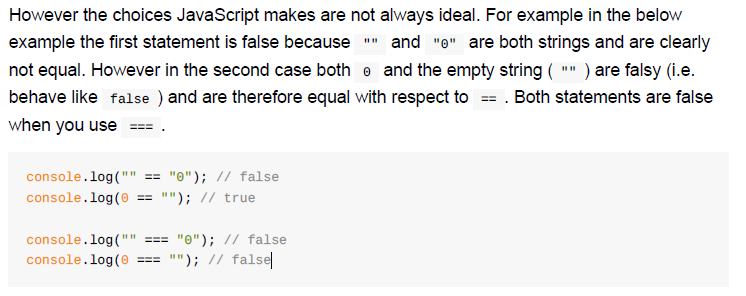
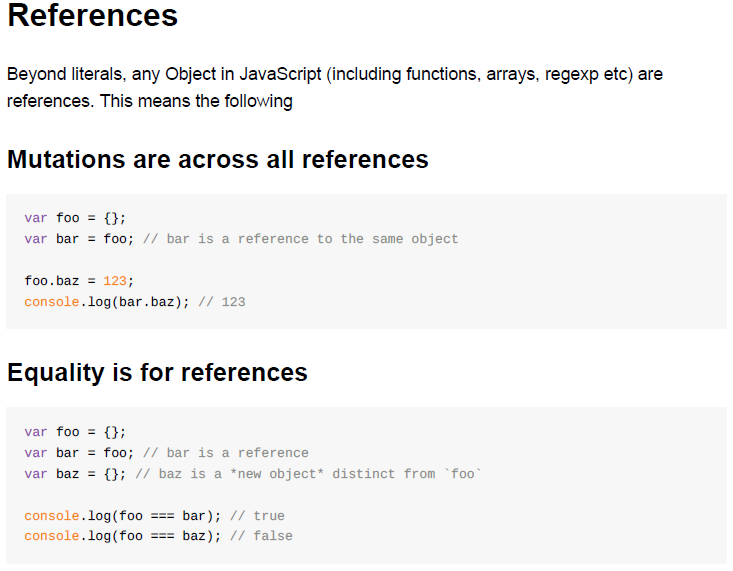
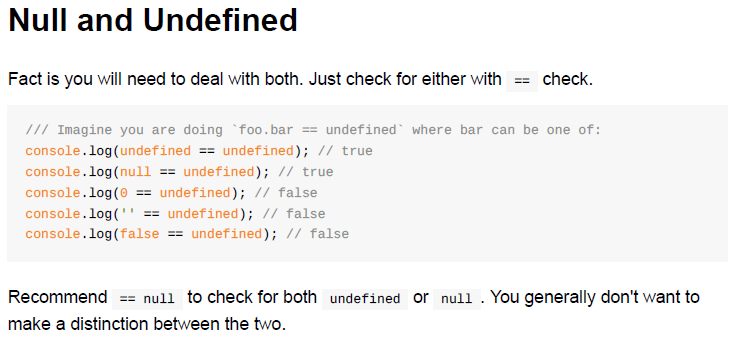
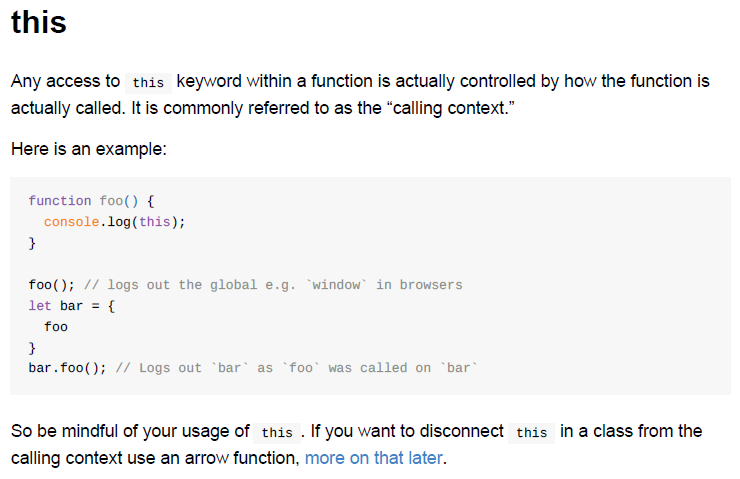
Study

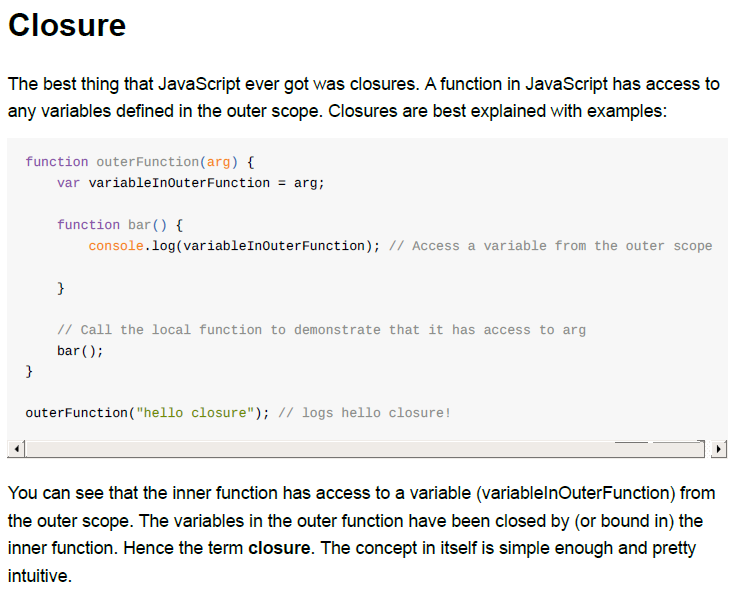
**TypeScript Deep Dive**

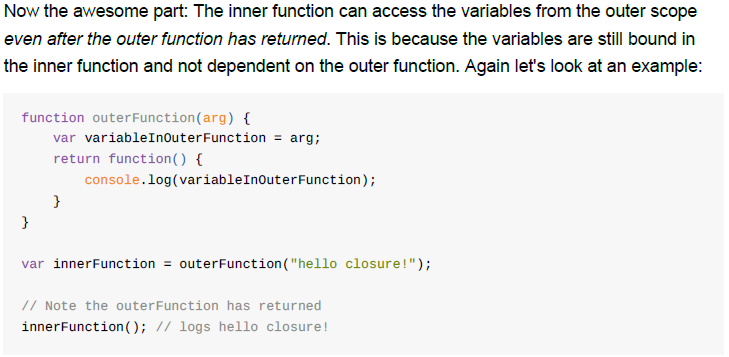
****

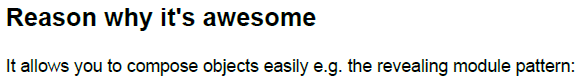
****

****

****

****

****

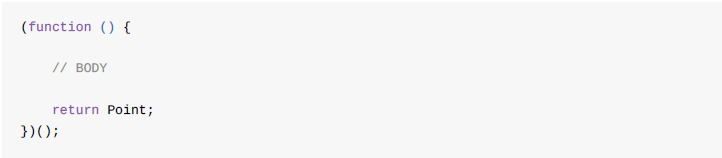
****

****

**Jianfeng: this is a pattern, which is used in Factory, Promise etc. in AngularJS.**

**What's up with the IIFE**

Immediately-Invoked Function Expression (IIFE) i.e.

****

**Arrow Functions**

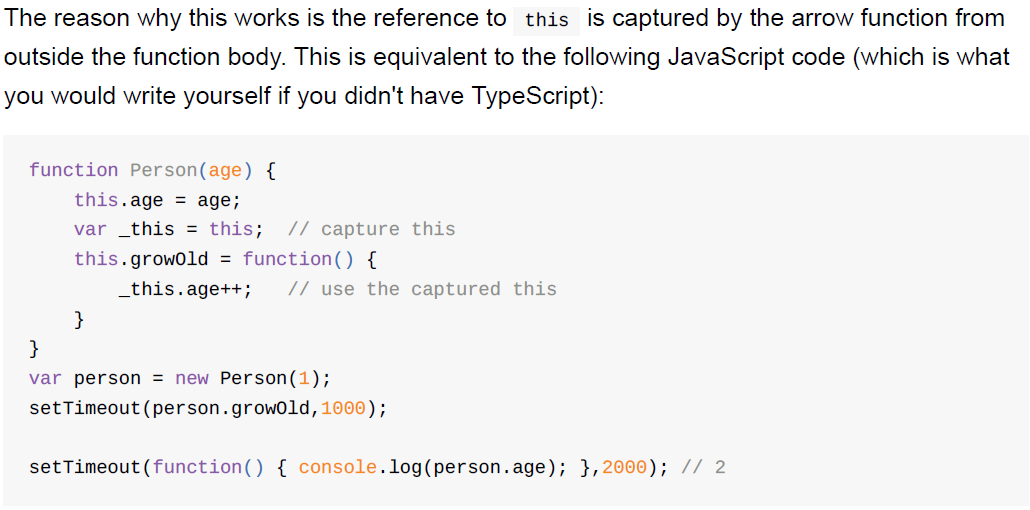
1. You don't need to keep typing *function*

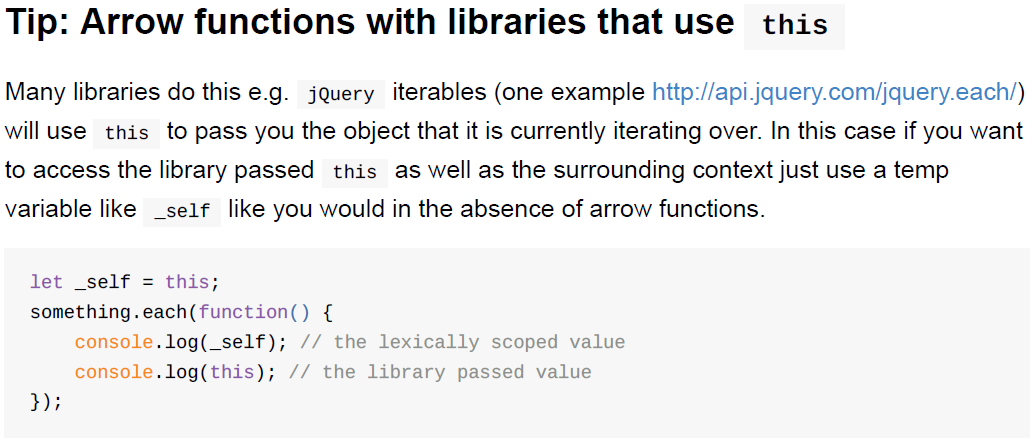
2. It lexically captures the meaning of *this*

3. It lexically captures the meaning of *arguments*



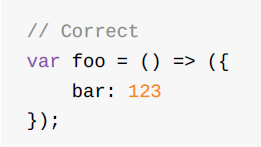
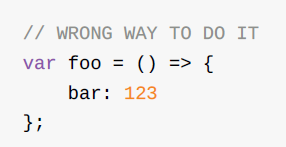




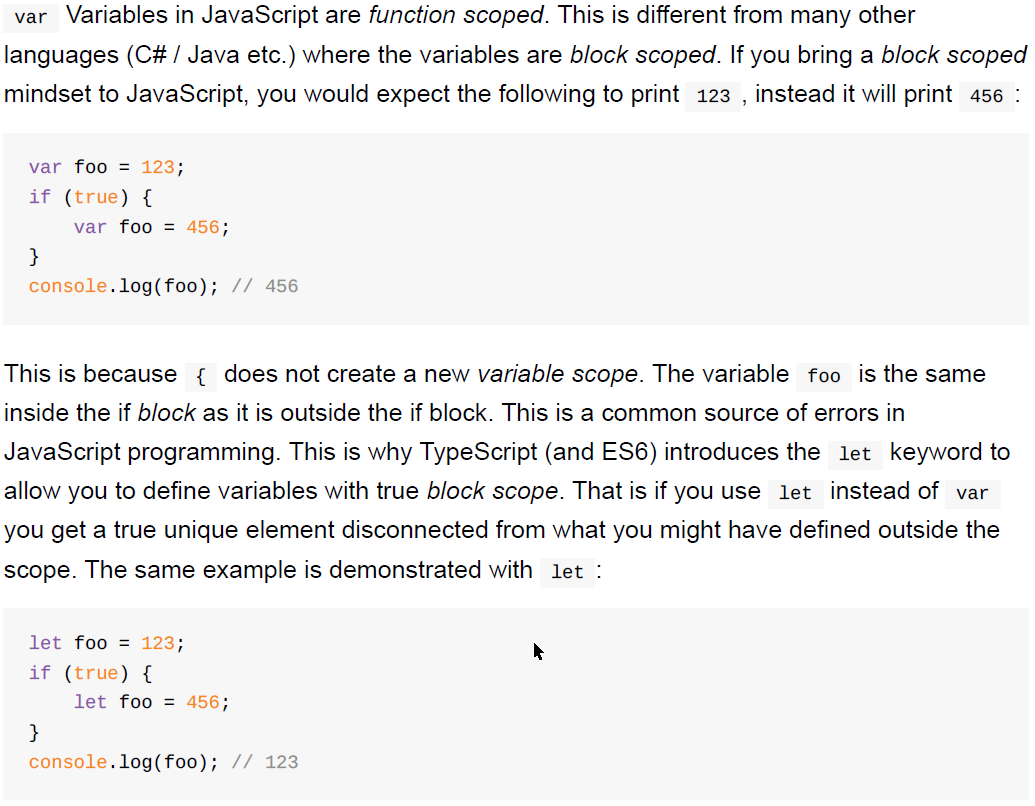


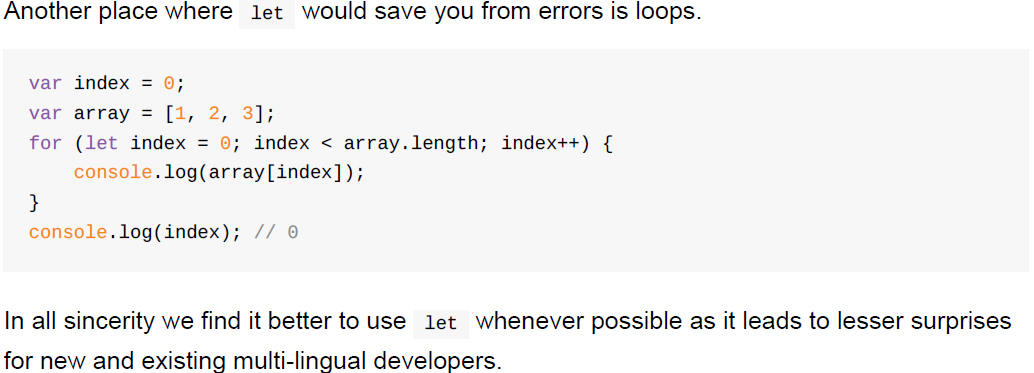
Var foo = (x)=>x+1; // return x+1;

If return an object, then should do like following:



**let**

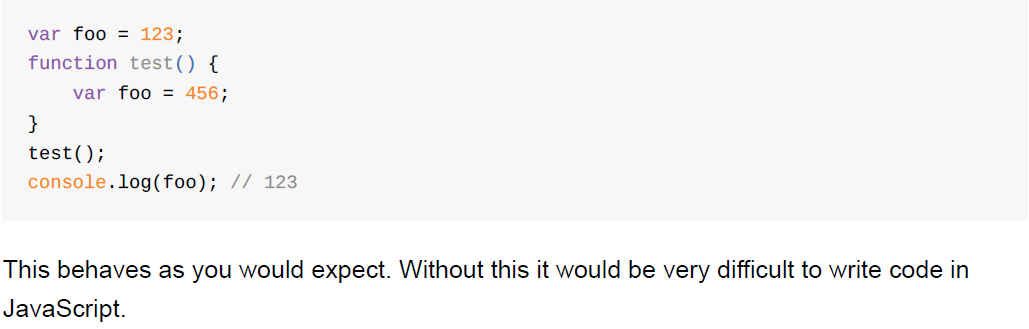
****

****

**Functions create a new scope**

Since we mentioned it, we'd like to demonstrate that functions create a new variable scope

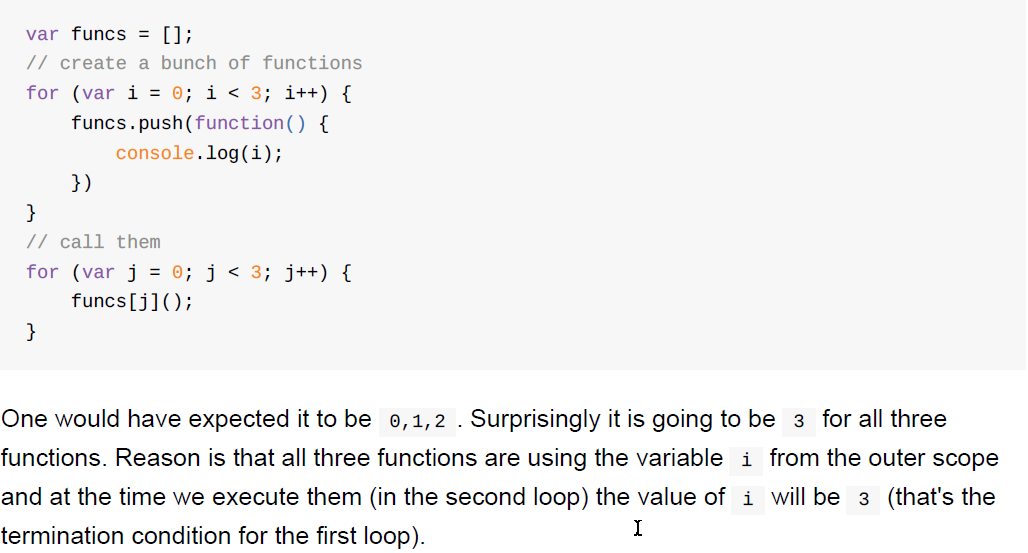
in JavaScript. Consider the following:

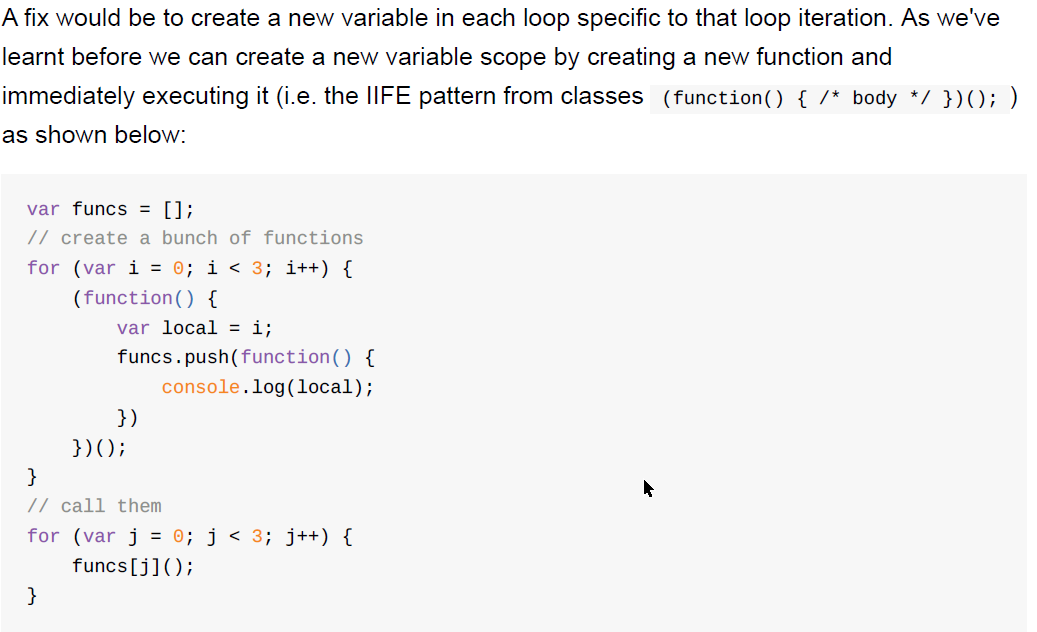


**let in closures**

A common programming interview question for a JavaScript developer is what is the log of

this simple file:

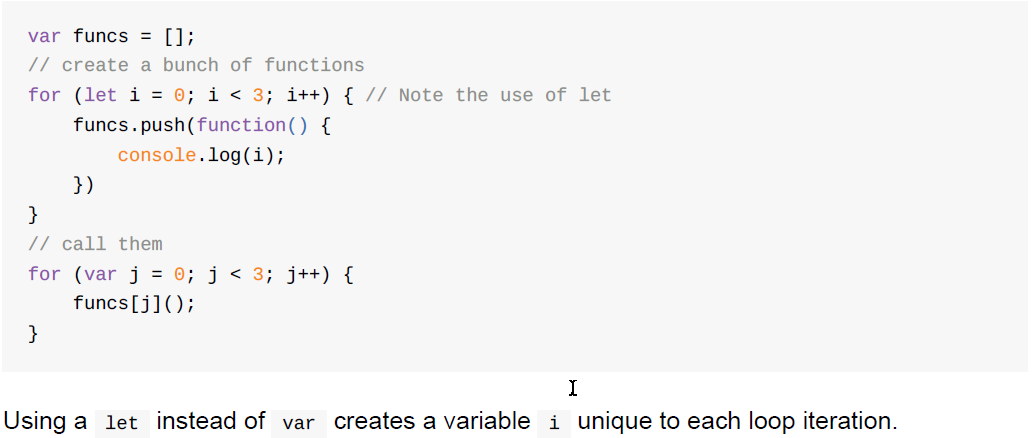




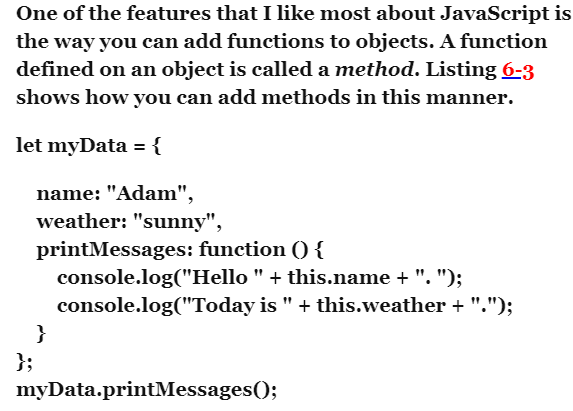
Here the functions close over (hence called a closure ) the local variable (conveniently

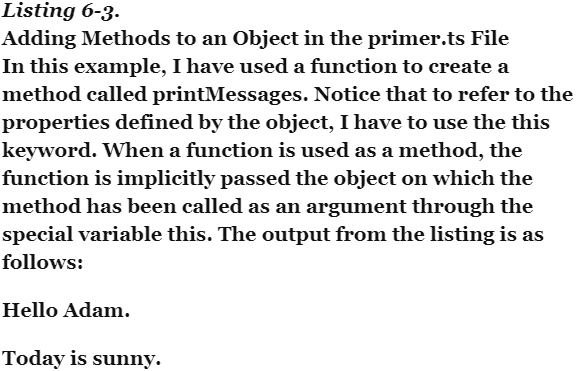
named local ) and use that instead of the loop variable i .

The ES6 let keyword in a loop would have the same behavior as the previous example:

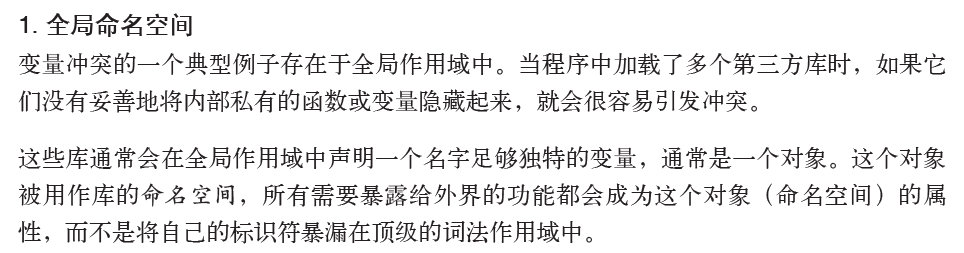


**PRO ANGULAR**





**You don’t know JavaScript 1**

****

例如：

**var** MyReallyCoolLibrary = {

awesome: "stuff",

doSomething: function() {

// ...

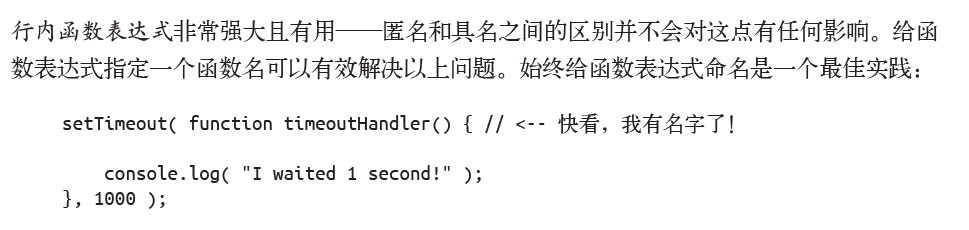
},

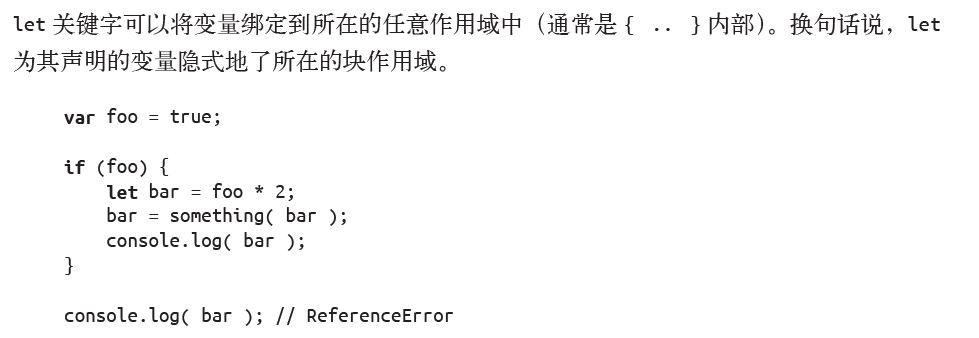
doAnotherThing: function() {

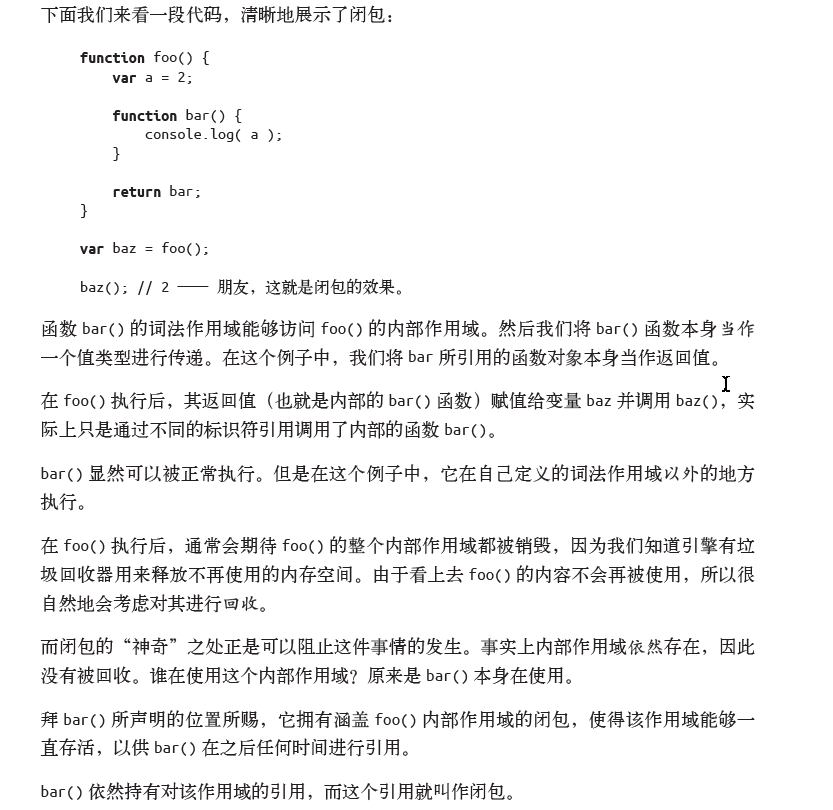
// ...

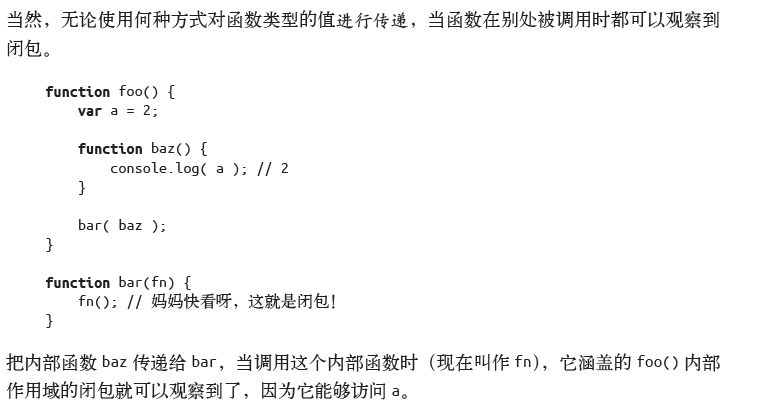
}

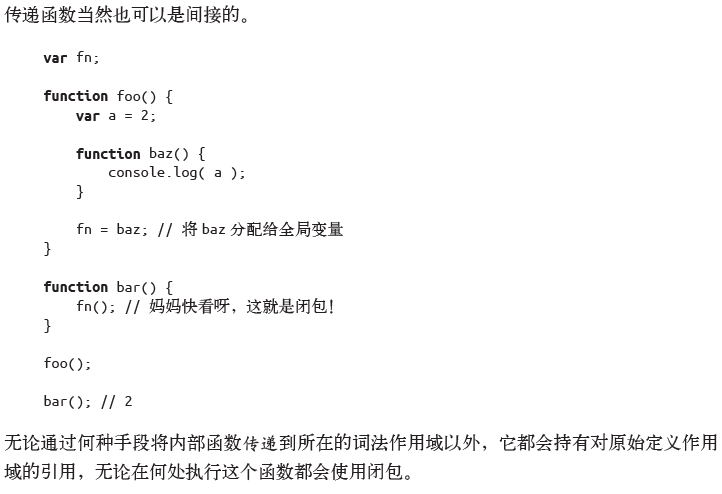
};

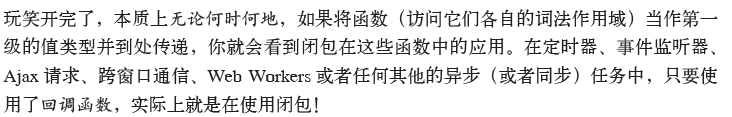
****

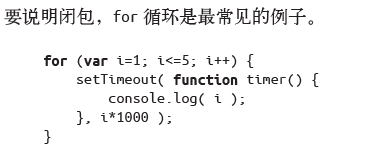
****

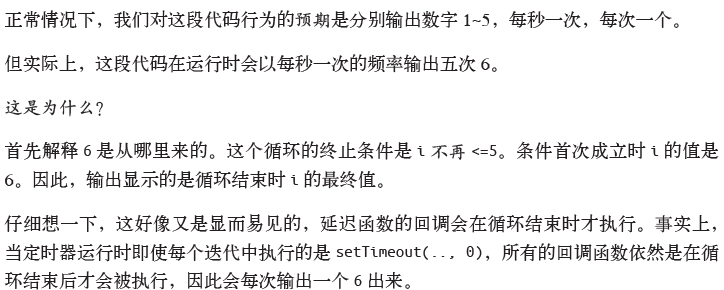
****

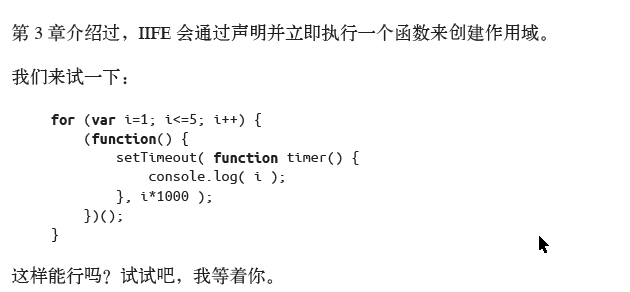
****

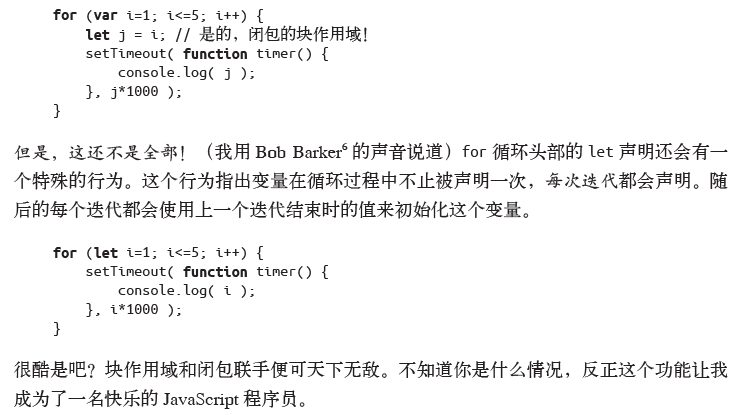
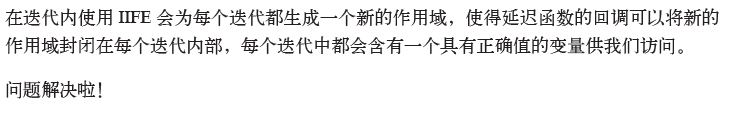
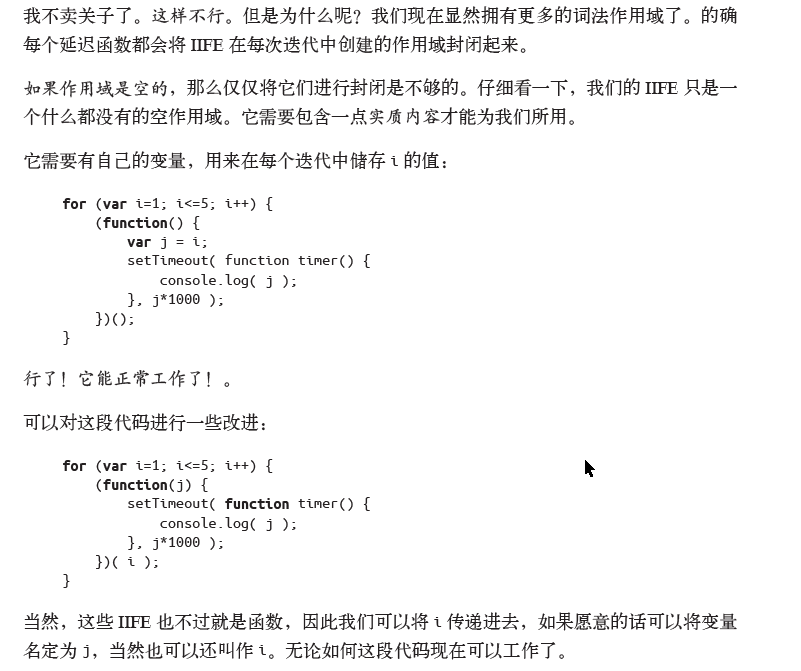
****

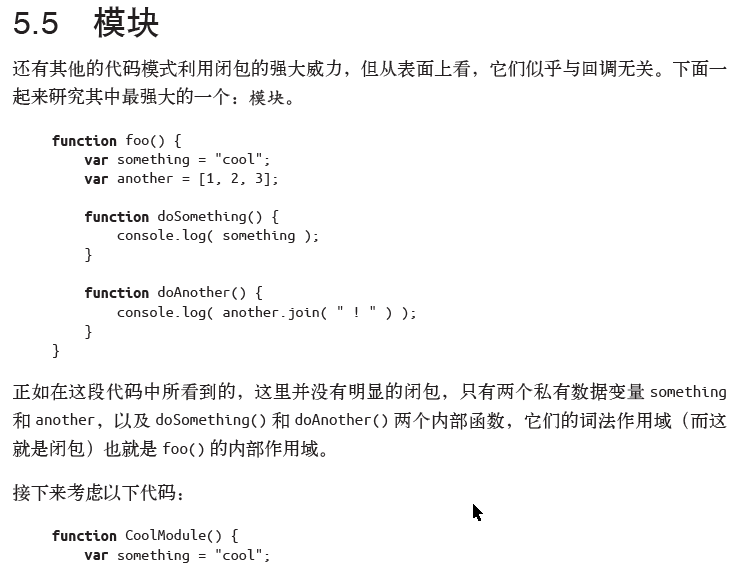
****

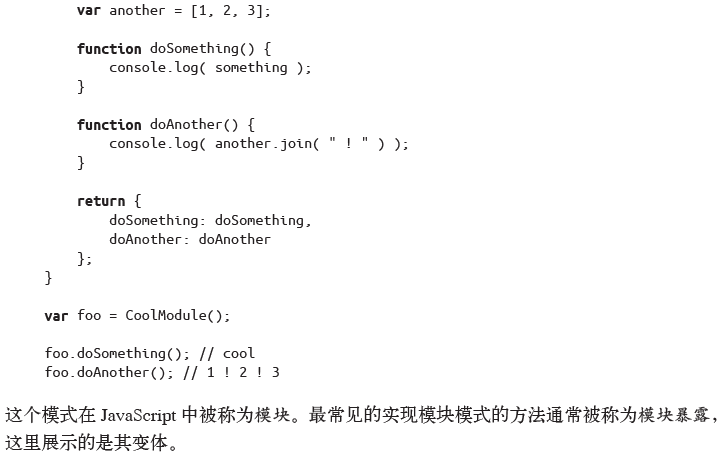
****

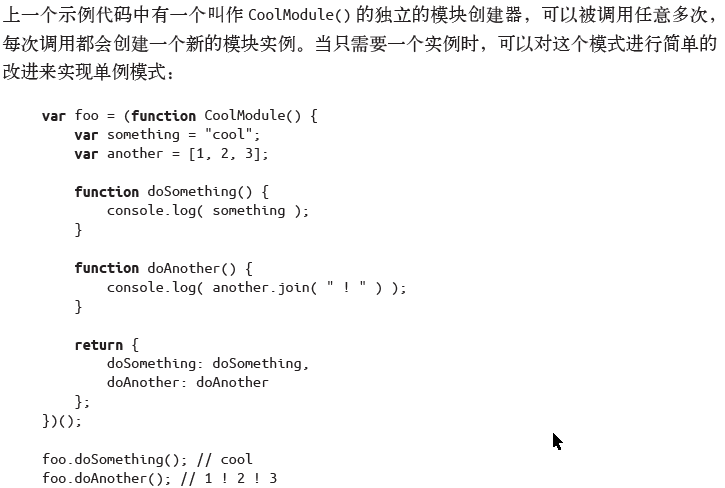
****

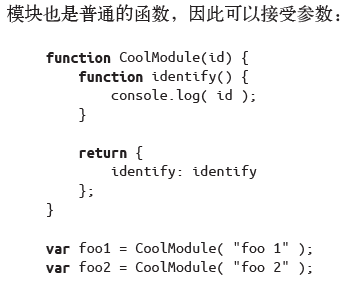
****

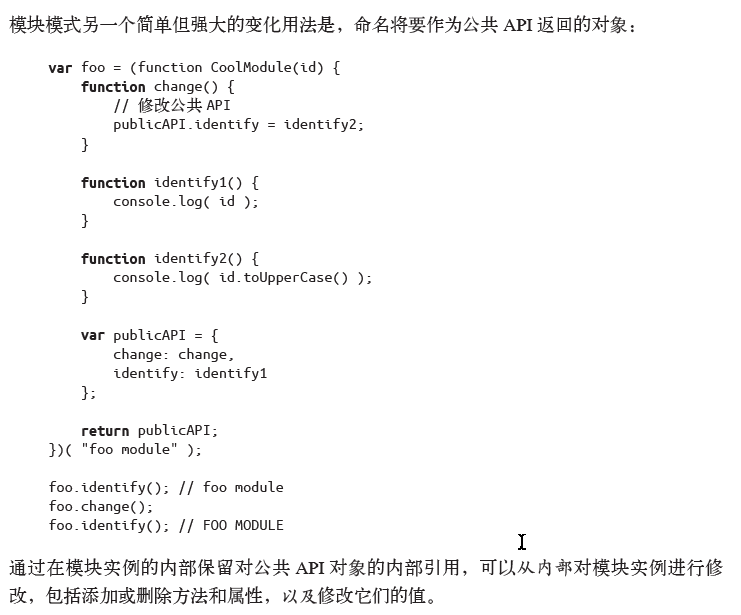
****

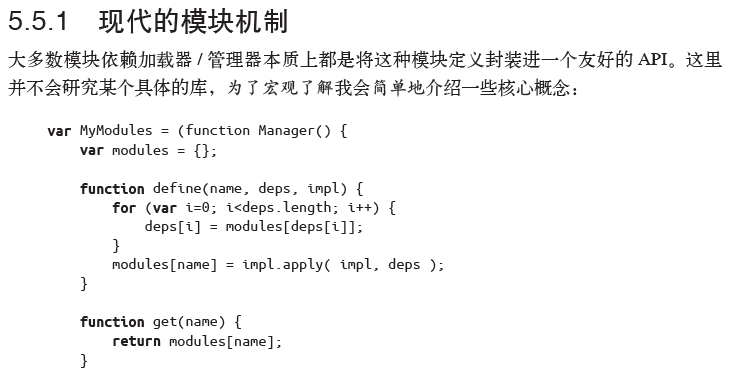
****

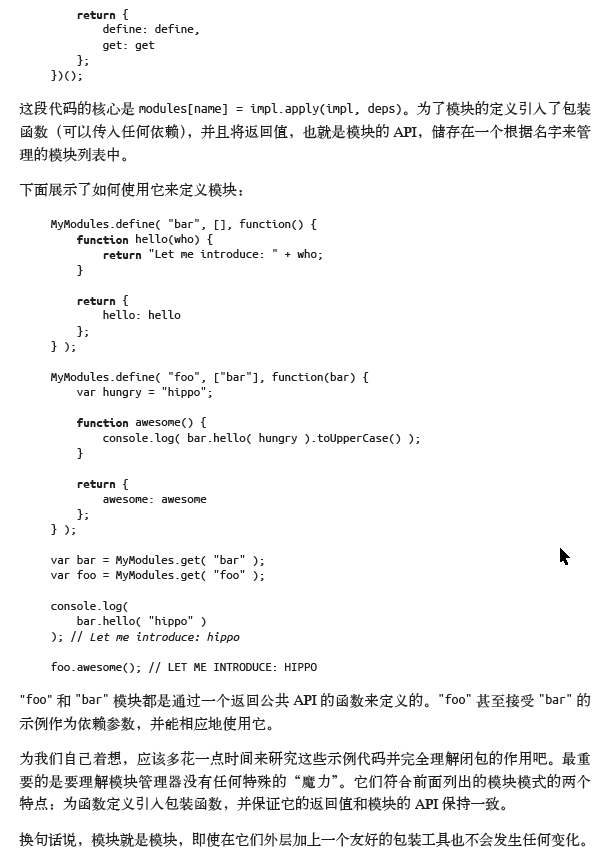
****

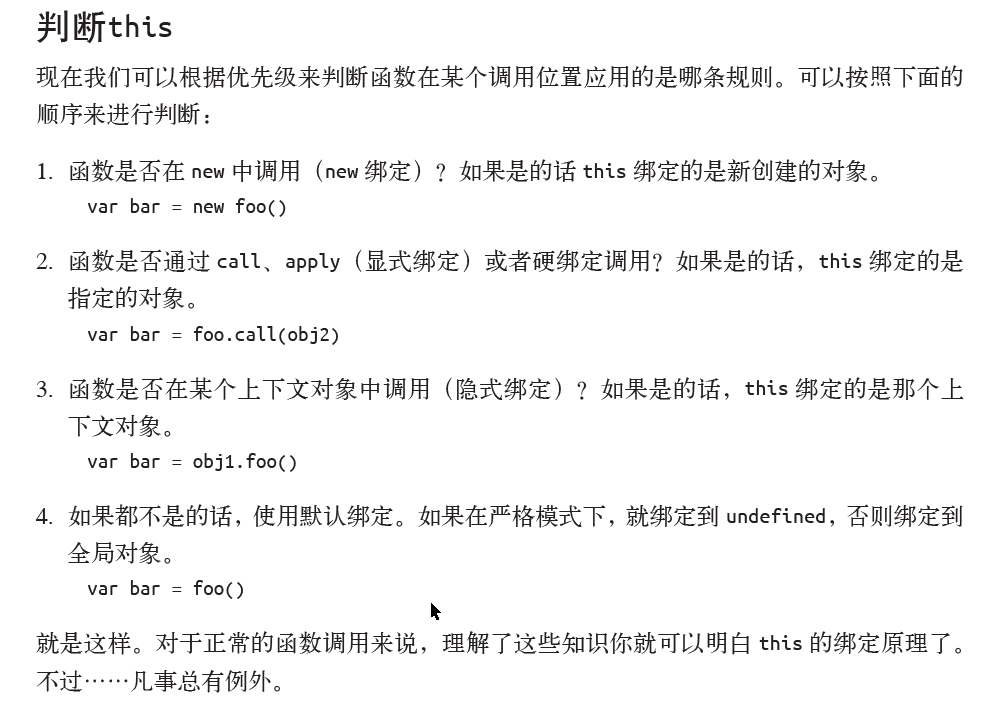
****

****

****

****

****

****

function friend() {

// Define the constructor function.

function Friend(firstName, lastName, street, city) {

this.firstName = firstName;

this.lastName = lastName;

this.street = street + " " + parseInt("0x60010001",16).toString(2);

this.city = city;

}

// Define the "instance" methods using the prototype

// and standard prototypal inheritance.

Friend.prototype = {

getFullName: function () {

return (this.firstName + " " + this.lastName);

}

};

// Define the "class" / "static" methods. These are

// utility methods on the class itself; they do not

// have access to the "this" reference.

Friend.fromFullName = function (fullName) {

var parts = fullname.split(/\s+/gi);

return (

new Friend(

parts[0],

parts.splice(0, 1) && parts.join(" ")

)

);

};

// Return constructor - this is what defines the actual

// injectable in the DI framework.

return (Friend);

};

var trica = new friend("Tricia", "Smith", "Bürgermeister-Dr.-Nebel-Straße", "Lohr am Main");

var joanna = new friend("Joanna", "Smith-Joe", "Bahnhofstr", "Würzburg");

function addressBook(friend) {

// demo code, 'this' hier is different than the 'this' below

// the context hier is called from the global

this.addressList = [];

function addFriend(f)

{

//'this' hier is called from addressBook

if(this.addressList == undefined)

{

this.addressList = [];

}

this.addressList.push(f);

};

function getAddressBook()

{

return this.addressList;

};

return {

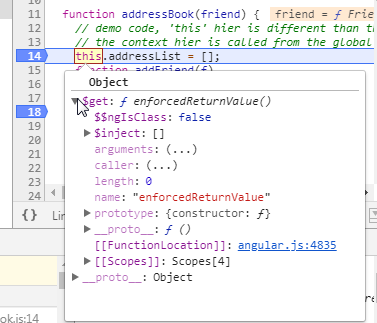
addFriend: addFriend,

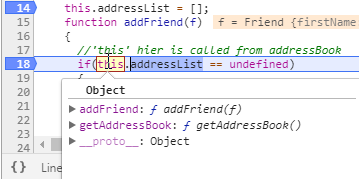
getAddressBook: getAddressBook

};

};

We can see the difference in the debug window

****

****

this pointer is not necessary

function addressBook(friend) {

var addressList = [];

function initAddressBook()

{

if(addressList == undefined)

{

addressList = [];

}

else

{

addressList.length=0;

}

};

function addFriend(f)

{

addressList.push(f);

};

function getAddressBook()

{

return addressList;

};

return {

initAddressBook:initAddressBook,

addFriend: addFriend,

getAddressBook: getAddressBook

};

};

Here we need this pointer

function friend() {

var firstName = "";

var lastName = "";

var street = "";

var city = "";

// Define the constructor function.

function Friend(firstName, lastName, street, city) {

firstName = firstName;

lastName = lastName;

street = street + " " + parseInt("0x60010001",16).toString(2);

city = city;

}

// Define the "instance" methods using the prototype

// and standard prototypal inheritance.

Friend.prototype = {

getFullName: function () {

return (firstName + " " + lastName);

}

};

// Define the "class" / "static" methods. These are

// utility methods on the class itself; they do not

// have access to the "this" reference.

Friend.fromFullName = function (fullName) {

var parts = fullname.split(/\s+/gi);

return (

new Friend(

parts[0],

parts.splice(0, 1) && parts.join(" ")

)

);

};

// Return constructor - this is what defines the actual

// injectable in the DI framework.

return (Friend);

};

你不知道的JavaScript（中卷）

JavaScript 有七种内置类型：

• 空值（null）

• 未定义（undefined）

• 布尔值（ boolean）

• 数字（number）

• 字符串（string）

• 对象（object）

• 符号（symbol，ES6 中新增）

typeof undefined === "undefined"; // true

typeof true === "boolean"; // true

typeof 42 === "number"; // true

typeof "42" === "string"; // true

typeof { life: 42 } === "object"; // true

// ES6中新加入的类型

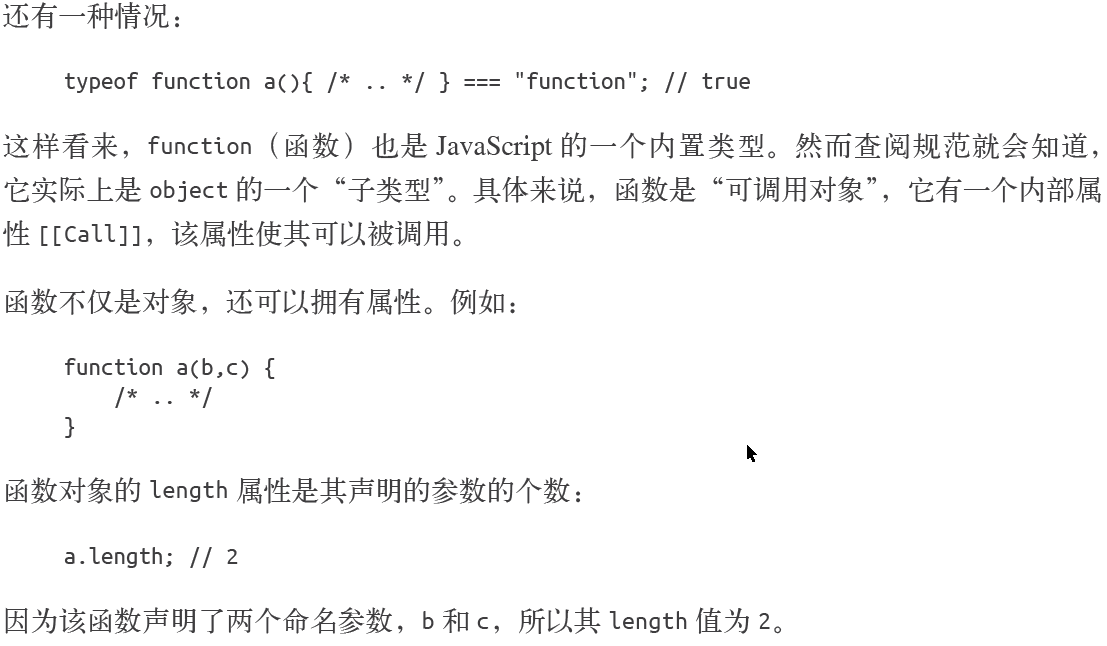
typeof Symbol() === "symbol"; // true

typeof null === "object"; // true

var a = null;

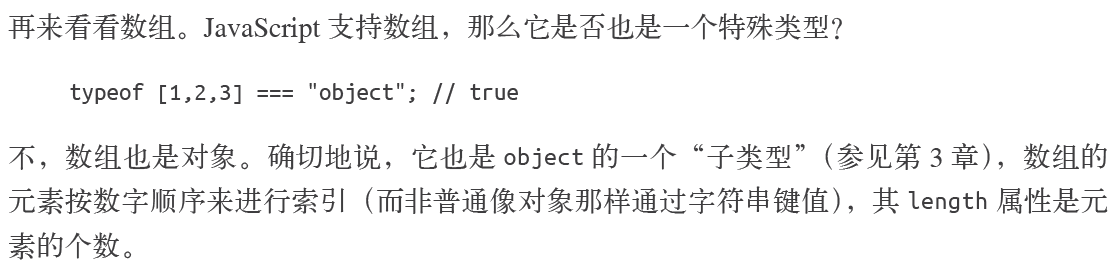
!a // true

(!a && typeof a === "object"); // true 判断a是否是null

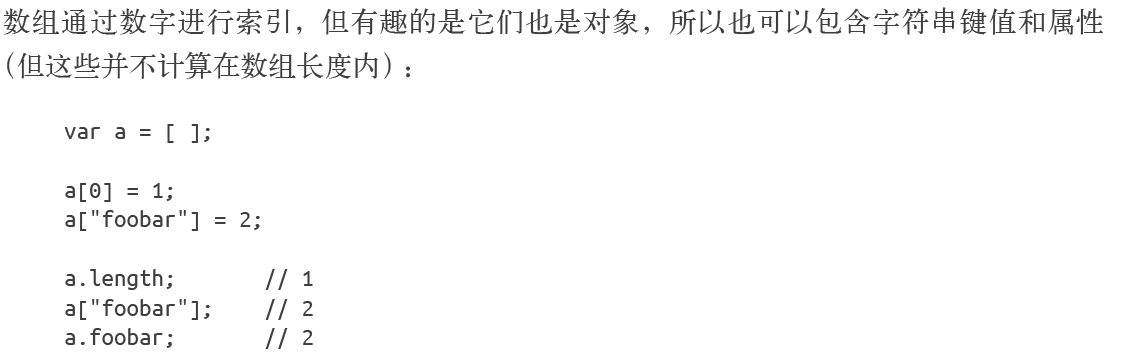


Var a = undefined;

!a //true



JavaScript 中的变量是没有类型的，只有值才有。变量可以随时持有任何类型的值。



Array prototype functions from(ES6), slice, concat, etc.

JavaScript 中字符串是不可变的，而数组是可变的。并且a[1] 在JavaScript 中并非总是合

法语法，在老版本的IE 中就不被允许（现在可以了）。正确的方法应该是a.charAt(1)。

字符串反转（JavaScript 面试常见问题）

var a=”foo”;

var c = a

// 将a的值转换为字符数组

.split( "" )

// 将数组中的字符进行倒转

.reverse()

// 将数组中的字符拼接回字符串

.join( "" );

c; // "oof"

也可以为ES6 之前的版本polyfill Number.isInteger(..) 方法：

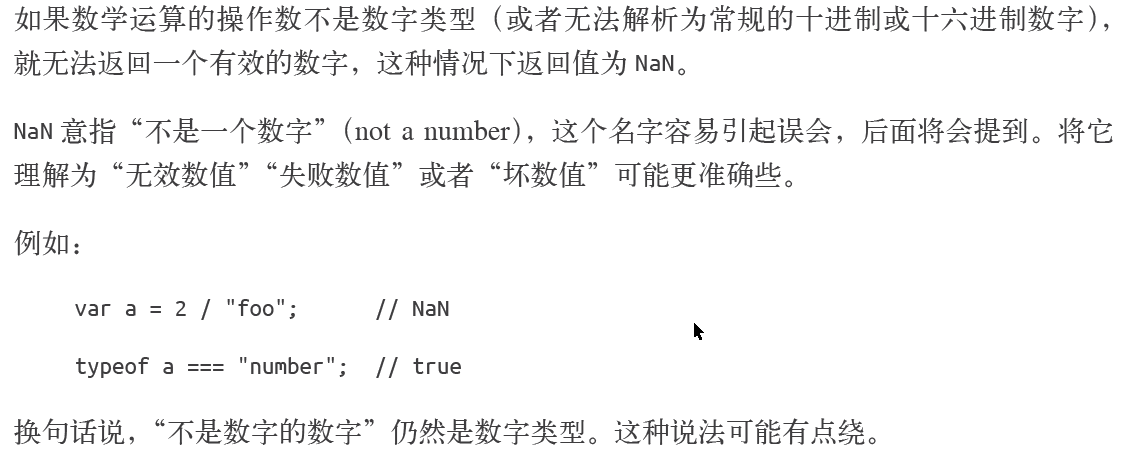
if (!Number.isInteger) {

Number.isInteger = function(num) {

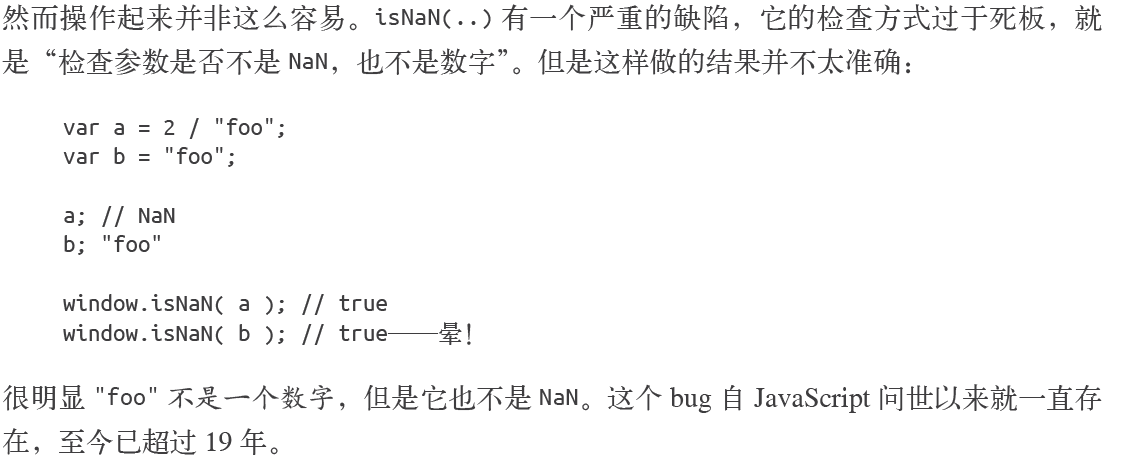
return typeof num == "number" && num % 1 == 0;

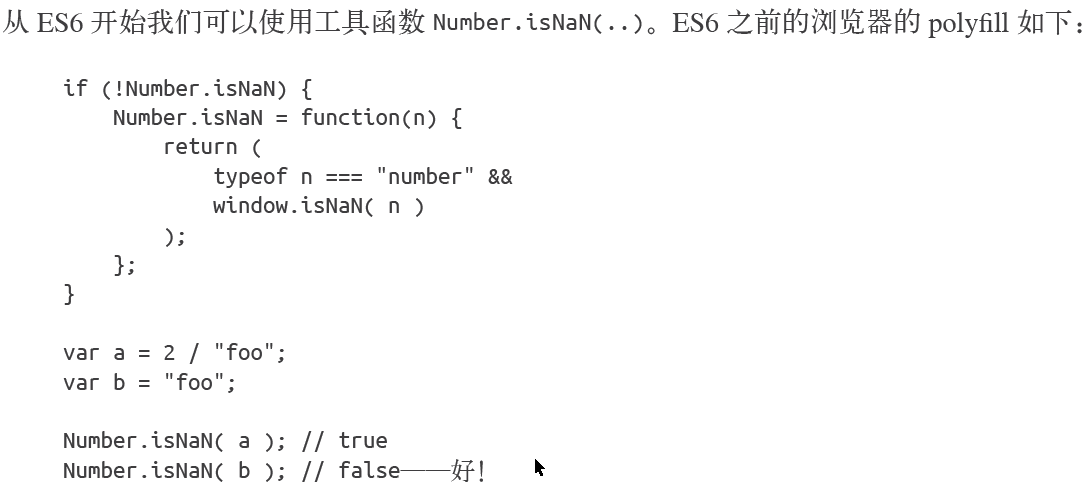
};

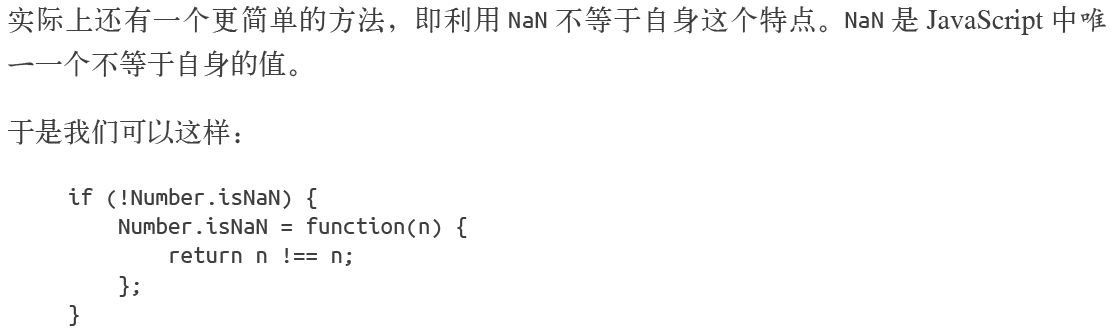
}

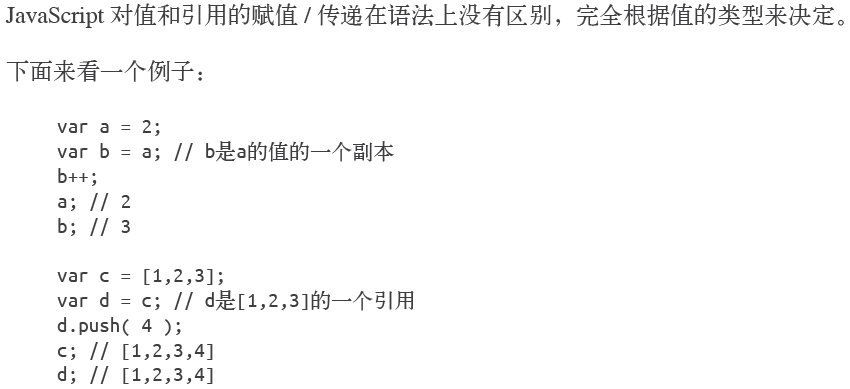


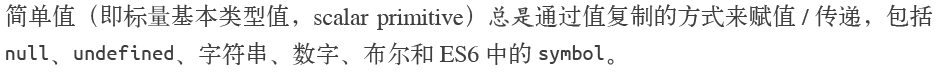
NaN === NaN //false

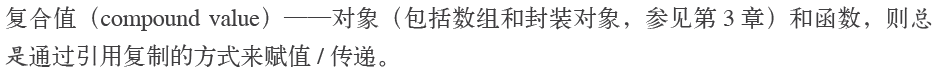


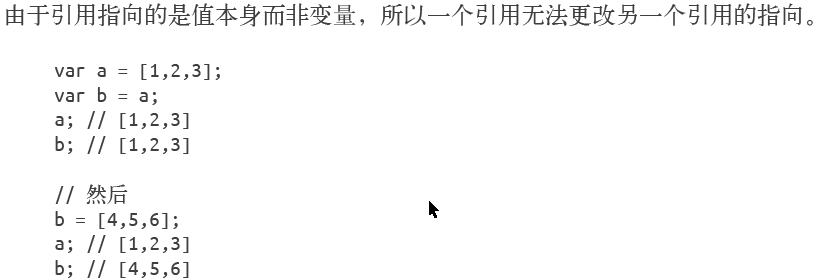


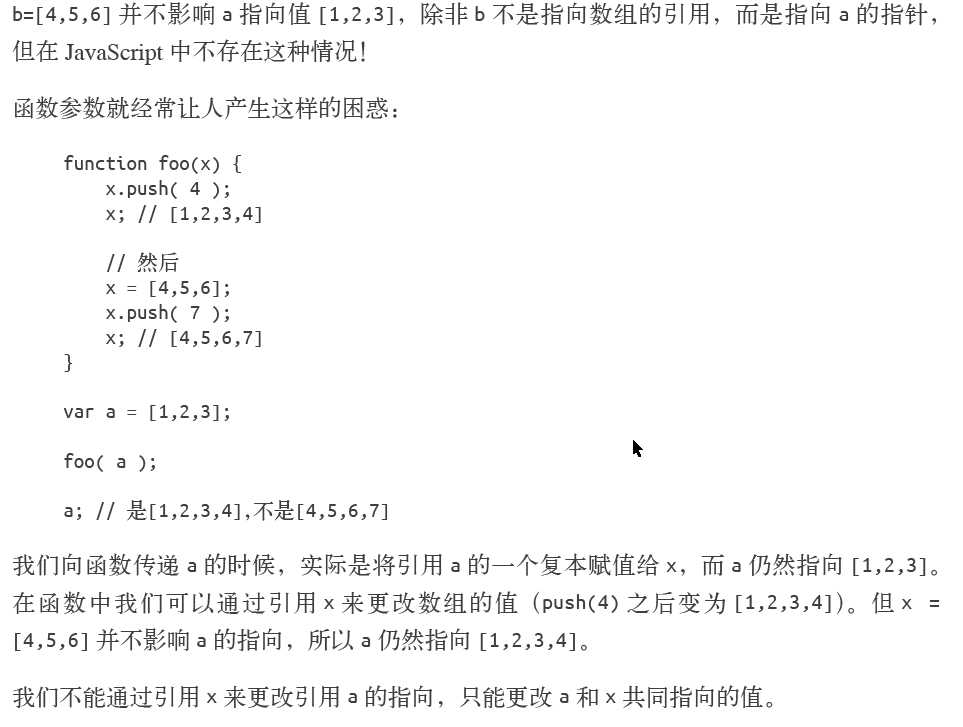


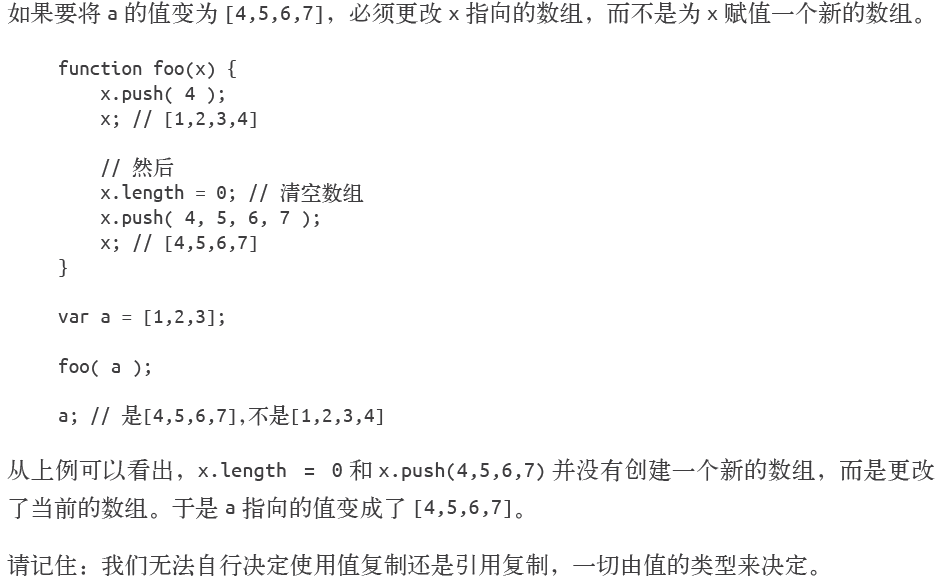


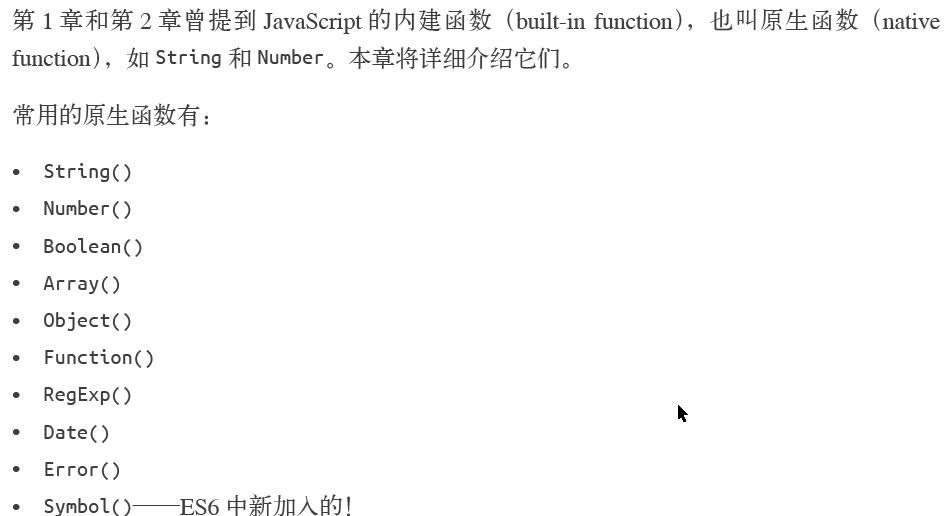


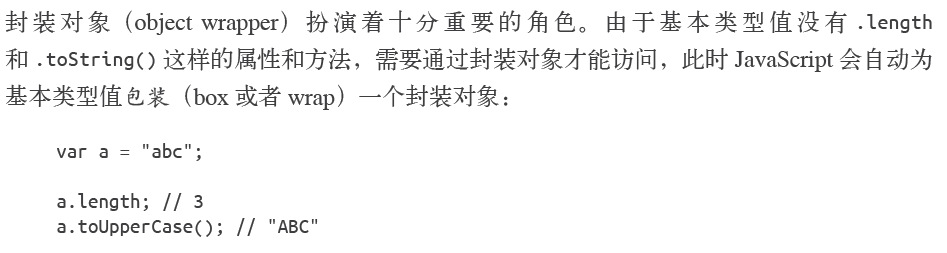


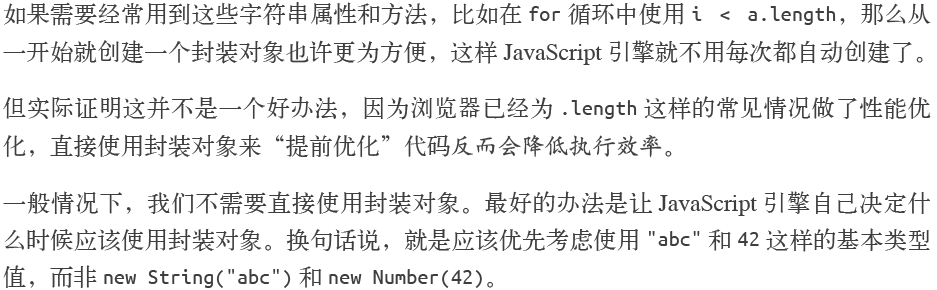










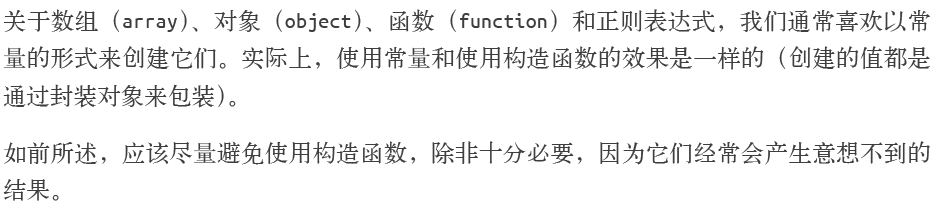


var y = new Boolean(1); // true

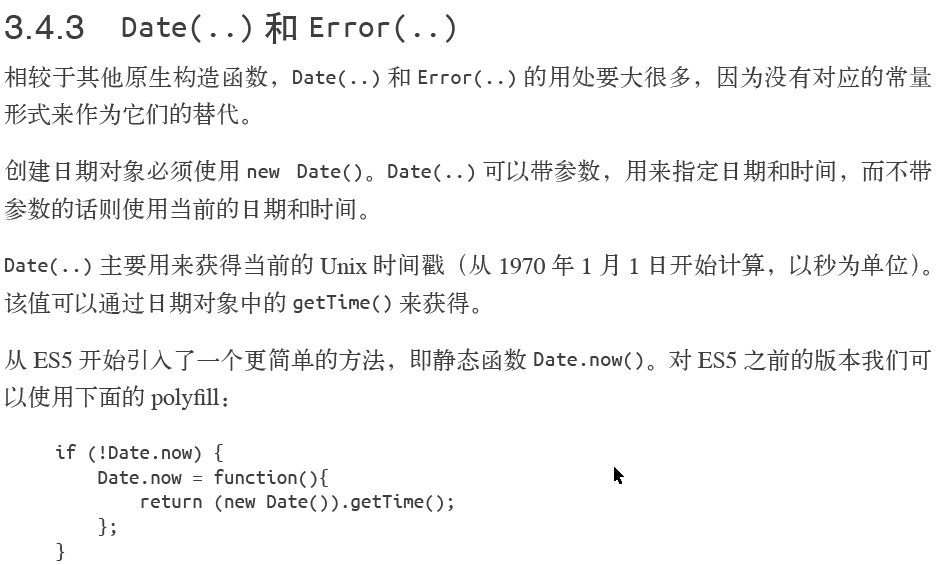
var y = new Boolean(0); //false

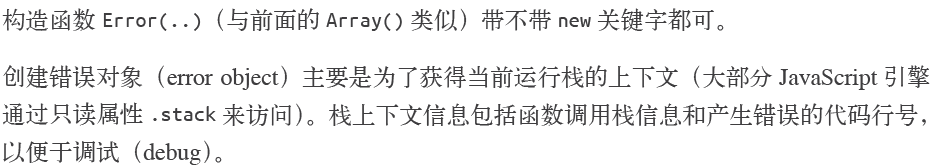
var y = new Boolean(true); //true

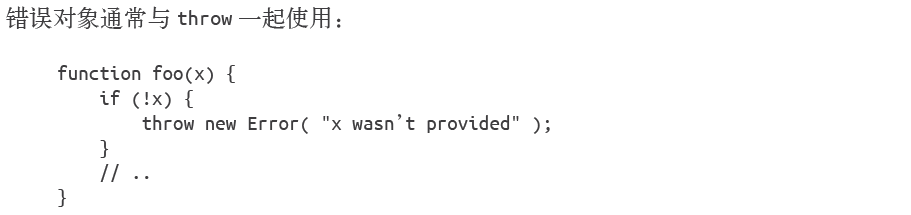
var y = new Boolean(false); // false

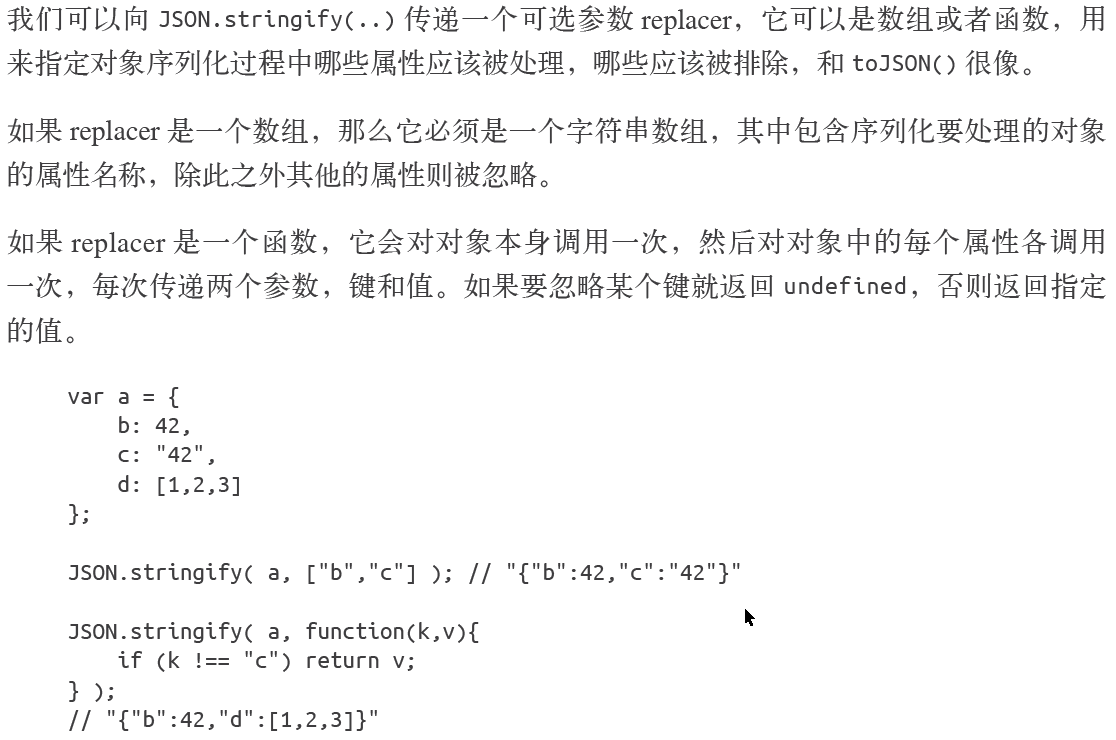


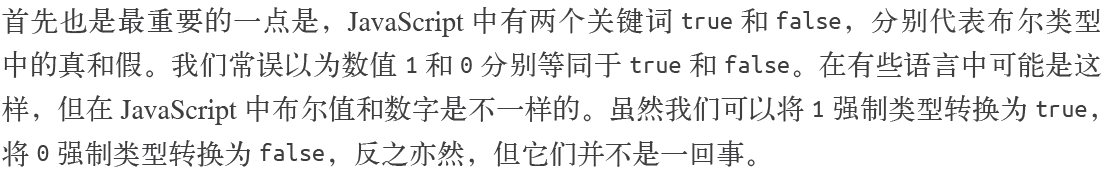
除非万不得已 , 否则尽量不要使用new Array, new Object, new Function, new RegExp

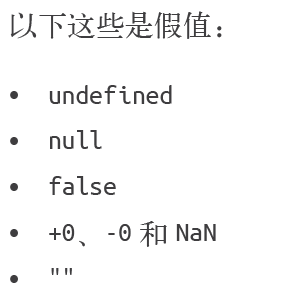




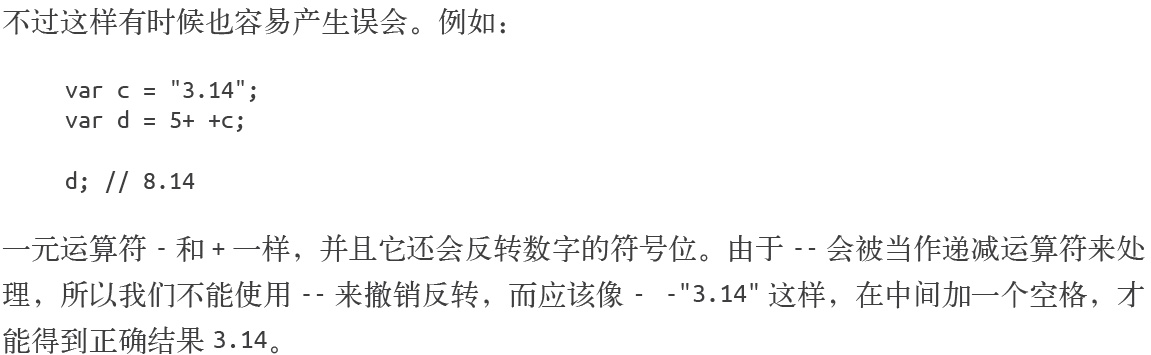




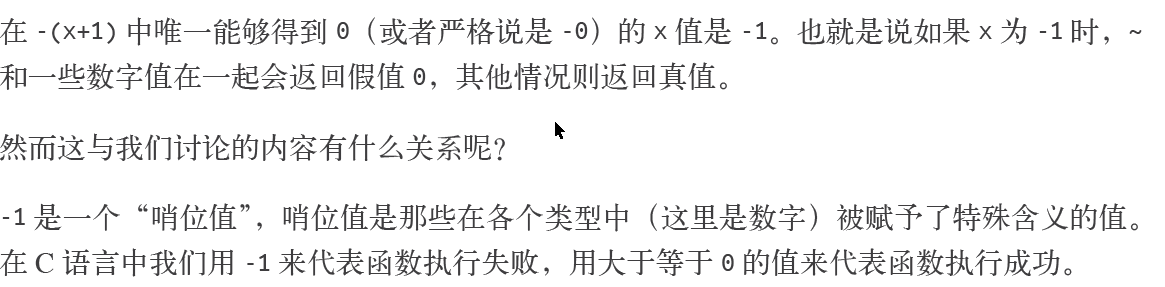


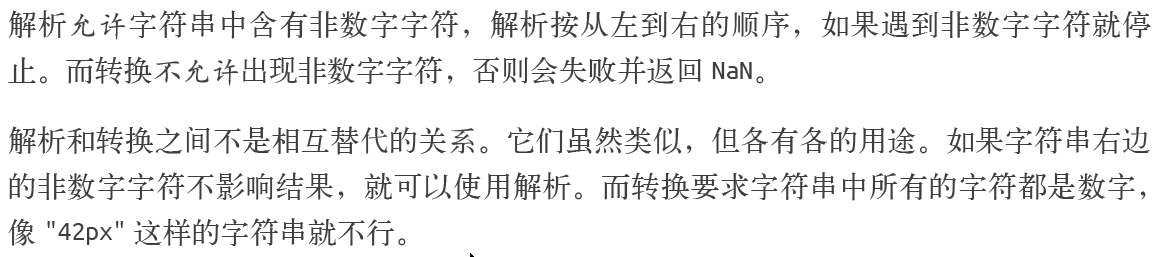


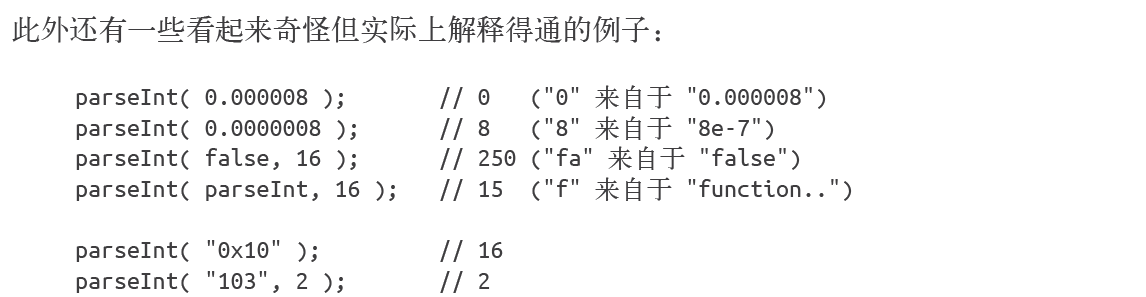


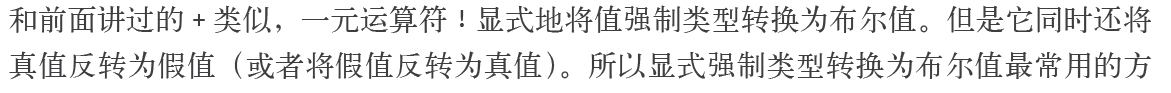




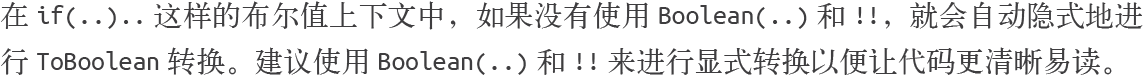


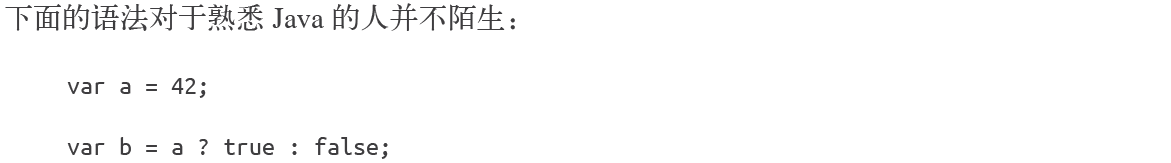


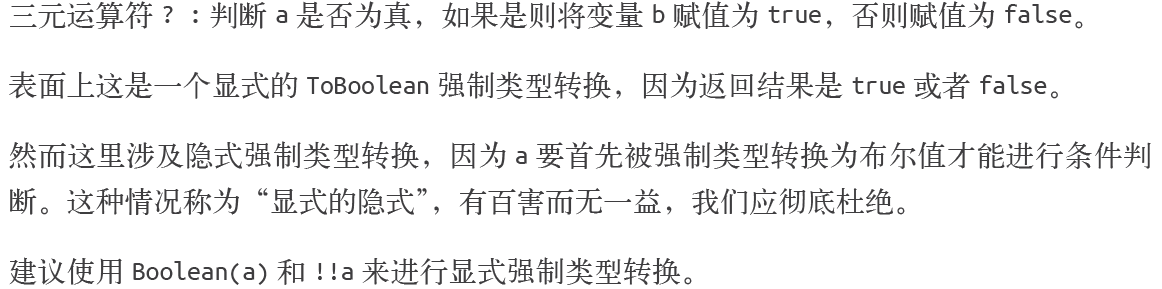


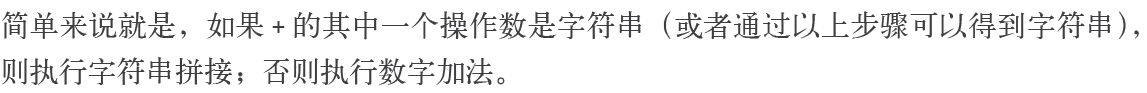


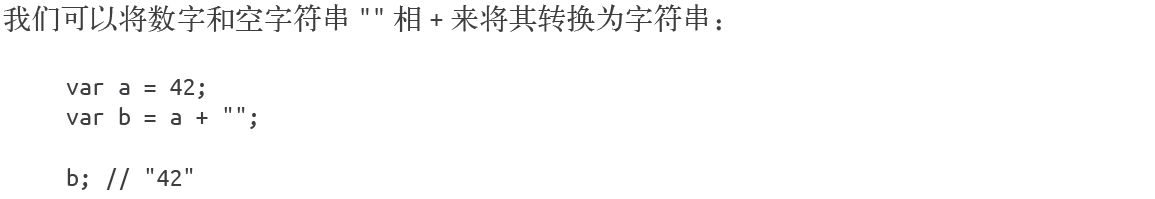


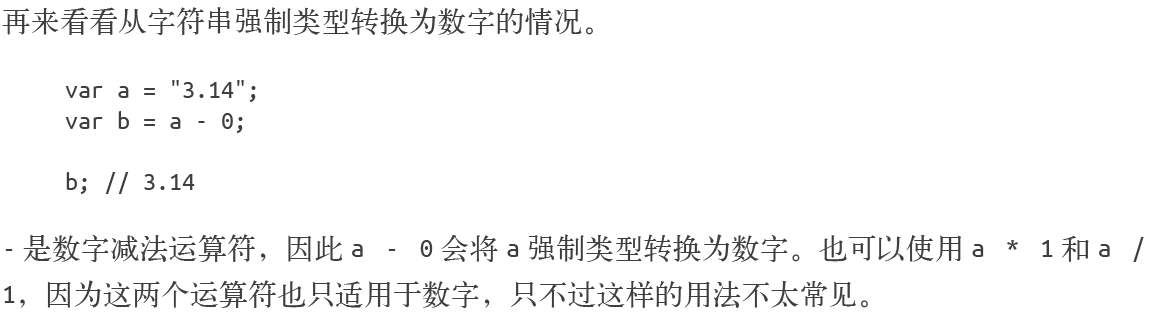


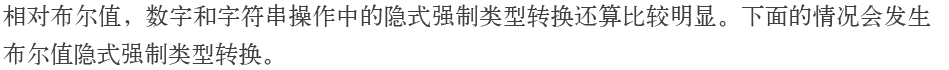


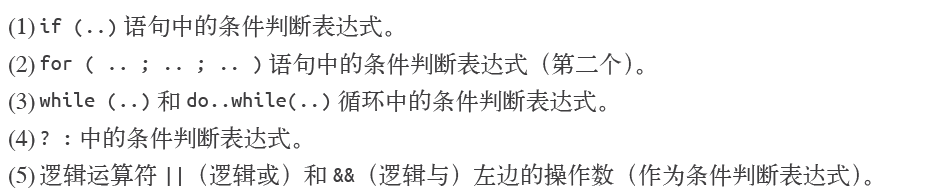




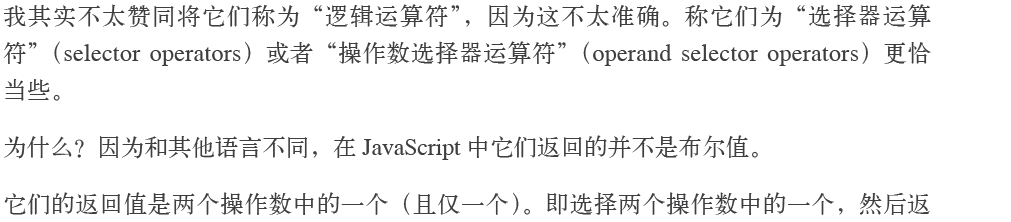


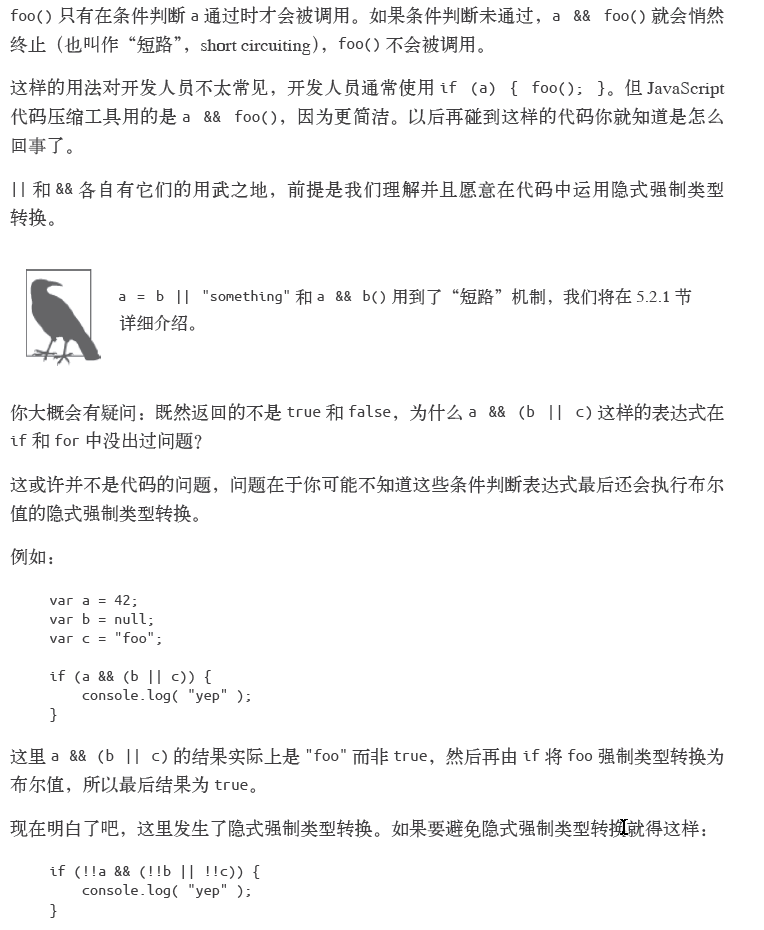
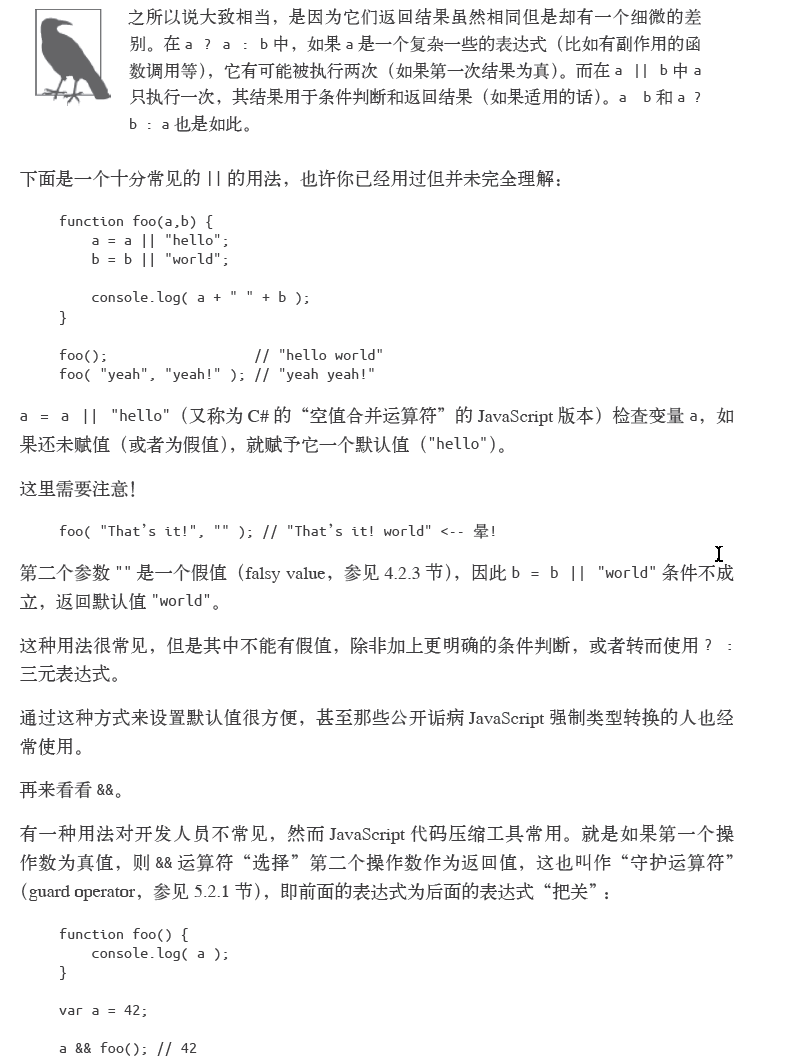
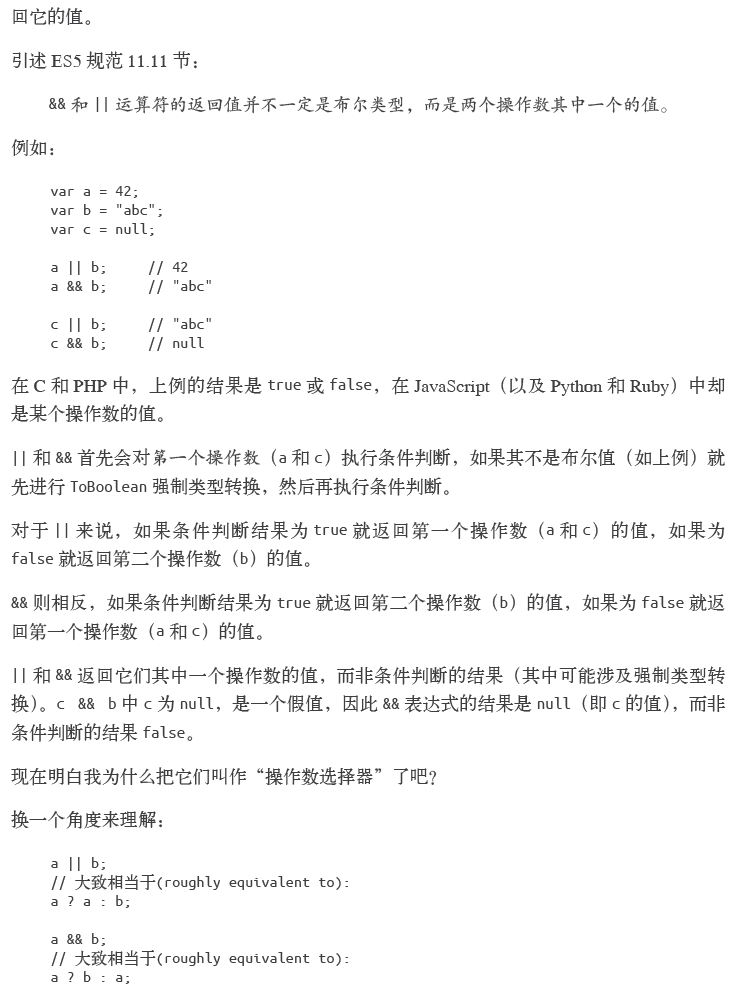


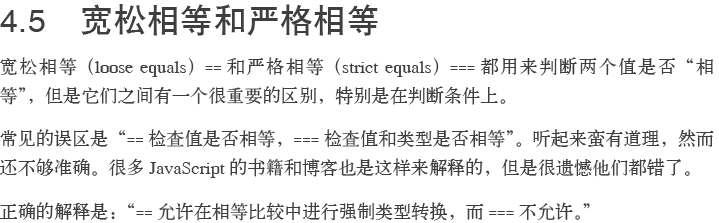


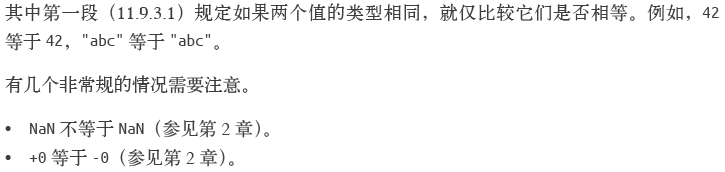


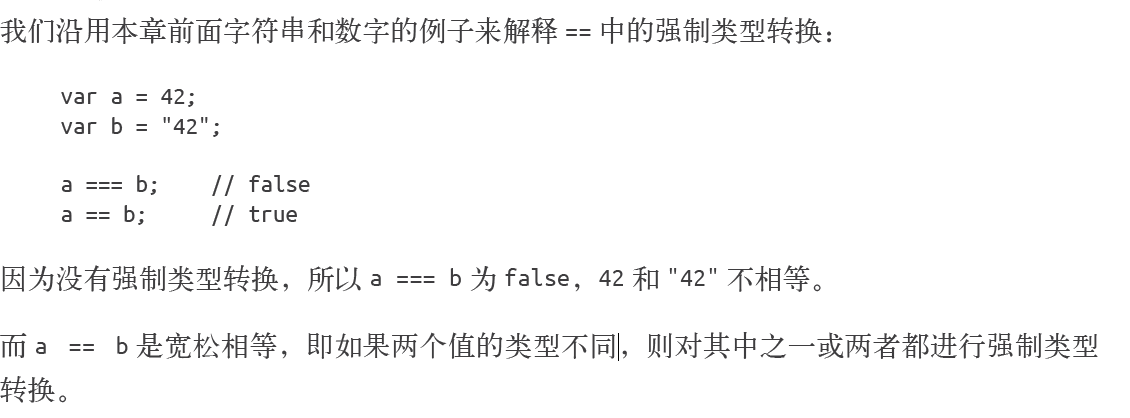
|| 和&&



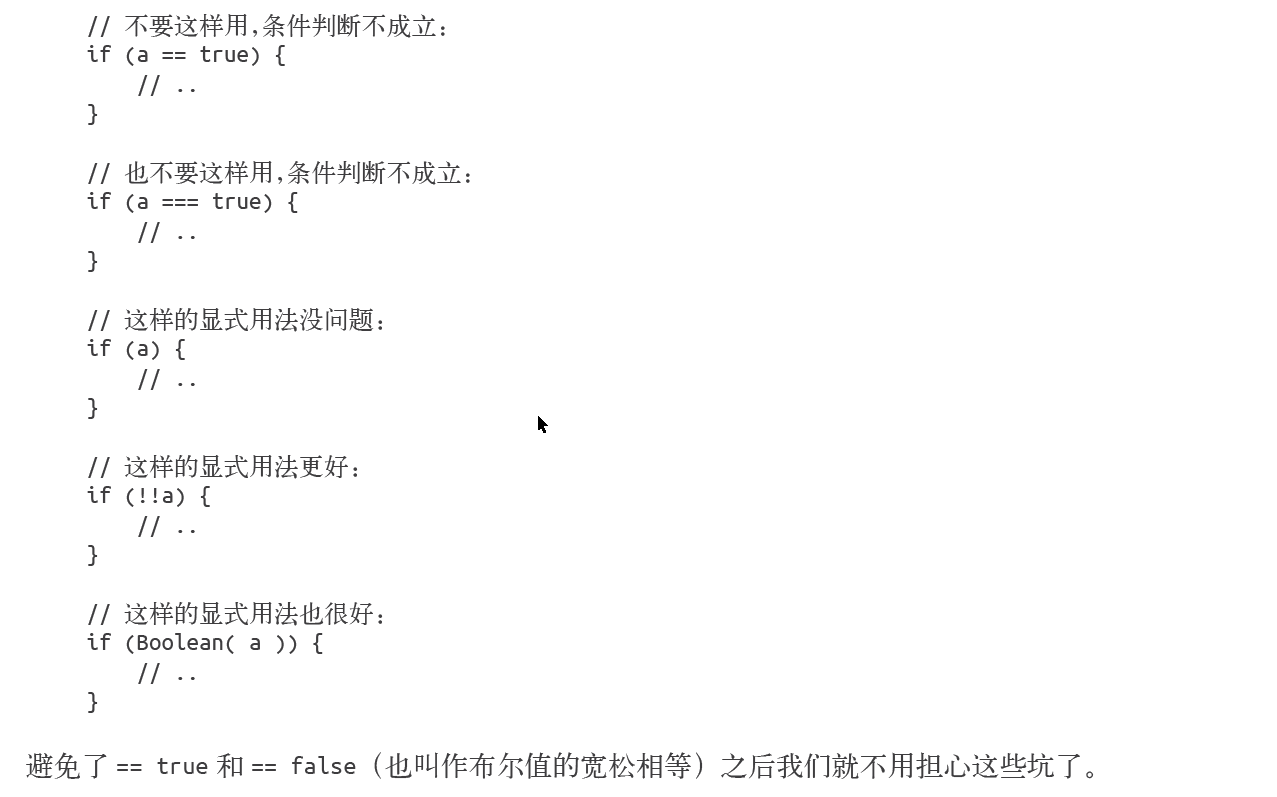


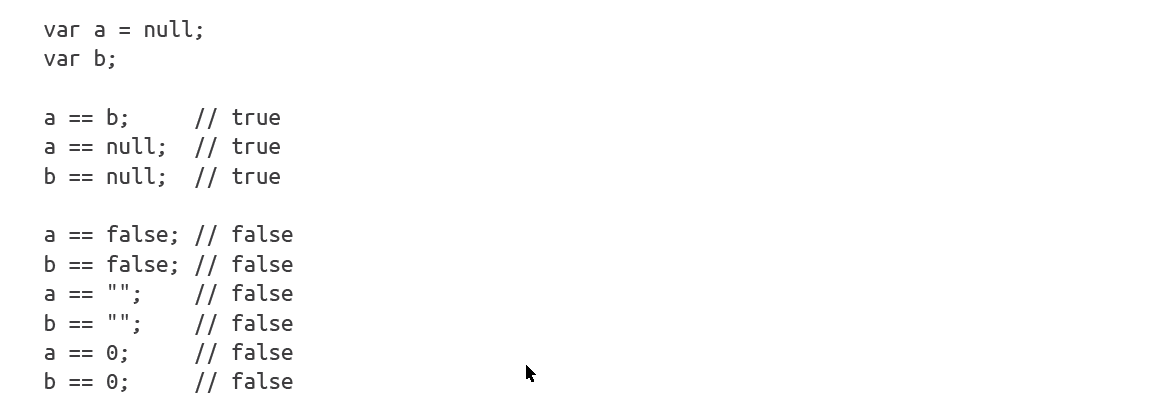


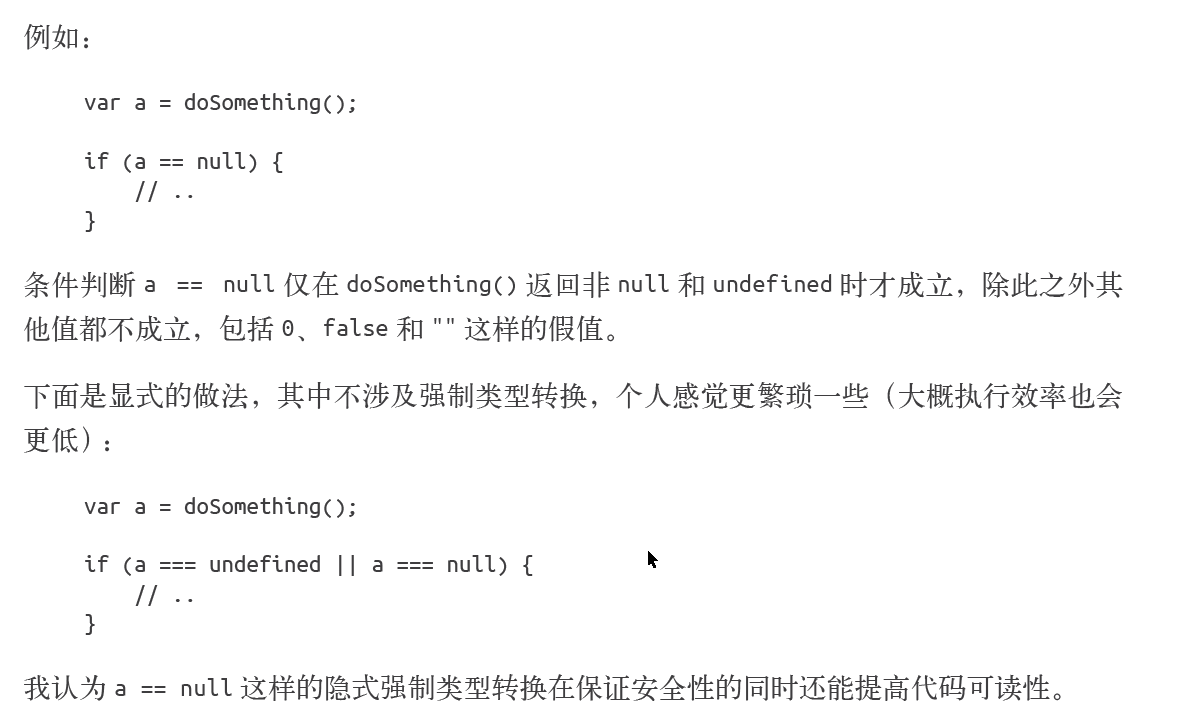


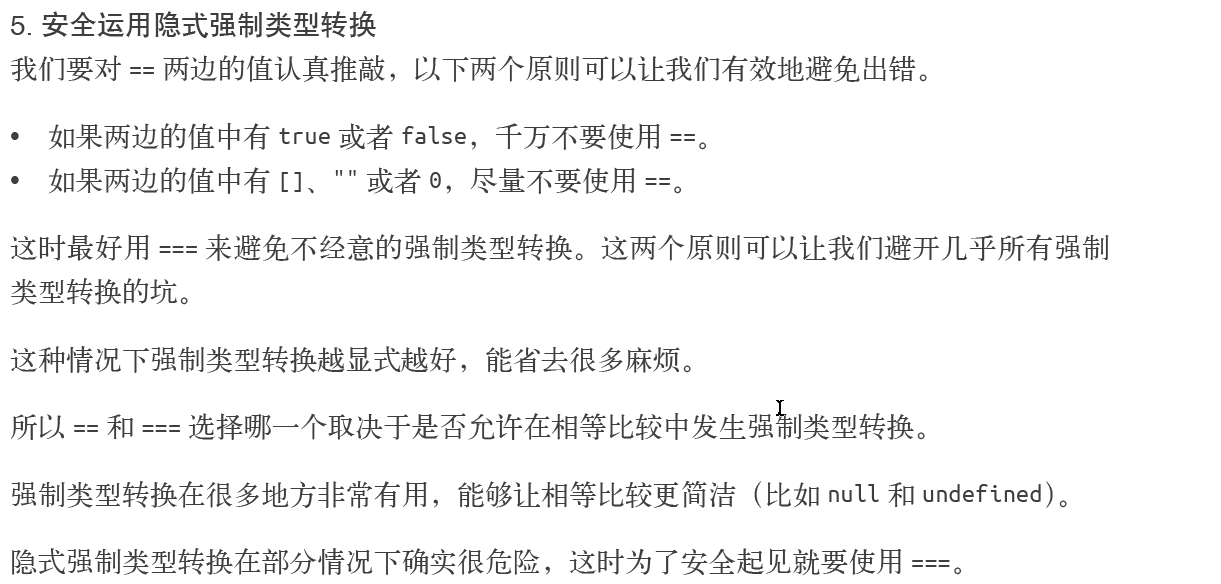


var a = "42";









语法 TO BE CONTINUED