

迭代器和生成器

可迭代性

当一个对象实现了 `Symbol.iterator` (`./symbols.html#symboliterator`)属性时，我们认为它是可迭代的。一些内置的类型如

`Array` , `Map` , `Set` , `String` , `Int32Array` , `Uint32Array` 等都已经实现了各自的 `Symbol.iterator` 。对象上的 `Symbol.iterator` 函数负责返回供迭代的值。

for..of 语句

`for..of` 会遍历可迭代的对象，调用对象上的 `Symbol.iterator` 方法。下面是在数组上使用 `for..of` 的简单例子：

```
let someArray = [1, "string", false];

for (let entry of someArray) {
  console.log(entry); // 1, "string", false
}
```

for..of vs. for..in 语句

`for..of` 和 `for..in` 均可迭代一个列表；但是用于迭代的值却不同，`for..in` 迭代的是对象的 *键* 的列表，而 `for..of` 则迭代对象的键对应的值。

下面的例子展示了两者之间的区别：

```
let list = [4, 5, 6];

for (let i in list) {
  console.log(i); // "0", "1", "2",
}

for (let i of list) {
  console.log(i); // "4", "5", "6"
}
```

另一个区别是 `for..in` 可以操作任何对象；它提供了查看对象属性的一种方法。但是 `for..of` 关注于迭代对象的值。内置对象 `Map` 和 `Set` 已经实现了 `Symbol.iterator` 方法，让我们可以访问它们保存的值。

```
let pets = new Set(["Cat", "Dog", "Hamster"]);
pets["species"] = "mammals";

for (let pet in pets) {
  console.log(pet); // "species"
}

for (let pet of pets) {
  console.log(pet); // "Cat", "Dog", "Hamster"
}
```

代码生成

目标为 ES5 和 ES3

当生成目标为ES5或ES3，迭代器只允许在 `Array` 类型上使用。在非数组值上使用 `for..of` 语句会得到一个错误，就算这些非数组值已经实现了 `Symbol.iterator` 属性。

编译器会生成一个简单的 `for` 循环做为 `for..of` 循环，比如：

```
let numbers = [1, 2, 3];
for (let num of numbers) {
  console.log(num);
}
```

生成的代码为：

```
var numbers = [1, 2, 3];
for (var _i = 0; _i < numbers.length; _i++) {
  var num = numbers[_i];
  console.log(num);
}
```

目标为 ECMAScript 2015 或更高

当目标为兼容ECMAScript 2015的引擎时，编译器会生成相应引擎的 `for..of` 内置迭代器实现方式。