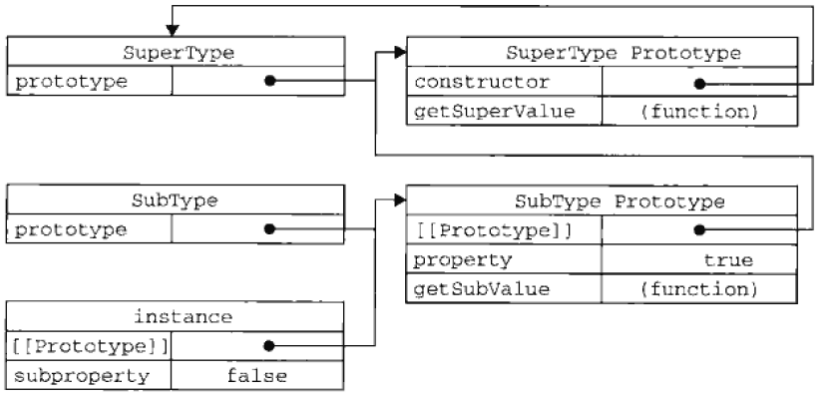
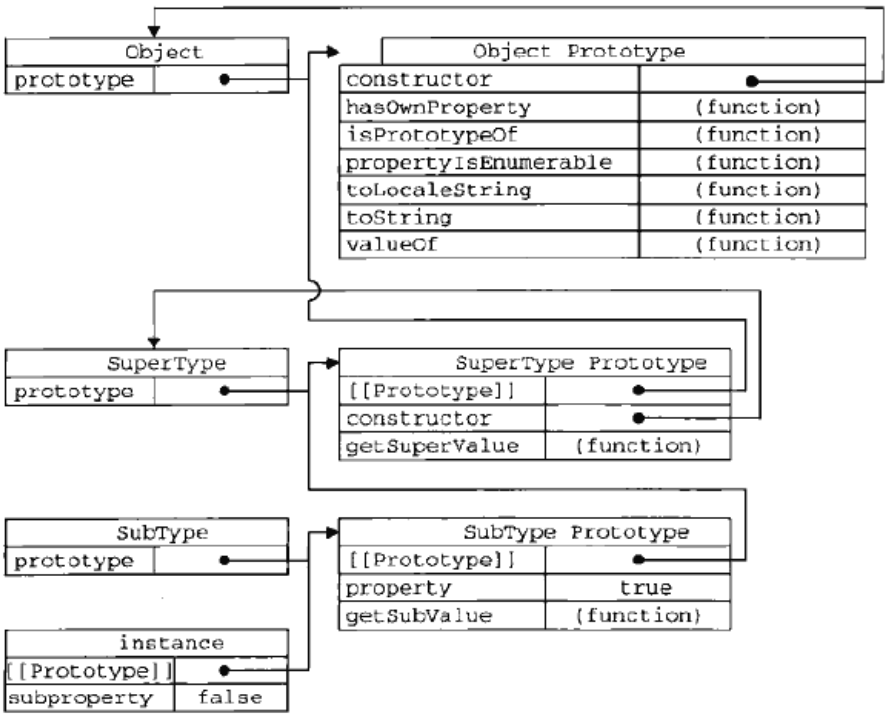
# 继承

**1. 原型链**

```
 //父类型
function SuperType () {
  this.property = true;          //父类型的实例属性
}

SuperType.prototype.getSuperValue = function () {
  return this.property;          //父类型的原型方法
};

//子类型
function SubType () {
  this.subProperty = false;
}

SubType.prototype = new SuperType ();      //用父类型的新实例改写子类型的原型对象，实现继承

SubType.prototype.getSubValue = function () {        //子类型添加新的原型方法
  return this.subProperty;
};

var instance = new SubType ();
alert (instance.getSuperValue);     //true
```




默认的原型Object




优点：

1. 父类型的原型方法属性和实例方法属性，子类型都能访问到。
2. 父类型新增的原型方法和属性，子类型也能访问到。
3. 实现了函数的复用，节省内存。

缺点：

1. 父类型中的实例属性变成了子类型中的原型属性，子类型的所有实例都会共享这些属性。如果该属性是引用类型值，那么如果对某一个实例中的该属性进行修改，则所有实例中的该属性都会发生变化。
2. 在创建子类型的实例的时，不能向父类型的构造函数传递参数。

**2.借用构造函数**

使用apply()或all()：

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

function SubType() {
    SuperType.call(this);            //在子类型构造函数的内部调用父类型构造函数，实现继承
}

var instance1 = new SubType();
//子类型构造函数SubType()创建了实例后，会在该实例环境下调用父类型构造函数SuperType()，然后执行SuperType()函数中定义的所有对象初始化代码
instance1.colors.push("black");
alert(instance1.colors);       //red,green,blue,black

var instance2 = new SubType();
alert(instance2.colors);         //red,green,blue
```

给父类型构造函数传递参数：

```
 function SuperType(name) {
        this.name = name;
    }

    function SubType() {
        //继承了SuperType
        SuperType.call(this, "yangjinjin");
        this.age = 27;
    }

    var obj = new SubType();

    alert(obj.name); //yangjinjin
    alert(obj.age); //27
```

优点：

1.解决了原型链方法中，子类型实例共享父类型引用值属性的问题。

2.可以在子类型构造函数中向父类型构造函数传递参数。

缺点：

1.方法都在构造函数内部定义，无法实现函数的复用。

2.在父类型的原型中定义的方法，对子类型是不可见的。

**3.组合继承**

将原型链和借用构造函数的技术组合到一块的模式。

使用原型链实现对原型属性和方法继承，而通过借用构造函数来实现实例属性的继承。

```
function SuperType (name) {
 this.name = name;          //要通过构造函数方式继承的属性
 this.colors = ["red","blue","green"];
}

SuperType.prototype.sayName = function () {        //要通过原型链方式继承的方法
 alert (this.name);
};

function SubType (name,age) {
 SuperType.call (this,name);        //构造函数方式继承属性
 this.age = age;        //给子类型添加新的实力属性
}



SubType.prototype = new SuperType ();          //原型链方式继承方法
SubType.prototype.sayAge = function () {        //给子类型添加新的原型方法
 alert (this.age);
};

var instance1 = new SubType ("a",17);
```

```
instance1.colors.push ("black");
alert (instance1.colors);    //"red,blue,green,black"
instance1.sayName ();    //"a"
instance1.sayAge ();    //17

var instance2 = new SubType ("b",21);
alert (instance2.colors);//"red,blue,green"
instance2.sayName();    //"b"
instance2.sayAge ();    //21
```

优点：

1. 实现了函数的复用。
2. 保证了每个实例都有它自己的属性。
3. 可以给父类型构造函数传递参数。