Function call forms and the value of `this`

Relevant Links

- Flanagan's book, section 8.2

Notes

There are 4 different ways that functions can be called, called "invocations". We describe them briefly here, and we will go deeper into them later. A big difference is how this is treated in each case.

function invoc. `f(...)` where `f` is a function.

this set to the global object. CAREFUL!

method invoc. `m.f(...)` where `m` is an object and `f` is a property of it with function value.

this set to `m`.

constructor invoc. `new F(...)` where `F` is a function. Constructors are by convention capitalized.

this set to a newly created object.

indirect invoc. `f.call(...)`, `f.bind(...)`, `f.apply(...)`.

this set to the first argument.

You need to be very careful when passing functions to some other part of your code, as you don't necessarily know how they are going to be called.

Here is an example of what can go wrong:

```
[1,2,3].forEach(console.log);
```

This produces an error in Chrome (but not in Node interestingly enough). It appears that in Chrome, console.log expects to be invoked with the this object set to console. But the functions passed in forEach are invoked as functions, not as methods. Try this to see it more clearly:

```
[1,2,3].forEach(function() { console.log(this); });
```

Indirect Invocations

The indirect invocations deserve further notice. There are mainly 3 functions:

Calls `f` with the first argument serving as `this` and any subsequent arguments passed as arguments to `f`.

e.g. f.call(null, 1, 2, 3);

Calls `f` with the first argument serving as this and the second argument being an array of the arguments to be used in the call.

e.g. f.apply(null, [1,2,3]);

Does not actually call `f`, but it returns a function which behaves like `f` except that it has "bound" the `this` object, and optionally has bound some number of arguments.

e.g. `f.bind(o, 1, 2)(3, 4)` is the same as `f.call(o, 1, 2, 3, 4)`.