# [ES6新特性：Javascript中的Reflect对象](https://www.cnblogs.com/diligenceday/p/5474126.html)

**阅读目录**

* [Reflect介绍：](https://www.cnblogs.com/diligenceday/p/5474126.html" \l "_label0)
* [什么要使用Reflect](https://www.cnblogs.com/diligenceday/p/5474126.html" \l "_label1)
* [Reflect.apply的使用](https://www.cnblogs.com/diligenceday/p/5474126.html" \l "_label2)
* [Reflect.construct()的使用：](https://www.cnblogs.com/diligenceday/p/5474126.html" \l "_label3)
* [Reflect.defineProperty的使用;](https://www.cnblogs.com/diligenceday/p/5474126.html" \l "_label4)
* [Reflect.deleteProperty的使用：](https://www.cnblogs.com/diligenceday/p/5474126.html" \l "_label5)
* [Reflect.get()方法的使用；](https://www.cnblogs.com/diligenceday/p/5474126.html" \l "_label6)
* [Reflect.getOwnPropertyDescritptor()方法的使用：](https://www.cnblogs.com/diligenceday/p/5474126.html" \l "_label7)
* [Reflect.getPrototypeOf()方法的使用：](https://www.cnblogs.com/diligenceday/p/5474126.html" \l "_label8)
* [Reflect.has的使用](https://www.cnblogs.com/diligenceday/p/5474126.html" \l "_label9)
* [Reflect.isExtensible()的使用](https://www.cnblogs.com/diligenceday/p/5474126.html" \l "_label10)
* [Reflect.ownKeys()方法的使用：](https://www.cnblogs.com/diligenceday/p/5474126.html" \l "_label11)
* [Reflect.set()](https://www.cnblogs.com/diligenceday/p/5474126.html" \l "_label12)
* [Reflect.setPrototypeOf()](https://www.cnblogs.com/diligenceday/p/5474126.html" \l "_label13)
* [参考：](https://www.cnblogs.com/diligenceday/p/5474126.html" \l "_label14)

[回到顶部](https://www.cnblogs.com/diligenceday/p/5474126.html" \l "_labelTop)

# **Reflect介绍：**

　　Reflect这个对象在我的node(v4.4.3)中还没有实现, babel(6.7.7)也没有实现 ,新版本的chrome是支持的， ff比较早就支持Proxy和Reflect了，要让node支持Reflect可以安装[harmony-reflect](https://github.com/tvcutsem/harmony-reflect/ " \t "https://www.cnblogs.com/diligenceday/p/_blank) ;

　　Reflect不是构造函数， 要使用的时候直接通过Reflect.method()调用， Reflect有的方法和Proxy差不多， 而且多数Reflect方法原生的Object已经重新实现了。

[回到顶部](https://www.cnblogs.com/diligenceday/p/5474126.html" \l "_labelTop)

# **什么要使用Reflect**

　　这里列举几个为什么要使用Reflect的原因， 译文地址：[Reflect](https://github.com/tvcutsem/harmony-reflect/wiki" \l "reflect" \t "https://www.cnblogs.com/diligenceday/p/_blank) ， 大概翻译了一遍：

　　1：**更加有用的返回值：** Reflect有一些方法和ES5中Object方法一样样的， 比如： Reflect.getOwnPropertyDescriptor和Reflect.defineProperty,  不过, Object.defineProperty(obj, name, desc)执行成功会返回obj， 以及其它原因导致的错误， Reflect.defineProperty只会返回false或者true来表示对象的属性是否设置上了， 如下代码可以重构：

运行下面代码

[IMG_256](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)

try {

Object.defineProperty(obj, name, desc);

// property defined successfully

} catch (e) {

// possible failure (and might accidentally catch the wrong exception)

}

[IMG_257](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)

　　重构成这样：

运行下面代码

if (Reflect.defineProperty(obj, name, desc)) {

// success

} else {

// failure

}

　　其余的方法， 比如Relect.set ， Reflect.deleteProperty, Reflect.preventExtensions, Reflect.setPrototypeOf， 都可以进行重构；

　　2：**函数操作**，  如果要判断一个obj有定义或者继承了属性name， 在ES5中这样判断：**name in obj** ； 或者删除一个属性 ：**delete obj[name]**,  虽然这些很好用， 很简短， 很明确， 但是要使用的时候也要封装成一个类；

　　有了Reflect， 它帮你封装好了， **Reflect.has(obj, name)**,  **Reflect.deleteProperty(obj, name);**

　　3：**更加可靠的函数式执行方式：** 在ES中， 要执行一个函数f，并给它传一组参数args， 还要绑定this的话， 要这么写：

运行下面代码

f.apply(obj, args)

　　但是f的apply可能被重新定义成用户自己的apply了，所以还是这样写比较靠谱：

运行下面代码

Function.prototype.apply.call(f, obj, args)

　　上面这段代码太长了， 而且不好懂， 有了Reflect， 我们可以更短更简洁明了：

运行下面代码

Reflect.apply(f, obj, args)

　　4：**可变参数形式的构造函数：** 想象一下， 你想通过**不确定长度的参数**实例化一个构造函数， 在ES5中， 我们可以使用扩展符号， 可以这么写：

运行下面代码

var obj = new F(...args)

　　不过在ES5中， 不支持扩展符啊， 所以， 我们只能用F.apply，或者F.call的方式传不同的参数， 可惜F是一个**构造函数**， 这个就坑爹了， 不过有了Reflect， 我们在ES5中能够这么写：

运行下面代码

var obj = Reflect.construct(F, args)

　　5：**控制访问器或者读取器的this：** 在ES5中， 想要读取一个元素的属性或者设置属性要这样：

运行下面代码

var name = ... // get property name as a string

obj[name] // generic property lookup

obj[name] = value // generic property update

　　Reflect.get和Reflect.set方法允许我们做同样的事情， 而且他增加了一个额外的参数reciver， 允许我们设置对象的setter和getter的上下**this**：

运行下面代码

var name = ... // get property name as a string

Reflect.get(obj, name, wrapper) // if obj[name] is an accessor, it gets run with `this === wrapper`

Reflect.set(obj, name, value, wrapper)

　　访问器中不想使用自己的方法，而是想要重定向this到wrapper：

运行下面代码

[IMG_258](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)

var obj = {

set foo(value) { return this.bar(); },

bar: function() {

alert(1);

}

};var wrapper = {

bar : function() {

console.log("wrapper");

}

}

Reflect.set(obj, "foo", "value", wrapper);

[IMG_259](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)

　　6：**避免直接访问 \_\_proto\_\_** ： ES5提供了 Object.getPrototypeOf(obj)，去访问对象的原型， ES6提供也提供了Reflect.getPrototypeOf(obj) 和  Reflect.setPrototypeOf(obj, newProto)， 这个是新的方法去**访问**和**设置对象的原型**：

[回到顶部](https://www.cnblogs.com/diligenceday/p/5474126.html" \l "_labelTop)

# **Reflect.apply的使用**

　　Reflect.apply其实就是ES5中的 Function.prototype.apply() 替身， 执行Reflect.apply需要三个参数  
　　第一个参数为： 需要执行的函数；  
　　第二个参数为： 需要执行函数的上下文this；  
　　第三个参数为： 是一个数组或者伪数组， 会作为执行函数的参数；

运行下面代码

[IMG_260](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)

<script>

let fn = function() {

this.attr = [0,1,2,3];

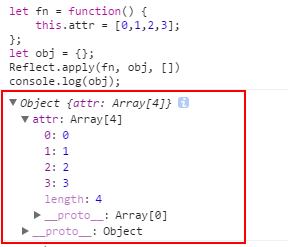
};

let obj = {};

Reflect.apply(fn, obj, [])

console.log(obj); </script>

[IMG_261](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)



　　Reflect.apply的DEMO：

运行下面代码

[IMG_263](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)

<script>

Reflect.apply(Math.floor, undefined, [1.75]); // 输出：1;

Reflect.apply(String.fromCharCode, undefined, [104, 101, 108, 108, 111]); // 输出："hello"

Reflect.apply(RegExp.prototype.exec, /ab/, ["confabulation"]).index; //输出： 4

Reflect.apply("".charAt, "ponies", [3]); // 输出："i"

</script>

[IMG_264](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)

　　Reflect可以与Proxy联合使用：

运行下面代码

[IMG_265](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)

{

var Fn = function(){

};

Fn.prototype.run = function() {

console.log( "runs out" );

};

var ProxyFn = new Proxy(Fn, {

construct (target ,arugments) {

console.log("proxy constructor");

var obj = new target(...arugments);

//Reflect.apply的使用方法; Reflect.apply(target.prototype.run, obj, arugments);

return obj;

}

});

new ProxyFn (); //会先输出: "proxy constructor" ； 再输出： runs out

}

[IMG_266](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)

[回到顶部](https://www.cnblogs.com/diligenceday/p/5474126.html" \l "_labelTop)

# **Reflect.construct()的使用：**

　　Reflect.construct其实就是实例化构造函数，通过传参形式的实现， 执行的方式不同， 效果其实一样， construct的第一个参数为构造函数， 第二个参数由参数组成的数组或者伪数组， 基本的使用方法为：

运行下面代码

[IMG_267](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)

var Fn = function(arg) {

this.args = [arg]

};

console.log( new Fn(1), Reflect.construct(Fn,[1]) ); // 输出是一样的

var d = Reflect.construct(Date, [1776, 6, 4]);

d instanceof Date; // true

d.getFullYear(); // 1776//所以Reflect.consturct和new 构

[IMG_268](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)

　　所以Reflect.consturct和new 构造函数是一样， 至少到目前为止..

　　我们可以给Reflect.construct传第三个参数 ， 第三个参数为一个超类， 新元素会继承这个超类；

运行下面代码

[IMG_269](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)

<script>

function someConstructor() {}var result = Reflect.construct(Array, [], someConstructor);

Reflect.getPrototypeOf(result); // someConstructor.prototype

Array.isArray(result); // true//orvar Fn = function() {

this.attr = [1];

};var Person = function() {

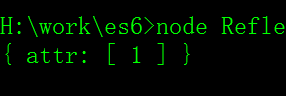
};

Person.prototype.run = function() {

};

console.log( Reflect.construct(Fn, [], Person) );</script>

[IMG_270](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)



　　所以我们可以用这个实现一个特殊的的数组， 继承数组， 但是也有自己的方法；

运行下面代码

[IMG_272](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)

var Fn = function() {

Array.apply(this, arguments);

this.shot = ()=> {

console.log("heheda");

};

};var arr = Reflect.construct(Fn, [])

[IMG_273](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)

[回到顶部](https://www.cnblogs.com/diligenceday/p/5474126.html" \l "_labelTop)

# **Reflect.defineProperty的使用;**

 　　Reflect.defineProperty返回的是一个**布尔值**， 通过直接赋值的方式把**属性**和**属性值**添加给对象返回的是一整个对象， **如果添加失败会抛错**；

运行下面代码

var obj = {};

obj.x = 10;

console.log(obj.x) //输出：10；

　　使用Reflect.defineProperty的方式添加值；

运行下面代码

[IMG_274](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)

<script>var obj = {};if( Reflect.defineProperty(obj, "x", {value : 7 }) ) {

console.log("added success");

}else{

console.log("添加失败");

};</script>

[IMG_275](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)

　　如果我们执行preventExtensions， 通过Object.defindProperty**定义新属性**报错了， 但是通过Reflect.defineProperty没有报错， 返回了一个false的值：

运行下面代码

[IMG_276](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)

var obj = {};

Object.preventExtensions(obj);

Object.defineProperty(obj, "x" , {

value: 101,

writable: false,

enumerable: false,

configurable: false

});// 直接抛错了;

console.log( Reflect.defineProperty(obj, "x", {value:101}) ) //返回false：

[IMG_277](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)

　　如果通过直接赋值的方式， 无论是否正确赋值， **都返回设置的值，** 除非我们手动确认对象的属性值是否设置成功；

运行下面代码

[IMG_278](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)

<script>var obj = {};

Object.preventExtensions(obj);

console.log( obj.aa = 1 ); //输出：1；

console.log(obj.aa) //输出：undefined；

</script>

[IMG_279](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)

[回到顶部](https://www.cnblogs.com/diligenceday/p/5474126.html" \l "_labelTop)

# **Reflect.deleteProperty的使用：**

　　Reflect.deleteProperty和Reflect.defineProperty的使用方法差不多， Reflect.deleteProperty和 delete obj.xx的操作结果是一样， 区别是使用形式不同：一个是操作符，一个是函数调用；

运行下面代码

Reflect.deleteProperty(Object.freeze({foo: 1}), "foo"); // false

delete Object.freeze({foo: 1}).foo; //输出：false；

[回到顶部](https://www.cnblogs.com/diligenceday/p/5474126.html" \l "_labelTop)

# **Reflect.get()方法的使用；**

　　这个方法的有两个必须的参数： 第一个为obj目标对象， 第二个为属性名对象， 第三个是可选的，是作为读取器的上下文(this);

运行下面代码

var obj = {};

obj.foo = 1;

console.log( obj.foo ); //输出：1;

console.log( Reflect.get(obj, "foo") ) //输出：1;

　　如果Reflect.get有第三个参数的话, 第三个参数会作为读取器的上下文：

运行下面代码

[IMG_280](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)

var Reflect = require('harmony-reflect');

var obj = {

"foo" : 1,

get bar() {

return this.foo;

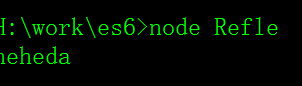
}

};var foo = {};

foo.foo = "heheda";

console.log(Reflect.get(obj, "bar", foo));

[IMG_281](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)



[回到顶部](https://www.cnblogs.com/diligenceday/p/5474126.html" \l "_labelTop)

# **Reflect.getOwnPropertyDescritptor()方法的使用：**

　　通过Reflect.getOwnPropertyDescritptor获取**属性描述**：

运行下面代码

[IMG_283](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)

Reflect.getOwnPropertyDescriptor({x: "hello"}, "x");//也可以这样获取：

Object.getOwnPropertyDescriptor({x:"1"},"x");//这两个的区别是一个会包装对象， 一个不会：

Reflect.getOwnPropertyDescriptor("hello",0); //抛出异常

Object.getOwnPropertyDescriptor("hello",0); //输出： {value: "h", writable: false, enumerable: true, configurable: false}

[IMG_284](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)

[回到顶部](https://www.cnblogs.com/diligenceday/p/5474126.html" \l "_labelTop)

# **Reflect.getPrototypeOf()方法的使用：**

　　Reflect.getPrototypeOf和Object.getPrototypeOf是一样的, 他们都是返回一个对象的原型

运行下面代码

Reflect.getPrototypeOf({}); // 输出：Object.prototype

Reflect.getPrototypeOf(Object.prototype); // 输出：null

Reflect.getPrototypeOf(Object.create(null)); // 输出： null

[回到顶部](https://www.cnblogs.com/diligenceday/p/5474126.html" \l "_labelTop)

# **Reflect.has的使用**

　　Reflect.has这个方法有点像操作符：in ， 比如这样： xx **in** obj;

运行下面代码

[IMG_285](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)

<script>

Reflect.has({x:0}, "x") //输出： true；

Reflect.has({y:0}, "y") //输出：true

； var obj = {x:0}; console.log( "x" in obj); var proxy = new Proxy(obj, { has : function(target, args) { console.log("执行has方法"); return Reflect.has(target,...args); } }); console.log( "x" in proxy); //输出：true； console.log(Reflect.has(proxy, "x")) //输出：true； </script>

[IMG_286](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)

　　这个demo的obj相当于变成了一个方法了， 没他什么用 ， 只是利用了他的has方法：

运行下面代码

obj = new Proxy({}, {

has(t, k) { return k.startsWith("door"); }

});

Reflect.has(obj, "doorbell"); // true

Reflect.has(obj, "dormitory"); // false

[回到顶部](https://www.cnblogs.com/diligenceday/p/5474126.html" \l "_labelTop)

# **Reflect.isExtensible()的使用**

运行下面代码

[IMG_287](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)

// 现在这个元素是可以扩展的；var empty = {};

Reflect.isExtensible(empty); // === true

// 使用preventExtensions方法， 让这个对象无法扩展新属性；Reflect.preventExtensions(empty);

Reflect.isExtensible(empty); // === false

// 这个对象无法扩展新属性， 可写的属性依然可以改动var sealed = Object.seal({});

Reflect.isExtensible(sealed); // === false

// 这个对象完全被冻结了var frozen = Object.freeze({});

Reflect.isExtensible(frozen); // === false

[IMG_288](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)

　　Reflect.isExtensible和Object.isExtensible的区别是， 如果参数不对，一个会抛错， 另一个只是返回true或者false：

运行下面代码

Reflect.isExtensible(1);// 抛错了: 1 is not an object

Object.isExtensible(1);// 返回false;

[回到顶部](https://www.cnblogs.com/diligenceday/p/5474126.html" \l "_labelTop)

# **Reflect.ownKeys()方法的使用：**

　　Reflect.ownKeys， Object可没有ownKeys方法, Reflect.ownKeysz他的作用是返回对象的keys;

运行下面代码

[IMG_289](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)

console.log(Reflect.ownKeys({"a":0,"b":1,"c":2,"d":3})); //输出 ：["a", "b", "c", "d"]

console.log(Reflect.ownKeys([])); // ["length"]var sym = Symbol.for("comet");var sym2 = Symbol.for("meteor");var obj = {[sym]: 0, "str": 0, "773": 0, "0": 0,

[sym2]: 0, "-1": 0, "8": 0, "second str": 0};

Reflect.ownKeys(obj); //输出：/ [ "0", "8", "773", "str", "-1", "second str", Symbol(comet), Symbol(meteor) ]

[IMG_290](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)

　　Reflect.ownKeys的排序是根据: **先显示数字， 数字根据大小排序，然后是 字符串根据插入的顺序排序， 最后是symbol类型的key也根据插入插入顺序排序;**  
　　出现这中排序是因为，你给一个对象属性赋值时候， 对象的key的排序规则就是先数字， 在字符串， 最后是symbol类型的数据；

　　Reflect.preventExtensions()的使用方法：

　　Object也有preventExtensions方法， 和Reflect.preventExtensions()有一点区别， 如果Reflect.preventExtensions参数不是对象会抛错；

运行下面代码

[IMG_291](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)

var empty = {};

Reflect.isExtensible(empty); // === true

// 执行preventExtensions后的对象可以修改；Reflect.preventExtensions(empty);

Reflect.isExtensible(empty); // === false

Reflect.preventExtensions(1);// TypeError: 1 is not an object

Object.preventExtensions(1);//不会抛错， 会返回：1

[IMG_292](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)

[回到顶部](https://www.cnblogs.com/diligenceday/p/5474126.html" \l "_labelTop)

# **Reflect.set()**

　　Reflect.set方法和get是差不多的；

运行下面代码

[IMG_293](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)

var obj = {};

Reflect.set(obj, "prop", "value"); // 输出：true

console.log( obj.prop ); // 输出："value"

var arr = ["duck", "duck", "duck"];

Reflect.set(arr, 2, "goose"); // true

console.log( arr[2] ); // "goose"

Reflect.set(arr, "length", 1); // true

console.log( arr );// ["duck"];

[IMG_294](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)

　　　　Reflect.set(obj)相当于 Reflect.set(obj, undefined, undefined);

运行下面代码

var obj = {};

Reflect.set(obj); // 输出：true//以上的代码相当于 Reflect.set(obj, undefined, undefined);

Reflect.getOwnPropertyDescriptor(obj, "undefined");// { value: undefined, writable: true, enumerable: true, configurable: true }

　　Reflect.set也可以有第四个参数， 这个参数会作为stter的this;

运行下面代码

[IMG_295](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)

var obj = {

value : 10,

set key( value ) {

console.log("setter");

this.value = value;

},

get key() {

return this.value;

}

};

Reflect.set(obj,"key","heheda", obj);

console.log(obj);

[IMG_296](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)

[回到顶部](https://www.cnblogs.com/diligenceday/p/5474126.html" \l "_labelTop)

# **Reflect.setPrototypeOf()**

　　Reflect.setPrototypeOf()方法和Object.setPrototypeOf差不多一样样的， 会给对象设置原型， 就是更改对象的**\_\_proto\_\_**属性了...;

运行下面代码

[IMG_297](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)

Reflect.setPrototypeOf({}, Object.prototype); // 输出true

// 给该对象数组[[Prototype]] 为null.

Reflect.setPrototypeOf({}, null); // true// 此时的obj.\_\_proto\_\_为undefine

//把对象冻结以后重新设置[[prototype]]

Reflect.setPrototypeOf(Object.freeze({}), null); // false

// 如果原型链循环依赖的话就会返回false.var target = {};var proto = Object.create(target);

Reflect.setPrototypeOf(target, proto); // false

[IMG_298](https://www.cnblogs.com/diligenceday/p/javascript:void(0);)

[回到顶部](https://www.cnblogs.com/diligenceday/p/5474126.html" \l "_labelTop)

# **参考：**

　　MDN：[mdn: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Reflect](mdn: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Reflect" \t "https://www.cnblogs.com/diligenceday/p/_blank)  
　　让node环境支持Proxy和Reflect：[https://github.com/tvcutsem/harmony-reflect/](https://github.com/tvcutsem/harmony-reflect/ " \t "https://www.cnblogs.com/diligenceday/p/_blank)