

Object.prototype.__proto__

- Deprecated: This feature is no longer recommended. Though some browsers might still support it, it may have already been removed from the relevant web standards, may be in the process of being dropped, or may only be kept for compatibility purposes. Avoid using it, and update existing code if possible; see the compatibility table at the bottom of this page to guide your decision. Be aware that this feature may cease to work at any time.
- Marning: Changing the [[Prototype]] of an object is, by the nature of how modern JavaScript engines optimize property accesses, currently a very slow operation in every browser and JavaScript engine. In addition, the effects of altering inheritance are subtle and far-flung, and are not limited to the time spent in the obj.__proto__ = ... statement, but may extend to *any* code that has access to any object whose [[Prototype]] has been altered. You can read more in <u>JavaScript engine fundamentals: optimizing prototypes</u> ☑.
- Note: The use of __proto__ is controversial and discouraged. Its existence and exact behavior have only been standardized as a legacy feature to ensure web compatibility, while it presents several security issues and footguns. For better support, prefer

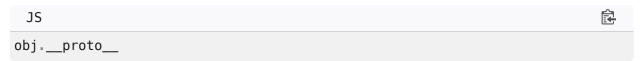
 Object.getPrototypeOf() / Reflect.getPrototypeOf() and

 Object.setPrototypeOf() / Reflect.setPrototypeOf() instead.

The __proto__ accessor property of <u>Object</u> instances exposes the <u>[[Prototype]]</u> (either an object or <u>null</u>) of this object.

The __proto__ property can also be used in an object literal definition to set the object [[Prototype]] on creation, as an alternative to <u>Object.create()</u>. See: <u>object initializer / literal syntax</u>. That syntax is standard and optimized for in implementations, and quite different from <code>Object.prototype.__proto__</code>.

Syntax



Return value

If used as a getter, returns the object's [[Prototype]].

Exceptions

<u>TypeError</u>

Thrown if attempting to set the prototype of a <u>non-extensible</u> object or an <u>immutable prototype exotic object</u>, such as <u>Object.prototype</u> or <u>window</u>.

Description

The __proto__ getter function exposes the value of the internal [[Prototype]] of an object. For objects created using an object literal (unless you use the prototype setter syntax), this value is <code>Object.prototype</code>. For objects created using array literals, this value is <code>Array.prototype</code>. For functions, this value is <code>Function.prototype</code>. You can read more about the prototype chain in <code>Inheritance</code> and the prototype chain.

The __proto__ setter allows the [[Prototype]] of an object to be mutated. The value provided must be an object or null. Providing any other value will do nothing.

Unlike Object.getPrototype0f(), which are always available on Object as static properties and always reflect the <a href="IPrototype]] internal property, the proto property doesn't always exist as a property on all objects, and as a result doesn't reflect IPrototype]] reliably.

The __proto__ property is just an accessor property on Object.prototype consisting of a getter and setter function. A property access for __proto__ that eventually consults Object.prototype will find this property, but an access that does not consult Object.prototype will not. If some other __proto__ property is found before Object.prototype is consulted, that property will hide the one found on Object.prototype.

```
null -prototype objects don't inherit any property from Object.prototype,
including the __proto__ accessor property, so if you try to read __proto__ on
such an object, the value is always undefined regardless of the object's actual
[[Prototype]], and any assignment to __proto__ would create a new property
called __proto__ instead of setting the object's prototype. Furthermore,
__proto__ can be redefined as an own property on any object instance through
Object.defineProperty() without triggering the setter. In this case, __proto__ will
no longer be an accessor for [[Prototype]]. Therefore, always prefer
Object.getPrototypeOf() and Object.setPrototypeOf() for setting and getting
the [[Prototype]] of an object.
```

Examples

Using __proto__

```
function Circle() {}
const shape = {};
const circle = new Circle();

// Set the object prototype.
// DEPRECATED. This is for example purposes only. DO NOT DO THIS in real code.
shape.__proto__ = circle;

// Get the object prototype
console.log(shape.__proto__ === Circle); // false
```

```
JS

const ShapeA = function () {};

const ShapeB = {
   a() {
```

```
console.log("aaa");
},

ShapeA.prototype.__proto__ = ShapeB;
console.log(ShapeA.prototype.__proto__); // { a: [Function: a] }

const shapeA = new ShapeA();
shapeA.a(); // aaa
console.log(ShapeA.prototype === shapeA.__proto__); // true
```

```
const ShapeC = function () {};
const ShapeD = {
    a() {
        console.log("a");
    },
};

const shapeC = new ShapeC();
shapeC.__proto__ = ShapeD;
shapeC.a(); // a
console.log(ShapeC.prototype === shapeC.__proto__); // false
```

```
function Test() {}
Test.prototype.myName = function () {
   console.log("myName");
};

const test = new Test();
console.log(test.__proto__ === Test.prototype); // true
test.myName(); // myName

const obj = {};
obj.__proto__ = Test.prototype;
obj.myName(); // myName
```

Specifications