好的，我已理解你的需求。你希望从零代码基础出发，系统地学习 WordPress 开发，最终目标是能够看懂开源博客的 WordPress 源码，并具备独立创建主题和插件的能力。你将使用 Local by Flywheel 作为本地开发环境。

这是一个非常有价值的学习目标，WordPress 是一个庞大且功能丰富的系统，掌握其开发需要循序渐进。我将扮演你的知识架构师，为你构建一个从基础到进阶的、像书籍/教程一样的知识体系。整个过程将分多轮输出，确保内容详尽且逻辑清晰。

【主题】：从零开始的 WordPress 主题与插件开发（面向无代码基础者）

【学习目标概述】：

1. 掌握 Web 开发基础知识（HTML, CSS, PHP - 核心）。
2. 理解 WordPress 的核心概念、架构与工作原理。
3. 熟练使用 Local by Flywheel 搭建本地开发环境。
4. 学会创建、修改和理解 WordPress 主题。
5. 学会创建、修改和理解 WordPress 插件。
6. 初步具备阅读和理解 WordPress 核心代码中与博客展示、主题、插件相关部分的能力。

【学习路线图/目录规划】：

为了达到上述目标，我们将按照以下结构逐步深入：

第一部分：启程与基石 1.1. 欢迎与教程导览 1.2. 为什么选择 WordPress 开发？ 1.3. WordPress 的本质与构成（非代码层面） 1.4. Web 开发基础概览（HTML, CSS, JavaScript, PHP 的角色） 1.5. 搭建本地开发环境：Local by Flywheel 的安装与使用 1.6. WordPress 的文件结构与数据库初步认知

第二部分：Web 开发基础速成（为 WordPress 开发服务） 2.1. HTML 基础：构建网页骨架 2.2. CSS 基础：美化网页样式 2.3. PHP 基础（核心）：动态生成内容 2.3.1. PHP 是什么？为什么 WordPress 用它？ 2.3.2. PHP 的基本语法、变量与数据类型 2.3.3. 控制结构（条件判断与循环） 2.3.4. 函数的使用与定义 2.3.5. 数组与关联数组 2.3.6. 包含文件 2.4. JavaScript 基础（了解）：客户端交互

第三部分：WordPress 核心概念深入 3.1. WordPress 请求生命周期（理解代码如何执行） 3.2. WordPress 核心机制：动作（Actions）与过滤（Filters）—— 钩子（Hooks）详解 3.3. WordPress 主题系统：模板层级结构 3.4. WordPress 插件系统：工作原理 3.5. 理解 The Loop (循环)：WordPress 内容展示的核心

第四部分：WordPress 主题开发实战（从零开始） 4.1. 主题文件的基本构成 (`style.css`, `index.php`, `functions.php`) 4.2. 创建一个最小化的 WordPress 主题 4.3. 使用 The Loop 显示文章列表 4.4. 常用的模板标签（Template Tags）介绍与使用 4.5. 拆分主题文件：`header.php`, `footer.php`, `sidebar.php` 4.6. 理解并实现不同的模板文件（`single.php`, `page.php`, `archive.php`, etc.） 4.7. 添加导航菜单与侧边栏小工具 4.8. 使用 `functions.php` 增强主题功能（初级） 4.9. 主题开发最佳实践与注意事项

第五部分：WordPress 插件开发实战 5.1. 插件文件的基本构成与插件头部信息 5.2. 创建一个最简单的 WordPress 插件 5.3. 利用动作（Actions）钩子：在特定时刻执行代码 5.4. 利用过滤（Filters）钩子：修改 WordPress 数据 5.5. 创建自定义文章类型（Custom Post Types） 5.6. 创建自定义分类法（Custom Taxonomies） 5.7. 为插件创建管理页面 5.8. Shortcode（短代码）的创建与使用 5.9. 插件开发中的数据库操作（WPDB 类介绍） 5.10. 在插件中正确引入样式和脚本 5.11. 插件开发最佳实践与安全性

第六部分：理解 WordPress 核心代码（初步）与进阶 6.1. WordPress 加载流程简介（`wp-config.php`, `wp-settings.php` 的作用） 6.2. 分析博客相关的核心文件（如 `index.php`, `wp-blog-header.php`, `wp-`

`includes/template-loader.php` 简化版） 6.3. 在核心代码中识别钩子点 6.4. 调试技巧与工具（`WP_DEBUG`, Query Monitor 插件） 6.5. 版本控制基础 (Git 简介) 6.6. 性能优化基础 6.7. 安全性基础 6.8. 持续学习与资源推荐

好的，这个目录规划非常详细，我们将一步步覆盖。请注意，从零基础到"看懂源码"并"独立开发复杂功能"需要时间和实践，这个教程将为你打下最坚实的基础和提供清晰的路径。我们将从第一部分开始。

第一部分：启程与基石

欢迎来到 WordPress 开发的世界！你做了一个非常棒的选择，WordPress 不仅是一个强大的内容管理系统，它背后还有一个庞大且活跃的开发者社区，提供了无限的可能性。

1.1. 欢迎与教程导览

你好！很高兴能引导你踏上 WordPress 开发之旅。你设定了一个非常实际且有挑战性的目标：从零开始，不仅学会如何"使用"WordPress，更能深入到"开发"层面，理解其内部机制，并最终能够定制化甚至贡献。

本教程就是为你量身定制的。它不是一个简单的"照着做"教程，而是一个尝试构建知识体系的过程。我们将从最พื้นฐานเบื้องต้นเลยจริงๆ层开始，逐步搭建你的知识楼阁。每一部分都会在前一部分的基础上进行，所以请耐心，逐步消化。

这个教程将：

- **系统化**：按照逻辑顺序组织知识，确保你理解每个概念的来龙去脉。
- **基础化**：从最基本的 Web 技术讲起，不跳过任何对理解 WordPress 至关重要的前置知识。
- **实践化**：理论知识将与实际操作（搭建环境、编写代码）紧密结合。
- **深入化**：不仅教你如何做，更解释为什么这样做，帮助你理解 WordPress 的设计哲学。
- **分阶段**：内容量大，我们会分多个部分交付，你可以按照自己的节奏学习。

请记住，学习编程和开发是一个需要不断练习和解决问题的过程。不要害怕犯错，错误是学习最好的催化剂。

1.2. 为什么选择 WordPress 开发？

在我们深入技术细节之前，花点时间思考一下：为什么是 WordPress？

- **市场份额巨大**：WordPress 驱动着全球超过 40% 的网站。这意味着巨大的就业机会、客户需求和丰富的资源（主题、插件、文档）。
- **强大的内容管理能力**：它最初是博客平台，但已经发展成为一个全功能的 CMS，可以构建各种类型的网站：企业站、电商、社区、作品集等等。
- **丰富的生态系统**：有数以万计的免费和付费主题、插件可用，极大地扩展了其功能。
- **活跃的社区**：遇到问题时，可以很容易地找到帮助、教程和开发者社区。
- **相对友好的入门曲线（作为用户）**：虽然开发有其复杂性，但作为用户，WordPress 是相对容易上手的。
- **PHP 的普及性**：PHP 是服务器端脚本语言中应用最广泛的之一，学习 PHP 不仅限于 WordPress 开发，对其他 Web 开发领域也有帮助。
- **开源**：你可以自由地使用、修改和分发 WordPress。最重要的是，它的代码是公开的，你可以研究它，学习它是如何工作的。

选择 WordPress 开发，意味着你进入了一个充满活力和机会的领域。

1.3. WordPress 的本质与构成（非代码层面）

在开始写代码之前，先从宏观上理解 WordPress 是什么：

本质上，WordPress 是一个基于 PHP 和 MySQL 构建的内容管理系统 (CMS)。

这听起来有点技术，但简单来说：

- **CMS (Content Management System)**：内容管理系统。它提供了一套用户友好的界面（WordPress 后台）来创建、编辑、组织和发布内容（文章、页面、图片、视频等），而无需直接与代码或数据库打交道。
- **PHP**：一种服务器端脚本语言。当你访问一个 WordPress 网站时，服务器会运行 PHP 代码。这些代码会从数据库中提取内容、处理逻辑，然后生成最终的 HTML、CSS 和 JavaScript 发送到你的浏览器，最终显示出你看到的网页。
- **MySQL**：一种关系型数据库管理系统。WordPress 将所有重要的信息存储在数据库中：文章内容、页面、用户、评论、设置等等。PHP 代码负责与数据库交互（读取、写入、更新、删除数据）。

所以，当你访问一个 WordPress 网站时，幕后发生的事情大致是：

1. 你的浏览器向服务器发送请求。
2. 服务器接收请求，识别出这是 WordPress 网站，并执行相应的 PHP 文件。
3. PHP 文件根据请求（比如你想看哪篇文章）查询 MySQL 数据库，获取数据。
4. PHP 将获取到的数据与主题模板结合，生成完整的 HTML 网页。
5. 服务器将生成的 HTML、以及相关的 CSS 和 JavaScript 文件发送回你的浏览器。
6. 你的浏览器解析并渲染这些文件，最终显示出网页。

理解这个基本的“请求 -> PHP -> 数据库 -> 生成 HTML -> 浏览器”流程，对于后续理解 WordPress 的代码结构和工作原理至关重要。

WordPress 由三大部分构成：

1. **核心程序 (Core)**：这是 WordPress 的基础，包含了运行一个网站所需的所有核心文件、函数和 API。这是由 WordPress 官方团队维护的。
2. **主题 (Themes)**：主题负责网站的**外观**。它控制着你的内容如何呈现给访问者，包括布局、颜色、字体、样式等等。主题是一系列模板文件（PHP, HTML, CSS, JavaScript）的集合。
3. **插件 (Plugins)**：插件负责网站的**功能**。它们可以添加新的特性、修改现有行为、与外部服务集成等等。插件也是一系列 PHP 文件（可能包含其他类型文件）的集合。

核心程序提供了基础设施和 API，主题决定了“长什么样”，插件决定了“能做什么”。主题和插件通过 WordPress 提供的“钩子”（Hooks）机制与核心程序进行交互，而不需要修改核心文件本身。这是 WordPress 灵活且可扩展的关键所在。

1.4. Web 开发基础概览（HTML, CSS, JavaScript, PHP 的角色）

虽然我们的目标是 WordPress 开发，但 WordPress 网站的最终呈现依赖于标准的 Web 技术。你需要对它们有一个基本的了解，特别是 PHP。

- **HTML (HyperText Markup Language)**：超文本标记语言。它是网页的**骨架**。HTML 使用标签（如 `<p>`, `<h1>`, `<a>`, `<img>`, `<div>`）来定义网页的结构和内容。例如，一个段落、一个标题、一张图片、一个链接都是用 HTML 标签来表示的。
    - *在 WordPress 开发中*：你的主题模板文件主要输出 HTML 结构。WordPress 的 PHP 函数会生成 HTML 片段。

- **CSS (Cascading Style Sheets)**：层叠样式表。它是网页的**外观**。CSS 用来描述 HTML 元素应该如何显示，包括颜色、字体、大小、布局、边距等等。
    - *在 WordPress 开发中*：你的主题 (`style.css` 文件以及其他可能的 CSS 文件) 控制着网站的视觉样式。
- **JavaScript (JS)**：一种客户端脚本语言。它负责网页的**交互**和动态效果。JS 可以在用户的浏览器中运行，响应用户的操作（点击按钮、鼠标移动），修改网页内容，发送异步请求等。
    - *在 WordPress 开发中*：虽然 PHP 是核心，但 JS 常用于主题和插件中实现前端交互、AJAX 请求、复杂的功能模块（如古腾堡编辑器底层也大量使用 React/JS）。我们将从基础开始，不需要深入复杂的 JS 框架，但了解其基本作用和如何在 WordPress 中引入是必要的。
- **PHP (Hypertext Preprocessor)**：一种服务器端脚本语言。它是 WordPress 的**大脑**和**逻辑**层。PHP 在服务器上运行，处理用户的请求，与数据库交互，动态生成 HTML 内容，处理表单提交，等等。
    - *在 WordPress 开发中*：这是你投入精力最多的地方。主题的模板文件 (`.php` 后缀) 和插件文件 (`.php` 后缀) 都是用 PHP 编写的。WordPress 的核心 API 也是通过 PHP 函数暴露的。

简单来说： HTML: **结构** (名词、骨骼) CSS: **样式** (形容词、皮肤) JavaScript: **行为** (动词、肌肉/神经) PHP: **逻辑** (思想、大脑/内脏 - 在服务器端)

由于你没有代码基础，我们将从 HTML 和 CSS 的最基本概念开始，然后会花更多时间在 PHP 上，因为它对 WordPress 开发至关重要。

1.5. 搭建本地开发环境：Local by Flywheel 的安装与使用

在开始编写任何代码之前，你需要一个可以在本地计算机上运行 WordPress 的环境。这样你就可以离线工作、测试新功能而不会影响线上网站。Local by Flywheel (通常简称 Local) 是一个非常流行且易于使用的本地 WordPress 开发工具。

为什么要用本地开发环境?

- **安全**：你可以在不受攻击的环境中随意试验和学习，即使弄坏了也不会有任何损失。
- **速度**：在本地运行比通过互联网访问远程服务器快得多。
- **方便**：无需上传文件到服务器，直接在本地编辑文件即可看到结果。
- **离线工作**：即使没有网络连接，你也可以继续开发。

Local by Flywheel 简介

Local 是一个免费的桌面应用程序，它提供了一个集成的开发环境，包含运行 WordPress 所需的一切：Web 服务器 (Nginx/Apache 可选)、PHP、MySQL 数据库，并能一键安装 WordPress。它的界面友好，特别适合初学者。

安装 Local by Flywheel

1. **下载**：访问 Local 的官方网站 (https://localwp.com/)。点击下载按钮。你需要选择你的操作系统 (Windows, macOS, Linux)。下载可能需要填写一些基本信息，但它是免费的。
2. **安装**：运行下载的安装程序。安装过程通常是标准的“下一步”流程。你可以选择安装路径，通常保持默认即可。
3. **运行**：安装完成后，启动 Local 应用程序。

使用 Local by Flywheel 创建一个新的 WordPress 站点

1. **打开 Local**：启动 Local 应用程序。

2. **点击 "+" 按钮**：通常在左下角或左侧菜单栏有一个大大的 "+" 按钮，用于“创建一个新网站”。
3. **选择创建新网站**：选择 "Create a new site"。
4. **命名网站**：给你的网站起一个名字（例如 `my-first-wp-project`）。Local 会根据这个名字在本地创建一个目录和相关的配置。
5. **选择环境**：这里通常有两个选项：
    - **Preferred (推荐)**：这是 Local 默认且优化的环境，通常包含 Nginx, PHP 最新版本, MySQL 8。对于大多数用户来说，选择这个即可。
    - **Custom (自定义)**：允许你选择不同的 Web 服务器 (Apache vs Nginx)、PHP 版本、MySQL 版本。如果你有特定需求或想模拟生产环境，可以选择这个。对于初学者，选择 **Preferred** 即可。
6. **设置 WordPress 用户**：设置 WordPress 管理员的用户名、密码和邮箱。**请务必记住这个用户名和密码！** 这是你登录 WordPress 后台所需的。你也可以修改网站标题。
7. **点击 "Add Site"**：Local 会开始下载 WordPress 并设置 Web 服务器、数据库。这个过程可能需要几分钟。
8. **完成**：当网站创建成功后，你会看到一个包含网站详细信息的页面。这里会显示网站名称、本地域名 (如 `my-first-wp-project.local`)、Web 服务器、PHP 版本、数据库版本、SSL 状态等。

管理你的本地 WordPress 站点

在 Local 应用中，你可以：

- **启动/停止网站**：每个网站旁边都有一个开关按钮。在开发时启动，不开发时停止可以节省资源。
- **访问网站**：点击网站信息页面上的 "Open Site" 按钮，会在浏览器中打开你的网站前台。
- **访问 WP Admin (后台)**：点击 "WP Admin" 按钮，会在浏览器中打开你的 WordPress 后台登录页面。输入你之前设置的管理员用户名和密码即可登录。
- **查看网站文件**：点击网站信息页面上的 "Go to Site Folder" 链接，会打开你的网站在本地文件系统中的根目录。这是你以后修改主题和插件文件的地方。通常路径会是 `Local Sites/你的网站名称/app/public`。
- **打开数据库管理工具**：点击 "Database" 旁边的 "Open Adminer" 或 "Sequel Ace/Pro" 按钮。Local 集成了一个数据库管理工具，让你无需命令行即可查看和修改数据库内容。对于初学者，暂时用不上，但知道它的存在即可。
- **开启 SSL**：Local 可以为你的本地站点生成 SSL 证书，使其通过 HTTPS 访问，更接近真实环境。通常 Local 会自动提示你信任证书。
- **One-click Admin**：勾选这个选项后，你可以直接点击 "WP Admin" 按钮而无需输入密码登录，非常方便开发（**但请勿在任何线上环境启用此功能！**）。

花点时间安装 Local，创建一个你的第一个本地 WordPress 网站，并尝试登录后台。熟悉 Local 的界面和基本操作是学习 WordPress 开发的第一步。

1.6. WordPress 的文件结构与数据库初步认知

成功搭建本地环境后，我们来初步了解一下你的 WordPress 网站在文件系统中的模样。

进入你的网站根目录（通过 Local 的 "Go to Site Folder" 打开的目录），通常在 `app/public` 子目录下，你会看到一些文件和文件夹：

```
your-site-name/
└── app/
    └── public/
```

```
├── index.php
├── wp-admin/           # WordPress 后台管理文件
├── wp-content/         # 主题、插件、上传文件、升级等内容都放在这里!
├── wp-includes/        # WordPress 核心功能文件，存放了大量的核心函数和类
├── wp-config.php       # 重要的配置文件，包含数据库连接信息等
├── wp-config-sample.php # 示例配置文件
├── wp-activate.php
├── wp-blog-header.php
├── wp-comments-post.php
├── wp-cron.php
├── wp-links-opml.php
├── wp-load.php
├── wp-login.php        # 登录页面文件
├── wp-mail.php
├── wp-settings.php     # WordPress 加载过程中最重要的文件之一
├── wp-signup.php
├── wp-trackback.php
└── xmlrpc.php
```

对于初学者，请**不要随意修改** `wp-admin` 和 `wp-includes` 文件夹内的文件，它们是 WordPress 的核心，任何错误的修改都可能导致网站崩溃。**所有你的开发工作（主题和插件）都将集中在 `wp-content` 文件夹内。**

`wp-content` 文件夹结构

进入 `wp-content` 文件夹，你会看到以下几个重要的子文件夹：

```
wp-content/
├── themes/       # 所有安装的主题都放在这里
├── plugins/      # 所有安装的插件都放在这里
├── uploads/      # 你上传的媒体文件（图片、文档等）默认放在这里
└── upgrade/      # WordPress 升级时使用的临时文件夹
```

- `themes/`: 你下载或自己开发的主题都将存放在这个文件夹下的独立子文件夹中。
- `plugins/`: 你下载或自己开发的插件都将存放在这个文件夹下的独立子文件夹中。
- `uploads/`: 这是一个非常重要的文件夹，你的媒体库中的所有文件（图片、视频等）都默认按年月目录结构存放在这里。

核心提示：你的所有定制开发工作（主题和插件的创建、修改）都只发生在 `wp-content` 文件夹内，绝不修改 `wp-admin` 或 `wp-includes`！

`wp-config.php` 文件

这是 WordPress 的配置文件，非常重要。它包含：

- 数据库连接信息 (数据库名, 用户名, 密码, 数据库主机地址)。Local 会在你创建网站时自动生成并填写这些信息。
- 用于加强安全性的密钥和盐值 (Keys and Salts)。
- 其他一些高级配置选项，例如是否开启调试模式 (`define('WP_DEBUG', true);`)。

暂时你只需要知道这个文件存在且很重要，它连接了你的 WordPress 文件和数据库。在后续学习调试时，我们会用到修改 `WP_DEBUG` 的配置。

WordPress 数据库 (MySQL)

虽然你不需要成为数据库专家，但了解 WordPress 如何使用数据库是理解其工作原理的关键。WordPress 将网站的大部分内容和设置存储在 MySQL 数据库中。

通过 Local 的数据库管理工具 (如 Adminer)，你可以看到 WordPress 数据库中有很多表，表名通常以 `wp_` 开头（这个前缀是可以在安装时修改的，或者在 `wp-config.php` 中定义，但默认是 `wp_`）。

一些重要的表（了解即可，无需记忆）：

- `wp_posts`: 存储文章 (posts), 页面 (pages), 自定义文章类型 (custom post types), 媒体附件 (attachments), 导航菜单项 (nav menu items) 等 **内容**。
- `wp_postmeta`: 存储文章/页面的元数据 (metadata)，例如特色图片、自定义字段等。
- `wp_comments`: 存储网站的评论。
- `wp_commentmeta`: 存储评论的元数据。
- `wp_terms`: 存储分类 (categories), 标签 (tags), 自定义分类法 (custom taxonomies) 中的“项”本身（例如，“科技”这个分类）。
- `wp_term_taxonomy`: 存储分类法（taxonomy）的定义，将 `wp_terms` 中的项与特定的分类法关联（例如，“科技”属于“分类”这个分类法）。
- `wp_term_relationships`: 将文章/页面与 `wp_terms` 中的项关联起来（例如，某篇文章属于“科技”分类）。
- `wp_users`: 存储网站注册用户的信息。
- `wp_usermeta`: 存储用户元数据，例如昵称、联系方式等。
- `wp_options`: 存储网站的各种设置和选项，例如网站标题、描述、固定链接设置、插件设置等。

理解这些表的存在和它们大致存储的内容，有助于你理解 WordPress 的数据是如何组织的，以及主题和插件如何与这些数据交互。当你使用 WordPress 后台修改内容或设置时，实际上就是在修改数据库中的数据。当你访问网站时，WordPress 的 PHP 代码就是从这些表中读取数据来生成网页。

好的，第一部分到此结束。我们了解了 WordPress 是什么，为什么学习它有价值，它由哪些部分构成，基本的 Web 技术扮演什么角色，以及如何搭建本地环境并初步认识了 WordPress 的文件结构和数据库。

本部分的实践作业：

1. 成功安装 Local by Flywheel。
2. 使用 Local 创建一个或多个本地 WordPress 网站。
3. 登录到你创建的本地网站的 WordPress 后台。
4. 通过 Local 打开网站的本地文件目录，并找到 `wp-content` 文件夹。
5. （可选，如果你好奇）使用 Local 的数据库管理工具（如 Adminer），查看一下 `wp_posts` 和 `wp_options` 表，初步感受一下数据库里的数据。

好的，我们继续。在第一部分，你已经了解了 WordPress 的宏观概念、组成部分，搭建了本地开发环境 Local，并初步认识了 WordPress 的文件结构。现在，是时候为你武装一些基础的 Web 开发语言知识了。

请记住，本部分是一个“速成”课程，旨在为你提供足够的基础，让你能够理解 WordPress 代码并开始编写自己的代码。我们不会深入每种语言的所有细节，而是聚焦于与 WordPress 开发最相关的部分。

第二部分：Web 开发基础速成（为 WordPress 开发服务）

WordPress 是一个基于 PHP 构建的系统，其前端展示依赖于 PHP 生成的 HTML，再由 CSS 负责样式，JavaScript 负责交互。因此，理解这三种语言的基本原理和它们如何在 WordPress 中协同工作，是至关重要的一步。

2.1. HTML 基础：构建网页骨架

HTML (HyperText Markup Language) 是用来创建网页**结构**的标记语言。它告诉浏览器网页上有什么内容（标题、段落、图片、链接等）以及它们的组织结构。

- **标记 (Tags)**：HTML 由一系列的标签组成。标签通常成对出现，有一个开始标签 `<tagname>` 和一个结束标签 `</tagname>`。内容放在开始标签和结束标签之间。例如：

  ```
  <p>这是一个段落。</p>
  <h1>这是一个一级标题</h1>
  ```

- **元素 (Elements)**：从开始标签到结束标签（包括标签本身和内容）构成一个 HTML 元素。
- **属性 (Attributes)**：标签可以有属性，属性提供了关于元素的额外信息。属性通常写在开始标签内，格式是 `name="value"`。例如：

  ```
  <a href="https://www.wordpress.org/">WordPress 官网</a>
  <img src="image.jpg" alt="描述图片的文字">
  ```

  这里的 `href` 和 `src`, `alt` 都是属性。
- **常见标签**：
  - 结构：`<html>`, `<head>`, `<body>`
  - 标题：`<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, `<h6>` (数字越小，标题级别越高，字体越大)
  - 段落：`<p>`
  - 链接：`<a>` (使用 `href` 属性指定目标 URL)
  - 图片：`<img>` (使用 `src` 属性指定图片路径，`alt` 属性提供替代文本)
  - 列表：`<ul>` (无序列表), `<ol>` (有序列表), `<li>` (列表项)
  - 分割容器：`<div>` (块级容器，常用于布局), `<span>` (行内容器)
  - 表单：`<form>`, `<input>`, `<textarea>`, `<button>`
- **HTML 文档基本结构**：

  ```
  <!DOCTYPE html>  <!-- 声明文档类型为 HTML5 -->
  <html lang="zh-CN"> <!-- html 根元素，lang 属性指定语言 -->
  <head>
      <meta charset="UTF-8"> <!-- 指定字符编码 -->
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <!-- 响应式设计用 -->
      <title>网页标题</title> <!-- 显示在浏览器标签页上的标题 -->
      <!-- 这里可以引入 CSS 文件和 JavaScript 文件 -->
      <link rel="stylesheet" href="style.css">
  </head>
  ```

```
<body>
    <!-- 网页的可见内容都放在这里 -->
    <header>
        <h1>网站标题</h1>
    </header>
    <nav>
        <!-- 导航菜单 -->
    </nav>
    <main>
        <article>
            <h2>文章标题</h2>
            <p>文章内容...</p>
        </article>
    </main>
    <footer>
        <!-- 底部信息 -->
    </footer>
    <script src="script.js"></script> <!-- 这里可以引入 JavaScript 文件 -->
</body>
</html>
```

WordPress 主题的模板文件 (`.php` 文件) 最终会输出这样的 HTML 结构。

在 WordPress 开发中理解 HTML 的意义：

- **模板输出**：WordPress 的 PHP 模板文件会包含大量的 HTML 标记，以及用 PHP 生成的动态内容（如文章标题、内容）。你需要能够读懂这些 HTML 来理解页面结构。
- **CSS 选择器**：你需要使用 CSS 来美化页面，CSS 选择器是基于 HTML 结构的。理解 HTML 结构才能正确地编写 CSS 来选中并美化特定的元素。
- **语义化**：使用正确的 HTML 标签（如 `<article>`, `<nav>`, `<footer>`）可以提高网页的可访问性和 SEO。

实践建议：

打开一个简单的 HTML 文件，修改其中的内容和标签，然后在浏览器中打开看看效果。这能帮助你快速熟悉 HTML 的基本语法。

2.2. CSS 基础：美化网页样式

CSS (Cascading Style Sheets) 用来控制 HTML 元素的**外观**和**布局**。

- **规则 (Rules)**：CSS 由一系列规则组成。每个规则包含一个**选择器**和一个或多个**声明块**。
- **选择器 (Selectors)**：选择器指定要应用样式的 HTML 元素。常见的选择器有：
    - 元素选择器：选择所有指定类型的元素（如 `p`, `h1`, `div`）。`p { ... }` 会选中所有 `<p>` 元素。
    - 类选择器：选择带有特定 `class` 属性的元素（`class="my-class"`）。以 `.` 开头。`.my-class { ... }` 会选中所有 class 为 `my-class` 的元素。
    - ID 选择器：选择带有特定 `id` 属性的元素（`id="my-id"`）。以 `#` 开头。`#my-id { ... }` 会选中 id 为 `my-id` 的元素（在一个页面中 ID 应该是唯一的）。
    - 后代选择器：选择作为另一个元素后代的元素。`div p { ... }` 会选中所有在 `<div>` 内部的 `<p>` 元素。
    - 子元素选择器：选择作为另一个元素直接子元素的元素。`div > p { ... }` 会选中所有直接位于 `<div>` 下的 `<p>` 元素。

- **声明块 (Declaration Block)**：包含在花括号 `{}` 中，由一个或多个**声明**组成，声明之间用分号 `;` 分隔。
- **声明 (Declaration)**：由一个**属性** (`property`) 和一个**值** (`value`) 组成，之间用冒号 `:` 分隔。`property: value;` 例如 `color: blue;`, `font-size: 16px;`。

```css
/* 这是一个 CSS 规则 */
p { /* 选择器：选中所有 <p> 元素 */
    color: #333; /* 声明：设置文本颜色 */
    font-size: 16px; /* 声明：设置字体大小 */
    line-height: 1.5; /* 声明：设置行高 */
}

.button { /* 选择器：选中所有 class 为 button 的元素 */
    display: inline-block; /* 设置显示类型 */
    padding: 10px 20px; /* 设置内边距 */
    background-color: blue;
    color: white;
    text-decoration: none; /* 移除下划线 */
    border-radius: 5px; /* 圆角 */
}

#site-header { /* 选择器：选中 id 为 site-header 的元素 */
    background-color: #f0f0f0;
    padding: 20px;
}
```

- **引入 CSS**：
  - 内联样式：直接写在 HTML 标签的 `style` 属性里（不推荐用于大型项目）。`<p style="color: red;">红色文字</p>`
  - 内部样式表：写在 HTML `<head>` 标签内的 `<style>` 标签里。
  - 外部样式表：将 CSS 写在单独的 `.css` 文件中，然后在 HTML `<head>` 中使用 `<link>` 标签引入（最常见和推荐的方式）。 `<link rel="stylesheet" href="style.css">` WordPress 主题通常使用外部样式表。

在 WordPress 开发中理解 CSS 的意义：

- **主题样式**：WordPress 主题的核心部分就是 CSS。你需要编写 CSS 来控制网站的布局、颜色、字体等。
- **样式层叠与优先级**：理解 CSS 的"层叠"特性（多个规则作用于同一个元素时，哪个规则生效）和优先级规则（ID > Class > 元素 > 内联样式等）对于调试样式问题非常重要。
- **响应式设计**：使用 CSS 媒体查询 (`@media`) 可以让你的网站在不同设备（手机、平板、桌面）上显示不同的样式，实现响应式布局。

实践建议：

在第一部分创建的本地 WordPress 网站上，尝试修改当前主题的 `style.css` 文件（在 `wp-content/themes/当前主题名称/style.css`）。例如，找到 `body` 选择器，修改一下 `background-color` 或 `color`，刷新页面看效果。这能让你快速看到 CSS 的作用。

2.3. PHP 基础（核心）：动态生成内容

PHP (Hypertext Preprocessor) 是一种广泛使用的开源服务器端脚本语言，特别适合 Web 开发。**WordPress 就是用 PHP 写的。** 你的主题和插件代码绝大部分都将是 PHP。

- **为什么 WordPress 用 PHP?**
    - 历史原因：WordPress 诞生时，PHP 是 Web 开发领域最流行的语言之一。
    - 易于学习和使用（相对于某些其他后端语言）。
    - 强大的数据库支持：与 MySQL 配合默契。
    - 庞大的社区和丰富的资源。
    - 非常适合处理动态内容：从数据库读取内容，根据用户请求生成不同的页面。
- **PHP 代码在哪里运行?** PHP 代码在**服务器端**运行。当浏览器请求一个 `.php` 文件时，Web 服务器（如 Local 中的 Nginx 或 Apache）会将请求交给 PHP 解释器。PHP 解释器执行代码，处理逻辑，然后将**生成的文本输出**（通常是 HTML）发送回 Web 服务器，最终服务器将这些输出发送给浏览器。浏览器只接收和渲染 HTML、CSS、JS，它并不知道背后的 PHP 代码是什么样的。
- **PHP 代码块**：PHP 代码通常嵌入在 HTML 文件中（或者一个纯粹的 PHP 文件）。PHP 代码必须写在 `<?php` 和 `?>` 标记之间。

```
<!DOCTYPE html>
<html>
<head>
    <title>我的 PHP 网页</title>
</head>
<body>
    <h1>欢迎</h1>
    <p>
        <?php
        // 这是一个 PHP 代码块
        echo "当前日期是: " . date("Y-m-d"); // echo 用于输出文本
        ?>
    </p>
    <p>这是 HTML 内容，不受 PHP 标记影响。</p>
    <?php
    // 另一个 PHP 代码块
    $name = "访客"; // 定义一个变量
    echo "<p>你好, " . $name . "! </p>"; // 输出变量的值
    ?>
</body>
</html>
```

`<?php ... ?>` 之间的内容会被服务器上的 PHP 解释器执行。标记之外的内容会被 PHP 解释器忽略，直接作为文本（通常是 HTML）输出。

2.3.1. PHP 是什么？为什么 WordPress 用它？（已在上面解释）

2.3.2. PHP 的基本语法、变量与数据类型

- **语句结束符**：PHP 语句以分号 `;` 结束。

```
echo "Hello World";
echo "Another statement";
```

- **注释**:
  - 单行注释: `// 这是一行注释` 或 `# 这也是一行注释`
  - 多行注释: `/* 这是多行注释 */`
- **变量 (Variables)**: 变量用于存储信息。在 PHP 中，变量以 `$` 符号开头。变量名是大小写敏感的 (`$name` 和 `$Name` 是不同的变量)。

```
$greeting = "你好";
$user_name = "张三";
$age = 30;
$is_logged_in = true;
```

  变量在使用前不需要提前声明类型，PHP 会根据赋给它的值自动判断类型。
- **数据类型 (Data Types)**: PHP 支持多种数据类型:
  - 字符串 (String): 文本，用单引号或双引号括起来。`"Hello"`, `'World'`
  - 整数 (Integer): 非小数的数字。`10`, `-5`
  - 浮点数 (Float/Double): 带小数的数字。`3.14`, `-0.001`
  - 布尔值 (Boolean): 真或假。`true`, `false` (不带引号)
  - 数组 (Array): 存储一组有序或关联的值。`array(1, 2, 3)`, `array('key' => 'value')`
  - 对象 (Object): 类的实例。
  - NULL: 表示没有值。
  - 资源 (Resource): 表示外部资源（如数据库连接、文件句柄）。

2.3.3. 控制结构（条件判断与循环）

- **条件判断 (Conditional Statements)**: 根据条件执行不同的代码块。
  - `if` 语句:

```
$score = 85;
if ($score > 60) {
    echo "及格";
}
```

  - `if...else` 语句:

```
$age = 16;
if ($age >= 18) {
    echo "成年人";
} else {
    echo "未成年人";
}
```

- `if...elseif...else` 语句：

```
$grade = "B";
if ($grade == "A") {
    echo "优秀";
} elseif ($grade == "B") {
    echo "良好";
} elseif ($grade == "C") {
    echo "及格";
} else {
    echo "不及格";
}
```

- `switch` 语句：用于基于不同条件执行不同动作。

```
$day = "周一";
switch ($day) {
    case "周一":
        echo "上班";
        break; // 跳出 switch
    case "周六":
    case "周日": // 可以匹配多个值
        echo "休息";
        break;
    default:
        echo "其他日子";
}
```

- **循环 (Loops)**：重复执行一段代码块。
  - `while` 循环：当条件为真时重复执行。

```
$i = 1;
while ($i <= 5) {
    echo $i . "<br>"; // . 是字符串连接符
    $i++; // 记住更新条件，否则会无限循环
}
```

  - `for` 循环：已知循环次数时常用。

```
for ($j = 0; $j < 3; $j++) {
    echo "循环次数: " . $j . "<br>";
}
```

  - `foreach` 循环：专门用于遍历数组，在 WordPress 开发中非常常用。

```php
$colors = array("红", "绿", "蓝");
foreach ($colors as $color) {
    echo $color . "<br>";
}

$person = array(
    "name" => "张三",
    "age" => 30,
    "city" => "北京"
);
// 遍历关联数组，同时获取键和值
foreach ($person as $key => $value) {
    echo $key . ": " . $value . "<br>";
}
```

在 WordPress 的主题模板中，你会大量看到 `while` 和 `foreach` 循环，用来遍历文章列表、评论等。最著名的就是用来显示文章列表的 **The Loop**，它就是一个 `while` 循环。

2.3.4. 函数的使用与定义

- **使用函数 (Calling Functions)**：PHP 内置了成千上万的函数，WordPress 也提供了大量的函数（称为模板标签或 API 函数）。使用函数可以执行特定任务。

```php
echo strlen("Hello World!"); // 使用内置的 strlen() 函数获取字符串长度
echo date("Y-m-d H:i:s"); // 使用内置的 date() 函数获取当前日期时间

// 在 WordPress 中调用函数（这是 WordPress 提供的，不是 PHP 内置的）
the_title(); // 显示当前文章标题
the_content(); // 显示当前文章内容
wp_footer(); // 调用这个函数通常会输出一些代码到页面底部
```

调用函数时，需要在函数名后加括号 `()`，如果函数需要参数，则在括号内传入。

- **定义函数 (Defining Functions)**：你可以创建自己的函数来封装可重用的代码块。

```php
function greet($name) {
    echo "你好，" . $name . "!";
}

// 调用自定义函数
greet("李四"); // 输出：你好，李四!
greet("王五"); // 输出：你好，王五!

function add_numbers($num1, $num2) {
    $sum = $num1 + $num2;
    return $sum; // 使用 return 返回一个值
}
```

```php
$result = add_numbers(5, 3);
echo "<p>5 + 3 = " . $result . "</p>"; // 输出: 5 + 3 = 8
```

在 WordPress 主题的 `functions.php` 文件和插件中，你会定义很多自己的函数来添加或修改功能。

2.3.5. 数组与关联数组

数组 (Array) 是 PHP 中用于存储多个值在一个变量里的重要数据结构。

- **索引数组 (Indexed Arrays)**：使用数字索引（默认从 0 开始）访问元素。

```php
$fruits = array("苹果", "香蕉", "橙子");
// 或者更现代的写法
$fruits = ["苹果", "香蕉", "橙子"];

echo $fruits[0]; // 输出: 苹果
echo $fruits[1]; // 输出: 香蕉

// 修改元素
$fruits[1] = "葡萄";
echo $fruits[1]; // 输出: 葡萄

// 添加元素
$fruits[] = "西瓜"; // 在末尾添加
echo $fruits[3]; // 输出: 西瓜

print_r($fruits); // 打印数组的结构，方便调试
/* 输出类似:
Array
(
    [0] => 苹果
    [1] => 葡萄
    [2] => 橙子
    [3] => 西瓜
)
*/
```

- **关联数组 (Associative Arrays)**：使用字符串键 (key) 来访问元素，而不是数字索引。

```php
$student = array(
    "姓名" => "小明",
    "学号" => "2023001",
    "专业" => "计算机"
);
// 或者更现代的写法
$student = [
    "姓名" => "小明",
    "学号" => "2023001",
    "专业" => "计算机"
```

```php
];

echo $student["姓名"]; // 输出：小明
echo $student["专业"]; // 输出：计算机

// 修改元素
$student["专业"] = "软件工程";
echo $student["专业"]; // 输出：软件工程

// 添加元素
$student["班级"] = "1班";

print_r($student);
/* 输出类似：
Array
(
    [姓名] => 小明
    [学号] => 2023001
    [专业] => 软件工程
    [班级] => 1班
)
*/
```

关联数组在 WordPress 中非常常见，许多函数的参数和返回值都是关联数组，用来传递复杂的配置或数据。

2.3.6. 包含文件

在 PHP 中，你可以将代码分散到多个文件中，然后在一个主文件中包含 (include) 或要求 (require) 这些文件。这有助于组织代码，避免重复。

- `include 'path/to/file.php';`：包含指定文件。如果文件不存在或有错误，会产生一个警告 (warning)，但脚本会继续执行。
- `require 'path/to/file.php';`：要求包含指定文件。如果文件不存在或有错误，会产生一个致命错误 (fatal error)，并停止脚本执行。
- `include_once 'path/to/file.php';` 和 `require_once 'path/to/file.php';`：与 `include` 和 `require` 类似，但会检查文件是否已经被包含过，如果已经被包含，则不再包含。在 WordPress 中，为了避免重复定义函数或类导致的错误，`_once` 版本更常用。

在 WordPress 开发中理解包含文件的意义：

- **模板文件**：WordPress 主题就是通过 `include` 或 `require` 来加载不同的模板文件（如 `header.php`, `footer.php`, `sidebar.php`）来构建完整页面的。
- **组织代码**：在主题和插件开发中，你会把功能相似的代码放在不同的文件中（例如，把所有与小工具相关的代码放在一个文件里），然后在主文件或 `functions.php` 中 `include` 或 `require` 它们。

2.3.7. PHP 与 HTML 的结合

PHP 最常用于动态生成 HTML。你可以根据 PHP 逻辑输出不同的 HTML 内容。

- **输出 HTML**：使用 `echo` 或 `print` 语句输出 HTML 字符串。

```php
$is_admin = true;
if ($is_admin) {
    echo "<p>欢迎回来，管理员！</p>";
} else {
    echo "<p>欢迎，访客！</p>";
}
?>

<?php // PHP 代码块结束，回到 HTML 模式 ?>
<p>这段是静态 HTML。</p>

<?php // 再次进入 PHP 代码块 ?>
<?php
$user_count = 150;
?>
<p>我们目前有 <?php echo $user_count; ?> 位注册用户。</p> <!-- 在 HTML 中输出变量值 -->
```

注意在 PHP 标记内外切换。在 PHP 标记 `<?php ... ?>` 之外，所有内容都被视为纯文本（HTML）。在 PHP 标记之内，代码被执行，其 `echo` 或 `print` 的输出会成为 HTML 的一部分。

- **简写形式 (Short Echo Tag)**：对于简单的变量输出，可以使用简写形式 `<?= $variable ?>`，它等同于 `<?php echo $variable ?>`。

```php
<p>网站名称：<?= get_bloginfo('name'); ?></p> <!-- WordPress 函数 get_bloginfo() 返回网站名称 -->
```

这种简写形式在 WordPress 主题模板中非常常见，可以使代码更简洁。

2.4. JavaScript 基础（了解）：客户端交互

JavaScript 是一种运行在用户浏览器中的脚本语言。它主要负责：

- 响应用户操作（点击、悬停、键盘输入）。
- 动态修改网页内容（在不重新加载整个页面的情况下更新部分页面）。
- 与服务器进行异步通信（AJAX），例如在不刷新页面的情况下提交表单或加载更多内容。
- 创建复杂的动画和交互效果。
- **引入 JavaScript**：
  - 写在 `<script>` 标签内，通常放在 `<body>` 结束标签之前（以避免阻塞页面渲染）。
  - 链接外部 `.js` 文件：`<script src="script.js"></script>`
- **基本语法**：

```
// 这是一个 JavaScript 注释

// 变量声明
let message = "你好! ";
const button = document.getElementById("myButton"); // 获取 ID 为 myButton 的
元素

// 函数定义
function handleClick() {
    alert(message); // 弹出提示框
}

// 事件监听：当按钮被点击时，执行 handleClick 函数
button.addEventListener("click", handleClick);
```

JavaScript 语法与 PHP 有些相似（变量、函数、控制结构），但它们运行的环境和目的不同。

在 WordPress 开发中理解 JavaScript 的意义：

- **前端交互**：主题和插件可能会使用 JavaScript 来实现前端的动态效果，例如幻灯片、表单验证、无限滚动等。
- **WordPress 后台**：WordPress 后台使用了大量的 JavaScript (包括 jQuery, React, Backbone.js 等库) 来实现其丰富的交互功能（如古腾堡编辑器）。
- **enqueue 脚本**：在 WordPress 中，推荐使用 `wp_enqueue_script()` 函数来加载 JavaScript 文件，而不是直接在模板中写 `<script>` 标签。这有助于管理依赖关系和避免冲突。

对于入门阶段，你不需要精通 JavaScript，但需要知道它的作用以及如何在 WordPress 中引入和使用它（后面会讲到）。PHP 是你当前需要重点掌握的语言。

好的，基础的 Web 开发语言速成学习已经完成。虽然只是皮毛，但这为你理解接下来 WordPress 的工作原理打下了基础。现在，我们将深入 WordPress 的核心，探讨它的一些关键概念和内部机制。理解这些，就像获得了看懂 WordPress “内部运作图纸”的能力。

第三部分：WordPress 核心概念深入

本部分旨在解释 WordPress 如何处理请求、组织代码、以及它为何如此灵活和可扩展。这是理解主题和插件如何与核心程序交互的基础。

3.1. WordPress 请求生命周期（理解代码如何执行）

当你或任何访问者在浏览器中输入你的 WordPress 网站地址并回车时，幕后发生了一系列复杂但有规律的事情。理解这个“请求生命周期”有助于你追踪代码的执行流程。

这是一个简化的流程（省略了很多细节，只关注核心路径）：

1. **用户请求 (Browser Request)**：用户在浏览器中访问 `http://your-site-name.local/` 或 `http://your-site-name.local/hello-world/` 等 URL。
2. **Web 服务器处理 (Web Server - Nginx/Apache)**：Local 中的 Web 服务器（例如 Nginx）接收到这个请求。对于大多数 WordPress 网站的请求（非静态文件如图片、CSS），服务器会将请求指向网站根目录

下的 `index.php` 文件。

3. **执行 `index.php`**：WordPress 根目录下的 `index.php` 文件非常简单，它主要做一件事：

```php
<?php
/**
 * Front to the WordPress application. This file doesn't do anything, but
grabs
 * wp-blog-header.php to load the WordPress environment and template.
 *
 * @package WordPress
 */

/**
 * Tells WordPress to load the WordPress theme and output it.
 *
 * @var bool
 */
define( 'WP_USE_THEMES', true ); // 告诉 WordPress 我们要加载并使用主题

/** Loads the WordPress Environment and Template */
require __DIR__ . '/wp-blog-header.php'; // 包含并执行 wp-blog-header.php 文件
```

正如注释所说，`index.php` 几乎什么都不做，它只是定义了一个常量 `WP_USE_THEMES` 为 `true`（表示这是一个需要加载主题的前台请求），然后 `require` 了 `wp-blog-header.php` 文件。

4. **执行 `wp-blog-header.php`**：这个文件是加载 WordPress 核心环境的起点。它主要做两件事：
    - `require __DIR__ . '/wp-load.php';`：加载 `wp-load.php` 文件。这是整个 WordPress 加载过程中最重要的文件之一。
    - 调用 `wp()` 函数：这个函数是 WordPress 查询、处理请求和加载模板的核心调度者。
5. **执行 `wp-load.php`**：这个文件层层深入，最终加载了 `wp-config.php`（获取数据库信息等）、`wp-settings.php`（加载核心函数、类、设置、激活的插件等等）。至此，整个 WordPress 环境（包括数据库连接、用户认证、选项设置、所有核心功能、已激活的主题和插件）基本都被加载到内存中了。
6. **执行 `wp-settings.php`**：这是一个非常长的文件，它完成了大量初始化工作：定义常量、加载核心文件(`wp-includes` 目录下的文件)、加载并初始化已激活的插件、加载主题的 `functions.php` 文件等等。**这个阶段是很多钩子（Hooks）被触发的地方。**
7. **`wp()` 函数执行**：回到 `wp-blog-header.php` 中调用的 `wp()` 函数。这个函数：
    - 解析用户请求的 URL (确定用户想看的是文章、页面、分类列表、首页等)。
    - 根据解析结果查询数据库，获取相关内容（例如，如果是单篇文章页，就去数据库里找那篇文章的数据）。
    - 设置全局变量（例如 `$post` 存储当前文章数据）。
    - 决定应该加载哪个主题模板文件来显示内容（根据模板层级结构，后面会讲）。
8. **加载主题模板文件 (Template Loading)**：`wp()` 函数内部或紧随其后（通过 `template-loader.php`，它由 `wp-settings.php` 加载）会根据第7步决定的模板文件路径，使用 `include` 或 `require` 函数加载相应的主题模板文件（例如 `index.php`, `single.php`, `page.php` 等）。
9. **执行主题模板文件**：加载的模板文件是 PHP 文件，它们通常包含 HTML 结构和大量的 WordPress PHP 函数（称为模板标签，如 `the_title()`, `the_content()`）。这些函数会从之前查询到的数据中提取信息，并使用 `echo` 输出 HTML 内容。**The Loop（循环）**就在这个阶段执行，负责迭代并显示多篇文章（如在首页或归档页）。

10. **生成最终 HTML**：主题模板文件的执行产生最终的 HTML、CSS、JavaScript 代码。
11. **服务器响应 (Server Response)**：Web 服务器将生成的 HTML 等内容发送回用户的浏览器。
12. **浏览器渲染 (Browser Rendering)**：用户的浏览器接收到内容，解析 HTML，应用 CSS 样式，执行 JavaScript 代码，最终将网页显示给用户。

总结这个流程：

`index.php` -> `wp-blog-header.php` -> `wp-load.php` -> `wp-config.php` (配置) -> `wp-settings.php` (加载核心、插件、主题 functions) -> `wp()` (解析请求、查询数据、决定模板) -> 加载主题模板文件 -> 执行模板文件 (包含 **The Loop**, 调用模板标签) -> 生成 HTML -> 发送给浏览器 -> 浏览器显示。

理解这个流程是至关重要的，因为它解释了为什么你修改 `wp-content/themes/你的主题/index.php` 或 `functions.php` 文件会影响网站，而修改 `wp-includes` 里的文件通常是禁止的。你的主题和插件代码都是在 `wp-settings.php` 加载后，在 `wp()` 函数处理请求的过程中，或者在主题模板文件执行时被调用的。

3.2. WordPress 核心机制：动作（Actions）与过滤（Filters）—— 钩子（Hooks）详解

WordPress 的强大扩展性很大程度上归功于其**钩子 (Hooks)** 系统。钩子是 WordPress 核心、主题和插件之间进行通信和交互的机制。通过钩子，你可以在不修改核心文件的情况下，在特定的时刻运行你的自定义代码（**动作 - Actions**），或者修改 WordPress 处理中的数据（**过滤 - Filters**）。

可以把钩子想象成 WordPress 执行流程中的一些“插槽”或“事件点”。当代码执行到某个点时，如果那里有一个钩子，WordPress 会查找是否有函数“挂载”到了这个钩子上，并执行这些函数。

钩子分为两种主要类型：

1. **动作 (Actions)**：
    - **用途**：在 WordPress 执行过程中的**特定事件发生时**执行你的自定义函数。动作不会修改数据，它们只是在某个时间点“做一些事情”。
    - **核心触发**：WordPress 核心代码在执行到某个特定点时，会调用 `do_action('钩子名称', [参数1, 参数2, ...]);` 来“触发”一个动作钩子。例如，在执行完主题的 `functions.php` 文件后，核心会触发 `after_setup_theme` 动作；在页面的 `<body>` 标签关闭之前，会触发 `wp_footer` 动作。
    - **你如何使用**：你通过 `add_action('钩子名称', '你的函数名', [优先级], [接受参数数量]);` 函数，将你的自定义 PHP 函数“挂载”到指定的动作钩子上。当这个钩子被触发时，你的函数就会被执行。
    - **移除钩子**：如果你想移除其他主题或插件挂载到某个钩子上的函数，可以使用 `remove_action('钩子名称', '要移除的函数名', [优先级]);`。

    动作示例 (简单，你将在 `functions.php` 或插件文件中写这样的代码)：

    假设你想在网站头部输出一段问候语：

```
// 将你的函数挂载到 wp_head 动作钩子上
add_action('wp_head', 'my_custom_greeting');

// 定义你的自定义函数
function my_custom_greeting() {
    // 可以在这里写 PHP 代码，并使用 echo 输出 HTML 或其他内容
```

```
    echo '<meta name="my-custom-meta" content="这是我通过钩子添加的内容">';
    echo '<!-- 欢迎来到我的网站!  -->';
}

// wp_head 是一个在主题的 header.php 文件中调用 do_action('wp_head'); 时触发的
钩子
// 它通常位于 <head> 标签内部，在加载 CSS 和 JS 之前或之后
```

2. **过滤 (Filters)**：

   - **用途**：在 WordPress 处理**数据**的过程中，**修改或过滤**这些数据。过滤总是会接收一个值，并期望你的函数返回一个值（通常是修改后的）。
   - **核心触发**：WordPress 核心代码在准备使用或返回某个数据之前，会调用 `apply_filters('钩子名称', $value, [参数1, 参数2, ...]);` 来“应用”一个过滤钩子。例如，在显示文章标题之前，核心会应用 `the_title` 过滤；在保存文章内容到数据库之前，会应用 `content_save_pre` 过滤。
   - **你如何使用**：你通过 `add_filter('钩子名称', '你的函数名', [优先级], [接受参数数量]);` 函数，将你的自定义 PHP 函数“挂载”到指定的过滤钩子上。你的函数将接收到原始数据（通常作为第一个参数），你可以处理它，然后必须 `return` 修改后的数据。
   - **移除钩子**：使用 `remove_filter('钩子名称', '要移除的函数名', [优先级]);`。

   过滤示例 (简单，你将在 `functions.php` 或插件文件中写这样的代码)：

   假设你想在所有文章标题末尾添加一段文字：

```
// 将你的函数挂载到 the_title 过滤钩子上
add_filter('the_title', 'add_suffix_to_title', 10, 2); // 优先级 10，接受 2 个
参数

// 定义你的自定义函数
function add_suffix_to_title($title, $id) {
    // $title 是原始的文章标题（第一个参数）
    // $id 是文章 ID（第二个参数）

    // 如果文章标题为空，不做处理（例如在首页或一些特殊地方标题可能为空）
    if (empty($title)) {
        return $title;
    }

    // 返回修改后的标题
    return $title . ' - 我的博客';
}

// the_title 钩子在 WordPress 显示文章标题时被触发
// 你的函数会在标题被输出到页面之前修改它
```

动作与过滤的关键区别：

- **目的不同**：动作是“做某事”，过滤是“修改数据”。

- **返回值**：动作函数通常不需要返回值（或者返回 `void`），过滤函数**必须返回**处理后的值。

优先级 (Priority)：当你使用 `add_action` 或 `add_filter` 时，可以指定一个可选的优先级数字（默认为 10）。优先级数字越小，函数执行得越早。如果多个函数挂载到同一个钩子且优先级相同，它们将按照挂载的顺序执行。

接受参数数量 (Accepted Args)：通过 `add_action` 或 `add_filter` 的第四个参数，你可以告诉 WordPress 你的函数需要接收几个参数。虽然钩子可能触发时提供了多个参数，但你的函数只会接收你指定数量的参数。如果你需要使用钩子提供的额外参数（比如 `the_title` 的 `$id`），你需要正确设置这个参数数量。

理解钩子系统的重要性：

- **扩展性**：它是 WordPress 生态系统的基石，让你可以在不修改核心代码的情况下，极大地扩展和修改 WordPress 的功能和外观。
- **阅读源码**：当你阅读 WordPress 核心、主题或插件代码时，你会看到大量的 `do_action()` 和 `apply_filters()` 调用。这些就是钩子点，你知道你可以在这些地方插入自己的代码。同样，你会看到大量的 `add_action()` 和 `add_filter()` 调用，这些是其他开发者"挂载"代码到钩子上的地方，通过这些你可以理解他们的代码是如何被触发和如何工作的。
- **编写代码**：你需要使用 `add_action()` 和 `add_filter()` 将你的主题或插件功能与 WordPress 的核心流程集成。

3.3. WordPress 主题系统：模板层级结构

WordPress 主题负责网站的**外观**。一个主题是一系列文件（主要是 PHP 模板文件、CSS、JavaScript、图片）的集合，存放在 `wp-content/themes/` 目录下的一个独立文件夹中。

当用户请求一个页面时，WordPress 会根据用户请求的类型（例如，是首页、单篇文章、某个分类的列表、还是一个独立的页面）以及你的主题中存在哪些文件，按照一个预设的**模板层级结构 (Template Hierarchy)** 来决定加载哪个 PHP 模板文件来生成页面。

这个层级结构就像一个决策树。WordPress 从最具体、最高优先级的模板文件开始查找，如果找到就使用它；如果找不到，就查找下一个优先级较低的文件，直到找到一个匹配的或者回退到最通用的 `index.php` 文件。

简化的模板层级示例 (核心部分)：

- **首页 (Homepage)**：
    1. `front-page.php` (如果设置了静态首页)
    2. `home.php` (如果设置为显示最新文章)
    3. `index.php` (回退)
- **单篇文章 (Single Post)**：
    1. `single-{post_type}-{slug}.php` (非常具体的，如 `single-post-hello-world.php`)
    2. `single-{post_type}.php` (按文章类型，如 `single-post.php`)
    3. `single.php` (所有单篇文章)
    4. `index.php` (回退)
- **单个页面 (Single Page)**：
    1. `custom-template.php` (自定义页面模板，你在后台编辑页面时可以选择)
    2. `page-{slug}.php` (按页面 slug，如 `page-about-us.php`)
    3. `page-{id}.php` (按页面 ID，如 `page-10.php`)
    4. `page.php` (所有独立页面)
    5. `index.php` (回退)

- **分类归档页 (Category Archive)**:
    1. `category-{slug}.php` (按分类 slug)
    2. `category-{id}.php` (按分类 ID)
    3. `category.php` (所有分类归档)
    4. `archive.php` (所有归档类型，包括标签、日期、作者等)
    5. `index.php` (回退)
- **标签归档页 (Tag Archive)**:
    1. `tag-{slug}.php`
    2. `tag-{id}.php`
    3. `tag.php`
    4. `archive.php`
    5. `index.php`
- **日期归档页 (Date Archive)**:
    1. `date.php`
    2. `archive.php`
    3. `index.php`
- **作者归档页 (Author Archive)**:
    1. `author-{nicename}.php`
    2. `author-{id}.php`
    3. `author.php`
    4. `archive.php`
    5. `index.php`
- **搜索结果页 (Search Results)**:
    1. `search.php`
    2. `index.php`
- **404 错误页 (404 Not Found)**:
    1. `404.php`
    2. `index.php`

这意味着什么?

- 如果你创建了一个 `single.php` 文件，所有单篇文章都会使用它来显示。
- 如果你想给某个特定分类（例如 ID 为 5 的分类）一个独特的布局，你可以创建一个 `category-5.php` 文件，WordPress 会优先使用它而不是通用的 `category.php` 或 `archive.php`。
- `index.php` 是最重要的回退文件。理论上，一个主题可以只有一个 `index.php` 和 `style.css` 文件就能运行（虽然功能会非常有限）。

主题文件的基本构成（至少需要的文件）：

- `style.css`：**必需。** 这是主题的样式文件，同时也包含主题的元信息（主题名称、作者、版本等），WordPress 通过这些信息识别主题。
- `index.php`：**必需（作为回退）。** 这是主要的模板文件，在没有更具体的模板文件时用于显示各种内容。

常用的其他主题文件:

- `header.php`：通常包含 HTML 文档的开头部分（`<!DOCTYPE html>`, `<html>`, `<head>`, 开启 `<body>` 标签）。在其他模板文件中通常用 `get_header()` 函数来包含它。

- `footer.php`：通常包含 HTML 文档的结尾部分（关闭 `<body>` 和 `<html>` 标签）。在其他模板文件中通常用 `get_footer()` 函数来包含它。
- `sidebar.php`：通常包含侧边栏的内容。用 `get_sidebar()` 函数包含。
- `functions.php`：这是一个非常重要的文件，用于添加主题特有的功能、注册菜单/侧边栏、定义图片尺寸、使用钩子等。它在 WordPress 加载主题时自动被加载和执行。
- 各种模板文件 (`single.php`, `page.php`, `archive.php`, `category.php`, etc.)：根据模板层级结构决定特定类型页面的布局。

当你阅读一个主题的代码时，首先看 `style.css` (了解主题信息和基础样式)，然后看 `index.php` 和 `functions.php`，再根据你想理解的页面类型去查找对应的模板文件（如单篇文章就看 `single.php`）。你会看到模板文件里包含 HTML 结构和大量的 WordPress PHP 函数。

3.4. WordPress 插件系统：工作原理

WordPress 插件用于向网站添加**功能**或修改现有功能。一个插件通常是一个或多个 PHP 文件的集合（也可能包含 CSS, JS, 图片等），存放在 `wp-content/plugins/` 目录下的一个独立文件夹中。

- **插件的识别**：WordPress 通过读取插件主文件（通常是与插件文件夹同名的 PHP 文件）顶部的特殊注释块来识别一个插件。这个注释块必须包含 `Plugin Name:` 行。

```php
<?php
/**
 * Plugin Name: 我的第一个插件
 * Description: 这是一个用于演示的简单插件。
 * Version: 1.0
 * Author: 你的名字
 * License: GPL2
 */

// 插件的代码从这里开始
// ...
?>
```

- **插件的加载**：当你在 WordPress 后台激活一个插件时，WordPress 会记住它。在每次页面加载的早期阶段（在 `wp-settings.php` 文件中），WordPress 会检查哪些插件被激活了，并使用 PHP 的 `include_once` 或 `require_once` 函数加载它们的 PHP 文件。这意味着插件的代码在主题文件被加载之前，甚至在许多核心函数被定义之后就已经可用了。
- **插件如何工作**：插件通过**钩子 (Hooks)** 与 WordPress 核心和主题进行交互。它们使用 `add_action()` 和 `add_filter()` 将自己的函数挂载到 WordPress 核心或主题触发的钩子上，从而在特定的时刻执行代码或修改数据。插件**不应该**直接修改 WordPress 核心文件。
- **插件的生命周期**：插件也有激活 (activation) 和禁用 (deactivation) 的概念。你可以在插件主文件中定义激活函数 (`register_activation_hook()`) 和禁用函数 (`register_deactivation_hook()`)，在插件被激活或禁用时执行一些初始化或清理工作（例如创建数据库表）。

理解插件系统的重要性：

- **添加功能**：插件是向 WordPress 网站添加任何非核心功能的标准方式。
- **模块化**：插件鼓励将功能模块化，与主题（负责外观）分离，使得功能可以在不同主题之间重用。

- **不修改核心**：通过钩子机制，插件可以在不触碰 WordPress 核心文件的情况下扩展功能，这使得更新 WordPress 核心变得容易，因为你的修改不会被覆盖。

当你阅读一个插件的代码时，首先看主 PHP 文件顶部的注释块（了解插件信息），然后看文件中使用了哪些 `add_action()` 和 `add_filter()` 调用，这能让你快速了解这个插件在哪些地方"做了手脚"。

3.5. 理解 The Loop (循环)：WordPress 内容展示的核心

The Loop (循环) 是 WordPress 主题中用于显示文章（或其他文章类型，如页面、自定义文章类型）列表或单篇文章内容的核心代码结构。它通常出现在主题的模板文件中（如 `index.php`, `archive.php`, `single.php`, `page.php` 等）。

The Loop 的目的是什么?

在列表页（如首页、分类归档页），WordPress 查询会返回多篇文章。The Loop 的作用就是**遍历**这些文章，在每次迭代中加载一篇文章的数据，然后允许主题使用**模板标签**来显示当前文章的标题、内容、日期、作者等信息。

在单篇文章页，查询通常只返回一篇文章，但 The Loop 仍然被使用，只是它只执行一次迭代。

The Loop 的基本结构 (使用 `while` 循环):

```
<?php
if ( have_posts() ) { // 检查是否有文章可以显示
    while ( have_posts() ) { // 开始循环，只要还有文章就继续
        the_post(); // 加载当前文章的数据，使其可用

        // 在这里写显示文章内容的 HTML 和 PHP 代码（模板标签）
        ?>
        <article id="post-<?php the_ID(); ?>" <?php post_class(); ?>> <!-- 使用模
板标签输出文章 ID 和 CSS 类 -->
            <h2><a href="<?php the_permalink(); ?>"><?php the_title(); ?></a></h2>
<!-- 输出文章标题和链接 -->
            <div class="entry-meta">
                <?php the_date(); ?> by <?php the_author(); ?> <!-- 输出日期和作者
-->
            </div>
            <div class="entry-content">
                <?php the_content(); ?> <!-- 输出文章内容 -->
            </div>
            <?php
            // 如果是单篇文章页，可以在这里显示评论等
            if ( is_single() ) {
                comments_template();
            }
            ?>
        </article>
        <?php

    } // 循环结束
} else {
    // 如果没有找到文章，显示一个提示信息
```

```
        ?>
        <p>抱歉，没有找到相关内容。</p>
        <?php
    }
    ?>
```

理解 The Loop 中用到的关键函数：

- `have_posts()`：这是一个条件函数。它检查当前的 WordPress 查询结果中是否还有下一篇文章可以处理。在 `if` 和 `while` 语句中都使用它。
- `the_post()`：这是一个核心函数。它"设置"当前的全局文章变量 (`$post`)，使得你可以使用其他的**模板标签**来获取和显示当前文章的数据。每次循环迭代时都必须调用它。
- **模板标签 (Template Tags)**：这些是 WordPress 提供的 PHP 函数，它们通常用于在主题模板文件中，特别是在 The Loop 内部，输出特定的数据。
    - `the_title()`：输出当前文章的标题。
    - `the_content()`：输出当前文章的内容。
    - `the_excerpt()`：输出当前文章的摘要。
    - `the_permalink()`：输出当前文章的永久链接 URL。
    - `the_ID()`：输出当前文章的 ID。
    - `the_date()` / `the_time()` / `the_modified_date()`：输出文章的日期/时间。
    - `the_author()`：输出文章作者的名称。
    - `the_category()`：输出文章所属分类的链接列表。
    - `the_tags()`：输出文章所属标签的链接列表。
    - `post_class()`：输出用于当前文章 `<article>` 或 `<div>` 元素的 CSS 类列表（非常有用，方便用 CSS 定位和样式化不同类型的文章）。
    - `body_class()`：类似 `post_class()`，用于 `<body>` 标签，输出页面类型的 CSS 类。
    - `wp_head()`：调用 `do_action('wp_head')`，用于在 `<head>` 标签内输出内容（主题和插件常在此挂载脚本、样式或其他元信息）。
    - `wp_footer()`：调用 `do_action('wp_footer')`，用于在 `</body>` 标签结束前输出内容（主题和插件常在此挂载脚本或其他需要在页面底部执行的代码）。
    - `get_header()` / `get_footer()` / `get_sidebar()`：用于包含主题的其他模板文件。
    - `bloginfo()` / `get_bloginfo()`：输出或返回网站的基本信息（如网站标题、描述、主题目录 URL 等）。
    - `home_url()`：输出网站首页的 URL。
    - `get_template_directory_uri()`：返回当前主题目录的 URL，常用于链接主题文件中的图片、CSS、JS。

理解 The Loop 和模板标签的重要性：

- **内容展示核心**：几乎所有显示文章/页面内容的地方都会用到 The Loop。
- **驱动模板**：模板文件的主要工作就是设置 The Loop 的结构，并在循环内调用模板标签来填充动态内容。
- **阅读主题**：当你阅读主题文件时，你会看到大量的模板标签，知道它们的作用就能理解这段代码在页面上显示了什么。

好的，第三部分关于 WordPress 核心概念的介绍到这里。我们深入了解了请求生命周期、钩子系统（动作与过滤），以及主题的模板层级和内容展示核心 The Loop。这些都是理解 WordPress 内部运作机制的关键。

本部分的实践作业：

1. **观察请求生命周期**：虽然我们不能“看”到服务器内部的执行过程，但你可以通过在 `wp-config.php` 文件中开启调试模式来观察一些错误或警告。找到 `define('WP_DEBUG', false);` 这一行，将其改为 `define('WP_DEBUG', true);`。如果网站出现问题，将会在页面上显示错误信息，这有助于你理解执行到哪个文件或哪一行出了问题。**注意：调试模式只在本地环境开启，线上环境务必关闭！**
2. **查找钩子**：在 Local 的 WordPress 网站根目录的文件中（特别是 `wp-settings.php`，或者一些核心文件），尝试搜索 `do_action(` 和 `apply_filters(`，初步感受一下 WordPress 在哪些地方预留了扩展点。
3. **查找模板文件**：浏览你的主题文件夹 (`wp-content/themes/你的主题`)，识别出 `index.php`, `header.php`, `footer.php`, `single.php`, `page.php` 等文件。尝试通过 Local 后台修改首页显示设置（设置为显示最新文章或静态页面），观察前台 URL，并根据模板层级结构推测 WordPress 使用了哪个模板文件来渲染首页。
4. **在主题文件中找到 The Loop**：打开你主题的 `index.php` 或 `single.php` 文件，找到 `if ( have_posts() )... while ( have_posts() )... the_post()... endwhile; endif;` 这样的结构，识别出 The Loop。观察在 The Loop 内部使用了哪些模板标签。

好的，让我们进入实践阶段！现在你对 WordPress 的基本构成、请求流程和钩子系统有了初步了解，并且掌握了基础的 HTML, CSS, PHP 知识。是时候把这些结合起来，从零开始构建一个 WordPress 主题了。

这将是一个功能非常基础的主题，但它将包含一个标准 WordPress 主题的核心组成部分，并让你理解模板文件如何协同工作。

第四部分：WordPress 主题开发实战（从零开始）

本部分的学习目标是让你亲手创建一个 WordPress 主题，理解其基本结构和工作流程，并学会使用常用的模板标签和功能。

4.1. 主题文件的基本构成 (`style.css`, `index.php`, `functions.php`)

一个 WordPress 主题至少需要两个文件才能被 WordPress 识别并激活：

1. **`style.css`**：这是主题的主要样式表文件。除了包含主题的 CSS 样式外，它**必须**包含一个特殊的注释头部，用来告诉 WordPress 这个主题的名字、作者、版本等信息。没有这个头部，WordPress 就不会在后台“外观”->“主题”列表中显示你的主题。
2. **`index.php`**：这是主题的**主模板文件**，也是最低优先级、最后回退的模板文件（根据模板层级结构）。理论上，一个主题可以只有 `index.css` 和 `index.php` 就能工作，尽管它会用同样的方式显示所有类型的页面。

除了这两个必需文件，一个稍微完整的主题通常还会包含：

- **`functions.php`**：用于添加主题特定的功能、注册菜单/侧边栏、引入脚本样式、使用钩子等。这是一个非常重要的文件，它在主题被激活时自动被 WordPress 加载执行。
- **`header.php`**：包含 HTML 文档的 `<head>` 部分和 `<body>` 标签的起始部分，以及网站头部内容（如 Logo, 导航）。
- **`footer.php`**：包含 HTML 文档 `</body>` 标签的结束部分以及网站底部内容（如版权信息）。
- **`sidebar.php`**：包含侧边栏内容（如小工具区域）。
- 各种**其他模板文件**（如 `single.php`, `page.php`, `archive.php`, `comments.php` 等）：根据模板层级结构用于显示特定类型的内容。

我们将从最基本的 `style.css` 和 `index.php` 开始，然后逐步添加其他文件。

4.2. 创建一个最小化的 WordPress 主题

让我们在 Local 环境中为你创建的 WordPress 网站里，亲手创建一个新的主题文件夹和必要文件。

步骤：

1. **找到主题目录**：打开你的 Local 网站。点击网站信息页上的 "Go to Site Folder"。进入 `app/public/wp-content/themes/` 目录。你会看到一些预装的主题文件夹（如 `twentytwentythree`, `twentytwentyfour` 等）。
2. **创建一个新的主题文件夹**：在这个 `themes` 目录里，创建一个新的文件夹，用来存放你的主题文件。文件夹名称是你主题的**唯一标识符**（slug），建议使用小写字母和连字符，并且不要包含空格或特殊字符。例如，创建一个名为 `my-first-theme` 的文件夹。
3. **创建 `style.css` 文件**：在 `my-first-theme` 文件夹内，创建一个名为 `style.css` 的新文件。使用你喜欢的代码编辑器（推荐使用功能丰富的编辑器如 VS Code, Sublime Text, Atom 等）。
4. **添加主题头部信息**：打开 `style.css` 文件，在文件的最顶部添加以下注释块。**请务必包含 `Theme Name:` 这一行。**

```
/*
 Theme Name: My First Theme
 Theme URI: https://your-website.com/my-first-theme/ (可选，主题主页)
 Author: Your Name (你的名字)
 Author URI: https://your-website.com/ (可选，作者主页)
 Description: 这是一个从零开始构建的第一个WordPress主题教程示例。
 Version: 1.0
 License: GNU General Public License v2 or later
 License URI: http://www.gnu.org/licenses/gpl-2.0.html
 Text Domain: my-first-theme (用于国际化/翻译，通常与主题文件夹名相同)
 Tags: blog, one-column, custom-colors, custom-menu (可选，用于WP主题目录筛选)
*/

/* 在这里开始写你的 CSS 样式 */
body {
    background-color: #f0f0f0;
    font-family: sans-serif;
}
```

   保存 `style.css` 文件。
5. **创建 `index.php` 文件**：在 `my-first-theme` 文件夹内，创建一个名为 `index.php` 的新文件。
6. **添加最基本的 PHP 代码**：打开 `index.php` 文件，添加以下代码：

```
<!DOCTYPE html>
<html <?php language_attributes(); ?>> <!-- 输出 HTML 语言属性，如 lang="zh-CN" -->
<head>
    <meta charset="<?php bloginfo('charset'); ?>"> <!-- 输出网站字符编码 -->
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title><?php wp_title('|', true, 'right'); bloginfo('name'); ?></title>
```

```
<!-- 输出页面标题 -->
    <link rel="stylesheet" href="<?php echo get_stylesheet_uri(); ?>"> <!--
引入 style.css -->
    <?php wp_head(); // 这是必须的！用于输出插件和核心在 <head> 中需要的内容，特
别是 wp_head 钩子 ?>
</head>
<body <?php body_class(); ?>> <!-- 输出 body 元素的 CSS 类 -->

    <h1><?php bloginfo('name'); ?></h1> <!-- 输出网站标题 -->
    <p><?php bloginfo('description'); ?></p> <!-- 输出网站描述 -->

    <?php
    // The Loop (后面会详细实现)
    if ( have_posts() ) {
        while ( have_posts() ) {
            the_post();
            ?>
            <h2><?php the_title(); ?></h2>
            <div><?php the_content(); ?></div>
            <hr>
            <?php
        }
    } else {
        ?>
        <p>没有找到文章。</p>
        <?php
    }
    ?>

    <?php wp_footer(); // 这是必须的！用于输出插件和核心在 </body> 结束前需要的内
容，特别是 wp_footer 钩子 ?>
</body>
</html>
```

保存 `index.php` 文件。

7. **激活主题**：打开你的 Local WordPress 网站后台（通过 Local 点击 "WP Admin"）。登录后，导航到“外观”->“主题”。你应该能在主题列表中看到一个名为 "My First Theme" 的主题，带有你设置的作者、描述等信息。点击“启用”按钮。
8. **查看网站**：主题启用后，访问你的网站前台（通过 Local 点击 "Open Site"）。你应该能看到一个非常简陋的页面，显示了网站标题、描述，以及你发布的所有文章的标题和内容（如果文章存在的话）。页面的背景颜色应该是你在 `style.css` 中设置的颜色。

恭喜！你已经成功创建并激活了你的第一个 WordPress 主题。虽然它看起来很简单，但这为你后续的开发打下了基础。

在 `index.php` 文件中，你已经看到了 PHP 代码、HTML 结构，以及一些 WordPress 特有的 PHP 函数，例如 `language_attributes()`, `bloginfo()`, `get_stylesheet_uri()`, `wp_head()`, `body_class()`, `have_posts()`, `the_post()`, `the_title()`, `the_content()`, `wp_footer()`。这些函数被称为**模板标签**，它们是 WordPress 提供给主题开发者用于获取和显示数据的便捷方式。

特别注意 `wp_head()` 和 `wp_footer()`。它们是分别触发 `wp_head` 和 `wp_footer` **动作钩子**的函数。WordPress 核心和许多插件会通过 `add_action()` 函数将自己的代码挂载到这两个钩子上，以便在页面的 `<head>` 或 `</body>` 结束前输出必要的样式表、脚本或元信息。因此，在你的主题中包含 `<?php wp_head(); ?>` 和 `<?php wp_footer(); ?>` 是**非常重要**的，否则一些插件可能无法正常工作。

`get_stylesheet_uri()` 函数用于获取当前主题的 `style.css` 文件的 URL。这是引入主题主要样式表的标准方法。

4.3. 使用 The Loop 显示文章列表

我们在上面的 `index.php` 中已经包含了一个简单的 The Loop 结构。现在我们来详细看看它。

```
        <?php
        // 1. 检查是否有文章可以显示
        if ( have_posts() ) {

            // 2. 开始循环，只要还有文章就继续
            while ( have_posts() ) {

                // 3. 加载当前文章的数据
                the_post();

                // 4. 在这里编写显示当前文章的 HTML 和调用模板标签的代码
                ?>
                <article id="post-<?php the_ID(); ?>" <?php post_class(); ?>> <!--
article 是 HTML5 语义标签，post_class() 输出文章相关的 CSS 类 -->
                    <h2><a href="<?php the_permalink(); ?>"><?php the_title(); ?>
</a></h2> <!-- 显示文章标题，并链接到单篇文章页 -->
                    <div class="entry-meta">
                        Posted on <?php the_date(); ?> by <?php the_author(); ?>
<!-- 显示发布日期和作者 -->
                    </div>
                    <div class="entry-content">
                        <?php the_content(); ?> <!-- 显示文章的完整内容 -->
                        <?php // 如果你想显示摘要而不是完整内容，可以使用
the_excerpt(); ?>
                    </div>
                </article>
                <?php

            } // 循环结束：endwhile; 也可以用 while(...) : ... endwhile; 这种替代语
法
        } else {
            // 5. 如果没有找到文章
            ?>
            <p>抱歉，没有找到相关内容。</p>
            <?php
        }
        ?>
```

解释：

- `if ( have_posts() )`: 这是 The Loop 的第一部分。`have_posts()` 函数会检查当前 WordPress 查询（根据用户请求的 URL，WordPress 在后台已经执行了数据库查询并获取了相关文章数据）是否有返回结果，即是否有文章可以用来显示。如果有，条件为真，进入 `if` 代码块。
- `while ( have_posts() )`: 这是 The Loop 的核心循环。只要 `have_posts()` 返回真（表示还有下一篇文章），循环就会继续。
- `the_post()`: 在 `while` 循环的**每次迭代**开始时，你**必须**调用 `the_post()` 函数。这个函数会将当前迭代的文章数据加载到 WordPress 的全局 `$post` 变量中。只有调用了 `the_post()`，后面的**模板标签**（如 `the_title()`, `the_content()` 等）才知道要显示哪篇文章的数据。
- 循环内部：在 `while` 循环的 `{ ... }` 内部，你可以自由地混合使用 HTML 和 PHP 模板标签来构建每篇文章的显示结构。每次循环，这里面的代码都会针对当前通过 `the_post()` 加载的文章执行一次。
- 模板标签：`the_ID()`, `the_permalink()`, `the_title()`, `the_date()`, `the_author()`, `the_content()`, `post_class()` 都是在循环内部非常常用的模板标签，用于输出当前文章的特定信息或属性。注意，像 `the_title()` 这样的函数是**直接输出**结果的 (使用了 `echo` 或 `print` 在内部)，而像 `get_the_title()` 这样的函数是**返回**结果，需要你手动 `echo` 或处理。在模板中，通常直接使用 `the_` 开头的输出函数更方便。
- `else` 部分：如果在 `if ( have_posts() )` 检查时发现没有文章，就会进入 `else` 代码块，你可以在这里显示一个“没有找到内容”的提示信息。

实践：

确保你的 `index.php` 文件包含上述 The Loop 结构，并在 Local 网站中创建几篇测试文章。访问网站首页，观察它们是否正确显示。尝试修改 `the_content()` 为 `the_excerpt()`，然后刷新首页，看看文章内容是否变成了摘要。

4.4. 常用的模板标签（Template Tags）介绍与使用

模板标签是 WordPress 开发者的利器，它们是预定义的 PHP 函数，用于在主题文件中获取和显示各种动态数据。我们已经在 The Loop 中看到了一些，这里列举更多常用的（以及一些非 Loop 相关的）：

在 The Loop 内部常用：

- `the_ID()`: 显示当前文章的 ID。
- `the_title( string $before = '', string $after = '', bool $echo = true )`: 显示当前文章的标题。`$before` 和 `$after` 参数可以在标题前后添加文本/HTML。`$echo` 参数控制是直接输出 (true) 还是返回字符串 (false)。
- `get_the_title( int|WP_Post $post = 0 )`: 返回当前文章的标题字符串，不直接输出。
- `the_content( string $more_link_text = null, bool $strip_teaser = false )`: 显示当前文章的完整内容。处理“阅读更多”标签。
- `get_the_content( string $more_link_text = null, bool $strip_teaser = false, int|WP_Post $post = 0 )`: 返回当前文章的完整内容字符串。
- `the_excerpt()`: 显示当前文章的摘要（如果你手动写了摘要就显示，否则自动生成一部分内容）。
- `get_the_excerpt( int|WP_Post $post = 0 )`: 返回当前文章的摘要字符串。
- `the_permalink()`: 显示当前文章的永久链接 URL。
- `get_permalink( int|WP_Post $post = 0, bool $leavename = false )`: 返回当前文章的永久链接 URL 字符串。
- `the_post_thumbnail( string|array $size = 'post-thumbnail', string|array $attr = '' )`: 如果文章设置了特色图片，显示它。需要主题支持特色图片（后面 `functions.php` 会讲）。`$size` 参数控制图片尺寸。

- `post_class( string|array $class = '', int|WP_Post $post = 0 )`: 输出用于文章容器元素的 CSS 类列表，包含很多有用的类（如文章类型、分类、标签等）。
- `comment_form()`: 显示评论表单。
- `comments_template( string $file = '/comments.php', bool $separate_comments = false )`: 包含并显示评论列表和评论表单。通常用在 `single.php` 和 `page.php` 中。

在 The Loop 外部或全局常用：

- `bloginfo( string $show = '', string $filter = 'raw' )`: 显示网站的各种信息。`$show` 参数可以是 `'name'` (网站标题), `'description'` (网站描述), `'url'` (网站 URL), `'charset'`, `'version'` 等。
- `get_bloginfo( string $show = '', string $filter = 'raw' )`: 返回网站信息字符串。
- `wp_head()`: **必须**放在 `<head>` 标签结束前。触发 `wp_head` 动作钩子。
- `wp_footer()`: **必须**放在 `</body>` 标签结束前。触发 `wp_footer` 动作钩子。
- `body_class( string|array $class = '' )`: 输出用于 `<body>` 元素的 CSS 类列表，包含页面类型、模板等信息。
- `home_url( string $path = '', string $scheme = null )`: 返回网站首页的 URL。常用于网站 Logo 链接。
- `get_template_directory_uri()`: 返回当前主题**主目录**的 URL。用于链接主题文件（如图片、主题自带的 CSS/JS）。注意：如果是子主题，这个函数返回的是**父主题**目录 URL。
- `get_stylesheet_directory_uri()`: 返回当前主题**样式表目录**的 URL。如果是主主题，和 `get_template_directory_uri()` 相同；如果是子主题，则返回**子主题**目录 URL。通常用这个函数来链接 `style.css` 和子主题自带的资源。
- `language_attributes()`: 输出 HTML 元素的语言属性，用于可访问性和 SEO。
- `is_front_page()`: 条件标签，判断当前页面是否为网站首页。
- `is_single()`: 条件标签，判断当前页面是否为单篇文章页。
- `is_page()`: 条件标签，判断当前页面是否为独立页面。
- `is_archive()`: 条件标签，判断当前页面是否为归档页（分类、标签、日期、作者等）。
- `is_404()`: 条件标签，判断当前页面是否为 404 错误页。

如何使用模板标签：

在你的 PHP 模板文件中，你需要将模板标签嵌入到 HTML 结构中。如果标签是直接输出的（如 `the_title()`），直接调用即可：

```
<h2><?php the_title(); ?></h2>
```

如果标签是返回字符串的（如 `get_the_title()`），你需要使用 `echo` 或简写形式 `<?= ... ?>` 来输出它：

```
<h2><?= get_the_title(); ?></h2>
```

或者

```
<h2><?php echo get_the_title(); ?></h2>
```

在主题开发中，通常倾向于使用 `the_` 开头的直接输出函数，除非你需要对返回的值进行进一步处理后再输出。

实践：

在你的 `index.php` 文件中，尝试使用更多上面提到的模板标签，例如在文章标题下方显示作者和日期，或者在 `<body>` 标签中使用 `body_class()`。刷新页面看看效果。

4.5. 拆分主题文件：`header.php`, `footer.php`, `sidebar.php`

为了更好地组织代码和提高可维护性，WordPress 主题通常会将页面的公共部分（头部、底部、侧边栏）拆分成单独的文件。这样，当你需要修改头部或底部时，只需修改一个文件，而不是修改所有模板文件。

我们将创建 `header.php`, `footer.php`, `sidebar.php`，并修改 `index.php` 来包含它们。

步骤：

1. **创建 `header.php`**：在 `my-first-theme` 文件夹内创建 `header.php`。将原来 `index.php` 中从 `<!DOCTYPE html>` 到网站标题/描述之后（但在 The Loop 之前）的所有代码剪切到 `header.php` 中。

```
<!DOCTYPE html>
<html <?php language_attributes(); ?>>
<head>
    <meta charset="<?php bloginfo('charset'); ?>">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title><?php wp_title('|', true, 'right'); bloginfo('name'); ?></title>
    <link rel="stylesheet" href="<?php echo get_stylesheet_uri(); ?>">
    <?php wp_head(); ?>
</head>
<body <?php body_class(); ?>>

    <header>
        <h1><a href="<?php echo home_url(); ?>"><?php bloginfo('name'); ?>
</a></h1> <!-- 网站标题链接到首页 -->
        <p><?php bloginfo('description'); ?></p>
        <!-- 这里以后可以放导航菜单 -->
    </header>

    <div id="content" class="site-content"> <!-- 常用的布局容器 -->
        <div id="primary" class="content-area">
            <main id="main" class="site-main">
                <!-- 主体内容 (The Loop) 将在这里开始 -->
```

保存 `header.php`。

2. **创建 `footer.php`**：在 `my-first-theme` 文件夹内创建 `footer.php`。将原来 `index.php` 中 The Loop 之后到 `</html>` 结束的所有代码剪切到 `footer.php` 中。

```
                </main><!-- #main -->
            </div><!-- #primary -->
```

```
            <?php get_sidebar(); // 这里包含 sidebar.php 文件 ?>

        </div><!-- #content -->

        <footer>
            <p>&copy; <?php echo date('Y'); ?> <?php bloginfo('name'); ?>.
All rights reserved.</p>
            <!-- 这里以后可以放底部菜单或其他信息 -->
        </footer>

        <?php wp_footer(); ?>
    </body>
</html>
```

注意，我在 `footer.php` 中添加了 `<?php get_sidebar(); ?>` 调用，这意味着侧边栏将放在主内容区域之后、页脚之前（这是一种常见的布局方式，主内容+侧边栏在一个容器内，页脚在外面）。如果你想要不同的布局（例如，侧边栏在主内容旁边），可能需要在 `header.php` 中开启容器并在 `footer.php` 中关闭。我们在这里使用了简单的两列/三列布局结构命名 (`#content`, `#primary`, `#secondary`/sidebar, `#main`)。 保存 `footer.php`。

3. **创建 `sidebar.php`**：在 `my-first-theme` 文件夹内创建 `sidebar.php`。

```
<?php
// 检查是否需要显示侧边栏
// 可以在 functions.php 中注册侧边栏区域，这里用来显示小工具
?>
<aside id="secondary" class="widget-area">
    <!-- 小工具将在这里显示 -->
    <?php dynamic_sidebar( 'sidebar-1' ); // dynamic_sidebar() 用于显示注册的
小工具区域 ?>
</aside><!-- #secondary -->
```

我们在这里使用了 `dynamic_sidebar( 'sidebar-1' )`，但这需要先在 `functions.php` 中注册一个 ID 为 `sidebar-1` 的小工具区域才能工作（后面会讲）。暂时它可能不显示任何内容。 保存 `sidebar.php`。

4. **修改 `index.php`**：现在 `index.php` 文件应该变得非常简洁了，它主要负责包含头部、尾部，以及执行 The Loop。

```
<?php get_header(); // 包含 header.php 文件 ?>

<?php
// The Loop (在 header.php 的 <main> 内部开始)
if ( have_posts() ) {
    while ( have_posts() ) {
        the_post();
        ?>
```

```
            <article id="post-<?php the_ID(); ?>" <?php post_class(); ?>>
                <h2><a href="<?php the_permalink(); ?>"><?php the_title(); ?>
</a></h2>
                <div class="entry-meta">
                    Posted on <?php the_date(); ?> by <?php the_author(); ?>
                </div>
                <div class="entry-content">
                    <?php the_content(); ?>
                </div>
            </article>
            <?php
        } // 循环结束
} else {
    ?>
    <p>抱歉，没有找到相关内容。</p>
    <?php
}
?>

<?php get_footer(); // 包含 footer.php 文件 ?>
```

保存 `index.php`。

5. **刷新网站**：访问你的网站前台。虽然页面结构变了（你可以通过浏览器开发者工具查看 HTML），但显示的内容应该与之前相同，因为我们只是重新组织了代码。现在你可以通过修改 `header.php` 来改变网站头部，修改 `footer.php` 来改变网站底部。

实践：

完成文件的拆分和修改。刷新网站确认没有错误。尝试在 `header.php` 的网站标题前面或后面添加一些文字，刷新看看是否生效。尝试在 `footer.php` 的版权信息后添加一段文字，刷新看看是否生效。

4.6. 理解并实现不同的模板文件（`single.php`, `page.php`, `archive.php`, etc.）

正如前面在模板层级结构中介绍的，WordPress 会根据请求的类型加载不同的模板文件。到目前为止，我们的主题只有一个 `index.php`，所以所有类型的页面（首页、单篇文章、独立页面、分类列表等）都使用了 `index.php` 来显示。

这显然不够，因为单篇文章页和文章列表页的布局需求通常是不同的（例如，单篇文章页不需要显示摘要，可能需要显示评论，而列表页只需要显示摘要和链接）。

现在，我们将为单篇文章页和独立页面创建特定的模板文件：`single.php` 和 `page.php`。

步骤：

1. **创建 `single.php`**：在 `my-first-theme` 文件夹内创建 `single.php` 文件。单篇文章页的结构通常与列表页类似，但它只显示**一篇**文章（当前请求的那篇），并且通常会包含评论区域。 将 `index.php` 的内容复制到 `single.php`。然后进行修改：
    - The Loop 结构保持不变 (`if ( have_posts() ) { while ( have_posts() ) { the_post(); ... } }`)。在单篇文章页，这个 Loop 只会执行一次。
    - 在 Loop 内部，确保使用了 `the_content()` 来显示文章的完整内容，而不是 `the_excerpt()`。

- 在 Loop 的 `<article>` 元素**内部**（在显示内容后），添加评论模板标签：`<?php comments_template(); ?>`。这个函数会根据评论相关的设置加载 `comments.php` 文件（如果你的主题有的话）来显示评论列表和评论表单。

修改后的 `single.php` 示例：

```
<?php get_header(); ?>

<?php
if ( have_posts() ) :
    while ( have_posts() ) : the_post();
        ?>
        <article id="post-<?php the_ID(); ?>" <?php post_class(); ?>>
            <header class="entry-header">
                <?php the_title( '<h1>', '</h1>' ); ?> <!-- 使用参数让标题在
<h1> 标签内 -->
                <div class="entry-meta">
                    Posted on <?php the_date(); ?> by <?php the_author(); ?>
                </div>
            </header><!-- .entry-header -->

            <div class="entry-content">
                <?php the_content(); ?>
                <?php
                // 分页导航（如果文章使用了分页符）
                wp_link_pages( array(
                    'before' => '<div class="page-links">' . esc_html__(
'Pages:', 'text-domain' ),
                    'after'  => '</div>',
                ) );
                ?>
            </div><!-- .entry-content -->

            <footer class="entry-footer">
                <?php // 这里可以显示分类、标签等信息 ?>
                <?php if ( has_category() ) { ?>
                    <span class="cat-links"><?php _e( 'Categories:', 'text-
domain' ); the_category( ', ' ); ?></span>
                <?php } ?>
                <?php if ( has_tag() ) { ?>
                    <span class="tags-links"><?php _e( 'Tags:', 'text-
domain' ); the_tags( '', ', ', '' ); ?></span>
                <?php } ?>
            </footer><!-- .entry-footer -->
        </article><!-- #post-<?php the_ID(); ?> -->

        <?php
        // 如果开启了评论，并且有评论或允许评论，显示评论模板
        if ( comments_open() || get_comments_number() ) :
            comments_template();
        endif;
```

```
    endwhile; // End of the loop.
else :
    ?>
    <p><?php esc_html_e( 'Sorry, no posts matched your criteria.', 'text-
domain' ); ?></p>
    <?php
endif;
?>

<?php get_footer(); ?>
```

注意我在上面的代码中引入了一些新的模板标签/函数，例如 `has_category()`, `the_category()`, `has_tag()`, `the_tags()`, `comments_open()`, `get_comments_number()`, `esc_html__()` 等。`esc_html__()` 是一个用于国际化和安全输出字符串的函数。`_e()` 是用于直接输出可翻译字符串的函数。文本域 `'text-domain'` 通常与你的主题文件夹名相同，用于后续的主题翻译。 保存 `single.php`。

2. **创建 `page.php`**：在 `my-first-theme` 文件夹内创建 `page.php` 文件。独立页面的显示通常比文章简单，它们没有分类、标签等，但可能也需要评论。 将 `index.php` 的内容复制到 `page.php`。进行类似 `single.php` 的修改：
    - Loop 结构保持不变。
    - Loop 内部使用 `the_content()`。
    - 在 Loop 的 `<article>` 元素内部，添加评论模板标签：`<?php comments_template(); ?>` (如果需要允许页面评论的话)。

    修改后的 `page.php` 示例：

```
<?php get_header(); ?>

<?php
if ( have_posts() ) :
    while ( have_posts() ) : the_post();
        ?>
        <article id="post-<?php the_ID(); ?>" <?php post_class(); ?>>
            <header class="entry-header">
                <?php the_title( '<h1>', '</h1>' ); ?>
            </header><!-- .entry-header -->

            <div class="entry-content">
                <?php the_content(); ?>
                <?php
                wp_link_pages( array(
                    'before' => '<div class="page-links">' . esc_html__(
'Pages:', 'text-domain' ),
                    'after'  => '</div>',
                ) );
                ?>
            </div><!-- .entry-content -->
```

```
            <footer class="entry-footer">
                <?php // 页面通常没有分类和标签，但这里可以放其他元信息 ?>
            </footer><!-- .entry-footer -->
        </article><!-- #post-<?php the_ID(); ?> -->

        <?php
        // 如果开启了评论，并且有评论或允许评论，显示评论模板（页面通常不开启评论，
但可以根据需要添加）
        // if ( comments_open() || get_comments_number() ) :
        //     comments_template();
        // endif;

    endwhile; // End of the loop.
else :
    ?>
    <p><?php esc_html_e( 'Sorry, no pages matched your criteria.', 'text-
domain' ); ?></p>
    <?php
endif;
?>

<?php get_footer(); ?>
```

保存 `page.php`。

3. **创建 `archive.php`**：归档页用于显示分类、标签、日期、作者等的文章列表。它的结构通常与 `index.php` 类似，但可能会在 The Loop 之前显示一些关于当前归档的信息（如分类名称）。 将 `index.php` 的内容复制到 `archive.php`。在 The Loop 的 `if ( have_posts() )` 之前，添加一些代码来显示归档标题：

```
<?php get_header(); ?>

<header class="page-header">
    <?php
    the_archive_title( '<h1 class="page-title">', '</h1>' ); // 显示归档标题
（如分类名、日期）
    the_archive_description( '<div class="archive-description">', '</div>'
); // 显示归档描述
    ?>
</header><!-- .page-header -->

<?php
// The Loop (在 header.php 的 <main> 内部开始)
if ( have_posts() ) {
    while ( have_posts() ) {
        the_post();
        ?>
        <article id="post-<?php the_ID(); ?>" <?php post_class(); ?>>
            <h2><a href="<?php the_permalink(); ?>"><?php the_title(); ?>
</a></h2>
            <div class="entry-meta">
```

```
                    Posted on <?php the_date(); ?> by <?php the_author(); ?>
                </div>
                <div class="entry-content">
                    <?php the_excerpt(); ?> <!-- 归档页通常只显示摘要 -->
                </div>
                <p><a href="<?php the_permalink(); ?>">阅读全文 &raquo;</a></p>
<!-- 添加“阅读全文”链接 -->
            </article>
            <?php
        } // 循环结束

        // 文章列表下方的分页导航
        the_posts_navigation(); // 显示“上一页/下一页”或页码链接（取决于设置）

    } else {
        ?>
        <p>抱歉，没有找到相关内容。</p>
        <?php
    }
    ?>

    <?php get_footer(); ?>
```

注意我们使用了 `the_archive_title()` 和 `the_archive_description()` 来显示归档页面的标题和描述。在 Loop 内部，我们改回了使用 `the_excerpt()` 来显示文章摘要，并添加了“阅读全文”链接。Loop 结束后，我们添加了 `the_posts_navigation()` 来显示分页链接，方便用户浏览更多文章。 保存 `archive.php`。

实践：

- 在 Local 网站后台创建几个分类和标签，给文章分配分类和标签。
- 创建几个独立页面。
- 访问单篇文章页、独立页面、某个分类的归档页。观察 WordPress 是否使用了你新建的 `single.php`, `page.php`, `archive.php` 文件来显示内容。页面的标题和结构应该与你修改后的文件相对应。

通过创建这些文件，你已经开始掌握如何利用 WordPress 的模板层级结构来为网站的不同部分提供不同的布局和显示方式。

4.7. 添加导航菜单与侧边栏小工具

用户通常期望网站有导航菜单（通常在头部或底部）和侧边栏（显示最新文章、分类列表、广告等）。这两部分在 WordPress 中是通过特定的功能来实现的。

4.7.1. 添加导航菜单 (Navigation Menus)

WordPress 允许用户在后台创建自定义菜单（包含链接到页面、文章、分类、自定义链接等），并将这些菜单分配到主题预定义的“菜单位置”。

步骤：

1. **在 `functions.php` 中注册菜单位置**：这是告诉 WordPress 你的主题支持菜单功能，并且你在主题的哪些地方预留了显示菜单的位置。在 `my-first-theme` 文件夹内创建 `functions.php` 文件（如果它还不存在）。添加以下代码：

```
<?php
// functions.php

// 在 setup 钩子中执行主题的初始化设置
add_action( 'after_setup_theme', 'my_first_theme_setup' );

function my_first_theme_setup() {

    // 注册导航菜单位置
    register_nav_menus( array(
        'primary' => esc_html__( 'Primary menu', 'my-first-theme' ), // 注册
一个名为 'primary' 的菜单位置
        'footer'  => esc_html__( 'Footer menu', 'my-first-theme' ),  // 注册
一个名为 'footer' 的菜单位置
    ) );

    // 添加主题对文章特色图片的支持
    add_theme_support( 'post-thumbnails' );

    // 添加主题对自定义 Logo 的支持
    add_theme_support( 'custom-logo' );

    // 添加主题对文章格式的支持 (Post Formats)
    add_theme_support( 'post-formats', array( 'aside', 'gallery' ) ); // 示
例：支持“日志”和“画廊”格式

    // 添加主题对HTML5标记结构的支持
    add_theme_support( 'html5', array( 'search-form', 'comment-form',
'comment-list', 'gallery', 'caption', 'script', 'style' ) );

    // 设置内容区域宽度 (可选，用于嵌入内容的最大宽度)
    if ( ! isset( $content_width ) ) {
        $content_width = 600; // 像素
    }

    // 注册编辑器样式 (可选，让后台编辑器显示主题的前台样式)
    // add_editor_style( 'editor-style.css' );

    // 加载主题的文本域，用于翻译
    load_theme_textdomain( 'my-first-theme', get_template_directory() .
'/languages' );

}

// 在 wp_enqueue_scripts 钩子中引入样式表和脚本
add_action( 'wp_enqueue_scripts', 'my_first_theme_scripts' );

function my_first_theme_scripts() {
```

```
    // 引入主题的 style.css (这是必须的，使用 get_stylesheet_uri() 获取)
    wp_enqueue_style( 'my-first-theme-style', get_stylesheet_uri() );

    // 如果你有其他的 CSS 文件，例如 main.css
    // wp_enqueue_style( 'my-first-theme-main', get_template_directory_uri()
. '/css/main.css' );

    // 如果你有需要引入的 JavaScript 文件，例如 script.js
    // wp_enqueue_script( 'my-first-theme-script',
get_template_directory_uri() . '/js/script.js', array('jquery'), '1.0', true
);
    // 注意第三个参数是依赖，第四个参数是版本号，第五个参数 true 表示在 footer 引
入
}


// 这里可以添加其他主题功能相关的函数，例如使用钩子修改内容等

?>
```

我们在 `my_first_theme_setup` 函数中使用 `register_nav_menus()` 注册了两个菜单位置：`primary` (主要菜单) 和 `footer` (底部菜单)。这个函数被挂载到 `after_setup_theme` 动作钩子上，这个钩子在 WordPress 加载主题后触发，适合做主题初始化设置。 我们还在 `my_first_theme_scripts` 函数中使用 `wp_enqueue_style()` 注册并加载了主题的 `style.css` 文件。这个函数被挂载到 `wp_enqueue_scripts` 动作钩子上，这是加载前端脚本和样式的标准方法，比直接在 `<head>` 中写 `<link>` 或 `<script>` 更推荐，因为它能更好地管理依赖和避免冲突。 保存 `functions.php`。

2. **在主题模板中显示菜单**：在 `header.php` 中找到你想要显示主要菜单的位置（通常在网站标题/描述下方），添加以下代码：

```
<nav id="site-navigation" class="main-navigation">
    <?php
    wp_nav_menu( array(
        'theme_location' => 'primary', // 指定要显示的菜单位置的 ID
        'menu_id'        => 'primary-menu', // 为生成的 <ul> 元素设置 ID
    ) );
    ?>
</nav><!-- #site-navigation -->
```

在 `footer.php` 中找到你想要显示底部菜单的位置，添加类似代码（修改 `theme_location`）：

```
<nav class="footer-navigation">
    <?php
    wp_nav_menu( array(
        'theme_location' => 'footer',
        'menu_id'        => 'footer-menu',
    ) );
    ?>
</nav>
```

`wp_nav_menu()` 是一个模板标签，用于显示指定位置分配的菜单。`theme_location` 参数非常重要，它必须和你 `register_nav_menus()` 中注册的 ID 对应。 保存 `header.php` 和 `footer.php`。

3. **在 WordPress 后台创建并分配菜单**：
    - 登录后台，“外观”->“菜单”。
    - 输入一个菜单名称（如“主导航”），点击“创建菜单”。
    - 在左侧选择你想要添加到菜单的页面、文章、分类或自定义链接，点击“添加到菜单”。
    - 在“菜单设置”下的“显示位置”中，勾选“Primary menu”（或你想分配的位置）。
    - 点击“保存菜单”。
    - 对底部菜单重复此过程。
4. **刷新网站**：访问网站前台。你应该能在 `header.php` 中添加 `wp_nav_menu()` 的位置看到你创建并分配的菜单了（样式可能很简陋，需要 CSS 来美化）。

4.7.2. 添加侧边栏小工具 (Sidebar Widgets)

小工具 (Widgets) 是 WordPress 中一种可拖拽的内容块（如最新文章、分类列表、自定义文本等），用户可以在后台“外观”->“小工具”中管理它们。主题需要**注册小工具区域 (Widget Areas)** 来告诉 WordPress 在哪里可以放置这些小工具。

步骤：

1. **在 `functions.php` 中注册小工具区域**：继续编辑 `functions.php` 文件，在 `my_first_theme_setup` 函数的下方（或者单独一个函数并挂载到 `widgets_init` 钩子上，后者更推荐）添加以下代码来注册一个侧边栏小工具区域：

```
// functions.php

// ... (前面的 my_first_theme_setup 函数和 wp_enqueue_scripts 函数)

// 注册小工具区域
add_action( 'widgets_init', 'my_first_theme_widgets_init' );

function my_first_theme_widgets_init() {
    register_sidebar( array(
        'name'          => esc_html__( 'Sidebar', 'my-first-theme' ), // 在后台显示的名字
        'id'            => 'sidebar-1', // 唯一 ID，用于 dynamic_sidebar() 调用
        'description'   => esc_html__( 'Add widgets here to appear in your sidebar.', 'my-first-theme' ), // 在后台的描述
        'before_widget' => '<section id="%1$s" class="widget %2$s">', // 每个小工具开始前的 HTML
        'after_widget'  => '</section>', // 每个小工具结束后的 HTML
        'before_title'  => '<h2 class="widget-title">', // 小工具标题开始前的 HTML
        'after_title'   => '</h2>', // 小工具标题结束后的 HTML
    ) );
```

```
    // 如果需要，可以注册更多小工具区域，例如页脚区域
    // register_sidebar( array(
    //  'name'          => esc_html__( 'Footer', 'my-first-theme' ),
    //  'id'            => 'footer-1',
    //  'description'   => esc_html__( 'Add widgets here to appear in your
footer.', 'my-first-theme' ),
    //  'before_widget' => '<section id="%1$s" class="widget %2$s">',
    //  'after_widget'  => '</section>',
    //  'before_title'  => '<h2 class="widget-title">',
    //  'after_title'   => '</h2>',
    // ) );
}

// ... (其他函数)
?>
```

`register_sidebar()` 函数用于注册小工具区域。`id` 参数是其唯一标识符，我们将在模板文件中使用它。`widgets_init` 钩子在所有默认小工具和注册的小工具区域被加载后触发，是注册小工具区域的正确时机。 保存 `functions.php`。

2. **在主题模板中显示小工具区域**：我们已经在 `sidebar.php` 文件中添加了显示小工具区域的代码：

```
<?php
// 检查这个侧边栏区域是否有活动的小工具
if ( is_active_sidebar( 'sidebar-1' ) ) { // 使用 is_active_sidebar() 检查
    ?>
    <aside id="secondary" class="widget-area">
        <?php dynamic_sidebar( 'sidebar-1' ); // 使用 dynamic_sidebar() 显示
小工具区域 ?>
    </aside><!-- #secondary -->
    <?php
}
?>
```

`dynamic_sidebar( 'sidebar-1' )` 函数用于显示指定 ID 的小工具区域中的所有小工具。我们还添加了 `is_active_sidebar( 'sidebar-1' )` 条件判断，这样只有当这个侧边栏区域在后台被添加了小工具时，整个 `<aside>` 元素才会被输出，避免在没有小工具时留下空的 HTML 结构。 保存 `sidebar.php`。

3. **在 WordPress 后台添加小工具**：
    - 登录后台，“外观”->“小工具”。
    - 在左侧的“可用小工具”列表中，拖拽一个你想要的小工具（例如“近期文章”、“分类”）到右侧你刚才注册的“Sidebar”区域中。
    - 配置小工具的设置（如标题、显示数量），点击“保存”。
4. **刷新网站**：访问网站前台（特别是列表页或单篇文章页，因为侧边栏通常显示在这些页面）。你应该能在 `sidebar.php` 文件被包含的位置看到你添加的小工具内容了。

实践：

完成菜单和小工具区域的注册和显示。在后台创建并分配菜单，添加小工具。刷新网站前台，确认菜单和小工具已经出现。尝试用浏览器开发者工具检查生成的 HTML 结构，找到菜单的 `nav` 标签和小工具的 `aside` 标签以及内部结构，对照你在 `wp_nav_menu()` 和 `register_sidebar()` 中设置的参数。

4.8. 使用 `functions.php` 增强主题功能（初级）

`functions.php` 是主题的“功能库”，你可以用它来：

- **注册菜单、侧边栏** (我们已经做了)。
- **添加主题支持**：开启特色图片、自定义 Logo、文章格式等 WordPress 内置功能。我们也在 `my_first_theme_setup` 函数中添加了一些示例。
- **引入脚本和样式** (我们已经用 `wp_enqueue_style()` 和 `wp_enqueue_script()` 做了)。
- **使用动作和过滤钩子**：这是 `functions.php` 最强大的用途之一。你可以通过 `add_action()` 和 `add_filter()` 在 WordPress 执行过程中的特定点插入或修改数据。

更多 `functions.php` 示例 (使用钩子)：

1. **修改文章摘要长度** (使用过滤钩子 `excerpt_length`)： 默认情况下，`the_excerpt()` 输出大约 55 个单词的摘要。你可以修改这个长度：

   ```
   // functions.php

   // ... (前面的代码)

   // 修改文章摘要的长度
   add_filter( 'excerpt_length', 'my_first_theme_excerpt_length', 999 ); // 优先级设高一点确保覆盖默认或低优先级设置

   function my_first_theme_excerpt_length( $length ) {
       return 20; // 将摘要长度设置为 20 个单词
   }

   // ... (其他函数)
   ```

   保存 `functions.php`，刷新首页（如果首页显示摘要），摘要长度应该变短了。`excerpt_length` 过滤钩子接收当前摘要长度作为参数，你的函数需要返回新的长度。

2. **修改文章摘要末尾的“阅读更多”文字** (使用过滤钩子 `excerpt_more`)：

   ```
   // functions.php

   // ... (前面的代码)

   // 修改文章摘要末尾的“阅读更多”文字
   add_filter( 'excerpt_more', 'my_first_theme_excerpt_more' );

   function my_first_theme_excerpt_more( $more ) {
   ```

```
    return ' <a href="' . get_permalink() . '" rel="bookmark">继续阅读
&raquo;</a>'; // 返回自定义的链接 HTML
}

// ... (其他函数)
```

保存 functions.php，刷新首页（如果首页显示摘要），摘要末尾的文字应该变了，并且链接到文章详情页。excerpt_more 过滤钩子接收默认的省略号或链接作为参数，你的函数需要返回新的文本或 HTML。

3. **在文章内容末尾添加一段文字** (使用过滤钩子 the_content):

```
// functions.php

// ... (前面的代码)

// 在单篇文章内容末尾添加一段文字
add_filter( 'the_content', 'my_first_theme_add_content_after' );

function my_first_theme_add_content_after( $content ) {
    // 只在单篇文章页添加
    if ( is_single() ) {
        $content .= '<p><em>这篇文章由 My First Theme 主题呈现。</em></p>'; //
.= 是字符串连接并赋值
    }
    return $content; // 必须返回内容
}

// ... (其他函数)
```

保存 functions.php，访问单篇文章页，文章内容末尾应该多了一段文字。the_content 过滤钩子接收文章的完整内容作为参数，你的函数需要返回修改后的内容。我们在这里用 is_single() 条件标签判断当前是否是单篇文章页，确保只在需要的地方添加内容。

这些例子展示了如何利用 add_filter() 钩子来修改 WordPress 核心函数输出的数据。类似地，你可以使用 add_action() 在特定的动作点执行代码，例如在保存文章时执行某个函数，或者在用户登录时显示一条消息。

实践:

尝试在你的 functions.php 文件中添加上面这些使用钩子的函数，观察网站前台的效果。思考你还可以在哪些地方使用钩子来修改 WordPress 的行为或数据。

4.9. 主题开发最佳实践与注意事项

在你进一步深入和开发更复杂的主题之前，了解一些最佳实践非常重要:

- **永远不要修改 WordPress 核心文件!** 这是最重要的规则。所有你的定制代码都应该在主题文件或插件文件中。

- **使用钩子 (Actions & Filters)**：尽量使用 WordPress 提供的钩子来扩展或修改功能，而不是直接修改核心或通过其他不标准的方式。
- **使用子主题 (Child Themes)**：如果你基于一个现有的主题进行修改（比如你想修改 Twenty Twenty-Four 主题），**不要直接修改原主题文件**。相反，你应该创建一个**子主题**。子主题继承父主题的所有模板和样式，你可以只覆盖或添加你需要修改/新增的文件。这样，当父主题更新时，你的修改不会丢失。创建一个子主题只需要一个包含特殊头部的 `style.css` 文件和一个引入父主题样式表的 `functions.php` 文件。
    - 子主题 `style.css` 头部示例：

```
/*
 Theme Name: My Twenty Twenty-Four Child Theme
 Theme URI: https://your-website.com/
 Description: My custom child theme.
 Author: Your Name
 Template: twentytwentyfour  // <-- 指定父主题的文件夹名称
 Version: 1.0.0
 Text Domain: my-twentytwentyfour-child
*/

/* 在这里写子主题的 CSS */
```

    - 子主题 `functions.php` (引入父主题样式)：

```
<?php
// 子主题 functions.php
add_action( 'wp_enqueue_scripts',
'my_twentytwentyfour_child_theme_enqueue_styles' );
function my_twentytwentyfour_child_theme_enqueue_styles() {
    $parenthandle = 'twentytwentyfour-style'; // 父主题的样式表 handle
(通常是主题文件夹名 + '-style')
    $theme = wp_get_theme();
    wp_enqueue_style( $parenthandle, get_template_directory_uri() .
'/style.css',
        array(), // 父主题样式表可能依赖的其他样式，可以留空
        $theme->parent()->get('Version')
    );
    wp_enqueue_style( 'child-style', get_stylesheet_uri(),
        array( $parenthandle ), // 子主题样式表依赖父主题样式表
        $theme->get('Version')
    );
}
?>
```

      然后你可以将父主题中你想修改的模板文件（如 `single.php`, `header.php`）复制到子主题文件夹中，并在那里进行修改。WordPress 会优先使用子主题中的文件。
- **安全是重中之重！** 特别是当你处理用户输入或与数据库交互时。
    - **验证 (Validation)**：检查用户输入的数据是否符合预期格式和类型。

    - **净化 (Sanitization)**：移除或转义用户输入数据中的潜在恶意内容（如 HTML 标签、JavaScript 代码），在将数据存入数据库之前进行。使用 WordPress 提供的函数，如 `sanitize_text_field()`, `esc_url()`, `wp_kses()` 等。
    - **转义 (Escaping)**：在将数据输出到页面上之前，转义特殊字符，以防止 XSS (Cross-Site Scripting) 攻击。使用 WordPress 提供的函数，如 `esc_html()`, `esc_attr()`, `esc_url()`, `wp_kses_post()` 等。例如，输出文章标题时使用 `<?php echo esc_html( get_the_title() ); ?>` 或者 `<?php the_title( '', '', true ); // the_title() 默认已处理转义 ?>`。 对于初学者，记住在输出可能包含用户输入或从数据库读取的数据时，**尽量使用 WordPress 提供的 `esc_` 开头的函数进行转义。**
- **国际化 (Internationalization - i18n)**：让你的主题能够被翻译成其他语言。这意味着所有面向用户显示的文本字符串都应该用 `__()` 或 `_e()` 函数包裹起来，并指定文本域（你的主题文件夹名）。
- **使用内置函数**：优先使用 WordPress 提供的函数（模板标签、API 函数），而不是直接写原生的 PHP 或 SQL 查询。这能保证代码的兼容性、安全性和效率。
- **组织代码**：使用拆分文件（header, footer, sidebar 等）、`functions.php` 的合理结构、甚至将复杂功能放入单独的插件来组织你的代码。
- **性能优化**：避免在 The Loop 中执行复杂的数据库查询或外部请求。谨慎使用插件。考虑图片优化、缓存等。
- **保持更新**：无论是 WordPress 核心、主题还是插件，都应该及时更新到最新版本，以获得安全补丁和新功能。
- **使用版本控制**：学习使用 Git 来管理你的代码。这能帮助你追踪修改历史，方便回退，并与他人协作。

好的，第四部分关于 WordPress 主题开发实战到这里结束。我们从零开始创建了一个基本主题，包含了必需的文件，学习了 The Loop 和常用模板标签，拆分了文件，并添加了菜单和小工具功能，最后介绍了一些重要的最佳实践。

你现在应该能够：

- 创建一个新的主题文件夹和基本文件 (`style.css`, `index.php`)。
- 理解 `style.css` 头部信息的作用。
- 理解并使用 The Loop 在模板文件中显示文章列表或单篇文章。
- 使用常用的模板标签获取和显示数据。
- 将主题拆分为 `header.php`, `footer.php`, `sidebar.php` 并使用 `get_template_part()` 或 `get_header()` 等包含它们。
- 为主题注册菜单位置和小工具区域。
- 在 `functions.php` 中使用 `add_action()` 和 `add_filter()` 钩子来添加或修改主题功能。
- 了解子主题、安全性和国际化的重要性。

本部分的实践作业：

1. **完善你的主题**：继续完善你创建的 `my-first-theme`，让它的结构更完整。尝试添加简单的 CSS 样式（直接写在 `style.css` 中，或者创建 `css` 文件夹并在 `functions.php` 中引入），让页面看起来更好一些。
2. **创建子主题**：选择一个 WordPress 默认主题（如 Twenty Twenty-Four），为其创建一个子主题。激活子主题，然后将父主题的 `index.php` 或 `single.php` 文件复制到你的子主题文件夹中，并进行一些修改（例如，在文章标题下方添加一行“这是我的子主题修改”）。验证修改是否生效。
3. **探索模板标签**：查阅 WordPress 官方文档关于模板标签的部分（https://developer.wordpress.org/themes/references/template-tags/），了解更多模板标签的功能，并

在你的主题中尝试使用一些新的标签。

4. **尝试更多钩子**：查阅 WordPress 官方文档关于动作钩子和过滤钩子的部分，找到一些感兴趣的钩子，并在你的 `functions.php` 中尝试使用 `add_action()` 或 `add_filter()` 挂载你的函数，看看能实现什么功能。

好的，我们已经打下了坚实的 WordPress 主题开发基础。你学会了如何控制网站的外观和内容的呈现方式。现在，我们将把注意力转向 WordPress 的另一个强大扩展方式——**插件**。

如果说主题决定了“长什么样”，那么插件就决定了“能做什么”。插件让你能够在不修改主题甚至不修改核心文件的情况下，为 WordPress 网站添加各种新功能。

第五部分：WordPress 插件开发实战

本部分的学习目标是让你理解 WordPress 插件的工作原理，亲手创建一个插件，并学习如何使用钩子、自定义文章类型、短代码等功能来为 WordPress 添加自定义功能。

5.1. 插件文件的基本构成与插件头部信息

一个 WordPress 插件可以是一个单独的 PHP 文件，也可以是一个包含多个文件和文件夹的复杂结构。但是，要让 WordPress 识别一个插件，**至少需要一个主 PHP 文件**，并且这个文件必须包含一个特殊的**插件头部注释块**。

这个主 PHP 文件通常放在 `wp-content/plugins/` 目录下以插件名称命名的文件夹中。例如，如果你的插件文件夹叫 `my-first-plugin`，那么主文件通常也叫 `my-first-plugin.php`。

插件头部注释块 (Plugin Header)：

这个注释块是 WordPress 识别插件的关键。它位于插件主 PHP 文件的最顶部，格式如下：

```php
<?php
/**
 * Plugin Name: My First Plugin
 * Description: 这是一个用于演示的简单 WordPress 插件。
 * Plugin URI: https://your-website.com/my-first-plugin/ (可选，插件主页 URL)
 * Author: Your Name (你的名字)
 * Author URI: https://your-website.com/ (可选，作者主页 URL)
 * Version: 1.0
 * License: GNU General Public License v2 or later (通常遵循 GPL)
 * License URI: http://www.gnu.org/licenses/gpl-2.0.html (通常指向 GPL 许可证 URL)
 * Text Domain: my-first-plugin (用于国际化/翻译，通常与插件文件夹名相同)
 * Domain Path: /languages (可选，如果翻译文件放在单独的 languages 文件夹)
 */

// 插件的代码从这里开始
// ...
```

- `Plugin Name:` 是**必需的**。WordPress 后台通过这个信息在插件列表中显示你的插件名称。
- 其他字段是可选的，但强烈推荐填写，它们提供了插件的基本信息，方便用户了解你的插件。
- `Text Domain:` 是用于国际化（让插件可翻译）的重要标识符。

- `Domain Path:` 指定了翻译 `.po` 文件所在的目录。

当你把一个包含这样头部注释的 PHP 文件上传到 `wp-content/plugins/` 目录（或其子文件夹中），WordPress 就能在后台“插件”列表中看到它了。

5.2. 创建一个最简单的 WordPress 插件

让我们来创建一个最基本的插件，它唯一的目的就是在网站页面的最底部添加一行文字。

步骤：

1. **找到插件目录**：打开你的 Local 网站文件夹。进入 `app/public/wp-content/plugins/` 目录。
2. **创建插件文件夹**：在这个 `plugins` 目录里，创建一个新的文件夹，例如 `my-first-plugin`。
3. **创建插件主文件**：在 `my-first-plugin` 文件夹内，创建一个名为 `my-first-plugin.php` 的新文件。
4. **添加插件头部和代码**：打开 `my-first-plugin.php` 文件，添加上面介绍的插件头部注释块，并在注释块下方添加以下 PHP 代码：

```php
<?php
/**
 * Plugin Name: My First Plugin
 * Description: 这是一个用于演示的简单 WordPress 插件。
 * Version: 1.0
 * Author: Your Name
 * Text Domain: my-first-plugin
 */

// 防止直接访问文件，提高安全性
if ( ! defined( 'ABSPATH' ) ) {
    exit; // Exit if accessed directly
}

// 使用 add_action() 钩子将我们的函数挂载到 wp_footer 动作上
add_action( 'wp_footer', 'my_first_plugin_add_text_to_footer' );

// 定义我们的自定义函数
function my_first_plugin_add_text_to_footer() {
    echo '<p style="text-align: center;">这是由我的第一个插件添加的内容！</p>';
}

?>
```

代码解释：

- `if ( ! defined( 'ABSPATH' ) ) { exit; }`: 这是一段标准的 WordPress 安全代码。`ABSPATH` 是 WordPress 加载时定义的一个常量（指向 WordPress 根目录）。如果一个 PHP 文件是直接通过 URL 访问而不是通过 WordPress 加载流程执行的，`ABSPATH` 就不会被定义。这段代码检查 `ABSPATH` 是否已定义，如果没有，就终止脚本执行，防止插件文件被直接恶意访问。你应该在所有插件和主题的功能性 PHP 文件的顶部加上这段代码。

    - `add_action( 'wp_footer', 'my_first_plugin_add_text_to_footer' );`: 这是核心部分。我们使用 `add_action()` 函数，将名为 `my_first_plugin_add_text_to_footer` 的函数挂载到 `wp_footer` 这个**动作钩子**上。这意味着当 WordPress 执行到 `wp_footer` 钩子时（通常在主题的 `footer.php` 文件中通过 `<?php wp_footer(); ?>` 调用触发），它会调用我们的 `my_first_plugin_add_text_to_footer` 函数。
    - `function my_first_plugin_add_text_to_footer() { ... }`: 这是我们的自定义函数，包含了我们希望在 `wp_footer` 钩子触发时执行的代码。这里我们简单地使用 `echo` 输出一段 HTML 文本。 保存 `my-first-plugin.php` 文件。

5. **激活插件**：打开你的 Local WordPress 网站后台，“插件”->“已安装的插件”。你应该能在列表中看到“My First Plugin”。点击“启用”。

6. **查看网站**：访问你的网站前台。滚动到页面底部，你应该能看到“这是由我的第一个插件添加的内容！”这段文字。

恭喜！你已经成功创建并激活了你的第一个 WordPress 插件。这个简单的例子虽然功能微不足道，但它演示了插件如何通过 `add_action()` 钩子在 WordPress 执行流程的特定点“插入”自己的代码。

5.3. 利用动作（Actions）钩子：在特定时刻执行代码

动作钩子是插件执行任务的主要方式。你使用 `add_action()` 函数将你的函数绑定到 WordPress 核心、主题或其它插件触发的动作钩子上。

一些常用的动作钩子（非常多，这只是一小部分）：

- `init`: WordPress 完成加载，但用户头信息和大部分功能尚未加载。**很多初始化工作都在这里完成**，例如注册自定义文章类型、分类法、Shortcode、后台菜单等。这是插件最常用的初始化钩子之一。
- `wp_enqueue_scripts`: 在前端页面加载脚本和样式时触发。你应该在这个钩子里使用 `wp_enqueue_style()` 和 `wp_enqueue_script()` 来引入你的 CSS 和 JS 文件。
- `admin_enqueue_scripts`: 在后台页面加载脚本和样式时触发。用于引入后台界面所需的 CSS 和 JS。
- `admin_menu`: 在生成后台管理菜单时触发。用于添加自定义顶级或子级管理菜单。
- `save_post`: 在文章（包括文章、页面、自定义文章类型）保存到数据库时触发。可以用来处理文章保存时的额外逻辑。
- `delete_post`: 在文章被删除时触发。
- `publish_post`: 在文章从草稿、待审或计划状态变为已发布时触发。
- `wp_login`: 用户成功登录时触发。
- `wp_logout`: 用户退出登录时触发。
- `plugins_loaded`: 所有激活的插件都已被加载时触发。
- `after_setup_theme`: 主题的基本功能（如菜单、特色图片支持）设置完毕后触发。
- `wp_head`: 在页面的 `<head>` 标签结束前触发（由主题的 `<?php wp_head(); ?>` 调用）。
- `wp_footer`: 在页面的 `</body>` 标签结束前触发（由主题的 `<?php wp_footer(); ?>` 调用）。

`add_action()` 函数语法：

```
add_action( string $hook_name, callable $callback, int $priority = 10, int
$accepted_args = 1 );
```

- `$hook_name`: 动作钩子的名称（字符串）。

- `$callback`: 当钩子触发时要执行的函数名（字符串）或匿名函数。
- `$priority`: （可选）执行优先级。数字越小，执行越早。默认为 10。
- `$accepted_args`: （可选）你的回调函数期望接收的参数数量。默认为 1。如果你的函数需要接收钩子传递的额外参数，需要在这里指定正确的数量。

示例：在后台菜单中添加一个顶级菜单项

这需要使用 `admin_menu` 钩子和 `add_menu_page()` 函数。

```
<?php
/**
 * Plugin Name: My Admin Menu Plugin
 * Description: 添加一个自定义后台菜单。
 * Version: 1.0
 * Author: Your Name
 * Text Domain: my-admin-menu-plugin
 */

if ( ! defined( 'ABSPATH' ) ) {
    exit;
}

// 将函数挂载到 admin_menu 动作钩子
add_action( 'admin_menu', 'my_admin_menu_create_menu' );

function my_admin_menu_create_menu() {
    // add_menu_page(
    //  string $page_title, // 页面标题（显示在浏览器标签页）
    //  string $menu_title, // 菜单标题（显示在侧边栏）
    //  string $capability, // 需要的权限，决定哪些用户能看到菜单
    //  string $menu_slug,  // 菜单的唯一 slug，用于 URL 和钩子
    //  callable $function = '', // 显示页面内容的函数
    //  string $icon_url = '',  // 菜单图标 URL
    //  int $position = null    // 菜单在侧边栏的位置
    // );
    add_menu_page(
        '我的自定义页面标题',          // 页面标题
        '我的插件菜单',              // 菜单标题
        'manage_options',            // 只有具有 'manage_options' 能力的用户才能看到（通常是管理员）
        'my-custom-plugin-page',     // 菜单 slug（唯一 ID）
        'my_admin_menu_page_content', // 显示页面内容的函数名
        'dashicons-admin-generic',    // 使用 WordPress 内置的通用图标
        99                           // 菜单位置，数值越大越靠下
    );
}

// 定义显示页面内容的函数
function my_admin_menu_page_content() {
    ?>
    <div class="wrap"> <!-- WordPress 后台页面的标准容器类 -->
        <h1>我的插件设置页面</h1>
```

```
            <p>这里是我的自定义插件的设置内容。</p>
            <?php // 这里可以添加表单等 ?>
        </div>
        <?php
}

?>
```

保存为一个新的 PHP 文件（例如 `my-admin-menu-plugin.php`）在 `plugins` 目录下，然后在后台激活它。你会看到后台侧边栏多了一个“我的插件菜单”项。点击它，就会看到我们定义的页面内容。

5.4. 利用过滤（Filters）钩子：修改 WordPress 数据

过滤钩子允许你在 WordPress 处理数据的过程中，修改或“过滤”这些数据。你使用 `add_filter()` 函数将你的函数绑定到过滤钩子上。

一些常用的过滤钩子：

- `the_content`: 在显示文章内容之前过滤内容。非常常用，可以用来在内容末尾添加广告、相关文章列表等。
- `the_title`: 在显示文章标题之前过滤标题。
- `excerpt_length`: 过滤自动生成的文章摘要长度。
- `excerpt_more`: 过滤文章摘要末尾的“阅读更多”链接文本。
- `wp_nav_menu_items`: 过滤导航菜单中的菜单项 HTML 列表。
- `upload_mimes`: 过滤允许上传的文件类型。
- `wp_mail`: 过滤 WordPress 发送的电子邮件的参数（收件人、主题、内容等）。
- `option_{option_name}`: 过滤从数据库 `wp_options` 表中获取的特定选项值。例如 `option_blogname` 过滤网站标题。
- `pre_get_posts`: 在执行主要文章查询之前，过滤查询参数。**非常强大和常用**，可以用来修改主查询（例如，在首页排除某个分类的文章，或者修改每页显示的文章数量）。

`add_filter()` 函数语法：

```
add_filter( string $hook_name, callable $callback, int $priority = 10, int
$accepted_args = 1 );
```

语法与 `add_action()` 完全相同，只是作用不同：过滤钩子期望你的回调函数接收一个值作为参数（通常是第一个参数），并**返回**一个值（修改后的）。

示例：修改网站标题

这需要使用 `option_blogname` 过滤钩子。

```
<?php
/**
 * Plugin Name: Modify Site Title
 * Description: 修改网站标题。
 * Version: 1.0
```

```
 * Author: Your Name
 * Text Domain: modify-site-title
 */

if ( ! defined( 'ABSPATH' ) ) {
    exit;
}

// 将函数挂载到 option_blogname 过滤钩子
add_filter( 'option_blogname', 'my_plugin_modify_site_title' );

// 定义过滤函数
function my_plugin_modify_site_title( $site_title ) {
    // $site_title 是原始的网站标题
    return $site_title . ' - 由我的插件修改'; // 返回修改后的标题
}

?>
```

保存为一个新的 PHP 文件（例如 `modify-site-title-plugin.php`）在 `plugins` 目录下，然后在后台激活它。访问网站前台，你会发现浏览器标签页和网站标题位置（如果主题使用了 `bloginfo('name')` 或 `wp_title()` 且未被其他代码覆盖）的网站标题变了。

示例：使用 `pre_get_posts` 修改首页显示的文章数量

默认情况下，首页显示的博客文章数量可以在后台“设置”->“阅读”中设置。你可以通过 `pre_get_posts` 过滤钩子在代码中覆盖这个设置。

```
<?php
/**
 * Plugin Name: Modify Posts Per Page
 * Description: 修改首页和归档页的文章显示数量。
 * Version: 1.0
 * Author: Your Name
 * Text Domain: modify-posts-per-page
 */

if ( ! defined( 'ABSPATH' ) ) {
    exit;
}

// 将函数挂载到 pre_get_posts 过滤钩子
add_action( 'pre_get_posts', 'my_plugin_change_posts_per_page' );

function my_plugin_change_posts_per_page( $query ) {
    // 确保我们只修改主查询，并且是在前台 (is_admin() 是检查是否在后台)
    // 并且是首页 (is_home()) 或归档页 (is_archive())
    if ( $query->is_main_query() && ! is_admin() && ( $query->is_home() || $query-
>is_archive() ) ) {
        $query->set( 'posts_per_page', 5 ); // 将每页显示的文章数量设置为 5
    }
```

```
    // 必须返回 $query 对象
    return $query;
}

?>
```

保存激活插件，将后台“设置”->“阅读”中的“博客页面最多显示”设置为大于 5 的数字，然后访问网站首页或归档页。你会发现实际显示的文章数量是 5。`pre_get_posts` 钩子提供了 `$query` 对象，你可以在执行查询之前修改它的参数。这是一个非常强大的钩子，但使用时要非常小心，确保你只修改了预期的查询，以免影响网站的其他部分。

5.5. 创建自定义文章类型（Custom Post Types - CPTs）

WordPress 最初只有“文章”(post) 和“页面”(page) 两种主要的内容类型。自定义文章类型允许你创建自己的内容类型，例如“产品”、“图书”、“电影”、“作品集”等。每个自定义文章类型都可以有自己的字段、分类法和归档页面。

注册自定义文章类型通常在 `init` 动作钩子中完成。

示例：注册一个“图书”自定义文章类型

```
<?php
/**
 * Plugin Name: Custom Post Type - Book
 * Description: 注册一个“图书”自定义文章类型。
 * Version: 1.0
 * Author: Your Name
 * Text Domain: custom-post-type-book
 */

if ( ! defined( 'ABSPATH' ) ) {
    exit;
}

// 将函数挂载到 init 动作钩子
add_action( 'init', 'my_plugin_register_book_post_type' );

function my_plugin_register_book_post_type() {
    $labels = array(
        'name'                  => _x( '图书', 'Post Type General Name', 'custom-
post-type-book' ), // _x() 和 __() 用于国际化
        'singular_name'         => _x( '图书', 'Post Type Singular Name', 'custom-
post-type-book' ),
        'menu_name'             => __( '图书', 'custom-post-type-book' ),
        'name_admin_bar'        => __( '图书', 'custom-post-type-book' ),
        'archives'              => __( '图书归档', 'custom-post-type-book' ),
        'attributes'            => __( '图书属性', 'custom-post-type-book' ),
        'parent_item_colon'     => __( '父级图书:', 'custom-post-type-book' ),
        'all_items'             => __( '所有图书', 'custom-post-type-book' ),
        'add_new_item'          => __( '添加新图书', 'custom-post-type-book' ),
```

```
        'add_new'               => __( '添加新图书', 'custom-post-type-book' ),
        'new_item'              => __( '新图书', 'custom-post-type-book' ),
        'edit_item'             => __( '编辑图书', 'custom-post-type-book' ),
        'update_item'           => __( '更新图书', 'custom-post-type-book' ),
        'view_item'             => __( '查看图书', 'custom-post-type-book' ),
        'view_items'            => __( '查看图书', 'custom-post-type-book' ),
        'search_items'          => __( '搜索图书', 'custom-post-type-book' ),
        'not_found'             => __( '未找到图书', 'custom-post-type-book' ),
        'not_found_in_trash'    => __( '回收站中未找到图书', 'custom-post-type-
book' ),
        'featured_image'        => __( '特色封面', 'custom-post-type-book' ),
        'set_featured_image'    => __( '设置特色封面', 'custom-post-type-book' ),
        'remove_featured_image' => __( '移除特色封面', 'custom-post-type-book' ),
        'use_featured_image'    => __( '使用特色封面', 'custom-post-type-book' ),
        'insert_into_item'      => __( '插入到图书', 'custom-post-type-book' ),
        'uploaded_to_this_item' => __( '上传到此图书', 'custom-post-type-book' ),
        'items_list'            => __( '图书列表', 'custom-post-type-book' ),
        'items_list_navigation' => __( '图书列表导航', 'custom-post-type-book' ),
        'filter_items_list'     => __( '过滤图书列表', 'custom-post-type-book' ),
    );
    $args = array(
        'label'                 => __( '图书', 'custom-post-type-book' ), // 文章类
型标签
        'description'           => __( '用于管理图书信息', 'custom-post-type-book'
), // 描述
        'labels'                => $labels, // 使用上面定义的标签数组
        'supports'              => array( 'title', 'editor', 'thumbnail',
'excerpt', 'comments' ), // 支持的功能：标题、编辑器、特色图片、摘要、评论
        'hierarchical'          => false, // 是否有父子关系（false 像文章，true 像页
面）
        'public'                => true, // 是否公开显示
        'show_ui'               => true, // 是否在后台显示 UI 界面
        'show_in_menu'          => true, // 是否在后台菜单中显示
        'menu_position'         => 5, // 菜单位置（文章下面）
        'menu_icon'             => 'dashicons-book', // 菜单图标
        'show_in_admin_bar'     => true, // 是否在管理工具栏中显示
        'show_in_nav_menus'     => true, // 是否在导航菜单中显示
        'can_export'            => true, // 是否可以导出
        'has_archive'           => true, // 是否有归档页面（例如
yoursite.com/book/）
        'exclude_from_search'   => false, // 是否从搜索结果中排除
        'publicly_queryable'    => true, // 是否可以通过 URL 查询
        'capability_type'       => 'post', // 权限类型（使用 post 的权限体系）
        'show_in_rest'          => true, // 是否在 REST API 中可用（古腾堡编辑器需要
这个）
        'rewrite'               => array('slug' => 'books'), // 设置 URL slug（例
如 yoursite.com/books/the-book-title）
    );
    register_post_type( 'book', $args ); // 注册文章类型，第一个参数是 slug，第二个参
数是配置数组

}

// 注册文章类型后，为了让其归档页面和单页面的永久链接生效，
```

```
// 需要刷新永久链接规则。这通常在插件激活/禁用时做。
// 但在开发中，你可以手动去后台“设置”->“永久链接”页面点一下“保存更改”来刷新。
// 后面讲插件激活钩子时会讲如何在激活时刷新。
?>
```

保存激活插件。你会看到后台侧边栏多了一个“图书”菜单。你可以像添加文章一样添加、编辑“图书”了。如果 `has_archive` 设置为 `true` 并设置了 `rewrite` slug，并且你刷新了永久链接，你就可以访问 `你的网站地址/books/` 查看图书列表归档页。

5.6. 创建自定义分类法（Custom Taxonomies）

自定义分类法允许你创建自己的分类系统来组织内容。WordPress 内置的分类法是“分类”(category) 和“标签”(tag)。你可以创建例如“作者”、“出版社”、“系列”等自定义分类法，并将它们关联到文章、页面或自定义文章类型上。

注册自定义分类法通常也在 `init` 动作钩子中完成。

示例：为“图书”文章类型注册一个“作者”自定义分类法

```php
<?php
/**
 * Plugin Name: Custom Taxonomy - Author
 * Description: 为“图书”自定义文章类型注册一个“作者”分类法。
 * Version: 1.0
 * Author: Your Name
 * Text Domain: custom-taxonomy-author
 */

if ( ! defined( 'ABSPATH' ) ) {
    exit;
}

// 将函数挂载到 init 动作钩子
add_action( 'init', 'my_plugin_register_author_taxonomy' );

function my_plugin_register_author_taxonomy() {
    $labels = array(
        'name'              => _x( '作者', 'taxonomy general name', 'custom-taxonomy-author' ),
        'singular_name'     => _x( '作者', 'taxonomy singular name', 'custom-taxonomy-author' ),
        'search_items'      => __( '搜索作者', 'custom-taxonomy-author' ),
        'all_items'         => __( '所有作者', 'custom-taxonomy-author' ),
        'parent_item'       => __( '父级作者', 'custom-taxonomy-author' ), // 如果是分层分类法
        'parent_item_colon' => __( '父级作者:', 'custom-taxonomy-author' ), // 如果是分层分类法
        'edit_item'         => __( '编辑作者', 'custom-taxonomy-author' ),
        'update_item'       => __( '更新作者', 'custom-taxonomy-author' ),
        'add_new_item'      => __( '添加新作者', 'custom-taxonomy-author' ),
        'new_item_name'     => __( '新作者名称', 'custom-taxonomy-author' ),
```

```
        'menu_name'         => __( '作者', 'custom-taxonomy-author' ),
        'not_found'         => __( '未找到作者', 'custom-taxonomy-author' ),
        'no_terms'          => __( '没有作者', 'custom-taxonomy-author' ),
        'items_list'        => __( '作者列表', 'custom-taxonomy-author' ),
        'items_list_navigation' => __( '作者列表导航', 'custom-taxonomy-author' ),
    );
    $args = array(
        'labels'            => $labels,
        'hierarchical'      => false, // 是否分层（true 像分类，false 像标签）
        'public'            => true, // 是否公开显示
        'show_ui'           => true, // 是否在后台显示 UI 界面
        'show_admin_column' => true, // 是否在列表页面显示为列
        'show_in_nav_menus' => true, // 是否在导航菜单中显示
        'show_tagcloud'     => true, // 是否显示在标签云小工具中
        'show_in_rest'      => true, // 是否在 REST API 中可用（古腾堡需要）
        'rewrite'           => array( 'slug' => 'author' ), // 设置 URL slug
    );
    // register_taxonomy( string $taxonomy, string|array $object_type, array $args
= array() );
    register_taxonomy( 'book_author', array( 'book' ), $args ); // 注册分类法，第一
个参数是 slug，第二个参数是关联的文章类型（这里关联到 'book' 文章类型），第三个参数是配置
数组

    // 同样，注册分类法后需要刷新永久链接才能让其归档页面生效。

}
?>
```

保存激活插件。如果你已经激活了上面创建的“图书”文章类型插件，现在去编辑图书，你会看到侧边栏多了一个“作者”的模块，可以添加作者了。如果 `hierarchical` 是 `true`，会是 checkbox 列表，如果是 `false` (如上面示例)，会是标签输入框。如果设置了 `rewrite` slug 并刷新了永久链接，你可以访问 `你的网站地址/author/作者名/` 查看某个作者的所有图书列表归档页。

CPT 和自定义分类法是 WordPress 开发中非常常见的需求，它们能让你以更结构化的方式管理不同类型的内容。

5.7. 为插件创建管理页面

如果你的插件有设置选项或需要一个专门的管理界面，你可以在后台创建一个自定义页面。我们已经在前面添加菜单的例子中初步展示了如何创建一个简单的页面。

更进一步，你可以使用 Settings API 或创建自己的表单来保存插件的设置。

使用 `add_options_page()` 添加一个设置子菜单

通常插件的设置页面会放在“设置”或“工具”下面作为子菜单。

```
<?php
/**
 * Plugin Name: Plugin Settings Page
 * Description: 添加一个插件设置页面。
```

```
 * Version: 1.0
 * Author: Your Name
 * Text Domain: plugin-settings-page
 */

if ( ! defined( 'ABSPATH' ) ) {
    exit;
}

// 将函数挂载到 admin_menu 动作钩子，用于添加菜单
add_action( 'admin_menu', 'my_plugin_settings_page_menu' );

function my_plugin_settings_page_menu() {
    // add_options_page(
    //  string $page_title,     // 页面标题
    //  string $menu_title,     // 菜单标题
    //  string $capability,     // 需要的权限
    //  string $menu_slug,      // 菜单 slug
    //  callable $function = '' // 显示页面内容的函数
    // );
    add_options_page(
        '我的插件设置',             // 页面标题
        '我的插件设置',             // 菜单标题（显示在“设置”菜单下）
        'manage_options',           // 权限
        'my-plugin-settings',       // 菜单 slug
        'my_plugin_settings_page_content' // 显示页面内容的函数
    );
}

// 定义显示页面内容的函数
function my_plugin_settings_page_content() {
    // 检查用户权限
    if ( ! current_user_can( 'manage_options' ) ) {
        return;
    }

    // 处理表单提交（如果页面上有表单）
    // check_admin_referer() 函数是用于安全检查的 nonce 验证
    if ( isset( $_POST['my_plugin_settings_submit'] ) && check_admin_referer(
'my_plugin_save_settings' ) ) {
        $setting_value = sanitize_text_field( $_POST['my_plugin_text_setting'] );
// 净化输入
        update_option( 'my_plugin_text_setting', $setting_value ); // 将设置保存到
数据库 wp_options 表
        ?>
        <div class="notice notice-success is-dismissible">
            <p>设置已保存！</p>
        </div>
        <?php
    }

    // 从数据库获取保存的设置值
    $saved_value = get_option( 'my_plugin_text_setting', '默认值' ); //
get_option( option_name, default_value )
```

```
    ?>
    <div class="wrap">
        <h1><?php echo esc_html( get_admin_page_title() ); ?></h1> <!-- 安全输出页面标题 -->

        <form action="" method="post"> <!-- 表单提交到当前页面 -->
            <?php settings_fields( 'my_plugin_settings_group' ); // 用于输出 nonce 字段等 ?>
            <?php // do_settings_sections( 'my_plugin_settings_page' ); // 如果使用 Settings API

            // 手动构建表单字段
            ?>
            <table class="form-table">
                <tr>
                    <th scope="row"><label for="my_plugin_text_setting">一个文本设置: </label></th>
                    <td>
                        <input type="text" name="my_plugin_text_setting" id="my_plugin_text_setting" value="<?php echo esc_attr( $saved_value ); // 安全输出属性 ?>" class="regular-text">
                        <p class="description">请输入一些文本。</p>
                    </td>
                </tr>
            </table>
            <?php wp_nonce_field( 'my_plugin_save_settings' ); // 添加 nonce 字段用于安全验证 ?>
            <?php submit_button( '保存设置', 'primary', 'my_plugin_settings_submit' ); // 添加标准提交按钮 ?>
        </form>
    </div>
    <?php
}

// 如果你的插件需要更复杂的设置页面（多个设置项，分组等），推荐学习和使用 WordPress 的 Settings API。
// 它提供了一套标准的方式来注册设置、设置区域、设置字段，并自动处理表单、验证和保存。
// 这里不做 Settings API 的详细讲解，因为篇幅所限且相对复杂，但请记住有这个更规范的方案。

?>
```

保存激活插件。在后台“设置”菜单下你会看到“我的插件设置”子菜单。点击进入页面，你可以输入文本并保存。保存的值会被存储在 `wp_options` 表中，并通过 `get_option()` 函数获取。注意代码中的安全函数 `sanitize_text_field()`, `esc_html()`, `esc_attr()`, `wp_nonce_field()`, `check_admin_referer()`。

5.8. Shortcode（短代码）的创建与使用

Shortcode 是 WordPress 提供的一种快捷方式，允许你在文章、页面或小工具的内容中插入一个简单的标签，WordPress 会在显示内容时用相应的动态内容替换这个标签。例如，`[gallery]` 会显示一个图片画廊。

你可以创建自己的 Shortcode 来方便用户在可视化编辑器中插入你的插件生成的内容或功能。

注册 Shortcode 通常在 `init` 动作钩子中完成，或者在 `wp_init` (别名钩子) 中。

示例：创建一个显示问候语的 Shortcode

```php
<?php
/**
 * Plugin Name: Greeting Shortcode
 * Description: 创建一个显示问候语的 Shortcode。
 * Version: 1.0
 * Author: Your Name
 * Text Domain: greeting-shortcode
 */

if ( ! defined( 'ABSPATH' ) ) {
    exit;
}

// 将函数挂载到 init 动作钩子 (或 wp_init)
add_action( 'init', 'my_plugin_register_greeting_shortcode' );

function my_plugin_register_greeting_shortcode() {
    // add_shortcode( string $tag, callable $callback );
    add_shortcode( 'greeting', 'my_plugin_greeting_shortcode_callback' ); // 注册一个名为 'greeting' 的 Shortcode
}

// Shortcode 回调函数
function my_plugin_greeting_shortcode_callback( $atts, $content = '', $tag = '' )
{
    // $atts 是 Shortcode 属性数组 (如 [greeting name="张三"])
    // $content 是包裹在 Shortcode 中的内容 (如 [greeting]Hello[/greeting])
    // $tag 是 Shortcode 的名称 (如 'greeting')

    // 解析属性 (使用 shortcode_atts() 函数来合并用户属性和默认属性，并进行安全处理)
    $atts = shortcode_atts( array(
        'name' => '访客', // 默认值
        'time' => 'any', // 示例属性
    ), $atts, 'greeting' ); // 第三个参数是 Shortcode 名称

    // 根据属性生成内容
    $output = '<p>你好, ' . esc_html( $atts['name'] ) . '! ';

    if ( $atts['time'] === 'morning' ) {
        $output .= ' 早上好! ';
    } elseif ( $atts['time'] === 'evening' ) {
        $output .= ' 晚上好! ';
    }

    $output .= '</p>';

    // Shortcode 函数必须返回内容，而不是直接输出
    return $output;
```

```
}
?>
```

保存激活插件。然后去创建或编辑一篇文章/页面，在内容编辑器中插入 `[greeting]`。保存并查看前台页面，`[greeting]` 应该被替换为“你好，访客！”。尝试插入 `[greeting name="李四"]`，会显示“你好，李四！”。尝试插入 `[greeting name="王五" time="morning"]`，会显示“你好，王五！ 早上好！”。

重要提示： Shortcode 回调函数**必须返回**要显示的内容，而不是使用 `echo` 直接输出。直接输出可能会导致内容出现在非预期的地方。使用 `shortcode_atts()` 函数处理 Shortcode 属性是推荐的安全做法。

5.9. 插件开发中的数据库操作（WPDB 类介绍）

WordPress 核心提供了一个 `$wpdb` 全局对象，它是 WordPress 数据库操作的抽象层。使用 `$wpdb` 类来与数据库交互是**强烈推荐**的，而不是直接使用原生的 PHP MySQL 函数。

为什么使用 `$wpdb`?

- **安全**：`$wpdb` 提供了准备语句 (prepared statements) 的方法，有助于防止 SQL 注入攻击。
- **抽象**：它兼容不同的数据库类型（虽然目前主要是 MySQL），并且处理了数据库前缀等问题。
- **方便**：提供了一些常用的查询和操作方法。

如何使用 `$wpdb`?

在使用 `$wpdb` 之前，需要通过 `global $wpdb;` 语句将其引入到你的函数作用域中。

```
<?php
/**
 * Plugin Name: WPDB Example
 * Description: 使用 $wpdb 查询数据库。
 * Version: 1.0
 * Author: Your Name
 * Text Domain: wpdb-example
 */

if ( ! defined( 'ABSPATH' ) ) {
    exit;
}

add_action( 'wp_footer', 'my_plugin_show_post_count' );

function my_plugin_show_post_count() {
    global $wpdb; // 引入 $wpdb 全局对象

    // 示例：查询 wp_posts 表中已发布的文章数量
    // 使用 $wpdb->prepare() 创建预处理语句，提高安全性
    $post_count = $wpdb->get_var( $wpdb->prepare(
        "SELECT COUNT(*) FROM $wpdb->posts WHERE post_type = %s AND post_status =
%s",
        'post', // %s 是字符串占位符
        'publish' // %s 是字符串占位符
    ) );
```

```
    // get_var() 用于获取查询结果的单个变量值（第一行第一列）

    if ( $post_count !== null ) {
        echo '<p style="text-align: center;">本站共有已发布的文章： ' . esc_html(
$post_count ) . ' 篇。</p>';
    } else {
        echo '<p style="text-align: center;">未能获取文章数量。</p>';
    }

    // 示例：获取最近 3 篇文章的标题和链接
    $recent_posts = $wpdb->get_results( $wpdb->prepare(
        "SELECT ID, post_title, post_name FROM $wpdb->posts WHERE post_type = %s
AND post_status = %s ORDER BY post_date DESC LIMIT %d",
        'post', 'publish', 3 // %d 是整数占位符
    ) );

    // get_results() 用于获取多行结果，返回一个对象数组
    // get_row() 用于获取单行结果，返回一个对象
    // get_col() 用于获取单列结果，返回一个数组

    if ( $recent_posts ) {
        echo '<h3>近期文章 (WPDB):</h3>';
        echo '<ul>';
        foreach ( $recent_posts as $post ) {
            // 使用 get_permalink() 和 esc_html() 安全地显示链接和标题
            echo '<li><a href="' . esc_url( get_permalink( $post->ID ) ) . '">' .
esc_html( $post->post_title ) . '</a></li>';
        }
        echo '</ul>';
    }

    // 示例：向 wp_options 表插入/更新一个选项（通常使用 update_option() 更方便）
    // $wpdb->insert( $wpdb->options, array('option_name' => 'my_new_option',
'option_value' => 'some value'), array('%s', '%s') );
    // $wpdb->update( $wpdb->options, array('option_value' => 'new value'),
array('option_name' => 'my_new_option'), array('%s'), array('%s') );
    // $wpdb->delete( $wpdb->options, array('option_name' => 'my_new_option'),
array('%s') );

    // 记住：对于文章、页面、用户等标准数据，优先使用 WordPress 提供的 API 函数（如
wp_insert_post(), update_post_meta(), get_users() 等），而不是直接操作数据库表。
    // $wpdb 主要用于查询或操作 WordPress 未提供方便 API 的自定义表或数据。

}

?>
```

保存激活插件。访问网站前台底部，你会看到显示文章数量和近期文章列表，这些数据是通过 `$wpdb` 直接从数据库查询获得的。注意 `prepare()` 的使用以及占位符 `%s` (字符串), `%d` (整数), `%f` (浮点数)。在需要直接执行 SQL 查询时，始终使用 `prepare()` 方法。

5.10. 在插件中正确引入样式和脚本

就像主题一样，插件也可能需要自己的 CSS 和 JavaScript 文件。你应该使用 WordPress 提供的 `wp_enqueue_style()` 和 `wp_enqueue_script()` 函数来引入它们，而不是直接在页面中写 `<link>` 或 `<script>` 标签。这样做的好处包括：

- **依赖管理**：可以指定你的脚本或样式依赖于哪些其他脚本或样式（例如，你的脚本可能依赖 jQuery）。
- **版本控制**：可以指定文件的版本号，有助于浏览器缓存管理。
- **避免冲突**：WordPress 会处理好文件加载顺序和重复加载的问题。
- **方便卸载**：当插件禁用或卸载时，这些文件不会被加载。

通常在 `wp_enqueue_scripts` 动作钩子（前端）或 `admin_enqueue_scripts` 动作钩子（后台）中调用这些函数。

示例：在前端引入一个 CSS 文件和一个 JS 文件

```
<?php
/**
 * Plugin Name: Enqueue Scripts Example
 * Description: 在前端引入样式和脚本。
 * Version: 1.0
 * Author: Your Name
 * Text Domain: enqueue-scripts-example
 */

if ( ! defined( 'ABSPATH' ) ) {
    exit;
}

// 将函数挂载到 wp_enqueue_scripts 动作钩子
add_action( 'wp_enqueue_scripts', 'my_plugin_enqueue_frontend_scripts' );

function my_plugin_enqueue_frontend_scripts() {
    // 获取插件目录的 URL
    // plugin_dir_url( __FILE__ ) 返回当前文件所在的目录 URL
    $plugin_url = plugin_dir_url( __FILE__ );

    // 引入 CSS 文件
    // wp_enqueue_style( string $handle, string $src, array $deps = array(),
string|bool|null $ver = false, string $media = 'all' );
    wp_enqueue_style(
        'my-plugin-style', // 唯一的 handle (标识符)
        $plugin_url . 'css/my-plugin-style.css', // 文件 URL
        array(), // 依赖的其他样式 handle 数组
        '1.0' // 版本号 (可以是插件版本或其他任意字符串, false 使用 WP 版本)
        // 'all' // 媒体类型
    );

    // 引入 JavaScript 文件
    // wp_enqueue_script( string $handle, string $src, array $deps = array(),
string|bool|null $ver = false, bool $in_footer = false );
    wp_enqueue_script(
```

```
        'my-plugin-script', // 唯一的 handle
        $plugin_url . 'js/my-plugin-script.js', // 文件 URL
        array('jquery'), // 依赖的脚本 handle 数组 (这里依赖 jQuery)
        '1.0', // 版本号
        true // 是否在页面底部加载 (true 推荐)
    );

    // 注意: wp_enqueue_style 和 wp_enqueue_script 只是"注册"了文件，它们实际输出
<link> 和 <script> 标签是在 wp_head() 和 wp_footer() 被调用时。所以主题中必须有这两个
函数调用。
}

// 为了演示，我们还需要创建 css 和 js 文件夹及相应文件
// 在你的插件文件夹 my-plugin-enqueue-scripts/ 下创建 css/ 和 js/ 文件夹
// 在 css/ 文件夹下创建 my-plugin-style.css
/*
   // my-plugin-style.css
   p { color: green; }
*/
// 在 js/ 文件夹下创建 my-plugin-script.js
/*
   // my-plugin-script.js
   jQuery(document).ready(function($) {
       console.log('My plugin script loaded!');
       $('p').on('click', function() {
           alert('Paragraph clicked!');
       });
   });
*/

?>
```

保存激活插件。在插件文件夹下创建 `css` 和 `js` 子文件夹，并在其中创建相应的 CSS 和 JS 文件并添加示例内容。刷新网站前台，查看页面源码，你应该能在 `<head>` 中看到你的 CSS 文件链接，在 `</body>` 结束前看到你的 JS 文件链接。同时，浏览器控制台应该输出"My plugin script loaded!"，点击段落应该弹出提示框。

5.11. 插件开发最佳实践与安全性

与主题开发类似，插件开发也有一些重要的最佳实践：

- **安全性是第一位的！** 插件代码通常比主题代码拥有更高的权限和更大的潜在影响范围。
    - **净化 (Sanitize)**：任何来自用户输入、外部 API 或不可信来源的数据，在**保存到数据库或进行处理**之前，都必须进行净化。使用 WordPress 提供的净化函数 (`sanitize_text_field()`, `esc_url_raw()`, `wp_kses()`, `sanitize_email()` 等)。
    - **转义 (Escape)**：任何可能包含特殊字符并要**输出到 HTML、属性、URL 或 JavaScript** 的数据，在输出之前都必须进行转义。使用 WordPress 提供的转义函数 (`esc_html()`, `esc_attr()`, `esc_url()`, `esc_js()`, `wp_kses_post()` 等)。
    - **Nonce 验证 (Nonce Verification)**：对于任何执行敏感操作（如保存设置、删除数据、执行某个动作）的请求，都必须进行 Nonce 验证，以防止 CSRF (Cross-Site Request Forgery) 攻击。使用 `wp_nonce_field()` 在表单中生成 Nonce 字段，使用 `check_ajax_referer()` (AJAX 请求) 或 `check_admin_referer()` (后台请求) 或 `wp_verify_nonce()` 进行验证。

    - **权限检查 (Capability Checks)**：在允许用户执行某个操作（如访问设置页面、保存选项）之前，务必检查用户是否具有相应的权限。使用 `current_user_can()` 函数。
    - **SQL 安全**：始终使用 `$wpdb->prepare()` 处理 SQL 查询中的变量，防止 SQL 注入。优先使用 WordPress API 函数操作标准数据。
- **使用钩子**：插件应该通过钩子与 WordPress 核心和其它组件交互，避免直接修改核心文件或全局变量（除非是文档说明可以修改的）。
- **代码组织**：对于复杂的插件，将代码拆分成多个文件和类，使用 PSR-4 标准进行自动加载，可以提高代码的可读性和可维护性。
- **唯一性**：你的插件中定义的函数、类、变量、常量、钩子名称、Shortcode 标签、CPT slug、分类法 slug、设置选项名、脚本/样式 handle 等都应该具有**唯一性**，以避免与其他主题或插件发生命名冲突。通常使用一个独特的前缀（例如你的插件文件夹名加上下划线 `my_plugin_`）来命名你的所有函数、变量和常量。
- **国际化**：让你的插件支持多种语言，使用 `__()` 和 `_e()` 函数标记所有可翻译字符串，并在插件头部指定 `Text Domain` 和 `Domain Path`。
- **卸载和激活钩子**：使用 `register_activation_hook()` 和 `register_deactivation_hook()` 在插件激活和禁用时执行代码，例如创建或删除数据库表、刷新永久链接 (`flush_rewrite_rules()`) 等。
- **Admin Notices**：在后台给用户显示消息时，使用 `admin_notices` 钩子和标准的 WordPress Notice HTML 结构，而不是直接 echo 错误信息。
- **避免阻塞操作**：插件的代码应该快速执行，避免进行长时间的外部请求或复杂的计算，以免拖慢网站速度。对于耗时操作，考虑使用 AJAX 或 Cron Job。
- **兼容性**：考虑你的插件与不同 WordPress 版本、PHP 版本、常见主题和插件的兼容性。
- **文档**：为你的插件编写清晰的文档，说明如何安装、使用和配置。

这是一个相当密集的插件开发基础介绍，但涵盖了创建功能性 WordPress 插件所需的核心概念和常用技术。

好的，第五部分关于 WordPress 插件开发实战到这里结束。你现在应该能够：

- 创建一个新的插件文件夹和主 PHP 文件，并添加插件头部信息。
- 理解插件如何通过钩子（动作和过滤）与 WordPress 交互。
- 使用 `add_action()` 和 `add_filter()` 挂载你的自定义函数。
- 了解常用的动作和过滤钩子。
- 创建自定义文章类型 (CPT) 和自定义分类法。
- 为插件添加一个简单的后台管理页面，并使用 `get_option()` 和 `update_option()` 保存/读取设置。
- 创建和使用 Shortcode。
- 了解并使用 `$wpdb` 类进行数据库操作。
- 使用 `wp_enqueue_style()` 和 `wp_enqueue_script()` 正确引入样式和脚本。
- 了解插件开发中的重要安全实践和最佳实践。

本部分的实践作业：

1. **创建并测试所有示例插件**：将本部分提供的所有代码示例分别保存为独立的 PHP 文件，放在 `wp-content/plugins/` 目录下的独立文件夹中。一个一个激活并测试它们的功能，确保你理解每一段代码的作用。
2. **结合 CPT 和分类法**：创建一个新的插件，同时注册“电影”文章类型和“导演”、“演员”分类法，并将它们关联起来。在后台添加一些电影数据，并尝试在主题的 `archive.php` 或创建一个新的 `archive-movie.php` 模板文件中显示电影列表。

3. **创建一个带设置的 Shortcode 插件**：创建一个插件，包含一个后台设置页面，允许用户设置一个问候语前缀。然后创建一个 Shortcode `[my_greeting]`，它使用设置页面保存的前缀，后面加上用户在 Shortcode 中指定的姓名（如 `[my_greeting name="张三"]`）。
4. **练习安全净化和转义**：在你的插件代码中，凡是处理用户输入的地方（如后台设置表单提交），确保使用了净化函数。凡是输出可能包含特殊字符的数据到页面的地方，确保使用了转义函数。

插件开发是一个广阔的领域，还有很多高级话题（如 AJAX, REST API, Cron Jobs, 自定义数据库表, 面向对象编程等），但掌握了本部分的内容，你就已经具备了开发大部分常见插件的基础能力。

下一部分预告： 在掌握了主题和插件开发基础后，我们将进入第六部分，初步探索如何阅读和理解 WordPress 核心代码中与主题、插件相关的部分，学习调试技巧，并展望进阶方向和推荐学习资源。

好的，我们来到了这个系统教程的第六部分。在前几部分，我们已经为你打下了坚实的 Web 基础，深入了解了 WordPress 的核心概念，并通过实践学会了如何创建和修改主题以及插件。

现在，是时候提升你的能力，让你能够更深入地理解 WordPress 的工作原理，学会如何排查问题，并为你指明持续学习的方向。本部分将带你初步领略 WordPress 核心代码的结构，学习调试技巧，并探讨一些进阶话题。

第六部分：理解 WordPress 核心代码（初步）与进阶

“看懂 WordPress 开源博客源码”是你的目标之一。这是一个具有挑战性但非常有益的目标。WordPress 核心代码是一个庞大且复杂的系统，要完全理解需要大量的时间和经验。本部分的目标是帮助你**入门**如何阅读和理解核心代码，让你不再觉得它是一个完全神秘的黑箱，而是可以探索和学习的资源。

6.1. WordPress 加载流程简介（`wp-config.php`, `wp-settings.php` 的作用）

在第三部分，我们宏观地描述了请求生命周期。现在，我们稍微深入一点，看看在请求的早期阶段，一些关键文件是如何被加载和执行的。这有助于你理解你的主题和插件代码是在何时被加载的。

回顾一下简化流程：

`index.php` -> `wp-blog-header.php` -> `wp-load.php` -> `wp-config.php` (配置) -> `wp-settings.php` (加载核心、插件、主题 functions) -> ...

1. **`index.php`**: 这是网站访问的入口文件。它定义 `WP_USE_THEMES` 常量，然后简单地 `require` 了 `wp-blog-header.php`。它的主要作用是将请求引导到 WordPress 的加载流程中。
2. **`wp-blog-header.php`**: 这个文件是加载 WordPress 环境的起点。它最重要的工作是 `require_once( __DIR__ . '/wp-load.php' );`。
3. **`wp-load.php`**: 这是 WordPress 加载流程中层层深入的核心文件。它的主要目的是找到并加载 `wp-config.php` 文件。它会尝试在当前目录和向上几级目录查找 `wp-config.php`。一旦找到并加载了 `wp-config.php`，就可以获取数据库连接信息等关键配置。然后 `wp-load.php` 会继续加载 `wp-settings.php`。
4. **`wp-config.php`**: 这个文件**不是**核心文件，它是你在安装 WordPress 时创建的配置文件。它包含了数据库连接信息、认证密钥、表格前缀等敏感和重要的配置。你也可以在这个文件中定义一些全局常量来修改 WordPress 的行为（例如 `WP_DEBUG`）。**理解 `wp-config.php` 的作用**非常重要，因为它决定了你的 WordPress 实例如何连接数据库以及一些基础行为。
5. **`wp-settings.php`**: 这是整个 WordPress 加载流程中**最核心、最关键、代码量最大**的文件之一。它完成了几乎所有的初始化工作：

    - 加载了 WordPress 核心库文件 (`wp-includes` 目录下的各种文件，定义了大量的核心函数、类、常量)。
    - 设置错误处理、时区、缓存等。
    - 加载并初始化**所有已激活的插件**。
    - 加载当前活动主题的 `functions.php` 文件。
    - 设置当前查询（根据 URL）。
    - 触发各种早期动作钩子（如 `plugins_loaded`, `setup_theme`, `after_setup_theme`, `init`, `wp_loaded` 等）。

你的主题 `functions.php` 和所有激活的插件代码，都是在 `wp-settings.php` 执行过程中被加载和执行的。 这解释了为什么你可以在 `functions.php` 和插件中通过 `add_action()` 和 `add_filter()` 挂载函数到 `init` 或更晚的钩子上，而这些函数会在后续的请求处理中被调用。

当你尝试阅读核心代码时，从 `index.php` 开始，按照 `require` 或 `include` 的顺序逐步深入 `wp-blog-header.php`, `wp-load.php`, `wp-settings.php`，你会对 WordPress 的启动过程有一个大致的了解。

6.2. 分析博客相关的核心文件（简化版）

WordPress 核心 (`wp-includes` 目录) 包含了海量文件，每个文件负责一部分核心功能。作为开发者，你不需要阅读所有文件，但了解一些与主题和插件开发密切相关的核心文件会非常有帮助。

请注意，不要修改这些核心文件！我们阅读它们是为了学习和理解。

- **`wp-includes/functions.php`**: 这是一个非常庞大的文件，包含了成千上万个 WordPress 核心函数，包括很多你在主题和插件中经常使用的**模板标签**（如 `get_the_title()`, `the_content()`, `wp_list_categories()` 等）以及其他辅助函数。当你遇到一个不熟悉的 WordPress 函数时，去 Code Reference (opens new window) 或直接查看这个文件（或其他相关的核心文件）是了解其作用和参数的最好方法。
- **`wp-includes/template-loader.php`**: 这个文件包含了实现**模板层级结构**的逻辑。它根据当前的查询 (`$wp_query` 对象) 和模板层级规则，决定应该加载主题中的哪个模板文件（`single.php`, `page.php`, `archive.php`, `index.php` 等），然后使用 `include` 或 `require` 加载它。如果你对 WordPress 如何决定使用哪个模板文件感到困惑，阅读这个文件能帮助你理解。
- **`wp-includes/class-wp-query.php`**: 这是定义 `$wp_query` 类的文件。`$wp_query` 是 WordPress 进行**主要文章查询**的核心对象，它存储了查询结果（文章列表）以及关于当前查询的信息（如是否是 404 页面，是否是首页等）。The Loop (`have_posts()` 和 `the_post()`) 实际上就是与 `$wp_query` 对象交互的方法。理解 `$wp_query` 有助于你更深入地理解 The Loop 和如何修改查询（例如使用 `pre_get_posts` 钩子）。
- **`wp-includes/plugin.php`**: 这个文件定义了 WordPress 的**钩子系统**！`add_action()`, `add_filter()`, `do_action()`, `apply_filters()` 这些核心函数都在这里定义。阅读这个文件（特别是这些函数的定义）可以让你更深入地理解钩子是如何工作的，优先级和参数是如何处理的。
- **`wp-includes/kses.php`**: 这个文件包含了用于**净化 HTML** 的函数，如 `wp_kses()`。理解这些函数如何工作对于编写安全的插件和主题至关重要。
- **`wp-includes/user.php` / `wp-includes/capabilities.php`**: 这些文件处理用户和权限相关的逻辑。当你需要处理用户登录、注册或检查用户权限时，这些文件是相关函数 (`current_user_can()`, `get_user_by()` 等) 的所在地。
- **`wp-admin/includes/` 目录**: 这个目录下的文件包含了所有后台界面的功能和逻辑。如果你需要开发后台相关的插件功能（如自定义设置页面、文章列表页面添加列等），你会经常与这个目录下的文件提供的函数和钩子打交道。

如何阅读核心代码：

1. **有目的性**：不要试图一次性读完所有代码。当你遇到一个特定的问题（例如，一个函数是做什么的，一个钩子在哪里触发，某个功能是如何实现的）时，再去查找相关的核心文件。
2. **使用代码编辑器**：一个好的代码编辑器（如 VS Code）有代码跳转、查找功能，可以方便你在文件之间导航和搜索函数名/变量名。
3. **从入口点开始**：当你追溯某个功能的实现时，从你已知的高层函数或模板文件开始，然后顺着函数调用、`require`/`include`、钩子触发点 (`do_action`, `apply_filters`) 逐步深入。
4. **查阅文档**：WordPress 官方开发者手册 (Developer Handbook) 和代码参考 (Code Reference) 是你最好的朋友。在阅读代码的同时查阅文档，可以帮助你理解代码的作用和用法。
5. **忽略细节**：初次阅读时，抓住主要逻辑和流程，不要纠结于每一个变量、每一个分支的细节。先建立整体概念。

6.3. 在核心代码中识别钩子点

前面提到，WordPress 的可扩展性依赖于钩子。当你阅读核心代码时，你会看到大量的 `do_action('hook_name', ...)` 和 `apply_filters('hook_name', $value, ...)` 调用。

识别这些钩子点的意义在于：

- 你知道在这些地方可以“插入”或“修改”WordPress 的行为。
- 当你想要在某个特定时刻执行代码或修改某个数据时，你可以通过搜索核心代码或查阅文档来找到合适的钩子。

如何在核心代码中查找钩子：

- **搜索文件**：在代码编辑器中，在 `wp-includes` 目录下搜索 `do_action(` 和 `apply_filters(`。
- **查看函数文档**：在 Code Reference 中查找 WordPress 函数，很多函数文档会说明该函数内部或执行前后触发了哪些钩子。

示例：

打开 `wp-includes/post-template.php` 文件，搜索 `apply_filters('the_content', ...)`，你会找到 `the_content()` 函数内部对这个过滤钩子的调用。这告诉你，如果你想修改文章显示的内容，可以将你的函数挂载到 `the_content` 过滤钩子上。

搜索 `wp-includes/general-template.php` 查找 `do_action('wp_head');` 和 `do_action('wp_footer');`，你会看到它们分别在 `wp_head()` 和 `wp_footer()` 函数中被调用。

当你遇到想要扩展或修改某个功能的需求时，第一步通常是去查找相关的 WordPress 函数或流程，然后找出它触发了哪些钩子。

6.4. 调试技巧与工具

编写代码不可避免地会遇到错误 (Errors) 和意料之外的行为 (Bugs)。掌握调试技巧是提高开发效率的关键。

- **`WP_DEBUG` 常量**：在 `wp-config.php` 文件中，将 `define('WP_DEBUG', false);` 修改为 `define('WP_DEBUG', true);`。这将开启 WordPress 的调试模式，在屏幕上显示 PHP 的错误、警告和通知。
    - `WP_DEBUG_DISPLAY`: 默认为 `true`。控制是否在页面上显示调试信息。在生产环境应设置为 `false`。

    - `WP_DEBUG_LOG`: 默认为 `false`。如果设置为 `true`，所有调试信息会被写入 `wp-content/debug.log` 文件，而不是显示在页面上。在生产环境并且需要记录错误时，可以设置为 `true`。
    - `SCRIPT_DEBUG`: 默认为 `false`。如果设置为 `true`，WordPress 会加载核心、主题和插件的开发版本（未压缩的 `.dev.js` 和 `.css` 文件），有助于调试前端脚本和样式。 **请注意：调试模式（特别是 `WP_DEBUG_DISPLAY` 为 `true`）只能在本地开发环境开启！** 在线网站开启调试模式会暴露敏感信息，存在安全风险，并且会影响用户体验。
- **PHP 错误日志**：除了 WordPress 自带的调试模式，你也可以配置 PHP 本身将错误写入日志文件。这通常在你的 Web 服务器配置 (`php.ini`) 中设置 `display_errors = Off` 和 `log_errors = On`，并指定 `error_log` 文件的路径。Local by Flywheel 通常会帮你配置好，你可以在 Local 网站的设置中找到查看日志的选项。
- **使用 `var_dump()`, `print_r()`, `error_log()`**：
    - `var_dump($variable);`：输出变量的类型和值，对于检查变量内容（特别是数组和对象）非常有用。
    - `print_r($variable);`：以更易读的方式输出数组和对象。
    - `echo "Debug message";`：简单地在页面输出一些标记，看看代码是否执行到此处。
    - `error_log('Debug message here');` 或 `error_log(print_r($variable, true));`：将调试信息写入 PHP 错误日志文件，这在不能直接在页面上输出信息时非常有用（例如，在 AJAX 请求或重定向发生时）。 **注意：** 在公共区域直接使用 `var_dump()` 或 `print_r()` 会破坏页面布局。使用 `error_log()` 是更“干净”的调试方式。调试完成后，务必删除这些调试代码。
- **浏览器开发者工具**：对于前端问题（HTML 结构、CSS 样式、JavaScript 错误），浏览器内置的开发者工具（按 F12 打开）是必备工具。你可以检查元素、修改 CSS、查看网络请求、调试 JavaScript。
- **专门的调试插件**：安装一些强大的调试插件可以极大地提升效率。
    - **Query Monitor**：**强烈推荐！** 这是一个免费插件，安装激活后会在后台管理工具栏显示当前页面加载的各种信息，包括：数据库查询（数量、耗时、具体查询语句）、PHP 错误和警告、钩子触发情况、Enqueue 的脚本和样式、HTTP API 调用、模板文件加载顺序等等。它能帮助你快速定位性能问题、错误来源和理解代码执行流程。
    - Debug Bar：另一个常用的调试工具条插件。
- **Xdebug**：更高级的 PHP 调试器，可以让你设置断点，逐步执行代码，检查变量状态。配置和使用相对复杂一些，但对于解决复杂问题非常强大。Local by Flywheel 通常内置了 Xdebug 支持，你可以在 Local 的网站设置中启用它。

调试流程建议：

1. **明确问题**：错误信息是什么？页面哪里不对劲？功能没有按预期工作？
2. **开启调试**：在 `wp-config.php` 中开启 `WP_DEBUG` 和 `WP_DEBUG_LOG`。
3. **查看错误**：检查页面上的错误信息或 `debug.log` 文件。错误信息通常会告诉你问题出在哪个文件哪一行。
4. **缩小范围**：根据问题和错误信息，尝试缩小可能出问题的代码范围（是主题问题？插件问题？是特定页面的问题？）。可以尝试暂时禁用主题或插件来排查。
5. **跟踪代码**：在可疑的代码位置插入 `error_log()` 或 `var_dump()` 来检查变量的值和代码执行流程。
6. **使用 Query Monitor**：检查数据库查询是否正确、钩子是否按预期触发、是否有 PHP 警告/通知。
7. **浏览器开发者工具**：检查前端元素的 HTML、CSS、JS 是否正确。
8. **查阅文档和搜索**：根据错误信息或问题描述搜索 WordPress Codex、Developer Handbook、Stack Exchange、博客文章等。
9. **逐步排除**：一次只修改或测试一个东西，避免引入新的问题。

6.5. 版本控制基础 (Git 简介)

版本控制是现代软件开发中不可或缺的工具，对于 WordPress 开发也不例外。Git 是目前最流行的分布式版本控制系统。

为什么要使用 Git?

- **追踪修改**：记录你代码的每一次改动，你可以随时查看历史，知道是谁在什么时候做了什么修改。
- **回退版本**：如果不小心引入了 bug 或破坏了功能，可以轻松地回退到之前的稳定版本。
- **协作开发**：多人协作开发时，Git 提供了合并不同开发者代码的机制。
- **分支管理**：可以在不影响主代码线的情况下，创建分支开发新功能或修复 bug，开发完成后再合并回去。
- **备份**：可以将代码库存储在远程仓库（如 GitHub, GitLab, Bitbucket），作为代码的备份。

Git 的基本概念:

- **Repository (仓库)**：存放项目代码和 Git 历史记录的地方。通常是项目根目录下的一个 `.git` 隐藏文件夹。
- **Commit (提交)**：记录一次或一组代码修改。每次提交都有一个唯一的 ID，并包含修改说明。
- **Branch (分支)**：代码库的一个独立开发线。`master` 或 `main` 通常是主分支。
- **Clone (克隆)**：从远程仓库复制一个本地仓库。
- **Add (添加)**：将工作目录中的修改添加到暂存区 (Staging Area)，准备进行提交。
- **Commit (提交)**：将暂存区的修改永久记录到本地仓库。
- **Push (推送)**：将本地仓库的提交上传到远程仓库。
- **Pull (拉取)**：从远程仓库下载最新的修改并合并到本地仓库。

基本的 Git 工作流程 (本地开发):

1. **初始化仓库**：在你的主题或插件文件夹根目录（不是 WordPress 根目录）打开终端或命令行，运行 `git init`。
2. **添加文件**：使用 `git add .` 添加所有文件到暂存区。
3. **提交修改**：使用 `git commit -m "Initial commit"` 提交你的第一次修改。
4. **继续修改**：编写代码。
5. **查看状态**：使用 `git status` 查看哪些文件被修改、添加或删除了。
6. **添加和提交**：对修改过的文件，使用 `git add <filename>` 或 `git add .`，然后使用 `git commit -m "Add new feature X"` 提交修改。写清晰的提交信息非常重要。
7. **查看提交历史**：使用 `git log` 查看所有提交记录。

学习 Git 需要一些时间和练习，但它是提升开发效率和规范性的重要一步。有很多在线教程和资源可以学习 Git。

6.6. 性能优化基础

网站速度对用户体验和 SEO 都非常重要。在 WordPress 开发中，你需要注意一些基本的性能优化原则。

- **优化数据库查询**：避免在 The Loop 或页面加载过程中执行过多的或低效的数据库查询。Query Monitor 是检查查询数量和耗时的强大工具。尽可能使用 WordPress 内置函数，它们通常是优化过的。
- **优化图片**：使用适当尺寸的图片，并对图片进行压缩优化。可以考虑使用 WebP 等新格式。使用 WordPress 的媒体库上传图片，它会自动生成不同尺寸的图片。

- **缓存**：使用缓存插件（如 WP Super Cache, W3 Total Cache, LiteSpeed Cache）可以生成网站的静态 HTML 文件，极大地减少服务器处理请求的时间。对象缓存和数据库缓存也能提高动态内容的加载速度。
- **最小化和合并文件**：将多个 CSS 文件合并成一个，多个 JavaScript 文件合并成一个，并对这些文件进行最小化（移除空格、注释等），可以减少 HTTP 请求次数和文件大小。许多优化插件可以帮你完成这个。
- **延迟加载脚本和样式**：非关键的 JavaScript 文件（如位于页面底部的脚本）应使用 `wp_enqueue_script()` 的第五个参数 `true` 在页脚加载，避免阻塞页面渲染。对于首屏不需要的样式，可以考虑异步加载。
- **使用 CDN**：使用内容分发网络 (CDN) 来分发静态资源（图片、CSS、JS），可以加快全球用户的访问速度。
- **选择好的主机**：主机的性能对网站速度有基础性的影响。

作为开发者，你的代码质量直接影响性能。避免在前端加载不必要的资源，避免在页面加载时执行耗时操作，是你在开发阶段就应该考虑的。

6.7. 安全性基础（重访）

我们在插件开发部分强调了安全性，这里再次强调其基础原则：

- **验证 (Validate)**：确认输入的数据是否是你期望的类型和格式。
- **净化 (Sanitize)**：移除或处理数据中不安全的部分，使其可以安全存储或在后台使用。
- **转义 (Escape)**：在将数据输出到页面上之前，处理特殊字符，防止跨站脚本攻击 (XSS)。
- **权限检查**：确保用户有权限执行他们正在尝试的操作。
- **Nonce 验证**：防止 CSRF 攻击，确保请求来自你的网站并且是用户主动发起的。
- **限制文件访问**：通过 `.htaccess` 或 Web 服务器配置限制对敏感文件（如 `.php` 文件，特别是 `/wp-content/uploads/` 目录下的 PHP 文件）的直接访问。我们插件示例中的 `if ( ! defined( 'ABSPATH' ) ) { exit; }` 就是一个基本的文件访问限制措施。
- **数据库安全**：使用 `$wpdb->prepare()` 防止 SQL 注入。

遵循这些原则，可以显著提高你的主题和插件的安全性，保护用户和网站数据。

6.8. 持续学习与资源推荐

WordPress 开发是一个不断发展的领域，新的版本、新的功能、新的技术层出不穷。要成为一名优秀的 WordPress 开发者，持续学习是必不可少的。

重要学习资源：

- **WordPress 官方开发者手册 (Developer Handbook)**：https://developer.wordpress.org/ 这是最权威的 WordPress 开发文档，涵盖了主题开发、插件开发、REST API、命令行工具 (WP-CLI) 等所有方面的详细信息和最佳实践。遇到问题时，首先应该查阅这里。
- **WordPress Code Reference**：https://developer.wordpress.org/reference/ 这是一个函数、类、方法、钩子等的参考大全。你可以查找任何 WordPress 函数的作用、参数、返回值、使用示例以及它在哪个版本中引入或修改。这是阅读核心代码和编写代码时的必备工具。
- **WordPress Codex**：https://codex.wordpress.org/ 虽然部分内容可能不如 Developer Handbook 新，但它仍然包含大量有用的信息，特别是关于早期功能和概念的解释。
- **WordPress Stack Exchange**：https://wordpress.stackexchange.com/ 这是一个问答网站，有很多 WordPress 开发者在这里提问和回答问题。你遇到的很多问题可能在这里都能找到答案。

- **WordPress 支持论坛**：https://wordpress.org/support/ 官方支持论坛，可以提问，也可以帮助他人。
- **官方 Make WordPress 博客**：https://make.wordpress.org/ 这是 WordPress 核心开发、设计、文档等各个团队的博客，了解 WordPress 的最新动态和未来的发展方向（特别是 Gutenberg/全站编辑）。
- **知名 WordPress 开发博客和教程网站**：例如 Smashing Magazine 的 WordPress 专栏、CSS-Tricks 的 WordPress 内容、Tuts+ 等。
- **阅读高质量的主题和插件代码**：去 WordPress.org 主题和插件目录下载一些受欢迎且评价好的主题和插件，阅读它们的源代码，学习它们是如何组织代码、使用钩子和实现功能的。WordPress 默认主题（如 Twenty Twenty-Four）的代码是很好的学习范例。
- **GitHub**：许多开源的 WordPress 插件和主题都在 GitHub 上托管，你可以查看它们的代码、提交历史、Issues、Pull Requests，学习开发者们是如何协作和解决问题的。WordPress 核心代码本身也在 GitHub 上 (https://github.com/WordPress/WordPress)。
- **WP-CLI**：https://wp-cli.org/ WordPress 命令行工具，可以让你通过命令行管理 WordPress（安装、更新、管理用户、文章、插件、主题等）。对于开发者来说非常高效。
- **学习更多 PHP, HTML, CSS, JavaScript 知识**：WordPress 是基于这些技术构建的，更深入地掌握基础技术会让你在 WordPress 开发中走得更远。推荐资源如 MDN Web Docs (HTML, CSS, JS), PHP Manual (php.net)。
- **参与社区**：参加 WordCamp (WordPress 线下活动)、贡献翻译、贡献文档、甚至贡献代码到核心、主题或插件，都是提升自己和回馈社区的好方式。

6.9. 总结与展望

恭喜你！你已经完成了这个从零开始的 WordPress 主题与插件开发的系统教程。

你现在应该具备以下能力：

- 理解 WordPress 的基本架构和工作流程。
- 掌握基本的 HTML, CSS, PHP 知识，能够读懂和编写简单的 Web 代码。
- 熟练搭建和使用 Local by Flywheel 本地开发环境。
- 理解 WordPress 的核心机制——钩子系统（动作和过滤）。
- 能够从零开始创建一个基本的 WordPress 主题，理解模板层级和常用模板标签，并添加菜单和侧边栏功能。
- 能够从零开始创建一个基本的 WordPress 插件，使用钩子添加功能，创建自定义文章类型、分类法、Shortcode，并初步了解数据库操作和文件引入规范。
- 了解如何初步阅读 WordPress 核心代码，找到钩子点。
- 掌握基本的调试技巧和工具。
- 理解版本控制、性能优化和安全性的基础重要性。
- 知道如何找到更多学习资源来持续提升。

达到“看懂 WordPress 开源博客源码”的目标：

通过本教程的学习，你应该已经不再惧怕打开 WordPress 核心代码文件了。你会认识其中的 PHP 语法、HTML 结构，能理解 `require` 和 `include` 的作用，能识别 `do_action` 和 `apply_filters` 这样的钩子点，也能找到 `have_posts()` 和 `the_post()` 这样的模板标签和核心函数。

要“完全看懂”所有源码，需要大量的实践和深入研究，逐步去理解核心文件的具体实现细节、不同模块之间的复杂交互。这是一个漫长的过程，但你已经迈出了最关键的第一步。

接下来的旅程：

- **大量实践**：不断练习创建更复杂的主题和插件。尝试实现你想到的各种功能。
- **阅读代码**：选择一些你使用或感兴趣的开源主题和插件，深入阅读它们的源代码，学习它们的设计思路和实现技巧。
- **解决问题**：在开发过程中你会遇到各种问题，学会独立查找资料、调试代码来解决它们。
- **深入特定领域**：根据你的兴趣，可以深入学习某个特定领域，例如：
    - **前端现代化**：使用 React/Vue 等前端框架与 WordPress REST API 结合开发 Headless WordPress 或 Block Editor (Gutenberg) 相关功能。
    - **性能优化**：深入研究各种性能调优技术。
    - **安全性**：更系统地学习 Web 安全和 WordPress 安全加固。
    - **电子商务**：学习 WooCommerce 插件的开发和定制。
    - **会员系统**：学习相关的插件和开发技术。
    - **贡献核心**：如果你对 WordPress 本身感兴趣，可以学习如何参与核心开发。

祝贺你完成了这个阶段的学习！你现在已经具备了成为一名 WordPress 开发者的基础。这个领域的潜力巨大，机会无限。保持好奇心，持续学习，享受创造的乐趣！

如果你在后续学习过程中有任何新的问题或需要进一步的指导，随时都可以再次提问。

整个教程到此结束。 我希望它为你构建了一个清晰、系统的 WordPress 开发知识体系。

如果你有其他想构建知识体系的主题，或者想针对本教程的某个部分进行更深入的探讨，请告诉我。