

# 1. 文章类别

## 1.1 点击编辑按钮展示修改文章分类的弹出层

1. 为编辑按钮添加 `btn-edit` 类名如下：

```
1 <button type="button" class="layui-btn layui-btn-xs btn-edit" data-id="{{ $value.Id }}">编辑</button>
```

2. 定义 `修改分类` 的弹出层：

```
1 <script type="text/html" id="dialog-edit">
2   <form class="layui-form" id="form-edit" lay-filter="form-edit">
3     <!-- 隐藏域，保存 Id 的值 -->
4     <input type="hidden" name="Id">
5     <div class="layui-form-item">
6       <label class="layui-form-label">分类名称</label>
7       <div class="layui-input-block">
8         <input type="text" name="name" required lay-verify="required"
placeholder="请输入分类名称" autocomplete="off" class="layui-input">
9       </div>
10    </div>
11    <div class="layui-form-item">
12      <label class="layui-form-label">分类别名</label>
13      <div class="layui-input-block">
14        <input type="text" name="alias" required lay-verify="required"
placeholder="请输入分类别名" autocomplete="off" class="layui-input">
15      </div>
16    </div>
17    <div class="layui-form-item">
18      <div class="layui-input-block">
19        <button class="layui-btn" lay-submit lay-filter="formDemo">确认修
改</button>
20      </div>
21    </div>
22  </form>
23 </script>
```

3. 通过 `代理` 的形式，为 `btn-edit` 按钮绑定点击事件：

```
1 var indexEdit = null
2 $('tbody').on('click', '.btn-edit', function() {
3   // 弹出一个修改文章分类信息的层
4   indexEdit = layer.open({
5     type: 1,
6     area: ['500px', '250px'],
7     title: '修改文章分类',
8     content: $('#dialog-edit').html()
9   })
10 })
```

## 1.2 为修改文章分类的弹出层填充表单数据

1. 为编辑按钮绑定 `data-id` 自定义属性:

```
1 <button type="button" class="layui-btn layui-btn-xs btn-edit" data-id="{{ $value.Id }}">编辑</button>
```

2. 在展示弹出层之后, 根据 id 的值发起请求获取文章分类的数据, 并填充到表单中:

```
1 var id = $(this).attr('data-id')
2 // 发起请求获取对应分类的数据
3 $.ajax({
4     method: 'GET',
5     url: '/my/article/cates/' + id,
6     success: function(res) {
7         form.val('form-edit', res.data)
8     }
9 })
```

## 1.3 更新文章分类的数据

1. 通过代理的形式, 为修改分类的表单绑定 submit 事件:

```
1 $('body').on('submit', '#form-edit', function(e) {
2     e.preventDefault()
3     $.ajax({
4         method: 'POST',
5         url: '/my/article/updatecate',
6         data: $(this).serialize(),
7         success: function(res) {
8             if (res.status !== 0) {
9                 return layer.msg('更新分类数据失败! ')
10            }
11            layer.msg('更新分类数据成功! ')
12            layer.close(indexEdit)
13            initArtCateList()
14        }
15    })
16 })
```

## 1.4 删除文章分类

1. 为删除按钮绑定 `btn-delete` 类名, 并添加 `data-id` 自定义属性:

```
1 <button type="button" class="layui-btn layui-btn-danger layui-btn-xs btn-delete" data-id="{{ $value.Id }}">删除</button>
```

2. 通过代理的形式, 为删除按钮绑定点击事件:

```
1 $('tbody').on('click', '.btn-delete', function() {
2     var id = $(this).attr('data-id')
3     // 提示用户是否要删除
4     layer.confirm('确认删除?', { icon: 3, title: '提示' }, function(index)
5     {
```

```

5      $.ajax({
6          method: 'GET',
7          url: '/my/article/deletcate/' + id,
8          success: function(res) {
9              if (res.status !== 0) {
10                 return layer.msg('删除分类失败! ')
11             }
12             layer.msg('删除分类成功! ')
13             layer.close(index)
14             initArtCateList()
15         }
16     })
17 })
18 })

```

## 2. 文章列表

### 2.1 创建文章列表页面

1. 新建 `/article/art_list.html` 页面结构如下:

```

1      <!DOCTYPE html>
2      <html lang="en">
3          <head>
4              <meta charset="UTF-8" />
5              <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6              <title>Document</title>
7              <link rel="stylesheet" href="/assets/lib/layui/css/layui.css" />
8              <link rel="stylesheet" href="/assets/css/article/art_list.css" />
9          </head>
10         <body>
11             <!-- 卡片区域 -->
12             <div class="layui-card">
13                 <div class="layui-card-header">文章列表</div>
14                 <div class="layui-card-body"></div>
15             </div>
16
17             <!-- 导入第三方的 JS 插件 -->
18             <script src="/assets/lib/layui/layui.all.js"></script>
19             <script src="/assets/lib/jquery.js"></script>
20             <script src="/assets/js/baseAPI.js"></script>
21             <!-- 导入自己的 JS 脚本 -->
22             <script src="/assets/js/article/art_list.js"></script>
23         </body>
24     </html>
25

```

2. 新建 `/assets/css/article/art_list.css` 样式表如下:

```

1  html,
2  body {
3      margin: 0;
4      padding: 0;
5  }
6
7  body {
8      padding: 15px;
9      background-color: #f2f3f5;
10 }

```

3. 新建 `/assets/js/article/art_list.js` 脚本文件。

## 2.2 定义查询参数对象q

1. 定义一个查询的参数对象如下：

```

1      // 定义一个查询的参数对象，将来请求数据的时候，
2      // 需要将请求参数对象提交到服务器
3      var q = {
4          pagenum: 1, // 页码值，默认请求第一页的数据
5          pagesize: 2, // 每页显示几条数据，默认每页显示2条
6          cate_id: '', // 文章分类的 Id
7          state: '' // 文章的发布状态
8      }

```

## 2.3 请求文章列表数据并使用模板引擎渲染列表结构

1. 定义获取文章列表数据的方法如下：

```

1  initTable()
2
3      // 获取文章列表数据的方法
4  function initTable() {
5      $.ajax({
6          method: 'GET',
7          url: '/my/article/list',
8          data: q,
9          success: function(res) {
10             if (res.status !== 0) {
11                 return layer.msg('获取文章列表失败！')
12             }
13             // 使用模板引擎渲染页面的数据
14             var htmlStr = template('tpl-table', res)
15             $('tbody').html(htmlStr)
16         }
17     })
18 }

```

2. 在页面中添加表格结构如下：

```

1  <!-- 列表区域 -->
2  <table class="layui-table">
3      <colgroup>

```

```

4      <col />
5      <col width="150" />
6      <col width="180" />
7      <col width="150" />
8      <col width="150" />
9  </colgroup>
10 <thead>
11   <tr>
12     <th>文章标题</th>
13     <th>分类</th>
14     <th>发表时间</th>
15     <th>状态</th>
16     <th>操作</th>
17   </tr>
18 </thead>
19 <tbody></tbody>
20 </table>

```

### 3. 定义列表数据的模板结构：

```

1  <script type="text/html" id="tpl-table">
2    {{each data}}
3    <tr>
4      <td>{{ $value.title }}</td>
5      <td>{{ $value.cate_name }}</td>
6      <td>{{ $value.pub_date | dateFormat }}</td>
7      <td>{{ $value.state }}</td>
8      <td>
9        <button type="button" class="layui-btn layui-btn-xs">编辑</button>
10       <button type="button" class="layui-btn layui-btn-danger layui-btn-
xs">删除</button>
11     </td>
12   </tr>
13   {{/each}}
14 </script>

```

## 2.4 定义美化时间格式的过滤器

### 1. 通过 `template.defaults.imports` 定义过滤器：

```

1  // 定义美化时间的过滤器
2  template.defaults.imports.dateFormat = function(date) {
3    const dt = new Date(date)
4
5    var y = dt.getFullYear()
6    var m = padZero(dt.getMonth() + 1)
7    var d = padZero(dt.getDate())
8
9    var hh = padZero(dt.getHours())
10   var mm = padZero(dt.getMinutes())
11   var ss = padZero(dt.getSeconds())
12
13   return y + '-' + m + '-' + d + ' ' + hh + ':' + mm + ':' + ss
14 }
15
16 // 定义补零的函数

```

```

17     function padZero(n) {
18         return n > 9 ? n : '0' + n
19     }

```

2. 在模板引擎中使用过滤器:

```

1     <td>{{ $value.pub_date | dateFormat }}</td>

```

## 2.5 绘制筛选区域的UI结构

1. 绘制 UI 结构:

```

1     <!-- 筛选区域 -->
2     <form class="layui-form" id="form-search">
3         <div class="layui-form-item layui-inline">
4             <select name="cate_id"></select>
5         </div>
6         <div class="layui-form-item layui-inline">
7             <select name="state">
8                 <option value="">所有状态</option>
9                 <option value="已发布">已发布</option>
10                <option value="草稿">草稿</option>
11            </select>
12        </div>
13        <div class="layui-form-item layui-inline">
14            <button class="layui-btn" lay-submit lay-filter="formDemo">筛选
15        </div>
16    </form>

```

## 2.6 发起请求获取并渲染文章分类的下拉选择框

1. 定义 `initCate` 函数请求文章分类的列表数据:

```

1     initCate()
2
3     // 初始化文章分类的方法
4     function initCate() {
5         $.ajax({
6             method: 'GET',
7             url: '/my/article/cates',
8             success: function(res) {
9                 if (res.status !== 0) {
10                    return layer.msg('获取分类数据失败!')
11                }
12                // 调用模板引擎渲染分类的可选项
13                var htmlStr = template('tpl-cate', res)
14                $('[name=cate_id]').html(htmlStr)
15                // 通过 layui 重新渲染表单区域的UI结构
16                form.render()
17            }
18        })
19    }

```

## 2. 定义分类可选项的模板结构：

```
1 <script type="text/html" id="tpl-cate">
2   <option value="">所有分类</option>
3   {{each data}}
4   <option value="{{ $value.Id }}">{{ $value.name }}</option>
5   {{/each}}
6 </script>
```

## 2.7 实现筛选的功能

### 1. 为筛选表单绑定 submit 事件：

```
1 $('#form-search').on('submit', function(e) {
2   e.preventDefault()
3   // 获取表单中选中项的值
4   var cate_id = $('[name=cate_id]').val()
5   var state = $('[name=state]').val()
6   // 为查询参数对象 q 中对应的属性赋值
7   q.cate_id = cate_id
8   q.state = state
9   // 根据最新的筛选条件，重新渲染表格的数据
10  initTable()
11 })
```

## 3. 分页

### 3.1 定义渲染分页的 renderPage 方法

#### 1. 定义渲染分页的方法：

```
1 function renderPage(total) {
2   console.log(total)
3 }
```

#### 2. 在 `initTable` 中调用 `renderPage` 方法：

```
1 function initTable() {
2   $.ajax({
3     method: 'GET',
4     url: '/my/article/list',
5     data: q,
6     success: function(res) {
7       if (res.status !== 0) {
8         return layer.msg('获取文章列表失败！')
9       }
10      // 使用模板引擎渲染页面的数据
11      var htmlStr = template('tpl-table', res)
12      $('tbody').html(htmlStr)
13      // 调用渲染分页的方法
14      renderPage(res.total)
15    }
16  })
17 }
```

```
16     })
17 }
```

### 3.2 调用 laypage.render 方法渲染分页的基本结构

1. 在页面中定义分页的区域：

```
1     <!-- 分页区域 -->
2     <div id="pageBox"></div>
```

2. 调用 laypage.render() 方法来渲染分页的结构：

```
1     // 定义渲染分页的方法
2     function renderPage(total) {
3         // 调用 laypage.render() 方法来渲染分页的结构
4         laypage.render({
5             elem: 'pageBox', // 分页容器的 Id
6             count: total, // 总数据条数
7             limit: q.pagesize, // 每页显示几条数据
8             curr: q.pagenum // 设置默认被选中的分页
9         })
10    }
```

### 3.3 在jump回调函数中通过obj.curr获取到最新的页码值

```
1     // 定义渲染分页的方法
2     function renderPage(total) {
3         // 调用 laypage.render() 方法来渲染分页的结构
4         laypage.render({
5             elem: 'pageBox', // 分页容器的 Id
6             count: total, // 总数据条数
7             limit: q.pagesize, // 每页显示几条数据
8             curr: q.pagenum, // 设置默认被选中的分页
9             // 分页发生切换的时候，触发 jump 回调
10            jump: function(obj) {
11                console.log(obj.curr)
12                // 把最新的页码值，赋值到 q 这个查询参数对象中
13                q.pagenum = obj.curr
14            }
15        })
16    }
```

### 3.4 解决 jump 回调函数发生死循环的问题

```
1     // 定义渲染分页的方法
2     function renderPage(total) {
3         // 调用 laypage.render() 方法来渲染分页的结构
4         laypage.render({
5             elem: 'pageBox', // 分页容器的 Id
6             count: total, // 总数据条数
7             limit: q.pagesize, // 每页显示几条数据
```



```

8      curr: q.pagenum, // 设置默认被选中的分页
9      // 分页发生切换的时候, 触发 jump 回调
10     // 触发 jump 回调的方式有两种:
11     // 1. 点击页码的时候, 会触发 jump 回调
12     // 2. 只要调用了 laypage.render() 方法, 就会触发 jump 回调
13     jump: function(obj, first) {
14         // 可以通过 first 的值, 来判断是通过哪种方式, 触发的 jump 回调
15         // 如果 first 的值为 true, 证明是方式2触发的
16         // 否则就是方式1触发的
17         console.log(first)
18         console.log(obj.curr)
19         // 把最新的页码值, 赋值到 q 这个查询参数对象中
20         q.pagenum = obj.curr
21         // 根据最新的 q 获取对应的数据列表, 并渲染表格
22         // initTable()
23         if (!first) {
24             initTable()
25         }
26     }
27 })
28 }

```

### 3.5 自定义分页的功能项

```

1  // 定义渲染分页的方法
2  function renderPage(total) {
3      // 调用 laypage.render() 方法来渲染分页的结构
4      laypage.render({
5          elem: 'pageBox', // 分页容器的 Id
6          count: total, // 总数据条数
7          limit: q.pagesize, // 每页显示几条数据
8          curr: q.pagenum, // 设置默认被选中的分页
9          layout: ['count', 'limit', 'prev', 'page', 'next', 'skip'],
10         limits: [2, 3, 5, 10],
11         // 分页发生切换的时候, 触发 jump 回调
12         // 触发 jump 回调的方式有两种:
13         // 1. 点击页码的时候, 会触发 jump 回调
14         // 2. 只要调用了 laypage.render() 方法, 就会触发 jump 回调
15         jump: function(obj, first) {
16             // 可以通过 first 的值, 来判断是通过哪种方式, 触发的 jump 回调
17             // 如果 first 的值为 true, 证明是方式2触发的
18             // 否则就是方式1触发的
19             console.log(first)
20             console.log(obj.curr)
21             // 把最新的页码值, 赋值到 q 这个查询参数对象中
22             q.pagenum = obj.curr
23             // 根据最新的 q 获取对应的数据列表, 并渲染表格
24             // initTable()
25             if (!first) {
26                 initTable()
27             }
28         }
29     })
30 }

```

## 3.6 实现切换每页展示多少条数据的功能

```
1 // 定义渲染分页的方法
2 function renderPage(total) {
3     // 调用 laypage.render() 方法来渲染分页的结构
4     laypage.render({
5         elem: 'pageBox', // 分页容器的 Id
6         count: total, // 总数据条数
7         limit: q.pagesize, // 每页显示几条数据
8         curr: q.pagenum, // 设置默认被选中的分页
9         layout: ['count', 'limit', 'prev', 'page', 'next', 'skip'],
10        limits: [2, 3, 5, 10],
11        // 分页发生切换的时候, 触发 jump 回调
12        // 触发 jump 回调的方式有两种:
13        // 1. 点击页码的时候, 会触发 jump 回调
14        // 2. 只要调用了 laypage.render() 方法, 就会触发 jump 回调
15        jump: function(obj, first) {
16            // 可以通过 first 的值, 来判断是通过哪种方式, 触发的 jump 回调
17            // 如果 first 的值为 true, 证明是方式2触发的
18            // 否则就是方式1触发的
19            console.log(first)
20            console.log(obj.curr)
21            // 把最新的页码值, 赋值到 q 这个查询参数对象中
22            q.pagenum = obj.curr
23            // 把最新的条目数, 赋值到 q 这个查询参数对象的 pagesize 属性中
24            q.pagesize = obj.limit
25            // 根据最新的 q 获取对应的数据列表, 并渲染表格
26            // initTable()
27            if (!first) {
28                initTable()
29            }
30        }
31    })
32 }
```

## 4. 删除文章

### 4.1 实现删除文章的功能

1. 为删除按钮绑定 `btn-delete` 类名和 `data-id` 自定义属性:

```
1 <button type="button" class="layui-btn layui-btn-danger layui-btn-xs btn-
  delete" data-id="{{ $value.Id }}">删除</button>
```

2. 通过代理的形式, 为删除按钮绑定点击事件处理函数:

```
1 $('tbody').on('click', '.btn-delete', function() {
2     // 获取到文章的 id
3     var id = $(this).attr('data-id')
4     // 询问用户是否要删除数据
```

```

5      layer.confirm('确认删除?', { icon: 3, title: '提示' }, function(index)
    {
6          $.ajax({
7              method: 'GET',
8              url: '/my/article/delete/' + id,
9              success: function(res) {
10                 if (res.status !== 0) {
11                     return layer.msg('删除文章失败! ')
12                 }
13                 layer.msg('删除文章成功! ')
14                 initTable()
15             }
16         })
17
18         layer.close(index)
19     })
20 })

```

## 4.2 解决删除文章时的小 Bug

```

1  $('tbody').on('click', '.btn-delete', function() {
2      // 获取删除按钮的个数
3      var len = $('.btn-delete').length
4      // 获取到文章的 id
5      var id = $(this).attr('data-id')
6      // 询问用户是否要删除数据
7      layer.confirm('确认删除?', { icon: 3, title: '提示' }, function(index) {
8          $.ajax({
9              method: 'GET',
10             url: '/my/article/delete/' + id,
11             success: function(res) {
12                 if (res.status !== 0) {
13                     return layer.msg('删除文章失败! ')
14                 }
15                 layer.msg('删除文章成功! ')
16                 // 当数据删除完成后, 需要判断当前这一页中, 是否还有剩余的数据
17                 // 如果没有剩余的数据了, 则让页码值 -1 之后,
18                 // 再重新调用 initTable 方法
19                 // 4
20                 if (len === 1) {
21                     // 如果 len 的值等于1, 证明删除完毕之后, 页面上就没有任何数据了
22                     // 页码值最小必须是 1
23                     q.pagenum = q.pagenum === 1 ? 1 : q.pagenum - 1
24                 }
25                 initTable()
26             }
27         })
28
29         layer.close(index)
30     })
31 })

```

## 5. 发布文章

### 5.1 创建文章发布页面的基本结构

1. 新建 `/article/art_pub.html` 页面结构如下：

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8" />
5     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6     <title>Document</title>
7     <link rel="stylesheet" href="/assets/lib/layui/css/layui.css" />
8     <link rel="stylesheet" href="/assets/css/article/art_pub.css" />
9   </head>
10  <body>
11    <!-- 卡片区域 -->
12    <div class="layui-card">
13      <div class="layui-card-header">写文章</div>
14      <div class="layui-card-body">
15        卡片式面板通常用于非白色背景色的主体内<br />
16        从而映衬出边框投影
17      </div>
18    </div>
19
20    <!-- 导入第三方的 JS 插件 -->
21    <script src="/assets/lib/layui/layui.all.js"></script>
22    <script src="/assets/lib/jquery.js"></script>
23    <script src="/assets/js/baseAPI.js"></script>
24    <!-- 导入自己的 JS -->
25    <script src="/assets/js/article/art_pub.js"></script>
26  </body>
27 </html>
28
```

2. 新建 `/assets/css/article/art_pub.css` 样式文件如下：

```
1 html,
2 body {
3   margin: 0;
4   padding: 0;
5 }
6
7 body {
8   padding: 15px;
9   background-color: #f2f3f5;
10 }
```

3. 新建 `/assets/js/article/art_pub.js` 脚本文件如下：

```
1 $(function() { })
```

### 5.2 新建基本的表单结构

```

1  <!-- 发布文章的表单 -->
2  <form class="layui-form">
3    <div class="layui-form-item">
4      <label class="layui-form-label">文章标题</label>
5      <div class="layui-input-block">
6        <input type="text" name="title" required lay-verify="required"
placeholder="请输入标题" autocomplete="off" class="layui-input" />
7      </div>
8    </div>
9  </form>

```

### 5.3 渲染文章类别对应的下拉选择框结构

1. 定义 UI 结构:

```

1  <!-- 第二行 -->
2  <div class="layui-form-item">
3    <label class="layui-form-label">文章类别</label>
4    <div class="layui-input-block">
5      <select name="cate_id" lay-verify="required"></select>
6    </div>
7  </div>

```

2. 导入 art-template:

```

1  <script src="/assets/lib/template-web.js"></script>

```

3. 定义模板结构:

```

1  <script type="text/html" id="tpl-cate">
2    <option value="">请选择文章类别</option>
3    {{each data}}
4    <option value="{{ $value.Id }}">{{ $value.name }}</option>
5    {{/each}}
6  </script>

```

4. 定义 `initCate` 方法:

```

1  $(function() {
2    var layer = layui.layer
3    var form = layui.form
4
5    initCate()
6    // 定义加载文章分类的方法
7    function initCate() {
8      $.ajax({
9        method: 'GET',
10       url: '/my/article/cates',
11       success: function(res) {
12         if (res.status !== 0) {
13           return layer.msg('初始化文章分类失败! ')
14         }
15         // 调用模板引擎, 渲染分类的下拉菜单
16         var htmlStr = template('tpl-cate', res)
17         $('[name=cate_id]').html(htmlStr)
18         // 一定要记得调用 form.render() 方法

```

```

19         form.render()
20     }
21 })
22 }
23 })

```

## 5.4 渲染富文本编辑器

参考 [素材/富文本和封面.md](#) 中的实现步骤

## 5.5 渲染封面裁剪区域

参考 [素材/富文本和封面.md](#) 中的实现步骤

## 5.6 渲染提交按钮区域

```

1     <!-- 第五行 -->
2     <div class="layui-form-item">
3         <div class="layui-input-block">
4             <button class="layui-btn lay-submit">发布</button>
5             <button class="layui-btn layui-btn-primary lay-submit">存为草稿</button>
6         </div>
7     </div>

```

## 5.7 点击选择封面按钮打开文件选择框

1. 修改 UI 结构，为 [选择封面](#) 按钮添加 `id`，并且在按钮后面添加 [文件选择框](#)：

```

1     <!-- 选择封面按钮 -->
2     <button type="button" class="layui-btn layui-btn-danger"
3         id="btnChooseImage">选择封面</button>
4     <!-- 隐藏的文件选择框 -->
5     <input type="file" id="coverFile" style="display: none;"
6         accept="image/png,image/jpeg,image/gif" />

```

2. 为选择封面的按钮，绑定点击事件处理函数：

```

1     $('#btnChooseImage').on('click', function() {
2         $('#coverFile').click()
3     })

```

## 5.8 将选择的图片设置到裁剪区域中

1. 监听 `coverFile` 的 `change` 事件，获取用户选择的文件列表：

```

1      // 监听 coverFile 的 change 事件，获取用户选择的文件列表
2      $('#coverFile').on('change', function(e) {
3          // 获取到文件的列表数组
4          var files = e.target.files
5          // 判断用户是否选择了文件
6          if (files.length === 0) {
7              return
8          }
9          // 根据文件，创建对应的 URL 地址
10         var newImgURL = URL.createObjectURL(files[0])
11         // 为裁剪区域重新设置图片
12         $image
13             .cropper('destroy') // 销毁旧的裁剪区域
14             .attr('src', newImgURL) // 重新设置图片路径
15             .cropper(options) // 重新初始化裁剪区域
16     })

```

## 5.9 分析发布文章的实现步骤

1. 为 存为草稿 按钮添加 id 属性：

```

1      <button class="layui-btn layui-btn-primary" lay-submit id="btnSave2">存为草稿</button>

```

2. 定义文章的发布状态：

```

1      var art_state = '已发布'

```

3. 为存为草稿按钮，绑定点击事件处理函数：

```

1      $('#btnSave2').on('click', function() {
2          art_state = '草稿'
3      })

```

## 5.10 基于Form表单创建FormData对象

1. 为发布文章的 Form 表单添加 id 属性：

```

1      <form class="layui-form" id="form-pub"></form>

```

2. 为表单绑定 submit 提交事件：

```

1      $('#form-pub').on('submit', function(e) {
2          // 1. 阻止表单的默认提交行为
3          e.preventDefault()
4          // 2. 基于 form 表单，快速创建一个 FormData 对象
5          var fd = new FormData($(this)[0])
6          // 3. 将文章的发布状态，存到 fd 中
7          fd.append('state', art_state)
8      })

```

## 5.11 将裁剪后的封面追加到FormData对象中

```
1 // 为表单绑定 submit 提交事件
2 $('#form-pub').on('submit', function(e) {
3     // 1. 阻止表单的默认提交行为
4     e.preventDefault()
5     // 2. 基于 form 表单，快速创建一个 FormData 对象
6     var fd = new FormData($(this)[0])
7     // 3. 将文章的发布状态，存到 fd 中
8     fd.append('state', art_state)
9     // 4. 将封面裁剪过后的图片，输出为一个文件对象
10    $image
11        .cropper('getCroppedCanvas', {
12            // 创建一个 Canvas 画布
13            width: 400,
14            height: 280
15        })
16        .toBlob(function(blob) {
17            // 将 Canvas 画布上的内容，转化为文件对象
18            // 得到文件对象后，进行后续的操作
19            // 5. 将文件对象，存储到 fd 中
20            fd.append('cover_img', blob)
21            // 6. 发起 ajax 数据请求
22        })
23    })
```

## 5.12 发起Ajax请求实现发布文章的功能

1. 定义一个发布文章的方法：

```
1 function publishArticle(fd) {
2     $.ajax({
3         method: 'POST',
4         url: '/my/article/add',
5         data: fd,
6         // 注意：如果向服务器提交的是 FormData 格式的数据，
7         // 必须添加以下两个配置项
8         contentType: false,
9         processData: false,
10        success: function(res) {
11            if (res.status !== 0) {
12                return layer.msg('发布文章失败! ')
13            }
14            layer.msg('发布文章成功! ')
15            // 发布文章成功后，跳转到文章列表页面
16            location.href = '/article/art_list.html'
17        }
18    })
19 }
```

2. 把裁剪的图片追加到 FormData 对象中之后，调用 publishArticle 方法：

```
1 // 为表单绑定 submit 提交事件
2 $('#form-pub').on('submit', function(e) {
3     // 1. 阻止表单的默认提交行为
```



```

4      e.preventDefault()
5      // 2. 基于 form 表单，快速创建一个 FormData 对象
6      var fd = new FormData($(this)[0])
7      // 3. 将文章的发布状态，存到 fd 中
8      fd.append('state', art_state)
9      // 4. 将封面裁剪过后的图片，输出为一个文件对象
10     $image
11     .cropper('getCroppedCanvas', {
12         // 创建一个 Canvas 画布
13         width: 400,
14         height: 280
15     })
16     .toBlob(function(blob) {
17         // 将 Canvas 画布上的内容，转化为文件对象
18         // 得到文件对象后，进行后续的操作
19         // 5. 将文件对象，存储到 fd 中
20         fd.append('cover_img', blob)
21         // 6. 发起 ajax 数据请求
22         publishArticle(fd)
23     })
24 })

```

## 5.13 将开发完成的项目代码推送到GitHub

1. 运行 `git add .` 命令
2. 运行 `git commit -m "完成文章管理相关功能的开发"` 命令
3. 运行 `git push -u origin article` 命令
4. 运行 `git checkout master` 命令
5. 运行 `git merge article` 命令
6. 运行 `git push` 命令