Array.prototype.reduce Dance

Basic

var assert = require("assert");

- forと違いimmutableなオブジェクトを使わないで合計を出せる
-

```
var total = [1, 2, 3, 4, 5].reduce(function (a, b) {
    return a + b;
});
assert.equal(total, 15);
```

初期値

- 初期値を指定することもできる

```
var initialTotal = [0, 1, 2, 3, 4].reduce(function (a, b) {
    return a + b;
}, 10);
assert.equal(initialTotal, 20);
```

例外

- 空の配列の場合は例外を吐く
- 15.4.4.21 Array.prototype.reduce
  - Step 8.c

```
assert.throws(
    function () {
        [].reduce(function (a, b) {
            return a + b
        });
    },
    TypeError,
    "k < len の場合はTypeError"
);
```

every by reduce

```
var assert = require("assert");
assert.ng = function(value, message){
    assert.equal(false, !!value, message);
};
```

- reduce があれば、補完mapやeveryなどは実装できる
- 柔軟性が高い

このうちreduceが一番強力で、mapやfilterやsumなど、他の関数もこれをもとに定義できます

via Functional JavaScript

Array.every

```
function every(array, predicate) {
    return array.reduce(function (prev, current, index, list) {
        if (prev) {
            return predicate(current, index, list);
        } else {
            return prev;
        }
    }, true);
}
function isBigEnough(element, index, list) {
    return (element >= 10);
}
assert.ok(every([12, 130, 44], isBigEnough));
assert.ng(every([1, 100, 200], isBigEnough));
```

map

var _ = require("underscore");

- map は基本的に入力の個数と出力の個数が同じ
- map だけだと filter (select) までは行えない

```
var filteredNumberToString = function (e) {
    if (typeof e === "number") {
        return String(e);
    }
};
var mappedArray = [1, null, 3].map(filteredNumberToString);
// 現実はundefinedになる
assert.deepEqual(mappedArray, ["1", undefined, "3"]);
```

flatMap

- reduce なら違う形(入力と出力の個数が異なる)ものを返せる

```
function flatMap(obj, iterator) {
    return _.reduce(obj, function (memo, value, index, list) {
        var items = iterator(value, index, list);
        if (items == null) {
            return memo;
        }
        return memo.concat(items);
    }, []);
}
var flatMappedArray = flatMap(["string", 1, null, 3], filteredNumberToString);
// Numberだけにfilter + NumberをStringに変換した結果を返す
assert.deepEqual(flatMappedArray, ["1", "3"]);
```

高階関数

- 関数を返す関数の事

```
var ComparisonResult = {
    ascending: -1,// <
    same: 0,       // ==
    descending: 1 // >
};
function comparator(predicate) {
    return function (x, y) {
        if (predicate(x, y)) {
            return ComparisonResult.ascending
        } else if (predicate(y, x)) {
            return ComparisonResult.descending;
        }
        return ComparisonResult.same;
    };
}
```

Predicates to ComparisonResult

- 真偽値 -> comparator を通して -> `ComparisonResult` を返す関数を作る
- 真偽値を返す関数を Predicates という
- 真偽値から < == > の3種類の状態を返せるのでシンプル

```
function isLessOrEqual(x, y) {
    return x <= y;
}
var values = [2, 3, -1, -6, 0, -108, 42, 10];
var expectedSortedValues = [-108, -6, -1, 0, 2, 3, 10, 42];
var results = values.sort(comparator(isLessOrEqual));
assert.deepEqual(results, expectedSortedValues);
```

Null Guard

- 配列に **falsy** な値が含まれてる意図しない結果になってしまう

```
var nums = [1, 2, 3, null, 5];
var multFn = function (total, n) {
    return total * n;
};
_.reduce(nums, multFn);// => 0 になってしまう...
```

Null Check?

- multFnにnullチェックを入れるのは本質的じゃない
- nullチェックを加える **高階関数** を作る

```
function fnull(fn, defaultValue) {
    return function () {
    // falsyだった場合はdefaultValueにしたものに引数を構築し直す
        var args = _.map(arguments, function (e) {
            return e != null ? e : defaultValue;
        });
        return fn.apply(null, args);
    }
}
```

safeMult

- falsyな値はdefalutValueに変更される

```
// var nums = [1, 2, 3, null, 5];
// falsyな値はdefalutValue(1)に変更される
var safeMultFn = fnull(multFn, 1);
var totalMult = _.reduce(nums, safeMultFn);
assert.equal(totalMult, 30);
```

おわり

サンプルコード

- azu/ReduceDance

参考

- functional javascript
- reduce関数は結構有用っていうお話 - あと味