20170730

```
//->复习20170729
var num = 2;
var obj = {num: 3};
obj.fn = (function (num) {
    this.num *= 2;
    num *= 2;
    return function () {
        this.num *= 2;
        num *= 3;
        console.log(num);
    }
})(obj.num);
var fn = obj.fn;
fn();//18
obj.fn();//54
console.log(num, obj.num);//8 6
```

1、单例模式

> 我们把描述东西特点特征的属性和方法汇总到一起，实现分组分类，避免了相互之间的冲突；我们把汇总在一起的那个对象称之为一个单独的个体或者是一个单独的实例 **=> “单例模式”**

```
var person1={
    name:"Junior",
    age:18
};
var person2={
    name:"Fancy",
    age:20
};

// ->person1和person2都是一个单独的个体（单独的实例），相互的name和age属性并不冲突，这就是基于“单例模式”的设计思想构造出来的
// ->person1或者person2不仅仅称之为对象名了，在单例模式下，他们称之为：“命名空间”
```

所谓的单例模式： 把描述事物的属性和方法放在同一个命名空间下，实现分组分类的作用，避免了相互之间的冲突（全局变量的污染）

```
//->项目中使用单例模式
/*
* 高级单例模式：基于js高阶编程技巧“惰性函数思想”封装的单例模式
*/
var nameSpace=(function(){
    //->自执行函数执行形成一个不销毁的私有作用域（闭包）
    return{
        init:function(){},
        ...
    }
})();
nameSpace.init();
```

单例模式是项目开发中最常用的设计模式之一，我们团队协作按模块化开发的时候，基本上都是基于单例的思想设计的，避免相互代码冲突

```
/*
* 模拟百度的开发：百度首页按功能可以分为以下几个模块：
* ->频道页卡 channelRender(channelModul)
* ->换肤
* ->天气
* ...
*/

//->public
//->utils：就是项目的公共方法库
var utils=(function(){//->utils常用公共方法库
    return {
        cookie:function(){},
        ...
    }
})();
//=>A
var channelRender=(function(){
    return {
        init:function(){...},
        drag:function(){...},
        ...
    }
})();

//=>B
var changeSkinRender=(function(){
    return {
        init:function(){...},
        change:function(){
            channelRender.drag();//调取别人的
            changeSkinRender.init();//->this.init(); [调取自己的]
            utils.cookie();//调取公共方法
        },
        ...
    }
})();
changeSkinRender.change();

//避免冲突：闭包（弊端：）/单例模式
```

2、工厂模式

> 体现出了函数的封装特点：把实现一个功能的代码进行封装，以后想实现这个功能，直接的执行函数即可，这样不仅仅实现了代码的“低耦合高内聚”，而且实现了批量生产的快速开发 **=> “工厂模式”**

3、面向对象编程思想（OOP）

> OOP：Object Oriented Programming
>
> - 对象：js中一切都是需要学习和研究的对象（泛指，抽象的名词）[万物皆对象]
> - 类：对象的具体细分（例如：自然界中分为植物类、动物类……，每一个大类还可以分为小类…）
> - 实例：每一个类别中具体的个体事物（例如：Junior就是人类中的一个实例）

js本身就是基于面向对象的思想构造出来的语言，所以js中肯定有很多的**“内置类”**

- 每一种数据类型都有一个自己所属的内置类，数据类型中的值都是所属类的实例
  - Number：每一个数字都是它的一个实例
  - String：每一个字符串都是它的一个实例
  - Boolean：
  - Null
  - Undefined
  - Object
    - Array
    - RegExp
    - Date
    - …
  - Function
  - …

> 每一个数据类型所用的方法，都是当前所属的类提前设定好的，实例就可以调取这个方法了，例如：Array数组类，数组为每一个实例都提前设定了一些方法
> Array.prototype

```
> Array.prototype
< ▼[constructor: function, toString: function, toLocaleString: function, join: function, pop: function…]
    ▶concat: function concat()
    ▶constructor: function Array()
    ▶copyWithin: function copyWithin()
    ▶entries: function entries()
    ▶every: function every()
    ▶fill: function fill()
    ▶filter: function filter()
    ▶find: function find()
    ▶findIndex: function findIndex()
    ▶forEach: function forEach()
    ▶includes: function includes()
    ▶indexOf: function indexOf()
    ▶join: function join()
    ▶keys: function keys()
    ▶lastIndexOf: function lastIndexOf()
     length: 0
    ▶map: function map()
    ▶pop: function pop()
    ▶push: function push()
    ▶reduce: function reduce()
    ▶reduceRight: function reduceRight()
    ▶reverse: function reverse()
    ▶shift: function shift()
    ▶slice: function slice()
    ▶some: function some()
    ▶sort: function sort()
    ▶splice: function splice()
    ▶toLocaleString: function toLocaleString()
    ▶toString: function toString()
    ▶unshift: function unshift()
    ▶Symbol(Symbol.iterator): function values()
    ▶Symbol(Symbol.unscopables): Object
    ▶__proto__: Object
```

> 以上这些方法就是数组给它的实例提供的，所以作为数组的实例，是可以调取这些方法使用的

- 元素或者节点集合类
    - HTMLCollection：元素集合类，通过DOM的方法获取到的元素集合都是它的实例，例如：getElementsByTagName、getElementsByClassName
    - NodeList：节点集合类，通过DOM方法获取到的节点集合都是它的一个实例，例如：getElementsByName、childNodes…
- DOM元素标签对应的内置类：每一个元素标签都有一个自己对应的内置类
    - div：HTMLDivElement ▶__proto__: HTMLDivElement
    - a：HTMLAnchorElement ▶__proto__: HTMLAnchorElement
    - p：HTMLParagraphElement __proto__: HTMLParagraphElement
    - …
- HTMLElement
- Element
- Node
- EventTarget
- Object
- …

4、构造函数设计模式

> **构造函数的作用：**基于面向对象的编程思想，创建js中的”自定义类“，基于自定义的类，创建相关的实例...

```

```

=> ▼ Person {}
▶ __proto__: Object

5、基于构造函数设计模式引申的原型模式

> 原型是构造函数的引申，构造函数中公共属性部分就是由原型来掌控的

```
1、所有的函数数据类型
    ->普通函数
    ->类
    天生具备一个属性：prototype，这个属性的值是一个对象,作用：当前类的公共属性和方法都在这个对象中存储着

2、  prototype天生具备一个属性：constructor（构造函数），这个属性存储的值就是当前类（函数）本身

3、每一个对象数据类型
    ->普通对象{}
    ->数组和正则等
    ->类的实例
    ->函数也是对象
    也会天生自带一个属性：__proto__，这个属性存储的值是它对应类的prototype属性的值

    //->理解
    //function Fn(){};
    //Fn.prototype={};
    Fn={
        prototype:{
            constructor:Fn
        }
    }
```