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：

```
<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：

```
<script type="text/html" id="dialog-edit">
  <form class="layui-form" id="form-edit" lay-filter="form-edit">
    <!-- 隐藏域，保存 Id 的值 -->
    <input type="hidden" name="Id">
    <div class="layui-form-item">
      <label class="layui-form-label">分类名称</label>
      <div class="layui-input-block">
        <input type="text" name="name" required  lay-verify="required" placeholder="请输入分类名称" autocomplete="off" class="layui-input">
      </div>
    </div>
    <div class="layui-form-item">
      <label class="layui-form-label">分类别名</label>
      <div class="layui-input-block">
        <input type="text" name="alias" required  lay-verify="required" placeholder="请输入分类别名" autocomplete="off" class="layui-input">
      </div>
    </div>
    <div class="layui-form-item">
      <div class="layui-input-block">
        <button class="layui-btn" lay-submit lay-filter="formDemo">确认修改</button>
      </div>
    </div>
  </form>
</script>
```

3. 通过 `代理` 的形式，为 `btn-edit` 按钮绑定点击事件：

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

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：

```
<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：

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

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：

```
$('body').on('submit', '#form-edit', function(e) {
    e.preventDefault()
    $.ajax({
          method: 'POST',
          url: '/my/article/updatecate',
          data: $(this).serialize(),
          success: function(res) {
            if (res.status !== 0) {
              return layer.msg('更新分类数据失败！')
            }
            layer.msg('更新分类数据成功！')
            layer.close(indexEdit)
            initArtCateList()
          }
    })
})
```

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：

```
<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：

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

```
        $.ajax({
          method: 'GET',
          url: '/my/article/deletecate/' + id,
          success: function(res) {
            if (res.status !== 0) {
              return layer.msg('删除分类失败！')
            }
            layer.msg('删除分类成功！')
            layer.close(index)
            initArtCateList()
          }
        })
      })
  })
```

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：

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <link rel="stylesheet" href="/assets/lib/layui/css/layui.css" />
    <link rel="stylesheet" href="/assets/css/article/art_list.css" />
  </head>
  <body>
    <!-- 卡片区域 -->
    <div class="layui-card">
      <div class="layui-card-header">文章列表</div>
      <div class="layui-card-body"></div>
    </div>

    <!-- 导入第三方的 JS 插件 -->
    <script src="/assets/lib/layui/layui.all.js"></script>
    <script src="/assets/lib/jquery.js"></script>
    <script src="/assets/js/baseAPI.js"></script>
    <!-- 导入自己的 JS 脚本 -->
    <script src="/assets/js/article/art_list.js"></script>
  </body>
</html>

```

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

```
html,
body {
  margin: 0;
  padding: 0;
}

body {
  padding: 15px;
  background-color: #f2f3f5;
}
```

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：

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

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：

```
initTable()

// 获取文章列表数据的方法
function initTable() {
    $.ajax({
      method: 'GET',
      url: '/my/article/list',
      data: q,
      success: function(res) {
        if (res.status !== 0) {
          return layer.msg('获取文章列表失败！')
        }
        // 使用模板引擎渲染页面的数据
        var htmlStr = template('tpl-table', res)
        $('tbody').html(htmlStr)
      }
    })
}
```

2. 在页面中添加表格结构如下：

```
<!-- 列表区域 -->
<table class="layui-table">
  <colgroup>
```

```
        <col />
        <col width="150" />
        <col width="180" />
        <col width="150" />
        <col width="150" />
      </colgroup>
      <thead>
        <tr>
          <th>文章标题</th>
          <th>分类</th>
          <th>发表时间</th>
          <th>状态</th>
          <th>操作</th>
        </tr>
      </thead>
      <tbody></tbody>
    </table>
```

3. 定义列表数据的模板结构：

```
    <script type="text/html" id="tpl-table">
      {{each data}}
      <tr>
        <td>{{$value.title}}</td>
        <td>{{$value.cate_name}}</td>
        <td>{{$value.pub_date|dataFormat}}</td>
        <td>{{$value.state}}</td>
        <td>
          <button type="button" class="layui-btn layui-btn-xs">编辑</button>
          <button type="button" class="layui-btn layui-btn-danger layui-btn-xs">删除</button>
        </td>
      </tr>
      {{/each}}
    </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：

```
      // 定义美化时间的过滤器
      template.defaults.imports.dataFormat = function(date) {
        const dt = new Date(date)

        var y = dt.getFullYear()
        var m = padZero(dt.getMonth() + 1)
        var d = padZero(dt.getDate())

        var hh = padZero(dt.getHours())
        var mm = padZero(dt.getMinutes())
        var ss = padZero(dt.getSeconds())

        return y + '-' + m + '-' + d + ' ' + hh + ':' + mm + ':' + ss
      }

      // 定义补零的函数
```

```
    function padZero(n) {
      return n > 9 ? n : '0' + n
    }
```

2. 在模板引擎中使用过滤器：

```
<td>{{$value.pub_date|dataFormat}}</td>
```

2.5 绘制筛选区域的UI结构

1. 绘制 UI 结构：

```
<!-- 筛选区域 -->
<form class="layui-form" id="form-search">
  <div class="layui-form-item layui-inline">
    <select name="cate_id"></select>
  </div>
  <div class="layui-form-item layui-inline">
    <select name="state">
      <option value="">所有状态</option>
      <option value="已发布">已发布</option>
      <option value="草稿">草稿</option>
    </select>
  </div>
  <div class="layui-form-item layui-inline">
    <button class="layui-btn" lay-submit lay-filter="formDemo">筛选
</button>
  </div>
</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：

```
    initCate()

    // 初始化文章分类的方法
    function initCate() {
      $.ajax({
        method: 'GET',
        url: '/my/article/cates',
        success: function(res) {
          if (res.status !== 0) {
            return layer.msg('获取分类数据失败！')
          }
          // 调用模板引擎渲染分类的可选项
          var htmlStr = template('tpl-cate', res)
          $('[name=cate_id]').html(htmlStr)
          // 通过 layui 重新渲染表单区域的UI结构
          form.render()
        }
      })
    }
```

2. 定义分类可选项的模板结构：

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

2.7 实现筛选的功能

1. 为筛选表单绑定 submit 事件：

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

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：

```
function renderPage(total) {
    console.log(total)
}
```

2. 在 `initTable` 中调用 `renderPage` 方法：

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

```
        })
    }
```

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：

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

2. 调用 laypage.render() 方法来渲染分页的结构：

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

3.3 在jump回调函数中通过obj.curr获取到最新的页码值

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

3.4 解决 jump 回调函数发生死循环的问题

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

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

3.5 自定义分页的功能项

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

3.6 实现切换每页展示多少条数据的功能

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

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：

```
<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：

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

```
        layer.confirm('确认删除?', { icon: 3, title: '提示' }, function(index) {
          $.ajax({
            method: 'GET',
            url: '/my/article/delete/' + id,
            success: function(res) {
              if (res.status !== 0) {
                return layer.msg('删除文章失败! ')
              }
              layer.msg('删除文章成功! ')
              initTable()
            }
          })

          layer.close(index)
        })
      })
```

4.2 解决删除文章时的小 Bug

```
$('tbody').on('click', '.btn-delete', function() {
    // 获取删除按钮的个数
    var len = $('.btn-delete').length
    // 获取到文章的 id
    var id = $(this).attr('data-id')
    // 询问用户是否要删除数据
    layer.confirm('确认删除?', { icon: 3, title: '提示' }, function(index) {
      $.ajax({
        method: 'GET',
        url: '/my/article/delete/' + id,
        success: function(res) {
          if (res.status !== 0) {
            return layer.msg('删除文章失败! ')
          }
          layer.msg('删除文章成功! ')
          // 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
          // 如果没有剩余的数据了,则让页码值 -1 之后,
          // 再重新调用 initTable 方法
          // 4
          if (len === 1) {
            // 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
            // 页码值最小必须是 1
            q.pagenum = q.pagenum === 1 ? 1 : q.pagenum - 1
          }
          initTable()
        }
      })

      layer.close(index)
    })
})
```

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：

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <link rel="stylesheet" href="/assets/lib/layui/css/layui.css" />
    <link rel="stylesheet" href="/assets/css/article/art_pub.css" />
  </head>
  <body>
    <!-- 卡片区域 -->
    <div class="layui-card">
      <div class="layui-card-header">写文章</div>
      <div class="layui-card-body">
        卡片式面板面板通常用于非白色背景色的主体内<br />
        从而映衬出边框投影
      </div>
    </div>

    <!-- 导入第三方的 JS 插件 -->
    <script src="/assets/lib/layui/layui.all.js"></script>
    <script src="/assets/lib/jquery.js"></script>
    <script src="/assets/js/baseAPI.js"></script>
    <!-- 导入自己的 JS -->
    <script src="/assets/js/article/art_pub.js"></script>
  </body>
</html>

```

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

```
html,
body {
  margin: 0;
  padding: 0;
}

body {
  padding: 15px;
  background-color: #f2f3f5;
}
```

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

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

5.2 新建基本的表单结构

```
<!-- 发布文章的表单 -->
<form class="layui-form">
  <div class="layui-form-item">
    <label class="layui-form-label">文章标题</label>
    <div class="layui-input-block">
      <input type="text" name="title" required lay-verify="required"
placeholder="请输入标题" autocomplete="off" class="layui-input" />
    </div>
  </div>
</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：

```
    <!-- 第二行 -->
    <div class="layui-form-item">
      <label class="layui-form-label">文章类别</label>
      <div class="layui-input-block">
        <select name="cate_id" lay-verify="required"></select>
      </div>
    </div>
```

2. 导入 art-template：

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

3. 定义模板结构：

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

4. 定义 `initCate` 方法：

```
  $(function() {
    var layer = layui.layer
    var form = layui.form

    initCate()
    // 定义加载文章分类的方法
    function initCate() {
      $.ajax({
        method: 'GET',
        url: '/my/article/cates',
        success: function(res) {
          if (res.status !== 0) {
            return layer.msg('初始化文章分类失败！')
          }
          // 调用模板引擎，渲染分类的下拉菜单
          var htmlStr = template('tpl-cate', res)
          $('[name=cate_id]').html(htmlStr)
          // 一定要记得调用 form.render() 方法
```

```
            form.render()
          }
        })
      }
    })
```

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

```
    <!-- 第五行 -->
    <div class="layui-form-item">
      <div class="layui-input-block">
        <button class="layui-btn" lay-submit>发布</button>
        <button class="layui-btn layui-btn-primary" lay-submit>存为草稿</button>
      </div>
    </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`：

```
    <!-- 选择封面按钮 -->
    <button type="button" class="layui-btn layui-btn-danger"
    id="btnChooseImage">选择封面</button>
    <!-- 隐藏的文件选择框 -->
    <input type="file" id="coverFile" style="display: none;"
    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：

```
    $('#btnChooseImage').on('click', function() {
        $('#coverFile').click()
    })
```

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：

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

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：

```
<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：

```
var art_state = '已发布'
```

3. 为存为草稿按钮，绑定点击事件处理函数：

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

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：

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

2. 为表单绑定 submit 提交事件：

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

5.11 将裁剪后的封面追加到FormData对象中

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

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：

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

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

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

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

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