ECMAScript Proposal: `Object.propertyCount`

Update

Champion: Ruben Bridgewater, Jordan Harband

Author: Ruben Bridgewater ruben@bridgewater.de

Stage: 1

Overview / Problem Statement

As described last time: Object.propertyCount proposal is mainly there to overcome *performance* and *correctness* issues in a variety of use cases / algorithms.

Use cases highlighted were:

1. Input validation (e.g., guarding against too big input)
2. Object comparison
3. Faster telemetry data
4. Testing utility
5. General fast paths for many different algorithms
6. Detecting non-index properties on array like objects
7. Sparse array detection

Precedent

Frequent patterns in widely-used JavaScript runtimes, frameworks, and libraries (Node.js, React, Angular, Lodash) demonstrate the common need for an optimized property counting mechanism.

The regular expression exec/match/matchAll methods produce a "match object" that is an Array, with non-index string properties on it (lastIndex, groups, etc).

Examples

Only enumerable symbols (needs the enumerable ones)

- *Next.js*
- *Angular*
- *TypeScript*
- *Lodash*

Any symbol (length only)

- *React*
- *Next.js*

Any non symbol property (length only)

- *React*

Examples 2

Any property (symbol & non-symbol; length only)

- *VS Code*

Any non-symbol reflects Object.keys depending on the enumerability

Array index (checks if it is a valid index)

- Lodash https://github.com/lodash/lodash/blob/main/dist/lodash.core.js#L1364-L1380
- Node.js assert.deepStrictEqual / partialDeepStrictEqual / ...
- Node.js console.log / util.inspect

Intermediate Conclusion

Frequent use cases next to the most common (enumerable strings) one are:

1. Symbol checks
2. Enumerability for symbols
3. Index properties

Simplified proposal

Separating specific use cases from the proposal:

- Detecting non-index properties on array like objects
- Sparse array detection

The new proposals all provide benefit on their own, while allowing to move this proposal forward independently.
That way the main use case is immediately addressed.
Combined, the proposals provide even more benefit by allowing to optimize even more algorithms.

Benefit

1. Differentiating non index string properties vs other string properties is difficult (TypedArray vs. Array).
2. Explicit dense / sparse array detection instead of implicit one

Explicit Semantics as before

- Only own properties are considered.
- Enumerability explicitly defined by the `enumerable` parameter.
- Avoids intermediate array allocation entirely when implemented natively.

Algorithmic Specification (details in spec proposal)

1. Initialize a numeric property counter to `0`.
2. Iterate directly over the object's own property descriptors
    - Access the internal property keys directly via the object's internal slots.
    - For each own property:
        - Determine if the key is a string or a symbol.
        - Check if the property type matches any specified in `keyTypes`.
        - If `enumerable` is not `'all'`, match the property's enumerability against the provided boolean value.
        - If the property meets all criteria, increment the counter.
3. Return the final count value.

Use Cases

- Improved readability and explicit intent
- Significant **performance** gains
- **Reduced memory** overhead
- **Simpler code**

Conclusion

`Object.propertyCount` offers substantial performance benefits by efficiently counting object properties, enhancing ECMAScript with clarity, performance, and reduced memory overhead.

Next steps

- Getting input about the separation
- Naming
- Addressing further comments
- Stage 2 or 2.7?