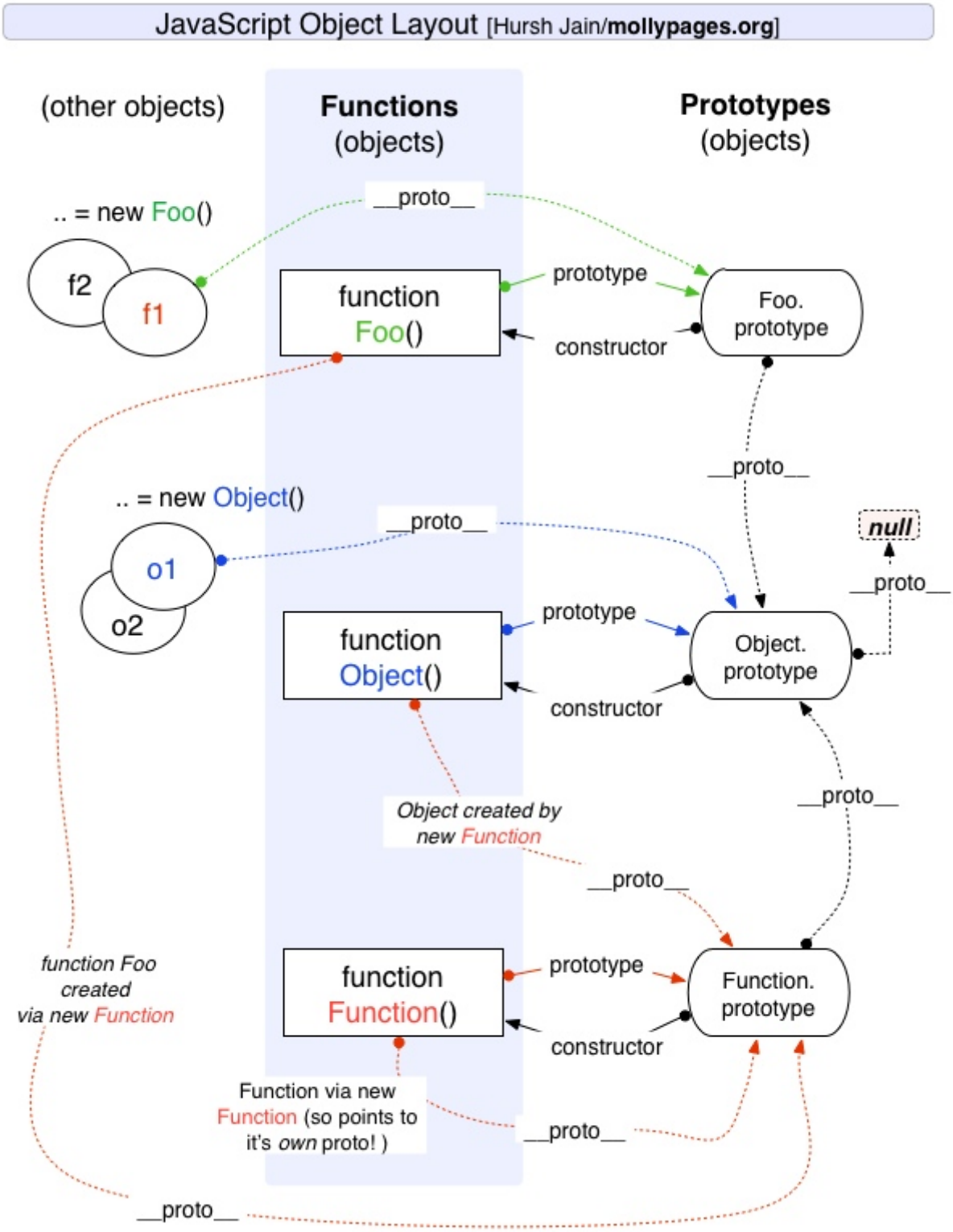
原型链



继承

1. 原型链继承：

   子类的原型是超类的实例

```
function SuperType() {
    this.colors = ['red', 'blue'];
}

function SubType() {

}

SubType.prototype = new SuperType();
```

缺点：本来在父对象中的实例属性，成了子对象的原型属性；在创建子对象时，没法给超类构造函数传参

2. 借用构造函数继承：
   直接在子类的构造函数中调用父类的构造函数

```
function SuperType() {
    this.colors = ['red', 'blue'];
}

function SubType() {
    SuperType.call(this);
}
```

缺点：方法都在构造函数中定义了，因此无法复用函数

3. 组合继承：
   使用原型链实现对原型属性和方法的继承，通过借用构造函数来实现对实例属性的继承

```
function SuperType(name) {
    this.name = name;
    this.colors = ['red', 'white'];
}

SuperType.prototype.sayName = function() {
    alert(this.name);
}

function SubType(name, age) {
    SuperType.call(this, name);
    this.age = age;
}

SubType.prototype = new SuperType();
SubType.prototype.constructor = SubType; // 一定要把原型的constructor属性指回子类构造函数
```

缺点：超类的构造函数被调用了两次，超类的实例属性是子类的原型属性，同时又被子类的实例属性重写，造成了内存浪费

4. 寄生组合式继承：
   在指定子类的原型时，不是调用超类的构造函数，而是用超类原型的一个浅拷贝副本来指定子类的原型

```
function inheritPrototype(SubType, SuperType) {
    var prototype = Object.assign({...SuperType.prototype}, {constructor: SubType});
    SubType.prototype = prototype;
}

function SuperType(name) {
    this.name = name;
    this.colors = ['red', 'blue'];
}

SuperType.prototype.sayName = function() {
    alert(this.name);
}

function SubType(name, age) {
    SuperType.call(this, name);
    this.age = age;
}

inheritPrototype(SubType, SuperType);

SubType.prototype.sayAge = function() {
    alert(this.age);
}
```

备注：《JavaScript高级教程》中的寄生组合式的实现事实上改变了父类的prototype, 其实是不正确的

5. 原型式继承：
   直接用 `Object.create()` ，将超类实例直接作为子类实例的原型

```
var superObj = {
    text: 'hello',
    color: []
};

var subObj = Object.create(superObj); // subObj.__proto__ === superObj is true
```

经典面试题

2019 面试准备 - JS 原型与原型链