

Proxy



- □□
- Proxy □□□□
 - get()
 - set()
 - apply()
 - has()
 - construct()
 - deleteProperty()
 - defineProperty()
 - getOwnPropertyDescriptor()
 - getPrototypeOf()
 - isExtensible()
 - ownKeys()
 - preventExtensions()
 - setPrototypeOf()
- Proxy.revocable()
- this □□
- □□ Web □□□□



Proxy

programming

“” meta

Proxy [] " "

[]

“ ”

Proxy

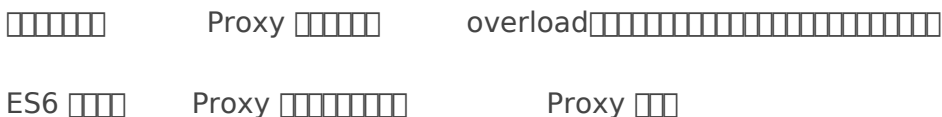
“ ”

“ ”

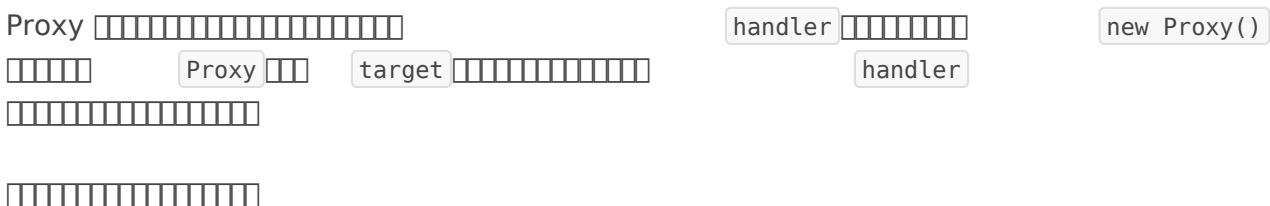
```
var obj = new Proxy({}, {
  get: function (target, propKey, receiver) {
    console.log(`getting ${propKey}!`);
    return Reflect.get(target, propKey, receiver);
  },
  set: function (target, propKey, value, receiver) {
    console.log(`setting ${propKey}!`);
    return Reflect.set(target, propKey, value, receiver);
  }
});
```



```
obj.count = 1
// setting count!
++obj.count
// getting count!
// setting count!
// 2
```



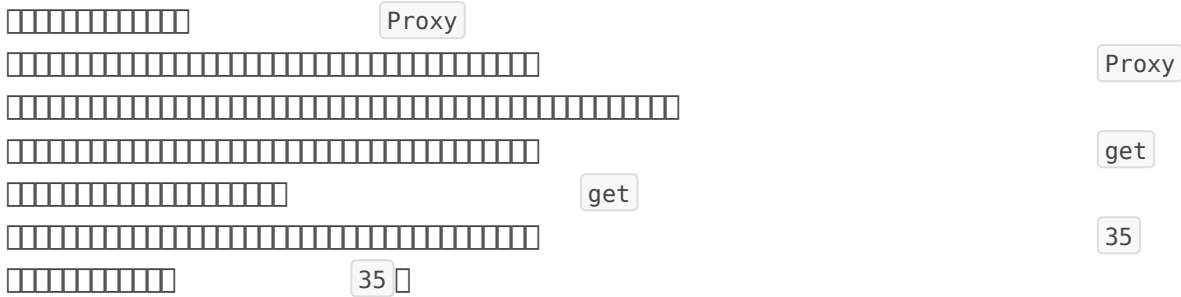
```
var proxy = new Proxy(target, handler);
```



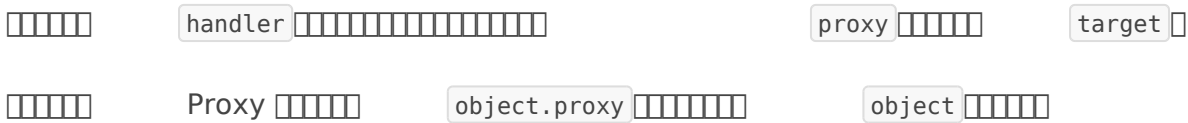
```
var proxy = new Proxy({}, {
  get: function(target, propKey) {
    return 35;
  }
});
```

```
proxy.time // 35
proxy.name // 35
```

```
proxy.title // 35
```



```
var target = {};  
var handler = {};  
var proxy = new Proxy(target, handler);  
proxy.a = 'b';  
target.a // "b"
```

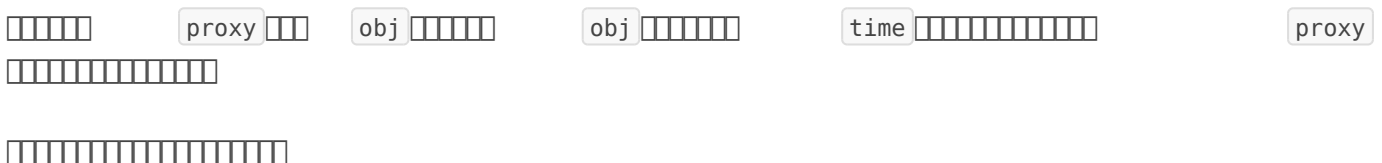


```
var object = { proxy: new Proxy(target, handler) };
```

Proxy

```
var proxy = new Proxy({}, {  
  get: function(target, propKey) {  
    return 35;  
  }  
});
```

```
let obj = Object.create(proxy);  
obj.time // 35
```



```
var handler = {
  get: function(target, name) {
    if (name === 'prototype') {
      return Object.prototype;
    }
    return 'Hello, ' + name;
  },

  apply: function(target, thisBinding, args) {
    return args[0];
  },

  construct: function(target, args) {
    return {value: args[1]};
  }
};

var fproxy = new Proxy(function(x, y) {
  return x + y;
}, handler);

fproxy(1, 2) // 1
new fproxy(1, 2) // {value: 2}
fproxy.prototype === Object.prototype // true
fproxy.foo === "Hello, foo" // true
```

[illegible]

Proxy 13

- **get(target, propKey, receiver)** `proxy.foo` `proxy['foo']`
- **set(target, propKey, value, receiver)** `proxy.foo = v`
- **has(target, propKey)** `propKey in proxy`
- **deleteProperty(target, propKey)** `delete proxy[propKey]`
- **ownKeys(target)** `Object.getOwnPropertyNames(proxy)`
`Object.getOwnPropertySymbols(proxy)` `Object.keys(proxy)` `for...in`
`Object.keys()`
- **getOwnPropertyDescriptor(target, propKey)**
`Object.getOwnPropertyDescriptor(proxy, propKey)`

- **defineProperty(target, propKey, propDesc)** `Object.defineProperty(proxy, propKey, propDesc)`
- **preventExtensions(target)** `Object.preventExtensions(proxy)`
- **getPrototypeOf(target)** `Object.getPrototypeOf(proxy)`
- **isExtensible(target)** `Object.isExtensible(proxy)`
- **setPrototypeOf(target, proto)** `Object.setPrototypeOf(proxy, proto)`
- **apply(target, object, args)** `Proxy` `proxy(...args)`
`proxy.call(object, ...args)` `proxy.apply(...)`
- **construct(target, args)** `Proxy` `new proxy(...args)`

Proxy

get()

get `proxy`

get

```
var person = {
  name: ""
};

var proxy = new Proxy(person, {
  get: function(target, propKey) {
    if (propKey in target) {
      return target[propKey];
    } else {
      throw new ReferenceError("Prop name \" + propKey + "\" does not exist.");
    }
  }
});

proxy.name // ""
proxy.age // 
```

undefined

```
let proto = new Proxy({}, {
  get(target, propertyKey, receiver) {
    console.log('GET ' + propertyKey);
    return target[propertyKey];
  }
});
```

obj

`get`

```
let target = [];  
target.push(...elements);  
return new Proxy(target, handler);  
}
```

[illegible]

-1

get

get DOM dom

```
const dom = new Proxy({}, {
  get(target, property) {
    return function(attrs = {}, ...children) {
      const el = document.createElement(property);
      for (let prop of Object.keys(attrs)) {
        el.setAttribute(prop, attrs[prop]);
      }
      for (let child of children) {
        if (typeof child === 'string') {
          child = document.createTextNode(child);
        }
        el.appendChild(child);
      }
    }
  }
});
```

```

        return el;
    }
}

});

const el = dom.div({},
    'Hello, my name is ',
    dom.a({href: '//example.com'}, 'Mark'),
    '. I like:',
    dom.ul({},
        dom.li({}, 'The web'),
        dom.li({}, 'Food'),
        dom.li({}, '...actually that\'s it')
    )
);

document.body.appendChild(el);

```

The diagram shows a Proxy object. It has a 'get' method (represented by a box with the text 'get') and a 'value' property (represented by a box with the text 'value'). The 'value' property is linked to a long array of 20 empty square boxes, representing an array of 20 elements. The entire Proxy object is enclosed in a larger box labeled 'Proxy'.

```
const proxy = new Proxy({}, {
  get: function(target, key, receiver) {
    return receiver;
  }
});

proxy.getReceiver === proxy // true
```

```

graph LR
    Proxy[proxy] -- getReceiver --> Receiver[getReceiver]
    Receiver -- get --> Proxy
    Proxy -- get --> Receiver
    Receiver -- get --> Proxy
  
```

```
const proxy = new Proxy({}, {
  get: function(target, key, receiver) {
    return receiver;
  }
});

const d = Object.create(proxy);

d.a === d // true
```

configurable
writable

configurable

writable

Proxy

Proxy

```
const target = Object.defineProperties({}, {
  foo: {
    value: 123,
    writable: false,
    configurable: false
  },
});
```

```
const handler = {
  get(target, propKey) {
    return 'abc';
  }
};
```

```
const proxy = new Proxy(target, handler);
```

```
proxy.foo
// TypeError: Invariant check failed
```

set()

set

Proxy

Person

age

200

Proxy

age

```
let validator = {
  set: function(obj, prop, value) {
    if (prop === 'age') {
      if (!Number.isInteger(value)) {
        throw new TypeError('The age is not an integer');
      }
      if (value > 200) {
        throw new RangeError('The age seems invalid');
      }
    }
  }
}
```

```

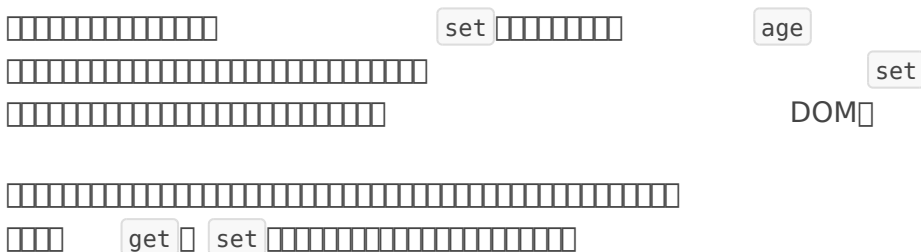
    // 验证 age 属性是否合法
    obj[prop] = value;
    return true;
  }
};

let person = new Proxy({}, validator);

person.age = 100;

person.age // 100
person.age = 'young' // 成功
person.age = 300 // 失败

```



```

const handler = {
  get (target, key) {
    invariant(key, 'get');
    return target[key];
  },
  set (target, key, value) {
    invariant(key, 'set');
    target[key] = value;
    return true;
  }
};

function invariant (key, action) {
  if (key[0] === '_') {
    throw new Error(`Invalid attempt to ${action} private "${key}" property`);
  }
}

const target = {};
const proxy = new Proxy(target, handler);
proxy._prop

```

```
// Error: Invalid attempt to get private "_prop" property
proxy._prop = 'c'
// Error: Invalid attempt to set private "_prop" property
```

[illegible]

Diagram illustrating a set structure. A box labeled "set" is connected to a row of 10 empty boxes, representing the elements of the set.

```
const handler = {
  set: function(obj, prop, value, receiver) {
    obj[prop] = value;
    return true;
  }
};

const proxy = new Proxy({}, handler);

proxy.foo = 'bar';

proxy.foo === proxy // true
```

```

graph LR
    set[set: 5 slots] --> receiver[receiver: 15 slots]
    proxy[proxy] -- set --> set
  
```

```
const handler = {
  set: function(obj, prop, value, receiver) {
    obj[prop] = value;
    return true;
  }
};

const proxy = new Proxy({}, handler);
const myObj = {};

Object.setPrototypeOf(myObj, proxy);

myObj.foo = 'bar';
myObj.foo === myObj // true
```

_____ set _____

```
const obj = {};  
Object.defineProperty(obj, 'foo', {
```

```

    value: 'bar',
    writable: false
  });

  const handler = {
    set: function(obj, prop, value, receiver) {
      obj[prop] = 'baz';
      return true;
    }
  };

  const proxy = new Proxy(obj, handler);
  proxy.foo = 'baz';
  proxy.foo // "bar"

```

obj.foo Proxy set set true

```

'use strict';
const handler = {
  set: function(obj, prop, value, receiver) {
    obj[prop] = receiver;
    // 阻止设置
    return false;
  }
};

const proxy = new Proxy({}, handler);
proxy.foo = 'bar';
// TypeError: 'set' on proxy: trap returned falsish for property 'foo'

```

set false undefined

apply()

apply call apply apply this

```
var handler = {
  apply (target, ctx, args) {
    return Reflect.apply(...arguments);
  }
};
```

□□□□□□□□

```
var target = function () { return 'I am the target'; };
var handler = {
  apply: function () {
    return 'I am the proxy';
  }
};

var p = new Proxy(target, handler);

p()
// "I am the proxy"
```

□□□□□□□□

p □ Proxy □□□□□□□□□□

p() □□□□

apply

□□□□□□□□□□

□□□□□□□□

```
var twice = {
  apply (target, ctx, args) {
    return Reflect.apply(...arguments) * 2;
  }
};

function sum (left, right) {
  return left + right;
};

var proxy = new Proxy(sum, twice);
proxy(1, 2) // 6
proxy.call(null, 5, 6) // 22
proxy.apply(null, [7, 8]) // 30
```

□□□□□□□□

proxy □□□□□□□□

call □ apply □□□□□□

apply □□□□

□□□□□□

Reflect.apply □□□□□□□□

```
Reflect.apply(proxy, null, [9, 10]) // 38
```

has()

has() ☐

HasProperty ☐

in ☐

has() ☐

☐

has() ☐

in ☐

```
var handler = {  
  has (target, key) {  
    if (key[0] === '_') {  
      return false;  
    }  
    return key in target;  
  }  
};  
var target = { _prop: 'foo', prop: 'foo' };  
var proxy = new Proxy(target, handler);  
'_prop' in proxy // false
```

☐

☐

proxy.has() ☐

false ☐

in ☐

☐

has() ☐

```
var obj = { a: 10 };  
Object.preventExtensions(obj);  
  
var p = new Proxy(obj, {  
  has: function(target, prop) {  
    return false;  
  }  
});  
  
'a' in p // TypeError is thrown
```

☐

obj ☐

has ☐

☐

has() ☐

“☐ ”

false

has()

HasProperty

HasOwnProperty

has()

for...in

in

has()

for...in

```
let stu1 = {name: '张三', score: 59};
let stu2 = {name: '李四', score: 99};

let handler = {
  has(target, prop) {
    if (prop === 'score' && target[prop] < 60) {
      console.log(`${target.name} 访问`);
      return false;
    }
    return prop in target;
  }
}
```

```
let oproxy1 = new Proxy(stu1, handler);
let oproxy2 = new Proxy(stu2, handler);
```

```
'score' in oproxy1
// 访问
// false
```

```
'score' in oproxy2
// true
```

```
for (let a in oproxy1) {
  console.log(oproxy1[a]);
}
// 访问
// 59
```

```
for (let b in oproxy2) {
  console.log(oproxy2[b]);
}
// 访问
// 99
```

`has()` `in` `for...in` `for...in`

construct()

`construct()` `new`

```
const handler = {
  construct (target, args, newTarget) {
    return new target(...args);
  }
};
```

`construct()`

- `target`
- `args`
- `newTarget` `new` `p`

```
const p = new Proxy(function () {}, {
  construct: function(target, args) {
    console.log('called: ' + args.join(', '));
    return { value: args[0] * 10 };
  }
});

(new p(1)).value
// "called: 1"
// 10
```

`construct()`

```
const p = new Proxy(function() {}, {
  construct: function(target, argumentsList) {
    return 1;
  }
});

new p() // 
// Uncaught TypeError: 'construct' on proxy: trap returned non-object ('1')
```


$$\left. \begin{array}{l} \{ \\ \{ \end{array} \right\}$$

```
var target = { _prop: 'foo' };
var proxy = new Proxy(target, handler);
delete proxy._prop
// Error: Invalid attempt to delete private "_prop" property
```

```

graph LR
    subgraph Initial_Object [Initial Object]
        direction TB
        A["a: 1"]
        B["b: 2"]
        C["c: 3"]
    end
    subgraph Deletion_Process [Deletion Process]
        direction TB
        DP["deleteProperty(c)"]
        D["delete"]
    end
    subgraph Resulting_Object [Resulting Object]
        direction TB
        A2["a: 1"]
        B2["b: 2"]
        C2["c: (deleted)"]
    end
    subgraph Configurable_Property [Configurable Property]
        direction TB
        CP["configurable"]
        DCP["deleteProperty"]
    end

```

defineProperty()

`defineProperty()` ☐☐☐☐ `Object.defineProperty()` ☐☐

```
var handler = {
  defineProperty (target, key, descriptor) {
    return false;
  }
};

var target = {};

var proxy = new Proxy(target, handler);

proxy.foo = 'bar' // 成功
```

```

{
  a: 1,
  b: false
}

```

The diagram illustrates three arrays with different properties:

- Array 1:** A short array (8 slots) labeled `non-extensible`. A `defineProperty()` call is shown with a red 'X' over it, indicating it fails.
- Array 2:** A long array (20 slots) labeled `writable`. A `defineProperty()` call is shown with a green checkmark over it, indicating it succeeds.
- Array 3:** A short array (5 slots) labeled `configurable`. A `defineProperty()` call is shown with a green checkmark over it, indicating it succeeds.

getOwnPropertyDescriptor()

getOwnPropertyDescriptor()  Object.getOwnPropertyDescriptor() 
undefined 

```
var handler = {
  getOwnPropertyDescriptor (target, key) {
    if (key[0] === '_') {
```

```

        return;
    }
    return Object.getOwnPropertyDescriptor(target, key);
}
};
var target = { _foo: 'bar', baz: 'tar' };
var proxy = new Proxy(target, handler);
Object.getOwnPropertyDescriptor(proxy, 'wat')
// undefined
Object.getOwnPropertyDescriptor(proxy, '_foo')
// undefined
Object.getOwnPropertyDescriptor(proxy, 'baz')
// { value: 'tar', writable: true, enumerable: true, configurable: true }

```

□□□□□
handler.getOwnPropertyDescriptor()
□□□□□□□□□□□□□□□□□□□□
undefined

getPrototypeOf()

getPrototypeOf()
□□□□□□□□□□□□□□□□□□□□

- Object.prototype.__proto__
- Object.prototype.isPrototypeOf()
- Object.getPrototypeOf()
- Reflect.getPrototypeOf()
- instanceof

□□□□□□□

```

var proto = {};
var p = new Proxy({}, {
  getPrototypeOf(target) {
    return proto;
  }
});
Object.getPrototypeOf(p) === proto // true

```

□□□□□
getPrototypeOf()
□□□□
Object.getPrototypeOf()
□□□
proto
□□□

□□□
getPrototypeOf()
□□□□□□□□□□□□
null
□□□□□□□□□□□□□□□□□□□□

non-extensible
□□
getPrototypeOf()
□□□□□□□□□□□□□□□□□□□□

isExtensible()

isExtensible() 

Object.isExtensible() 

```
var p = new Proxy({}, {  
  isExtensible: function(target) {  
    console.log("called");  
    return true;  
  }  
});
```

```
Object.isExtensible(p)  
// "called"  
// true
```



isExtensible() 

Object.isExtensible 

called 





isExtensible 

```
Object.isExtensible(proxy) === Object.isExtensible(target)
```



```
var p = new Proxy({}, {  
  isExtensible: function(target) {  
    return false;  
  }  
});
```

```
Object.isExtensible(p)  
// Uncaught TypeError: 'isExtensible' on proxy: trap result does not reflect extensibility of  
proxy target (which is 'true')
```

ownKeys()

ownKeys() 

- `Object.getOwnPropertyNames()`

- `Object.getOwnPropertySymbols()`
- `Object.keys()`
- `for...in`

Object.keys()

```
let target = {
  a: 1,
  b: 2,
  c: 3
};

let handler = {
  ownKeys(target) {
    return ['a'];
  }
};

let proxy = new Proxy(target, handler);

Object.keys(proxy)
// [ 'a' ]
```

target Object.keys() a b c a

```
let target = {
  _bar: 'foo',
  _prop: 'bar',
  prop: 'baz'
};

let handler = {
  ownKeys(target) {
    return Reflect.ownKeys(target).filter(key => key[0] !== '_');
  }
};

let proxy = new Proxy(target, handler);

for (let key of Object.keys(proxy)) {
```

```

    console.log(target[key]);
  }
  // "baz"

```

□□□□

Object.keys() □□□□□□□□

ownKeys() □□□□□□□□

- □□□□□□□□
- □□□ Symbol □
- □□□□ enumerable □□□□

```

let target = {
  a: 1,
  b: 2,
  c: 3,
  [Symbol.for('secret')]: '4',
};

Object.defineProperty(target, 'key', {
  enumerable: false,
  configurable: true,
  writable: true,
  value: 'static'
});

let handler = {
  ownKeys(target) {
    return ['a', 'd', Symbol.for('secret'), 'key'];
  }
};

let proxy = new Proxy(target, handler);

Object.keys(proxy)
// ['a']

```

□□□□

ownKeys() □□□□□□□□□□

□□□□□□□□

key □□□□□□□□

d □□ Symbol □□

Symbol.for('secret')

ownKeys() □□□□□□

Object.getOwnPropertyNames() □

```
var p = new Proxy({}, {
  ownKeys: function(target) {
    return ['a', 'b', 'c'];
  }
});
```

```
Object.getOwnPropertyNames(p)
// [ 'a', 'b', 'c' ]
```

for...in 

ownKeys() 

```
const obj = { hello: 'world' };
const proxy = new Proxy(obj, {
  ownKeys: function () {
    return ['a', 'b'];
  }
});

for (let key in proxy) {
  console.log(key); // 
}
```



ownkeys() 

a  b 

obj 

for...in



ownKeys() 

Symbol



```
var obj = {};

var p = new Proxy(obj, {
  ownKeys: function(target) {
    return [123, true, undefined, null, {}, []];
  }
});

Object.getOwnPropertyNames(p)
// Uncaught TypeError: 123 is not a valid property name
```



ownKeys() 

Symbol

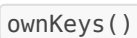


ownKeys() 

```
var obj = {};  
Object.defineProperty(obj, 'a', {  
  configurable: false,  
  enumerable: true,  
  value: 10 }  
);  
  
var p = new Proxy(obj, {  
  ownKeys: function(target) {  
    return ['b'];  
  }  
});  
  
Object.getOwnPropertyNames(p)  
// Uncaught TypeError: 'ownKeys' on proxy: trap result did not include 'a'
```

  obj  a  ownKeys()  a



 non-extensible  ownKeys() 



```
var obj = {  
  a: 1  
};  
  
Object.preventExtensions(obj);  
  
var p = new Proxy(obj, {  
  ownKeys: function(target) {  
    return ['a', 'b'];  
  }  
});  
  
Object.getOwnPropertyNames(p)  
// Uncaught TypeError: 'ownKeys' on proxy: trap returned extra keys but proxy target is non-extensible
```

b

setPrototypeOf()

--	--	--	--	--	--	--	--

--	--	--

--	--	--	--	--	--	--	--

```
var handler = {
  setPrototypeOf(target, proto) {
    throw new Error('Changing the prototype is forbidden');
  }
};

var proto = {};

var target = function () {};

var proxy = new Proxy(target, handler);

Object.setPrototypeOf(proxy, proto);

// Error: Changing the prototype is forbidden
```

[illegible][illegible][illegible]

non-extensible

☐ `setPrototypeOf()` ☐

Proxy.revocable()

Proxy.revocable()

Proxy ☐ ☐ ☐

```
let target = {};  
  
let handler = {};  
  
let {proxy, revoke} = Proxy.revocable(target, handler);  
  
proxy.foo = 123;  
  
proxy.foo // 123  
  
revoke();  
  
proxy.foo // TypeError: Revoked
```

[illegible]

proxy ☐ ☐ ☐

Proxy ☐ ☐ ☐

revoke

[illegible]

Proxy									
-------	--	--	--	--	--	--	--	--	--

[illegible]

Proxy

[illegible]

Proxy.revocable()

██

████

this

Proxy

██

████████████████████

Proxy ████████████████████

this █████

Proxy

```
const target = {
  m: function () {
    console.log(this === proxy);
  }
};

const handler = {};

const proxy = new Proxy(target, handler);

target.m() // false
proxy.m()  // true
```

████████ proxy target target.m() this proxy target
proxy proxy target.m() proxy.m()

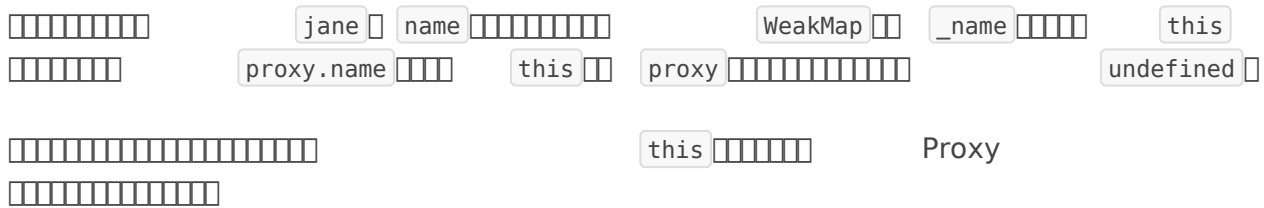
████████ this Proxy

```
const _name = new WeakMap();

class Person {
  constructor(name) {
    _name.set(this, name);
  }
  get name() {
    return _name.get(this);
  }
}

const jane = new Person('Jane');
jane.name // 'Jane'
```

```
const proxy = new Proxy(jane, {});
proxy.name // undefined
```



```
const target = new Date();
const handler = {};
const proxy = new Proxy(target, handler);

proxy.getDate();
// TypeError: this is not a Date object.
```



```
const target = new Date('2015-01-01');
const handler = {
  get(target, prop) {
    if (prop === 'getDate') {
      return target.getDate.bind(target);
    }
    return Reflect.get(target, prop);
  }
};
const proxy = new Proxy(target, handler);

proxy.getDate() // 1
```



```
const handler = {
  get: function (target, key, receiver) {
    console.log(this === handler);
    return 'Hello, ' + key;
  },
  set: function (target, key, value) {
```

□□□□□ get() □ set() □□□□□ this □□□□□ handler □□□

Proxy [] Web []

Web: 8 buffers (top row), 8 buffers (bottom row). Proxy: 8 buffers.

☐ Proxy ☐ ORM ☐

□□ #1

□ □□ □□ 2020-04-16 10:29:52 UTC

□ □□ □□ 2023-08-24 14:16:36 UTC