

概要

AltJS としての言語 TypeScript の特徴をかいつまんで説明します。

対象者

- JavaScript でコードを書いたことがある。
 - JavaScript でプログラムを書くのが辛い or もっと楽したい。
 - TypeScript ってなに？聞いたことはあるけど・・・
- ・・・な人。

TypeScriptの参考資料

- [TypeScript - JavaScript that scales.](https://www.typescriptlang.org/) (公式サイト)

- TypeScriptってどんなもの？ プロ生ちゃんと始めてみよう！ - Build Insider
- Insider.NET > TypeScriptで学ぶJavaScript入門 - @IT
- TypeScriptの型入門 - Qiita
- ワイ「いうても型なんて面倒くさいだけやろ？」 - Qiita

AltJS (Alternative JavaScript) とは

AltJS とは、JavaScript の代わりとなる 言語の総称です。

その言語で書いたものを JavaScript に変換して使用します。

TypeScript の他には、[CoffeeScript](#), [Opal](#)(Ruby から JavaScript に変換するツール)などがあります。

なぜ、そんな面倒くさいことをするのか、JavaScript がある定常以上の規模となると、下記の理由から実装・保守の効率が非常に悪くなります。

- 型の定義がないので、意図しない値が入ることがある。

- null safety でないので、意図しない null や undefined が入ることがある。
- オブジェクト指向言語だが、インターフェースやクラス定義がなく、プロパティ名を間違っている場合でも実行時までエラーにならず、エラーになっても原因の解析に時間がかかることが多い。
- 型やインターフェース、クラス定義がないので、エディタによる入力補完があまり受けられない。

TypeScriptとは

マイクロソフトが開発したオープンソース言語で、一言で言うと、「型定義できるJavaScript」。

他の Alt JavaScript と比べて後発ながら人気が高く、Google の6番目の社内標準言語としても採用されました。

AltJS としては、Coffee Script が牽引してきましたが、最近では TypeScript に取って代わられた感じがあります。

参考：

- * [Githubの調査で、2018年急成長している言語ランキング3位](#)
- * [Google社内の標準言語としてTypeScriptが承認される。n g-conf 2017 - Publickey](#)

CoffeeScript について

- * [CoffeeScriptは本当に駄目なのか？ - Qiita](#)

引用：CoffeeScript が駄目でない事を理解していただけ
たと思う。いつまでも素の JavaScript しか使えない JSer
はそのうちいらなくなることは必至だろう。今すぐ、素
の JavaScript を捨てて、**TypeScript** を使うことをお勧め
する。

はい、、、あ?、え?

AltJS とは全く別なアプローチとしてWebクライアント開発
問題を解消するものとして、WebAssembly というものもあり
ます。これは、JavaScriptでなくC/C++、Rustで書いたコ
ードをバイナリコードに変換し、ブラウザで動かすことが

できる仕様です。

Chrome, FireFox, Edge, Safari など主要なブラウザでサポートされています。・・・がみんな大好き IE はサポートしていませんし、将来も無いでしょう。

将来的には、大規模なWebアプリはこちらに移っていく可能性もあるので、要ウォッチです。

[WebAssemblyの紹介（2018年冬版） - Qiita](#)

特徴としては

- JavaScript のスーパーセット（上方互換）となっている。
 - JavaScriptの最新仕様である、"ES2018" の構文仕様が使える。
- 型定義が使える。
- インターフェース、クラスがつかえる。
- null/undefined safe にできる。
- 型定義ファイルがあれば外部ライブラリも型を利用できる。
- ジェネリックが使える
- エディタによる入力補完が強力。(Visual Studio Code など対応しているエディタ)

上で書いたように、TypeScriptで書いたコードをコンパイルをかけることで JavaScript に変換し、ブラウザ等で利用することになります。

TypeScriptでは、型の指定を矯正するなどの制約を設けることができ、コンパイルをするときに、制約のチェックを行います。そうすることで、JavaScript では実行時にしかわからないバグを未然に防ぐことができるのです。

各特徴について、詳しく説明していきます。

JavaScript のスーパーセット（上方互換）となっている

TypeScript は、JavaScriptの仕様を拡張したものになっているので、JavaScriptで書いたものは、TypeScript としても有効です。

例えば、if や for, case といったステートメントや、`{a: "hoge"}` や `["a", "b"]` といったオブジェクトや配列のリテラルはそのまま使えます。

ただし、TypeScript のコンパイラオプションで、制約をつけている場合、その制約によって、コンパイルでエラーになる場合があります。

例えば、`allowUnusedLabels` を `false` にしていると、未使用の変数があったらコンパイルでエラーになります。(このプロパティはデフォルトで `off` ですが)

ですので、JavaScriptを使ったことがあれば、TypeScript の学習コストは低いです。

また、TypeScript は、JavaScript の最新の仕様(正確には、ECMAScript の仕様)を取り込んでいっているので、その学習にもなります。

さらに、それを ES5 などの古いバージョンの JavaScript に変換するトランスパイラとしても機能もあります。

Interface や Class の定義は、C# や Java のそれに似ているので、それらの言語を知っていれば、さらに学習しやすいでしょう。

型定義が使える

型定義の基本

JavaScript からの拡張として、変数の型定義があります。型定義には、さまざまな仕様があります。

変数に定義した型と割り当てた値の型が違う場合、コンパイルする際にエラーとなり、事前にバグに気づくことができます。

基本的な変数の型定義はこちら。

```
let name: string; // name を文字列型として宣言
name = "ebihara";
name = 0; // エラー: 文字列ではない
```

配列の場合、JavaScriptでは 文字列や数値を混在できますが、基本的には 1 つの型の値が入るとおもいます。TypeScriptではそのような制限ができます。


```
const array: string[] = [];  
array.push("ebihara");  
array.push(1); // エラー：配列の型と合わない
```

関数の引数と戻り値の型

関数の引数と戻り値にも型を指定できます。

戻り値の型を指定することで、関数を利用するときだけでなく、関数内の `return` の値もチェックされます。

```
function getName(id: string): string {  
    // なんか処理  
    return "hoge";  
}  
  
const name = getName("xxx");  
console.log(name.length); // 4  
  
function getAge(id: string): number {  
    // なんか処理  
    return 20;  
}  
  
const age = getAge("xxx");  
console.log(age.length); // エラー： age は number 型で length は
```

```
const age2 = getAge(); // エラー： 定義された引数を渡していない

function getAge2(): number {
    return "hoge"; // エラー： 戻り値の方が異なるのでエラー
}
```

引数が必須でない場合もあります。その場合は、引数名の後ろに ? をつけます。

```
function getName(id?: string): string {
    // id は、string か undefined の可能性がある。
    return "hoge";
}

const name = getName("xxx");
console.log(name.length); // 4
// 型定義の後ろに = で値を渡すと、デフォルト値となる

function getName2(id?: string = "xxx"): string {
    // id に値が渡されない場合、"xxx" が入る。
    return "hoge";
}
```

オブジェクト型

いくつかのプロパティを持ったオブジェクトを変数や関数で扱う場合があります。その場合もプロパティの名前とその型が参照されます。

```
const user: { name: string, age: number } = {  
  name: "ebihara",  
  age: 44,  
};  
console.log(user.name); // ebihara  
console.log(user.aga); // エラー: プロパティ名が間違っている  
console.log(user.age.length); // エラー: age は number 型で len
```

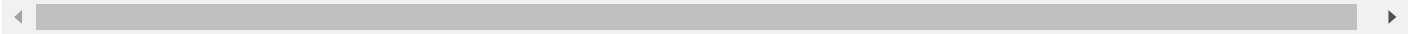
関数型

JavaScript は関数型言語の仕様も一部取り混んでいることもあり、Java や C# と異なり、関数をオブジェクトのように扱うことができます。

つまり、関数だけで独立できたり、関数の引数に関数を与える、オブジェクトのプロパティとして関数を割り当てる、といったことができます。

TypeScriptではその関数の仕様も定義できます。

```
interface IUser {  
    name: string;  
    // 1つの文字列の引数、戻り値が文字列の関数を定義している  
    getName: (keisho: string) => string;  
}  
  
class User implements IUser {  
    name: string;  
    // ここで、引数の型や数が違ったり、戻り値の型が違うと エラーとなる  
    getName (keisho: string) {  
        return `${this.name} (${keisho})`;  
    }  
}  
  
const user = new User();  
user.name = "ebiahra";  
console.log(user.getName("さん"));
```



interface については、後述します。

また、アロー演算子(=>)が出てきていますが、これは、function の別な書き方です。

全く同じか、というところでもないのですが、これについては、こちらを御覧ください。

【JavaScript】アロー関数式を学ぶついでにthisも復習する話 - Qiita

型の推論

変数の宣言時に値を渡すと、型指定をしなくても自動的に型が宣言されたように振る舞います。

```
const age = 10;
console.log(age.length); // エラー: age は number 型となるので、
const user = {
  name: "ebihara",
  age: 44,
};
console.log(user.age.length); // エラー: age は number 型で len
```

関数の戻り値も同様です。

```
function getUser() {  
    return {  
        name: "ebihara",  
        age: 44,  
    };  
}  
  
const user = getUser();  
console.log(user.age.length); // エラー: age は number 型で len
```

union型

JavaScript の場合、変数や引数に複数の型の値が入ることが多いです。

TypeScript ではそれも実現できますが、プロパティを参照するときなど、キャストが必要になる場合があります。

```
let nameOrAge: string | number = "ebihara";  
nameOrAge = 12;  
nameOrAge = true; // エラー: string か number の値のみ入る  
console.log(nameOrAge.toString()); // どちらも持っているメンバーを  
console.log(nameOrAge.length); // エラー: 型が特定されていない  
if (typeof nameOrAge === "string") { // length プロパティを持つて
```

```
console.log(nameOrAge.length); // 自動的に string にキャスト  
}  
console.log((nameOrAge as string).length); // 明示的にキャストし
```

文字列リテラル型

文字列型だけど、決まった値しか入らない、ということも多いかと思います。

例えば、データの状態を示す値などです。これも TypeScript で制御することができます。

```
let status: "create" | "edit" | "view";  
status = "creata"; // エラー: スペルミスで宣言にない文字列が割り当
```

その他

それ以外にも、タプル型、インデックスシグネチャ などまだ多くの機能があります。

こちらのページが詳しいので、参考にしてください。

TypeScriptの型入門 - Qiita

インターフェース、クラスがつかえる

Java や C# などと同じようにインターフェースとクラスの定義ができます。

実は、クラス定義は JavaScript の ES2015 というバージョンで導入されていますが、TypeScriptは更にそれを拡張しています。

```
interface IUser {  
    name: string;  
    age: number;  
}
```

```
class User implements IUser {  
    // インターフェースの実装なので、同じプロパティ定義が必要  
    public name: string;  
    public age: number;  
    // getter setter の指定が可能
```



```
public get displayName() {
    return `${this.name} ${this.keisho} (${this.age})`;
}
// public private のアクセサが使用可能
private keisho = "さん";
// コンストラクタの定義 new したときに呼ばれる
public constructor() {
    this.name = "";
    this.age = 0;
}
}

const user = new User();
user.name = "ebihara";
user.age = 44;
console.log(user.displayName);
user.displayName = "aa"; // エラー: 参照専用のプロパティ
user.keisho = "くん"; // エラー: 公開されていないプロパティ

// インターフェースから直接リテラルでオブジェクトを生成できる
const user2: IUser = {
    // インターフェースで定義されたプロパティが指定されなかったり、型
    name: "ebihara",
    age: 44,
}
```

null/undefined safe にできる

JavaScript を書いていて、"Uncaught TypeError: Cannot read property 'propname' of undefined" というエラーに遭遇したことのない人はいないでしょう。

JavaScript は簡単に undefined が入り込んで、実行時エラーを引き起こします。

参考：[null安全でない言語は、もはやレガシー言語だ - Qiita](#)

それを防ぐために、コンパイラオプション

で、"strictNullChecks" を on にすると、null や undefined である可能性の変数がチェックできるようになります。

具体的には、コードを見てみます。

```
class Sample {  
    public name: string; // エラー： 初期値を定義していないので、  
    public name2: string = ""; // 初期値を指定しているので、OK  
    public name3: string | undefined; // 明示的に undefined のこ
```

```
public name4?: string; // name3 と同じ意味
public name5: string | null; // エラー: null と undefined |
public fn() {
    console.log(this.name3.length); // エラー: undefined の
    if (!!this.name3) {
        console.log(this.name3.length); // undefined の可能
    }
    console.log((this.name3 as string).length); // 明示的に
}
}
```

！重要！ null safety だからといって、null や undefined のエラーが張り込まないわけではありません。サーバーからのデータの受信、ユーザーの入力など外部からの入力があるからです。当然それらの値について、チェックを掛ける必要があります。

型定義ファイルがあれば外部ライブラリも型を利用できる

JavaScript でWebアプリ等を作成する場合は、React や jQuery、moment などのライブラリを利用することが多いと思います。

それらのライブラリの言語は当然、JavaScript で提供されているので、そのライブラリにこういったオブジェクトや関数が用意されているので、関数の引数、戻り値は何なのかは、ライブラリの仕様書を見ないとわかりません。

TypeScript では、ライブラリがTypeScript用の型定義ファイルを提供していることがあり、それがあれば TypeScript でライブラリのAPI情報がわかり、コンパイル時にチェックすることができます。

TypeScript 用の型定義ファイルは、npm リポジトリに `@types/[ライブラリ名]` で提供されています。

例

```
> npm install --save-dev @types/jquery
```

`@types/jquery`
<https://www.npmjs.com>



また、ライブラリに同梱されている場合もあります。

最近では、TypeScript で書いて JavaScript で提供するライブラリも多いことから、同梱している物も増えています。

TypeScript が急速にシェアを伸ばしていることから、ほとんどのメジャーな JavaScript ライブラリに、型定義ファイルが提供されています。

型定義ファイルがない場合は、自分で ".d.ts" を書くことで回避することもできます。大抵は、その中に any 型（型定義されない型）を定義してコンパイルを通す手段となります。その場合、型定義のチェックがされないことになるので、十分に注意してコーディングします。

型定義ファイルの拡張子は、".d.ts" です。

ジェネリックが使える

薬の話・・・ではなく、

ジェネリックとは、クラスや関数で、その中で使う型を抽象化し、外部からその型を指定できるようにすることで、そのふるまいを変えることができるものです。

って言われても、はじめての人には難しいですね。

ジェネリックは、主に汎用的な関数やクラスを定義するときに使われます。

C# や Java ではなじみがある方は判ると思いますが、それと同等の機能です。

例えば、任意の型の配列と、呼び出すたびに配列を順番に取り出していく関数をもつクラスを作ってみます。

ジェネリックを使えば、扱う配列の型を抽象化することで、このクラスを作成する時点で決めることができます。

```
class Iterator<T> { // <T> がジェネリックの宣言 Tは仮の名前
  // クラス内では、T型 として扱う
  private currentIndex: number = 0;
  private array: T[] = []; // 外から与えられる型で配列を定義する
  // 配列に値を登録 引数は、T型 である必要がある
  public push = (value: T) => {
```

```
        this.array.push(value);
    }

    public get = (): T => { // 関数の戻り値の型は T型
        // this.currentIndex が配列の数を超えたときの確認とかは省略
        // this.array は、Tの配列なので、value は T型
        const value = this.array[this.currentIndex];
        this.currentIndex += 1;
        return value;
    };
}
```

```
const numberIterator = new Iterator<number>(); // 数値型を扱うと
numberIterator.push(999);
numberIterator.push(998);
numberIterator.push("ghi"); // number 型ではないのでエラー
```

```
const a = numberIterator.get();
console.log(a.length); // a は number 型でないのでエラー
```

エディタによる入力補完が強力

エディタが TypeScript の入力支援機能をサポートしている場合、型定義から入力候補を表示する事ができます。

このために、TypeScriptを採用していると言ってもいいくらい便利な機能で、開発効率が格段に違います。

TypeScript と同じマイクロソフトが開発している、 Visual Studio Code がおすすめです。

入力補完の例：



まとめ

TypeScript の主だった機能を紹介しました。

ここで説明した以外にも機能はまだあるので、公式ページ等と見てください。

近年、JavaScript は Webアプリケーション のみにかかわらず node.js でのサーバーサイド処理や、Electron でのデスクトップアプリ、React Native を使ったモバイルアプリなど、その活用範囲は広がっています。

TypeScript はそういった中〜大規模になるプロジェクトで活躍する事ができる言語ですので、是非身につけてください。