面向对象

面向对象的三大特性

> 面向对象的三大特性 → **封装、继承、多态**

封装

作用 **复用 | 信息隐蔽**

```
//01 封装的简单说明
//001 观察以下杂乱无关的语句,它们仅仅只是一堆无意义的全局变量
var name = "乘风波浪";
var actors = ["彭于晏","赵丽颖","邓超","李荣浩"];
var showTime = "2017-1-28 - 2017-2-28";
var director = "韩寒";
var type = "喜剧";
var play = function () {
    //播放......
};

//002 把上面的变量封装到对象中(更具体&有意义)
var film  = {
    name:"乘风破浪",
    actors:["彭于晏","赵丽颖","邓超","李荣浩"],
    showTime:"2017-1-28 - 2017-2-28",
    director:"韩寒",
    type:"喜剧",
    play:function () {
    //播放......
    }
}
```

继承

作用 获取已经存在的对象已有属性和方法的一种方式(获取他人已有财富和资源的一种方式)。

```
//继承的简单示例
//001 创建一个空的对象
var nullObj = {};

// 上面的对象film拥有了很多的属性和方法
//002 设法让nullObj拥有film中所有的属性和方法
//方法一:自己重新写一遍
//方法二:通过某种方式来获得,比如遍历该对象,然后完成赋值
for(var k in film)
```

```
{
    nullObj[k] = film[k];
}

for(var k in nullObj)
{
    console.log(nullObj[k]);
}
```

多态

多态 : polymorphism = poly(复数) + morph(形态) + ism

作用 多用于强类型语言中,JavaScript具备与生俱来的多态特性。

HTML

多态表现为:
① 同一操作,作用于不同的对象,会产生不同的解释和行为。
② 隐藏不同。

对象的创建

001 字面量方式创建对象

基本写法

```
var  book1 = {
        name:"声名狼藉者的生活",
        price:42.00,
        author:"福柯",
        press:"北京大学出版社",
        read:function () {
            console.log("我的书名为:声名狼藉者的的生活,作者为福柯....");
        }
    };
```

创建说明

[01] 代码复用性差

[02] 如果要创建大量的同类型对象，则需要些大量重复性代码

002 内置构造函数创建对象

JS中的内置构造函数

- String
- Number
- Boolean
- Date

- Array
- Function
- Object
- RegExp

注意 基本包装类型需要区别于 `string | number | boolean`

基本写法

```
var book1 = new Object();
book1.name = "声名狼藉者的生活";
book1.price = 42.00;
book1.author = "福柯";
book1.press = "北京大学出版社";
book1.read = function () {
    console.log("我的书名为:声名狼藉者的的生活,作者为福柯....");
};
```

创建说明

[01] 创建的对象无法复用,复用性差

[02] 如果需要创建多个同类型的对象,如(书籍)则需要写大量重复的代码,代码的冗余度高

003 工厂函数创建对象

基本写法

```
function createBookNew (name,price,author,press) {

    var book = new Object();
    book.name = name;
    book.price = price;
    book.author = author;
    book.press = press;
    book.read = function () {
        console.log("我的书名为:"+book.name+",作者为"+book.author+"....");
    };

    return book;
}

//使用工厂函数来创建对象
var book1 = createBookNew("声名狼藉者的的生活","42.00","福柯","北京大学出版社");
var book2 = createBookNew("人性的枷锁","49.00","毛姆","华东师范大学出版社");
var book3 = createBookNew("悟空传","28.00","今何在","湖南文艺出版社");

//打印对象的属性,调用对象的方法
console.log(book1.name);
console.log(book2.name);
console.log(book3.name);

book1.read();
```

```
book2.read();
book3.read();
```

创建说明

[01] 工厂函数方式创建对象其本质是对内置构造函数创建对象的过程进行了封装。

[02] 适用于大规模“批量生产”同类型的对象。

```
function createBook (name,price,author,press) {

    //001 参数 = 原料
    var book = new Object();

    //002 创建对象并设置对象的属性和方法 = 对原料进行加工
    book.name = name;
    book.price = price;
    book.author = author;
    book.press = press;
    book.read = function () {
        console.log("我的书名为:"+book.name+",作者为"+book.author+"....");
    };

    //003 把处理好的对象返回给我们 == 产品出厂
    return book;
}
```

封装思路 使用函数把固定不变的部分封装起来，变化的部分提取为函数的参数

工厂函数创建对象的实现过程

HTML

① 提供一个创建对象的函数（参数）
② 在该函数内部使用new 关键字和Object构造器创建对象
③ 设置对象的属性
④ 设置对象的方法
⑤ 返回对象

004 自定义构造函数创建对象

基本写法

```
function 构造函数名(参数1,参数2,参数3...) {
    //设置对象的属性
    this.属性01 = 参数1;
    this.属性02 = 参数2;

    //设置对象的方法
    this.方法01 = function () {
        //.....
    };
    this.方法02 = function () {
        //.....
    }
```

```
}

//自定义构造函数方式创建对象
var 对象01 = new 构造函数名(实参01,实参02,实参03...);
var 对象02 = new 构造函数名(实参01,实参02,实参03...);
```

代码示例

```
function CreateBook (name,price,author,press) {
    this.name = name;
    this.price = price;
    this.author = author;
    this.press = press;
    this.read = function () {
        console.log("我的书名为:"+this.name+",作者为"+this.author+"....");
    };
}

var b1 = new CreateBook("声名狼藉者的的生活","42.00","福柯","北京大学出版社");
var b2 = new CreateBook("人性的枷锁","49.00","毛姆","华东师范大学出版社");
var b3 = new CreateBook("悟空传","28.00","今何在","湖南文艺出版社");

//打印对象的属性,并调用对象的方法测试
console.log(b1.author);
console.log(b2.author);
console.log(b3.author);
b1.read();
b2.read();
b3.read();
```

构造函数与new关键字

构造函数的作用	用于完成对象的初始化
new关键字的作用	用于创建对象（Object类型）
构造函数和函数的区别	函数的首字母大写

自定义构造函数和简单工厂函数的对比

HTML

① 函数的首字母大写(用于区别构造函数和普通函数)
② 创建对象的过程是由`new关键字`实现
③ 在构造函数内部会自动的创建新对象,并赋值给`this指针`
④ 自动返回创建出来的对象

> **➥ 构造函数的执行过程**
>
> ① 使用new关键字创建对象
> ② 把新创建出来的对象赋值给 `this`
> ③ 在构造函数内部，使用 `this` 为新创建出来的对象设置属性和方法
> ④ 默认返回新创建的对象(普通函数如果不显示的 `return` 则默认返回 `undefined`)。

> **注意** 简单使用构造函数的方式来创建对象也非完美，因为每次创建对象的时候都会重新创建函数，那么如果创建的对象数量很多，而对象方法内部的实现一模一样就会造成资源浪费。更好的做法是把构造函数和原型对象结合在一起使用。

> **➥ 构造函数的返回值**
>
> ➢ 如果没有return，则默认返回的是新创建出来的对象
> ➢ 如果在构造函数中显示的return，则依照具体的情况处理
>
> - [01] `return 的是对象` ，则直接返回该对象(替换默认对象)。
> - [02] `return 的是null或基本数据类型值` ，则返回新创建的对象。

构造函数和原型对象

构造函数的注意点

函数传值 可以把构造函数的对象方法抽取为参数

```
//001 创建一个构造函数
function Person(name,age,toDoSomeThing) {

    //002 在构造函数内部设置对象的属性和方法
    this.name = name;
    this.age = age;

    this.sayName = function () {
        console.log(this.name);
    };

    this.toDoSomeThing = toDoSomeThing;
}

//003 使用构造函数创建对象
var zhangsan = new Person("张三",18,function () {
    console.log("张三在读书");
});

var lisi = new Person("李四",20,function () {
    console.log("李四在玩耍");
});
```

实例对象的类型

❑ **检查对象的类型** `instanceOf` 和 `typeof`
❑ **获取对象的类型** `Object.prototype.toString.call(instance)`

构造器属性

```
function Dog(name) {
        this.name = name;
        this.color = "黄色";
    }
console.log(dog.constructor);    //Dog函数
//注意：实例对象自身并不拥有constructor属性，该属性总是从原型对象中获取
```

❏ **属性的名称**： constructor
❏ **属性的作用**：指向创建该对象的构造函数，类似于现实生活中所有的产品都标有生产厂家一样

构造函数的调用

01 构造函数可以像普通函数一样不通过new关键字直接调用 。
02 在使用构造函数创建对象的时候，如果没有传递参数，则 () 可以省略 。

```
//01 创建构造函数
function Person() {
    this.name = "张三";
    this.age = 20;
    this.sayName = function () {
        console.log(this.name);
    }
}

//02 使用构造函数创建对象
var p1 = new Person();
var p2 = new Person;     //说明:如果不需要传递参数,则在调用构造函数的时候()可以省略

//备注：this的指向
//01 如果使用new 构造函数的方式调用，则this指向内部默认创建出来的空对象
//02 如果像调用普通函数一样调用构造函数，则this指向全局对象window(不要这样使用)
```

原型对象

原型对象

在构造函数创建出来的时候,系统会默认帮构造函数创建并关联的一个新对象。
自定义构造函数的原型对象默认是一个空对象，Object类型的空对象(类似于 new Object())。

原型对象的作用 构造函数原型对象中的属性和方法可以被使用该构造函数创建出来的对象使用。即以自定义构造函数方式创建出来的所有对象都能够共享该构造函数的原型对象中的所有属性和方法。

> ➥ **访问构造函数的原型对象**
>
> ❏ 构造函数. protoType
> ❏ 对象. __proto__ (ES6)
> ❏ Object.getPrototypeOf(instance)

设置原型对象的属性和方法

① 直接替换原型对象
② 利用对象的动态特性来为构造函数的原型对象添加属性和方法

实例和实例化

实例化	通过构造函数创建具体对象的过程。
实例对象	通过构造函数实例化出来的对象，我们称之为该构造函数的一个实例。

> **注意** 在说实例的时候,一定要指定是某个具体构造函数的实例

原型的使用方法

① 直接替换原型对象
② 利用对象的动态特性给原型添加属性|方法，如果要添加的方法过多,则有大量重复代码

```HTML
01 替换前后创建的对象所指向的原型对象不一致
02 替换原型对象会切断和之前的原型对象之间的关系
```

原型对象的使用注意

```HTML
① 访问属性:构造函数创建出来的对象在访问属性的时候,会先在实例内查找,没有则到原型对象中查找
② 设置属性:
   在使用点语法进行赋值的时候,无法操作到对应的原型对象
                    如果该属性在对象中已经存在,则修改该属性的值
                    如果该属性在对象中尚未存在,则新增该属性
③ 设置原型对象的属性:
   [01] 设置原型对象的属性,只能通过构造函数.Prototype的方式|替换原型对象的方式设置
   [02] 如果原型对象的属性是值类型,那么只能通过Person.prototype.属性的方式修改其值

        如果原型对象的属性是引用类型,那么可以通过对象名.引用对象.属性名的方式设置|修改
          (1) 使用构造函数创建出来的多个对象的原型对象中的该属性指向的是同一块数据
          (2) 某个对象对该原型对象属性进行了修改会影响到其他的对象
```

proto属性说明

`__proto__` 早期是一个非标准属性，现已经被纳入ES6规范。在ES6之前ECMAScript中并不包含该属性，这只是某些浏览器为了方便开发人员开发和调试而提供的一个属性并不具备通用性，在调试代码的时候使用该属性可以更方便的获取原型对象，在正式代码中需要考虑到版本兼容问题。