

弹层组件文档 - layui.layer

layer至今仍作为layui的代表作，她的受众广泛并非偶然，而是这五年多的坚持，不断完善和维护、不断建设和提升社区服务，使得猿们纷纷自发传播，乃至成为今天的Layui最强劲的源动力。目前，layer已成为国内最多人使用的web弹层组件，[GitHub](#)自然Stars3000+，官网累计下载量达30w+，大概有20万Web平台正在使用layer。

之所以列举上面这些数字，并非是在炫耀（也没什么好炫耀的），而是懂layer的人都知道，这是一种怎样不易的坚持。由于layer在Layui体系中的位置比较特殊，甚至让很多人都误以为 Layui === Layer ui，所以再次强调layer只是作为Layui的一个弹层模块，由于其用户基数较大，所以至今仍把她作为独立组件来维护。不过请注意：无论是独立的layer，还是作为内置模块的layer，文档都以本页为准。

模块加载名称：[layer](#)，layer独立组件官网：[layer.layui.com](#)

使用场景

由于layer可以独立使用，也可以通过Layui模块化使用。所以请按照你的实际需求来选择。

不同点	作为独立组件使用	Layui模块化使用
用前准备	如果你不想使用Layui，而只是想使用layer，你可以去layer独立组件官网下载组件包。你需要在你的页面引入jQuery1.8以上的任意版本，并引入layer.js。	如果你使用的是Layui，那么你直接在官网下载layui框架即可，无需引入jQuery和layer.js，但需要引入layui.css和layui.js
调用方式	通过script引入layer.js后，直接用即可。 参考： 快速上手	需要通过layui.use('layer', callback)加载模块如下代码所示

作为独立组件使用layer[layui.code](#)

```
1. 引入好layer.js后，直接用即可
2. <script src="layer.js"></script>
3. <script>
4. layer.msg('hello');
5. </script>
6.
```

在layui中使用layer[layui.code](#)

```
1. layui.use('layer', function(){
2.     var layer = layui.layer;
3.
4.     layer.msg('hello');
5. });
6.
```

除了上面有所不同，其它都完全一致。

基础参数

我们提到的基础参数主要指调用方法时用到的配置项，如：layer.open({content: ''})layer.msg('', {time: 3})等，其中的content和time即是基础参数，以键值形式存在，基础参数可合理应用于任何层类型中，您不需要所有都去配置，大多数都是可选的。而其中的layer.open、layer.msg就是内置方法。注意，从2.3开始，无需通过layer.config来加载拓展模块

type - 基本层类型

类型：Number，默认：0

layer提供了5种层类型。可传入的值有：0（信息框，默认）1（页面层）2（iframe层）3（加载层）4（tips层）。若您采用layer.open({type: 1})方式调用，则type为必填项（信息框除外）

title - 标题

类型：String/Array/Boolean，默认：'信息'

title支持三种类型的值，若您传入的是普通的字符串，如title: '我是标题'，那么只会改变标题文本；若您还需要自定义标题区域样式，那么您可以title: ['文本', 'font-size: 18px;']，数组第二项可以写任意css样式；如果您不想显示标题栏，您可以title: false

content - 内容

类型：String/DOM/Array，默认：''

content可传入的值是灵活多变的，不仅可以传入普通的html内容，还可以指定DOM，更可以随着type的不同而不同。譬如：

code[layui.code](#)

```
1. /*
2. 如果是页面层
3. */
4. layer.open({
5.     type: 1,
```

```

6.   content: '传入任意的文本或html' //这里content是一个普通的String
7.   });
8.   layer.open({
9.     type: 1,
10.    content: $('#id') //这里content是一个DOM，注意：最好该元素要存放在body最外层，否则可能被其它的相对元素所影响
11.  });
12.  //Ajax获取
13.  $.post('url', {}, function(str){
14.    layer.open({
15.      type: 1,
16.      content: str //注意，如果str是object，那么需要字符拼接。
17.    });
18.  });
19.  /*
20.   如果是iframe层
21.   */
22.  layer.open({
23.    type: 2,
24.    content: 'http://sentsin.com' //这里content是一个URL，如果你不想让iframe出现滚动条，你还可以content: ['http://sen
tsin.com', 'no']
25.  });
26.  /*
27.   如果是用layer.open执行tips层
28.   */
29.  layer.open({
30.    type: 4,
31.    content: ['内容', '#id'] //数组第二项即吸附元素选择器或者DOM
32.  });
33.

```

skin - 样式类名

类型：**String**，默认：''

skin不仅允许你传入**layer**内置的样式**class**名，还可以传入您自定义的**class**名。这是一个很好的切入点，意味着你可以借助**skin**轻松完成不同的风格定制。目前**layer**内置的**skin**有：*layui-layer-lan**layui-layer-molv*，未来我们还会选择性地内置更多，但更推荐您自己来定义。以下是一个自定义风格的简单例子

code_layui.code

```

1.
2. //单个使用
3. layer.open({
4.   skin: 'demo-class'
5. });
6. //全局使用。即所有弹出层都默认采用，但是单个配置skin的优先级更高
7. layer.config({
8.   skin: 'demo-class'
9. })
10. //CSS
11. body .demo-class .layui-layer-title{background:#c00; color:#fff; border: none;}
12. body .demo-class .layui-layer-btn{border-top:1px solid #E9E7E7}
13. body .demo-class .layui-layer-btn a{background:#333;}
14. body .demo-class .layui-layer-btn .layui-layer-btn1{background:#999;}
15. ...
16. 加上body是为了保证优先级。你可以借助Chrome调试工具，定义更多样式控制层更多的区域。
17.

```

你也可以[去看看layer皮肤制作说明](#)

area - 宽高

类型：**String/Array**，默认：'auto'

在默认状态下，**layer**是宽高都自适应的，但当你只想定义宽度时，你可以`area: '500px'`，高度仍然是自适应的。当你宽高都要定义时，你可以`area: ['500px', '300px']`

offset - 坐标

类型：**String/Array**，默认：垂直水平居中

offset默认情况下不用设置。但如果你不想垂直水平居中，你还可以进行以下赋值：

值	备注
offset: '100px'	只定义top坐标，水平保持居中
offset: ['100px', '50px']	同时定义top、left坐标
offset: 't'	快捷设置顶部坐标
offset: 'r'	快捷设置右边缘坐标
offset: 'b'	快捷设置底部坐标
offset: 'l'	快捷设置左边缘坐标

值	备注
offset: 'lt'	快捷设置左上角
offset: 'lb'	快捷设置左下角
offset: 'rt'	快捷设置右上角
offset: 'rb'	快捷设置右下角

icon - 图标。信息框和加载层的私有参数

类型：Number，默认：-1（信息框）/0（加载层）

信息框默认不显示图标。当你想显示图标时，默认皮肤可以传入0-6如果是加载层，可以传入0-2。如：

code`layui.code`

```
1. //eg1
2. layer.alert('酷毙了', {icon: 1});
3. //eg2
4. layer.msg('不开心。。', {icon: 5});
5. //eg3
6. layer.load(1); //风格1的加载
7.
```

btn - 按钮

类型：String/Array，默认：'确认'

信息框模式时，**btn**默认是一个确认按钮，其它层类型则默认不显示，加载层和**tips**层则无效。当您只想自定义一个按钮时，你可以以**btn: '我知道了'**，当你要定义两个按钮时，你可以**btn: ['yes', 'no']**。当然，你也可以定义更多按钮，比如：**btn: ['按钮1', '按钮2', '按钮3', ...]**，按钮1的回调是**yes**，而从按钮2开始，则回调为**btn2: function(){}，以此类推。如：**

code`layui.code`

```
1. //eg1
2. layer.confirm('纳尼?', {
3.   btn: ['按钮一', '按钮二', '按钮三'] //可以无限个按钮
4.   ,btn3: function(index, layer){
5.     //按钮【按钮三】的回调
6.   }
7. }, function(index, layer){
8.   //按钮【按钮一】的回调
9. }, function(index){
10.   //按钮【按钮二】的回调
11. });
12.
13. //eg2
14. layer.open({
15.   content: 'test'
16.   ,btn: ['按钮一', '按钮二', '按钮三']
17.   ,yes: function(index, layer){
18.     //按钮【按钮一】的回调
19.   }
20.   ,btn2: function(index, layer){
21.     //按钮【按钮二】的回调
22.   }
23.   //return false 开启该代码可禁止点击该按钮关闭
24. }
25.   ,btn3: function(index, layer){
26.     //按钮【按钮三】的回调
27.   }
28.   //return false 开启该代码可禁止点击该按钮关闭
29. }
30.   ,cancel: function(){
31.     //右上角关闭回调
32.   }
33.   //return false 开启该代码可禁止点击该按钮关闭
34. }
35. });
36.
```

btnAlign - 按钮排列

类型：String，默认：r

你可以快捷定义按钮的排列位置，**btnAlign**的默认值为**r**，即右对齐。该参数可支持的赋值如下：

值	备注
btnAlign: 'l'	按钮左对齐
btnAlign: 'c'	按钮居中对齐
btnAlign: 'r'	按钮右对齐。默认值，不用设置

closeBtn - 关闭按钮

类型: **String/Boolean**, 默认: 1

layer提供了两种风格的关闭按钮, 可通过配置1和2来展示, 如果不显示, 则`closeBtn: 0`

shade - 遮罩

类型: **String/Array/Boolean**, 默认: 0.3

即弹层外区域。默认是0.3透明度的黑色背景（'#000'）。如果你想定义别的颜色, 可以`shade: [0.8, '#393D49']`; 如果你不想显示遮罩, 可以`shade: 0`

shadeClose - 是否点击遮罩关闭

类型: **Boolean**, 默认: false

如果你的`shade`是存在的, 那么你可以设定`shadeClose`来控制点击弹层外区域关闭。

time - 自动关闭所需毫秒

类型: **Number**, 默认: 0

默认不会自动关闭。当你想自动关闭时, 可以`time: 5000`, 即代表5秒后自动关闭, 注意单位是毫秒（1秒=1000毫秒）

id - 用于控制弹层唯一标识

类型: **String**, 默认: 空字符

设置该值后, 不管是什么类型的层, 都只允许同时弹出一个。一般用于页面层和`iframe`层模式

anim - 弹出动画

类型: **Number**, 默认: 0

我们的出场动画全部采用CSS3。这意味着除了ie6-9, 其它所有浏览器都是支持的。目前`anim`可支持的动画类型有0-6 如果不想显示动画, 设置 `anim: -1` 即可。另外需要注意的是, 3.0之前的版本用的是 `shift` 参数

isOutAnim - 关闭动画（layer 3.0.3新增）

类型: **Boolean**, 默认: true

默认情况下, 关闭层时会有一个过度动画。如果你不想开启, 设置 `isOutAnim: false` 即可

maxmin - 最大最小化。

类型: **Boolean**, 默认: false

该参数值对`type:1`和`type:2`有效。默认不显示最大化按钮。需要显示配置`maxmin: true`即可

fixed - 固定

类型: **Boolean**, 默认: true

即鼠标滚动时, 层是否固定在可视区域。如果不想, 设置`fixed: false`即可

resize - 是否允许拉伸

类型: **Boolean**, 默认: true

默认情况下, 你可以在弹层右下角拖动来拉伸尺寸。如果对指定的弹层屏蔽该功能, 设置 `false`即可。该参数对`loading`、`tips`层无效

resizing - 监听窗口拉伸动作

类型: **Function**, 默认: null

当你拖拽弹层右下角对窗体进行尺寸调整时, 如果你设定了该回调, 则会执行。回调返回一个参数: 当前层的DOM对象

`code`[layui.code](#)

```
1. resizing: function(layero){
2.   console.log(layero);
3. }
```

scrollbar - 是否允许浏览器出现滚动条

类型: **Boolean**, 默认: **true**

默认允许浏览器滚动, 如果设定`scrollbar: false`, 则屏蔽

maxWidth - 最大宽度

类型: **Number**, 默认: **360**

请注意: 只有当`area: 'auto'`时, `maxWidth`的设定才有效。

zIndex - 层叠顺序

类型: **Number**, 默认: **19891014** (贤心生日 0.0)

一般用于解决和其它组件的层叠冲突。

move - 触发拖动的元素

类型: **String/DOM/Boolean**, 默认: **'layui-layer-title'**

默认是触发标题区域拖拽。如果你想单独定义, 指向元素的选择器或者DOM即可。如`move: '.mine-move'`。你还配置设定`move: false`来禁止拖拽

moveOut - 是否允许拖拽到窗口外

类型: **Boolean**, 默认: **false**

默认只能在窗口内拖拽, 如果你想让拖到窗外, 那么设定`moveOut: true`即可

moveEnd - 拖动完毕后的回调方法

类型: **Function**, 默认: **null**

默认不会触发`moveEnd`, 如果你需要, 设定`moveEnd: function(layero){}`即可。其中`layero`为当前层的DOM对象

tips - tips方向和颜色

类型: **Number/Array**, 默认: **2**

`tips`层的私有参数。支持上右下左四个方向, 通过1-4进行方向设定。如`tips: 3`则表示在元素的下面出现。有时你还可能会定义一些颜色, 可以设定`tips: [1, '#c00']`

tipsMore - 是否允许多个tips

类型: **Boolean**, 默认: **false**

允许多个意味着不会销毁之前的`tips`层。通过`tipsMore: true`开启

success - 层弹出后的成功回调方法

类型: **Function**, 默认: **null**

当你需要在层创建完毕时即执行一些语句, 可以通过该回调。`success`会携带两个参数, 分别是当前层DOM当前层索引。如:

code_layui.code

```
1. layer.open({
2.   content: '测试回调',
3.   success: function(layero, index){
4.     console.log(layero, index);
5.   }
6. });
7.
```

yes - 确定按钮回调方法

类型: **Function**, 默认: **null**

该回调携带两个参数, 分别为当前层索引、当前层DOM对象。如:

code_layui.code

```

1. layer.open({
2.   content: '测试回调',
3.   yes: function(index, layero){
4.     //do something
5.     layer.close(index); //如果设定了yes回调，需进行手工关闭
6.   }
7. });
8.

```

cancel - 右上角关闭按钮触发的回调

类型：Function，默认：null

该回调携带两个参数，分别为：当前层索引参数（index）、当前层的DOM对象（layero），默认会自动触发关闭。如果不想关闭，`return false`即可，如：

code_layui.code

```

1. cancel: function(index, layero){
2.   if(confirm('确定要关闭么')){ //只有当点击confirm框的确定时，该层才会关闭
3.     layer.close(index)
4.   }
5.   return false;
6. }
7.

```

end - 层销毁后触发的回调

类型：Function，默认：null

无论是确认还是取消，只要层被销毁了，`end`都会执行，不携带任何参数。

full/min/restore - 分别代表最大化、最小化、还原 后触发的回调

类型：Function，默认：null

携带一个参数，即当前层DOM

layer.config(options) - 初始化全局配置

这是一个可以重要也可以不重要的方法，重要的是，它的权利真的很大，尤其是在模块化加载layer时，你会发现你必须要用到它。它不仅配置一些诸如路径、加载的模块，甚至还可以决定整个弹层的默认参数。而说它不重要，是因为多数情况下，你会发现，你似乎不是那么十分需要它。但你真的需要认识一下这位伙计。

如果您是采用seajs或者requirejs加载layer，您需要执行该方法来完成初始化的配置。比如：

code_layui.code

```

1. layer.config({
2.   path: '/res/layer/' //layer.js所在的目录，可以是绝对目录，也可以是相对目录
3. });
4. //这样的话，layer就会去加载一些它所需要的配件，比如css等。
5. //当然，你即便不用seajs或者requirejs，也可以通过上述方式设定路径
6.

```

如果你是采用`<script src="?a.js&layer.js">`这种合并的方式引入layer，那么您需要在script标签上加一个自定义属性`merge="true"`。如：

code_layui.code

```

1. <script src="?a.js&layer.js" merge="true">
2. 这样的话，layer就不会去自动去获取路径，但你需要通过以下方式来完成初始化的配置
3. layer.config({
4.   path: '/res/layer/' //layer.js所在的目录，可以是绝对目录，也可以是相对目录
5. });
6.

```

注意：如果采用 layui 加载 layer，无需设置 path。所以前置工作都是自动完成。

但layer.config的作用远不止上述这样。它还可以配置层默认的基础参数，如：

code_layui.code

```

1. layer.config({
2.   anim: 1, //默认动画风格
3.   skin: 'layui-layer-molv' //默认皮肤
4.   ...
5. });
6. //除此之外，extend还允许你加载拓展的css皮肤，如：
7. layer.config({
8.   //如果是独立版的layer，则将myskin存放在./skin目录下
9.   //如果是layui中使用layer，则将myskin存放在./css/modules/layer目录下
10.  extend: 'myskin/style.css'
11. });
12. //具体的皮肤定制，可以参见：skin参数说明

```

layer.ready(callback) - 初始化就绪

由于我们的layer内置了轻量级加载器，所以你根本不需要单独引入css等文件。但是加载总是需要过程的。当你在页面一打开就要执行弹层时，你最好是将弹层放入ready方法中，如：

code layui.code

```
1. //页面一打开就执行弹层
2. layer.ready(function() {
3.     layer.msg('很高兴一开场就见到你');
4. });
5.
```

我是华丽的酱油：介绍完上面两位引导者，接下来我们真正的主角闪亮登场了。此处应有掌声 ^^

layer.open(options) - 原始核心方法

基本上是露脸率最高的方法，不管是使用哪种方式创建层，都是走layer.open()，创建任何类型的弹层都会返回一个当前层索引，上述的options即是基础参数，另外，该文档统一采用options作为基础参数的标识例子：

code layui.code

```
1. var index = layer.open({
2.     content: 'test'
3. });
4. //拿到的index是一个重要的凭据，它是诸如layer.close(index)等方法的必传参数。
5.
```

噢，请等等，上面这位主角的介绍篇幅怎么看怎么都觉得跟它的地位不符，作者在文档中只给了它如此可怜的一块地？？这是因为，它真的已经大众得不能再大众了，你真正需要了解的，是它的内部器官，即上面一大篇幅介绍的各种基础参数。←_←

layer.alert(content, options, yes) - 普通信息框

它的弹出似乎显得有些高调，一般用于对用户造成比较强烈的关注，类似系统alert，但却比alert更灵便。它的参数是自动向左补齐的。通过第二个参数，可以设定各种你所需要的基础参数，但如果你不需要的話，直接写回调即可。如

code layui.code

```
1. //eg1
2. layer.alert('只想简单的提示');
3. //eg2
4. layer.alert('加了个图标', {icon: 1}); //这时如果你还想执行yes回调，可以放在第三个参数中。
5. //eg3
6. layer.alert('有了回调', function(index) {
7.     //do something
8.
9.     layer.close(index);
10. });
11.
```

layer.confirm(content, options, yes, cancel) - 询问框

类似系统confirm，但却远胜confirm，另外它不是和系统的confirm一样阻塞你需要把交互的语句放在回调体中。同样的，它的参数也是自动补齐的。

code layui.code

```
1. //eg1
2. layer.confirm('is not?', {icon: 3, title: '提示'}, function(index) {
3.     //do something
4.
5.     layer.close(index);
6. });
7. //eg2
8. layer.confirm('is not?', function(index) {
9.     //do something
10.
11.     layer.close(index);
12. });
13.
```

layer.msg(content, options, end) - 提示框

我们在源码中用了相对较大的篇幅来定制了这个msg，目的是想将其打造成露脸率最高的提示框。而事实上我的确也在大量地使用它。因为它简单，而且足够得自觉，它不仅占据很少的面积，而且默认还会3秒后自动消失所有这一切都决定了我对msg的爱。因此我赋予了她许多可能在外形方面，它坚持简陋的变化，在作用方面，他坚持零用户操作。而且它的参数也是自动补齐的。

code layui.code

```
1. //eg1
2. layer.msg('只想弱弱提示');
3. //eg2
```

```

4. layer.msg('有表情地提示', {icon: 6});
5. //eg3
6. layer.msg('关闭后想做什么', function(){
7.     //do something
8. });
9. //eg
10. layer.msg('同上', {
11.     icon: 1,
12.     time: 2000 //2秒关闭（如果不配置，默认是3秒）
13. }, function(){
14.     //do something
15. });
16.

```

layer.load(icon, options) - 加载层

type:3的深度定制。**load**并不需要你传太多的参数，但如果你不喜欢默认的加载风格，你还有选择空间。**icon**支持传入0-2如果是0，无需传。另外特别注意一点：**load默认是不会自动关闭的**，因为你一般会在**ajax**回调体中关闭它。

code [layui.code](#)

```

1. //eg1
2. var index = layer.load();
3. //eg2
4. var index = layer.load(1); //换了种风格
5. //eg3
6. var index = layer.load(2, {time: 10*1000}); //又换了种风格，并且设定最长等待10秒
7. //关闭
8. layer.close(index);
9.

```

layer.tips(content, follow, options) - tips层

type:4的深度定制。也是我本人比较喜欢的一个层类型，因为它拥有和**msg**一样的低调和自觉，而且会**智能定位**，即灵活地判断它应该出现在哪边。默认是在元素右边弹出

code [layui.code](#)

```

1. //eg1
2. layer.tips('只想提示地精准些', '#id');
3. //eg 2
4. $('#id').on('click', function(){
5.     var that = this;
6.     layer.tips('只想提示地精准些', that); //在元素的事件回调体中，follow直接赋予this即可
7. });
8. //eg 3
9. layer.tips('在上面', '#id', {
10.     tips: 1
11. });
12.

```

上面主要是一些弹层的调用方式，而下面介绍的是一些辅助性的方法

layer.close(index) - 关闭特定层

关于它似乎没有太多介绍的必要，唯一让你疑惑的，可能就是这个**index**了吧。事实上它非常容易得到。

code [layui.code](#)

```

1. //当你想关闭当前页的某个层时
2. var index = layer.open();
3. var index = layer.alert();
4. var index = layer.load();
5. var index = layer.tips();
6. //正如你看到的，每一种弹层调用方式，都会返回一个index
7. layer.close(index); //此时你只需要把获得的index，轻轻地赋予layer.close即可
8.
9. //如果你想关闭最新弹出的层，直接获取layer.index即可
10. layer.close(layer.index); //它获取的始终是最新弹出的某个层，值是由layer内部动态递增计算的
11.
12. //当你在iframe页面关闭自身时
13. var index = parent.layer.getFrameIndex(window.name); //先得到当前iframe层的索引
14. parent.layer.close(index); //再执行关闭
15.

```

layer.closeAll(type) - 关闭所有层

如果你很懒，你**不想去获取index**你只想关闭。那么**closeAll**真的可以帮上你。如果你不指向层类型的话，它会销毁掉当前页所有的**layer**层。当然，如果你只想关闭某个类型的层，那么你可以

code [layui.code](#)

```

1. layer.closeAll(); //疯狂模式，关闭所有层
2. layer.closeAll('dialog'); //关闭信息框
3. layer.closeAll('page'); //关闭所有页面层

```



```
4. layer.closeAll('iframe'); //关闭所有的iframe层
5. layer.closeAll('loading'); //关闭加载层
6. layer.closeAll('tips'); //关闭所有的tips层
7.
```

layer.style(index, cssStyle) - 重新定义层的样式

该方法对loading层和tips层无效。参数index为层的索引，cssStyle允许你传入任意的css属性

code layui.code

```
1. //重新给指定层设定width、top等
2. layer.style(index, {
3.   width: '1000px',
4.   top: '10px'
5. });
6.
```

layer.title(title, index) - 改变层的标题

使用方式: `layer.title('标题变了', index)`

layer.getChildFrame(selector, index) - 获取iframe页的DOM

当你试图在当前页获取iframe页的DOM元素时，你可以用此方法。selector即iframe页的选择器

code layui.code

```
1. layer.open({
2.   type: 2,
3.   content: 'test/iframe.html',
4.   success: function(layero, index){
5.     var body = layer.getChildFrame('body', index);
6.     var iframeWin = window[layero.find('iframe')[0]['name']]; //得到iframe页的窗口对象，执行iframe页的方法：iframeWin.method();
7.     console.log(body.html()) //得到iframe页的body内容
8.     body.find('input').val('Hi, 我是从父页来的')
9.   }
10. });
11.
```

layer.getFrameIndex(windowName) - 获取特定iframe层的索引

此方法一般用于在iframe页关闭自身时用到。

code layui.code

```
1. //假设这是iframe页
2. var index = parent.layer.getFrameIndex(window.name); //先得到当前iframe层的索引
3. parent.layer.close(index); //再执行关闭
4.
```

layer.iframeAuto(index) - 指定iframe层自适应

调用该方法时，iframe层的高度会重新进行适应

layer.iframeSrc(index, url) - //重置特定iframe url

似乎不怎么常用的样子。使用方式: `layer.iframeSrc(index, 'http://sentsin.com')`

layer.setTop(layero) - 置顶当前窗口

非常强大的一个方法，虽然一般很少用。但是当你的页面有很多很多layer窗口，你需要像Window窗体那样，点击某个窗口，该窗体就置顶在上面，那么setTop可以来轻松实现。它采用巧妙的逻辑，以使这种置顶的性能达到最优

code layui.code

```
1. //通过这种方式弹出的层，每当它被选择，就会置顶。
2. layer.open({
3.   type: 2,
4.   shade: false,
5.   area: '500px',
6.   maxmin: true,
7.   content: 'http://www.layui.com',
8.   zIndex: layer.zIndex, //重点1
9.   success: function(layero){
10.    layer.setTop(layero); //重点2
11.  }
12. });
13.
```

layer.full()、layer.min()、layer.restore() - 手工执行最大小化

(这三个酱油又一次被并列 ==。) 一般用于在自定义元素上触发最大化、最小化和全屏。

请注意, 从2.3开始, 无需通过layer.config来加载拓展模块。如果您是之前版本, 则需通过下述方式来加载

code [layui.code](#)

```
1. layer.config({
2.   extend: 'extend/layer.ext.js'
3. });
4.
```

layer.prompt(options, yes) - 输入层

prompt的参数也是向前补齐的。options不仅可支持传入基础参数, 还可以传入prompt专用的属性。当然, 也可以不传。yes携带value 表单值index 索引elem 表单元素

code [layui.code](#)

```
1. //prompt层新定制的成员如下
2. {
3.   formType: 1, //输入框类型, 支持0(文本) 默认1(密码) 2(多行文本)
4.   value: '', //初始时的值, 默认空字符
5.   maxlength: 140, //可输入文本的最大长度, 默认500
6. }
7.
8. //例子1
9. layer.prompt(function(value, index, elem){
10.   alert(value); //得到value
11.   layer.close(index);
12. });
13.
14. //例子2
15. layer.prompt({
16.   formType: 2,
17.   value: '初始值',
18.   title: '请输入值',
19.   area: ['800px', '350px'] //自定义文本域宽高
20. }, function(value, index, elem){
21.   alert(value); //得到value
22.   layer.close(index);
23. });
24.
```

layer.tab(options) - tab层

tab层只单独定制了一个成员, 即tab: [], 这个好像没有什么可介绍的, 简单粗暴看例子

code [layui.code](#)

```
1. layer.tab({
2.   area: ['600px', '300px'],
3.   tab: [{
4.     title: 'TAB1',
5.     content: '内容1'
6.   }, {
7.     title: 'TAB2',
8.     content: '内容2'
9.   }, {
10.    title: 'TAB3',
11.    content: '内容3'
12.  }]
13. });
14.
```

layer.photos(options) - 相册层

相册层, 也可以称之为图片查看器。它的出场动画从layer内置的动画类型中随机展现。photos支持传入json和直接读取页面图片两种方式。如果是json传入, 如下:

code [layui.code](#)

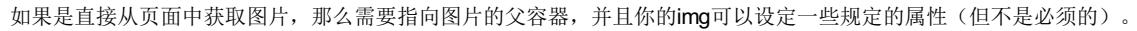
```
1. $.getJSON('/jquery/layer/test/photos.json', function(json){
2.   layer.photos({
3.     photos: json
4.     ,anim: 5 //0-6的选择, 指定弹出图片动画类型, 默认随机(请注意, 3.0之前的版本用shift参数)
5.   });
6. });
7. //而返回的json需严格按照如下格式:
8. code layui.code
9. {
10.   "title": "", //相册标题
11.   "id": 123, //相册id
```

```

4.  "start": 0, //初始显示的图片序号，默认0
5.  "data": [   //相册包含的图片，数组格式
6.    {
7.      "alt": "图片名",
8.      "pid": 666, //图片id
9.      "src": "", //原图地址
10.     "thumb": "" //缩略图地址
11.   }
12. ]
13. }

```

9.

如果是直接从页面中获取图片，那么需要指向图片的父容器，并且你的可以设定一些规定的属性（但不是必须的）。

code_layui.code

```

1.  //HTML示例
2.  <div id="layer-photos-demo" class="layer-photos-demo">
3.    
4.    
5.  </div>
6.
7.  <script>
8.  //调用示例
9.  layer.photos({
10.    photos: '#layer-photos-demo'
11.    ,anim: 5 //0-6的选择，指定弹出图片动画类型，默认随机（请注意，3.0之前的版本用shift参数）
12.  });
13. </script>
14.

```

星空如此深邃

看看一个实例呗：

第二种方式的图片查看器显然更加简单，因为无需像第一种那样返回规定的json，但是他们还是有各自的应用场景的，你可以按照你的需求进行选择。另外，**photos**还有个**tab**回调，切换图片时触发。

code_layui.code

```

1.  layer.photos({
2.    photos: json/选择器,
3.    tab: function(pic, layero){
4.      console.log(pic) //当前图片的一些信息
5.    }
6.  });
7.

```

结语

合理地设定基础参数，合理地选择内置方法，合理的心态，合理地阅读，只要一切都在合理的前提下，你才会感知到layer许许多多令人愉悦的地方，她真的是否如你所愿，取决于你对她了解的深远。愿layer能给你的web开发带来一段美妙的旅程。别忘了在线调试。