

Layui 2.x 使用教程

layui (谐音：类UI) 是一款采用自身模块规范编写的前端 UI 框架，遵循原生 HTML/CSS/JS 的书写与组织形式，门槛极低，拿来即用。其外在极简，却又不失饱满的内在，体积轻盈，组件丰盈，从核心代码到 API 的每...



下载手机APP
畅享精彩阅读

目 录

致谢

基础说明

开始使用 Getting Started

底层方法 基础支撑

页面元素 规范 / 公共类 / 属性

模块规范 使用 / 扩展

常见问题 FAQ

更新日志

页面元素

布局 栅格 / 后台布局

颜色 主题色设计感 / 内置背景色

图标 字体图标

动画 内置 CSS3 动画

按钮 button 组

表单 form 元素集合

导航 菜单 / 面包屑

选项卡 Tabs 切换

进度条 progress

面板 折叠 / 手风琴

表格 静态 table

徽章 小圆点 / 小边框

时间线 timeline

辅助 引用 / 字段集 / 横线等

内置模块

弹出层 layer

日期与时间选择 laydate

即时通讯 layim

分页 laypage

模板引擎 laytpl

数据表格 table

表单 form

文件上传 upload

穿梭框 transfer

树形组件 tree

颜色选择器 colorpicker

常用元素操作 [element](#)

滑块 [slider](#)

评分 [rate](#)

轮播 [carousel](#)

流加载 [flow](#)

工具集 [util](#)

代码修饰器 [code](#)

致谢

当前文档《Layui 2.x 使用教程》由 进击的皇虫 使用 书栈网 (BookStack.CN) 进行构建，生成于 2020-05-05。

书栈网仅提供文档编写、整理、归类等功能，以及对文档内容的生成和导出工具。

文档内容由网友们编写和整理，书栈网难以确认文档内容知识点是否错漏。如果您在阅读文档获取知识的时候，发现文档内容有不恰当的地方，请向我们反馈，让我们共同携手，将知识准确、高效且有效地传递给每一个人。

同时，如果您在日常工作、生活和学习中遇到有价值有营养的知识文档，欢迎分享到书栈网，为知识的传承献上您的一份力量！

如果当前文档生成时间太久，请到书栈网获取最新的文档，以跟上知识更新换代的步伐。

内容来源：Layui <https://www.layui.com/doc/>

文档地址：<http://www.bookstack.cn/books/Layui-2.x>

书栈官网：<https://www.bookstack.cn>

书栈开源：<https://github.com/TruthHun>

分享，让知识传承更久远！感谢知识的创造者，感谢知识的分享者，也感谢每一位阅读到此处的读者，因为我们都将成为知识的传承者。

- [开始使用](#) [Getting Started](#)
- [底层方法](#) [基础支撑](#)
- [页面元素](#) [规范](#) / [公共类](#) / [属性](#)
- [模块规范](#) [使用](#) / [扩展](#)
- [常见问题](#) [FAQ](#)
- [更新日志](#)

开始使用 - 入门指南

layui (谐音: 类UI) 是一款采用自身模块规范编写的前端 UI 框架, 遵循原生 HTML/CSS/JS 的书写与组织形式, 门槛极低, 拿来即用。其外在极简, 却又不失饱满的内在, 体积轻盈, 组件丰盈, 从核心代码到 API 的每一处细节都经过精心雕琢, 非常适合界面的快速开发。layui 首个版本发布于 2016 年金秋, 她区别于那些基于 MVVM 底层的 UI 框架, 却并非逆道而行, 而是信奉返璞归真之道。准确地说, 她更多是为服务端程序员量身定做, 你无需涉足各种前端工具的复杂配置, 只需面对浏览器本身, 让一切你所需要的元素与交互, 从这里信手拈来。



兼容性和面向场景

layui 兼容人类正在使用的全部浏览器 (IE6/7 除外), 可作为 PC 端后台系统与前台界面的速成开发方案。

获得 layui

1. 官网首页下载

你可以在我们的 [官网首页](#) 下载到 layui 的最新版, 它经过了自动化构建, 更适合用于生产环境。目录结构如下:

```
1.  |—css //css目录
2.  |   |—modules //模块css目录 (一般如果模块相对较大, 我们会单独提取, 比如下面三个: )
3.  |   |   |—laydate
4.  |   |   |—layer
5.  |   |   |—layim
6.  |   |—layui.css //核心样式文件
7.  |—font //字体图标目录
8.  |—images //图片资源目录 (目前只有layim和编辑器用到的GIF表情)
9.  |—lay //模块核心目录
10. |   |—modules //各模块组件
11. |—layui.js //基础核心库
12. |—layui.all.js //包含layui.js和所有模块的合并文件
13.
```

2. Git 仓库下载

你也可以通过 [GitHub](#) 或 [码云](#) 得到 layui 的完整开发包, 以便于你进行二次开发, 或者 Fork layui 为我们贡献方案

[GitHub](#) [码云](#)

3. npm 安装

```
1.
2. npm i layui-src
```

3.

一般用于 WebPack 管理

快速上手

获得 layui 后，将其完整地部署到你的项目目录（或静态资源服务器），你只需要引入下述两个文件：

```
1. ./layui/css/layui.css
2. ./layui/layui.js //提示：如果是采用非模块化方式（最下面有讲解），此处可换成：./layui/layui.all.js
3.
```

没错，不用去管其它任何文件。因为他们（比如各模块）都是在最终使用的时候才会自动加载。这是一个基本的入门页面：

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4.   <meta charset="utf-8">
5.   <meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1">
6.   <title>开始使用layui</title>
7.   <link rel="stylesheet" href="../layui/css/layui.css">
8. </head>
9. <body>
10.
11. <!-- 你的HTML代码 -->
12.
13. <script src="../layui/layui.js"></script>
14. <script>
15.   //一般直接写在一个js文件中
16.   layui.use(['layer', 'form'], function(){
17.     var layer = layui.layer
18.     ,form = layui.form;
19.
20.     layer.msg('Hello World');
21.   });
22. </script>
23. </body>
24. </html>
25.
```

如果你想采用非模块化方式（即所有模块一次性加载，尽管我们并不推荐你这么 做），你也可以按照下面的方式使用：

```
1. <!DOCTYPE html>
2. <html>
```

```

3. <head>
4.   <meta charset="utf-8">
5.   <meta name="viewport" content="width=device-width, initial-scale=1, maximum-
  scale=1">
6.   <title>非模块化方式使用layui</title>
7.   <link rel="stylesheet" href="../../layui/css/layui.css">
8. </head>
9. <body>
10.
11. <!-- 你的HTML代码 -->
12.
13. <script src="../../layui/layui.all.js"></script>
14. <script>
15.   //由于模块都一次性加载，因此不用执行 layui.use() 来加载对应模块，直接使用即可：
16.   ;!function(){
17.     var layer = layui.layer
18.     ,form = layui.form;
19.
20.     layer.msg('Hello World');
21.   }();
22. </script>
23. </body>
24. </html>
25.

```

经典，因返璞归真

layui 定义为「经典模块化」，并非是自吹她自身有多优秀，而是有意避开当下 JS 社区的主流方案，试图以最简单的方式去诠释高效！她的所谓经典，是在于对返璞归真的执念，她以当前浏览器普通认可的方式去组织模块！我们认为，这恰是符合当下国内绝大多数程序员从旧时代过渡到未来新标准的最佳指引。所以 layui 本身也并不是完全遵循于AMD时代，准确地说，她试图建立自己的模式，所以你会看到：

```

1. //layui模块的定义
2. layui.define([mods], function(exports){
3.
4.   //.....
5.
6.   exports('mod', api);
7. });
8.
9. //layui模块的使用
10. layui.use(['mod1', 'mod2'], function(args){
11.   var mod = layui.mod1;
12.
13.   //.....
14.
15. });
16.

```


没错，她具备早前 AMD 的影子，又并非受限于 CommonJS 的那些条条框框，layui 认为这种轻量的组织方式，比 WebPack 更符合绝大多数场景。所以她坚持采用经典模块化，也正是能让人避开工具的复杂配置，回归简单，安静高效地编织原生态的 HTML / CSS / JavaScript。

但是 layui 又并非是 RequireJS 那样的模块加载器，而是一款 UI 解决方案，与 Bootstrap 的不同又在于：layui 糅合了自身对经典模块化的理解。

模块化的用法（一般用于开发环境）

我们推荐你遵循 layui 的模块规范建立一个入口文件，并通过 `layui.use()` 方式来加载该入口文件，如下所示：

```
1. <script>
2. layui.config({
3.   base: '/res/js/modules/' //你存放新模块的目录，注意，不是layui的模块目录
4. }).use('index'); //加载入口
5. </script>
6.
```

上述的 `index` 即为你 `/res/js/modules/` 目录下的 `index.js`，它的内容应该如下：

```
1. /**
2.   项目JS主入口
3.   以依赖layui的layer和form模块为例
4. **/
5. layui.define(['layer', 'form'], function(exports){
6.   var layer = layui.layer
7.   , form = layui.form;
8.
9.   layer.msg('Hello World');
10.
11.   exports('index', {}); //注意，这里是模块输出的核心，模块名必须和use时的模块名一致
12. });
13.
```

全模块用法（一般用于线上环境）

事实上 layui 的「模块化加载」十分适用于开发环境，它方便团队开发和代码调试。但对于「线上环境」，我们更推荐您采用「全模块加载」，即直接引入

`layui.all.js`，它包含了 layui 所有的内置模块，且无需再通过 `layui.use()` 方法加载模块，直接调用即可。如：

```
1. <script src="../../layui/layui.all.js"></script>
```

```

2. <script>
3. ;!function(){
4.   //无需再执行layui.use()方法加载模块，直接使用即可
5.   var form = layui.form
6.   ,layer = layui.layer;
7.
8.   //...
9. }();
10. </script>
11.

```

除了 layui 内置的全模块加载，layui 的扩展模块同样可以合并为一个文件来加载。我们假设你的项目存放了很多个扩展模块（可以简单理解为一个 js 文件就是一个模块），那么你只需要根据其依赖关系将其合并，即可一次性加载所有扩展模块，如：

```

1. //mod1.js
2. layui.define('layer', function(exports){
3.   //...
4.   exports(mod1, {});
5. });
6.
7. //mod2.js, 假设依赖 mod1 和 form
8. layui.define(['mod1', 'form'], function(exports){
9.   //...
10.   exports(mod2, {});
11. });
12.
13. //mod3.js
14. //...
15.
16. //main.js 主入口模块
17. layui.define('mod2', function(exports){
18.   //...
19.   exports('main', {});
20. });
21.

```

现在我们可以借助 Gulp 将上述的 `mod1、mod2、mod3、main` 等扩展模块构建合并到一个模块文件：`main`，此时你只需要加载它即可：

```

1. <script src="../../layui/layui.all.js"></script>
2. <script>
3.   layui.config({
4.     base: '../js/' //你的扩展模块所在目录
5.   }).use('main'); //这里的 main 模块包含了 mod1、mod2、mod3 等等扩展模块
6.   }();
7. </script>
8.

```

可以看到，通过「全模块」的用法，我们最多只需要加载两个 JS 文件，这将大幅度减少静态资源的请求。

通过上面的阅读，也许你已经大致了解如何使用 layui 了，但真正用于项目远不止如此，你需要继续阅读后面的文档，尤其是「基础说明」。

那么，从现在开始，尽情地拥抱 layui 吧！但愿她能成为你长远的开发伴侣，化作你方寸屏幕前的亿万字节！

layui - 用心与你沟通

底层方法

本篇主要介绍核心基础库 `layui.js` 所发挥的作用，其中过滤了大部分在外部可能不是太常用的 API，侧重罗列了最常用的框架支撑。

全局配置

方法：`layui.config(options)`

你可以在使用模块之前，全局化配置一些参数，尽管大部分时候它不是必须的。所以我们目前提供的全局配置项非常少，这也是为了减少一些不必要的工作，尽可能让使用变得更简单。目前支持的全局配置项如下：

```
1. layui.config({
2.   dir: '/res/layui/' //layui.js 所在路径（注意，如果是 script 单独引入 layui.js，无需
   设定该参数。），一般情况下可以无视
3.   ,version: false //一般用于更新模块缓存，默认不开启。设为 true 即让浏览器不缓存。也可以
   设为一个固定的值，如：201610
4.   ,debug: false //用于开启调试模式，默认 false，如果设为 true，则JS模块的节点会保留在页
   面
5.   ,base: '' //设定扩展的 layui 模块的所在目录，一般用于外部模块扩展
6. });
7.
```

定义模块

方法：`layui.define([mods], callback)`

通过该方法可定义一个 `layui` 模块。参数 `mods` 是可选的，用于声明该模块所依赖的模块。`callback` 即为模块加载完毕的回调函数，它返回一个 `exports` 参数，用于输出该模块的接口。

```
1. layui.define(function(exports){
2.   //do something
3.
4.   exports('demo', function(){
5.     alert('Hello World!');
6.   });
7. });
8.
```

跟 `RequireJS` 最大不同的地方在于接口输出，`exports` 是一个函数，它接受两个参数，第一个参数为模块名，第二个参数为模块接口，当你声明了上述的一个模块后，你就可以在外部使用了，`demo` 就会注册到 `layui` 对象下，即可通过 `layui.demo()` 去执行该模块的接口。

你也可以在定义一个模块的时候，声明该模块所需的依赖，如：

```
1. layui.define(['layer', 'laypage'], function(exports){
2.     //do something
3.
4.     exports('demo', function(){
5.         alert('Hello World!');
6.     });
7. });
8.
```

上述的 `['layer', 'laypage']` 即为本模块所依赖的模块，它并非只能是一个数组，你也可以直接传一个字符型的模块名，但是这样只能依赖一个模块。

加载所需模块

方法：`layui.use([mods], callback)`

layui 的内置模块并非默认就加载的，他必须在你执行该方法后才会加载。它的参数跟上述的 `define` 方法完全一样。

另外请注意，`mods` 里面必须是一个合法的模块名，不能包含目录。如果需要加载目录，建议采用 `extend` 建立别名（详见模块规范）

```
1. layui.use(['laypage', 'layedit'], function(){
2.     var laypage = layui.laypage
3.     ,layedit = layui.layedit;
4.
5.     //do something
6. });
7.
```

该方法的函数其实返回了所加载的模块接口，所以你其实也可以不通过 `layui` 对象赋值获得接口（这一点跟 `Sea.js` 很像哈）：

```
1. layui.use(['laypage', 'layedit'], function(laypage, layedit){
2.
3.     //使用分页
4.     laypage();
5.
6.     //建立编辑器
7.     layedit.build();
8. });
9.
```

动态加载 CSS

方法: `layui.link(href)`

href 即为 css 路径。注意：该方法并非是你使用 layui 所必须的，它一般只是用于动态加载你的外部 CSS 文件。

本地存储

本地存储是对 localStorage 和 sessionStorage 的友好封装，可更方便地管理本地数据。

- localStorage 持久化存储: `layui.data(table, settings)`，数据会永久存在，除非物理删除。
- sessionStorage 会话性存储: `layui.sessionData(table, settings)`，页面关闭后即失效。
注: layui 2.2.5 新增

上述两个方法的使用方式是完全一样的。其中参数 `table` 为表名，`settings` 是一个对象，用于设置 key、value。下面以 `layui.data` 方法为例：

```
1. //【增】：向 test 表插入一个 nickname 字段，如果该表不存在，则自动建立。
2. layui.data('test', {
3.   key: 'nickname'
4.   ,value: '贤心'
5. });
6.
7. //【删】：删除 test 表的 nickname 字段
8. layui.data('test', {
9.   key: 'nickname'
10.  ,remove: true
11. });
12. layui.data('test', null); //删除test表
13.
14. //【改】：同【增】，会覆盖已经存储的数据
15.
16. //【查】：向 test 表读取全部的数据
17. var localTest = layui.data('test');
18. console.log(localTest.nickname); //获得“贤心”
19.
```

获取设备信息

方法: `layui.device(key)`，参数key是可选的

由于 layui 的一些功能进行了兼容性处理和响应式支持，因此该方法同样发挥了至关重要的作用。尤其是在 layui mobile 模块中的作用可谓举足轻重。该方法返回了丰富的设备信息：

```
1. var device = layui.device();
2.
3. //device即可根据不同的设备返回下述不同的信息
```

```

4.  {
5.    os: "windows" //底层操作系统, windows、linux、mac等
6.    ,ie: false //ie6-11的版本, 如果不是ie浏览器, 则为false
7.    ,weixin: false //是否微信环境
8.    ,android: false //是否安卓系统
9.    ,ios: false //是否ios系统
10. }

```

有时你的 App 可能会对 userAgent 插入一段特定的标识, 譬如:

```

Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/53.0.2785.143 myapp/1.8.6 Safari/537.36

```

你要验证当前的 WebView 是否在你的 App 环境, 即可通过上述的myapp (即为 Native 给 Webview 插入的标识, 可以随意定义) 来判断。

```

1. var device = layui.device('myapp');
2. if(device.myapp){
3.   alert('在我的App环境');
4. }
5.

```

其它

除上述介绍的方法之外, layui.js 内部还提供了许多底层引擎, 他们同样是整个 layui 体系的有力支撑, 在日常应用中也许会用到:

方法/属性	描述
layui.cache	静态属性。获得一些配置及临时的缓存信息
layui.extend(options)	拓展一个模块别名, 如: layui.extend({test: '/res/js/test'})
layui.each(obj, fn)	对象 (Array、Object、DOM 对象等) 遍历, 可用于取代for语句
layui.getStyle(node, name)	获得一个原始 DOM 节点的 style 属性值, 如: layui.getStyle(document.body, 'font-size')
layui.img(url, callback, error)	图片预加载
layui.sort(obj, key, desc)	将数组中的对象按某个成员重新对该数组排序, 如: layui.sort([{a: 3}, {a: 1}, {a: 5}], 'a')
layui.router()	获得 location.hash 路由结构, 一般在单页面应用中发挥作用。
layui.url(href)	用于将一段 URL 链接中的 pathname、search、hash 属性值进行对象化处理 参数: href 可选。若不传, 则自动读取当前页面的 url (即: location.href) 示例: <code>var url = layui.url();</code> 注意: 系 layui 2.5.6 新增
layui.hint()	向控制台打印一些异常信息, 目前只返回了 error 方法: layui.hint().error('出错啦')

<code>layui.stope(e)</code>	阻止事件冒泡
<code>layui.onevent(modName, events, callback)</code>	增加自定义模块事件。有兴趣的同学可以阅读 <code>layui.js</code> 源码以及 <code>form</code> 模块
<code>layui.event(modName, events, params)</code>	执行自定义模块事件，搭配 <code>onevent</code> 使用
<code>layui.factory(modName)</code>	用于获取模块对应的 <code>define</code> 回调函数

第三方支撑

layui 部分模块依赖 *jQuery* (比如 `layer`)，但是你并不用去额外加载 *jQuery*。layui 已经将 *jQuery* 最稳定的一个版本改为 layui 的内部模块，当你去使用 `layer` 之类的模块时，它会首先判断你的页面是否已经引入了 *jQuery*，如果没有，则加载内部的 *jQuery* 模块，如果有，则不会加载。

另外，我们的图标取材于阿里巴巴矢量图标库 (*iconfont*)，构建工具采用 *Gulp*。除此之外，不依赖于任何第三方工具。

layui - 用心与你沟通

页面元素规范与说明

layui 提倡返璞归真，遵循于原生态的元素书写规则，所以通常而言，你仍然是在写基本的 HTML 和 CSS 代码，不同的是，在 HTML 结构上及 CSS 定义上需要小小遵循一定的规范。

CSS内置公共基础类

类名 (class)	说明
布局 / 容器	
layui-main	用于设置一个宽度为 1140px 的水平居中块（无响应式）
layui-inline	用于将标签设为内联块状元素
layui-box	用于排除一些UI框架（如Bootstrap）强制将全部元素设为box-sizing: border-box所引发的尺寸偏差
layui-clear	用于消除浮动（一般不怎么常用，因为layui几乎没用到浮动）
layui-btn-container	用于定义按钮的父容器。（layui 2.2.5 新增）
layui-btn-fluid	用于定义流体按钮。即宽度最大化适应。（layui 2.2.5 新增）
辅助	
layui-icon	用于图标
layui-elip	用于单行文本溢出省略
layui-unselect	用于屏蔽选中
layui-disabled	用于设置元素不可点击状态
layui-circle	用于设置元素为圆形
layui-show	用于显示块状元素
layui-hide	用于隐藏元素
文本	
layui-text	定义一段文本区域（如文章），该区域内的特殊标签（如a、li、em等）将会进行相应处理
layui-word-aux	灰色标注性文字，左右会有间隔
背景色	
layui-bg-red	用于设置元素赤色背景
layui-bg-orange	用于设置元素橙色背景
layui-bg-green	用于设置元素墨绿色背景（主色调）
layui-bg-cyan	用于设置元素藏青色背景
layui-bg-blue	用于设置元素蓝色背景
layui-bg-black	用于设置元素经典黑色背景
layui-bg-gray	用于设置元素经典灰色背景

其它的类一般都是某个元素或模块所特有，因此不作为我们所定义的公共类。

CSS命名规范

class命名前缀：*layui*，连接符：*-*，如：*class="layui-form"*

命名格式一般分为两种：一：*layui*-模块名-状态或类型，二：*layui*-状态或类型。因为有些类并非是某个模块所特有，他们通常会是一些公共类。如：一（定义按钮的原始风格）：*class="layui-btn layui-btn-primary"*、二（定义内联块状元素）：*class="layui-inline"*

大致记住这些简单的规则，会让你在填充HTML的时候显得更加得心应手。另外，如果你是开发Layui拓展（模块），你最好也要遵循于类似的规则，并且请勿占用Layui已经命名好的类，假设你是在帮Layui开发一个markdown编辑器，你的css书写规则应该如下：

```
1. .layui-markdown{border: 1px solid #e2e2e2;}
2. .layui-markdown-tools{}
3. .layui-markdown-text{}
4.
```

HTML规范：结构

Layui在解析HTML元素时，必须充分确保其结构是被支持的。以Tab选项卡为例：

```
1. <div class="layui-tab">
2.   <ul class="layui-tab-title">
3.     <li class="layui-this">标题一</li>
4.     <li>标题二</li>
5.     <li>标题三</li>
6.   </ul>
7.   <div class="layui-tab-content">
8.     <div class="layui-tab-item layui-show">内容1</div>
9.     <div class="layui-tab-item">内容2</div>
10.    <div class="layui-tab-item">内容3</div>
11.  </div>
12. </div>
13.
```

你如果改变了结构，极有可能会导致Tab功能失效。所以在嵌套HTML的时候，你应该细读各个元素模块的相关文档（如果你不是拿来主义）

HTML规范：常用公共属性

很多时候，元素的基本交互行为，都是由模块自动开启。但不同的区域可能需要触

发不同的动作，这就需要你设定我们所支持的自定义属性来作为区分。如下面的 *lay-submit*、*lay-filter*即为公共属性（即以 *lay-* 作为前缀的自定义属性）：

```
1. <button class="layui-btn" lay-submit lay-filter="login">登入</button>
2.
```

目前我们的公共属性如下所示（即普遍运用于所有元素上的属性）

属性	描述
lay-skin=""	定义相同元素的不同风格，如checkbox的开关风格
lay-filter=""	事件过滤器。你可能会在很多地方看到他，他一般是用于监听特定的自定义事件。你可以把它看作是一个ID选择器
lay-submit	定义一个触发表单提交的button，不用填写值

额，好像有点少的样子（反正你也基本不会看文档。。（づ￣▽￣）づ）。其它的自定义属性基本都在各自模块的文档中有所介绍。

结语

其实很多时候并不想陈列条条框框（除了一些特定需要的），所以你会发现本篇的篇幅较短。（哈哈哈哈哈，其实是写文档写得想吐了）

模块规范

layui 的模块是基于 `layui.js` 内部实现的异步模块加载方式，它并不遵循于AMD（没有为什么，毕竟任性呀！），而是自己定义了一套更轻量的模块规范。并且这种方式在经过了大量的实践后，成为 layui 最核心的模块加载引擎。

预先加载

开门见山，还是直接说使用比较妥当。Layui的模块加载采用核心的 `layui.use(mods, callback)` 方法，当你的JS 需要用到Layui模块的时候，我们更推荐你采用预先加载，因为这样可以避免到处写`layui.use`的麻烦。你应该在最外层如此定义：

```
1.  /*
2.    Demo1.js
3.    使用Layui的form和upload模块
4.  */
5.  layui.use(['form', 'upload'], function(){ //如果只加载一个模块，可以不填数组。
6.    如：layui.use('form')
7.    var form = layui.form //获取form模块
8.    ,upload = layui.upload; //获取upload模块
9.
10.   //监听提交按钮
11.   form.on('submit(test)', function(data){
12.     console.log(data);
13.   });
14.
15.   //实例化一个上传控件
16.   upload({
17.     url: '上传接口url'
18.     ,success: function(data){
19.       console.log(data);
20.     }
21.   });
22.  });
```

按需加载（不推荐）

如果你有强迫症，你对网站的性能有极致的要求，你并不想预先加载所需要的模块，而是在触发一个动作的时候，才去加载模块，那么，这是允许的。你不用在你的JS最外层去包裹一个大大的 `layui.use`，你只需要：

```
1.  /*
2.    Demo2.js
3.    按需加载一个Layui模块
```

```

4.  */
5.
6.  //.....
7.  //你的各种JS代码什么的
8.  //.....
9.
10. //下面是在一个事件回调里去加载一个Layui模块
11. button.addEventListener('click', function(){
12.     layui.use('laytpl', function(laytpl){ //温馨提示：多次调用use并不会重复加载
        laytpl.js, Layui内部有做模块cache处理。
13.         var html = laytpl('').render({});
14.         console.log(html);
15.     });
16. });
17.

```

注意：如果你的 JS 中需要大量用到模块，我们并不推荐你采用这种加载方式，因为这意味着你要写很多 `layui.use()`，代码可维护性不高。建议还是采用：预先加载。即一个JS文件中，写一个use即可。

模块命名空间

layui 的模块接口会绑定在 layui 对象下，内部由 `layui.define()` 方法来完成。每一个模块都会一个特有的名字，并且无法被占用。所以你无需担心模块的空间被污染，除非你主动 `delete layui.{模块名}`。调用模块可通过 `layui.use` 方法来实现，然后再通过 layui 对象获得模块接口。如：

```

1. layui.use(['layer', 'laypage', 'laydate'], function(){
2.     var layer = layui.layer //获得layer模块
3.     ,laypage = layui.laypage //获得laypage模块
4.     ,laydate = layui.laydate; //获得laydate模块
5.
6.     //使用模块
7. });
8.

```

我们推荐你将所有的业务代码都写在一个大的 use 回调中，而不是将模块接口暴露给全局，比如下面的方式我们是极不推荐的：

```

1. //强烈不推荐下面的做法
2. var laypage, laydate;
3. layui.use(['laypage', 'laydate'], function(){
4.     laypage = layui.laypage;
5.     laydate = layui.laydate;
6. });
7.

```

你之所以想使用上面的错误方式，是想其它地方使用不在执行一次 `layui.use`？

但这种理解本身是存在问题的。因为如果一旦你的业务代码是在模块加载完毕之前执行，你的全局对象将获取不到模块接口，因此这样用不仅不符合规范，还存在报错风险。建议在你的 js 文件中，在最外层写一个 `layui.use` 来加载所依赖的模块，并将业务代码写在回调中，见：[预先加载](#)。

事实上，如果你不想采用 `layui.use`，你可以引入 `layui.all.js` 来替代 `layui.js`，见：[全模块用法](#)

扩展一个 layui 模块

layui 官方提供的模块有时可能还无法满足你，或者你试图按照layer的模块规范来扩展一个模块。那么你有必要认识`layui.define()`方法，相信你在文档左侧的“底层方法”中已有所阅读。下面就让我们一起扩展一个Layui模块吧：

第一步：确认模块名，假设为：`mymod`，然后新建一个`mymod.js` 文件放入项目任意目录下（注意：不用放入layui目录）

第二步：编写`test.js` 如下：

```
1.  /**
2.   * 扩展一个test模块
3.   */
4.
5.  layui.define(function(exports){ //提示：模块也可以依赖其它模块，如：
6.    layui.define('layer', callback);
7.    var obj = {
8.      hello: function(str){
9.        alert('Hello ' + (str || 'mymod'));
10.     }
11.   };
12.   //输出test接口
13.   exports('mymod', obj);
14. });
15.
```

第三步：设定扩展模块所在的目录，然后就可以在别的JS文件中使用了

```
1.  //config的设置是全局的
2.  layui.config({
3.    base: '/res/js/' //假设这是你存放拓展模块的根目录
4.  }).extend({ //设定模块别名
5.    mymod: 'mymod' //如果 mymod.js 是在根目录，也可以不用设定别名
6.    , mod1: 'admin/mod1' //相对于上述 base 目录的子目录
7.  });
8.
9.  //你也可以忽略 base 设定的根目录，直接在 extend 指定路径（主要：该功能为 layui 2.2.0 新增）
10. layui.extend({
11.   mod2: '{/}http://cdn.xxx.com/lib/mod2' // {/}的意思即代表采用自有路径，即不跟随
```

```
    base 路径
12. })
13.
14. //使用拓展模块
15. layui.use(['mymod', 'mod1'], function(){
16.     var mymod = layui.mymod
17.     ,mod1 = layui.mod1
18.     ,mod2 = layui.mod2;
19.
20.     mymod.hello('World!'); //弹出 Hello World!
21. });
22.
```

大体上来说，layui 的模块定义很类似 Require.js 和 Sea.js，但跟他们又有着明显的不同，譬如在接口输出等地方。

结语

其实关于模块的核心，就是 layui.js 的两个底层方法：一个用于定义模块的 *layui.define()*，一个加载模块的 *layui.use()*。

layui - 用心与你沟通

常见问题

本篇将主要讲解使用过程中普遍遇到的“问题”，这些问题并非是 BUG，通常是需要我们自己去注意的一些点。（会结合用户反馈持续补充）

哪里有 layui 未压缩源代码？

之所以在下载包里没有提供未压缩的源代码，是为了方便直接用于生产环境。
layui 源代码可通过以下平台获取：

[GitHub](#) [码云](#)

应该如何加载模块？

事实上我们在模块规范已经有明确地说明，你可以采用预先加载和按需加载两种模式，但后者我们并不推荐（文档也解释原因了）。因此我们强烈推荐的方式是：你应该在你js文件的代码最外层，就把需要用到的模块 `layui.use`以 一下，如：

```
1.  /**
2.   你的js文件
3.  **/
4.
5.  //我们强烈推荐你在代码最外层把需要用到的模块先加载
6.  layui.use(['layer', 'form', 'element'], function(){
7.    var layer = layui.layer
8.    ,form = layui.form
9.    ,element = layui.element
10.
11.    //.....
12.    //你的代码都应该写在这里面
13.  });
14.
15.
```

layui 与 layuiAdmin 有什么区别？

- layui 是一款开源免费的前端 UI 框架，遵循 MIT 协议，可随意使用。
- [layuiAdmin](#) 是完全基于 layui 搭建的一款通用型后台管理模板系统，由 layui 官方出品，需购买授权使用。

为什么表单不显示？

当你使用表单时，layui 会对 select、checkbox、radio 等原始元素隐藏，从而进行美化修饰处理。但这需要依赖于 form 组件，所以你必须加载 *form*，并且执行一个实例。[#详见说明](#)

```
1. layui.use('form', function(){
2.   var form = layui.form; //只有执行了这一步，部分表单元素才会自动修饰成功
3.
4.   //.....
5.
6.   //但是，如果你的HTML是动态生成的，自动渲染就会失效
7.   //因此你需要在相应的地方，执行下述方法来进行渲染
8.   form.render();
9. });
10.
```

同理的还有 [element](#) 模块

遇到各种问题怎么办？

求助社区 抱团取暖

[layui 社区](#)

layui - 用心与你沟通

更新日志

•

2.5.x

2020 年了，很多人问 layui 有什么计划？答：努力寻求突破，并顺势召唤出 layui 的新生妹妹：***UI。
人生建议：千万不要硬着头皮做自己根本不擅长的工作，而是应该花更多的精力做自己所擅长的事情。

◦ v2.5.6 2020-01-15

- [优化] layui.use() 方法，以支持加载非内置模块的合并请求（如您在线上环境采用「非模块化加载」的方式，那么最多可以只加载两个文件，即：layui.all.js、main.js(你的扩展模块的合并文件)。这将大幅度减少文件请求) [#详见文档](#)
- [新增] layui.url() 底层方法，用于将 url 中的 pathname、search、hash 属性进行对象化获取 [#详见文档](#)
- [优化] 栅格的列间隔类 .layui-col-space，支持 1-30 区间所有双数间隔，并支持 1、5、15、25 单数间隔
- [优化] table 组件的合计行，若接口直接返回了合计行数据，则优先读取，否则由前端自动合计当前行数据 [#详见文档](#)
- [修复] upload 组件因上个版本的 progress（进度条）功能导致的部分情况无法跨域上传的问题
- [优化] upload 组件 progress 回调，在第二个参数中返回了当前触发的元素对象
- [修复] form 模块的 select 组件在 lay-disabled 和 lay-search 共用时出现可编辑问题
- [修复] flow.load() 多次执行时的重复加载的问题
- [修复] util 组件的 event 方法重复绑定事件的问题
- [新增] 28 个字体图标 [#详见文档](#)

◦ v2.5.5 2019-09-10

之前大家在《[2.5.x 问题集中收集](#)》中反馈的内容，我已如数看到。其中有些被采纳在本次版本中，有些被规划到了 v2.6、v3.0 中。我深知之前反复的断更对 layui 带来的是怎样无可挽回的损失，然而 layui 终于还是不得不重新审视它所处的位置，行进的「前端河」分出了两条支流：一条是滚滚洪流，没有人能够逃脱它的流速，我们必须要在拥抱中重生；一条是波澜不惊，那是我们最初的方向，返璞归真、自诩的「经典」和不灭的执念。顺应潮流而不背离初心，layui 注定要同时流向这两条支流，虽千万里之行，亦愿独闯。

- [优化] form 组件的 val() 方法，除了之前版本的赋值，目前还可支持取值 [#详见文档](#)

- [新增] tree 组件的数据源参数 field, 用于定义数据字段名
- [优化] tree 组件的节点删除前的提示功能
- [修复] tree 组件的数据源参数 checked 在父子节点同时设定时的异常问题
- [优化] table 组件的 defaultToolbar 参数, 可以扩展头部工具栏右侧图标按钮 [#详见文档](#)
- [优化] table 组件的局部代码, 减少重复的全局事件引发的卡顿问题
- [修复] table 组件的合计行未按照对应列显示自定义模板的问题
- [修复] table 组件导出数据时未包含合计行的问题
- [修复] transfer 组件的右侧面板在使用搜索转移数据后, 无法再获取右侧数据的问题
- [新增] upload 组件的 progress 回调, 用于监听文件的上传进度 [#详见文档](#)

◦ v2.5.4 2019-06-06

- [修复] table 组件 reload 后的各种可能发生的异常问题
- [新增] tree 组件的 id 参数, 用于定义实例的唯一索引
- [新增] tree 组件的 text 参数, 用于定义一些默认文本
- [新增] tree 组件的数据源参数: spread、checked, 以定义节点初始的: 选中状态、展开状态
- [剔除] tree 组件的 spread、checked 参数, 采用数据源中的 spread、checked 属性替代
- [剔除] tree 组件的 key 参数, 因为多此一举
- [优化] tree 组件的 getChecked 方法, 让其返回选中的数据, 而非节点 DOM
- [优化] tree 组件中数据源 label 参数, 改名为 title
- [优化] tree 组件的 defaultNodeName、emptyText 参数, 将其移至到新增的 text 参数中
- [修复] transfer、tree 组件在 layui-form 中的样式异常问题

◦ v2.5.2 / v2.5.3 2019-06-04

- [全面重写] transfer 组件, 加强功能, 并修复了已知的所有问题 [#详见文档](#)
- [回滚] table 组件的 reload 方法在 2.5.0 的机制, 因大量场景测试存在不可控异常, 基于稳定考虑, 特此恢复 2.5.0 之前机制
- [修复] table 组件已知的若干紧急小问题
- [修复] upload 组件 reload 重载时, 如果传入 elem 出现报错的问题

◦ v2.5.1 2019-06-03

- [修复] table 组件，当不开启分页且出现滚动条，底部出现无边框的问题
- [修复] table 组件，当 reload 仅重载 data 时的若干小问题
- [调整] tree 组件，如果子节点有任意一条选中，则父节点为选中状态
- [新增] tree 组件的 defaultNodeName 属性，用于定义节点的默认名称
- [优化] tree 组件的 renderContent 属性，改名为 edit，可自由配置“增改删”按钮的显示状态
- [优化] tree 组件的 expandClick 属性，改名为 onlyIconControl，用于表示“是否仅允许节点左侧图标控制展开收缩”，默认 false
- [优化] tree 组件方法命名：getCheck 改名为 getChecked、setCheck 改名为 setChecked
- [优化] tree 组件多处异常样式
- [修复] tree 组件的 click 回调的若干问题
- [新增] transfer 组件的 width、height 参数，用于设定左右穿梭框的宽高
- [修复] transfer 组件在 form 元素下，穿梭框中的按钮触发了提交的 bug
- [优化] transfer 组件方法命名：getValue 改名为 getData

◦ v2.5.0 2019-05-31

此“更”时隔半载，其间个中缘由，尽在不言...

[新增] tree 组件

- [支持] 无限层级
- [支持] 自定义节点数据源：节点名称、勾选状态、禁用状态、拖拽禁止状态、新窗口跳转链接等
- [支持] 复选框勾选功能
- [支持] 层级连接线功能
- [支持] 节点的“增删改”内置操作
- [支持] 手风琴展开模式
- [支持] 设定默认展开的节点
- [提供] 节点被点击、复选框勾选、节点过滤、“增删改”等多种操作场景下的回调
- [提供] 设置指定的节点为勾选状态的方法
- [提供] 获取当前勾选的全部节点的方法

- [提供] 数据为空的自定义文本提示功能
- [新增] transfer 穿梭组件
 - [支持] 设定左右栏初始化数据
 - [支持] 列表搜索功能
 - [支持] 自定义左右栏标题
 - [提供] 左右穿梭交互时的回调
 - [提供] 获取选中数据（右侧列表）的方法
- [加强] table 组件
 - [优化] table.reload(id, options, type) 方法，可通过设置 type 为 "data"，只刷新数据部分（无抖动）（注：2.5.2 已取消）
 - [优化] 打印功能，可包含“合计行”打印
 - [修复] 导出表格时，如果内容出现逗号出现数据错位的问题
 - [修复] 无法导出自定义模版（templet）中的数据，而是导出了原始数据的问题
 - [优化] 多处核心代码
- [加强] form 组件
 - [新增] 元素属性 lay-reqText="", 可用于自定义必填项（即设定了 lay-verify="required" 的表单）的提示文本
 - [优化] form 组件的 val() 方法，以避免 radio 元素无法赋值数值型数字的问题
 - [修复] form 中绑定了 layDate 组件的元素在验证 date 失败时，点击不弹出 layDate 组件的问题
- [完善] upload 组件
 - [增加] inst.reload(options) 方法，用于重置实例的属性值（比如动态改变 acceptMime 等）。inst 是 upload.render() 返回的实例
 - [修复] 在回调中无法获取到 this.item 的问题
- [完善] 其他
 - [新增] util 模块的 util.event(attr, obj, eventType) 方法，用于更好地处理事件动作
 - [修复] 在使用 layui.all.js 的情况下，use 一个依赖了非内置模块的自定义模块出现的回调异常问题

- [修复] carousel 轮播组件 interval 属性可能存在的异常问题

•

2.4.x

欢迎来到 layui 2.4。本次除了 table 组件较大幅度的加强，还特别添加了业界比较常见的两款新组件：colorpicker（颜色选择器）和 slider（滑块）。依然没有看到 tree？多选框等等你想要的功能性组件？不必着急，我们推出了 [layui 第三方扩展组件平台](#)，它用于收集来自 layui 官方以外的组件，由贡献者自己维护，以群众之力共同完善 layui！

◦ v2.4.5 2018-11-01

- [修复] table 组件中勾选全选后，再点击任意行复选框无法获取到值的问题
- [修复] table 组件的 toolbar: true 时，在高版本 jQuery 下出现的报错问题
- [修复] table 组件的固定列高在某些情况未能铺满的问题
- [优化] table 组件的单元格溢出下拉框，让其不触发行点击事件
- [修复] slider 组件在 max 参数设为某些数字（如 20）时，点击 input 增减按钮出现的异常问题
- [优化] slider 组件局部代码，以自动纠正某些异常参数值
- [修复] form 组件的 select 在空值状态下双击，导致赋值了空值提示文本的问题

◦ v2.4.4 2018-10-29

- [新增] table 组件的 autoSort 参数，用于控制是否在前端自动排序。若 false，则需自己（服务端）返回排序后的数据
- [新增] table 组件的 [resize](#) 方法，用于重置表格尺寸等视图结构（该优化来自“layui 社区之光”@[岁月小偷](#)）
- [优化] table 组件合计行浮点型数据精度
- [优化] table 组件列筛选弹出框的高度问题，限制最大高度为表格高度
- [修复] table 组件的隐藏列未能同步“合计行”状态的问题
- [修复] form 组件的 val() 方法没找到相关表单出现报错的问题

◦ v2.4.3 2018-09-03

- [修复] table 组件在设列有 hide: true 情况下的宽度自适应问题
- [修复] table 组件的多级表头，设置了 hide 的各种问题
- [修复] table 组件的合计行有值为 null、空字符、小数点等情况时未正确统计的问题

- [修复] table 组件设置为 size: 'sm' 时, 行中 radio 单选框样式的不雅问题
- [优化] colorpicker 颜色面板拖动体验

◦ v2.4.2 2018-08-30

- [修复] table 组件初始设定 hide 属性, 表格未能正常显示的问题
- [修复] table 组件的多级表头下, 点击筛选列出现的异常问题
- [修复] table 组件在隐藏列后, 执行 table.reload() 方法重载后表头后的异常问题
- [修复] table 组件的 loading 参数设为 false 的异常问题
- [剔除] table 组件的 height 参数最小高度的判断, 改回之前的自由设定
- [优化] table 组件局部代码, 以适应复杂表头下的各种功能
- [优化] table 组件的打印, 不包含非数据列
- [优化] slider 组件局部代码
- [修复] 2.4.1 改动的层级优先级导致的各种堆叠异常问题 (尤其是对 layuiAdmin 的影响)

◦ v2.4.1 2018-08-29

- [新增] table 组件的 defaultToolbar 参数, 可自由配置头部工具栏右侧的图标 [#详见文档](#)
- [优化] table 组件的筛选下拉框在表格高度较短时被遮挡的问题
- [优化] table 组件的 toolbar 参数, 可通过设置 "default" 值来显示内置图标
- [优化] table 组件的 height 参数, 加入最小高度的判断, 以保证任何情况都能良好展示
- [优化] 各组件、元素之间的 z-index 层级优先级
- [优化] 颜色选择器的状态可即时跟随面板中的输入框值改变而改变
- [优化] 颜色选择器的局部代码和外观
- [修复] 颜色选择器的 value 参数设置 hex 部分初始值时的报错问题

◦ v2.4.0 2018-08-28

[加强] table 组件

- [新增] 工具栏区域, 通过 toolbar 参数开启, 可自定义工具按钮 [#详见文档](#)
- [新增] 列显示隐藏功能, 位于工具栏右侧

- [新增] 表格导出功能，位于工具栏右侧。也可通过 `table.exportFile()` 方法导出
- [新增] 表格打印功能，位于工具栏右侧
- [新增] 合计行功能，通过 `totalRow` 参数开启
- [新增] 单选框列，通过 `type: 'radio'` 开启
- [新增] 工具栏事件 (`toolbar`)、行单击事件 (`row`)、行双击事件 (`rowDouble`)
- [新增] 数据预解析回调函数: `parseData`，用于将返回的任意数据格式解析成 `table` 组件规定的数据格式 [#详见文档](#)
- [新增] 数据初始时的 `loading`，并更换了风格
- [新增] `title` 参数，用于定义 `table` 的大标题（在文件导出等地方会用到）
- [新增] 表头参数 `cols` 的 `hide` 子参数，用于控制隐藏列
- [新增] 表头参数 `cols` 的 `totalRow` 子参数，用于开启该列的合计功能
- [新增] 表头参数 `cols` 的 `totalRowText` 子参数，用于显示自定义的合计文本
- [优化] 自动分配列宽导致无故出现横向滚动条的问题
- [优化] 自动分配的列宽可跟随浏览器窗体尺寸改变而重新适配
- [优化] 单元格溢出状态的展开体验
- [优化] 多处细节问题
- [修复] `templet` 为函数时，单元格编辑和 `tool` 事件的 `update` 均未执行模板解析的问题
- [修复] `form` 组件中的元素在 `table` 中一些不友好的显示问题
- [修复] 表格容器在隐藏区域时（如 `Tabs` 中），未能显示分页栏的问题

。[新增] colorpicker 颜色选择器组件

- [支持] hex、rgb、rgba 三类色彩模式
- [支持] 自由拖拽设置
- [支持] 透明度拖拽设置，需配合 rgba 色值
- [支持] 预定义颜色列表，可自由配置
- [支持] 三种颜色框风格，可用于和不同尺寸的表单和按钮搭配
- [支持] 颜色被改变和选择完毕的回调
- [注意] colorpicker 组件不支持ie10以下版本，其它高级浏览器均支持
- [#详见文档](#)

。[新增] slider 滑块组件

- [支持] 水平和垂直两种滑块类型
- [支持] 支持自定义滑块数值区间
- [支持] 开启滑块范围模式拖拽

- [支持] 开启输入框动态改变滑块数值
- [支持] 自定义滑块间隔长度
- [支持] 自定义数值提示文本
- [支持] 自定义主题色
- [支持] 滑块数值被改变的回调，以及动态改变滑块的外部方法
- [#详见文档](#)

◦ [完善] form 组件

- [修复] checkox 组件的 primary 风格在文字很长的情况下出现的显示异常问题
- [修复] select 组件搜索后，上下快捷键选择失效的问题

◦ 其它

- [修复] layDate 组件在设置了非法 初始 value 时，点击空白处报错的问题
- [剔除] css 中备选字体 \5FAE\8F6F\96C5\9ED1 (微软雅黑)，以免版权问题。尽管它并不会优先使用

•

2.3.0

2018-05-25

你好，layui 2.3。它除了包含许多久违的细节之更，更是新增了 rate 评分组件，这是 layui 团队新成员 [@star1029](#) 的第一款组件。

◦ rate 评分

- 新增的全新组件，用于进行评分/星操作
- 支持是否开启半星
- 支持自定义星星个数
- 支持自定义星星主题色
- 支持自定义评级对应的说明，如：文本/图标等
- 支持初始值的设定
- 支持是否只读，即：只展示、不可点
- [#详见文档](#)

◦ laypage 通用分页

- 对 layout 参数新增 refresh 支持，用于显示刷新当前页图标

◦ laydate 日期与时间

- 新增 `isInitValue` 参数，用于控制是否自动向元素填充初始值（需配合 `value` 参数使用）（2.3.0 正式版改为默认 `true`）

◦ form 表单

- 增加 `form.val(filter, fields)` 方法，用于给指定表单集合的元素初始赋值。[#详见文档](#)
- 对 `select` 组件新增上下键（↑ ↓）回车键（Enter）选择功能（2.3.0 正式版修复了滚动条不跟随的问题）。
- 优化 `switch` 开关组件，让其能根据文本自由伸缩宽
- 修复 `checkbox` 复选框组件在高分辨屏下出现的样式不雅问题

◦ upload 文件上传

- 对 `choose`、`before` 回调返回的参数 `obj` 增加 `resetFile` 方法，可用于重命名文件名
- 修复开启 `size` 参数后，文件超出规定大小时，提示信息有误的问题

◦ flow 流加载

- 修复开启 `isLazyimg:true` 后，图片懒加载但是图片不存在的报错问题

◦ 其它改动

- 新增 21 个字体图标
- 字体图标开始支持规范化的 `font-class` 命名方式 [#详见文档](#)
- 新增 卡片面板 `class="layui-card"` 元素结构（需配合“非白”背景凸显效果）[#详见文档](#)
- `util` 模块新增 `escape` 方法，用于转义 `xss` 字符

•

2.2.6

2018-04-03

- 新增 `table` 的数据请求时的 `headers` 参数支持，用于添加请求头
- 新增 `nav` 垂直导航菜单的无限级子菜单功能（注意：水平导航菜单暂不支持无限级）

- 新增 nav 导航菜单基础属性 lay-shrink="all", 用于开启: 展开子菜单时, 收缩兄弟节点已展开的子菜单
- 新增 upload 的数据请求时的 headers 参数支持, 用于添加请求头
- 优化 upload 组件的 data 参数, 支持方法写法, 用于传递动态值。#[详见文档](#)
- 优化 element 的 nav 事件, 并解决了之前存在的父菜单无法触发事件的问题
- 新增 upload 组件的 acceptMime 参数, 规定打开文件选择框时, 筛选显示的文件类型 #91
- layui.build() 第一个参数支持 html 原始对象 #146
- Support post+json for table module #194

•

2.2.5

2018-01-03

2018年的第一个新版本

- 新增 table 的 templet 方法支持, 即现在自定义模板可以采用方法的形式替代内容 #[详见文档](#)
- 新增 table 的 text 参数, 用于自定义各种默认提示文本
- 新增 layui.factory(modName) 方法, 用于获取模块对应的工厂, 以便回收利用
- 新增 layui.sessionData(table, settings) 方法, 用于存储本地会话性数据
- 新增用于定义按钮容器的 class 选择器: layui-btn-container
- 新增用于定义流体按钮的 class 选择器: layui-btn-fluid
- 优化 table 的分页栏, 如果无数据则不显示
- 优化 layer 的 prompt 层, 初始赋值 value 时的光标会出现在最后 (之前版本会在最前)
- 优化 layui.event() 和 layui.onevent() 方法, 处理自定义事件更加灵活
- 优化 layui.router() 方法, 新增返回拼接后的 href
- 修复 table 的异步数据在 code 为非0时, 未执行 done 回调的问题
- 修复 element.tabChange() 方法的 this 指向问题

•

2.2.4

2017-12-07

该版本对之前存在的较多问题进行了一次清理, 推荐升级。

- 新增 form 的 name 为数组格式 (如: name="arr[]") 的支持
- 修复 form 的 select/checkbox/radio 等元素设定 lay-verType="tips" 时的提示层定位问题
- 修复 table 的自适应列宽可能引发的报错: Cannot read property 'defaultView' of null
- 修复 table 中渲染已知数据时执行 reload 不会清除之前数据的问题
- 修复 upload 的 number 在 size 未设定时无效的问题
- 修复 upload 的 allDone 回调在 auto: false 时的异常问题
- 修复 upload 在 IE11 下设定 exts 的某些异常提示问题
- 鉴于大家对 table 在 2.2.0 中的边框线普遍反映太浅, 特此加深, 回归统一边框

- 2.2.45 (2017-12-08 补充):
- 进一步优化 form 的 name 数组支持
- 修复 2.2.4 对 table reload 造成错误影响的问题

•

2.2.3

2017-11-27

- 新增 upload 模块的 number 参数, 用于设定同时允许上传的文件数量 (默认不限制) [#详见文档](#)
- 新增 util 模块的 util.toDateString(time, format) 方法, 用于转化时间戳或日期对象为日期格式字符 [#详见文档](#)
- 新增 util 模块的 util.digit(num, length) 方法, 用于对数字前置补零, 如: 6 ==> 06 [#详见文档](#)
- 修复 table 特殊情况出现的: 'cssRules' of undefined 报错问题
- 修复 table 无数据时, 点击列头的 checkbox 和排序出现的报错问题
- 修复 laydate 的 set 方法报错问题
- 恢复 form 的 lay-filter 在 layui 2.2.0 之前的验证规则, 即: 值为空也会触发内置的验证规则。选填项由开发者通过自定义验证来把控。

•

2.2.2

2017-11-17

- 优化 table 的列宽自动适配算法

- 修复 table 的重载后，分页总条数未更新的问题
- 修复 table 的删除行后，再点击全选，导致 table.checkStatus() 返回的数据异常问题
- 修复 table 的选择条数在 Firefox 浏览器下失效的问题
- 修复 table 的表头在同样的列宽下，Chrome 下正常，Firefox 和 ie 低版本下却出现省略号的差异性问题的
- 修复 layDate 中选择 datetime 选择器的月份列表报错的问题（尽管之前的报错不影响正常使用）
- 去除 h1/h2/h3 的 14px 的重置样式

- 注意事项：
- 转换静态表格，请务必确保 table.init() 方法的 limit 参数（默认：10）是与你服务端限定的数据条数一致
- 覆盖升级后注意清除浏览器缓存

•

2.2.1

2017-11-16

- 新增 table 的基础参数：cellMinWidth，用于全局定义所有常规单元格的最小宽度（默认 60）[#详见文档](#)
- 新增 table 的表头参数：minWidth，用于局部定义常规单元格的最小宽度（默认 60），优先级大于 cellMinWidth [#详见文档](#)
- 优化 table 内容超出单元格宽度时的一些小细节
- 优化 table 选中行样式
- 修复 table 的 page 参数传入 laypage 对象时的各种问题。这次可以真正做到分页的各种自定义，[官网示例](#) 页面中也有相关演示
- 修复 table 的表元素模板在 lg 尺寸单元格中未垂直居中的问题（注意，sm 尺寸的表格不做相关兼容）
- 修复 table 的查看更多的单元格弹层在自定义模板的情况下，无效的问题
- 修复在 ie8 下路径计算异常导致的所有 layui 内置模块失效的严重 bug
- 覆盖升级后注意清除浏览器缓存

•

2.2.0

2017-11-15

本次升级幅度较大，主要核心还是在 table 模块。无论是从代码，还是文档和示例的撰写上，都进行了

大量调整。除此之外，form 以及其它的几项改进，也堪称良心之举，这应该是一次令人愉悦的更新吧。特别说明的是：原计划与 layuiAdmin 同时发布，由于会员群对 layui 2.2 的呼声太高，经投票最终还是决定先发布 layui 2.2。而 layuiAdmin 和 Fly社区模板3.0，都将陆续推出，请耐心等待！

。table 改进

- 新增列宽自动适应功能，允许列宽设置百分比甚至不填写，内部自动分配
- 新增序号列支持（type: 'numbers'）[#详见文档](#)
- 新增表头的基础参数：type，用于定义列类型（normal/checkbox/space/numbers等）
- 优化 page 参数，支持传入一个对象，里面可包含 laypage 组件所有支持的参数（jump、elem除外）
- limit 改为默认 10
- 自定义模板中，可通过 undefined 获得序号
- 优化主题样式，风格更加清爽，进一步接近 layui 极简理念
- 优化单元格编辑样式（之前编辑时的样式不大明显）
- 修复单元格弹出的更多内容浮层的若干问题
- 修复 表单元素（如：复选框/开关/单选框）在数据表格单元格中的样式异常问题
- 修复表格重载时的左右抖动问题

。form 改进

- 设定 lay-verify 时，不再强制必填，除非同时设定了 required，如：lay-verify="phone|required"（注：2.2.3 已取消）
- 对表单元素新增 lay-verType="" 属性，用于定义异常提示层模式，可选值有：tips（吸附）、alert（对话框），msg（默认）[#详见文档](#)
- 搜索选择框匹配到相应选项时，如果未选择相应项，则失去焦点时清空内容（这样做的目的，是为了确保 select 是 option元素中定义的 value，而不是 text）
- 复选框禁用状态样式优化

。其它改进

- carousel 轮播当只有一个条目时，不会再显示指示器
- 导航条允许设置 lay-unselect 属性，点击指定导航菜单标题时，将不会出现选中效果
- layui.extend() 方法支持拓展一个自定义路径的 js 模块（可加载远程文件）
- 优化 layDate 组件范围选择样式
- 修复 layui.data() 方法无法存储空字符的值的问题

。 注意事项

- 为了统一规范，定义按钮尺寸的 CSS 类：layui-btn-mini 改为 layui-btn-xs、layui-btn-small 改为 layui-btn-sm、layui-btn-big 改为 layui-btn-lg，升级版本时务必注意
- 覆盖升级后，注意清理浏览器缓存

•

2.1.7

2017-11-02

- 新增 upload 组件的 allDone 回调，用于监听多文件上传完毕后的状态（如：总文件数、成功数、失败数）
- 新增 progress 进度条的分数设定（如：1/3）
- 修复 layDate 组件当前面板为 1 月时，上个月（12月）的最后一天显示为 30 的 bug
- 修复 layDate 组件输入一个非法值再点击其它同类控件，未校验上一个控件日期是否非法的问题
- 修复 progress 进度条的 lay-percent 设定百分比在某些情况下存在的 bug
- 优化 badge 徽章默认样式

•

2.1.6

2017-10-30

- 新增 element.tab(options) 方法，用于绑定自定义 Tab 元素（即非 layui 自带的 tab 结构，[#详见文档](#)）
- 新增 tab 选项卡删除的事件监听（tabDelete）
- 新增 laytpl 的忽略分隔符 `{{! template !}}`，用于忽略指定区域的模板解析
- 新增用于替代 element.init 的 element.render() 方法，并增加了两个参数：type、filter，以便对元素进行局部初始化。[#详见文档](#)
- 为了编码统一，element.init() 方法可用 element.render() 方法替代。但 init 方法仍会兼容。
- 优化 util 模块的 util.timeAgo() 方法的返回字符
- 优化二级导航的箭头高度在移动设备偏下的问题
- 修复垂直导航的一级导航行高没居中的问题
- 修复 table 在列宽足够的情况下点击列可能会弹出 tips 的问题 [#68](#)

•

2.1.5

2017-09-20

本次更新内容较少，主要对一些细节进行优化。

- 优化按钮样式，默认不再有 0.9 的透明度，因为这样看上去会更鲜艳些
- 优化 carousel 轮播逻辑，以使条目切换的衔接动画更加自然些
- 修复 carousel 模块的轮播在切换浏览器窗口后，出现空白 loading 的问题
- 修复 input/textarea 等表单元素在搜狗浏览器下的 placeholder 行高问题
- 修复 select 搜索框在 ios 设备和 safari 下无法输入的问题
- 解决 后台布局 中滑动顶部最右导航，引发水平滚动条的不良体验问题（需向 body 加入一段样式，[#详见文档](#)）

•

2.1.4

2017-09-15

- 优化原始按钮行高偏下问题
- 优化 table 多级表头并加强，使其真正做到无限级，且功能稳定。[#见示例](#)
- 修复 table 如果有一页空数据，导致所有分页都显示为空数据的问题
- 修复 table 的多级表头设定 checkbox 列后的异常问题
- 修复 table 的多级表头如果组合列后面还有列，表格主体与表头未对应上的问题
- 修复 table 中设置了 response 的 dataName，点击排序执行 table.reload() 方法后，无数据的问题
- 修复 layDate 在 requirejs 下使用的报错问题
- 如果 layDate 在 requirejs 下使用，需要通过 laydate.path = "" 来设定 laydate.js 所在目录 [#见文档](#)

•

2.1.3

2017-09-13

2.0 发布已经过去三周了，相比功能的继续丰富，也许更重要的是稳定，我们深知这一点。所以请尝试从“layui新版本存在许多问题”的错觉中清醒，事实上，当你真正适应了这一切，你会对 layui 有一个

全新的认知。而今天更新的 2.1.3 主要还是力求稳定，但在下一个较大版本（2.2.0），你将会看到如你所愿的改进，也是社区呼声最高的存在，想象一下吧！

- 优化 内置动画，以避免在 Chrome 最新版(61.0.3163.79) 中出现的各种奇异现象
- 优化 layDate 的时间选择器（time）的范围选择体验，不再校验“结束时间超出开始时间”，以支持跨越凌晨的情况
- 优化 layDate 的控件关闭，以避免focus事件下出现的未关闭上一个控件的问题
- 优化 table 在行数据未填满表格高度时，固定列仍然100%高度所形成的不雅感
- 修复 layDate 中自定义无符号分割的日期格式（如yyyyMM、yyyyMMdd），二次选择后控件存在的异常问题
- 修复 layDate 中的日期时间选择器（datetime）开启范围选择时，选择当天未校验“结束时间超出开始时间”的问题
- 修复 table 中开启单元格编辑，在执行 update 方法后，值未同步编辑框的问题
- 修复 table 中转换多级表头的静态表格有数据未展示的问题
- 修复 layer 在最新版 Chrome（61.0.3163.79）下的遮罩层出现的奇异花屏现象
- 修复 form 中设定 lay-verify="number" 不能验证负数和小数的问题

•

2.1.2

2017-09-03

感谢来自百度MIP团队的 [前端小武](#) 提供自动化测试，目前已正式邀请其成为 layui collaborator

- 【layDate】修复 value 传入日期对象时，初始的input值异常问题
- 【layDate】修复设置了min/max，年月列表的确定不可点击问题
- 【table】优化 loading，改为显示在表格容器的正中位置
- 【table】修复右侧固定列在部分设备（如手机/Mac）无效的问题
- 【table】修复切换分页，未关闭上一个“查看全部”的提示层的问题
- 【form】修复 input 输入框的 placeholder 在部分浏览器（如：Safari）下行高问题
- 【底层】修复 layui.sort() 方法处理负数排序的问题 [#57](#)

•

2.1.1

2017-09-01

。功能新增

- [layDate] 对 mark 参数增加 0-0-date 支持，以标记每月的某天 [#见文档](#)
- [table] 新增 initSort 参数，用于初始设置排序字段与状态 [#见文档](#)

。完善与优化

- [layDate] 如果设置了 value，则默认给被绑定的元素（如 input）赋值

。Bug Fixes

- [layDate] 修复年列表的 disabled 小问题
- [layPage] 修复在某些情况下，选择不同每页条数时，分页结构存在的小问题
- [table] 修复 table sort 事件在执行 table.reload() 死循环的问题
- [table] 修复多级表头开启复选框或设置左右固定列时的主体与表头未对齐问题
- [upload] 修复多文件模式下，设置 size，出现重复请求的问题

•

2.1.0

2017-08-30

在过去一周对 2.0 的反馈收集，我很惊讶地看到关于数据表格的多种吐槽。layui 向来以易用著称，但这次，大家普遍对 layui table 的态度却似乎并不是这样。这让我一度懊恼与不解，尽管可能的答案是：layui 的使用群体正在变得更加广泛。但我更多的是在反思。于是，在这个版本中，一半坚持原有的理念，一半妥协，默默并努力改造成我和你们都希望的样子。而这，仅仅只是开始...

。功能新增

- [table] 新增 table.reload() 方法，以便对自动渲染的表格完成重载 [#见文档](#)
- [table] 新增 height 最大化减去差值的自动换算功能（语法：full-差值），以让容器始终适应屏幕高度 [#见文档](#)
- [table] 新增 response 参数，用于对返回的数据格式的自定义支持 [#见文档](#)
- [table] 新增 request 参数，用于对分页请求的参数：page/limit 的自定义名称支持 [#见文档](#)

- [table] 新增sort事件，用于监听排序动作 [#见文档](#)
- [upload] 开始支持直接在元素上配置基础参数
- [upload] 回调中可通过 `this.item` 读取到当前触发上传的元素，一般用于 `elem` 绑定 `class` 的情况

◦ 完善与优化

- [响应式] 平板设备的临界值改为以iPad为准（768px）
- [table] 去除选中的数据中的临时字段：LAY_CHECKED
- [table] 优化局部代码
- [tab选项卡] 点击 `target="_blank"` 类型的a标签时，不会对该项标记下标

◦ Bug Fixes

- [底层] 修复 `layui.sort()` 方法在个别特殊情况排序异常的问题
- [table] 修复编辑单元格或删除行后，对列进行排序出现刚初始时的数据的Bug
- [table] 修复在无数据的情况下开启 `toolbar` 且固定在右侧，出现重复的问题
- [layDate] 修复点击时间选择器的标题区域，出现报错的问题
- [layDate] 修复限制可选日期时，年列表和月列表存在的小问题
- [tab选项卡] 修复执行添加tabs时，当出现下拉，再删除最后面几个tabs出现的Bug

•

2.0.2

2017-08-24

跟上个版本一样，重心还是放在 `layDate` 和 `table` 上

◦ layDate 日期时间选择器

- 增加开始日期超出结束日期时点击确定出现的提示
- 修复回车事件影响其它元素（如 `textarea`）正常回车的失误性bug
- 修复：选择日期范围确定后，再重新打开并选择开始日期（却未点结束日期），再点击空白区域关闭，又重新打开，并选择更小一点的日期所出现的异常问题（卧槽，这个bug描述起来好累！你们读懂了吗？至少我都被自己说晕了。发现这个bug的同学心细如夸克）
- 修复时间选择器在“360极(keng)速(die)浏览器”下因显示隐藏滚

动条导致的样式异常问题

◦ Table 表格

- 增加对工具条模板的自定义 JS 脚本支持
- 新增列“空隙”类型，以定义一个 15px 宽度无任何内容的列
- 优化右固定列的左侧边框线、数据还未获取造成的分页落差感等细节问题
- 去除在首列值为 null 或 undefined 时的自动加序号功能
- 修复工具条，只有设置了 fixed 参数才会显示的 bug
- 修复列单元格设置居中时，点击超出文本，未弹出查看更多的浮层的问题

◦ 其它

- 新增用于在不同设备下显示和隐藏的响应式公共类（[见文档](#)）
- 优化导航样式
- 修复富文本编辑器多次上传图片的覆盖问题

•

2.0.1

2017-08-22

尽管昨天发布了大版本，但是有 Bug 还是得速修

◦ layDate 日期时间选择器

- 日期时间选择器改为只能点确定关闭（以便选择时间）
- 修复选择了当月的最后一天（比如29/31），再切换年月出现日期值超出的bug
- 修复ready回调返回的月份值没有加1的问题

◦ Table 表格

- 修复Java端的异步接口未指定json类型，前端无法获取到数据的问题
- 修复字符型数字排序异常的bug
- 修复如果数字为0，单元格未显示0的bug
- 修复勾选行，再删除行后，仍然可以获取该选中行数据的bug

。其它

- 修复 Upload模块 正常返回了JSON，仍然抛出提示“请对上传接口返回有效JSON”的bug
- 修复 富文本上传图片异常的BUG
- 修复 layui.sort(obj, key, desc) 方法对字符型数字排序异常的bug

•

2.0.0

2017-08-21

里程碑式版本。核心机制未变，更多是对 1.x 的扩充和完善。它的意义并不只是那几项更新，而是自此开始，layui 进入一段重生之旅。
因该版本改动较大，为避免代码冗余，特不兼容 1.x，请勿覆盖升级，你也可以阅读：[1.x升2.0注意事项](#)

。Table 表格

- 新增的全新模块，用于对表格进行一些列功能和动态数据操作
- 支持固定表头、固定行、固定列左/列右
- 支持拖拽改变列宽度
- 支持多级表头
- 支持大表格、小表格、默认表格的任意尺寸设定
- 支持多种表格风格设定
- 支持“Ajax接口获取”、“直接赋值数据”、“转化现有表格”三种初始化渲染方式
- 支持单元格的自定义模板
- 支持对表格重载（比如搜索、条件筛选等）
- 内置checkbox复选框功能
- 内置自定义工具条及相关操作功能
- 内置分页功能
- 内置字段排序功能
- 内置单元格编辑功能
- 内置显示单元格更多内容功能

。Carousel 轮播

- 新增的全新模块，用于处理页面轮播逻辑
- 支持图片、文字列表等任意内容的切换
- 支持普通轮播和全屏轮播（FullPage）的设定
- 支持多种切换动画的设定

- 支持是否自动切换、自动切换的时间间隔的设定
- 支持初始开始的条目索引的设定
- 支持箭头和指示器的风格和位置设定

◦ Layout 布局

- 新增栅格布局系统
- 栅格采用12等分，内置移动设备、平板、桌面中等和大型屏幕的响应式处理
- 栅格支持分栏间隔、列偏移、嵌套，流体布局等
- 栅格最低能支持到ie8
- 新增后台大框架布局现成方案

◦ layDate 日期时间选择器

- 全面重写，可作为独立组件（版本直接跃升为 5.0）
- 依旧采用原生JavaScript编写，零依赖，可在layui中作为模块使用，也可作为独立组件使用
- 支持单独显示年选择器、年月选择器、日期选择器、时间选择器、日期时间选择器
- 支持双控件，用于选择年/年月/日期/时间/日期时间五种类型选择器的范围（可顺时、逆时）
- 支持日期格式的自定义
- 支持日期是否合法的自动校验
- 支持有效日期范围的设定
- 支持内置事件（可自定义）、外部事件、直接显示等多种调用方式
- 支持中文版和国际版的语言设定
- 支持开启公历节日和标记重要日期
- 支持直接嵌套在页面的某个容器中
- 支持底部按钮的任意顺序排版
- 支持智能显示在最佳可视坐标
- 支持回车快捷键选择
- 支持多种内置主题的设定，支持自定义主题色，且可单独定制主题

◦ Upload 上传

- 全面重写
- 可指定任意元素（如按钮、普通div等）来触发上传
- 支持选择后自动上传和手工上传两种模式
- 支持附加参数、支持自定义文件名等
- 支持多文件上传（ie8/9除外）
- 支持拖拽文件上传（ie8/9除外）
- 支持文件大小限制，单位kb（ie8/9除外）

- 支持图片上传前预览（ie8/9除外）
- 支持文件跨域上传（ie8/9除外）

。 layPage 分页

- 核心代码和接口重写
- 新增“数据总数”、“每页条数”显示区域
- 支持自定义排版
- 新增count参数，用于得到数据总数，并剔除了pages参数（分页总数）
- 新增limits参数，用于设定每页条数的选择项
- 新增limit参数，用于设定每页条数的默认项
- 优化跳页框在输入非数字时的校验
- 总页数低于2时，仍然输出分页结构（前面版本不会显示）
- 尾页文本默认显示为总页数
- 跳页框如果输入的页码大于最大页数，则自动跳到最大页
- 样式优化

。 Form 表单集合

- select组件增加自动上下判断，用于显示在最佳可视区域
- select组件允许出现“请选择”的空值选项
- form.render(type, filter)方法增加第二个参数，用于指定某个区域进行局部渲染
- 优化复选框样式，以更友好地用于非form场景中
- form.on方法支持链式写法

。 Layer 弹层

- 同步到最新的 layer v3.1.0
- 增加maxHeight参数，用于设定弹层的最大高度
- 对默认按钮颜色、Tips层、Prompt层、Tab层等进行了样式微调，以便更显大气，且更符合layui风格

。 Element 页面元素

- 新增时间轴元素
- 新增徽章元素
- 新增动画CSS类文档
- 导航UI细节优化，并新增三种主题色支持：墨绿/藏青/蓝
- 导航支持加入图片
- 分割线新增可支持的颜色：赤/橙/墨绿/藏青/蓝/黑/灰
- Tab选项卡UI微调

- element模块输出的接口由先前的函数改为对象

。 Util 工具集

- 新增倒计时方法: `util.countdown()`
- 新增用于得到“某个时间在当前时间的多久前”的方法:
`util.timeAgo()`
- [固定块] 新增 `showHeight` 参数, 用于控制出现TOP按钮的滚动条高度临界值

。 底层方法

- 新增 `layui.sort(obj, key, desc)` 方法, 用于将数组中的对象按某个成员重新对该数组排序
- 改写`layui.router()`方法, 以更好地解析`location.hash`的单页URL规则

。 其它更改

- 新增28个字体图标
- 剔除全局滚动条样式
- 获取内置的jQuery接口, 可通过 `var $ = layui.$;` 得到, 之前的 `layui.jquery`仍然可用
- `layui.css`大量样式结构优化

。 Bug fixes

- 修复select组件在没有任何option的情况下报错的问题
- 修复导航多个排列在一起时, hover出现异常的问题
- 修复`layui.device()`方法在Chrome设备模式无法识别ios环境的问题
- 修复IE下, 多次执行`layui.use`加载同一个模块时, 控制台出现多条重复请求的问题 (实际上不是真实请求)

。 1.x 升 2.0 特别注意事项

- `layDate`日期模块、`layPage`分页模块、`Upload`上传模块等等, 均已完全重写, 请按照最新文档修改
- 获取 Form 模块接口, 由之前的 `var form = layui.form()` 改为: `var form = layui.form`
- 获取 Element 模块接口, 由之前的 `var element = layui.element()` 改为: `var element = layui.element`
- `layui.all.js` 的目录调整到跟 `layui.js` 的同级目录, 如有使

用到 layui.all.js，请注意修改路径

- 由于改动较大，2.0其实并不兼容1.x，强烈不推荐覆盖升级。官网仍会保留 1.x 的存档，最好按需升级。

•

1.0.1 到 1.0.9

2016-10-18 - 2017-02-28

- [1.0.9](#)
- [1.0.8](#)
- [1.0.7/1.0.6/1.0.5](#)
- [1.0.4](#)
- [1.0.3](#)
- [1.0.2](#)
- 据可靠消息：1.0.1 版本的日志已被土卫六人劫走，并无任何踪迹

•

1.0.0

2016-10-14

首个正式版本出蛋

•

孵化

layui - 用心与你沟通

- 布局 栅格 / 后台布局
- 颜色 主题色设计感 / 内置背景色
- 图标 字体图标
- 动画 内置 CSS3 动画
- 按钮 button 组
- 表单 form 元素集合
- 导航 菜单 / 面包屑
- 选项卡 Tabs 切换
- 进度条 progress
- 面板 折叠 / 手风琴
- 表格 静态 table
- 徽章 小圆点 / 小边框
- 时间线 timeline
- 辅助 引用 / 字段集 / 横线等

栅格系统与后台布局

如你所愿，在 layui 2.0 的版本中，我们加入了强劲的栅格系统和后台布局方案，这意味着你终于可以着手采用 layui 排版你的响应式网站和后台系统了。layui 的栅格系统采用业界比较常见的 12 等分规则，内置移动设备、平板、桌面中等和大型屏幕的多终端适配处理，最低能支持到 ie8。而你应当更欣喜的是，layui 终于开放了它经典的后台布局方案，快速搭建一个属于你的后台系统将变得十分轻松自如。

栅格系统

为了丰富网页布局，简化 HTML/CSS 代码的耦合，并提升多终端的适配能力，layui 在 2.0 的版本中引进了自己的一套具备响应式能力的栅格系统。我们将容器进行了 12 等分，预设了 4*12 种 CSS 排列类，它们在移动设备、平板、桌面中/大尺寸四种不同的屏幕下发挥着各自的作用。

一、栅格布局规则：

1.	采用 <code>layui-row</code> 来定义行，如： <code><div class="layui-row"></div></code>
2.	采用类似 <code>layui-col-md</code> 这样的预设类来定义一组列（ <code>column</code> ），且放在行（ <code>row</code> ）内。其中： <ul style="list-style-type: none"> • 变量 <code>md</code> 代表的是不同屏幕下的标记（可选值见下文） • 变量 代表的是该列所占用的12等分数（如6/12），可选值为 1 - 12 • 如果多个列的“等分数值”总和等于12，则刚好满行排列。如果大于12，多余的列将自动另起一行。
3.	列可以同时出现最多四种不同的组合，分别是： <code>xs</code> （超小屏幕，如手机）、 <code>sm</code> （小屏幕，如平板）、 <code>md</code> （桌面中等屏幕）、 <code>lg</code> （桌面大型屏幕），以呈现更加动态灵活的布局。
4.	可对列追加类似 <code>layui-col-space5</code> 、 <code>layui-col-md-offset3</code> 这样的预设类来定义列的间距和偏移。
5.	最后，在列（ <code>column</code> ）元素中放入你自己的任意元素填充内容，完成布局！

示例（这里只是大致列举两个，更多实例请前往 [示例-栅格](#) 查看）

你的内容 9/12						你的内容 3/12		
50% 33.33% 33.33%			50% 66.67% 33.33%			33.33% 100% 33.33%		
33.33% 50% 66.67%						33.33% 50% 33.33%		

```

1. <div class="layui-container">
2.   常规布局（以中型屏幕桌面为例）：
3.   <div class="layui-row">
4.     <div class="layui-col-md9">
5.       你的内容 9/12
6.     </div>
7.     <div class="layui-col-md3">
```

```
8.         你的内容 3/12
9.         </div>
10.    </div>
11.
12.    移动设备、平板、桌面端的不同表现：
13.    <div class="layui-row">
14.        <div class="layui-col-xs6 layui-col-sm6 layui-col-md4">
15.            移动：6/12 | 平板：6/12 | 桌面：4/12
16.        </div>
17.        <div class="layui-col-xs6 layui-col-sm6 layui-col-md4">
18.            移动：6/12 | 平板：6/12 | 桌面：4/12
19.        </div>
20.        <div class="layui-col-xs4 layui-col-sm12 layui-col-md4">
21.            移动：4/12 | 平板：12/12 | 桌面：4/12
22.        </div>
23.        <div class="layui-col-xs4 layui-col-sm7 layui-col-md8">
24.            移动：4/12 | 平板：7/12 | 桌面：8/12
25.        </div>
26.        <div class="layui-col-xs4 layui-col-sm5 layui-col-md4">
27.            移动：4/12 | 平板：5/12 | 桌面：4/12
28.        </div>
29.    </div>
30. </div>
31.
```

二、响应式规则：

栅格的响应式能力，得益于CSS3媒体查询（Media Queries）的强力支持，从而针对四类不同尺寸的屏幕，进行相应的适配处理

	超小屏幕 (手机<768px)	小屏幕 (平板 ≥768px)	中等屏幕 (桌面 ≥992px)	大型屏幕 (桌面≥1200px)
.layui-container 的值	auto	750px	970px	1170px
标记	xs	sm	md	lg
列对应类 为1-12的等分数值	layui-col-xs	layui-col-sm	layui-col-md	layui-col-lg*
总列数	12			
响应行为	始终按设定的比例水平排列	在当前屏幕下水平排列，如果屏幕大小低于临界值则堆叠排列		

三、响应式公共类：

类名 (class)	说明
layui-show--block	定义不同设备下的 display: block; 可选值有: xs、sm、md、lg
layui-show--inline	定义不同设备下的 display: inline; 可选值同上

layui-show--inline-block	定义不同设备下的 display: inline-block; 可选值同上
layui-hide-	定义不同设备下的隐藏类，即： display: none; 可选值同上

四、布局容器：

将栅格放入一个带有 `class="layui-container"` 的特定的容器中，以便在小屏幕以上的设备中固定宽度，让列可控。

```
1. <div class="layui-container">
2.   <div class="layui-row">
3.     .....
4.   </div>
5. </div>
6.
```

当然，你还可以不固定容器宽度。将栅格或其它元素放入一个带有 `class="layui-fluid"` 的容器中，那么宽度将不会固定，而是 100% 适应

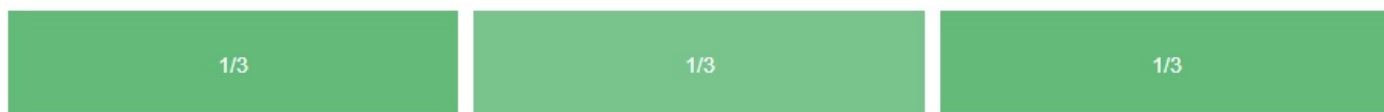
```
1. <div class="layui-fluid">
2.   .....
3. </div>
4.
```

五、列间距：

通过“列间距”的预设类，来设定列之间的间距。且一行中最左的列不会出现左边距，最右的列不会出现右边距。列间距在保证排版美观的同时，还可以进一步保证分列的宽度精细程度。我们结合网页常用的边距，预设了 12 种不同尺寸的边距，分别是：	
layui-col-space1 layui-col-space2 layui-col-space4 layui-col-space5 layui-col-space6 layui-col-space8 layui-col-space10 layui-col-space12 layui-col-space14 layui-col-space15 layui-col-space16 layui-col-space18	支持列之间为 1px-30px 区间的所有双数间隔，以及 1px、5px、15px、25px 的单数间隔

```
layui-col-space20
layui-col-space22
layui-col-space24
layui-col-space25
layui-col-space26
layui-col-space28
layui-col-space30
```

下面是一个简单的例子，列间距为10px：



```
1. <div class="layui-row layui-col-space10">
2.   <div class="layui-col-md4">
3.     1/3
4.   </div>
5.   <div class="layui-col-md4">
6.     1/3
7.   </div>
8.   <div class="layui-col-md4">
9.     1/3
10.  </div>
11. </div>
12.
```

如果需要的间距高于30px（一般不常见），请采用偏移，下文继续讲解

六、列偏移：

对列追加 类似 `layui-col-md-offset**` 的预设类，从而让列向右偏移。其中 `*` 号代表的是偏移占据的列数，可选中为 `1 - 12`。

如：`layui-col-md-offset3*`，即代表在“中型桌面屏幕”下，让该列向右偏移 3 个列宽度

下面是一个采用「列偏移」机制让两个列左右对齐的实例



```
1. <div class="layui-row">
2.   <div class="layui-col-md4">
```

```

3.      4/12
4.    </div>
5.    <div class="layui-col-md4 layui-col-md-offset4">
6.      偏移4列，从而在最右
7.    </div>
8.  </div>
9.

```

请注意，列偏移可针对不同屏幕的标准进行设定，比如上述的例子，只会在桌面屏幕下有效，当低于桌面屏幕的规定的临界值，就会堆叠排列。

七、栅格嵌套：

理论上，你可以对栅格进行无穷层次的嵌套，这更加增强了栅格的表现能力。而嵌套的使用非常简单。在列元素（`layui-col-md*`）中插入一个行元素（`layui-row*`），即可完成嵌套。下面是一个简单的例子：



```

1. <div class="layui-row layui-col-space5">
2.   <div class="layui-col-md5">
3.     <div class="layui-row grid-demo">
4.       <div class="layui-col-md3">
5.         内部列
6.       </div>
7.       <div class="layui-col-md9">
8.         内部列
9.       </div>
10.      <div class="layui-col-md12">
11.        内部列
12.      </div>
13.    </div>
14.  </div>
15.  <div class="layui-col-md7">
16.    <div class="layui-row grid-demo grid-demo-bg1">
17.      <div class="layui-col-md12">
18.        内部列
19.      </div>
20.      <div class="layui-col-md9">
21.        内部列
22.      </div>
23.      <div class="layui-col-md3">
24.        内部列
25.      </div>

```

```

26.     </div>
27.   </div>
28. </div>
29.

```

八、让IE8/9兼容栅格：

事实上IE8和IE9并不支持媒体查询 (Media Queries)，但你可以使用下面的补丁完美兼容！该补丁来自于开源社区：

```

1.  <!-- 让IE8/9支持媒体查询，从而兼容栅格 -->
2.  <!--[if lt IE 9]>
3.    <script src="https://cdn.staticfile.org/html5shiv/r29/html5.min.js"></script>
4.    <script src="https://cdn.staticfile.org/respond.js/1.4.2/respond.min.js">
      </script>
5.  <![endif]-->
6.

```

将上述代码放入你页面 `<head>` 标签内的任意位置

后台布局

layui 之所以赢得如此多人的青睐，更多是在于它前后台系统通吃的能力。既可编织出绚丽的前台页面，又可满足繁杂的后台功能需求。layui 致力于让每一位开发者都能轻松搭建自己的后台。下面是 layui 提供的一个现场的方案，你可以前往示例页面，[预览后台布局效果](#)

```

1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.    <meta charset="utf-8">
5.    <meta name="viewport" content="width=device-width, initial-scale=1, maximum-
      scale=1">
6.    <title>layout 后台大布局 - Layui</title>
7.    <link rel="stylesheet" href="../../src/css/layui.css">
8.  </head>
9.  <body class="layui-layout-body">
10. <div class="layui-layout layui-layout-admin">
11.   <div class="layui-header">
12.     <div class="layui-logo">layui 后台布局</div>
13.     <!-- 头部区域（可配合layui已有的水平导航） -->
14.     <ul class="layui-nav layui-layout-left">
15.       <li class="layui-nav-item"><a href="">控制台</a></li>
16.       <li class="layui-nav-item"><a href="">商品管理</a></li>
17.       <li class="layui-nav-item"><a href="">用户</a></li>
18.       <li class="layui-nav-item">
19.         <a href="javascript:;">其它系统</a>
20.         <dl class="layui-nav-child">
21.           <dd><a href="">邮件管理</a></dd>
22.           <dd><a href="">消息管理</a></dd>

```



```

23.         <dd><a href="">授权管理</a></dd>
24.     </dl>
25. </li>
26. </ul>
27. <ul class="layui-nav layui-layout-right">
28.     <li class="layui-nav-item">
29.         <a href="javascript:;">
30.             
31.                 贤心
32.             </a>
33.             <dl class="layui-nav-child">
34.                 <dd><a href="">基本资料</a></dd>
35.                 <dd><a href="">安全设置</a></dd>
36.             </dl>
37.         </li>
38.         <li class="layui-nav-item"><a href="">退了</a></li>
39.     </ul>
40. </div>
41.
42. <div class="layui-side layui-bg-black">
43.     <div class="layui-side-scroll">
44.         <!-- 左侧导航区域（可配合layui已有的垂直导航） -->
45.         <ul class="layui-nav layui-nav-tree" lay-filter="test">
46.             <li class="layui-nav-item layui-nav-itemed">
47.                 <a class="" href="javascript:;">所有商品</a>
48.                 <dl class="layui-nav-child">
49.                     <dd><a href="javascript:;">列表一</a></dd>
50.                     <dd><a href="javascript:;">列表二</a></dd>
51.                     <dd><a href="javascript:;">列表三</a></dd>
52.                     <dd><a href="">超链接</a></dd>
53.                 </dl>
54.             </li>
55.             <li class="layui-nav-item">
56.                 <a href="javascript:;">解决方案</a>
57.                 <dl class="layui-nav-child">
58.                     <dd><a href="javascript:;">列表一</a></dd>
59.                     <dd><a href="javascript:;">列表二</a></dd>
60.                     <dd><a href="">超链接</a></dd>
61.                 </dl>
62.             </li>
63.             <li class="layui-nav-item"><a href="">云市场</a></li>
64.             <li class="layui-nav-item"><a href="">发布商品</a></li>
65.         </ul>
66.     </div>
67. </div>
68.
69. <div class="layui-body">
70.     <!-- 内容主体区域 -->
71.     <div style="padding: 15px;">内容主体区域</div>
72. </div>
73.
74. <div class="layui-footer">
75.     <!-- 底部固定区域 -->

```

```
76.      © layui.com - 底部固定区域
77.      </div>
78. </div>
79. <script src="../../src/layui.js"></script>
80. <script>
81. //JavaScript代码区域
82. layui.use('element', function(){
83.     var element = layui.element;
84.
85. });
86. </script>
87. </body>
88. </html>
89.
```

结语

layui - 用心与你沟通

layui 颜色设计感

视觉疲劳的形成往往是由于颜色过于丰富或过于单一形成的麻木感，而 layui 提供的颜色，清新而不乏深沉，互相柔和，不过分刺激大脑皮层的神经反应，形成越久越耐看的微妙影像。合理搭配，可与各式各样的网站避免违和，从而使你的 Web 平台看上去更为融洽。

常用主色

#009688 通常用于按钮、及任何修饰元素	#5FB878 一般用于选中状态	#393D49 通常用于导航	#1E9FFF 比较适合一些鲜艳色系的页面
---------------------------	---------------------	-------------------	--------------------------

layui 主要是以象征包容的墨绿作为主色调，由于它给人以深沉感，所以通常会以浅黑色的作为其陪衬，又会以蓝色这种比较鲜艳的色调来弥补它的色觉疲劳，整体让人清新自然，愈发耐看。【取色意义】：我们执着于务实，不盲目攀比，又始终不忘绽放活力。这正是 layui 所追求的价值。

场景色

#FFB800 暖色系，一般用于提示性元素	#FF5722 比较引人注意的颜色	#01AAED 用于文字着色，如链接文本	#2F4056 侧边或底部普遍采用的颜色
--------------------------	----------------------	-------------------------	-------------------------

事实上，layui 并非不敢去尝试一些亮丽的颜色，但许多情况下一个它可能并不是那么合适，所以我们把这些颜色归为“场景色”，即按照实际场景来呈现对应的颜色，比如你想给人以警觉感，可以尝试用上面的红色。他们一般会出现在 layui 的按钮、提示和修饰性元素，以及一些侧边元素上。

极简中性色

他们一般用于背景、边框等

#F0F0F0	#f2f2f2	#eeeeee	#e2e2e2	#dddddd	#d2d2d2	#c2c2c2
---------	---------	---------	---------	---------	---------	---------

layui 认为灰色系代表极简，因为这是一种神奇的颜色，几乎可以与任何元素搭配，不易形成视觉疲劳，且永远不会过时。低调而优雅！

内置的背景色CSS类

layui 内置了七种背景色，以便你用于各种元素中，如：徽章、分割线、导航等等

赤色：	class="layui-bg-red"
橙色：	class="layui-bg-orange"
墨绿：	class="layui-bg-green"
藏青：	class="layui-bg-cyan"
蓝色：	class="layui-bg-blue"
雅黑：	class="layui-bg-black"
银灰：	class="layui-bg-gray"

结语

“不热衷于视觉设计的程序猿不是一个好作家！” — 贤心

layui - 用心与你沟通

字体图标











































layui 的所有图标全部采用字体形式，取材于阿里巴巴矢量图标库（iconfont）。因此你可以把一个 icon 看作是一个普通的文字，这意味着你直接用 css 控制文字属性，如 color、font-size，就可以改变图标的颜色和大小。你可以通过 font-class 或 unicode 来定义不同的图标。







使用方式











































通过对一个内联元素（一般推荐用 i 标签）设定 class="layui-icon"，来定义一个图标，然后对元素加上图标对应的 font-class（注意：layui 2.3.0 之前只支持采用 unicode 字符），即可显示出你想要的图标，譬如：











































1. 从 layui 2.3.0 开始，支持 font-class 的形式定义图标：
2. `<i class="layui-icon layui-icon-face-smile"></i>`
3. 注意：在 layui 2.3.0 之前的版本，只能设置 unicode 来定义图标
4. `<i class="layui-icon"></i>`
5. 其中的 ；即是图标对应的 unicode 字符
- 6.
7. 你可以去定义它的颜色或者大小，如：
8. `<i class="layui-icon layui-icon-face-smile" style="font-size: 30px; color: #1E9FFF;"></i>`
- 9.

内置图标一览表（168个）

						
实心  layui-icon-heart-fill	空心  layui-icon-heart	亮度/晴  layui-icon-light	时间/历史  layui-icon-time	蓝牙  layui-icon-bluetooth	@艾特  layui-icon-at	静音  layui-icon-mute
						
录音/麦克风  layui-icon-mike	密钥/钥匙  layui-icon-key	礼物/活动  layui-icon-gift	邮箱  layui-icon-email	RSS  layui-icon-rss	WiFi  layui-icon-wifi	退出/注销  layui-icon-logout
						
Android 安卓  layui-icon-android	Apple iOS 苹果  layui-icon-ios	Windows  layui-icon-windows	穿梭框  layui-icon-transfer	客服  layui-icon-service	减  layui-icon-subtraction	加  layui-icon-addition
						
滑块  layui-icon-slider	打印  layui-icon-print	导出  layui-icon-export	列  layui-icon-cols	退出全屏  layui-icon-screen-restore	全屏  layui-icon-screen-full	半星  layui-icon-rate-half
						
星星-空心  layui-icon-rate	星星-实心  layui-icon-rate-solid	手机  layui-icon-cellphone	验证码  layui-icon-vercode	微信  layui-icon-login-wechat	QQ  layui-icon-login-qq	微博  layui-icon-login-weibo
						
密码  layui-icon-password	用户名  layui-icon-username	刷新-粗  layui-icon-refresh-3	授权  layui-icon-aux	左向右伸缩菜单  layui-icon-spread-left	右向左伸缩菜单  layui-icon-shrink-right	雪花  layui-icon-snowflake

						
提示说明  layui-icon-tips	便签  layui-icon-note	主页  layui-icon-home	高级  layui-icon-senior	刷新  layui-icon-refresh	刷新  layui-icon-refresh-1	旗帜  layui-icon-flag
						
主题  layui-icon-theme	消息-通知  layui-icon-notice	网站  layui-icon-website	控制台  layui-icon-console	表情-惊讶  layui-icon-face-surprised	设置-空心  layui-icon-set	模板  layui-icon-template-1
						
应用  layui-icon-app	模板  layui-icon-template	赞  layui-icon-praise	踩  layui-icon-tread	男  layui-icon-male	女  layui-icon-female	相机-空心  layui-icon-camera
						
相机-实心  layui-icon-camera-fill	菜单-水平  layui-icon-more	菜单-垂直  layui-icon-more-vertical	金额-人民币  layui-icon-rmb	金额-美元  layui-icon-dollar	钻石-等级  layui-icon-diamond	火  layui-icon-fire
						
返回  layui-icon-return	位置-地图  layui-icon-location	办公-阅读  layui-icon-read	调查  layui-icon-survey	表情-微笑  layui-icon-face-smile	表情-哭泣  layui-icon-face-cry	购物车  layui-icon-cart-simple
						
购物车  layui-icon-cart	下一页  layui-icon-next	上一页  layui-icon-prev	上传-空心-拖拽  layui-icon-upload-drag	上传-实心  layui-icon-upload	下载-圆圈  layui-icon-download-circle	组件  layui-icon-component

 文件-粗  layui-icon-file-b	 用户  layui-icon-user	 发现-实心  layui-icon-find-fill	 loading  layui-icon-loading	 loading  layui-icon-loading-1	 添加  layui-icon-add-1	 播放  layui-icon-play
 暂停  layui-icon-pause	 音频-耳机  layui-icon-headset	 视频  layui-icon-video	 语音-声音  layui-icon-voice	 消息-通知-喇叭  layui-icon-speaker	 删除线  layui-icon-fonts-del	 代码  layui-icon-fonts-code
 HTML  layui-icon-fonts-html	 字体加粗  layui-icon-fonts-strong	 删除链接  layui-icon-unlink	 图片  layui-icon-picture	 链接  layui-icon-link	 表情-笑-粗  layui-icon-face-smile-b	 左对齐  layui-icon-align-left
 右对齐  layui-icon-align-right	 居中对齐  layui-icon-align-center	 字体-下划线  layui-icon-fonts-u	 字体-斜体  layui-icon-fonts-i	 Tabs 选项卡  layui-icon-tabs	 单选框-选中  layui-icon-radio	 单选框-候选  layui-icon-circle
 编辑  layui-icon-edit	 分享  layui-icon-share	 删除  layui-icon-delete	 表单  layui-icon-form	 手机-细体  layui-icon-cellphone-fine	 聊天 对话 沟通  layui-icon-dialogue	 文字格式化  layui-icon-fonts-clear
 窗口  layui-icon-layer	 日期  layui-icon-date	 水 下雨  layui-icon-water	 代码-圆圈  layui-icon-code-circle	 轮播组图  layui-icon-carousel	 翻页  layui-icon-prev-circle	 布局  layui-icon-layouts

 工具  layui-icon-util	 选择模板  layui-icon-template-1	 上传-圆圈  layui-icon-upload-circle	 树  layui-icon-tree	 表格  layui-icon-table	 图表  layui-icon-chart	 图标 报表 屏幕  layui-icon-chart-screen
 引擎  layui-icon-engine	 下三角  layui-icon-triangle-d	 右三角  layui-icon-triangle-r	 文件  layui-icon-file	 设置-小型  layui-icon-set-sm	 减少-圆圈  layui-icon-reduce-circle	 添加-圆圈  layui-icon-add-circle
 404  layui-icon-404	 关于  layui-icon-about	 箭头 向上  layui-icon-up	 箭头 向下  layui-icon-down	 箭头 向左  layui-icon-left	 箭头 向右  layui-icon-right	 圆点  layui-icon-circle-dot
 搜索  layui-icon-search	 设置-实心  layui-icon-set-fill	 群组  layui-icon-group	 好友  layui-icon-friends	 回复 评论 实心  layui-icon-reply-fill	 菜单 隐身 实心  layui-icon-menu-fill	 记录  layui-icon-log
 图片-细体  layui-icon-picture-fine	 表情-笑-细体  layui-icon-face-smile-fine	 列表  layui-icon-list	 发布 纸飞机  layui-icon-release	 对 OK  layui-icon-ok	 帮助  layui-icon-help	 客服  layui-icon-chat
 top 置顶  layui-icon-top	 收藏-空心  layui-icon-star	 收藏-实心  layui-icon-star-fill	 关闭-实心  layui-icon-close-fill	 关闭-空心  layui-icon-close	 正确  layui-icon-ok-circle	 添加-圆圈-细体  layui-icon-add-circle-fine

跨域问题的解决

由于浏览器存在同源策略，所以如果 layui（里面含图标字体文件）所在的地址与你当前的页面地址不在同一个域下，即会出现图标跨域问题。所以要么你就把 layui 与网站放在同一服务器，要么就对 layui 所在的资源服务器的 Response Headers 加上属性：Access-Control-Allow-Origin: *

layui - 用心与你沟通

CSS3动画类

在实用价值的前提之下，我们并没有内置过多花俏的动画。而他们同样在 layui 的许多交互元素中，发挥着重要的作用。layui 的动画全部采用 CSS3，因此不支持ie8和部分不支持ie9（即ie8/9无动画）

使用方式

动画的使用非常简单，直接对元素赋值动画特定的 class 类名即可。如：

1. 其中 layui-anim 是必须的，后面跟着的即是不同的动画类
2. `<div class="layui-anim layui-anim-up"></div>`
- 3.
4. 循环动画，追加：layui-anim-loop
5. `<div class="layui-anim layui-anim-up layui-anim-loop"></div>`
- 6.

内置CSS3动画一览表

下面是不同的动画类名，数量可能有点少的样子？但正如开头所讲的，拒绝冗余花俏，拥抱精简实用。点击下述绿色块，可直接预览动画

 <p>从最底部往上滑入</p> <p>layui-anim-up</p>	 <p>微微往上滑入</p> <p>layui-anim-upbit</p>	 <p>平滑放大</p> <p>layui-anim-scale</p>	 <p>弹簧式放大</p> <p>layui-anim-scaleSpring</p>
 <p>渐现</p> <p>layui-anim-fadein</p>	 <p>渐隐</p> <p>layui-anim-fadeout</p>	 <p>360度旋转</p> <p>layui-anim-rotate</p>	 <p>循环动画</p> <p>追加：layui-anim-loop</p>

结语

物不在多，有用则精。

layui - 用心与你沟通

按钮 - 页面元素

向任意HTML元素设定`class="layui-btn"`，建立一个基础按钮。通过追加格式为`layui-btn-{type}`的class来定义其它按钮风格。内置的按钮class可以进行任意组合，从而形成更多种按钮风格。

用法

```
1. <button type="button" class="layui-btn">一个标准的按钮</button>
2. <a href="http://www.layui.com" class="layui-btn">一个可跳转的按钮</a>
3.
```

主题



名称	组合
原始	class="layui-btn layui-btn-primary"
默认	class="layui-btn"
百搭	class="layui-btn layui-btn-normal"
暖色	class="layui-btn layui-btn-warm"
警告	class="layui-btn layui-btn-danger"
禁用	class="layui-btn layui-btn-disabled"

尺寸



尺寸	组合
大型	class="layui-btn layui-btn-lg"
默认	class="layui-btn"
小型	class="layui-btn layui-btn-sm"
迷你	class="layui-btn layui-btn-xs"



尺寸	组合
大型百搭	class="layui-btn layui-btn-lg layui-btn-normal"
正常暖色	class="layui-btn layui-btn-warm"
小型警告	class="layui-btn layui-btn-sm layui-btn-danger"
迷你禁用	class="layui-btn layui-btn-xs layui-btn-disabled"



```
1.  
2. <button type="button" class="layui-btn layui-btn-fluid">流体按钮（最大化适  
   应）</button>  
3.
```

圆角



主题	组合
原始	class="layui-btn layui-btn-radius layui-btn-primary"
默认	class="layui-btn layui-btn-radius"
百搭	class="layui-btn layui-btn-radius layui-btn-normal"
暖色	class="layui-btn layui-btn-radius layui-btn-warm"
警告	class="layui-btn layui-btn-radius layui-btn-danger"
禁用	class="layui-btn layui-btn-radius layui-btn-disabled"



尺寸	组合
大型百搭	class="layui-btn layui-btn-lg layui-btn-radius layui-btn-normal"
小型警告	class="layui-btn layui-btn-sm layui-btn-radius layui-btn-danger"
迷你禁用	class="layui-btn layui-btn-xs layui-btn-radius layui-btn-disabled"

哈哈哈哈哈，这组合名长得简直没朋友~ ☐ ☐

图标

```

1. <button type="button" class="layui-btn">
2.   <i class="layui-icon">#xe608;</i> 添加
3. </button>
4.
5. <button type="button" class="layui-btn layui-btn-sm layui-btn-primary">
6.   <i class="layui-icon">#x1002;</i>
7. </button>
8.

```

温馨提示：各种图标字体请移步文档左侧【页面元素 - 图标】阅览

按钮组



将按钮放入一个`class="layui-btn-group"`元素中，即可形成按钮组，按钮本身仍然可以随意搭配

```

1. <div class="layui-btn-group">
2.   <button type="button" class="layui-btn">增加</button>
3.   <button type="button" class="layui-btn">编辑</button>
4.   <button type="button" class="layui-btn">删除</button>
5. </div>
6.
7. <div class="layui-btn-group">
8.   <button type="button" class="layui-btn layui-btn-sm">
9.     <i class="layui-icon">#xe654;</i>
10.  </button>
11.   <button type="button" class="layui-btn layui-btn-sm">
12.     <i class="layui-icon">#xe642;</i>
13.   </button>
14.   <button type="button" class="layui-btn layui-btn-sm">
15.     <i class="layui-icon">#xe640;</i>
16.   </button>
17.   <button type="button" class="layui-btn layui-btn-sm">

```

```

18.     <i class="layui-icon">&#xe602;</i>
19.   </button>
20. </div>
21.
22. <div class="layui-btn-group">
23.   <button type="button" class="layui-btn layui-btn-primary layui-btn-sm">
24.     <i class="layui-icon">&#xe654;</i>
25.   </button>
26.   <button type="button" class="layui-btn layui-btn-primary layui-btn-sm">
27.     <i class="layui-icon">&#xe642;</i>
28.   </button>
29.   <button type="button" class="layui-btn layui-btn-primary layui-btn-sm">
30.     <i class="layui-icon">&#xe640;</i>
31.   </button>
32. </div>
33.

```

按钮容器

尽管按钮在同节点并排时会自动拉开间距，但在按钮太多的情况，效果并不是很美好。因为你需要用到按钮容器



```

1. <div class="layui-btn-container">
2.   <button type="button" class="layui-btn">按钮一</button>
3.   <button type="button" class="layui-btn">按钮二</button>
4.   <button type="button" class="layui-btn">按钮三</button>
5. </div>
6.

```

结语

你是否发现，主题、尺寸、图标、圆角的交叉组合，可以形成难以计算的按钮种类。另外，你可能最关注的是配色，Layui内置的六种主题的按钮颜色都是业界常用的标准配色，如果他们仍然无法与你的网站契合，那么请先允许我“噗”一声。。。然后你就大胆地自撷一个颜色吧！比如：粉红色或者菊花色（一个有味道的颜色）

表单 - 页面元素

在一个容器中设定 `class="layui-form"` 来标识一个表单元素块，通过规范好的HTML结构及CSS类，来组装成各式各样的表单元素，并通过内置的 `form`模块 来完成各种交互。

依赖加载模块：`form`（请注意：如果不加载form模块，select、checkbox、radio等将无法显示，并且无法使用form相关功能）

小睹为快

输入框

请输入标题

密码框

请输入密码

辅助文字

选择框

请选择

复选框

写作

阅读

发呆

开关

OFF

单选框

男

女

文本域

请输入内容

立即提交

重置

通过上述的小小演示，你已经大致了解了一波 `layui` 的表单模块，你可能会觉得她还算不错，但并不太过瘾？譬如你希望看到日期选择、图片上传等等。然而你必须认识到，本篇文档核心介绍的是表单元素，对于日期、上传等更多丰富的元素，其实也是可以很方便地穿插在内的。

下述是【小睹为快】的HTML结构：

```
1. <form class="layui-form" action="">
2.   <div class="layui-form-item">
3.     <label class="layui-form-label">输入框</label>
4.     <div class="layui-input-block">
5.       <input type="text" name="title" required lay-verify="required"
6.         placeholder="请输入标题" autocomplete="off" class="layui-input">
7.     </div>
```



```

7.     </div>
8.     <div class="layui-form-item">
9.         <label class="layui-form-label">密码框</label>
10.        <div class="layui-input-inline">
11.            <input type="password" name="password" required lay-verify="required"
placeholder="请输入密码" autocomplete="off" class="layui-input">
12.        </div>
13.        <div class="layui-form-mid layui-word-aux">辅助文字</div>
14.    </div>
15.    <div class="layui-form-item">
16.        <label class="layui-form-label">选择框</label>
17.        <div class="layui-input-block">
18.            <select name="city" lay-verify="required">
19.                <option value=""></option>
20.                <option value="0">北京</option>
21.                <option value="1">上海</option>
22.                <option value="2">广州</option>
23.                <option value="3">深圳</option>
24.                <option value="4">杭州</option>
25.            </select>
26.        </div>
27.    </div>
28.    <div class="layui-form-item">
29.        <label class="layui-form-label">复选框</label>
30.        <div class="layui-input-block">
31.            <input type="checkbox" name="like[write]" title="写作">
32.            <input type="checkbox" name="like[read]" title="阅读" checked>
33.            <input type="checkbox" name="like[dai]" title="发呆">
34.        </div>
35.    </div>
36.    <div class="layui-form-item">
37.        <label class="layui-form-label">开关</label>
38.        <div class="layui-input-block">
39.            <input type="checkbox" name="switch" lay-skin="switch">
40.        </div>
41.    </div>
42.    <div class="layui-form-item">
43.        <label class="layui-form-label">单选框</label>
44.        <div class="layui-input-block">
45.            <input type="radio" name="sex" value="男" title="男">
46.            <input type="radio" name="sex" value="女" title="女" checked>
47.        </div>
48.    </div>
49.    <div class="layui-form-item layui-form-text">
50.        <label class="layui-form-label">文本域</label>
51.        <div class="layui-input-block">
52.            <textarea name="desc" placeholder="请输入内容" class="layui-textarea">
</textarea>
53.        </div>
54.    </div>
55.    <div class="layui-form-item">
56.        <div class="layui-input-block">
57.            <button class="layui-btn" lay-submit lay-filter="formDemo">立即提交
</button>

```



```

58.     <button type="reset" class="layui-btn layui-btn-primary">重置</button>
59.     </div>
60. </div>
61. </form>
62.
63. <script>
64. //Demo
65. layui.use('form', function(){
66.     var form = layui.form;
67.
68.     //监听提交
69.     form.on('submit(formDemo)', function(data){
70.         layer.msg(JSON.stringify(data.field));
71.         return false;
72.     });
73. });
74. </script>
75.

```

UI的最终呈现得益于 Form模块 的全自动渲染，她将原本普通的诸如select、checkbox、radio等元素重置为你所看到的模样。或许你可以移步左侧导航的内置模块 中的 表单 对其进行详细的了解。

而本篇介绍的是表单元素本身，譬如规定的区块、CSS类、原始控件等。他们共同组成了一个表单体系。

下述是基本的行区块结构，它提供了响应式的支持。但如果你不大喜欢，你可以换成你的结构，但必须要在外层容器中定义`class="layui-form"`，form模块才能正常工作。

```

1. <div class="layui-form-item">
2.   <label class="layui-form-label">标签区域</label>
3.   <div class="layui-input-block">
4.     原始表单元素区域
5.   </div>
6. </div>
7.

```

输入框

请输入标题

```

1. <input type="text" name="title" required lay-verify="required" placeholder="请
   输入标题" autocomplete="off" class="layui-input">
2.

```

required: 注册浏览器所规定的必填字段

lay-verify: 注册form模块需要验证的类型

class="layui-input": layui.css提供的通用CSS类

这些在下文都不再做重复介绍

下拉选择框

请选择一个城市 ▼	杭州 ▼	select分组 ▼	带搜索的选择框 ▼
-----------	------	------------	-----------

```

1. <select name="city" lay-verify="">
2.   <option value="">请选择一个城市</option>
3.   <option value="010">北京</option>
4.   <option value="021">上海</option>
5.   <option value="0571">杭州</option>
6. </select>
7.
```

上述option的第一项主要是占个坑，让form模块预留“请选择”的提示空间，否则将会把第一项（存在value值）作为默认选中项。你可以在option的空值项中自定义文本，如：请选择分类。

你可以通过设定 *selected* 来设定默认选中项：

```

1. <select name="city" lay-verify="">
2.   <option value="010">北京</option>
3.   <option value="021" disabled>上海（禁用效果）</option>
4.   <option value="0571" selected>杭州</option>
5. </select>
6.
```

你还可以通过 *optgroup* 标签给select分组：

```

1. <select name="quiz">
2.   <option value="">请选择</option>
3.   <optgroup label="城市记忆">
4.     <option value="你工作的第一个城市">你工作的第一个城市？</option>
5.   </optgroup>
6.   <optgroup label="学生时代">
7.     <option value="你的工号">你的工号？</option>
8.     <option value="你最喜欢的老师">你最喜欢的老师？</option>
9.   </optgroup>
10. </select>
11.
```

以及通过设定属性 *lay-search* 来开启搜索匹配功能

```

1. <select name="city" lay-verify="" lay-search>
2.   <option value="010">layer</option>
3.   <option value="021">form</option>
4.   <option value="0571" selected>layim</option>
5.   .....
6. </select>
7.

```

属性*selected*可设定默认项
属性*disabled*开启禁用，select和option标签都支持

复选框



```

1. 默认风格：
2. <input type="checkbox" name="" title="写作" checked>
3. <input type="checkbox" name="" title="发呆">
4. <input type="checkbox" name="" title="禁用" disabled>
5.
6. 原始风格：
7. <input type="checkbox" name="" title="写作" lay-skin="primary" checked>
8. <input type="checkbox" name="" title="发呆" lay-skin="primary">
9. <input type="checkbox" name="" title="禁用" lay-skin="primary" disabled>
10.

```

属性*title*可自定义文本（温馨提示：如果只想显示复选框，可以不用设置*title*）

属性*checked*可设定默认选中

属性*lay-skin*可设置复选框的风格

设置*value="1"*可自定义值，否则选中时返回的就是默认的on

开关



其实就是checkbox复选框的“变种”，通过设定 *lay-skin="switch"* 形成了开关风格

```

1. <input type="checkbox" name="xxx" lay-skin="switch">
2. <input type="checkbox" name="yyy" lay-skin="switch" lay-text="ON|OFF" checked>
3. <input type="checkbox" name="zzz" lay-skin="switch" lay-text="开启|关闭">
4. <input type="checkbox" name="aaa" lay-skin="switch" disabled>

```

5.

属性`checked`可设定默认开

属性`disabled`开启禁用

属性`lay-text`可自定义开关两种状态的文本

设置`value="1"`可自定义值，否则选中时返回的就是默认的on

单选框

☐ 男 ☒ 女 ☐ 中性

```
1. <input type="radio" name="sex" value="nan" title="男">
2. <input type="radio" name="sex" value="nv" title="女" checked>
3. <input type="radio" name="sex" value="" title="中性" disabled>
4.
```

属性`title`可自定义文本

属性`disabled`开启禁用

设置`value="xxx"`可自定义值，否则选中时返回的就是默认的on

文本域

请输入

```
1. <textarea name="" required lay-verify="required" placeholder="请输入"
   class="layui-textarea"></textarea>
2.
```

`class="layui-textarea"`: layui.css提供的通用CSS类

组装行内表单

范围

¥

-

¥

密码

```
1. <div class="layui-form-item">
2.
```

```

3.     <div class="layui-inline">
4.         <label class="layui-form-label">范围</label>
5.         <div class="layui-input-inline" style="width: 100px;">
6.             <input type="text" name="price_min" placeholder="¥" autocomplete="off"
class="layui-input">
7.         </div>
8.         <div class="layui-form-mid">-</div>
9.         <div class="layui-input-inline" style="width: 100px;">
10.            <input type="text" name="price_max" placeholder="¥" autocomplete="off"
class="layui-input">
11.        </div>
12.    </div>
13.
14.    <div class="layui-inline">
15.        <label class="layui-form-label">密码</label>
16.        <div class="layui-input-inline" style="width: 100px;">
17.            <input type="password" name="" autocomplete="off" class="layui-input">
18.        </div>
19.    </div>
20.
21. </div>
22.

```

`class="layui-inline"`: 定义外层行内

`class="layui-input-inline"`: 定义内层行内

忽略美化渲染

你可以对表单元素增加属性 `lay-ignore` 设置后，将不会对该标签进行美化渲染，即保留系统风格，比如：

```

1. <select lay-ignore>
2.     <option>...</option>
3. </select>
4.

```

一般不推荐这样做。事实上form组件所提供的接口，对其渲染过的元素，足以应付几乎所有的业务需求。如果忽略渲染，可能会让UI风格不和谐

表单方框风格

输入框	请输入标题	
密码框	请输入密码	辅助文字
选择框	请选择	
开关	<input type="checkbox"/>	
单选框	<input type="radio"/> 男 <input checked="" type="radio"/> 女	
文本域		
请输入内容		
<div>立即提交</div>		

通过追加 `layui-form-pane` 的class，来设定表单的方框风格。内部结构不变。我们的F1y社区用的就是这个风格。

```
1. <form class="layui-form layui-form-pane" action="">
2.   内部结构都一样，值得注意的是 复选框/开关/单选框 这些组合在该风格下需要额外添加 pane属性
   (否则会看起来比较别扭)，如：
3.   <div class="layui-form-item" pane>
4.     <label class="layui-form-label">单选框</label>
5.     <div class="layui-input-block">
6.       <input type="radio" name="sex" value="男" title="男">
7.       <input type="radio" name="sex" value="女" title="女" checked>
8.     </div>
9.   </div>
10. </form>
11.
```

结语


Layui版本稳定后，会抽空推出一个表单元素生成工具，这样似乎就更方便地组装你的表单了呀。

导航相关 - 页面元素

导航一般指页面引导性频道集合，多以菜单的形式呈现，可应用于头部和侧边，是整个网页画龙点睛般的存在。面包屑结构简单，支持自定义分隔符。千万不要忘了加载 `element` 模块。虽然大部分行为都是在加载完该模块后自动完成的，但一些交互操作，如呼出二级菜单等，需借助 `element` 模块才能使用。你可以移步文档左侧【内置模块 - 常用元素操作 `element`】了解详情

依赖加载模块：`element`

水平导航


A horizontal navigation bar with a dark background. It contains five items: '最新活动' (Latest Activities), '产品' (Products), '大数据' (Big Data), '解决方案' (Solutions) with a dropdown arrow, and '社区' (Community). The '产品' item is highlighted with a green underline.

```
1. <ul class="layui-nav" lay-filter="">
2.   <li class="layui-nav-item"><a href="">最新活动</a></li>
3.   <li class="layui-nav-item layui-this"><a href="">产品</a></li>
4.   <li class="layui-nav-item"><a href="">大数据</a></li>
5.   <li class="layui-nav-item">
6.     <a href="javascript:;">解决方案</a>
7.     <dl class="layui-nav-child"> <!-- 二级菜单 -->
8.       <dd><a href="">移动模块</a></dd>
9.       <dd><a href="">后台模版</a></dd>
10.      <dd><a href="">电商平台</a></dd>
11.    </dl>
12.  </li>
13.  <li class="layui-nav-item"><a href="">社区</a></li>
14. </ul>
15.
16. <script>
17. //注意：导航 依赖 element 模块，否则无法进行功能性操作
18. layui.use('element', function(){
19.   var element = layui.element;
20.
21.   //...
22. });
23. </script>
24.
```

设定 `layui-this` 来指向当前页面分类。

导航中的其它元素

除了一般的文字导航，我们还内置了图片和徽章的支持，如：

控制台 9 个人中心 ●  我 ▼

```

1. <ul class="layui-nav">
2.   <li class="layui-nav-item">
3.     <a href="">控制台<span class="layui-badge">9</span></a>
4.   </li>
5.   <li class="layui-nav-item">
6.     <a href="">个人中心<span class="layui-badge-dot"></span></a>
7.   </li>
8.   <li class="layui-nav-item">
9.     <a href="">我</a>
10.    <dl class="layui-nav-child">
11.      <dd><a href="javascript:;">修改信息</a></dd>
12.      <dd><a href="javascript:;">安全管理</a></dd>
13.      <dd><a href="javascript:;">退了</a></dd>
14.    </dl>
15.  </li>
16. </ul>
17.

```

是否瞬间上了个档次呢？

导航主题

藏青导航 产品 大数据 解决方案 ▼ 社区

墨绿导航 产品 大数据 解决方案 ▼ 社区

艳蓝导航 产品 大数据 解决方案 ▼ 社区

通过对导航追加CSS背景类，让导航呈现不同的主题色

```

1. //如定义一个墨绿背景色的导航
2. <ul class="layui-nav layui-bg-green" lay-filter="">
3.   ...
4. </ul>
5.

```

水平导航支持的其它背景主题有：*layui-bg-cyan*（藏青）、*layui-bg-molv*（墨绿）、*layui-bg-blue*（艳蓝）

垂直导航支持的其它背景主题有：*layui-bg-cyan*（藏青）

垂直/侧边导航



```

1. <ul class="layui-nav layui-nav-tree" lay-filter="test">
2. <!-- 侧边导航: <ul class="layui-nav layui-nav-tree layui-nav-side"> -->
3.   <li class="layui-nav-item layui-nav-itemed">
4.     <a href="javascript:;">默认展开</a>
5.     <dl class="layui-nav-child">
6.       <dd><a href="javascript:;">选项1</a></dd>
7.       <dd><a href="javascript:;">选项2</a></dd>
8.       <dd><a href="">跳转</a></dd>
9.     </dl>
10.  </li>
11.  <li class="layui-nav-item">
12.    <a href="javascript:;">解决方案</a>
13.    <dl class="layui-nav-child">
14.      <dd><a href="">移动模块</a></dd>
15.      <dd><a href="">后台模版</a></dd>
16.      <dd><a href="">电商平台</a></dd>
17.    </dl>
18.  </li>
19.  <li class="layui-nav-item"><a href="">产品</a></li>
20.  <li class="layui-nav-item"><a href="">大数据</a></li>
21. </ul>
22.

```

水平、垂直、侧边三个导航的HTML结构是完全一样的，不同的是：

垂直导航需要追加class: *layui-nav-tree*

侧边导航需要追加class: *layui-nav-tree layui-nav-side*

导航可选属性

对导航元素结构设定可选属性，可让导航菜单项达到不同效果。目前支持的属性如下：

属性名	可选值	说明
lay-shrink	<ul style="list-style-type: none">空值（默认） 不收缩兄弟菜单子菜单all 收缩全部兄弟菜单子菜单	展开子菜单时，是否收缩兄弟节点已展开的子菜单（注：layui 2.2.6 开始新增） 如：<ul class="layui-nav layui-nav-tree" lay-shrink="all"> ...
lay-unselect	无需填值	点击指定导航菜单时，不会出现选中效果（注：layui 2.2.0 开始新增） 如：<li class="layui-nav-item" lay-unselect>刷新

面包屑

首页 / 国际新闻 / 亚太地区 / 正文

```
1. <span class="layui-breadcrumb">
2.   <a href="">首页</a>
3.   <a href="">国际新闻</a>
4.   <a href="">亚太地区</a>
5.   <a><cite>正文</cite></a>
6. </span>
7.
```

你还可以通过设置属性 `lay-separator="-"` 来自定义分隔符。如： [首页](#) [国际新闻](#) [亚太地区](#) [正文](#)

```
1. <span class="layui-breadcrumb" lay-separator="-">
2.   <a href="">首页</a>
3.   <a href="">国际新闻</a>
4.   <a href="">亚太地区</a>
5.   <a><cite>正文</cite></a>
6. </span>
7.
```

当然，你还可以作为小导航来用，如：

娱乐 | 八卦 | 体育 | 搞笑 | 视频 | 游戏 | 综艺

```
1. <span class="layui-breadcrumb" lay-separator="|">
2.   <a href="">娱乐</a>
3.   <a href="">八卦</a>
4.   <a href="">体育</a>
```

```
5.     <a href="">搞笑</a>
6.     <a href="">视频</a>
7.     <a href="">游戏</a>
8.     <a href="">综艺</a>
9. </span>
10.
```

layui - 用心与你沟通


Tab选项卡 - 页面元素

导航菜单可应用于头部和侧边，Tab选项卡提供多套风格，支持响应式，支持删除选项卡等功能。面包屑结构简单，支持自定义分隔符。

依赖加载组件：[element](#)（请注意：必须加载element模块，相关功能才能正常使用，详见：[内置组件 - 常用元素操作](#)）

默认Tab选项卡

Tab广泛应用于Web页面，因此我们也对其进行了良好的支持。Layui内置多种Tab风格，支持删除选项卡、并提供响应式支持。这是一个最基本的例子：



如果需要对Tab进行外部新增、删除、切换等操作，请移步到“[内置组件-常用元素操作](#)”页面中查阅：基础方法

```
1. <div class="layui-tab">
2.   <ul class="layui-tab-title">
3.     <li class="layui-this">网站设置</li>
4.     <li>用户管理</li>
5.     <li>权限分配</li>
6.     <li>商品管理</li>
7.     <li>订单管理</li>
8.   </ul>
9.   <div class="layui-tab-content">
10.    <div class="layui-tab-item layui-show">内容1</div>
11.    <div class="layui-tab-item">内容2</div>
12.    <div class="layui-tab-item">内容3</div>
13.    <div class="layui-tab-item">内容4</div>
14.    <div class="layui-tab-item">内容5</div>
15.  </div>
16. </div>
17.
18. <script>
19.   //注意：选项卡 依赖 element 模块，否则无法进行功能性操作
20.   layui.use('element', function(){
21.     var element = layui.element;
22.
23.     //...
24.   });
25. </script>
26.
```

Tab简洁风格

网站设置

用户管理

权限分配

商品管理

订单管理

```

1. <div class="layui-tab layui-tab-brief" lay-filter="docDemoTabBrief">
2.   <ul class="layui-tab-title">
3.     <li class="layui-this">网站设置</li>
4.     <li>用户管理</li>
5.     <li>权限分配</li>
6.     <li>商品管理</li>
7.     <li>订单管理</li>
8.   </ul>
9.   <div class="layui-tab-content"></div>
10. </div>
11.

```

通过追加class: *layui-tab-brief* 来设定简洁风格。

值得注意的是，如果存在 *layui-tab-item* 的内容区域，在切换时自动定位到对应内容。如果不存在内容区域，则不会定位到对应内容。你通常需要设置过滤器，通过 *element* 模块的监听tab事件来进行切换操作。详见文档左侧【内置组件 - 基本元素操作 element】

Tab卡片风格

网站设置

用户管理

权限分配

商品管理

订单管理

1

```

1. <div class="layui-tab layui-tab-card">
2.   <ul class="layui-tab-title">
3.     <li class="layui-this">网站设置</li>
4.     <li>用户管理</li>
5.     <li>权限分配</li>
6.     <li>商品管理</li>
7.     <li>订单管理</li>
8.   </ul>
9.   <div class="layui-tab-content" style="height: 100px;">
10.    <div class="layui-tab-item layui-show">1</div>
11.    <div class="layui-tab-item">2</div>
12.    <div class="layui-tab-item">3</div>
13.    <div class="layui-tab-item">4</div>
14.    <div class="layui-tab-item">5</div>

```

```

15.     <div class="layui-tab-item">6</div>
16.     </div>
17. </div>
18.

```

通过追加class: `layui-tab-card`来设定卡片风格

Tab响应式

当容器的宽度不足以显示全部的选项时，即会自动出现展开图标，如下以卡片风格为例（注意：所有Tab风格都支持响应式）：



额，感觉像是打了个小酱油。而事实上在自适应的页面中（不固宽），它的意义才会呈现。

带删除的Tab

你可以对父层容器设置属性 `lay-allowClose="true"` 来允许Tab选项卡被删除



```

1. <div class="layui-tab" lay-allowClose="true">
2.   <ul class="layui-tab-title">
3.     <li class="layui-this">网站设置</li>
4.     <li>用户基本管理</li>
5.     <li>权限分配</li>
6.     <li>全部历史商品管理文字长一点试试</li>
7.     <li>订单管理</li>
8.   </ul>
9.   <div class="layui-tab-content">
10.    <div class="layui-tab-item layui-show">1</div>
11.    <div class="layui-tab-item">2</div>
12.    <div class="layui-tab-item">3</div>
13.    <div class="layui-tab-item">4</div>

```

```

14.     <div class="layui-tab-item">5</div>
15.     <div class="layui-tab-item">6</div>
16.   </div>
17. </div>
18.

```

与默认相比没有什么特别的结构，就是多了个属性 `lay-allowClose="true"`

ID焦点定位

你可以通过对选项卡设置属性 `lay-id="xxx"` 来作为唯一的匹配索引，以用于外部的定位切换，如后台的左侧导航、以及页面地址 hash 的匹配。

```

1. <div class="layui-tab" lay-filter="test1">
2.   <ul class="layui-tab-title">
3.     <li class="layui-this" lay-id="111" >文章列表</li>
4.     <li lay-id="222">发送信息</li>
5.     <li lay-id="333">权限分配</li>
6.     <li lay-id="444">审核</li>
7.     <li lay-id="555">订单管理</li>
8.   </ul>
9.   <div class="layui-tab-content">
10.    <div class="layui-tab-item layui-show">1</div>
11.    <div class="layui-tab-item">2</div>
12.    <div class="layui-tab-item">3</div>
13.    <div class="layui-tab-item">4</div>
14.    <div class="layui-tab-item">5</div>
15.  </div>
16. </div>
17.

```

属性 `lay-id` 是扮演这项功能的主要角色，它是动态操作的重要凭据，如：

```

1. <script>
2. layui.use('element', function(){
3.   var element = layui.element;
4.
5.   //获取hash来切换选项卡，假设当前地址的hash为lay-id对应的值
6.   var layid = location.hash.replace(/^#test1=/, '');
7.   element.tabChange('test1', layid); //假设当前地址为：http://a.com#test1=222，那
   么选项卡会自动切换到“发送消息”这一项
8.
9.   //监听Tab切换，以改变地址hash值
10.  element.on('tab(test1)', function(){
11.    location.hash = 'test1='+ this.getAttribute('lay-id');
12.  });
13. });
14. </script>
15.

```

同样的还有增加选项卡和删除选项卡，都需要用到 `lay-id`，更多动态操作请阅

读：element 模块

提示

无论是导航、还是Tab，都需依赖 *element* 模块，大部分行为都是在加载完该模块后自动完成的，但一些交互操作，如Tab事件监听等，需按照场景选择性使用。你可以移步文档左侧【内置组件 - 基本元素操作 element】了解详情

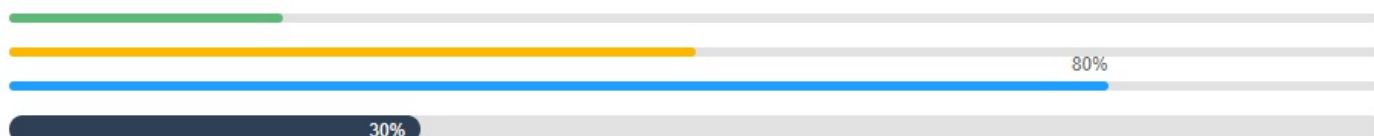
layui - 用心与你沟通

进度条 - 页面元素

进度条可应用于许多业务场景，如任务完成进度、loading等等，是一种较为直观的表达元素。

依赖加载组件：[element](#)

常规用法



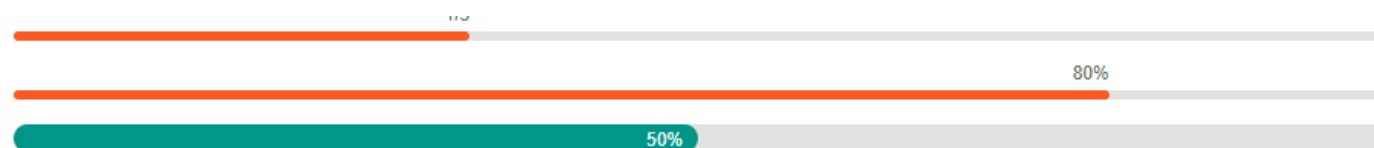
我们进度条提供了两种尺寸及多种颜色的显示风格，其中颜色的选值可参考：[背景色公共类](#)。基本元素结构如下

```
1. <div class="layui-progress">
2.   <div class="layui-progress-bar" lay-percent="10%"></div>
3. </div>
4.
5. <script>
6. //注意进度条依赖 element 模块，否则无法进行正常渲染和功能性操作
7. layui.use('element', function(){
8.   var element = layui.element;
9. });
10. </script>
11.
```

属性 `lay-percent`：代表进度条的初始百分比，你也可以动态改变进度，详见：[进度条的动态操作](#)

正如上述你见到的，当对元素设置了class为 `layui-progress-big` 时，即为大尺寸的进度条风格。默认风格的进度条的百分比如果开启，会在右上角显示，而大号进度条则会在内部显示。

显示进度比文本



通过对父级元素设置属性 `lay-showPercent="yes"` 来开启进度比的文本显示，支持：普通数字、百分数、分数（layui 2.1.7 新增）

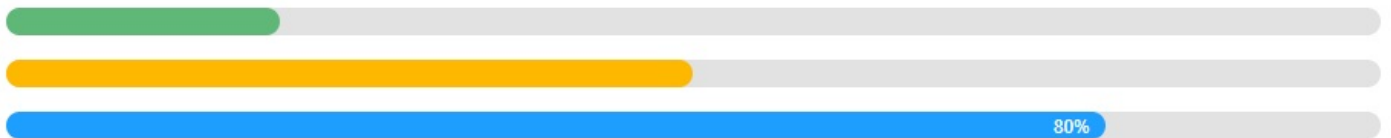
```

1. <div class="layui-progress" lay-showPercent="true">
2.   <div class="layui-progress-bar layui-bg-red" lay-percent="1/3"></div>
3. </div>
4.
5. <div class="layui-progress" lay-showPercent="yes">
6.   <div class="layui-progress-bar layui-bg-red" lay-percent="30%"></div>
7. </div>
8.
9. <div class="layui-progress layui-progress-big" lay-showPercent="yes">
10.   <div class="layui-progress-bar layui-bg-green" lay-percent="50%"></div>
11. </div>
12.

```

注意：默认情况下不会显示百分比文本，如果你想开启，对属性lay-showPercent设置任意值即可，如：yes。但如果不显示，千万别设置no或者false，直接剔除该属性即可。

大号进度条



如果短小细长的它不大适合追求激情与视觉冲击的你，那么你可以选择大而粗，尽情地销魂于活塞运动。研究表明：上述尺寸刚刚好。

```

1. <div class="layui-progress layui-progress-big">
2.   <div class="layui-progress-bar" lay-percent="20%"></div>
3. </div>
4.
5. <div class="layui-progress layui-progress-big">
6.   <div class="layui-progress-bar layui-bg-orange" lay-percent="50%"></div>
7. </div>
8.
9. <div class="layui-progress layui-progress-big" lay-showPercent="true">
10.   <div class="layui-progress-bar layui-bg-blue" lay-percent="80%"></div>
11. </div>
12.

```

正如上述你见到的，当对元素设置了class为 layui-progress-big 时，即为大尺寸的进度条风格。默认风格的进度条的百分比如果开启，会在右上角显示，而大号进度条则会在内部显示。

如果你需要对进度条进行动态操作，如动态改变进度，那么你需要阅读：[element 模块](#)

面板 - 页面元素

一般的面板通常是指一个独立的容器，而折叠面板则能有效地节省页面的可视面积，非常适合应用于：QA说明、帮助文档等

依赖加载组件：[element](#)

卡片面板

卡片面板

卡片式面板面板通常用于非白色背景色的主体内
从而映衬出边框投影

卡片面板

结合 layui 的栅格系统
轻松实现响应式布局

```
1. <div class="layui-card">
2.   <div class="layui-card-header">卡片面板</div>
3.   <div class="layui-card-body">
4.     卡片式面板面板通常用于非白色背景色的主体内<br>
5.     从而映衬出边框投影
6.   </div>
7. </div>
8.
```

如果你的网页采用的是默认的白色背景，不建议使用卡片面板。

折叠面板

▼ 杜甫

杜甫的思想核心是儒家的仁政思想，他有“致君尧舜上，再使风俗淳”的宏伟抱负。杜甫虽然在世时名声并不显赫，但后来声名远播，对中国文学和日本文学都产生了深远的影响。杜甫共有约1500首诗歌被保留了下来，大多集于《杜工部集》。

▼ 李清照

李清照出生于书香门第，早期生活优裕，其父李格非藏书甚富，她小时候就在良好的家庭环境中打下文学基础。出嫁后与夫赵明诚共同致力于书画金石的搜集整理。金兵入据中原时，流寓南方，境遇孤苦。所作词，前期多写其悠闲生活，后期多悲叹身世，情调感伤。形式上善用白描手法，自辟途径，语言清丽。

> 鲁迅

通过对内容元素设置class为 `layui-show` 来选择性初始展开某一个面板，`element` 模块会自动呈现状态图标。

```

1. <div class="layui-collapse">
2.   <div class="layui-colla-item">
3.     <h2 class="layui-colla-title">杜甫</h2>
4.     <div class="layui-colla-content layui-show">内容区域</div>
5.   </div>
6.   <div class="layui-colla-item">
7.     <h2 class="layui-colla-title">李清照</h2>
8.     <div class="layui-colla-content layui-show">内容区域</div>
9.   </div>
10.  <div class="layui-colla-item">
11.    <h2 class="layui-colla-title">鲁迅</h2>
12.    <div class="layui-colla-content layui-show">内容区域</div>
13.  </div>
14. </div>
15.
16. <script>
17.  //注意：折叠面板 依赖 element 模块，否则无法进行功能性操作
18.  layui.use('element', function(){
19.    var element = layui.element;
20.
21.    //...
22.  });
23. </script>
24.

```

开启手风琴

▼ 杜甫

杜甫的思想核心是儒家的仁政思想，他有“致君尧舜上，再使风俗淳”的宏伟抱负。杜甫虽然在世时名声并不显赫，但后来声名远播，对中国文学和日本文学都产生了深远的影响。杜甫共有约1500首诗歌被保留了下来，大多集于《杜工部集》。

› 李清照

› 鲁迅

在折叠面板的父容器设置属性 `lay-accordion` 来开启手风琴，那么在进行折叠操作时，始终只会展现当前的面板。

```

1. <div class="layui-collapse" lay-accordion>
2.   <div class="layui-colla-item">
3.     <h2 class="layui-colla-title">杜甫</h2>
4.     <div class="layui-colla-content layui-show">内容区域</div>
5.   </div>
6.   <div class="layui-colla-item">
7.     <h2 class="layui-colla-title">李清照</h2>
8.     <div class="layui-colla-content layui-show">内容区域</div>
9.   </div>
10.  <div class="layui-colla-item">

```

```
11.     <h2 class="layui-colla-title">鲁迅</h2>
12.     <div class="layui-colla-content layui-show">内容区域</div>
13.   </div>
14. </div>
15.
```

layui - 用心与你沟通

表格 - 页面元素

本篇为你介绍表格的HTML使用，你将通过内置的自定义属性对其进行风格的多样化设定。请注意：这仅仅局限于呈现基础表格，如果你急切需要的是数据表格（*datatable*），那么你应该详细阅读：[table模块](#)

常规用法

昵称	加入时间	签名
贤心	2016-11-29	人生就像是一场修行
许闲心	2016-11-28	于千万人之中遇见你所遇见的人，于千万年之中，时间的无涯的荒野里...
sentsin	2016-11-27	Life is either a daring adventure or nothing.

```
1. <table class="layui-table">
2.   <colgroup>
3.     <col width="150">
4.     <col width="200">
5.   </col>
6. </colgroup>
7. <thead>
8.   <tr>
9.     <th>昵称</th>
10.    <th>加入时间</th>
11.    <th>签名</th>
12.  </tr>
13. </thead>
14. <tbody>
15.   <tr>
16.     <td>贤心</td>
17.     <td>2016-11-29</td>
18.     <td>人生就像是一场修行</td>
19.   </tr>
20.   <tr>
21.     <td>许闲心</td>
22.     <td>2016-11-28</td>
23.     <td>于千万人之中遇见你所遇见的人，于千万年之中，时间的无涯的荒野里...</td>
24.   </tr>
25. </tbody>
26. </table>
27.
```

基础属性

静态表格支持以下基础属性，可定义不同风格/尺寸的表格样式：

属性名	属性值	备注
lay-even	无	用于开启 隔行 背景，可与其它属性一起使用
lay-skin="属性值"	line （行边框风格） row （列边框风格） nob （无边框风格）	若使用默认风格不设置该属性即可
lay-size="属性值"	sm （小尺寸） lg （大尺寸）	若使用默认尺寸不设置该属性即可

将你所需要的基础属性写在table标签上即可，如（一个带有隔行背景，且行边框风格的大尺寸表格）：

```
1. <table lay-even lay-skin="line" lay-size="lg">
2. ...
3. </table>
4.
```

表格其它风格

除了默认的表格风格外，我们还提供了其它几种风格，你可以按照实际需求自由设定

昵称	加入时间	签名
贤心	2016-11-29	人生就像是一场修行
许闲心	2016-11-28	于千万人之中遇见你所遇见的人，于千万年之中，时间的无涯的荒野里...
sentsin	2016-11-27	Life is either a daring adventure or nothing.

昵称	加入时间	签名
贤心	2016-11-29	人生就像是一场修行
许闲心	2016-11-28	于千万人之中遇见你所遇见的人，于千万年之中，时间的无涯的荒野里...
sentsin	2016-11-27	Life is either a daring adventure or nothing.

昵称	加入时间	签名
贤心	2016-11-29	人生就像是一场修行
许闲心	2016-11-28	于千万人之中遇见你所遇见的人，于千万年之中，时间的无涯的荒野里...
sentsin	2016-11-27	Life is either a daring adventure or nothing.

```
1. <table class="layui-table" lay-skin="line">
2.   行边框表格（内部结构参见右侧目录“常规用法”）
3. </table>
4.
5. <table class="layui-table" lay-skin="row">
6.   列边框表格（内部结构参见右侧目录“常规用法”）
```



```
7. </table>
8.
9. <table class="layui-table" lay-even lay-skin="nob">
10.   无边框表格（内部结构参见右侧目录“常规用法”）
11. </table>
12.
```

表格其它尺寸

除了默认的表格尺寸外，我们还提供了其它几种尺寸，你可以按照实际需求自由设定

昵称	加入时间	签名
贤心	2016-11-29	人生就像是一场修行
许闲心	2016-11-28	于千万人之中遇见你所遇见的人，于千万年之中，时间的无涯的荒野里...
sentsin	2016-11-27	Life is either a daring adventure or nothing.

昵称	加入时间	签名
贤心	2016-11-29	人生就像是一场修行
许闲心	2016-11-28	于千万人之中遇见你所遇见的人，于千万年之中，时间的无涯的荒野里...
sentsin	2016-11-27	Life is either a daring adventure or nothing.

```
1. <table class="layui-table" lay-size="sm">
2.   小尺寸表格（内部结构参见右侧目录“常规用法”）
3. </table>
4.
5. <table class="layui-table" lay-size="lg">
6.   大尺寸表格（内部结构参见右侧目录“常规用法”）
7. </table>
8.
```

结语

再次温馨提醒：如果你需要对表格进行排序、数据交互等一系列功能性操作，你需要进一步阅读 layui 的重要组成：[table模块](#)

徽章

徽章是一个修饰性的元素，它们本身细小而并不显眼，但掺杂在其它元素中就显得尤为突出了。页面往往因徽章的陪衬，而显得十分和谐。

快速使用

不妨先来看看 徽章 这个小小的大家族吧：



你可能已经敏锐地发现，除去花枝招展的七种颜色，徽章具有三种不同的风格类型：小圆点、椭圆体、边框体

```

1.
2. 小圆点, 通过 layui-badge-dot 来定义, 里面不能加文字
3. <span class="layui-badge-dot"></span>
4. <span class="layui-badge-dot layui-bg-orange"></span>
5. <span class="layui-badge-dot layui-bg-green"></span>
6. <span class="layui-badge-dot layui-bg-cyan"></span>
7. <span class="layui-badge-dot layui-bg-blue"></span>
8. <span class="layui-badge-dot layui-bg-black"></span>
9. <span class="layui-badge-dot layui-bg-gray"></span>
10.
11. 椭圆体, 通过 layui-badge 来定义。事实上我们把这个视为主要使用方式
12. <span class="layui-badge">6</span>
13. <span class="layui-badge">99</span>
14. <span class="layui-badge">61728</span>
15.
16. <span class="layui-badge">赤</span>
17. <span class="layui-badge layui-bg-orange">橙</span>
18. <span class="layui-badge layui-bg-green">绿</span>
19. <span class="layui-badge layui-bg-cyan">青</span>
20. <span class="layui-badge layui-bg-blue">蓝</span>
21. <span class="layui-badge layui-bg-black">黑</span>
22. <span class="layui-badge layui-bg-gray">灰</span>
23.
24. 边框体, 通过 layui-badge-rim 来定义
25. <span class="layui-badge-rim">6</span>
26. <span class="layui-badge-rim">Hot</span>
27.

```

与其它元素的搭配

徽章主要是修饰作用，因此必不可少要与几乎所有的元素搭配。我们目前对以下元素内置了徽章的排版支持：

按钮



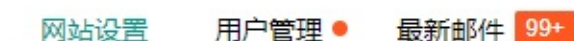
1. `<button class="layui-btn">查看消息1</button>`
2. `<button class="layui-btn">动态</button>`
- 3.

导航



1. `<ul class="layui-nav" style="text-align: right;"> <-- 小Tips : 这里有没有发现, 设置导航靠右对齐 (或居中对齐) 其实非常简单 -->`
2. `<li class="layui-nav-item">`
3. `控制台9`
4. ``
5. `<li class="layui-nav-item">`
6. `个人中心`
7. ``
8. ``
- 9.

选项卡（所有风格都支持，这里以简约风格为例）



1. `<div class="layui-tab layui-tab-brief">`
2. `<ul class="layui-tab-title">`
3. `<li class="layui-this">网站设置`
4. `用户管理`
5. `最新邮件99+`
6. ``
7. `<div class="layui-tab-content"></div>`
8. `</div>`
- 9.

而至于与其它更多元素的搭配，就由你自由把控吧！

结语

其实，在与其它元素的搭配中，你要做的，无非就是合理运用这几点：边距 背景色，徽章必然大显神威！

layui - 用心与你沟通

时间线

将时间抽象到二维平面，垂直呈现一段从过去到现在的故事。

快速使用

8月18日

layui 2.0 的一切准备工作似乎都已到位。发布之弦，一触即发。
不枉近百个日日夜夜与之相伴。因小而大，因弱而强。
无论它能走多远，抑或如何支撑？至少我曾倾注全心，无怨无悔 😊

8月16日

杜甫的思想核心是儒家的仁政思想，他有“*致君尧舜上，再使风俗淳*”的宏伟抱负。个人最爱的名篇有：

- 《登高》
- 《茅屋为秋风所破歌》

8月15日

中国人民抗日战争胜利日
常常在想，尽管对这个国家有这样那样的抱怨，但我们可能的确生在了最好的时代
铭记、感恩
所有为中华民族浴血奋战的英雄将士
永垂不朽

过去

```

1. <ul class="layui-timeline">
2.   <li class="layui-timeline-item">
3.     <i class="layui-icon layui-timeline-axis">&#xe63f;</i>
4.     <div class="layui-timeline-content layui-text">
5.       <h3 class="layui-timeline-title">8月18日</h3>
6.       <p>
7.         layui 2.0 的一切准备工作似乎都已到位。发布之弦，一触即发。
8.         <br>不枉近百个日日夜夜与之相伴。因小而大，因弱而强。
9.         <br>无论它能走多远，抑或如何支撑？至少我曾倾注全心，无怨无悔 <i class="layui-
icon"></i>
10.       </p>
11.     </div>
12.   </li>
13.   <li class="layui-timeline-item">
14.     <i class="layui-icon layui-timeline-axis">&#xe63f;</i>
15.     <div class="layui-timeline-content layui-text">
16.       <h3 class="layui-timeline-title">8月16日</h3>
17.       <p>杜甫的思想核心是儒家的仁政思想，他有“<em>致君尧舜上，再使风俗淳</em>”的宏伟抱负。

```

个人最爱的名篇有：

```

18.         <ul>
19.             <li>《登高》</li>
20.             <li>《茅屋为秋风所破歌》</li>
21.         </ul>
22.     </div>
23. </li>
24. <li class="layui-timeline-item">
25.     <i class="layui-icon layui-timeline-axis">&#xe63f;</i>
26.     <div class="layui-timeline-content layui-text">
27.         <h3 class="layui-timeline-title">8月15日</h3>
28.         <p>
29.             中国人民抗日战争胜利72周年
30.             <br>常常在想，尽管对这个国家有这样那样的抱怨，但我们的确生在了最好的时代
31.             <br>铭记、感恩
32.             <br>所有为中华民族浴血奋战的英雄将士
33.             <br>永垂不朽
34.         </p>
35.     </div>
36. </li>
37. <li class="layui-timeline-item">
38.     <i class="layui-icon layui-timeline-axis">&#xe63f;</i>
39.     <div class="layui-timeline-content layui-text">
40.         <div class="layui-timeline-title">过去</div>
41.     </div>
42. </li>
43. </ul>
44.

```

关于时间线，似乎也没有什么太多可介绍的东西。你需要留意的是以下几点

- 图标可以任意定义（但并不推荐更改）
- 标题区域并不意味着一定要加粗
- 内容区域可自由填充。

结语

授之以鱼不如授之以渔，时间线怎么用，就看你的了。

layui - 用心与你沟通

简单辅助元素 - 页面元素

本篇主要集中罗列页面中的一些简单辅助元素，如：引用块、字段集区块、横线等等，这些元素都无需依赖任何模块

引用区块

引用区域的文字

引用区域的文字

目前内置了上述两种风格

1. `<blockquote class="layui-elem-quote">引用区域的文字</blockquote>`
2. `<blockquote class="layui-elem-quote layui-quote-nm">引用区域的文字</blockquote>`
- 3.

字段集区块

字段集区块 - 默认风格

内容区域。

同样内置了两种风格，另一种风格即该文档的标题横线：字段集一行

1. `<fieldset class="layui-elem-field">`
2. `<legend>字段集区块 - 默认风格</legend>`
3. `<div class="layui-field-box">`
4. 内容区域
5. `</div>`
6. `</fieldset>`
- 7.
8. `<fieldset class="layui-elem-field layui-field-title">`
9. `<legend>字段集区块 - 横线风格</legend>`
10. `<div class="layui-field-box">`
11. 内容区域
12. `</div>`
13. `</fieldset>`
14. 你可以把它看作是一个有标题的横线
- 15.

横线

默认分割线

赤色分割线

橙色分割线

墨绿分割线

青色分割线

蓝色分割线

黑色分割线

灰色分割线

1. 默认分割线
2. `<hr>`
- 3.
4. 赤色分割线
5. `<hr class="layui-bg-red">`
- 6.
7. 橙色分割线
8. `<hr class="layui-bg-orange">`
- 9.
10. 墨绿分割线
11. `<hr class="layui-bg-green">`
- 12.
13. 青色分割线
14. `<hr class="layui-bg-cyan">`
- 15.
16. 蓝色分割线
17. `<hr class="layui-bg-blue">`
- 18.
19. 黑色分割线
20. `<hr class="layui-bg-black">`
- 21.
22. 灰色分割线
23. `<hr class="layui-bg-gray">`
- 24.

layui - 用心与你沟通

- 弹出层 layer
- 日期与时间选择 laydate
- 即时通讯 layim
- 分页 laypage
- 模板引擎 laytpl
- 数据表格 table
- 表单 form
- 文件上传 upload
- 穿梭框 transfer
- 树形组件 tree
- 颜色选择器 colorpicker
- 常用元素操作 element
- 滑块 slider
- 评分 rate
- 轮播 carousel
- 流加载 flow
- 工具集 util
- 代码修饰器 code

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，而是这数年来的坚持、不弃的执念，将那些不屑的眼光转化为应得的尊重，不断完善和维护、不断建设和提升社区服务，在 Web 开发者的圈子里口口相传，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 Stars 5000+，官网累计下载量达 50w+，大概有 30 万不同规模的 Web 平台使用过 layer。

之所以列举上面这些数字，并非是在夸夸其谈，而是懂 layer 的人都知道，这是一种怎样不易的沉淀。而由于 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，独立版本：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。

场景	用前准备	调用方式
1. 作为独立组件使用	如果你只是单独想使用 layer，你可以去 layer 独立版本官网下载组件包。你需要在你的页面引入 jQuery_1.8 以上的任意版本，并引入 <code>_layer.js</code> 。	通过 script 标签引入 layer.js 后，直接用即可。参考： 快速上手
2. layui 模块化使用	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 <code>layui.css</code> 和 <code>layui.js</code>	通过 <code>layui.use('layer', callback)</code> 加载模块

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

```
layui.use('layer', function(){
    var layer = layui.layer;

    layer.msg('hello');
});
```

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）。若你采用`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，若你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；若你还需要自定义标题区域样式，那么你可以`title: ['文本', 'font-size:18px;']`，数组第二项可以写任意css样式；如果你不想显示标题栏，你可以`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：

```

/*
 如果是页面层
 */
layer.open({
  type: 1,
  content: '传入任意的文本或html' //这里content是一个普通的String
});
layer.open({
  type: 1,
  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
});
//Ajax获取
$.post('url', {}, function(str){
  layer.open({
    type: 1,
    content: str //注意，如果str是object，那么需要字符拼接。
  });
});
/*
 如果是iframe层
 */
layer.open({
  type: 2,
  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://sentsin.com', 'no']
});
/*
如果是用layer.open执行tips层
*/
layer.open({
  type: 4,
  content: ['内容', '#id'] //数组第二项即吸附元素选择器或者DOM
});

```

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

```

//单个使用
layer.open({
  skin: 'demo-class'
});
//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
layer.config({
  skin: 'demo-class'
})
//CSS
body .demo-class .layui-layer-title{background:#c00; color:#fff; border: none;}
body .demo-class .layui-layer-btn{border-top:1px solid #E9E7E7}
body .demo-class .layui-layer-btn a{background:#333;}
body .demo-class .layui-layer-btn .layui-layer-btn1{background:#999;}
...

```

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。

你也可以[去查看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: 'auto'	默认坐标，即垂直水平居中
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
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：

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

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：

```
//eg1
layer.confirm('纳尼?', {
  btn: ['按钮一', '按钮二', '按钮三'] //可以无限个按钮
  , btn3: function(index, layero){
```

```

    //按钮【按钮三】的回调
  }, function(index, layero){
    //按钮【按钮一】的回调
  }, function(index){
    //按钮【按钮二】的回调
  });

//eg2
layer.open({
  content: 'test'
  ,btn: ['按钮一', '按钮二', '按钮三']
  ,yes: function(index, layero){
    //按钮【按钮一】的回调
  }
  ,btn2: function(index, layero){
    //按钮【按钮二】的回调

    //return false 开启该代码可禁止点击该按钮关闭
  }
  ,btn3: function(index, layero){
    //按钮【按钮三】的回调

    //return false 开启该代码可禁止点击该按钮关闭
  }
  ,cancel: function(){
    //右上角关闭回调

    //return false 开启该代码可禁止点击该按钮关闭
  }
});

```

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

值	备注
anim: 0	平滑放大。默认
anim: 1	从上掉落
anim: 2	从最底部往上滑入
anim: 3	从左滑入
anim: 4	从左翻滚
anim: 5	渐显
anim: 6	抖动

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。默认不显示最大最小化按钮。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

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

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。

maxHeight - 最大高度

类型: Number, 默认: 无

请注意: 只有当高度自适应时, `maxHeight`的设定才有效。

zIndex - 层叠顺序

类型: , 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:

```
layer.open({
  content: '测试回调',
  success: function(layero, index){
    console.log(layero, index);
  }
});
```

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:

```
layer.open({
  content: '测试回调',
  yes: function(index, layero){
    //do something
    layer.close(index); //如果设定了yes回调, 需进行手工关闭
  }
});
```

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:

```
cancel: function(index, layero){
  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
    layer.close(index)
  }
  return false;
}
```

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。它不仅可以配置一些诸如路径、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，你需要执行该方法来完成初始化的配置。比如：

```
layer.config({  
  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  
});  
//这样的话，layer就会去加载一些它所需要的配件，比如css等。  
//当然，你即便不用seajs或者requirejs，也可以通过上述方式设定路径
```

如果你是采用

日期和时间组件文档 - layui.laydate

如你所见，layDate 在 layui 2.0 的版本中迎来一次重生。无论曾经它给你带来过多么糟糕的体验，从今往后，所有的旧坑都将弥合。全面重写的 layDate 包含了大量的更新，其中主要以：年选择器、年月选择器、日期选择器、时间选择器、日期时间选择器 五种类型的选择方式为基本核心，并且均支持范围选择（即双控件）。内置强劲自定义日期格式解析和合法校正机制，含中文版和国际版，主题简约却又不失灵活多样。由于内部采用的是零依赖的原生 JavaScript 编写，因此又可作为独立组件使用。毫无疑问，这是 layui 的虚心之作。

模块加载名称：laydate，独立版本：<http://www.layui.com/laydate/>

快速使用

和 layer 一样，你可以在 layui 中使用 layDate，也可直接使用 layDate 独立版，请按照你的实际需求来选择。

场景	用前准备	调用方式
1. 在 layui 模块中使用	下载 layui 后，引入layui.css和layui.js即可	通过layui.use('laydate', callback)加载模块后，再调用方法
2. 作为独立组件使用	去 layDate 独立版官网下载组件包，引入 laydate.js 即可	直接调用方法使用

这是一个最简单的示例：



对应的代码如下：

```
1. <!DOCTYPE html>
2. <html>
3. <head>
```

```

4.   <meta charset="utf-8">
5.   <title>layDate快速使用</title>
6.   <link rel="stylesheet" href="/static/build/layui.css" media="all">
7. </head>
8. <body>
9.
10. <div class="layui-inline"> <!-- 注意：这一层元素并不是必须的 -->
11.   <input type="text" class="layui-input" id="test1">
12. </div>
13.
14. <script src="/static/build/layui.js"></script>
15. <script>
16.   layui.use('laydate', function(){
17.     var laydate = layui.laydate;
18.
19.     //执行一个laydate实例
20.     laydate.render({
21.       elem: '#test1' //指定元素
22.     });
23.   });
24. </script>
25. </body>
26. </html>
27.

```

```

1. <!DOCTYPE html>
2. <html>
3. <head>
4.   <meta charset="utf-8">
5.   <title>使用 layDate 独立版</title>
6. </head>
7. <body>
8.
9. <input type="text" id="test1">
10.
11. <script src="laydate.js"></script>
12. <script>
13.   //执行一个laydate实例
14.   laydate.render({
15.     elem: '#test1' //指定元素
16.   });
17. </script>
18. </body>
19. </html>
20.

```

除了在组件加载方式有一些小小的不同，其它都完全类似

基础参数选项

通过核心方法：`laydate.render(options)` 来设置基础参数，也可以通过方法：`laydate.set(options)` 来设定全局基础参数。

elem - 绑定元素

类型：`String/DOM`，默认值：无

必填项，用于绑定执行日期渲染的元素，值一般为选择器，或DOM对象

```
1. laydate.render({
2.   elem: '#test' //或 elem: document.getElementById('test')、elem: lay('#test')
   等
3. });
4.
```

type - 控件选择类型

类型：`String`，默认值：`date`

用于单独提供不同的选择器类型，可选值如下表：

type可选值	名称	用途
year	年选择器	只提供年列表选择
month	年月选择器	只提供年、月选择
date	日期选择器	可选择：年、月、日。type默认值，一般可不填
time	时间选择器	只提供时、分、秒选择
datetime	日期时间选择器	可选择：年、月、日、时、分、秒

```
1. //年选择器
2. laydate.render({
3.   elem: '#test'
4.   ,type: 'year'
5. });
6.
7. //年月选择器
8. laydate.render({
9.   elem: '#test'
10.  ,type: 'month'
11. });
12.
13. //日期选择器
14. laydate.render({
15.   elem: '#test'
16.   //,type: 'date' //默认，可不填
17. });
18.
19. //时间选择器
```

```

20. laydate.render({
21.   elem: '#test'
22.   ,type: 'time'
23. });
24.
25. //日期时间选择器
26. laydate.render({
27.   elem: '#test'
28.   ,type: 'datetime'
29. });
30.

```

range - 开启左右面板范围选择

类型: *Boolean/String*, 默认值: *false*

如果设置 *true*, 将默认采用 “ - ” 分割。 你也可以直接设置 分割字符。五种选择器类型均支持左右面板的范围选择。

```

1. //年范围选择
2. laydate.render({
3.   elem: '#test'
4.   ,type: 'year'
5.   ,range: true //或 range: '~' 来自定义分割字符
6. });
7.
8. //年月范围选择
9. laydate.render({
10.  elem: '#test'
11.  ,type: 'month'
12.  ,range: true //或 range: '~' 来自定义分割字符
13. });
14.
15. //日期范围选择
16. laydate.render({
17.  elem: '#test'
18.  ,range: true //或 range: '~' 来自定义分割字符
19. });
20.
21. //时间范围选择
22. laydate.render({
23.  elem: '#test'
24.  ,type: 'time'
25.  ,range: true //或 range: '~' 来自定义分割字符
26. });
27.
28. //日期时间范围选择
29. laydate.render({
30.  elem: '#test'
31.  ,type: 'datetime'
32.  ,range: true //或 range: '~' 来自定义分割字符

```

```
33. });
34.
```

format - 自定义格式

类型：*String*，默认值：*yyyy-MM-dd*

通过日期时间各自的格式符和长度，来设定一个你所需要的日期格式。layDate支持的格式如下：

格式符	说明
yyyy	年份，至少四位数。如果不足四位，则前面补零
y	年份，不限制位数，即不管年份多少位，前面均不补零
MM	月份，至少两位数。如果不足两位，则前面补零。
M	月份，允许一位数。
dd	日期，至少两位数。如果不足两位，则前面补零。
d	日期，允许一位数。
HH	小时，至少两位数。如果不足两位，则前面补零。
H	小时，允许一位数。
mm	分钟，至少两位数。如果不足两位，则前面补零。
m	分钟，允许一位数。
ss	秒数，至少两位数。如果不足两位，则前面补零。
s	秒数，允许一位数。

通过上述不同的格式符组合成一段日期时间字符串，可任意排版，如下所示：

格式	示例值
yyyy-MM-dd HH:mm:ss	2017-08-18 20:08:08
yyyy年MM月dd日 HH时mm分ss秒	2017年08月18日 20时08分08秒
yyyyMMdd	20170818
dd/MM/yyyy	18/08/2017
yyyy年M月	2017年8月
M月d日	8月18日
北京时间：HH点mm分	北京时间：20点08分
yyyy年的M月某天晚上，大概H点	2017年的8月某天晚上，大概20点

```
1. //自定义日期格式
2. laydate.render({
3.   elem: '#test'
4.   , format: 'yyyy年MM月dd日' //可任意组合
5. });
```


6.

value - 初始值

类型: *String*, 默认值: *new Date()*

支持传入符合format参数设定的日期格式字符, 或者 *new Date()*

```

1. //传入符合format格式的字符给初始值
2. laydate.render({
3.   elem: '#test'
4.   ,value: '2018-08-18' //必须遵循format参数设定的格式
5. });
6.
7. //传入Date对象给初始值
8. laydate.render({
9.   elem: '#test'
10.  ,value: new Date(1534766888000) //参数即为: 2018-08-20 20:08:08 的时间戳
11. });
12.
```

isInitValue - 初始值填充

类型: *Boolean*, 默认值: *true*

用于控制是否自动向元素填充初始值 (需配合 *value* 参数使用)

```

1. laydate.render({
2.   elem: '#test'
3.   ,value: '2017-09-10'
4.   ,isInitValue: false //是否允许填充初始值, 默认为 true
5. });
6.
```

注意: 该参数为 layui 2.3.0 新增。

min/max - 最小/大范围内的日期时间值

类型: *string*, 默认值: *min: '1900-1-1'、max: '2099-12-31'*

设定有限范围内的日期或时间值, 不在范围内的将不可选中。这两个参数的赋值非常灵活, 主要有以下几种情况:

1.	如果值为字符类型, 则: 年月日必须用 - (中划线) 分割、时分秒必须用 : (半角冒号) 号分割。这里并非遵循 <i>format</i> 设定的格式
2.	如果值为整数类型, 且数字 < 86400000, 则数字代表天数, 如: <i>min: -7</i> , 即代表最小日期在7天前, 正数代表若干天后

3.	如果值为整数类型，且数字 ≥ 86400000 ，则数字代表时间戳，如：max: 4073558400000，即代表最大日期在：公元3000年1月1日
----	----------------------------------------------------------------------------------

```

1. //日期有效范围只限定在：2017年
2. laydate.render({
3.   elem: '#test'
4.   ,min: '2017-1-1'
5.   ,max: '2017-12-31'
6. });
7.
8. //日期有效范围限定在：过去一周到未来一周
9. laydate.render({
10.   elem: '#test'
11.   ,min: -7 //7天前
12.   ,max: 7 //7天后
13. });
14.
15. //日期时间有效范围的设定：
16. laydate.render({
17.   elem: '#test'
18.   ,type: 'datetime'
19.   ,min: '2017-8-11 12:30:00'
20.   ,max: '2017-8-18 12:30:00'
21. });
22.
23. //时间有效范围设定在：上午九点半到下午五点半
24. laydate.render({
25.   elem: '#test'
26.   ,type: 'time'
27.   ,min: '09:30:00'
28.   ,max: '17:30:00'
29. });
30.

```

毫不保留地说，min和max参数是两个非常强大的存在，合理运用，可帮助用户在日期与时间的选择上带来更为友好的约束与体验。

trigger - 自定义弹出控件的事件

类型：*String*，默认值：*focus*，如果绑定的元素非输入框，则默认事件为：*click*

```

1. //自定义事件
2. laydate.render({
3.   elem: '#test'
4.   ,trigger: 'click' //采用click弹出
5. });
6.

```

show - 默认显示

类型：*Boolean*，默认值：*false*

如果设置：*true*，则控件默认显示在绑定元素的区域。通常用于外部事件调用控件，如：

```
1. //默认显示
2. laydate.render({
3.   elem: '#test'
4.   , show: true //直接显示
5. });
6.
7. //外部事件调用
8. lay('#test1').on('click', function(e){ //假设 test1 是一个按钮
9.   laydate.render({
10.    elem: '#test'
11.    , show: true //直接显示
12.    , closeStop: '#test1' //这里代表的意思是：点击 test1 所在元素阻止关闭事件冒泡。如果不
    设定，则无法弹出控件
13.   });
14. });
15.
```

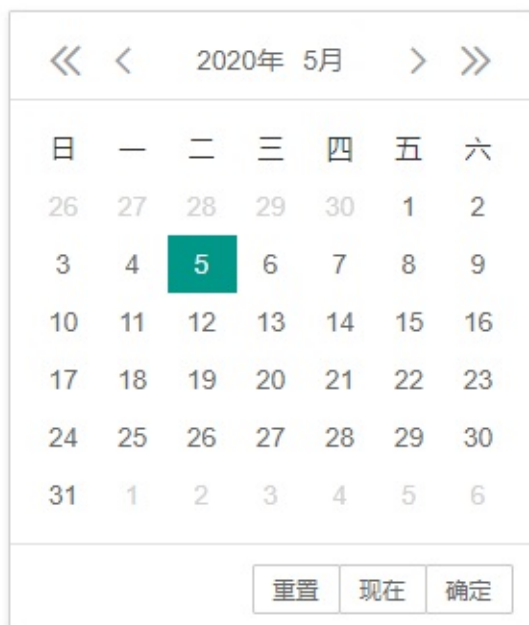
position - 定位方式

类型：*String*，默认值：*absolute*

用于设定控件的定位方式，有以下三种可选值：

position 可选值	说明
absolute	绝对定位，始终吸附在绑定元素周围。默认值
fixed	固定定位，初始吸附在绑定元素周围，不随浏览器滚动条所左右。一般用于在固定定位的弹层中使用。
static	静态定位，控件将直接嵌套在指定容器中。 注意：请勿与 <i>show</i> 参数的概念搞混淆。 <i>show</i> 为 <i>true</i> 时，控件仍然是采用绝对或固定定位。而这里是直接嵌套显示

下面是一个直接嵌套显示的例子：



```

1.  【HTML】
2.  <span id="testView"></span>
3.  <div id="test2"></div>
4.
5.  【JS】：
6.  //嵌套在指定容器中
7.  laydate.render({
8.    elem: '#test2'
9.    ,position: 'static'
10.    ,change: function(value, date){ //监听日期被切换
11.      lay('#testView').html(value);
12.    }
13.  });
14.

```

zIndex - 层叠顺序

类型：*Number*，默认值：66666666

一般用于解决与其它元素的互相被遮掩的问题。如果 `position` 参数设为 `static` 时，该参数无效。

```

1.  //设定控件的层叠顺序
2.  laydate.render({
3.    elem: '#test'
4.    ,zIndex: 99999999
5.  });
6.

```

showBottom - 是否显示底部栏

类型: *Boolean*, 默认值: *true*

如果设置 *false*, 将不会显示控件的底部栏区域

```
1. //不显示底部栏
2. laydate.render({
3.   elem: '#test'
4.   ,showBottom: false
5. });
6.
```

btns - 工具按钮

类型: *Array*, 默认值: *['clear', 'now', 'confirm']*

右下角显示的按钮, 会按照数组顺序排列, 内置可识别的值有: *clear*、*now*、*confirm*

```
1. //只显示清空和确认
2. laydate.render({
3.   elem: '#test'
4.   ,btns: ['clear', 'confirm']
5. });
6.
```

lang - 语言

类型: *String*, 默认值: *cn*

我们内置了两种语言版本: *cn* (中文版)、*en* (国际版, 即英文版)。这里并没有开放自定义文字, 是为了避免繁琐的配置。

```
1. //国际版
2. laydate.render({
3.   elem: '#test'
4.   ,lang: 'en'
5. });
6.
```

theme - 主题

类型: *String*, 默认值: *default*

我们内置了多种主题, *theme*的可选值有: *default* (默认简约)、*molv* (墨绿背景)、*#颜色值* (自定义颜色背景)、*grid* (格子主题)

```

1.  //墨绿背景主题
2.  laydate.render({
3.    elem: '#test'
4.    ,theme: 'molv'
5.  });
6.
7.  //自定义背景色主题 - 非常实用
8.  laydate.render({
9.    elem: '#test'
10.   ,theme: '#393D49'
11.  });
12.
13. //格子主题
14. laydate.render({
15.   elem: '#test'
16.   ,theme: 'grid'
17. });
18.

```

另外，你还可以传入其它字符，如：theme: 'xxx'，那么控件将会多出一个 `class="laydate-theme-xxx"` 的CSS类，以便于你单独定制主题。

calendar - 是否显示公历节日

类型: *Boolean*，默认值: *false*

我们内置了一些我国通用的公历重要节日，通过设置 *true* 来开启。国际版不会显示。

```

1.  //允许显示公历节日
2.  laydate.render({
3.    elem: '#test'
4.    ,calendar: true
5.  });
6.

```

mark - 标注重要日子

类型: *Object*，默认值: 无

calendar 参数所代表的公历节日更多情况下是一个摆设。因此，我们还需要自定义标注重要日子，比如结婚纪念日？日程等？它分为以下两种：

标注	格式	说明
每年的日期	{ '0-9-18': '国耻' }	0 即代表每一年
每月的日期	{ '0-0-15': '中旬' }	0-0 即代表每年每月 (layui 2.1.1/layDate 5.0.4 新增)

特定的日期	{'2017-8-21': '发布'})	-
-------	----------------------	---

可同时设定多个，如：

```

1. //标注重要日子
2. var ins1 = laydate.render({
3.   elem: '#test'
4.   ,mark: {
5.     '0-10-14': '生日'
6.     , '0-12-31': '跨年' //每年12月31日
7.     , '0-0-10': '工资' //每个月10号
8.     , '2017-8-15': '' //具体日期
9.     , '2017-8-20': '预发' //如果为空字符，则默认显示数字+徽章
10.    , '2017-8-21': '发布'
11.   }
12.   ,done: function(value, date){
13.     if(date.year === 2017 && date.month === 8 && date.date === 15){ //点击2017年
14.       ins1.hint('中国人民抗日战争胜利72周年');
15.     }
16.   }
17. });
18.

```

非常实用的存在，是时候利用它制作你的日程表了。

控件初始打开的回调

控件在打开时触发，回调返回一个参数：初始的日期时间对象

```

1. laydate.render({
2.   elem: '#test'
3.   ,ready: function(date){
4.     console.log(date); //得到初始的日期时间对象:{year: 2017, month: 8, date: 18,
5.       hours: 0, minutes: 0, seconds: 0}
6.   }
7. });

```

日期时间被切换后的回调

年月日时间被切换时都会触发。回调返回三个参数，分别代表：生成的值、日期时间对象、结束的日期时间对象

```

1. laydate.render({
2.   elem: '#test'
3.   ,change: function(value, date, endDate){

```

```

4.     console.log(value); //得到日期生成的值，如：2017-08-18
5.     console.log(date); //得到日期时间对象：{year: 2017, month: 8, date: 18, hours:
0, minutes: 0, seconds: 0}
6.     console.log(endDate); //得结束的日期时间对象，开启范围选择（range: true）才会返回。
对象成员同上。
7.   }
8. });
9.

```

控件选择完毕后的回调

点击日期、清空、现在、确定均会触发。回调返回三个参数，分别代表：生成的值、日期时间对象、结束的日期时间对象

```

1. laydate.render({
2.   elem: '#test'
3.   ,done: function(value, date, endDate){
4.     console.log(value); //得到日期生成的值，如：2017-08-18
5.     console.log(date); //得到日期时间对象：{year: 2017, month: 8, date: 18, hours:
0, minutes: 0, seconds: 0}
6.     console.log(endDate); //得结束的日期时间对象，开启范围选择（range: true）才会返回。
对象成员同上。
7.   }
8. });
9.

```

弹出控件提示

事实上，执行核心方法 `laydate.render(options)` 会返回一个当前实例对象。其中包含一些成员属性和方法，比如：hint方法

```

1. var ins1 = laydate.render({
2.   elem: '#test'
3.   ,change: function(value, date, endDate){
4.     ins1.hint(value); //在控件上弹出value值
5.   }
6. });
7.

```

配置基础路径

如果你不是采用 layui 或者普通 script 标签方式加载的 laydate.js，而是采用 requirejs 等其它方式引用 laydate，那么你需要设置基础路径，以便 laydate.css 完成加载。

```

1. laydate.path = '/static/xxx/'; //laydate.js 所在目录
2.

```



```
3. //配置好路径后，再调用
4. laydate.render(options);
5.
```

其它方法

方法名	备注
laydate.getEndDate(month, year)	获取指定年月的最后一天，month默认为当前月，year默认为当前年。如： var endDate1 = laydate.getEndDate(10); //得到31 var endDate2 = laydate.getEndDate(2, 2080); //得到29

结语

layDate最早发布于2014年6月，但当时只迭代了一个版本，就再也没有更新。而时至今日，作为 layui 2.0 的核心组成之一，layDate 再度强势复活，不禁让人感慨万千！layDate曾被我定义为：“最失败的一个组件”，被我无情搁置了整整三年。现在，是时候卸下这个标签了。

LayIM 开发者文档

当前文档适用于 LayIM PC端 最新版，如果你正在了解的并非该版本，你可以前往其它版本阅览：[LayIM 移动版文档](#)

在WebIM似乎已被打入冷宫的今天，LayIM正试图重新为网页带来一些社交想象。作为一款Web即时通讯前端解决方案（服务端需自写），LayIM提供了全方位的前端接口支撑，不仅能让您更高效地接入到自己的通讯服务中，更能让你轻松地与 环信、融云、野狗 等第三方通讯服务平台对接。LayIM始终坚持极简的体验，致力于拉近你的用户在web间的距离。

LayIM兼容除IE6/7以外的所有浏览器，如果你的网站仍需兼容ie6/7，那么强烈建议你说服你的老板或者客户。

模块加载名称：layim，官网地址：layim.layui.com

开始使用

LayIM基于layui模块体系，因此你获得的其实是一个包含LayIM的layui框架，不同的是，开源版的layui并不包含LayIM。捐赠后，将您获得的压缩包解压，将layui整个目录文件放入你的项目后，不用再对其代码做任何修改（方便下次升级）。然后您只需引入下述两个文件即可。

1. `./build/css/layui.css`
2. `./build/layui.js`

假如你将layui放入你的/static/目录中，并且你的html页面在根目录，那么一个最直接的例子是：

```
1. <!doctype html>
2. <html>
3. <head>
4. <meta charset="utf-8">
5. <title>LayIM测试</title>
6. <link rel="stylesheet" href="/static/build/layui.css" media="all">
7. </head>
8. <body>
9. <script src="/static/build/layui.js"></script>
10. <script>
11. layui.use('layim', function(layim){
12.     //先来个客服模式压压精
13.     layim.config({
14.         brief: true //是否简约模式（如果true则不显示主面板）
15.     }).chat({
16.         name: '客服姐姐'
17.         , type: 'friend'
18.         , avatar: 'http://tp1.sinaimg.cn/5619439268/180/40030060651/1'
19.         , id: -2
20.     });
21. });
```

```

22. </script>
23. </body>
24. </html>
25.

```

通过上述方式，便可成功加载layim。当然，你仅仅只是看到了一个“客服姐姐”的聊天面板，这等同于：Hello World!

向“客服姐姐”问好后就忘了它吧，这份文档才刚刚开始。 

初始化配置

一个你必须认识的方法：layim.config(options)

控制着许多重要的配置，基本上一个完整LayIM实例的构成由它而开始，它允许你自由设定以下参数：

```

1. layui.use('layim', function(layim){
2.   //基础配置
3.   layim.config({
4.
5.     init: {} //获取主面板列表信息，下文会做进一步介绍
6.
7.     //获取群员接口（返回的数据格式见下文）
8.     ,members: {
9.       url: '' //接口地址（返回的数据格式见下文）
10.      ,type: 'get' //默认get，一般可不填
11.      ,data: {} //额外参数
12.    }
13.
14.    //上传图片接口（返回的数据格式见下文），若不开启图片上传，剔除该项即可
15.    ,uploadImage: {
16.      url: '' //接口地址
17.      ,type: 'post' //默认post
18.    }
19.
20.    //上传文件接口（返回的数据格式见下文），若不开启文件上传，剔除该项即可
21.    ,uploadFile: {
22.      url: '' //接口地址
23.      ,type: 'post' //默认post
24.    }
25.    //扩展工具栏，下文会做进一步介绍（如果无需扩展，剔除该项即可）
26.    ,tool: [{
27.      alias: 'code' //工具别名
28.      ,title: '代码' //工具名称
29.      ,icon: '&#xe64e;' //工具图标，参考图标文档
30.    }]
31.
32.    ,msgbox: layui.cache.dir + 'css/modules/layim/html/msgbox.html' //消息盒子页面地址，若不开启，剔除该项即可
33.    ,find: layui.cache.dir + 'css/modules/layim/html/find.html' //发现页面地址，若不开启，剔除该项即可

```

```
34.         ,chatLog: layui.cache.dir + 'css/modules/layim/html/chatlog.html' //聊天记录
           页面地址，若不开启，剔除该项即可
35.     });
36. });
37.
```

layim.config(options)的更多可选配置项见下表：

可选项	默认值	类型	用途
brief	false	boolean	是否简约模式，如果设为 true，则主面板不会显示。一般可用于客服
title	我的LayIM	string	主面板最小化后显示的名称
min	false	boolean	用于设定主面板是否在页面打开时，始终最小化展现
right	0px	string	用于设定主面板右偏移量。该参数可避免遮盖你页面右下角已经的bar
minRight	无	string	用户控制聊天面板最小化时、及新消息提示层的相对right的px坐标。 如：minRight：‘200px’
initSkin	空字符	string	设置初始背景，默认不开启。可设置./css/modules/layim/skin目录下的图片文件名 如：initSkin：‘5.jpg’
isAudio	false	boolean	是否开启聊天工具栏音频
isVideo	false	boolean	是否开启开启聊天工具栏视频
notice	false	boolean	是否开启桌面消息提醒，即在浏览器之外的提醒
voice	default.mp3	string/boolean	设定消息提醒的声音文件（所在目录：./layui/css/modules/layim/voice/） 若不开启，设置 false 即可
isfriend	true	boolean	是否开启好友
isgroup	true	boolean	是否开启群组
maxLength	3000	number	可允许的消息最大字符长度
skin	null	object	拓展背景，如： <pre>1. skin: [2. 'http://xxx.com/skin.jpg', 3. 'b.png' 4. ... 5.]</pre>
copyright	false	boolean	是否授权。如果非授权获得，或将LayIM应用在第三方，建议保留，即不设置。

init数据格式

通过layim.config来设定init参数可获得：我的信息、好友列表、群组列表。你可以采用Ajax配置方式，以及直接赋值列表数据。

```

1.  /获取主面板列表信息
2.  init: {
3.      url: '' //接口地址（返回的数据格式见下文）
4.      ,type: 'get' //默认get，一般可不填
5.      ,data: {} //额外参数
6.  }
7.

```

那么该接口所返回的信息（response）应该严格按照下述的JSON格式：

```

1.  {
2.      "code": 0 //0表示成功，其它表示失败
3.      , "msg": "" //失败信息
4.      , "data": {
5.
6.          //我的信息
7.          "mine": {
8.              "username": "纸飞机" //我的昵称
9.              , "id": "100000" //我的ID
10.             , "status": "online" //在线状态 online：在线、hide：隐身
11.             , "sign": "在深邃的编码世界，做一枚轻盈的纸飞机" //我的签名
12.             , "avatar": "a.jpg" //我的头像
13.         }
14.
15.         //好友列表
16.         , "friend": [{
17.             "groupname": "前端码屌" //好友分组名
18.             , "id": 1 //分组ID
19.             , "list": [{ //分组下的好友列表
20.                 "username": "贤心" //好友昵称
21.                 , "id": "100001" //好友ID
22.                 , "avatar": "a.jpg" //好友头像
23.                 , "sign": "这些都是测试数据，实际使用请严格按照该格式返回" //好友签名
24.                 , "status": "online" //若值为offline代表离线，online或者不填为在线
25.             }, ..... ]
26.         }, ..... ]
27.
28.         //群组列表
29.         , "group": [{
30.             "groupname": "前端群" //群组名
31.             , "id": "101" //群组ID
32.             , "avatar": "a.jpg" //群组头像
33.         }, ..... ]
34.     }
35. }
36.

```

Demo: [getList.json](#)

1. //里面的字段格式 同 上文的 data 中的格式。
2. //注意：采用该方式时，不可以再配置url参数，否则会走Ajax模式

```

3.  init: {
4.      mine: {}
5.      ,friend: []
6.      ,group: []
7.  }
8.

```

members数据格式

通过layim.config来设定members参数可获取群员列表，假设你进行了下述配置

```

1.  ,members: {
2.      url: ' '/api/im/getMembers/'
3.      ,data: {}
4.  }
5.

```

那么当点击群聊面板查看成员时，将会向members的url发送Ajax请求，并自动传递一个id参数（群组id）。该接口所返回的信息（response）应该严格按照下述的JSON格式：

```

1.  {
2.      "code": 0 //0表示成功，其它表示失败
3.      , "msg": "" //失败信息
4.      , "data": {
5.          "list": [{
6.              "username": "马小云" //群员昵称
7.              , "id": "168168" //群员id
8.              , "avatar": "http://tp4.sinaimg.cn/2145291155/180/5601307179/1" //群员头像
9.              , "sign": "让天下没有难写的代码" //群员签名
10.          }, ..... ]
11.      }
12.  }
13.

```

Demo: [getMembers.json](#)

上传图片

通过layim.config来设定uploadImage参数可进行图片上传，假设你进行了下述配置

```

1.  ,uploadImage: {
2.      url: ''
3.  }
4.

```

那么当点击聊天面板上传图片时，将会向uploadImage的url发送HTTP请求，进行图片上传。我们会给你传递一个name="file"的文件表单域，你接受即可。该接口所返回的信息（response）应该严格按照下述的JSON格式：

```

1.  {
2.    "code": 0 //0表示成功，其它表示失败
3.    , "msg": "" //失败信息
4.    , "data": {
5.      "src": "http://cdn.xxx.com/upload/images/a.jpg" //图片url
6.    }
7.  }
8.

```

上传文件

通过layim.config来设定uploadImage参数可进行文件上传，事实上跟图片上传非常类似有木有？假设你进行了下述配置

```

1.  ,uploadFile: {
2.    url: ' '
3.  }
4.

```

那么当点击聊天面板上传文件时，将会向uploadFile的url发送HTTP请求，进行文件上传。我们会给你传递一个name="file"的文件表单域，你接受即可。该接口所返回的信息（response）应该严格按照下述的JSON格式：

```

1.  {
2.    "code": 0 //0表示成功，其它表示失败
3.    , "msg": "" //失败信息
4.    , "data": {
5.      "src": "http://cdn.xxx.com/upload/file/LayIM.zip" //文件url
6.      , "name": "LayIM.zip" //文件名
7.    }
8.  }
9.

```

扩展工具栏

你可以对聊天面板的工具栏区域进行扩展，以便实现更丰富的聊天内容。首先在你layim.config中设定好要扩展的图标、名称等信息，如：

```

1.  layim.config({
2.    //可同时配置多个

```

```

3.     tool: [{
4.         alias: 'code' //工具别名
5.         ,title: '代码' //工具名称
6.         ,icon: '&#xe64e;' //工具图标，参考图标文档
7.     }]
8. })
9.

```

那么在工具栏中将会出现一个代码的图标，如何在点击它时实现你想要的功能呢？很简单，你只需要：

```

1. //监听自定义工具栏点击，以添加代码为例
2. layim.on('tool(code)', function(insert, send, obj){ //事件中的tool为固定字符，而
   code则为过滤器，对应的是工具别名 (alias)
3.     layer.prompt({
4.         title: '插入代码'
5.         ,formType: 2
6.         ,shade: 0
7.     }, function(text, index){
8.         layer.close(index);
9.         insert('[pre class=layui-code]' + text + '[/pre]'); //将内容插入到编辑器，主要
   由insert完成
10.        //send(); //自动发送
11.    });
12.    console.log(this); //获取当前工具的DOM对象
13.    console.log(obj); //获得当前会话窗口的DOM对象、基础信息
14. });
15.

```

消息盒子

你可以配置msgbox指向一个盒子页面地址，从而为主面板赋予各种系统消息机制。

```

1. layim.config({
2.     msgbox: layui.cache.dir + 'css/modules/layim/html/msgbox.html' //消息盒子页面地
   址，若不开启，剔除该项即可
3. });
4.

```

它指向的是我们内置的一个盒子模版（你也可以换成你自己的页面），里面写好了所有代码，你只需要按照里面的注释配置好一些接口即可。

当你试图在盒子的图标上实时获取到最新的系统消息数时，你可以借助以下方法完成。

```

1. layim.msgbox(5); //数字即为你通过websocket或者Ajax实时获取到的最新消息数量
2. //它将在主面板的消息盒子icon上不断显隐提示，直到点击后自动消失
3.

```


查找页面

为了避免联调的复杂性且需求不一，所以官方暂不提供该模版结构与样式，实际使用时，可移至该文件到你的项目中，对页面自行把控

```
1. layim.config({
2.   find: layui.cache.dir + 'css/modules/layim/html/find.html' //发现页面地址，若不
   开启，剔除该项即可
3. });
4.
```

查看更多聊天记录

LayIM会在本地存储每个会话的最新 20 条记录。但一般服务端也要存储一份，为了避免联调的复杂性，我们并不内置更多聊天记录的展示。而是弹出一个聊天记录的layer窗口，这意味着你可以随意发挥聊天记录的页面。通过 layim.config配置chatLog来指向聊天记录的URL。假设你进行了下述设定：

```
1. layim.config({
2.   //以下为我们内置的模版，也可以换成你的任意页面。若不开启，剔除该项即可
3.   chatLog: layui.cache.dir + 'css/modules/layim/html/chatlog.html'
4. });
5.
```

那么，我们会在chatLog所对应的URL后面动态追加当前聊天窗口的ID（即好友或群组ID）和类型，如：/chat/log?id=123&type=friend
详细情况可以打开 *chatlog.html* 去了解，里面有我们写好的代码。

富文本内容格式

LayIM的聊天多类型内容解析采用内部特定的文本格式，你可以借助它在聊天内容区域呈现多样化的内容。目前我们支持的内容如下：

1. 超链接格式： <i>a</i> (地址)[文本]	如： <i>a</i> (http://www.layui.com)[layui]
2. 图片格式： <i>img</i> [地址]	如： <i>img</i> [http://cdn.layui.com/xxx/a.jpg]
3. 文件格式： <i>file</i> (地址)[文本] [layim.zip]	如： <i>file</i> (http://cdn.layui.com/download/layim.zip)
4. 音频格式： <i>audio</i> [地址]	如： <i>audio</i> [http://cdn.layui.com/xxx/a.mp3]
5. 视频格式： <i>video</i> [地址]	如： <i>video</i> [http://cdn.layui.com/xxx/a.avi]
6.	

接入WebSocket

一般来说，对接到第三方通讯平台是个更靠谱的选择，比如：环信、融云、野狗。

它们都可以是 LayIM 的最佳拍档，你会省去许多工作。

但如果你想保证绝对的数据隐私，你也可以自己搭建通讯服务。目前所有的主流 Web 语言都有对 WebSocket 做很好的支持，其中也有很多优秀的通讯框架可以任你挑选，因此万万不可畏惧它的存在，除非你真的想用 Ajax 轮询去解决消息 I/O。WebSocket 会使得你的消息接受和传递变得极其轻松，最重要的一点是，性能卓绝。WebSocket 的建立非常简单：

```

1.  layui.use('layim', function(layim){
2.
3.      //建立WebSocket通讯
4.      //注意：如果你要兼容ie8+，建议你采用 socket.io 的版本。下面是以原生WS为例
5.
6.      var socket = new WebSocket('ws://localhost:8090');
7.
8.      //发送一个消息
9.      socket.send('Hi Server, I am LayIM!');
10.     //更多情况下，一般是传递一个JSON
11.     socket.send(JSON.stringify({
12.         type: '' //随便定义，用于在服务端区分消息类型
13.         ,data: {}
14.     }));
15.
16.     //连接成功时触发
17.     socket.onopen = function(){
18.         socket.send('XXX连接成功');
19.     };
20.
21.     //监听收到的消息
22.     socket.onmessage = function(res){
23.         //res为接受到的值，如 {"emit": "messageName", "data": {}}
24.         //emit即为发出的事件名，用于区分不同的消息
25.     };
26.
27.     另外还有onclose、onerror，分别是在链接关闭和出错时触发。
28.
29.     //基本上常用的就上面几个了，是不是非一般的简单？
30.
31.
32. });
33.

```

服务端层面，不用害怕，事情也远没有那么复杂。这些文档也许可以帮助你：
[LayIM 接入案例/教程大集锦，含Node.js/Java/.NET/PHP](#)

事件监听

方法：layim.on(event, callback)

用于LayIM事件监听。接受两个参数。第一个参数event即事件名，第二个参数

callback即事件回调。

ready事件

事件名：ready，用于监听LayIM初始化就绪。由于主面板的渲染，需建立在init接口请求完毕的基础上，而一些操作必须等到主面板渲染完毕后才能操作，所以这个时候就可以放入ready事件的回调体中来执行。其回调接受一个object类型的参数，携带一些基础配置信息、我的用户信息、好友/群列表信息、本地数据库信息等，调用方式：

```
1. layim.on('ready', function(options){
2.     console.log(options);
3.     //do something
4. });
5. //注意：以下情况不会触发 ready事件，即代码无需写在ready里面
6. * 简约模式（即brief: true时）不会触发该事件
7. * init直接赋值mine、friend的情况下（只有设置了url才会执行 ready 事件）
8.
```

监听在线状态切换

事件名：online，看到主面板你的昵称后的icon么，没错，就是它。当前支持“在线”、“隐身”两种状态切换。分别以online和hide的string类型的值传递给回调参数。如：

```
1. layim.on('online', function(status){
2.     console.log(status); //获得online或者hide
3.
4.     //此时，你可以通过Ajax将这个状态值记录到数据库中了。
5.     //服务端接口需自写。
6. });
7.
```

监听修改签名

事件名：sign

当主面板的签名被改动后触发，并返回新的签名

```
1. layim.on('sign', function(value){
2.     console.log(value); //获得新的签名
3.
4.     //此时，你可以通过Ajax将新的签名同步到数据库中了。
5. });
6.
```

监听更换背景皮肤

事件名：setSkin

当点击更换背景皮肤时触发，返回特定目录下的图片文件名和src路径

```
1. layim.on('setSkin', function(filename, src){
2.     console.log(filename); //获得文件名，如：1.jpg
3.     console.log(src); //获得背景路径，如：
      http://res.layui.com/layui/src/css/modules/layim/skin/1.jpg
4. });
5.
```

监听发送的消息

事件名：sendMessage，每当你发送一个消息，都可以通过该事件监听到。回调参数接受一个object类型的值，携带发送的聊天信息。如：

```
1. layim.on('sendMessage', function(res){
2.     var mine = res.mine; //包含我发送的消息及我的信息
3.     //mine的结构如下：
4.     {
5.         avatar: "avatar.jpg" //我的头像
6.         ,content: "你好吗" //消息内容
7.         ,id: "100000" //我的id
8.         ,mine: true //是否我发送的消息
9.         ,username: "纸飞机" //我的昵称
10.    }
11.
12.    var to = res.to; //对方的信息
13.
14.    //to的结构如下：
15.    {
16.        avatar: "avatar.jpg"
17.        ,id: "100001"
18.        ,name: "贤心"
19.        ,sign: "这些都是测试数据，实际使用请严格按照该格式返回"
20.        ,type: "friend" //聊天类型，一般分friend和group两种，group即群聊
21.        ,username: "贤心"
22.    }
23.
24.    //监听到上述消息后，就可以轻松地发送socket了，如：
25.    socket.send(JSON.stringify({
26.        type: 'chatMessage' //随便定义，用于在服务端区分消息类型
27.        ,data: res
28.    }));
29.
30. });
```

监听接受的消息

事件的监听并非layim提供，而是WebSocket提供。

检测到WebSocket事件后，执行layim的内置方法：

`layim.getMessage(options)`

即可显示消息到聊天面板（如果消息所指定的聊天面板没有打开，则会进入本地的消息队列中，直到指定的聊天面板被打开，方可显示。），这是一个对你有用的例子：

```

1. //监听收到的聊天消息，假设你服务端emit的事件名为：chatMessage
2. socket.onmessage = function(res){
3.   res = JSON.parse(res);
4.   if(res.emit === 'chatMessage'){
5.     layim.getMessage(res.data); //res.data即你发送消息传递的数据（阅读：监听发送的消息）
6.   }
7. };
8.
9. //如果是来自于用户的聊天消息，它必须接受以下字段
10. layim.getMessage({
11.   username: "纸飞机" //消息来源用户名
12.   ,avatar: "http://tp1.sinaimg.cn/1571889140/180/40030060651/1" //消息来源用户头像
13.   ,id: "100000" //消息的来源ID（如果是私聊，则是用户id，如果是群聊，则是群组id）
14.   ,type: "friend" //聊天窗口来源类型，从发送消息传递的to里面获取
15.   ,content: "嗨，你好！本消息系离线消息。" //消息内容
16.   ,cid: 0 //消息id，可不传。除非你要对消息进行一些操作（如撤回）
17.   ,mine: false //是否我发送的消息，如果为true，则会显示在右方
18.   ,fromid: "100000" //消息的发送者id（比如群组中的某个消息发送者），可用于自动解决浏览器多窗口时的一些问题
19.   ,timestamp: 1467475443306 //服务端时间戳毫秒数。注意：如果你返回的是标准的 unix 时间戳，记得要 *1000
20. });
21.
22. //如果是来自于系统的聊天面板的消息
23. layim.getMessage({
24.   system: true //系统消息
25.   ,id: 1111111 //聊天窗口ID
26.   ,type: "friend" //聊天窗口类型
27.   ,content: '对方已掉线'
28. });

```

监听查看群员

事件名：members，在群聊面板中查看全部成员时触发，该事件返回获取群员接口（即layim.config中的members）的response信息。

```
1. layim.on('members', function(data){
2.     console.log(data);
3. });
4.
```

监听聊天窗口的切换

事件名: chatChange, 坦白而言, 似乎没什么卵用。不过有总比没有好。该事件返回一个object类型的参数, 携带当前聊天面板的容器、基础信息等。

```
1. layim.on('chatChange', function(obj){
2.     console.log(obj);
3. });
4.
```

自定义一个聊天窗口

方法名: layim.chat(options), 是否似曾相识, 没错, 我们见过文档最开始的“客服姐姐”用的就是该方法。它允许你自定义任意模式的聊天窗口, 先看例子吧:

```
1. //自定义聊天窗口
2. layim.chat({
3.     name: '张三' //名称
4.     ,type: 'friend' //聊天类型
5.     ,avatar: 'http://tp1.sinaimg.cn/5619439268/180/40030060651/1' //头像
6.     ,id: 11111 //好友id
7. })
8.
9. //自定义客服窗口
10. layim.config({
11.     brief: true //简约模式, 不显示主面板
12. }).chat({
13.     name: '在线客服二' //名称
14.     ,type: 'friend' //聊天类型
15.     ,avatar: 'http://tp1.sinaimg.cn/5619439268/180/40030060651/1' //头像
16.     ,id: -2 //定义唯一的id方便你处理信息
17. });
18. layim.setChatMin(); //收缩聊天面板
19.
20.
21. //自定义群聊 (对于想搞一个临时性的房间, 貌似是挺有意思的)
22. layim.chat({
23.     name: 'LayIM畅聊'
24.     ,type: 'group' //群组类型
25.     ,avatar: 'http://tp2.sinaimg.cn/5488749285/50/5719808192/1'
26.     ,id: 10000000 //定义唯一的id方便你处理信息
```

```

27.     ,members: 123 //成员数, 不好获取的话, 可以设置为0
28. });

```

该方法结合**brief: true** (简约模式), 可以免去较为复杂的数据配置。轻量地建立一个聊天面板。拥有较大的平台实用性。可以预见的是, 它应该会成为LayIM一个露脸率最高的存在。想象一下吧, 对你而言是否如此?

初始最小化聊天界面

方法名: `layim.setChatMin()`, 如果你在初始的状态下不想展开聊天面板 (譬如悬浮的在线客服), 那么该方法会派上用场, 使用很简单, 就不过多啰嗦了。

```

1. layim.setChatMin();

```

更新当前会话状态

方法名: `layim.setChatStatus(str)`, 可用于显示: 对方输入状态、在线离线状态等, 如:

```

1. //每次窗口打开或切换, 即更新对方的状态
2. layim.on('chatChange', function(res){
3.     var type = res.data.type;
4.     if(type === 'friend'){
5.         layim.setChatStatus('<span style="color:#FF5722;">在线</span>'); //模拟标注好友在线状态
6.     } else if(type === 'group'){
7.         //模拟系统消息
8.         layim.getMessage({
9.             system: true //系统消息
10.            ,id: 111111111
11.            ,type: "group"
12.            ,content: '贤心加入群聊'
13.        });
14.    }
15. });
16.

```

弹出添加面板

方法名: `layim.add(data)`

执行该方法将弹出一个我们内置的添加面板, 可以帮助你完成: 申请添加好友、申请添加群。参数格式如下:

```

1. layim.add({
2.     type: 'friend' //friend: 申请加好友、group: 申请加群

```



```

3.     ,username: 'xxx' //好友昵称, 若申请加群, 参数为: groupname
4.     ,avatar: 'a.jpg' //头像
5.     ,submit: function(group, remark, index){ //一般在此执行Ajax和WS, 以通知对方
6.         console.log(group); //获取选择的好友分组ID, 若为添加群, 则不返回值
7.         console.log(remark); //获取附加信息
8.         layer.close(index); //关闭改面板
9.     }
10. });
11.

```

好友分组面板

方法名: `layim.setFriendGroup(data)`

执行该方法将弹出一个好友分组面板, 以完成将好友追加到主面板的操作:

```

1. layim.setFriendGroup({
2.     type: 'friend'
3.     ,username: 'xxx' //好友昵称, 若申请加群, 参数为: groupname
4.     ,avatar: 'a.jpg' //头像
5.     ,group: layim.cache().friend //获取好友列表数据
6.     ,submit: function(group, index){
7.         //一般在此执行Ajax和WS, 以通知对方已经同意申请
8.         //.....
9.
10.        //同意后, 将好友追加到主面板
11.        layim.addList(data); //见下文
12.    }
13. });
14.

```

添加好友/群到主面板

方法名: `layim.addList(options)`, 当你的WebSocket监听到有好友或者群新增时, 需让LayIM的主面板同步添加的信息, 可用该方法。先看看例子:

```

1. layim.on('ready', function(res){
2.
3.
4.     //监听添加列表的socket事件, 假设你服务端emit的事件名为: addList
5.     socket.onmessage = function(res){
6.         if(res.emit === 'addList'){
7.             layim.addList(res.data); //如果是在iframe页, 如LayIM设定的add面板, 则为
            parent.layui.layim.addList(data);
8.         }
9.     };
10.
11.     //需要特别注意的是回调返回的res
12.

```



```

13.    //如果添加的是好友，res的结构必须是这样的
14.    {
15.      type: 'friend' //列表类型，只支持friend和group两种
16.      ,avatar: "a.jpg" //好友头像
17.      ,username: '冲田杏梨' //好友昵称
18.      ,groupid: 2 //所在的分组id
19.      ,id: "1233333312121212" //好友id
20.      ,sign: "本人冲田杏梨将结束AV女优的工作" //好友签名
21.    }
22.
23.    //如果添加的是群组，res的结构必须是这样的
24.    {
25.      type: 'group' //列表类型，只支持friend和group两种
26.      ,avatar: "a.jpg" //群组头像
27.      ,groupname: 'Angular开发' //群组名称
28.      ,id: "12333333" //群组id
29.    }
30.
31.  });
32.

```

从主面板移除好友/群

方法名: `layim.removeList(options)`

当你的WebSocket监听到有好友或者群删除时，需让LayIM的主面板同步删除的信息，可用该方法。它的调用非常简单，只需要传两个key：

```

1. layim.removeList({
2.   type: 'friend' //或者group
3.   ,id: 1238668 //好友或者群组ID
4. });
5. //如果是在iframe页，如LayIM设定的add面板，则为：
6. parent.layui.layim.removeList({
7.   type: 'friend' //或者group
8.   ,id: 1238668 //好友或者群组ID
9. });
10.

```

实时更新好友列表离线状态

方法名: `layim.setFriendStatus(id, type)`

```

1. layim.setFriendStatus(11111, 'online'); //设置指定好友在线，即头像取消置灰
2. layim.setFriendStatus(11111, 'offline'); //设置指定好友在线，即头像置灰
3.

```

获取cache数据

方法名：`layim.cache()`，获取LayIM的cache信息，返回的信息结构和ready事件获得的信息一样，不同的是，该方法始终会获取到最新的cache。

1. `//输出的信息不妨在你的Chrome控制台看看（需在引有LayIM的页面中），在此就不做列举了`
2. `console.log(layim.cache())`

删除本地数据

```
1. var cache = layui.layim.cache();
2. var local = layui.data('layim')[cache.mine.id]; //获取当前用户本地数据
3.
4. //这里以删除本地聊天记录为例
5. delete local.chatlog;
6.
7. //向localStorage同步数据
8. layui.data('layim', {
9.   key: cache.mine.id
10.  , value: local
11. });
12.
```

关于版权

LayIM目前并非开源产品，因此如果你是通过官网捐赠渠道获得LayIM，你将成为LayIM的终身会员，并可以将LayIM应用在任意正规平台。如果你通过非捐赠渠道获得LayIM，我们并不会进行过多追究，但是请勿对 LayIM 本身二次出售。版权最终解释权为：layui.com 所有

layui - 用心与你沟通

分页模块文档 - layui.laypage

layPage 致力于提供极致的分页逻辑，既可轻松胜任异步分页，也可作为页面刷新式分页。自 layui 2.0 开始，无论是从核心代码还是API设计，layPage 都完成了一次蜕变。清爽的UI、灵活的排版，极简的调用方式，这一切的优质元素，都将毫无违和感地镶嵌在你的页面之中。

模块加载名称：*laypage*

快速使用

laypage 的使用非常简单，指向一个用于存放分页的容器，通过服务端得到一些初始值，即可完成分页渲染：

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4.   <meta charset="utf-8">
5.   <title>layPage快速使用</title>
6.   <link rel="stylesheet" href="/static/build/layui.css" media="all">
7. </head>
8. <body>
9.
10. <div id="test1"></div>
11.
12. <script src="/static/build/layui.js"></script>
13. <script>
14.   layui.use('laypage', function(){
15.     var laypage = layui.laypage;
16.
17.     //执行一个laypage实例
18.     laypage.render({
19.       elem: 'test1' //注意，这里的 test1 是 ID，不用加 # 号
20.       ,count: 50 //数据总数，从服务端得到
21.     });
22.   });
23. </script>
24. </body>
25. </html>
26.
```

基础参数选项

通过核心方法：*laypage.render(options)* 来设置基础参数。由于使用非常简单，本篇直接罗列核心接口的参数选项

参数选项	说明	类型	默认值
------	----	----	-----

elem	指向存放分页的容器，值可以是容器ID、DOM对象。如： 1. elem: 'id' 注意：这里不能加 # 号 2. elem: document.getElementById('id')	String/Object	-
count	数据总数。一般通过服务端得到	Number	-
limit	每页显示的条数。laypage将会借助 count 和 limit 计算出分页数。	Number	10
limits	每页条数的选择项。如果 layout 参数开启了 limit，则会出现每页条数的select选择框	Array	[10, 20, 30, 40, 50]
curr	起始页。一般用于刷新类型的跳页以及HASH跳页。如： <pre>1. //开启location.hash的记录 2. laypage.render({ 3. elem: 'test1' 4. ,count: 500 5. ,curr: location.hash.replace('#!fencye=', '') //获取起始页 6. ,hash: 'fencye' //自定义hash值 7. });</pre>	Number	1
groups	连续出现的页码个数	Number	5
prev	自定义“上一页”的内容，支持传入普通文本和HTML	String	上一页
next	自定义“下一页”的内容，同上	String	下一页
first	自定义“首页”的内容，同上	String	1
last	自定义“尾页”的内容，同上	String	总页数数值
layout	自定义排版。可选值有：count（总条目输出区域）、prev（上一页区域）、page（分页区域）、next（下一页区域）、limit（条目选项区域）、refresh（页面刷新区域。注意：layui 2.3.0 新增）、skip（快捷跳页区域）	Array	['prev', 'page', 'next']
theme	自定义主题。支持传入：颜色值，或任意普通字符。如： 1. theme: '#c00' 2. theme: 'xxx' //将会生成 class="layui-laypage-xxx" 的CSS类，以便自定义主题	String	-
hash	开启location.hash，并自定义 hash 值。如果开启，在触发分页时，会自动对url追加：#!hash值={curr} 利用这个，可以在页面载入时就定位到指定页	String/Boolean	false

jump - 切换分页的回调

当分页被切换时触发，函数返回两个参数：obj（当前分页的所有选项值）、first（是否首次，一般用于初始加载的判断）

```
1. laypage.render({
2.   elem: 'test1'
3.   ,count: 70 //数据总数，从服务端得到
4.   ,jump: function(obj, first){
5.     //obj包含了当前分页的所有参数，比如：
6.     console.log(obj.curr); //得到当前页，以便向服务端请求对应页的数据。
```

```
7.     console.log(obj.limit); //得到每页显示的条数
8.
9.     //首次不执行
10.    if(!first){
11.        //do something
12.    }
13. }
14. });
15.
```

结束

正如你看到对的，laypage 只负责分页本身的逻辑，具体的数据请求与渲染需要另外去完成。laypage 不仅能应用在一般的异步分页上，还可直接对一段已知数据进行分页展现，更可以取代传统的超链接分页，如果你无需考虑SEO的话（因为laypage 的分页是JS动态生成的）。

layui - 用心与你沟通

模板引擎文档 - layui.laytpl

laytpl 是 JavaScript 模板引擎，在字符解析上有着比较出色的表现，欠缺之处在于异常调试上。由于传统意义的前端模板引擎已经变得不再流行，所以 laytpl 后续可能会进行重写，目前方向暂时还没有想好，预计会在layui比较稳定后开始实施。

模块加载名称: laytpl，在线调试: <http://www.layui.com/demo/laytpl.html>

快速使用

与一般的字符拼接不同的是，laytpl 的模板可与数据分离，集中把逻辑处理放在 View 层，提升代码可维护性，尤其是针对大量模板渲染的情况。

```
1. layui.use('laytpl', function(){
2.   var laytpl = layui.laytpl;
3.
4.   //直接解析字符
5.   laytpl('{{ d.name }}是一位公猿').render({
6.     name: '贤心'
7.   }, function(string){
8.     console.log(string); //贤心是一位公猿
9.   });
10.
11.   //你也可以采用下述同步写法，将 render 方法的回调函数剔除，可直接返回渲染好的字符
12.   var string = laytpl('{{ d.name }}是一位公猿').render({
13.     name: '贤心'
14.   });
15.   console.log(string); //贤心是一位公猿
16.
17.   //如果模板较大，你也可以采用数据的写法，这样会比较直观一些
18.   laytpl([
19.     '{{ d.name }}是一位公猿'
20.     , '其它字符 {{ d.content }} 其它字符'
21.   ].join(''))
22. });
23.
```

你也可以将模板存储在页面或其它任意位置：

```
1. //第一步：编写模版。你可以使用一个script标签存放模板，如：
2. <script id="demo" type="text/html">
3.   <h3>{{ d.title }}</h3>
4.   <ul>
5.     {{# layui.each(d.list, function(index, item){ }}
6.       <li>
7.         <span>{{ item.modname }}</span>
8.         <span>{{ item.alias }} : </span>
9.         <span>{{ item.site || '' }}</span>
```

```
10.     </li>
11.     {{#   }); }}
12.     {{#   if(d.list.length === 0){ }}
13.         无数据
14.     {{#   }}
15. </ul>
16. </script>
17.
18. //第二步：建立视图。用于呈现渲染结果。
19. <div id="view"></div>
20.
21. //第三步：渲染模版
22. var data = { //数据
23.     "title": "Layui常用模块"
24.     , "list": [{"modname": "弹层", "alias": "layer", "site": "layer.layui.com"},
25.     {"modname": "表单", "alias": "form"}]
26. };
27. var getTpl = demo.innerHTML
28. , view = document.getElementById('view');
29. laytpl(getTpl).render(data, function(html){
30.     view.innerHTML = html;
31. });
```

模版语法

语法	说明	示例
<code>{{ d.field }}</code>	输出一个普通字段，不转义html	<pre>1. <div>{{ d.content }} 2. </div></pre>
<code>{{= d.field }}</code>	输出一个普通字段，并转义html	<pre>1. <h2>{{= d.title }} 2. </h2></pre>
<code>{{# JavaScript表达式 }}</code>	<p>JS 语句。一般用于逻辑处理。用分隔符加 # 号开头。</p> <p>注意：如果你想输出一个函数，正确的写法是：<code>{{ fn() }}</code>，而不是：<code>{{# fn() }}</code></p>	<pre>1. {{# 2. var fn = function() 3. { 4. return '2017-08-18'; 5. }; 6. }} 7. {{# if(true){ }} 8. 开始日期: {{ fn() }} 9. {{# }} else { }} 10. 已截止 11. {{# }}</pre>

<pre>{{! template !}}</pre>	<p>对一段指定的模板区域进行过滤，即不解析该区域的模板。注：layui 2.1.6 新增</p>	<ol style="list-style-type: none"> 1. <code><div> {{! 这里的模板不会被解析 !}}</div></code> 2.
-----------------------------	---------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------

分隔符

如果模版默认的 `{{ }}` 分隔符与你的其它模板（一般是服务端模板）存在冲突，你也可以重新定义分隔符：

```
1. laytpl.config({
2.   open: '<%',
3.   close: '%>'
4. });
5.
6. //分割符将必须采用上述定义的
7. laytpl([
8.   '<%# var type = "公"; %>' //JS 表达式
9.   , '<% d.name %>是一位<% type %>猿。'
10. ].join('')).render({
11.   name: '贤心'
12. }, function(string){
13.   console.log(string); //贤心是一位公猿
14. });
15.
```

结语

laytpl 应用在 layui 的很多模块中，如：layim、table 等。尽管传统意义的前端模板引擎已经变得不再流行，但 laytpl 仍然可以发挥一定作用，不妨尝试一下吧。

layui - 用心与你沟通

table 数据表格文档 - layui.table

table 模块是我们的又一走心之作，在 layui 2.0 的版本中全新推出，是 layui 最核心的组成之一。它用于对表格进行一些列功能和动态化数据操作，涵盖了日常业务所涉及的几乎全部需求。支持固定表头、固定行、固定列左/列右，支持拖拽改变列宽度，支持排序，支持多级表头，支持单元格的自定义模板，支持对表格重载（比如搜索、条件筛选等），支持复选框，支持分页，支持单元格编辑等等一些列功能。尽管如此，我们仍将对其进行完善，在控制代码量和性能的前提下，不定期增加更多人性化功能。table 模块也将是 layui 重点维护的项目之一。

模块加载名称：*table*

快速使用

创建一个table实例最简单的方式是，在页面放置一个元素，然后通过 *table.render()* 方法指定该容器，如下所示：

ID	用户名	性别	城市	签名	积分	评分	职业	财富
10000	user-0	女	城市-0	签名-0	255	57	作家	82830700
10001	user-1	男	城市-1	签名-1	884	27	词人	64928690
10002	user-2	女	城市-2	签名-2	650	31	酱油	6298078
10003	user-3	女	城市-3	签名-3	362	68	诗人	37117017
10004	user-4	男	城市-4	签名-4	807	6	作家	76263262
10005	user-5	女	城市-5	签名-5	173	87	作家	60344147

< 1 2 3 ... 100 >

到第 1 页

确定

共 1000 条

10 条/页 ▼

上面你已经看到了一个简单数据表格的基本样子，你一定迫不及待地想知道它的使用方式。下面就是它对应的代码：

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4.   <meta charset="utf-8">
5.   <title>table模块快速使用</title>
6.   <link rel="stylesheet" href="/layui/css/layui.css" media="all">
7. </head>
8. <body>
9.
10. <table id="demo" lay-filter="test"></table>
11.
12. <script src="/layui/layui.js"></script>
13. <script>
14.   layui.use('table', function(){
```

```
15.   var table = layui.table;
16.
17.   //第一个实例
18.   table.render({
19.     elem: '#demo'
20.     ,height: 312
21.     ,url: '/demo/table/user/' //数据接口
22.     ,page: true //开启分页
23.     ,cols: [[ //表头
24.       {field: 'id', title: 'ID', width:80, sort: true, fixed: 'left'}
25.       ,{field: 'username', title: '用户名', width:80}
26.       ,{field: 'sex', title: '性别', width:80, sort: true}
27.       ,{field: 'city', title: '城市', width:80}
28.       ,{field: 'sign', title: '签名', width: 177}
29.       ,{field: 'experience', title: '积分', width: 80, sort: true}
30.       ,{field: 'score', title: '评分', width: 80, sort: true}
31.       ,{field: 'classify', title: '职业', width: 80}
32.       ,{field: 'wealth', title: '财富', width: 135, sort: true}
33.     ]]
34.   });
35.
36. });
37. </script>
38. </body>
39. </html>
```

一切都并不会陌生：绑定容器、设置数据接口、在表头设定对应的字段，剩下的...就交给 layui 吧。你的牛刀是否早已饥渴难耐？那么不妨现在就去小试一波吧。数据接口可参考：[返回的数据](#)，规则在下文也有讲解。

三种初始化渲染方式

在上述“快速使用”的介绍中，你已经初步见证了 table 模块的信手拈来，但其使用方式并不止那一种。我们为了满足各种情况下的需求，对 table 模块做了三种初始化的支持，你可按照个人喜好和实际情况灵活使用。

	方式	机制	适用场景
01.	方法渲染	用JS方法的配置完成渲染	（推荐）无需写过多的 HTML，在 JS 中指定原始元素，再设定各项参数即可。
02.	自动渲染	HTML配置，自动渲染	无需写过多 JS，可专注于 HTML 表头部分
03.	转换静态表格	转化一段已有的表格元素	无需配置数据接口，在JS中指定表格元素，并简单地给表头加上自定义属性即可

方法渲染

其实这是“自动化渲染”的手动模式，本质类似，只是“方法级渲染”将基础参数的

设定放在了JS代码中，且原始的 table 标签只需要一个 选择器：

```
1.
2. <table id="demo" lay-filter="test"></table>
3.
```

```
1. var table = layui.table;
2.
3. //执行渲染
4. table.render({
5.   elem: '#demo' //指定原始表格元素选择器（推荐id选择器）
6.   ,height: 315 //容器高度
7.   ,cols: [{}] //设置表头
8.   //,... //更多参数参考右侧目录：基本参数选项
9. });
10.
```

事实上我们更推荐采用“方法级渲染”的做法，其最大的优势在于你可以脱离HTML文件，而专注于JS本身。尤其对于项目的频繁改动及发布，其便捷性会体现得更为明显。而究竟它与“自动化渲染”的方式谁更简单，也只能由各位猿猿们自行体味了。

备注：`table.render()`方法返回一个对象：`var tableIns = table.render(options)`，可用于对当前表格进行“重载”等操作，详见目录：[表格重载](#)

自动渲染

所谓的自动渲染，即：在一段 table 容器中配置好相应的参数，由 table 模块内部自动对其完成渲染，而无需你写初始的渲染方法。其特点在上文也有阐述。你需要关注的是以下三点：

- 1) 带有 `class="layui-table"` 的 \ 标签。
- 2) 对标签设置属性 `lay-data=""` 用于配置一些基础参数
- 3) 在 \ 标签中设置属性`lay-data=""`用于配置表头信息

按照上述的规范写好table原始容器后，只要你加载了layui 的 table 模块，就会自动对其建立动态的数据表格。下面是一个示例：

```
1. <table class="layui-table" lay-data="{height:315, url:'/demo/table/user/',
2.   page:true, id:'test'}" lay-filter="test">
3.   <thead>
4.     <tr>
5.       <th lay-data="{field:'id', width:80, sort: true}">ID</th>
6.       <th lay-data="{field:'username', width:80}">用户名</th>
7.       <th lay-data="{field:'sex', width:80, sort: true}">性别</th>
8.       <th lay-data="{field:'city'}">城市</th>
9.       <th lay-data="{field:'sign'}">签名</th>
```

```

9.         <th lay-data="{field:'experience', sort: true}">积分</th>
10.        <th lay-data="{field:'score', sort: true}">评分</th>
11.        <th lay-data="{field:'classify'}">职业</th>
12.        <th lay-data="{field:'wealth', sort: true}">财富</th>
13.    </tr>
14. </thead>
15. </table>
16.

```

转换静态表格

假设你的页面已经存在了一段有内容的表格，它由原始的table标签组成，这时你需要赋予它一些动态元素，比如拖拽调整列宽？比如排序等等？那么你同样可以很轻松地去实现。如下所示：

昵称	积分	签名
贤心166	人生就像是一场修行a	
贤心288	人生就像是一场修行b	
贤心333	人生就像是一场修行c	

转换上述表格

通过上面的小例子，你已经初步见识了这一功能的实际意义。尝试在你的静态表格的 `th` 标签中加上 `lay-data=""` 属性，代码如下：

```

1. <table lay-filter="demo">
2.   <thead>
3.     <tr>
4.       <th lay-data="{field:'username', width:100}">昵称</th>
5.       <th lay-data="{field:'experience', width:80, sort:true}">积分</th>
6.       <th lay-data="{field:'sign'}">签名</th>
7.     </tr>
8.   </thead>
9.   <tbody>
10.    <tr>
11.      <td>贤心1</td>
12.      <td>66</td>
13.      <td>人生就像是一场修行a</td>
14.    </tr>
15.    <tr>
16.      <td>贤心2</td>
17.      <td>88</td>
18.      <td>人生就像是一场修行b</td>
19.    </tr>
20.    <tr>
21.      <td>贤心3</td>
22.      <td>33</td>
23.      <td>人生就像是一场修行c</td>
24.    </tr>

```

```
25.     </tbody>
26. </table>
27.
```

然后执行用于转换表格的JS方法，就可以达到目的了：

```
1. var table = layui.table;
2.
3. //转换静态表格
4. table.init('demo', {
5.     height: 315 //设置高度
6.     ,limit: 10 //注意：请务必确保 limit 参数（默认：10）是与你服务端限定的数据条数一致
7.     //支持所有基础参数
8. });
9.
```

在前面的“方法渲染”和“自动渲染”两种方式中，你的数据都来源于异步的接口，这可能并不利于所谓的seo（当然是针对于前台页面）。而在这里，你的数据已和页面同步输出，却仍然可以转换成动态表格，是否感受到一丝惬意呢？

基础参数一览表

基础参数并非所有都要出现，有必选也有可选，结合你的实际需求自由设定。基础参数一般出现在以下几种场景中：

```
1. 场景一：下述方法中的键值即为基础参数项
2. table.render({
3.     height: 300
4.     ,url: '/api/data'
5. });
6.
7. 场景二：下述 lay-data 里面的内容即为基础参数项，切记：值要用单引号
8. <table lay-data="{height:300, url: '/api/data'}" lay-filter="demo"> ..... </table>
9.
10. 更多场景：下述 options 即为含有基础参数项的对象
11. > table.init('filter', options); //转化静态表格
12. > var tableObj = table.render({});
13.     tableObj.reload(options); //重载表格
14.
```

下面是目前 table 模块所支持的全部参数一览表，我们对重点参数进行了详细的说明，你可以点击下述表格最右侧的“示例”去查看

参数	类型	说明	示例值
elem	String/DOM	指定原始 table 容器的选择器或 DOM，方法渲染方式必填	"#demo"
cols	Array	设置表头。值是一个二维数组。方法渲染方式必填	详见表头参数

url (等)	-	异步数据接口相关参数。其中 url 参数为必填项	详见 异步接口
toolbar	String/DOM/Boolean	<p>开启表格头部工具栏区域，该参数支持四种类型值：</p> <ul style="list-style-type: none"> toolbar: '#toolbarDemo' // 指向自定义工具栏模板选择器 toolbar: '<div>xxx</div>' // 直接传入工具栏模板字符 toolbar: true // 仅开启工具栏，不显示左侧模板 toolbar: 'default' // 让工具栏左侧显示默认的内置模板 <p>注意：</p> <ol style="list-style-type: none"> 该参数为 layui 2.4.0 开始新增。 若需要“列显示隐藏”、“导出”、“打印”等功能，则必须开启该参数 	false
defaultToolbar	Array	该参数可自由配置头部工具栏右侧的图标按钮	详见 头工具栏图标
width	Number	设定容器宽度。table容器的默认宽度是跟随它的父元素铺满，你也可以设定一个固定值，当容器中的内容超出了该宽度时，会自动出现横向滚动条。	1000
height	Number/String	设定容器高度	详见 height
cellMinWidth	Number	(layui 2.2.1 新增) 全局定义所有常规单元格的最小宽度（默认：60），一般用于列宽自动分配的情况。其优先级低于表头参数中的 minWidth	100
done	Function	数据渲染完的回调。你可以借此做一些其它的操作	详见 done回调
data	Array	直接赋值数据。既适用于只展示一页数据，也非常适用于对一段已知数据进行多页展示。	[[{}], {}, {}, {}, ...]
totalRow	Boolean	是否开启合计行区域。layui 2.4.0 新增	false
page	Boolean/Object	开启分页（默认：false） 注：从 layui 2.2.0 开始，支持传入一个对象，里面可包含 laypage 组件所有支持的参数（jump、elem除外）	{theme: '#c00'}
limit	Number	每页显示的条数（默认：10）。值务必对应 limits 参数的选项。 注意：优先级低于 page 参数中的 limit 参数	30
limits	Array	每页条数的选择项，默认： [10, 20, 30, 40, 50, 60, 70, 80, 90]。 注意：优先级低于 page 参数中的 limits 参数	[30, 60, 90]
loading	Boolean	是否显示加载条（默认：true）。如果设置 false，则在切换分页时，不会出现加载条。该参数只适用于 url 参数开启的方式	false
title	String	定义 table 的大标题（在文件导出等地方会用到） layui 2.4.0 新增	“用户表”
text	Object	自定义文本，如空数据时的异常提示等。 注：layui 2.2.5 开始新增。	详见 自定义文本

autoSort	Boolean	默认 true，即直接由 table 组件在前端自动处理排序。 若为 false，则需自主排序，通常由服务端直接返回排序好的数据。 注意：该参数为 layui 2.4.4 新增	详见监听排序
initSort	Object	初始排序状态。用于在数据表格渲染完毕时，默认按某个字段排序。	详见初始排序
id	String	设定容器唯一 id。id 是对表格的数据操作方法是必要的传递条件，它是表格容器的索引，你在下文诸多地方都将会见识它的存在。 值得注意的是：从 layui 2.2.0 开始，该参数也可以自动从 <code><table id="test"></table></code> 中的 id 参数中获取。	test
skin (等)	-	设定表格各种外观、尺寸等	详见表格风格

cols - 表头参数一览表

相信我，在你还尚无法驾驭 layui table 的时候，你的所有焦点都应放在这里，它带引领你完成许多可见和不可见甚至你无法想象的工作。如果你采用的是方法渲染，cols 是一个二维数组，表头参数设定在数组内；如果你采用的自动渲染，表头参数的设定应放在 标签上

参数	类型	说明	示例值
field	String	设定字段名。字段名的设定非常重要，且是表格数据列的唯一标识	username
title	String	设定标题名称	用户名
width	Number/String	设定列宽，若不填写，则自动分配；若填写，则支持值为：数字、百分比 请结合实际情况，对不同列做不同设定。	200 30%
minWidth	Number	局部定义当前常规单元格的最小宽度（默认：60），一般用于列宽自动分配的情况。其优先级高于基础参数中的 cellMinWidth	100
type	String	设定列类型。可选值有： <ul style="list-style-type: none">normal（常规列，无需设定）checkbox（复选框列）radio（单选框列，layui 2.4.0 新增）numbers（序号列）space（空列）	任意一个可选值
LAY_CHECKED	Boolean	是否全选状态（默认：false）。必须复选框列开启后才有效，如果设置 true，则表示复选框默认全部选中。	true
fixed	String	固定列。可选值有：left（固定在左）、right（固定在右）。一旦设定，对应的列将会被固定在左或右，不随滚动条而滚动。 注意：如果是固定在左，该列必须放在表头最前面；如果是固定在右，该列必须放在表头最后面。	left（同 true） right

hide	Boolean	是否初始隐藏列，默认：false。layui 2.4.0 新增	true
totalRow	Boolean/Object	<p>是否开启该列的自动合计功能，默认：false。</p> <p>当开启时，则默认由前端自动合计当前行数据。 从 layui 2.5.6 开始：若接口直接返回了合计行数据，则优先读取接口合计行数据，格式如下：</p> <pre> 1. { 2. "code": 0, 3. "msg": "", 4. "count": 1000, 5. "data": [{}, {}] 6. "totalRow": { 7. "score": "666" 8. , "experience": "999" 9. } 10. } 11. </pre> <p>如上，在 totalRow 中返回所需统计的列字段名和值即可。 另外，totalRow 字段同样可以通过 parseData 回调来解析成为 table 组件所规定的数据格式。</p>	true
totalRowText	String	用于显示自定义的合计文本。layui 2.4.0 新增	“合计：”
sort	Boolean	<p>是否允许排序（默认：false）。如果设置 true，则在对应的表头显示排序 icon，从而对列开启排序功能。</p> <p>注意：不推荐对值同时存在“数字和普通字符”的列开启排序，因为会进入字典序比对。比如：‘贤心’ > ‘2’ > ‘100’，这可能并不是你想要的结果，但字典序排列算法（ASCII码比对）就是如此。</p>	true
unresize	Boolean	是否禁用拖拽列宽（默认：false）。默认情况下会根据列类型（type）来决定是否禁用，如复选框列，会自动禁用。而其它普通列，默认允许拖拽列宽，当然你也可以设置 true 来禁用该功能。	false
edit	String	单元格编辑类型（默认不开启）目前只支持：text（输入框）	text
event	String	自定义单元格点击事件名，以便在 tool 事件中完成对该单元格的业务处理	任意字符
style	String	自定义单元格样式。即传入 CSS 样式	background-color: #5FB878; color: #fff;
align	String	单元格排列方式。可选值有：left（默认）、center（居中）、right（居右）	center

colspan	Number	单元格所占列数（默认：1）。一般用于多级表头	3
rowspan	Number	单元格所占行数（默认：1）。一般用于多级表头	2
templet	String	自定义列模板，模板遵循 laytpl 语法。这是一个非常实用的功能，你可借助它实现逻辑处理，以及将原始数据转化成其它格式，如时间戳转化为日期字符等	详见自定义模板
toolbar	String	绑定工具条模板。可在每行对应的列中出现一些自定义的操作性按钮	详见行工具事件

下面是一些方法渲染和自动渲染的配置方式：

```
1. //方法渲染：
2. table.render({
3.   cols: [[ //标题栏
4.     {checkbox: true}
5.     ,{field: 'id', title: 'ID', width: 80}
6.     ,{field: 'username', title: '用户名', width: 120}
7.   ]]
8. });
9.
10. 它等价于自动渲染：
11. <table class="layui-table" lay-data="{基础参数}" lay-filter="test">
12.   <thead>
13.     <tr>
14.       <th lay-data="{checkbox:true}"></th>
15.       <th lay-data="{field:'id', width:80}">ID</th>
16.       <th lay-data="{field:'username', width:180}">用户名</th>
17.     </tr>
18.   </thead>
19. </table>
20.
```

以下是一个二级表头的例子：

```
1. JS：
2. table.render({
3.   cols: [[ //标题栏
4.     {field: 'username', title: '联系人', width: 80, rowspan: 2} //rowspan即纵向跨
      越的单元格数
5.     ,{field: 'amount', title: '金额', width: 80, rowspan: 2}
6.     ,{align: 'center', title: '地址', colspan: 3} //colspan即横跨的单元格数，这种情
      况下不用设置field和width
7.   ], [
8.     {field: 'province', title: '省', width: 80}
9.     ,{field: 'city', title: '市', width: 120}
10.    ,{field: 'county', title: '详细', width: 300}
11.  ]]
12. });
13.
14. 它等价于：
15. <table class="layui-table" lay-data="{基础参数}">
16.   <thead>
```

```

17.     <tr>
18.         <th lay-data="{field:'username', width:80}" rowspan="2">联系人</th>
19.         <th lay-data="{field:'amount', width:120}" rowspan="2">金额</th>
20.         <th lay-data="{align:'center'}" colspan="3">地址</th>
21.     </tr>
22.     <tr>
23.         <th lay-data="{field:'province', width:80}">省</th>
24.         <th lay-data="{field:'city', width:120}">市</th>
25.         <th lay-data="{field:'county', width:300}">详细</th>
26.     </tr>
27. </thead>
28. </table>
29.

```

需要说明的是，table模块支持无限极表头，你可按照上述的方式继续扩充。核心点在于 *rowspan* 和 *colspan* 两个参数的使用。

templet - 自定义列模板

类型：*String*，默认值：无

在默认情况下，单元格的内容是完全按照数据接口返回的content原样输出的，如果你想对某列的单元格添加链接等其它元素，你可以借助该参数来轻松实现。这是一个非常实用且强大的功能，你的表格内容会因此而丰富多样。

templet 提供了三种使用方式，请结合实际场景选择最合适的一种：

- 如果自定义模板的字符量大，我们推荐你采用【方式一】；
- 如果自定义模板的字符量适中，或者想更方便地调用外部方法，我们推荐你采用【方式二】；
- 如果自定义模板的字符量很小，我们推荐你采用【方式三】

方式一：绑定模版选择器。

```

1. table.render({
2.     cols: [[
3.         {field:'title', title: '文章标题', width: 200, templet: '#titleTpl'} //这里的
         templet值是模板元素的选择器
4.         ,{field:'id', title: 'ID', width:100}
5.     ]]
6. });
7.
8. 等价于：
9. <th lay-data="{field:'title', width: 200, templet: '#titleTpl'}">文章标题</th>
10. <th lay-data="{field:'id', width:100}">ID</th>
11.

```

下述是templet对应的模板，它可以存放在页面的任意位置。模板遵循于 *laytpl* 语法，可读取到返回的所有数据

```

1. <script type="text/html" id="titleTpl">
2.     <a href="/detail/{d.id}" class="layui-table-link">{{d.title}}</a>

```

```

3. </script>
4.
5. 注意：上述的 {{d.id}}、{{d.title}} 是动态内容，它对应数据接口返回的字段名。除此之外，你还可以读取到以下额外字段：
    序号：{{ d.LAY_INDEX }} （该额外字段为 layui 2.2.0 新增）
6.
7. 由于模板遵循 laytpl 语法（建议细读 laytpl文档），因此在模板中你可以写任意脚本语句（如 if else/for等）：
8. <script type="text/html" id="titleTpl">
9.     {{# if(d.id < 100){ }}
10.         <a href="/detail/{{d.id}}" class="layui-table-link">{{d.title}}</a>
11.     {{# }} else { }}
12.         {{d.title}}(普通用户)
13.     {{# }} }}
14. </script>
15.

```

方式二：函数转义。自 layui 2.2.5 开始，templet 开始支持函数形式，函数返回一个参数 d，包含接口返回的所有字段和数据。如下所示：

```

1. table.render({
2.     cols: [[
3.         {field: 'title', title: '文章标题', width: 200
4.           ,templet: function(d){
5.               return 'ID: '+ d.id + ', 标题: <span style="color: #c00;">' + d.title
6.           }
7.         }
8.         , {field: 'id', title: 'ID', width: 100}
9.     ]]
10. });
11.

```

方式三：直接赋值模版字符。事实上，templet 也可以直接是一段 html 内容，如：

```

1.
2. templet: '<div><a href="/detail/{{d.id}}" class="layui-table-link">{{d.title}}</a></div>'
3.
4. 注意：这里一定要被一层 <div></div> 包裹，否则无法读取到模板
5.

```

toolbar - 绑定工具条模板

类型：String，默认值：无

通常你需要在表格的每一行加上 查看、编辑、删除 这样类似的操作按钮，而 tool 参数就是为此而生，你因此可以非常便捷地实现各种操作功能。tool 参数和 templet 参数的使用方式完全类似，通常接受的是一个选择器，也可以是一

段HTML字符。

```

1. table.render({
2.   cols: [[
3.     {field:'id', title:'ID', width:100}
4.     ,{fixed: 'right', width:150, align:'center', toolbar: '#barDemo'} //这里的
      toolbar值是模板元素的选择器
5.   ]]
6. });
7.
8. 等价于：
9. <th lay-data="{field:'id', width:100}">ID</th>
10. <th lay-data="{fixed: 'right', width:150, align:'center', toolbar:
      '#barDemo'}"></th>
11.

```

下述是 toolbar 对应的模板，它可以存放在页面的任意位置：

```

1. <script type="text/html" id="barDemo">
2.   <a class="layui-btn layui-btn-xs" lay-event="detail">查看</a>
3.   <a class="layui-btn layui-btn-xs" lay-event="edit">编辑</a>
4.   <a class="layui-btn layui-btn-danger layui-btn-xs" lay-event="del">删除</a>
5.
6.   <!-- 这里同样支持 laytpl 语法, 如: -->
7.   {{# if(d.auth > 2){ }}
8.     <a class="layui-btn layui-btn-xs" lay-event="check">审核</a>
9.   {{# }} }}
10. </script>
11.
12. 注意：属性 lay-event="" 是模板的关键所在，值可随意定义。
13.

```

接下来我们可以借助 table模块的工具条事件，完成不同的操作功能：

```

1. //监听工具条
2. table.on('tool(test)', function(obj){ //注：tool 是工具条事件名，test 是 table 原始
      容器的属性 lay-filter="对应的值"
3.   var data = obj.data; //获得当前行数据
4.   var layEvent = obj.event; //获得 lay-event 对应的值（也可以是表头的 event 参数对应
      的值）
5.   var tr = obj.tr; //获得当前行 tr 的 DOM 对象（如果有的话）
6.
7.   if(layEvent === 'detail'){ //查看
8.     //do something
9.   } else if(layEvent === 'del'){ //删除
10.    layer.confirm('真的删除行么', function(index){
11.      obj.del(); //删除对应行（tr）的DOM结构，并更新缓存
12.      layer.close(index);
13.      //向服务端发送删除指令
14.    });
15.   } else if(layEvent === 'edit'){ //编辑
16.     //do something
17.

```

```
18.      //同步更新缓存对应的值
19.      obj.update({
20.          username: '123'
21.          ,title: 'xxx'
22.      });
23.  } else if(layEvent === 'LAYTABLE_TIPS'){
24.      layer.alert('Hi, 头部工具栏扩展的右侧图标。');
25.  }
26. });
27.
```

异步数据接口

数据的异步请求由以下几个参数组成：

参数名	功能
url	接口地址。默认会自动传递两个参数： <i>?page=1&limit=30</i> （该参数可通过 request 自定义） <i>page</i> 代表当前页码、 <i>limit</i> 代表每页数据量
method	接口http请求类型，默认：get
where	接口的其它参数。如： <i>where: {token: 'sasasas', id: 123}</i>
contentType	发送到服务端的内容编码类型。如果你要发送 json 内容，可以设置： <i>contentType: 'application/json'</i>
headers	接口的请求头。如： <i>headers: {token: 'sasasas'}</i>
parseData	<p>数据格式解析的回调函数，用于将返回的任意数据格式解析成 table 组件规定的数据格式。</p> <p>table 组件默认规定的数据格式为（参考：/demo/table/user/）：</p> <pre>1. { 2. "code": 0, 3. "msg": "", 4. "count": 1000, 5. "data": [{}, {}] 6. } 7.</pre> <p>很多时候，您接口返回的数据格式并不一定都符合 table 默认规定的格式，比如：</p> <pre>1. { 2. "status": 0, 3. "message": "", 4. "total": 180, 5. "data": { 6. "item": [{}, {}] 7. } 8. } 9.</pre>

那么你需要借助 `parseData` 回调函数将其解析成 `table` 组件所规定的数据格式

```
1. table.render({
2.   elem: '#demp'
3.   ,url: ''
4.   ,parseData: function(res){ //res 即为原始返回的数据
5.     return {
6.       "code": res.status, //解析接口状态
7.       "msg": res.message, //解析提示文本
8.       "count": res.total, //解析数据长度
9.       "data": res.data.item //解析数据列表
10.    };
11.  }
12.  //,..... //其他参数
13. });
```

该参数非常实用，系 layui 2.4.0 开始新增

request

用于对分页请求的参数：`page`、`limit`重新设定名称，如：

```
1. table.render({
2.   elem: '#demp'
3.   ,url: ''
4.   ,request: {
5.     pageNum: 'curr' //页码的参数名称，默认：page
6.     ,limitName: 'nums' //每页数据量的参数名，默认：limit
7.   }
8.   //,..... //其他参数
9. });
```

那么请求数据时的参数将会变为：`?curr=1&nums=30`

response

您还可以借助 `response` 参数来重新设定返回的数据格式，如：

```
1. table.render({
2.   elem: '#demp'
3.   ,url: ''
4.   ,response: {
5.     statusName: 'status' //规定数据状态的字段名称，默认：code
6.     ,statusCode: 200 //规定成功的状态码，默认：0
7.     ,msgName: 'hint' //规定状态信息的字段名称，默认：msg
8.     ,countName: 'total' //规定数据总数的字段名称，默认：count
9.     ,dataName: 'rows' //规定数据列表的字段名称，默认：data
10.  }
11.  //,..... //其他参数
12. });
```

那么上面所规定的格式为：

```
1. {
2.   "status": 200,
3.   "hint": "",
4.   "total": 1000,
```

```
5.   "rows": []
6. }
7.
```

注意：*request* 和 *response* 参数均为 *layui 2.1.0* 版本新增

调用示例：

```
1.  //“方法级渲染”配置方式
2.  table.render({ //其它参数在此省略
3.    url: '/api/data/'
4.    //where: {token: 'sasasas', id: 123} //如果无需传递额外参数，可不加该参数
5.    //method: 'post' //如果无需自定义HTTP类型，可不加该参数
6.    //request: {} //如果无需自定义请求参数，可不加该参数
7.    //response: {} //如果无需自定义数据响应名称，可不加该参数
8.  });
9.
10.  等价于（“自动化渲染”配置方式）
11.  <table class="layui-table" lay-data="{url: '/api/data/'}" lay-filter="test"> .....
12.  </table>
```

done - 数据渲染完的回调

类型：*Function*，默认值：无

无论是异步请求数据，还是直接赋值数据，都会触发该回调。你可以利用该回调做一些表格以外元素的渲染。

```
1.  table.render({ //其它参数在此省略
2.    done: function(res, curr, count){
3.      //如果是异步请求数据方式，res即为你接口返回的信息。
4.      //如果是直接赋值的方式，res即为：{data: [], count: 99} data为当前页数据、count为数据总长度
5.      console.log(res);
6.
7.      //得到当前页码
8.      console.log(curr);
9.
10.     //得到数据总量
11.     console.log(count);
12.   }
13. });
14.
```

defaultToolbar - 头部工具栏右侧图标

类型：*Array*，默认值：*["filter", "exports", "print"]*

该参数可自由配置头部工具栏右侧的图标按钮，值为一个数组，支持以下可选值：

- filter：显示筛选图标
- exports：显示导出图标
- print：显示打印图标

可根据值的顺序显示排版图标，如：

```
defaultToolbar: ['filter', 'print', 'exports']
```

另外你还可以无限扩展图标按钮（layui 2.5.5 新增）：

```
1. table.render({ //其它参数在此省略
2.   defaultToolbar: ['filter', 'print', 'exports', {
3.     title: '提示' //标题
4.     ,layEvent: 'LAYTABLE_TIPS' //事件名，用于 toolbar 事件中使用
5.     ,icon: 'layui-icon-tips' //图标类名
6.   }]
7. });
8.
```

扩展的图标可通过 toolbar 事件监听（[详见行工具事件](#)），其中 layEvent 的值会在事件的回调参数中返回，以便区分不同的触发动作。

text - 自定义文本

类型：Object

table 模块会内置一些默认的文本信息，如果想修改，你可以设置以下参数达到目的。

```
1. table.render({ //其它参数在此省略
2.   text: {
3.     none: '暂无相关数据' //默认：无数据。注：该属性为 layui 2.2.5 开始新增
4.   }
5. });
6.
```

initSort - 初始排序

类型：Object，默认值：无

用于在数据表格渲染完毕时，默认按某个字段排序。注：该参数为 layui 2.1.1 新增

```
1. //“方法级渲染”配置方式
2. table.render({ //其它参数在此省略
3.   initSort: {
```



```
4.     field: 'id' //排序字段, 对应 cols 设定的各字段名
5.     ,type: 'desc' //排序方式  asc: 升序、desc: 降序、null: 默认排序
6.   }
7. });
8.
9. 等价于 (“自动化渲染”配置方式)
10. <table class="layui-table" lay-data="{initSort:{field:'id', type:'desc'}}" lay-
    filter="test"> ..... </table>
11.
```

height - 设定容器高度

类型：*Number/String*，可选值如下：

可选值	说明	示例
不填写	默认情况。高度随数据列表而适应，表格容器不会出现纵向滚动条	-
固定值	设定一个数字，用于定义容器高度，当容器中的内容超出了该高度时，会自动出现纵向滚动条	height: 315
full-差值	高度将始终铺满，无论浏览器尺寸如何。这是一个特定的语法格式，其中 <i>full</i> 是固定的，而 差值 则是一个数值，这需要你来预估，比如：表格容器距离浏览器顶部和底部的距离“和” PS：该功能为 <i>layui 2.1.0</i> 版本中新增	height: 'full-20'

示例：

```
1.  //“方法级渲染”配置方式
2.  table.render({ //其它参数在此省略
3.    height: 315 //固定值
4.  });
5.  table.render({ //其它参数在此省略
6.    height: 'full-20' //高度最大化减去差值
7.  });
8.
9. 等价于 (“自动化渲染”配置方式)
10. <table class="layui-table" lay-data="{height:315}" lay-filter="test"> .....
    </table>
11. <table class="layui-table" lay-data="{height:'full-20'}" lay-filter="test"> .....
    </table>
12.
```

设定表格外观

控制表格外观的主要由以下参数组成：

参数名	可选值	备注
skin	line （行边框风格） row （列边框风格） nob （无边框风格）	用于设定表格风格，若使用默认风格不设置该属性即可
even	true/false	若不开启隔行背景，不设置该参数即可

size	sm (小尺寸) lg (大尺寸)	用于设定表格尺寸，若使用默认尺寸不设置该属性即可
------	----------------------	--------------------------

```

1.  //“方法级渲染”配置方式
2.  table.render({ //其它参数在此省略
3.    skin: 'line' //行边框风格
4.    ,even: true //开启隔行背景
5.    ,size: 'sm' //小尺寸的表格
6.  });
7.
8.  等价于（“自动化渲染”配置方式）
9.  <table class="layui-table" lay-data="{skin:'line', even:true, size:'sm'}" lay-
    filter="test"> ..... </table>
10.

```

基础方法

基础用法是table模块的关键组成部分，目前所开放的所有方法如下：

```

1. > table.set(options); //设定全局默认参数。options即各项基础参数
2. > table.on('event(filter)', callback); //事件监听。event为内置事件名（详见下文），
    filter为容器lay-filter设定的值
3. > table.init(filter, options); //filter为容器lay-filter设定的值，options即各项基础
    参数。例子见：转换静态表格
4. > table.checkStatus(id); //获取表格选中行（下文会有详细介绍）。id 即为 id 参数对应的值
5. > table.render(options); //用于表格方法级渲染，核心方法。应该不用再过多解释了，详见：方法
    级渲染
6. > table.reload(id, options); //表格重载
7. > table.resize(id); //重置表格尺寸
8. > table.exportFile(id, data, type); //导出数据
9.

```

获取选中行

该方法可获取到表格所有的选中行相关数据

语法：`table.checkStatus('ID')`，其中 `ID` 为基础参数 `id` 对应的值（见：[设定容器唯一ID](#)），如：

```

1.  【自动化渲染】
2.  <table class="layui-table" lay-data="{id: 'idTest'}"> ..... </table>
3.
4.  【方法渲染】
5.  table.render({ //其它参数省略
6.    id: 'idTest'
7.  });
8.

```

```

1.  var checkStatus = table.checkStatus('idTest'); //idTest 即为基础参数 id 对应的值

```

```

2.
3. console.log(checkStatus.data) //获取选中行的数据
4. console.log(checkStatus.data.length) //获取选中行数量，可作为是否有选中行的条件
5. console.log(checkStatus.isAll ) //表格是否全选
6.

```

重置表格尺寸

该方法可重置表格尺寸和结构，其内部完成了：固定列高度平铺、动态分配列宽、容器滚动条宽高补丁 等操作。它一般用于特殊情况下（如“非窗口 `resize`”导致的表格父容器宽度变化而引发的列宽适配异常），以保证表格在此类特殊情况下依旧能友好展示。

语法：`table.resize('ID')`，其中 `ID` 为基础参数 `id` 对应的值（见：[设定容器唯一ID](#)），如：

```

1. table.render({ //其它参数省略
2.   ,elem: '#demo'
3.   //,..... //其它参数
4.   ,id: 'idTest'
5. });
6.
7. //执行表格“尺寸结构”的重置，一般写在对应的事件中
8. table.resize('idTest');
9.

```

表格重载

很多时候，你需要对表格进行重载。比如数据全局搜索。以下方法可以帮你轻松实现这类需求（可任选一种）。

语法	说明	适用场景
<code>table.reload(ID, options)</code>	参数 <code>ID</code> 即为基础参数 <code>id</code> 对应的值，见： 设定容器唯一ID 参数 <code>options</code> 即为各项基础参数	所有渲染方式
<code>tableIns.reload(options)</code>	参数同上 <code>tableIns</code> 可通过 <code>var tableIns = table.render()</code> 得到	仅限方法级渲染

```

1. 【HTML】
2. <table class="layui-table" lay-data="{id: 'idTest'}"> ..... </table>
3.
4. 【JS】
5. table.reload('idTest', {
6.   url: '/api/table/search'
7.   ,where: {} //设定异步数据接口的额外参数
8.   //,height: 300

```

```

9. });
10.

```

```

1. //所获得的 tableIns 即为当前容器的实例
2. var tableIns = table.render({
3.   elem: '#id'
4.   ,cols: [] //设置表头
5.   ,url: '/api/data' //设置异步接口
6.   ,id: 'idTest'
7. });
8.
9. //这里以搜索为例
10. tableIns.reload({
11.   where: { //设定异步数据接口的额外参数，任意设
12.     aaaaaa: 'xxx'
13.     ,bbb: 'yyy'
14.     //...
15.   }
16.   ,page: {
17.     curr: 1 //重新从第 1 页开始
18.   }
19. });
20. //上述方法等价于
21. table.reload('idTest', {
22.   where: { //设定异步数据接口的额外参数，任意设
23.     aaaaaa: 'xxx'
24.     ,bbb: 'yyy'
25.     //...
26.   }
27.   ,page: {
28.     curr: 1 //重新从第 1 页开始
29.   }
30. }); //只重载数据
31.

```

注意：这里的表格重载是指对表格重新进行渲染，包括数据请求和基础参数的读取

导出任意数据

尽管 table 的工具栏内置了数据导出按钮，但有时你可能需要通过方法去导出任意数据，那么可以借助以下方法：

语法：`table.exportFile(id, data, type)`

```

1. var ins1 = table.render({
2.   elem: '#demo'
3.   ,id: 'test'
4.   //,..... //其它参数
5. })
6.
7. //将上述表格示例导出为 csv 文件

```

```
8. table.exportFile(ins1.config.id, data); //data 为该实例中的任意数量的数据
9.
```

事实上，该方法也可以不用依赖 table 的实例，可直接导出任意数据：

```
1. table.exportFile(['名字', '性别', '年龄'], [
2.   ['张三', '男', '20'],
3.   ['李四', '女', '18'],
4.   ['王五', '女', '19']
5. ], 'csv'); //默认导出 csv, 也可以为 :xls
6.
```

事件监听

语法：`table.on('event(filter)', callback)`；注：event为内置事件名，filter为容器lay-filter设定的值
table模块在Layui事件机制中注册了专属事件，如果你使用layui.onevent()自定义模块事件，请勿占用table名。目前所支持的所有事件见下文

默认情况下，事件所监听的是全部的table模块容器，但如果你只想监听某一个容器，使用事件过滤器即可。

假设原始容器为：

那么你的事件监听写法如下：

```
1. //以复选框事件为例
2. table.on('checkbox(test)', function(obj){
3.   console.log(obj)
4. });
5.
```

监听头部工具栏事件

点击头部工具栏区域设定了属性为 `lay-event=""` 的元素时触发（该事件为layui 2.4.0 开始新增）。如：

```
1. 原始容器
2. <table id="demo" lay-filter="test"></table>
3.
4. 工具栏模板：
5. <script type="text/html" id="toolbarDemo">
6.   <div class="layui-btn-container">
7.     <button class="layui-btn layui-btn-sm" lay-event="add">添加</button>
8.     <button class="layui-btn layui-btn-sm" lay-event="delete">删除</button>
9.     <button class="layui-btn layui-btn-sm" lay-event="update">编辑</button>
```

```

10.     </div>
11. </script>
12.
13. <script;>
14.   //JS 调用 :
15.   table.render({
16.     elem: '#demo'
17.     ,toolbar: '#toolbarDemo'
18.     //,... //其他参数
19.   });
20.
21.   //监听事件
22.   table.on('toolbar(test)', function(obj){
23.     var checkStatus = table.checkStatus(obj.config.id);
24.     switch(obj.event){
25.       case 'add':
26.         layer.msg('添加');
27.         break;
28.       case 'delete':
29.         layer.msg('删除');
30.         break;
31.       case 'update':
32.         layer.msg('编辑');
33.         break;
34.     };
35.   });
36. </script>
37.

```

监听复选框选择

点击复选框时触发，回调函数返回一个object参数，携带的成员如下：

```

1. table.on('checkbox(test)', function(obj){
2.   console.log(obj.checked); //当前是否选中状态
3.   console.log(obj.data); //选中行的相关数据
4.   console.log(obj.type); //如果触发的是全选，则为：all，如果触发的是单选，则为：one
5. });
6.

```

监听单元格编辑

单元格被编辑，且值发生改变时触发，回调函数返回一个object参数，携带的成员如下：

```

1. table.on('edit(test)', function(obj){ //注：edit是固定事件名，test是table原始容器的
   属性 lay-filter="对应的值"
2.   console.log(obj.value); //得到修改后的值
3.   console.log(obj.field); //当前编辑的字段名

```

```

4.     console.log(obj.data); //所在行的所有相关数据
5. });
6.

```

监听行单双击事件

点击或双击行时触发。该事件为 layui 2.4.0 开始新增

```

1. //监听行单击事件
2. table.on('row(test)', function(obj){
3.     console.log(obj.tr) //得到当前行元素对象
4.     console.log(obj.data) //得到当前行数据
5.     //obj.del(); //删除当前行
6.     //obj.update(fields) //修改当前行数据
7. });
8.
9. //监听行双击事件
10. table.on('rowDouble(test)', function(obj){
11.     //obj 同上
12. });
13.

```

监听行中工具条点击事件

具体用法见：[绑定工具条](#)

监听排序切换

点击表头排序时触发，它通用在基础参数中设置 `autoSort: false` 时使用，以完成服务端的排序，而不是默认的前端排序。该事件的回调函数返回一个 `object` 参数，携带的成员如下：

```

1. //禁用前端自动排序，以便由服务端直接返回排序好的数据
2. table.render({
3.     elem: '#id'
4.     ,autoSort: false //禁用前端自动排序。注意：该参数为 layui 2.4.4 新增
5.     //,... //其它参数省略
6. });
7.
8. //监听排序事件
9. table.on('sort(test)', function(obj){ //注：sort 是工具条事件名，test 是 table 原始容器的属性 lay-filter="对应的值"
10.     console.log(obj.field); //当前排序的字段名
11.     console.log(obj.type); //当前排序类型：desc（降序）、asc（升序）、null（空对象，默认排序）
12.     console.log(this); //当前排序的 th 对象
13.

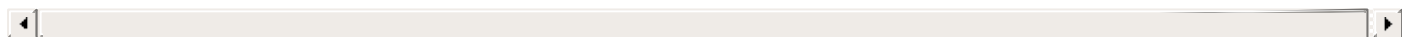
```

```
14. //尽管我们的 table 自带排序功能，但并没有请求服务端。
15. //有些时候，你可能需要根据当前排序的字段，重新向服务端发送请求，从而实现服务端排序，如：
16. table.reload('idTest', {
17.   initSort: obj //记录初始排序，如果不设的话，将无法标记表头的排序状态。
18.   ,where: { //请求参数（注意：这里面的参数可任意定义，并非下面固定的格式）
19.     field: obj.field //排序字段
20.     ,order: obj.type //排序方式
21.   }
22. });
23.
24. layer.msg('服务端排序。order by '+ obj.field + ' ' + obj.type);
25. });
26.
```

结语

table 模块自推出以来，因某些功能的缺失，一度饱受非议，也背负了各种莫须有的锅，然而我始终未曾放弃对它的优化。为了迎合 layui 开箱即用的理念，我的工作并不是那么轻松。无论是从代码，还是文档和示例的撰写上，我都进行了多次调整，为的只是它能被更多人认可。——贤心

layui - 用心与你沟通



表单模块文档 - layui.form

我们通常会在最常用的模块上耗费更多的精力，用尽可能简单的方式诠释 layui 所带来的便捷性。显而易见，form 是我们非常看重的一块。于是它试图用一贯极简的姿态，去捕获你对它的深深青睐。寄托在原始表单元素上的属性设定，及其全自动的初始渲染，和基于事件驱动的接口书写方式，会让你觉得，传统模式下的组件调用形式，也可以是那样的优雅、简单。然而文字的陈述始终是苍白的，所以用行动慢慢感受 layui.form 给你的项目带来的效率提升吧。

模块加载名称: *form*

使用

layui 针对各种表单元素做了较为全面的UI支持，你无需去书写那些 UI 结构，你只需要写 HTML 原始的 input、select、textarea 这些基本的标签即可。我们在 UI 上的渲染只要求一点，你必须给表单体系所在的父元素加上 `class="layui-form"`，一切的工作都会在你加载完form模块后，自动完成。如下是一个最基本的例子：

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. <meta charset="utf-8">
5. <title>layui.form小例子</title>
6. <link rel="stylesheet" href="layui.css" media="all">
7. </head>
8. <body>
9. <form class="layui-form"> <!-- 提示：如果你不想用form，你可以换成div等任何一个普通元素 -->
10.   <div class="layui-form-item">
11.     <label class="layui-form-label">输入框</label>
12.     <div class="layui-input-block">
13.       <input type="text" name="" placeholder="请输入" autocomplete="off"
class="layui-input">
14.     </div>
15.   </div>
16.   <div class="layui-form-item">
17.     <label class="layui-form-label">下拉选择框</label>
18.     <div class="layui-input-block">
19.       <select name="interest" lay-filter="aihao">
20.         <option value="0">写作</option>
21.         <option value="1">阅读</option>
22.       </select>
23.     </div>
24.   </div>
25.   <div class="layui-form-item">
26.     <label class="layui-form-label">复选框</label>
27.     <div class="layui-input-block">
28.       <input type="checkbox" name="like[write]" title="写作">
29.       <input type="checkbox" name="like[read]" title="阅读">
30.     </div>
```

```

31.     </div>
32.     <div class="layui-form-item">
33.         <label class="layui-form-label">开关关</label>
34.         <div class="layui-input-block">
35.             <input type="checkbox" lay-skin="switch">
36.         </div>
37.     </div>
38.     <div class="layui-form-item">
39.         <label class="layui-form-label">开关开</label>
40.         <div class="layui-input-block">
41.             <input type="checkbox" checked lay-skin="switch">
42.         </div>
43.     </div>
44.     <div class="layui-form-item">
45.         <label class="layui-form-label">单选框</label>
46.         <div class="layui-input-block">
47.             <input type="radio" name="sex" value="0" title="男">
48.             <input type="radio" name="sex" value="1" title="女" checked>
49.         </div>
50.     </div>
51.     <div class="layui-form-item layui-form-text">
52.         <label class="layui-form-label">请填写描述</label>
53.         <div class="layui-input-block">
54.             <textarea placeholder="请输入内容" class="layui-textarea"></textarea>
55.         </div>
56.     </div>
57.     <div class="layui-form-item">
58.         <div class="layui-input-block">
59.             <button class="layui-btn" lay-submit lay-filter="*">立即提交</button>
60.             <button type="reset" class="layui-btn layui-btn-primary">重置</button>
61.         </div>
62.     </div>
63.     <!-- 更多表单结构排版请移步文档左侧【页面元素-表单】一项阅览 -->
64. </form>
65. <script src="layui.js"></script>
66. <script>
67. layui.use('form', function(){
68.     var form = layui.form;
69.
70.     //各种基于事件的操作，下面会有进一步介绍
71. });
72. </script>
73. </body>
74. </html>
75.

```

正如你上述看到的，你必须放入 layui 所规范的元素结构，form 模块才会对其进行重置渲染。值得注意的是，在具体的每一块表单元素中，你仍是像往日一样写 input 等标签即可。另外，我们对我们所规范的结构进行了响应式的支持，而针对一些不同的表单排版，比如行内表单，你也只需要设定我们所定义好的 class 即可。关于这一块，你可以移步页面元素的表单文档中做详细了解。而事实上你的

大部分焦点可能也在那里，但当前这篇文档主要还是介绍 form 整体模块。

更新渲染

有些时候，你的有些表单元素可能是动态插入的。这时 form 模块 的自动化渲染是会对其失效的。虽然我们没有双向绑定机制（因为我们叫经典模块化框架，偷笑.gif） 但没有关系，你只需要执行 `form.render(type, filter);` 方法即可。

第一个参数：type，为表单的 type 类型，可选。默认对全部类型的表单进行一次更新。可局部刷新的 type 如下表：

参数（type）值	描述
select	刷新select选择框渲染
checkbox	刷新checkbox复选框（含开关）渲染
radio	刷新radio单选框框渲染

```
1. form.render(); //更新全部
2. form.render('select'); //刷新select选择框渲染
3. //.....
4.
```

第二个参数：filter，为 `class="layui-form"` 所在元素的 `lay-filter=""` 的值。你可以借助该参数，对表单完成局部更新。

```
1. 【HTML】
2. <div class="layui-form" lay-filter="test1">
3.   ...
4. </div>
5.
6. <div class="layui-form" lay-filter="test2">
7.   ...
8. </div>
9.
10. 【JavaScript】
11. form.render(null, 'test1'); //更新 lay-filter="test1" 所在容器内的全部表单状态
12. form.render('select', 'test2'); //更新 lay-filter="test2" 所在容器内的全部 select
    状态
13. //.....
14.
```

预设元素属性

事实上在使用表单时，你的一半精力可能会在元素本身上。所以我们把一些基础属性的配置恰恰安放在了标签本身上。如：

```
1. <input type="text" lay-verify="email">
2. <input type="checkbox" checked lay-skin="switch" lay-filter="encrypt" title="是否加密">
3. <button lay-submit>提交</button>
4.
```

上述的`lay-verify`、`lay-skin`、`lay-filter`、`lay-submit`神马的都是我们所说的预设的元素属性，他们可以使得一些交互操作交由form模块内部、或者你来借助form提供的JS接口精确控制。目前我们可支持的属性如下表所示：

属性名	属性值	说明
title	任意字符	设定元素名称，一般用于checkbox、radio框
lay-skin	switch（开关风格） primary（原始风格）	定义元素的风格，目前只对 <code>checkbox</code> 元素有效，可将其转变为开关样式
lay-ignore	任意字符或不设值	是否忽略元素美化处理。设置后，将不会对该元素进行初始化渲染，即保留系统风格
lay-filter	任意字符	事件过滤器，主要用于事件的精确匹配，跟选择器是比较类似的。其实它并不私属于form模块，它在 <code>layui</code> 的很多基于事件的接口中都会应用到。
lay-verify	required（必填项） phone（手机号） email（邮箱） url（网址） number（数字） date（日期） identity（身份证） 自定义值	同时支持多条规则的验证，格式： <code>lay-verify="验证A 验证B"</code> 如： <code>lay-verify="required phone number"</code> 另外，除了我们内置的校验规则，你还可以给他设定任意的值，比如 <code>lay-verify="pass"</code> ，那么你就需要借助 <code>form.verify()</code> 方法对 <code>pass</code> 进行一个校验规则的定义。 详见表单验证
lay-verType	tips（吸附层） alert（对话框） msg（默认）	用于定义异常提示层模式。
lay-reqText	任意字符	用于自定义必填项（即设定了 <code>lay-verify="required"</code> 的表单）的提示文本 注意：该功能为 <code>layui 2.5.0</code> 新增
lay-submit	无需填写值	绑定触发提交的元素，如button

事件监听

语法：`form.on('event(过滤器值)', callback);`

form模块在 `layui` 事件机制中注册了专属事件，所以当你使用 `layui.onevent()` 自定义模块事件时，请勿占用form名。form支持的事件如下：

event	描述

select	监听select下拉选择事件
checkbox	监听checkbox复选框勾选事件
switch	监听checkbox复选框开关事件
radio	监听radio单选框事件
submit	监听表单提交事件

默认情况下，事件所监听的是全部的form模块元素，但如果你只想监听某一个元素，使用事件过滤器即可。

如：

```
1. form.on('select(test)', function(data){
2.   console.log(data);
3. });
4.
```

监听select选择

下拉选择框被选中时触发，回调函数返回一个object参数，携带两个成员：

```
1. form.on('select(filter)', function(data){
2.   console.log(data.elem); //得到select原始DOM对象
3.   console.log(data.value); //得到被选中的值
4.   console.log(data.othis); //得到美化后的DOM对象
5. });
6.
```

请注意：如果你想监听所有的select，去掉过滤器(*filter*)即可。下面将不再对此进行备注。

监听checkbox复选

复选框点击勾选时触发，回调函数返回一个object参数，携带两个成员：

```
1. form.on('checkbox(filter)', function(data){
2.   console.log(data.elem); //得到checkbox原始DOM对象
3.   console.log(data.elem.checked); //是否被选中， true或者false
4.   console.log(data.value); //复选框value值， 也可以通过data.elem.value得到
5.   console.log(data.othis); //得到美化后的DOM对象
6. });
7.
```

监听switch开关

开关被点击时触发，回调函数返回一个object参数，携带两个成员：

```
1. form.on('switch(filter)', function(data){
2.   console.log(data.elem); //得到checkbox原始DOM对象
3.   console.log(data.elem.checked); //开关是否开启, true或者false
4.   console.log(data.value); //开关value值, 也可以通过data.elem.value得到
5.   console.log(data.othis); //得到美化后的DOM对象
6. });
7.
```

监听radio单选

radio单选框被点击时触发，回调函数返回一个object参数，携带两个成员：

```
1. form.on('radio(filter)', function(data){
2.   console.log(data.elem); //得到radio原始DOM对象
3.   console.log(data.value); //被点击的radio的value值
4. });
5.
```

监听submit提交

按钮点击或者表单被执行提交时触发，其中回调函数只有在验证全部通过后才会进入，回调返回三个成员：

```
1. form.on('submit(*)', function(data){
2.   console.log(data.elem) //被执行事件的元素DOM对象，一般为button对象
3.   console.log(data.form) //被执行提交的form对象，一般在存在form标签时才会返回
4.   console.log(data.field) //当前容器的全部表单字段，名值对形式：{name: value}
5.   return false; //阻止表单跳转。如果需要表单跳转，去掉这段即可。
6. });
7.
```

再次温馨提示：上述的`submit(\)`中的 `**` 号为事件过滤器的值，是在你绑定执行提交的元素时设定的，如：

```
1. <button lay-submit lay-filter="*">提交</button>
2.
```

你可以把*号换成任意的值，如：`lay-filter="go"`，但监听事件时也要改成`form.on('submit(go)', callback);`

表单赋值 / 取值

语法: `form.val('filter', object);`

用于给指定表单集合的元素赋值和取值。如果 `object` 参数存在，则为赋值；如果 `object` 参数不存在，则为取值。

注：其中「取值」功能为 layui 2.5.5 开始新增

```

1.  //给表单赋值
2.  form.val("formTest", { //formTest 即 class="layui-form" 所在元素属性 lay-
    filter="" 对应的值
3.    "username": "贤心" // "name": "value"
4.    , "sex": "女"
5.    , "auth": 3
6.    , "check[write]": true
7.    , "open": false
8.    , "desc": "我爱layui"
9.  });
10.
11. //获取表单区域所有值
12. var data1 = form.val("formTest");
13.

```

第二个参数中的键值是表单元素对应的 `name` 和 `value`。

表单验证

我们对表单的验证进行了非常巧妙的支持，大多数时候你只需要在表单元素上加上 `lay-verify=""` 属性值即可。如：

```

1. <input type="text" lay-verify="email">
2.
3. 还同时支持多条规则的验证，如下：
4. <input type="text" lay-verify="required|phone|number">
5.

```

上述对输入框定义了一个邮箱规则的校验，它会在 `form` 模块内部完成。目前我们内置的校验支持见上文的：[预设元素属性](#)

除了内置的校验规则外，你还可以自定义验证规则，通常对于比较复杂的校验，这是非常有必要的。

```

1. form.verify({
2.   username: function(value, item){ //value：表单的值、item：表单的DOM对象
3.     if(!new RegExp("^[a-zA-Z0-9_\u4e00-\u9fa5\\s]+$").test(value)){
4.       return '用户名不能有特殊字符';
5.     }
6.     if(/(^_|$|_|$)/.test(value)){
7.       return '用户名首尾不能出现下划线\'_\'';
8.     }
9.     if(/^d+|d+$/.test(value)){

```

```
10.         return '用户名不能全为数字';
11.     }
12. }
13.
14. //我们既支持上述函数式的方式，也支持下述数组的形式
15. //数组的两个值分别代表：[正则匹配、匹配不符时的提示文字]
16. ,pass: [
17.     /^[\\S]{6,12}$/
18.     , '密码必须6到12位，且不能出现空格'
19. ]
20. });
21.
```

当你自定义了类似上面的验证规则后，你只需要把 `key` 赋值给输入框的 `lay-verify` 属性即可：

```
1. <input type="text" lay-verify="username" placeholder="请输入用户名">
2. <input type="password" lay-verify="pass" placeholder="请输入密码">
3.
```

再次温馨提示一下

针对一些不同的表单排版，比如行内表单、整体表单风格、按钮风格等等，请移步到文档导航左侧【页面元素】下的指定分类即可。

layui - 用心与你沟通

图片/文件上传 - layui.upload

上传模块自 layui 2.0 的版本开始，进行了全面重写，这使得它不再那么单一，它所包含的不仅是更为强劲的功能，还有灵活的UI。任何元素都可以作为上传组件来调用，譬如按钮、图片、普通的DIV等等，而不再是一个单调的file文件域。

模块加载名称：*upload*

快速使用

一切从小试牛刀开始。通常情况下，我们上传文件是借助 type="file" 的 input 标签来完成的，但非常遗憾的是，它不能很好地与其它表单元素并存，所以我们常常要单独为它做一个业务层面的“异步上传”，即先让图片上传，再和其它表单一起提交保存。下面是一个小示例：

上传图片

这原本只是一个普通的 button，正是 upload 模块赋予了它“文件选择”的特殊技能。当然，你还可以随意定制它的样式，而不是只局限于按钮。

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4.   <meta charset="utf-8">
5.   <title>upload模块快速使用</title>
6.   <link rel="stylesheet" href="/static/build/layui.css" media="all">
7. </head>
8. <body>
9.
10. <button type="button" class="layui-btn" id="test1">
11.   <i class="layui-icon">&#xe67c;</i>上传图片
12. </button>
13.
14. <script src="/static/build/layui.js"></script>
15. <script>
16. layui.use('upload', function(){
17.   var upload = layui.upload;
18.
19.   //执行实例
20.   var uploadInst = upload.render({
21.     elem: '#test1' //绑定元素
22.     ,url: '/upload/' //上传接口
23.     ,done: function(res){
24.       //上传完毕回调
25.     }
```

```

26.         ,error: function(){
27.             //请求异常回调
28.         }
29.     });
30. });
31. </script>
32. </body>
33. </html>

```

一切看起来是那样的简单，乃至我不得不凑一段文字来填充这一行的版面。这样好像与下文衔接起来会比较谐调的样子（自我感觉）

核心方法与基础参数选项

使用 upload 模块必须与 `upload.render(options)` 方法打交道，其中的 `options` 即为基础参数，它是一个对象。

```

1.  var upload = layui.upload; //得到 upload 对象
2.
3.  //创建一个上传组件
4.  upload.render({
5.      elem: '#id'
6.      ,url: ''
7.      ,done: function(res, index, upload){ //上传后的回调
8.
9.      }
10.     //,accept: 'file' //允许上传的文件类型
11.     //,size: 50 //最大允许上传的文件大小
12.     //,.....
13. })
14.

```

从 layui 2.1.0 开始，允许你直接在元素上设定基础参数，如：

```

1.  【HTML】
2.  <button class="layui-btn test" lay-data="{url: '/a/'}">上传图片</button>
3.  <button class="layui-btn test" lay-data="{url: '/b/', accept: 'file'}">上传文件
4.  </button>
5.  【JS】
6.  upload.render({
7.      elem: '.test'
8.      ,done: function(res, index, upload){
9.          //获取当前触发上传的元素，一般用于 elem 绑定 class 的情况，注意：此乃 layui 2.1.0 新
10.         增
11.         var item = this.item;
12.     }
13. })

```

更多支持的参数详见下表，合理的配置它们，应对各式各样的业务需求。

参数选项	说明	类型	默认值
elem	指向容器选择器，如：elem: '#id'。 也可以是DOM对象	string/object	-
url	服务端上传接口，返回的数据规范请详见下文	string	-
data	请求上传接口的额外参数。如：data: {id: 'xxx'} 从 layui 2.2.6 开始，支持动态值，如： <pre> 1. data: { 2. id: function(){ 3. return \$('#id').val(); 4. } 5. } 6. </pre>	object	-
headers	接口的请求头。如：headers: {token: 'sasasas'}。注：该参数为 layui 2.2.6 开始新增		
accept	指定允许上传时校验的文件类型，可选值有：images（图片）、file（所有文件）、video（视频）、audio（音频）	string	images
acceptMime	规定打开文件选择框时，筛选出的文件类型，值为用逗号隔开的 MIME 类型列表。如： acceptMime: 'image/*'（只显示图片文件） acceptMime: 'image/jpg, image/png'（只显示 jpg 和 png 文件） 注：该参数为 layui 2.2.6 开始新增	string	images
exts	允许上传的文件后缀。一般结合 accept 参数类设定。假设 accept 为 file 类型时，那么你设置 exts: 'zip rar 7z' 即代表只允许上传压缩格式的文件。如果 accept 未设定，那么限制的就是图片的文件格式	string	jpg png gif bmp jpeg
auto	是否选完文件后自动上传。如果设定 false，那么需要设置 bindAction 参数来指向一个其它按钮提交上传	boolean	true
bindAction	指向一个按钮触发上传，一般配合 auto: false 来使用。值为选择器或 DOM对象，如：bindAction: '#btn'	string/object	-
field	设定文件域的字段名	string	file
size	设置文件最大可允许上传的大小，单位 KB。不支持ie8/9	number	0（即不限制）
multiple	是否允许多文件上传。设置 true即可开启。不支持ie8/9	boolean	false
number	设置同时可上传的文件数量，一般配合 multiple 参数出现。	number	0（即不限制）

number	注意：该参数为 <i>layui 2.2.3</i> 开始新增	number	0 (即不限制)
drag	是否接受拖拽的文件上传，设置 <i>false</i> 可禁用。不支持ie8/9	boolean	true
回调			
choose	选择文件后的回调函数。返回一个 <i>object</i> 参数，详见下文	function	-
before	文件提交上传前的回调。返回一个 <i>object</i> 参数（同上），详见下文	function	-
done	执行上传请求后的回调。返回三个参数，分别为： <i>res</i> （服务端响应信息）、 <i>index</i> （当前文件的索引）、 <i>upload</i> （重新上传的方法，一般在文件上传失败后使用）。详见下文	function	-
error	执行上传请求出现异常的回调（一般为网络异常、URL 404等）。返回两个参数，分别为： <i>index</i> （当前文件的索引）、 <i>upload</i> （重新上传的方法）。详见下文	function	-

上传接口

设定一个 URL 地址给 *url* 参数，用来告诉 *upload* 模块的服务端上传接口。像你平时使用Ajax一样。如：

```

1. upload.render({
2.   elem: '#id'
3.   ,url: '/api/upload/' //必填项
4.   ,method: '' //可选项。HTTP类型，默认post
5.   ,data: {} //可选项。额外的参数，如：{id: 123, abc: 'xxx'}
6. });
7.
```

该接口返回的相应信息（*response*）必须是一个标准的 JSON 格式，如：

```

1. {
2.   "code": 0
3.   , "msg": ""
4.   , "data": {
5.     "src": "http://cdn.layui.com/123.jpg"
6.   }
7. }
8.
```

注意1：你不一定非得按照上述格式返回，只要是合法的 JSON 字符即可。其响应信息会转化成JS对象传递给 *done* 回调。

注意2：如果上传后，出现文件下载框（一般为ie下），那么你需要在服务端对 *response* 的 *header* 设置 *Content-Type: text/html*

选择文件的回调

在文件被选择后触发，该回调会在 `before` 回调之前。一般用于非自动上传（即 `auto: false`）的场景，比如预览图片等。

```

1. upload.render({
2.   elem: '#id'
3.   ,url: '/api/upload/'
4.   ,auto: false //选择文件后不自动上传
5.   ,bindAction: '#testListAction' //指向一个按钮触发上传
6.   ,choose: function(obj){
7.     //将每次选择的文件追加到文件队列
8.     var files = obj.pushFile();
9.
10.    //预读本地文件，如果是多文件，则会遍历。(不支持ie8/9)
11.    obj.preview(function(index, file, result){
12.      console.log(index); //得到文件索引
13.      console.log(file); //得到文件对象
14.      console.log(result); //得到文件base64编码，比如图片
15.
16.      //obj.resetFile(index, file, '123.jpg'); //重命名文件名, layui 2.3.0 开始新
增
17.
18.      //这里还可以做一些 append 文件列表 DOM 的操作
19.
20.      //obj.upload(index, file); //对上传失败的单个文件重新上传，一般在某个事件中使用
21.      //delete files[index]; //删除列表中对应的文件，一般在某个事件中使用
22.    });
23.  }
24. });
25.

```

事实上这是一个非常实用的存在，可轻松应对复杂的列表文件上传管理。具体可移步到 [示例](#) 页面，里面有一个文件列表的小例子。

文件上传前的回调

在 `choose` 回调之后、`done/error` 回调之前触发。返回的参数完全类似 `choose` 回调。一般用于上传完毕前的loading、图片预览等。

```

1. upload.render({
2.   elem: '#id'
3.   ,url: '/api/upload/'
4.   ,before: function(obj){ //obj参数包含的信息，跟 choose回调完全一致，可参见上文。
5.     layer.load(); //上传loading
6.   }
7.   ,done: function(res, index, upload){
8.     layer.closeAll('loading'); //关闭loading
9.   }

```

```

10.     ,error: function(index, upload){
11.         layer.closeAll('loading'); //关闭loading
12.     }
13. });
14.

```

上传接口请求成功的回调

在上传接口请求完毕后触发，但文件不一定是上传成功的，只是接口的响应状态正常（200）。回调返回三个参数，分别为：服务端响应信息、当前文件的索引、重新上传的方法

```

1.  upload.render({
2.      elem: '#id'
3.      ,url: '/api/upload/'
4.      ,done: function(res, index, upload){
5.          //假设code=0代表上传成功
6.          if(res.code == 0){
7.              //do something （比如将res返回的图片链接保存到表单的隐藏域）
8.          }
9.
10.         //获取当前触发上传的元素，一般用于 elem 绑定 class 的情况，注意：此乃 layui 2.1.0 新
增
11.         var item = this.item;
12.
13.         //文件保存失败
14.         //do something
15.     }
16. });
17.

```

上传请求失败的回调

当请求上传时出现异常时触发（如网络异常、404/500等）。回调返回两个参数，分别为：当前文件的索引、重新上传的方法

```

1.  upload.render({
2.      elem: '#id'
3.      ,url: '/api/upload/'
4.      ,error: function(index, upload){
5.          //当上传失败时，你可以生成一个“重新上传”的按钮，点击该按钮时，执行 upload() 方法即可实现重新上传
6.      }
7.  });
8.

```

多文件上传完毕后的状态回调

只有当开启多文件时（即 `multiple: true`），该回调才会被触发。回调返回一个 `object` 类型的参数，包含一些状态数据：

```
1. upload.render({
2.   elem: '#id'
3.   ,url: '/api/upload/'
4.   ,multiple: true
5.   ,allDone: function(obj){ //当文件全部被提交后，才触发
6.     console.log(obj.total); //得到总文件数
7.     console.log(obj.successful); //请求成功的文件数
8.     console.log(obj.aborted); //请求失败的文件数
9.   }
10.  ,done: function(res, index, upload){ //每个文件提交一次触发一次。详见“请求成功的回调”
11.
12.   }
13. });
14.
```

文件上传进度的回调

在网速一般的情况下，大文件的上传通常需要一定时间的等待，而浏览器并不会醒目地告知你它正在努力地上传中，此时为了提升用户体验，我们可以通过该回调制作一个进度条。注：该回调为 layui 2.5.5 新增

```
1. upload.render({
2.   elem: '#id'
3.   ,url: '/api/upload/'
4.   ,progress: function(n, elem){
5.     var percent = n + '%' //获取进度百分比
6.     element.progress('demo', percent); //可配合 layui 进度条元素使用
7.
8.     //以下系 layui 2.5.6 新增
9.     console.log(elem); //得到当前触发的元素 DOM 对象。可通过该元素定义的属性值匹配到对应的进度条。
10.  }
11. });
12.
```

重载实例

有时你可能需要对 `upload.render()` 实例进行重载，通过改变一些参数（如将上传文件重置为只上传图片等场景）来重置功能。如：

```
1. //创建一个实例
2. var uploadInst = upload.render({
3.   elem: '#id'
```

```

4.     ,url: '/api/upload/'
5.     ,size: 1024*5 //限定大小
6. });
7.
8. //重载该实例，支持重载全部基础参数
9. uploadInst.reload({
10.   accept: 'images' //只允许上传图片
11.   ,acceptMime: 'image/*' //只筛选图片
12.   ,size: 1024*2 //限定大小
13. });
14.

```

注意：该方法为 layui 2.5.0 开始新增

重新上传

在执行 `upload.render(options)` 方法时，其实有返回一个实例对象，以便对完成重新上传等操作。注意：这是对当前上传队列的全局重新上传，而 `choose` 回调返回的 `obj.upload(index, file)` 方法则是对单个文件进行重新上传。如：

```

1. var uploadInst = upload.render({
2.   elem: '#id'
3.   ,url: '/api/upload/'
4.   ,choose: function(obj){
5.     obj.preview(function(index, file, result){
6.       //对上传失败的单个文件重新上传，一般在某个事件中使用
7.       //obj.upload(index, file);
8.     });
9.   }
10. });
11.
12. //重新上传的方法，一般在某个事件中使用
13. //uploadInst.upload();
14.

```

跨域上传

有些时候，可能会涉及到文件跨域操作，过去版本的 `upload` 模块最大的缺陷恰恰在于这里。而从 `layui 2.0` 的版本开始，我们已经对跨域做了支持。但鉴于代码的冗余度等多方面考虑，在IE9以下版本环境中，仍然不支持跨域。其它所有版本的IE和Chrome/FireFox等高级浏览器均支持。

那么，需要你怎么做？通常来说，是借助 `CORS`（跨资源共享）方案，即对接口所在的服务器设置：`Access-Control-Allow-Origin` 详见Google，配置起来还是挺简单的。而至于域名限制，一般是服务端程序中去处理。这里不做过多赘述。

穿梭框组件文档 - layui.transfer

穿梭框组件的初衷来源于 layui 社区的扩展组件平台，并且在 layui 2.5.0 的版本中开始登场。其适用的业务场景多样，不妨一试。

模块加载名称：*transfer*

快速使用

transfer 组件可以进行数据的交互筛选

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4.   <meta charset="utf-8">
5.   <title>穿梭框组件</title>
6.   <link rel="stylesheet" href="../src/css/layui.css">
7. </head>
8. <body>
9.   <div id="test1"></div>
10.  <script src="../src/layui.js"></script>
11.  <script>
12.    layui.use('transfer', function(){
13.      var transfer = layui.transfer;
14.
15.      //渲染
16.      transfer.render({
17.        elem: '#test1' //绑定元素
18.        ,data: [
19.          {"value": "1", "title": "李白", "disabled": "", "checked": ""}
20.          ,{"value": "2", "title": "杜甫", "disabled": "", "checked": ""}
21.          ,{"value": "3", "title": "贤心", "disabled": "", "checked": ""}
22.        ]
23.        ,id: 'demo1' //定义索引
24.      });
25.    });
26.  </script>
27. </body>
28. </html>
29.
```

基础参数

目前 transfer 组件提供以下基础参数，可根据需要进行相应的设置

参数选项	说明	类型	默认值

elem	指向容器选择器	String/Object	-
title	穿梭框上方标题	Array	['标题一', '标题二']
data	数据源	Array	[{}, {}, ...]
parseData	用于对数据源进行格式解析	Function	详见数据源格式解析
value	初始选中的数据（右侧列表）	Array	-
id	设定实例唯一索引，用于基础方法传参使用。	String	-
showSearch	是否开启搜索	Boolean	false
width	定义左右穿梭框宽度	Number	200
height	定义左右穿梭框高度	Number	340
text	自定义文本，如空数据时的异常提示等。 <pre>1. text: { 2. none: '无数据' //没有数据时的文案 3. ,searchNone: '无匹配数据' //搜索无匹配数据 4. 时的文案 4. }</pre>	Object	-
onchange	左右数据穿梭时的回调	Function	详见穿梭时的回调

数据源格式解析

数据格式解析的回调函数，用于将任意数据格式解析成 transfer 组件规定的
数据格式，以下是合法的数据格式如下：

```
1. [  
2.   {"value": "1", "title": "李白", "disabled": "", "checked": ""}  
3.   , {"value": "2", "title": "杜甫", "disabled": "", "checked": ""}  
4.   , {"value": "3", "title": "贤心", "disabled": "", "checked": ""}  
5. ]  
6.
```

然而很多时候你返回的数据格式可能并不符合规范，比如：

```
1. [  
2.   {"id": "1", "name": "李白"}  
3.   , {"id": "2", "name": "杜甫"}  
4.   , {"id": "3", "name": "贤心"}  
5. ]  
6.
```

那么您需要将其解析成 transfer 组件所规定的
数据格式：

```
1. transfer.render({  
2.   elem: '#text1'
```

```

3.     ,data: [
4.         {"id": "1", "name": "李白"}
5.         ,{"id": "2", "name": "杜甫"}
6.         ,{"id": "3", "name": "贤心"}
7.     ]
8.     ,parseData: function(res){
9.         return {
10.             "value": res.id //数据值
11.             , "title": res.name //数据标题
12.             , "disabled": res.disabled //是否禁用
13.             , "checked": res.checked //是否选中
14.         }
15.     }
16. });
17.

```

左右穿梭的回调

当数据在左右穿梭时触发，回调返回当前被穿梭的数据

```

1. transfer.render({
2.     elem: '#text'
3.     ,data: [] //数据源
4.     ,onChange: function(data, index){
5.         console.log(data); //得到当前被穿梭的数据
6.         console.log(index); //如果数据来自左边, index 为 0, 否则为 1
7.     }
8. });
9.

```

基础方法

基础用法是组件关键组成部分，目前所开放的所有方法如下：

```

1. var transfer = layui.transfer;
2.
3. transfer.set(options); //设定全局默认参数。options 即各项基础参数
4. transfer.getData(id); //获得右侧数据
5. transfer.reload(id, options); //重载实例
6.

```

获得右侧数据

穿梭框的右侧数据通常被认为是选中数据，因此你需要得到它并提交到后台。

```

1. transfer.render({

```

```
2.     elem: '#test'
3.     ,data: []
4.     ,id: 'demo1' //定义索引
5. });
6.
7. //获得右侧数据
8. var getData = transfer.getData('demo1');
9.
```

实例重载

重载一个已经创建的组件实例，被覆盖新的基础属性

```
1. transfer.render({
2.     elem: '#test'
3.     ,data: []
4.     ,id: 'demo1' //定义索引
5. });
6.
7. //可以重载所有基础参数
8. transfer.reload('demo1', {
9.     title: ['新列表1', '新列表2']
10. });
11.
```

结语

穿梭框组件极易上手，在浩瀚的业务需求中，值得一用。

layui - 用心与你沟通

树形组件文档 - layui.tree

在一段漫长的雪藏后，我们在 layui 2.5.0 的版本中重新创作了 tree，以便它能够更加适用于绝大多数业务场景，而风格依然遵循 layui 独有的极简和清爽。需要提醒的是，如果您的项目中仍然采用了 layui 2.5 版本之前的 tree，它将不被兼容，请尽快修改为以下新的调用方式。

模块加载名称：*tree*

快速使用

通过 `tree.render()` 方法指定一个元素，便可快速创建一个 tree 实例。

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4.   <meta charset="utf-8">
5.   <title>树组件</title>
6.   <link rel="stylesheet" href="../src/css/layui.css">
7. </head>
8. <body>
9.   <div id="test1"></div>
10.  <script src="../src/layui.js"></script>
11.  <script>
12.    layui.use('tree', function(){
13.      var tree = layui.tree;
14.
15.      //渲染
16.      var inst1 = tree.render({
17.        elem: '#test1' //绑定元素
18.        ,data: [{
19.          title: '江西' //一级菜单
20.          ,children: [{
21.            title: '南昌' //二级菜单
22.            ,children: [{
23.              title: '高新区' //三级菜单
24.              //..... //以此类推，可无限层级
25.            }]
26.          }]
27.        },{
28.          title: '陕西' //一级菜单
29.          ,children: [{
30.            title: '西安' //二级菜单
31.          }]
32.        }]
33.      });
34.    });
35.  </script>
```

```
36. </body>
37. </html>
38.
```

基础参数

目前 tree 组件提供以下基础参数，可根据需要进行相应的设置

参数选项	说明	类型	示例值
elem	指向容器选择器	String/Object	-
data	数据源	Array	详见数据选项
id	设定实例唯一索引，用于基础方法传参使用。	String	-
showCheckbox	是否显示复选框	Boolean	false
edit	是否开启节点的操作图标。默认 false。 <ul style="list-style-type: none">若为 true，则默认显示“改删”图标若为 数组，则可自由配置操作图标的显示状态和顺序，目前支持的操作图标有：add、update、del，如： edit: ['add', 'update', 'del']	Boolean/Array	['update', 'del']
accordion	是否开启手风琴模式，默认 false	Boolean	false
onlyIconControl	是否仅允许节点左侧图标控制展开收缩。默认 false（即点击节点本身也可控制）。若为 true，则只能通过节点左侧图标来展开收缩	Boolean	false
isJump	是否允许点击节点时弹出新窗口跳转。默认 false，若开启，需在节点数据中设定 link 参数（值为 url 格式）	Boolean	false
showLine	是否开启连接线。默认 true，若设为 false，则节点左侧出现三角图标。	Boolean	true
text	自定义各类默认文本，目前支持以下设定： 1. text: { 2. defaultNodeName: '未命名' //节点默认名称 3. , none: '无数据' //数据为空时的提示文本 4. }	Object	-

数据源属性选项

我们将 data 参数称之为数据源，其内部支持设定以下选项

属性选项	说明	类型	示例值
title	节点标题	String	未命名
id	节点唯一索引值，用于对指定节点进行各类操作	String/Number	任意唯一的字符或数字

field	节点字段名	String	一般对应表字段名
children	子节点。支持设定选项同父节点	Array	[{title: '子节点1', id: '111'}]
href	点击节点弹出新窗口对应的 url。需开启 isJump 参数	String	任意 URL
spread	节点是否初始展开，默认 false	Boolean	true
checked	节点是否初始为选中状态（如果开启复选框的话），默认 false	Boolean	true
disabled	节点是否为禁用状态。默认 false	Boolean	false

节点被点击的回调

在节点被点击后触发，返回的参数如下：

```
1. tree.render({
2.   elem: '#test1'
3.   ,click: function(obj){
4.     console.log(obj.data); //得到当前点击的节点数据
5.     console.log(obj.state); //得到当前节点的展开状态：open、close、normal
6.     console.log(obj.elem); //得到当前节点元素
7.
8.     console.log(obj.data.children); //当前节点下是否有子节点
9.   }
10. });
11.
```

复选框被点击的回调

点击复选框时触发，返回的参数如下：

```
1. tree.render({
2.   elem: '#test1'
3.   ,oncheck: function(obj){
4.     console.log(obj.data); //得到当前点击的节点数据
5.     console.log(obj.checked); //得到当前节点的展开状态：open、close、normal
6.     console.log(obj.elem); //得到当前节点元素
7.   }
8. });
9.
```

操作节点的回调

通过 operate 实现函数，对节点进行增删改等操作时，返回操作类型及操作节点

```

1. tree.render({
2.   elem: '#test1'
3.   ,operate: function(obj){
4.     var type = obj.type; //得到操作类型：add、edit、del
5.     var data = obj.data; //得到当前节点的数据
6.     var elem = obj.elem; //得到当前节点元素
7.
8.     //Ajax 操作
9.     var id = data.id; //得到节点索引
10.    if(type === 'add'){ //增加节点
11.      //返回 key 值
12.      return 123;
13.    } else if(type === 'update'){ //修改节点
14.      console.log(elem.find('.layui-tree-txt').html()); //得到修改后的内容
15.    } else if(type === 'del'){ //删除节点
16.
17.    };
18.  }
19. });
20.

```

返回选中的节点数据

很多时候 tree 可能不仅仅只是一个树菜单，它还用于各种权限控制等场景，这个时候你需要获得选中的节点。

语法：tree.getChecked(id)

```

1. tree.render({
2.   elem: '#test'
3.   ,data: [] //数据源
4.   ,id: 'demoId' //定义索引
5. });
6.
7. //获得选中的节点
8. var checkData = tree.getChecked('demoId');
9.

```

设置节点勾选

除了通过 checked 参数对节点进行初始勾选之外，你还可以通过方法动态对节点执行勾选

语法：tree.setChecked(id, checkedId)

参数 checkedId：代表的是数据源中的节点 id

```

1. tree.render({
2.   elem: '#test'
3.   ,data: [] //数据源

```



```
4.     ,id: 'demoId' //定义索引
5.   });
6.
7.   //执行节点勾选
8.   tree.setChecked('demoId', 1); //单个勾选 id 为 1 的节点
9.   tree.setChecked('demoId', [2, 3]); //批量勾选 id 为 2、3 的节点
10.
```

实例重载

重载一个已经创建的组件实例，被覆盖新的基础属性

```
1. tree.render({
2.   elem: '#test'
3.   ,data: []
4.   ,id: 'demoId' //定义索引
5. });
6.
7.   //可以重载所有基础参数
8. tree.reload('demoId', {
9.   //新的参数
10. });
11.
```

结语

树组件还在持续完善。

layui - 用心与你沟通

颜色选择器文档 - layui.colorpicker

在主题定制的应用场景中，自然离不开颜色的自定义。而你往往需要的是关于它的直观选择，于是 colorpicker 模块姗姗来迟，它支持 hex、rgb、rgba 三类色彩模式，在代码中简单的调用后，便可在你的网页系统中自由拖拽去选择你中意的颜色。

模块加载名称：*colorpicker*

注意：*colorpicker* 为 *layui 2.4.0* 新增模块，不支持 *ie10* 以下版本，其它高级浏览器均支持。

使用

colorpicker 是一款颜色选择器，如下是一个最基本的用法：

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4.   <meta charset="utf-8">
5.   <title>颜色选择器</title>
6.   <link rel="stylesheet" href="../src/css/layui.css">
7. </head>
8. <body>
9. <div id="test1"></div>
10. <script src="../src/layui.js"></script>
11. <script>
12.   layui.use('colorpicker', function(){
13.     var colorpicker = layui.colorpicker;
14.     //渲染
15.     colorpicker.render({
16.       elem: '#test1' //绑定元素
17.     });
18.   });
19. </script>
20. </body>
21. </html>
22.
```

基础参数

colorpicker 组件目前支持以下参数

参数选项	说明	类型	默认值
elem	指向容器选择器	string/object	-
color	默认颜色，不管你是使用 hex、rgb 还是 rgba 的格式输入，最终会以指定的格式显示。	string	-

format	颜色显示/输入格式，可选值： <i>hex</i> 、 <i>rgb</i> 若在 <i>rgb</i> 格式下开启了透明度，格式会自动变成 <i>rgba</i> 。在没有输入颜色的前提下，组件会默认为 <i>#000</i> 也就是黑色。	string	hex（即 16 进制色值）
alpha	是否开启透明度，若不开启，则不会显示透明框。开启了透明度选项时，当你的默认颜色为 <i>hex</i> 或 <i>rgb</i> 格式，组件会默认加上值为 1 的透明度。相同的，当你没有开启透明度，却以 <i>rgba</i> 格式设置默认颜色时，组件会默认没有透明度。 注意：该参数必须配合 <i>rgba</i> 颜色值使用	boolean	false
predefine	预定义颜色是否开启	boolean	false
colors	预定义颜色，此参数需配合 <i>predefine: true</i> 使用。	Array	此处列举一部分： ['#ff4500', '#1e90ff', 'rgba(255, 69, 0, 0.68)', 'rgb(255, 120, 0)']
size	下拉框大小，可以选择： <i>lg</i> 、 <i>sm</i> 、 <i>xs</i> 。	string	-

预定义颜色

预定义颜色，可以被认为是提供的参考色，因此除了我们默认的预定义颜色之外，你还可以自己定义

```

1. layui.use('colorpicker', function(){
2.   var colorpicker = layui.colorpicker;
3.
4.   colorpicker.render({
5.     elem: '#test1'
6.     ,predefine: true
7.     ,colors: ['#F00', '#0F0', '#00F', 'rgb(255, 69, 0)', 'rgba(255, 69, 0, 0.5)']
8.   });
9. });
10.
```

颜色被改变的回调

回调名：*change*

当颜色在选择器中发生选择改变时，会进入 *change* 回调，你可以通过它来进行所需操作，下面的例子就是实时的输出当前选择器的颜色

```

1. layui.use('colorpicker', function(){
2.   var colorpicker = layui.colorpicker;
```

```
3.
4.     colorpicker.render({
5.         elem: '#test1'
6.         ,change: function(color){
7.             console.log(color)
8.         }
9.     });
10. });
11.
```

颜色选择后的回调

回调名: *done*

点击颜色选择器的“确认”和“清除”按钮，均会触发 `done` 回调，回调返回当前选择的色值。

```
1. layui.use('colorpicker', function(){
2.     var colorpicker = layui.colorpicker;
3.
4.     colorpicker.render({
5.         elem: '#test1'
6.         ,done: function(color){
7.             console.log(color)
8.             //譬如你可以在回调中把得到的 color 赋值给表单
9.         }
10.     });
11. });
12.
```

结语

颜色选择器不仅仅是独立使用，它更多情况可能是跟表单结合使用。

layui - 用心与你沟通

常用元素操作 - layui.element

页面中有许多元素需要自动去完成一些处理，譬如导航菜单的小滑块、Tab的切换等操作，他们往往不需要去单独调用一个方法来开启一项功能，而页面上恰恰有太多这样的小交互，所以我们统一归类为element组件。跟表单一样，基于元素属性和事件驱动接口书写方式。

模块加载名称：*element*

使用

元素功能的开启只需要加载element模块即会自动完成，所以不用跟其它模块一样为某一个功能而调用一个方法。她只需要找到她支持的元素，如你的页面存在一个Tab元素块，那么element模块会自动赋予她该有的功能。

```
1. <div class="layui-tab" lay-filter="demo">
2.   <ul class="layui-tab-title">
3.     <li class="layui-this">网站设置</li>
4.     <li>商品管理</li>
5.     <li>订单管理</li>
6.   </ul>
7.   <div class="layui-tab-content">
8.     <div class="layui-tab-item layui-show">内容1</div>
9.     <div class="layui-tab-item">内容2</div>
10.    <div class="layui-tab-item">内容3</div>
11.  </div>
12. </div>
13.
```

前提是你加载element模块

```
1. layui.use('element', function(){
2.   var element = layui.element;
3.
4.   //一些事件监听
5.   element.on('tab(demo)', function(data){
6.     console.log(data);
7.   });
8. });
9.
```

预设元素属性

我们通过自定义元素属性来作为元素的功能参数，他们一般配置在容器外层，如：

```
1. <div class="layui-tab" lay-allowClose="true" lay-filter="demo">...</div>
2. <span class="layui-breadcrumb" lay-separator="|"></span>
3.
4.   And So On
5.
```

element 模块支持的元素如下表：

属性名	可选值	说明
lay-filter	任意字符	事件过滤器（公用属性），主要用于事件的精确匹配，跟选择器是比较类似的。
lay-allowClose	true	针对于Tab容器，是否允许选项卡关闭。默认不允许，即不用设置该属性
lay-separator	任意分隔符	针对于面包屑容器

基础方法

基础方法允许你在外部主动对元素发起一起操作，目前element模块提供的方法如下：

方法名	描述
<code>var element = layui.element;</code>	element模块的实例 返回的 <code>element</code> 变量为该实例的对象，携带一些用于元素操作的基础方法
<code>element.on(filter, callback);</code>	用于元素的一些事件监听
<code>element.tabAdd(filter, options);</code>	用于新增一个Tab选项 参数 <code>filter</code> ：tab元素的 <code>lay-filter="value"</code> 过滤器的值（value） 参数 <code>options</code> ：设定可选值的对象，目前支持的选项如下述示例： <pre>1. element.tabAdd('demo', { 2. title: '选项卡的标题' 3. ,content: '选项卡的内容' //支持传入html 4. ,id: '选项卡标题的lay-id属性值' 5. });</pre>
<code>element.tabDelete(filter, layid);</code>	用于删除指定的Tab选项 参数 <code>filter</code> ：tab元素的 <code>lay-filter="value"</code> 过滤器的值（value） 参数 <code>layid</code> ：选项卡标题列表的 属性 <code>lay-id</code> 的值 <pre>1. element.tabDelete('demo', 'xxx'); //删除 lay-id="xxx" 的这一项</pre>
<code>element.tabChange(filter, layid);</code>	用于外部切换到指定的Tab项上，参数同上，如： <code>element.tabChange('demo', 'layid');</code> //切换到 <code>lay-id="yyy"</code> 的这一项
	用于绑定自定义 Tab 元素（即非 layui 自带的 tab 结构）。该方法为 layui 2.1.6 新增

<code>element.tab(options);</code>	<p>参数<code>options</code>：设定可选值的对象，目前支持的选项如下述示例：</p> <pre>1. //HTML 2. <ul id="tabHeader"> 3. 标题1 4. 标题2 5. 标题3 6. 7. <div id="tabBody"> 8. <div class="xxx">内容1</div> 9. <div class="xxx">内容2</div> 10. <div class="xxx">内容4</div> 11. </div> 12. 13. //JavaScript 14. element.tab({ 15. headerElem: '#tabHeader>li' //指定tab头元素项 16. ,bodyElem: '#tabBody>.xxx' //指定tab主体元素项 17. });</pre>
<code>element.progress(filter, percent);</code>	<p>用于动态改变进度条百分比： <code>element.progress('demo', '30%');</code></p>

更新渲染

跟表单元素一样，很多时候你的页面元素可能是动态生成的，这时element的相关功能将不会对其有效，你必须手工执行 `element.init(type, filter)` 方法即可。注意：2.1.6 开始，可以用 `element.render(type, filter);` 方法替代

第一个参数：type，为表单的type类型，可选。默认对全部类型的表单进行一次更新。可局部刷新的type如下表：

参数（type）值	描述
tab	重新对tab选项卡进行初始化渲染
nav	重新对导航进行渲染
breadcrumb	重新对面包屑进行渲染
progress	重新对进度条进行渲染
collapse	重新对折叠面板进行渲染

```
1. element.init(); //更新全部 2.1.6 可用 element.render() 方法替代
2. element.render('nav'); //重新对导航进行渲染。注：layui 2.1.6 版本新增
3. //.....
4.
```

第二个参数：filter，为元素的 `lay-filter=""` 的值。你可以借助该参数，完成指定元素的局部更新。

```

1.  【HTML】
2.  <div class="layui-nav" lay-filter="test1">
3.    ...
4.  </div>
5.
6.  <div class="layui-nav" lay-filter="test2">
7.    ...
8.  </div>
9.
10. 【JavaScript】
11. //比如当你对导航动态插入了二级菜单，这时你需要重新去对它进行渲染
12. element.render('nav', 'test1'); //对 lay-filter="test1" 所在导航重新渲染。注：layui
    2.1.6 版本新增
13. //.....
14.

```

事件监听

语法：`element.on('event(过滤器值)', callback);`

`element` 模块在 `layui` 事件机制中注册了 `element` 模块事件，所以当你使用 `layui.onevent()` 自定义模块事件时，请勿占用 `element` 名。目前 `element` 模块所支持的事件如下表：

event	描述
tab	监听 Tab 选项卡切换事件
tabDelete	监听 Tab 监听选项卡删除事件
nav	监听导航菜单的点击事件
collapse	监听折叠面板展开或收缩事件

默认情况下，事件所监听的是全部的元素，但如果你只想监听某一个元素，使用事件过滤器即可。

如：

```

1. element.on('tab(test)', function(data){
2.   console.log(data);
3. });
4.

```

监听选项卡切换

Tab选项卡点击切换时触发，回调函数返回一个object参数，携带两个成员：

```

1. element.on('tab(filter)', function(data){
2.   console.log(this); //当前Tab标题所在的原始DOM元素

```



```

3.     console.log(data.index); //得到当前Tab的所在下标
4.     console.log(data.elem); //得到当前的Tab大容器
5. });
6.

```

监听选项卡删除

Tab选项卡被删除时触发，回调函数返回一个object参数，携带两个成员：

```

1. element.on('tabDelete(filter)', function(data){
2.     console.log(this); //当前Tab标题所在的原始DOM元素
3.     console.log(data.index); //得到当前Tab的所在下标
4.     console.log(data.elem); //得到当前的Tab大容器
5. });
6.

```

注：该事件为 *layui 2.1.6* 新增

监听导航菜单的点击

当点击导航父级菜单和二级菜单时触发，回调函数返回所点击的菜单DOM对象：

```

1. element.on('nav(filter)', function(elem){
2.     console.log(elem); //得到当前点击的DOM对象
3. });
4.

```

监听折叠面板

当折叠面板点击展开或收缩时触发，回调函数返回一个object参数，携带三个成员：

```

1. element.on('collapse(filter)', function(data){
2.     console.log(data.show); //得到当前面板的展开状态，true或者false
3.     console.log(data.title); //得到当前点击面板的标题区域DOM对象
4.     console.log(data.content); //得到当前点击面板的内容区域DOM对象
5. });
6.

```

动态操作进度条

你肯定不仅仅是满足于进度条的初始化显示，通常情况下你需要动态改变它的进度值，element模块提供了这样的基础方法：*element.progress(filter,*

percent);。

1. `<div class="layui-progress layui-progress-big" lay-filter="demo" lay-showPercent="true">`
2. `<div class="layui-progress-bar" lay-percent="0%"></div>`
3. `</div>`
- 4.
5. 上述是一个已经设置了过滤器（`lay-filter="demo"`）的进度条
6. 现在你只需要在某个事件或者语句中执行方法：`element.progress('demo', '50%');`
7. 即可改变进度
- 8.

如果你需要进度条更直观的例子，建议浏览：[进度条演示页面](#)

结语

事实上元素模块的大部分操作都是内部自动完成的，所以目前你发现他的接口很少呢。当然，我们也会不断增加element模块所支持的页面元素。

layui - 用心与你沟通

滑块文档 - layui.slider

作为一个拖拽式的交互性组件，滑块往往能给产品带来更好的操作体验。layui 深以为然，slider 模块包含了你能想到的大部分功能，尽管它可以作为一个独立的个体，但很多时候它往往会出现 form 元素中，想象一下吧。

模块加载名称：*slider*

注意：*slider* 为 *layui 2.4.0* 新增模块

使用

通过对 slider 模块的使用，你可以在页面构建出可拖动的滑动元素，如下是一个最基本的用法：

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4.   <meta charset="utf-8">
5.   <title>滑块</title>
6.   <link rel="stylesheet" href="../../src/css/layui.css">
7. </head>
8. <body>
9. <div id="slideTest1"></div>
10. <script src="../../src/layui.js"></script>
11. <script>
12.   layui.use('slider', function(){
13.     var slider = layui.slider;
14.
15.     //渲染
16.     slider.render({
17.       elem: '#slideTest1' //绑定元素
18.     });
19.   });
20. </script>
21. </body>
22. </html>
23.
```

基础参数

slider 组件支持以下参数

参数选项	说明	类型	默认值
elem	指向容器选择器	string/object	-
type	滑块类型，可选值有： <i>default</i> （水平滑块）、 <i>vertical</i> （垂直滑块）	string	default

min	滑动条最小值，正整数，默认为 0	number	0
max	滑动条最大值	number	100
range	是否开启滑块的范围拖拽，若设为 true，则滑块将出现两个可拖拽的环	boolean	false
value	滑块初始值，默认为数字，若开启了滑块为范围拖拽（即 range: true），则需赋值数组，异表示开始和结尾的区间，如：value: [30, 60]	number/Array	0
step	拖动的步长	number	1
showstep	是否显示间断点	boolean	false
tips	是否显示文字提示	boolean	true
input	是否显示输入框（注意：若 range 参数为 true 则强制无效） 点击输入框的上下按钮，以及输入任意数字后回车或失去焦点，均可动态改变滑块	boolean	false
height	滑动条高度，需配合 type: "vertical" 参数使用	number	200
disabled	是否将滑块禁用拖拽	boolean	false
theme	主题颜色，以使用在不同的主题风格下	string	#009688

自定义提示文本

当鼠标放在圆点和滑块拖拽时均会触发提示层。其默认显示的文本是它的对应数值，你也可以自定义提示内容：

```

1. slider.render({
2.   elem: '#slideTest1'
3.   ,setTips: function(value){ //自定义提示文本
4.     return value + '%';
5.   }
6. });
7.
```

数值改变的回调

在滑块数值被改变时触发。该回调非常重要，可动态获得滑块当前的数值。你可以将得到的数值，赋值给隐藏域，或者进行一些其它操作。

```

1. //当滑块为普通模式，回调返回的 value 是一个数值
2. slider.render({
3.   elem: '#slideTest1'
4.   ,change: function(value){
5.     console.log(value) //动态获取滑块数值
6.     //do something
7.   }
8. });
9.
```

```

10. //当滑块为范围模式，回调返回的 value 是一个数组，包含开始和结尾
11. slider.render({
12.   elem: '#slideTest1'
13.   ,range: true
14.   ,change: function(value){
15.     console.log(value[0]) //得到开始值
16.     console.log(value[1]) //得到结尾值
17.     //do something
18.   }
19. });
20.

```

实例方法

执行 slider 实例时，会返回一个当前实例的对象，里面包含针对当前实例的方法和属性。

语法：`var ins1 = slider.render(options);`

```

1. var ins1 = slider.render(options); //获得实例对象
2.
3. ins1.config //获得当前实例的配置项
4. ins1.setValue(nums); //动态给滑块赋值
5.

```

动态改变滑块数值

你可以通过外部方法动态改变滑块数值，如：

```

1. var ins1 = slider.render({
2.   elem: '#test1'
3.   //...
4. });
5.
6. //改变指定滑块实例的数值
7. ins1.setValue(20)
8.
9. //若滑块开启了范围（range: true）
10. ins1.setValue(20, 0) //设置开始值
11. ins1.setValue(60, 1) //设置结尾值
12.

```

结语

layui.slider 可以大幅度提升你 web 应用中的很多操作体验，可尽情发挥。

评分组件文档 - layui.rate

rate 评分组件在电商和 O2O 平台尤为常见，一般用于对商家进行服务满意度评价。rate 组件是 layui 团队新成员 @star1029 的第一款组件，外形依然小巧自然，功能依旧灵活实用。其中评分对应的自定义内容功能，可让它有更多的发挥空间。该组件为 2.3.0 版本新增

模块加载名称：*rate*

使用

rate 组件可以用来进行展示或评价，你只需要通过更改参数设定来开启你想要的功能，如下是一个最基本的例子：

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4.   <meta charset="utf-8">
5.   <title>评分组件</title>
6.   <link rel="stylesheet" href="../src/css/layui.css">
7. </head>
8. <body>
9.   <div id="test1"></div>
10.  <script src="../src/layui.js"></script>
11.  <script>
12.    layui.use('rate', function(){
13.      var rate = layui.rate;
14.
15.      //渲染
16.      var ins1 = rate.render({
17.        elem: '#test1' //绑定元素
18.      });
19.    });
20.  </script>
21. </body>
22. </html>
23.
```

这真的就是个小例子，所以下文对组件的参数进行了说明，请仔细阅读奥

基础参数

目前 rate 组件提供了以下基础参数，你可根据实际场景进行相应的设置

参数选项	说明	类型	默认值
elem	指向容器选择器	string/object	-
length	评分组件中具体星星的个数。个数当然是整数啦，残缺的星星	number	5

length	很可怜的，所以设置了小数点的组件我们会默认向下取整	number	5
value	评分的初始值	number	0
theme	主题颜色。我们默认的组件颜色是#FFB800，你可以根据自身喜好来更改组件的颜色，以适用不同场景	string	#FFB800
half	设定组件是否可以选半星	boolean	false
text	是否显示评分对应的内容	boolean	false
readonly	是否只读，即只用于展示而不可点	boolean	false

分数设置

如若你设置分数，我们会根据你是否开启半星功能，来做一个具体的规范：

关闭半星功能：

- 小数值大于 0.5 ：分数向上取整，如 3.6 分，则系统自动更改为 4 分
- 小数值小于等于 0.5 ：分数向下取整，如 3.2 分，则系统自动更改为 3 分
- 如果在关闭半星功能的情况下开启了文本，你会发现你的分数也相应的变成了整数

开启半星功能：

- 不论你的小数值是 0.1 还是 0.9，都统一规划为 0.5，在文本开启的情况下，你可以看见你的分数并没有发生变化

自定义文本的回调

通过 `setText` 函数，在组件初次渲染和点击后时产生回调。我们默认文本以星级显示，你可以根据自己设定的文字来替换我们的默认文本，如“讨厌”“喜欢”。若用户选择分数而没有设定对应文字的情况下，系统会使用我们的默认文本

```

1.
2. rate.render({
3.   elem: '#test1'
4.   ,setText: function(value){
5.     var arrs = {
6.       '1': '极差'
7.       , '2': '差'
8.       , '3': '中等'
9.       , '4': '好'
10.    };
11.    this.span.text(arrs[value] || ( value + "星"));
12.  }
13. });
14.
```

当你点击 3 星时，文本内容是中等，点击 5 星时，由于没有设定对应文字，所以文本会显示 5 星

点击产生的回调

通过 choose 实现函数，在组件被点击后触发，回调分数，用户可根据分数来设置效果，比如出现弹出层

```
1. rate.render({
2.   elem: '#test1'
3.   ,choose: function(value){
4.     if(value > 4) alert( '么么哒' )
5.   }
6. });
7.
```

那么当你点击 5 星或更高星级时，页面就会弹出“么么哒”啦，你可根据相应需求在 choose 里完善你的代码

结语

评分组件非常简单，重点在于参数选项的设置，你可以前往示例页面进行更为直观的了解。

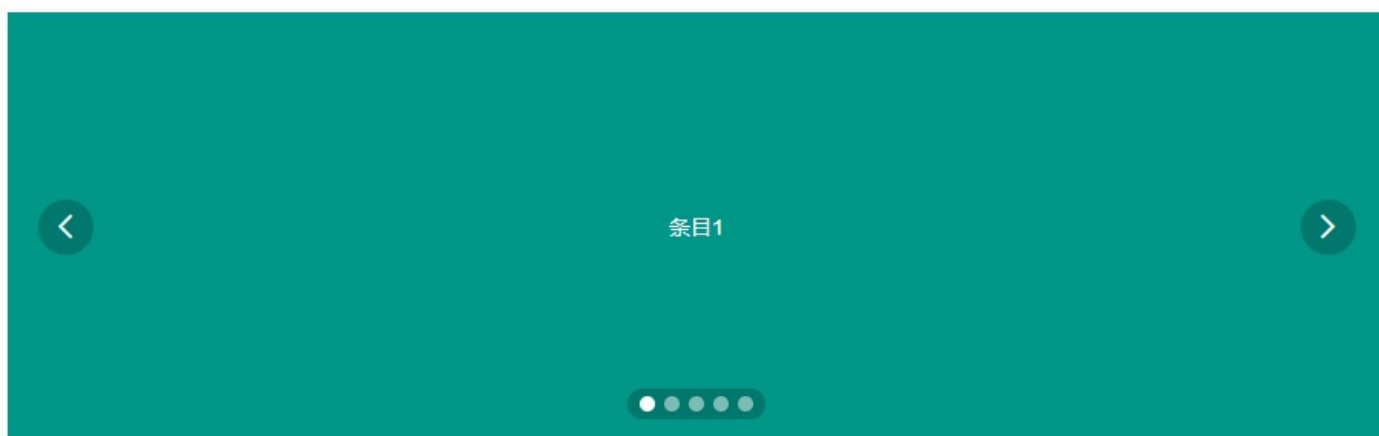
通用轮播组件文档 - layui.carousel

carousel 是 layui 2.0 版本中新增的全新模块，主要适用于跑马灯/轮播等交互场景。它并非单纯地为焦点图而生，准确地说，它可以满足任何类型内容的轮播式切换操作，更可以胜任 FullPage（全屏上下轮播）的需求，简洁而不失强劲，灵活而优雅。

模块加载名称：*carousel*

快速使用

如下是几个常用的轮播示例，其中背景色是为了区分条目单独加的，在layui框架中并不会包含。条目区域可以放上文字列表、图片等任意内容



```
1. <!DOCTYPE html>
2. <html>
3. <head>
4.   <meta charset="utf-8">
5.   <title>carousel模块快速使用</title>
6.   <link rel="stylesheet" href="/static/build/layui.css" media="all">
7. </head>
8. <body>
9.
10. <div class="layui-carousel" id="test1">
11.   <div carousel-item>
12.     <div>条目1</div>
13.     <div>条目2</div>
14.     <div>条目3</div>
15.     <div>条目4</div>
16.     <div>条目5</div>
17.   </div>
18. </div>
19. <!-- 条目中可以是任意内容，如：<img src=""> -->
20.
21. <script src="/static/build/layui.js"></script>
22. <script>
23.   layui.use('carousel', function(){
```

```

24.   var carousel = layui.carousel;
25.   //建造实例
26.   carousel.render({
27.     elem: '#test1'
28.     ,width: '100%' //设置容器宽度
29.     ,arrow: 'always' //始终显示箭头
30.     //,anim: 'updown' //切换动画方式
31.   });
32. });
33. </script>
34. </body>
35. </html>

```

在HTML结构中，只需要简单地注意这两项：

- 1) 外层元素的 `class="layui-carousel"` 用来标识为一个轮播容器
- 2) 内层元素的属性 `carousel-item` 用来标识条目

而 `id` 则用于carousel模块建造实例的元素指向，剩下的工作，就是按照你的实际需求，给方法设置不同的基础参数了。

基础参数选项

通过核心方法：`carousel.render(options)` 来对轮播设置基础参数，也可以通过方法：`carousel.set(options)` 来设定全局基础参数。

可选项	说明	类型	默认值
elem	指向容器选择器，如：elem: '#id'。也可以是DOM对象	string/object	无
width	设定轮播容器宽度，支持像素和百分比	string	'600px'
height	设定轮播容器高度，支持像素和百分比	string	'280px'
full	是否全屏轮播	boolean	false
anim	轮播切换动画方式，可选值为： <ul style="list-style-type: none"> • default（左右切换） • updown（上下切换） • fade（渐隐渐显切换） 	string	'default'
autoplay	是否自动切换	boolean	true
interval	自动切换的时间间隔，单位：ms（毫秒），不能低于800	number	3000
index	初始开始的条目索引	number	0
arrow	切换箭头默认显示状态，可选值为： <ul style="list-style-type: none"> • hover（悬停显示） • always（始终显示） • none（始终不显示） 	string	'hover'
indicator	指示器位置，可选值为： <ul style="list-style-type: none"> • inside（容器内部） • outside（容器外部） • none（不显示） 	string	'inside'

	注意：如果设定了 <code>anim: 'updown'</code> ，该参数将无效		
trigger	指示器的触发事件	string	'click'

切换事件

轮播的每一次切换时触发，回调函数返回一个object参数，携带的成员如下：

```
1. var carousel = layui.carousel;
2.
3. //监听轮播切换事件
4. carousel.on('change(test1)', function(obj){ //test1来源于对应HTML容器的 lay-
   filter="test1" 属性值
5.   console.log(obj.index); //当前条目的索引
6.   console.log(obj.prevIndex); //上一个条目的索引
7.   console.log(obj.item); //当前条目的元素对象
8. });
9.
```

重置轮播

事实上，在执行 `carousel.render(options)` 方法时，有返回一个当前实例的对象。该对象包含了用于操作当前轮播的一些属性和方法。

```
1. var ins = carousel.render(options);
2.
3. //重置轮播
4. ins.reload(options);
5.
```

结语

由于轮播的使用非常简单，所以本篇不做过于详细的讲解，核心在于基础参数选项的设置。你也可以前往示例页面进行更为直观的了解。

layui - 用心与你沟通

流加载文档 - layui.flow

该模块包含信息流加载和图片懒加载两大核心支持，无论是对服务端、还是前端体验，都有非常大的性能帮助。你可能已经在太多的地方看到她们的身影了，但不妨现在开始，体验一下Layui更为简单和高效的Flow吧。

模块加载名称：*flow*

使用

flow模块包含两个核心方法，如下所示：

```
1. layui.use('flow', function(){
2.   var flow = layui.flow;
3.   //信息流
4.   flow.load(options);
5.
6.   //图片懒加载
7.   flow.lazyimg(options);
8. });
9.
```

下面将对她们进行详细介绍。

信息流

信息流即异步逐页渲染列表元素，这是你页面已经存在的一段列表，你页面初始时只显示了6个

```
1.
2.   <li>1</li>
3.   <li>2</li>
4.   .....
5.   <li>6</li>
6.
7.
```

你想通过加载更多来显示余下列表，那么你只需要执行方法：
flow.load(options) 即可

```
1. layui.use('flow', function(){
2.   var $ = layui.jquery; //不用额外加载jQuery，flow模块本身是有依赖jQuery的，直接用即可。
3.   var flow = layui.flow;
```

```
4.   flow.load({
5.     elem: '#demo' //指定列表容器
6.   ,done: function(page, next){ //到达临界点（默认滚动触发），触发下一页
7.     var lis = [];
8.     //以jQuery的Ajax请求为例，请求下一页数据（注意：page是从2开始返回）
9.     $.get('/api/list?page='+page, function(res){
10.      //假设你的列表返回在data集合中
11.      layui.each(res.data, function(index, item){
12.        lis.push('
13. '+ item.title + '
14. ');
14.      });
15.
16.      //执行下一页渲染，第二参数为：满足“加载更多”的条件，即后面仍有分页
17.      //pages为Ajax返回的总页数，只有当前页小于总页数的情况下，才会继续出现加载更多
18.      next(lis.join(''), page < res.pages);
19.    });
20.  }
21. });
22. });
23.
```

上述是一个比较简单的例子，以下是信息流完整的参数支撑（即options对象），它们将有助于你更灵活地应对各种场景

参数	类型	描述
elem	string	指定列表容器的选择器
scrollElem	string	滚动条所在元素选择器，默认document。如果你不是通过窗口滚动来触发流加载，而是页面中的某一个容器的滚动条，那么通过该参数指定即可。
isAuto	boolean	是否自动加载。默认true。如果设为false，点会在列表底部生成一个“加载更多”的button，则只能点击它才会加载下一页数据。
end	string	用于显示末页内容，可传入任意HTML字符。默认为：没有更多了
isLazyimg	boolean	是否开启图片懒加载。默认false。如果设为true，则只会对在可视区域的图片进行按需加载。但与此同时，在拼接列表字符的时候，你不能给列表中的img元素赋值src，必须要用lay-src取代，如： <div>1. layui.each(res.data, function(index, item){ 2. lis.push(''); 3. });</div>
mb	number	与底部的临界距离，默认50。即当滚动条与底部产生该距离时，触发加载。注意：只有在isAuto为true时有效。额，等等。。mb=margin-bottom，可不是骂人的呀。
done	function	到达临界点触发加载的回调。信息流最重要的一个存在。携带两个参数： <div>1. done: function(page, next){ 2. //请注意：layui 1.0.5 之前的版本是从第2页开始返回，也就是说你的第一页数据并非done来触发加载 3. （为之前这个愚蠢的设计表示抱歉） 4. //从 layui 1.0.5 的版本开始，page是从1开始返回，初始时即会执行一次done回调。 5. //console.log(page) //获得当前页 6.</div>

```

7.    //执行下一页渲染，第二参数为：满足“加载更多”的条件，即后面仍有分页
8.    //只有当前页小于总页数的情况下，才会继续出现加载更多
9.    next( '列表HTML片段', page < res.pages);
10. }
11.

```

图片懒加载

应该说比当前市面上任何一个懒加载的实现都更为强劲和轻量，她用不足80行代码巧妙地提供了一个始终加载当前屏图片的高性能方案（无论上滑还是下滑）。对你的网站因为图片可能带来的压力，可做出很好的应对。

语法：flow.lazyimg(options)

```

1. layui.use('flow', function(){
2.     var flow = layui.flow;
3.     //当你执行这样一个方法时，即对页面中的全部带有lay-src的img元素开启了懒加载（当然你也可以指定相关img）
4.     flow.lazyimg();
5. });
6.

```

如上所述，它只会针对以下img元素有效：

```

1. 
2. 
3. 
4.

```

图片懒加载的使用极其简单，其参数（options对象）可支持的key如下表所示：

参数	类型	描述
elem	string	指定开启懒加载的img元素选择器，如 elem: '.demo img' 或 elem: 'img.load'
scrollElem	string	滚动条所在元素选择器，默认document。如果你不是通过窗口滚动来触发流加载，而是页面中的某一个容器的滚动条，那么通过该参数指定即可。

结语

如果还有各种流行的流加载，你可以给我们提供建议，我们会在layui后续版本中选择性加入。

工具集文档 - layui.util

我们将一些工具性元素放入 *util* 模块中，以供选择性使用。其内部由多个小工具组件组成，他们也许不是必须的，但很多时候却能为你的页面提供良好的辅助作用。

模块加载名称：*util*

固定块

看到页面右下角的那个包含top的bar了吗？对，就是她。她通常会出现在那个固定位置，由两个可选的bar和一个默认必选的TopBar组成。

语法：*util.fixbar(options)*
其中参数 *options* 是一个对象，可支持的key如下表：

参数	类型	描述
bar1	Boolean/String	默认false。如果值为true，则显示第一个bar，带有一个默认图标。如果值为图标字符，则显示第一个bar，并覆盖默认图标
bar2	Boolean/String	默认false。如果值为true，则显示第二个bar，带有一个默认图标。如果值为图标字符，则显示第二个bar，并覆盖默认图标
bgcolor	String	自定义区块背景色
showHeight	Number	用于控制出现TOP按钮的滚动条高度临界值。默认：200
css	Object	你可以通过重置bar的位置，比如 <code>css: {right: 100, bottom: 100}</code>
click	Function	点击bar的回调，函数返回一个type参数，用于区分bar类型。支持的类型有：bar1、bar2、top

```
1. layui.use('util', function(){
2.   var util = layui.util;
3.
4.   //执行
5.   util.fixbar({
6.     bar1: true
7.     ,click: function(type){
8.       console.log(type);
9.       if(type === 'bar1'){
10.        alert('点击了bar1')
11.      }
12.    }
13.  });
14. });
15.
```

倒计时

这是一个精致的封装，它并不负责 UI 元素的呈现，而仅仅只是返回倒计时的数据，这意味着你可以将它应用在任何倒计时相关的业务中。

语法：`util.countdown(endTime, serverTime, callback)`

参数	说明
endTime	结束时间戳或Date对象，如：4073558400000，或：new Date(2099,1,1)。
serverTime	当前服务器时间戳或Date对象
callback	回调函数。如果倒计时尚在运行，则每一秒都会执行一次。并且返回三个参数： <code>date</code> （包含天/时/分/秒的对象）、 <code>serverTime</code> （当前服务器时间戳或Date对象）， <code>timer</code> （计时器返回的ID值，用于clearTimeout）

```
1. <div id="test"></div>
2.
3. <script>
4. layui.use('util', function(){
5.   var util = layui.util;
6.
7.   //示例
8.   var endTime = new Date(2099,1,1).getTime() //假设为结束日期
9.   ,serverTime = new Date().getTime(); //假设为当前服务器时间，这里采用的是本地时间，实际使用一般是取服务端的
10.
11.   util.countdown(endTime, serverTime, function(date, serverTime, timer){
12.     var str = date[0] + '天' + date[1] + '时' + date[2] + '分' + date[3] +
13.     '秒';
14.     layui.$('#test').html('距离2099年1月1日还有：'+ str);
15.   });
16. </script>
17.
```

其它方法

方法	说明
util.timeAgo(time, onlyDate)	某个时间在当前时间的多久前。 参数 <code>time</code> ：即为某个时间的时间戳或日期对象 参数 <code>onlyDate</code> ：是否在超过30天后，只返回日期字符，而不返回时分秒 如果在3分钟以内，返回：刚刚 如果在30天以内，返回：若干分钟前、若干小时前、若干天前，如：5分钟前 如果在30天以上，返回：日期字符，如：2017-01-01
util.toDateString(time, format)	转化时间戳或日期对象为日期格式字符 参数 <code>time</code> ：可以是日期对象，也可以是毫秒数 参数 <code>format</code> ：日期字符格式（默认：yyyy-MM-dd HH:mm:ss），可随意定义，如：yyyy年MM月dd日
util.digit(num, length)	数字前置补零 参数 <code>num</code> ：原始数字 参数 <code>length</code> ：数字长度，如果原始数字长度小于 <code>length</code> ，则前面补零，如：util.digit(7, 3) //007

util.escape(str)	转义 xss 字符 参数 str: 任意字符
util.event(attr, obj, eventType)	<p>用于更好地批量处理事件。 参数 attr: 绑定需要监听事件的元素属性 参数 obj: 事件回调链 参数 eventType: 事件类型 (默认 click)</p> <p>示例:</p> <pre> 1. HTML : 2. <button class="layui-btn" lay-active="e1">事件1</button> 3. <button class="layui-btn" lay-active="e2">事件2</button> 4. <button class="layui-btn" lay-active="e3">事件3</button> 5. 6. JavaScript : 7. <script> 8. layui.use('util', function(){ 9. var util = layui.util; 10. 11. //处理属性 为 lay-active 的所有元素事件 12. util.event('lay-active', { 13. e1: function(){ 14. alert('触发了事件1'); 15. } 16. ,e2: function(){ 17. alert('触发了事件2'); 18. } 19. ,e3: function(){ 20. alert('触发了事件3'); 21. } 22. }); 23. }); 24. </script> 25. </pre>

结语

工具模块往往应用在边边角角，将不定期增加。

layui - 用心与你沟通

代码修饰器文档 - layui.code

code模块通常针对于程序员，它是layui中一个极其轻量的组成。通俗而言，该模块就是对你的pre元素进行一个修饰，从而保证你展现的代码更具可读性。目前它没有对不同的语言进行颜色高亮（因为目前感觉没有太大必要，后面layui全面稳定后，可能会完善该功能），但这丝毫不会影响它对你带来的便捷。

模块加载名称：*code*

使用

code模块的使用非常简单，请直接看代码，假设你在页面有这样一段pre标签：

```
1. <pre class="layui-code">
2. //代码区域
3. var a = 'hello layui';
4. </pre>
5.
```

那么你只需要经过下面的方式：

```
1. layui.use('code', function(){ //加载code模块
2.   layui.code(); //引用code方法
3. });
4.
```

就可以将那段pre区块显示成你现在看到的这个样子：

```
1. //代码区域
2. var a = 'hello layui';
3.
```

基础参数

方法：layui.code(options)

它接受一个对象参数options，支持以下key的设定

参数	类型	描述
elem	string	指定元素的选择器
title	string	设定标题
height	string	设置最大高度
encode	boolean	是否转义html标签，默认false
skin	string	风格选择（值见下文）

about	boolean	是否剔除右上关于
-------	---------	----------

特别提示：除了上述方式的设置，我们还允许你直接在`pre`标签上设置属性来替代，如：

```
1. <pre class="layui-code" lay-title="" lay-height="" lay-skin="" lay-encode="">
2. 这样有木有觉得更方便些
3. </pre>
4.
```

下面将针对每一个参数做进一步讲解。

指定元素

code模块会去自动查找class为layui-code的类，如果你初始给的不是该类，仅仅只是一个pre标签，那么需要通过elem设置选择器来指向元素：

```
1. layui.code({
2.   elem: 'pre' //默认值为.layui-code
3. });
4.
```

设置标题

即左上角显示的文本，默认值为code

```
1. layui.code({
2.   title: 'JavaScript'
3. });
4.
```

或者直接在pre标签上设置属性 `<pre lay-title="JavaScript"></pre>`

设置最大高度

你可以设置以下key来控制修饰器的最大高度。如果内容低于该高度，则会自适应内容高度；如果内容超过了该高度，则会自动出现滚动条。

```
1. layui.code({
2.   height: '100px' //请注意必须加px。如果该key不设定，则会自适应高度，且不会出现滚动条。
3. });
4.
5.
6.
7.
8. Hi, 我是充数的 ^_^
9.
10.
```

或者直接在pre标签上设置属性 `<pre lay-height="100px"></pre>`

转义html标签

事实上很多时候你都需要在pre标签中展现html标签，而不希望它被浏览器解析。那么code模块允许你这么，只需要开启encode即可，如：

```
1. layui.code({
2.   encode: true //是否转义html标签。默认不开启
3. });
4.
```

开启了encode后的效果如下：

```
1.
2.
3.   1. HTML将不会被解析
4.
5.   1. 有木有感觉非常方便
6.
```

或者直接在pre标签上设置属性 `<pre lay-encode="true"></pre>`

风格选择

你肯定不会满足于code的某一种显示风格，而skin参数则允许你设定许多种显示风格，我们目前内置了两种，分别为默认和notepad

```
1. layui.code({
2.   title: 'NotePad++的风格'
3.   , skin: 'notepad' //如果要默认风格，不用设定该key。
4. });
5.
```

上述的设定后，你会看到下面的样子

```
1. i'm code.
2. i'm code too.
3.
```

或者直接在pre标签上设置属性 `<pre lay-skin="notepad"></pre>`

剔除关于

如果你不喜欢出现右上角的layui.code字眼，你是可以剔除的。设置about:false即可，请叫我雷锋。

layui - 用心与你沟通