看一个简单demo：

function A(name){

this.name = name;

}

var aa=new A(“lxw”);

这里面new到底做了什么？先看一个很熟悉的函数

function B(name){

var obj=new Object();

obj.name = name;

return obj;

}

var bb=B(“lxw”); //看着是不是很面熟，调用函数B 得到返回的对象

aa和bb的调用只是差了一个 new ，所以可以这么想像

function A(name){

//当使用new 时，浏览器默认增加名为this的空对象

//var this = new Object();

this.name = name;

//最后，默认返回this

return this;

}

var aa=new A(“lxw”);

这里面this相当于B中内部的obj，唯一的区别是：B的调用是普通函数调用，A的调用是通过new 实现。此时的A被专家命名为构造函数，作用是构造一个对象，说白了还是函数

但是，我在想如果通过new去实例化某个函数，然后函数内部返回的是一个自定义的东西，会出现什么情况：

function C(){

return {

name: “lxw”

}

}

var cc=new C;

console.log(cc.name); //lxw，感觉还挺对的，但是，如果返回的是非对象类型：

function C(){

return 123;

}

var cc=new C;

console.log(cc); //返回 { } 一个空对象，所以这里就很奇怪了

所以，new做的还不止上面的那些

最终new之后所做的事情：

//注释部分的内容都是浏览器内部的默认实现

function C(name){

//1 默认创建空对象

//var this = new Object();

//2 将A中原型上的prototype原型对象放到\_\_proto\_\_属性

//这样，可以保证实例化对象可以访问A这个类中的公用属性和方法

//\_\_proto\_\_只有在Firefox下打印出来

//this.\_\_proto\_\_ = A.prototype;

//3 默认返回this对象（前提：类中没有自定义 return ）

//return this;

//4 如果类中有自定义return 返回result，则会检测自定义的结果result是否是Object类型，如Function、Array、Object

//如果是Object类型，返回自定义对象result；如果不是，仍返回this

//return (typeof result === Object) ? result: this ;

}

var cc=new C(“lxw”);