原型

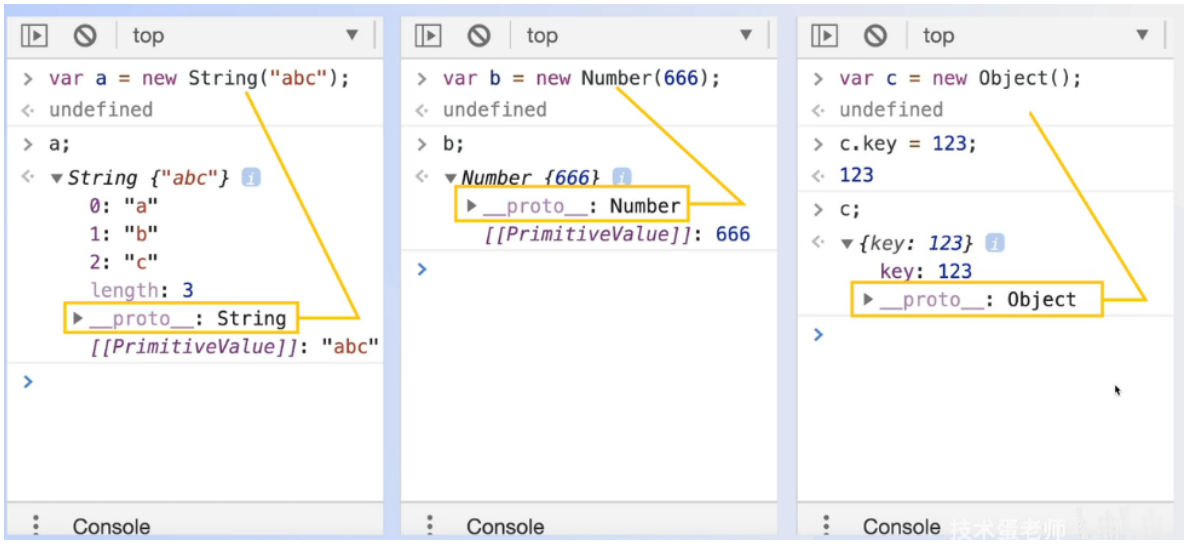
prototype和`__proto__`

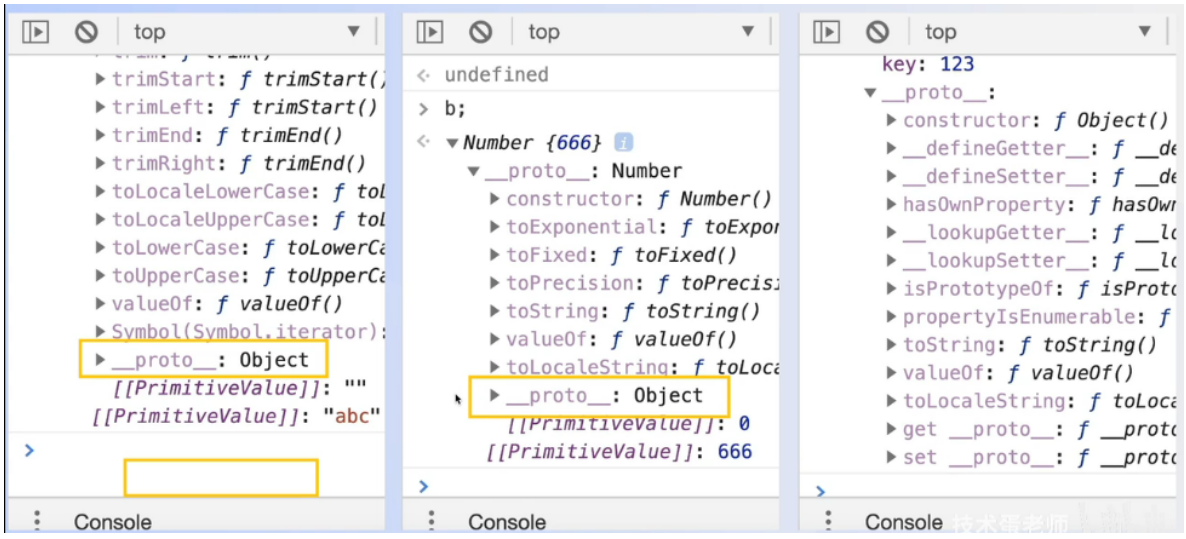
是什么?

prototype:显式原型

`__proto__`:隐式原型

新对象被创建的时候，除了各自的属性以外，还有一个隐式的`__proto__`属性被创建。这个`__proto__`是一个Object。



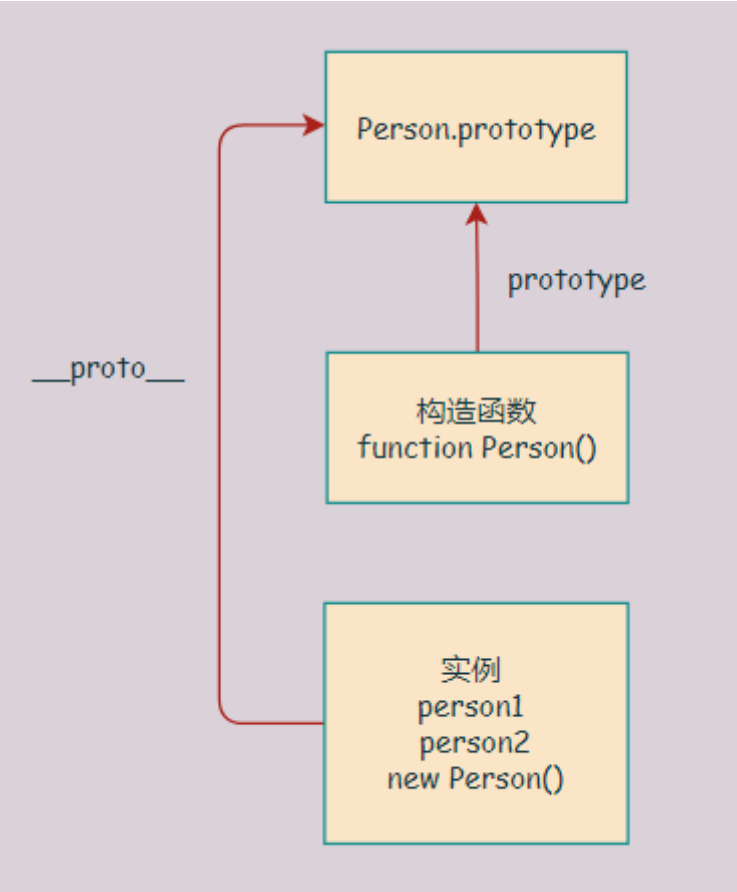


关系

`构造函数的`prototype和其`实例`的**proto**是指向同一个地方的，这个地方就叫做`原型对象`。

Person.`prototype` === person1.`__propto__`

```
function Person(name, age) { // 这个就是构造函数
  this.name = name
  this.age = age
}

const person1 = new Person('小明', 20) // 这个是Person构造函数的实例
const person2 = new Person('小红', 30) // 这个也是Person构造函数的实例
```



函数

平时定义函数的方法，有下面几种：

```
1.
function fn1(name, age){
    console.log(`${name}今年${age}岁`)
}

2.
const fn2 = function(name, age){
    console.log(`${name}今年${age}岁`)
}

3.
const fn3 = (name, age) => {
    console.log(`${name}今年${age}岁`)
}
```

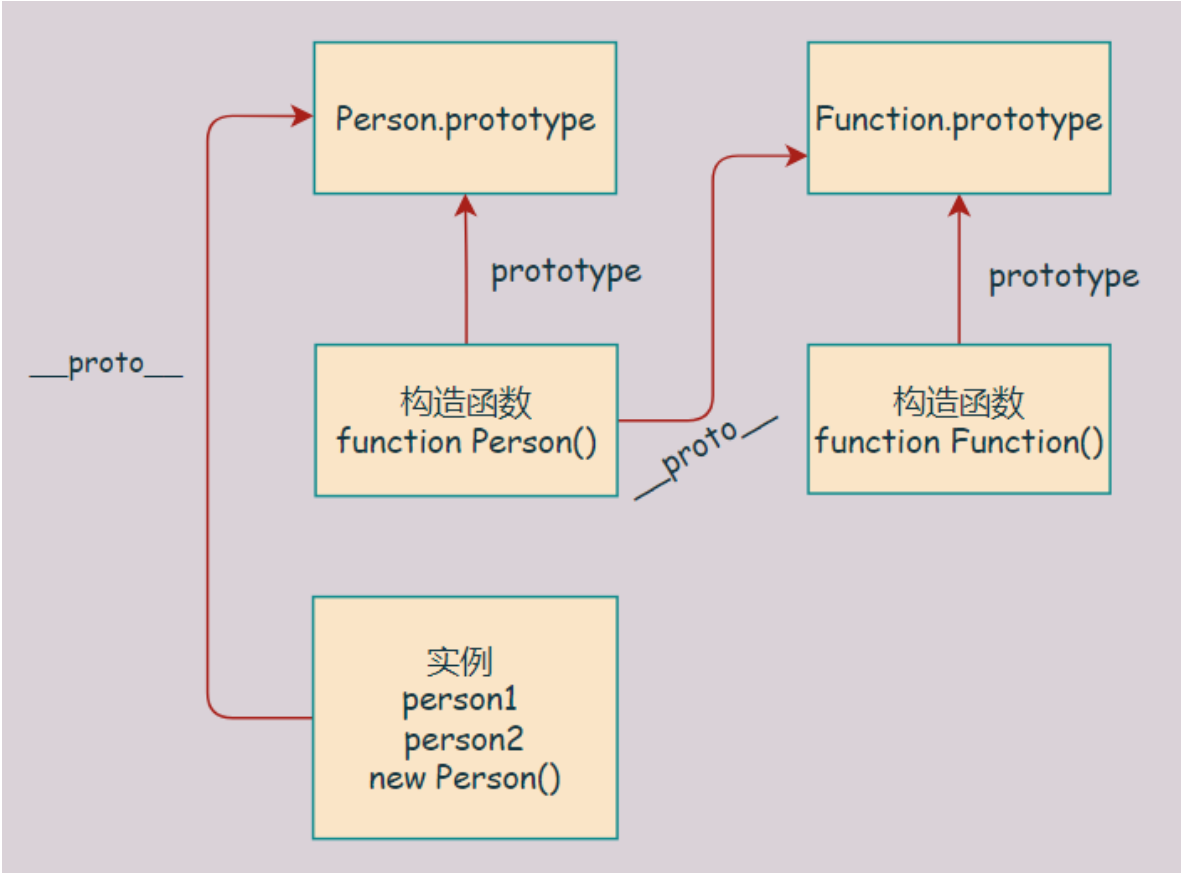
上面这些方法都可以使用`new Function`来声明。这么说`Function`也是一个构造函数。

```
const fn1 = new Function('name', 'age', 'console.log(`${name}今年${age}岁`)')
const fn2 = new Function('name', 'age', 'console.log(`${name}今年${age}岁`)')
const fn3 = new Function('name', 'age', 'console.log(`${name}今年${age}岁`)')
```

那么fn1，fn2，fn3就是`构造函数`Function的`实例`。

就有：

```
Function.prototype === fn1.__proto__
Function.prototype === fn2.__proto__
Function.prototype === fn3.__proto__
```



对象

创建对象的方法：

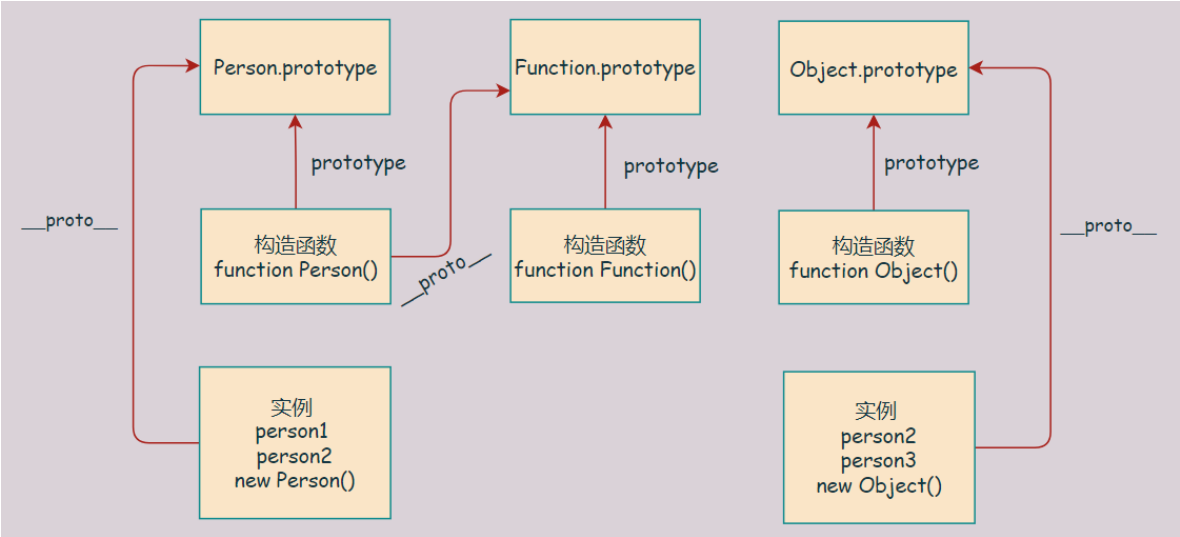
- `构造函数创建对象`（Function构造函数的实例，不讨论）
- `字面量创建对象`
- `new Object创建对象`
- `Object.create创建对象`(创建出来的是一个空原型的对象，不讨论)

```
// 第一种：构造函数创建对象
function Person(name, age) {
  this.name = name
  this.age = age
}

// 第二种：字面量创建对象
const person2 = {name: 'scout', age: 24}

// 第三种：new Object创建对象
```

```
const person3 = new Object()
person3.name = 'scout'
person3.age = 24

// 第四种：Object.create创建对象
const person4 = Object.create({})
person4.name = 'scout'
person4.age = 24
```

这里`字面量创建对象`和`new Object`两种的方式，本质都是`new Object`创建对象。

那么person2, person3都是`Object构造函数`的`实例`。



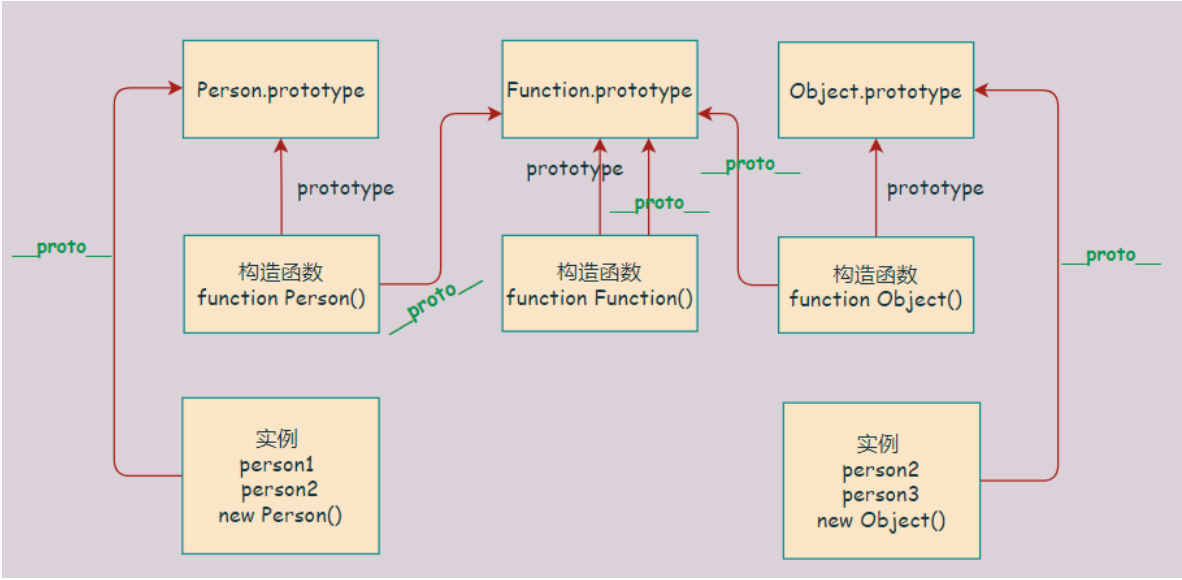
Function和Object

- `函数`是`Function构造函数`的实例
- `对象`是`Object构造函数`的实例

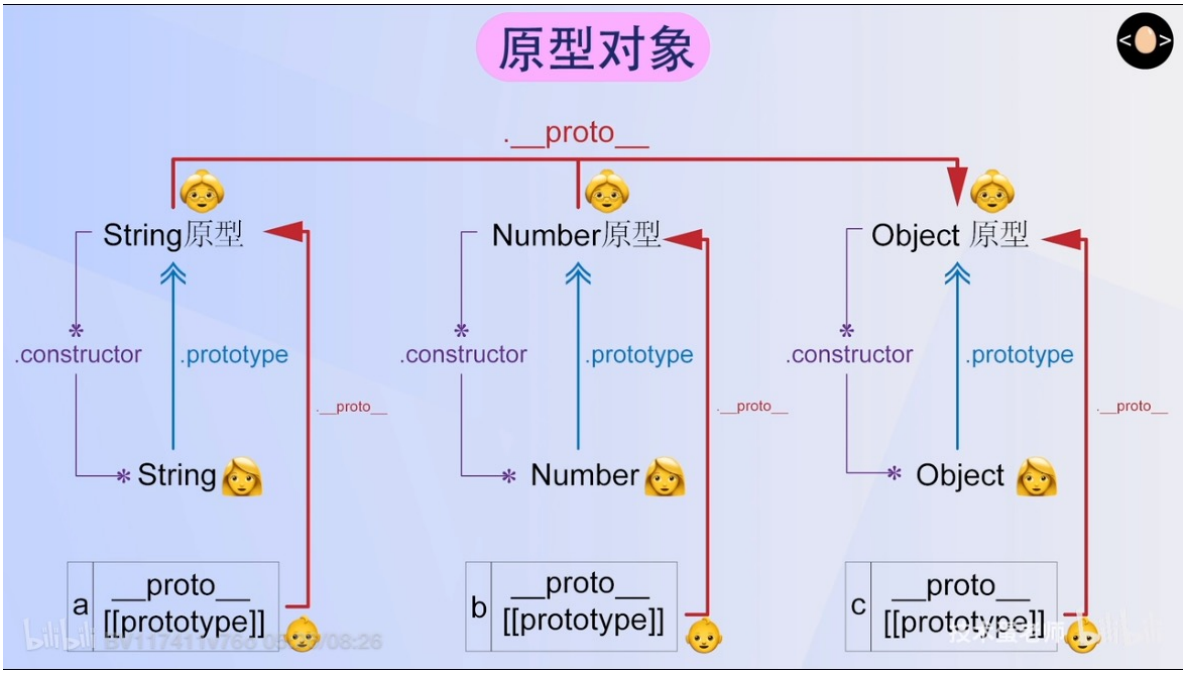
那`Function构造函数`和`Object构造函数`的实例?

- `function Object()`其实也是个函数，所以他是`Function构造函数`的实例
- `function Function()`其实也是个函数，所以他也是`Function构造函数`的实例，他是他自己本身的实例

```
Function.prototype === Object.__proto__
Function.prototype === Function.__proto__
```



b站上的一张图也贴出来：



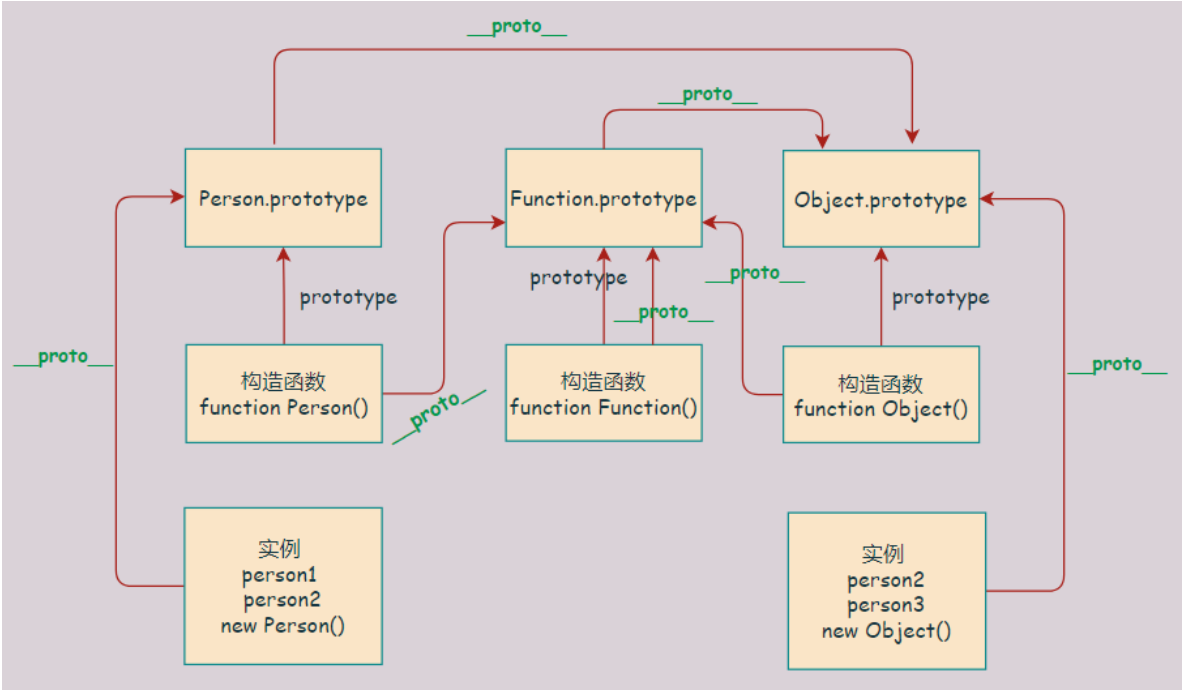
原型链

Person.prototype 和 Function.prototype

- Person.prototype，它是`构造函数Person`的原型对象
- Function.prototype，他是`构造函数Function`的原型对象

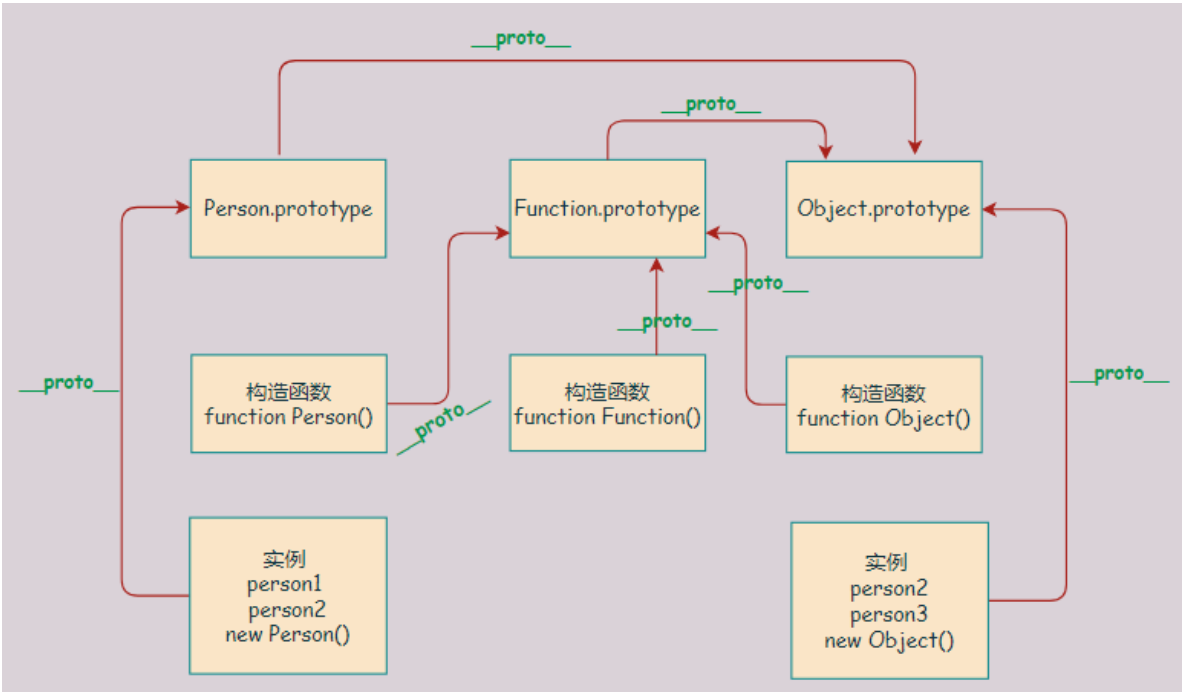
`原型对象`本质也是`对象`，是通过`new Object()`创建出来的。说明`Person.prototype`和`Function.prototype`都是`构造函数Object`的实例。所以：

```
Person.prototype.__proto__ === Object.prototype
Function.prototype.__proto__ === Object.prototype
```



什么是原型链?

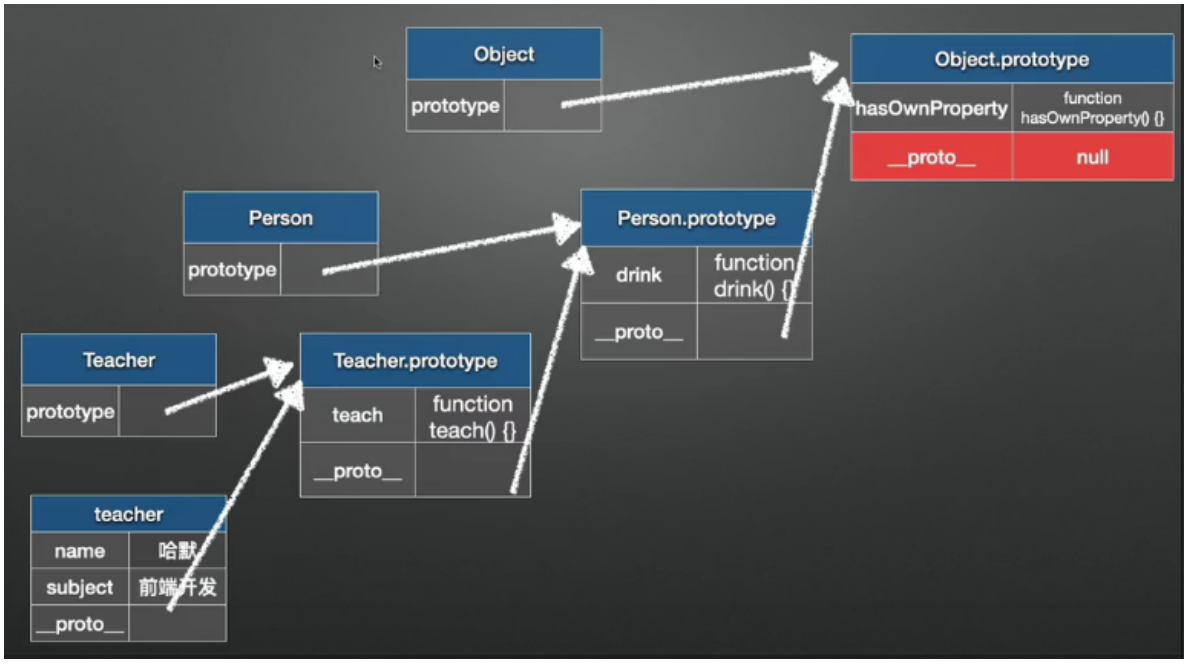
`__proto__`的路径 就叫原型链。



再整理一下:

三条原型链。

当访问一个对象的属性时，如果这个对象内部不存在这个属性，那么它会去它的原型对象里找这个属性，这个原型对象又会有自己的原型，于是就这样一直找下去，这样一层一层往上找的过程就形成了原型链。



原型链终点

`Object.prototype`的`__proto__`指向null，就是原型链的终点。

原型继承

`原型继承`就是，`实例`可以使用`构造函数上的prototype`中的方法。

参考

[这可能是掘金讲「原型链」，讲的最好最通俗易懂的了，附练习题！]

[【JS】图解原型链相关练习题，带你彻底搞懂原型链！！！（这可能是掘金画原型链画的最正的😀）]