# 迭代器协议

**该特性目前仍处于ECMAScript 6规范草案中**
目前的实现在未来可能会发生改变, 甚至被完全删除, 请谨慎使用.

ES6 里的迭代器并不是一种新的语法或者是新的内置对象(构造函数), 而是一种协议 (protocol). 所有遵循了这个协议的对象都可以称之为迭代器对象.

## 描述

一个迭代器(对象)会有一个名为 `next` 的方法, 调用该方法后会返回一个拥有两个属性的对象, 一个是 `value` 属性, 值可以是任意值, 以及一个 `done` 属性, 布尔值, 表示该迭代器是否已经被迭代完毕.

## 示例

### 制作迭代器

```
function makeIterator(array){
    var nextIndex = 0;

    return {
       next: function(){
           return nextIndex < array.length ?
               {value: array[nextIndex++], done: false} :
               {done: true};
       }
    }
}

var it = makeIterator(['yo', 'ya']);

console.log(it.next().value);console.log(it.next().value);console.log(it.next().done);
```

### 制作无穷迭代器

```
function idMaker(){
    var index = 0;

    return {
       next: function(){
           return {value: index++, done: false};
       }
    }
}

var it = idMaker();

console.log(it.next().value);console.log(it.next().value);console.log(it.next().value);
```

使用生成器制作迭代器

```
function* makeSimpleGenerator(array){
    var nextIndex = 0;

    while(nextIndex < array.length){
        yield array[nextIndex++];
    }
}

var gen = makeSimpleGenerator(['yo', 'ya']);

console.log(gen.next().value);console.log(gen.next().value);console.log(gen.next().done);



function* idMaker(){
    var index = 0;
    while(true)
        yield index++;
}

var gen = idMaker();

console.log(gen.next().value);console.log(gen.next().value);console.log(gen.next().value);
```

**Firefox**私有的并且已经废弃的迭代器协议

Firefox 在 26 之前实现过另一种非标准的迭代器协议: 在调用 `next` 方法时它会直接返回想要的值, 而不是一个拥有 `value` 属性的占位符对象, 且在迭代完毕后会抛出 `StopIteration` 异常.

被废弃的迭代器协议示例

```
function makeIterator(array){
    var nextIndex = 0;

    return {
       next: function(){
           if(nextIndex < array.length){
               return array[nextIndex++];
           else
               throw new StopIteration();
       }
    }
}

var it = makeIterator(['yo', 'ya']);
```

```
console.log(it.next());console.log(it.next());try{
    console.log(it.next());
}
catch(e){
    if(e instanceof StopIteration){
            }
}
```