# 整体架构-jQuery3.2.1

# 创建jQuery对象----(L94)

# jQuery原型添加方法与属性

# 核心函数

# 选择器sizzle引擎

# 回调对象

## 概述

jQuery.Callbacks()是在版本1.7中新加入的，一个多用途的回调列表对象，提供了强大的的方式来管理回调函数列表

### 主要使用场景

控制一系列的函数顺序执行；

最简单的思路是，把函数添加到数组中，当需要时从数组中拿出来进行运行

为了可以增加程序松耦合，jquery使用了基于发布订阅的观察者设计模式，因为使用了观察者模式，故add到list中的函数，当fire时，会将fire的参数全部传入到list中每个函数中，依次获得结果

### 举例

* + - 1. 通常情况下函数队列的处理方式：

// 执行函数  
function runlist(func) {  
 for(var i = 0 ; i <func.length ; i++){  
 func[i]();  
 }  
 func.length = 0;  
}  
// 添加  
var list = [];  
list.push(console1);  
list.push(console2);  
list.push(console3);  
  
runlist(list);  
function console1() {  
 console.log('1')  
}  
function console2() {  
 console.log('2')  
}  
function console3() {  
 console.log('3')  
}

* + - 1. 利用jQuery回调函数

var callback = $.Callbacks();  
callback.add(console1);  
callback.add(console2);  
callback.add(console3);  
callback.fire();

## 源码---（L3274）

### 设置属性分析

* + - 1. Once：确保这个回调列表只执行（.fire() ）一次.

var f1 = function (val) {  
 console.log(val);  
};  
var callback = $.Callbacks('once');  
callback.add(f1);  
callback.fire(1);  
callback.add(f1);  
callback.fire('2');// 结果只会显示1次1

* + - * 1. 如设置once属性，则在fire中会将locked设置为true，因memory为false，故会将list=””，则再add时无法add函数，再调用fire时，fireWith内部也不会调用
      1. Memory:保持以前的值（参数），将函数添加到这个列表的后面，并使用先前保存的参数立即执行该函数。他有一个特点，就是在第一次fire之前使用add添加的回调都不会马上执行，只有调用了一次fire之后使用add添加的回调会马上执行。

var fn1 = function(value) { console.log(value); };  
var callbacks = $.Callbacks("memory");  
callbacks.add( fn1 );//无执行结果  
callbacks.fire( "1" );//执行结果1。保存场景参数1  
console.log('\*\*');  
callbacks.add( fn1 );//执行结果1。使用上次保存的场景参数1  
callbacks.fire( "2" );//执行结果2,2。保存场景参数2

* + - * 1. 第一次调用add，add函数中的memory为undefined，故仅仅是将函数加到list中；当调用fire时，会将此时的参数”1”添加到queue数组中，执行fn1函数
        2. 第二次调用add，会直接fire，利用上一次fire的参数
      1. Unique：确保一次只能添加一个回调(所以在列表中没有重复的回调).通过jQuery.Callbacks里面的has函数，判断list是否已经有func添加
      2. stopOnFalse: 当一个回调返回false 时中断调用

### 源码

/\*  
 \* 一个多用途的回调列表对象，提供了强大的的方式来管理回调函数列表。  
 \* 使用如下参数，构建一个callbasc列表  
 \*  
 \* options: 传入options会改变回调列表的行为方式  
 \*  
 \* 默认的回调列表充当事件回调列表，并且可以fired多次  
 \*  
 \* 可选的options:  
 \*  
 \* once: 确保回调列表只能fired一次（如Deferred）  
 \*  
 \* memory: 保存之前的值，当新函数add到list时，无需fire，自动执行  
 \*  
 \* unique: 确保callback只能被添加一次（list中无重复的）  
 \*  
 \* stopOnFalse: 当回调返回false时，中断调用  
 \*  
 \*/  
jQuery.Callbacks = function( options ) {  
  
 // 如需要将字符串格式的options转为object格式  
 options = typeof options === "string" ?  
 // 将字符串格式的options转为object格式  
 // 类似options=memory，则转换为options.memory = true  
 createOptions( options ) :  
 jQuery.extend( {}, options );  
  
 var // list是否现在firing的标识  
 firing,  
  
 // 对于 non-forgettable lists最后一个fire值  
 // $.Callbacks("memory"),调用过fire后，再add会自动调用fire  
 // add函数中有设置，即memory为true，则调用fire()  
 memory,  
  
 // list是否已经fired  
 fired,  
  
 // 阻止firing的标识  
 locked,  
  
 // 实际的回调列表  
 list = [],  
  
 // 主要是存放memory值,即参数值  
 queue = [],  
  
 // 当前调用的回调index ，如需要会被add或remove修改  
 firingIndex = -1,  
 // fire() 方法作为私有方法被封装在函数中,外部不可直接访问  
 // 因此像 memory、firing、fired 这些状态对于外部上下文来说是不可更改的  
 fire = function() {  
  
 // 如otions.once设置，表示只能fire一次，设置loacked值  
 locked = locked || options.once;  
  
 // 执行list中的全部回调函数  
 fired = firing = true;  
 // 当callback.fire()方法调用时，fireWith会将[context,arugments]，push到queue中，queue.length=1  
 for ( ; queue.length; firingIndex = -1 ) {  
 memory = queue.shift();  
 // 根据fireWith的分析， memory[ 0 ], memory[ 1 ]分别为context与arugments  
 while ( ++firingIndex < list.length ) {  
  
 if ( list[ firingIndex ].apply( memory[ 0 ], memory[ 1 ] ) === false &&  
 options.stopOnFalse ) {  
  
 // stopOnFalse为true，则跳到list最后，  
 // 再调用add，也不会再次fire  
 firingIndex = list.length;  
 memory = false;  
 }  
 }  
 }  
  
 // 如未传入memory参数，则将memory设置为false  
 if ( !options.memory ) {  
 memory = false;  
 }  
  
 firing = false;  
  
 // Clean up if we're done firing for good  
 if ( locked ) {  
  
 // 保留一个空list，为了之后再添加回调时使用  
 if ( memory ) {  
 list = [];  
  
 } else {  
 list = "";  
 }  
 }  
 },  
  
 // 实际的回调对象  
 self = {  
  
 // 添加回调或回调集合到list中  
 add: function() {  
 if ( list ) {  
  
 // 如options为memory参数，应在add后直接firing  
 if ( memory && !firing ) {  
 firingIndex = list.length - 1;  
 queue.push( memory );  
 }  
 // 将add参数arguments传入这个立即运行函数  
 // 如args为([bar,[foo,foo1]])，each循环2次，第一次为bar  
 // 第二次为[foo,foo1]，通过递归，将这个数组再次调用add()  
 // 最后将bar,foo,foo1添加到list中  
 // arguments比较特殊，如参数为[1,2,3],但arguments.length=1,故each不会循环多次  
 ( function add( args ) {  
 jQuery.each( args, function( \_, arg ) {  
 if ( jQuery.isFunction( arg ) ) {  
 // options.unique只能添加一个回调（不能有重复的）  
 if ( !options.unique || !self.has( arg ) ) {  
 list.push( arg );  
 }  
 } else if ( arg && arg.length && jQuery.type( arg ) !== "string" ) {  
 // 如args是嵌套数组，则需要用递归将参数全部加入到list中  
 // 检查递归  
 add( arg );  
 }  
 } );  
 } )( arguments );  
  
 if ( memory && !firing ) {  
 fire();  
 }  
 }  
 return this;  
 },  
  
 // 删除list中的一个回调  
 remove: function() {  
 jQuery.each( arguments, function( \_, arg ) {  
 var index;  
 while ( ( index = jQuery.inArray( arg, list, index ) ) > -1 ) {  
 list.splice( index, 1 );  
  
 // Handle firing indexes  
 if ( index <= firingIndex ) {  
 firingIndex--;  
 }  
 }  
 } );  
 return this;  
 },  
  
 // 检查一个回调是否在list中  
 has: function( fn ) {  
 return fn ?  
 jQuery.inArray( fn, list ) > -1 :  
 list.length > 0;  
 },  
  
 // 清空list中的所有回调  
 // 直接将list=[]则是清空  
 empty: function() {  
 if ( list ) {  
 list = [];  
 }  
 return this;  
 },  
  
 // 禁用 .fire and .add  
 // 与lock主要区别是，不会判断！mermory与！firing  
 disable: function() {  
 locked = queue = [];  
 list = memory = "";  
 return this;  
 },  
 disabled: function() {  
 return !list;  
 },  
  
 // 禁用 .fire  
 // 禁用fire同时会禁用add，除非设置了options=memory  
 // 锁定一个回调列表，以避免进一步的修改列表状态  
 // 即，lock之后，fire不会再有函数运行结果  
 lock: function() {  
 locked = queue = [];  
 if ( !memory && !firing ) {  
 list = memory = "";  
 }  
 return this;  
 },  
 locked: function() {  
 return !!locked;  
 },  
  
 // 访问给定的上下文和参数列表中的所有回调  
 fireWith: function( context, args ) {  
 // callback调用lock()方法后，locked=[]  
 if ( !locked ) {  
 args = args || [];  
 args = [ context, args.slice ? args.slice() : args ];  
 queue.push( args );  
 if ( !firing ) {  
 fire();  
 }  
 }  
 return this;  
 },  
  
 // 用给定的arguments调用list中的所有回调函数  
 fire: function() {  
 self.fireWith( this, arguments );  
 return this;  
 },  
  
 // 判断list是否调用过fire  
 fired: function() {  
 return !!fired;  
 }  
 };  
  
 return self;  
};