Partial function application in JavaScript

What!?, how, why (and why not)

Adrian Lang
@adrianlang
xmpp:adrian@kleinrot.net
mail@adrianlang.de

November 17., 2011

CC BY-SA

What is partial function application?

Partial function application is the process of applying a function to less arguments than it expects.

```
f.length === 2
f(1)
```

Instead of an actual result, such an application returns a function expecting the remaining arguments.

```
typeof f(1) === 'function'
f(1).length === 1
```

When all arguments are specified, the original function is executed.

```
f(1)(2) === f(1, 2)
```


CC BY-SA

A Haskell example

```
times :: t -> (t -> t)
times a b = a * b

times5 :: t -> t
times5 = times 5 -- OMG IT'S ONLY 1 PARAM!

times5_2 :: t
times5_2 = times5 2
```


CC BY-SA

Now JavaScript

```
function times (a, b) {
    return a * b;
}

var times5 = times(5); // NaN

var times5_2 = times5(2); // Type error
```


CC BY-SA

bind

```
function times (a, b) {
    return a * b;
}

var times5 = times.bind(undefined, 5);

var times5_2 = times5(2);
```


CC BY-SA

Adding support for partial function application

```
function times (a, b) {
    if (arguments.length < 2) {
        var argv = Array.prototype.slice.call(arguments);
        argv.unshift(times);
        return bind.apply(argv);
    }
    return a * b;
}

var times5 = times(5);

var times5_2 = times5(2);
```


CC BY-SA

Abstracting away

```
function partial(callee, argc, argv) {
    if (argv.length < argc) {
        argv = Array.prototype.slice.apply(argv);
        argv.unshift(callee);
        return bind.apply(this, argv);
    }
}

function times(a, b) {
    return partial(times, times.length, arguments) ||
           a * b;
}

var times5 = times(5);

var times5_2 = times5(2);
```


CC BY-SA

. . . far away

```
function partialize(func) {
    return function wrapper(/* ... */) {
        return partial(wrapper, func.length, arguments) ||
                func.apply(this, arguments);
    };
}

var times = partialize(function (a, b) {
    return a * b;
});

var times5 = times(5);

var times5_2 = times5(2);
```


CC BY-SA

Use cases (1/3)

You can use partial function application everywhere you use `bind` (if you only bind formal arguments, not `this`, and if you do not bind all arguments). For example:

- Event handlers

```
// Do stuff
function menuAction(item, e) {}

$.each(['login', 'search', 'index'], function (item) {
    $('#menu__' + item).click(menuAction(item));
});
```


CC BY-SA

Use cases (2/3)

- Preparing (node.js) callbacks

```
// Retrieve wiki page
function getWikiPage(pagename, callback) {}

// Parse some data out of a text
function parseTextForData(text, callback) {}

function getDataFromWikiPage(pagename, callback) {
    async.waterfall([
        getWikiPage(pagename),
        parseTextForData
    ], callback);
}
```


CC BY-SA

Use cases (3/3)

- Giving functions the place they deserve – first class objects with usable syntax

```
var times3plus5 = _.compose(plus(5), times(3));

var doubled = _.map(values, times(2));

_.each(['Crappy usage examples done',
        'Let\'s talk about problems'], appendTo(log));
```


CC BY-SA

Optional params and variadic functions

Many functions handle a dynamic number of arguments. For example, about half of underscore.js's functions have a non-fixed number of arguments.

- At some point, `partialize` has to recognize an application as complete – usually when all mandatory arguments are given. So, no partial application of exclusively optional arguments. Another option is to make all optional parameters mandatory.
- Distinguishing mandatory from optional arguments cannot be done reliable through `f.length`. Creating a function supporting partial application gets uglier.

CC BY-SA

`this`

What to do with `this`?

- Late bind: `obj.f(a).call(eTarget, b)` has `eTarget` as `this`; `partialize` has to use a hand-crafted bind.
- Early bind: `obj.f(a).call(eTarget, b)` has `obj` as `this`; `partialize` can use native `bind`, where available.

Both ways are reasonable and have their use-cases.

CC BY-SA

Performance

- Partializing based on V8's native `bind` is around 2.5 times faster than using my hand-crafted `bind`, which supports late-bind of `this`.
- Partially applying is slower the more variables already have been bound. This is not the case with plain `bind`.
- Even based on the native `bind`, partial application takes around 1.4 times as long as a plain `bind`.
- Partial application does not get slower if done multiple times (i. e. binding three arguments each in single function call). With plain `bind`, the final execution is slower with multiple binds.

CC BY-SA

Social problems

The biggest problem is probably adoption, even in a specific project, or, put different: People won't get it. This is amplified by some things:

- Built-in functions; They may be partialized, but that's tedious work.
- Partial application? Variadic function? Optional parameters left out? What is the signature of this function?
- The error condition of not specifying enough parameters gets even worse than most functions do right now.

CC BY-SA

More to think about (1/3)

Currently, you have to specify parameters in the order they are defined. Stuff like the following would be nice:

```
var addLogLine = append(UNBOUND, log);

addLogLine("Just an idea");
```

Another idea is being able to perform non-trivial computations with the arguments already given. This could greatly enhance performance in some cases.

CC BY-SA

More to think about (2/3)

Partial function application comes from a functional background and works better with functions without side effects. JavaScript is not purely functional, and so there is a difference between executing a function and binding all parameters – the latter creates a function without any parameters. However, parameters specifying event objects or node.js callbacks help differentiating between the two.

CC BY-SA

More to think about (3/3)

Supporting partial application means `f(1)(2)` works like `f(1, 2)`. There is some usage for the reverse, i. e. making `f(1, 2)` work like `f(1)(2)`;

```
function getDataFromWikiPage(pagename, callback) {
    async.waterfall([
        getWikiPage(pagename),
        parseTextForData
    ], callback);
}

function getDataFromWikiPage(pagename) {
    async.waterfall([
        getWikiPage(pagename),
        parseTextForData
    ]);
}
```


CC BY-SA

When you see it . . .

```
$foo.click(function (event) {
    return clickhandler(a, event);
});
```

. . . you'll apply partially.

CC BY-SA