# 函数

## 递归

一、函数调用

function isPalindrome( text ) {

if ( text.length <=1 ) return true

if( text.charAt(0) !== text.charAt(text.length -1) ) return false

return isPalindrome( text.substr( 1, text.length - 2 ) )

}

二、方法调用

var ninja = {

chirp: function(n) {

return n > 1 ? ninja.chirp(n - 1) + "-chirp" : "chirp";

}

};

var samurai = { chirp: ninja.chirp };

**优化**

return n > 1 ? this.chirp(n - 1) + "-chirp" : "chirp";

1. **内联命名函数**

上面的问题在于，其他对象引用时必须给方法命名为chirp，而用内联命令函数则不会有这个问题

var ninja = {

chirp: function signal(n) { //#1

return n > 1 ? signal(n - 1) + "-chirp" : "chirp";

}

};

内联函数的名称在函数外是不可用的。只在函数内部是可用的。如：

var a = function b() {

}

b -- 》 undefined

这也是为什么全局属性会被作为window的方法进行创建的原因，不使用window的属性，无法引用这些函数。

注意： arguments.callee() 也可以调用当前的函数，但是，es5规范中禁止使用这个属性。

## 函数存储与对象存储

问题：如何存储一组独立的函数？

关键： 可以将函数视为对象，在其上添加属性

函数属性存储优势

将状态和缓存信息存储在独立位置

利于代码组织

外部存储或缓存对象无需污染作用域

var store = {

nextId: 1, //#1

cache: {}, //#2

add: function(fn) { //#3

if (!fn.id) { //#3

fn.id = store.nextId++; //#3

return !!(store.cache[fn.id] = fn); //#3

} //#3

}

};

**自记忆函数**

用于缓存记忆昂贵的计算结果

function isPrime(value) {

if (!isPrime.answers) isPrime.answers = {}; //#1

if (isPrime.answers[value] != null) { //#2

return isPrime.answers[value]; //#2

} //#2

var prime = value != 1; // 1 can never be prime

for (var i = 2; i < value; i++) {

if (value % i == 0) {

prime = false;

break;

}

}

return isPrime.answers[value] = prime; //#3

}

但是也要权衡一下缺点

节省了计算性能，但是任何类型的缓存肯定会牺牲内存。

缓存是否应该与业务逻辑放在一起？（函数或方法应该只做一件事？）

缓存记忆DOM元素

function getElements( name ) {

if( !getElements.cache ) getElements.cache = {};

return getElements.cache[name] ||

document.getElementsByTagName(name)

}

类数组缓存

var elems = {

length: 0,

add: function(elem) {

Array.prototype.push.call(this, elem)

},

gather: function(id) {

this.add(document.getElementById(id))

}

}

## 利用参数进行函数重载

由于js函数没有重载的概念，可以借用参数模仿重载

注：函数的length属性，可以知道声明了多少形参

arguments.length属性，可以知道传入了多少实参

1. 更具实参个数进行判断进而重载

## 函数判定

如何判断一个对象是一个函数的实例，并且可以调用

1. typeof 的问题

一个非期望的函数结果返回

safari 返回nodeList为function

ie 。。。

1. Object.prototype.toString.call(fn) === ‘[object Function]’

返回一个对象内部的描述符

IE会将DOM元素的方法报告为object（测试情况不符合）

# 闭包

## 闭包如何工作

一个函数在创建时，允许该函数访问并操作该函数之外的变量所创建的作用域。

也就是说：函数调用时可以访问，该函数声明时的作用域里的变量和函数。

实际上是一种状态封装

在某一方面保护了，函数执行完毕后要被垃圾回收的变量和函数

## 闭包使用

### 访问私有变量

访问函数内部的变量

function Ninja() { //#1

var feints = 0; //#2

this.getFeints = function(){ //#3

return feints; //#3

}; //#3

this.feint = function(){ //#4

feints++; //#4

}; //#4

}

var ninja = new Ninja();

### 回调与计时器回调

### 绑定函数上下文

function bind(context,name){ //#1

return function(){ //#1

return context[name].apply(context,arguments); //#1

}; //#1

} //#1

var button = {

clicked: false,

click: function(){

this.clicked = true;

assert(button.clicked,"The button has been clicked");

console.log(this);

}

};

var elem = document.getElementById("test");

elem.addEventListener("click",bind(button,"click"),false); //#2

### 偏应用函数

也就是柯里化函数，

柯里化：在一个函数中首先填充几个参数（然后在返回一个新函数）的技术就是柯里化

偏应用方程返回一个含有预处理参数的新函数