闭包的用途：

可以读取函数内部的变量；让这些变量的值始终保持在内存中

function f1(){

　　　　var n=999;

　　　　nAdd=function(){n+=1}

　　　　function f2(){

　　　　　　alert(n);

　　　　}

　　　　return f2;

　　}

　　var result=f1();

　　result(); // 999

　　nAdd();

　　result(); // 1000

使用闭包的注意点

1）由于闭包会使得函数中的变量都被保存在内存中，内存消耗很大，

所以不能滥用闭包，否则会造成网页的性能问题，在IE中可能导致内存泄露。

解决方法是，在退出函数之前，将不使用的局部变量全部删除。

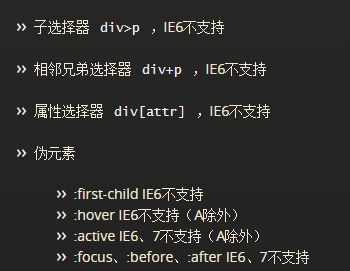
2）闭包会在父函数外部，改变父函数内部变量的值。

所以，如果你把父函数当作对象（object）使用，把闭包当作它的公用方法（Public Method），

把内部变量当作它的私有属性（private value），这时一定要小心，不要随便改变父函数内部变量的值。

**浏览器兼容性问题小结（包括html css js）：内容出处**[**http://luics.com/fe-compatibility.html**](http://luics.com/fe-compatibility.html)

[**http://www.qdfuns.com/notes/18372/bf9885f9902262654cd4d12cd1b65a11**](http://www.qdfuns.com/notes/18372/bf9885f9902262654cd4d12cd1b65a11)



* HTML对象获取问题  
  FireFox：document.getElementById("idName");  
  ie:document.idname或者document.getElementById("idName").  
  解决办法：统一使用document.getElementById("idName");
* 说明:Firefox下,可以使用const关键字或var关键字来定义常量;  
  IE下,只能使用var关键字来定义常量.   
  解决方法：统一使用var关键字来定义常量.
* event.x与event.y问题  
  说明:IE下,event对象有x,y属性,但是没有pageX,pageY属性;  
  Firefox下,event对象有pageX,pageY属性,但是没有x,y属性.   
  解决方法：使用mX(mX   =   event.x   ?   event.x   :   event.pageX;)来代替IE下的event.x或者Firefox下的event.pageX.
* css中的width和padding  
  在IE7和FF中width宽度不包括padding，在Ie6中包括padding
* FF和IEBOX模型解释不一致导致相差2px  
  box.style{width:100;border 1px;}   
  ie理解为box.width = 100   
  ff理解为box.width = 100 + 1\*2 = 102  //加上边框2px
* 设置为float的div在ie下设置的margin会加倍。这是一个ie6都存在的bug。解决方案是在这个div里面加上display:inline;   
  例如：  
  <div id=”imfloat”>  
  相应的css为  
  #imfloat{   
  float:left;   
  margin:5px;   
  display:inline;}
* IE与宽度和高度的问题  
  IE不认得min-这个定义，但实际上它把正常的width和height当作有min的情况来使。这样问题就大了，如果只用宽度和高度，正常的浏览器里这两个值就不会变，如果只用min-width和min-height的话，IE下面根本等于没有设置宽度和高度。  
  比如要设置背景图片，这个宽度是比较重要的。要解决这个问题，可以这样：  
  #box{ width: 80px; height: 35px;}html>body #box{ width: auto; height: auto; min-width: 80px; min-height: 35px;}
* IE6下图片下有空隙产生  
  解决这个BUG的技巧有很多,可以是改变html的排版,或者设置img为display:block或者设置vertical-align属性为vertical-align:top/bottom/middle/text-bottom 都可以解决.
* LI中内容超过长度后以省略号显示  
  此技巧适用与IE、Opera、safari、chrom浏览器，FF暂不支持。  
    
  <style type="text/css">  
  <!--  
  li {   
  width:200px;   
  white-space:nowrap;   
  text-overflow:ellipsis;   
  -o-text-overflow:ellipsis;   
  overflow: hidden;   
  }  
    
  -->  
  </style>
* IE和FireFox 对空格的尺寸解释不同，FireFox为4px,IE为8px; FireFox对div与div之间的空格是忽略的，但是IE是处理的。因此在两个相邻div之间不要有空格跟回车，否则可能造成不同浏览间之间格式不正确，比如著名的3px偏差（多个img标签连着，然后定义float: left;结果在firefox里面正常，而IE里面显示的每个img都相隔了3px。我把标签之间的空格都删除都没有作用。解决方法是在img外面套li，并且对li定义margin: 0; 避免方式：在必要的时候不要无视 list 标签）而且原因难以查明

**从输入 URL 到页面加载完成的过程中都发生了什么事情？**

1. 浏览器向DNS服务器查找输入URL对应的IP地址。
2. DNS服务器返回网站的IP地址。
3. 浏览器根据IP地址与目标web服务器在80端口上建立TCP连接
4. 浏览器获取请求页面的html代码。
5. 浏览器在显示窗口内渲染HTML。
6. 窗口关闭时，浏览器终止与服务器的连接。

1、浏览器的url请求   
2、递归寻找DNS服务器   
3、连接目标IP并建立TCP连接   
4、向目标服务器发送http请求   
5、web服务器接收请求后处理   
6、web服务器返回相应的结果【无效、重定向、正确页面等】 

7、浏览器接收返回的http内容

================================前端解析分割线=========================

8、开始解析html文件，当然是自上而下，先是头部，后是body

9、当解析到头部css外部链接时，同步去下载，如果遇到外部js链接也是下载【不过js链接不建议放在头部，因为耽误页面第一展现时间】

10、接着解析body部分，边解析边开始生成对应的DOM树，同时等待css文件下载

11、一旦css文件下载完毕，那么就同步去用已经生成的DOM节点+CSS去生成渲染树

12、渲染树一旦有结构模型了，接着就会同步去计算渲染树节点的布局位置

13、一旦计算出来渲染的坐标后，又同步去开始渲染

14、10-13步进行过程中如果遇到图片则跳过去渲染下面内容，等待图片下载成功后会返回来在渲染原来图片的位置

15、同14步，如果渲染过程中出现js代码调整DOM树机构的情况，也会再次重新来过，从修改DOM那步开始

16、最终所有节点和资源都会渲染完成

=============================分析结束分割线======================

17、渲染完成后开始page的onload事件   
18、整个页面load完成

**清除float问题；**

**方法一：添加新的元素 、应用 clear：both；**

<div class="outer">

<div class="div1">1</div>

<div class="div2">2</div>

<div class="div3">3</div>

<div class="clear"></div>

</div>

.clear{clear:both; height: 0; line-height: 0; font-size: 0}

**方法二：父级div定义 overflow: auto（注意：是父级div也就是这里的 div.outer）**

<div class="outer over-flow"> //这里添加了一个class

<div class="div1">1</div>

<div class="div2">2</div>

<div class="div3">3</div>

<!--<div class="clear"></div>-->

</div>

.over-flow{ overflow: auto; zoom: 1; //zoom: 1; 是在处理兼容性问题}

**方法三：  :after 方法：（注意：作用于浮动元素的父亲）**

.outer {zoom:1;} /\*==for IE6/7 Maxthon2==\*/

.outer :after {clear:both;content:'.';display:block;width: 0;height: 0;visibility:hidden;}/\*==for FF/chrome/opera/IE8==\*/

**div居中问题**

**对于有宽高的div垂直居中：**

position: absolute;

left: 50%;

top: 50%;

width:200px;

height:100px;

margin-left:-100px;

margin-top:-50px;

**对于不固定宽高的div垂直居中：**

**position: fixed;**

**top: 50%;**

**left: 50%;**

**background-color: #000;**

**width:50%;**

**height: 50%;**

**-webkit-transform: translateX(-50%) translateY(-50%);**

### css3不定宽高水平垂直居中

实现不定宽高水平垂直居中。

justify-content:center;//子元素水平居中

align-items:center;//子元素垂直居中

display:-webkit-flex;

在父级元素上面加上上面3句话，就可以实现子元素水平垂直居中。

**浏览器的标准模式和怪异模式**

各个浏览器在对页面的渲染上存在差异，甚至同一浏览器在不同版本中，对页面的渲染也不同。

由于W3C标准的推出，浏览器渲染页面有了统一的标准，产生了Quircks mode和Standars mode，两种渲染方法共存在一个浏览器上。

火狐一直工作在标准模式下，但IE（6，7，8）标准模式与怪异模式差别很大，主要体现在对盒子模型的解释上。

* 浏览器对没有doctype声明的网页采用quirks mode解析。
* 对于那些浏览器不能识别的doctype声明，浏览器采用strict mode解析。
* 没有使用DTD声明或者使用HTML4以下（不包括HTML4）的DTD声明时，基本所有的浏览器都是使用quirks mode呈现
* 在现有有doctype声明的网页，绝大多数是采用strict mode进行解析的
* 在ie6中，如果在doctype声明前有一个xml声明(比如:<?xml version=”1.0″ encoding=”iso-8859-1″?>)，则采用quirks mode解析

如何判定现在是标准模式还是怪异模式：

方法一：执行以下代码  
alert(window.top.document.compatMode) ;  
//BackCompat  表示怪异模式  
//CSS1Compat  表示标准模式

方法二：jquery为我们提供的方法，如下：  
alert($.boxModel)  
alert($.support.boxModel)

盒模型

盒模型默认的值是content-box, 新增的值是padding-box和border-box，几种盒模型计算元素宽高的区别如下：

### content-box（默认）

布局所占宽度Width：

Width = width + padding-left + padding-right + border-left + border-right

布局所占高度Height:

Height = height + padding-top + padding-bottom + border-top + border-bottom

### padding-box

布局所占宽度Width：

Width = width(包含padding-left + padding-right) + border-top + border-bottom

布局所占高度Height:

Height = height(包含padding-top + padding-bottom) + border-top + border-bottom

### border-box

布局所占宽度Width：

Width = width(包含padding-left + padding-right + border-left + border-right)

布局所占高度Height:

Height = height(包含padding-top + padding-bottom + border-top + border-bottom)