JavaScript高级教程
—— 原型链

主讲人和课程设计：耕耕

本节课提纲

- 学习原型链的原因和背景
- 函数基本知识
- 对象基本知识
- 什么是原型链
- __proto__
- prototype
- constructor
- 总结与原理解释

学习原型链的原因和背景

- 面试可能会遇到的问题
- 打好JavaScript的基础
- 实际的运用
- 最重要的是什么？
  - 应该很少有人把这个讲清楚
  - 有点难度吗？
  - 其实看着复杂，理解了其实非常简单

函数基本知识

```
function sayHello(p1, p2, p3) {
    console.log('Hello genggeng!')
    return "hello"
}

let getName = () => {
    return "genggeng"
}
```

```
function Person(name) {
    this.name = name
}

// callback
function getPoint(callback) {
    console.log("local point")
    callback()
}

getPoint(() => {
    console.log("callback")
})
```

函数 – 来点高级货

```
function moreFeatures() {
    return function () {
        return function () {
            return "genggeng"
        }
    }
}


(() => {
    console.log("genggeng-invoke")
})()
```

```
if (() => { }) {
    console.log("ok")
} else {
    console.log("error")
}

let sum = new Function('a', 'b’, 'return a + b');
console.log(sum(1, 2))
```

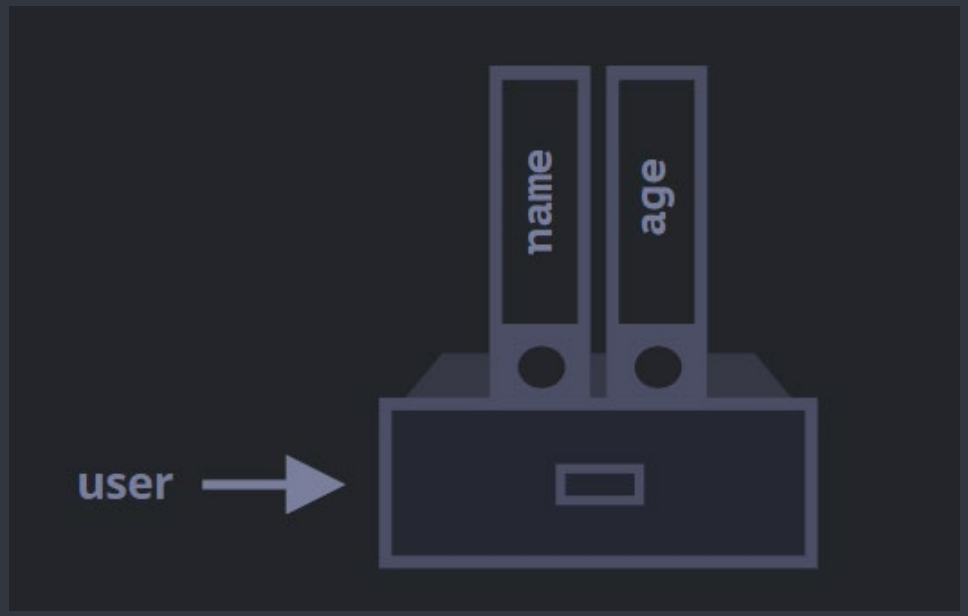
函数 – 来点高级货

```
let doSth = new Function("console.log(123)");
doSth()

function hard1() {
    let value = "test";
    let func = function () {
       console.log(value);
    };
    return func;
}

hard1()()
```

```
//only global
let value = "999"
function hard2() {
    let value = "test";
    let func = new Function("console.log(value)");
    return func;
}

hard2()()
```

对象基本知识

```
// Object 常见创建方法
let user1 = new Object();
let user2 = {};
let special = Object.create(null)

let user3 = {
    name: "John",
    age: 30,
    getName: function () {
        return this.name
    }
};

for (const key in user3) {
     console.log(key);
 }
```



对象 — 来点高级货

```
let id = Symbol("id");

let user5 = {
    name: "John",
    [id]: 123 // not "id": 123
};

for (const key in user5) {
    console.log(key);
}
```

```
let user6 = {};
console.log(user?.address?.street);

let userAdmin = {
    admin() {
        console.log("I am admin");
    }
};

let userGuest = {};
userAdmin.admin?.();
userGuest.admin?.();
```

对象 — 来点高级货

```
let user7 = {
    name: "John"
};

Object.defineProperty(user, "name", {
    writable: false
});

user.name = "Pete";
```

```
let user8 = {
    name: "John",
    surname: "Smith",
    get fullName() {
        return `${this.name} ${this.surname}`;
    },
    set fullName(value) {
        [this.name, this.surname] = value.split(" ");
    }
};

user8.fullName = "Alice Cooper";
console.log(user8.fullName);
console.log(user8.name);
console.log(user8.surname);
```

什么是原型链

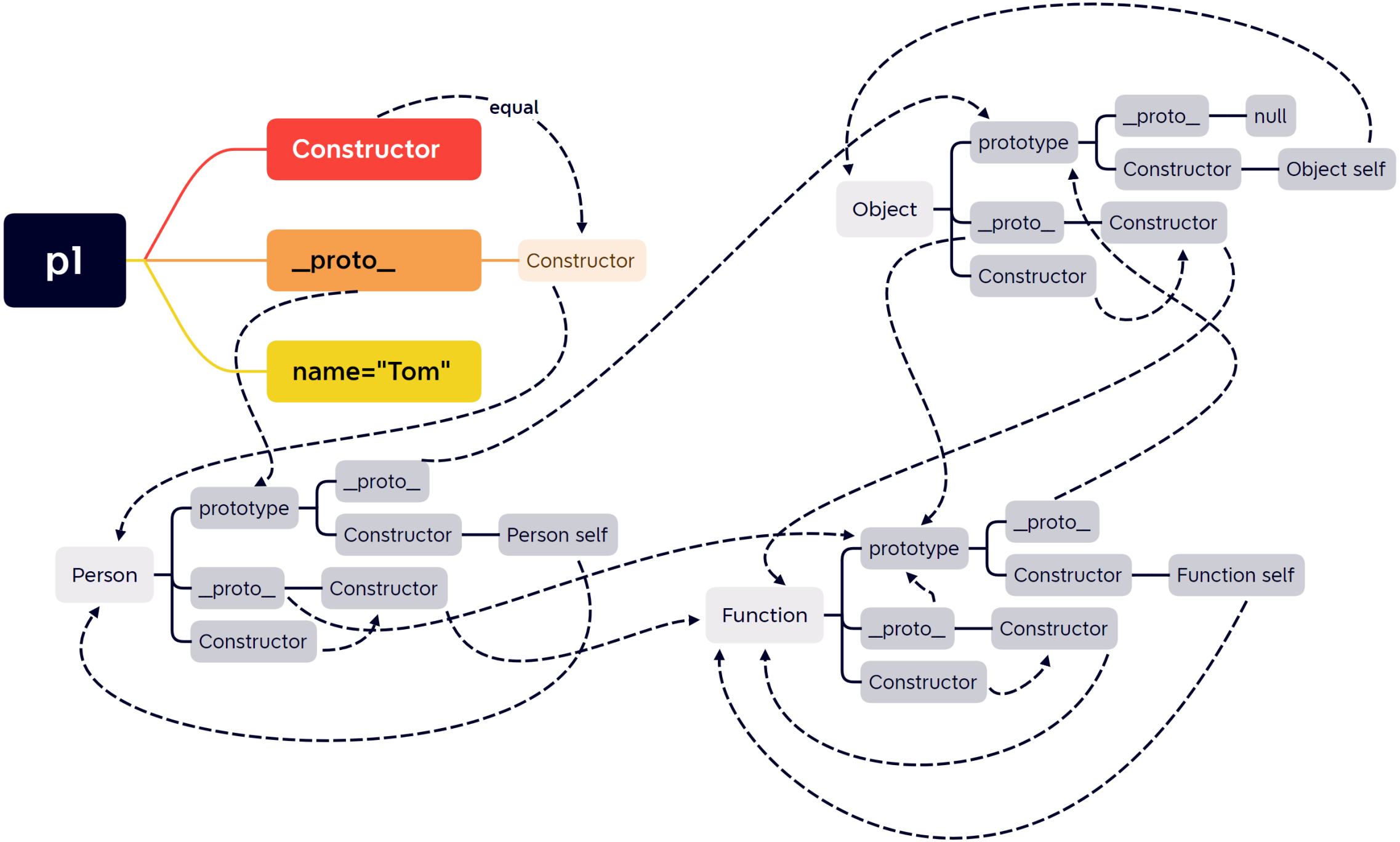
原型：每一个对象都与另一个对象相关联，那个关联的对象就称为原型。
例如：函数Person有一个属性**prototype**，指向一个对象，对于普通函数来说没多意义，对于构造函数就有作用了，当使用new操作符时，会把Person.prototype（原型对象）赋值给实例的**__proto__**（原型实例）属性。

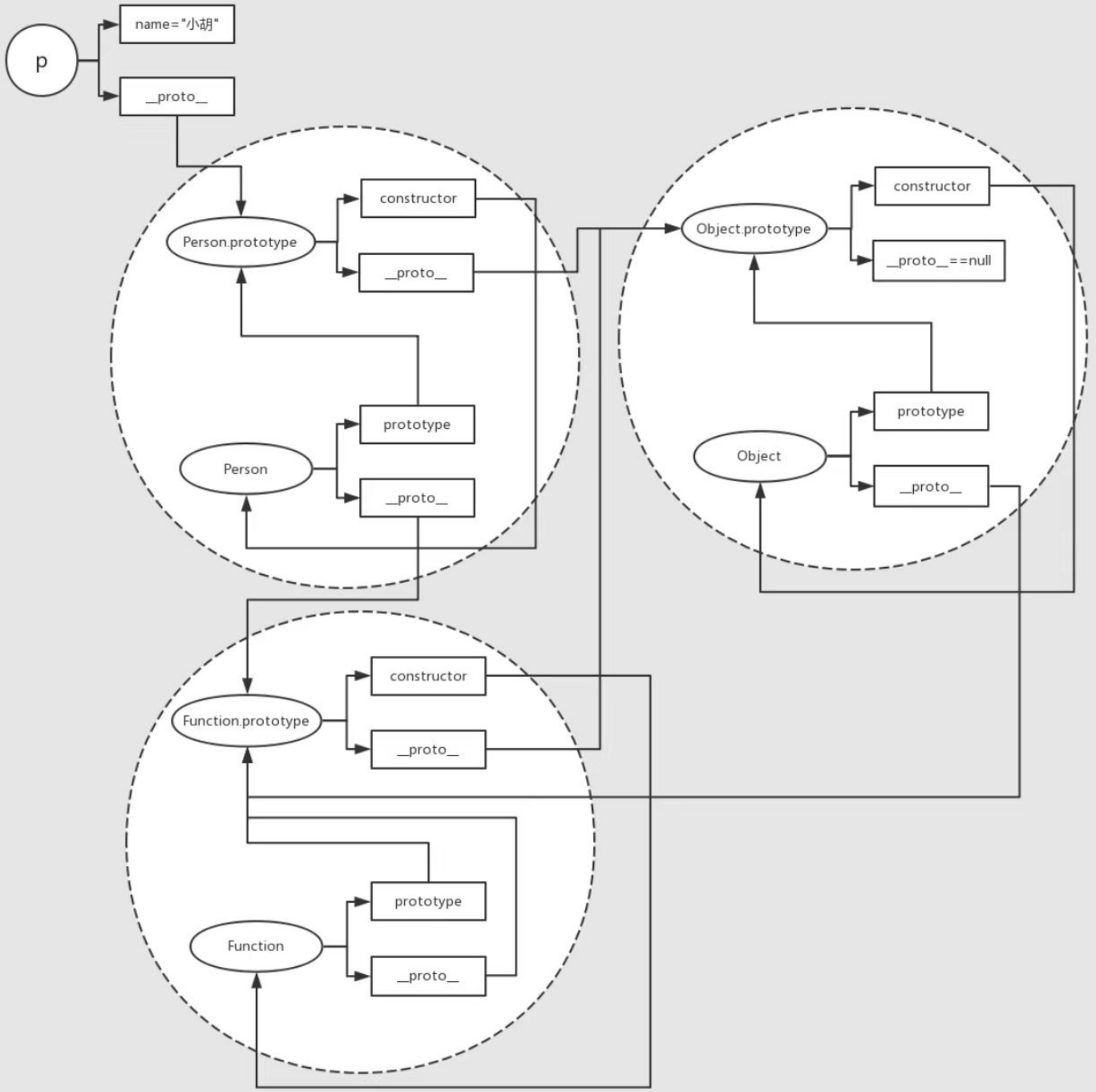
JavaScript有一个原型查找机制，把原来定义在实例上的方法，放到原型对象上去，通过构造函数的new操作，会把原型对象赋值给实例的__proto__属性，那么当使用返回的实例去调用某一个方法的时候，如果实例本身上没有，就去自动去实例的__proto__上去查找，这样达到方法的复用，减少内存开销。

实例解释

- 理解__proto__
  - 实例对象的属性
- 理解prototype
  - 构造对象的属性
- 理解constructor
  - 构造器，用来构造对象的

```
function Person(name) {
    this.name = name
}

const p1 = new Person("Tom")
```


p1
Constructor
proto
name="Tom"
equal
Constructor
Person
prototype
proto
Constructor
Person self
proto
Constructor
Constructor
Function
prototype
proto
Constructor
Function self
proto
Constructor
Constructor
Object
prototype
proto
null
Constructor
Object self
proto
Constructor
Constructor


p
name="小胡"
__proto__
Person.prototype
constructor
__proto__
prototype
Person
__proto__
Object.prototype
constructor
__proto__==null
prototype
Object
__proto__
Function.prototype
constructor
__proto__
prototype
Function
__proto__

总结

- 理解__proto__
  - 实例对象的属性
  - 一个实例对象的__proto__总有一个创造它的构造函数prototype对应它，**prototype的实例对象一般是Object.prototype**
  - **p1**. __proto__ === Person.prototype
  - **Person.prototype**. __proto__ === Object.prototype
  - **Object.prototype**. __proto__ == null
- 理解prototype
  - 构造对象的属性，跟上面的对应
- 理解constructor
  - 构造器，用来构造对象的
  - **p1.constructor === p1.__proto__.constructor**
  - **Person.prototype.constructor = Person**

总结2

p1是实例对象，谁创造了p1, 它的原型实例__proto__就对应谁的prototype

p1. __proto__ **Person**.prototype

Person.prototype.__proto__ **Object**.prototype

Object.prototype .__proto__ **到头了null, 特殊就这么一个**

Person.__proto__ 是谁创造了Person? Person()实质上是一个函数，是Function创造了它 **Function**.prototype

Object.__proto__ Object() 也是一个函数 **Function**.prototype

Function.__proto__ Function() 也是一个函数 **Function**.prototype

Person.prototype.constructor === Person 规则1

p1.constructor === p1 .__proto__.constructor 规则2，可以理解为p1的构造器是谁？Person咯

总结3

一句话总结了，A.__proto__，是谁搞出来了A?
就去找那个创造出A的prototype