基础

函数：工具 --- 封装细节： 名称 参数 作用

对象：工具包 --- 多个工具包和属性

框架：浏览器兼容 封装一些常用的代码 多个对象

传统开发四要素

1. 定义变量
2. 获取元素
3. 绑定元素
4. 绑定事件

对象的基础

对象的基本写法 -- 原型

如何使用对象 -- 实例化

抽象 -- 类

具体 -- 实例

类可以有多个实例

当我们new 一个实例后，系统自动做了如下事情

创建一个空对象 var p = {}

拷贝构造函数中的方法属性到空对象中

自动成生一个属性_proto_指向类的原型p.__proto__ = XXXX.prototype

call：借用另一个对象、改变this指向

实例化本质：拷贝构造函数属性、方法；原型对象通过一个属性继承

判断数据类型

typeof

识别基本类型

无法识别 引用类型的具体类型

instanceof

判断已知对象类型或函数

不能用来判断字符串和数字等

constructor

判断对象的构造函数

无法判断是否为直接继承或间接继承

toString.call()

识别未知object的类型

无法识别实例化对象的类型

构造函数

构造函数和普通函数的区别：使用new调用，无return

使用构造函数创建属性的四种方式

1. 参数形式
2. 默认值形式
3. 动态添加形式
4. 混合模式

get set 属性取值器

Get set 用于对一个属性进行包装

公有属性和私有属性

私有属性：只能在对象构造函数调用

公有属性：在对象实例化后调用

constructor

用于检测某个实例的构造函数是哪个

构造函数的隐藏属性

实例的属性拷贝子构造函数

实例也拥有该属性

原型

原型对象不管实例化多少次，都只会生成一次。

通过原型创建对象，其实创建的是两个对象：
-构造函数对象
-原型对象
当我们实例化的时候，该实例自动拷贝构造函数的所有属性和方法，
而对于原型对象，则不拷贝，而是通过一个属性‘铁链’

其实js中本来没有对象这个概念
利用函数实现了对象

原型对象本质：

原型对象的属性和方法可以被所有实例共享

实例属性和原型属性判断 ：
hasOwnProperty() 方法
如果是实例对象—ture
如果是原型对象 - false

创建对象的几种方式

1. 传统方式

```
var obj = new Objert();
obj.name = "obj";

var obj2 = obj;
obj2.name = "This is obj2";
alert(obj.name);    //obj2
```

缺点：创建类似对象，代码冗余

1. 工厂模式

```
function createObject (name, age) {
    var obj = new Object();
obj.name = name;
obj.age = age;
obj.run = function () {
    return this.name + this.age;
};
return obj;
}

var obj1 = createObject("ojb1", 100);
var obj2 = createObject("ojb2", 100);

alert(typeof obj1);
alert(obj2 instanceof createObject); //false
```

优点：调用时传参，可以重复多次实例化；

缺点：识别问题，无法知晓是哪个对象的实例

1. 构造函数

```
function CreateObj (name, age) {
this.name = name;
this.age = age;
this.run = function () {
    return this.name + this.age;
}
}

var obj1 = new CreateObj1("obj1", 10);
var obj2 = new CreateObj2("obj2", 20);

alert(obj2 instanceof CreateObj);    //true
```

优点：解决了识别问题

缺点：每次调用都会初始化，浪费内存

1. 原型对象

```
//使用构造函数创建原型对象
function Create1 () {}
Create.prototype.name = "obj1";
Create.prototype.age = 10;
```

```
var box1 = new Create1();

//使用字面量创建原型对象
function Create2 () {}
Create.prototype = {
name : "Obj",
age : 10
run : function () {
    return this.name + this.age;
}
}
var box1 = new Create2();
```

//使用构造函数创建原型对象和使用字面量创建原型对象的区别
//字面量创建的方式使用constructor 属性不会指向实例，而会指向Object
//构造函数创建的方式则相反。

优点：共享， 省略构造函数传承初始化
缺点：传递引用类型会出错

1. 混合模式

```
function Box (name, age) {
    this.name = name;
    this.age = age;
    this.family = ['父亲', '母亲', '妹妹'];
}

Box.prototype = {
    coustructor : Box,
    run : function () {
        return this.name + this.age + this.family;
    }
}
```

优点：支持传递初始参数
构造函数模式用于定义实例属性
原型模式用于定义方法和共享属性
每个实例都会有自己的一份实例属性，同时又共享着方法

最大限度的节省了内存

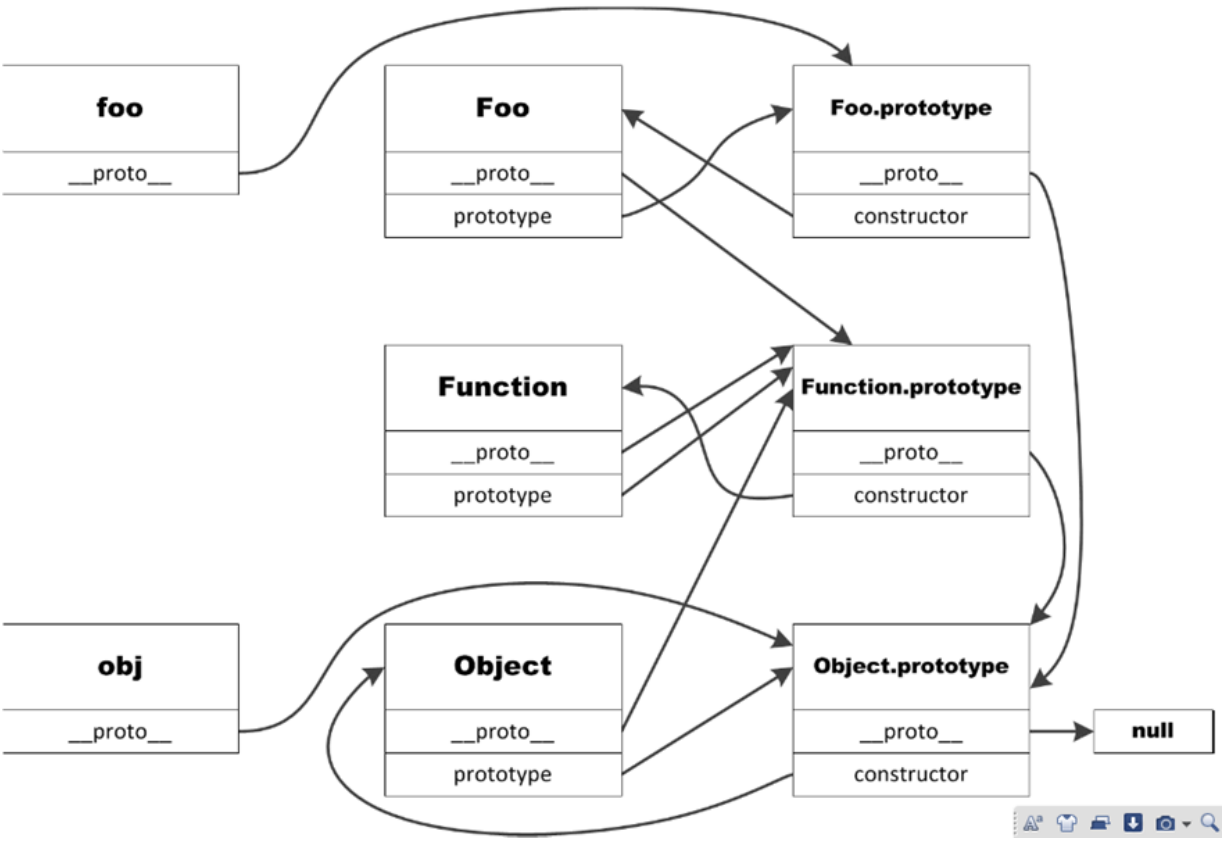
1. 内置对象

String Data Math Array RegExp Number Object Function Null

Booleean Error Cookie Session Navigutor

1. 拷贝
2. extend
3. 第三方框架

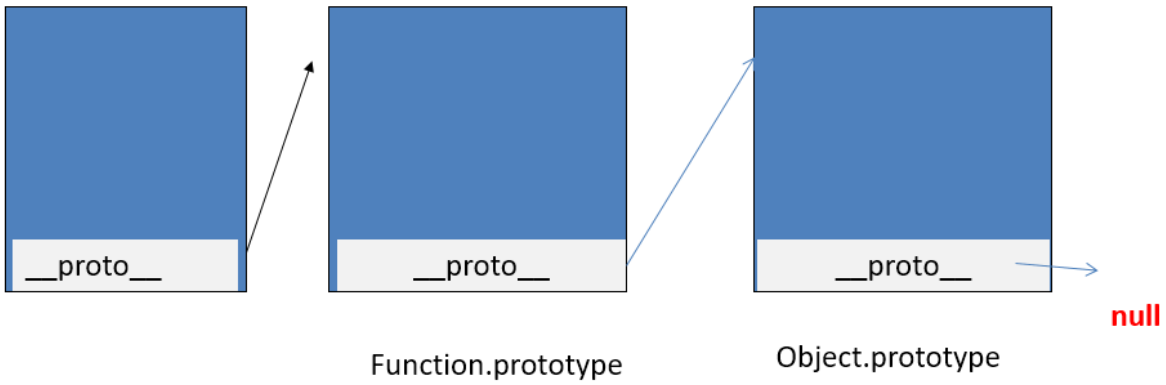
继承



属性访问搜索法则

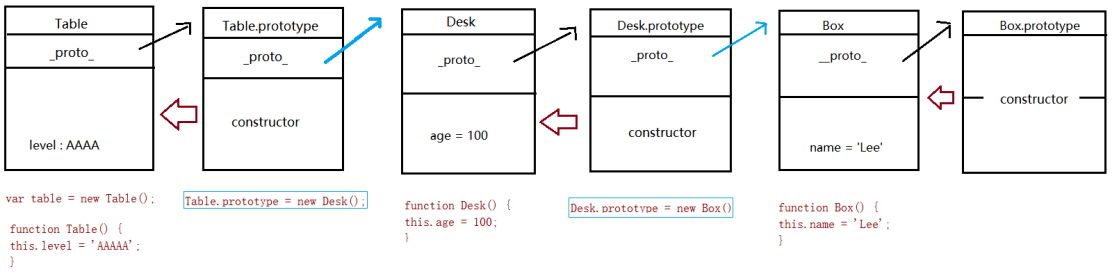
object.__proto__ > Function.prototype.__proto__ > Object.prototype.__proto__ > null

所有内置对象都是Function对象的实例



继承的九重境界

1. Object中的继承



- 最简单的继承

```
var obj = new Object();
```

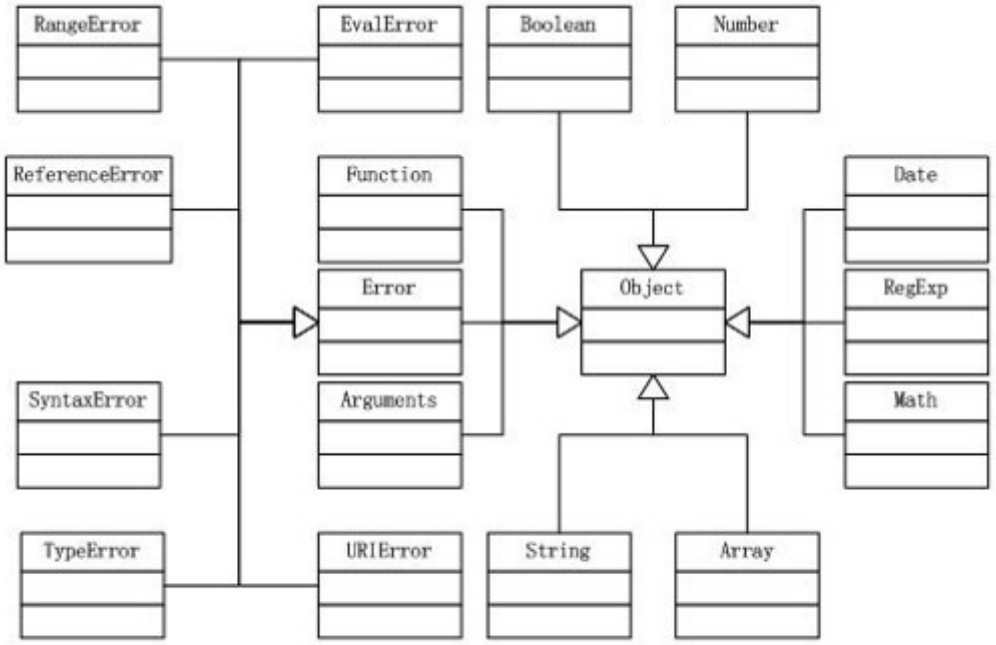
新建一个对象，继承Object的所有属性和方法

1. 内置对象的继承

所有的内置对象都继承在Object

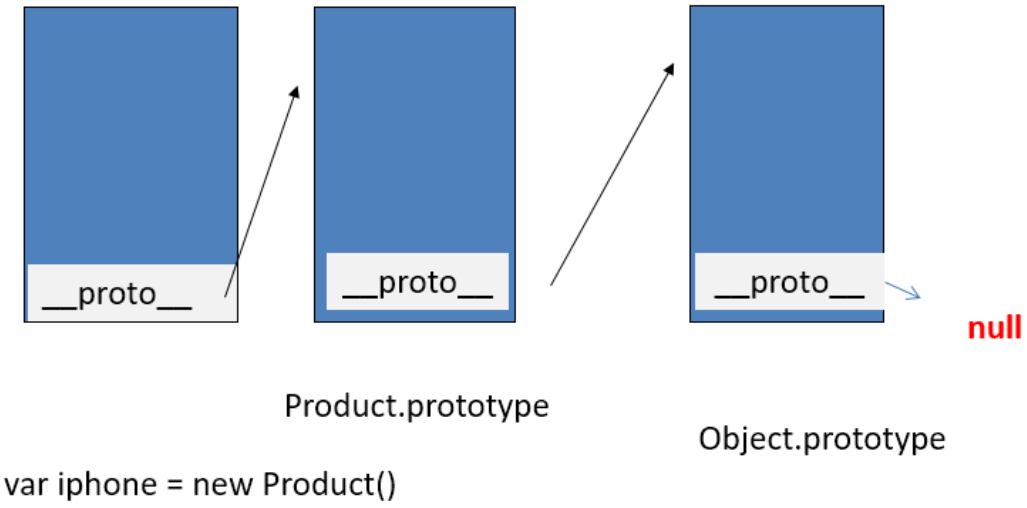
除了拥有自身的方法和属性之外

还拥有Object的所有属性和方法



1. 自定义对象的继承

继承的本质就是通过__proto__指针指向某个对象
系统能够自动链式访问所指向的对象的属性方法。
让第二个对象的__proto__指向另一个对象即可



1. 多种继承实现方式

原型继承无法传参

构造函数继承：
使用call方法，第一个参数为上下文；

但无法继承原型中的方法

组合继承：实现了继承属性和方法
但会调用两次超类构造函数
一次是在创建子类原型时
另一次是在子类构造函数内部

寄生组合继承

拷贝继承

1. 第三方框架实现继承

多继承

1. 面向对象三大特性和继承

封装

将属性方法归类，条理清晰
当出现错误，容易发现问题，先找到其属于哪个对象
隔离作用，当代码出现错误，不会影响其他对象
封装变化，对于一个需求经常变动的地方，封装起来，这样当代码需要经常修改，只需要修改单一方法
封装复杂：将一些复杂的功能封装起来，方便使用

继承

多态

同类‘事物’，多种形态
继承本身就是多态的一种实现。
同类事物---多态

1. 面向未来变化编程和继承

面向接口编程

面向接口编程步骤

规定接口

按照接口编写子模块

面向接口编程

面向抽象编程

面向未来编程

面向对象解决的问题

封装世界，应对变化

1. 设计模式和继承

了解即可

工作个3-5年 知道

工作5-10 熟悉

精通？？ 没有

面向未来编程

1. 忘记模式 挥洒自如 无招胜有招

基础：必须修炼完成前四层

中级：第五层，第六层，第七层

高级：第八层，第九层

BOM对象

Window location Document History Navigator Screen Error

window内置全局属性和方法

全局常量

Infinity, NaN, undefined

全局方法

eval(), is Finite(), isNaN(), parseFloat(), prseInt(), decodeURI()
decodeURIComponent(), encodeURI(), encodeURIcComponent()

window方法-对话框

alert() 函数：弹出消息对话框

confirm() 函数：弹出消息对话框

prompt() 函数：弹出消息对话框

Window方法 - 时间等待与间隔函数

setTimeout() 函数

clearTimeout() 函数

setInterval() 函数

clearInterval() 函数

Window方法 - 获取失去焦点

focus() 函数：使窗体或空间获得焦点

blur() 函数：使窗体或控件失去焦点

Window方法 - 新窗口

open() 函数：打开(弹出)一个新的窗体

close() 函数：关闭窗体