関数型指向のプログラミング

JavaScriptの世界には長らく、クラスはありませんでした。プロトタイプを使ったクラスのようなものはありましたが、Javaなどに慣れた人からは不満を持たれていました。 プロトタイプが分かっていれば、インスタンス作成能力には問題はなく、親を継承したオブジェクトも作れるため、能力が劣っていたわけではありませんでしたが、オブジェクト指向としては使いにくい。 そんなふうに言われることも多々ありました。

一方で昔からJavaScriptで関数型プログラミングを行おう、という一派はそれなりにいました。

JavaScriptの出自からして、Schemeという関数型言語が好きだったブレンダン・アイクです。 開発時の会社の方針でJava風の文法を備え、Javaに影響を受けた言語にはなっていますが、Javaのような静的型付けのオブジェクト指向のコンパイル言語とはやや遠い、プロトタイプ指向のインタプリタになっています。 関数もオブジェクトとして、変数に入れたり自由に呼び出したりできる一級関数ですし、無名関数を気軽に作ったり、その関数が作られた場所の変数を束縛するクロージャとしても使えたり、関数型言語の要素も数多く備えています。

おそらく、jQueryも関数型の思想で作られたのではないかと思います。 オライリーからも、2013年に『JavaScriptで学ぶ関数型プログラミング』の原著が出ています。

関数型の言語を触ったことがない人も、特に怖がる必要はありません。 JavaScriptで実現できる関数型プログラミングのテクニックはかなり限られています。ループを再帰で書くと、すぐにスタックを使い果たしてエラーになります。 関数型でコードを書くと言っても、あまり極端なことはできずに、オブジェクト指向と組み合わせたハイブリッドな実装方法になりますし、バグを産みにくいコードを書くための指針集といった趣になります。

イミュータブル

イミュータブル（immutable）は「変更できない」という意味です。 データを加工するのではなく、元のデータはそのままに、複製しつつ変化させたバージョンを作ります。 ここで活躍するのが、配列のメソッドの、 `map()` 、 `forEach()` 、 `filter()` です。

たとえば、1から10までの数値が入った配列があったとして、それぞれの値を2倍にした配列を作ります。

```
// 元のデータ
const source = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
// 2倍の要素を作る
const doubles = source.map(value => {
  return value * 2;
});
```

変換処理の関数を書き、`map()`に渡すと、要素1つずつをその関数にわたし、その結果の配列を新たに作って返します。

- `map()`: 元のデータの各要素に何かしらの加工を行った新しい配列を作る
- `forEach()`: 元のデータの各要素に対するループを行う
- `fileter`: 元のデータの書く要素を評価し、選択されたものだけを保持する新しい配列を作る

これは配列の例ですが、複合型の場合に値そのものを書き換えるのではなく、書き換えた別のバージョンを作っていくのがポイントです。 オブジェクト指向はデータと操作が結びついていますが、関数型はデータとそれに対する操作を切り離すと共に、データを複製するという異なったアプローチを取ります。 実行効率的にもったいないのでは、と思われる方もいると思いますが、JavaScriptエンジン開発競争のおかげで、JavaScriptエンジンは動的言語の中ではトップクラスのパフォーマンスは持ちます。そこそこ大きなデータであっても高速に処理ができます。多少ペナルティがあったとしても、管理のしやすさや安全を重視してこの方法を取ります。