ES6 PPT1512081: Symbol 数据类型

Thursday, November 5, 2015 10:15 AM

Symbol 数据类型：

1. 原始类型 Symbol 为全局下独一无二值：Symbol(), Symbol(param)

   ```
   var mySymbol1 = Symbol(), mySymbol2 = Symbol(), mySymbol3 = Symbol('三'), mySymbol4 = Symbol({});
   var mySymbol5 = mySymbol1;
   console.log(mySymbol1 === mySymbol2); // false
   console.log(mySymbol1 === mySymbol3); // false
   console.log(mySymbol3 === mySymbol4); // false
   console.log(mySymbol3 === Symbol('三')); // false
   console.log(mySymbol1 === mySymbol5); // true
   console.log(typeof Symbol(), typeof Symbol); // symbol function
   new Symbol(); // TypeError: Symbol is not a constructor
   ```

2. Symbol 值不能与其他类型的值进行运算，但可以被转换为 String 与 Boolean 值：

   ```
   var mySymbol0 = Symbol(0), mySymbolA = Symbol([1, 2, 3, 4]);
   console.log('mySymbol0 is: ' + mySymbol0); // TypeError: Cannot convert a Symbol value to a string
   console.log('Stringified mySymbol0 is: ' + String(mySymbol0)); // Stringified mySymbol0 is: Symbol(0)
   console.log('Stringified mySymbolA is: ' + mySymbolA.toString()); // Stringified mySymbolA is: Symbol(1,2,3,4)
   console.log('Bool value for Symbol() is: ' + Boolean(Symbol())); // Bool value for Symbol() is: true
   console.log('mySymbol0 的布尔值为: ' + (mySymbol0 ? '真' : '假')); // mySymbol0 的布尔值为: 真
   ```

3. Symbol 可作为对象之属性名：{ [mySymbol]: myValue }

   ```
   var mySymbol101 = Symbol(101);
   var myObj1 = { [mySymbol101]: 'Obj1-101' };
   var myObj2 = {}; myObj2[mySymbol101] = 'Obj2-101';
   var myObj3 = {}; Object.defineProperty(myObj3, mySymbol101, { value: 'Obj3-101' });
   console.log(myObj1[mySymbol101]); // Obj1-101
   console.log(myObj2[mySymbol101]); // Obj2-101
   console.log(myObj3[mySymbol101]); // Obj3-101
   console.log(myObj1.mySymbol101); // undefined
   ```

4. 使用 Symbol.for() 与 Symbol.keyFor() 对 Symbol 值登记进行复用：
   a. Symbol.for(param) 生成类似唯一值单例：

      ```
      var s0 = Symbol('foo');
      var s1 = Symbol.for('foo');
      var s2 = Symbol.for('foo');
      console.log(s1 === s2); // true
      console.log(s1 === s0); // false
      console.log(Symbol.for('bar') === Symbol.for('bar')); // true
      console.log(Symbol.for('bar') === Symbol('bar')); // false
      console.log(Symbol.keyFor(s0), Symbol.keyFor(s1), Symbol.keyFor(s2)); // undefined "foo" "foo"
      ```

   b. Symbol.for(param) 登记至全局环境，iframe 中可用：

      ```
      iframe = document.createElement('iframe');
      ```

```
iframe.src = String(window.location);
document.body.appendChild(iframe);
console.log(iframe.contentWindow.Symbol.for('foo') === Symbol.for('foo')); // true
```

5. Symbol 属性无法被遍历，除非使用 Object.getOwnPropertySymbols(myObj) 或 Reflect.ownKeys(myObj)：
   a. Object.getOwnPropertySymbols(myObj) 和 Reflect.ownKeys(myObj) 返回一个数组：

```
var mySymbolM = Symbol('M');
var mySymbolN = Symbol.for('N');
var myObj101 = { l: 'love', [mySymbolM]: 'man', [mySymbolN]: 'new', o: 'open' };
for (var key in myObj101) console.log(myObj101[key]); // love // open
console.log(Object.keys(myObj101)); // ["l", "o"]
console.log(Object.getOwnPropertyNames(myObj101)); // ["l", "o"]
for (var symbol of Object.getOwnPropertySymbols(myObj101))
  console.log(symbol, myObj101[symbol]); // Symbol(M) "man" // Symbol(N) "new"
console.log(Object.getOwnPropertySymbols(myObj101)); // [Symbol(M), Symbol(N)]
console.log(Reflect.ownKeys(myObj101)); // ["l", Symbol(M), Symbol(N), "o"]
```

   b. 解析 function 下 arguments 的结构：

```
(function () {
  console.log(arguments); // {"0":1,"1":"2","2":3,"3":"四"}
  for (var key in arguments) console.log(arguments[key]); // 0 1 2 3
  for (var arg of arguments) console.log(arg); // 1 2 3 四
  console.log(Object.keys(arguments)); // ["0", "1", "2", "3"]
  console.log(Object.getOwnPropertyNames(arguments)); // ["0", "1", "2", "3", "length",
"callee"]
  console.log(Object.getOwnPropertySymbols(arguments)); // [Symbol(Symbol.iterator)]
})(1, '2', 3, '四');
```

6. 内置 Symbol 方法：Symbol.hasInstance(), Symbol.isConcatSpreadable(), Symbol.species(), Symbol.match(), Symbol.replace(), Symbol.search(), Symbol.split(), Symbol.iterator(), Symbol.toPrimitive(), Symbol.toStringTag(), Symbol.unscopables()

各类应用：

1. 定义仅供内部使用的对象属性或方法：

```
let myUser;
{
 const password = Symbol();
 const getPassword = Symbol();
 class User {
  constructor(usr, pwd) {
   this.username = usr;
   this[password] = pwd;
  }
  getUserName() { return this['username']; }
  [getPassword]() { return this[password]; }
 }
 myUser = new User('usr007', 'pwd007');
 console.log(myUser.getUserName()); // usr007
 console.log(myUser[getPassword]()); // pwd007
}
console.log(myUser.getUserName()); // usr007
console.log(myUser[getPassword]()); // ReferenceError: getPassword is not defined
```

2. 消除魔术字符串：

```
// 使用魔术字符串传入 String 类型参数
(function count1(language) {
  if (language === 'en') console.log('one-two-three');
  if (language === 'zh') console.log('一-二-三');
})('en'); // one-two-three
// 使用非魔术字符串传入 Symbol 类型参数
var LanguageType = { en: Symbol(), zh: Symbol() };
(function count2(language) {
  if (language === LanguageType.en) console.log('one-two-three');
  if (language === LanguageType.zh) console.log('一-二-三');
})(LanguageType.zh); // 一-二-三
```