# 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

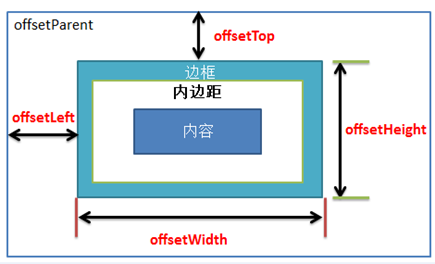
# 样式操作

# class操作

# 位置尺寸操作

## 基础概述

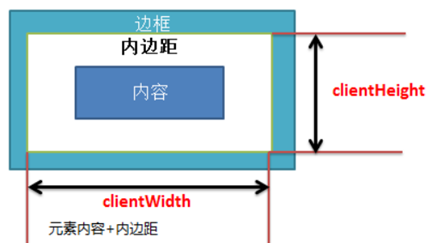
### 偏移量(offsetWidth,offsetHeight,offsetLeft,offsetTop)

[](https://images0.cnblogs.com/blog/329084/201402/242021021483205.png)

* + - 1. offsetHeight/offsetWidth: 表述元素的外尺寸：元素内容+内边距+边框(不包括外边距)
      2. offsetLeft/offsetTop: 表示该元素的左上角（边框外边缘）与**已定位的父容器**（offsetParent对象）左上角的距离。
      3. offsetParent元素是指元素最近的定位（relative,absolute）祖先元素,可递归上溯。

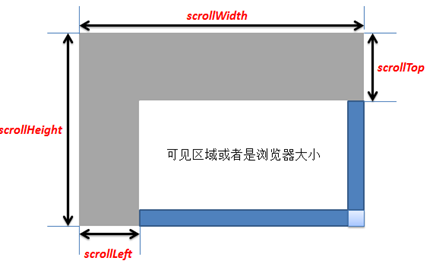
### 客户区大小 (clientWidth,clientHeight)

* + - 1. clientWidth/clientHeight: 用于描述元素的内尺寸：元素内容+两边内边距

[](https://images0.cnblogs.com/blog/329084/201402/242021048359691.png)

* + - 1. **document.documentElement.clientWidth不包括滚动条**
      2. document.body.clientHeight：<body>元素的宽度和高度（注意，包括了不可见的区域）

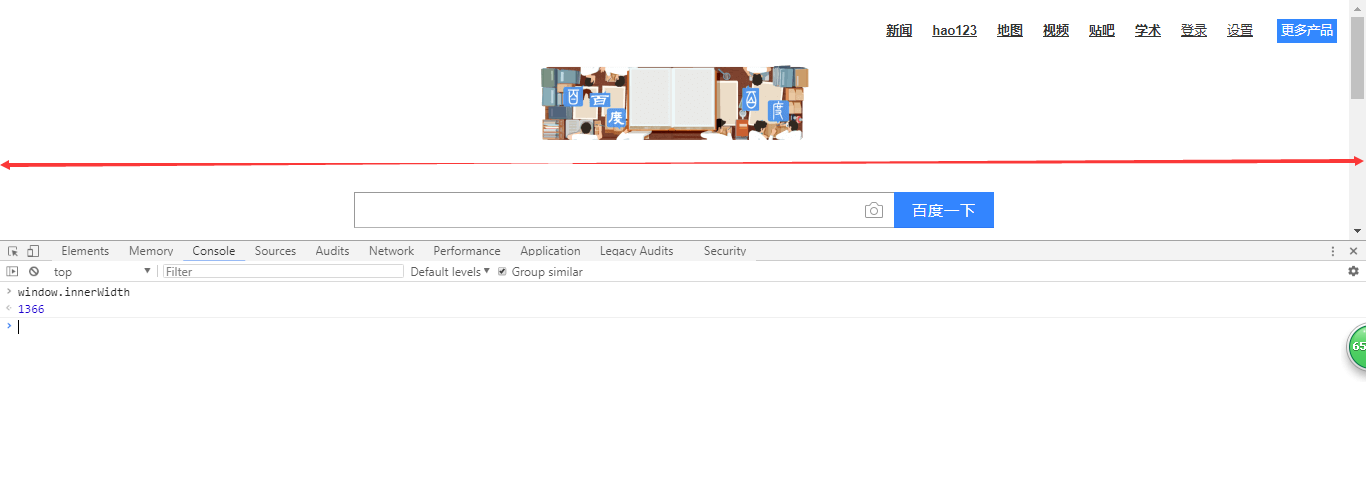
### 滚动大小(scrollWidth,scrollHeight,scrollLeft,scrollTop)

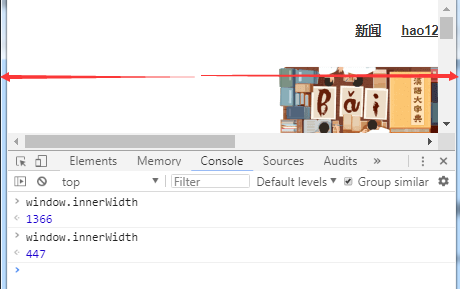
[](https://images0.cnblogs.com/blog/329084/201402/242021067376237.png)

* + - 1. scrollHeight/scrollWidth: 元素内容的总高度或宽度
      2. scrollLeft/scrollTop：是指元素滚动条位置，它们是可写的(被隐藏的内容区域左侧/上方的像素)
      3. 浏览器窗口的滚动条位置：window对象的pageXoffset和pageYoffset, IE 8及更早版本可以通过scrollLeft和scrollTop属性获得滚动条位置

### Window.innerHeight,window.innerWidth

* + - 1. 浏览器窗口的视口（viewport）高度（以像素为单位），**如果存在水平滚动条，则包括它。**



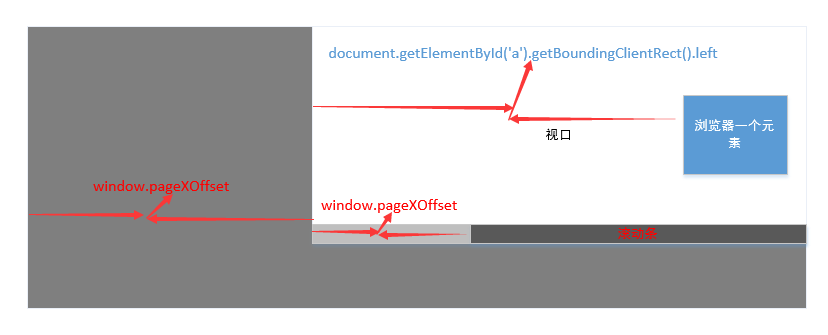


### window.pageXOffset, window.pageYOffset

* + - 1. 当前页面相对于窗口显示区左上角的 X /Y位置，**即水平/垂直滚动条已滚动的距离**

### Window.pageXOffset与getBoundingClientRect()关系

* + - 1. Window.pageXOffset是滚动条位置，图中灰色为document的大小，白色为视口大小，实际上有滚动条实际是将视图相对于document右移动pageXOffset
      2. getBoundingClientRect().left实际是元素具体视图左边的距离
      3. 因此document.getElementById('a').getBoundingClientRect().left+window.pageXOffset即为，元素相对于document左边的距离

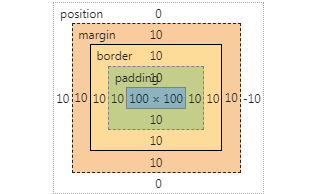


### getBoundingClientRect().left获取的值

* + - 1. 当前html结构

<div id="div2" >  
 <p id="a" class="a b c" >asdf</p>  
</div>

* + - 1. Div2的left：100px，p具有如下，具有padding，border，margin都是10px

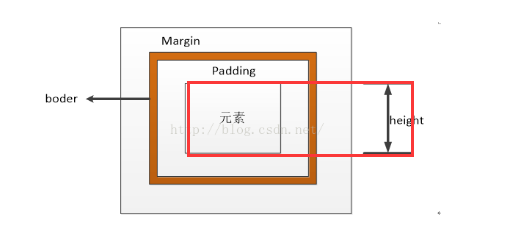


* + - 1. getBoundingRect()=120px，即计算的是div2偏移的100px与margin+border；这有利于理解下面offset，position等计算

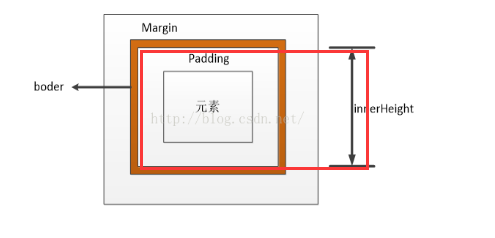
## 尺寸操作(innerHeight, innerWidth, height, width, outerHeight and outerWidth)

### 方法介绍

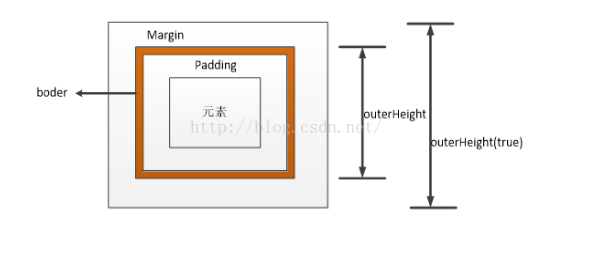
* + - 1. 注意
         1. 这些尺寸方法都是jQuery的方法
         2. 因为有些样式不是简单的读写属性就可以的，比如width就不是简单地读取el.style.width，所以才弄了这些方法
      2. Height,width
         1. 元素本身宽高，不包括padding和margin



* + - 1. InnerHeight,innerWidth
         1. 包括内容+padding，**无border**



* + - 1. OuterHeight,outerWidth
         1. 如果参数为true时，宽高为：元素自身宽高 + padding + border +margin
         2. 无参数时，元素自身宽高 + padding + border



### 源码分析--------------------(L10120)

/\*\*  
 \* 构建jQuery的innerHeight, innerWidth, height, width, outerHeight and outerWidth 方法  
 \*/  
jQuery.each( { Height: "height", Width: "width" }, function( name, type ) {  
 // {padding:"innerHeight",content:height,"":outerHeight}  
 // {padding:"innerWidth",content:width,"":outerWidth}  
 jQuery.each( { padding: "inner" + name, content: type, "": "outer" + name },  
 // 因此defalutExtra为padding，content，"",funcName为innerHeight，height，outerHeight  
 function( defaultExtra, funcName ) {  
  
 // 外边距只添加到outerHeight, outerWidth  
 jQuery.fn[ funcName ] = function( margin, value ) {  
 var chainable = arguments.length && ( defaultExtra || typeof margin !== "boolean" ),  
 extra = defaultExtra || ( margin === true || value === true ? "margin" : "border" );  
  
 return access( this, function( elem, type, value ) {  
 var doc;  
 // 如是elem为window对象，  
 if ( jQuery.isWindow( elem ) ) {  
  
 // $( window ).outerWidth/Height 会返回包含scrollbars的宽高值 (gh-1729)  
 return funcName.indexOf( "outer" ) === 0 ?  
 elem[ "inner" + name ] : // 如调用的是outerHeight，则用window.innerHeight代替，包括滚动条  
 // Document对象是每个DOM树的根，但是它并不代表树中的一个HTML元素  
 // document.documentElement属性引用了作为文档根元素的html标记  
 // document.body属性引用了body标记  
 elem.document.documentElement[ "client" + name ];// 否则用documentElement.clientHeight,不包括滚动条  
 }  
  
 // 获取document的宽高，nodeType=9为document  
 if ( elem.nodeType === 9 ) {  
 doc = elem.documentElement;// 返回文档根  
  
 // document.documentElement.scrollWidth返回整个文档的宽度  
 // document.documentElement.offsetWidth返回整个文档的可见宽度  
 // document.documentElement.clientwidth返回整个文档的可见宽度（不包含边框），clientwidth = offsetWidth - borderWidth  
 // 一般不会给document.documentElement设置边框，故clientwidth 与 offsetWidth一致  
 return Math.max(  
 elem.body[ "scroll" + name ], doc[ "scroll" + name ],  
 elem.body[ "offset" + name ], doc[ "offset" + name ],  
 doc[ "client" + name ]  
 );  
 }  
  
 return value === undefined ?// 根据是否提供value值，判断是取值还是设置值  
  
 // 获取元素宽高，但不强制转换为数值  
 jQuery.css( elem, type, extra ) :  
  
 // 设置宽高  
 jQuery.style( elem, type, value, extra );  
 }, type, chainable ? margin : undefined, chainable );  
 };  
 } );  
} );

### 以.outerHeight()为例

* + - 1. jQuery.each( { Height: "height", Width: "width" }这段代码
         1. 根据outerHeight，参数为true，extra会得到margin值，不传参数或false得到border值

// 根据不同的方法，产生不同的extra，为了传递给jQuery.css()  
// 如outerHeight(),不包括margin；未传入参数， 故margin，value都为undefined，extra返回border  
// 如outerHeight(true),包括margin，故extra返回的是margin  
extra = defaultExtra || ( margin === true || value === true ? "margin" : "border" );

* + - * 1. 又因调用outerHeight并未传入value值，会调用jQuery.css，参数分别为匹配到的元素elem，type为height，extra为border或margin

return value === undefined ?// 根据是否提供value值，判断是取值还是设置值  
  
 // 获取元素宽高，但不强制转换为数值  
 jQuery.css( elem, type, extra ) :  
  
 // 设置宽高  
 jQuery.style( elem, type, value, extra );

* + - 1. jQuery.css方法中
         1. 因为height具有hooks.get方法，因此会通过调用hooks.get获取值

if ( hooks && "get" in hooks ) {   
 val = hooks.get( elem, true, extra );  
}

* + - 1. jQuery.each( [ "height", "width" ]这段
         1. 函数开始调用hooks.get方法，因为当前此时元素不是display:none等情况，直接会调用getWidthOrHeight获取高度

get: function( elem, computed, extra ) {  
 if ( computed ) {   
 return rdisplayswap.test( jQuery.css( elem, "display" ) ) &&   
 ( !elem.getClientRects().length || !elem.getBoundingClientRect().width ) ?  
 swap( elem, cssShow, function() {  
 return getWidthOrHeight( elem, name, extra );  
 } ) :  
 getWidthOrHeight( elem, name, extra );  
 }  
},

* + - 1. getWidthOrHeight方法中
         1. 将参数进行调，elem为匹配元素，name为height，extra为border或margin，测试的元素非borderbox，故valueIsBorderBox为false，styles为当前元素的计算样式集

return ( val +  
   
 augmentWidthOrHeight(  
 elem,  
 name,  
 extra || ( isBorderBox ? "border" : "content" ),  
 valueIsBorderBox,  
 styles  
 )  
) + "px";

* + - 1. augmentWidthOrHeight方法中
         1. 如extra为margin，会调用jQuery.css计算margin-top，margin-bottom，然后将计算结果返回

if ( extra === "margin" ) {  
 // cssExpand:padding，margin的4个方向  
 val += jQuery.css( elem, extra + cssExpand[ i ], true, styles );  
}

## 位置操作

### $().offsetParent()--------------------------(L10043)

// 获取已经定位的最近的祖先元素  
// 相当于对this.offsetParent,html方法，定义了jQuery逻辑  
// 如下情况此方法会返回documentElement  
// 1) iframe里面没有offsetParent的元素，这个方法会返回父级window对象的documentElement  
// 2) 对于隐藏元素或独立元素  
// 3) 对于body或html元素  
//  
// 但上述说的例外一般会发生，只是被认为这些情况返回documentElement更合理  
// 然而，这个逻辑不能保证完全可靠，将来会改变  
offsetParent: function() {  
 return this.map( function() {  
 // HTMLElement.offsetParent :返回一个指向最近的（closest，指包含层级上的最近）包含该元素的定位元素  
 // HTMLElement.offsetParent :如果没有定位的元素，则 offsetParent 为最近的 table, table cell 或根元素（标准模式下为 html；quirks 模式下为 body）  
 // 由于返回table，table cell jQuery认为不合理，因此，重新书写了offsetParent逻辑  
 var offsetParent = this.offsetParent;  
  
 while ( offsetParent && jQuery.css( offsetParent, "position" ) === "static" ) {  
 offsetParent = offsetParent.offsetParent;  
 }  
  
 return offsetParent || documentElement;  
 } );  
}

### $().position()---------------------(L10002)

/\*\*  
 \* 获取匹配元素(匹配集合的第一个元素)相对父元素的偏移。  
 \* 返回值可能是分数，  
 \* @return {{top: number, left: number}}  
 \*/  
position: function() {  
 // 未获得匹配元素，直接返回  
 if ( !this[ 0 ] ) {  
 return;  
 }  
  
 var offsetParent, offset,  
 elem = this[ 0 ],  
 parentOffset = { top: 0, left: 0 };// 默认父级元素的top和left为0  
  
 // fixed元素是根据window偏移的 ，window对象是offset的父级(parentOffset = {top:0, left: 0},  
 if ( jQuery.css( elem, "position" ) === "fixed" ) {  
  
 // 使用getBoundingClientRect 计算  
 offset = elem.getBoundingClientRect();  
  
 } else {  
  
 // 获取实际的已定位父级元素  
 offsetParent = this.offsetParent();  
  
 // 获取正确的偏移量  
 offset = this.offset();  
 if ( !nodeName( offsetParent[ 0 ], "html" ) ) {  
 parentOffset = offsetParent.offset(); // 获取已定位父级相对document的偏移  
 }  
  
 // 因为offset得到的是针对borderbox的  
 parentOffset = {  
 top: parentOffset.top + jQuery.css( offsetParent[ 0 ], "borderTopWidth", true ),  
 left: parentOffset.left + jQuery.css( offsetParent[ 0 ], "borderLeftWidth", true )  
 };  
 }  
  
 // 减去父级偏移与元素margin  
 return {  
 top: offset.top - parentOffset.top - jQuery.css( elem, "marginTop", true ),  
 left: offset.left - parentOffset.left - jQuery.css( elem, "marginLeft", true )  
 };  
},

### $().offset()----------------------------(L9964)

* + - 1. jQuery.offset. setOffset------------------(L9912)

jQuery.offset = {  
 // 根据opitons，为元素设置相对document的偏移，i为每一个匹配元素序号  
 setOffset: function( elem, options, i ) {  
 var curPosition, curLeft, curCSSTop, curTop, curOffset, curCSSLeft, calculatePosition,  
 position = jQuery.css( elem, "position" ),  
 curElem = jQuery( elem ),  
 props = {};  
  
 // 首先设置postion属性，因为默认值static，会忽略top, bottom, left, right  
 if ( position === "static" ) {  
 elem.style.position = "relative";  
 }  
  
 curOffset = curElem.offset();// 获取当前元素相对于document的偏移量  
 curCSSTop = jQuery.css( elem, "top" );  
 curCSSLeft = jQuery.css( elem, "left" );  
 // relative的auto是0，而absolute与fixed的auto并不相当于0，因为它们是相对最近的有定位祖先节点或根元素的  
 calculatePosition = ( position === "absolute" || position === "fixed" ) &&  
 ( curCSSTop + curCSSLeft ).indexOf( "auto" ) > -1;  
  
 // 如postion为absolute或fixed，兵器top或left不是atuo，主要计算位置  
 if ( calculatePosition ) {  
 // 为auto时，获取相对最近的有定位祖先节点或根元素的距离  
 curPosition = curElem.position();  
 curTop = curPosition.top;  
 curLeft = curPosition.left;  
  
 } else {  
 // relative无需计算，需要去除单位  
 curTop = parseFloat( curCSSTop ) || 0;  
 curLeft = parseFloat( curCSSLeft ) || 0;  
 }  
 // options为函数时，调用得到修正后需要设置的offset()坐标，并赋值给options  
 if ( jQuery.isFunction( options ) ) {  
  
 // fn(i, options) 可自定义修正坐标  
 options = options.call( elem, i, jQuery.extend( {}, curOffset ) );  
 }  
  
 if ( options.top != null ) {  
 props.top = ( options.top - curOffset.top ) + curTop;  
 }  
 if ( options.left != null ) {  
 props.left = ( options.left - curOffset.left ) + curLeft;  
 }  
 // 可以通过using属性定义钩子函数，取代默认的写入  
 if ( "using" in options ) {  
 options.using.call( elem, props );  
  
 } else {// 默认直接通过{top: x, left: x}对象形式调用css写入  
 curElem.css( props );  
 }  
 }  
};

* + - 1. 源码分析

/\*\*  
 \* 获取匹配元素相对于document的偏移量，无论如何改变浏览器大小，滚动，位置不变  
 \* .offset()：获取相对于document的偏移量  
 \* .offset( {top:'',left:''} )：根据对象设置偏移  
 \* @param options  
 \* @return {\*}  
 \*/  
offset: function( options ) {  
  
 // 为了保证链式调用,根据options设置值  
 if ( arguments.length ) {  
 return options === undefined ?  
 this :  
 this.each( function( i ) {  
 jQuery.offset.setOffset( this, options, i );  
 } );  
 }  
  
 var doc, docElem, rect, win,  
 elem = this[ 0 ];  
  
 if ( !elem ) {  
 return;  
 }  
  
 // 针对不在dom树中的元素或hidden元素（display：none）返回0(gh-2310)  
 // Support: IE <=11 only  
 // 在IE中对非DOM树元素使用getBoundingClientRect 会抛出错误  
 if ( !elem.getClientRects().length ) {  
 return { top: 0, left: 0 };  
 }  
 // 注意：返回元素的大小及其相对于视口的位置。故会随滚动条滚动left，top等值不同  
 // 这个方法是把元素看做为border-box，即，padding和border都在width和height内部  
 rect = elem.getBoundingClientRect();  
  
 doc = elem.ownerDocument;// 节点的顶层document对象  
 docElem = doc.documentElement;// html  
 // 为何要使用document.defalutView.pageYOffset,而不直接用pageYOffset，或window.pageYOffset  
 // 修复document对象不在页面中，而可能在内存中；或pageYOffset被同名对象覆盖，或docuemnt在inframe或popup中使用  
 // 浏览器中document.defalutView返回window  
 win = doc.defaultView;  
  
 return {  
 // pageYOffset滚动条位置  
 top: rect.top + win.pageYOffset - docElem.clientTop,  
 left: rect.left + win.pageXOffset - docElem.clientLeft  
 };  
},

## $().scrollLeft,$().scrollTop--------------(L10066)

// 构建 scrollLeft and scrollTop 方法  
// 获取匹配元素相对滚动条顶部的偏移。  
// 相对于Element.scrollLeft，兼容了document与window对象的偏移量设置与取值  
jQuery.each( { scrollLeft: "pageXOffset", scrollTop: "pageYOffset" }, function( method, prop ) {  
 var top = "pageYOffset" === prop;  
  
 jQuery.fn[ method ] = function( val ) {  
 return access( this, function( elem, method, val ) {  
  
 // 判断elem是否为window对象,如不是window对象，只是普通elme，则win为undefined  
 var win;  
 if ( jQuery.isWindow( elem ) ) {  
 win = elem;  
 } else if ( elem.nodeType === 9 ) {  
 win = elem.defaultView;  
 }  
 // 取值，如是window对象，则用window.pageXOffset获取滚动位置  
 // 否则再调用elem.scrollLeft,Element.scrollLeft 属性可以读取或设置元素滚动条到元素左边的距离。  
 if ( val === undefined ) {  
 return win ? win[ prop ] : elem[ method ];  
 }  
  
 if ( win ) {  
 // scrollTo滚动到文档中的某个坐标。  
 // window.scrollTo(x-coord,y-coord )  
 // 因为方法是只滚动x，或y，因此，滚动x时，需要给scrollTo的第二个参数传入当前y偏移值  
 win.scrollTo(  
 !top ? val : win.pageXOffset,  
 top ? val : win.pageYOffset  
 );  
  
 } else {  
 elem[ method ] = val;  
 }  
 }, method, val, arguments.length );  
 };  
} );

## jQuery.cssHooks.left，jQuery.cssHooks.top

// Support: Safari <=7 - 9.1, Chrome <=37 - 49  
// 使用jQuery.fn.position增加jQuery.cssHooks.top，jQuery.cssHooks.left  
// Webkit bug: https://bugs.webkit.org/show\_bug.cgi?id=29084  
// Blink bug: https://bugs.chromium.org/p/chromium/issues/detail?id=589347  
// getComputedStyle 会对指定了top/left/bottom/right返回百分数值，这个值并不是是当前模块相对于偏移模块的值，利用position检查下  
// 通过curCSS，判断top与left值是否为百分数，如是，则使用position获得与定位父级元素的偏移，否则使用原值  
jQuery.each( [ "top", "left" ], function( i, prop ) {  
 jQuery.cssHooks[ prop ] = addGetHookIf( support.pixelPosition,  
 function( elem, computed ) {  
 if ( computed ) {  
 computed = curCSS( elem, prop );  
  
 // 如当前返回百分数值，使用offset计算偏移量  
 return rnumnonpx.test( computed ) ?  
 jQuery( elem ).position()[ prop ] + "px" :  
 computed;  
 }  
 }  
 );  
} );