Object.prototype.hasOwnProperty()

The **`hasOwnProperty()`** method returns a boolean indicating whether the object has the specified property.

Syntax

```
obj.hasOwnProperty(prop)
```

Parameters

`prop`

The name of the property to test.

Description

Every object descended from `Object` inherits the `hasOwnProperty` method. This method can be used to determine whether an object has the specified property as a direct property of that object; unlike the `in` operator, this method does not check down the object's prototype chain.

Examples

Using `hasOwnProperty` to test for a property's existence

The following example determines whether the `o` object contains a property named `prop`:

```
o = new Object();
o.prop = 'exists';

function changeO() {
  o.newprop = o.prop;
  delete o.prop;
}

o.hasOwnProperty('prop');   // returns true
changeO();
o.hasOwnProperty('prop');   // returns false
```

Direct versus inherited properties

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

```
o = new Object();
o.prop = 'exists';
o.hasOwnProperty('prop');             // returns true
o.hasOwnProperty('toString');         // returns false
o.hasOwnProperty('hasOwnProperty');   // returns false
```

Iterating over the properties of an object

The following example shows how to iterate over the properties of an object without executing on inherit properties. Note that the `for...in` loop is already only iterating enumerable items, so one should not assume based on the lack of non-enumerable properties shown in the loop that `hasOwnProperty` itself is confined strictly to enumerable items (as with `Object.getOwnPropertyNames()`).

```
var buz = {
  fog: 'stack'
};

```

```
for (var 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
  }
}
```

Using `hasOwnProperty` as a property name

JavaScript does not protect the property name `hasOwnProperty`; thus, if the possibility exists that an object might have a property with this name, it is necessary to use an *external* `hasOwnProperty` to get correct results:

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

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

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

// It's also possible to use the hasOwnProperty property from the Object prototype for this purpose
Object.prototype.hasOwnProperty.call(foo, 'bar'); // true
```

Note that in the last case there are no newly created objects.

Specifications

Specification	Status	Comment
ECMAScript 3rd Edition (ECMA-262)	ST Standard	Initial definition. Implemented in JavaScript 1.5.
ECMAScript 5.1 (ECMA-262) The definition of 'Object.prototype.hasOwnProperty' in that specification.	ST Standard	
ECMAScript 2015 (6th Edition, ECMA-262) The definition of 'Object.prototype.hasOwnProperty' in that specification.	ST Standard	

Browser compatibility

Desktop Mobile

Feature	Chrome	Firefox (Gecko)	Internet Explorer	Opera	Safari
Basic support	(Yes)	(Yes)	(Yes)	(Yes)	(Yes)

See also

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