JS原型及原型链详解

烈风逍遥
一个默默前进的前端开发
关注他

6 人赞同了该文章

前言

作为一个初入前端的菜鸟，每次说起JS的原型，原型链，继承机制之类的东西都很头疼。想深入学习前端的话这些基础是避免不了的，为此，只能下苦功夫将其了解吃透。

原型(prototype)的来源和定义

当我最初学习JS接触到原型的时候，我感觉它就像是一个凭空出现，难以理解的复杂概念。完全不像数组Array，对象Object 等等这个我们可以联想到生活中的具体示例的东西。比如说数组我们可以类比为一个购物车，里面的数据就是我们所购买的东西，鞋子，面包之类的。对象的话可以把一个人看成一个对象，姓名，性别，年龄等可以作为数据，对应的数据就作为属性的值。而原型感觉与之前我们所学的东西，完全割裂开了，不能找到与之对应或者类似的东西，很抽象，难以理解，只能硬着头皮去记录它的作用，当下可能记住了，可时间一长又忘记了。主要是我没有形成对它的映射，没有联想记忆。

原型的来源

在面向对象编程中，**继承**是非常实用也非常核心的功能，很多面向对象语言都支持两种继承：接口继承和实现继承。前者只继承方法签名，后者继承实现的方法。例如C++，Java语言中的类，**子类可以继承父类中的某些属性和方法**，这样可以实现数据的共享。但**JavaScript中只有对象没有类**(ES6中添加了类Class，之前是没有的)，为了解决共享数据的问题，JavaScript的开发者们提出了原型这一概念，来实现数据和方法的共享。

原型的定义

ECMAScript标准中原型的解释

在ECMAScript2019(262.ecma-international.org...)中是这样定义原型的

4.3.5prototype

> **object that provides shared properties for other objects** NOTE When a constructor creates an object, that object implicitly references the constructor's **prototype** property for the purpose of resolving property references. The constructor's **prototype** property can be referenced by the program expression **constructor.prototype**, and properties added to an object's prototype are shared, through inheritance, by all objects sharing

▲ 赞同 6 ▼

章

> the prototype. Alternatively, a new object may be created with an explicitly specified prototype by using the **Object.create** built-in function.

在ECMAScript2019规范中，原型(prototype)被定义为：**给其他对象提供共享属性的对象**

《 javascript高级程序设计 》中这样描述原型

> 每个**函数**都会创建一个**prototype属性**，这个属性是一个**对象**，包含应该由特定引用类型的实例共享的属性和方法。实际上，这个对象就是通过调用构造函数创建的对象的原型。使用原型对象的好处是，在它上面定义的属性和方法都可以被对象实例共享。原来在构造函数中直接赋给对象实例的值，可以直接赋值给它们的原型。

《 你不知道的javascript 》中对原型的描述

> javascript中的**对象**有一个特殊的 **[[Prototype]] 内置属性**，其实就是对其他对象的引用。几乎所有的对象在创建时 [[Prototype]] 都会被赋予一个非空的值。

小结

1，**原型是一个对象，为其他对象提供共享属性的对象，又称原型对象。**可以说原型不是一个固定的对象，它只是承担了某种职责。当某个对象，承担了为其他对象提供共享属性的职责时，它就成了该对象的原型。换而言之，不同对象的原型可能都是不一样的。 2，**几乎所有对象**在创建时都会被赋予一个非空的值作为原型对象的引用，来实现共享数据的效果。 3，在不同的对象上原型存放的方式也有所差别

- **函数**(function)：函数是一种特殊的对象，函数的原型存放在**prototype**属性上
- **对象**(Object): 普通对象的原型是存放到内置属性**[[Prototype]]**上，可以通过对象的**__proto__**来访问对象的原型。
- **数组**(Array): 数组也是一种特殊的对象，但与函数不同的是它的原型和普通对象一样，也是存放到内置属性**[[Prototype]]**上，可以通过数组的**__proto__**来访问数组的原型。

原型的创建

函数的原型的创建

无论何时，只要创建一个函数，就会按照特定的规则为这个函数创建一个prototype属性(指向原型对象)。 如下，创建了一个空的函数，函数上有对应的原型

```
let Person = function(){};
console.log(Person.prototype);
```

创建时函数的初始原型，只有一个名为**constructor**的属性和**[[Prototype]]**的内置属性。

image.png

我们也可以在函数的原型上添加数据和方法，通过构造函数的实例来访问。

```
let Person = function(){};
Person.prototype.a = '10';
Person.prototype.say = function(){
    console.log('你好，我是函数原型上的方法');
```

```
}
let person = new Person();
console.log(person.a);
person.say();
console.log(Person.prototype);
```

image.png

对象的原型的创建

前面已经说到过**几乎所有对象**（一些特殊的对象在之后的原型链中会提到）创建时都会被赋予一个非空的值作为原型。对象的原型是存放到内置对象**[[Prototype]]**上的，可以通过**__proto__**访问。

```
let obj = {};
console.log(obj);
console.log(obj.__proto__);
```

可以看到对象的原型上有很多的属性和方法，其中一些常用的如toString，valueOf等等，平时我们调用对象上的这些方法时就是调用对象原型上的方法。

image.png

数组的原型的创建

数组是一种特殊的对象，这里直接创建一个空数组查看它的原型

```
let arr = [];
console.log(arr);
console.log(arr.__proto__);
```

可以看到数组原型上有很多方法，我们平常用到的数组方法push、pop、join、forEach等等方法都是调用的数组原型上的方法。

image.png

小结

通过对函数、对象、数组不同对象原型的查看，我们可以看到，每当我们新创建一个对象或数组时，都可以通过它们的原型来共享一些公共的方法和属性。通过这种特殊的机制(原型)实现了数据和方法的共享。 这里需要注意的一点是：创建函数、对象和数组时为其创建的属性**prototype**或**[[Prototype]]**是**对其他对象的引用。**如果我们通过某个对象修改了它的原型，那其他对象的原型也是引用的同一个的话也会对应变化。 举个例子，创建了两个对象obj1和obj2，并且在obj1的原型上添加了say方法，但在obj2也可以调用这个方法，因为它们是对同一个原型对象的引用。

```
let obj1 = {};
let obj2 = {};

obj1.__proto__.say = function(){
  console.log('obj1的原型上添加了一个say方法');
}
obj1.say();  //obj1的原型上添加了一个say方法
obj2.say();  //obj1的原型上添加了一个say方法
```

其示意图如下：

image.png

原型的层级

属性和方法的访问路径

在通过对象访问属性时，会按照这个**属性的名称**开始搜索。首先会在**对象实例**身上查找，找到了给定的名称，则返回该名称对应的值。如果没有找到这个属性，则搜索会沿着指针进入**原型对象**，然后在原型对象上找到该属性后，再返回对应的值。这就是原型用于在多个对象实例间共享属性和方法的原理。

```
let obj1 = {};
obj1.__proto__.a = 100;
console.log(obj1.a) // 100  ，来自原型对象
```

属性和方法的覆盖

如果给对象实例身上添加一个属性，这个属性就会遮蔽原型对象上的同名属性，虽然不会修改它，但会屏蔽对它的访问。JavaScript提供了 `hasOwnProperty()` 方法来确定某个属性是在实例上还是在原型对象上。

```
let obj1 = {a:50};
obj1.__proto__.a = 100;
console.log(obj1.a) // 100  ，来自实例对象

let obj2 = {b:10};
obj2.__proto__.c = 5;

console.log(obj1.hasOwnProperty('a'))  // true
console.log(obj2.hasOwnProperty('b'))  //true
console.log(obj2.hasOwnProperty('c'))  //false
```

构造函数，原型对象和函数实例三者间的关系

constructor属性

通过前面的内容我们已经了解到函数在创建时会生成一个prototype属性，这个属性是指向原型对象的。**默认情况下，所有函数的原型对象会自动获得一个名为 **constructor** 的属性，指回与之关联的构造函数。** 看下面一段代码

```
//创建一个构造函数
const Person = function(){};
//创建一个函数实例
let per = new Person();

console.log(Person.prototype === per.__proto__);  // true
console.log(Person.prototype.constructor === Person);  // true
```

看着代码有点绕，我们来看下看图理解下。

image.png

构造函数的原型和函数实例对象的原型是同一个对象，**实例和构造函数原型之间有直接关系，但实例和构造函数之间没有。**

constructor属性的作用

1. 可以用来判断类型

```
//我们常用的是简写方式，最初写法是: let arr = new Array(10,20,30);
let arr = [10, 20, 30];
console.log(arr.constructor === Array); // true
```

1. 获取构造函数的信息

```
//创建一个构造函数
const Person = function(name,age){
    this.name = name;
    this.age = age;
};
//创建一个函数实例
let per = new Person('张三',25);
//获取构造函数的参数名称
let args = per.constructor.toString().match(/\(.*\)/).pop().slice(1,-1).split(',');
console.log(args); //['name','age']
```

原型链

原型链的终点

让我们回顾一下前面原型定义的小结：**几乎所有对象在创建时都会被赋予一个非空的值作为原型对象的引用。**那么原型对象也是一个对象，这个原型对象被创建时是不是也会被赋予一个非空的值作为其原型对象（即原型对象的原型）的引用？如此循环下去是否会构造一条无限长的访问链路？

image.png

原型链就是类似于这种访问链路，但它是有终点的。有一个特殊的对象在创建时会被赋予一个null值作为其原型对象的引用，这个对象的原型为空。可以把这个对象看做**最初原型**，作为所有原型链的终点(非手动设置的终点)。这个对象就是创建普通对象时的原型。

```
let obj1 = {};
console.log(obj1.__proto__);
```

image.png

普通函数，数组，对象三者原型之间的联系

对象

我们平时创建一个对象时通常是使用简写方式，完整的写法是通过构造函数Object创建的，每个普通对象都是构造函数Object的实例。

```
//简写
let obj1 = {};
//完整写法
let obj = new Object();
console.log(obj1.__proto__ === Object.prototype); //true
```

它们之间的关系如图所示。

image.png

构造函数Object的原型是一个比较特殊的对象，我把这个对象成为**最初原型对象**，这个对象的**__proto__**属性是指向**null**的。JS其他对象上的原型对象都是从它演变而来的。

数组

数组和对象类似，数组是构造函数Array的实例。数组构造函数Array和对象构造函数Object的关联是：**构造函数Array的原型对象的原型对象和构造函数Object的原型对象是同一个对象。**

```
let obj1 = {};
let arr = [];
console.log(arr.__proto__ === Array.prototype);  //true
console.log(obj1.__proto__ === arr.__proto__.__proto__); //true
```

关系示意图如下。

image.png

函数

普通函数的原型对象和Array，Object等构造函数的原型对象是不同的，但它们又最终汇交于**最初原型对象。**

```
let obj1 = {};
const Person = function(name,age){
    this.name = name;
    this.age = age;
};
let per = new Person('张三',25);
console.log(Person.prototype === per.__proto__);  //true
console.log(Person.prototype.__proto__ === obj1.__proto__);  //true
```

其关系图如下所示。

image.png

图中红色的链路就是数组，对象，函数构成的原型链。它们的终点是最初原型对象。

小结

原型链是通过对象特有的**原型**构成的一种链式结构，主要用来继承多个引用类型的属性和方法。默认情况下，所有引用类型都继承自Object。

参考文章

blog.csdn.net/zz_jesse/...

mp.weixin.qq.com/s/1UDI...

zhuanlan.zhihu.com/p/56...

jianshu.com/p/7dda47907...

blog.csdn.net/lvdou1120...

发布于 2023-07-19 17:19 · IP 属地湖北

JavaScript 原生 JavaScript js 原型

发布一条带图评论吧

1 条评论 默认 最新

kasser

属性和方法的覆盖
这里打印的obj.a 应该是50，你解释对了，但是数据错了😘

11-07 · IP 属地上海 回复 喜欢

推荐阅读

JS中的原型和原型链详解

JS中的原型和原型链是大家彻底搞懂JS面向对象及JS中继承相关知识模块非常重要的一个模块，一旦突破这块知识点，相信大家对JS会有一个更新、更全面的认识。 一、 什么是原型？任何对象都有一...

游戏开发 发表于cocos...

浅谈JS

极乐君