27-第二十七章 面向对象OOP ECMAscrtipt5(三)

一、 ECMAScript 对象类型

> 在 ECMAScript 中，所有对象并非同等创建的。
> 一般来说，可以创建并使用的对象有三种：`本地对象`、`内置对象`和`宿主对象`、`自定义对象`。

本地对象包括：

```
1. Object
2. Function
3. Array
4. String
5. Boolean
6. Number
7. Date
8. RegExp
9. Error
10. EvalError
11. RangeError
12. ReferenceError
13. SyntaxError
14. TypeError
15. URIError
```

内置对象：

```
1. ECMA-262 只定义了两个内置对象，即 Global（window） 和 Math （它们也是本地对象，根据定义，每个内置对象都是本地对象）。
```

宿主对象:

1. 所有非本地对象都是宿主对象（host object），即由 ECMAScript 实现的宿主环境提供的对象。
2. 所有 BOM 和 DOM 对象都是宿主对象。

二、 对象中函数改写及this指向

- 对象中函数改写

```
var obj = {
        index:888,
          get:function(){
              console.log('this is  get');
          },
          set:function(){
              console.log('this is set');
          }
      }
      obj.get();//this is set
```

> 可以改写成如下

```
var obj = {
  index:888,
   get:function(){
        console.log('this is  get');
    },
    set:function(){
        console.log('this is set');
    }
 }
 obj.get();//this is set
```

- 对象中this指向，谁调用指向谁

```
var obj = {
      show:function(){
           console.log(this);//指向obj
       },
       name:{
           getname:function(){
               console.log(this);//指向name
           }
       }
   }
   obj.show();//指向obj
   obj.name.getname();//指向name
```

三、ECMAScript5 对象的属性方法

一、对象属性

1、`constructor`
对创建对象的函数的引用（指针）。对于 Object 对象，该指针指向原始的 Object() 函数。

二、对象方法

1、`hasOwnProperty(property)`
obj.`hasOwnProperty(` name `)` 来判断一个属性是否是自有属性，自身属性还是继承原型属性。必须用字符串指定该属性。返回true 或 false

```
  function fn(index){
       this.index = index;
   }
   fn.prototype.name = '二狗';
   var obj = new fn(88);
   console.log( obj.hasOwnProperty('index') );//true
   console.log( obj.hasOwnProperty('name') );//false
```

2、`isPrototypeOf(object)`

> obj.`isPrototypeOf(` obj.prototype `)` 判断该对象的原型是否为xxxxx。 返回true 或 false

```
1.    Obj.constructor.prototype.isPrototypeOf(Obj) //true
```

3、`propertyIsEnumerable()`

> `obj.propertyIsEnumerable(` 'name' `)` 判断对象给定的属性`是否可枚举`，即是否可用 `for...in` 语句遍历到,返回true 或 false

```
1. obj.propertyIsEnumerable('name')
```

> **getter** `/setter` ，函数

4.`getter`,`setter`: 返回property的值得方法，值：`function(){}`或`undefined`默认是`undefined`

```
1. var obj = {
2.         _name:'hello',
3.          get name(){
4.              console.log('get');
5.              return this._name;
6.          },
7.          set name(val){
8.              console.log('set');
9.              this._name = val;
10.            // return val;
11.         }
12.     };
13.
14.   console.log(obj.name);//'get'  'hello' //获取值会触发get 函数
15.   console.log(obj.name = '二狗');//'set'  '二狗' //设置值会触发set函
    数
```

5.`__defineGetter__()`,`__defineSetter__()` 定义setter getter 函数

> 在对象定义后给对象添加getter或setter方法要通过两个特殊的方法`__defineGetter__`和`__defineSetter__`。这两个函数要求第一个是getter或setter的名称，以string给出，第二个参数是作为getter或setter的函数。

```
var  obj = {
     _name : '二狗',
  };
  obj.__defineGetter__('name',function(){ return this._name});//定义 get name
  obj.__defineSetter__('name',function(val){ this._name = val});//set name
  console.log( obj.name );//'二狗'
  console.log( obj.name = '小黑' );//'小黑'
```

---

6. `__lookupGetter__`, `__lookupSetter__` 返回getter setter所定义的函数

语法：

> 1 obj.**lookupGetter**(sprop)

```
var  obj = {
       _name : '二狗',
        get name(){     return this._name;  },
        set name(val){  return this._name=val;  }
    }
    console.log( obj.__lookupGetter__('name') );
    //function get name(){return this._name;}
    console.log( obj.__lookupSetter__('name') );//'小黑'
    //function set name(val){ return this._name=val; }
```

六、ECMAScript5 Object的新属性方法

1、Object.defineProperty(O,Prop,descriptor) /

Object.defineProperties(O,descriptors)

> 定义对象属性

- `O` ————————————为已有`对象`
- `Prop` ————————————为`属性`
- `descriptor` ————————为属性`描述符`
- `descriptors` ————————-多个属性`描述符`？

> 在之前的JavaScript中对象字段是对象属性，是一个键值对，而在ECMAScript5中引入property，property有几个特征
> `Object.defineProperty` 及 `Object.defineProperties` 定义默认为:

1. `value`：值，默认是`undefined`
2. `writable`：是否可写，默认是`false`，
3. `enumerable`：是否可以被枚举(for in)，默认`false`
4. `configurable`：是否可以被删除，默认`false`

普遍定义的为

- 下面利用defineProperty为o对象定义age属性，并且添加`描述符`

```
var o ={}
Object.defineProperty(o,'age', {
        value: 24,
        writable: true,
        enumerable: true,
        configurable: true
    });
    alert(o.age);//24
```

- 下面defineProperties为o对象添加 多个`描述符`

```
var o ={}
 Object.defineProperties(o,{
     age:{
          value: 24,
             writable: true,
              enumerable: true,
              configurable: true
      },
      name:{
           value: 'hello',
           writable: true,
           enumerable: true,
           configurable: true
      }
  });
  var val = o.age;//''get'
  alert(val);//24
```

- 下面defineProperties中特殊的get set

```
function fn(name){
 this._name = name;
 }

 var obj = new fn('二狗');

 Object.defineProperties(obj,{
     index:{value:1},
     _age:{value:123,writable:true},
     age:{
         //此处不能写value了
         get:function(){
             console.log('get');
             return this._age;
         },
         set:function(val){
             console.log('set');
             this._age = val;//_age 属性必须是writable:true，否则为set函数失效

         }
         /*此处get set 可以写成如下方式
         get(){
             console.log('get');
             return this._age;
         },
         set(val){
             console.log('set');
             this._age = val;//_age 属性必须是writable:true，否则为set函数失效

         }
         */



     }

 });
 console.log( obj.age );// get 123
 obj.age = 888;//set
 console.log( obj.age );// get 888

```

2、Object.getOwnPropertyDescriptor(O,property)

> 获取对象的自有的指定的`属性描述符`

```
var Des = Object.getOwnPropertyDescriptor(obj,'hello');
alert(Des);//{value: undefined, writable: true, enumerable: true, configurable: true}
```

3、Object.keys(O,property)

> 获取所有的可枚举的属性名，非继承,返回数组

```
console.log(Object.kes( obj ); //["hello"]
```

4、Object.getOwnPropertyNames(O)

> 获取所有自有的属性名，非继承

```
console.log(Object.getOwnPropertyNames(obj)); //["hello", "index"]
```

5、`Object.create(` O, descriptors `)`

> `Object.create(` O,descriptors `)` 这个方法用于`创建一个对象`，并把其prototype属性赋值为第一个参数，同时可以设置`多个descriptors`，第二个参数为可选，

- 以第一个参数为原型创建一个对象，即让新对象继承O

```
 var obj =Object.create({
      name:'小黑',
      age:20
});
```

- 以第一个参数为原型创建一个对象,并且多个`属性描述符`

```
var obj = Object.create({
    name:   '小黑',
    age:20

 },
 {
        hello:{
            value:'00000',
            writable: true,
          enumerable: true,
          configurable: true},
        index:{
            value:'8888',
            writable: false,
          enumerable: false,
          configurable: false }

});
 alert( obj.index);//8888
```

6、`Object.preventExtensions(O)` / Object.isExtensible()

> `Object.preventExtensions(O)` `阻止对象拓展`，即：不能增加新的属性，但是属性的值仍然可以更改，也可以把属性删除，
> `Object.isExtensible(O)` 用于判断对象是否可拓展

```
        console.log(Object.isExtensible(o)); //true
        o.lastName = 'Sun';
        console.log(o.lastName);    //Sun ,此时对象可以拓展
        ////////////////////////////
        Object.preventExtensions(o);
        console.log(Object.isExtensible(o)); //false

        o.lastName = "ByronSun";
        console.log(o.lastName); //ByronSun，属性值仍然可以修改

        //delete o.lastName;
        console.log(o.lastName); //undefined仍可删除属性

         o.firstname = 'Byron'; //Can't add property firstname, o
bject is not extensible 不能够添加属性
```

7、`Object.seal(O)` / Object.isSealed()

> `Object.seal(O)`方法用于把对象`密封`，也就是让对象既不可以拓展也不可以删除属性（把每个属性的`configurable`设为false）,单数属性值仍然可以修改，`Object.isSealed()`由于判断对象是否被密封

```
        Object.seal(o);
        o.age = 25; //仍然可以修改
        delete o.age; //Cannot delete property 'age' of #<Object>
```

8、`Object.freeze(O)` / Object.isFrozen()

> 终极神器，完全`冻结对象`，在seal的基础上，属性值也不可以修改（每个属性的`wirtable`也被设为`false`）