11-第十一章 对象(JSON) for/in function[all apply bind]

—JSON

1 、创建对象（ JSON ）

> 对象是Javascript的基本数据结构，对象是引用类型
> 创建对象的三种方式 `对象直接量` ， `new Oject()` , `Object.create({})` [ES5],create创建需要一个对象参数

```
//直接量
var  obj = {}
//new
var  obj = new Object()
//ES5
var  obj = Object.create()
```

- 对象都是一个 `key(键)` : `value( 值 )` 一一对应
- age为对象的 `key(键)` ，或obj的age属性
- 20为age 的 `value( 值 )`

> `严格是对象为JSON`

```
//对象
var obj = {  age: 20,name: '小黑', }
//JSON
var obj = {  'age' : 20,'name': '小黑', }

```

2 、访问JSON的值

obj`.`attribute
`obj[`attribute`]`

```
var  obj = {
    age: 20,
    name: '小黑',
    sex: '男'
}

alert( obj.age ) //20
//或
alert( obj[age] ) //20
```

3 、修改JSON的属性值

```
var obj = { name: '小黑'};
obj.name  = '二狗';
//或
obj[name ] = '二狗';
```

3 、添加JSON属性

```
var obj = {};
obj.name = 'hello';
```

4 、删除JSON属性

`delete` 可以删除对象属性

```
var obj = {};
obj.name = 'hello';

delete obj.name
//或
delete obj[name]
```

5 、JSON数字属性

```
var obj = {
    age: 20,
    name: '小黑',
    sex: '男'

};
obj[0] = 'hello';
obj[1] = 'AAA';

```

6、 `in` 判断对象是否存在某个属性

```
var   obj = { hello:123};
 alert( 'hello' in obj );//true
```

二 、for in遍历json

1、for in 遍历JSON

```
var person = {
    age: 20,
    name: '小黑',
    sex: '男'

}

for( var attr in person){//attr 为属性，attr不是必须的，可以为任意变量名
    console.log( attr );//属性名：  age,name,sex
    console.log( person[attr] )//对应的属性值：20 小黑 男
}
```

2、for in 也可以遍历数组

```
var  arr = [8,2,4,6];

for ( var index in arr){
    console.log( index );                    //0 1 2 3
    console.log( arr[index] );              // 8 2 4 6
}
```

!!!! for 循环不能遍历JSON

三、JSON.parse() 对象化 / JSON.stringify() 对象字符化

1、`JSON.parse()`

> `JSON.parse(` obj `)` 方法解析一个JSON `字符串` ，构造由字符串描述的JavaScript值或对象。可以提供可选的reviver函数以在返回之前对所得到的对象执行变换。

```
var obj =  '{
    "age": 20,
    "name": "小黑",
    "sex": "男"

}';

JSON.parse( obj );

//解析后的值为
obj =  {
    age: 20,
    name: "小黑",
    sex: "男"

};

```

1、`JSON.stringify()`

> `JSON.stringify( obj )`与`JSON.parse()`进行的是反操作

```
JSON.stringify({});                   // '{}'
JSON.stringify(true);                 // 'true'
JSON.stringify("foo");                // '"foo"'
JSON.stringify([1, "false", false]);  // '[1,"false",false]'
JSON.stringify({ x: 5 });             // '{"x":5}'

JSON.stringify({x: 5, y: 6});
// "{"x":5,"y":6}"
```

四、JSON对象仿`jQuery` 链式操作 css html

```
1.
2.  function $(option){
3.      var t = typeof option;
4.      if(t=='function'){
5.          window.onload = option;
6.      }else if(t.toLowerCase() == 'string' ){
7.
8.          var el = option.substring(1,option.length);
9.          //alert(option.length);
10.         el = document.getElementById(el);
11.
12.     }
13.     var obj = {
14.         css:function(attr,val){
15.             el.style[attr] = val;
16.             return obj;
17.         },
18.         html:function(val){
19.             el.innerHTML = val;
20.             return obj;
21.         }
22.     }
23.     return obj;
24. }
25. $('#box').css('backgroundColor','red').html('hello');
26.
```

七、Function `call()` `applay()` `bind()`方法

> 函数的`call()`,`apply()`,`bind()`方法都是用于`改变`函数内部`this指向`
> 用法`Function.call()`,`Function.apply()`,`Function.bind()`,

1、`call()`和`apply`都用于函数调用

```
function fn(){  alert(this) }

fn();//window
fn.call('hello');//'hello'
fn.apply('8888');//'8888'
```

区别：
`call(` thisvalue, val1，val2，…. `)`

- `thisvalue` 是函数内部`this的值`
- 后面是参数列表

`apply(` thisvalue, [val1，val2，….] `)`

- `thisvalue` 是函数内部`this的值`
- 后面是参数`数组`，所有参数放数组里面

2、`bind()`都用于创建中

1)、适用匿名函数

```
        var fn = function (a,b ){
            console.log(this,a,b);
        }.bind('hello',1,2);
        fn();
        ////////////////////////////

        var fn = function (a,b ){
            console.log(this,a,b);
        }.bind('hello');
        fn(1,2);

        ////////////////////////////////////
        obj.onclick = function(){

        }.bind()

```

2)、有名函数，有些特殊

```
    function fn(){
      console.log(this);
    }
    fn.bind('hello')();
```

3）、自执行函数

```
(function abc(){
   console.log(this);
}.bind('hello')());
```

```

(function abc(){
    console.log(this);
 }.bind('hello'))();


```

```
(function abc(){
        console.log(this);
 }).bind('hello')();
```

五、ES5对象属性/方法

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。

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

```
obj.hasOwnProperty("name")
```

2、`isPrototypeOf(object)`

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

```
obj.isPrototypeOf( object )
```

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

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

> **getter** `/setter` ，函数

5. `get`: 返回property的值得方法，值： `function(){}` 或 `undefined` 默认是 `undefined`
6. `set` ： 为property设置值的方法，值： `function(){}` 或 `undefined` 默认是 `undefined`

```
var  obj = {
       _name:123,
       get hell(){
         alert('get');
         return this._name;
       },
       set hell(val){
          alert('set');
         this._name = val;
       }
    };
    obj.hell;//get
    obj.hell =  555;//set
```

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` ————————-属性及描述`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

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

    - 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为添加getter

```
var o ={}
Object.defineProperties(o, {
      age:{
                  value: 24,
                writable: true,
                enumerable: true,
                configurable: true
         },
     name:{
                  value: '小黑',
              writable: true,
              enumerable: true,
              configurable: true,
              get value(){//函数名一定得一样
                       alert('get');
                      //alert(this.value);
                      return this._value;
                  },

         }
    });

 o.name;//‘get’ 访问o的name属性会访问getvalue函数

```

2、`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

```
 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
```

3、Object.getOwnPropertyDescriptor(O,property)

> 获取对象的指定的`属性描述符`

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

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.keys(O,property)

> 获取所有的可枚举的属性名，非继承

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

5、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

```
1.          console.log(Object.isExtensible(o)); //true
2.          o.lastName = 'Sun';
3.          console.log(o.lastName);    //Sun ,此时对象可以拓展
4.          ////////////////////////////
5.          Object.preventExtensions(o);
6.          console.log(Object.isExtensible(o)); //false
7.
8.          o.lastName = "ByronSun";
9.          console.log(o.lastName); //ByronSun，属性值仍然可以修改
10.
11.         //delete o.lastName;
12.         console.log(o.lastName); //undefined仍可删除属性
13.
14.          o.firstname = 'Byron'; //Can't add property firstname, o
    bject is not extensible 不能够添加属性
```

6、`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

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

7、`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`）