JAVASCRIPT-OBJECT

Color
- 经常使用
- 比较实用
- 偶尔需要

提取数据

Object.keys()

```
const object1 = {
  a: 'somestring',
  b: 42,
  c: false
};

console.log(Object.keys(object1));
// expected output: Array ["a", "b", "c"]
```

Object.keys(obj)

给出key值数组

Object.values()

```
const object1 = {
  a: 'somestring',
  b: 42,
  c: false
};

console.log(Object.values(object1));
// expected output: Array ["somestring", 42, false]
```

Object.values(obj)

给出value值数组

Object.entries()

```
const object1 = {
  a: 'somestring',
  b: 42
};

for (const [key, value] of Object.entries(object1)) {
  console.log(`${key}: ${value}`);
}

// expected output:
// "a: somestring"
// "b: 42"
// order is not guaranteed
```

Object.entries(obj)

给出key value所组成数组的数组

Object.fromEntries()

```
const entries = new Map([
  ['foo', 'bar'],
  ['baz', 42]
]);

const obj = Object.fromEntries(entries);

console.log(obj);
// expected output: Object { foo: "bar", baz: 42 }
```

Object.fromEntries(iterable)

可以将entries()产生的结果还原为obj

产生新Object

Object.assign()

```
const target = { a: 1, b: 2 };
const source = { b: 4, c: 5 };

const returnedTarget = Object.assign(target, source);

console.log(target);
// expected output: Object { a: 1, b: 4, c: 5 }

console.log(returnedTarget);
// expected output: Object { a: 1, b: 4, c: 5 }
```

Object.assign(target, ...sources)

将sources内容替换掉target

Object.create()

```
const person = {
  isHuman: false,
  printIntroduction: function() {
    console.log(`My name is ${this.name}. Am I human? ${this.isHuman}`);
  }
};

const me = Object.create(person);

me.name = 'Matthew'; // "name" is a property set on "me", but not on "person"
me.isHuman = true; // inherited properties can be overwritten

me.printIntroduction();
// expected output: "My name is Matthew. Am I human? true"
```

Object.create(proto)
Object.create(proto,propertiesObject)

根据给出的原型创建新数组
创建的新数组打印是空的，其内容都在它的原型链中。

操作属性描述

Object.getOwnPropertyDescriptor()
查自己的某一个属性

```
const object1 = {
  property1: 42
};

const descriptor1 = Object.getOwnPropertyDescriptor(object1, 'property1');

console.log(descriptor1.configurable);
// expected output: true

console.log(descriptor1.value);
// expected output: 42
```

Object.getOwnPropertyDescriptor(obj, prop)

可以查看一个属性的所有描述属性

Object.getOwnPropertyDescriptors()
查所有自己的属性

```
const object1 = {
  property1: 42
};

const descriptors1 = Object.getOwnPropertyDescriptors(object1);

console.log(descriptors1.property1.writable);
// expected output: true

console.log(descriptors1.property1.value);
// expected output: 42
```

Object.getOwnPropertyDescriptors(obj)

查看所有属性的描述属性

这里只能查看，并不能修改

描述属性相同

Object.defineProperty()

```
const object1 = {};

Object.defineProperty(object1, 'property1', {
  value: 42,
  writable: false
});

object1.property1 = 77;
// throws an error in strict mode

console.log(object1.property1);
// expected output: 42
```

Object.defineProperty(obj, prop, descriptor)

可定义和修改制定属性

Object.defineProperties()

```
const object1 = {};

Object.defineProperties(object1, {
  property1: {
    value: 42,
    writable: true
  },
  property2: {}
});

console.log(object1.property1);
// expected output: 42
```

Object.defineProperties(obj, props)

可定义和修改所有属性

可以在定义时设定，也可以在随后做修改

描述属性
- value — 属性值
- writable — 是否可写
- get — 返回getter，没有就undefined
- set — 返回setter，没有就undefined
- configurable — true则这个属性本身可更改，也可以删除
- enumerable — true如果是枚举类型

禁固属性

Object.preventExtensions()
不可增
可以删
可以改

```
// New objects are extensible.
var empty = {};
Object.isExtensible(empty); // === true

// ...but that can be changed.
Object.preventExtensions(empty);
Object.isExtensible(empty); // === false

// Sealed objects are by definition non-extensible.
var sealed = Object.seal({});
Object.isExtensible(sealed); // === false

// Frozen objects are also by definition non-extensible.
var frozen = Object.freeze({});
Object.isExtensible(frozen); // === false
```

Object.preventExtensions(obj)

阻止可可拓展性，这个obj将不可拓展，不过原型链是可以添加新属性的。

Object.seal()
不可增
不可删
可以改

```
const object1 = {
  property1: 42
};

Object.seal(object1);
object1.property1 = 33;
console.log(object1.property1);
// expected output: 33

delete object1.property1; // cannot delete when sealed
console.log(object1.property1);
// expected output: 33
```

Object.seal(obj)

seal封印了这个obj，obj它将不会添加或删除属性

Object.isSealed(obj) — 可判断是否被封印

Object.freeze()
不可增
不可删
不可改（一层）

```
const obj = {
  prop: 42
};

Object.freeze(obj);

obj.prop = 33;
// Throws an error in strict mode

console.log(obj.prop);
// expected output: 42
```

Object.freeze(obj)

freeze冻结了这个obj，不仅属性无法添加修改，连内部值都不能修改

Object.isFrozen(obj) — 可判断是否被冻结

一旦被封印，冻结，阻止拓展，则无法还原。这都是一次性的操作。
但一个obj创造出来到它被禁锢属性锁定之前这段空间，它做什么都可以。