基本的な構文

TypeScriptを扱ううえで登場する制御構文です。 JavaScriptと変わりませんし、JavaやC++とかともほぼ変わりません。 すでに知っている方は飛ばしても問題ありません。

制御構文

`if`

一番基本的な条件分岐です。JavaやC++を使ったことがあれば一緒です。

- `if (条件) ブロック` です。
- `else if (条件)` を追加することで、最初のケースで外れた場合に追加で条件分岐させることができます。
- `else` をつけると、マッチしなかったケースで処理される節を追加できます。
- ブロックは `{ }` でくくってもいいですし、1つしか文がないなら `{ }` を省略することもできます。

```
if (task === "休憩中") {
  console.log("サーフィンに行く");
} else if (task === "デスマ中") {
  console.log("睡眠時間を確保する");
} else {
  console.log("出勤する");
}
```

なお、昔、よくバグの原因となると有名だった、条件文の中で比較演算子ではなく、間違って代入を書いてしまうことでプログラムの動きがおかしくなってしまう問題ですが、ESLintの推奨設定で有効になる `no-cond-assign` という項目で検出できます。

`switch`

条件文の中の値と、 `case` で設定されている値を `===` 演算子で前から順番に探索し、最初にマッチした節を実行します。一致した値がなく `default` 節があった場合にはそこが実行されます。

```
switch (task) {
case "休憩中":
  console.log("サーフィンに行く");
  break;
case "デスマ中":
  console.log("睡眠時間を確保する");
  break;
default:
  console.log("出勤する");
}
```

`case` の条件が重複している `case` はESLintの推奨設定でも有効になる `no-duplicate-case` オプションで検知できます。 また、 `break` を忘れると、次の `case` が実行されてしまいますが、こちらもESLintの推奨設定で有効になる `no-fallthrough` オプションで検知できます。

`for`

一番使うループ構文です。4通りの書き方があります。

C言語風のループ変数を使う書き方

フラグの数値をインクリメントしながらループする方式です。昔は `var` を変数宣言に使っていましたが、 `let` が推奨です。 `let` の変数は、このforの条件式とブロックの中だけで有効になります。

C言語風のループ変数を使う方式

```
for (let i = 0; i < 5; i++) {
  console.log(i);
}
```

`for..in`

オブジェクトのプロパティを列挙するループです。 プロトタイプまで探索しにいくため、想定外の値がループ変数に代入される可能性があります。 そのため、 `hasOwnProperty()` メソッドを呼んで、想定外の値が入らないようにブロックする書き方が一般的でした。 今では使うことはないでしょう。 次に紹介する `for..of` を使うべきです。

for in

```
for (let key in obj) {
  if (obj.hasOwnProperty(key))
    console.log(key, obj[key]);
  }
}
```

JavaScript時代との違いは、 `let` で定義された変数の範囲です。その `key` の値も含めて、条件文とボディの中以外から見えることはありません。

注釈

配列のループに `for..in` を使うことも不可能ではありませんが、現在使われている各種ブラウザでは、通常の `for` ループと比べて50倍から100倍遅くなります。配列のループの手段として使うのはやめましょう。

`for..of`

`for..in` より新しい記法です。イテレータという各データ構造がループ用に持っている機能を使うため、想定外の値が入ることはありません。

for of

```
for (const value of array) {
  console.log(value);
}
```

`of` の右側には、イテレータプロトコルをサポートする、次のような要素が書けます。

- 配列、 `Map` 、 `Set` 、文字列

また、それ以外に、イテレータを返すメソッドや関数があり、これらの呼び出しを右辺に持ってくることもできます。

- `array.entries()` （配列のインデックスと値がセットで返ってくる）
- `Object.keys(obj)` （オブジェクトのキーが返ってくる）
- `Object.values(obj)` （オブジェクトの値が返ってくる）
- `Object.entries(obj)` （オブジェクトのキーと値が返ってくる）
- `map.keys()` （ `Map` のキーが返ってくる）
- `map.values()` （ `Map` の値が返ってくる）
- `map.entries()` （ `Map` のキーと値が返ってくる）

キーと値の両方が帰ってくるメソッドは、分割代入を用いて変数に入れます。

for of

```
for (const [key, value] of Object.entries(obj)) {
  console.log(key, value);
}
```

なお上記に列挙したものの中では、 `Object.keys()` がES5に入っています。他のものを使うときは、ターゲットバージョンをES2015以上にするか、ターゲットバージョンを低くする代わりにPolyfillを設定する必要があります。

このイテレータは、配列以外にも、配列のような複数の値を含むデータ構造（シーケンス）が共通で備えるインタフェースです。このインタフェースを実装することで、ユーザークラスでも `for..of` ループと一緒に使えるようになります。現在はそれほどではないですが、言語標準であるため、何かしらの最適化が行われる可能性があります。

`for await of`

ES2018で導入されました。ループごとに非同期の待ち処理を入れます。 これに対応するには、asyncIterator に対応した要素を条件文の右辺に持ってくる必要がありますが、現在サポートしているのは `ReadableStream` ぐらいしかありません。 このクラスは、 `fetch()` のレスポンスの `body` プロパティぐらいでしか見かけません。 対応するクラスを自作することもできます。

```
for await (const body of response.body) {
  console.log(body);
}
```

並行して処理を投げる場合は、非同期の章で紹介するように `Promise.all()` を使い、すべてのリクエストはすべて待たずに投げてしまう方が効率的です。 `for await of` は同期的な仕事でのみ利用されることを想定しています。

`while` 、 `do .. while`

条件にあっている限り回り続けるループです。 `while` はブロックに入る前にチェックが入る方式、 `do .. while` はブロックの後でチェックをします。

以前は、無限ループを実現するために `while (true)` と書くこともありましたが、ESLintでは推奨設定で設定される `no-constant-condition` オプションで禁止されます。

`try .. catch`

例外をキャッチする文法です。Javaと違うのは、JavaScriptは型を使って複数の `catch` 節を振り分けることができない、という点です。 `catch` には1つだけ入れ条件文を書きます。 非同期処理が多いJavaScriptでは、例外でうまくエラーを捕まえられることはまれでしたが、ES2017で導入された `async` 関数は非同期処理の中のエラーを例外として投げるため、再びこの文法の利用価値が高まっています。 例外に関しては特別に章を分けて説明します。

```
try {
  // 処理
  throw new Error("例外投げる")
} catch (e) {
  // ここに飛んでくる
  console.log(e);
} finally {
  // エラーがあってもなくてもここにはくる
}
```

式

基本的な演算子などは、他の言語と変わらないので省略します。他の言語ユーザーが困りそうなポイントは次の2つぐらいです。

- 比較の演算子: `===` と `==` がある（それぞれ否定は `!==` と `!=` ）。前者は一致を厳密に見るが、後者は、文字列に変換してから比較する。なお、配列やオブジェクトで厳密な一致（ `===` ）は、インスタンスが同一かどうか、で判定されます。
- `**` 演算子: `x ** y` で `Math.pow(x, y)` と同じ累乗計算を行う

いまどきのウェブフレームワークでコードを書く上で大事な式は2つあり、論理積（ `&&` ）と、三項演算子ですね。それぞれ、 `（条件）&& 真の時の値` 、 `（条件）? 真の時の値 : 偽の時の値` という、条件分岐を1行で書きます。

三項演算子（わかりやすくするためにかっこを入れましたが省略可能です）

```
const result = (day === "金曜日") ? "明日休みなので鳥貴族に行く" : "大人しく帰る";
```

とくに、Reactは1行の一筆書き（1つの `return` 文の中で）で、仮想DOMという巨大なJavaScriptのオブジェクトを生成します。このときに条件分岐のコードとして役に立つのが三項演算子というわけです。

Reactの中の条件分岐

```
render() {
  return (
    <div>
      { this.state.loggedIn ? <p>ようこそ</p> : <p>ログインが必要です</p> }
    </div>
  );
}
```

参考までに、ループは配列の `map` メソッドを使うことが多いです。

Reactの中のループ

```
render() {
  return (
    <ul>
      { this.state.users.map(user => {
        <li>{user.name}</li>
      })
    </ul>
  );
}
```

まとめ

基本的な部分は他の言語、特にC++やJavaといった傾向の言語を使っている人からすればあまり大きな変化に感じないでしょう。

`for` ループだけはいくつか拡張がされたりしていましたので紹介しました。 また、今時のウェブフレームワークで使う、1行のコード内で使える条件分岐とループも紹介しました。