kind-of

Get the native type of a value.

Please consider following this project's author, Jon Schlinkert, and consider starring the project to show your :heart: and support.

Install

Install with npm:

Install with bower

Why use this?

it's fast | optimizations
better type checking

Usage

es5, es6, and browser ready

```
var kindOf = require('kind-of');

kindOf(undefined);
//=> 'undefined'

kindOf(null);
//=> 'null'

kindOf(true);
//=> 'boolean'

kindOf(false);
//=> 'boolean'

kindOf(new Buffer(''));
//=> 'buffer'

kindOf(42);
//=> 'number'

kindOf('str');
//=> 'string'

kindOf(arguments);
//=> 'arguments'

kindOf({});
//=> 'object'
```

Author

Jon Schlinkert

- GitHub Profile
- Twitter Profile
- LinkedIn Profile

License

This file was generated by verb-generate-readme, v0.8.0, on January 16, 2020.


```
kindOf(Object.create(null));
//=> 'object'

kindOf(new Test());
//=> 'object'

kindOf(new Date());
//=> 'date'

kindOf([1, 2, 3]);
//=> 'array'

kindOf(/foo/);
//=> 'regexp'

kindOf(new RegExp('foo'));
//=> 'regexp'

kindOf(new Error('error'));
//=> 'error'

kindOf(function () {});
//=> 'function'

kindOf(function * () {});
//=> 'generatorfunction'

kindOf(Symbol('str'));
//=> 'symbol'

kindOf(new Map());
//=> 'map'

kindOf(new WeakMap());
```

```
//=> 'weakmap'

kindOf(new Set());
//=> 'set'

kindOf(new WeakSet());
//=> 'weakset'

kindOf(new Int8Array());
//=> 'int8array'

kindOf(new Uint8Array());
//=> 'uint8array'

kindOf(new Uint8ClampedArray());
//=> 'uint8clampedarray'

kindOf(new Int16Array());
//=> 'int16array'

kindOf(new Uint16Array());
//=> 'uint16array'

kindOf(new Int32Array());
//=> 'int32array'

kindOf(new Uint32Array());
//=> 'uint32array'

kindOf(new Float32Array());
//=> 'float32array'

kindOf(new Float64Array());
//=> 'float64array'
```


Related projects

You might also be interested in these projects:

- is-glob: Returns `true` if the given string looks like a glob pattern or an extglob pattern… more | homepage
- is-number: Returns true if a number or string value is a finite number. Useful for regex… more | homepage
- is-primitive: Returns `true` if the value is a primitive. | homepage

Contributors

Commits	Contributor
102	jonschlinkert
3	aretecode
2	miguelmota
1	doowb
1	dtothefp
1	ianstormtaylor
1	ksheedlo
1	pdehaan
1	laggingreflex
1	tunnckoCore
1	xiaofen9

```
//=> 'object'
console.log(typeOf(new WeakMap()));
//=> 'object'
console.log(typeOf(new WeakSet()));
//=> 'object'
```

About

Contributing

Pull requests and stars are always welcome. For bugs and feature requests, please create an issue.

Running Tests

Running and reviewing unit tests is a great way to get familiarized with a library and its API. You can install dependencies and run tests with the following command:

```
$ npm install && npm test
```

Building docs

(This project's readme.md is generated by verb, please don't edit the readme directly. Any changes to the readme must be made in the .verb.md readme template.)

To generate the readme, run the following command:

```
$ npm install -g verbose/verb#dev verb-
        generate-readme && verb
```


Benchmarks

Benchmarked against typeof and type-of.

```
# arguments (32 bytes)
  kind-of x 17,024,098 ops/sec ±1.90%
        (86 runs sampled)
  lib-type-of x 11,926,235 ops/sec
        ±1.34% (83 runs sampled)
  lib-typeof x 9,245,257 ops/sec ±1.22%
        (87 runs sampled)

  fastest is kind-of (by 161% avg)

# array (22 bytes)
  kind-of x 17,196,492 ops/sec ±1.07%
        (88 runs sampled)
  lib-type-of x 8,838,283 ops/sec
        ±1.02% (87 runs sampled)
  lib-typeof x 8,677,848 ops/sec ±0.87%
        (87 runs sampled)

  fastest is kind-of (by 196% avg)

# boolean (24 bytes)
  kind-of x 16,841,600 ops/sec ±1.10%
        (86 runs sampled)
  lib-type-of x 8,096,787 ops/sec
        ±0.95% (87 runs sampled)
  lib-typeof x 8,423,345 ops/sec ±1.15%
        (86 runs sampled)
```

```
  fastest is kind-of (by 204% avg)

# buffer (38 bytes)
  kind-of x 14,848,060 ops/sec ±1.05%
        (86 runs sampled)
  lib-type-of x 3,671,577 ops/sec
        ±1.49% (87 runs sampled)
  lib-typeof x 8,360,236 ops/sec ±1.24%
        (86 runs sampled)

  fastest is kind-of (by 247% avg)

# date (30 bytes)
  kind-of x 16,067,761 ops/sec ±1.58%
        (86 runs sampled)
  lib-type-of x 8,954,436 ops/sec
        ±1.40% (87 runs sampled)
  lib-typeof x 8,488,307 ops/sec ±1.51%
        (84 runs sampled)

  fastest is kind-of (by 184% avg)

# error (36 bytes)
  kind-of x 9,634,090 ops/sec ±1.12%
        (89 runs sampled)
  lib-type-of x 7,735,624 ops/sec
        ±1.32% (86 runs sampled)
  lib-typeof x 7,442,160 ops/sec ±1.11%
        (90 runs sampled)

  fastest is kind-of (by 127% avg)

# function (34 bytes)
```


typeof lib

Incorrectly identifies instances of custom constructors (pretty common):

```
var typeOf = require('typeof');
function Test() {}
console.log(typeOf(new Test()));
//=> 'test'
```

Returns `object` instead of `arguments`:

```
function foo() {
  console.log(typeOf(arguments)) //=>
        'object'
}
foo();
```

type-of lib

Incorrectly returns `object` for generator functions, buffers, `Map`, `Set`, `WeakMap` and `WeakSet`:

```
function * foo() {}
console.log(typeOf(foo));
//=> 'object'
console.log(typeOf(new Buffer('')));
//=> 'object'
console.log(typeOf(new Map()));
//=> 'object'
console.log(typeOf(new Set()));
```

some time to understand how and why `typeof` checks were being used in my own libraries and other libraries I use a lot.

2. Optimize around bottlenecks - In other words, the order in which conditionals are implemented is significant, because each check is only as fast as the failing checks that came before it. Here, the biggest bottleneck by far is checking for plain objects (an object that was created by the `Object` constructor). I opted to make this check happen by process of elimination rather than brute force up front (e.g. by using something like `val.constructor.name`), so that every other type check would not be penalized it.
3. Don't do uneccessary processing - why do `.slice(8, -1).toLowerCase()`; just to get the word `regex`? It's much faster to do `if (type === '[object RegExp]') return 'regex'`
4. There is no reason to make the code in a microlib as terse as possible, just to win points for making it shorter. It's always better to favor performant code over terse code. You will always only be using a single `require()` statement to use the library anyway, regardless of how the code is written.

Better type checking

kind-of seems to be more consistently "correct" than other type checking libs I've looked at. For example, here are some differing results from other popular libs:


```
  kind-of x 10,031,494 ops/sec ±1.27%
        (86 runs sampled)
  lib-type-of x 9,502,757 ops/sec
        ±1.17% (89 runs sampled)
  lib-typeof x 8,278,985 ops/sec ±1.08%
        (88 runs sampled)

  fastest is kind-of (by 113% avg)

# null (24 bytes)
  kind-of x 18,159,808 ops/sec ±1.92%
        (86 runs sampled)
  lib-type-of x 12,927,635 ops/sec
        ±1.01% (88 runs sampled)
  lib-typeof x 7,958,234 ops/sec ±1.21%
        (89 runs sampled)

  fastest is kind-of (by 174% avg)

# number (22 bytes)
  kind-of x 17,846,779 ops/sec ±0.91%
        (85 runs sampled)
  lib-type-of x 3,316,636 ops/sec
        ±1.19% (86 runs sampled)
  lib-typeof x 2,329,477 ops/sec ±2.21%
        (85 runs sampled)

  fastest is kind-of (by 632% avg)

# object-plain (47 bytes)
  kind-of x 7,085,155 ops/sec ±1.05%
        (88 runs sampled)
  lib-type-of x 8,870,930 ops/sec
        ±1.06% (83 runs sampled)
```

```
  lib-typeof x 8,716,024 ops/sec ±1.05%
        (87 runs sampled)

  fastest is lib-type-of (by 112% avg)

# regex (25 bytes)
  kind-of x 14,196,052 ops/sec ±1.65%
        (84 runs sampled)
  lib-type-of x 9,554,164 ops/sec
        ±1.25% (88 runs sampled)
  lib-typeof x 8,359,691 ops/sec ±1.07%
        (87 runs sampled)

  fastest is kind-of (by 158% avg)

# string (33 bytes)
  kind-of x 16,131,428 ops/sec ±1.41%
        (85 runs sampled)
  lib-type-of x 7,273,172 ops/sec
        ±1.05% (87 runs sampled)
  lib-typeof x 7,382,635 ops/sec ±1.17%
        (85 runs sampled)

  fastest is kind-of (by 220% avg)

# symbol (34 bytes)
  kind-of x 17,011,537 ops/sec ±1.24%
        (86 runs sampled)
  lib-type-of x 3,492,454 ops/sec
        ±1.23% (89 runs sampled)
  lib-typeof x 7,471,235 ops/sec ±2.48%
        (87 runs sampled)

  fastest is kind-of (by 310% avg)
```



```
# template-strings (36 bytes)
  kind-of x 15,434,250 ops/sec ±1.46%
        (83 runs sampled)
  lib-type-of x 7,157,907 ops/sec
        ±0.97% (87 runs sampled)
  lib-typeof x 7,517,986 ops/sec ±0.92%
        (86 runs sampled)

  fastest is kind-of (by 210% avg)

# undefined (29 bytes)
  kind-of x 19,167,115 ops/sec ±1.71%
        (87 runs sampled)
  lib-type-of x 15,477,740 ops/sec
        ±1.63% (85 runs sampled)
  lib-typeof x 19,075,495 ops/sec
        ±1.17% (83 runs sampled)

  fastest is lib-typeof,kind-of
```

Optimizations

In 7 out of 8 cases, this library is 2x-10x faster than other top libraries included in the benchmarks. There are a few things that lead to this performance advantage, none of them hard and fast rules, but all of them simple and repeatable in almost any code library:

1. Optimize around the fastest and most common use cases first. Of course, this will change from project-to-project, but I took