

面向对象（一）

对象的概念

上面是通过语义的角度来理解什么是面向对象

如果它是一个对象，那么它应该具备 以下几个特点

1. 对象具备属性

属性就是用于描述当前对象的特征的，可以理解为上在面的【数据】

2. 对象具备方法

方法就是一个对象的能力，例如它可以做什么事情。可以理解为上面的【能力】

3. 对象应该是可以继承的

父级对象里面的某些方法或属性或以在子级对象里面，继续使用

🤔思考：为什么需要对象，如果没有对象会怎么办？

现在我想描述我们班同学的信息，应该怎么办呢

```
1 //颜一鸣同学的相关信息
2 var stuName = "颜一鸣";
3 var sex = "男";
4 var age = 18;
5 var hobby = "看书, 睡觉";
6
7 //现在我想描述一下另一位同学 曹慧的信息 怎么办
8 var stuName2 = "曹慧";
9 var sex2 = "女";
10 var age2 = 18;
11 var hobby2 = "做饭, 逛街";
```

在没有面向对象的情况下，如果我们想形容一些对象的数据【属性】我们必须定义大量的变量，我们现在迫切需要有一种方式来进行集中式的数据管理

对象创建

对于同学们来说对象同学们已经接触过了，之前的数组就是对象。在JavaScript里面，对象的创建也叫对象的定义，它有很多方式

通过 Object 来创建

```
1 //创建了一个对象
2 var obj1 = new Object();
3 //我现在把这个对象当成颜一鸣，去描述颜一鸣的数据
4 obj1.stuName = "颜一鸣";
5 obj1.sex = "男";
6 obj1.age = 18;
7 obj1.hobby = "看书, 睡觉";
```

这种方式去创建对象是非常简单的，它直接通过 `new Object()` 就可以得到一个空的对象，然后在这个空的对象上面赋值属性就可以了

通过字面量 {} 来创建

这种创建方式与之前数组的创建方式很相似，我们在讲数组的时候我们说过 `new Array()` 就相当于 `[]`，所以 `new Object()` 就相当于 `{}`

语法格式

```
1 var 对象名 = {
2     属性名1:属性值1,
3     属性名2:属性值2
4 }
```

通过上面的语法，可以得到下面的结果

```
1 var obj1 = new Object();
2 obj1.stuName = "颜一鸣";
3 //等价于
4 var obj2 = {};
5 obj2.stuName = "曹慧";
```

使用字面量创建的时候还可以使用下面的方式创建

```

1  var obj1 = new Object();
2  obj1.stuName = "颜一鸣";
3  obj1.sex = "男";
4  obj1.age = 18;
5  obj1.hobby = "看书, 睡觉";
6
7  //我们可以直接在花括号里面把所需要的属性写进去
8  var obj2 = {
9      stuName: "曹慧",
10     sex: "女",
11     age: 18
12 };
13 obj2.hobby= "做饭, 逛街";

```

对象属性的调用方式

通过上面的学习, 我们知道对象里面是有数据, 这些数据叫属性, 一个对象如何去调用性呢。这里有2种方式给大家介绍一下

通过 `.` 的方式来调用

正常情况一下, 通过 `对象.属性` 就可以进行调用了, 如下所示

```

1  var obj1 = {
2      stuName: "颜一鸣",
3      sex: "男",
4      age: 18,
5      hobby: "看书, 睡觉"
6  }
7
8
9  //打印obj1的姓名
10 console.log(obj1.stuName);
11 console.log(obj1.sex);
12 console.log(obj1.age);
13 console.log(obj1.hobby);

```

注意: 这一种方式去调用属性的时候只能调用常规的属性, 对于特殊的属性则不能通过 `.` 的方式来调用, 要使用 `[]` 来调用



通过 `[]` 的方式来调用

中括号的调用方式其实同学们之前已经接触过了, 就是数组里面的索引, 数组里面的索引其实也算是一个属性, 所以我们在调用数字的属性的时候我们使用了 `[]`

对于特殊的属性, 我们是不能够使用 `.` 的试来调用的, 如下所示

```
1  var obj1 = {
2      "stu-name": "颜一鸣",
3      sex: "男",
4      age: 18,
5      0:"hello"
6  }
7  //所有的属性都可以通过[]来调用
8  console.log(obj1.sex);
9  console.log(obj1["sex"]);
10 // console.log(obj1.0); //报错
11 console.log(obj1[0]);
12 console.log(obj1["stu-name"]);
```

> obj1

<  {0: 'hello', stu-name: '颜一鸣', sex: '男', age: 18} 
 0: "hello"
 age: 18
 sex: "男"
 stu-name: "颜一鸣"
 ▶ [[Prototype]]: Object

说明: [] 中括号调用属性的方式其实是可以调用所有的属性的

总结: 所有的属性都可以通过 `[]` 来进行调用, `.` 点这种方式只能调用常规属性

变量做为属性名

```
1  var a = "hello";
2  var obj1 = {
3      stu: a          //上面的a就是一个变量a
4  }
5
6  var obj2 = {
7      a: "world"      //这里的a就是一个普通的字符串a
8  }
9
10 //有没有一种办法把a用变量的形式放在属性当中
11 var obj3 = {
12     [a]: "world"     //这里的a加了中括号，代表的就是变量a的值
13 }
```

> obj1

< ▶ {stu: 'hello'}

> obj2

< ▶ {a: 'world'}

> obj3

< ▶ {hello: 'world'}

在上面的学习的2种方式里面，我们都已经知道了怎么样去创建对象，对象可以把某些数据集中管理。但是我们仍然面对一个问题，如果我现在需要创建很多个对象，怎么办？我现在想把我们班65位同学们的信息全部都用对象表示，这又怎么办？如果我们还是使用原来的方式去创建，这样我们就要把所有的学生的属性都一一的输入一遍，这样非常不好，效率太低了

我们现在需要一种技术，快速的创建对象

使用工厂模式创建对象

一谈起工厂，脑海之中不自觉的就会想起批量生产，并且生产的东西应该都是相同的。工厂模式其实就是批量生产对象，并且生产相同的对象

```

1  // 现在我要批量的创建相同类型的对象
2  function createStudent(_name,_sex,_age) {
3      var obj = {
4          name: _name,
5          sex: _sex,
6          age: _age
7      }
8      return obj;
9  }
10
11 var stu1 = createStudent("李心悦","女",17);
12 var stu2 = createStudent("颜一鸣","男",20);

```

在上面的工厂模式里面，我们可以快速的去创建对象，并且对象的格式也是一样的，这样就解决了我们需要创建重复对象的问题

函数 `createPerson()` 能够根据接受的参数来构建一个包含所有必要信息的 `Person` 对象。可以无数次地调用这个函数，而每次它都会返回一个包含三个属性一个方法的对象。工厂模式虽然解决了创建多个相似对象的问题，但却没有解决对象识别的问题（即怎样知道一个对象的类型）。随着 JavaScript 的发展，又一个新模式出现了。

```

1  // 现在我要批量的创建相同类型的对象
2  //专门生产学生的工厂
3  function createStudent(_name, _sex, _age) {
4      var obj = {
5          name: _name,
6          sex: _sex,
7          age: _age
8      }
9      return obj;
10 }
11
12 var stu1 = createStudent("李心悦", "女", 17);
13 var stu2 = createStudent("颜一鸣", "男", 20);
14
15 //专门生产牛的工厂
16 function createCow(_name,_sex,_age) {
17     var obj = {
18         name: _name,
19         sex: _sex,
20         age: _age
21     }
22     return obj;
23 }
24
25 var cow1 = createCow("小牛1号","母",1);
26 var cow2 = createCow("公牛2号","公",2);

```

在上面的代码里面，我们有2个工厂，一个是专门生产学生的对象的，一个是专门生产牛的对象，但是这两个对象最终所表现出来的类型竟然是相同的

```
> cow1
< ▶ {name: '小牛1号', sex: '母', age: 1}
> cow2
< ▶ {name: '公牛2号', sex: '公', age: 2}
> stu1
< ▶ {name: '李心悦', sex: '女', age: 17}
> stu2
< ▶ {name: '颜一鸣', sex: '男', age: 20}
```

```
1  typeof stu1;           //"object"
2  typeof cow1;           //"object"
3  //之前也给大家讲过，基本数据类型才使用`typeof`，而复杂数据类型使用`instanceof`
4  //现在我们使用`instanceOf`来检测
5  stu1 instanceof Object; //true
6  cow1 instanceof Object; //true
```

通过上面的代码我们可以看到，我们不能够去识别工厂模式创建的对象类型，无论是通过 `typeof` 或 `instanceof` 都不能区分

使用构造函数创建对象【重点】

重要的事情说三遍，这个章节是重点，重点，重点

要弄清楚构造函数就首先先弄清楚函数

函数之前我们已经学过了，就是一个通过 `function` 去定义的东西，同时我们也学习了函数的调用方式，函数是通过函数名+()的形式去调用的

现在我们来讲函数的另一种调用方式

```
1  var arr = new Array();
2  // 这会创建一个数组对象
3  var obj = new Object();
4  // 为什么new一个Object也会得到对象?
```

在上面的代码里面，我们就要思考一下，为什么 `new` 一个东西就会得到对象，同时 `new` 的这个东西又是什么？

现在我们就来看一下 `Array` 与 `Object` 到底是个什么东西

```
1 console.log(typeof Array);           //function
2 console.log(typeof Object);          //function
```

我们现在可以看到 `Array` 与 `Object` 都是一个 `function` 函数，所以我们知道了一点 **new 一个函数就会得到一个对象**，`Array` 与 `Object` 就是系统当中内置的构造函数

所谓的构造其实就是通过 `function` 关键字定义的普通函数，只是它的调用方式不一样而已。**如果一个函数使用了 `new` 关键字去调用，那么这个函数就叫构造函数**

```
1 function abc(){
2     console.log("abc");
3     return 123;
4 }
5
6 var x = abc();           //普通函数    返回值x 123
7 var y = new abc();       //new调用的,构造函数  返回了一个对象
```

通过上面的代码我们可以发现，`new` 应该就是调用一个函数，然后再拿到对象

```
1 var 对象 = new 函数();
```

? 问题：为什么用 `new` 去调用的函数就叫构造函数

如果我们现在想通过上面的 `new 函数()` 的方式来创建对象，那么我们必须要知道构造函数是怎么调用的，而构造函数的调用主要点还是在 `new` 上面，所以我们需知道 `new` 到底干了什么事情

要创建 `Person` 的新实例，必须使用 `new` 操作符。以这种方式调用构造函数实际上会经历以下 4 个步骤：

- (1) 创建一个新对象；
- (2) 将构造函数的作用域赋给新对象（因此 `this` 就指向了这个新对象）；
- (3) 执行构造函数中的代码（为这个新对象添加属性）；
- (4) 返回新对象。


```

1 // 现在我要创建一个学生的对象
2 function Student(_name,_age) {
3     // console.log(this);
4     // this就是新创建的对象stu1
5     this.name = _name;
6     this.age = _age;
7     // return this;
8 }
9
10 var stu1 = new Student("颜一鸣",18);           //第一个对象
11 var stu2 = new Student("李心悦",20);           //第二个对象

```

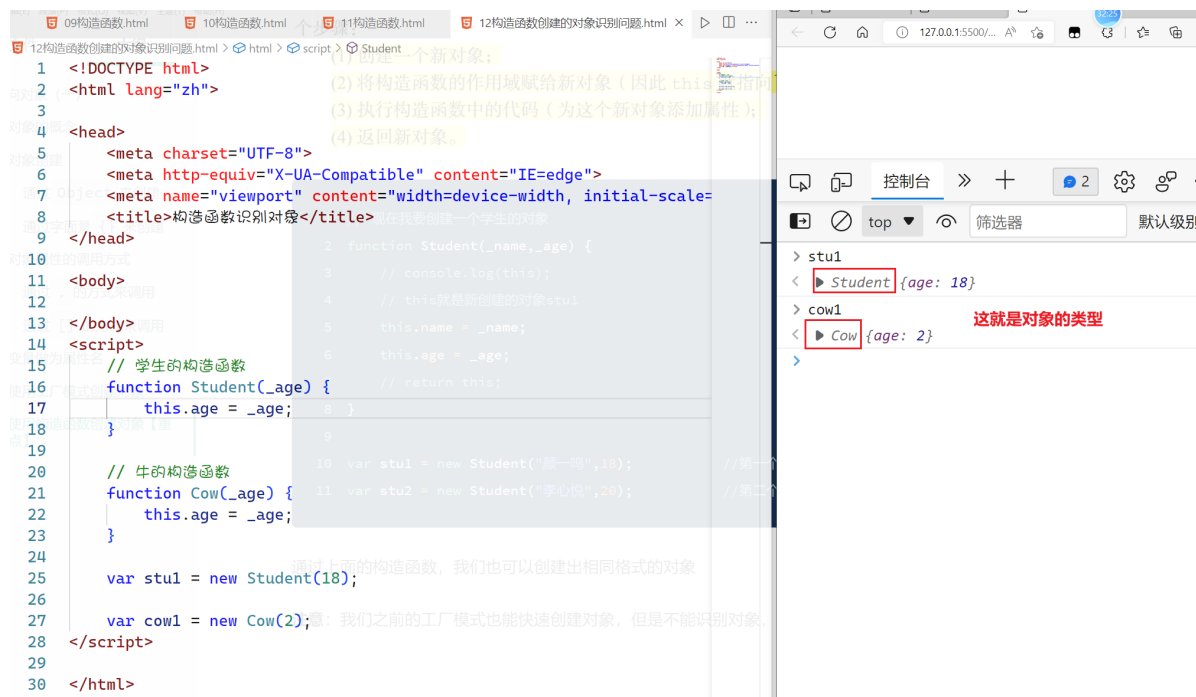
通过上面的构造函数，我们也可以创建出相同格式的对象

注意：我们之前的工厂模式也能快速创建对象，但是不能识别对象，这个时候的构造函数是否可以识别对象呢

```

1 // 学生的构造函数
2 function Student(_age) {
3     this.age = _age;
4 }
5
6 // 牛的构造函数
7 function Cow(_age) {
8     this.age = _age;
9 }
10
11 var stu1 = new Student(18);
12 var cow1 = new Cow(2);
13
14 console.log(stu1 instanceof Object);           //true
15 console.log(cow1 instanceof Object);           //true
16 console.log("-----");
17 //现在开始真正的识别
18 console.log(stu1 instanceof Student);          //true
19 console.log(cow1 instanceof Cow);               //true
20 console.log(stu1 instanceof Cow);               //false
21 console.log(cow1 instanceof Student);          //false

```



这个时候我们再去使用 `instanceof` 去检测的时候就可以识别对象的类型了，就可以把2个对象区分开了

构造函数与普通函数的区别

构造函数与普通函数在定义上面是没有任何区别的，关键是看他的调用形式

1. 一个函数如果以 `new` 去调用那么它就是构造函数，如果只是通过 `函数名+()` 这种形式调用它就是普通函数
2. 普通函数的返回值是通过 `return` 来完成的，而构造函数的返回值是**自动返回**的

构造函数的返回值如果内部返回的是一个基本数据类型则不生效，它会自动返回构造函数创建的对象

构造函数的返回值如果内部返回的是一个对象，则放弃构造函数自动创建的对象，以`return`为主

```
1 function Student(name){
2   this.name = name;
3   return 123;           //123是基本数据类型，所以构造函数不要
4 }
5
6 var x = Student("张三"); //普通函数调用，所以x接收的就是return返回的结果
7 var y = new Student("李四"); //构造函数调用，默认返回当前创建的对象
```

注意： 这里有个天大的坑

```

1  function Student(name) {
2      this.name = name;
3      var arr = ["a", "b", "c"];
4      //arr是一个数组，数组是对象
5      return arr;          //这里返回的是一个对象
6  }
7
8  var x = Student("张三");    //普通函数调用    接收return的结果 arr
9  var y = new Student("李四"); //构造函数调用    它到底接收什么

```

在上面的代码里面，因为 `Student` 的函数内部返回的是一个对象 `arr` 数组，所以在构造函数进行 `new` 的返回的时候，它会放弃自己创建的对象，而返回 `arr` 给外边

构造函数并不会接收 `return` 的基本数据类型的返回值，但是如果你返回的是一个对象，它那我要了（这个时候就不再返回默认的构造函数创建的对象）

3. 构造函数里面的 `this` 指向了当前构造函数所创建的对象，而普通函数里面的 `this` 指向了浏览器的全局对象 `window` 【这个知识点在后面的DOM里面会讲到】
4. 普通函数在调用的时候是需要通过 `函数名+()` 来调用，而构造函数如果不需要传递参数，则可以省略括号 `()`

```

1  var arr = new Array;
2  var obj = new Object;
3
4  function Student(){
5      this.name = "张三";
6  }
7
8  Student();    //普通函数
9  var stu1 = new Student;    //构造函数调用

```

在构造函数里面，它的 `()` 主要是为了传递参数的，如果我们不需要传递参数，则这个小括号 `()` 可以省略不写

约定俗成：构造函数的函数名首字母大写，而普通函数的首字母是小写的，这不是规范，但这是约定。在系统当中如果你要是发现一个函数名大小了，则它肯定是构造函数

保证函数以构造的方式执行

```

1  //我们想把下面的函数定义成构造函数
2  function Student(name) {
3      // 如何保证当前的函数只能被当成构造函数调用
4      if (new.target === Student) {
5          this.name = name;
6      }
7      else{
8          console.error("当前的函数只能以构造函数执行");
9      }
10 }
11 //Student("张三");           //普通限制
12 var x = new Student("李四");  //构造函数调用

```

对象中的方法

每个对象上面或多或少的都具备一些能力，这些能力其实说法是对象上面的方法（方法就是函数，在面向过程的编程里面函数，在面向对象里面的叫方法）

Object的方式

```

1  var stu1 = new Object();
2  stu1.name = "江海丽";
3  stu1.sayHello = function(){
4      console.log("你好啊,我叫: " + this.name);
5  }

```

字面量的方式

```

1  var stu2 = {
2      name:"曹慧",
3      sayHello:function(){
4          console.log("hello world! My name is " + this.name);
5          // console.log(this === stu2);
6      }
7  }


```

构造函数创建对象里面的方法

```

1  function Student(name) {
2      this.name = name;
3      // this指向新创建的对象
4      this.sayHello = function(){
5          console.log("我是一个曹同学,我的姓名是: " + this.name);
6      }
7  }
8
9  var stu3 = new Student("曹方");

```

 **小技巧:** 在对象方法的内部, `this` 是指向当前这个对象的, 如果想在方法里拿到自己的某一个属性, 则可以通过 `this` 来调用

基础篇总结

1. 面向对象的概念是什么?

面向结果的编程方式, 集中式的数据管理方式, 高内聚的特征, 集中数据的这个过程就是封装对象的过程

2. 对象创建的几方式是什么? 最常见的方式有哪些?

- `new Object()` 创建对象【不常用】
- 通过 `{}` 花括号创建【很常见】

上面的两种试都只是适合创建少量的对象, 如果批量则不适用了

- 工厂模式【不常用】
- 构造函数的创建【很常用】

3. 构造函数与普通函数的区别在哪里?

- 调用方式的不同, 特别是 `new`
- 返回值的情况不同
- `this` 指向会不同
- 通过 `new` 调用的构造函数可以省略 `()`

4. 对象调用属性或方法的方式

```

1  //[]是可以调用任何属性
2  var stu1 = {
3      userName: "江海丽",
4      sayHello: function () {
5          console.log("大家好,我叫" + this.userName);
6      }
7  }
8  stu1.sayHello();
9
10 //stu1.sayHello === stu1["sayHello"];
11 stu1["sayHello"]();
12
13
14 var arr = ["标哥哥"];
15 arr.push("江海丽");

```

```
16
17 arr["push"]("曹慧");
18 //arr["push"] === arr.push
19 //arr["push"]("曹慧")===arr.push("曹慧")
```

5. new 到底干了什么事情？【new干的4件事情】
6. 记住 new.target 保证函数以构造函数方式执行