mdn web docs

Object.prototype.hasOwnProperty()

The **`hasOwnProperty()`** method returns a boolean indicating whether the object has the specified property as its own property (as opposed to inheriting it).

Try it

JavaScript Demo: Object.prototype.hasOwnProperty()

```
const object1 = {};
object1.property1 = 42;

console.log(object1.hasOwnProperty('property1'));
// expected output: true

console.log(object1.hasOwnProperty('toString'));
// expected output: false

console.log(object1.hasOwnProperty('hasOwnProperty'));
// expected output: false

```

Run › Reset

```
> true
> false
> false
```

Note: `Object.hasOwn()` is recommended over `hasOwnProperty()`, in browsers where it is supported.

Syntax

```
hasOwnProperty(prop)
```

Parameters

`prop`

The `String` name or Symbol of the property to test.

Return value

Returns `true` if the object has the specified property as own property; `false` otherwise.

Description

The **`hasOwnProperty()`** method returns `true` if the specified property is a direct property of the object — even if the value is `null` or `undefined`. The method returns `false` if the property is inherited, or has not been declared at all. Unlike the `in` operator, this method does not check for the specified property in the object's prototype chain.

The method can be called on *most* JavaScript objects, because most objects descend from `Object`, and hence inherit its methods. For example `Array` is an `Object`, so you can use `hasOwnProperty()` method to check whether an index exists:

```
const fruits = ['Apple', 'Banana','Watermelon', 'Orange'];
fruits.hasOwnProperty(3);   // true ('Orange')
fruits.hasOwnProperty(4);   // false - not defined
```

The method will not be available in objects where it is reimplemented, or on objects created using `Object.create(null)` (as these don't inherit from `Object.prototype`). Examples for these cases are given below.

Examples

Using hasOwnProperty to test for an own property's existence

The following code shows how to determine whether the `example` object contains a property named `prop`.

```
const example = {};
example.hasOwnProperty('prop');   // false

example.prop = 'exists';
example.hasOwnProperty('prop');   // true - 'prop' has been defined

example.prop = null;
example.hasOwnProperty('prop');   // true - own property exists with value of null

example.prop = undefined;
example.hasOwnProperty('prop');   // true - own property exists with value of undefined
```

Direct vs. inherited properties

The following example differentiates between direct properties and properties inherited through the prototype chain:

```
const example = {};
example.prop = 'exists';

// `hasOwnProperty` will only return true for direct properties:
example.hasOwnProperty('prop');             // returns true
example.hasOwnProperty('toString');         // returns false
example.hasOwnProperty('hasOwnProperty');   // returns false

// The `in` operator will return true for direct or inherited properties:
'prop' in example;                          // returns true
'toString' in example;                      // returns true
'hasOwnProperty' in example;                // returns true
```

Iterating over the properties of an object

The following example shows how to iterate over the enumerable properties of an object without executing on inherited properties.

```
const buz = {
  fog: 'stack',
};

for (const name in buz) {
  if (buz.hasOwnProperty(name)) {
    console.log(`this is fog (${name}) for sure. Value: ${buz[name]}`);
  } else {
    console.log(name); // toString or something else
  }
}
```

Note that the `for...in` loop only iterates enumerable items: the absence of non-enumerable properties emitted from the loop does not imply that `hasOwnProperty` itself is confined strictly to enumerable items (as with `Object.getOwnPropertyNames()`).

Using hasOwnProperty as a property name

JavaScript does not protect the property name `hasOwnProperty`; an object that has a property with this name may return incorrect results:

```
const foo = {
  hasOwnProperty() {
    return false;
  },
  bar: 'Here be dragons',
};

foo.hasOwnProperty('bar'); // reimplementation always returns false
```

The recommended way to overcome this problem is to instead use `Object.hasOwn()` (in browsers that support it). Other alternatives include using an *external* `hasOwnProperty`:

```
const foo = { bar: 'Here be dragons' };

// Use Object.hasOwn() method - recommended
Object.hasOwn(foo, "bar");  // true

// Use the hasOwnProperty property from the Object prototype
Object.prototype.hasOwnProperty.call(foo, 'bar'); // true

// Use another Object's hasOwnProperty
// and call it with 'this' set to foo
({}).hasOwnProperty.call(foo, 'bar'); // true
```

Note that in the first two cases there are no newly created objects.

Objects created with Object.create(null)

Objects created using `Object.create(null)` do not inherit from `Object.prototype`, making `hasOwnProperty()` inaccessible.

```
const foo = Object.create(null);
foo.prop = 'exists';
foo.hasOwnProperty("prop"); // Uncaught TypeError: foo.hasOwnProperty is not a function
```

The solutions in this case are the same as for the previous section: use `Object.hasOwn()` by preference, otherwise use an external object's `hasOwnProperty()`.

Specifications

Specification
ECMAScript Language Specification # sec-object.prototype.hasownproperty

Browser compatibility

Report problems with this compatibility data on GitHub

	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	Chrome Android	Firefox for Android	
`hasOwnProperty`	Chrome 1	Edge 12	Firefox 1	Internet Explorer 5.5	Opera 5	Safari 3	Chrome Android 18	Firefox for Android 4	Op An

Full support

See also

- `Object.hasOwn()`
- Enumerability and ownership of properties
- `Object.getOwnPropertyNames()`
- `for...in`
- `in`
- JavaScript Guide: Inheritance revisited

Last modified: Aug 9, 2022, by MDN contributors