

管理后台文章管理

1. 展示全部文章数据列表

1.1. 查询文章数据

```
1 select * from posts;
```

在文章列表页通过执行以上 SQL 语句获取全部文章数据：

```
1 // 查询全部文章数据
2 $posts = xiu_query('select * from posts');
```

1.2. 基本的文章数据绑定

在表格中将多余的 `<tr>` 标记移除，通过 `foreach` 遍历查询到的 `$posts` 变量，绑定数据：

名词解释：**数据绑定**是指将一个有结构的数据输出到特定结构的 HTML 上。

```
1 <?php foreach ($posts as $item) { ?>
2 <tr>
3     <td class="text-center"><input type="checkbox"></td>
4     <td><?php echo $item['title']; ?></td>
5     <td><?php echo $item['user_id']; ?></td>
6     <td><?php echo $item['category_id']; ?></td>
7     <td><?php echo $item['created']; ?></td>
8     <td><?php echo $item['status']; ?></td>
9     <td class="text-center">
10         <a href="post-add.php" class="btn btn-default btn-xs">编辑</a>
11         <a href="javascript:;" class="btn btn-danger btn-xs">删除</a>
12     </td>
13 </tr>
14 <?php } ?>
```

2. 数据过滤输出

Sometimes, 我们希望在界面上看到的数据并不是数据库中原始的数据，而是经过一个特定的转换逻辑转换过后的结果。这种情况经常发生在输出时间、状态和关联数据时。

2.1. 文章状态友好展示

一般情况下，我们在数据库存储标识都用英文或数字方式存储，但是在界面上应该展示成中文方式，所以我们需要在输出的时候做一次转换，转换方式就是定义一个转换函数：

```

1  /**
2   * 将英文状态描述转换为中文
3   * @param string $status 英文状态
4   * @return string      中文状态
5   */
6  function convert_status ($status) {
7      switch ($status) {
8          case 'drafted':
9              return '草稿';
10         case 'published':
11             return '已发布';
12         case 'trashed':
13             return '回收站';
14         default:
15             return '未知';
16     }
17 }
```

在输出时调用这个函数：

```

1  <td class="text-center"><?php echo convert_status($item['status']); ?></td>
```

2.2. 日期格式化展示

如果需要自定义发布时间的展示格式，可以通过 `date()` 函数完成，而 `date()` 函数所需的参数除了控制输出格式的 `format` 以外，还需要一个整数类型的 `timestamp`。

<http://php.net/manual/zh/function.date.php>

我们已有的是一个类似于 `2012-12-12 12:00:00` 具有标准时间格式的字符串，必须转换成整数类型的 `timestamp`，才能调用 `date()` 函数。我们可以借助 `strtotime()` 函数完成时间字符串到时间戳（`timestamp`）的转换：

<http://php.net/manual/zh/function.strtotime.php>

- 在使用 `strtotime()` 之前，确保通过 `date_default_timezone_set()` 设置默认时区

于是乎，定义一个格式化日期的转换函数：

```
1  /**
2   * 格式化日期
3   * @param string $created 时间字符串
4   * @return string      格式化后的时间字符串
5   */
6  function format_date ($created) {
7      // 设置默认时区!!!
8      date_default_timezone_set('UTC');
9
10     // 转换为时间戳
11     $timestamp = strtotime($created);
12
13     // 格式化并返回 由于 r 是特殊字符，所以需要 \r 转义一下
14     return date('Y年m月d日 <b\r> H:i:s', $timestamp);
15 }
```

在输出的位置调用此函数：

```
1  <td class="text-center"><?php echo format_date($item['created']); ?></td>
```

▶ 源代码: step-20

2.3. 关联数据查询展示

2.3.1. 分类信息展示

我们在文章表中存储的分类信息就只是分类的 ID，直接输出到表格中，并不友好，所以我们需要在输出分类信息时再次根据分类 ID 查询对应文章的分类信息，将查询到的结果输出到 HTML 中。

定义一个根据分类 ID 获取分类信息的函数：

```
1  /**
2   * 根据 ID 获取分类信息
3   * @param integer $id 分类 ID
4   * @return array      分类信息关联数组
5   */
6  function get_category ($id) {
7      $sql = sprintf('select * from categories where id = %d', $id);
8      return xiu_query($sql)[0];
9  }
```

在输出分类名称的位置通过调用该函数输出分类名称：

```
1  <td><?php echo get_category($item['category_id'])['name']; ?></td>
```

2.3.2. 作者信息展示

同上所述，按照相同的方式查询文章的作者信息并输出：

```
1  /**
2   * 根据 ID 获取用户信息
3   * @param integer $id 用户 ID
4   * @return array      用户信息关联数组
5   */
6  function get_author ($id) {
7      $sql = sprintf('select * from users where id = %d', $id);
8      return xiu_query($sql)[0];
9  }
```

在输出作者的位置通过调用该函数输出作者昵称：

```
1  <td><?php echo get_author($item['user_id'])['nickname']; ?></td>
```

► 源代码: step-21

2.4. 联合查询，一步到位

按照以上的方式，可以正常输出分类和作者信息，但是过程中需要有大量的数据库连接和查询操作。

在实际开发过程中，一般不这么做，通常我们会使用联合查询的方式，同时把我们需要的信息查询出来：

```
1 select *
2 from posts
3 inner join users on posts.user_id = users.id
4 inner join categories on posts.category_id = categories.id;
```

以上这条语句可以把 `posts`、`users`、`categories` 三张表同时查询出来（查询到一个结果集中）。

所以我们可以移除 `get_category` 和 `get_author` 两个函数，将查询语句改为上面定义的内容，完成一次性查询。

▶ 源代码: step-22

这样查询也有一些小问题：如果这几个表中有相同名称的字段，在查询过后转换为关联数组就会有问题（关联数组的键是不能重复的），所以我们需要指定需要查询的字段，同时还可以给每一个字段起一个别名，避免冲突：

```
1 select
2     posts.id,
3     posts.title,
4     posts.created,
5     posts.status,
6     categories.name as category_name,
7     users.nickname as author_name
8 from posts
9 inner join users on posts.user_id = users.id
10 inner join categories on posts.category_id = categories.id;
```

从另外一个角度来说：指定了具体的查询字段，也会提高数据库检索的速度。

▶ 源代码: step-23

MAKE IT BETTER

3. 分页加载文章数据

导入更多的测试数据

[测试数据导入脚本](#)

3.1. 查询一部分数据

当数据过多过后，如果还是按照以上操作每次查询全部数据，页面就显得十分臃肿，加载起来也非常慢，所以必须要通过分页加载的方式改善（每次只显示一部分数据）。

操作方式也非常简单，就是在原有 SQL 语句的基础之上加上 `limit` 和 `order by` 子句：

```
1 select
2   posts.id,
3   posts.title,
4   posts.created,
5   posts.status,
6   categories.name as category_name,
7   users.nickname as author_name
8 from posts
9 inner join users on posts.user_id = users.id
10 inner join categories on posts.category_id = categories.id
11 order by posts.created desc
12 limit 0, 10
```

limit 用法：limit [offset,]rows

- `limit 10` -- 只取前 10 条数据
- `limit 5, 10` -- 从第 5 条之后，第 6 条开始，向后取 10 条数据

3.2. 分页参数计算

`limit` 子句中的 `0` 和 `10` 不是一成不变的，应该跟着页码的变化而变化，具体的规则就是：

- 第 1 页 `limit 0, 10`
- 第 2 页 `limit 10, 10`
- 第 3 页 `limit 20, 10`
- 第 4 页 `limit 30, 10`
- ...

根据以上规则得出公式：`offset = (page - 1) * size`

```

1 // 处理分页
2 // =====
3
4 $size = 10;
5 $page = 2;
6
7 // 查询数据
8 // =====
9
10 // 查询全部文章数据
11 $posts = xiu_query(sprintf('select
12     posts.id,
13     posts.title,
14     posts.created,
15     posts.status,
16     categories.name as category_name,
17     users.nickname as author_name
18 from posts
19 inner join users on posts.user_id = users.id
20 inner join categories on posts.category_id = categories.id
21 order by posts.created desc
22 limit %d, %d', ($page - 1) * $size, $size));

```

▶ 源代码: step-24

3.3. 获取当前页码

一般分页都是通过 URL 传递一个页码参数（通常使用 `querystring`）

也就是说，我们应该在页面开始执行的时候获取这个 URL 参数：

```

1 // 处理分页
2 // =====
3
4 // 定义每页显示数据量（一般把这一项定义到配置文件中）
5 $size = 10;
6
7 // 获取分页参数 没有或传过来的不是数字的话默认为 1
8 $page = isset($_GET['p']) && is_numeric($_GET['p']) ? intval($_GET['p']) : 1;
9
10 if ($page <= 0) {
11     // 页码小于 1 没有任何意义，则跳转到第一页
12     header('Location: /admin/posts.php?p=1');
13     exit;
14 }

```

▶ 源代码: step-25

3.4. 展示分页跳转链接

用户在使用分页功能时不可能通过地址栏改变要访问的页码，必须通过可视化的链接点击访问，所以我们需要根据数据的情况在界面上显示分页链接。



目标效果演示：<http://esimakin.github.io/twbs-pagination/>

组合一个分页跳转链接的必要条件：

- 一共有多少页面（或者一共有多少条数据、每页显示多少条）
- 当前显示的时第几页
- 一共要在界面上显示多少个分页跳转链接，一般为单数个，如：3、5、7

以上必要条件中只有第一条需要额外处理，其余的目前都可以拿到，所以重点攻克第一条：

3.4.1. 获取总页数

一共有多少页面取决于一共有多少条数据和每一页显示多少条，计算公式为：`$total_pages = ceil($total_count / $size)`。

通过查询数据库可以得到总条数：

```
1 // 查询总条数
2 $total_count = intval(xiu_query('select count(1)
3 from posts
4 inner join users on posts.user_id = users.id
5 inner join categories on posts.category_id = categories.id')[0][0]);
6
7 // 计算总页数
8 $total_pages = ceil($total_count / $size);
```

思考：为什么要在查询条数的时候也用联合查询

知道了总页数，就可以对 URL 中传递过来的分页参数做范围校验了（`$page <= $total_pages`）

```
1 if ($page > $total_pages) {
2     // 超出范围，则跳转到最后一页
3     header('Location: /admin/posts.php?p=' . $total_pages);
4     exit;
5 }
```

▶ 源代码: step-26

3.4.2. 循环输出分页链接

```
1 <ul class="pagination pagination-sm pull-right">
2     <?php if ($page - 1 > 0) : ?>
3     <li><a href="?p=<?php echo $page - 1; ?>">上一页</a></li>
4     <?php endif; ?>
5     <?php for ($i = 1; $i <= $total_pages; $i++) : ?>
6     <li><?php echo $i === $page ? ' class="active"' : '' ?><a href="?p=<?php echo $i; ?>"><?php
7     echo $i; ?></a></li>
8     <?php endfor; ?>
9     <?php if ($page + 1 <= $total_pages) : ?>
10    <li><a href="?p=<?php echo $page + 1; ?>">下一页</a></li>
11    <?php endif; ?>
12 </ul>
```

▶ 源代码: step-27

由于分页功能在不同页面都会被使用到，所以也提取到 `functions.php` 文件中，封装成一个 `xiu_pagination` 函数：

```
1  /**
2   * 输出分页链接
3   * @param integer $page 当前页码
4   * @param integer $total 总页数
5   * @param string $format 链接模板，%d 会被替换为具体页数
6   * @example
7   * <?php xiu_pagination(2, 10, '/list.php?page=%d'); ?>
8   */
9  function xiu_pagination ($page, $total, $format) {
10     // 上一页
11     if ($page - 1 > 0) {
12         printf('<li><a href="%s">上一页</a></li>', sprintf($format, $page - 1));
13     }
14
15     // 数字页码
16     for ($i = 1; $i <= $total; $i++) {
17         $activeClass = $i === $page ? ' class="active"' : '';
18         printf('<li>%s<a href="%s">%d</a></li>', $activeClass, sprintf($format, $i), $i);
19     }
20
21     // 下一页
22     if ($page + 1 <= $total) {
23         printf('<li><a href="%s">下一页</a></li>', sprintf($format, $page + 1));
24     }
25 }
```

📄 源代码: step-28

3.4.3. 控制显示页码个数

按照目前的实现情况，已经可以正常使用分页链接了，但是当总页数过多，显示起来也会有问题，所以需要控制显示页码的个数，一般情况下，我们是根据当前页码在中间，左边和右边各留几位。

实现以上需求的思路：主要就是想办法根据当前页码知道应该从第几页开始显示，到第几页结束，另外需要注意不能超出范围。

以下是具体实现：

```

1  /**
2  * 输出分页链接
3  * @param integer $page    当前页码
4  * @param integer $total    总页数
5  * @param string $format    链接模板, %d 会被替换为具体页数
6  * @param integer $visible  可见页码数量 (可选参数, 默认为 5)
7  * @example
8  *   <?php xiu_pagination(2, 10, '/list.php?page=%d', 5); ?>
9  */
10 function xiu_pagination ($page, $total, $format, $visible = 5) {
11     // 计算起始页码
12     // 当前页左侧应有几个页码数, 如果一共是 5 个, 则左边是 2 个, 右边是两个
13     $left = floor($visible / 2);
14     // 开始页码
15     $begin = $page - $left;
16     // 确保开始不能小于 1
17     $begin = $begin < 1 ? 1 : $begin;
18     // 结束页码
19     $end = $begin + $visible - 1;
20     // 确保结束不能大于最大值 $total
21     $end = $end > $total ? $total : $end;
22     // 如果 $end 变了, $begin 也要跟着一起变
23     $begin = $end - $visible + 1;
24     // 确保开始不能小于 1
25     $begin = $begin < 1 ? 1 : $begin;
26
27     // 上一页
28     if ($page - 1 > 0) {
29         printf('<li><a href="%s">&laquo;</a></li>', sprintf($format, $page - 1));
30     }
31
32     // 省略号
33     if ($begin > 1) {
34         print('<li class="disabled"><span>...</span></li>');
35     }
36
37     // 数字页码
38     for ($i = $begin; $i <= $end; $i++) {
39         // 经过以上的计算 $i 的类型可能是 float 类型, 所以此处用 == 比较合适
40         $activeClass = $i == $page ? ' class="active"' : '';
41         printf('<li><a href="%s">%d</a></li>', $activeClass, sprintf($format, $i), $i);
42     }
43
44     // 省略号
45     if ($end < $total) {
46         print('<li class="disabled"><span>...</span></li>');
47     }
48

```

```
49 // 下一页
50 if ($page + 1 <= $total) {
51     printf('<li><a href="%s">&raquo;</a></li>', sprintf($format, $page + 1));
52 }
53 }
```

▸ 源代码: step-29

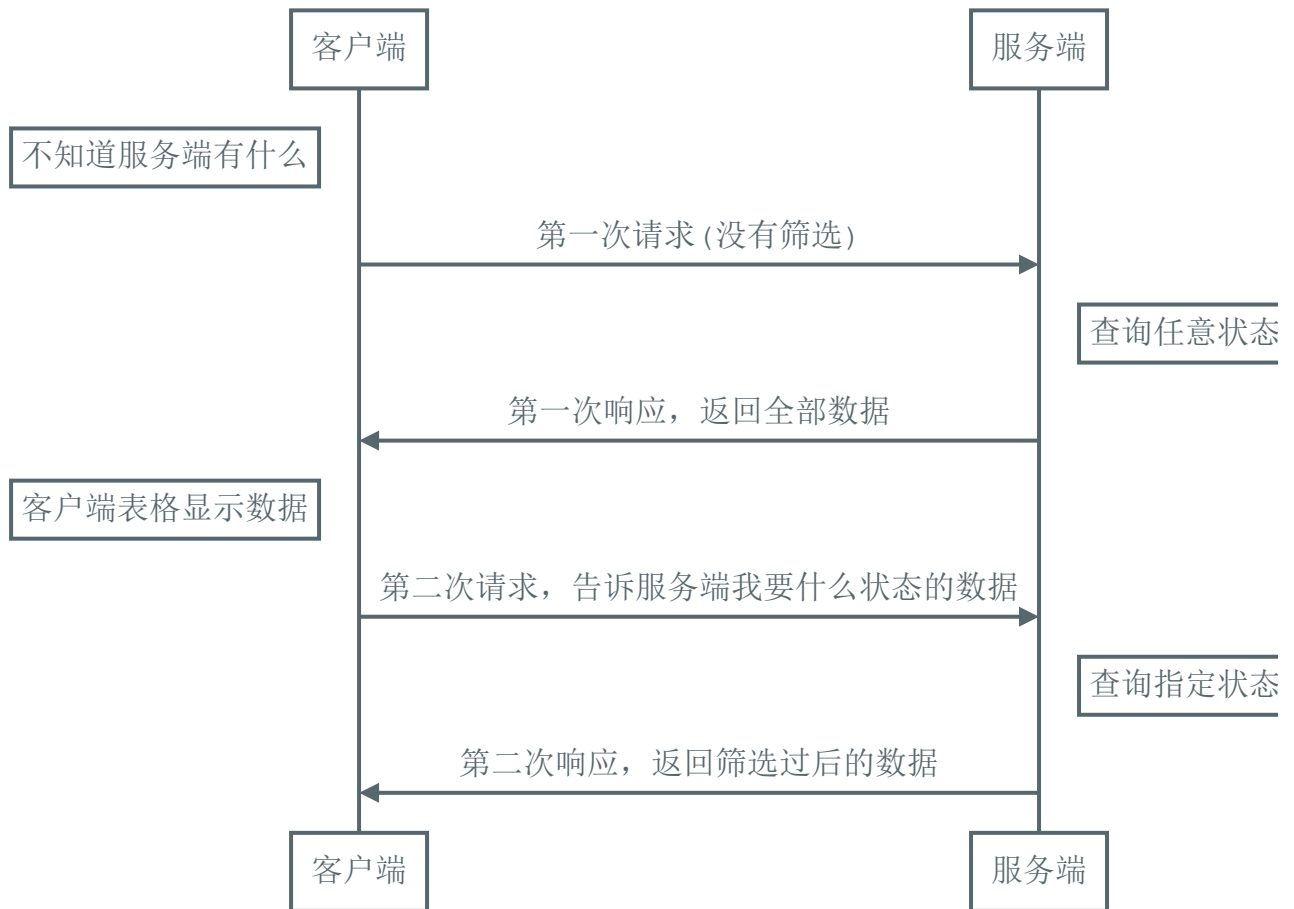
MAKE IT BETTER

4. 数据筛选

4.1. 状态筛选

注意：不要立即联想 AJAX 的问题，AJAX 是后来诞生的，换言之不是必须的，我们这里讨论的只是传统的方式（历史使人明智）

实现思路：用户只有在未筛选的情况下知道一共有哪些状态，当用户选择其中某个状态过后，必须让服务端知道用户选择的**状态是什么，从而返回指定的状态下的数据。



好有一比：去商店买钻石，你不可能直接说来颗 5 斤的，正常情况，你都是先问有没有钻石，售货员拿出一排，顺便告诉你有哪几种重量的，你再告诉售货员你选其中哪一种，售货员再拿出指定种类的让你挑。

所以我们先在页面上添加一个表单（用于接收用户后续的意愿），然后提供一个 `<select>` 列出全部状态，让用户选择。用户选择完了再把表单提交回来，此时服务端就知道你需要什么状态的数据。

注意：永远都不要说历史上的事 low，历史永远是伟大的。

4.1.1. 表单 HTML 处理

不设置表单的 `method` 默认就是 `get`，此处就应该使用 `get`，原因有二：

- 效果上：`get` 提交的参数会体现在 URL 中，方便用户使用（刷新、书签）
- 性质上：这一次请求主观上还是在**拿（获取）**服务端的数据

```

1 <form class="form-inline" action="/admin/posts.php">
2   <select name="s" class="form-control input-sm">
3     <option value="all">所有状态</option>
4     <option value="drafted">草稿</option>
5     <option value="published">已发布</option>
6     <option value="trashed">回收站</option>
7   </select>
8   <button class="btn btn-default btn-sm">筛选</button>
9 </form>

```

4.1.2. 获取提交参数

在查询数据之前，接受参数，组织查询参数：

```

1 // 处理筛选逻辑
2 // =====
3
4 // 数据库查询筛选条件（默认为 1 = 1，相当于没有条件）
5 $where = '1 = 1';
6
7 // 状态筛选
8 if (isset($_GET['s']) && $_GET['s'] != 'all') {
9     $where .= sprintf(" and posts.status = '%s'", $_GET['s']);
10 }
11
12 // $where => " and posts.status = 'drafted'"

```

4.1.3. 添加查询参数

然后在进行查询时添加 `where` 子句：

```

1 // 查询总条数
2 $total_count = intval(xiu_query('select count(1)
3 from posts
4 inner join users on posts.user_id = users.id
5 inner join categories on posts.category_id = categories.id
6 where ' . $where)[0][0]);
7
8 ...
9
10 // 查询全部文章数据
11 $posts = xiu_query(sprintf('select
12     posts.id,
13     posts.title,
14     posts.created,
15     posts.status,
16     categories.name as category_name,
17     users.nickname as author_name
18 from posts
19 inner join users on posts.user_id = users.id
20 inner join categories on posts.category_id = categories.id
21 where %s
22 order by posts.created desc
23 limit %d, %d', $where, ($page - 1) * $size, $size));

```

▶ 源代码: step-30

4.1.4. 记住筛选状态

筛选后，`<select>` 中被选中的 `<option>` 应该在展示的时候默认选中：

```

1 <select name="s" class="form-control input-sm">
2     <option value="all">所有状态</option>
3     <option value="drafted">?php echo isset($_GET['s']) && $_GET['s'] == 'drafted' ? ' selected' :
4     ''; ?>>草稿</option>
5     <option value="published">?php echo isset($_GET['s']) && $_GET['s'] == 'published' ? '
6     selected' : ''; ?>>已发布</option>
7     <option value="trashed">?php echo isset($_GET['s']) && $_GET['s'] == 'trashed' ? ' selected' :
8     ''; ?>>回收站</option>
9 </select>

```

▶ 源代码: step-31

4.2. 分类筛选

 作业：仿照状态筛选功能的实现过程，实现分类筛选功能

 源代码: step-32

4.3. 结合分页


目前来说，单独看筛选功能和分页功能都是没有问题，但是同时使用会有问题：

1. 筛选过后，页数不对（没有遇到，但是常见）。
 - 原因：查询总条数时没有添加筛选条件
2. 筛选过后，点分页链接访问其他页，筛选状态丢失。
 - 原因：分类链接的 URL 中只有页码信息，不包括筛选状态

4.3.1. 分页链接加入筛选参数

只要在涉及到分页链接的地方加上当前的筛选参数即可解决问题，所以我们在接收状态筛选参数时将其记录下来：

```
1 // 记录本次请求的查询参数
2 $query = '';
3
4 // 状态筛选
5 if (isset($_GET['s']) && $_GET['s'] != 'all') {
6     $where .= sprintf(" and posts.status = '%s'", $_GET['s']);
7     $query .= '&s=' . $_GET['s'];
8 }
9
10 // 分类筛选
11 if (isset($_GET['c']) && $_GET['c'] != 'all') {
12     $where .= sprintf(" and posts.category_id = %d", $_GET['c']);
13     $query .= '&c=' . $_GET['c'];
14 }
15
16 // $query => "&s=drafted&c=2"
```

 源代码: step-33