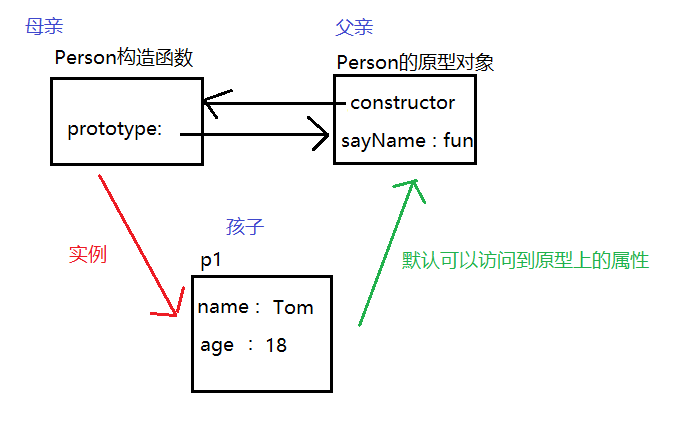
### 原型与原型链

1. **原型**
2. 定义：每一个函数创建出来的时候，浏览器会自动创建这个函数对应的原型对象。
3. 使用原型对象：
4. 获取到函数自身的原型对象，用prototype属性；
5. 原型对象上有一个constructor属性，可以指回自己的函数；
6. 要解决构造函数的问题，我们可以把函数赋值给原型对象：构造函数.protottype.xxx = function(){}；
7. 构造函数的实例会默认去构造函数的原型上查找对应的方法；



1. 内置对象的原型链：

内置对象主要有：Array、String、Date、Math;

这里拿数组举例：所有的数组都是Array这个函数的实例；

给Array的原型进行排序，那么所有的数组都可以调用这个排序方法；

例如:

*Array*.prototype.mySort = *function*(){

for(*var* i = 0; i < this.length - 1; i++) {

for(*var* j = 0; j < this.length - 1 - i; j++) {

if(this[j] > this[j + 1]){

*var* temp = this[j];

this[j] = this[j + 1];

this[j + 1] = temp;

}

}

}

return this;

}

*var* arr = [1,2,13,7];

*console*.log(arr.mySort());

打印出的结果就是排序好的数组。

1. 原型的简单使用方式

*function* Person(){

}

*Person*.prototype = {

constructor: Person,

say: *function*(){

*console*.log('shuohua');

},

eat: *function*(){

*console*.log('eat');

}

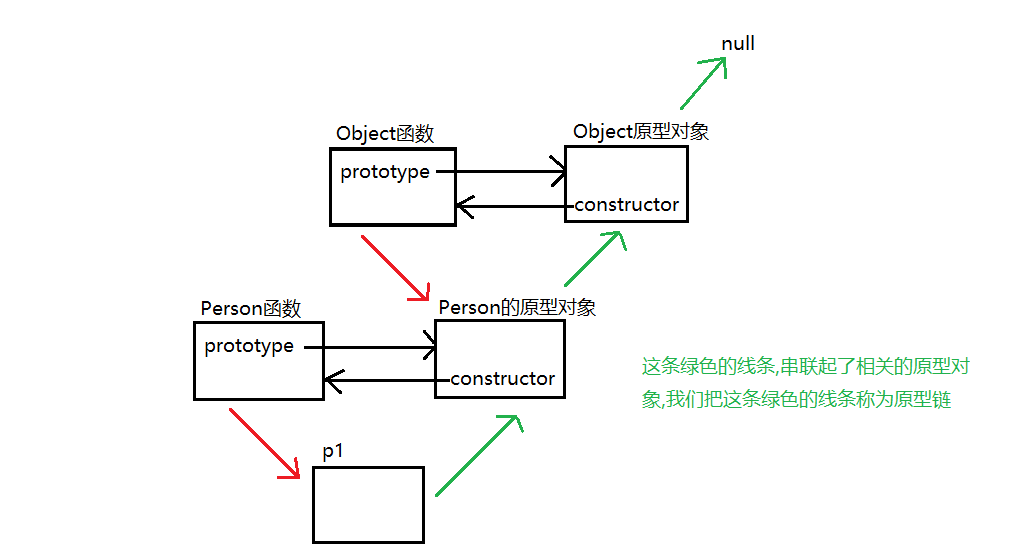
}

*var* p = new Person();

p.say();

p.eat();

1. **原型链：**
2. 原型链:实例可以默认访问到自己构造函数的原型对象,原型对象又可以访问自己构造函数的原型对象,这样的就形成一条原型链。



1. 对象属性的查找规则: 先找自己身上找,找不到就顺着原型链往上找,都没有就返回undefined
2. Instanceof:

作用: 判断一个函数的原型对象是否在对像的原型链上，

语法：对象 instanceof 函数 是则返回true,不是则返回false;

4、原型链不可变: 一旦实例创建完成,那么这个实例的原型链就固定. 不会随着自己构造函数的原型的变化,而变化

例如：

*function* Person()

}

*var* p1 = new Person();

*console*.log(p1 instanceof Person); //true

*Person*.prototype = {

constructor: Person,

say:*function*(){

*console*.log('shuohau');

}

}

*var* p2 = new Person();

*console*.log(p1 instanceof Person); //false

*console*.log(p2 instanceof Person); // true

p1.say();

p2.say();

