评论管理

经过之前的文章管理和分类管理两个功能的开发过程，我们发现基本上都是相同的套路，而且对于任何一个业务最终都还是这些基础的**增删改查**。

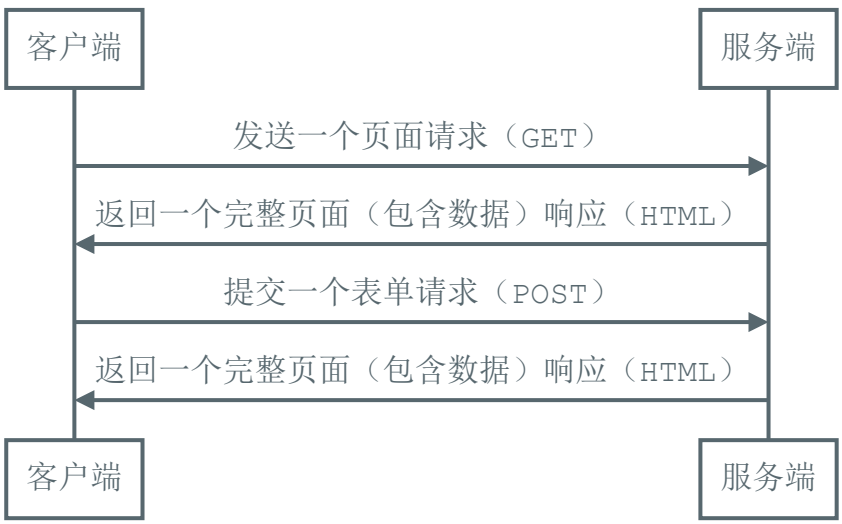
这里的评论管理我们就不在按照之前的实现方式（传统的动态网站方式）去完成了，取而代之的是 AJAX 方式。

> 咱们这个过程的设计就体现了 Web 技术的大致发展历史，了解并掌握这些有助于我们更好的使用当下最新最优的方案完成当下应用的开发。

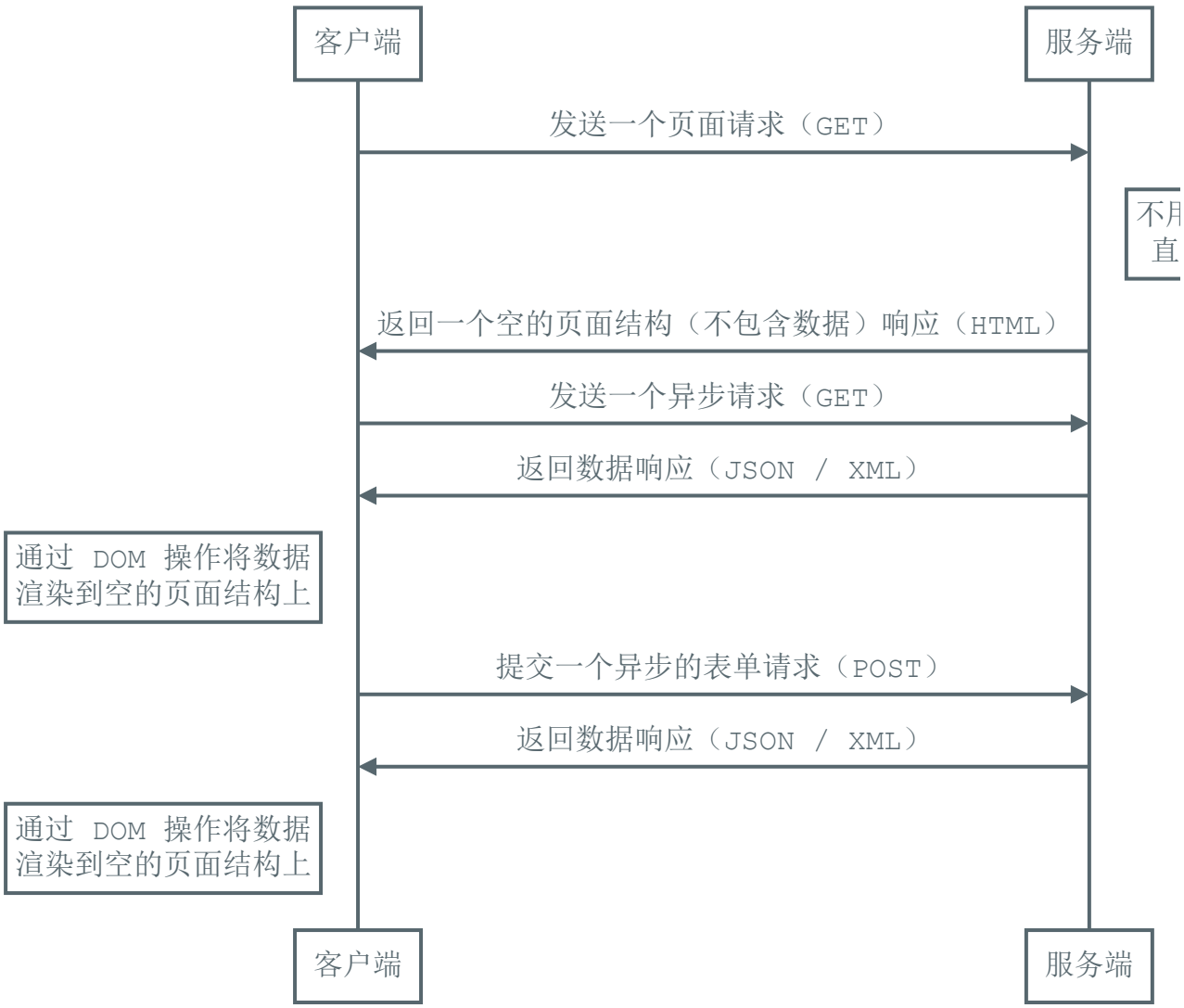
1. 前提说明

这里我们最后一次对比一下传统的动态网站方式和 AJAX 方式之间的差异：

传统方式



AJAX 方式



MAKE IT BETTER

2. 通过 AJAX 方式实现评论管理

2.1. 返回空页面结构

早在开始开发的时候，我们就已经将评论管理的静态页面整合进来了，这里我们只需要简单调整一下，包括：

1. 删除静态页开发时页面上写死的假数据
2. 该加 id 的加 id，该加 name 的加 name

既然是返回空页面，那么页面执行过程中就不需要有 PHP 代码的执行了，那么 PHP 对于这个过程就没有意义了。

> 提问：这里的评论管理页是应该用 html 文件，还是 php 文件？

> 源代码: step-58

2.2. 数据接口

既然是通过 AJAX 获取数据，然后在通过 DOM 操作渲染数据，我们首先第一前提就是要有一个可以获取评论数据的接口，那么接下来我们就需要开发一个可以返回评论数据的接口。

2.2.1. 设计结构的特性

> 从使用者的角度考虑每一个所需功能，反推出来对接口的要求，然后具体实现每个要求，这就是所谓的逆向工程。

对于评论管理页面，我们的需求是：

1. 可以分页查看评论数据列表（作者、评论、文章标题、评论时间、评论状态）
2. 可以通过分页导航访问特定分页的数据，
3. 可以通过操作按钮批准、拒绝、删除每一条评论

根据需求得知，这个功能开发过程中需要三个的接口（endpoint），我们创建三个 php 文件：

1. comment-list.php: 分页加载评论数据
2. comment-delete.php: 删除评论
3. comment-status.php: 改变评论状态

> 名词解释：对于 Web API，我们把每一个接入点称之为 endpoint

2.2.2. comment-list.php

分页查询数据的逻辑可以参考文章管理模块的数据加载

```php
// 处理分页参数
// ========================================

// 页码
$page = isset($_GET['p']) && is_numeric($_GET['p']) ? intval($_GET['p']) : 1;

// 检查页码最小值
if ($page <= 0) {
  header('Location: /admin/comment-list.php?p=1');
  exit;
}

// 页大小
$size = isset($_GET['s']) && is_numeric($_GET['s']) ? intval($_GET['s']) : 20;

// 查询总条数
$total_count = intval(xiu_query('select count(1) from comments
inner join posts on comments.post_id = posts.id')[0][0]);

// 计算总页数
$total_pages = ceil($total_count / $size);

// 检查页码最大值
if ($page > $total_pages) {
  // 跳转到最后一页
  header('Location: /admin/comment-list.php?p=' . $total_pages);
  exit;
}

// 查询数据
// ========================================

// 分页查询评论数据
$comments = xiu_query(sprintf('select
  comments.*, posts.title as post_title
from comments
inner join posts on comments.post_id = posts.id
order by comments.created desc
limit %d, %d', ($page - 1) * $size, $size));

// 响应 JSON
// ========================================

// 设置响应类型为 JSON
header('Content-Type: application/json');

// 输出 JSON
echo json_encode(array(
```

```
  'success' => true,
  'data' => $comments,
  'total_count' => $total_count
));
```

> 源代码: step-59

2.2.3. comment-delete.php

参考分类删除或者文章删除。

```
// 设置响应类型为 JSON
header('Content-Type: application/json');

if (empty($_GET['id'])) {
  // 缺少必要参数
  exit(json_encode(array(
    'success' => false,
    'message' => '缺少必要参数'
  )));
}

// 拼接 SQL 并执行
$affected_rows = xiu_execute(sprintf('delete from comments where id in (%s)', $_GET['id']));

// 输出结果
echo json_encode(array(
  'success' => $affected_rows > 0
));
```

> 源代码: step-60

2.2.4. comment-status.php

```php
// 设置响应类型为 JSON
header('Content-Type: application/json');

// 不是说 POST 方式就不能使用 GET 传参数，不要固化思维
if (empty($_GET['id']) || empty($_POST['status'])) {
  // 缺少必要参数
  exit(json_encode(array(
    'success' => false,
    'message' => '缺少必要参数'
  )));
}

// 拼接 SQL 并执行
$affected_rows = xiu_execute(sprintf("update comments set status = '%s' where id in (%s)", $_POST['status'], $_GET['id']));

// 输出结果
echo json_encode(array(
  'success' => $affected_rows > 0
));
```

> 源代码: step-61

有了接口过后，我们就可以通过在页面中执行 AJAX 操作调用这些接口，实现相对应的功能了。

2.3. 评论数据加载

页面加载完成过后，发送异步请求获取评论数据

```js
$.get('/admin/comment-list.php', { p: 1, s: 30 }, function (res) {
  console.log(res)
  // => { success: true, data: [ ... ], total_count: 100 }
})
```

将数据渲染（客户端渲染）到表格中：

```
var $alert = $('.alert')
var $tbody = $('tbody')

// 页面加载完成过后，发送异步请求获取评论数据
$.get('/admin/comment-list.php', { p: 1, s: 30 }, function (res) {
  console.log(res)
  // => { success: true, data: [ ... ], total_count: 100 }
  if (!res.success) {
    // 加载失败 提示消息 并结束运行
    return $alert.text(res.message)
  }

  // 将数据渲染到表格中
  $(res.data).each(function (i, item) {
    // 每一个数据对应一个 tr
    $tbody.append('<tr class="' + '' + '">' +
    '  <td class="text-center"><input type="checkbox"></td>' +
    '  <td>' + item.author + '</td>' +
    '  <td>' + item.content + '</td>' +
    '  <td>《' + item.post_title + '》</td>' +
    '  <td>' + item.created + '</td>' +
    '  <td>' + item.status + '</td>' +
    '  <td class="text-center">' +
    '    <a href="javascript:;" class="btn btn-info btn-xs">批准</a>' +
    '    <a href="javascript:;" class="btn btn-danger btn-xs">删除</a>' +
    '  </td>' +
    '</tr>')
  })
})
```

> 源代码: step-62

不用往下写了，一旦涉及到这种数据加载渲染的问题，就会涉及到大量的字符串拼接问题，费劲还容易错，总之很不爽。

之前在服务端渲染数据的时候，没有太多这种感觉，而现在到了客户端渲染就十分恶心，根本原因是因为我们的方法过于原始，对于简单的数据渲染还是可以接受的，但是一旦数据复杂了，结构复杂了，过程就十分恶心，而后端使用的实际上是一种“套模板”过程。

脑子稍微灵光一点的同学就应该想得明白：作为一个天天骑自行车上下班的人看着旁边的人都骑摩托车，久而久之这个骑自行车的也会换成摩托车。

当然这里只是打个比方，可以体会我想表达的意思：作为一个积极的人，在明知道有更好方案的情况下是不会一直使用自己以前的老旧方案的，这是社会进步的核心，也是技术进步的核心。

前端也有模板引擎，而且从使用上来说，更多更好更方便。

> 换而言之，模板引擎的本质其实就是各种恶心的字符串操作。 而这里我为什么扯了这么多，主要目的是希望**你**能够有所**体会**，有所**感悟**，不要荒废了你的**思考**能力。

这里我们借助一个非常轻量的模板引擎 jsrender 解决以上问题，模板引擎的使用套路也都类似：

1. 引入模板引擎库文件
2. 准备一个模板
3. 准备一个需要渲染的数据
4. 调用一个模板引擎库提供的方法，把数据通过模板渲染成 HTML

载入模板引擎库：

```
<script src="/static/assets/vendors/jsrender/jsrender.js"></script>
```

准备模板：

```
<script id="comment_tmpl" type="text/x-jsrender">
  <tr class="danger">
    <td class="text-center"><input type="checkbox"></td>
    <td>大大</td>
    <td>楼主好人，顶一个</td>
    <td>《Hello world》</td>
    <td>2016/10/07</td>
    <td>未批准</td>
    <td class="text-center">
      <a href="javascript:;" class="btn btn-info btn-xs">批准</a>
      <a href="javascript:;" class="btn btn-danger btn-xs">删除</a>
    </td>
  </tr>
</script>
```

> 提问：一般模板引擎都要求把模板定义在 `<script>` 标签中，为什么？

调用模板方法：

```
var html = $('#comment_tmpl').render(data)
// html => 渲染后的结果
// 设置到页面中
$tbody.html(html)
```

借助模板语法输出变量：

```
<script id="comment_tmpl" type="text/x-jsrender">
  {{if success}}
  {{for data}}
  <tr class="{{: status === 'held' ? 'warning' : status === 'rejected' ? 'danger' : '' }}">
    <td class="text-center"><input type="checkbox" data-id="{{: id }}"></td>
    <td>{{: author }}</td>
    <td>{{: content }}</td>
    <td>《{{: post_title }}》</td>
    <td>{{: created}}</td>
    <td>{{: status === 'held' ? '待审' : status === 'rejected' ? '拒绝' : '准许' }}</td>
    <td class="text-center">
      {{if status ===  'held'}}
      <button class="btn btn-info btn-xs" data-id="{{: id }}">批准</button>
      <button class="btn btn-warning btn-xs" data-id="{{: id }}">拒绝</button>
      {{/if}}
      <button class="btn btn-danger btn-xs" data-id="{{: id }}">删除</button>
    </td>
  </tr>
  {{/for}}
  {{else}}
  <tr>
    <td colspan="7">{{: message }}</td>
  </tr>
  {{/if}}
</script>
```

> 源代码: step-63

2.4. 分页组件展示

我们之前写的生成分页 HTML 是在服务端渲染分页组件，只能作用在服务端渲染数据的情况下。但是当下的情况我们采用的是客户端渲染的方案，自然就用不了之前的代码了，但是思想是相通的，我们仍然可以按照之前的套路来实现，只不过是在客户端，借助于 JavaScript 实现。

> 作业： 实现一个 JavaScript 版本的分页组件

这里我们就不自己再写了，前端行业最大的特点就是轮子多，实际开发过程中我们多是使用已有的轮子。

> 思考： 为什么不要造轮子，造轮子有什么优点，又有什么缺点？

这里使用的是 twbs-pagination，使用方法就不在这里赘述了。

```
// 页面加载完成过后，发送异步请求获取评论数据
$.get('/admin/comment-list.php', { p: 1, s: size }, function (res) {
  // 通过模板引擎渲染数据
  var html = $tmpl.render(res)

  // 设置到页面中
  $tbody.html(html)

  // 分页组件
  $pagination.twbsPagination({
    totalPages: Math.ceil(res.total_count / size),
    onPageClick: function (event, page) {
      // 页码发生变化时执行
      console.log(page)
    }
  })
})
```

> 源代码: step-64

我们后续都会在 `onPageClick` 加载指定页的数据：

```
// 页面加载完成过后，发送异步请求获取评论数据
$.get('/admin/comment-list.php', { p: 1, s: size }, function (res) {
  // 通过模板引擎渲染数据
  var html = $tmpl.render(res)

  // 设置到页面中
  $tbody.html(html)

  // 分页组件
  $pagination.twbsPagination({
    initiateStartPageClick: false, // 否则 onPageClick 第一次就会触发
    totalPages: Math.ceil(res.total_count / size),
    onPageClick: function (e, page) {
      $.get('/admin/comment-list.php', { p: page, s: size }, function (res) {
        // 通过模板引擎渲染数据
        var html = $tmpl.render(res)
        // 设置到页面中
        $tbody.html(html)
      })
    }
  })
})
```

> 源代码: step-65

2.5. 删除评论

如果是采用同步的方式，则与文章或分类管理的删除相同，但是此处我们的方案是采用 AJAX 方式。

万变不离其宗，想要删除掉评论，客户端肯定是做不到的，因为数据在服务端。可以通过客户端发送一个请求（信号）到服务端，服务端执行删除操作，服务端业务已经实现，现在的问题就是客户端发请求的问题。

2.5.1. 给删除按钮绑定点击事件

常规思路：

1. 为删除按钮添加一个 `btn-delete` 的 class
2. 为所有 `btn-delete` 注册点击事件

```
$('.btn-delete').on('click', function () {
  console.log('btn delete clicked')
})
```

但是经过测试发现，在点击删除按钮后控制台不会输出任何内容，也就是说按钮的点击事件没有执行。

> 提问：为什么按钮的点击事件不会执行

问题的答案也非常简单：当执行注册事件代码时，表格中的数据还没有初始化完成，那么通过 `$('.btn-delete')` 就不会选择到后来界面上的删除按钮元素，自然也就没办法注册点击事件了。

解决办法：

1. 控制注册代码的执行时机；
2. 另外一种事件方式：委托事件；

> http://www.jquery123.com/on/#direct-and-delegated-events

```
// 删除评论
$tbody.on('click', '.btn-delete', function () {
  console.log('btn delete clicked')
})
```

> 源代码: step-66

2.5.2. 发送删除评论的异步请求

点击事件执行 -> 发送异步请求 -> 移除当前点击按钮所属行

```
$tbody.on('click', '.btn-delete', function () {
  var $tr = $(this).parent().parent()
  var id = parseInt($tr.data('id'))
  $.get('/admin/comment-delete.php', { id: id }, function (res) {
    res.success && $tr.remove()
  })
})
```

> 源代码: step-67

个人认为删除成功过后，不应该单单从界面上的表格中移除当前行，而是重新加载当前页数据。

我们重新调整一下代码：

```
var $alert = $('.alert')
var $tbody = $('tbody')
var $tmpl = $('#comment_tmpl')
var $pagination = $('.pagination')

// 页大小
var size = 30
// 当前页码
var currentPage = 1

/**
 * 加载指定页数据
 */
function loadData () {
  $.get('/admin/comment-list.php', { p: currentPage, s: size }, function (res) {
    // 通过模板引擎渲染数据
    var html = $tmpl.render(res)
    // 设置到页面中
    $tbody.html(html)
  })
}

// 页面加载完成过后，发送异步请求获取评论数据
$.get('/admin/comment-list.php', { p: 1, s: size }, function (res) {
  console.log(res)
  // => { success: true, data: [ ... ], total_count: 100 }

  // 通过模板引擎渲染数据
  var html = $tmpl.render(res)

  // 设置到页面中
  $tbody.html(html)

  // 分页组件
  $pagination.twbsPagination({
    initiateStartPageClick: false, // 否则 onPageClick 第一次就会触发
    totalPages: Math.ceil(res.total_count / size),
    onPageClick: function (e, page) {
      currentPage = page
      loadData()
    }
  })
})

// 删除评论
$tbody.on('click', '.btn-delete', function () {
  var $tr = $(this).parent().parent()
  var id = parseInt($tr.data('id'))
```

```
  $.get('/admin/comment-delete.php', { id: id }, function (res) {
    res.success && loadData()
  })
})
```

> 源代码: step-68

2.6. 修改评论状态

```
$tbody.on('click', '.btn-edit', function () {
  var id = parseInt($(this).parent().parent().data('id'))
  var status = $(this).data('status')
  $.post('/admin/comment-status.php?id=' + id, { status: status }, function (res) {
    res.success && loadData()
  })
})
```

> 源代码: step-69

2.7. 批量操作

2.7.1. 批量操作显示

当选中了一个或一个以上的行时，显示批量操作按钮：

```
var $btnBatch = $('.btn-batch')

// 选中项集合
var checkedItems = []

// 批量操作按钮
$tbody.on('change', 'td > input[type=checkbox]', function () {
  var id = parseInt($(this).parent().parent().data('id'))
  if ($(this).prop('checked')) {
    checkedItems.push(id)
  } else {
    checkedItems.splice(checkedItems.indexOf(id), 1)
  }
  checkedItems.length ? $btnBatch.fadeIn() : $btnBatch.fadeOut()

})
```

> 源代码: step-70

2.7.2. 全选 / 全不选

点击表头中的复选框，切换表格中全部数据选中状态

```
// 全选 / 全不选
$('th > input[type=checkbox]').on('change', function () {
  var checked = $(this).prop('checked')
  $('td > input[type=checkbox]').prop('checked', checked).trigger('change')
})
```

> 源代码: step-71

2.7.3. 批量操作按钮点击

点击不同按钮，执行不同请求

```
// 批量操作
$btnBatch
  // 批准
  .on('click', '.btn-info', function (e) {
    $.post('/admin/comment-status.php?id=' + checkedItems.join(','), { status: 'approved' }, function (res) {
      res.success && loadData()
    })
  })
  // 拒绝
  .on('click', '.btn-warning', function (e) {
    $.post('/admin/comment-status.php?id=' + checkedItems.join(','), { status: 'rejected' }, function (res) {
      res.success && loadData()
    })
  })
  // 删除
  .on('click', '.btn-danger', function (e) {
    $.get('/admin/comment-delete.php', { id: checkedItems.join(',') }, function (res) {
      res.success && loadData()
    })
  })
```

> 源代码: step-72

> 作业： 解决刷新过后继续加载指定页码下的数据

> 源代码: step-73