

校企合作项目记录表（跟岗学习专用）

| | | |
|------------|---|---------|
| 记录时段 | 2019 年 11 月 25 日 —— 2019 年 11 月 30 日 | |
| 企业名称 | 广东元心科技有限公司 | |
| 记录类型 | 学习记录 | |
| 学生姓名 | 苏俊贤 | 指导老师签名： |
| 记录内容（可另页附） | <div>JavaScript 原型和原型链</div> <div>一. 原型 prototype 和__proto__ 每个实例对象都有一个__proto__属性，并指向它的 prototype 原型对象。 每个构造函数都有一个 prototype 原型对象，prototype 原型对象里的 constructor 指向构造函数本身。 关系如下图：</div> <div><pre>graph TD; C[构造函数] -- "prototype" --> PO[原型对象]; PO -- "constructor" --> C; C -- "new" --> IO[实例对象]; IO -- "__proto__" --> PO;</pre><p>The diagram illustrates the JavaScript prototype chain. It features three yellow rounded rectangular boxes: '构造函数' (Constructor) on the left, '原型对象' (Prototype Object) at the top right, and '实例对象' (Instance Object) at the bottom right. A dashed arrow labeled 'prototype' points from the Constructor to the Prototype Object. A dashed arrow labeled 'constructor' points from the Prototype Object back to the Constructor. A dashed arrow labeled 'new' points from the Constructor to the Instance Object. A dashed arrow labeled '__proto__' points from the Instance Object up to the Prototype Object.</p></div> <div>代码示例：</div> | |

```
function Person(nick, age){
    this.nick = nick;
    this.age = age;
}
Person.prototype.sayName = function(){
    console.log(this.nick);
}

var p1 = new Person('Byron', 20);

var p2 = new Person('Casper', 25);

p1.sayName() // Byron

p2.sayName() // Casper

p1.__proto__ === Person.prototype //true

p2.__proto__ === Person.prototype //true

p1.__proto__ === p2.__proto__ //true

Person.prototype.constructor === Person //true
```

二. 原型链

定义一个数组 arr

```
var arr = [1,2,3]

arr.valueOf() // [1, 2, 3]
```

控制台打印如下：

```
> console.dir(arr)
```

VM1240:1

```
▼ Array(3) ⓘ
  0: 1
  1: 2
  2: 3
  length: 3
  ▼ __proto__: Array(0)
    ▶ concat: f concat()
    ▶ constructor: f Array()
    ▶ copyWithin: f copyWithin()
    ▶ entries: f entries()
    ▶ every: f every()
    ▶ fill: f fill()
    ▶ filter: f filter()
    ▶ find: f find()
    ▶ findIndex: f findIndex()
    ▶ forEach: f forEach()
    ▶ includes: f includes()
    ▶ indexOf: f indexOf()
    ▶ join: f join()
    ▶ keys: f keys()
    ▶ lastIndexOf: f lastIndexOf()
    length: 0
    ▶ map: f map()
    ▶ pop: f pop()
    ▶ push: f push()
    ▶ reduce: f reduce()
    ▶ reduceRight: f reduceRight()
    ▶ reverse: f reverse()
    ▶ shift: f shift()
    ▶ slice: f slice()
    ▶ some: f some()
    ▶ sort: f sort()
    ▶ splice: f splice()
    ▶ toLocaleString: f toLocaleString()
    ▶ toString: f toString()
    ▶ unshift: f unshift()
    ▶ Symbol(Symbol.iterator): f values()
    ▶ Symbol(Symbol.unscopables): {copyWithin: true, entries: true, fill: true, find: t
    ▶ __proto__: Object
```

想要找到 valueOf 方法的话，一般会去 Array.prototype 对象寻找，但是却发现 arr.__proto__ 没有这个方法，那么 valueOf 方法是从哪里来的？看下图：

```
▼ __proto__:
  ▶ constructor: f Object()
  ▶ hasOwnProperty: f hasOwnProperty()
  ▶ isPrototypeOf: f isPrototypeOf()
  ▶ propertyIsEnumerable: f propertyIsEnumerable()
  ▶ toLocaleString: f toLocaleString()
  ▶ toString: f toString()
  ▶ valueOf: f valueOf()
  ▶ __defineGetter__: f __defineGetter__()
  ▶ __defineSetter__: f __defineSetter__()
  ▶ __lookupGetter__: f __lookupGetter__()
  ▶ __lookupSetter__: f __lookupSetter__()
  ▶ get __proto__: f __proto__()
  ▶ set __proto__: f __proto__()
```

原来你在这里

当试图访问一个对象的属性时，它不仅仅在该对象上搜寻，还会搜寻该对象的原型，以及该对象的原型的原型，依次层层向上搜索，直到找到一个名字匹配的属性或到达原型链的末尾。

查找 valueOf 方法的大致流程如下：

1. 当前实例对象 obj，查找 obj 的属性或方法，找到后返回
2. 没有找到，通过 obj.__proto__，找到 obj 构造函数的 prototype 并且查找上面的属性和方法，找到后返回
3. 没有找到，把 Array.prototype 当做 obj，重复以上步骤

当然不会一直找下去，原型链是有终点的，最后查找到 Object.prototype 时 Object.prototype.__proto__ === null，意味着查找结束

他们的关系如下：

```
arr.__proto__ === Array.prototype
true
Array.prototype.__proto__ === Object.prototype
true
arr.__proto__.__proto__ === Object.prototype
true

// 原型链的终点
Object.prototype.__proto__ === null
true
```

原型链如下：

arr ---> Array.prototype ---> Object.prototype ---> null

这就是原型链，层层向上查找，最后还没有就返回 undefined

这周的工作：

1. 完善 titan 小程序 day-dish 页面的逻辑，优化 ui
2. 修改 mars 消费者小程序的订单查询页面，优化 ui