# 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

# Deferred对象(异步队列)

# 数据缓存

## 概述

### 内存泄露

* + - 1. 循环引用很常见且大部分情况下是无害的，但当参与循环引用的对象中有DOM对象或者ActiveX对象时，循环引用将导致内存泄露。如下为普通的循环引用，引用自身，大部分情况无害；但将new Object替换成document.getElementById或者document.createElement就会发生内存泄露了。

var a=new Object;  
a.r=a;

### 数据缓存

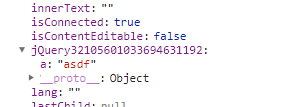
* + - 1. 一般jQuery开发，我们都喜欢便捷式的把很多属性，比如状态标志都写到dom节点中,也就是HTMLElement，这样做好处是直观，坏处则是：
         1. 循环引用
         2. 直接暴露数据，安全性？
         3. 增加一堆的自定义属性标签，对浏览器来说是没意义的
         4. 取数据的时候要对HTML节点做操作
      2. **更好的解决方法是使用一种低耦合的方式让DOM和缓存数据能够联系起来。**
      3. jQuery引入缓存的作用
         1. 允许我们在DOM元素上附加任意类型的数据,避免了循环引用的内存泄漏风险
         2. 用于存储跟dom节点相关的数据，包括事件，动画等
         3. 一种低耦合的方式让DOM和缓存数据能够联系起来

### jQuery实现缓存思路

* + - 1. 在对象上绑定expando属性，并jQuery定义了一个属性cache = {}来保存所有的缓存数据
      2. 通过expando来查找jQuery.cache上对应的缓存数据
      3. 为了保证了id 的全局唯一性，这个id使用jQuery.guid自增

## jQuery.data()与.data()区别

* + - 1. 看jQuery.data(element,[key],[value])，每一个element都会有自己的一个{key:value}对象保存着数据，所以新建的对象就算有key相同它也不会覆盖原来存在的对象key所对应的value，因为新对象保存是是在另一个{key:value}对象中；并且每次调用$.data()，内部data函数，会创建this.expando是不同的
      2. $("div").data("a","aaaa") 它是把数据绑定每一个匹配div节点的元素上，因此相同的key会相互覆盖
      3. 数据存储的位置：



## 源码分析

/\*\*  
 \* jQuery很多函数具有写入和读取功能，如jQuery.fn.css等，  
 \* 此函数为抽象出的入口函数；有时jQuery可以传入obj，或者传入function等，很多都是通过这个函数进行处理的  
 \* 以jquery.fn.extend({data:..})返回access为例  
 \* access首先重要逻辑是处理key为obj形式，如key为obj，则将key再次调用access  
 \* 之后，将处理逻辑分为有key还是无key；如有key，则bulk为false  
 \* 注：用此方法包装的函数调用，提供了四种传值方式：name，(name,value)，({key:value,key:value})，(name,function(index,attr){})

\* @param elems jquery匹配的dom元素  
 \* @param fn fn函数  
 \* @param key null  
 \* @param value value  
 \* @param chainable arguments.length > 1  
 \* @param emptyGet null  
 \* @param raw true  
 \* @return {\*}  
 \*/  
  
var access = function( elems, fn, key, value, chainable, emptyGet, raw ) {  
 var i = 0,  
 len = elems.length,  
 // 通过bulk将处理逻辑分为2大类，key有值，bulk为false，否则为true  
 // 这样封装，主要是为了两类能使用相同的逻辑  
 bulk = key == null; // == 优先级高于=，如key==null，则将bulk赋值为true  
  
 // 如key为object，则递归方法，一个个key设置  
 // 类似，$('a').css({'height':'10px','width':'10px'})  
 // 具体为何不用仔细理解，jQuery.fn.css代码分析会有，jQuery.fn.css内部调用了access,  
 if ( jQuery.type( key ) === "object" ) {  
 chainable = true;  
 for ( i in key ) {  
 access( elems, fn, i, key[ i ], true, emptyGet, raw );  
 }  
  
 // 如value有值，即类似$('a').css({'height':'10px'})的调用方式，key为height，value为10px  
 } else if ( value !== undefined ) {  
 chainable = true;  
 // 如value不是函数，raw设为true，raw标识为value是否为函数  
 if ( !jQuery.isFunction( value ) ) {  
 raw = true;  
 }  
 // key==null，则将bulk赋值为true  
 if ( bulk ) {  
  
 // value不是函数  
 // Bulk 操作针对整个集合operations run against the entire set  
 if ( raw ) {  
 fn.call( elems, value );  
 fn = null;  
  
 // value为函数，elem为普通值，用jquery封装后重新定义fn  
 } else {  
 bulk = fn;  
 fn = function( elem, key, value ) {  
 return bulk.call( jQuery( elem ), value );  
 };  
 }  
 }  
  
 if ( fn ) {  
 for ( ; i < len; i++ ) {  
 fn(  
 elems[ i ], key, raw ?  
 value :  
 value.call( elems[ i ], i, fn( elems[ i ], key ) )  
 );  
 }  
 }  
 }  
 // 如value !== undefined，即有value值都是链式调用；  
 // 上面会将chainable设为true， 则返回elems  
 if ( chainable ) {  
 return elems;  
 }  
  
 // key有值，bulk为false  
 if ( bulk ) {  
 return fn.call( elems ); //以jquery.fn.extend({data:..})返回access为例,则fun的参数value为elems  
 }  
 // 如elem.length不存在，则返回emptyGet  
 return len ? fn( elems[ 0 ], key ) : emptyGet;  
};

/\*\*  
 \* 判断绑定数据的目标owner类型是否符合  
 \* @param owner  
 \* @return {boolean}  
 \*/  
var acceptData = function( owner ) {  
  
 // Accepts only:  
 // - Node  
 // - Node.ELEMENT\_NODE  
 // - Node.DOCUMENT\_NODE  
 // - Object  
 // - Any +owner.nodeType转换为number，但+{}.nodeType === NaN !NaN = true  
 return owner.nodeType === 1 || owner.nodeType === 9 || !( +owner.nodeType );  
};  
  
  
  
  
function Data() {  
 this.expando = jQuery.expando + Data.uid++;  
}  
  
Data.uid = 1;  
/\*\*  
 \* 数据缓存，先在jQuery内部创建一个cache对象{}, 来保存缓存数据。   
 \* 然后往需要进行缓存的DOM节点上扩展一个值为expando的属性  
 \* @type {{  
 \* cache: Data.cache, 创建cache缓存  
 \* set: Data.set, 在owner设置data  
 \* get: Data.get, 获取数据  
 \* access: Data.access, 提供set，get统一访问接口，根据不同情况，调用get，set方法  
 \* remove: Data.remove, 删除值，删除key对应的值  
 \* hasData: Data.hasData 判断owner是否包含数据  
 \* }}  
 \*/  
Data.prototype = {  
 // 创建cache缓存  
 cache: function( owner ) {  
  
 // 检查DOM对象是否有expando属性，判断对象是否有cache  
 var value = owner[ this.expando ];  
  
 // 如无cache，创建一个  
 if ( !value ) {  
 value = {};  
  
 // 现代浏览器可以接收非元素节点绑定数据，但由于see #8335.问题，返回空对象  
 if ( acceptData( owner ) ) {  
  
 // 如owner是元素节点，就在expando上绑定value值  
 if ( owner.nodeType ) {  
 owner[ this.expando ] = value;  
  
 // 将owner的this.expando配置为非枚举属性，为了保证当数据删除时需要将属性同时删除，  
 } else {  
 Object.defineProperty( owner, this.expando, {  
 value: value,  
 configurable: true  
 } );  
 }  
 }  
 }  
  
 return value;  
 },  
 // 在owner设置data，参数可以为[owner, data, value],或者[owner,{properties}]  
 set: function( owner, data, value ) {  
 var prop,  
 cache = this.cache( owner );  
  
 // Handle: [ owner, key, value ] args  
 // 总是使用驼峰标记法的key (gh-2257)  
 if ( typeof data === "string" ) {  
 cache[ jQuery.camelCase( data ) ] = value;  
  
 // Handle: [ owner, { properties } ] args  
 } else {  
  
 // 复制属性到cache对象  
 for ( prop in data ) {  
 cache[ jQuery.camelCase( prop ) ] = data[ prop ];  
 }  
 }  
 return cache;  
 },  
 // 获取数据，如key==undefined，直接返回全部数据，  
 get: function( owner, key ) {  
 return key === undefined ?  
 this.cache( owner ) :  
  
 // 总是使用驼峰标记法 (gh-2257)  
 owner[ this.expando ] && owner[ this.expando ][ jQuery.camelCase( key ) ];  
 },  
 // 提供set，get统一访问接口，根据不同情况，调用get，set方法  
 access: function( owner, key, value ) {  
  
 // 处理如下情况：  
 // 1、未指定key  
 // 2、字符串类型key指定，但未提供value值  
 //  
 // 利用get方法决定何值返回  
 if ( key === undefined ||  
 ( ( key && typeof key === "string" ) && value === undefined ) ) {  
  
 return this.get( owner, key );  
 }  
  
 // 如key不是String，或key与value都指定了，利用set方法，设置值  
 this.set( owner, key, value );  
 // 根据不同情况返回不同值  
 return value !== undefined ? value : key;  
 },  
 // 删除值，删除key对应的值，当key=undefined时，删除owner[ this.expando ]  
 remove: function( owner, key ) {  
 var i,  
 cache = owner[ this.expando ];  
  
 if ( cache === undefined ) {  
 return;  
 }  
  
 if ( key !== undefined ) {  
  
 // 支持keys数组，与空格分隔字符串  
 if ( Array.isArray( key ) ) {  
  
 // 如key是数组，将key全部转换为驼峰标记法的  
 key = key.map( jQuery.camelCase );  
 } else {  
 key = jQuery.camelCase( key );  
  
 // 如key具有空格，则直接使用它  
 // 否则构造一个无空格的array  
 key = key in cache ?  
 [ key ] :  
 ( key.match( rnothtmlwhite ) || [] );  
 }  
  
 i = key.length;  
  
 while ( i-- ) {  
 delete cache[ key[ i ] ];  
 }  
 }  
  
 // 如未传入key或cache无数据，则删除expando  
 //??????????????????????????????????当key为undefined，只是删除了dom的expando属性，未在cache中删除对应数据？？？？？？？  
 if ( key === undefined || jQuery.isEmptyObject( cache ) ) {  
  
 // Support: Chrome <=35 - 45  
 // Webkit & Blink 当DOM节点删除属性时会造成性能下降，故将属性设置为undefined  
 // https://bugs.chromium.org/p/chromium/issues/detail?id=378607 (bug restricted)  
 if ( owner.nodeType ) {  
 owner[ this.expando ] = undefined;  
 } else {  
 delete owner[ this.expando ];  
 }  
 }  
 },  
 // 判断owner是否包含数据  
 hasData: function( owner ) {  
 var cache = owner[ this.expando ];  
 return cache !== undefined && !jQuery.isEmptyObject( cache );  
 }  
};  
// 相当于私有数据  
var dataPriv = new Data();  
// 用户数据  
var dataUser = new Data();  
  
  
  
// 实现概要  
// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*学习下如何不暴露实现细节给用户  
// 1. 增加api接口，语义上兼容1.9x分支  
// 2. 通过减少存储路径到单个，以提高模块可维护性  
// 3. 使用相同的机制支持private与user数据  
// 4. 不会暴露私有数据给用户代码 (**TODO: Drop \_data, \_removeData)**// 5. 避免暴露实现细节给用户对象 (eg. expando properties)  
// 6. 提供清晰方法来实现2014年的WeakMap升级 https://www.debuggex.com/  
  
// \w :匹配包括下划线的任何单词字符,等价于 [A-Z a-z 0-9\_],  
// \W :匹配任何非单词字符,等价于 [^A-Z a-z 0-9\_]  
var rbrace = /^(?:\{[\w\W]\*\}|\[[\w\W]\*\])$/,  
 rmultiDash = /[A-Z]/g;  
// 对数据进行转换，dataAttr中使用，将html中data-\*的数据进行下处理  
function getData( data ) {  
 if ( data === "true" ) {  
 return true;  
 }  
  
 if ( data === "false" ) {  
 return false;  
 }  
  
 if ( data === "null" ) {  
 return null;  
 }  
  
 // 只有在不改变字符串的情况下才转换成数字。  
 if ( data === +data + "" ) {  
 return +data;  
 }  
 // 在此函数外进行了捕获  
 // /^(?:\{[\w\W]\*\}|\[[\w\W]\*\])$/.test('{asd}');=====>true但JSON.parse会报错？  
 if ( rbrace.test( data ) ) {  
 return JSON.parse( data );  
 }  
  
 return data;  
}  
  
// 从HTML的data-\*获取数据，将数据设置到dataUser里面  
function dataAttr( elem, key, data ) {  
 var name;  
  
 // 如内部找不到任何数据，则从html5中的data-\*查找数据  
 if ( data === undefined && elem.nodeType === 1 ) {  
 // key.replace( rmultiDash, "-$&" )类似于key.replace( /([A-Z])/g, "-$1" );不知这样写有何意义？？？？？？？？？？？？？  
 name = "data-" + key.replace( rmultiDash, "-$&" ).toLowerCase();  
 data = elem.getAttribute( name );  
  
 if ( typeof data === "string" ) {  
 try {  
 data = getData( data );  
 } catch ( e ) {} // 对于getData报错处理  
  
 // 确保我们设置了数据，这样它以后不会改变。  
 dataUser.set( elem, key, data );  
 } else {  
 data = undefined;  
 }  
 }  
 return data;  
}  
// $.data()方法，每次调用都会针对无this.expando创建cache存放缓存数据  
jQuery.extend( {  
 hasData: function( elem ) {  
 return dataUser.hasData( elem ) || dataPriv.hasData( elem );  
 },  
 //data方法可以同时set或get数据  
 data: function( elem, name, data ) {  
 return dataUser.access( elem, name, data );  
 },  
  
 removeData: function( elem, name ) {  
 dataUser.remove( elem, name );  
 },  
  
 // **TODO: Now that all calls to \_data and \_removeData have been replaced** // with direct calls to dataPriv methods, these can be deprecated.  
 \_data: function( elem, name, data ) {  
 return dataPriv.access( elem, name, data );  
 },  
  
 \_removeData: function( elem, name ) {  
 dataPriv.remove( elem, name );  
 }  
} );  
// $('aa').data()方法，与$.data()方法是不一样的  
jQuery.fn.extend( {  
 data: function( key, value ) {  
 var i, name, data,  
 elem = this[ 0 ],// $匹配到的第一个元素  
 attrs = elem && elem.attributes;// 返回指定节点的属性集合  
  
 // 获取所有值,即$().data()调用形式  
 if ( key === undefined ) {  
 if ( this.length ) {// jquery有匹配到元素  
 data = dataUser.get( elem );  
 // 主要是将html属性上的data-数据存储在data中  
 if ( elem.nodeType === 1 && !dataPriv.get( elem, "hasDataAttrs" ) ) {  
 i = attrs.length;  
 while ( i-- ) {  
  
 // Support: IE 11 only  
 // attrs 元素可以为null (#14894)  
 if ( attrs[ i ] ) { // 获取html上data  
 name = attrs[ i ].name;  
 if ( name.indexOf( "data-" ) === 0 ) {  
 name = jQuery.camelCase( name.slice( 5 ) );  
 dataAttr( elem, name, data[ name ] );  
 }  
 }  
 }  
 dataPriv.set( elem, "hasDataAttrs", true );//只需要存储data-上数据一个次  
 }  
 }  
  
 return data;  
 }  
  
 // 设置多组值  
 if ( typeof key === "object" ) {  
 return this.each( function() {  
 dataUser.set( this, key );  
 } );  
 }  
  
 return access( this, function( value ) {  
 var data;  
  
 // 如$()获取到dom，不是空，则this[0]不为空，value这个参数也不会是undefined  
 // 对于空的jquery对象，如$('#a')[0]会返回undefined，即elem = this[0]会在试图读取data缓存时抛出异常  
 if ( elem && value === undefined ) {  
  
 // 尝试从cahce中获取数据  
 // key在Data中都是驼峰标记法  
 data = dataUser.get( elem, key );  
 if ( data !== undefined ) {  
 return data;  
 }  
  
 // 尝试在html的data中获取数据  
 // HTML5 custom data-\* attrs  
 data = dataAttr( elem, key );  
 if ( data !== undefined ) {  
 return data;  
 }  
  
 // 确实无数据，直接return  
 return;  
 }  
  
 // 设置数据  
 this.each( function() {  
  
 // 总是存储驼峰标记法的key  
 dataUser.set( this, key, value );  
 } );  
 }, null, value, arguments.length > 1, null, true );  
 },  
 // 由于each,则每个节点都会删除key对应的值  
 removeData: function( key ) {  
 return this.each( function() {  
 dataUser.remove( this, key );  
 } );  
 }  
} );