TypeScript入門

目次

1. TypeScriptとは

JavaScriptの文法に型を追加したJavaScriptのスーパーセット

`interface` や `type` など他のプログラミング言語でおなじみの文法が `JavaScript` でも使用できるようになる

2. TypeScriptを使うメリット

- メンテナンス性、保守性の高いコードがかける
- VSCodeなどのエディタの恩恵が受けられる

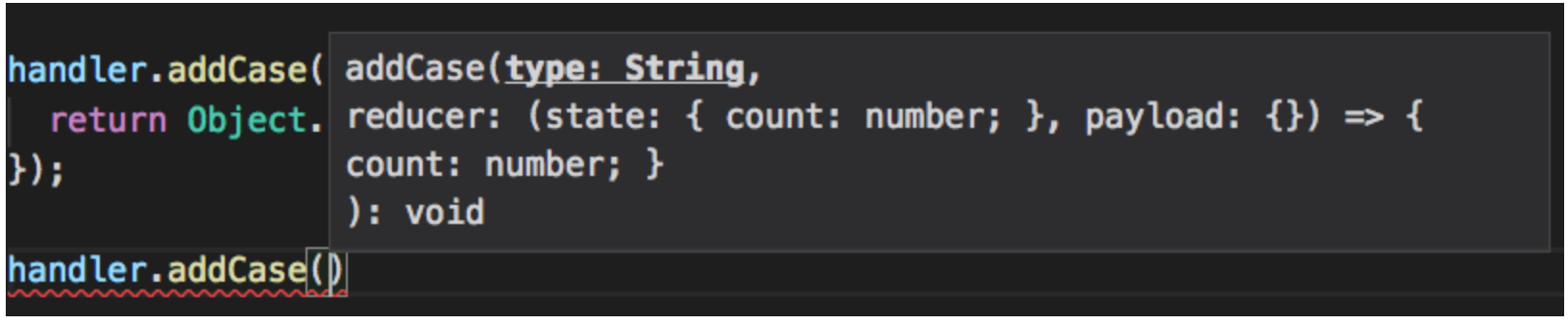
2. TypeScriptを使うメリット

VSCodeなどのエディタの恩恵が受けられる

- メソッドや変数の型情報表示
- プロパティの一括置換
- 変数のエラーチェック
- 型定義している場所に移動

2. TypeScriptを使うメリット

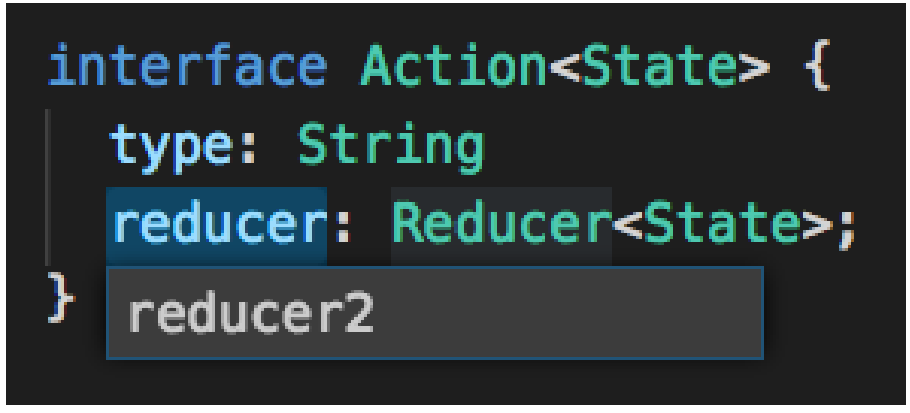
メソッドや変数の型情報表示



2. TypeScriptを使うメリット

VSCodeなどのエディタの恩恵が受けられる

プロパティの一括置換



2. TypeScriptを使うメリット

VSCodeなどのエディタの恩恵が受けられる

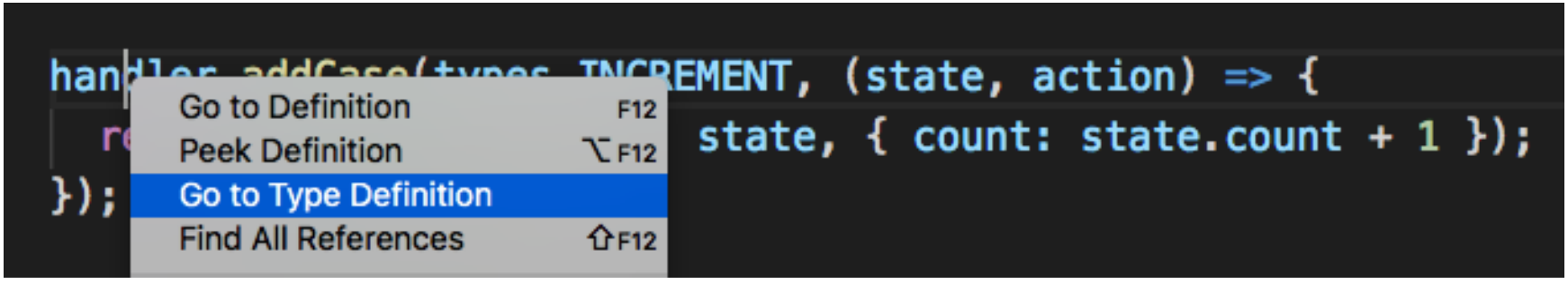
変数のエラーチェック

```
handler.addCase(types.RENAME, (state, action) => {
  return Object.assign({}, state, { rename: action.rename });
});
```

2. TypeScriptを使うメリット

VSCodeなどのエディタの恩恵が受けられる

型定義している場所に移動



3. 文法

type

```
let str: string = 'godai';
str = 3; //error!

const plus = (a: number, b: number) : number => {
  return `${a} ${b}`; //error!
}
```

3. 文法

inteface

```
interface Human {
  age: number,
  name: string
}

const hoge: Human = {
  gender: 'man' //error genderは定義されていない
}
```

3. 文法

Generics

```
class Hoge<State> {
  state: State,
  constructor(state: State) {
    this.state = state;
  }
}

const hoge = new Hoge({ name: 'hoge' });
hoge.age // error!
hoge.name // 'hoge'
```

4. 環境構築

1. TypeScript only
2. TypeScript + Babel7

4. 環境構築

TypeScript only

```
npm install webpack webpack-cli ts-loader --save
```

4. 環境構築

TypeScript only

```
module.exports = {
    resolve: {
      extensions: ['.ts', '.tsx', '.js', '.json']
    },
    module: {
        rules: [
            {
                test: /\.ts|\.tsx$/,
                exclude: /node_modules/,
                use: {
                    loader: "ts-loader"
                }
            }
        ]
    }
};
```

4. 環境構築

TypeScript only

```
{
    "compilerOptions": {
        "sourceMap": true,
        // TSはECMAScript 5に変換
        "target": "es5",
        // TSのモジュールはES Modulesとして出力
        "module": "es2015",
        // JSXの書式を有効に設定
        "jsx": "react",
        "moduleResolution": "node",
        "lib": [
        "es2018",
        "dom"
        ]
    }
}
```

4. 環境構築

TypeScript + Babel7

https://github.com/Microsoft/TypeScript-Babel-Starter

```
npm install --save-dev typescript
npm install --save-dev @babel/core
npm install --save-dev @babel/cli
npm install --save-dev @babel/preset-stage-3
npm install --save-dev @babel/preset-env
npm install --save-dev @babel/preset-typescript
```

4. 環境構築

TypeScript + Babel7

```
tsc
--init
--declaration
--allowSyntheticDefaultImports
--target esnext
--outDir lib
```

4. 環境構築

TypeScript + Babel7

```
{
  "presets": [
    ["@babel/env", {
      "targets": {
        "browsers": ["last 2 versions"]
      }
    }],
    "@babel/stage-3",
    "@babel/react",
    "@babel/typescript"
  ]
}
```

4. 環境構築

TypeScript + Babel7

```
module.exports = {
    entry: './src/index',
    output: {
        path: path.resolve(__dirname, 'dist'),
        filename: 'app.bundle.js'
    },
    resolve: {
        extensions: ['.ts', '.tsx', '.js', '.json']
    },
    module: {
        rules: [{
            test: /\.(tsx?)|(js)$/,
            exclude: /node_modules/,
            loader: 'babel-loader',
        }],
    }
};
```

5. 最新のTypeScript事情

3.0の最新機能

rest paramsの型定義

```
function foo(...args: [number, string, boolean]): void;
```

JSXでのdefaultProps対応

```
export interface Props {
    name: string;
}

export class Greet extends React.Component<Props> {
    render() {
        const { name } = this.props;
        return <div>Hello ${name.toUpperCase()}!</div>;
    }
    static defaultProps = { name: "world"};
}
```

5. 最新のTypeScript事情

もっと気軽にTypeScriptを導入しよう！

1. すべての変数に型をつける必要はない。（型推論機能もある！）
2. npm経由で型情報もインストールできる
3. Reactともすごく相性がいい

6. ライブラリとして配布する際のコツ

TypeScriptのプロジェクトをnpmとして配布するときには`package.json`に定義ファイルへのパスを記述する。`types`

```
{
  "name": "redux-action-handler",
  "version": "0.0.3",
  "description": "handle actions without using switch statement
  "main": "./lib/index.js",
  "types": "./index.d.ts",
  ...
```

6. ライブラリとして配布する際のコツ

index.d.tsの設置

```
declare module 'redux-action-handler' {
  interface Payload {
    [x: string]: any
  }
  interface Action<State> {
    type: string;
    reducer: Reducer<State>;
  }
  type Reducer<State> = (state: State, payload: Payload) => Sta
  export default class ActionHandler<State> {
    private initialState;
    private actions;
    constructor(initialState: State);
    addCase(type: string, reducer: Reducer<State>): void;
    create(): (state: State, action: Action<State>) => State;
  }
}
```

6. ライブラリとして配布する際のコツ

サンプル

https://github.com/steelydylan/redux-action-handler

ありがとうございました！