ES6 PPT1606201: 遍历器 Iterator

Thursday, November 5, 2015 10:15 AM

遍历器（迭代器）Iterator：

1. 遍历器是一种接口（Object 类型），为各种不同的数据结构提供统一的可遍历操作其所有成员的机制
2. 遍历过程
   a. 创建一个指针对象即遍历器 myItr，将其指针指向传入遍历器参数的数据起始位置 -> var myItr = makeIteratorAB(['a', 'b'])
   b. 第一次调用指针对象的 next 方法，返回数据结构第一个成员并将指针移向第二个成员 -> myItr.next() // {value: "a", done: false}
   c. 第二次调用指针对象的 next 方法，返回数据结构第二个成员并将指针移向第三个成员 -> myItr.next() // {value: "b", done: false}
   d. 不断调用指针对象的 next 方法，直到指针移向数据结构的结束位置 -> myItr.next() // {value: undefined, done: true}

```
// 遍历器实现
var it = makeIteratorAB(['a', 'b']);
it.next() // { value: "a", done: false }
it.next() // { value: "b", done: false }
it.next() // { value: undefined, done: true }
function makeIteratorAB(array) {
 var nextIndex = 0;
 return {
  next: function() {
   return nextIndex < array.length ?
    {value: array[nextIndex++], done: false} :
    {value: undefined, done: true};
  }
 };
}
```

3. 遍历器每次 next 方法的返回值对象中的 value 和 done 属性
   a. 属性 value 为当前成员值，其值为 undefined 时可省略
   b. 属性 done 表示是否已结束遍历，其值为 false 时可省略
   c. 实例：优化的 makeIteratorAB 遍历器实现

```
function makeIteratorAB(array) {
 var nextIndex = 0;
 return {
  next: function() {
   return nextIndex < array.length ?
    {value: array[nextIndex++]} :
    {done: true};
  }
 };
}
```

   d. 应用：ID 生成器，无限运行的遍历器

```
var it = idMaker();
it.next().value // "0"
it.next().value // "1"
it.next().value // "2"
// ...
function idMaker() {
 var index = 0;
 return {
  next: function() {
   return {value: index++, done: false};
  }
 };
}
```

4. 默认情况下遍历器通过对象的 Symbol.iterator 方法生成。只要正确部署了该方法，对象就被认为是可迭代（可遍历）的(Iterable)
   a. 特性1：可遍历对象可以通过扩展运算符展开
   b. 特性2：可以使用 for ... of 对可遍历对象进行遍历操作：for (itemValue of iterable) ;

```
var myIterableABC = {data: ['A', 'B', 'C']}
myIterableABC[Symbol.iterator] = makeIteratorABC;
function makeIteratorABC() {
 var nextIndex = 0;
```

```
const self = this
var len =self.data.length
var array = ['A', 'B', 'C'];
return {
  next: function() {
    return nextIndex < array.length ? {value: array[nextIndex++]} : {done: true};
  }
return nextIndex < len ? {value: self.data[nextIndex++]} : {done:true }
 };
}
console.log([...myIterableABC]); // ["A", "B", "C"]
for(var letter of myIterableABC) console.log(letter); // "A" // "B" // "C"
var [myFirst, ...myRest] = myIterableABC;
console.log(myFirst, myRest); // "A", ["B", "C"]
var myInvalidItr = {[Symbol.iterator]: () => 1};
console.log([...myInvalidItr]); // myInvalidItr[Symbol.iterator] is not a function -> Symbol.iterator方法未返回一个遍历器，无法遍历
```

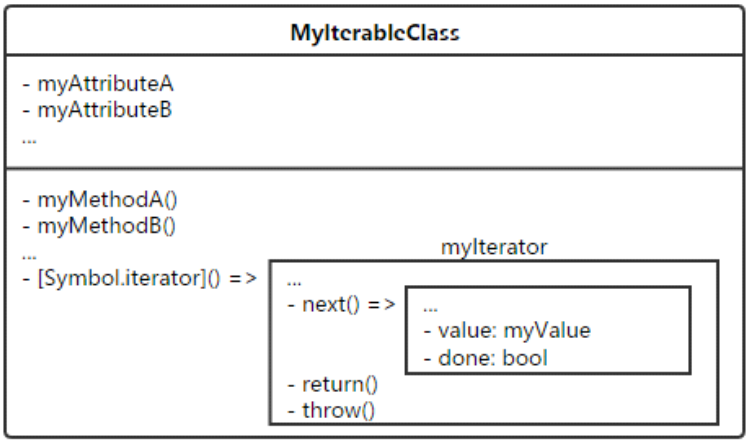
5. 数组、字符串、Set/Map、function 内的 arguments 等类数组对象都原生部署了 Symbol.iterator 遍历器生成方法，它们均是可遍历的

```
(function() {
  var itr = arguments[Symbol.iterator]();
  console.log(itr.next(), itr.next(), itr.next(), itr.next());
})('o', 'k'); // {value: "o", done: false}, {value: "k", done: false}, {value: undefined, done: true}, {value: undefined, done: true}
console.log([...'my string']); // ["m", "y", " ", "s", "t", "r", "i", "n", "g"]
```

6. 遍历器的 return 方法（可选）
   a. 如果遍历器定义了 return 方法，在 for...of 未完成遍历而提前退出（eg. 报错或调用 break）时该方法会被调用

```
function readNumber(numbers) {
  var index = -1;
  return {
    [Symbol.iterator]() {
      return {
        next() {
          if (++index >= numbers.length) return {done: true};
          var currentNumber = numbers[index];
          return {value: currentNumber};
        },
        return() {
          console.log('发现一个非数字: ' + numbers[index]);
          return {done: true};
        }
      };
    }
  };
}
for (let number of readNumber([3, 4, 'C', 8])) {
  if(isNaN(number)) break;
} // 发现一个非数字: C
```

7. 可遍历(Iterable)对象结构图示：



8. 应用：使用遍历器实现对象属性的遍历

```
class IterableClass {
  [Symbol.iterator]() {
    var keys = [], currentIndex = 0, self = this;
    for (var key in this) keys.push(key);
    return {
      next() {
        var currentKey = keys[currentIndex];
```

```
        return currentIndex++ < keys.length ? {value: `${currentKey}: ${self[currentKey]}`} : {done: true}
      }
    }
  }
}
class Cat extends IterableClass {
  constructor(name, age, gender = 'unknown') {
    super();
    this.name = name;
    this.age = age;
    this.gender = gender;
  }
}
console.log(...new Cat('Tom', 3)); // {name: "Tom", age: 3, gender: "unknown"}
```

9. 应用：将类数组对象转化为可遍历对象

```
let iterable = {
  0: 'a',
  1: 'b',
  2: 'c',
  length: 3,
  [Symbol.iterator]: Array.prototype[Symbol.iterator]
};
console.log(...iterable);  // "a", "b", "c"
```