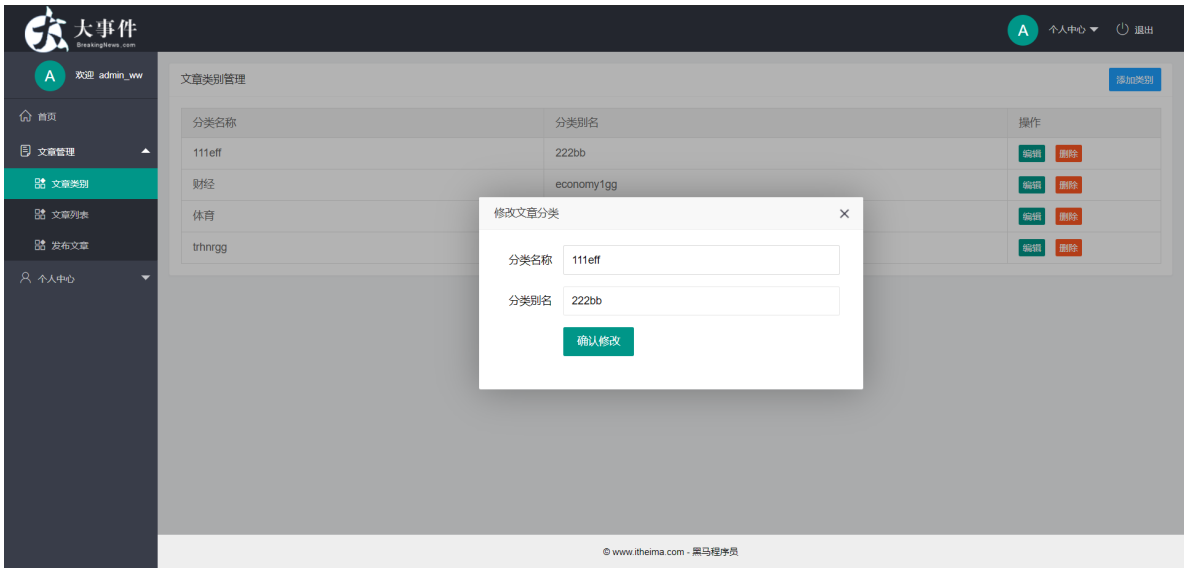
目标

- 能够完成文章类别的编辑功能
- 能够完成文章类别的删除功能
- 能够知道文章列表的分页查询参数的作用
- 能够完成文章列表的筛选功能
- 能够参照文档使用 `LayUI` 提供的自定义分页 `UI` 效果
- 知道 `laypage` 内置模块里面 `layout` 属性和 `limits` 属性的作用
- 能够解决 `jump` 回调函数中死循环的问题
- 能够获取 `jump` 回调中关键数据，页码值和每页条目数
- 能够实现分页查询功能
- 能够完成删除文章功能
- 能够完成新增文章功能

1. 文章类别(☆☆☆)

1.1 编辑功能



1.1.1 点击编辑按钮展示修改文章分类的弹出层

- 为编辑按钮添加 `btn-edit` 类名如下

```
<button type="button" class="layui-btn layui-btn-xs btn-edit">编辑</button>
```

- 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>
```

- 通过 **事件委派** 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()
  })
})
```

1.1.2 为修改文章分类的弹出层填充表单数据

点击不同的条目，那么编辑里面的内容应该是对应的，目前问题，在代码层面，是不知道到底点击了哪个按钮，这里我们可以通过给对应 编辑按钮绑定id来实现

- 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>
```

- 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

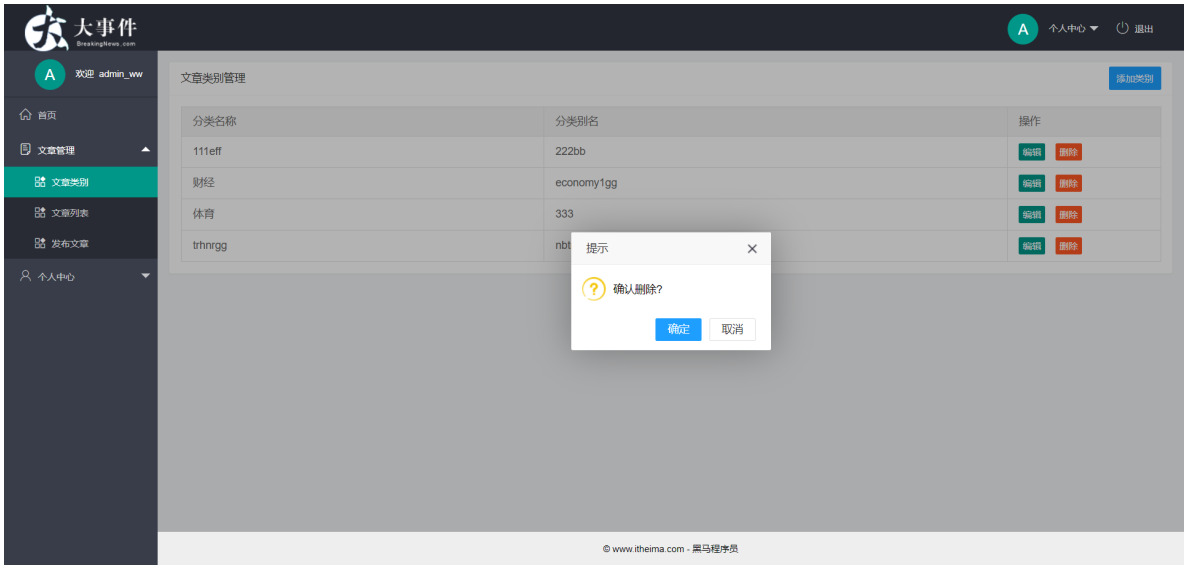
```
var id = $(this).attr('data-id')
// 发起请求获取对应分类的数据
$.ajax({
  method: 'GET',
  url: '/my/article/cates/' + id,
  success: function(res) {
    form.val('form-edit', res.data)
  }
})
```

1.1.3 更新文章分类的数据

- 通过 事件委派 的方式，给修改按钮绑定点击事件
- 查阅接口文档，利用 `$.ajax()` 发送 `post` 请求
- 判断服务器返回的 status 是否等于0，如果不等于，利用 `layer.msg` 提示用户
- 如果等于0，代表编辑成功，关闭当前弹出层，调用 `initArtCateList()` 方法获取最新的数据

```
$('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()
          }
    })
})
```

1.2 删除功能



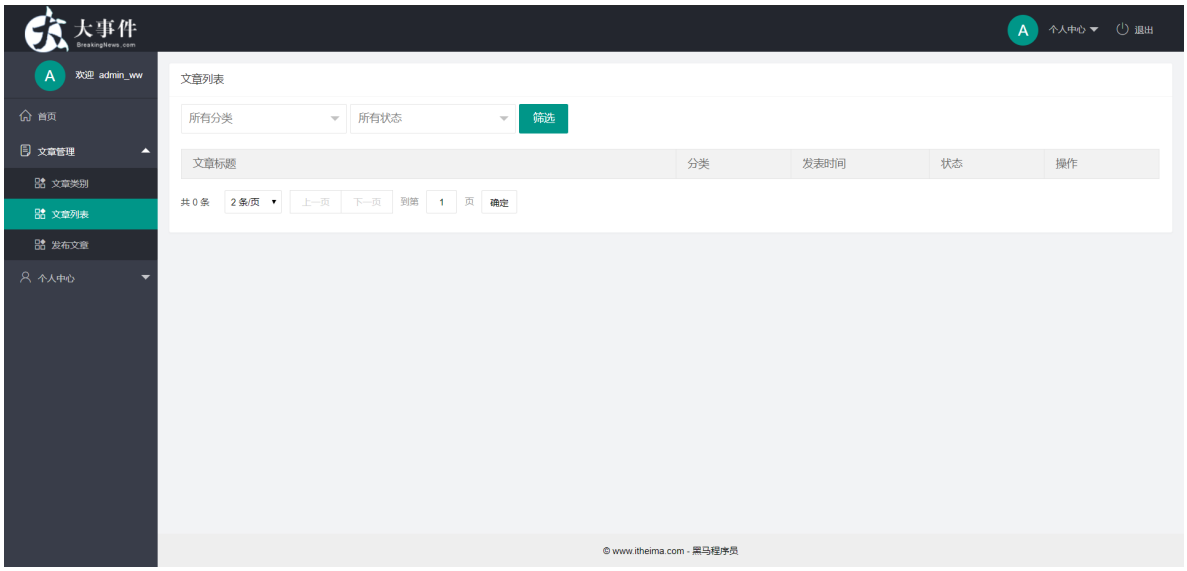
1.2.1 删除文章分类

- 为删除按钮绑定 `btn-delete` 类名，并添加 `data-id` 自定义属性
- 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. 文章列表功能



2.1 创建文章列表页面

- 新建 `/article/art_list.html` 页面
  - 利用 卡片区域搭建结构
  - 导入第三方的 `JS` 插件

```
<!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>
```

- 新建 `/assets/css/article/art_list.css` 样式表

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

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

- 新建 `/assets/js/article/art_list.js` 脚本文件

2.2 定义查询参数对象q

查阅接口文档我们发现，需要提交的参数是比较多的，所以我们可以先来定义一下这个查询的参数对象，由于服务器文章的数量会很多，不能一次性请求回来，这种功能也是很常见的，分页查询

请求URL:

- `/my/article/list`

请求方式:

- GET

Header:

```
Authorization:Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwidXNlcm5hbWUiOiJhZG1pbiIsInBhc3N3b3JkIjoiIiwibmlj
EA8LNrtMG04llKdZ33S9KBL3XeuBxuI
```

参数:

参数名	必选	类型	说明
pagenum	是	int	页码值
pagesize	是	int	每页显示多少条数据
cate_id	否	string	文章分类的 Id
state	否	string	文章的状态，可选值有：已发布、草稿

- `pagenum` 告知服务器我们需要的是第几页数据
- `pagesize` 每页能够显示的条目数
- `cate_id` 你需要找哪一个分类下的文章，默认是所有文章
- `state` 你需要查找的是什么状态的文章

```
// 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

- 定义获取文章列表数据的方法 `initTable()`

```
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)
      }
    })
}
```

- 在页面中添加表格结构

```
<!-- 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>
```

- 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

- 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
}
```

- 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

- 绘制 `UI` 结构
  - 默认的 两个select 和 button 都是独自占据一行的
  - 我们可以给三个模块设置 `layui-inline`
  - 分类的数据是动态进行获取的，第一个下拉框我们先不设置内容
  - 第二个下拉选择框里面内容是死的，我们可以定义三个 option 添加对应的内容即可

```
<!-- 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

- 定义 `initCate` 函数请求文章分类的列表数据
  - 利用 `$.ajax()` 发送get请求
  - 判断服务器返回的 `status` 是否等于0，如果不等于，提示用户
  - 如果等于，通过 `template` 得到渲染完毕后的 `html` 结构
  - 利用属性选择器渲染到第一个 `select` 里面

```
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()
    }
  })
}
```

- 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(☆☆☆)

- 需要先绑定表单的 `submit` 事件
- 在事件里面获取到表单中选中的值
- 然后把这个值同步到我们 参数对象 `q` 里面
- 再次调用 `initTable()` 方法即可

```
$('#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. 分页功能(☆☆☆)

3.1 定义渲染分页的 `renderPage` 方法

- 定义渲染分页的方法，接收一个总数量的参数

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

- 在渲染表格完后调用这个方法

```
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

- 借助 `layui` 提供的分页的方式

- 在页面中定义分页的区域

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

- 调用 `laypage.render()` 方法来渲染分页的结构
  - `elem` 分页容器的 id
  - `count` 总数据条数
  - `limit` 每页显示几条数据
  - `curr` 设置默认被选中的分页

```
// 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

- 分页发生切换的时候，触发 `jump` 回调
- 把最新的页码值赋值给请求参数 `q`

```
// 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

- 目前有个问题，默认这个 jump 是一个循环不断在触发，所以我们还不能直接进行网络请求
- 触发 `jump` 的原因
  - 点击页码的时候会触发 `jump` 的回调
  - 只要调用了 `laypage.render()` 方法就会触发 `jump` 回调
    - 我们在定义分页结构的时候，我们调用了 `laypage.render()` 方法，一旦调用，我们触发了 jump回调
    - 我们在 jump回调里面又调用了 `initTable()` 方法
    - `initTable()` 里面请求成功之后，又调用了 `laypage.render()` 方法，所以发生了死循环
- 判断 jump 回调的触发方式就可以解决了
- jump 的第二个参数就能帮我们判断出来，如果第二个参数为true，就代表是利用 `laypage.render()` 方法触发的

```
// 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

查阅了 `layui` 内置模块分页区域的文档后，我们发现可以通过一个参数来配置 `layout`

- 里面传递的是一个数组，分页的顺序是通过我们配置数组的顺序来显示的
- 通过 `limits` 参数可以来设置每页来显示多少条

```
// 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],// 每页展示多少条
      ...
    })
}
```

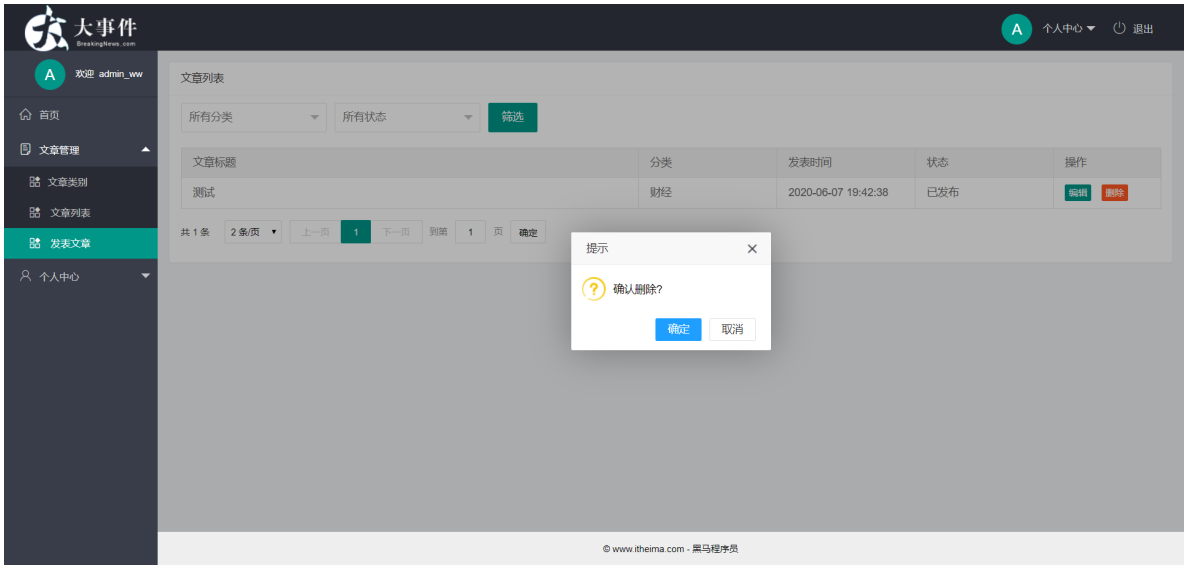
3.6 实现切换每页展示多少条数据的功能

- 通过 `jump` 回调来实现，当我们切换了 每页显示条目数后，会触发这个回调
- 通过 `obj.limit` 就能获取用户选择的是第几条
- 拿到最新条目数后，我们设置给请求对象 `q` 里面
- 然后重新调用 `initTable()` 来获取最新数据

```
// 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
        // 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
        if (!first) {
          initTable()
        }
      }
    })
}
```

4. 删除文章(☆☆☆)



4.1 实现删除文章的功能

- 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>
```

- 通过代理的形式，为删除按钮绑定点击事件处理函数
  - 弹出确认取消框提示用户
  - 用户点击确认，发送请求，删除当前文章，携带文章 `id`
  - 请求成功之后，获取最新的文章列表信息
  - 关闭当前弹出层

```
$('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

问题：删除后，页码值虽然正常，但是当前页码的数据没有渲染出来

解决：

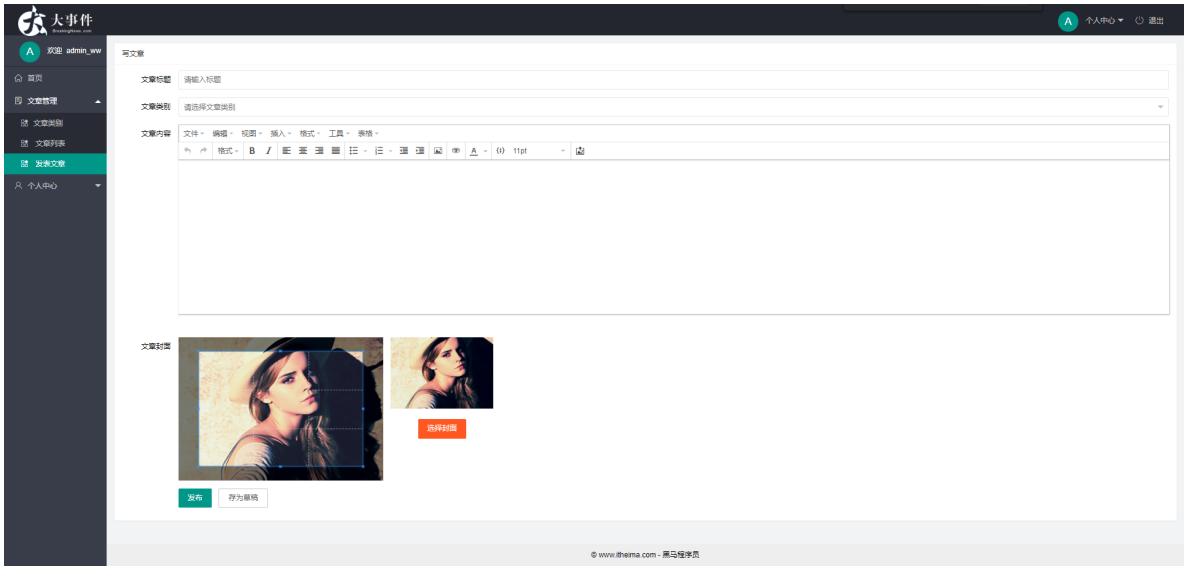
- 判断删除后，页面上是否还有数据
  - 判断当前页面的删除按钮的长度是否等于1
  - 如果等于1，那么我们让当前页码-1即可，如果不等于1，不用处理

```
$('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

- 新建 `/article/art_pub.html` 页面结构
  - 利用卡片区域进行结构布局
  - 导入第三方的 `JS` 插件

```
<!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>
```

- 新建 `/assets/css/article/art_pub.css` 样式文件

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

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

- 新建 `/assets/js/article/art_pub.js` 脚本文件

5.2 新建基本的表单结构

- 自上而下 文章标题 -> 文章类别 -> 文章内容 -> 文章封面 -> 表单操作按钮

```
<!-- 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

- 第二行右侧内容应该是下拉选择框 `select`，必填项，里面的内容是动态进行获取的

```
<!-- 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>
```

- 导入 `art-template`

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

- 定义模板结构，多个条目，利用列表渲染

```
<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>
```

- 定义 `initCate` 方法，请求文章类别的列表，利用模板引擎渲染到页面

```
$(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

- 添加如下的 `layui` 表单行

```
<div class="layui-form-item">
  <!-- 左侧的 label -->
  <label class="layui-form-label">文章内容</label>
  <!-- 为富文本编辑器外部的容器设置高度 -->
  <div class="layui-input-block" style="height: 400px;">
    <!-- 重要：将来这个 textarea 会被初始化为富文本编辑器 -->
    <textarea name="content"></textarea>
  </div>
</div>
```

- 导入富文本必须的 `script` 脚本

```
<!-- 富文本 -->
<script src="/assets/lib/tinymce/tinymce.min.js"></script>
<script src="/assets/lib/tinymce/tinymce_setup.js"></script>
```

- 调用 `initEditor()` 方法，初始化富文本编辑器

```
// 初始化富文本编辑器
initEditor()
```

5.5 渲染封面裁剪区域

- 在 `<head>` 中导入 `cropper.css` 样式表

```
<link rel="stylesheet" href="/assets/lib/cropper/cropper.css" />
```

- 在 `<body>` 的结束标签之前，按顺序导入如下的 `js` 脚本：

```
<script src="/assets/lib/jquery.js"></script>
<script src="/assets/lib/cropper/Cropper.js"></script>
<script src="/assets/lib/cropper/jquery-cropper.js"></script>
```

- 在表单中，添加如下的表单行结构

```
<div class="layui-form-item">
  <!-- 左侧的 label -->
  <label class="layui-form-label">文章封面</label>
  <!-- 选择封面区域 -->
  <div class="layui-input-block cover-box">
    <!-- 左侧裁剪区域 -->
    <div class="cover-left">
      <img id="image" src="/assets/images/sample2.jpg" alt="" />
    </div>
    <!-- 右侧预览区域和选择封面区域 -->
    <div class="cover-right">
      <!-- 预览的区域 -->
      <div class="img-preview"></div>
      <!-- 选择封面按钮 -->
      <button type="button" class="layui-btn layui-btn-danger">选择封面</button>
    </div>
  </div>
</div>
```

- 美化的样式

```
/* 封面容器的样式 */
.cover-box {
  display: flex;
}

/* 左侧裁剪区域的样式 */
.cover-left {
  width: 400px;
  height: 280px;
  overflow: hidden;
  margin-right: 20px;
}

/* 右侧盒子的样式 */
.cover-right {
  display: flex;
```

```
  flex-direction: column;
  align-items: center;
}

/* 预览区域的样式 */
.img-preview {
  width: 200px;
  height: 140px;
  background-color: #ccc;
  margin-bottom: 20px;
  overflow: hidden;
}
```

- 实现基本裁剪效果

```
// 1. 初始化图片裁剪器
var $image = $('#image')

// 2. 裁剪选项
var options = {
  aspectRatio: 400 / 280,
  preview: '.img-preview'
}

// 3. 初始化裁剪区域
$image.cropper(options)
```

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>
```

5.6 点击选择封面按钮打开文件选择框

- 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" />
```

- 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

- 监听 `coverFile` 的 `change` 事件，获取用户选择的文件列表
- 跟进文件，创建对应的 URL 地址
- 为裁剪区域重新设置图片
  - 销毁旧的裁剪区域
  - 重新设置图片路径
  - 重新初始化裁剪区域

```
// 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

- 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>
```

- 定义文章的发布状态

```
var art_state = '已发布'
```

- 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

- 为发布文章的 Form 表单添加 `id` 属性

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

- 为表单绑定 `submit` 提交事件

- 阻止表单默认提交行为
- 基于 `form` 表单，快速创建一个 `FormData` 对象
- 将文章的发布状态，存到 `FormData` 对象中

```
$('#form-pub').on('submit', function(e) {
  // 1. 阻止表单的默认提交行为
  e.preventDefault()
  // 2. 基于 form 表单，快速创建一个 FormData 对象
  var fd = new FormData($(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

- 将裁剪后的图片输出成一个文件
- 把文件追加到 `formData` 中即可

```
// 为表单绑定 submit 提交事件
$('#form-pub').on('submit', function(e) {
  ...
  // 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

- 定义一个发布文章的方法
- 注意：如果是提交的是`FormData`格式数据，需要添加 `contentType：false ，processData：false`

```
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'
      }
    })
}
```

- 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