1.构造函数和原型

- JS ES6之前没有类的概念，大多浏览器对ES6的支持不是很好。
- ES6之前使用构造函数来模拟类，
- 创建对象的可以通过以下三种方式
  1. 对象字面量 `{}`
  2. `new Object()`
  3. 自定义构造函数 配合`new`调用

2. 构造函数注意点：

1. 构造函数用于创建某一类对象，首字母要大写。
2. 构造函数要和`new` 一起使用才有意义。 `new Person();`

- `new`在执行时会做四件事情：
  1. 在内存中创建一个新的空对象。
  2. 让`this`指向这个新的对象。
  3. 执行构造函数里面的代码，给这个对象添加属性和方法。
  4. 返回这个新对象（所以构造函数里面不需要`return`，但是如果显性返回一个对象，就不会返回这个对象）

3. 构造函数的 静态成员 和 实例成员

- **构造函数中的 属性 和 方法 我们称为成员，成员可以添加**
- **静态成员**：在构造函数上添加的成员，只能由构造函数本身来访问

```
Person.sex = '男';
```

  - `sex`就是静态成员，只能通过`Person`构造函数来访问
- **实例成员**：在构造函数内部通过`this`添加的成员，只能由实例化的对象来访问

```
function Person(){
  this.name = 1;
}   // 这里的name就是实例成员
```

4.构造函数原型对象 prototype

- 构造函数的问题
  - 存在**浪费内存**的问题，每创建一个实例对象，对象中的方法(函数)都会开辟一块新空间
  - 希望所有的实例对象都使用同一个函数(方法),这样就比较节省空间
- 构造函数通过原型分配的函数(方法)是所有实例对象共享的。
- 我们每创建一个函数，都有一个`prototype`(原型)属性，这个属性是一个指针，这个属性指向函数的原型对象，原型对象包含特定类型所有实例共享的属性和方法。
- 可以把不变的方法，直接定义在`prototype`对象上，这样所有对象的实例，就可以共享这些方法。

- 一般情况下，我们的**公共属性定义到构造函数里面，公共的方法放到原型对象身上。**

5.对象原型 proto

- 对象都会有一个属性 `__proto__` 指向构造函数的 `prototype` 原型对象，之所以我们对象可以使用构造函数 `prototype` 原型对象的属性和方法，就是因为对象有 `__proto__` 原型的存在
- `对象的.__proto__` == `构造函数.prototype`；他俩都指向同一个原型对象

```
console.log(ldh.__proto__ === Star.prototype);  //true
```

- 方法的查找规则：首先看 实例对象 上是否有 `sing` 方法，如果有就执行这个对象上的 `sing`，如果没有，因为有 `__proto__` 的存在，就去构造函数原型对象 `prototype` 身上去查找 `sing` 这个方法
- 实例对象可以使用 构造函数的原型对象 `prototype` 方法的原因
  - `Star` 构造函数的公共方法放在 `prototype` 原型对象上
  - `Star` 构造出来的对象实例，会有一个对象原型 `__proto__` 指向 `prototype`。

- `__proto__` 对象原型的意义就在于为对象的查找提供了一个方向/线路，但是它是一个非标准属性，**实际开发中，不可以使用**

6.constructor 构造函数

- 构造函数原型对象(`prototype`)里面都有一个属性 `constructor` 属性，`constructor` 我们称之为构造函数，因为他**指回构造函数本身。**
- `JS` 中的 `constructor` 是原型对象 `prototype` 上的一个属性（如果该对象是实例对象，则在其 `__proto__` 上,因为实例对象的 `__proto__` 指向构造函数原型对象），通常指向生成这个对象的构造函数。

- 构造函数的原型对象指向构造

```
console.log(Star.prototype.constructor.name);   //Star
console.log(Star.constructor.name)      //Function
```

- 如果构造函数原型对象 `prototype` 要添加的方法很多，可以用对象字面量的形式添加，之后使用 `constructor` 手动指回原来的 构造函数

```
Star.prototype = {
  //constructor会被覆盖掉，要手动指回原来的 构造函数
  constructor: Star,
  sing: function(){}
  movie: function(){}
}
```

7.构造函数、实例、原型对象三者之间的关系

- 构造函数

    - 属性：`prototype`
        - `prototype`指向`prototype`原型对象
- `prototype`原型对象
    - 属性：`constructor`
        - `constructor` 指向构造函数
- 实例对象
    - 属性：`__proto__`
        - `__proto__`指向`prototype`原型对象

8.原型链

- 对象就会拥有`__proto__`原型，指向原型对象
- 函数就会拥有`prototype`

1. 只要是对象 就有`__proto__`原型，指向原型对象(`prototype`原型也是一个对象)

```
console.log(Star.prototype.__proto__);
console.log(Star.prototype.__proto__ === Object.prototype); //true
```

2. 我们`Star`原型对象里面的`__proto__`原型指向的是`Object.prototype`
    - 构造函数的`prototype`和实例对象的`__proto__` 都指向构造函数原型对象
3. `Object.prototype`原型对象的`__proto__`指向 为 空`null`

```
console.dir(Object.prototype.__proto__) //null
console.log(Object.prototype.__proto__ === null);  //true
```

    - 所有的系统构造函数应该都是由`Object`构造函数构造来的

9.JS的成员查找机制

- 构造函数中的属性和方法我们称为成员，成员可以添加

1. 当访问一个对象的属性 (包括方法) 时，首先查找这个对象自身有没有该属性
2. 如果没有查找到，就查找它的原型(也就是`__proto__`指向的`prototype`原型对象)
3. 如果还没有查找到，就查找原型对象的原型(`Object`的原型对象)。
4. 以此类推一直找到`Object`为止(`null`)
5. _`_proto__`对象原型的意义就在于为对象成员查找机制提供一个方向 /路线

10.原型对象this指向

- `this`指向的两条原则
    1. 只有调用函数时才能确定`this`的指向。
    2. 一般情况下`this`指向函数的调用者

```
var that;
Star.prototype.sing = function(){
  console.log("我会唱歌");
  that = this;
}
var ldh = new Star("刘德华",18,"女");

ldh.sing();
console.log(ldh === that);  //true
```

1. 在构造函数中的`this`指向的是对象实例 `ldh`
2. 原型对象函数里面的`this` 指向的是 实例对象`ldh`(因为是`ldh`调用的)

11.扩展内置对象

- 可以通过原型对象，对原来的内置对象进行自定义的方法，比如给数组增加自动求偶数和的功能
- 对于原来存在的内置对象(`Array`,`Number`等)，不要通过字面量给原型对象添加方法，会覆盖原来的方法和属性。

12 .继承

- ES6之前并没有给我们提供 `extends` 继承。我们可以通过 **构造函数+原型对象** 模拟实现继承，被称为**组合继承**。

1. `call()`;
   - `show.call("call",10,20);` ： `this`指向 `"call"` 字符串
2. 借用构造函数继承父类型属性
   - 核心原理： 通过 `call()` 把父类型的`this`指向 子类型的`this`,这样就可以实现子类型继承父类型的属性

```
// 1. 父构造函数
function Father(uname,age){
  // this 指向父构造函数的对象实例
  this.uname = uname;
  this.age = age;
}

//2. 子构造函数
function Son(name, age, score){
  //this 指向子构造函数的对象实例
  this.score = score;

  Father.call(this, name, age);
  // 将父构造函数的this指向子构造函数的this
  // 注意传参
}
```

3. 借用原型对象继承父类型方法
   - 错误的方法

```
Son.prototype = Father.prototype;
```

- 这种方法 就是让子原型对象拥有了 父原型对象的指针,修改子原型对象，就相当于修改父原型对象，不可取

- 正确的方法

```
Son.prototype = new Father();
```

- 如果利用了对象的形式修改了原型对象，别忘了利用`constructor`指回原来的构造函数

```
Son.prototype = new Father(); //这时的`Son.prototype`指向`Father`
Son.prototype.constructor = Son;// 用`constructor`指回`Son`构造函数。
```