



TELEPASEE

함수형 프로그래밍 라이브러리 Ramda.js와 함께하는 간단 API 서버 만들기

한현섭 (han@telepasee.com)

발표자 소개

- 현 TELEPASEE 개발자
- 어쩌다 보니 처음 배우려고 한 프로그래밍 언어가...  Scala
 - But... 뭔 소린지 몰라서 결국 열혈C 부터 하게 됨
- OOP, 함수형 다 시도해 해볼 수 있는 JavaScript가 너무 좋습니다. (node.js 만세)

함수형 프로그래밍



위키백과
우리 모두의 백과사전

대문
사용자 모임
요즘 화제
최근 바뀜
모든 문서 보기
임의 문서로
도움말
기부

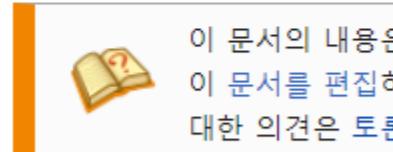
도구

여기로 가리키는 문서

문서 토론

함수형 프로그래밍

위키백과, 우리 모두의 백과사전.



이 문서의 내용은
이 문서를 편집하
대한 의견은 토론

함수형 프로그래밍은 자료 처리를 수학적
로그래밍에서는 상태를 바꾸는 것을 강조하
가능성, 결정문제, 함수정의, 함수응용과 재
들은 람다 연산을 발전시킨 것으로 볼 수 있

잠깐...

- Arrow Function

```
function (x) {  
    return x * 2;  
}
```

=

x => x * 2
(x) => (x * 2)
(x) => { return x * 2; }

함수형 프로그래밍 특징 – 순수함수

- Pure Function

```
let double = x => x * 2; ( 0 )
```

```
let idx = 0;
```

```
let moveIdx = x => idx + x; ( X )
```

함수형 프로그래밍 특징 – 일급함수

- First-Class Function

```
let callFn =  
  (fn, args) => fn.call({}, args);  
  
callFn(x => x * 2, 3); // 6
```

함수형 프로그래밍 특징

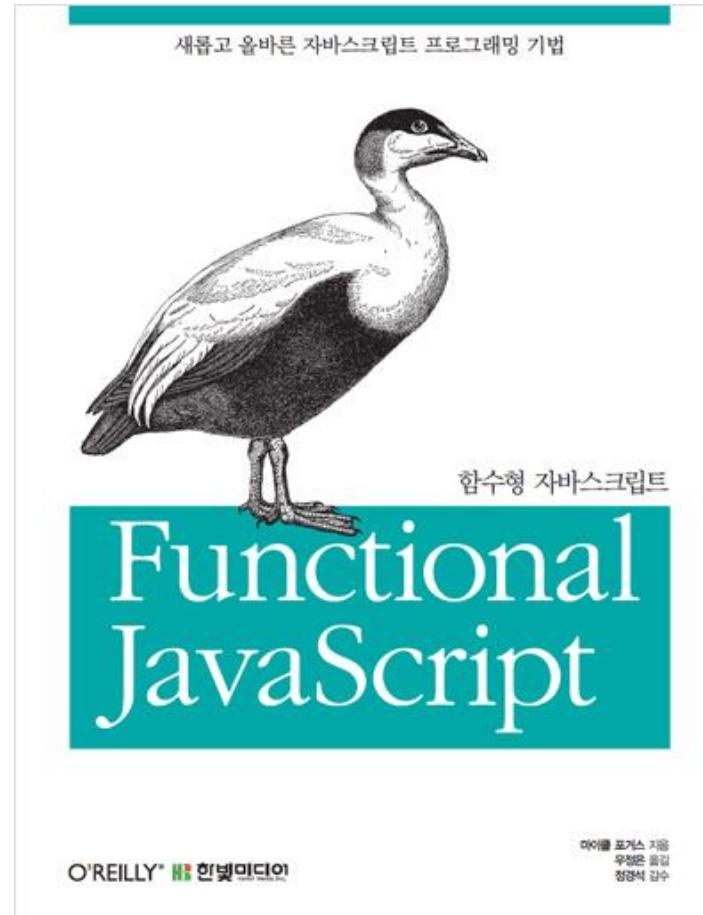
- Persistent Immutable Data
- 영속적이고 불변한 데이터

```
let origin = [1, 2, 3, 4];
```

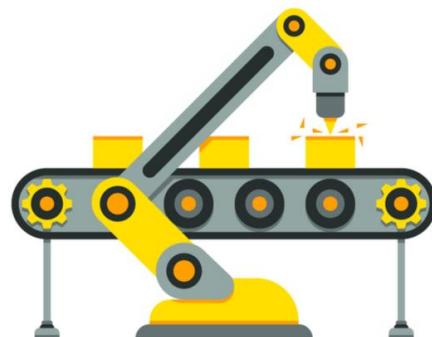
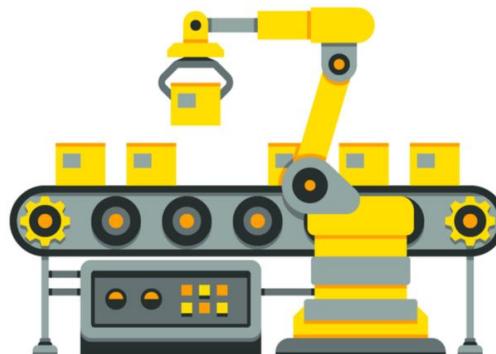
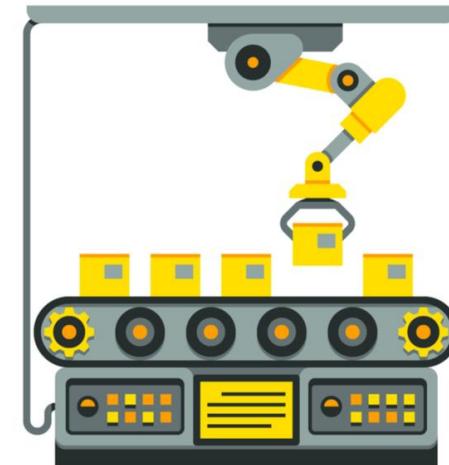
```
let newArr = add5(origin); // [1, 2, 3, 4, 5]
```

```
console.log(origin); // [1, 2, 3, 4]
```

함수형 프로그래밍 특징



함수형 프로그래밍 특징

 $f(x)$  $g(x)$  $h(x)$  $f \circ g \circ h (x)$

함수형 프로그래밍

- 더 나은 강의들
 - Functional Programming Principles in Scala
 - <https://www.coursera.org/learn/progfun1>
 - 자바스크립트로 알아보는 함수형 프로그래밍
 - <https://www.inflearn.com/course/함수형-프로그래밍>

오늘의 Library



Ramda

<http://ramdajs.com/>



Ramda v0.25.0

[Home](#)[Documentation](#)[Try Ramda](#)[GitHub](#)[Discuss](#)

Ramda

A practical functional library for JavaScript programmers.

[build](#) passing [npm package](#) 0.25.0 [dependencies](#) none [gitter](#) [join chat](#)

Why Ramda?

There are already several excellent libraries with a functional flavor. Typically, they are meant to be general-purpose toolkits, suitable for working in multiple paradigms. Ramda has a more focused goal. We wanted a library designed specifically for a functional programming style, one that makes it easy to create functional pipelines, one that never mutates user data.

What's Different?

The primary distinguishing features of Ramda are:

- Ramda emphasizes a purer functional style. Immutability and side-effect free functions are at the heart of its design philosophy. This can help you get the job done with simple, elegant code.
- Ramda functions are automatically curried. This allows you to easily build up new functions from old ones simply by not supplying the final parameters.
- The parameters to Ramda functions are arranged to make it convenient for currying. The data to be operated on is generally supplied last.

The last two points together make it very easy to build functions as sequences of simpler functions, each of which transforms the data and passes it along to the next. Ramda is designed to support this style of coding.





UNDERSCORE.JS

underscore로는
제대로 된
함수형 코딩이 어렵다!

Maintenance문제도 있고...
성능 향상을 시켜보자



Lodash
A modern JavaScript utility library delivering modularity, performance & extras.



좀 더 함수형 코딩 스타일을
지원하자



Lodash
A modern JavaScript utility library delivering modularity, performance & extras.

lodash/fp



vs

UNDERSCORE.JS

```
> R.map( x => x * 2, [1, 2, 3] );
```

```
> [2, 4, 6]
```

```
> _.map( [1, 2, 3], x => x * 2 );
```

```
> [2, 4, 6]
```

Ramda의 함수들은 모두 자동적으로 Curry 됩니다.

```
let double = x => x * 2;
```

```
R.map(double, [1, 2, 3]); // -> [2, 4, 6]
```

```
let doubleMap = R.map(double);
```

```
doubleMap([1, 2, 3]); // -> [2, 4, 6]
```

```
R.map(double)([1, 2, 3]); // -> [2, 4, 6]
```

 와 함께하는 간단 API

간단한 채팅 REST API Spec.

Messegining API

```
GET /messages
queries =
  room : number //채팅방 번호
  time : number //기준시간 unixstamp
```

POST /message

```
body : {
  room      : number, //채팅방 번호
  userKey   : string, //UUID
  text      : string
}
```



와 함께하는 간단 API

우선 서버를 설정합시다.

```
$ mkdir r_exam && cd r_exam  
$ npm install ramda express body-parser
```

위와 같이 패키지들을 설치해주시고

```
let http = require('http');  
let express = require('express');  
let app = express();  
let bodyParser = require('body-parser');  
let router = require('./router');  
  
app.use(bodyParser.json());  
app.use('/', router);  
http.createServer(app).listen('3005');
```

r_exam/router.js

Router GET /messages | POST /message

```
router.get('/messages', (req, res, next) => {
  //validation
  .....
  //getMessages
  let result = repo.getMessages(room, time);

  //return
  res.status(200);
  return res.send({ messages : result
});
```

```
router.post('/message', (req, res, next) => {
  //validation
  .....
  //postMessage
  let isSuccess = repo.postMessage(room, userKey, text);

  //return
  if(isSuccess){
    return res.status(200).send({ isSuccess });
  }else{
    return res.status(500).send({ isSuccess });
  }
});
```

Validator **GET** /messages

```
let room = req.query.room;
let time = req.query.time;

if(room === undefined){
    return res.status(400).send({ error : "query 'room' is missing" });
}else if(time === undefined){
    return res.status(400).send({ error : "query 'time' is missing" });
}else if(isNaN(room)){
    return res.status(400).send({ error : "query 'room' is not Number." });
}else if(isNaN(time)){
    return res.status(400).send({ error : "query 'time' is not Number." });
}
```

Validator **POST** /message

```
let room      = req.body.room;
let userKey   = req.body.userKey;
let text       = req.body.text;

if(room === undefined) {
    return res.status(400).send({ error : "missing 'room' params" });
}else if(userKey === undefined){
    return res.status(400).send({ error : "missing 'userKey' params" });
}else if(text === undefined) {
    return res.status(400).send({ error : "missing 'text' params" });
}else if(isNaN(room)){
    return res.status(400).send({ error : "param 'room' is not Number." });
}else if(!(typeof userKey === 'string')){
    return res.status(400).send({ error : "param 'userKey' is not String." });
}else if(!uuidValidRegex.test(userKey)){
    return res.status(400).send({ error : "param 'userKey' is not UUID." });
}else if(!(typeof text === 'string')){
    return res.status(400).send({ error : "param 'text' is not String." });
}
```

Validator Refactoring

```
let room = req.query.room;
let time = req.query.time;

let validMissingParam = (item)=>{
    return item === undefined;
}

let validInvalidParam = (item, fn)=>{
    return fn(item);
}

if(validMissingParam(room)){
    return res.status(400).send({ error : "query 'room' is missing" });
}else if(validMissingParam(time)){
    return res.status(400).send({ error : "query 'time' is missing" });
}else if(validInvalidParam(room, isNaN)){
    return res.status(400).send({ error : "query 'room' is not Number." });
}else if(validInvalidParam(time, isNaN)){
    return res.status(400).send({ error : "query 'time' is not Number." });
}
```

Validator Refactoring

```
let validMissingParam = (item)=>{
    return item === undefined;
}

let validInvalidParam = (item, fn)=>{
    return fn(item);
}

if(validMissingParam("room", { req, res })){
    //return res.status(400).send({ error : "query 'room' is missing" });
}else if(validMissingParam("time", { req, res })){
    //return res.status(400).send({ error : "query 'time' is missing" });
}else if(validInvalidParam("room", isNaN, { req, res })){
    //return res.status(400).send({ error : "query 'room' is not Number." });
}else if(validInvalidParam("time", isNaN, { req, res })){
    //return res.status(400).send({ error : "query 'time' is not Number." });
}
```

Validator Refactoring

```
let validMissingParam = (itemName, httpParam)=>{
  if(httpParam.req.query[itemName] === undefined){
    httpParam.res.status(400)
      .send({error : "query " + itemName + " is missing."});
    return false;
  }else{
    return true;
}
}
```

R.path

path

Object



[Idx] → {a} → a | Undefined

Idx = String | Int

EXPAND PARAMETERS

Added in v0.2.0

Retrieve the value at a given path.

See also [prop](#).

```
R.path(['a', 'b'], {a: {b: 2}}); //=> 2
```

[Open in REPL](#)

[Run it here](#)

```
R.path(['a', 'b'], {c: {b: 2}}); //=> undefined
```

R.path

```
let validMissingParam = (itemName, httpParam)=>{
  if(R.path(["req", "query", itemName], httpParam) === undefined){
    httpParam.res.status(400)
      .send({error : "query " + itemName + " is missing."});
    return false;
  }else{
    return true;
  }
}
```

R.isNil

isNil

Type 

* → Boolean

[EXPAND PARAMETERS](#)

Added in v0.9.0

Checks if the input value is `null` or `undefined`.

```
R.isNil(null); //=> true  
R.isNil(undefined); //=> true  
R.isNil(0); //=> false  
R.isNil([]); //=> false
```

[Open in REPL](#) [Run it here](#)

R.isNil

```
let validMissingParam = (itemName, httpParam)=>{
  if(R.isNil(R.path(["req", "query", itemName], httpParam))) {
    httpParam.res.status(400)
      .send({error : "query " + itemName + " is missing."});
    return false;
  }else{
    return true;
  }
}
```

R.compose

compose

Function

((y → z), (x → y), …, (o → p), ((a, b, …, n) → o)) → ((a, b, …, n) → z)

EXPAND PARAMETERS

Added in v0.1.0

Performs right-to-left function composition. The rightmost function may have any arity; the remaining functions must be unary.

Note: The result of compose is not automatically curried.

See also [pipe](#).

```
var classyGreeting = (firstName, lastName) => "The name's " + lastName + ", " + firstName + " " + lastName
var yellGreeting = R.compose(R.toUpperCase, classyGreeting);
yellGreeting('James', 'Bond'); //=> "THE NAME'S BOND, JAMES BOND"
```

```
R.compose(Math.abs, R.add(1), R.multiply(2))(-4) //=> 7
```

R.compose

$$f(g(h(x))) = f \circ g \circ h (x)$$

```
let result = a(b(c(d(x))));
```

```
let result = R.compose(a,b,c,d)(x);
```

R.compose

```
R.isNil(R.path(["req", "query", itemName], httpParam));  
  
R.compose(R.isNil, R.path(["req", "query", itemName]))(httpParam);
```

```
let validMissingParam = (itemName, httpParam)=>{  
  if(R.compose(R.isNil, R.path(["req", "query", itemName]))(httpParam)){  
    httpParam.res.status(400)  
    .send({error : "query " + itemName + " is missing."});  
    return false;  
  }else{  
    return true;  
  }  
}
```

R.last

last

[List](#)

[a] → a | Undefined

String → String

EXPAND PARAMETERS

Added in v0.1.4

Returns the last element of the given list or string.

See also [init](#), [head](#), [tail](#).

```
R.last(['fi', 'fo', 'fum']); //=> 'fum'  
R.last([]); //=> undefined
```

[Open in REPL](#) [Run it here](#)

```
R.last('abc'); //=> 'c'  
R.last(''); //=> ''
```

Validator with

```
let validMissingParam = (path, httpParam)=>{
  if(R.compose(R.isNil, R.path(path))(httpParam)){
    httpParam.res.status(400)
      .send({error : "Missing Parameter : " + R.last(path)});
    return false;
  }else{
    return true;
  }
}

let validInvalidParam = (path, validFn, httpParam)=>{
  if(R.compose(validFn, R.path(path))(httpParam)){
    httpParam.res.status(400)
      .send({error : "Invalid Parameter : " + R.last(path)});
    return false;
  }else{
    return true;
  }
}
```

R.curry

curry

Function



`(* → a) → (* → a)`

[EXPAND PARAMETERS](#)

Added in v0.1.0

Returns a curried equivalent of the provided function. The curried function has two unusual capabilities. First, its arguments needn't be provided one at a time. If `f` is a ternary function and `g` is `R.curry(f)`, the following are equivalent:

- `g(1)(2)(3)`
- `g(1)(2, 3)`
- `g(1, 2)(3)`
- `g(1, 2, 3)`

See also [curryN](#).

```
var addFourNumbers = (a, b, c, d) => a + b + c + d;
```

[Open in REPL](#)

[Run it here](#)

```
var curriedAddFourNumbers = R.curry(addFourNumbers);
var f = curriedAddFourNumbers(1, 2);
var g = f(3);
g(4); //=> 10
```

R.curry

```
let validParam = R.curry((message, validFn, path, httpParam)=>{
  if(R.compose(validFn, R.path(path))(httpParam)){
    httpParam.res.status(400).send({error : message + R.last(path)});
    return false;
  }else{
    return true;
  }
});

let validMissingParam = validParam("Missing Parameter : ", R.isNil);
let validInvalidParam = validParam("Invalid Parameter : ");
```

Curried

```
let validMissingParam = validParam("Missing Parameter : ", R.isNil);
let validInvalidParam = validParam("Invalid Parameter : ");

router.get('/messages', (req, res, next) => {
    //validation
    if(validMissingParam(["req", "query", "room"], {req, res}) &&
        validMissingParam(["req", "query", "room"], {req, res}) &&
        validInvalidParam(isNaN, ["req", "query", "room"], {req, res}) &&
        validInvalidParam(isNaN, ["req", "query", "room"], {req, res}))
    {
        //getMessages
        let result = repo.getMessages(room, time);

        //return
        res.status(200).send({ messages : result });
    }
});
```

R.allPass

allPass

Logic



`[(*... → Boolean)] → (*... → Boolean)`

EXPAND PARAMETERS

Added in v0.9.0

Takes a list of predicates and returns a predicate that returns true for a given list of arguments if every one of the provided predicates is satisfied by those arguments.

The function returned is a curried function whose arity matches that of the highest-arity predicate.

See also [anyPass](#).

```
var isQueen = R.propEq('rank', 'Q');
var isSpade = R.propEq('suit', '♠');
var isQueenOfSpades = R.allPass([isQueen, isSpade]);
```

[Open in REPL](#)

[Run it here](#)

```
isQueenOfSpades({rank: 'Q', suit: '♣'}); //=> false
isQueenOfSpades({rank: 'Q', suit: '♠'}); //=> true
```

R.allPass

```
router.get('/messages', (req, res, next) => {
  //validation
  if(R.allPass([validMissingParam(["req", "query", "room"]),
    validMissingParam(["req", "query", "time"]),
    validInvalidParam(isNaN, ["req", "query", "room"]),
    validInvalidParam(isNaN, ["req", "query", "time"])])([req, res]))
  {
    //getMessages
    let result = repo.getMessages(R.compose(R.pick(["room", "time"]),
      R.prop("query")))(req);

    //return
    res.status(200).send({ messages : result });
  }
});
```

Validator **POST** /message

```
router.post('/message', (req, res, next) => {
  //validation
  if(R.allPass([
    validMissingParam(["req", "body", "room"]),
    validMissingParam(["req", "body", "userKey"]),
    validMissingParam(["req", "body", "text"]),
    validInvalidParam(isNaN, ["req", "body", "room"]),
    validInvalidParam(R.is(String), ["req", "body", "userKey"]),
    validInvalidParam(uuidValidRegex, ["req", "body", "userKey"]),
    validInvalidParam(R.is(String), ["req", "body", "text"])
  ])([req, res])){
    //post Message
    let isSuccess =
      repo.postMessage(
        R.compose(R.pick(["room", "userKey", "text"]), R.prop("body"))(req)
      );
    if(isSuccess){
      return res.status(200).send({ isSuccess });
    }else{
      return res.status(500).send({ isSuccess });
    }
  }
});
```

Try Ramda - Ramda REPL

Ramda v0.25.0

[Home](#)[Documentation](#)[Try Ramda](#)[GitHub](#)**INPUT** [Reset](#) [Share](#)**OUTPUT** [Clear](#) [Tidy](#)

```
1 R.reduceRight(R.subtract, 0, [1, 2, 3, 4]) // => (1 - (2  
2 //      -                                -2  
3 //      / \                             / \  
4 //      1   -                            1   3  
5 //      / \                             / \  
6 //      2   -      ==>      2   -1  
7 //      / \                             / \  
8 //      3   -                            3   4  
9 //      / \                             / \  
10 //     4   0                            4   0
```

-2

Try Ramda - 이럴때 좋아요.

- 아래는 모두 실 사용예입니다.
 - 문자열 합칠 때
 - 중첩배열을 다루는 스크립트 짤 때
 - 함수형으로 짜다가 디버깅 할 때
 - Doc내용이 뭔 소린지 도통 모를 때

Ramda Cookbook

Cookbook

Vladimir Gorej edited this page 7 days ago · 156 revisions

Recipes

- `pickIndexes` : Pick values from a list by indexes
 - `list` : Create a `list` function
 - `cssQuery` / `setStyle` : Mess with the DOM
 - Apply a list of functions in a specific order into a list of values
 - `dotPath` / `propsDotPath` : Derivative of `R.props` for deep fields
 - Use ramda-fantasy Future to wrap AJAX
 - Get n function calls as a list
 - `defaults` : Set properties only if they don't exist
-



INPUT [Reset](#) [Share](#)

1

2

3

4 `R.compose(R.join(" "), R.pair)('Thank', 'You!')`

OUTPUT [Clear](#) [Tidy](#)

"Thank You!"