管理后台首页

一般情况下我们在管理后台的首页展示的都是一些系统信息和数据统计。

1. 统计数据加载

站点内容统计版块的数据，需要通过查询数据库得到具体的数值。

1.1. SQL 语句

```sql
-- 文章总数:
select count(1) from posts

-- 草稿总数:
select count(1) from posts where status = 'drafted'

-- 分类总数:
select count(1) from categories

-- 评论总数:
select count(1) from comments

-- 待审核的评论总数:
select count(1) from comments where status = 'held'
```

1.2. 执行 SQL 查询数据

在 `index.php` 页面加载（执行）过程中通过代码执行 SQL 语句，获取所需数据：

```php
// 查询文章总数

$connection = mysqli_connect(DB_HOST, DB_USER, DB_PASS, DB_NAME);

if (!$connection) {
  die('<h1>Connect Error (' . mysqli_connect_errno() . ') ' . mysqli_connect_error() .
'</h1>');
}

if ($result = mysqli_query($connection, 'select count(1) from posts')) {
  $post_count = mysqli_fetch_row($result)[0];
  mysqli_free_result($result);
}

mysqli_close($connection);

// 其余几个查询只是查询语句不同，所以这里不列举了。
```

> 源代码: step-13

MAKE IT BETTER

2. 封装查询函数

由于接下来的操作（开发）过程中会有很多需要查询数据库的地方。如果每次都按照最原始的方法去写过于麻烦。我们应该将重复的地方提取出来，封装一个公共的 `xiu_query` 函数，方便后期调用和维护。

> 说明：由于 PHP 内置函数特别多，我们在自定义函数时，函数名称最好有个性一点，否则会冲突（冲突了会有警告）。

```php
/**
 * 执行一个查询语句，返回查询到的数据（关联数组混合索引数组）
 * @param  string $sql 需要执行的查询语句
 * @return array       查询到的数据（二维数组）
 */
function xiu_query ($sql) {
  // 建立数据库连接
  $connection = mysqli_connect(DB_HOST, DB_USER, DB_PASS, DB_NAME);

  if (!$connection) {
    // 如果连接失败报错
    die('<h1>Connect Error (' . mysqli_connect_errno() . ') ' . mysqli_connect_error() . '</h1>');
  }

  // 定义结果数据容器，用于装载查询到的数据
  $data = array();

  // 执行参数中指定的 SQL 语句
  if ($result = mysqli_query($connection, $sql)) {
    // 查询成功，则获取结果集中的数据

    // 遍历每一行的数据
    while ($row = mysqli_fetch_array($result)) {
      // 追加到结果数据容器中
      $data[] = $row;
    }

    // 释放结果集
    mysqli_free_result($result);
  }

  // 关闭数据库连接
  mysqli_close($connection);

  // 返回容器中的数据
  return $data;
}
```

> 源代码: step-14

2.1. 抽象 functions.php 文件

而且这个函数不仅仅在当前页面会使用到，其他的页面中也会使用，所以我们把这个函数抽离到一个单独的 `functions.php` 文件中，并将这个文件放到根目录下（前台代码也会用到）。

> 源代码: step-15

2.2. 抽象创建数据库连接函数

按照以上方法封装很方便，但是也有一些问题：

1. 无法重用一个数据库连接对象，每次查询都是创建一个新的数据库连接，非常消耗资源。
2. 如果希望使用其他的查询函数，比如 `mysqli_fetch_assoc`、`mysqli_fetch_row`。

为此，我们将以上封装的 `xiu_query` 函数拆分成两个函数：

1. `xiu_connect` 函数，用于创建一个数据库连接对象；
2. `xiu_query` 函数，执行一个查询操作，并返回查询到的数据结果；

> 源代码: step-16

MAKE IT BETTER

3. 获取当前登录用户信息

在管理后台一般我们都需要获取当前登录用户的信息，用于界面展示和后续业务逻辑（例如新增文章）中使用，所以我们必须要能获取到当前登录用户信息。

而登录用户信息只有在登录表单提交那次请求中知道，如果后续请求也需要的话，就必须借助 Session 去保存状态，在之前开发登录状态保持功能时，我们往 Session 中存放的只是一个标识变量，可以调整一下，改为保存用户标识（用户 ID）：

```
// $_SESSION['is_logged_in'] = true;
// 保存用户 ID 到 Session 中
$_SESSION['current_logged_in_user_id'] = $user['id'];
```

需要访问控制的位置也需要调整：

```
if (empty($_SESSION['current_logged_in_user_id'])) {
  ...

}
```

> 源代码: step-17

3.1. 封装获取当前登录用户信息

我们对访问控制逻辑代码稍加改造，抽象成一个公共的函数，便于后续在其他地方公用：

```php
/**
 * 获取当前登录用户的信息
 * 如果没有获取到的话则跳转到登录页
 * 也可以通过全局变量访问返回结果
 * @return array 包含用户信息的关联数组
 */
function xiu_get_current_user () {
  if (isset($GLOBALS['current_user'])) {
    // 已经执行过了（重复调用导致）
    return $GLOBALS['current_user'];
  }

  // 启动会话
  session_start();

  if (empty($_SESSION['current_logged_in_user_id']) ||
!is_numeric($_SESSION['current_logged_in_user_id'])) {
    // 没有登录标识就代表没有登录
    // 跳转到登录页
    header('Location: /admin/login.php');
    exit; // 结束代码继续执行
  }

  // 根据 ID 获取当前登录用户信息（定义成全局的，方便后续使用）
  $GLOBALS['current_user'] = xiu_query(sprintf('select * from users where id = %d limit 1',
intval($_SESSION['current_logged_in_user_id'])))[0];

  return $GLOBALS['current_user'];
}
```

在需要访问控制的位置调用，调用之后就可以使用 `$current_user` 访问当前登录用户信息：

```php
// 获取登录用户信息
xiu_get_current_user();

...

<!-- 访问登录用户信息 -->
<div class="profile">
  <img class="avatar" src="<?php echo $current_user['avatar']; ?>">
  <h3 class="name"><?php echo $current_user['nickname']; ?></h3>
</div>
```

> 源代码: step-18