Object.propertyCount()

Addressing a common JS bottleneck.

Champion: **Jordan Harband**

Author: **Ruben Bridgewater**

107th ECMA Meeting | April 2025

Motivation

Almost any big library / framework uses `Object.keys(object).length` in different ways.

`Object.getOwnSymbols` and `Object.getOwnPropertyNames` are also frequently used that way.

Many algorithms would be faster having such API.
Especially useful for fath paths.

Performance / Memory impact

Performance often changes (JIT, C++, Cross platform Assembler, GC, etc. all have a big impact)

Cost:

- Initial API call cost (*cpu*) 🔴
- Cost for traversing the keys (*cpu*) 🟡
- Cost for allocating the array (cpu_ & _memory) 🟢
- GC (*cpu & memory*) 🟢
- (Cost for converting index keys to strings) (*cpu*) 🟢

Effective Performance

Shape and algorithms determine overhead.

```javascript
const empty = {}
Object.keys(empty).length

const array = Array.from({ length: 10000 })
array.key = true
Object.keys(array).length !== array.length

const bigObject = array.reduce((obj, _, i) => { obj[`key_${i}`] = i; return obj }, {})
Object.keys(bigObject).length
```

Use cases

1. Input validation and guarding against too big input
2. Object comparison (*Frequent case*)
3. Sparse array detection
    - Mostly not done (false results vs. bad runtime)
4. Detecting extra properties on array like objects
    - Mostly not done (false results vs. bad runtime)
5. Fast telemetry data
6. Testing utility (check for the number of properties)
7. General fast paths for many algorithms

API - Object.propertyCount(target[, options])

- **target**: The object whose properties will be counted.
    - Throws TypeError if target is not an object.
- **options**?: An object specifying filtering criteria:
    - **keyTypes**?: Array specifying property types to include:
        - Possible values: *'index', 'nonIndexString', 'symbol'*.
        - Defaults to *['index', 'nonIndexString']* (aligning closely with Object.keys).
    - **enumerable**?: Indicates property enumerability:
        - *true* to count only enumerable properties (default).
        - *false* to count only non-enumerable properties.
        - *'all'* to count both enumerable and non-enumerable properties.
    - Throws TypeError if any option provided contains invalid key or value.

Alternative for nested options

- **options**?: An object specifying filtering criteria:
  - **indexKeys**?: Boolean (*default: true*)
  - **nonIndexKeys**: Boolean (*default: true*)
  - **symbolKeys**: Boolean (*default: false*)
  - **enumerable**: Indicates property enumerability:
    - *true* to count only enumerable properties (default).
    - *false* to count only non-enumerable properties.
    - *'all'* to count both enumerable and non-enumerable properties.
  - Throws TypeError if any option provided contains invalid key or value.

Options vs. multiple methods

- Adoption of all methods is slower than a single one
- Simplicity
- Expert API

Why only own properties?

- Use case for inherited properties is rare
- Can still be added at a later point with the current proposal

Example uses

- Angular
  - type: **['index', 'nonIndexString']**, enumerable: **true** // Object.keys
  - type: **['symbol']**, enumerable: **true** // Object.getOwnPropertySymbols & filter
- React
  - type: **['index', 'nonIndexString']**, enumerable: **true** // Object.keys
  - type: **['index', 'nonIndexString']**, enumerable: ***'all'*** // Object.getOwnPropertyNames
  - type: **['symbol']**, enumerable: **true** // Object.getOwnPropertySymbols & filter
  - type: **['symbol']**, enumerable: ***'all'*** // Object.getOwnPropertySymbols
- Lodash
  - type: **['index', 'nonIndexString']**, enumerable: **true** // Object.keys

More Examples

- Next.js
  - type: **['index', 'nonIndexString']**, enumerable: **true** // Object.keys
  - type: **['symbol']**, enumerable: **true** // Object.getOwnPropertySymbols & filter
  - type: **['symbol']**, enumerable: ***'all'*** // Object.getOwnPropertySymbols
- TypeScript
  - type: **['index', 'nonIndexString']**, enumerable: **true** // Object.keys
  - type: **['symbol']**, enumerable: **true** // Object.getOwnPropertySymbols & filter
- vscode
  - type: **['index', 'nonIndexString']**, enumerable: **true** // Object.keys
  - type: **['index', 'nonIndexString']**, enumerable: ***'all'*** // Object.getOwnPropertyNames

And even more examples

- Node.js
  - type: **['index']**, enumerable: **true** // Object.keys & filter
  - type: **['index', 'nonIndexString']**, enumerable: **true** // Object.keys
  - type: **['nonIndexString']**, enumerable: **true** // Object.keys & filter
  - type: **['symbol']**, enumerable: **true** // Object.getOwnPropertySymbols & filter
  - type: **['index', 'nonIndexString', 'symbol']**, enumerable: **true**
    // Object.getOwnPropertySymbols & Object.getOwnPropertyNames

Only production code, test code excluded & Possible fast paths included as examples.

Options with few real world examples

- Any option with **enumerable: false**
  - Needed to validate only enumerable properties, especially for symbols
  - Currently not found due to the lack of the API
- **Index properties**
  - Too costly to validate if other properties exist
  - Likely a frequent case as soon as this API exists
  - Difficult to determine correct behavior

Edge cases

Index properties

- Array indices
- TypedArray indices
- Indices on other objects

Prototype

```js
const obj = Object.create(null);
obj.property = 1;
Object.propertyCount(obj); // returns 1

const obj2 = { __proto__: null });
obj2.property = 1;
Object.propertyCount(obj2); // returns 1
```

Considerations

- Backwards compatibility
- Performance
- Simplicity
- Flexibility
- Map vs. Object

Next steps

- Getting input
- Addressing comments
- Stage 1 or 2?