

实例对象和 new 命令

1. 对象是什么

面向对象编程（Object Oriented Programming，缩写为 OOP）将真实世界各种复杂的关系，抽象为一个个对象。

（1）对象是单个实物的抽象。

一本书、一辆汽车、一个人都可以是对象，一个数据库、一张网页、一个远程服务器连接也可以是对象。当实物被抽象成对象，实物之间的关系就变成了对象之间的关系，从而就可以模拟现实情况，针对对象进行编程。

（2）对象是一个容器，封装了属性（property）和方法（method）。

属性是对象的状态，方法是对象的行为（完成某种任务）。我们可以把动物抽象为 `animal` 对象，使用“属性”记录具体是哪一种动物，使用“方法”表示动物的某种行为（奔跑、捕猎、休息等等）。

2. 构造函数

面向对象编程的第一步，就是要生成对象。典型的面向对象编程语言（比如 C++ 和 Java），都有“类”（class）这个概念。所谓“类”就是对象的模板，对象就是“类”的实例。但是，JavaScript 语言的对象体系，不是基于“类”的，而是基于构造函数（constructor）和原型链（prototype）。

JavaScript 语言使用构造函数（constructor）作为对象的模板。**所谓“构造函数”，就是专门用来生成实例对象的函数。**它就是对象的模板，描述实例对象的基本结构。一个构造函数，可以生成多个实例对象，这些实例对象都有相同的结构。构造函数就是一个普通的函数，但具有自己的特征和用法。**为了与普通函数区别，构造函数名字的第一个字母通常大写。**

```
let Vehicle = function () {  
  this.price = 1000;  
};
```

构造函数的特点有两个：

- 生成对象的时候，必须使用 `new` 命令。
- 函数体内部使用了 `this` 关键字，代表了所要生成的对象实例。

3. new 命令

3.1. 基本用法

`new` 命令的作用，就是执行构造函数，返回一个实例对象。

```
let Vehicle = function () {  
  this.price = 1000;  
};
```

```
let v = new Vehicle();  
v.price; // 1000
```

上例通过 `new` 命令，让构造函数 `Vehicle` 生成一个实例对象，保存在变量 `v` 中。这个新生成的实例对象，从构造函数 `Vehicle` 得到了 `price` 属性。`new` 命令执行时，构造函数内部的 `this`，就代表了新生成的实例对象，`this.price` 表示实例对象有一个 `price` 属性，值是 1000。

使用 `new` 命令时，根据需要，构造函数也可以接受参数。

```
let Vehicle = function (p) {  
  this.price = p;  
};  
let v = new Vehicle(500);
```

如果忘了使用 `new` 命令，直接调用构造函数，构造函数就变成了普通函数，并不会生成实例对象。`this` 这时代表全局对象，将造成一些意想不到的结果。

```
let Vehicle = function () {  
  this.price = 1000;  
};  
let v = Vehicle();  
v; // undefined  
price; // 1000
```

上例中，调用 `Vehicle` 构造函数时，忘了加上 `new` 命令。结果，变量 `v` 变成了 `undefined`，而 `price` 属性变成了全局变量。

构造函数内部使用严格模式，即第一行加上 `use strict`。这样的话，一旦忘了使用 `new` 命令，直接调用构造函数就会报错。

```
function Fubar(foo, bar) {  
  'use strict';  
  this._foo = foo;  
  this._bar = bar;  
}  
Fubar(); // TypeError: Cannot set property '_foo' of undefined
```

由于严格模式中，函数内部的 `this` 不能指向全局对象，默认等于 `undefined`，导致不加 `new` 调用会报错（JavaScript 不允许对 `undefined` 添加属性）。

另一个解决办法，构造函数内部判断是否使用 `new` 命令，如果发现没有使用，则直接返回一个实例对象。

```
function Fubar(foo, bar) {  
  if (!(this instanceof Fubar)) {
```

```
    return new Fubar(foo, bar);
  }
  this._foo = foo;
  this._bar = bar;
}
Fubar(1, 2)._foo;           // 1
(new Fubar(1, 2))._foo;    // 1
```

上例中的构造函数，不管加不加 `new` 命令，都会得到同样的结果。

3.2. new 命令的原理

使用 `new` 命令时，它后面的函数依次执行下面的步骤。

1. 创建一个空对象，作为将要返回的对象实例。
2. 将这个空对象的原型，指向构造函数的 `prototype` 属性。
3. 将这个空对象赋值给函数内部的 `this` 关键字。
4. 开始执行构造函数内部的代码。

也就是说，构造函数内部，`this` 指的是一个新生成的空对象，所有针对 `this` 的操作，都会发生在这个空对象上。构造函数之所以叫“构造函数”，就是说这个函数的目的，就是操作一个空对象（即 `this` 对象），将其“构造”为需要的样子。

如果构造函数内部有 `return` 语句，而且 `return` 后面跟着一个对象，`new` 命令会返回 `return` 语句指定的对象；否则，就会不管 `return` 语句，返回 `this` 对象。

```
let Vehicle = function () {
  this.price = 1000;
  return 1000;
};
(new Vehicle()) === 1000; // false
```

上例中，构造函数 `Vehicle` 的 `return` 语句返回一个数值。这时，`new` 命令就会忽略这个 `return` 语句，返回“构造”后的 `this` 对象。

但是，如果 `return` 语句返回的是一个跟 `this` 无关的新对象，`new` 命令会返回这个新对象，而不是 `this` 对象。

```
let Vehicle = function () {
  this.price = 1000;
  return { price: 2000 };
};

(new Vehicle()).price; // 2000
```

上例中，构造函数 `Vehicle` 的 `return` 语句，返回的是一个新对象。`new` 命令会返回这个对象，而不是 `this` 对象。

另一方面，如果对普通函数（内部没有 `this` 关键字的函数）使用 `new` 命令，则会返回一个空对象。

```
function getMessage() {  
  return 'this is a message';  
}  
let msg = new getMessage();  
msg // {}  
typeof msg // "object"
```

这是因为 `new` 命令总是返回一个对象，要么是实例对象，要么是 `return` 语句指定的对象。本例中，`return` 语句返回的是字符串，所以 `new` 命令就忽略了该语句。

`new` 命令简化的内部流程，可以用下面的代码表示。

```
function _new(constructor, params) { // constructor 构造函数, params 构造函数参数  
  // 将 arguments 对象转为数组  
  let args = [].slice.call(arguments);  
  // 取出构造函数  
  let constructor = args.shift();  
  // 创建一个空对象，继承构造函数的 prototype 属性  
  let context = Object.create(constructor.prototype);  
  // 执行构造函数  
  let result = constructor.apply(context, args);  
  // 如果返回结果是对象，就直接返回，否则返回 context 对象  
  return (typeof result === 'object' && result !== null) ? result : context;  
}  
  
// 实例  
let actor = _new(Person, '张三', 28);
```

3.3. new.target

函数内部可以使用 `new.target` 属性。如果当前函数是 `new` 命令调用，`new.target` 指向当前函数，否则为 `undefined`。

```
function f() {  
  console.log(new.target === f);  
}  
f(); // false  
new f(); // true
```

使用这个属性，可以判断函数调用的时候，是否使用 `new` 命令。

```
function f() {  
  if (!new.target) {  
    throw new Error('请使用 new 命令调用!');  
  }  
}
```

```
}  
// ...  
}  
f() // Uncaught Error: 请使用 new 命令调用!
```

上例中，构造函数 `f` 调用时，没有使用 `new` 命令，就抛出一个错误。

4. Object.create() 创建实例对象

构造函数作为模板，可以生成实例对象。但是，有时拿不到构造函数，只能拿到一个现有的对象。我们希望以这个现有的对象作为模板，生成新的实例对象，这时就 **可以使用 `Object.create()` 方法将参数作为模版生成实例对象。**

```
let person1 = {  
  name: '张三',  
  age: 38,  
  greeting: function() {  
    console.log('Hi! I\'m ' + this.name + '.');  
  }  
};  
let person2 = Object.create(person1);  
person2.name; // 张三  
person2.greeting(); // Hi! I'm 张三.  
person1.constructor === person2.constructor; // true
```

上例中，对象 `person1` 是 `person2` 的模板，后者继承了前者的属性和方法。且两者拥有相同的构造函数。