## 我所理解的从原型到原型链

先上代码：

```
function Person() {
    console.log('这个是函数');
};
let person = new Person();
console.log(Person.prototype === person.__proto__); // true
```

> 在这个例子中，`Person`就是一个构造函数，使用`new`创建了一个实例对象`person`，但是...什么是`prototype`、`__proto__`？为什么他们相等？

# prototype

在JavaScript中，我们创建一个函数A（**就是声明一个函数**），那么浏览器就会在内存中创建一个对象B，而且每个函数都默认会有一个属性`prototype`指向了这个对象（即：**prototype的属性的值是这个对象**）。这个对象B就是函数A的原型对象，简称函数的原型。这个原型对象B 默认会有一个属性 **constructor** 指向了这个函数A （意思就是说：constructor属性的值是函数A ）。

当把一个函数作为构造函数 （理论上任何函数都可以作为构造函数）使用new创建对象的时候，那么这个对象会自动添加一个不可见的属性`__proto__`，而且这个属性指向了构造函数的原型对象，新创建的对象会从原型对象上继承属性和方法。

代码：

```
function Person() {
    console.log('这个是函数');
};
Person.prototype.age = 22;
Person.prototype.name = 'Tom';
let person1 = new Person();
let person2 = new Person();
console.log(person1.age, person1.name); // 22, Tom
console.log(person2.age, person2.name); // 22, Tom
console.log(person2.age === person2.name); // true
```

> 上面代码可以理解：
>
> 1. 创建函数`Person`的时候，浏览器内存会创建一个对象，这个对象是函数`Person`的原型对象，因为`Person`函数的`prototype`属性指向了这个原型对象，所以可以使用`Person.prototype`直接访问到原型对象。
> 2. `Person.prototype.age`：给`Person`函数的原型对象中添加一个属性`age`并且值是`22`
> 3. `Person.prototype.name`：给`Person`函数的原型对象中添加一个属性`name`并且值是`Tom`
> 4. 当`Person`函数使用`new`创建了实例对象`person1`和`person2`时候，因为`person1`和

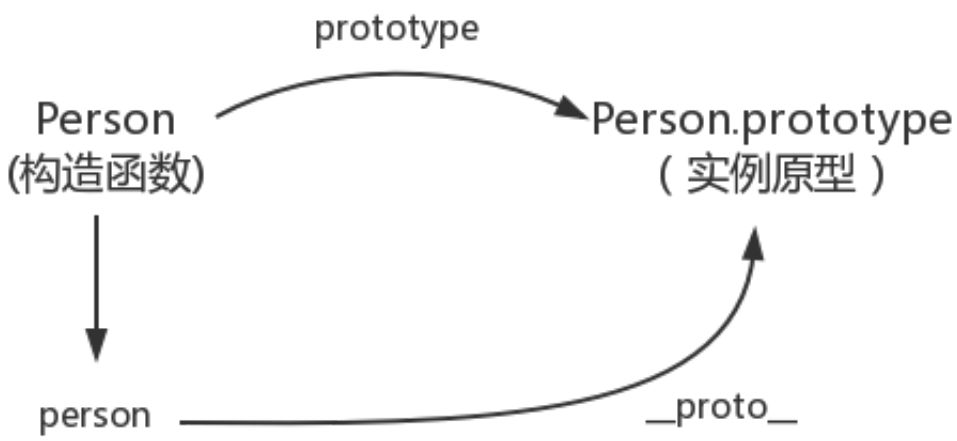
> `person2`的原型就是`Person`函数的`prototype`属性指向了那个对象（原型对象），与此同时，因为`person1`和`person2`中没有`name`和`age`属性，所以`person1`和`person2`"继承"了`age`和`name`的属性。

> 原型对象默认只有属性：`constructor`。其他都是从Object继承而来，暂且不用考虑。

## proto

这是每一个`javaScript`对象（除了`null`）都具有的一个属性，叫`__proto__`，这个属性会指向该对象的原型。

```
function Person() {
    console.log('这个是函数');
};
let person = new Person();
console.log(person.__proto__ === Person.prototype); // true
```




## proto指向谁?

1. 字面量方式：

```
let a = {};
console.log(a.constructor); //function Object() { [native code] } (即构造器Object)
console.log(a.__proto__ === a.constructor.prototype); //true
```

2. 构造器方式：

```
let A = function () {

};
let a = new A();
console.log(a.constructor); // function(){}（即构造器function A）
console.log(a.__proto__ === a.constructor.prototype); //true
```

3. Object.create()方式：

```
let a1 = {a:1};
let a2 = Object.create(a1);
console.log(a2.constructor); //function Object() { [native code] }（即构造器Object）
console.log(a2.__proto__ === a1);// true
console.log(a2.__proto__ === a2.constructor.prototype); //false（此处即为例外情况）
```

## prototype 和 proto 的区别

1. `prototype` 是函数(function) 的一个属性, 它指向函数的原型，
2. `__proto__` 是对象的内部属性, 它指向构造器的原型, 对象依赖它进行原型链查询。
3. `prototype`只有函数才有，其他(非函数)对象不具有该属性。而`__proto__`是对象的内部属性，任何对象都拥有该属性。

```
let a = {};
console.log(a.prototype); //undefined
console.log(a.__proto__);  //Object {}

let b = function(){}
console.log(b.prototype); //b {}
console.log(b.__proto__);  //function() {}
```

---

既然实例对象和构造函数都可以指向原型，那么原型是否有属性指向构造函数或者实例呢?

## constructor

指向实例倒是没有，因为一个构造函数可以生成多个实例，但是原型指向构造函数倒是有的，这就要讲到第三个属性：`constructor`，每个原型都有一个`constructor`属性指向关联的构造函数：

```
function Person() {
    console.log('这个是函数');
};
console.log(Person === Person.prototype.constructor); // true
```

综上得出：

```
function Person() {
    console.log('这个是函数');
}
var person = new Person();
console.log(person.__proto__ === Person.prototype) // true
console.log(Person.prototype.constructor === Person) // true
// ES5的方法,可以获得对象的原型
console.log(Object.getPrototypeOf(person) === Person.prototype) //true
```

有时候根据需要，可以`Person.prototype`属性指定新的对象，来作为`Person`的原型对象。但是这个时候新的对象的constructor属性则不再指向Person构造函数了。

```
//直接给Person的原型指定对象字面量。则这个对象的constructor属性不再指向Person函数
Person.prototype = {
    name:"志玲",
    age:20
};
let p1 = new Person();
console.log(p1.name); // 志玲
console.log(Person.prototype.constructor === Person); //false
// 如果constructor对你很重要，你应该在Person.prototype中添加一行这样的代码：
Person.prototype = {
    constructor : Person    //让constructor重新指向Person函数
}
```

知道了构造函数、实例原型、和实例之间的关系，接下来说说实例和原型的关系。

## 实例与原型

当读取实例的属性时，如果找不到，就会查找与对象关联的原型中的属性，如果还查不到，就去找原型的原型，一直找到最顶层为止。

```
function Person() {
    console.log('这个是函数');
}
Person.prototype.name = 'Tom';

let person = new Person();
person.name = 'Rose';

console.log(person.name) // Tom
delete person.name;
console.log(person.name) // Rose
```

在上面代码中，设置了`person`的`name`属性，所以可以读取到为`Rose`，当删除了`person`的`name`属性时，读取`person.name`，从`person`中找不到，就会从`person`的原型也就是`person.__proto__ == Person.prototype`中查找，幸运的是找到了为`Tom`，但是万一还没有找到呢？原型的原型又是什么呢？

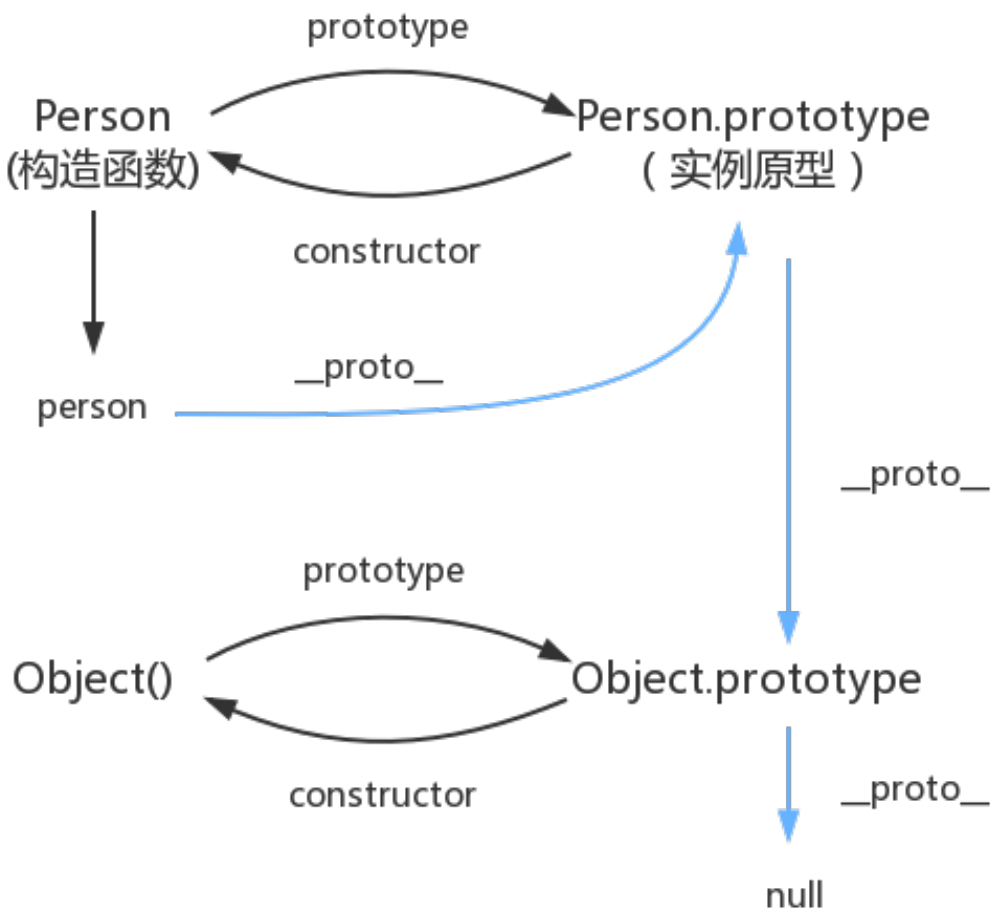
原型也是一个对象，既然是对象，我们就可以用最原始的方式创建它，那就是：

```
var obj = new Object();
obj.name = 'name'
console.log(obj.name) // name
```

原型对象是通过`Object`构造函数生成的，结合之前所讲,实例的`__proto__`指向构造函数的`prototype`,所以：

## 原型链

那`Object.prototype`的原型呢？`null`，嗯，就是`null`,所以查到`Object.prototype`就可以停止查找了，所以：




图中由相互关联的原型组成的链状结构就是原型链，也就是蓝色的这条线。原型链是基于`__proto__`的。js对象有一个指向一个原型对象的链。当试图访问一个对象的属性时，它不仅仅在该对象上搜寻，还会搜寻该对象的原型，以及该对象的原型的原型，依此层层向上搜索，直到找到一个

名字匹配的属性或到达原型链的末尾。

由于`__proto__`是任何对象都有的属性，而js里面万物皆为对象，所以会形成一条`__proto__`连起来的链条，递归访问`__proto__`必须最终到头，并且值为`null`。当js引擎查找对象的属性时，先查找对象本身是否存在该属性，如果不存在，会在原型链上查找，但不会查找自身的`prototype`

```
let A = function(){

};
let a = new A();
console.log(a.__proto__); //Object {}（即构造器function A 的原型对象）
console.log(a.__proto__.__proto__); //Object {}（即构造器function Object 的原型对象）
console.log(a.__proto__.__proto__.__proto__); //null
```

在javasrcript中，原型链的主要目的就是实现继承，其基本思想是利用原型让一个引用类型继承另一个引用类型的属性和方法。代码：

```
// 定义一个构造函数
function Father() {
    // 添加name属性，默认直接赋值了。当然也可以通过构造函数传递过来
    this.name = '马云';
}
// 给Father的原型添加giveMoney方法
Father.prototype.giveMoney = function () {
    console.log('我是Father原型中定义的方法');
}
// 再定义一个构造函数
function Son() {
    //添加age属性
    this.age = 18;
}
// 关键地方：把Son构造方法的原型替换成Father的对象。
Son.prototype = new Father();
// 给Son的原型添加getMoney方法
Son.prototype.getMoney = function () {
    console.log('我是Son的原型中定义的方法');
}
// 创建Son类型的对象
let son1 = new Son();
// 发现不仅可以访问Son中定义属性和Son原型中定义的方法，也可以访问Father中定义的属性和Father原型中的方法。
// 这样就通过继承完成了类型之间的继承。
// Son继承了Father中的属性和方法，当然还有Father原型中的属性和方法。

son1.giveMoney(); // 我是Father原型中定义的方法
son1.getMoney(); // 我是Son的原型中定义的方法
console.log('Father定义的属性：' + son1.name); // 马云
console.log('Son中定义的属性：' + son1.age); // 18
```

上面代码可以理解：

1. 定义`Son`构造函数后，我们没有再使用`Son`的默认原型，而是把他的默认原型更换成了`Father`类型对象。
2. 这时，如果这样访问`son1.name`，则先在`son1`中查找`name`属性，没有然后去他的原型(`Father`对象)中找到了，所以是”马云”。
3. 如果这样访问`son1.giveMoney()`，则现在`son1`中这个方法，找不到去他的原型中找，仍然找不到，则再去这个原型的原型中去找，然后在`Father`的原型对象找到了。
4. 从图中可以看出来，在访问属性和方法的时候，查找的顺序是这样的：对象->原型->原型的原型->...->原型链的顶端。 就像一个链条一样，这样**由原型连成的”链条”，就是我们经常所说的原型链。**
5. 从上面的分析可以看出，通过原型链的形式就完成了JavaScript的继承。

## `new`关键词到底起了什么作用呢

`new`关键词的作用就是完成实例与父类原型之间关系的串接，并创建一个新的对象。简单来说，`new`就是把`父类`的`prototype`赋值给实例的`__proto__`