Object

Share: Twitter Facebook Google+

The **`Object`** constructor creates an object wrapper.

Syntax

```
// Object initialiser or literal
{ [ nameValuePair1[, nameValuePair2[, ...nameValuePairN] ] ] }

// Called as a constructor
new Object([value])
```

Parameters

`nameValuePair1, nameValuePair2, ... nameValuePairN`
Pairs of names (strings) and values (any value) where the name is separated from the value by a colon.

`value`
Any value.

Description

The `Object` constructor creates an object wrapper for the given value. If the value is `null` or `undefined`, it will create and return an empty object, otherwise, it will return an object of a Type that corresponds to the given value. If the value is an object already, it will return the value.

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

See also the object initializer / literal syntax.

Properties of the `Object` constructor

`Object.length`
Has a value of 1.

`Object.prototype`
Allows the addition of properties to all objects of type Object.

Methods of the `Object` constructor

`Object.assign()`
Creates a new object by copying 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 of a given object's own enumerable property `[key, value]` pairs.

`Object.freeze()`
Freezes an object: other code can't delete or change any properties.

`Object.getOwnPropertyDescriptor()`

Returns a property descriptor for a named property on 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 of the specified object.

`Object.is()`
: Compares if two values are distinguishable (ie. the same)

`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 properties.

`Object.observe()` ⚠
: Asynchronously observes changes to an object.

`Object.getNotifier()` ⚠
: Get a notifier with which to create object changes manually.

`Object.preventExtensions()`
: Prevents any extensions of an object.

`Object.seal()`
: Prevents other code from deleting properties of an object.

`Object.setPrototypeOf()`
: Sets the prototype (i.e., the internal `[[Prototype]]` property)

`Object.unobserve()` ⚠
: Unobserves changes to an object.

`Object.values()` ⚗
: Returns an array of a given object's own enumerable values.

`Object` instances and `Object` prototype object

All objects in JavaScript are descended from `Object`; all objects inherit methods and properties from `Object.prototype`, although they may be overridden. For example, other constructors' prototypes override the `constructor` property and provide their own `toString()` methods. Changes to the `Object` prototype object are propagated to all objects unless the properties and methods subject to those changes are overridden further along the prototype chain.

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.

`Object.prototype.__noSuchMethod__` ⚠
: Allows a function to be defined that will be executed when an undefined object member is called as a method.

`Object.prototype.__count__`
Used to return the number of enumerable properties directly on a user-defined object, but has been removed.

`Object.prototype.__parent__`
Used to point to an object's context, but has been removed.

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 indication whether the specified object is in the prototype chain of the object this method is called upon.

`Object.prototype.propertyIsEnumerable()`
Returns a boolean indicating if the internal ECMAScript [[Enumerable]] attribute is set.

`Object.prototype.toSource()`
Returns string containing the source of an object literal representing the object that this method is called upon; you can use this value to create a new object.

`Object.prototype.toLocaleString()`
Calls `toString()`.

`Object.prototype.toString()`
Returns a string representation of the object.

`Object.prototype.unwatch()`
Removes a watchpoint from a property of the object.

`Object.prototype.valueOf()`
Returns the primitive value of the specified object.

`Object.prototype.watch()`
Adds a watchpoint to a property of the object.

`Object.prototype.eval()`
Used to evaluate a string of JavaScript code in the context of the specified object, but has been removed.

Examples

Using `Object` given `undefined` and `null` types

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

```
var o = new Object();
```

```
var o = new Object(undefined);
```

```
var o = new Object(null);
```

Using `Object` to create `Boolean` objects

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

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

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

Specifications

Specification	Status	Comment
ECMAScript 1st Edition (ECMA-262)	ST Standard	Initial definition. Implemented in JavaScript 1.0.
ECMAScript 5.1 (ECMA-262) The definition of 'Object' in that specification.	ST Standard	
ECMAScript 2015 (6th Edition, ECMA-262) The definition of 'Object' in that specification.	ST Standard	Added Object.assign, Object.getOwnPropertySymbols, Object.setPrototypeOf
ECMAScript 2016 Draft (7th Edition, ECMA-262) The definition of 'Object' in that specification.	D Draft	Added Object.entries and Object.values.

Browser compatibility

Desktop Mobile

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

See also

- Object initializer