mdn web docs

Object

The **`Object`** type represents one of JavaScript's data types. It is used to store various keyed collections and more complex entities. Objects can be created using the `Object()` constructor or the object initializer / literal syntax.

Description

Nearly all objects in JavaScript are instances of **`Object`**; a typical object inherits properties (including methods) from `Object.prototype`, although these properties may be shadowed (a.k.a. overridden). However, an `Object` may be deliberately created for which this is not true (e.g. by `Object.create(null)`), or it may be altered so that this is no longer true (e.g. with `Object.setPrototypeOf`).

Changes to the `Object` prototype object are seen by **all** objects through prototype chaining, unless the properties and methods subject to those changes are overridden further along the prototype chain. This provides a very powerful although potentially dangerous mechanism to override or extend object behavior.

The `Object` constructor's behavior depends on the input's type.

- If the value is `null` or `undefined`, it will create and return an empty object.
- If the value is an object already, it will return the value.
- Otherwise, it will return an object of a Type that corresponds to the given value.

When called in a non-constructor context, `Object` behaves identically to `new Object()`.

See also the object initializer / literal syntax.

Deleting a property from an object

There isn't any method in an Object itself to delete its own properties (such as `Map.prototype.delete()`). To do so, one must use the delete operator.

Constructor

`Object()`

Turns the input into an object.

Static methods

`Object.assign()`

Copies the values of all enumerable own properties from one or more source objects to a target object.

`Object.create()`

Creates a new object with the specified prototype object and properties.

`Object.defineProperty()`

Adds the named property described by a given descriptor to an object.

`Object.defineProperties()`

Adds the named properties described by the given descriptors to an object.

`Object.entries()`

Returns an array containing all of the `[key, value]` pairs of a given object's **own** enumerable string properties.

`Object.freeze()`

Freezes an object. Other code cannot delete or change its properties.

`Object.fromEntries()`

Returns a new object from an iterable of `[key, value]` pairs. (This is the reverse of `Object.entries`).

`Object.getOwnPropertyDescriptor()`

Returns a property descriptor for a named property on an object.

`Object.getOwnPropertyDescriptors()`

Returns an object containing all own property descriptors for an object.

`Object.getOwnPropertyNames()`

Returns an array containing the names of all of the given object's **own** enumerable and non-enumerable properties.

`Object.getOwnPropertySymbols()`

Returns an array of all symbol properties found directly upon a given object.

`Object.getPrototypeOf()`

Returns the prototype (internal `[[Prototype]]` property) of the specified object.

`Object.is()`

Compares if two values are the same value. Equates all `NaN` values (which differs from both `IsLooselyEqual` used by `==` and `IsStrictlyEqual` used by `===`).

`Object.isExtensible()`

Determines if extending of an object is allowed.

`Object.isFrozen()`

Determines if an object was frozen.

`Object.isSealed()`

Determines if an object is sealed.

`Object.keys()`

Returns an array containing the names of all of the given object's **own** enumerable string properties.

`Object.preventExtensions()`

Prevents any extensions of an object.

`Object.seal()`

Prevents other code from deleting properties of an object.

`Object.setPrototypeOf()`

Sets the object's prototype (its internal `[[Prototype]]` property).

`Object.values()`

Returns an array containing the values that correspond to all of a given object's **own** enumerable string properties.

Instance properties

`Object.prototype.constructor`

Specifies the function that creates an object's prototype.

`Object.prototype.__proto__`

Points to the object which was used as prototype when the object was instantiated.

Instance methods

`Object.prototype.__defineGetter__()`

Associates a function with a property that, when accessed, executes that function and returns its return value.

`Object.prototype.__defineSetter__()`

Associates a function with a property that, when set, executes that function which modifies the property.

`Object.prototype.__lookupGetter__()`

Returns the function associated with the specified property by the `__defineGetter__()` method.

`Object.prototype.__lookupSetter__()`

Returns the function associated with the specified property by the `__defineSetter__()` method.

`Object.prototype.hasOwnProperty()`

Returns a boolean indicating whether an object contains the specified property as a direct property of that object and not inherited through the prototype chain.

`Object.prototype.isPrototypeOf()`

Returns a boolean indicating whether the object this method is called upon is in the prototype chain of the specified object.

`Object.prototype.propertyIsEnumerable()`

Returns a boolean indicating if the internal ECMAScript [[Enumerable]] attribute is set.

`Object.prototype.toLocaleString()`

Calls `toString()`.

`Object.prototype.toString()`

Returns a string representation of the object.

`Object.prototype.valueOf()`

Returns the primitive value of the specified object.

Examples

Constructing empty objects

The following examples store an empty `Object` object in `o`:

```
const o1 = new Object();
const o2 = new Object(undefined);
const o3 = new Object(null);
```

Using Object to create Boolean objects

The following examples store `Boolean` objects in `o`:

```
// equivalent to const o = new Boolean(true)
const o = new Object(true);

// equivalent to const o = new Boolean(false)
const o = new Object(Boolean());
```

Object prototypes

When altering the behavior of existing `Object.prototype` methods, consider injecting code by wrapping your extension before or after the existing logic. For example, this (untested) code will pre-conditionally execute custom logic before the built-in logic or someone else's extension is executed.

When modifying prototypes with hooks, pass `this` and the arguments (the call state) to the current behavior by calling `apply()` on the function. This pattern can be used for any prototype, such as `Node.prototype`, `Function.prototype`, etc.

```
const current = Object.prototype.valueOf;

// Since my property "-prop-value" is cross-cutting and isn't always
// on the same prototype chain, I want to modify Object.prototype:
Object.prototype.valueOf = function (...args) {
  if (Object.hasOwn(this, '-prop-value')) {
    return this['-prop-value'];
  } else {
    // It doesn't look like one of my objects, so let's fall back on
    // the default behavior by reproducing the current behavior as best we can.
    // The apply behaves like "super" in some other languages.
    // Even though valueOf() doesn't take arguments, some other hook may.
    return current.apply(this, args);
```

```
  }
}
```

> **Warning:** Modifying the `prototype` property of any built-in constructor is considered a bad practice and risks forward compatibility.

You can read more about prototypes in Inheritance and the prototype chain.

Specifications

Specification
ECMAScript Language Specification # sec-object-objects

Browser compatibility

Report problems with this compatibility data on GitHub

	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	Chrome Android
`Object`	Chrome 1	Edge 12	Firefox 1	Internet Explorer 3	Opera 3	Safari 1	Chrome Android
`Object()` constructor	Chrome 1	Edge 12	Firefox 1	Internet Explorer 3	Opera 3	Safari 1	Chrome Android
`assign`	Chrome 45	Edge 12	Firefox 34	Internet Explorer No	Opera 32	Safari 9	Chrome Android
`constructor`	Chrome 1	Edge 12	Firefox 1	Internet Explorer 8	Opera 4	Safari 1	Chrome Android
`create`	Chrome 5	Edge 12	Firefox 4	Internet Explorer 9	Opera 11.6	Safari 5	Chrome Android
`__defineGetter__`	Chrome 1	Edge 12	Firefox 1	Internet Explorer 11	Opera 9.5	Safari 3	Chrome Android
`defineProperties`	Chrome 5	Edge 12	Firefox 4	Internet Explorer 9	Opera 11.6	Safari 5	Chrome Android
`defineProperty`	Chrome 5	Edge 12	Firefox 4	Internet Explorer 9	Opera 11.6	Safari 5.1	Chrome Android

	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	Chrome Android
__defineSetter__	Chrome 1	Edge 12	Firefox 1	Internet Explorer 11	Opera 9.5	Safari 3	Chrome Android
entries	Chrome 54	Edge 14	Firefox 47	Internet Explorer No	Opera 41	Safari 10.1	Chrome Android
freeze	Chrome 6	Edge 12	Firefox 4	Internet Explorer 9	Opera 12	Safari 5.1	Chrome Android
fromEntries	Chrome 73	Edge 79	Firefox 63	Internet Explorer No	Opera 60	Safari 12.1	Chrome Android
getOwnPropertyDescriptor	Chrome 5	Edge 12	Firefox 4	Internet Explorer 9	Opera 12	Safari 5	Chrome Android
getOwnPropertyDescriptors	Chrome 54	Edge 15	Firefox 50	Internet Explorer No	Opera 41	Safari 10	Chrome Android
getOwnPropertyNames	Chrome 5	Edge 12	Firefox 4	Internet Explorer 9	Opera 12	Safari 5	Chrome Android
getOwnPropertySymbols	Chrome 38	Edge 12	Firefox 36	Internet Explorer No	Opera 25	Safari 9	Chrome Android
getPrototypeOf	Chrome 5	Edge 12	Firefox 3.5	Internet Explorer 9	Opera 12.1	Safari 5	Chrome Android
hasOwn	Chrome 93	Edge 93	Firefox 92	Internet Explorer No	Opera 79	Safari 15.4	Chrome Android
hasOwnProperty	Chrome 1	Edge 12	Firefox 1	Internet Explorer 5.5	Opera 5	Safari 3	Chrome Android
is	Chrome 19	Edge 12	Firefox 22	Internet Explorer No	Opera 15	Safari 9	Chrome Android
isExtensible	Chrome 6	Edge 12	Firefox 4	Internet Explorer 9	Opera 12	Safari 5.1	Chrome Android
isFrozen	Chrome 6	Edge 12	Firefox 4	Internet Explorer 9	Opera 12	Safari 5.1	Chrome Android

	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	Chrome Android
isPrototypeOf	Chrome 1	Edge 12	Firefox 1	Internet Explorer 9	Opera 4	Safari 3	Chrome Android
isSealed	Chrome 6	Edge 12	Firefox 4	Internet Explorer 9	Opera 12	Safari 5.1	Chrome Android
keys	Chrome 5	Edge 12	Firefox 4	Internet Explorer 9	Opera 12	Safari 5	Chrome Android
__lookupGetter__	Chrome 1	Edge 12	Firefox 1	Internet Explorer 11	Opera 9.5	Safari 3	Chrome Android
__lookupSetter__	Chrome 1	Edge 12	Firefox 1	Internet Explorer 11	Opera 9.5	Safari 3	Chrome Android
preventExtensions	Chrome 6	Edge 12	Firefox 4	Internet Explorer 9	Opera 12	Safari 5.1	Chrome Android
ES2015 behavior for non-object argument	Chrome 44	Edge 12	Firefox 35	Internet Explorer 11	Opera 31	Safari 9	Chrome Android
propertyIsEnumerable	Chrome 1	Edge 12	Firefox 1	Internet Explorer 5.5	Opera 4	Safari 3	Chrome Android
__proto__	Chrome 1	Edge 12	Firefox 1	Internet Explorer 11	Opera 10.5	Safari 3	Chrome Android
seal	Chrome 6	Edge 12	Firefox 4	Internet Explorer 9	Opera 12	Safari 5.1	Chrome Android
setPrototypeOf	Chrome 34	Edge 12	Firefox 31	Internet Explorer 11	Opera 21	Safari 9	Chrome Android
toLocaleString	Chrome 1	Edge 12	Firefox 1	Internet Explorer 5.5	Opera 4	Safari 1	Chrome Android
toString()	Chrome 1	Edge 12	Firefox 1	Internet Explorer 3	Opera 3	Safari 1	Chrome Android
valueOf	Chrome 1	Edge 12	Firefox 1	Internet Explorer 4	Opera 3	Safari 1	Chrome Android

	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	Chrome Android
`values`	Chrome 54	Edge 14	Firefox 47	Internet Explorer No	Opera 41	Safari 10.1	Chrome Android

Full support Partial support No support Deprecated. Not for use in new websites. See implementation notes.

User must explicitly enable this feature.

See also

- Object initializer

Last modified: Aug 9, 2022, by MDN contributors