# hasOwnProperty函数

hasOwnProperty是Object.prototype的一个方法，它可是个好东西，他能判断一个对象是否包含自定义属性而不是原型链上的属性，因为hasOwnProperty 是 JavaScript 中唯一一个处理属性但是不查找原型链的函数。

// 修改Object.prototype

Object.prototype.bar = 1;

var foo = {goo: undefined};

foo.bar; // 1

'bar' in foo; // true

foo.hasOwnProperty('bar'); // false

foo.hasOwnProperty('goo'); // true

只有 hasOwnProperty 可以给出正确和期望的结果，这在遍历对象的属性时会很有用。 没有其它方法可以用来排除原型链上的属性，而不是定义在对象自身上的属性。

但有个恶心的地方是：JavaScript 不会保护 hasOwnProperty 被非法占用，因此如果一个对象碰巧存在这个属性，就需要使用外部的 hasOwnProperty 函数来获取正确的结果。

var foo = {

hasOwnProperty: function() {

return false;

},

bar: 'Here be dragons'

};

foo.hasOwnProperty('bar'); // 总是返回 false

// 使用{}对象的 hasOwnProperty，并将其上下为设置为foo

{}.hasOwnProperty.call(foo, 'bar'); // true

当检查对象上某个属性是否存在时，hasOwnProperty 是唯一可用的方法。同时在使用 for in loop 遍历对象时，推荐总是使用 hasOwnProperty 方法，这将会避免原型对象扩展带来的干扰，我们来看一下例子：

// 修改 Object.prototype

Object.prototype.bar = 1;

var foo = {moo: 2};

for(var i in foo) {

console.log(i); // 输出两个属性：bar 和 moo

}

我们没办法改变for in语句的行为，所以想过滤结果就只能使用hasOwnProperty 方法，代码如下：

// foo 变量是上例中的

for(var i in foo) {

if (foo.hasOwnProperty(i)) {

console.log(i);

}

}

这个版本的代码是唯一正确的写法。由于我们使用了 hasOwnProperty，所以这次只输出 moo。如果不使用 hasOwnProperty，则这段代码在原生对象原型（比如 Object.prototype）被扩展时可能会出错。

总结：推荐使用 hasOwnProperty，不要对代码运行的环境做任何假设，不要假设原生对象是否已经被扩展了。