# 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

# 队列操作

# 事件系统

# DOM操作

# 样式操作

# class操作

## function getClass( elem )----------(L7761)

// 获取元素elem的class属性值  
// 如<p id="a" class="a b c">asdf</p>，document.getElementById('a').getAttribute('class')  
// 取到值为'a b c';  
function getClass( elem ) {  
 return elem.getAttribute && elem.getAttribute( "class" ) || "";  
}

## function stripAndCollapse( value )------------(L7753)

// 根据HTML spec协议处理空白，将value数组再拼接为'a b c'形式  
// https://html.spec.whatwg.org/multipage/infrastructure.html#strip-and-collapse-whitespace  
function stripAndCollapse( value ) {  
 var tokens = value.match( rnothtmlwhite ) || [];  
 return tokens.join( " " );  
}

## addClass等源码分析-------------------------(L7765)

jQuery.fn.extend( {  
 /\*\*  
 \* 为每个匹配的元素添加指定的类名。  
 \* 注意：方法不会替换一个class，只会在现有元素class后面添加新的  
 \*  
 \* @param value 一个或多个要添加到元素中的CSS类名，请用空格分开  
 \* value为function，如：可以为每个li的样式后面增加个序号  
 \* $( "ul li" ).addClass(function( index,curClass ) {// curClass，当前匹配元素的class值  
 \* return "item-" + index;  
 \* });  
 \* @return {\*}  
 \*/  
 addClass: function( value ) {  
 var classes, elem, cur, curValue, clazz, j, finalValue,  
 i = 0;  
 // 如value为函数，递归调用addClass  
 if ( jQuery.isFunction( value ) ) {  
 // 因此func(index,curClass);第一个参数为jQuery匹配数组的各元素，curClass，当前匹配元素的class值  
 return this.each( function( j ) {  
 jQuery( this ).addClass( value.call( this, j, getClass( this ) ) );  
 } );  
 }  
 // value为String类型，并且不是""  
 if ( typeof value === "string" && value ) {  
 classes = value.match( rnothtmlwhite ) || [];//将以空格分隔的class转换为数组  
  
 while ( ( elem = this[ i++ ] ) ) {  
 curValue = getClass( elem );// 获取当前jQuery匹配元素的Class值  
 // 为了更好的可压缩性？？？？？  
 cur = elem.nodeType === 1 && ( " " + stripAndCollapse( curValue ) + " " );  
  
 if ( cur ) { // 如当前节点存在class  
 j = 0;  
 // 循环遍历要添加的class，利用indexOf确保不添加重复的样式  
 while ( ( clazz = classes[ j++ ] ) ) {  
 if ( cur.indexOf( " " + clazz + " " ) < 0 ) {  
 cur += clazz + " ";  
 }  
 }  
  
 // 只有有新增样式时，才重新设定元素的class属性，避免浏览器不必要的渲染  
 finalValue = stripAndCollapse( cur );  
 if ( curValue !== finalValue ) {  
 elem.setAttribute( "class", finalValue );  
 }  
 }  
 }  
 }  
  
 return this;  
 },  
 /\*\*  
 \* 删除class  
 \* @param value value未指定值，删除全部class  
 \* @return {\*}  
 \*/  
 removeClass: function( value ) {  
 var classes, elem, cur, curValue, clazz, j, finalValue,  
 i = 0;  
 // 如value为函数，递归调用removeClass  
 if ( jQuery.isFunction( value ) ) {  
 return this.each( function( j ) {  
 jQuery( this ).removeClass( value.call( this, j, getClass( this ) ) );  
 } );  
 }  
 // 如未传入参数  
 if ( !arguments.length ) {  
 // 设置元素的class属性值为""  
 return this.attr( "class", "" );  
 }  
  
 if ( typeof value === "string" && value ) {  
 classes = value.match( rnothtmlwhite ) || [];  
  
 while ( ( elem = this[ i++ ] ) ) {  
 curValue = getClass( elem );  
  
 cur = elem.nodeType === 1 && ( " " + stripAndCollapse( curValue ) + " " );  
  
 if ( cur ) {  
 j = 0;  
 while ( ( clazz = classes[ j++ ] ) ) {  
  
 // 删除class，如有相同的class值，用repalce替换为""  
 while ( cur.indexOf( " " + clazz + " " ) > -1 ) {  
 cur = cur.replace( " " + clazz + " ", " " );  
 }  
 }  
  
 // 避免重新渲染  
 finalValue = stripAndCollapse( cur );  
 if ( curValue !== finalValue ) {  
 elem.setAttribute( "class", finalValue );  
 }  
 }  
 }  
 }  
  
 return this;  
 },  
 /\*\*  
 \* 切换样式  
 \* .toggleClass( className )：如有元素有className，则删除此className，无则添加上  
 \* .toggleClass( className, state )：state为true等价于addClass(className);false等价于removeClass(className)  
 \* .toggleClass( func )：如div.foo父级有bar类，则切换happy，否则切换sad  
 \* $( "div.foo" ).toggleClass(function() {  
 \* if ( $( this ).parent().is( ".bar" ) ) {  
 \* return "happy";  
 \* } else {  
 \* return "sad";  
 \* }  
 \* });  
 \* @param value  
 \* @param stateVal  
 \* @return {\*}  
 \*/  
 toggleClass: function( value, stateVal ) {  
 var type = typeof value;  
 // 如stateVal是true或false，则调用addClass或removeClass  
 if ( typeof stateVal === "boolean" && type === "string" ) {  
 return stateVal ? this.addClass( value ) : this.removeClass( value );  
 }  
 // 如value为函数  
 if ( jQuery.isFunction( value ) ) {  
 return this.each( function( i ) {  
 jQuery( this ).toggleClass(  
 value.call( this, i, getClass( this ), stateVal ),  
 stateVal  
 );  
 } );  
 }  
  
 return this.each( function() {  
 var className, i, self, classNames;  
  
 if ( type === "string" ) {  
  
 // 切换个别类名  
 i = 0;  
 self = jQuery( this );  
 classNames = value.match( rnothtmlwhite ) || [];  
  
 while ( ( className = classNames[ i++ ] ) ) {  
  
 // 利用hasClass判断，如果有，则删除，没有添加上  
 if ( self.hasClass( className ) ) {  
 self.removeClass( className );  
 } else {  
 self.addClass( className );  
 }  
 }  
  
 // 切换全部类名,.toggleClass( boolean )在3.0版本deprecated了（官网API）  
 } else if ( value === undefined || type === "boolean" ) {  
 className = getClass( this );  
 if ( className ) {  
  
 // 内部存储className  
 dataPriv.set( this, "\_\_className\_\_", className );  
 }  
  
 // 如$('#a').toggleClass();或$('#a').toggleClass(false);调用方式；会删除a上的类名  
 // 但将删除的类名存储在dataPriv上  
 // 再通过$('#a').toggleClass(true);调用方式会取回dataPriv上之前删除的类名添加到a上  
 if ( this.setAttribute ) {  
 this.setAttribute( "class",  
 className || value === false ?  
 "" :  
 dataPriv.get( this, "\_\_className\_\_" ) || ""  
 );  
 }  
 }  
 } );  
 },  
 /\*\*  
 \* 判断是否有slector给定的样式在匹配元素上  
 \* 注意：因为hasClass并为对多个class进行空格拆分，故，如<p id="a" class="a b c">asdf</p>  
 \* $('#a').hasClass('b a');返回false  
 \* @param selector  
 \* @return {boolean}  
 \*/  
 hasClass: function( selector ) {  
 var className, elem,  
 i = 0;  
  
 className = " " + selector + " ";  
 while ( ( elem = this[ i++ ] ) ) {  
 if ( elem.nodeType === 1 &&  
 ( " " + stripAndCollapse( getClass( elem ) ) + " " ).indexOf( className ) > -1 ) {  
 return true;  
 }  
 }  
  
 return false;  
 }  
} );