# 一 jQuery事件

## 1 jQuery事件机制

jQuery的事件机制：

指jQuery对JS操作DOM事件的封装，包括事件绑定、事件解绑、事件触发。

jQuery事件机制发展历程：

简单事件绑定 >> bind事件绑定 >> delegate事件绑定 >> on【重点】

* 简单事件绑定：

click(handler) 单击事件

blur(handler) 失去焦点事件

mouseenter(handler) 鼠标进入事件

mouseleave(handler) 鼠标离开事件

dbclick(handler) 双击事件

change(handler) 改变事件，如：文本框值改变，下来列表值改变等

focus(handler) 获得焦点事件

keydown(handler) 键盘按下事件

* bind方式 可绑定多个事件，事件元素必须在文档中；1.7后被on取代

$("p").bind("click mouseenter", function(e){

//事件响应方法

});

* delegate方式 支持动态创建的元素绑定事件；性能高（多个子组件只需给 父组件注册）。

$(".parentBox").delegate("p", "click", function(){

//为 .parentBox下面的所有的p标签绑定事件

});

## 2 on方式绑定事件

jQuery**1.7版本后**，jQuery用on统一了所有的事件处理的方法，on方式绑定事件是最现代的方式，兼容zepto。

$('div').on('click mouseover',function(){  
 alert(123);  
});  
  
$('div').on({  
 'click' : function(){  
 alert(123);  
 },  
 'mouseover' : function(){  
 alert(456);  
 }  
});  
  
$('div').on('click mouseover',function(){  
 alert(123);  
 $('div').off('mouseover');  
});

*//注册委托事件：让子元素li执行事件*$('div').on('click','li',function(){  
 alert(123);  
});

如果一个元素被父级绑定了委托事件，自己也绑定了普通事件，优先执行委托事件。

注意：on还有一个可选参数data，在事件函数内，data的访问方式是：e.data

## 3 one绑定一次事件

$( "p" ).one( "click", function() {

alert( $( this ).text() );

});

## 4 jQuery移除事件绑定

* unbind() 方式 解绑 bind方式绑定的事件

$(selector).unbind(); //解绑所有的事件

$(selector).unbind(“click”); //解绑指定的事件

* undelegate() 方式 解绑delegate方式绑定的事件

$( selector ).undelegate(); //解绑所有的delegate事件

$( selector).undelegate( “click” ); //解绑所有的click事件

* **offf()方式** **解绑on方式绑定的事件**

// 解绑匹配元素的所有事件

$(selector).off();

// 解绑匹配元素的所有click事件

$(selector).off(“click”);

// 解绑所有代理的click事件，元素本身的事件不会被解绑

$(selector).off( “click”, “\*\*” );

## 5 事件对象

jQuery的事件对象ev已经是兼容的，不需要像原生一样再写兼容格式。

ev.pageX----相对于文档

ev.clientX----相对于可视区

以上同理还有Y轴对应方法。

ev.which:keycode

event.data 传递给事件处理程序的额外数据

event.currentTarget 等同于this，当前DOM对象

event.pageX  鼠标相对于文档左部边缘的位置

**event.target** 触发事件源，不一定===this

**event.stopPropagation()；** 阻止事件冒泡

**event.preventDefault();**  阻止默认行为

event.type 事件类型：click，dbclick…

event.which 鼠标的按键类型：左1 中2 右3

event.keyCode 键盘按键代码

## 6 事件触发

$(selector).click(); //简单事件触发：触发 click事件

$(selector).trigger(“click”); //trigger方法触发事件

$(selector).triggerHandler(“focus”); //此方式不触发浏览器行为如文本框获焦点

## 7 阻止冒泡与默认行为

event.stopPropagation() //阻止事件冒泡

event.preventDefault();  //阻止默认行为

如果：return false 则直接阻止全部

## 8 节流阀

当onkeydown事件触发时，用户不停的按按按键，会反复触发，为了保证只触发一次，需要添加节流阀：

*//按下1-9这几个数字键，能触发对应的mouseenter事件*$(document).on("keydown", function (e) {  
 if(flag) {  
 flag = false;  
 *//获取到按下的键* var code = e.keyCode;  
 if(code >= 49 && code <= 57){  
 *//触发对应的li的mouseenter事件* $(".nav li").eq(code - 49).mouseenter();  
 }  
 }  
   
});  
  
$(document).on("keyup", function (e) {  
 flag = true;  
   
 *//获取到按下的键* var code = e.keyCode;  
 if(code >= 49 && code <= 57){  
 *//触发对应的li的mouseenter事件* $(".nav li").eq(code - 49).mouseleave();  
 }  
});

# 二 jQuery对象方法补充

## 1 jQuery与原生JS混用

jQuery与原生JS可以同时使用，但是最好不要混用：

<script src="jquery-3.3.1.min.js"></script>  
<script>  
 $(function () {  
 $('#div').click(function () {  
 alert($(this).html());  
 });  
 });  
</script>

这里使用$(this)来避免直接使用this引起的原生JS与jquery的混用。

## 2 filter has not

$(function () {  
 $('div').filter('#div1').css('background','red');  
 $('div').has('span').css('background','green');  
});

filter(): 过滤  
 not()： filter的反义词  
 has()： 是否包含，has查看的是当前元素是否包含，filter过滤的是所有同级元素

## 3 delay() 延时

# 三 $下的常用方法

$下的方法大多数为工具类方法，不仅可以给jQuery使用，也可以给原生JS使用。

## 1 type()

判断类型，与typeof区别是可以判断对象类型，比如时间对象返回Date，而typeof返回的都是Object。

## 2 trim()

去除空白；

## 3 inArray()

类似indexOf();

    var arr = ['a','b','c','d'];

    alert( $.inArray('b',arr) );

## 4 proxy()

改变this指向。

$(function(){

    function show(n1,n2){

        alert(n1);

        alert(n2);

        alert(this);

    }

    //show();

    //$.proxy(show , document,3)(4);

    $(document).click( $.proxy(show,window,3,4) );

});

## 5 parseJSON()

将字符串数据转换成json对象

var str = '{"name":"hello"}';

alert($.parseJSON( str ).name);

## 6 makeArray()

将类数组转换成真正的数组。

    var aDiv = document.getElementsByTagName('div'); //类数组

    $.makeArray(aDiv).push();

## 7 map函数

$.map(arry,function(object,index){}) 返回一个新的数组

$("li").map(function(index, element){}) 注意参数的顺序是反的

var newArr = $.map($("li"), function(i, e) {

return $(e).text() + i;//每一项返回的结果组成新数组

});

var newArr = $("li").map(function(elem, index){

console.log("elem:" + elem);

console.log("index:" + index);

retrun index;

});

# 四 jQuery补充

## 1 链式编程

链式编程原理：return this;

通常情况下，只有设置操作才能把链式编程延续下去。因为获取操作的时候，会返回获取到的相应的值，无法返回 this。

end(); // 结束当前链最近的一次过滤操作，并且返回匹配元素之前的状态。

## 2 隐式迭代

在方法的内部会为匹配到的所有元素进行循环遍历，执行相应的方法；而不用我们再进行循环，简化我们的操作，方便我们调用。

如果获取的是多元素的值，大部分情况下返回的是第一个元素的值。

## 3 each方法

each(): 可以看做for循环的加强版：

    $('li').each(function(i,elem){ //i：下标 elem : 每个元素

        $(elem).html(i);

    });

有了隐式迭代，为什么还要使用each函数遍历？

大部分情况下是不需要使用each方法的，因为jQuery的隐式迭代特性。

如果要对每个元素做不同的处理，这时候就用到了each方法

作用：遍历jQuery对象集合，为每个匹配的元素执行一个函数

// 参数一表示当前元素在所有匹配元素中的索引号

// 参数二表示当前元素（DOM对象）

$(selector).each(function(index,element){});

$( "li" ).each(function() {

$( this ).addClass( "foo" );

});

$( "li" ).each(function( index ) {

console.log( index + ": " + $( this ).text() );

});

$( "div" ).each(function( index, element ) {});

## 4 多库共存

多库共存指的是：jQuery占用了$ 和jQuery这两个变量。当在同一个页面中引用了jQuery这个js库，并且引用的其他库（或者其他版本的jQuery库）中也用到了$或者jQuery这两个变量，那么，要保证每个库都能正常使用，这时候就有了多库共存的问题。

// 模拟另外的库使用了 $ 这个变量，就与jQuery库产生了冲突

var $ = { name : “itecast” };

解决方式：

// 作用：让jQuery释放对$的控制权，让其他库能够使用$符号，此后，只能使用jQuery来调用jQuery提供的方法

$.noConflict();

var myQuery = $.noConflict();

var $ = 10;

myQuery(function(){

    myQuery('body').css('background','red');

});

## 5 jQuery插件机制

，通过插件的方式，可以扩展jQuery的功能。

$.extend: 扩展工具方法下的插件形式

$.fn.extend: 扩展到JQ对象下的插件形式 （fn是 prototype的简写）

<script>

$.extend({

    leftTrim : function(str){

        return str.replace(/^\s+/,'');

    },

    rightTrim : function(){

        //方法体

    }

});

</script>

<script>

var str = ' hello ';

alert( '('+$.leftTrim(str)+')' );

</script>

## 1 缓存

### 1.1 缓存简介

缓存：cache

缓存的作用，就是将一些常用的数据，存储起来，提供使用，提升性

CDN Content Delivery Network

数据库 高并发

非关系型数据库（内存型数据库） MongoDB Redis

网站静态页面缓存机制

将网页静态化，存储在服务器端

### 1.2 斐波那契数列性能问题

//f(n) = f(n-1) + f(n - 2)

var count = 0;

function fib(n){

count ++;

if(n <= 2){

return 1;

}

return fib(n - 1) + fib(n - 2);

}

fib(5);

console.log(count);

我们发现，随着计算数值越大，计算量增长速度远远超过我们的预期。

使用缓存优化：

定义一个缓存数组，存储已经计算出来的斐波那契数

1.计算的步骤

1.先从cache数组中去取想要获取的数字

2.如果获取到了，直接使用

3.如果没有获取到，就去计算，计算完之后，把计算结果存入cache，然后将结果返回

var cache = [];

function fib(n){

//1.从cache中获取数据

if(cache[n] !== undefined){

//如果缓存中有 直接返回

return cache[n];

}

//如果缓存中没有 就计算

if(n <= 2){

//把计算结果存入数组

cache[n] = 1;

return 1;

}

var temp = fib(n - 1) + fib(n - 2);

//把计算结果存入数组

cache[n] = temp;

return temp;

}

console.log(fib(6));

这样创建的缓存暴露了出来，不安全，可以使用闭包方式：

var count =0 ;

function createFib(){

var cache = [];

function fib(n){

count ++;

//1.从cache中获取数据

if(cache[n] !== undefined){

//如果缓存中有 直接返回

return cache[n];

}

//如果缓存中没有 就计算

if(n <= 2){

//把计算结果存入数组

cache[n] = 1;

return 1;

}

var temp = fib(n - 1) + fib(n - 2);

//把计算结果存入数组

cache[n] = temp;

return temp;

}

return fib;

}

var fn = createFib();

fn(6);

console.log(count);

jQuery源码分析：

function createCache() {

var keys = [];

function cache( key, value ) {

// 使用(key + " ") 是为了避免和原生（本地）的原型中的属性冲突

if ( keys.push( key + " " ) > 3 ) {

// 只保留最新存入的数据

delete cache[ keys.shift() ];

}

// 1 给 cache 赋值

// 2 把值返回

return (cache[ key + " " ] = value);

}

return cache;

}

var typeCache = createCache();

typeCache("monitor");

console.log(typeCache["monitor" + " "]);

typeCache("monitor","张学友");

console.log(typeCache["monitor1" + " "]);

typeCache("monitor","刘德华");

console.log(typeCache["monitor2" + " "]);

typeCache("monitor3","彭于晏");

console.log(typeCache["monitor3 "]);

// console.log(typeCache["monitor "]);

### 1.3 jQuery缓存处理

jQuery的查找也是从缓存中进行查找，

//eleCache

//typeCache

//classCache

//eventCache

function createCache(){

//cache对象中以键值对的形式存储我们的缓存数据

var cache = {};

//index数组中该存储键，这个键是有顺序，可以方便我们做超出容量的处理

var index = [];

return function (key, value) {

//如果传了值，就说名是设置值

if(value !== undefined){

//将数据存入cache对象，做缓存

cache[key] = value;

//将键存入index数组中，以和cache中的数据进行对应

index.push(key);

//判断缓存中的数据数量是不是超出了限制

if(index.length >= 50){

//如果超出了限制

//删除掉最早存储缓存的数据

//最早存入缓存的数据的键是在index数组的第一位

//使用数组的shift方法可以获取并删除掉数组的第一个元素

var tempKey = index.shift();

//获取到最早加入缓存的这个数据的键，可以使用它将数据从缓存各种删除

delete cache[tempKey];

}

}

//如果没有传值，只穿了键，那就是获取值

// else{

// return cache[key];

// }

return cache[key];

}

}

var eleCache = createCache();

eleCache("name","高金彪");

console.log(eleCache("name"));

var typeCche = createCache();

### 1.4斐波那契数列最终版

// 创建缓存容器

// function createCache(){

// var cache = {};

// return function (key, value) {

// //如果传了值，就说名是设置值

// if(value !== undefined){

// cache[key] = value;

// return cache[key];

// }

// //如果没有传值，只穿了键，那就是获取值

// else{

// return cache[key];

// }

// }

// }

var count =0 ;

function createFib(){

var fibCache = createCache();

function fib(n){

count ++;

//1.从cache中获取数据

if(fibCache(n) !== undefined){

//如果缓存中有 直接返回

return fibCache(n) ;

}

//如果缓存中没有 就计算

if(n <= 2){

//把计算结果存入数组

fibCache(n , 1) ;

return 1;

}

var temp = fib(n - 1) + fib(n - 2);

//把计算结果存入数组

fibCache(n, temp) ;

return temp;

}

return fib;

}