JS面向对象编程笔记整理（一）

作者：flhuoshan
原文链接：https://ld246.com/article/1440063515896
来源网站：链滴

一、最简单的对象，创建时进行初始化。

<pre class="prettyprint lang-js">var human={
    name:"张三",
    sex:"男"
}
console.log(human.name+human.sex);</pre>

  或者先创建一个空对象，再对对象进行填充:

<pre class="prettyprint lang-js">var human1 = {};
human1.name="张三";
human1.sex="男";
var human2 ={};
human2.name="李四";
human2.sex="女";</pre>

  这样，human和human1,human2间没有任何关系。如果创建个对象，代码量是非常恐怖的。

  二、函数定义对象

<pre class="prettyprint lang-js">function human(name,sex){
  return {
    name:name,
    sex:sex
  }
}
var human1 = human("张三","男");
var human2 = human("李四","女");</pre>

  此方法，代码量减少了很多，但是human1和human2之间找不到联系。

<pre class="prettyprint lang-js">console.log(human1 instanceof human) //false
console.log(human2 instanceof human) //false</pre>

  三、进一步改进，使用构造函数来定义对象(此处我使用大写的Human来区别于普通函数human):

<pre class="prettyprint lang-js">function Human(name,sex){
  this.name = name;
  this.sex = sex;
}</pre>

构造函数其实就是一个普通函数，但是内部使用了</pan>this变量。对构造函数使用new运算符，就能生成实例，并且this变量会绑定在实例对象上。

<pre class="prettyprint lang-js">var human1 = new Human("张三","男");
var human2 = new Human("李四","女");</pre>

这样构造出的对象，都具有constructor属性：

<pre class="prettyprint lang-js">console.log(human1.constuctor == Human);//true
console.log(human2.constuctor == Human);//true</pre>

且可以看出，human1和human2都是Human的实例对象

<pre class="prettyprint lang-js">console.log(human1 instanceof Human) //true
console.log(human2 instanceof Human) //true</pre>

四、上一步使用构造函数来构造对象，已经符合了面向对象的基本要求，但还是存在一个问题，看：

<pre class="prettyprint lang-js">function Human(name,sex){
this.name = name;
this.sex = sex;
this.type ="人类";
this.skill = function(){
console.log("说话");
};
}</pre>

this.type和this.skill属于Human的共性，使用new的方法来构造对象的时候，每产生一个实对象，浏览器都会新开辟两块内存存储type和skill属性。

<pre class="prettyprint lang-js">console.log(human1 .type == human2 .type); //false</pre>

五、是否可以改进？Javascript规定，每一个构造函数都有个prototype属性，指向另一个对象。这个对象的所有属性和方法，都会被构造函数的实例继承。</san>这意味着，我们可以把那些不变的属性和方法，直接定义在prootype对象上。

```
    </div>
</div>
<p style="font-family:微软雅黑;font-size:14px;background-color:#FFFFFF;">
    <br />
</p>
<pre class="prettyprint lang-js">function Human(name,sex){
    this.name = name;
    this.sex = sex;
}
Human.prototype.type ="人类";
Human.prototype.skill = function(){ <span></span>console.log<span></span>("说话");
}</pre>
<p>
    <br />
</p>
<p style="font-family:微软雅黑;font-size:14px;background-color:#FFFFFF;">
    注意:prototype的处理放在构造函数之外进行。
</p>
<p style="font-family:微软雅黑;font-size:14px;background-color:#FFFFFF;">
    <br />
</p>
<pre class="prettyprint lang-js">var human1= new Human("张三","男");
var human2= new Human("李四","女");
console.log(human1.type); // 人类
human1.skill(); // 说话
console.log(human1 .type == human2 .type ); //true</pre>
<p>
    <br />
</p>
<p style="font-family:微软雅黑;font-size:14px;background-color:#FFFFFF;">
    <span style="background-color:inherit;">此时human1.type和human2.type指向同一块内存
</span>
</p>
<p style="font-family:微软雅黑;font-size:14px;background-color:#FFFFFF;">
    <span style="background-color:inherit;"><br />
</span>
</p>
<p style="font-family:微软雅黑;font-size:14px;background-color:#FFFFFF;">
    六、Prototype模式的验证方法
</p>
<p style="font-family:微软雅黑;font-size:14px;background-color:#FFFFFF;">
    为了配合prototype属性，Javascript定义了一些辅助方法，帮助我们使用它。，
</p>
<p style="font-family:微软雅黑;font-size:14px;background-color:#FFFFFF;">
    6.1 isPrototypeOf()
</p>
<p style="font-family:微软雅黑;font-size:14px;background-color:#FFFFFF;">
    这个方法用来判断，某个proptotype对象和某个实例之间的关系。
</p>
<div style="font-family:微软雅黑;font-size:14px;background-color:#FFFFFF;">
    <p style="background-color:inherit;">
        <br />
    </p>
<pre class="prettyprint lang-js">console.log(Human.prototype.isPrototypeOf(human1)); //tru
```

```
console.log(Human.prototype.isPrototypeOf(human2)); //true</pre>
    <p>
        <br />
    </p>
</div>
<p style="font-family:微软雅黑;font-size:14px;background-color:#FFFFFF;">
    6.2 hasOwnProperty()
</p>
<p style="font-family:微软雅黑;font-size:14px;background-color:#FFFFFF;">
    每个实例对象都有一个hasOwnProperty()方法，用来判断某一个属性到底是本地属性，还是继承prototype对象的属性。
</p>
<div style="font-family:微软雅黑;font-size:14px;background-color:#FFFFFF;">
    <p style="background-color:inherit;">
        <br />
    </p>
<pre class="prettyprint lang-js">console.log(Human.hasOwnProperty("name")); // true
console.log(Human.hasOwnProperty("type")); // false</pre>
    <p>
        <br />
    </p>
</div>
<p style="font-family:微软雅黑;font-size:14px;background-color:#FFFFFF;">
    6.3 in运算符
</p>
<p style="font-family:微软雅黑;font-size:14px;background-color:#FFFFFF;">
    in运算符可以用来判断，某个实例是否含有某个属性，不管是不是本地属性。
</p>
<div style="font-family:微软雅黑;font-size:14px;background-color:#FFFFFF;">
    <p style="background-color:inherit;">
        <br />
    </p>
<pre class="prettyprint lang-js">console.log("name" in human1); // true
console.log("type" in human1); // true</pre>
    <p>
        <br />
    </p>
</div>
<p style="font-family:微软雅黑;font-size:14px;background-color:#FFFFFF;">
    in运算符还可以用来遍历某个对象的所有属性。
</p>
<p>
    <span></span>
</p>
<p style="background-color:inherit;">
    <br />
</p>
<pre class="prettyprint lang-js">for(var prop in human1) { alert("human1["+prop+"]="+hum
n1[prop]); }</pre>
<p>
    <br />
</p>
<p style="background-color:inherit;">
```

```
   本文整理自：http://www.ruanyifeng.com/blog/（阮一峰的网络日志）
</p>
<span></span>
<p>
   <br />
</p>
<p style="font-family:微软雅黑;font-size:14px;background-color:#FFFFFF;">
   <br />
</p>
<p style="font-family:微软雅黑;font-size:14px;background-color:#FFFFFF;">
   <br />
</p>
<p style="font-family:微软雅黑;font-size:14px;background-color:#FFFFFF;">
   <br />
</p>
<div style="font-family:微软雅黑;font-size:14px;background-color:#FFFFFF;">
   <div style="background-color:inherit;">
      <div style="background-color:inherit;">
         <br />
      </div>
<br />
   </div>
   <div style="background-color:inherit;">
      <br />
   </div>
<br />
</div>
<div style="font-family:微软雅黑;font-size:14px;background-color:#FFFFFF;">
   <br />
</div>
<div style="font-family:微软雅黑;font-size:14px;background-color:#FFFFFF;">
   <br />
</div>
```