

2026 CakePHP-Day3

CakePHP CMS 教學1

修改 PHP 時區

- 在 Dockerfile 中加入以下程式碼：

```
RUN echo "date.timezone=Asia/Taipei" > /usr/local/etc/php/conf.d/timezone.ini
```

內容管理系統(CMS)

- 本教學將引導您建立一個簡單的 CMS 應用程式。
- 首先，我們將安裝 CakePHP、建立資料庫並建立簡單的文章管理。
- 以下是您需要的：
 1. 資料庫伺服器。我們將在本教程中使用 MySQL 伺服器。您需要有足夠的 SQL 知識才能建立資料庫並執行教程中的 SQL 程式碼片段。CakePHP 將負責建立您的應用程式所需的所有查詢。由於我們使用的是 MySQL，因此也要確保在 PHP 中啟用了 pdo_mysql。
 2. 基本的 PHP 知識。

建立第一個模型

- 模型是 CakePHP 應用程式的核心。
- 它們讓我們能夠讀取和修改資料，建立資料之間的關聯，驗證資料，並應用應用程式規則。
- 模型提供了建立控制器動作和模板所需的基礎。
- CakePHP 的模型由 **Table** 和 **Entity** 物件組成。
- **Table** 物件用於訪問特定資料表中存儲的實體集合，它們存放於 `src/Model/Table` 資料夾中。
- 我們將建立的檔案將儲存在 `src/Model/Table/ArticlesTable.php`。完整的檔案內容如下：

```
<?php
// 檔案: src/Model/Table/ArticlesTable.php
declare(strict_types=1);

namespace App\Model\Table;

use Cake\ORM\Table;

class ArticlesTable extends Table
{
    public function initialize(array $config): void
    {
        parent::initialize($config);
        $this->addBehavior('Timestamp');
    }
}
```

ArticlesTable.php

建立第一個模型

- 我們為模型附加了 **Timestamp** 行為，該行為會自動填充資料表中的 **created** 和 **modified** 欄位。
- 透過將 **Table** 物件命名為 **ArticlesTable**，CakePHP 能根據命名慣例判斷此模型對應 **articles** 資料表。
- CakePHP 也會根據慣例假設 **id** 欄位是資料表的主鍵。



如果在 `src/Model/Table` 中找不到相應的檔案，CakePHP 會動態生成模型物件。但這也意味著如果你的檔案命名錯誤（例如：`articlestable.php` 或 `ArticleTable.php`），CakePHP 將無法識別你的設定，並改用動態生成的模型。

```
<?php
// 檔案: src/Model/Table/ArticlesTable.php
declare(strict_types=1);

namespace App\Model\Table;

use Cake\ORM\Table;

class ArticlesTable extends Table
{
    public function initialize(array $config): void
    {
        parent::initialize($config);
        $this->addBehavior('Timestamp');
    }
}
```

ArticlesTable.php

建立第一個模型

- 接著，我們將為 Articles 建立一個 **Entity** 類別。
- Entity 代表資料庫中的單筆記錄，提供資料的行級別行為。
- 我們的 Entity 將儲存在 src/Model/Entity/Article.php。完整檔案內容如下：
- 目前，我們的 Entity 非常簡單。我們僅設定了 `_accessible` 屬性，該屬性控制哪些屬性可以透過[批次賦值 \(Mass Assignment\)](#)被修改。
- 目前這個模型的功能還很有限。接下來，我們將建立第一個**控制器**和**模板**，以便與模型進行互動。



ArticlesTable 和 Article Entity 類別可以透過終端機指令生成：

```
bin/cake bake model articles
```

```
<?php
// 檔案: src/Model/Entity/Article.php
declare(strict_types=1);

namespace App\Model\Entity;

use Cake\ORM\Entity;

class Article extends Entity
{
    protected array $_accessible = [
        'user_id' => true,
        'title' => true,
        'slug' => true,
        'body' => true,
        'published' => true,
        'created' => true,
        'modified' => true,
        'user' => true,
        'tags' => true,
    ];
}
```

建立文章控制器

- 在建立模型後，我們需要為 `Articles` 新增一個控制器。
- `CakePHP` 的控制器負責處理 `HTTP` 請求並執行模型方法中的業務邏輯，以準備回應。
- 我們會將這個新控制器儲存在 `src/Controller` 資料夾中的一個名為 `ArticlesController.php` 的檔案內。以下是基本控制器的範例：

```
<?php
// 檔案: src/Controller/ArticlesController.php

namespace App\Controller;

class ArticlesController extends AppController
{
}
```

建立文章控制器

- 現在，讓我們在控制器中新增一個動作（action）。
- 動作是控制器的方法，與路由相連。
- 例如，當使用者請求 `www.example.com/articles/index`（或 `www.example.com/articles`）時，CakePHP 會調用 `ArticlesController` 的 `index` 方法。
- 這個方法應該查詢模型層，並透過在 `View` 中渲染模板來準備回應。以下是該動作的程式碼：

```
<?php
// 檔案: src/Controller/ArticlesController.php

namespace App\Controller;

class ArticlesController extends AppController
{
    public function index()
    {
        $articles = $this->paginate($this->Articles);
        $this->set(compact('articles'));
    }
}
```

建立文章控制器

- 透過在 `ArticlesController` 中定義 `function index()`，使用者現在可以透過請求 `www.example.com/articles/index` 訪問該邏輯。
- 同樣地，如果我們定義了一個名為 `foobar()` 的函式，使用者就可以透過請求 `www.example.com/articles/foobar` 訪問它。
- 你可能會想將控制器和動作命名為能夠直接獲得特定 URL 的方式，但請避免這樣做。
- 應遵循 `CakePHP` 的命名慣例，創建可讀且有意義的動作名稱。之後你可以使用路由 (`Routing`) 將你希望的 URL 連接到所創建的動作。
- 我們的控制器動作非常簡單。它使用自動載入的 `Articles` 模型，從資料庫中獲取分頁的文章集合，然後使用 `set()` 方法將文章傳遞給模板 (我們稍後會建立模板)。
- 在控制器動作執行完成後，`CakePHP` 會自動渲染該模板。

```
<?php                                     ArticlesController.php
// 檔案: src/Controller/ArticlesController.php

namespace App\Controller;

class ArticlesController extends AppController
{
    public function index()
    {
        $articles = $this->paginate($this->Articles);
        $this->set(compact('articles'));
    }
}
```

建立文章列表模板

- 現在，我們的控制器已經從模型中提取資料並準備好視圖內容，接下來讓我們為 `index` 動作建立一個視圖模板。
- **CakePHP** 的視圖模板是用來展示內容的 **PHP** 程式碼，會插入到應用程式的版面配置 (`layout`) 中。
- 雖然我們在這裡會建立 **HTML**，但視圖也可以生成 **JSON**、**CSV**，甚至像 **PDF** 這樣的二進位檔案。
- 版面配置 (`layout`) 是包裹在視圖外的展示程式碼，包含網站的常見元素，例如標頭、頁尾和導航欄。
- 應用程式可以有多個版面配置，並在它們之間切換。不過，現在我們只需使用預設的版面配置即可。
- **CakePHP** 的模板檔案存儲在 `templates` 資料夾中，並且會按照它們對應的控制器名稱進行分類。
- 因此，在這個案例中，我們需要建立一個名為 `Articles` 的資料夾，並新增以下程式碼到你的應用程式中：

```
<!-- 檔案: templates/Articles/index.php -->
<h1>文章</h1>
<table>
  <tr>
    <th>標題</th>
    <th>建立時間</th>
  </tr>

  <!-- 在這裡迭代 $articles 查詢物件，輸出文章資訊 -->

  <?php foreach ($articles as $article): ?>
  <tr>
    <td>
      <? = $this->Html->link($article->title,
        ['action' => 'view', $article->slug]) ?>
    </td>
    <td>
      <? = $article->created
        ->i18nFormat('yyyy年MM月dd日 EEEE HH:mm') ?>
    </td>
  </tr>
  <?php endforeach; ?>
</table>
```

index.php

建立文章列表模板

- 在上一節中，我們使用 `set()` 將 `articles` 變數傳遞到視圖中。傳遞到視圖的變數可以作為區域變數使用，我們在上述程式碼中已經使用了這些變數。
- 你可能注意到程式碼中有一個名為 `$this->Html` 的物件。這是一個 CakePHP 的 `HtmlHelper` 實例。CakePHP 附帶了一組視圖輔助類別 (`helpers`)，它們可以簡化建立連結、表單和分頁按鈕等任務。在這裡，重要的是 `link()` 方法會生成一個包含指定連結文字 (第一個參數) 和 URL (第二個參數) 的 HTML 連結。
- 在 CakePHP 中指定 URL 時，建議使用陣列或命名路由的語法。這些語法可以利用 CakePHP 提供的反向路由功能。
- 到這一步，你應該可以在瀏覽器中訪問 `http://localhost:8080/CMS/articles/index`。你將會看到一個包含文章標題和表格列表的視圖，格式已正確顯示出來。

```
<!-- 檔案: templates/Articles/index.php -->
<h1>文章</h1>
<table>
  <tr>
    <th>標題</th>
    <th>建立時間</th>
  </tr>

  <!-- 在這裡迭代 $articles 查詢物件，輸出文章資訊 -->

  <?php foreach ($articles as $article): ?>
  <tr>
    <td>
      <?= $this->Html->link($article->title,
        ['action' => 'view', $article->slug]) ?>
    </td>
    <td>
      <?= $article->created
        ->i18nFormat('yyyy年MM月dd日 EEEE HH:mm') ?>
    </td>
  </tr>
  <?php endforeach; ?>
</table>
```

index.php

建立 View 動作

- 如果你點擊文章列表頁面中的其中一個「view」連結，會看到一個錯誤頁面，提示該動作尚未實作。現在讓我們來修正這個問題：

// 加入到現有的 src/Controller/ArticlesController.php 檔案

ArticlesController.php

```
public function view(?string $slug = null)
{
    $article = $this->Articles->findBySlug($slug)->firstOrFail();
    $this->set(compact('article'));
}
```

建立 View 動作

- 雖然這是一個簡單的動作，但我們使用了一些強大的 CakePHP 功能。首先，我們在動作中使用了 `findBySlug()`，這是一個動態查詢方法 (Dynamic Finder)。該方法允許我們根據給定的 `slug` 建立一個基礎查詢來查找文章。接著，我們使用 `firstOrFail()` 來取得第一筆記錄，或者在找不到記錄時拋出 `\Cake\Datasource\Exception\RecordNotFoundException`。
- 我們的動作接收了一個 `$slug` 參數，但這個參數是從哪裡來的呢？當用戶請求 `/articles/view/first-post` 時，CakePHP 的路由和分派層會將值 `first-post` 傳遞給 `$slug`。
- 如果我們儲存新的動作並重新加載瀏覽器，會看到另一個 CakePHP 錯誤頁面，提示我們缺少視圖模板。接下來，讓我們修正這個問題。

```
// 加入到現有的 src/Controller/ArticlesController.php 檔案

public function view(?string $slug = null)
{
    $article = $this->Articles->findBySlug($slug)->firstOrFail();
    $this->set(compact('article'));
}
```

ArticlesController.php

建立 View 模板

- 接下來，我們為新的 `view` 動作建立一個視圖模板，並將其放置在 `templates/Articles/view.php` 中：

```
<!-- 檔案: templates/Articles/view.php -->
```

`view.php`

```
<h1><?= h($article->title) ?></h1>
<p><?= h($article->body) ?></p>
<p><small>創建於: <?= $article->created->i18nFormat('yyyy年MM月dd日 EEEE HH:mm') ?></small></p>
<p><?= $this->Html->link('編輯', ['action' => 'edit', $article->slug]) ?></p>
```

- 現在，你可以通過訪問 `/articles/index` 中的連結或手動請求文章，例如 `/articles/view/first-post`，來驗證這是否正常運作。

新增文章



你需要在任何使用 Flash 的控制器中包含 Flash 元件。通常，將它加入到 `AppController` 中是個不錯的選擇，而這已經在本教學中完成了。

- 在完成基本的讀取視圖後，我們需要讓使用者能夠新增文章。首先，在 `ArticlesController` 中建立一個 `add()` 動作。以下是更新後的控制器內容：

// 加入到現有的 `src/Controller/ArticlesController.php` 檔案

`ArticlesController.php`

```
public function add()
{
    $article = $this->Articles->newEmptyEntity();
    if ($this->request->is('post')) {
        $article = $this->Articles->patchEntity($article, $this->request->getData());

        // 暫時寫死 user_id，後續我們將建構驗證功能來取代此設定。
        $article->user_id = 1;

        if ($this->Articles->save($article)) {
            $this->Flash->success(__('您的文章已經儲存。'));

            return $this->redirect(['action' => 'index']);
        }
        $this->Flash->error(__('無法新增您的文章。'));
    }
    $this->set('article', $article);
}
```

`add()` 動作的功能

1. 如果收到的請求是 HTTP POST，系統會嘗試使用 `Articles` 模型儲存資料。
2. 如果儲存失敗，則直接渲染視圖，讓我們有機會向使用者顯示驗證錯誤或其他警告。

新增文章

- 每個 CakePHP 請求都會包含一個可透過 `$this->request` 存取的請求物件。
- 此物件包含剛收到的請求相關資訊。我們使用 [Cake\Http\ServerRequest::is\(\)](#) 方法檢查請求是否為 HTTP POST。
- POST 資料可透過 `$this->request->getData()` 取得。
- 如果想查看其內容，可以使用 `pr()` 或 `debug()` 函數來列印。
- 為了儲存資料，我們首先將 POST 資料“編組”為一個 `Article` 實體，接著使用我們之前建立的 `ArticlesTable` 將實體資料儲存下來。
- 儲存成功後，我們使用 `FlashComponent` 的 `success()` 方法，將一條訊息設定到 `session` 中。
- 該訊息會在重新導向後的下一個頁面顯示。在佈局檔案中，我們使用 `<?= $this->Flash->render() ?>` 顯示 Flash 訊息並清除相應的 `session` 變數。
- 最後，儲存完成後，我們使用 [Cake\Controller\Controller::redirect](#) 將使用者導回文章列表頁面。
- 參數 `['action' => 'index']` 會被轉換為 URL `/articles`，即 `ArticlesController` 的 `index` 動作。
- 可以參考 [API](#) 文件中的 `Cake\Routing\Router::url()` 函數，了解指定 URL 的多種格式和用法。

建立新增文章模板

- 以下是我們的新增文章視圖模板：

```
<!-- 檔案: templates/Articles/add.php -->
```

add.php

```
<h1>新增文章</h1>
```

```
<?php
```

```
    echo $this->Form->create($article);
```

```
    // 暫時寫死使用者。
```

```
    echo $this->Form->control('user_id', ['type' => 'hidden', 'value' => 1]);
```

```
    echo $this->Form->control('title', ['label' => '標題']);
```

```
    echo $this->Form->control('body', ['label' => '內文', 'rows' => '3']);
```

```
    echo $this->Form->button(__('儲存文章'));
```

```
    echo $this->Form->end();
```

```
?>
```

- 我們使用 FormHelper 來生成 HTML 表單的開頭標籤。以下是 `$this->Form->create()` 生成的 HTML：

```
<form method="post" action="/articles/add">
```

建立新增文章模板

- 以下是我們的新增文章視圖模板：

```
<!-- 檔案: templates/Articles/add.php -->
```

add.php

```
<h1>新增文章</h1>
```

```
<?php
```

```
    echo $this->Form->create($article);
```

```
    // 暫時寫死使用者。
```

```
    echo $this->Form->control('user_id', ['type' => 'hidden', 'value' => 1]);
```

```
    echo $this->Form->control('title', ['label' => '標題']);
```

```
    echo $this->Form->control('body', ['label' => '內文', 'rows' => '3']);
```

```
    echo $this->Form->button(__('儲存文章'));
```

```
    echo $this->Form->end();
```

```
?>
```

- 由於我們在調用 `create()` 時沒有提供 `URL` 選項，`FormHelper` 假設我們希望表單提交回當前的動作。
- `$this->Form->control()` 方法用於創建對應於模型字段的表單元素。第一個參數指定字段名稱，第二個參數則允許設置多種選項，例如此處指定了文字區域的行數。該方法根據模型字段的類型生成不同的表單元素，並自動生成標籤文字。你可以使用選項自定義標籤、輸入框或表單控制項的其他方面。
- 最後，`$this->Form->end()` 用於關閉表單標籤。

建立新增文章模板

- 現在，讓我們回到 `templates/Articles/index.php` 視圖，為頁面新增一個「新增文章」的連結。在 `<table>` 標籤之前，添加以下內容：

```
<!-- 加入到現有的 templates/Articles/index.php 檔案 -->

<h1>文章</h1>
<?= $this->Html->link('新增文章', ['action' => 'add']) ?>
<table>
    <tr>
        <th>標題</th>
        <th>建立時間</th>
    </tr>

    <!-- 在這裡迭代 $articles 查詢物件，輸出文章資訊 -->
    <!-- 下面還有其他程式碼 -->
```

`index.php`

新增簡單的 slug 生成功能

- 如果我們現在儲存一篇文章，由於沒有建立 slug 屬性且該欄位設置為 NOT NULL，因此儲存將失敗。slug 通常是文章標題的 URL 安全版本。我們可以使用 ORM 的 beforeSave() 回呼來填充 slug 欄位：

```
<?php
// 加入到現有的 src/Model/Table/ArticlesTable.php 檔案
namespace App\Model\Table;

use Cake\ORM\Table;
// 引入 Text 類別
use Cake\Utility\Text;
// 引入 EventInterface 類別
use Cake\Event\EventInterface;

// 新增以下方法

public function beforeSave(EventInterface $event, $entity, $options)
{
    if ($entity->isNew() && !$entity->slug) {
        $sluggedTitle = Text::slug($entity->title);
        // 將 slug 修剪為 schema 中定義的最大長度
        $entity->slug = substr($sluggedTitle, 0, 191);
    }
}
```

ArticlesTable.php

這段程式碼非常簡單，目前尚未考慮重複的 slug 問題，但稍後我們會修正這部分功能。

編輯文章

- 我們的應用程式現在可以儲存文章，但無法編輯它們。讓我們來解決這個問題。在 `ArticlesController` 中新增以下動作：

// 加入到現有的 `src/Controller/ArticlesController.php` 檔案

`ArticlesController.php`

// 新增以下方法

```
public function edit(?string $slug)
{
    $article = $this->Articles
        ->findBySlug($slug)
        ->firstOrFail();

    if ($this->request->is(['post', 'put'])) {
        $this->Articles->patchEntity($article, $this->request->getData());
        if ($this->Articles->save($article)) {
            $this->Flash->success(__('您的文章已經更新。'));

            return $this->redirect(['action' => 'index']);
        }
        $this->Flash->error(__('無法更新您的文章。'));
    }

    $this->set('article', $article);
}
```

`edit($slug)` 功能說明

1. 確認文章是否存在
2. 處理 POST 和 PUT 請求
3. 成功或失敗的回饋
4. 傳遞資料至視圖

編輯文章

1. 確認文章是否存在：

此動作首先確保使用者嘗試存取的文章記錄存在。如果沒有傳入 `$slug` 參數，或者該文章不存在，將會拋出 **RecordNotFoundException**，而 CakePHP 的 **ErrorHandler** 會顯示適當的錯誤頁面。

2. 處理 POST 和 PUT 請求：

動作接著檢查請求是否為 **POST** 或 **PUT** 請求。如果是，則使用 **patchEntity()** 方法來更新文章實體，並嘗試儲存變更。

3. 成功或失敗的回饋：

- 如果儲存成功，顯示成功的訊息並導向回文章列表頁面。
- 如果儲存失敗，顯示錯誤訊息，並讓使用者檢視驗證錯誤。

4. 傳遞資料至視圖：

在最後，將 `$article` 資料傳遞到視圖中，供模板顯示編輯表單。

建立編輯文章模板

- 編輯模板應如下所示：

```
<!-- 檔案: templates/Articles/edit.php -->
```

edit.php

```
<h1>編輯文章</h1>
```

```
<?php
```

```
    echo $this->Form->create($article);
    echo $this->Form->control('user_id', ['type' => 'hidden']);
    echo $this->Form->control('title', ['label' => '標題']);
    echo $this->Form->control('body', ['label' => '內文', 'rows' => '3']);
    echo $this->Form->button(__('儲存文章'));
    echo $this->Form->end();
```

```
?>
```

- 此模板會輸出編輯表單（包含已填入的值），並顯示任何必要的驗證錯誤訊息。

修改視圖

- 現在可以更新 `index` 視圖，為特定文章新增編輯連結：

```
<!-- 加入到現有的 templates/Articles/index.php 檔案 -->
```

`index.php`

```
<table>
```

```
<tr>
```

```
<th>標題</th>
```

```
<th>建立時間</th>
```

```
<th>操作</th>
```

```
</tr>
```

```
<!-- 在這裡迭代 $articles 查詢物件，輸出文章資訊 -->
```

```
<!-- 這裡還有其他程式碼 -->
```

```
<td>
```

```
<?= $article->created->format(DATE_RFC850) ?>
```

```
</td>
```

```
<td>
```

```
<?= $this->Html->link('編輯', ['action' => 'edit', $article->slug]) ?>
```

```
</td>
```

```
</tr>
```

```
<?php endforeach; ?>
```

```
</table>
```

更新文章的驗證規則

- 到目前為止，我們的文章 (Articles) 尚未進行任何輸入驗證。現在讓我們透過使用驗證器 (Validator) 來修正這個問題：

```
// 加入到現有的 src/Model/Table/ArticlesTable.php 檔案

// 在 namespace 宣告下方，新增以下 use 宣告以匯入 Validator 類別
use Cake\Validation\Validator;

// 新增以下方法
public function validationDefault(Validator $validator): Validator
{
    $validator
        ->notEmptyString('title', '請輸入標題')
        ->minLength('title', 2, '標題至少需要 2 個字元')
        ->maxLength('title', 255, '標題最多 255 個字元')
        ->notEmptyString('body', '請輸入內容');

    return $validator;
}
```

index.php

功能說明

1. validationDefault() 方法：
此方法告訴 CakePHP 在執行 save() 方法時如何驗證資料。
2. 驗證規則：
 - title (標題) :
 - 必須為非空字串
 - 至少 2 個字元
 - 最多 255 個字元
 - body (內容) :
 - 必須為非空字串

更新文章的驗證規則

CakePHP 驗證引擎的特性

- CakePHP 的驗證引擎功能強大且靈活，提供了一套常用的規則（例如電子郵件地址、IP 地址驗證等），並允許您新增自訂的驗證規則。更多設置資訊，請參考[驗證文件](#)。

測試驗證規則

- 現在規則已設置完成，您可以嘗試新增標題或內容為空的文章，看看會發生什麼事。
- 由於我們使用了 `Cake\View\Helper\FormHelper::control()` 方法來建立表單元素，因此驗證錯誤訊息將會自動顯示在表單中，讓使用者知道需要修正的地方。

刪除文章

- 接下來，我們為使用者新增刪除文章的功能。首先，在 `ArticlesController` 中新增 `delete()` 方法：

```
// 加入到現有的 src/Controller/ArticlesController.php 檔案
```

`ArticlesController.php`

```
// 新增以下方法
```

```
public function delete(?string $slug)
{
    $this->request->allowMethod(['post', 'delete']);

    $article = $this->Articles->findBySlug($slug)->firstOrFail();
    if ($this->Articles->delete($article)) {
        $this->Flash->success(__('文章 {0} 已被刪除。', $article->title));

        return $this->redirect(['action' => 'index']);
    }
}
```

刪除文章

1. 刪除邏輯：

- 此邏輯會刪除指定的文章（透過 `$slug` 傳遞），並使用 `$this->Flash->success()` 顯示成功訊息，然後重新導向至 `/articles`。
- 如果使用者嘗試以 `GET` 請求刪除文章，`allowMethod()` 會拋出例外錯誤。CakePHP 內建的例外處理器會捕捉這些錯誤，並顯示友好的錯誤頁面。

2. 安全性：

- 允許使用 `GET` 請求刪除內容非常危險，例如網頁爬蟲可能會意外刪除所有內容。
- 因此，我們在控制器中使用了 `allowMethod()` 確保只接受 `POST` 或 `DELETE` 請求來刪除文章。

3. 無模板：

- 由於此操作僅執行邏輯並重定向至另一個操作，因此不需要專門的模板。

修改視圖

- 您可以更新 `index` 視圖，為使用者提供刪除文章的連結：

```
<!-- 加入到現有的 templates/Articles/index.php 檔案 -->

<!-- 這裡還有其他程式碼 -->
    <td>
        <?= $article->created->i18nFormat(yyyy年MM月dd日 EEEE HH:mm:ss) ?>
    </td>
    <td>
        <?= $this->Html->link('編輯', ['action' => 'edit', $article->slug]) ?>
        <?= $this->Form->postLink(
            '刪除',
            ['action' => 'delete', $article->slug],
            ['confirm' => '您確定要刪除這篇文章?'])
        ?>
    </td>
</tr>
<?php endforeach; ?>
</table>
```

`index.php`

功能說明

1. `postLink()` 方法：
使用 `postLink()` 建立一個連結，該連結會透過 JavaScript 執行 POST 請求來刪除文章。
2. 刪除確認提示：
此視圖使用 `FormHelper`，在使用者嘗試刪除文章前，彈出 JavaScript 確認對話框，避免意外刪除。

透過 `bake` 建立控制器



您可以使用以下指令快速生成 **ArticlesController**：

```
bin/cake bake controller articles
```

但請注意，此指令無法生成 `templates/Articles/*.php` 模板文件。

接下來的步驟

完成基本的文章管理功能後，我們將為 **Tags** 和 **Users** 資料表建立基本操作。