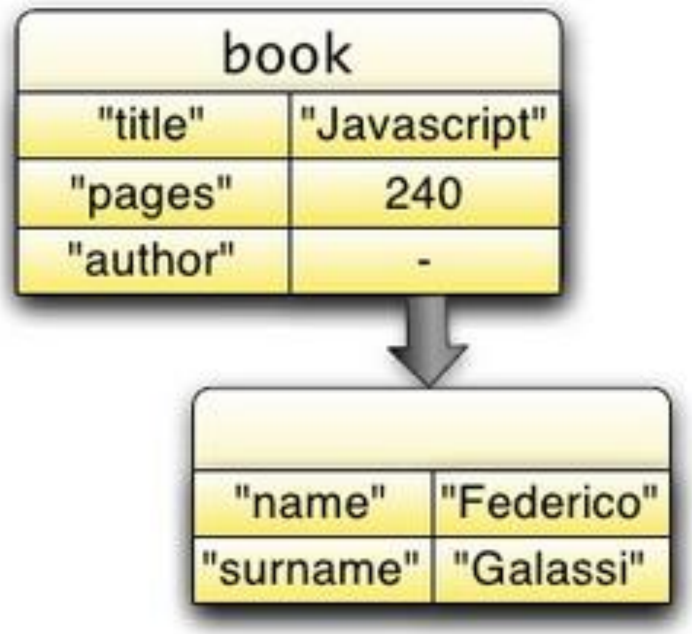
乐 美 无 限
nemo-tec.com

JavaScript对象原型链

豆连军 @八月虎baidu
北京乐美无限科技有限公司

对象Object

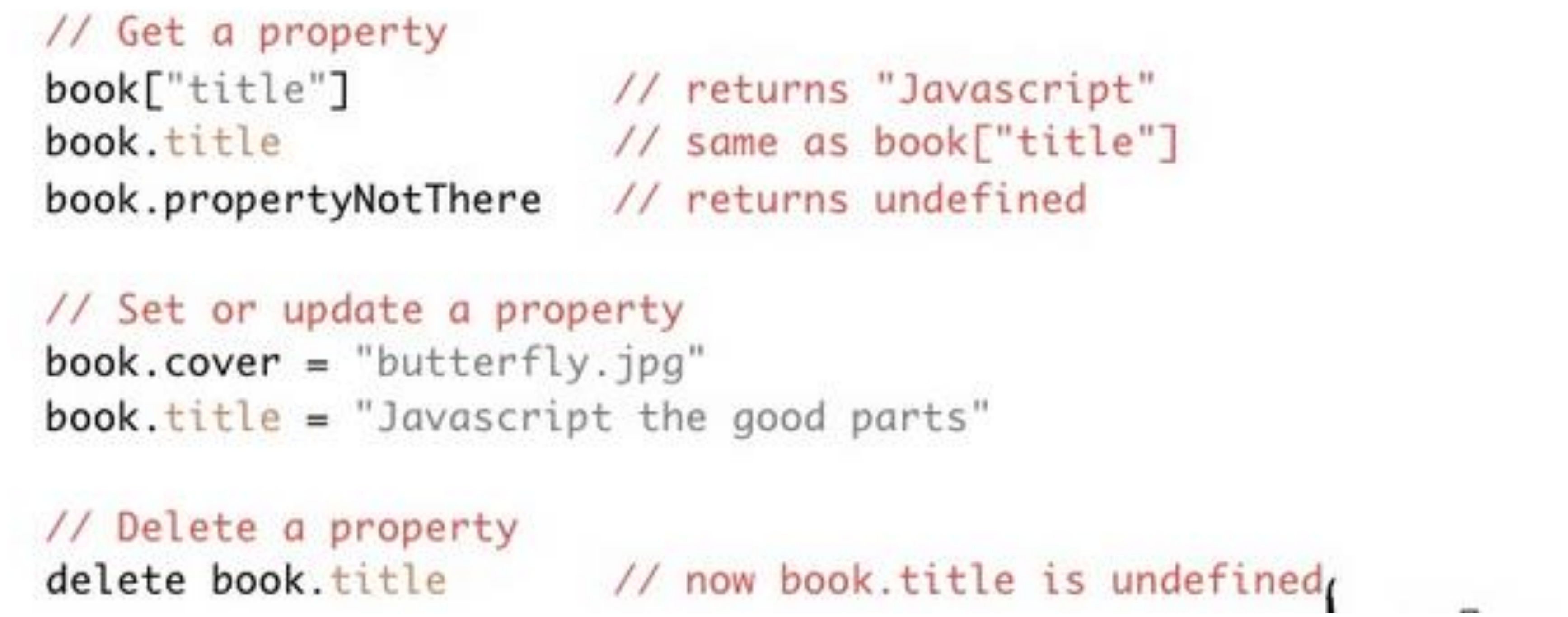
- Containers of key/value pairs （键值对）
  - keys are strings
  - values are anything



```
// Creation with literal
var book = {
    title: "Javascript",
    pages: 240,
    author: {
        name: "Federico",
        surname: "Galassi"
    }
}
```

对象的动态性

- Objects are dynamic
  - Properties can be added and removed at runtime
  - No class constraints

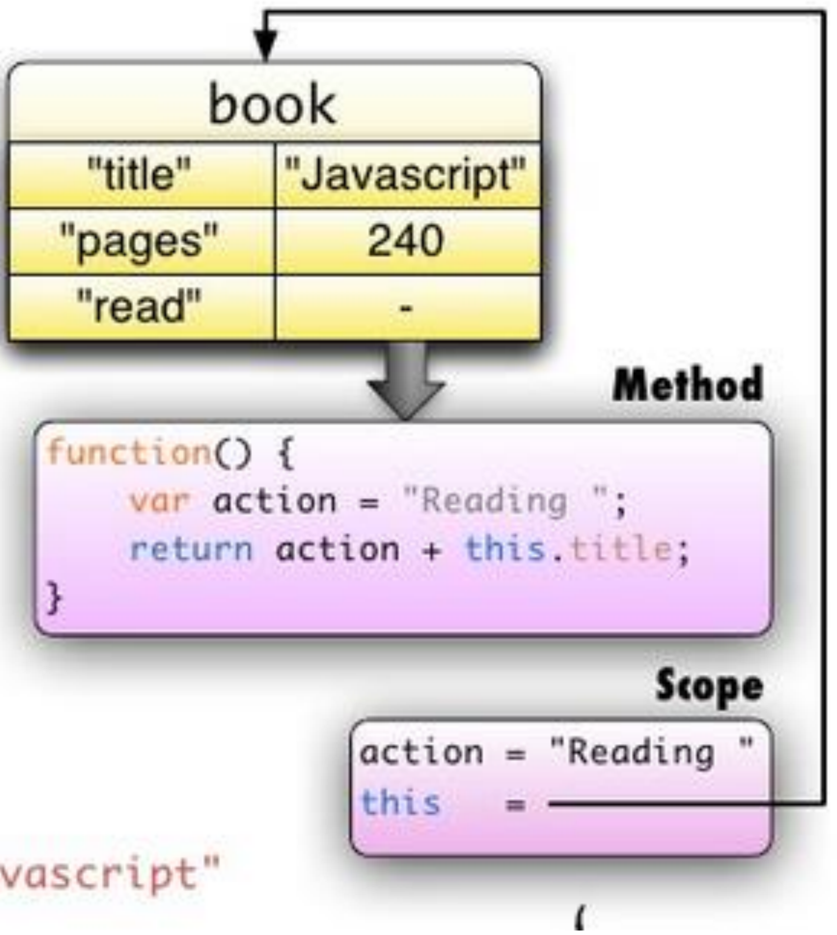


```
// Get a property
book["title"]              // returns "Javascript"
book.title                 // same as book["title"]
book.propertyNotThere      // returns undefined

// Set or update a property
book.cover = "butterfly.jpg"
book.title = "Javascript the good parts"

// Delete a property
delete book.title          // now book.title is undefined
```

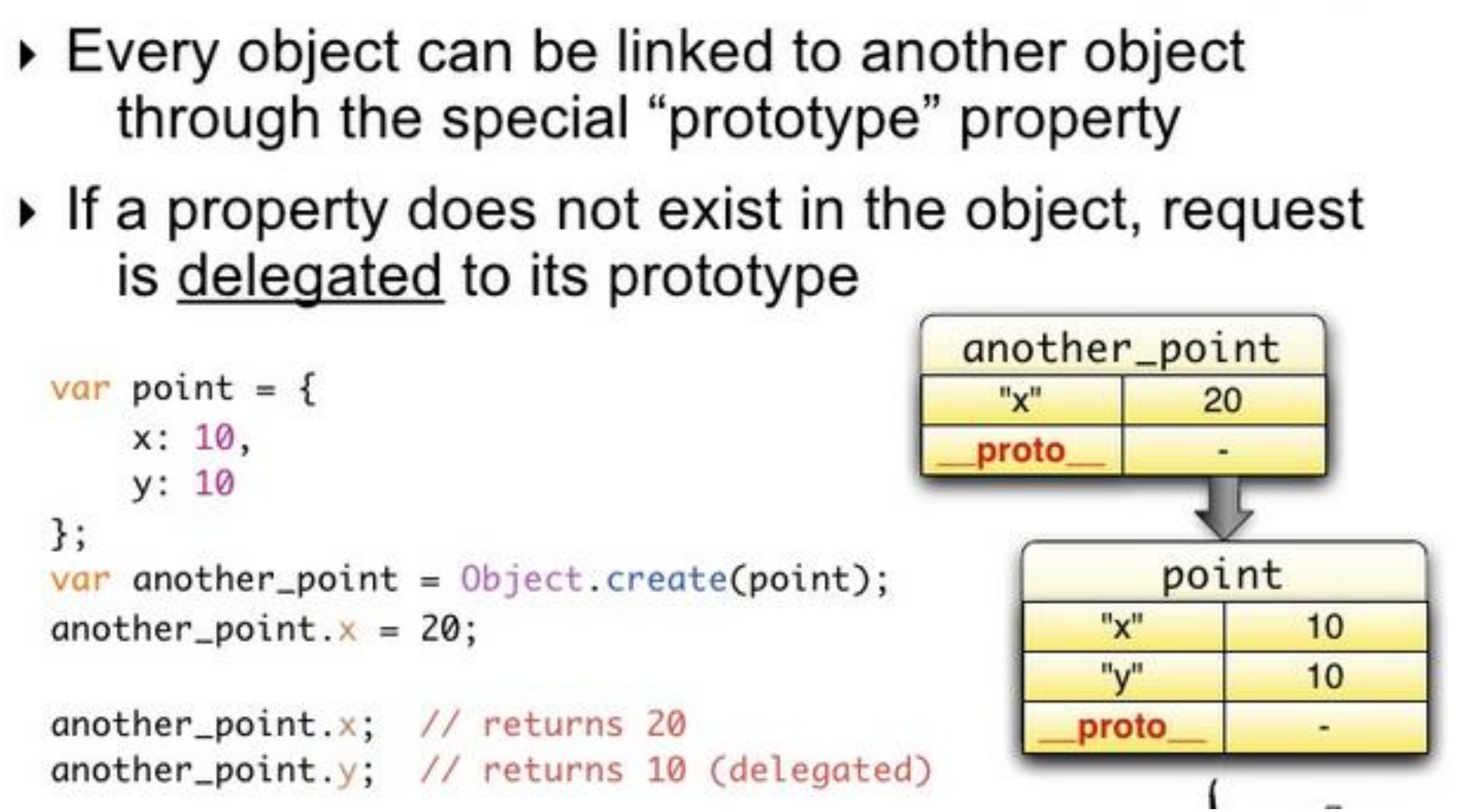
对象的方法

- Methods are function valued properties
- Inside methods **this** is bound to object “on the left”

```
book.read = function() {
    var action = "Reading ";
    return action + this.title;
}

book.read(); // returns "Reading Javascript"
```



对象属性的继承机制

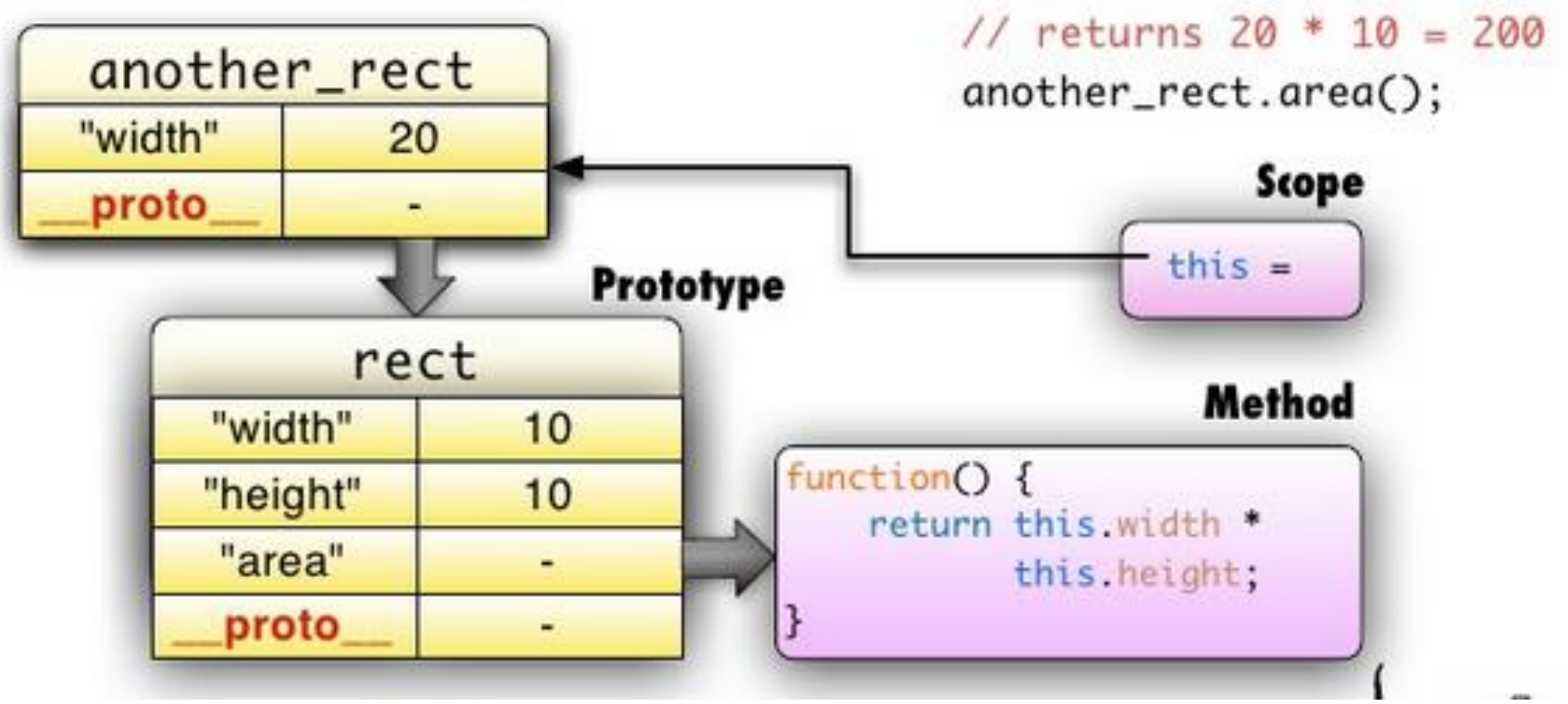


- Every object can be linked to another object through the special “prototype” property
- If a property does not exist in the object, request is delegated to its prototype

```
var point = {
    x: 10,
    y: 10
};
var another_point = Object.create(point);
another_point.x = 20;

another_point.x;  // returns 20
another_point.y;  // returns 10 (delegated)
```

对象属性的继承机制

- Delegation works for methods too
- **this** is always bound to the “first object”

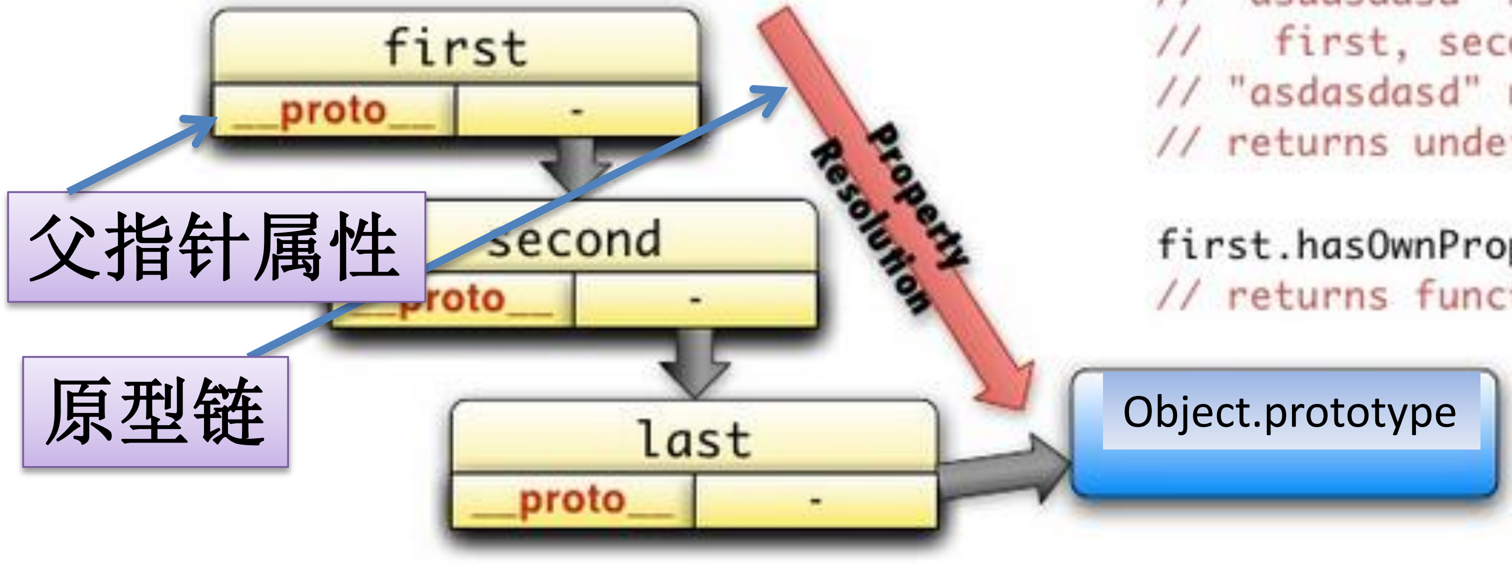


```
// returns 20 * 10 = 200
another_rect.area();
```

```
function() {
    return this.width *
           this.height;
}
```

对象属性的继承机制

- Properties are resolved by following the Prototype Chain
- Prototype Chains always ends with Object
- Beyond there’s undefined



```
first.asdasdasd;
// "asdasdasd" not in
//    first, second, last
// "asdasdasd" not in {}
// returns undefined

first.hasOwnProperty
// returns function() ...
```

原型继承

- Prototypes are javascript way to share
  - Data
  - Behavior

原型继承

- Prototypal Inheritance
  - Vs Classical Inheritance
  - Simpler
    - No classes and objects, only objects
  - Easier
    - Work by examples, not abstractions
  - Powerful !!
    - Can simulate classical
    - Reverse not true
  - Shhhh, Don’t tell anyone
    - Easier to write spaghetti code

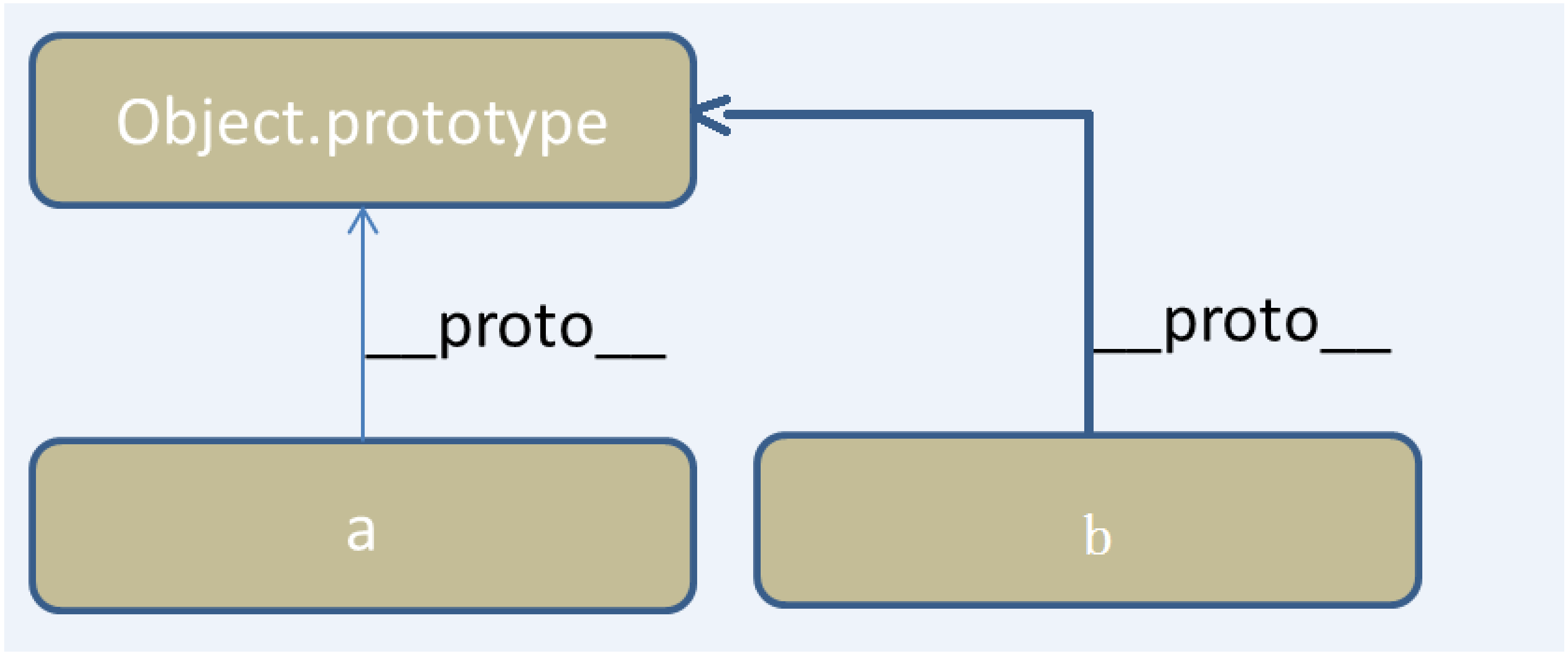
限

nemo-tec.com

对象创建方法1：对象字面量方式

- var a = {} ;
- var b = {} ;

__proto__: 所有对象都有的属性名称，用于指向其原型，称为“原型指针”。__proto__此属性是隐含创建的，其属性名称是非标准的，因此不用“属性”来称呼它，而用“指针”来代称。对象之间通过这种原型属性的指向关系构成了**原型链**。

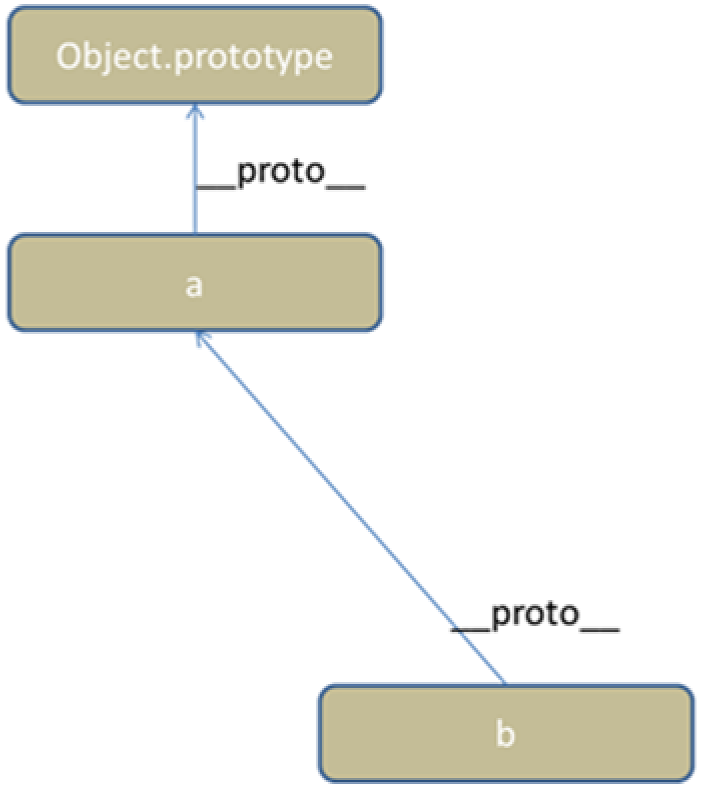


对象创建方法2：Object.create()

- Ok, I cheated
  - __proto__ available in mozilla only
  - Object.create coming in next revision of language
- Javascript is schizophrenic
  - Prototypal nature
  - Wannabe classical

对象创建方法2： Object.create()

- var b = Object.create(a) ;



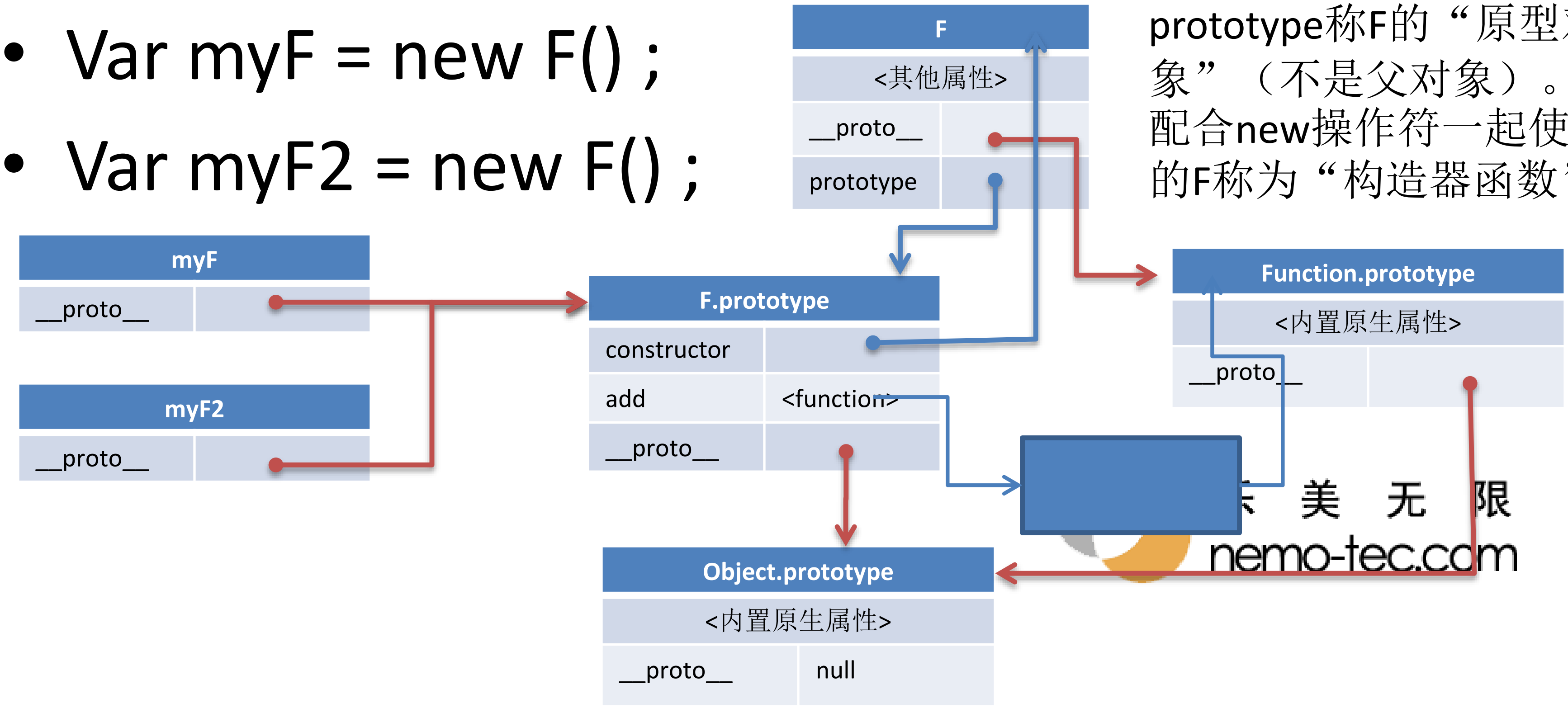
对象创建方法3：构造器函数方式

- var F = function () {};

或者：function F(){} ;

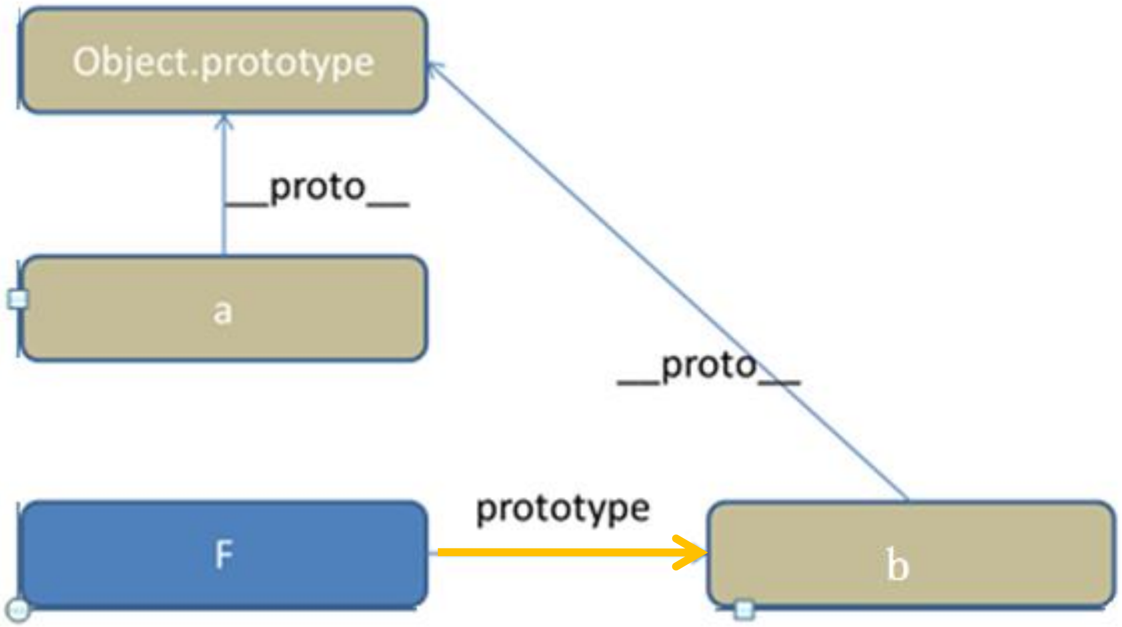
- F.prototype.add = function(){} ;
- Var myF = new F() ;
- Var myF2 = new F() ;

prototype：该属性名称是函数对象的内置属性，普通对象没有该属性名称。prototype属性用于配合new操作符使用，用于指明新创建的对象实例的原型指针指向哪里。prototype称F的“原型对象”（不是父对象）。配合new操作符一起使用的F称为“构造器函数”。



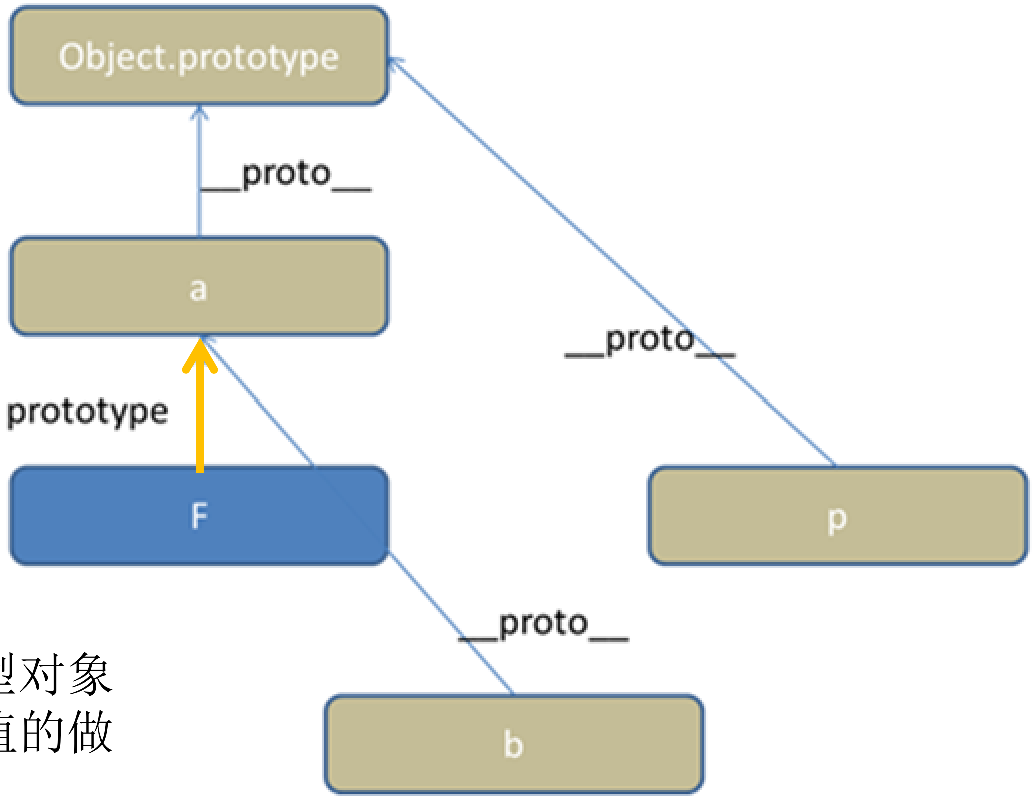
对象创建方法3：构造器函数方式 cont.

- Var b = {} ;
- var F = function () {};
- F.prototype = b ;



对象创建方法3：构造器函数方式 cont.

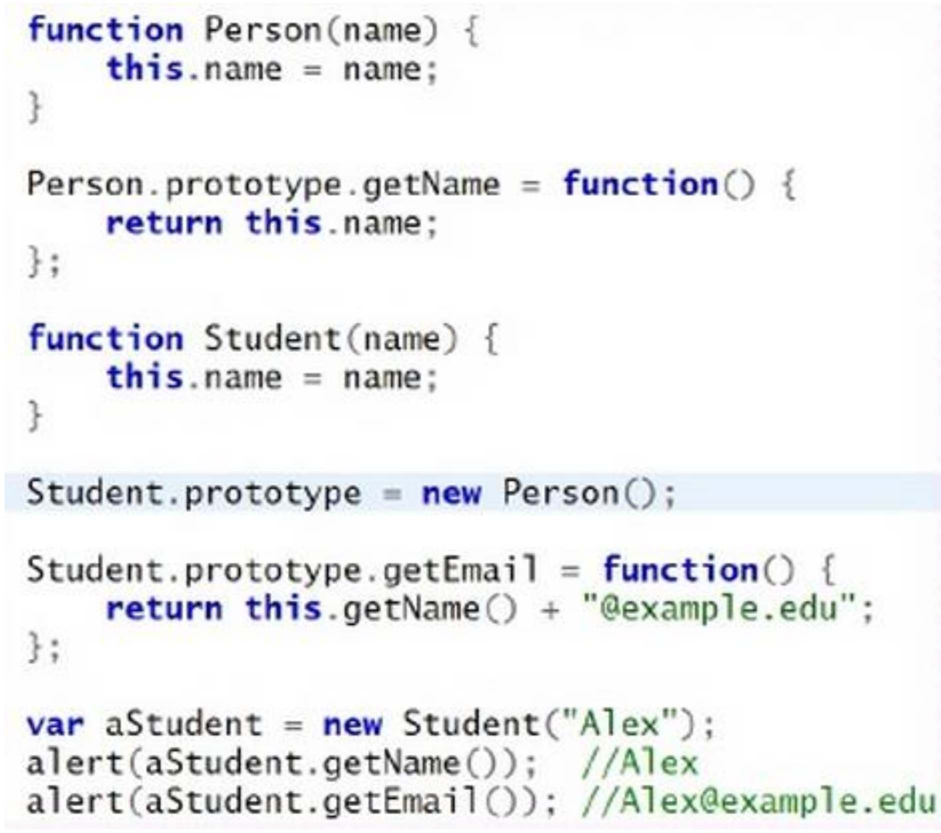
- var a={}, p = {} ;
- F.prototype = a ;
- var b = new F();



建议：对象的缺省值都放到原型对象中，替代在构造函数中设缺省值的做法。

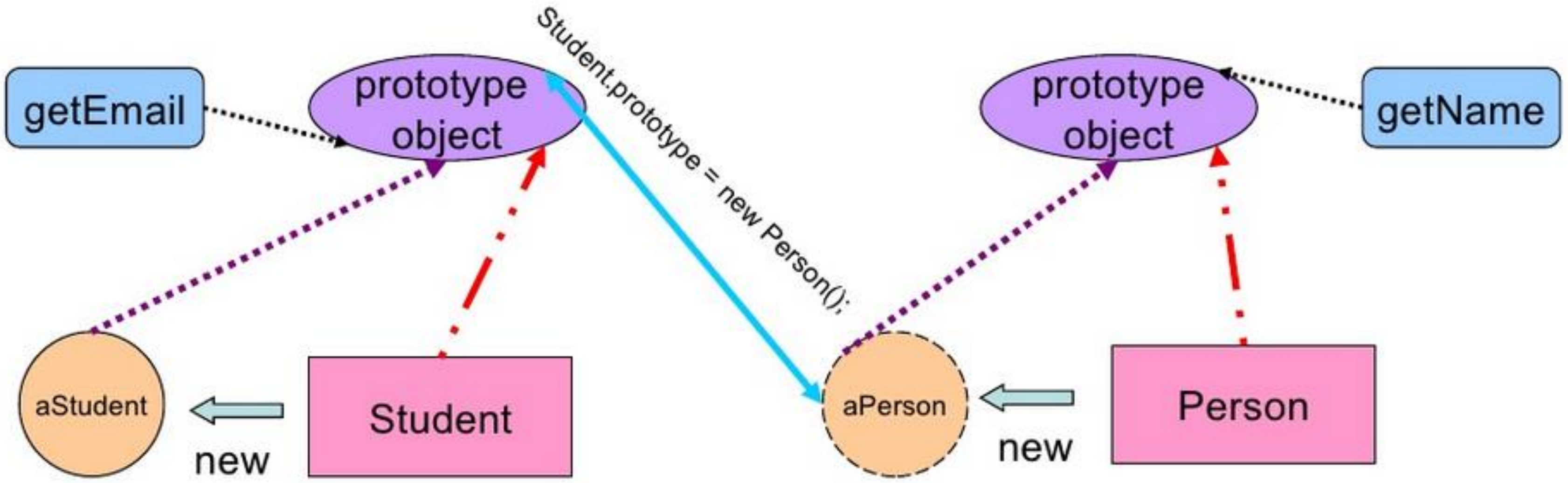
父类与子类

- Person是父类，Student是子类
- Student子类从父类继承getName方法，并声明了自己的方法getEmail



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

Person.prototype.getName = function() {
    return this.name;
};

function Student(name) {
    this.name = name;
}

Student.prototype = new Person();

Student.prototype.getEmail = function() {
    return this.getName() + "@example.edu";
};

var aStudent = new Student("Alex");
alert(aStudent.getName());  //Alex
alert(aStudent.getEmail()); //Alex@example.edu
```

父类与子类cont.



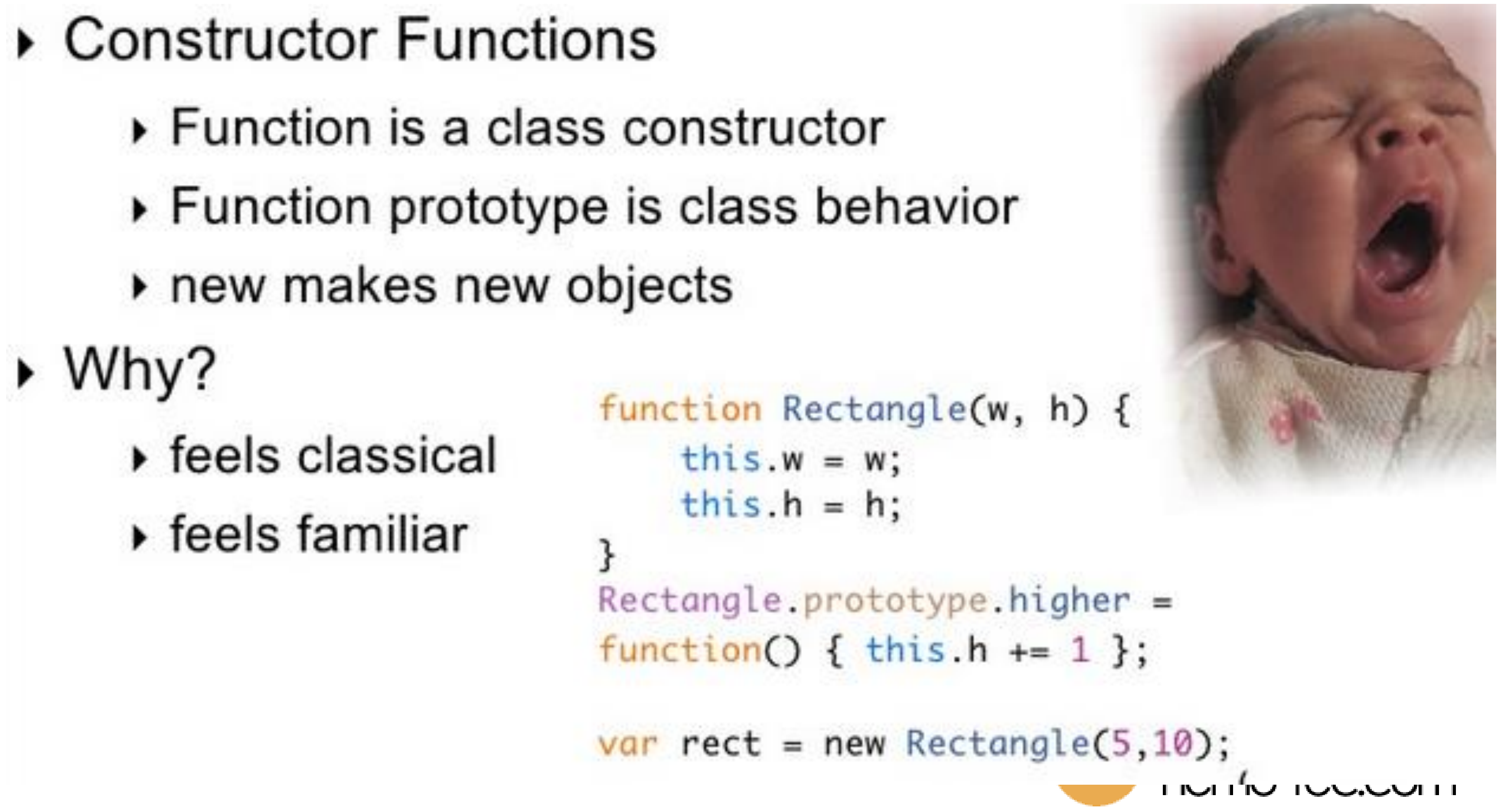
⋯▶	Implicit references to constructor's prototype
‐ ·▶	Constructor's prototype
🟪	Constructor
●	Objects created by *new* operator
🟦	Properties

<u>Search path of *aStudent.getName*</u>

Student's prototype object ->
Person's prototype object

Note: aPerson doesn't exist in the code.

总结：构造器函数



- Constructor Functions
  - Function is a class constructor
  - Function prototype is class behavior
  - new makes new objects
- Why?
  - feels classical
  - feels familiar

```
function Rectangle(w, h) {
    this.w = w;
    this.h = h;
}
Rectangle.prototype.higher =
function() { this.h += 1 };

var rect = new Rectangle(5,10);
```