

# オーバーロード関数の話

@Mita.ts #2

# ken7253

Frontend developer

技術記事を書いたりするのが趣味。

最近はReactを使ったアプリケーションを書いています。

ユーザーインターフェイスやブラウザが好き。

 <https://github.com/ken7253>

 <https://zenn.dev/ken7253>

 <https://dairoku-studio.com>



オーバーロード関数使ってますか？

---

# オーバーロード関数 is 何？

関数のシグネチャーを複数宣言できる機能

## シグネチャー

関数名・引数の型・返り値の型をまとめた情報のこと。

- 関数宣言(`function`)のみで利用可能。
- 引数の型に応じて返り値の型を変えることができる。
- `Conditional types` を使わないので記述が分かりやすい。

# オーバーロード関数 is 何？

こういう感じに定義する。

```
// シグネチャーのパターンを定義
export function foo(): void;
export function foo(value: number): number;
export function foo(value: string): Error;

// 全てのシグネチャーを満たすように関数を実装
export function foo(value?: number | string): void | number | Error {
    if (typeof value === 'number') {
        return value * 2;
    } else if (typeof value === 'string') {
        return new TypeError('invalid Type');
    };
    console.log("void");
};
```



何が嬉しいんだ... ?

---

## オーバーロード関数の使いどころ

例として、任意のオブジェクトをマージする処理を実装したとする。

```
function merge(obj1: Obj1, obj2: Obj2): Record<string, unknown>;
function merge(obj1: Obj1 | null, obj2: Obj1 | null): Error | Record<string, unknown>;
function merge(
  obj1: Obj1 | null,
  obj2: Obj1 | null
): Error | Record<string, unknown> {
  // マージ処理を実装
};
```

こうすると関数の利用者側に2つの選択を提示できる。

- 先にオブジェクトの存在チェックを行って安全な状態で渡してもらう。
- とりあえず値を突っ込んで後でエラーハンドリングを行う。

存在チェックを行えば返り値から `Error` 型が除去される。

# 設計時の注意点

---

# 設計時の注意点

---

## 神関数にならないように

- 小さいことをうまくやる
- オーバーロードは最小限で

## 型推論のコストに注意

- オーバーロード関数とパターンマッチでTSが型推論を諦めた

## 理由なく使っていいわけではない

- ジェネリクスや単純なunion型で解決できる場合が多い
- サバイバルTypeScript | オーバーロード以外も検討しよう

