super

Jump to: Syntax Description Example Specifications Browser compatibility See also

The **super** keyword is used to access and call functions on an object's parent.

The `super.prop` and `super[expr]` expressions are valid in any method definition in both classes and object literals.

Syntax

```
super([arguments]); // calls the parent constructor.
super.functionOnParent([arguments]);
```

Description

When used in a constructor, the `super` keyword appears alone and must be used before the `this` keyword is used. The `super` keyword can also be used to call functions on a parent object.

Example

Using `super` in classes

This code snippet is taken from the classes sample (live demo). Here `super()` is called to avoid duplicating the constructor parts' that are common between `Rectangle` and `Square`.

```
class Rectangle {
  constructor(height, width) {
    this.name = 'Rectangle';
    this.height = height;
    this.width = width;
  }
  sayName() {
    console.log('Hi, I am a ', this.name + '.');
  }
  get area() {
    return this.height * this.width;
  }
  set area(value) {
    this.height = this.width = Math.sqrt(value);
  }
}

class Square extends Rectangle {
  constructor(length) {
    this.height; // ReferenceError, super needs to be called first!

    // Here, it calls the parent class' constructor with lengths
```

```
    // provided for the Rectangle's width and height
    super(length, length);

    // Note: In derived classes, super() must be called before you
    // can use 'this'. Leaving this out will cause a reference error.
    this.name = 'Square';
  }
}
```

Super-calling static methods

You are also able to call super on static methods.

```
class Rectangle {
  constructor() {}
  static logNbSides() {
    return 'I have 4 sides';
  }
}

class Square extends Rectangle {
  constructor() {}
  static logDescription() {
    return super.logNbSides() + ' which are all equal';
  }
}
Square.logDescription(); // 'I have 4 sides which are all equal'
```

Deleting super properties will throw an error

You cannot use the delete operator and `super.prop` or `super[expr]` to delete a parent class' property, it will throw a `ReferenceError`.

```
class Base {
  constructor() {}
  foo() {}
}
class Derived extends Base {
  constructor() {}
  delete() {
    delete super.foo; // this is bad
  }
}

new Derived().delete(); // ReferenceError: invalid delete involving 'super'.
```

`super.prop` cannot overwrite non-writable properties

When defining non-writable properties with e.g. `Object.defineProperty`, `super` cannot overwrite the value of the property.

```
class X {
  constructor() {
    Object.defineProperty(this, 'prop', {
      configurable: true,
      writable: false,
      value: 1
    });
  }
}

class Y extends X {
  constructor() {
    super();
  }
  foo() {
    super.prop = 2;   // Cannot overwrite the value.
  }
}

var y = new Y();
y.foo(); // TypeError: "prop" is read-only
console.log(y.prop); // 1
```

Using `super.prop` in object literals

Super can also be used in the object initializer / literal notation. In this example, two objects define a method. In the second object, `super` calls the first object's method. This works with the help of `Object.setPrototypeOf()` with which we are able to set the prototype of `obj2` to `obj1`, so that `super` is able to find `method1` on `obj1`.

```
var obj1 = {
  method1() {
    console.log('method 1');
  }
}

var obj2 = {
  method2() {
   super.method1();
  }
}

Object.setPrototypeOf(obj2, obj1);
obj2.method2(); // logs "method 1"
```

Specifications

Specification	Status	Comment
ECMAScript 2015 (6th Edition, ECMA-262) The definition of 'super' in that specification.	ST Standard	Initial definition.
ECMAScript Latest Draft (ECMA-262) The definition of 'super' in that specification.	D Draft	

Browser compatibility

New compatibility tables are in beta

Basic support													
42	Yes	45	No	Yes	Yes	42	42	Yes	45	Yes	Yes	4.0	Yes

Full support No support

See also

- Classes