instanceof

The **`instanceof` operator** tests whether the `prototype` property of a constructor appears anywhere in the prototype chain of an object.

JavaScript Demo: Expressions - instanceof

```
function Car(make, model, year) {
  this.make = make;
  this.model = model;
  this.year = year;
}
var auto = new Car('Honda', 'Accord', 1998);

console.log(auto instanceof Car);
// expected output: true

console.log(auto instanceof Object);
// expected output: true

```

Run ›

Reset

Syntax

> *object* instanceof *constructor*

Parameters

`object`
The object to test.

`constructor`
Function to test against

Description

The `instanceof` operator tests the presence of `constructor.prototype` in `object`'s prototype chain.

```
// defining constructors
function C() {}
function D() {}

var o = new C();

// true, because: Object.getPrototypeOf(o) === C.prototype
o instanceof C;

// false, because D.prototype is nowhere in o's prototype chain
o instanceof D;

o instanceof Object; // true, because:
C.prototype instanceof Object // true
```

```
C.prototype instanceof Object // true

C.prototype = {};
var o2 = new C();

o2 instanceof C; // true

// false, because C.prototype is nowhere in
// o's prototype chain anymore
o instanceof C;

D.prototype = new C(); // add C to [[Prototype]] linkage of D
var o3 = new D();
o3 instanceof D; // true
o3 instanceof C; // true since C.prototype is now in o3's prototype chain
```

Note that the value of an `instanceof` test can change based on changes to the `prototype` property of constructors, and it can also be changed by changing an object prototype using `Object.setPrototypeOf`. It is also possible using the non-standard `__proto__` pseudo-property.

`instanceof` and multiple context (e.g. frames or windows)

Different scopes have different execution environments. This means that they have different built-ins (different global object, different constructors, etc.). This may result in unexpected results. For instance, `[] instanceof window.frames[0].Array` will return `false`, because `Array.prototype !== window.frames[0].Array` and arrays inherit from the former.

This may not make sense at first but when you start dealing with multiple frames or windows in your script and pass objects from one context to another via functions, this will be a valid and

strong issue. For instance, you can securely check if a given object is, in fact, an Array using `Array.isArray(myObj)`

For example checking if a Nodes is a SVGElement in a different context you can use `myNode instanceof myNode.ownerDocument.defaultView.SVGElement`

> **Note for Mozilla developers:**
> In code using XPCOM `instanceof` has special effect: `obj instanceof xpcomInterface` (e.g. `Components.interfaces.nsIFile`) calls `obj.QueryInterface(xpcomInterface)` and returns `true` if QueryInterface succeeded. A side effect of such call is that you can use `xpcomInterface`'s properties on `obj` after a successful `instanceof` test. Unlike standard JavaScript globals, the test `obj instanceof xpcomInterface` works as expected even if `obj` is from a different scope.

Examples

Demonstrating that `String` and `Date` are of type `Object` and exceptional cases

The following code uses `instanceof` to demonstrate that `String` and `Date` objects are also of type `Object` (they are derived from `Object`).

However, objects created with the object literal notation are an exception here: Although the prototype is undefined, `instanceof Object` returns `true`.

```
var simpleStr = 'This is a simple string';
var myString  = new String();
var newStr    = new String('String created with constructor');
var myDate    = new Date();
var myObj     = {};
var myNonObj  = Object.create(null);

simpleStr instanceof String; // returns false, checks the prototype chain, finds undefined
myString  instanceof String; // returns true
newStr    instanceof String; // returns true
myString  instanceof Object; // returns true

myObj    instanceof Object;    // returns true, despite an undefined prototype
({})     instanceof Object;    // returns true, same case as above
myNonObj instanceof Object;    // returns false, a way to create an object that is not an instance of Ob

myString instanceof Date;   // returns false

myDate instanceof Date;     // returns true
myDate instanceof Object;   // returns true
myDate instanceof String;   // returns false
```

Demonstrating that `mycar` is of type `Car` and type `Object`

The following code creates an object type `Car` and an instance of that object type, `mycar`. The `instanceof` operator demonstrates that the `mycar` object is of type `Car` and of type `Object`.

```
function Car(make, model, year) {
```

```
  this.make = make;
  this.model = model;
  this.year = year;
}
var mycar = new Car('Honda', 'Accord', 1998);
var a = mycar instanceof Car;    // returns true
var b = mycar instanceof Object; // returns true
```

Not an instanceof

To test if an object is not an instanceof a specific Constructor, you can do

```
if (!(mycar instanceof Car)) {
  // Do something, like mycar = new Car(mycar)
}
```

This is really different from

```
if (!mycar instanceof Car)
```

that will always be false (`!mycar` will be treated before `instanceof`, so you always try to know if a boolean is an instance of `Car`).

Specifications

Specification	Status	Comment
ECMAScript Latest Draft (ECMA-262) The definition of 'Relational Operators' in that specification.	D Draft	
ECMAScript 2015 (6th Edition, ECMA-262) The definition of 'Relational Operators' in that specification.	ST Standard	
ECMAScript 5.1 (ECMA-262) The definition of 'The instanceof operator' in that specification.	ST Standard	
ECMAScript 3rd Edition (ECMA-262) The definition of 'The instanceof operator' in that specification.	ST Standard	Initial definition. Implemented in JavaScript 1.4.

Browser compatibility

Update compatibility data on GitHub

`instanceof`

Chrome	Yes
Edge	Yes
Firefox	1
IE	Yes
Opera	Yes
Safari	Yes

WebView Android	Yes
Chrome Android	Yes
Firefox Android	4
Opera Android	Yes
Safari iOS	Yes
Samsung Internet Android	Yes
nodejs	Yes

Flag as incorrect

Full support

See also

- `typeof`
- `Symbol.hasInstance`