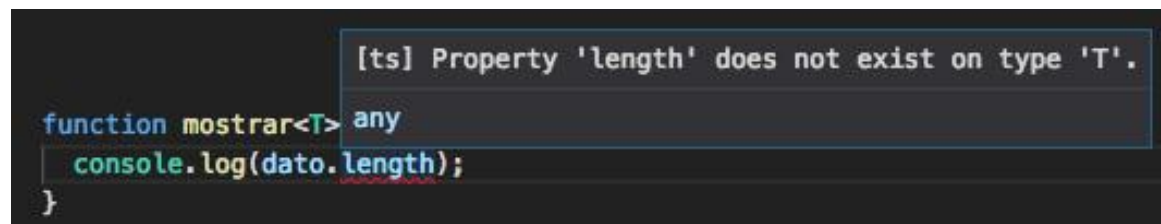
Añadir restricciones a los genéricos

El problema que hemos empezado a observar en el ejemplo anterior puede ser resuelto añadiendo una restricción a los genéricos.

Vamos a observar este código:

```
function mostrar<T>(dato: T) {  
    console.log(dato.length);  
}
```

Si mi genérico recibido como parámetro fuera una cadena, o un array, podría estar seguro que en el cuerpo de la función se puede acceder a su propiedad length. Sin embargo, el genérico te puede aceptar cualquier cosa, por lo que el compilador se queja porque cualquier cosa no necesariamente tendrá la propiedad length.



```
function mostrar<T> any  
    console.log(dato.length);  
}
```

[ts] Property 'length' does not exist on type 'T'.

recuerda que el genérico no es un tipo "any". Si fuera declarado como "any" el parámetro el compilador no se quejaría. La diferencia es que el genérico, aunque técnicamente pueda ser cualquier cosa, el compilador se debe preocupar porque todo lo que reciba, sea lo que sea, pueda realizar las operaciones que se marcan en la implementación de la función.

Una alternativa para solucionarlo es restringir el genérico por medio de una interfaz. Luego, podemos decir en la función que el genérico puede ser cualquier cosa que adopte esa interfaz.

```
interface conLength {  
  length: number;  
}  
  
function mostrar<T extends conLength>(dato: T) {  
  console.log(dato.length);  
}
```

Ahora, si intento hacer algo como esto:

```
mostrar(3);
```

El compilador se quejará porque el "number" 3 no es algo que disponga de la propiedad length. "[ts] Argument of type '3' is not assignable to parameter of type 'conLength'."

Sin embargo, si le paso una cadena, no habrá ningún problema.

```
mostrar("test");
```

También podremos pasar un objeto cualquiera, con tal que tenga la propiedad length:

```
mostrar({ length: 3, otraCosa: 'test' });
```

Sin embargo, si el objeto tiene la propiedad length pero no es de tipo number, el compilador te arrojará un error.

```
mostrar({ length: "no es number", otraCosa: 'test' });
```

Restringir con otro genérico

Ya para terminar, vemos algo todavía más raro, que es restringir el tipo a partir de otro genérico.

```
function restringir<T extends U, U>(param1: T, param2: U): U {  
  return param2;  
}
```

En este caso estás diciendo que lo que envíes como dato a "param1" debe tener, al menos, las mismas propiedades (y mismos tipos) que se encuentran en el tipo de "param2".

Por tanto, esto sería correcto:

```
let obj1 = {  
  a: 1,  
  b: 2  
}  
let obj2 = {  
  a: 1  
}  
restringir(obj1, obj2);
```

Pero esto otro no sería, ya que el objeto obj2 contiene propiedades que no están en obj1.

```
let obj1 = {  
  a: 1,  
  b: 2  
}  
let obj2 = {  
  noSeEncuentra: 1  
}  
restringir(obj1, obj2);
```

Tampoco sería válido este código, porque, a pesar de contener las mismas propiedades, los tipos de obj1 no son los mismos de los tipos de obj2.

```
let obj1 = {  
  a: 1,  
  b: 2  
}  
let obj2 = {  
  a: "55",  
  b: "4"  
}  
restringir(obj1, obj2);
```

Hasta aquí hemos visto muchas posibilidades de los genéricos en TypeScript, muchas de bastante utilidad, aunque algunas otras sin duda bastante extrañas. Esperamos que, aunque no hubieras oído hablar de tipos generics, con esta ayuda hayas despejado tus dudas.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en 06/11/2017
Disponible online en <http://desarrolloweb.com/articulos/generics-typescript.html>