[笔记][LIKE-JS][4-对象和面向对象(初级)]

`JavaScript`

---

054. JS基础-对象-字面量对象(掌握)

`JavaScript` 是基于对象的语言。
无法创建自定义的类型，不能很好的支持继承和多态。

`JavaScript` 的对象是 无序属性的集合。

- 其属性可以包含基本值、对象或函数。
- 对象就是一组没有顺序的值。
- 我们可以把 `JavaScript` 中的对象想象成键值对，其中值可以是数据和函数。

对象的行为和特征

- 特征：属性
- 行为：方法

```
<script>
    var dog = {
        name: '旺财',
        age: 10,
        dogFriends: ['阿黄', '飞毛腿'],

        eat: function() {
            console.log('吃');
        },
        run: function() {
            console.log('跑');
        }
    };
    console.log(dog);
    console.log(dog.name);
    dog.run();
    dog.name = '张三';
    console.log(dog.name);
</script>
```

---

055. JS基础-对象-JSON(掌握)

数据交换格式：

- `xml`：体积大
- `json`：体积小，效率高，用的多

什么是 `json`

- `JavaScript Object Notation`（`JavaScript` 对象表示形式）
- `JavaScript` 的子集

`json` 和对象字面量的区别

- `JSON` 的属性必须用双引号引起来，对象字面量可以省略。
- `JSON` 本质上是一个数据交换格式。
  有两种结构：对象和数组。两种结构相互组合从而形成各种复杂的数据结

构。（可以层层嵌套）

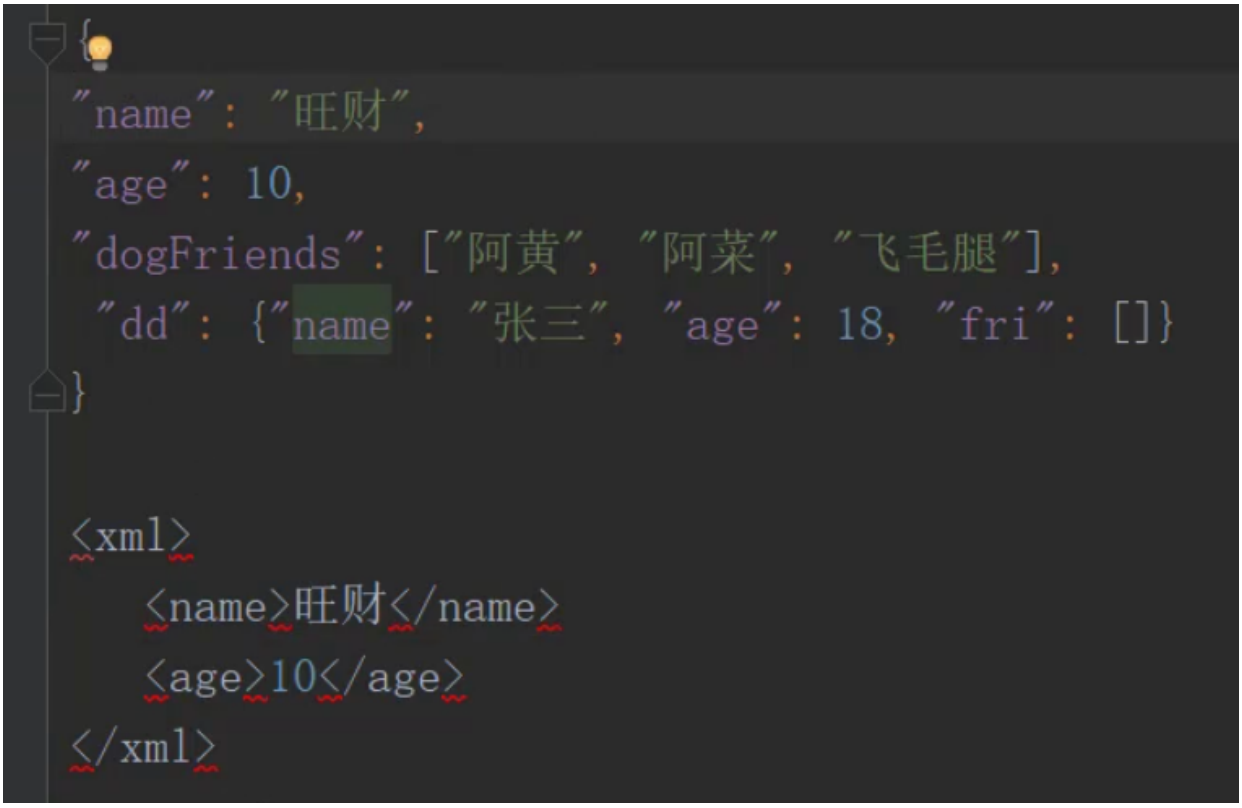
对象的遍历

```
for (var key in 对象) {
    console.log(对象[key]);
}
```

---

056. JS基础-构造函数产生对象-方式1(掌握)

> **注意**：`JSON` 里面键和值都要用双引号！



普通函数不能产生对象，构造函数可以产生对象。
`new` 后面调用函数，我们称为构造函数。
其他语言中的类可以理解为 `JSON` 中的构造函数。
`new Object()`

`Object()` 就是一个构造函数，本质就是函数，只不过构造函数的目的是为了创建新对象，为新对象进行初始化。

构造函数名，首字母大写。

```
<script>
    // 1. 构造函数
    function Person() {
        var obj = new Object();
        // 属性
        obj.name = null;
        obj.age = null;
        obj.sex = null;

        // 方法
        obj.study = function () {
            console.log(this.name + '学习');
        };

        return obj;
    }

    var p = Person();
    console.log(p);
</script>
```

---

057. JS基础-构造函数产生对象-方式2(掌握)

- `this` 所在的函数在哪个对象中，`this` 就代表这个对象
- 谁调用 `this` 就是谁
- 构造函数中的 `this` 始终是 `new` 当前对象。

```
<script>
    // 1. 构造函数
    function Dog(name, age, dogFriends) {
```

```
        this.name = name;
        this.age = age;
        this.dogFriends = dogFriends;

        this.eat = function (something) {
            console.log(this.name + '在吃' + something);
        }

    }

    var smallDog = new Dog('小花', 1);
    console.log(smallDog.name);
    smallDog.eat('肉骨头');

</script>
```

058. JS基础-构造函数-prototype(掌握)

通过对象传参

```
<script>
    // 1. 构造函数
    function Dog(option) {
        this.name = option.name;
        this.age = option.age;
        this.dogFriends = option.dogFriends;

        this.eat = function (something) {
            console.log(this.name + '在吃' + something);
        }

    }

    var bigDog = new Dog({name: '旺财', age: 5});
    console.log(bigDog);
</script>
```

构造器 `constructor` 和原型属性 `prototype`

- 在任何一个对象中都有构造器和原型属性，包括原生的对象
- `constuctor` 返回创建此对象的构造函数
- `prototype` 让我们有能力动态地给对象添加属性和方法

```
<script>
    var arr = [];
    var obj = {};

    var arr1 = new Array();
    var obj1 = new Object();
</script>
```

构造函数 `Array` 本身就是一个对象。

```
<script>
    Array.prototype.eat = function () {
        alert('我会吃东西');
    };

    var arr = new Array();
    arr.eat();
</script>
```

> `Python` 中拿到类名就可以动态添加属性和方法了。

---

059. JS基础-构造函数-完善构造函数(掌握)

```
<script>
```

```
    console.log(Array.constructor);  // function Function
()
    var arr = [];
    console.log(arr.constructor);  // function Array()
</script>
```

通过 `this` 来指定的属性和方法，是每个对象都有一份的。
所以共享的属性、方法，需要使用 `prototype` 指定。

```
<script>
    function Dog(option) {
        this.name = option.name;
        this.age = option.age;
        this.dogFriends = option.dogFriends;
    }
    Dog.prototype.eat = function(something) {
        console.log(this.name + '在吃' + something);
    };

    // 或者这么写
    Dog.prototype = {
      run: function() {
          console.log(this.name + '在跑');
      }
    };

    mydog = new Dog({name: '叮咚', age: 2});
    console.log(mydog);
    mydog.run();
    mydog.eat('饼干');  // 这里会报错，因为后面的 prototype 把
前面的覆盖了
</script>
```

还有一种写法：

```
<script>
    function Dog(option) {
        this._init(option);
    }
```

```
    Dog.prototype = {
        _init: function(option) {
            this.name = option.name;
            this.age = option.age;
        },
        eat: function(sth) {
            console.log(this.name + '在吃' + sth);
        }
    };
</script>
```

---

060. JS基础-数据类型分析(掌握)

简单数据类型/基本数据类型
复杂数据类型/引用数据类型

区别：在内存中的存储方式不同。

栈区（stack）：由编译器自动分配释放，存放函数的参数值，局部变量的值。
堆区（heap）：一般由程序员分配释放，若开发者不释放，程序结束时可能由`OS`回收。

栈区存放体积小，可以快速读取的数据。
堆区存放体积大，读取较慢的数据。
类似于内存和硬盘的关系。

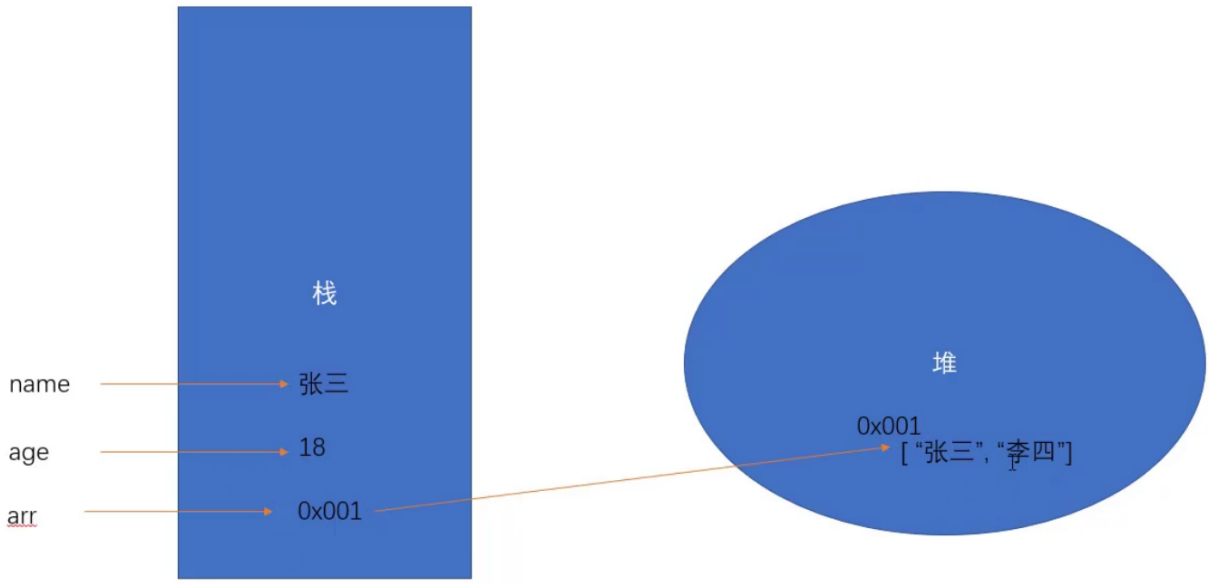
简单数据类型

- `Number`
- `String`
- `Boolean`
- `Undefined`
- `Null`

复杂数据类型

- `Object`

- `Array`
- `Date`



061. JS基础-数组的高级API-上(掌握)

`js` 已经帮我们写好的对象就是内置对象。
常见的内置对象：`Date`、`Array`、`Math`、`RegExp`、`Error`、`String`

`Array` 高级 `API`：

- `sort`：从小到大排序，升序，会改变原数组

  > `Python` 中用法相同

升序 `a-b`，降序 `b-a`，内部用的是冒泡排序

```
<script>
    var arr = [1, 4, 2, 8, 5, 7];
    console.log(arr);

    var arr = [1, 4, 2, 8, 5, 7];
    console.log(arr.sort(function (a, b) {
        return a - b;
    }));
```

```
        var arr = [1, 4, 2, 8, 5, 7];
        console.log(arr.sort(function (a, b) {
            return b - a;
        }));
    </script>
```

- `slice`
  接收一个或者两个参数。
  两个参数：左闭右开。
  一个参数：左闭到末尾。
  也可以传负号，负号会加上一个数组的长度

---

062. JS基础-数组的高级API-下(掌握)

- `splice`
  删除、插入和替换，改变原数组
  删除：两个参数，要删除的第一项，和要删除的项数
  插入：起始位置、0（要删除的项数）、要插入任意数量的项。
  替换：起始位置、要删除的项数、要插入任意数量的项。

清空数组：

- `arr = [];`
- `arr.splice(0);` （第二个参数不写，默认删除后面所有的东西）

`forEach`

```
    <script>
        var arr = [1, 3, 5, 7];
        arr.forEach(function (value, index) {
            console.log(index + ':' + value);
        })
    </script>
```

`map`

```
    var arr = [1, 3, 5];
    var arr2 = arr.map(function (value, index) {
        return value * value;
    });
    console.log(arr2);
```

`filter`

```
<script>
    var arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11];
    var newArr = arr.filter(function (value, index) {
        return index % 3 === 0 || value >= 9;
    });
    console.log(newArr);
</script>
```

`every`：所有项满足，返回 `true`

> 类似 `Python` 的 `all`

`some`：有一项满足，返回 `true`

> 类似 `Python` 的 `any`

```
<script>
    var arr = [1, 2, 3, 4, 5];
    var result = arr.every(function (value, index) {
        return value < 10;
    });
    console.log(result);  // true

    var arr = [1, 2, 3, 100];
    var result = arr.some(function (value, index) {
        return value > 10;
    })
    console.log(result);  // true
</script>
```

完成于 `20190226`