Partial function application in JavaScript

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

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

November 21., 2011

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)
```

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
```

Now JavaScript

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

var times5 = times(5); // NaN

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

bind

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

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

var times5_2 = times5(2);
```

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);
```

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);
```

. . . 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);
```

Use cases

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).
If you don't use `bind`, well . . . ignore this ›partial‹ stuff I'm talking about and try to understand that ›bind‹ stuff.

Event handlers

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

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

Preparing (node.js) callbacks

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

Preparing (node.js) callbacks

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

Preparing (node.js) callbacks

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

Working with functions

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));
```

Side effects

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.

```
times5_2 = times 5 2 -- Value
```

```
var times5_2 = times.bind(undefined, 5, 2); // Function
```

```
var times5_2 = times(5, 2); // Value
```

```
var times5_2 = times(5)(2); // Value
```

Side effects

However, parameters specifying event objects or callbacks help differentiating between execution and just binding all parameters.

```
function rollMenu(menu, e) {
}

$.hover('#helpmenu', rollMenu('help'));

// ...

function parseData(data, callback) {
}

var parseCurrentFile = parseData(currentFile);
```

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`, so you would have to specify the function's length explicitly when partializing.

`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.
However, the native bind is around 2.5 times faster (where it's available), so I use it, and hence early-bind.

Performance

It's really not that slow (takes 1.4 times as long as a simple bind). Some specialties:

- Partially applying is slower the more variables already have been bound. This is not the case with 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.

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; You could partialize them as well, 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.

partial-js

```
var partial = require('partial');

Math.pow = partial.partialize(Math.pow);
var powersOf2 = Math.pow(2);
```

partial-js; less intrusive

```
partial.prototypeP();
var powersOf2 = Math.pow.p(2);

// or

partial.functionP(Math.pow);
var powersOf2 = Math.pow.p(2);

// or

var powersOf2 = partial(Math.pow, 2);
```

More to think about (1/2)

Currently, you have to pass parameters in the order they are declared in the function signature.

```
var addLogLine = function (entry) {
    return append(entry, log);
};
```

If there would be a function `appendTo` instead of `append`, this could be way easier with partial function application:

```
var addLogLine = appendTo(log);
```

It's unreasonable to create different functions just for reordering the arguments. How about this:

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

More to think about (2/2)

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);
getDataFromWikiPage('somepage', console.log);

function getDataFromWikiPage(pagename) {
    return async.waterfall([
        getWikiPage(pagename),
        parseTextForData
    ]);
}
getDataFromWikiPage('somepage', console.log);
```

When you see it . . .

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

. . . you'll apply partially.

Adrian Lang

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

Partial function application

https://github.com/adrianlang/partial-js
https://adrianlang.de/talks/partial2.pdf