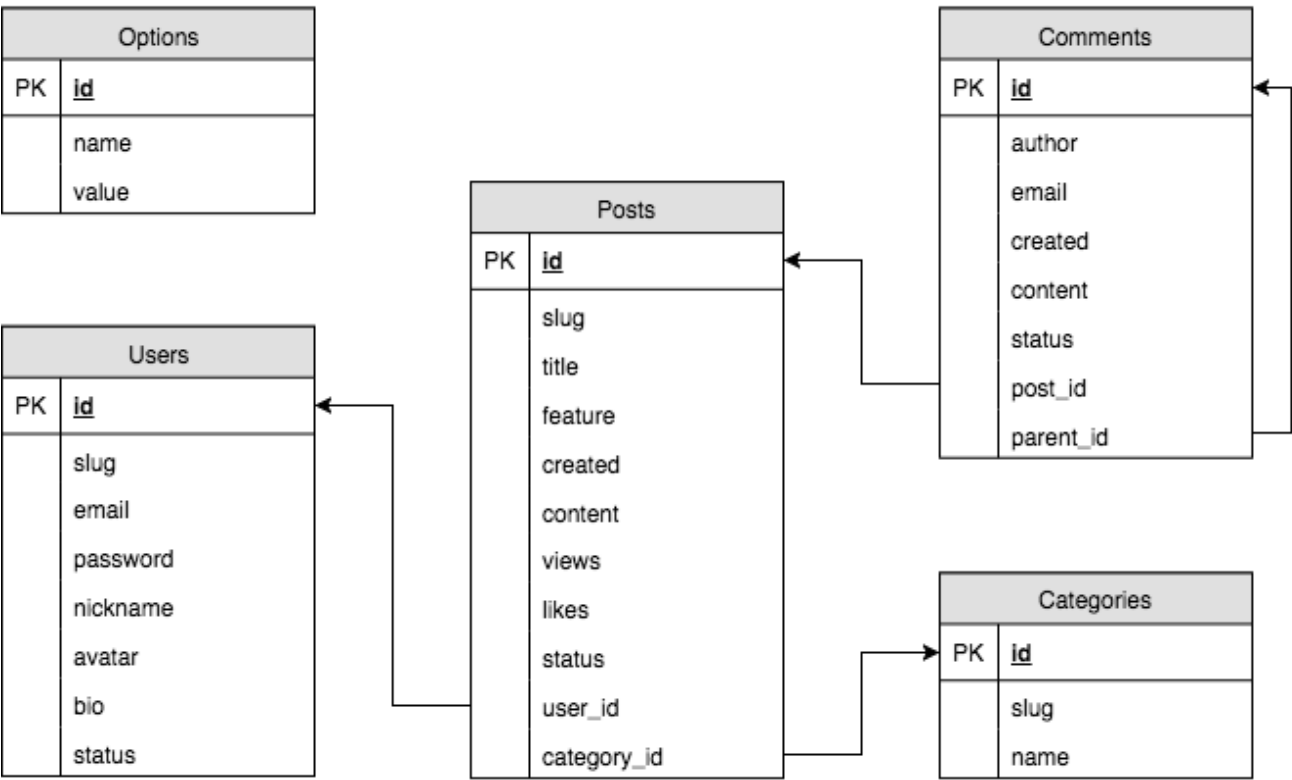


# 准备工作

- 数据库设计
- 基础结构搭建

## 1. 数据库设计

根据我们的业务需要设计数据库的结构，这个过程是每个项目开始时所必须的，一般由专门的 DBA 角色完成（很多没有划分的非常具体的公司由后端开发人员兼任）。



### 1.1. 选项表（options）

用于记录网站的一些配置属性信息，如：站点标题，站点描述等

字段	描述	备注
id	🔑 主键	
key	属性键	snake_case
value	属性值	JSON 格式

## 1.2. 用户表 ( users )

用于记录用户信息

字段	描述	备注
id	🔑 主键	
slug	URL 别名	
email	邮箱	亦做登录名
password	密码	
nickname	昵称	
avatar	头像	图片 URL 路径
bio	简介	
status	状态	未激活 ( unactivated ) / 激活 ( activated ) / 禁止 ( forbidden ) / 回收站 ( trashed )

## 1.3. 文章表 ( posts )

用于记录文章信息

字段	描述	备注
id	🔑 主键	
slug	URL 别名	
title	标题	
feature	特色图像	图片 URL 路径
created	创建时间	
content	内容	
views	浏览次数	
likes	点赞数	
status	状态	草稿 ( drafted ) / 已发布 ( published ) / 回收站 ( trashed )
user_id	👤 用户 ID	当前文章的作者 ID
category_id	👤 分类 ID	当前文章的分类 ID

提问：为什么要用关联 ID ？

一般情况下，表与表之间的数据肯定会产生某种联系，我们一般在当前表中存放被关联的数据ID，这样做的目的，保证两个表之间的数据联系，用ID存的目的是增强可维护性




### 1.4. 分类表 ( categories )

用于记录文章分类信息

字段	描述	备注
id	🔑 主键	
slug	URL 别名	
name	分类名称	

### 1.5. 评论表 ( comments )

用于记录文章评论信息

字段	描述	备注
id	 主键	
author	作者	
email	邮箱	
created	创建时间	
content	内容	
status	状态	待审核 ( held ) / 准许 ( approved ) / 拒绝 ( rejected ) / 回收站 ( trashed )
post_id	 文章 ID	
parent_id	 父级 ID	

点击下载：[初始化数据库脚本](#)

MAKE IT BETTER

## 2. 搭建项目架构

项目最基本的分为两个大块，前台（对大众开放）和后台（仅对管理员开放）。

一般在实际项目中，很多人会把前台和后台分为两个项目去做，部署时也会分开部署：

- 前台访问地址：`https://www.wedn.net/`
- 后台访问地址：`https://admin.wedn.net/`

这样做相互独立不容易乱，也更加安全。但是有点麻烦，所以我们采用更为常见的方案：让后台作为一个子目录出现。这样的话，大体结构就是：

- 前台访问地址：`https://www.wedn.net/`
- 后台访问地址：`https://www.wedn.net/admin/`

### 2.1. 基本的目录结构

```

1  └─ baixiu ..... 项目文件夹（网站根目录）
2    └─ admin ..... 后台文件夹
3      └─ index.php ..... 后台脚本文件
4    └─ static ..... 静态文件夹
5      └─ assets ..... 资源文件夹
6      └─ uploads ..... 上传文件夹
7    └─ index.php ..... 前台脚本文件

```

### 2.1.1. 基本原则

- 先明确一共有多少个页面
- 一个页面就对应一个 php 文件去处理

▶ 源代码: step-01

## 2.2. 整合静态资源文件

### 2.2.1. 静态文件 vs. 动态文件

- 静态文件指的就是服务器不会经过任何处理就返回给客户端浏览器的文件，比如：图片、样式表、字体文件等
- 动态文件指的就是服务器会对请求的文件进行处理，并将处理后的结果返回给客户端浏览器的文件，比如：PHP 文件、ASP 文件、JSP 文件

### 2.2.2. 具体操作

由于 Apache / Nginx 这一类 Web Server 本身可以处理静态文件请求，所以不需要 PHP 处理静态文件请求。只需要将静态资源放到网站目录中，即可访问

```

1  └─ baixiu ..... 项目文件夹（网站根目录）
2    └─ .....
3    └─ static ..... 静态文件夹
4      └─ assets ..... 资源文件夹
5  +  └─ └─ css ..... 样式文件夹
6  +  └─ └─ img ..... 图片文件夹
7  +  └─ └─ js ..... 脚本文件夹
8  +  └─ └─ venders ..... 第三方资源
9      └─ uploads ..... 上传文件夹
10 +  └─ └─ 2017 ..... 2017 年上传文件目录
11 └─ .....

```

## 注意

- `static` 目录中只允许出现静态文件。
- `assets` 目录中放置网页中所需的资源文件。
- `uploads` 目录中放置网站运营过程中上传的文件，如果担心文件过多，可以按年归档（一年一个文件夹）。

📄 源代码: step-02

## 2.3. 项目配置文件

由于在接下来的开发过程中，肯定又一部分公共的成员，例如数据库名称，数据库主机，数据库用户名密码等，这些数据应该放到公共的地方，抽象成一个配置文件 `config.php` 放到项目根目录下。

这个配置文件采用定义常量的方式定义配置成员：

```
1  /**
2   * 数据库主机
3   */
4  define('DB_HOST', '127.0.0.1');
5
6  /**
7   * 数据库用户名
8   */
9  define('DB_USER', 'root');
10
11 /**
12 * 数据库密码
13 */
14 define('DB_PASS', 'wanglei');
15
16 /**
17 * 数据库名称
18 */
19 define('DB_NAME', 'baixiu');
```

注意：这种只有服务端代码的 PHP 文件应该去除结尾处的 `?>`，防止输出内容

在需要的时候可以通过 `require` 载入：

```
1 // 载入配置文件
2 require 'config.php';
3
4 ...
5
6 // 用到的时候
7 echo DB_NAME;
```

▶ 源代码: step-03

### 2.3.1. 载入脚本的几种方式对比

- `require`
- `require_once`
- `include`
- `include_once`

- 共同点：
  - 都可以在当前 PHP 脚本文件执行时载入另外一个 PHP 脚本文件。
- `require` 和 `include` 不同点：
  - 当载入的脚本文件不存在时，`require` 会报一个致命错误（结束程序执行），而 `include` 不会
- 有 `once` 后缀的特点：
  - 判断当前载入的脚本文件是否已经载入过，如果载入了就不再执行

提问：由以上几种方式的对比可以得出：在载入 `config.php` 时使用哪种方式更合适？

### 2.3.2. 显示 PHP 错误信息

当执行 PHP 文件发生错误时，如果在页面上不显示错误信息，只是提示 500 Internal Server Error 错误，应该是 PHP 配置的问题，解决方案就是：找到 `php.ini` 配置文件中 `display_errors` 选项，将值设置为 `On`

```
1 ; http://php.net/display-errors
2 ; display_errors = Off
3 display_errors = On
4
5 ; The display of errors which occur during PHP's startup sequence are handled
```





## 4.2. 抽离公共部分

由于每一个页面中都有一部分代码（侧边栏）是相同的，分散到各个文件中，不便于维护，所以应该抽象到一个公共的文件中。

于是我们在 `admin` 目录中创建一个 `inc`（includes 的简称）子目录，在这个目录里创建一个 `sidebar.php` 文件，用于抽象公共的侧边栏 `<div class="aside"> ... </div>`，然后在每一个需要这个模块的页面中通过 `include` 载入：

```
1 ...
2 <?php include 'inc/sidebar.php' ;?>
3 ...
```

提问：为什么使用 `include` 而不是 `require` ？

▶ 源代码: step-05

### 4.2.1. 侧边栏的焦点状态

由于侧边栏在不同页面时，active class name 出现的位置不尽相同，所以我们需要区分当前 `sidebar.php` 文件是在哪个页面中载入的，从而明确焦点状态。

所以目前的关键问题就出现在了如何在 `sidebar.php` 中知道当前被哪个文件载入了。

通过查看 `include` 函数的文档发现：如果 `a.php` 通过 `include` 载入了 `b.php` 文件，那么在 `b.php` 文件中可以访问到 `a.php` 中定义的变量。

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

借助这个特性，我们可以在各个页面中定义一个标识变量，然后在 `sidebar.php` 中通过这个标识变量区别不同页面的载入：

每一个菜单项 `<li>` 元素：

```
1 ...
2 <li<?php echo $current_page == 'dashboard' ? ' class="active"' : ''; ?>
3   <a href="index.php"><i class="fa fa-dashboard"></i>仪表盘</a>
4 </li>
5 ...
```

对于有子菜单的菜单项，有一点例外：

```

1 ...
2 <li>?php echo in_array($current_page, array('posts', 'post-add', 'categories')) ? '
  class="active"' : ''; ?>
3   <a href="#menu-posts">?php echo in_array($current_page, array('posts', 'post-add',
  'categories')) ? '' : ' class="collapsed"'; ?> data-toggle="collapse">
4     <i class="fa fa-thumb-tack"></i>文章<i class="fa fa-angle-right"></i>
5   </a>
6   <ul id="menu-posts" class="collapse">?php echo in_array($current_page, array('posts', 'post-
  add', 'categories')) ? ' in' : ''; ?>
7     <li>?php echo $current_page == 'posts' ? ' class="active"' : ''; ?><a href="posts.php">所
  有文章</a></li>
8     <li>?php echo $current_page == 'post-add' ? ' class="active"' : ''; ?><a href="post-
  add.php">写文章</a></li>
9     <li>?php echo $current_page == 'categories' ? ' class="active"' : ''; ?><a
  href="categories.php">分类目录</a></li>
10   </ul>
11 </li>
12 ...

```

▶ 源代码: step-06