# 正式课第五天

## 单例模式

> 单例：单独的实例；
> 实例：把相同的事务总结（归纳、抽象）出来形成一类事物，把具体的属性和方法具体化，这个具体的描述就是实例

### 命名冲突

- 1、封闭空间 把一段代码放入一个函数内，当执行函数的时候，函数内的域和外界是互不干扰的；`(function(){})()`
- 2、命名空间： 把一些变量或者方法

单例模式的优势：

- 1.解决命名冲突
- 2.把相同事物归为一类

## 工厂模式

> 批量生产多个实例；通过传参去描述具体的实例；把生产后的对象返回到外界；

## 构造函数（new）

> new是一元运算符，专门运算函数的；运算别的就报错；

1、执行函数

- 不使用（）【用new fn】调用也是可以执行函数的，此时（）只是为了传参；
  2、构造函数（fn）中this指向了**当前实例**（此构造函数的返回值）
  该构造函数的返回值是“fn{}”；没有return也不是undefined；
  **new把函数变为实例化对象**
  3.return 的返回值默认指向当前实例this；
  有return 如果后面跟着的是一个基本类型，结果依然是实例，如果后边跟着的是一个引用类型，那么结果就是这个引用类型；

## 面向对象编程：

> 把描述相同的事物抽象出来，归为一类，把描述这个类的属性和方法挂在这个类的原型上的 一种编程方式就叫面向对象编程；
> 抽象：抽离出长得相像的部分；

- js的面向对象的特征
  - 抽象
  - 封装
  - 继承
  - 多态

## 原型模式

**使用原型的目的：**

> 当构造函数中写上方法的时 候，每new一次就生成一个方法，在同一个类生成的两个实例，他们相同的方法是不相等的，这就导致如果new若干次，那么就会生成若干个一模一样的方法，这样对性能不好；所以我们要使用原型的方式去吧方法挂在原型上
> **定义：**
> 当定义一个函数的时候，这个函数自身有一些属性或者方法，其中有一个属性叫做prototype。这个属性就叫原型；
> prototype 它是一个对象，它的值又是一个对象{prototype：{ } }
> **它的用处是如果实例化对象上没有某个属性或者方法还会去这个实例化对象的构造函数中的原型下去查找该属性或者方法**；
> **如果构造函数的原型上没有这个方法，那么还会去原型的原型链（*_proto_*）中查找,最终会找到 object身上；**

小 总结：构造函数下肯定有原型。实例化对象下肯定有原型链；

**实例化对象的原型链 `===` 构造函数的原型**
对象. **proto** `===` 构造函数.prototype;