插件开发全攻略

Posted By Charles On 2008 年 7 月 6 日 @ 11:33

隔了这么久，终于将这个系列文章都翻译完了，这么一遍走下来，实在觉得自己很浅薄，技术和语言水平都十分有限。恐怕也没有全部传达原作者的意图。

我打算陆续将该系列的文章校对一遍，并且打算将这个系列里面的例子改写成自己的例子，这样，我可以对文章有更准确的理解。

这里，我罗列一下这个系列文章的目录吧。英文名字就是原文，底下的中文名字就是译文。

1. How to Write a WordPress Plugin - Introduction [1]
2. 介绍 [2]
3. Seven Reasons to Write a WordPress Plugin [3]
4. 编写插件的七个理由 [4]
5. How to Get Ideas for Wordpress Plugins [5]
6. 怎样获得 WordPress 插件的创意 [6]
7. Structure of a Wordpress Plugin [7]
8. WordPress 插件的结构 [8]
9. WordPress Plugin Actions [9]
10. WordPress 插件 Actions [10]
11. WordPress Plugin Filters [11]
12. WordPress 插件 Filter [12]
13. Constructing a WordPress Plugin Admin Panel [13]
14. 构造一个 WordPress 插件管理员面板 [14]
15. Constructing a WordPress Plugin User’s Panel [15]
16. 构建一个 WordPress 插件用户面板 [16]
17. WordPress Plugins and Database Interaction [17]
18. WordPress 插件和数据库交互 [18]
19. Using JavaScript and CSS with your WordPress Plugin [19]
20. 在你的 WordPress 插件中使用 Javascript 和 CSS [20]
21. Using AJAX with your WordPress Plugin [21]
22. 在你的 WP 插件中使用 AJAX [22]
23. Releasing and Promoting Your WordPress Plugin [23]
24. 发布并推广你的 WordPress 插件 [24]

原来的目录页面在：这里 [25]

还提供 PDF 版本的下载哦。

作者：Charles [26]

原文链接：插件开发全攻略（目录）[27]

《Becomin' Charles》[26]版权所有，转载时必须以链接形式注明作者和原始出处及本声明。

插件开发全攻略（01）---介绍

Posted By Charles On 2008 年 5 月 28 日 @ 15:15 In WordPress | No Comments

对于许多 WordPress [1]用户来说，插件是必须的。WordPress 插件 [2]使得那些只有很少或者没有编程能力的用户可以扩展他们博客的功能。插件的形式多种多样，在 WordPress [3]中，插件几乎可以做任何事情。

即便是 WordPress [3]这样一个优秀的独立应用程序，仍旧有许多方面有缺憾。用户所要求的越来越多的 WordPress [3]所应该具有的特性，都很可能被开发成插件。此外，还有很多没有实现的创意，而且，每一天还有更多的创意被提出。

在已经发布了三款插件之后（不包括我自己写给自己用的那些），我意识到了一些 WordPress [3]的局限性，我希望能够分享一些我已经学到的（我仍旧在学）关于创建 WordPress [3]插件的一些经验。最终，我将会开始写一个系列，专门讨论编写你自己的 WordPress [3]插件的过程中遇到的各种各样的问题。这个系列将会从非常基础的话题开始，并且假设你的插件知识是零基础。

这个系列是为谁准备的？

这个系列是为了任何一个好奇或者想要学习怎样编写他们自己的 WordPress [3]插件的用户撰写的。这个系列的读者应该有中级程度的 PHP 知识，知道一点点 JavaScript，并且有相当的 CSS 知识。

这个插件系列将会使主题模板设计者受益，还有那些想要修补插件代码的人，还有那些想从零开始编写自己的插件的人。

完成工作必备利器

使用任何文本编辑器都可以写插件。这里是一些我个人使用的创建插件的工具。

- Dreamweaver
- Firefox
- Firebug（Firefox 的插件） [4]
- Web Developer（Firefox 的插件） [5]
- XAMPP [6]和一个本地安装的 WordPress [1]

这个系列假设你使用的是 WordPress [3] 2.1.x 或者更新的版本。

代码范例

我所使用的所有的代码在每篇文章的结论一节都提供下载。我会随着内容推进，逐步完善我的代码，所以每一份下载都会不同。我将会创建一个实际上什么都做不了的插件，但是足够向你展示一个插件工作的基础了。

由于这个系列里的每篇文章都是在上一篇的基础之上，所以推荐按照顺序来阅读这个系列里面的文章。

我极力建议你使用本地 WordPress [3]来安装调试测试插件，而不是在一个你用来发表你的文章的正规的 WordPress [3]上。

话题

我计划从非常基础的内容开始，然后快速推进到更加核心的 WordPress [3]插件函数的内容。这个系列不会关注插件开发中太过细节的东西，但是希望能够给你一个很好的开始开发你自己的插件的基础。如果你有任何问题或者建议，请留言，或者使用 Devlounge Contact Form [7]插件给我发送 Email（译者：也可以给我留言）。我请你不要依赖 Devlounge [8]或者支持，而是使用 WordPress 支持论坛 [9]。

技术

我在我的代码范例中使用的某些技术可能不是最好的代码表达方式。你可能会对我没有使用许多的便捷写法而有所微词。我在这里提前抱歉。每个人有不同的编码风格。

只要是插件技术，结构，行为，或者其他讨厌的东西，如果有更好的方式被我忽略了，那么我非常乐意倾听。

插件开发全攻略（02）---编写插件的七个理由

Posted By Charles On 2008 年 5 月 29 日 @ 10:29 In WordPress | No Comments

当写作《插件开发全攻略》系列文章的时候，我想，首先罗列一些为什么 WordPress [1]用户想要编写一个 WordPress 插件 [2]的理由是有好处的。

下面罗列了为什么一个 WordPress [1]用户应该考虑编写一个 WordPress [1]插件的七个理由。

1. 你喜欢一个插件的创意，但是不喜欢这个插件的实现方式

无论是在 Weblog Tools Collection [3]还是在官方的 WordPress 插件目录 [4]或者 WordPress 插件数据库 [5]寻找插件，你将不可避免的找到一个插件，能够满足你的需求，但是只能在某种程度上满足你的需求。

你很喜欢这个插件的创意，但是却并不是那么喜欢插件作者使用的实现方法。为什么不创建你自己的分立的版本来实现你最初的那个创意呢？

2. 你想修改已经存在的插件的代码

有时候，一个插件的输出需要修改一下，或者你想要的某个功能正好没有。虽然你可以尝试说服插件作者来添加你要的特性，但是插件作者通常都非常忙或者他们不喜欢你的建议。一个插件作者要为一个免费插件提供支持，维护需要付出很多努力。有时候，一个插件很可能已经没有任何人去维护它了。

在插件作者不能满足你的需求的这种情况下，就全靠你自己采取主动来修改存在的插件代码了。如果你的工作做得足够好并且做了足够多的修改，你可以再次发布这个插件，只要原来这个插件发布于与 GPL 相兼容的许可 [6]就行。

通常当我安装或者测试一个新的插件的时候，我要做的第一件事情，就是查看一下代码，看看有什么我可以修改的，有什么我不可以修改的，我有可能往里面添加什么或者拿掉什么。

3. 你想要扩展一个插件

有时候一个插件本身很好，但是你想要在它的基础上建立并且发布你自己的版本。比如说，你可能认为一个插件在使用了 AJAX 后会工作得更好，或者添加更多的钩子以使它能够与其他插件兼容。你可能想要给它添加一个管理面板使得你不需要深入到代码里去修改输出。

就想前面提出的一样，如果一个插件是兼容 GPL 许可的，你可以自由地发布你自己的版本。

4. 你想要可移植的主题代码

对于像我们这样选择自己从零开始创建主题的玩家，你可能会发现你自己在很多地方重用着一段代码。编写你自己的插件，组合所有的代码小片段使得你可以像使用模板标签一样来使用那些代码，不是更好吗？

模板标签的美丽就在于你可以一次又一次地在你的主题或者任何一个你将来要创建的主题中使用它们。并且，你只需要在一个地方修改这些代码，而不是很多处地方。

5. 你是一个主题设计者

如果你是一个 WordPress [1]模板设计者，我可能会劝说你，符合逻辑的下一步是成为一个插件作者。编写插件将使得你对于 WordPress [1]的行为有更进一步的了解，接着，你就可以扩展那些你已经发布的模板的功能了。

6. 你想要赚钱

一个好的插件作者通常可以因为他的工作而获得回报。一些插件作者也可以获得捐赠或者为提供额外的支持或者咨询而收取费用。

7. 你想获得链接

当我发布 Reader Appreciation Project [7]的时候，我的其中一个目的就是快速建立传入链接。我知道的最好的方法就是写一些 WordPress [1]插件并且提高它们的功能。我的其中一个插件（WP Ajax Edit Comments [8]）竟然非常地流行，目前已经为我带来了超过 100 个传入链接了。

插件开发全攻略（03）---怎样获得 WordPress 插件的创意

Posted By Charles On 2008 年 5 月 30 日 @ 22:01 In WordPress | No Comments

如果你已经被说服而想要研究一下创建你自己的 WordPress [1]插件的可能性，那么找到一个可以让你开始行动的一个创意可能是非常困难的。幸运的是，有很多地方可以让你找到开发你自己的插件的灵感。在本文中，我会罗列好几个方法使你可以找到开发你自己的 WordPress [1]插件的创意。

倾听你的读者

你的读者是你获得插件创意的宝库。比如说，一个读者可能需要一个简单的方式来回复或者编辑评论。因为博客的读者是使用你博客最多的人，他们往往对于你的博客还缺乏哪些功能有着独特的洞察力。就在那天，我的一个读者请求我能不能让评论在发布前可以预览一下。幸运地是已经有一些插件可以提供这个功能了，但是有时候，你的读者会建议一些还没有被插件实现的功能。

倾听你自己的心声

"如果 WordPress [1]可以……就好了"

如果你发现 WordPress [1]缺少一个你非常需要的特性，为什么不自己写一个插件来实现呢？机遇就在于如果你渴望添加这个特性，那么别人也一样。

检查一下博客资源

像 The Blog Herald [2]和 Weblog Tools Collection [3]，正是插件创意的宝库。周三，The Blog Herald 有一个专栏，叫《WordPress [1]星期三》。这个专栏里面是插件需求和"愿望列表"。而 Weblog Tools Collection 几乎每一天都会发布一款插件，从那里，你可以了解人们需要什么样的插件。

检查 **WordPress** [1]支持论坛

WordPress 支持论坛 [4]里面到处都是寻找帮助以扩展他们的 WordPress [1]博客的人。一个典型的找寻插件创意的板块是 Requests and Feedback 论坛。另一个地方是 WordPress ideas page [5]。

研究 **API**

诸如 Flickr，FeedBurner，Google Maps 这样的在线服务，还有其他提供 API 服务，使得第三方应用拥有使用他们服务的能力。通过这些 API，你可以开始通过编程创建你自己的 WordPress [1]解决方案。

如果有一个你非常喜欢的服务，但是你想把它集成到你的 WordPress [1]中，那么调查一下这个服务的 API 看看能不能开发一个好的插件。

第三方应用

伴随着 WordPress [1]博客，人们可能已经装了许多的第三方应用。这一类的例子有 Mint [6]（译者注：网站分析程序），Vanilla [7]（译者注：论坛程序），其他还有许多。为什么不开发一个插件来集成第三方应用到一个 WordPress [1]博客呢？

存在的 WordPress [1]插件

如果你找到一个你非常喜欢的 WordPress [1]插件，并且想要发布一个带有你自己的创意的分支版本，那么请放手干吧。如果你不喜欢某个特定的插件的实现方式，建立你自己的实现吧。有许多的插件做得都是相同的事情，但是还是有一些细微地差别的。

插件开发全攻略（04）---WordPress 插件的结构

Posted By Charles On 2008 年 6 月 1 日 @ 19:32 In WordPress | No Comments

开发一个 WordPress [1]插件的一个更重要的方面，是你怎样设计它的结构。本文将研究几个关于设计插件结构的提示，以帮助你组织你的插件资源，避免名字冲突。每一个插件作者的插件的结构都不尽相同，所以这些提示只是我的个人偏好。我将首先简单地描述一下一个 WordPress [1]插件是怎样工作的，然后介绍一个插件的结构。

WordPress [1]插件怎样工作

在将一个插件放入到 wp-content/plugins [2]/目录后，插件应该自动的处于可以安装的状态 [3]。

当一个插件被"启用"，等同于告知 WordPress [1]将你的代码装载到"每"个页面（包括管理页面）。这也就是为什么当你启用了很多的插件的时候，你的 WordPress [1]可能非常慢的原因，这是由它所引入的代码的量决定的。

从你的插件被启用，WordPress [1]自动将你的代码装载后开始，你可以利用 WordPress 插件 API [4]。你还可以使用 WordPress 模板标签 [5]或者创建你自己的函数。

如果你计划开发一个改变文章内容或者评论的插件，我建议你读一下 WordPress loop [6]。WordPress [1] loop 是一个显示你的文章的循环。有些模板标签在这个循环外面无法工作，所以，你准确地知道你代码在哪里执行是非常必要的。你可通过使用 Actions [7]和 Filters [8]来控制这一点，这将会在将来的文章中解释。

文件夹结构

所有的 WordPress [1]插件都会被安装到 wp-content/plugins [2]目录中。有些插件作者的插件只包含一个文件，但是我推荐你总是创建一个文件夹来保存你的插件。

典型地，我会把我的插件放在如下的目录结构中：

- 插件文件夹名称（你插件的名字，没有空格或者特殊字符）
  - 插件 PHP 文件
  - js 文件夹（存放 javascript 文件）
  - css 文件夹（存放样式表文件）
  - php 文件夹（存放其他的 PHP 文件）

举例来说，这是一个我创建的样例结构：

- devlounge-plugin-series

- devlounge-plugin-series.php
- js
- css
- php

在 devlounge-plugin-series 文件夹中，我将只包含主要的 PHP 文件，把其他的文件都放到他们各自属于的文件夹中。这个结构将帮助其他插件作者在查看你的代码的时候，能够分辨哪个是主要的插件文件，哪些是支持插件正常工作的附属文件。

WordPress [1]还建议将图片放到一个文件夹中，并且包含一个 readme 文件在你的插件中。

主要插件文件

当你开始编写一个插件的时候，前面七行用来描述你的插件。

```
1. <?php
2. /*
3. Plugin Name: Your Plugin Name Here
4. Plugin URI: Your Plugin URI
5. Version: Current Plugin Version
6. Author: Who Are You?
7. Description: What does your plugin do?
8. */
```

第 3 行是让你命名你的插件的。第 4 行是向用户指出插件所在的网址的。第 5 行让你指定当前的版本。第 6 行让你设定插件的作者。第 7 行是对插件的描述。

下面展示的是一个已经填写完毕的范例：

```
1.  /*
2. Plugin Name: Devlounge Plugin Series
3. Plugin URI: http://www.devlounge.net/
4. Version: v1.00
5. Author: Ronald Huereca
6. Description: A sample plugin for a Devlounge series.
7. */
8. ?>
```

下面展示的是插件在 WordPress [1]插件面板上出现的截图。

设定一个类结构

开发一个插件，并不需要你对 PHP 类 [9]了若指掌，但是如果你真的很熟悉，那会很有帮助。为了避免与其他的 WordPress [1] 插件发生名字冲突，一个类结构是必须的。如果别人在插件当中使用了和你一样的函数名，那么就会发生一个错误，WordPress [1]可能会无法响应直到你删除那个插件。

为了避免名字冲突，所有的插件都使用一个 PHP 类结构是急需的。这里有一些"骨架"代码，可以帮助你创建一个类结构。

```
1.  if (!class_exists("DevloungePluginSeries")) {
2.      class DevloungePluginSeries {
3.          function DevloungePluginSeries() { //constructor
4.          }
5.      }
6.  } //End Class DevloungePluginSeries
7.  ?>
```

上述代码的含义是，首先检查是否有一个名叫 DevloungePluginSeries 的类存在。如果这个类不存在，那么创建这个类。

初始化你的类

下面的代码将会初始化你的类。

```
1.  if (class_exists("DevloungePluginSeries")) {
2.      $dl_pluginSeries = new DevloungePluginSeries();
3.  }
4.  ?>
```

上面的代码检查是否类 DevloungePluginSeries 已经存在了。如果已经存在，那么一个变量$dl_pluginSeries 将会被创建，并且会使用一个 DevloungePluginSeries 类的对象给它赋值。

设置 Actions 和 Filters

下面的代码段是用来放置 WordPress [1] Actions 和 Filters 的地方（我会在后续的文章中详细讲）。

```
1.  //Actions and Filters
2.  if (isset($dl_pluginSeries)) {
3.      //Actions
4.      //Filters
5.  }
6.  ?>
```

上述代码首先确认$dl_pluginSeries 是否已经被设定。如果设定了（仅当类存在的时候），那么就会设定合适的 actions 和 filters。

插件开发全攻略（05）---WordPress 插件 Actions

Posted By Charles On 2008 年 6 月 3 日 @ 14:47 In WordPress | 1 Comment

WordPress actions [1]允许作为插件作者的你插入到 WordPress [2]应用中并且执行一段代码。一个 Action 的例子就是，你想要在一个用户发布完一篇文章或者留下一篇留言的时候执行一个动作。

一些我使用极其频繁的 Action 有：

- admin_menu：允许你给你的插件设置一个管理面板。
- wp_head：允许你将代码插入到博客的<head>标签内。

Action 在行动

当定义一个 WordPress 插件的结构 [3]的时，我为某些 Action 留下了一块地方。在这个例子中，我们将使得一段代码可以在 WordPress [2]博客中的<head>标签内运行。

首先，我们需要在 **DevloungePluginSeries** 类中添加一个函数：

```
1.  function addHeaderCode(){
2.      <!-- Devlounge Was Here-->
3.  }
4.  ?>
```

上面一段代码的作用是输出 HTML 注释。相当简单，但是你实际上可以输出任何东西。为了调用这个函数，我们要添加一个 Action。

```
1.  //Actions and Filters
2.  if (isset($dl_pluginSeries)){
3.      //Actions
4.      add_action('wp_head',array(&$dl_pluginSeries,'addHeaderCode'),1);
5.      //Filters
6.  }
7.  ?>
```

从 WordPress 插件 API [4]中我们可以知道，add_action 的结构如下：

```
add_action('hook_name','your_function_name',[priority],[accepted_args]);
```

由于我们是在一个类的内部调用一个函数，我们传递给 Action 一个数组，里面包含我们的类对象的引用（dl_pluginSeries）和我们想要调用的函数名（addHeaderCode）。我们已经给我们的插件设定了一个优先级为 1，数字越小，执行时间越靠前。

运行代码

如果 Devlounge Plugin Series 插件被启用了，使用【查看->源文件】应该可以在你的博客的源码中看到注释“Devlounge was here”。

移除 Action

如果你的插件动态地添加 Action，你也可以使用 remove_action 来动态地移除 action。用法如下：

remove_action('action_hook','action_function')。

插件开发全攻略（06）---WordPress 插件 Filter

Posted By Charles On 2008 年 6 月 4 日 @ 18:48 In WordPress | No Comments

Filter [1]是一组使得你的插件可以插入来修改文字的函数。被修改的文字通常是要插入到数据库或者显示给终端用户看的。

WordPress [2] Filter 允许你修改几乎任何类型的显示文字，而且其功能十分强劲。通过 Filter 你可以修改文章，feed，怎么样在评论中的作者，还有很多，很多。

为了说明 WordPress [2] Filter 的用处，我们会继续在已经存在的 Devlounge Plugin Series 代码上工作。

添加一个内容 Filter

有一个你可以使用的很 Cool 的 Filter 是'the_content'。这个 filter 在文章内容被显示在浏览器之前执行。我们将要添加一行文字到文章内容的末尾。

根据 WordPress 插件 API [3]，添加一个 filter 的格式为：

add_filter('hook_name','your_filter',[priority],[accepted_args]);

我们只需要往 **DevloungePluginSeries** 类中添加一个函数。让我叫它 **addContent**。

```
1.  function addContent($content=''){
2.      $content.="<p>Devlounge Was Here</p>";
3.      return $content;
4.  }
5.  ?>
```

在上面的代码中，发生了以下的事件：

- 上述函数会接受一个参数叫做 **content**
- 如果没有参数传递尽力，那么会设定默认值（空串）
- 参数 **content** 被附加了一行我们自定义的文本
- 然后重新返回了 content

在向类添加了上述函数后，下一步要做的就是将它插入到'**the_content**'中，使得这个函数被调用。

```
1.  //Actions and Filters
```

```
2.  if(isset($dl_pluginSeries)){
3.      //Actions
4.      add_action('wp_head',array(&$dl_pluginSeries,'addHeaderCode'),1);
5.      //Filters
6.      add_filter('the_content',array(&$dl_pluginSeries,'addContent'));
7.  }
8.  ?>
```

在这段代码的第 6 行你可以看到，添加了一个叫做**'the_content'**的 filter，通过这个 filter，我们的函数**'addContent'**会被调用。

如果插件启用，当一篇文章被显示的时候，文本"Devlounge Was Here"将会被显示到文章的末尾。

添加一个作者 **Filter**

我在这里要展示另一个使用 Filter 的例子，我要展示一下怎么操纵显示一个评论的作者。我这里只是简单地让所有的作者名字都用大写显示。

我们只需要往 **DevloungePluginSeries** 类中添加一个函数。我们叫它 **authorUpperCase**。

```
1.   function authorUpperCase($author=''){
2.      return strtoupper($author);
3.  }
4.  ?>
```

在上述代码中，做了这些事情：

上述函数会接受一个参数，叫做 **author**

如果没有参数，会被设定成默认值

author 字符串会以大写的形式返回

在函数被添加到类后，下一步是将它插入到**'get_comment_author'**中使它被调用。

```
1.   //Actions and Filters
2.  if(isset($dl_pluginSeries)){
3.      //Actions
4.      add_action('wp_head',array(&$dl_pluginSeries,'addHeaderCode'),1);
5.      //Filters
6.      add_filter('the_content',array(&$dl_pluginSeries,'addContent'));
7.      add_filter('get_comment_author',array(&$dl_pluginSeries,'authorUpperCase'));
8.  }
9.  ?>
```

像你在第 7 行看到的，我们添加了一个'**get_comment-author**' filter，然后，我们的函数'**authorUpperCase**'会被调用 。

如果插件已经启用，并且一篇日志的评论是可见的，那么评论的作者名字会以大写显示。

应用 **Filter**

使用 Filter 可以做的更强大的是，你可以动态地调用他们。没有必要每次运行都添加一次 filter。你可以在任何时候在你的代码里面调用 filter。

apply_filters 的格式是：apply_filter('filter name','you text');

你会在这个系列的后续文章中看到 **apply_filters** 的例子。

插件开发全攻略（07）---构造一个 WordPress 插件管理员面板

Posted By Charles On 2008 年 6 月 6 日 @ 16:26 In WordPress | 5 Comments

任何需要用户输入（诸如改变一个变量）的插件，都需要某种管理面板。建立一个管理面板 [1]，并不是那么难的，所以，一个插件作者决定不创建管理面板，而是让用户自己去修改 PHP 代码的行为让我很是苦恼。让一个用户（TA 的 PHP 只是可能是 0）去修改代码通常来说不是一个好主意。本文将深入探讨成功地为你的插件创建管理面板，到底需要些什么。

[2]

存储变量的地方

当你想要给你的插件创建一个管理面板的时候，你首先会碰到的问题之一就是到底在哪里存储变量值。非常幸运，WordPress [3] 通过 options 使得这件事变得非常容易。我将会在后续的系列文章中解释 options 和数据库存储。现在来说，所有你需要做的事情就是点点你的头，然后跟着我的指导，把你自己的管理变量存储到 WordPress [3]数据库中。

通常我考虑到 options 的时候做的第一件事情就是，给我的管理 options 取一个独特的名字。我把这个名字以成员变量的形式存储在我的类中。在这个 Devlounge Plugin Series 插件的例子中，我把这个变量声明添加到 **DevloungePluginSeries** 类中。

命名你的管理 **Options**

```
class DevloungePluginSeries {
        var $adminOptionsName = "DevloungePluginSeriesAdminOptions";
        function DevloungePluginSeries() { //constructor
        }
```

第 2 行显示了我存储我的成员变量的地方。我将我的变量命名为 **adminOptionsName** 并且给予它一个很长而且独特的值 **DevloungePluginSeriesAdminOptions**。

给你的管理 **Options** 设定缺省值

你需要一个地方来初始化你的管理选项，尤其是当一个用户第一次激活你的插件的时候。然而，你还必须包正这些 options 在升级中是安全的，以防万一你决定将来添加更多的 options。我使用的技巧是，提供一个专门的函数来调用你的管理 options。你的插件的需求可能不一样，但是绝大多数的管理面板不会难以置信地复杂，所以一个函数对你的管理 options 来说，已经足够了。

下面这个函数是我插入到类 DevloungePluginSeries 中的：

```
1.    //Returns an array of admin options
2.          function getAdminOptions() {
3.              $devloungeAdminOptions = array('show_header' => 'true',
4.                  'add_content' => 'true',
5.                  'comment_author' => 'true',
6.                  'content' => '');
7.              $devOptions = get_option($this->adminOptionsName);
8.              if (!empty($devOptions)) {
9.                  foreach ($devOptions as $key => $option)
10.                     $devloungeAdminOptions[$key] = $option;
11.             }
12.             update_option($this->adminOptionsName, $devloungeAdminOptions);
13.             return $devloungeAdminOptions;
14.         }
```

```
15.  ?>
```

这个函数做的事情是：

- 给你的管理 options 赋默认值（3-6 行）
- 尝试从数据库中找出以前存储过的值（第 7 行）
- 如果 options 曾经存储过，那么存储的值会覆盖默认值（8-11 行）
- options 被存储到 WordPress [3]数据库中（第 12 行）
- options 被返回供你使用（第 13 行）

初始化管理 Options

getAdminOptions 函数可以在任何时候调用来获得管理 options。然而，当插件第一次安装的时候呢？应该有某个函数被调用也能获得管理 options。我在 DevloungePluginSeries 类中添加了这个函数：

```
1.  function init() {
2.      $this->getAdminOptions();
3.  }
4.  ?>
```

短小简单。但是必须有一个 action 来调用这个函数。

```
add_action('activate_devlounge-plugin-series/devlounge-plugin-series.php',  array(&$dl_pluginSeries, 'init'));
```

这个 action 有些复杂，但是很容易理清楚。下面是这个 action 做的事情：

- 当一个插件被激活的时候你告诉它运行
- 你给它主要的插件 PHP 文件，就是 devlounge-plugin-series/devlounge-plugin-series.php。这当然是假设你把插件放到了 wp-content/plugins [4]/目录中。
- 你传递了一个引用给实例变量 dl_pluginSeries 并且调用了 init 函数。

所以，任何时候插件被激活，init 函数都会被调用。

管理面板怎样工作

在我深入到创建实际的管理面板的代码的时候，描述一下管理面板的行为是有好处的。这里有一些你设定你管理面板的步骤：

- 检查任何表单数据是否已经被提交。
- 如果表单数据存在，输出提醒。
- 显示管理面板 Options。

在管理面板中，一个非常困扰你的问题可能是_e 的使用。_e 函数允许 WordPress [3]搜索你文本的本地化版本。这将帮助 WordPress [3]在将来翻译你的插件。这个函数的工作类似一个普通的 echo，但是你传给它的是你的文本和一个标识变量（典型的是你的插件名称）。一个例子：

```
_e('Update Settings','DevloungePluginSeries');
```

这句代码和：

```
echo "Update Settings";
```

是相同的。

设置你的管理面板函数

我们想要做的第一件事情就是设定一个函数，来真正地打印出一个管理面板。函数名字叫做 printAdminPage。下一段代码将要读入我们早先设定的 options 并且检查是否所有的发布 options 已经被提交。这一段的所有代码假设写在 printAdminPage 函数中。

```
1.  //Prints out the admin page
2.      function printAdminPage() {
3.          $devOptions = $this->getAdminOptions();
4.          if (isset($_POST['update_devloungePluginSeriesSettings'])) {
5.              if (isset($_POST['devloungeHeader'])) {
6.                  $devOptions['show_header'] = $_POST['devloungeHeader'];
7.              }
8.              if (isset($_POST['devloungeAddContent'])) {
9.                  $devOptions['add_content'] = $_POST['devloungeAddContent'];
10.             }
11.             if (isset($_POST['devloungeAuthor'])) {
12.                 $devOptions['comment_author'] = $_POST['devloungeAuthor'];
13.             }
14.             if (isset($_POST['devloungeContent'])) {
15.                 $devOptions['content'] = apply_filters('content_save_pre',
    $_POST['devloungeContent']);
16.             }
17.             update_option($this->adminOptionsName, $devOptions);
18.             ?>
19.  <div class="updated"><p><strong><?php _e("Settings Updated.",
    "DevloungePluginSeries");?></strong></p></div>
20.             <?php
21.  }
22.  ?>
```

上面所有代码做的是装载 options 并且测试是否表单的每个部分都已经被正确提交。if 语句并不是必须的，但是有的时候对于调试来说很有用。第一被测试的表单变量是 update_devloungePluginSeriesSettings。这个变量被赋值给我们的“提交”按钮。如果它没有被设定，那么说明这个表单没有被提交。

就像说好的一样，在第 16 行，我使用了 apply_filters 函数来格式化内容，以便存入数据库。

下一段代码将会显示一个 HTML 表单，那是管理面板锁必须的。它有一些复杂，所以我会这里概括一下。所有的代码做的事情，就是显示表单元素和读入 options。

```
<div class=wrap>
<form method="post" action="<?php echo $_SERVER["REQUEST_URI"]; ?>">
<h2>Devlounge Plugin Series</h2>
<h3>Content to Add to the End of a Post</h3>
<textarea name="devloungeContent" style="width: 80%; height: 100px;"><?php _e(apply_filters('format_to_edit',$devOptions['content']), 'DevloungePluginSeries') ?></textarea>
<h3>Allow Comment Code in the Header?</h3>
<p>Selecting "No" will disable the comment code inserted in the header.</p>
<p><label for="devloungeHeader_yes"><input type="radio" id="devloungeHeader_yes" name="devloungeHeader" value="true" <?php if ($devOptions['show_header'] == "true") { _e('checked="checked"', "DevloungePluginSeries"); }?> /> Yes</label>  <label for="devloungeHeader_no"><input type="radio" id="devloungeHeader_no" name="devloungeHeader" value="false" <?php if ($devOptions['show_header'] == "false") { _e('checked="checked"', "DevloungePluginSeries"); }?>/> No</label></p>
<h3>Allow Content Added to the End of a Post?</h3>
<p>Selecting "No" will disable the content from being added into the end of a post.</p>
<p><label for="devloungeAddContent_yes"><input type="radio" id="devloungeAddContent_yes" name="devloungeAddContent" value="true" <?php if ($devOptions['add_content'] == "true") { _e('checked="checked"', "DevloungePluginSeries"); }?> /> Yes</label>  <label for="devloungeAddContent_no"><input type="radio" id="devloungeAddContent_no" name="devloungeAddContent" value="false" <?php if ($devOptions['add_content'] == "false") { _e('checked="checked"', "DevloungePluginSeries"); }?>/> No</label></p>
<h3>Allow Comment Authors to be Uppercase?</h3>
<p>Selecting "No" will leave the comment authors alone.</p>
<p><label for="devloungeAuthor_yes"><input type="radio" id="devloungeAuthor_yes" name="devloungeAuthor" value="true" <?php if ($devOptions['comment_author'] == "true") { _e('checked="checked"', "DevloungePluginSeries"); }?> /> Yes</label>  <label for="devloungeAuthor_no"><input type="radio" id="devloungeAuthor_no" name="devloungeAuthor" value="false" <?php if ($devOptions['comment_author'] == "false") { _e('checked="checked"', "DevloungePluginSeries"); }?>/> No</label></p>
<div class="submit">
<input type="submit" name="update_devloungePluginSeriesSettings" value="<?php _e('Update Settings', 'DevloungePluginSeries') ?>" /></div>
</form>
 </div>
                    <?php
                }//End function printAdminPage()
```

需要在上述代码中观察的是 options 的引用，和 HTML 和 PHP 是怎样集成的。

设置管理面板 **Action**

现在，printAdminPage 函数已经添加了，我们需要通过一个 action 调用它。首先函数必须被设定在正好在 action 的上面，在类的范围之外。

```
1.  //Initialize the admin panel
2.  if (!function_exists("DevloungePluginSeries_ap")) {
3.      function DevloungePluginSeries_ap() {
4.          global $dl_pluginSeries;
5.          if (!isset($dl_pluginSeries)) {
6.              return;
7.          }
8.          if (function_exists('add_options_page')) {
9.      add_options_page('Devlounge Plugin Series', 'Devlounge Plugin Series', 9,
    basename(__FILE__), array(&$dl_pluginSeries, 'printAdminPage'));
10.         }
11.     }
12. }
13. ?>
```

上面的代码做了下面这些事：

- 创建了一个叫做 DevloungePluginSeries_ap 的函数。
- 测试变量 dl_pluginSeries 是否存在（4-7 行）。这个变量是我们的类的引用。
- 一个叫做“Devlounge Plugin Series”的管理页面被初始化了，并且，我们的 printAdminPage 函数被引用了。（8-10 行）。

add_options_page 函数的调用方法为：

add_options_page(page_title,menu_title,access_level/capability,file,[function]);

访问级别（在这个例子中是 9）在 WordPress [3] Codex 的 Users Levels page 页面有详细的描述。

必须设置一个 action 来调用 DevloungePluginSeries_ap 函数：

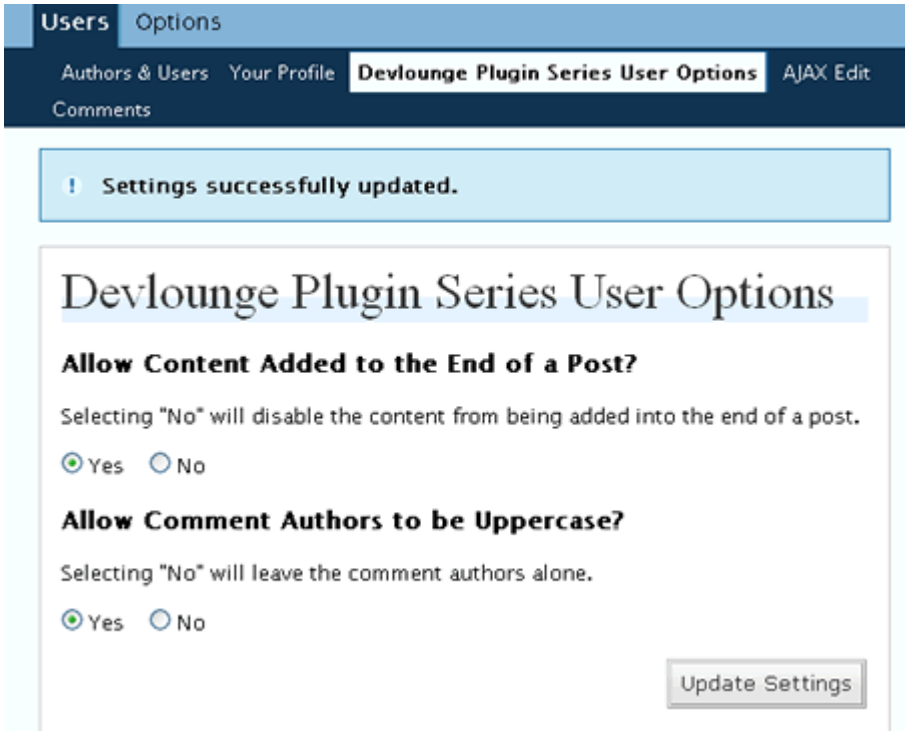
```
add_action('admin_menu', 'DevloungePluginSeries_ap');
```

插件开发全攻略（08）---构建一个 WordPress 插件用户面板

Posted By Charles On 2008 年 6 月 22 日 @ 18:16 In WordPress | 1 Comment

将会有这么一种情况，你将有一个主要的管理面板，但是希望能够让独立的用户设定他们自己的偏好。在 Devlounge Plugin Series 这个例子中，我们添加了一个是否把文字添加到文章末尾的选项。然而，假如一个登录用户不希望看到这段文字呢？为什么不给他们一个选择，而且不影响到所有其他的用户呢？

这篇文章将会涉及到这个问题，让你可以添加你自己的用户面板。



命名你的选项

```
1.  class DevloungePluginSeries {
2.      var $adminOptionsName = "DevloungePluginSeriesAdminOptions";
3.      var $adminUsersName = "DevloungePluginSeriesAdminUsersOptions";
4.  ?>
```

第 3 行显示，我添加了一个成员函数，叫做 **adminUsersName**。我给这个变量取了一个长而且独特的名字 **DevloungePluginSeriesAdminUsersOptions**。

设定你的缺省用户选项

你将需要一个地方来初始化你的选项，尤其是当一个用户第一次激活了你的插件的时候。然而，这些选项在管理面板之外也应该起作用，不论用户是登录了还是没有登录。

这里是一个我引入到 **DevloungePluginSeries** 类中的函数：

```
1.  //Returns an array of user options
2.      function getUserOptions() {
3.          global $user_email;
4.          if (empty($user_email)) {
5.              get_currentuserinfo();
6.          }
7.          if (empty($user_email)) { return ''; }
```

```
8.              $devOptions = get_option($this->adminUsersName);
9.              if (!isset($devOptions)) {
10.                 $devOptions = array();
11.             }
12.             if (empty($devOptions[$user_email])) {
13.                 $devOptions[$user_email] = 'true,true';
14.                 update_option($this->adminUsersName, $devOptions);
15.             }
16.             return $devOptions;
17.         }
18.  ?>
```

这个函数完成以下任务：

- 检查用户是否已经登陆（3-7 行）。这个检查仅需要查看变量 user_email 是否设定即可。
- 尝试找到以前存储在数据库中的选项（第 8 行）。
- 如果没有找到 option，设定默认值（第 9-15 行）。
- 返回 option 供你使用

初始化管理用户 **option**

可以在任何时候调用 getUserOptions 来取得用户管理 options。然而，那么插件第一次安装的时候呢？应该有某种函数被调用，以取得用户选项。我添加了下面这个函数到 init 函数：

```
1.    function init() {
2.        $this->getAdminOptions();
3.        $this->getUserOptions();
4.    }
5.    ?>
```

第 3 行调用了新函数 getUserOptions。由于已经有了一个 action 调用了 init 函数，不需要额外的步骤了。

怎样使管理面板和用户面板一起工作

你会回想起上一篇关于设定一个管理面板 [1]的文章中，WordPress [2]管理员可以设定文章末尾的内容，代码是否是否在头部显示，评论中的作者名字是否大写。而用户面板允许那些不是管理员的用户有能力设定他们是否希望这些 options。

我们将允许用户决定，如果他们：

- 想要内容在文章的末尾显示（仅当管理员启用了这个 option）
- 想要评论作者的名字大写（仅当管理员启用了这个 option）

设定用户面板函数

我们要做的第一件事情就是编写一个函数来显示用户面板。函数名字为 printAdminUsersPage。下一段代码将会读入我们早先设定的 options 并且检查一下是否有任何文章选项已经被提交。这一节中所有的代码都默认包含于 printAdminUsersPage 函数。

```
1.  //Prints out the admin page
2.  function printAdminUsersPage() {
3.      global $user_email;
4.      if (empty($user_email)) {
5.          get_currentuserinfo();
6.      }
7.      $devOptions = $this->getUserOptions();
8.      //Save the updated options to the database
9.      if (isset($_POST['update_devloungePluginSeriesSettings']) &&
    isset($_POST['devloungeAddContent']) && isset($_POST['devloungeAuthor'])) {
10.         if (isset($user_email)) {
11.             $devOptions[$user_email] = $_POST['devloungeAddContent'] . "," .
    $_POST['devloungeAuthor'];
12.             ?>
13.             <div class="updated"><p><strong>Settings successfully
    updated.</strong></p></div>
14.             <?php
15.             update_option($this->adminUsersName, $devOptions);
16.         }
17.     }
18.     //Get the author options
19.     $devOptions = $devOptions[$user_email];
20.     $devOptions = explode(",", $devOptions);
21.     if (sizeof($devOptions)>= 2) {
22.         $content = $devOptions[0];
23.         $author = $devOptions[1];
24.     }
25.             ?>
```

上述代码：

- 取回用户 options（第 7 行）
- 存储发送的数据到数据库（第 9-18 行）
- 为用户读入用逗号分隔的变量

下一段代码将会显示用户面板必须的 HTML 表单。所有的代码做的事情就是显示表单元素，并且读入已经取得的选项。

```
1.  <div class=wrap>
2.  <form method="post" action="<?php echo $_SERVER["REQUEST_URI"]; ?>">
3.  <h2>Devlounge Plugin Series User Options</h2>
4.  <h3>Allow Content Added to the End of a Post?</h3>
```

```
<p>Selecting "No" will disable the content from being added into the end of a post.</p>
<p><label for="devloungeAddContent_yes"><input type="radio" id="devloungeAddContent_yes" name="devloungeAddContent" value="true" <?php if ($content == "true") { _e('checked="checked"', "DevloungePluginSeries"); }?> /> Yes</label>  <label for="devloungeAddContent_no"><input type="radio" id="devloungeAddContent_no" name="devloungeAddContent" value="false" <?php if ($content == "false") { _e('checked="checked"', "DevloungePluginSeries"); }?>/> No</label></p>
<h3>Allow Comment Authors to be Uppercase?</h3>
<p>Selecting "No" will leave the comment authors alone.</p>
<p><label for="devloungeAuthor_yes"><input type="radio" id="devloungeAuthor_yes" name="devloungeAuthor" value="true" <?php if ($author == "true") { _e('checked="checked"', "DevloungePluginSeries"); }?> /> Yes</label>  <label for="devloungeAuthor_no"><input type="radio" id="devloungeAuthor_no" name="devloungeAuthor" value="false" <?php if ($author == "false") { _e('checked="checked"', "DevloungePluginSeries"); }?>/> No</label></p>
<div class="submit">
<input type="submit" name="update_devloungePluginSeriesSettings" value="<?php _e('Update Settings', 'DevloungePluginSeries') ?>" /></div>
</form>
</div>
                    <?php
                }//End function printAdminUsersPage()
```

设定用户面板 **Action**

当设定管理面板[1]的时候，我们定义了一个函数叫做 DevloungePluginSeries_ap，以帮助初始化管理面板。我们现在将要再次求助于此函数来添加我们的用户面板。

```
//Initialize the admin and users panel
if (!function_exists("DevloungePluginSeries_ap")) {
    function DevloungePluginSeries_ap() {
        global $dl_pluginSeries;
        if (!isset($dl_pluginSeries)) {
            return;
        }
        if (function_exists('add_options_page')) {
            add_options_page('Devlounge Plugin Series', 'Devlounge Plugin Series', 9, basename(__FILE__), array(&$dl_pluginSeries, 'printAdminPage'));
        }
        if (function_exists('add_submenu_page')) {
            add_submenu_page('profile.php', "Devlounge Plugin Series User Options","Devlounge Plugin Series User Options", 0, basename(__FILE__), array(&$dl_pluginSeries, 'printAdminUsersPage'));
        }
    }
```

```
15.  }
16.  ?>
```

在第 12 行，你能看到一行代码：

- 往 profile.php 页面中添加了一个子菜单
- 让一个用户级别高于或者等于 0 的用户访问用户面板
- 调用我们的 printAdminUsersPage 函数

关于访问级别的（此例中为 0）更详细的描述，可以参考用户级别 [3]。

插件开发全攻略（09）---WordPress 插件和数据库交互

Posted By Charles On 2008 年 6 月 23 日 @ 15:58 In WordPress | 3 Comments

当你编写一个插件的时候，你将不可避免地要将一些变量存储到数据库，或者将它们从数据库中取出。幸运的是，WordPress [1] 通过 options 和一个数据库对象，使得存取数据变得很简单。本文将会谈及如何在一个 WordPress [1]数据库中存储或者取回数据。

在数据库中存储数据

将数据存储到 WordPress [1]数据库，主要有两种方法：

1. 创建你自己的表。
2. 使用 Options

由于绝大多数插件不需要它们自己的表，所以，我将只讨论使用 options 的方法。然而，WordPress [1] Codex 上面详细讨论了怎样设定你自己的表的方法。

WordPress [1] Options

使用 WordPress [1] Options，在数据库存储和取回数据就跟函数调用一样简单。WordPress [1]为 options 提供了四个函数：

- add_option
- get_option
- update_option
- delete_option

add_option

add_option 函数接受四个参数，option 的名字是必须的。四个参数是：

```
add_option($name, $value, $description, $autoload);
```

使用这个函数来添加将来要从数据库中取出的数据是很有好处的。

参数 **$name** 必须是独一无二的，否则你就会覆盖别人的 option，或者别人会覆盖你的 option。

我通常不用这个函数，因为 **update_function** 完全可以做相同的事情。

get_option

get_option 函数允许你取回事先存储在数据库里的 option。它只接受一个参数，就是 option 的名字。函数的格式是：

```
get_option($option_name);
```

update_option

update_option 函数工作方式和 **add_option** 是一样的，除此之外，如果 option 已经存在，该函数会更新 option 的值。当往数据库中存储数据的时候，我个人喜欢使用这个双重功能的函数，尤甚于 **add_option** 。

delete_option

delete_option 函数从数据库中删除 options。函数的格式是： `delete_option($option_name);`

一个代码范例

你可能会回忆起这个系列以前的文章中，我把 options 以 array 的形式存入数据库。这里是一个例子，并且还有一些说明分析：

```
1.  //Returns an array of admin options
2.  function getAdminOptions() {
3.  $devloungeAdminOptions = array('show_header' => 'true',
4.  'add_content' => 'true',
5.  'comment_author' => 'true',
6.  'content' => '');
7.  $devOptions = get_option($this->adminOptionsName);
8.  if (!empty($devOptions)) {
9.  foreach ($devOptions as $key => $option)
10. $devloungeAdminOptions[$key] = $option;
11. }
12. update_option($this->adminOptionsName, $devloungeAdminOptions);
13. return $devloungeAdminOptions;
14. }
15. ?>
```

在第 3-6 行，我创建了一个最终要作为 option 存入 WordPress [1]数据库（ 12 行）的数组。我这么做是因为我不需要存储多个选项（每个选项都要查询一次数据库）。这个技术对代码臃肿，数据库查询和名字冲突都有所助益。

WordPress [1] 数据库类

另一个在 WordPress [1]数据库存储和取回数据的强大的方法是使用 WordPress [1]数据库类对象。在一个函数中，这个类对象的引用方式如下：

```
1.  function sample_function() {
2.  global $wpdb;
3.  }
4.  ?>
```

在这个变量被引用了以后，你可以访问《wpdb 类中许多有用的函数 [2]》。

举例来说，假如我们想要取回 WordPress [1]博客的评论的总数。这是一个函数，通过使用 WPDB 类来实现这个目的：

```
1.  function sample_function() {
2.  global $wpdb;
3.  $comments = $wpdb->get_row("SELECT count(comment_approved) comments_count FROM
    $wpdb->comments where comment_approved = '1' group by comment_approved", ARRAY_A);
4.  echo $comments['comments_count'];
5.  }
6.  ?>
```

上述函数完成了如下工作：

- 第 2 行，我们添加了一个 $wpdb 的引用。
- 第 3 行，我们调用了一个 wpdb 类内部的函数 **get_row**
- 第 3 行，我们从评论表（ $wpdb->comments ）中取回数据。这里我们指定返回的数据为一个联合数组（ARRAY_A）。
- 第 4 行，我们打印出结果。因为我希望数据的放回形式是联合数组，我只要调用我在 SQL 语句中赋值的变量就可以了，也就是 **comments_count**

wpdb 类是一个有着许多功能的非常大的类。我建议查看一下 WPDB 类页面 [2] ，看看 **wpdb 类**到底可以干些什么。

插件开发全攻略（10）---在你的 WordPress 插件中使用 Javascript 和 CSS

Posted By Charles On 2008 年 6 月 25 日 @ 21:24 In WordPress | 1 Comment

现今的许多插件对 javascript 和层叠样式表依赖更多了。将你插件中的 javascript 和 css 放置到分离的文件中是非常重要的，那样做会使插件维护起来更加容易。此系列中的这个部分将介绍怎样在你的插件中加载 javascript 和 CSS 文件。

添加你的 javascript

你的插件可能需要装载 prototype 类库，或者一个自定义的脚本。这一节将向你展示几个 WordPress [1]函数，它们可以帮助你装载脚本，并且避免脚本冲突。

wp_register_script 函数

wp_register_script 函数允许你注册要调用的脚本，并且允许你设定先决条件。比如说，如果你的脚本需要 prototype 事先加载，那么你可以通过这个函数来指定。

这里是 **wp_register_script** 函数的参数：

```
wp_register_script($handle, $src, $deps=array(), $ver=false);
```

- **handle** 是一个独一无二的名字，后面会用此名字引用你的脚本。这个参数是必须的。
- **src** 是你的 javascript 文件的绝对路径。这个参数是必须的。
- **deps** 是一个依赖数组。比如说，如果你的脚本需要使用 prototype，你要把它罗列在这里。这个参数是可选的。
- **ver** 是一个字符串，标明了脚本的版本。这个变量是可选的。

举个例子，如果你有一个脚本要装载：*http://yourdomain.com/wp-content/plugins [2]/your-plugin-directory/js/script.js*

让我们做一些假设：

- 你希望 handle 的名字为“my_script_handle”。
- 你的脚本依赖于 prototype 类库
- 你的版本是 1.0

你将在你的插件代码初始化调用那个函数，或者在 **wp_head** action 之后调用：

```
wp_register_script('my_script_handle','http://yourdomain.com/wp-content/plugins/your-plugin-directory/js/script.js', array('prototype'),'1.0');
```

wp_enqueue_script 函数

wp_enqueue_script 函数除了 **src** 参数是可选的以外，其他与 **wp_register_script** 是一样的。如果提供一个 **src**，入队函数会*自动地注册*脚本，因此使用 **wp_register_script** 函数并不是必须的。然而，**wp_register_script** 函数允许你手动注册你的脚本，这样，你可以仅使用一次 **wp_enqueue_script** 函数就将你所有的脚本装载。

如果我们像上一个例子中那样调用脚本，那么看起来像下面的样子：

```
wp_enqueue_script('my_script_handle','http://yourdomain.com/wp-content/plugins/your-plugin-directory/js/script.js', array('prototype'),'1.0');
```

一个 javascript 的例子

对于 Devlounge Plugin Series 插件，我们将添加一个 javascript 文件，后面的文章中会用它。使用这个文件的目的，就是为了说明怎样使用 **wp_enqueue_script** 函数。

- 文件将会放下下列地址：http://yourdomain.com/wp-content/plugins [2]/devlounge-plugin-series/js/devlounge-plugin-series.js
- 这个文件依赖于 prototype
- 版本是 1.0

你可能会回忆起，这个系列文章早期的一篇文章中，我们添加了一个 action 叫 **wp_head**。那个 action 添加后，会调用一个函数，叫做 **addHeaderCode**。让我们来修改这个函数，添加我们新的 javascript：

```
1.  function addHeaderCode() {
2.          if (function_exists('wp_enqueue_script')) {
3.              wp_enqueue_script('devlounge_plugin_series', get_bloginfo('wpurl') . '/wp-content/plugins/devlounge-plugin-series/js/devlounge-plugin-series.js', array('prototype'), '0.1');
4.          }
5.          $devOptions = $this->getAdminOptions();
6.          if ($devOptions['show_header'] == "false") { return; }
7.          ?>
8.  <!-- Devlounge Was Here -->
9.          <?php
10. }
```

上述代码的工作如下：

- 首先，检查 **wp_enqueue_script** 函数的存在性
- 然后调用该函数
- 我们使用 **get_bloginfo('wpurl')** 来得到 WordPress [1]安装根目录，然后手写路径中剩下的部分

加载层叠样式表

我已经往我的样式目录里面添加了一个新的样式表。这里是我们的一些前提：

- 这个文件存储在下列位置：http://yourdomain.com/wp-content/plugins [2]/devlounge-plugin-series/css/devlounge-plugin-series.css
- 我在这个 CSS 文件中指定了一个 ID 叫做#devlounge-link
- **你已经添加了下面的代码** 在文章的末尾：取得博客留言的数量

在样式表文件中，我已经添加了下面的 ID：

```
1.  #devlounge-link{
2.      background-color:#006;
3.      color:#fff;
4.  }
```

在插件中添加样式表，就和在 **addHeaderCode** 函数中添加一行代码一样简单：

```
1.  function addHeaderCode() {
```

```
2.              echo '<link type="text/css" rel="stylesheet" href="' . get_bloginfo('wpurl') . '/wp-content/plugins/devlounge-plugin-series/css/devlounge-plugin-series.css" />' . "\n";
3.              if (function_exists('wp_enqueue_script')) {
4.                  wp_enqueue_script('devlounge_plugin_series', get_bloginfo('wpurl') . '/wp-content/plugins/devlounge-plugin-series/js/devlounge-plugin-series.js', array('prototype'), '0.1');
5.              }
6.              $devOptions = $this->getAdminOptions();
7.              if ($devOptions['show_header'] == "false") { return; }
8.              ?>
9.  <!-- Devlounge Was Here -->
10.             <?php
11. }
```

在第 2 行，我只是简单地打印出了新的样式表。

插件开发全攻略(11)---在你的 WP 插件中使用 AJAX

Posted By Charles On 2008 年 7 月 1 日 @ 13:41 In WordPress | 3 Comments

越来越多的插件开始使用 AJAX 技术。我个人并没有在大多数的插件中看到过 AJAX，但是使用 AJAX 来完成某个任务对你的插件来说可能是必要的。这篇文章将像你展示怎样在你的插件中使用 AJAX。

这篇文章将在上一篇文章《在插件中添加 js 和 css》 [1]的基础上继续。

建立一个新的 PHP 文件

Devlounge Plugin Series 插件已经有了如下的目录结构了：

- devloung-plugin-series
  - devlounge-plugin-series.php(main plugin file)
  - js
    - devlounge-plugin-series.js.php
  - css
    - devlounge-plugin-series.css
  - php
    - dl-plugin-ajax.php(新 php 文件)

注意，我的 javascript 文件的扩展名是 **php**。我会在这篇文章后面解释这个变化的来历。

我已经建立了一个新的文件，并且把它放到了 **php** 文件夹中，并且命名为 **dl-plugin-ajax.php**。我已经在这个文件中放了如下的代码：

```
1.  <?php
2.  if (!function_exists('add_action'))
3.  {
```

```
    require_once("../../../../wp-config.php");
}
if (isset($dl_pluginSeries)) {
    $dl_pluginSeries->showComments();
}
?>
```

这个段代码非常简单，并且仅为做 AJAX 调用而写。它确保了配置结构存在，从而我们可以调用类对象 **dl_pluginSeries** 引用其他 WordPress [2]函数和变量。然而，**showComments** 函数还没有创建，我们下一个议程就是来做这件事。

定义 showComments 函数

showComments 函数将放在我们的 **DevloungPluginSeries** 类中：

```
function showComments() {
    global $wpdb;
    $devloungecomments = $wpdb->get_row("SELECT count(comment_approved) comments_count FROM $wpdb->comments where comment_approved = '1' group by comment_approved", ARRAY_A);
    echo "You have " . $devloungecomments['comments_count'] . " comments on your blog";
}
?>
```

可能已经认出来了，这段代码在数据库交互这篇文章中出现过。此函数输出你博客上留言的数量。

让 JavaScript 知道你的博客在哪里

使用 AJAX 时，一个烦人的事情就是外部 JavaScript 文件不知道你博客安装路径是什么。我是通过在 js 后面添加 php 扩展名来处理这个问题的，因为这样，我就可以调用 WordPress [2]函数了。在 **addHeaderCode** 函数中，我把代码从：

```
if (function_exists('wp_enqueue_script')) {
    wp_enqueue_script('devlounge_plugin_series', get_bloginfo('wpurl') . '/wp-content/plugins/devlounge-plugin-series/js/devlounge-plugin-series.js', array('prototype'), '0.1');
}
?>
```

换成：

```
if (function_exists('wp_enqueue_script')) {
    wp_enqueue_script('devlounge_plugin_seriess', get_bloginfo('wpurl') . '/wp-content/plugins/devlounge-plugin-series/js/devlounge-plugin-series.js.php', array('prototype'), '0.3');
}
?>
```

我唯一改变的是一个版本号码，还有就是给 JavaScript 文件添加了一个 php 扩展名。

编写 JavaScript

这段脚本的目的是找到博客的 URL，调用 PHP 文件，然后返回结果给用户。

```
<?php
if (!function_exists('add_action'))
{
    require_once("../../../../wp-config.php");
}
?>
Event.observe(window, 'load', devloungePluginSeriesInit, false);
function devloungePluginSeriesInit() {
    $('devlounge-link').onclick = devloungePluginSeriesClick;
}
function devloungePluginSeriesClick(evt) {
    var url =  "<?php bloginfo('wpurl') ?>/wp-content/plugins/devlounge-plugin-series/php/dl-plugin-ajax.php";
    var success = function(t){devloungePluginSeriesClickComplete(t);}
    var myAjax = new Ajax.Request(url, {method:'post', onSuccess:success});
    return false;
}
function devloungePluginSeriesClickComplete(t) {
    alert(t.responseText);
}
```

上述代码做了下面这些事情（记住，我们在使用 Prototype）：

- 确定配置结构是存在的，这样我们才能访问 WordPress [2]函数
- 在文档已经装载后，**devloungePluginSeriesInit** 函数被调用了
- 给你添加到文章末尾的链接上绑定了一个事件。如果你忘了，现在可以加进去。简单地找到文章，然后添加这段代码：Get the Number of Blog Comments
- 找到 PHP 文件的绝对路径
- 调用 PHP 文件
- 将反馈输出给用户

结果

下一步，我们假设你已经把那个链接添加好了。我们点击链接"**Get the Number of Blog Comments**"，脚本使用 AJAX 调用了 **DevlongePluginSeries** 类中的函数，并且以对话框的形式返回了结果。

就如你看到的那样，我的本地安装版本，并没有多少评论。

插件开发全攻略（12）---发布并推广你的 WordPress 插件

Posted By Charles On 2008 年 7 月 3 日 @ 12:10 In WordPress | 5 Comments

在你完成了你了不起的 WordPress [1]插件后，在你发布和推广你的插件之前，还有一些事情需要考虑。

发布前

努力遵循标准

虽然遵循 WordPress [1]编码规范不是必须的，但是有些东西确实可以让你的生活变得简单。其中最有价值的一个建议是**永远不要使用简写的 PHP**。原因呢？并非每个人都会将简写开启。

所以，不要这样写：

```
1.  <? /*your php code*/ ?>
```

而要这样写：

```
1.  <?php /*your php code*/ ?>
```

确保你已经完全测试了你的插件

找一些爱好者来测试你的插件。技术上来说，专门的测试人员是好的，但是你还是希望有一些普通的并不懂得编程语言的用户来测试。

找到所有的 bug 是不可能的，但是至少努力推出一个稳定的版本。

确保你有一个 Readme 文件

在你把插件向大众公开之前，确认一下你是否有一个 Readme 文件。这个文件应该包含最基本的安装流程介绍。对于一个要求严格的 readme 文件版本，请检查 WordPress [1]推荐 Readme 格式。甚至有一个绝妙的 Readme 文件验证器。

设立一个专门的 WordPress [1]插件页面

Ajay D'Souza 曾经写过如何发布 WordPress [1]主题的建议。他的建议也可以在某种程度上用在插件之上。

确保你设立了一个专门的插件页面。在这个页面上，用户可以找到关于这个插件的任何东西。这个插件的页面至少要包含以下的东西：

- 对你的插件的一个简单描述
- 下载链接

- 特性列表
- 安装指南
- 版本历史
- 已知的 bugs 和冲突
- 截图或者 demo
- 联系或者支持信息（或者允许评论
-

上述信息将帮助你提升你的插件，尤其是描述和特性部分。

有一个好的目录结构

我会力劝你总是把自己的插件放在一个目录里面，除了主要插件文件外的任何文件都应该放在子目录里面。确保你 zip，gzip 或者 rar 你的插件，使得人们可以非常简单的安装你的插件。

你的插件需要修改模板或者文件修改吗？

如果你的插件需要修改模板的设置或者文件，做好被 bug 报告和等待的帮助猛攻的准备。我认为，一个好的插件绝对不需要修改主题或者某个文件的。除非这个插件往 WordPress [1]内核中添加了模板标签。

如果你的插件确实需要模板修改或者文件修改，在你的下载页面包含一些详细的例子，并且尽可能在你的发布中包含一些 demo。

推广你的插件

在你拥有了你专门的下载页面后，是时候发布你插件的信息使得人们来下载你的插件了。你花在你的插件描述和特性的撰写上的时间是非产关键的，因为这决定了人们会否采用你的插件。其他人链接到你的插件也非常重要。

在 Weblog Tools Collection 推广

一个推广你的插件的很好的地方是在 Weblog Tools Collection。在他们的插件发布栏目，你可以将你的详细描述投递，推荐你的插件。

在 WordPress [1]插件数据库推广

WordPress [1]插件数据库是另一个添加你的插件的好去处。添加你的插件的过程不是那么一帆风顺的，但是有详细的指导。

在 WordPress [1]官方插件库推广

WordPress [1]自己提供了主机来存放你的插件。不过，在你被允许添加你的插件之前，你必须满足几个条件。记住，任何宣传都是好的宣传。

使用社会网络推广

将你的插件添加到 delicious，digg，和 Stumble Upon。让你的朋友来帮忙。如果你的插件足够好，推荐就会广为流传。

在你自己的博客上推广

如果你的插件是人们会注意到的，在你自己的博客上使用吧。人们会开始问，你使用的到底是什么插件。口口相传是非常有效的，尤其是在博客社区内。

结论

你可以拥有世界上最好的插件，但是如果没有正确的发布和推广，只有很少的人会下载它。一旦你开始你的推广过程，倾听特性需求和 bug 报告是非常重要的，尤其是你的插件非常年轻的时候。如果你的插件不起作用，或者太多的人使用的时候遇到问题，人们下载的时候就会非常担忧。所以，在早期修复那些 bug 和添加重要的特性是非常重要的。绝大多数这样的问题可以在测试的过程中解决，但是有些 bug 只有在官方发布后才能看到。

写在《插件开发全攻略》系列的结尾

感谢您阅读插件开发系列文章的最后一篇。希望这个系列能够让您受益，并且帮助你为撰写自己的插件建立基础。非常感谢您的阅读。