深度剖析new并手动封装（进阶）

jCodeLife
0.395 2020.09.15 12:40:17 字数 1,536 阅读 69

编辑文章

昨晚做了一个梦，梦见有道关于`new`的笔试题，梦中的情景让我很困惑！
所以今天再来彻底的理解`new`关键字到底做了什么，然后手动封装其功能

首先，我们应该知道`new`的基本作用是用于构造函数生产对象

```
function Person(name, age) {
    this.name = name;
    this.age = age;
}
let p1 = new Person('alice', 12);
console.log(p1)//Person {name: "alice", age: 12}
```

上述例子中，通过`new`构造函数`Person()`产生`p1`对象，生产的对象可以继承`Person`原型上的属性和方法

```
Person.prototype.show = function (name, age) {
    console.log(`我叫${this.name}，我今年${this.age}岁了！`)
}
p1.show()
```

上面代码执行结果是：`我叫alice，我今年12岁了！`
补充说明一下什么是原型
说白了原型（`prototype`）其实就是`function`对象的一个属性，它定义了构造函数构造出的对象的共有祖先，凡是该构造函数构造出的对象，都能继承原型上的属性和方法

接着来看`new`关键字的内部原理
我们知道，在`JavaScript`中，函数执行前面加`new`和不加`new`，执行的结果会变得不一样；那`new`到底做了什么？
我们先来回顾一下，我之前提过通过new构造函数构造出对象内部原理

1. 隐式创建this对象；在函数体最前面隐式的加上this = {}
2. 执行this.xxx = xxx
3. 隐式的返回this对象

昨晚那个梦告诉我这不太对，有几个问题
第一，生成的对象为什么会继承自构造函数的原型？如果按照上述步骤构建对象，构建过程跟原型并无关联；
第二，执行的只是`this.xxx=xxx`这种格式的代码吗，那 其他语句执行吗？
第三，`new`最终的结果都是返回`this`对象？如果构造函数有毛病，自己显示的`return`了呢，那结果是返回隐式的`this`对象和是返回显示`return`的东西呢？

针对上诉三个疑问，我们一起来看看
第一个问题，生成的对象为什么会继承自构造函数的原型？
其实最开始构建对象的时候，隐式创建的`this`对象并非完全是空对象`{}`，这个对象里面有个属性`__proto__`指向其构造函数的`prototype`，这就是对象能继承原型上的属性和方法的真实原因

```
this.__proto__ : Student.prototype;
```

推荐阅读
5个技巧让你更好的编写JavaScript(ES6) 中条件语句
阅读 92
Es6基础语法
阅读 1,172
js 判断字符串中是否包含某个字符串
阅读 370
vue项目实现登录携带token
阅读 347
JavaScript基础篇（三）作用域和闭包
阅读 57

```
function Person(name, age) {
    console.log(this)//{name: "alice", age: 12}
    this.name = name;
    this.age = age;
    alert(1)//弹出1
    console.log(2);//控制台输出2
    function inner() {
        if (1) {
            console.log('inner')//inner函数执行打印出：inner
        }
    }
    inner();
}
let p1 = new Person('alice', 12);
```

上面代码中，我们发现并非指执行this.xxx=xxx这种格式的代码，而是跟正常函数执行一致，会执行函数中的每条代码

第三个问题，如果构造函数有显示return，结果返回的结果是显示的return还是隐式的this对象？

我来显示的返回一些东西

(1) 显示返回的类型为基础类型中原始值时，即 `number、string、boolean、null、undefined` 中的一种

```
function Person(name, age) {
    this.name = name;
    this.age = age;
    return 1;
    //return 'abc';
    //return true;
    //return null;
    //return undefined;
}
let p1 = new Person('alice', 12);
console.log(p1);
```

显示返回原始值数据时，最终返回都原本的 `this` 对象

图片获取失败，请点击重试

显示返回原始值数据类型，返回结果都是隐式的this对象

(2) 显示返回数组、对象、function

```
function Person(name, age) {
    this.name = name;
    this.age = age;
    return [];
}
let p1 = new Person('alice', 12);
console.log(p1);//[]
```

若显示返回数组，返回的就是该数组

```
function Person(name, age) {
    this.name = name;
    this.age = age;
    return {};
}
let p1 = new Person('alice', 12);
console.log(p1);//{}
```

推荐阅读

5个技巧让你更好的编写JavaScript(ES6) 中条件语句
阅读 92

Es6基础语法
阅读 1,172

js 判断字符串中是否包含某个字符串
阅读 370

vue项目实现登录携带token
阅读 347

JavaScript基础篇（三）作用域和闭包
阅读 57

```
    this.age = age;
    function fn(){
    }
    return fn;
}
let p1 = new Person('alice', 12);
console.log(p1);//fn(){}
```

若显示返回一个函数，最终返回的就是函数

以上几个例子，可以得出跟我们最开始讲得不一致的地方，就是：
如果构造函数本身有显示返回，显示返回的情况分两种
第一，如果显示返回的数据类型是原始类型（`number、string、boolean、undefined、null`），那构造对象最终返回我们构建的`this`对象；
第二，如果显示返回的值如果是引用值（比如`array、object、function`），那么`new`得出的结果不再是`this`对象，而是显示返回的东西；需特别注意这点

所以，重新总结梳理一下new的内部原理：

`1. 在函数体最前头隐式创建this对象；相当于 let this = {}`

`2. 给this对象身上添加__proto__属性指向构造函数的原型prototype`

`3. 执行构造函数中的每一条语句（注意构造函数执行时，里面this指向绑定为新对象哦）`

`4. 判断构造函数本身是否有显示的return数据：`

`如果没有显示return数据，那么返回隐式的this对象`

`如果有显示return数据，判断return的数据的是原始值还是引用值，如果是原始值，返回结果还是隐式的this对象；如果是引用值，返回的就是显示返回的东西`

经过上述总结梳理，可以模拟实现`new`的功能如下：

```
Function.prototype._new = function (...arg) {
    let _this = {};//第一步，创建this对象
    _this.__proto__ = this.prototype;//第二步，添加属性指向构造函数原型
    let _constructorRes = this.apply(_this, arg);//第三步，让构造函数执行，并绑定构造函数中的this指
    //第四步，判断返回结果是什么类型
    if (typeof _constructorRes === 'number'
        || typeof _constructorRes === 'string'
        || typeof _constructorRes === 'boolean'
        || _constructorRes === null
        || _constructorRes === undefined) {//判断为原始值，返回_this对象
        return _this;
    } else {
        return _constructorRes;
    }
}
```

测试一下

```
function Person(name, age) {
    this.name = name;
    this.age = age;
}
let p1 = Person._new('alice', 12);
console.log(p1);
```

以上测试代码结果打印`{name: "alice", age: 12}`，即`let p1 = Person._new('alice', 12)`和`let p1 = new Person('alice', 12)`执行结果一致。
经过多次测试以上模拟实现`new`没有问题！

最后来解决我梦见的那道题目（大概）

```
function Foo() {
    getName = function () {
```

推荐阅读

5个技巧让你更好的编写JavaScript(ES6) 中条件语句
阅读 92

Es6基础语法
阅读 1,172

js 判断字符串中是否包含某个字符串
阅读 370

vue项目实现登录携带token
阅读 347

JavaScript基础篇（三）作用域和闭包
阅读 57

```
}
Foo.prototype.getName = function () {
    alert(3)
}
var getName = function () {
    alert(4)
}
function getName() {
    alert(5)
}
Foo.getName();
getName()
Foo().getName()
getName()
new Foo.getName()
new Foo().getName()
new new Foo().getName()
```

前面四个问题，都比较容易，分别得出：`2,4,1,1`
主要来看后面三个

1. new Foo.getName()
   这里将Foo上的静态方法getName函数作为构造函数执行，所以得出结果为：`2`
   注意：这里是先Foo.getName,然后再new
2. new Foo().getName()
   这里是先计算new Foo()，即调用的是Foo.prototype上的getName()，即得出结果为：`3`
3. new new Foo().getName()
   以上代码相当于new((new Foo()).getName)();答案输出：`3`

这题除了明白new的内部原理之外，得注意js运算符的优先级；
以 new new Foo().getName()为例，小结一下上述new的优先级顺序：

1. 先`new Foo()`
2. 然后new Foo()`.getName`
3. 最后`new` new Foo().getName`()`

以上就是关于new的全部内容！

jCodeLife 书山有路勤为径，学海无涯苦作舟
总资产73 (约5.46元) 共写了13.6W字 获得340个赞 共29个粉丝

被以下专题收入，发现更多相似内容

投稿管理

+收入我的专题 前段开发那些事儿

推荐阅读

5个技巧让你更好的编写JavaScript(ES6) 中条件语句
阅读 92

Es6基础语法
阅读 1,172

js 判断字符串中是否包含某个字符串
阅读 370

vue项目实现登录携带token
阅读 347

JavaScript基础篇（三）作用域和闭包
阅读 57