继承模式

1. 通过Person.prototype = {}
2. 通过person,call()　　并不节省效率
3. Person.prototype = Son.prototype　　这种方法son不能单独改变 同一区域两个名字
4. F.prototype = Father.prototype son.prototype = new F(); Best **圣杯模式**

```
function Father(){
}
function Son(){
}
function Inherit(Target,Origin) {
    function F(){}
    F.prototype = Origin.prototype;
    Target.prototype = new F();
    Target.prototype.constructor = Target;  //这两条是为了方便找继承
    //的对象及原型
    Target.prototype.uber = Origin.prototype;
}
Inherit(Son,Father);
var son = Son();
var father = Father();
```

立即执行函数封装，闭包变量私有化

```
var inherit = (function (){
    var F(){};
    return function(Target,Origin) {
        F.prototype = Origin.prototype;
        Target.prototype = new F();
        Target.prototype.constructor = Target;
        Target.prototype.uber = Origin.prototype;
    }
}());
```

命名空间

1.

曾经对于变量重名的方法(现在已经不用)

```
var org = {
    department1 : {
        jicheng : {
```

```
        name : '123'
    }
    zhangsan : {
        afge : 'abc'
    }
  }
}
var jicheng = org.deparment.jicheng
jicheng.name
```

2.

管理变量，防止污染全局，适用于模块化开发

利用闭包

```
name = 'abc';
var init = (function(){
var name = 'bcd';
function sayname(){
  console.log(name);
}
return function(){
  sayname();
}
}());
```

实现方法的连续调用

```
var deng = {
  smoke : function(){
    console.log('Smoking Now...cool');
   return this;
  },
  drink : function(){
    console.log('Drinking Now...so cool');
     return this;
  },
  perm : function (){
    console.log('Perming Now...yeah cool');
     return this;
  }
}
deng.drink().perm().smoke();
```

不用switch ifelse实现对象属性的选择性调用

首先 类似person.name的方式调用对象属性等同于person["属性名"]（前一种也会在内部实行转化）

底层原则

```
var deng = {
  wife1 : {name : 'xiaohong'},
  wife2 : {name : 'xiaoli'},
  wife3 : {name : 'xiaozhang'},
```

```
    wife4 : {name : 'xiaowang'}
    saywife : function (num){
        return this["wife + num"]
    }
}
deng.saywife(2).name
```

遍历对象属性名

```
var obj = {
    name : 'name',
    age : 13,
    sex : 'male',
    weight : 98,
    height : 190
}
for (var prop in obj){
    console.log(prop)   //遍历属性名
}
for(var prop in obj){
    console.log(obj.prop)    //错误--->obj['prop'] 然而并没有这个
    //属性,只好打印undefined
    console.log(obj[prop])   //遍历属性
}
```

hasOwnProperty()

判断属性是否属于自身　　能够区分原型

```
var obj = {
    name : 'name',
    age : 13,
    sex : 'male',
    weight : 98,
    height : 190,
    __proto__ : {
        lastname : 'ji'
    }
}
for (var prop in obj){
    if (obj.hasOwnProperty(prop)){
        console.log(obj[prop])
    }
}
```

in

只能判断属性是否能被对象调用 就算是对象的原型的也是属于对象也会返回TRUE

A instanceof B

判断A对象 是不是B构造函数 构造出来的 对象

看A的原型链上有没有B的原型

判断一变量arr是数组还是对象

```
Object.prototype.toString.call([])    //将this指向改为数组
Object.prototype.toString = function(){
    //识别this
    //返回相应的结果
}
```