面向对象（二）

之前我们已经讲过了对象的封装（也就是对象的创建过程），那么有没有一种可能对象创建以后会发生变化

```
var obj1 = {
    stuName: "曹方",
    sex: "女",
    age: 18,
    height: 168,
    weight: 50
}
```

问题:

1. 当这个对象创建好了以后，这个对象里面有些属性是不能公开，如我们希望`age`这个属性不要，怎么办?
2. 当这个对象创建好了以后，我们希望`sex`这个属性的值不能更改，怎么办呢?

针对上面的问题，如果我们还使用之前的方式去创建对象就会有问题

delete关键字

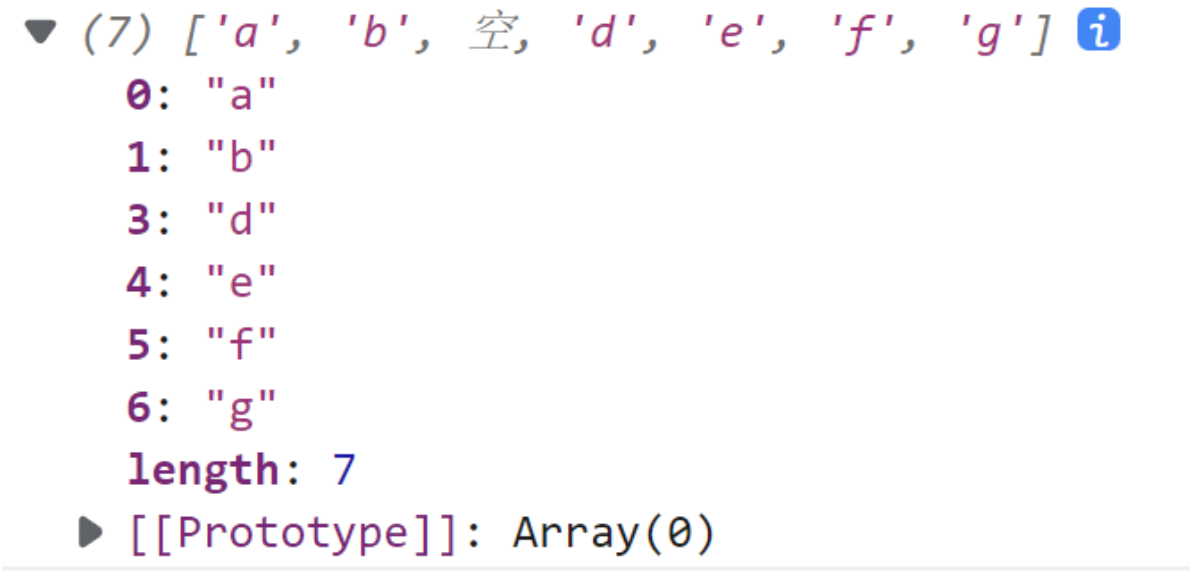
如果我们希望某一个属性不要了，就可以使用`delete`把这个属性把它删掉

```
delete obj1.age;            //删除obj1对象上面的age属性
```

> 上面的操作会返回一个布尔值，如果返回的是`true`就代表这个属性删除成功，如果返回`false`就代表这个属性删除失败

扩展

```
var arr = ["a","b","c","d","e","f","g"];
//我想删除"c"怎么办
delete arr[2];
```



因为 `delete` 是可以删除属性的，而数组的索引也是一个属性，也就可以附表和 `delete` 去删除，不过我也发现了通过 `delete` 删除以后数组是不会呈现队列效应（沙漏效应）的

❓ **思考一下**：为什么我们执行 `delete arr.length` 的时候会得到 `false`，也就是删除失败?

如果想定义一个不能被删除的属性，则必须要使用特殊的方式来定义属性

Object.defineProperty来定义属性

如果想定义一些特殊的属性，我们就需要使用刚刚说的这个方法，它可以定义一些不能删除的属性，不能更改的属性等一系列操作

所有的属性其实都可以分为2类

1. 数据属性
2. 访问器属性

数据属性

1. 数据属性

数据属性包含一个数据值的位置。在这个位置可以读取和写入值。数据属性有 4 个描述其行为的特性。

- `[[Configurable]]`：表示能否通过 `delete` 删除属性从而重新定义属性，能否修改属性的特性，或者能否把属性修改为访问器属性。像前面例子中那样直接在对象上定义的属性，它们的这个特性默认值为 `true`。
- `[[Enumerable]]`：表示能否通过 `for-in` 循环返回属性。像前面例子中那样直接在对象上定义的属性，它们的这个特性默认值为 `true`。
- `[[Writable]]`：表示能否修改属性的值。像前面例子中那样直接在对象上定义的属性，它们的这个特性默认值为 `true`。
- `[[Value]]`：包含这个属性的数据值。读取属性值的时候，从这个位置读；写入属性值的时候，把新值保存在这个位置。这个特性的默认值为 `undefined`。

在定义属性的时候，我们可以通过上面的4个设置来定义特殊的属性，现在我们来定义第一个特殊的对象的属性

```
1  var stu1 = {
```

```
    age: 18
}
//现在，我们希望`stu1`对象上面有一个属性stuName，不能被删除
//如果想定义一个特殊的属性，要使用Object.defineProperty来完成
Object.defineProperty(stu1, "stuName", {
    //这一个对象就描述了属性stuName它的特征
    value: "张三",
    configurable: false            //决定当前属性不可以被删除
});

//现在我想定义一个性别sex的属性，但是这个属性不可以更改它的值
Object.defineProperty(stu1, "sex", {
    value: "男",
    writable: false                //当前属性值不可以被更改，相当于只读
})
```

访问器属性

2. 访问器属性

访问器属性不包含数据值；它们包含一对儿 getter 和 setter 函数（不过，这两个函数都不是必需的）。在读取访问器属性时，会调用 getter 函数，这个函数负责返回有效的值；在写入访问器属性时，会调用 setter 函数并传入新值，这个函数负责决定如何处理数据。访问器属性有如下 4 个特性。

- [[Configurable]]：表示能否通过 delete 删除属性从而重新定义属性，能否修改属性的特性，或者能否把属性修改为数据属性。对于直接在对象上定义的属性，这个特性的默认值为 true。
- [[Enumerable]]：表示能否通过 for-in 循环返回属性。对于直接在对象上定义的属性，这个特性的默认值为 true。
- [[Get]]：在读取属性时调用的函数。默认值为 undefined。
- [[Set]]：在写入属性时调用的函数。默认值为 undefined。

```
// 这就是一个普通的对象
var obj1 = {
    birthday: "2002-1-10"
}
//我们现在定义一个年龄的属性
Object.defineProperty(obj1, "age", {
    configurable: false,          //不可以被删除
    get: function () {
        //在获取这个属必值的时候自动调用
        console.log("我要取age属性的值");
    },
    set: function () {
        //在设置这个属性的值的时候自动调用
        console.log("你现在是不是要对age属性赋值")
    }
});

obj1.age;            //在调用obj1的age属性的时候,get方法会被调用
obj1.age = 18;       //在设置obj1的age属性的值的时候,set方法会被调用
```

> 上面的 `age` 属性就是一个访问器属性
>
> 1. `get` 函数与 `set` 函数是在对 `age` 属性进行访问 或设置的时候自动调用的
> 2. `age` 属性是没有 `value` ，所以它没有存储具体的值

问题：访问器属性的用处到底在哪里？

```
var obj1 = {
    // 这是一个身份证号
    IDCard: "420984199009081014"
}
//现在定义一个属性sex,这个sex的值是根据当前的身份证号来决定的
//字符串可以像数组一样操作
Object.defineProperty(obj1, "sex", {
    configurable: false,
    get: function () {
        //取值的时候调用
        return this.IDCard[16] % 2 == 0 ? "女" : "男";
    }
});

console.log(obj1.sex);                      //男

obj1.IDCard = "420984199009081082";
console.log(obj1.sex);                      //女
```

通过上面的访问器属性，我们可以看到，当我们设置了一个对象的身份证号属性 `IDCard` 以后，我们再去获取 `sex` 的时候，它会自动调用 `get` 函数，而 `get` 函数会根据 `IDCard` 来计算性别，这样会非常方便

> 有一句话是这么说的，访问器属性也叫联动属性

访问器的优点：

1. 通过 `get` 函数我们可以让一个属性值以另一个属性值为参照
2. 我们可以通过 `get/set` 来决定属性的取值与赋值操作。在上面的代码里面，我们没有设置 `set` 方法，就说明 `sex` 这个属性不能被赋值，只能被取值

现在我们来看一下访问器属性里面的 `set` 的函数的用法。

```
var obj1 = {
    // 定义了姓的属性
    firstName: "杨",
    //定义了名的属性
    lastName: "标"
```

```
}

//现在我们想定义一个`userName`的访问器属性
Object.defineProperty(obj1, "userName", {
    configurable: false,
    get: function () {
        return this.firstName + this.lastName;
    },
    set: function (v) {
        //在赋值的时候会自动调用set函数
        console.log("赋值");
        // console.log(v);
        this.firstName = v[0];          //张
        this.lastName = v[1];           //三
    }
});

obj1.userName = "张三";
```

```
<script>
    var obj1 = {
        // 定义了姓的属性
        firstName: "杨",
        //定义了名的属性
        lastName: "标"
    }

    //现在我们想定义一个`userName`的访问器属性
    Object.defineProperty(obj1, "userName", {
        configurable: false,
        get: function () {
            return this.firstName + this.lastName;
        },
        set: function (v) {
            //在赋值的时候会自动调用set函数
            console.log("赋值");
            // console.log(v);
            this.firstName = v[0];          //张
            this.lastName = v[1];           //三
        }
    });

    obj1.userName = "张三";
</script>
```

这里要注意，参数v就是要赋的值

通过上面的上面的代码我们可以很清楚的看到一点，访问器属性本身是没有任何值存储的，它所有的值都依赖于其它的属性，`get`负责取值的操作，`set`负责赋的操作，它们都是自动的

场景：通过访问器属性来控制属性值的设置与读取

```
// 想定义一个年龄age，这年龄可以取值，但是赋值的时候只能大于或等于18
var stu1 = {
```

```
    userName: "张三",
    _aaa: 18              //这是一个普通的属性，可以存数据
}

//访问器属性本身不存储任何数据，它要依赖于其它属性`_aaa`
Object.defineProperty(stu1, "age", {
    configurable: false,
    get: function () {
        //get函数负责取值，在调用age属性的时候返回了`_aaa`的值
        return this._aaa;
    },
    set: function (v) {
        //在赋值的时候 v就是要赋的值，如果这个值大于等于18.就正常赋值
        //同时因为访问器属性本身不存储任何数据，所以它只能依赖于其它属性
        //最终这值是赋到了`_aaa`上面
        if (v >= 18) {
            //正常赋值
            this._aaa = v;
        }
    }
});
```

Object.defineProperties来定义属性

之前我们已经使用`Object.defineProperty`来实现特殊属性的定义，但是这个方法只能一次性的定义一个属性，如果我们要定义多个特殊的属性则要多次的调用这个方法，比较麻烦

`Object.defineProperties`这个方法就可以一将从 性的定义多个特殊的属性，它的语法格式如下

```
Object.defineProperties(对象,{
   属性名1:{
       //相关的特性
   },
   属性名2:{
        //相关的特性
   }
});
```

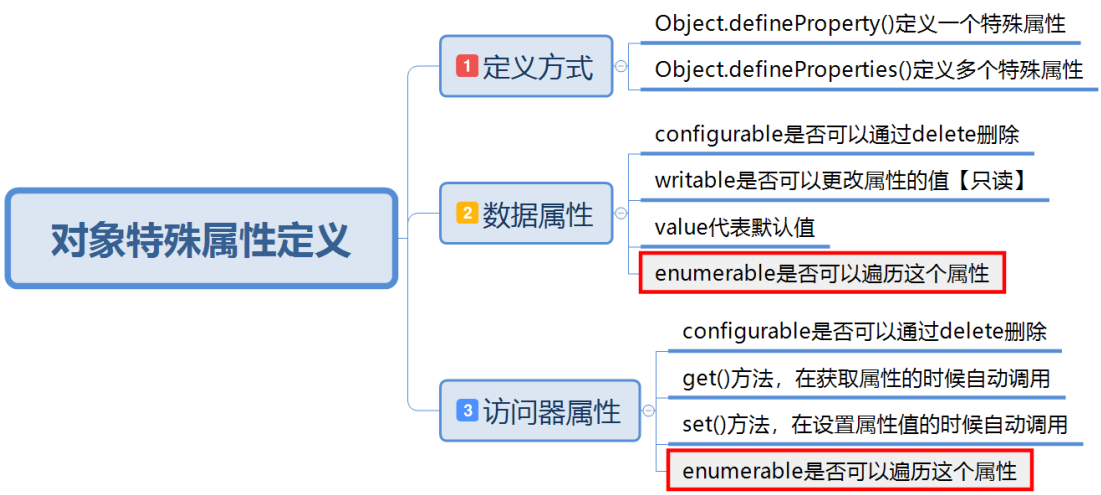
通过上面的方式我们就可以同时定义多个特殊属性

```
var obj1 = {
    userName: "张三",
}
//obj1上面有一个age属性，这个属性不能被删除  默认值是18
//obj1上面有一个sex属性，这属性不能被修改【只读】  默认值是男

```

```
Object.defineProperties(obj1, {
    age: {
        configurable: false,
        value: 18
    },
    sex: {
        value: "男",
        writable: false
    }
});

```



遍历对象的属性

> 其实这个东西早看应该讲了，只是时机不成熟

遍历对象的属性就是把对象当中的所有属性名都拿出来

for...in遍历对象

3.6.5 for-in语句

for-in 语句是一种精准的迭代语句，可以用来枚举对象的属性。以下是 for-in 语句的语法：

```
for (property in expression) statement
```

下面是一个示例：

```
for (var propName in window) {
     document.write(propName);
}
```

ForInStatementExample01.htm

在这个例子中，我们使用 for-in 循环来显示了 BOM 中 window 对象的所有属性。每次执行循环时，都会将 window 对象中存在的一个属性名赋值给变量 propName。这个过程会一直持续到对象中的所有属性都被枚举一遍为止。与 for 语句类似，这里控制语句中的 var 操作符也不是必需的。但是，为了保证使用局部变量，我们推荐上面例子中的这种做法。

ECMAScript 对象的属性没有顺序。因此，通过 for-in 循环输出的属性名的顺序是不可预测的。具体来讲，所有属性都会被返回一次，但返回的先后次序可能会因浏览器而异。

但是，如果表示要迭代的对象的变量值为 null 或 undefined，for-in 语句会抛出错误。ECMAScript 5 更正了这一行为；对这种情况不再抛出错误，而只是不执行循环体。为了保证最大限度的

上面的话是“红宝书”当中的话，如果想遍历对象里面的属性我们需要使用 `for...in`

```
var obj1 = {
    userName: "张三",
    age: 18,
    sex: "男"
}
//现在我希望把上面所有的属性都拿出来打印一遍，怎么办?
for(var i in obj1){
    console.log(i);             //这个时候的i就是所有的属性名
    //userName,age,sex三个打印结果
}
```

`for...in` 是我们第一种遍历对象属性的方式，只这个属性的 `enumerable` 不为 `false` 则可以正常的遍历出来

```
var obj1 = {
    userName: "张三",
    age: 18
}

//单独的去定义一个特殊属性sex
Object.defineProperty(obj1,"sex",{
    value:"男",
    enumerable:false        //这里因为设置false，所以就不能被for...in遍历
})

for(var i in obj1){
   console.log(i);
    //userName,age
```

```
}
```

扩展

```
var arr = ["a", "b", "c", "d", "e", "f"];
// 遍历上面的数组
/*
    for (var i = 0; i < arr.length; i++) {
        console.log(arr[i]);
    }
*/

/*
    arr.forEach(function (item, index, _arr) {
        console.log(item);
    })
*/

for (var i in arr) {
    console.log(arr[i]);
}
```

通过Object.keys()遍历对象

在`JavaScript`的内部，有一个方法叫`Object.keys()`

之前我们在学习对象的创建的时候有一种字面量创建法，如下

```
var obj1 = {
    属性名:属性值
}
//其实也有另一种叫法
var obj2 = {
    键:值
}
//这种写法叫 键值对 ，键的英文单词是key，值的英文单词是value
//属性名也叫key
```

顾名思义`Object.keys()`就是获取所有的属性名，但是它只能获取`enumerable:true`的属性名

```
var obj1 = {
    userName: "张三",
    age: 18
}

```

```
//单独的去定义一个特殊属性sex
Object.defineProperty(obj1, "sex", {
    value: "男",
    enumerable: false       //不能被遍历
});

var arr = Object.keys(obj1);
//这个时候的arr就会得到`obj1`对象里面所有`enumerable:true`的属性名["userName","age"]
arr.forEach(function(item){
    console.log(item,obj1[item]);
    //如果想打印属性值，怎么办?
})
```

`Object.keys(对象)`它可以把当前这个对象里面所有`enumerable:true`的属性拿过来组成一数组，然后我们再遍历这个属性数组就相当于遍历了整个对象

注意：上面的2种方式只能遍历`enumerable:true`的属性

思考：有没有什么办法可以遍历对象当中的所有属性？【把`enumerable:false`的也遍历出来】

Object.getOwnPropertyNames()方法遍历

这一个方法可以获取某一个对象上面的所有的属性名，它会忽略掉`enumerable`这个特性

```
var obj1 = {
    userName: "张三",
    age: 18
}

//单独的去定义一个特殊属性sex
Object.defineProperty(obj1, "sex", {
    value: "男",
    enumerable: false       //不能被遍历
});

//for...in
//Object.keys(obj1)

//这个方法会获取`obj1`对象上面自身所有的属性
var arr = Object.getOwnPropertyNames(obj1);
// ['userName', 'age', 'sex']
arr.forEach(function(item,index,_arr){
    console.log(item);
})
```

构造函数与特殊属性结合

构造函数是可以帮我们快速的创建对象，而`Object.defineProperty`可以帮我们来创建一些特殊的属性，那么这两个东西是可以结合的

场景：现在我希望创建我们班所有学生的对象【65人】，每个学生都具备`userName,sex,age,birthday`这4个属性，其中`sex`属性一旦设置就不可更改，`age`是一个仅仅只可以访问的属性，并且`age`属性的值要通过`birthday`来决定，`birthday`这个属性一定设置也不可以更改

对于上面的场景需要，我们应该怎么办呢?

```
//先不考虑特殊的属性，只考虑普通的属性
function Student(userName, sex, birthday) {
    this.userName = userName;
    this.sex = sex;
    this.birthday = birthday;
}

var s1 = new Student("颜一鸣", "男", "2000-12-1");
```

上面的代码肯定是不符合我们的要求的，怎么办呢?

现在我们已经知道构造函数所创建的对象是一个普通属性的对象，如果想包含特殊的属性只能使用`Object.defineProperty`去完成。现在关键问题就是如果将2者结合

要完成上面的问题先思考一下，我们的构造函数内部主要干了什么事情?

1. 创建了一个新的对象
2. `this`指向了这个新的对象
3. 执行构造函数中的代码（为这个新对象添加属性）；
4. 返回这个新对象

问题就在第3步，我要对这个新对象添加特殊属性

```
function Student(userName, sex, birthday) {
    this.userName = userName;
    Object.defineProperties(this, {
        sex: {
            value: sex,
```

```
            writable: false
        },
        birthday: {
            value: birthday,
            writable: false
        },
        age: {
            get: function () {
                // 希望通过生日来获取年龄
                // 用当前日期 - 生日 = 年龄
                var now = new Date();
                var birthdayDate = new Date(this.birthday);
                var times = now - birthdayDate;
                var yearCount = ~~(times / 1000 / 60 / 60 / 24 / 365);
                return yearCount;
            }
        }
    });
}

var s1 = new Student("颜一鸣", "男", "2000-12-1");
```

1. `age` 是一个访问器属性通过 `get` 函数来计算年龄，它依赖于 `birthday` 属性（这个计算过程没有学，可以先不管）
2. 我们将构造函数与特殊属性进行了结合
3. 像上面这种方式我们平常在工作中经常使用

获取对象属性的描述信息

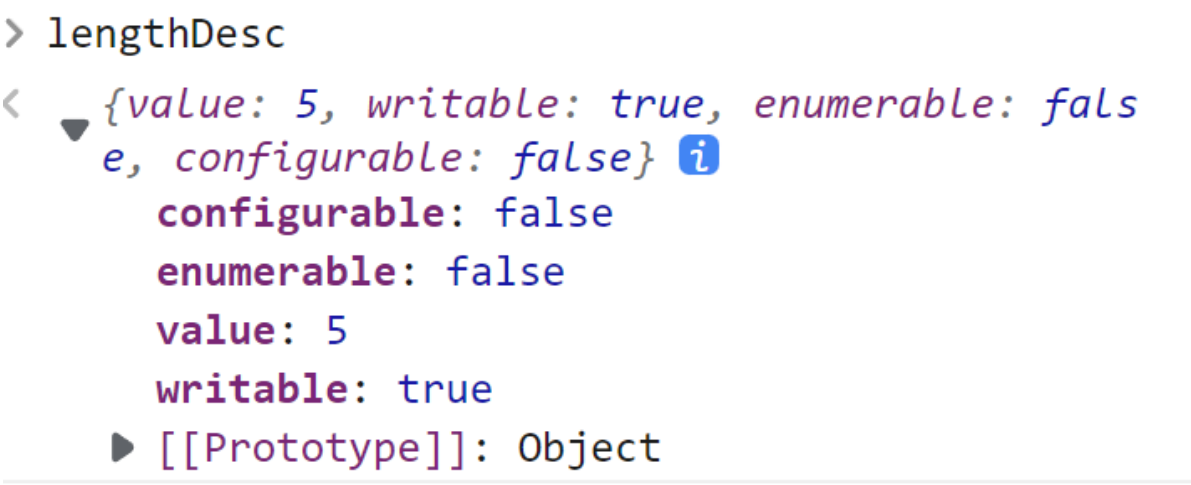
获取对象属性的描述信息也叫获取对象属性的特征，当我们得到一个对象以后，我们想知道某些属性是否可以更改，某一个是否可以被删除，怎么办呢？

在 `JS` 里面，有一个方法是可以获取对象属性的描述信息的

```
var 描述信息 = Object.getOwnPropertyDescriptor(对象,属性);
```

```
var arr = ["a", "b", "c", "d", "e"];
//length可以遍历吗?
//length可以删除吗?
// 如果想获取某珍上属性的特征，是可以通过一个方法的

var lengthDesc = Object.getOwnPropertyDescriptor(arr, "length");
```



看到了上面的描述信息，我们就知道了`length`属性不能被删除，也不能被`for...in`遍历，但是可以更改长度

判断对象是否具备某一个属性

假设现在我们有一个对象，我们需要判断某一个对象是否具备某一个属性，怎么办呢?

```
var stu1 = {
    userName: "李心悦",
    sex: "女"
}
Object.defineProperty(stu1, "age", {
    value: 18,
    enumerable: false,          //是否可遍历
    configurable: false,
    writable: false
});
```

当判断某一个对象是否具备某一个属性的时候一定要注意我们之前所学习的对象的属性的遍历`Object.keys()`以及`for...in`都只能够将`enumerable:true`的属性遍历出来，所以在判断一个对象是否具备一个属性的时候是不能够使用这2个方法的

通过`in`关键字来检测

判断某一个对象是否具备 某一从此属性我们可以使用关键字`in`来完成，它的语法格式如下

```
属性 in 对象;
```

如果上面这个结果是`true`就代表这个对象具备这个属性，如果是`false`就代表这个对象不具备这个属性

```
console.log("userName" in stu1);        //true
console.log("sex" in stu1);             //true
console.log("age" in stu1);             //true
console.log("a" in stu1);               //false
```

通过 hasOwnProperty() 来完成检测

在每个对象的内部，都会有一个方法叫 `hasOwnProperty("属性名")` 用来检测当前这个对象是否包含这个属性，如果得到的结果是 `true` 就代表当前这个对象包含这个属性， `false` 则不包含

```
stu1.hasOwnProperty("userName");        //true
stu1.hasOwnProperty("sex");             //true
stu1.hasOwnProperty("age");             //true
stu1.hasOwnProperty("aaa");             //false
```

在上面的2种判断方法里面，我们都可以完成一个对象对于属性是否存在的检测。**对于是否具备某一个属性应该忽略掉 `enumerable:false` 这个条件，因为这个条件只是用来控制遍历的，不是用于来控制是否存在**

区别:

1. `in` 关键字去检测属性是否存在的时候它会到父级对象去检测
2. `hasOwnProperty()` 只是用于检测自己有不有这个属性，不会到父级对象上面去检测

```
    var arr = ["a", "b", "c", "d", "e"];

    console.log("length" in arr);                   //true
    console.log(arr.hasOwnProperty("length"));      //true

    console.log("push" in arr);                     //true
    console.log(arr.hasOwnProperty("push"));        //false  自身没有，push方法是在父级的
```

练习

1. 根据要求创建对象
    - 定义一构造函数Person,它有5个属性，分别是姓名(userName)，性别(sex)，年龄(age)，生日(birthday)，身份证号(IDCard)，
    - 用户在创建对象的时候，这个构造函数接收二个参数，分别是姓名(userName)，和身份证号(IDCard)
    - 所有的属性都不能被delete,也不能更改它的初始值
    - 所有的属性都可以被 for...in遍历出来
    - 根据身份证号来得到用户的性别
    - 根据身份证号来得到用户的生日（身份证号里面是包含生日的，如420984199905041016）。
    - 根据生日来得到年龄（这里可以参考标哥之前的代码）

- 根据上面定义好的构造函数去创建一个对象，然后再去遍历这个对象，打印这个对象里面的属性名与当前属性的值

提示：字符串可以通过索引取到里面的某个值`var str ="abc"; str[0]=="a"`，字符串也可以像数组一样去操作，有`slice`提取的方法

```
function Person(userName, IDCard) {
    Object.defineProperties(this, {
        userName: {
            configurable: false,
            writable: false,
            value: userName,
            enumerable: true
        },
        IDCard: {
            configurable: false,
            writable: false,
            value: IDCard,
            enumerable: true
        },
        sex: {
            configurable: false,
            enumerable: true,
            get: function () {
                // return ["女","男"][this.IDCard[16] % 2];
                return this.IDCard[16] % 2 == 0 ? "女" : "男";
            }
        },
        birthday: {
            configurable: false,
            enumerable: true,
            get: function () {
                var year = this.IDCard.slice(6, 10);
                var month = this.IDCard.slice(10, 12);
                var day = this.IDCard.slice(12, 14);
                // return year + "-" + month + "-" + day;
                return [year, month, day].join("-");
            }
        },
        age: {
            configurable: false,
            enumerable: true,
            get: function () {
                var now = new Date();
                var birthdayDate = new Date(this.birthday);
                var times = now - birthdayDate;
                var yearCount = ~~(times / 1000 / 60 / 60 / 24 / 365);
                return yearCount;
            }
        }
```

```
    });
}

var p1 = new Person("张珊", "420984199801021019");
for (var i in p1) {
    console.log(i, p1[i]);
}
```

2. 请从下面的对象当中遍历所有的属性，如果这个属性是一个方法就调用这个方法

   同时，在遍历的过程当中要把所有`enumerable:false`的属性标记出来

```
var obj1 = {
    userName: "张珊"
}

Object.defineProperties(obj1, {
    sex: {
        enumerable: false,
        value: "男"
    },
    age: {
        enumerable:false,
        get: function () {
            return 19;
        }
    },
    sayHello: {
        enumerable: false,
        value: function () {
            console.log("大家好，我叫" + this.userName);
        }
    }
})
```

3. 请将下面对象当中所有的属性值全部打印出来

```
var teacherInfo = {
    teahcerName: "标哥哥",
    age: 19,
    sex: "男",
    money: 1000
}
var stu1 = {
    userName: "张珊",
    sex: "女",
    age: 18,
```

```
    aaa: teacherInfo,
    hobby: ["看书", "睡觉"]
}

//请将stu1所有的属性值全部打印一遍
//张珊 女 18 标哥哥 19 男 1000 看书 睡觉
```