2020面试收获 - js原型及原型链

一、前言

2020年是特殊的一年，由于疫情原因，大家都窝在家办公。而我则怀着梦想，从天津来到了北京，开启了人生的第一次跳槽。

在面试过程中，频频被原型相关知识问住，每次回答都支支吾吾。后来有家非常心仪的公司，在二面时，果不其然，又问原型了！

我痛下决心用了两天时间钻研了下原型，弄明白后发现世界都明亮了，原来这么简单 ~

有些理解还比较浅薄，随着时间的推移和理解的深入，以后还会补充。如果大家发现我理解的有问题，欢迎大家在评论中指正。话不多说，切入正题。

二、构造函数

讲原型则离不开构造函数，让我们先来认识下构造函数。

2.1 构造函数分为 实例成员 和 静态成员

让我们先来看看他们分别是什么样子的。

实例成员： 实例成员就是在构造函数内部，通过this添加的成员。实例成员只能通过实例化的对象来访问。

静态成员： 在构造函数本身上添加的成员，只能通过构造函数来访问

```ini 复制代码
function Star(name,age) {
    //实例成员
    this.name = name;
    this.age = age;
}
//静态成员
Star.sex = '女';
```

```
let stars = new Star('小红',18);
console.log(stars);      // Star {name: "小红", age: 18}
console.log(stars.sex);  // undefined      实例无法访问sex属性

console.log(Star.name); //Star      通过构造函数无法直接访问实例成员
console.log(Star.sex);  //女        通过构造函数可直接访问静态成员
```

2.2 通过构造函数创建对象

该过程也称作实例化

2.2.1 如何通过构造函数创建一个对象?

ini 复制代码

```
function Father(name) {
    this.name = name;
}
let son = new Father('Lisa');
console.log(son); //Father {name: "Lisa"}
```

此时，son就是一个新对象。

2.2.2 new一个新对象的过程，发生了什么?

(1) 创建一个空对象 son `{}`
(2) 为 son 准备原型链连接 `son.__proto__ = Father.prototype`
(3) 重新绑定this，使构造函数的this指向新对象 `Father.call(this)`
(4) 为新对象属性赋值 `son.name`
(5) 返回this `return this` ，此时的新对象就拥有了构造函数的方法和属性了

2.2.3 每个实例的方法是共享的吗?

这要看我们如何定义该方法了，分为两种情况。

方法1：在构造函数上直接定义方法（不共享）

javascript 复制代码

```
function Star() {
    this.sing = function () {
        console.log('我爱唱歌');
```

```
    }
}
let stu1 = new Star();
let stu2 = new Star();
stu1.sing();//我爱唱歌
stu2.sing();//我爱唱歌
console.log(stu1.sing === stu2.sing);//false
```

很明显，stu1 和 stu2 指向的不是一个地方。 所以 在构造函数上通过this来添加方法的方式来生成实例，每次生成实例，都是新开辟一个内存空间存方法。这样会导致内存的极大浪费，从而影响性能。

方法2：通过原型添加方法（共享）

构造函数通过原型分配的函数，是所有对象共享的。

ini 复制代码

```
function Star(name) {
    this.name = name;
}
Star.prototype.sing = function () {
    console.log('我爱唱歌', this.name);
};
let stu1 = new Star('小红');
let stu2 = new Star('小蓝');
stu1.sing();//我爱唱歌 小红
stu2.sing();//我爱唱歌 小蓝
console.log(stu1.sing === stu2.sing);//true
```

2.2.4 实例的属性为基本类型是，它们是共享的吗?

属性存储的是如果存储的是基本类型，不存在共享问题，是否相同要看值内容。

ini 复制代码

```
let stu1 = new Star('小红');
let stu2 = new Star('小红');
console.log(stu1.name === stu2.name);//true

let stu1 = new Star('小红');
let stu2 = new Star('小蓝');
console.log(stu1.name === stu2.name);//false
```

2.2.5 定义构造函数的规则

公共属性定义到构造函数里面，公共方法我们放到原型对象身上。

三、原型

前面我们在 实例化 和 实例共享方法 时，都提到了原型。那么现在聊聊这个神秘的原型到底是什么?

3.1 什么是原型?

Father.prototype 就是原型，它是一个对象，我们也称它为原型对象。

3.2 原型的作用是什么?

原型的作用，就是共享方法。
我们通过 `Father.prototype.method` 可以共享方法，不会反应开辟空间存储方法。

3.3 原型中this的指向是什么?

原型中this的指向是实例。

四、原型链

4.1 什么是原型链?

原型与原型层层相链接的过程即为原型链。

4.2 原型链应用

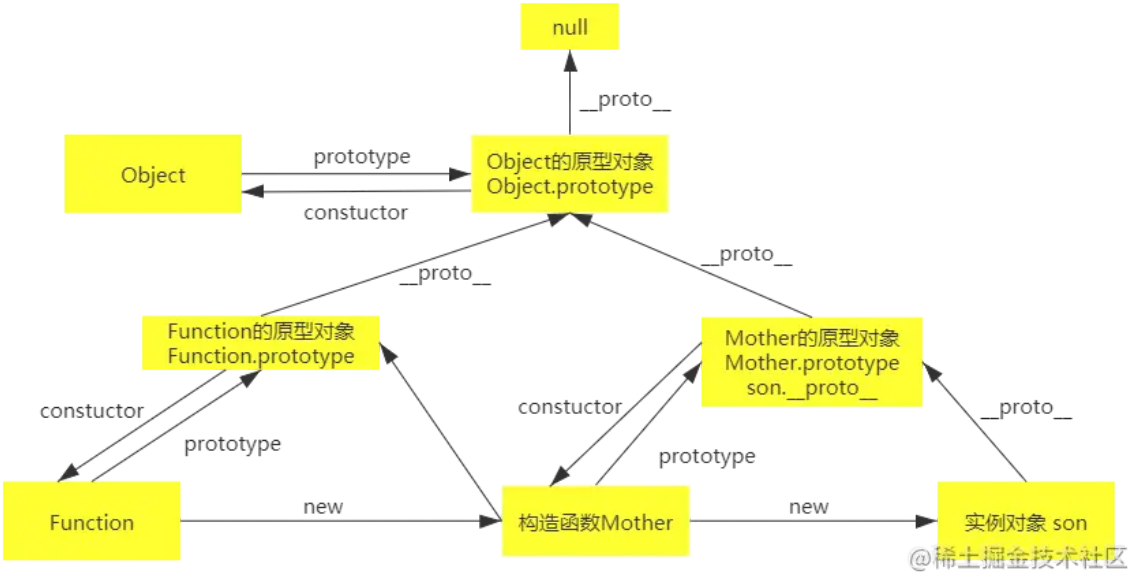
对象可以使用构造函数prototype原型对象的属性和方法，就是因为对象有__proto__原型的存在
每个对象都有__proto__原型的存在

ini 复制代码

```
function Star(name,age) {
    this.name = name;
    this.age = age;
```

```
}
Star.prototype.dance = function(){
    console.log('我在跳舞',this.name);
};
let obj = new Star('张萌',18);
console.log(obj.__proto__ === Star.prototype);//true
```

4.3 原型链图



4.4 原型查找方式

例如：查找obj的dance方法

javascript 复制代码

```
function Star(name) {
    this.name = name;

    (1)首先看obj对象身上是否有dance方法，如果有，则执行对象身上的方法
    this.dance = function () {
        console.log(this.name + '1');
    }
}

(2)如果没有dance方法，就去构造函数原型对象prototype身上去查找dance这个方法。
Star.prototype.dance = function () {
    console.log(this.name + '2');
};

(3)如果再没有dance方法，就去Object原型对象prototype身上去查找dance这个方法。
Object.prototype.dance = function () {
```

```
    console.log(this.name + '3');
};
(4)如果再没有，则会报错。
let obj = new Star('小红');
obj.dance();
```

(1)首先看obj对象身上是否有dance方法，如果有，则执行对象身上的方法。

(2)如果没有dance方法，就去构造函数原型对象prototype身上去查找dance这个方法。

(3)如果再没有dance方法，就去Object原型对象prototype身上去查找dance这个方法。

(4)如果再没有，则会报错。

4.5 原型的构造器

原型的构造器指向构造函数。

ini 复制代码

```ini
function Star(name) {
    this.name = name;
}
let obj = new Star('小红');
console.log(Star.prototype.constructor === Star);//true
console.log(obj.__proto__.constructor === Star); //true
```

4.5.1 在原型上添加方法需要注意的地方

方法1： 构造函数.prototype.方法 在原型对象上直接添加方法，此时的原型对象是有 constructor 构造器的，构造器指向构造函数本身

javascript 复制代码

```javascript
function Star(name) {
    this.name = name;
}
Star.prototype.dance = function () {
    console.log(this.name);
};
let obj = new Star('小红');
console.log(obj.__proto__);  //{dance: f, constructor: f}
console.log(obj.__proto__.constructor);  // Star
```

方法2： `Star.prototype = {}` 给原型重新赋值，此时会丢失构造器，我们需要手动定义构造器，指回构造函数本身

```ini 复制代码
function Star(name) {
    this.name = name;
}
Star.prototype = {
    dance: function () {
        console.log(this.name);
    }
};
let obj = new Star('小红');
console.log(obj.__proto__); //{dance: f}
console.log(obj.__proto__.constructor); //  f Object() { [native code] }
Star.prototype.constructor = Star;
```

4.5.2 一般不允许直接改变原型 `prototype` 指向

改变原型指向，会使原生的方法都没了，所以Array、String这些内置的方法是不允许改变原型指向的。如果改变了，就会报错。

```ini 复制代码
Array.prototype.getSum = function (arr) {
    let sum = 0;
    for (let i = 0; i < this.length; ++i) {
        sum += this[i];
    }
    return sum;
};
let arr = [1, 2, 3, 4, 5, 6, 7, 8, 9];
console.log(arr.getSum());//45
```

如果改变prototype指向，会报错!

```ini 复制代码
Array.prototype = {
    getSum: function (arr) {
        let sum = 0;
        for (let i = 0; i < this.length; ++i) {
            sum += this[i];
        }
        return sum;
    }
};
let arr = [1, 2, 3, 4, 5, 6, 7, 8, 9];
```

```
console.log(arr.getSum());//45
```

五、继承 - ES5方法

ES6之前并没有给我们提供extends继承，我们可以通过构造函数+原型对象模拟实现继承。

继承属性，利用call改变this指向。但该方法只可以继承属性，实例不可以使用父类的方法。

ini 复制代码

```
function Father(name) {
    this.name = name;
}
Father.prototype.dance = function () {
  console.log('I am dancing');
};
function Son(name, age) {
    Father.call(this, name);
    this.age = age;
}
let son = new Son('小红', 100);
son.dance();   //报错
```

如何继承父类的方法呢?

解决方法1：利用 `Son.prototype = Father.prototype` 改变原型指向，但此时我们给子类增加原型方法，同样会影响到父类。

javascript 复制代码

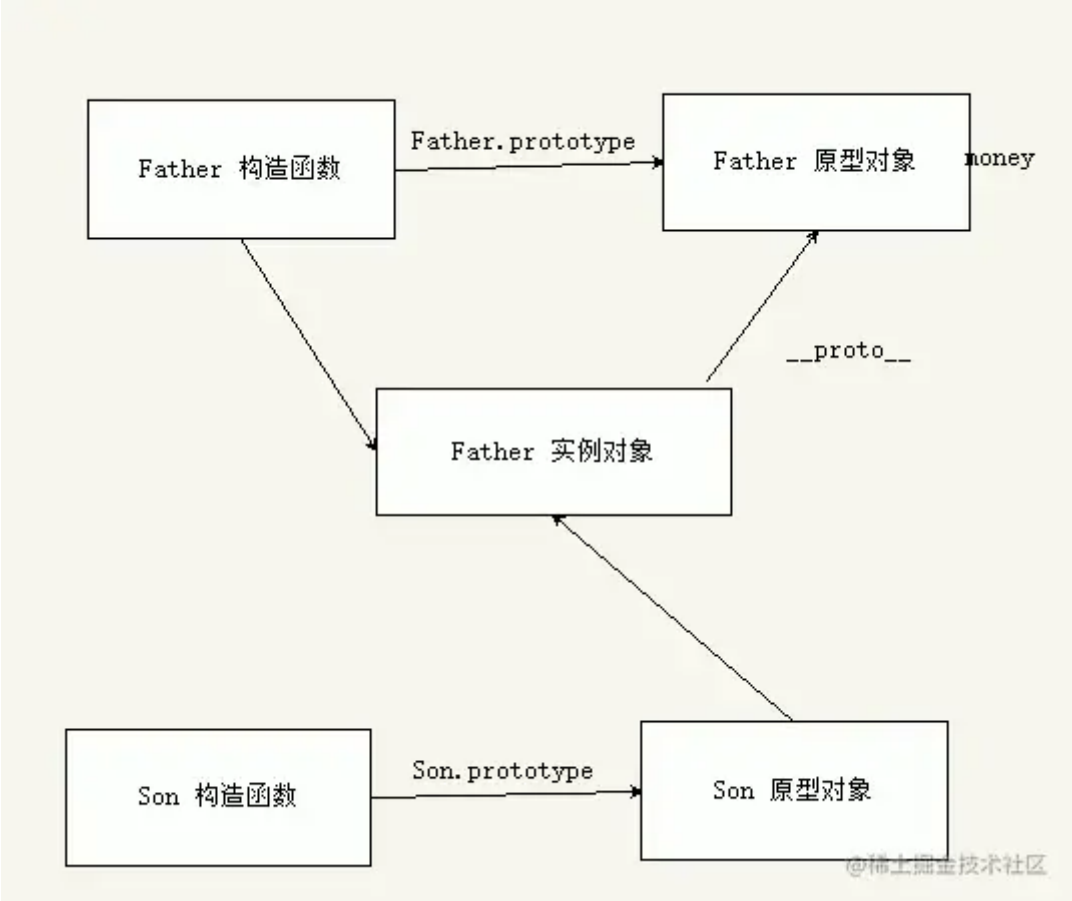
```
function Father(name) {
    this.name = name;
}
Father.prototype.dance = function () {
    console.log('I am dancing');
};
function Son(name, age) {
    Father.call(this, name);
    this.age = age;
}
Son.prototype = Father.prototype;
//为子类添加方法
Son.prototype.sing = function () {
```

```
    console.log('I am singing');
};
let son = new Son('小红', 100);
//此时父类也被影响了
console.log(Father.prototype) //{dance: f, sing: f, constructor: f}
```

解决方法2：子类的原型指向父类的实例，这样就可以顺着原型链共享父类的方法了。并且为子类添加原型方法的时候，不会影响父类。

javascript 复制代码

```
function Father(name) {
    this.name = name;
}
Father.prototype.dance = function () {
    console.log('I am dancing');
};
function Son(name, age) {
    Father.call(this, name);
    this.age = age;
}
Son.prototype = new Father();
Son.prototype.sing = function () {
    console.log('I am singing');
};
let son = new Son('小红', 100);
console.log(Father.prototype) //{dance: f, constructor: f}
```



七、类

什么是类?

类的本质还是一个函数，类就是构造函数的另一种写法。

javascript 复制代码

```
function Star(){}
console.log(typeof Star); //function

class Star {}
console.log(typeof Star); //function
```

ES6中类没有变量提升

通过构造函数创建实例，是可以变量提升的。 es6中的类，必须先有类，才可以实例化。

类的所有方法都定义在类的prototype属性上面

让我们来测试一下。

```javascript 复制代码
class Father{
    constructor(name){
        this.name = name;
    }
    sing(){
        return this.name;
    }
}
let red = new Father('小红');
let green = new Father('小绿');
console.log(red.sing === green.sing); //true
```

向类中添加方法

通过Object.assign，在原型上追加方法。

```javascript 复制代码
class Father{
    constructor(name){
        this.name = name;
    }
    sing(){
        return this.name;
    }
}
//在原型上追加方法
Object.assign(Father.prototype,{
    dance(){
        return '我爱跳舞';
    }
});
let red = new Father('小红');
let green = new Father('小绿');
console.log(red.dance());//我爱跳舞
console.log(red.dance === green.dance); //true
```

constructor方法

constructor方法是类的默认方法，通过new命令生成对象实例时，自动调用该方法。一个类必须有constructor方法，如果没有显式定义，一个空的constructor方法会被默认添加。

八、继承 - ES6方法

javascript 复制代码

```
class Father {
    constructor(name){
        this.name = name;
    }
    dance(){
        return '我在跳舞';
    }
}
class Son extends Father{
    constructor(name,score){
        super(name);
        this.score = score;
    }
    sing(){
        return this.name +','+this.dance();
    }
}
let obj = new Son('小红',100);
```

九、类和构造函数的区别

(1) 类必须使用new调用，否则会报错。这是它跟普通构造函数的一个主要区别，后者不用new也可以执行。

(2) 类的所有实例共享一个原型对象。

(3) 类的内部，默认就是严格模式，所以不需要使用use strict指定运行模式。

十、总结

构造函数特点：

1.构造函数有原型对象prototype。

2.构造函数原型对象prototype里面有constructor，指向构造函数本身。

3.构造函数可以通过原型对象添加方法。

4.构造函数创建的实例对象有__proto__原型，指向构造函数的原型对象。

类：

1.class本质还是function

2.类的所有方法都定义在类的prototype属性上

3.类创建的实例，里面也有__proto__指向类的prototype原型对象

4.新的class写法，只是让对象原型的写法更加清晰，更像面向对象编程的语法而已。

5.ES6的类其实就是语法糖。

十一、什么是语法糖

什么是语法糖？加糖后的代码功能与加糖前保持一致，糖在不改变其所在位置的语法结构的前提下，实现了运行时的等价。

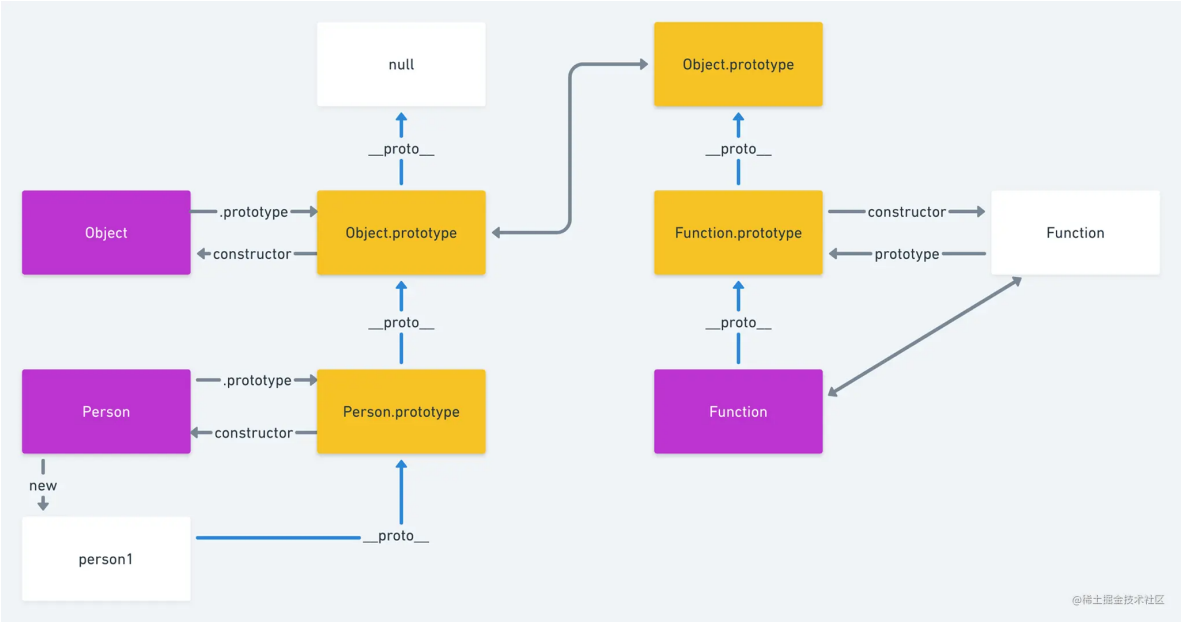
语法糖没有改变语言功能，但增加了程序员的可读性。

十二、面试题分享

面试题1

```javascript
Object.prototype.__proto__     //null
Function.prototype.__proto__  //Object.prototype
Object.__proto__              //Function.prototype
```

讲解： 这里涉及到Function的原型问题，附一张图，这图是一个面试官发给我的，我也不知道原作者在哪里~



面试题2

给大家分享那道我被卡住的面试题，希望大家在学习完知识后，可以回顾一下。

css 复制代码

```
按照如下要求实现Person 和 Student 对象
 a)Student 继承Person
 b)Person 包含一个实例变量 name， 包含一个方法 printName
 c)Student 包含一个实例变量 score， 包含一个实例方法printScore
 d)所有Person和Student对象之间共享一个方法
```

es6类写法

javascript 复制代码

```
class Person {
    constructor(name) {
        this.name = name;
    }
    printName() {
        console.log('This is printName');
    }
    commonMethods(){
        console.log('我是共享方法');
    }
}

class Student extends Person {
    constructor(name, score) {
        super(name);
        this.score = score;
    }
```

```
    printScore() {
        console.log('This is printScore');
    }
}

let stu = new Student('小红');
let person = new Person('小紫');
console.log(stu.commonMethods===person.commonMethods);//true
```

原生写法

ini 复制代码

```
function Person (name){
    this.name = name;
    this.printName=function() {
        console.log('This is printName');
    };
}
Person.prototype.commonMethods=function(){
    console.log('我是共享方法');
};

function Student(name, score) {
    Person.call(this,name);
    this.score = score;
    this.printScore=function() {
        console.log('This is printScore');
    }
}
Student.prototype = new Person();
let person = new Person('小紫',80);
let stu = new Student('小红',100);
console.log(stu.printName===person.printName);//false
console.log(stu.commonMethods===person.commonMethods);//true
```