WordPress The Right Way（多语版）

书栈(BookStack.CN)

目　录

Créditos

致谢

当前文档 《WordPress The Right Way（多语版）》 由 进击的皇虫 使用 书栈(BookStack.CN) 进行构建，生成于 2018-07-11。

书栈(BookStack.CN) 仅提供文档编写、整理、归类等功能，以及对文档内容的生成和导出工具。

文档内容由网友们编写和整理，书栈(BookStack.CN) 难以确认文档内容知识点是否错漏。如果您在阅读文档获取知识的时候，发现文档内容有不恰当的地方，请向我们反馈，让我们共同携手，将知识准确、高效且有效地传递给每一个人。

同时，如果您在日常工作、生活和学习中遇到有价值有营养的知识文档，欢迎分享到 书栈(BookStack.CN) ，为知识的传承献上您的一份力量！

如果当前文档生成时间太久，请到 书栈(BookStack.CN) 获取最新的文档，以跟上知识更新换代的步伐。

文档地址：http://www.bookstack.cn/books/WordPress-The-Right-Way

书栈官网：http://www.bookstack.cn

书栈开源：https://github.com/TruthHun

分享，让知识传承更久远！ 感谢知识的创造者，感谢知识的分享者，也感谢每一位阅读到此处的读者，因为我们都将成为知识的传承者。

README

WordPress The Right Way

This book is a condensed resource of best practices for and by WordPress developers, intended to fast track developers past common mistakes and painful problems.

This is a living document and will continue to be updated with more helpful information and examples as they become available.

How to Contribute

You can contribute on GitHub. Changes will be pushed to Gitbook.io automatically when the main repository changes.

Editing the book can be done either by updating the markdown files with a text editor, or opening the repository in the Gitbook desktop app. The desktop app will give you a live preview option.

License

来源(书栈小编注)

> *https://github.com/tomjn/WordPress-The-Right-Way*

英文版

Introduction

WordPress The Right Way

This book is a condensed resource of best practices for and by WordPress developers, intended to fast track developers past common mistakes and painful problems.

This is a living document and will continue to be updated with more helpful information and examples as they become available.

How to Contribute

You can contribute on GitHub. Changes will be pushed to Gitbook.io automatically when the main repository changes.

Editing the book can be done either by updating the markdown files with a text editor, or opening the repository in the Gitbook desktop app. The desktop app will give you a live preview option.

License

Getting Started

Getting Started

Basic PHP

It’s assumed that you have a basic knowledge of PHP. This will include a knowledge of:

- functions
- arrays
- variables
- loops and conditionals
- classes and objects
- class inheritance
- polymorphism
- POST and GET
- variable scope

If you don’t have a good grasp of those concepts, you should make sure you have a firm understanding before continuing.

It’s also assumed you have a code editor that has PHP syntax highlighting, although these will be beneficial:

- Auto Indenting
- Auto-completion
- Brace matching
- Syntax checking

Local Development Environments

It’s important to have a local development environment. Gone are the old days of changing a PHP file then updating it on the live server and hoping for the best.

With a local environment, you can work faster, no more uploading and downloading files, being at the mercy of a dodgy internet connection, or waiting for pages to load from the open web. With a local server stack you can work on a train in a tunnel with no wifi or phone signal, and test your work before deploying it to the live server.

Here are a few options for setting up a local development environment. They fall into two categories:

- Virtual Machines
- Native Server Stacks

The first type of environment usually involves projects such as Vagrant, and gives you a standardised consistent virtual machine to work with.

The second, installs the server software directly into your operating system. There are various tools that make this easy, but your environment will be unique and more difficult to debug. These are sometimes called LAMP stacks, which stands for Linux Apache MySQL PHP.

IIS

Microsoft Internet Information Services is the server software that powers Windows based servers. Variants of it come with Windows if you install the appropriate components, but knowledge of IIS setup in the WordPress community is rare. Most remote servers run an Apache or Nginx setup, and developer knowledge is geared in that direction.

IIS is not the easiest route to take.

Version Control

A vital part of working in teams and contributing is version control. Version control systems track changes over time and allow developers to collaborate and undo changes.

Git

Created by Linus Torvalds the creator of Linux, Git is a popular decentralised system, if you've ever been on GitHub, you've encountered git.

Subversion

Also known as svn, this is a centralised version control system, used for the plugin and theme repositories on WordPress.org

Code Style Guide

Code Style Guide

Clean Code

It’s important to keep code readable and maintainable. This prevents small but critical errors from becoming hidden in your code, while making whole classes of bugs incredibly obvious (missing closing braces are easy to spot when you indent consistently).

While it’s best to use the same standard as everybody else, if you’re more comfortable using a PSR standard, then use that. If you do though, do it consistently.

Indenting

Indenting in WordPress is done using tabs, representing 4 spaces visually. Indenting is important for readable code, and each statement should be on its own line. Without indenting, it becomes very difficult to understand what’s happening, and mistakes are easier to make. This also makes support requests on the forums and stack exchange difficult to answer.

A good editor will auto-indent for you, most can re-indent a file if you’ve older code that needs fixing.

A good way to ensure that all members on a team are using the same styles is to use Editor Config. It contains plugins for different editors, so everyone can use their favorite editor.

For instance, the following `.editorconfig` file enforces the above rule, indentation as tabs of width 4 spaces.

```
[*.php]
indent_style = tabs
indent_size = 4
```

PHP Tag Spam

The `<?php` and `?>` tags should be used sparingly. For example:

```
<?php while( have_posts() ) { ?>
    <?php the_post(); ?>
    <?php the_title(); ?>
```

```
    <strong><?php the_date(); ?></strong>
    <?php the_content(); ?>
<?php } ?>
```

Would be easier to read as:

```
<?php
while( have_posts() ) {
    the_post();
    the_title();
    ?>
    <strong><?php the_date(); ?></strong>
    <?php
    the_content();
} ?>
```

A good guideline is to calculate what needs to be displayed, then display it all in one go rather than mixing the two.

Linting

A lot of editors support or have built in syntax checkers. These are called Linters. When using a good editor, syntax errors are highlighted or pointed out.

For example, in PHPStorm, a syntax error is given a red underline.

Coding Standards

WordPress follows a set of coding standards. These differ from the PSR standards. For example, WordPress uses tabs rather than spaces, and places the opening bracket on the same line.

The WordPress Contributor Handbook covers the coding standards in more details. Click below to read more:

- HTML Coding Standards
- PHP Coding Standards
- JavaScript Coding Standards
- CSS Coding Standards

PHP Code Sniffer & PHP CS Fixer

PHP Code Sniffer is a tool that finds violations of the coding standard. Many editors integrate support, including support for a second tool that fixes those violations automatically.

To use this, you will need the WordPress Coding Standards definition.

Debugging

Debugging

When developing for WordPress, it’s important that your code works, but when it fails it can be like a needle in a haystack. It doesn’t need to be that way.

This chapter covers:

- How to find out what errors have occurred
- How to debug the issue
- Plugins and tools to make your life easier
- Features in WordPress that make debugging easier
- How to prevent problems from occurring to begin with and easy automated tools to catch mistakes for you

But before you continue, a word on White Screens of Death

White Screens of Death

A common issue with new WordPress developers is the white screen of death. This happens when a fatal error occurs in PHP. Many new developers respond to this by making changes and hoping the problem goes away, but there are better ways of dealing with this.

When an error occurs in PHP, it gets logged somewhere, and you can find out what went wrong and where.

A good starting point for developers is to enable `WP_DEBUG` .

Error Logging

Error Logging

There are several kinds of error logging, but the most basic are:

- Displaying errors on the frontend
- Writing errors to a log file
- Not displaying anything at all

In a production/live environment, you want to write errors to a log file.

Warnings vs Errors

Depending on how PHP is configured, warnings will also be shown. A warning is something that does not stop PHP from running but indicates a problem might have occurred. For example:

```
1. $my_array = array(
2.     'alice' => 5,
3.     'bob' => 6
4. );
5. echo $my_array['eve'];
```

Here, I am echoing the ‘eve’ entry in `$my_array`, but there is no such entry. PHP responds by creating an empty value and logging a warning. Warnings are indicators of bugs and mistakes.

PHP Error Reporting

Depending on what was defined in your `php.ini`, PHP will have an error reporting level. Everything below that level will be ignored or considered a warning. Everything above it will be considered an error. This can vary from server to server.

The `@` operator

Never use the `@` operator. It’s used to hide errors and warnings in code, but it doesn’t do what people expect it to do.

`@` works by setting the error reporting level on a command so that no error is logged. It doesn’t prevent the error from happening, which is what people expect it to do. This can mean fatal errors are not caught or logged. Avoid using the `@` operator, and treat all instances of it with suspicion.

Handling Errors

Handling Errors

While using the Core APIs, it’s a good idea to check return values. For example, when creating a post, if something goes wrong you should be able to handle that outcome. Not handling errors and failures can lead to unstable code and unpredictable behavior.

Return values

Many functions return error and success values. You should always check these values after making a call. For example `get_post_meta` returns a custom field value, but if that custom field/post meta does not exist, it returns an error value.

Different APIs return different error values, and can include:

- `null` values
- false
- `WP_Error` objects

WordPress API calls at the time of writing do not throw exceptions. However if you hook into actions such as `save_post` and throw an exception, it may not be caught due to this expectation, so do not throw exceptions unless you’re sure you know what you’re doing.

`WP_Error`

The `WP_Error` object is a catch all error message object returned by some APIs. It has internal storage for multiple error messages and error codes.

`is_wp_error`

This is a helpful method to simplify error checking. It checks if a returned value was a `WP_Error` object, and also checks for a handful of other error values. It returns a true or false value, allowing checks such as these:

This is a helpful method to simplify error checking. It checks if a returned value was a `WP_Error` object, but does not check for other error values. It’s shorthand for `if ( get_class( $variable ) == 'WP_Error' )` . For example:

```
1. if ( !is_wp_error( $value ) ) {
2.     // do things
3. } else {
4.     // display a warning to the user and abort
5. }
```

While this is a useful function, remember, not every API returns the same error value, and you should check first.

Tools

Tools

Debugging tools fall into two categories:

- Tools to diagnose issues when they arise and reveal problems
- Tools that prevent mistakes and errors from ever happening to begin with

The age old adage still applies: **prevention is better than cure**

Debugging Tools / Plugins

Installing the plugin Developer from the WordPress repository will give you quick access to a broad range of debugging tools. The following debugging plugins are quite useful:

- Log Deprecated Notices Logs usage of deprecated functions.
- Debug Bar Provides an interface for debugging PHP Notices/Warnings/Errors, reviewing SQL Queries, analysing caching behaviour and much more. It's also extendable with plugins.
- Debug Console The Debug Console for example is really useful.
- Query Monitor View debugging and performance information on database queries, hooks, conditionals, HTTP requests, redirects and more.

Xdebug and Remote Debugging

The Xdebug PHP Extension allows for enhanced debugging, function and method tracing, and profiling of PHP applications. This is installed with VVV and can be turned on/off.

With PHPStorm, you can install a browser extension to access Xdebug (or Zend Debugger) from within the IDE.

Rather than manually adding `var_dump` statements and reloading the page, you can add a breakpoint anywhere in your PHP code, execution will stop and you can see a stack trace, inspect (and modify) the values of all variables and objects or manually evaluate (test) a PHP expression.

With zero-configuration debugging (controlled via cookies and bookmarklets) you don't need to add `?XDEBUG_SESSION_START` to your URLs and you can also debug HTTP post requests.

PHP Debuggers

- DBG - PHP Debugger and Profiler

Browser Web Inspectors

- Chrome DevTools for Google Chrome
- Firebug for Mozilla Firefox
- F12 developer tools for Internet Explorer
- Opera Dragonfly for Opera

Prevention

There are a number of tools dedicated to analysing code and catching semantic mistakes, or pointing out problems in code.

PHP Mess Detector for example, will highlight long variable names, npath and cyclomatic complexity, classes that are too large, unused variables, and other problems. SCheck is a tool provided by Facebook, and performs similar checks, such as finding dead statements and unused classes.

If you can’t type hint, you can make use of a tool such as phantm to infer types and find clashes. Many others exist though, and integrate with your editor/IDE, so look around

wp-config.php

Constants of `wp-config.php`

Currently there are several PHP constants on the `wp-config.php` that will allow you to improve you WordPress code and help you debug.

`WP_DEBUG`

This is an Option included in WordPress version 2.3.1.

By default this will be set to `false` which will prevent warnings and errors from been shown, but **all WordPress developers should have this option active**.

Activates the Logs

```
1. define( 'WP_DEBUG', true );
```

Deactivates the Logs

```
1. define( 'WP_DEBUG', false );
```

*Check that the values must be **bool** instead of **string***

A minor patch later the on Wordpress version 2.3.2, the system allowed us to have a more granular control over the Database error logs.

Later on in the version 2.5, WordPress raised the error reporting level to E_ALL, that will allow to see logs for Notices and Deprecation messages.

Notes:

If you have this option turned on, you might encounter problems with AJAX requests, this problem is related to Notices been printed on the output of the AJAX response, that **will break XML and JSON**.

`WP_DEBUG_LOG`

When you use `WP_DEBUG` set to `true` you have access to this constant, and this will allow you to log your notices and warnings to a file.

`WP_DEBUG_DISPLAY`

When you use `WP_DEBUG` set to `true` you have access to this constant, with it you can choose to display or not the notices and warnings on the screen.

Note:

If these variables don’t produce the output you are expecting check out the Codex Section about ways to setup your logging.

`SCRIPT_DEBUG`

When you have a WordPress plugin or theme that is including the Minified version of your CSS or JavaScript files by default you are doing it wrong!

Following the WordPress idea of creating a file for development and its minified version is very good and you should have both files in your plugin, and based on this variable you will enqueue one or the other.

By default this constant will be set to `false`, and if you want to be able to debug CSS or JavaScript files from WordPress you should turn it to `true`.

Activates the Logs

```
1. define( 'SCRIPT_DEBUG', true );
```

*Check that the values must be **bool** instead of **string***

WordPress default files `wp-includes` and `wp-admin` will be set to its development version if set to `true`.

`CONCATENATE_SCRIPTS`

On your WordPress administration you will have all your JavaScript files concatenated in to one single request based on the dependencies and priority of enqueue.

To remove this feature all around you can set this constant to `false`.

```
1. define( 'CONCATENATE_SCRIPTS', false );
```

`SAVEQUERIES`

When you are dealing with the database you might want to save your queries so that you can debug what is happening inside of your plugin or theme.

Make `$wpdb` save Queries

```
1. define( 'SAVEQUERIES', true );
```

***Note:** this will slowdown your WordPress*

Core

Core

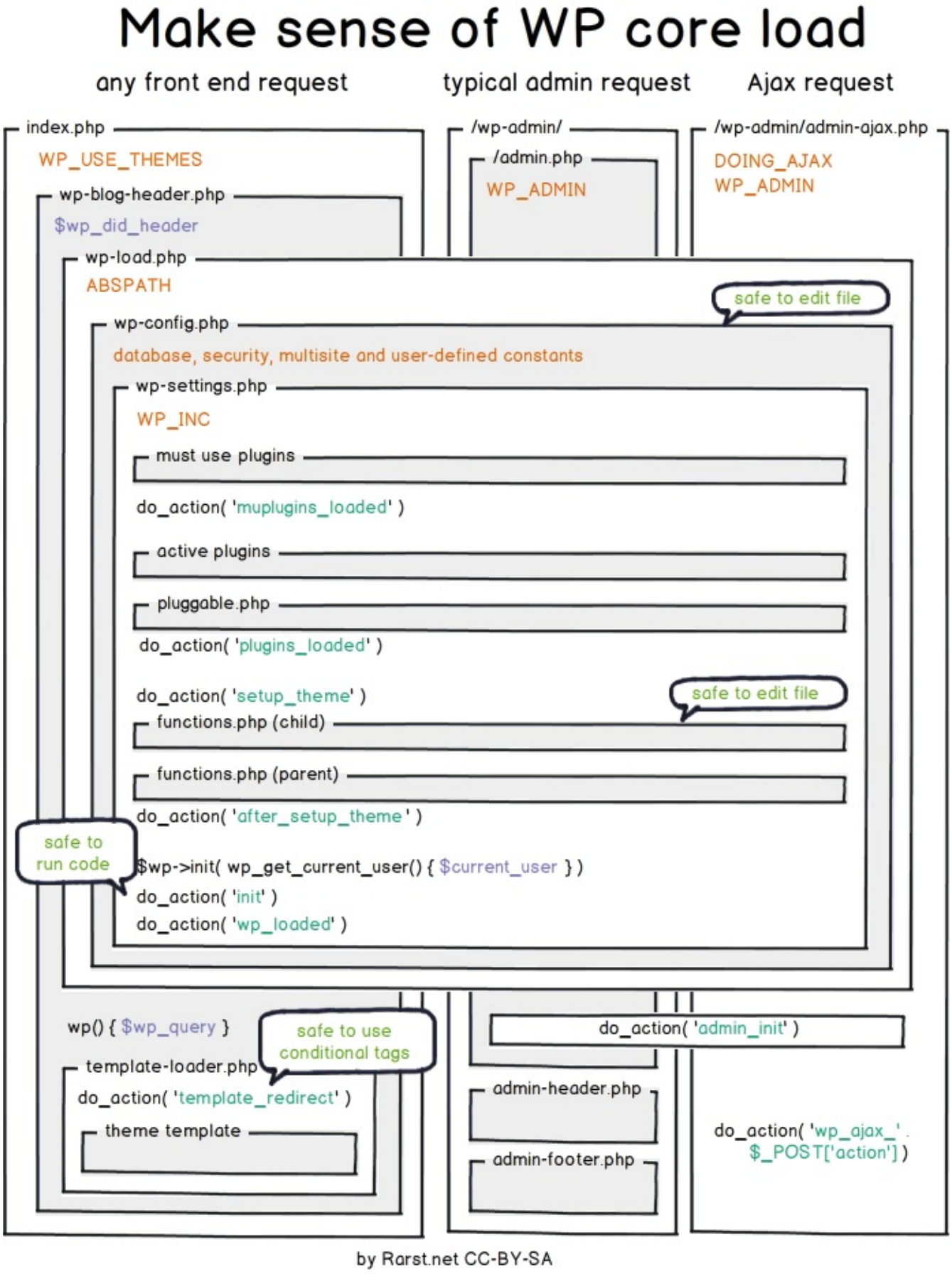
WordPress core is the code that powers WordPress itself. It is what you get when downloading WordPress from wordpress.org, minus the themes and plugins.

Load Process

At the most basic, the WordPress core loading follows this pattern:

- Load MU plugins
- Load Activated plugins
- load theme functions.php
- Run init hook
- Run main query
- Load template

Administration and AJAX requests follow a similar but lighter process. This diagram covers the specifics:



Deregistering jQuery

Many plugin and theme developers attempt to unregister the jQuery that comes with

core, and add their own copy, normally the jQuery on the Google CDN. Do not do this, it can cause compatability issues.

Instead use the copy of jQuery that comes with WordPress and aim for the version used in the latest WordPress when testing. This ensures maximum compatability across plugins.

Modifying Core

It’s tempting to modify parts of Core to remove or add things, but this must never be done. When WordPress updates, all your changes will be lost.

Instead, use Hooks/Actions and Filters to modify Core behaviour.

Further Reading

- Making Sense of Core Load

Data

Data

There are multiple data types in WordPress, but the basic data types stored in the database are:

- Post
- Comments
- Terms
- User
- Blogs
- Network
- Links
- Options
- Site Options

Some of these data types can have additional data attached to them in the form of meta data, some of them have types and statuses.

There are also roles and capabilities, which are not considered content and apply exclusively to users.

Meta

Meta data is data with a key/name and a value, attached to another piece of data. Some people will know them as custom fields. Others will know them as user meta.

Post Meta (or Custom fields) are normally shown in the post edit screen, but if the post meta's key/name begins with an underscore, that field is hidden. This way features such as featured images can be built with their own user interfaces.

Post types

Posts, pages, attachments, and menus are all different kinds of posts. You can also register your own post types, and these are referred to as custom post types. Remember to flush permalinks (manually: Dashboard > Settings > Permalinks > Save Changes or programmatically via flush_rewrite_rules) if you modify or add a post type. If you do not do this, some links will generate 404 errors.

*Warning: Developers sometimes attempt to work around the rewrite rules by flushing rewrite rules on the `init` action using the `flush_rewrite_rules`, but this is a mistake. It can lead to unexpected behaviours, and has a large negative performance impact. Rewrite rules are expensive to build.

Default Post Types

The default post types are as follows:

- Post (`post`)
- Page (`page`)
- Attachment (`attachment`)
- Revision (`revision`)
- Navigation menu (`nav_menu_item`)

Menus

Menu items are stored as `nav_menu_item` posts, however, the menu itself is a term in a custom taxonomy that contains `nav_menu_item` posts.

Revisions

A post of type `revision` , revisions are historical copies of posts, and are tied to their original post via the post_parent field. To get a posts revisions, grab all its children of type `revision` . If a database is growing very large or a particular post/page is frequently/automatically edited, you can limit the number of revisions stored.

Uploaded Files & Images

When you upload a file, WordPress does not reference the image or file using its URL, it uses an attachment. Attachments are posts of type `attachment` , and are referred to by their post ID. For example, when you set a featured image on a post, it stores your chosen image’s post ID in that post’s meta (`_thumbnail_id`).

If a file is uploaded whilst editing a post (rather than in the Media Library itself), it’s `post_parent` field is set to that of the post.

WordPress generates and saves resized versions of the original at the time of upload for better performance. They can be cropped and manipulated independently and the dimensions and filenames are stored in the attachment postmeta.

The default image sizes are configured in Dashboard > Settings > Media. If you need more, you can use add_image_size and also control cropping. You can request a specific size of image using the attachment ID and the image size name (e.g. ‘medium’, ‘large’.)

The Regenerate Thumbnails plugin is useful if you change sizes at a later date.

Comments

Comments have their own table, and are attached to a post. Comments are not a type of

post however, but they are capable of storing meta data. This is rarely used by developers but allows for interesting things.

Terms and Taxonomies

A taxonomy is a way of categorising or organising things. Items are organised using terms in that taxonomy.

For example, yellow is a term in the colour taxonomy. Big and small are both terms in the size taxonomy.

The Tags and categories that come with WordPress are both taxonomies. Individual tags and categories are called terms.

You can register your own taxonomies, but remember to flush permalinks (see *Post Types* above) if you make changes.

Taxonomy terms are tied to Object IDs, where an object ID can be any kind of data. This includes posts, users, or comments. These IDs are normally post IDs, but this is purely convention. There is nothing preventing a user or a comment taxonomy. A user taxonomy would be useful for grouping users into locations or job roles.

Options

Options are stored as key value pairs in their own table. Some options have an autoload flag set and are loaded on every page load to reduce the number of queries.

Transients

Transients are stored as options and are used to cache things temporarily

Object Cache

By default WordPress will use in memory caching that does not persist between page loads. There are plugins available that extend this to use APC or Memcache amongst others.

Data Overview

Here’s a table showing the full spectrum of data types in WordPress that are stored in the database:

	Post	Comments	Term	User
Description	Content, e.g. articles	Commentary on a post	A type of objects, for classifying	Users and Authors

Supported	Yes	Yes	Yes	Yes
Meta	Custom fields	Comment meta	Planned	User Meta
Meta Access	`get_post_meta`	`get_comment_meta`	Planned	get_user_me
Type	Post type	Comment type	Taxonomy	Roles and Capabilitie
Type registration	`register_post_type`	defined on use	`register_taxonomy`	`add_role` `add_cap`
Taxonomy UI?	Yes	No	Yes	No
Default Types	post page attachment nav_menu nav_menu_item	none pingback trackback	cat tag	admin edito author contributor subscriber
Query Class	`WP_Query`	`WP_Comment_Query`	`get_terms`	`WP_User_Query`
Has Archives	Yes	No	Per blog	Yes
Has Widget	Recent Posts	Recent Comments	Categories and Tags	No
Data Availability	Per blog	Per blog	Per blog	Per install
Set current	`setup_postdata`	n/a	n/a	n/a
Database Table	`wp_posts`	`wp_comments`	`wp_terms` `wp_term_relationships` `wp_term_taxonomy`	`wp_users`
Meta Table	`wp_postmeta`	`wp_commentmeta`	Planned	`wp_usermeta`

Queries

Queries

This chapter talks about several kinds of query. Post queries, taxonomy queries, comment queries, user queries, and general SQL queries.

Whenever possible, use the query APIs that WordPress provides, rather than directly calling the database. This allows the internal cache system to speed up your queries, and for caching plugins to help out.

Not using the WordPress APIs to perform queries means that 3rd party plugins are unable to intercept and modify requests, leading to compatibility issues, and broken or incomplete functionality.

Query Limits and Performance

Some queries are more expansive than others, they simply do more work and don’t scale. No amount of MySQL optimisation will fix them. For example, complex meta queries are more expensive.

One issue that most developers don’t realise is scale. For example, you are listing terms in a custom taxonomy in a dropdown, and you have 5 or 10 terms. In that example the query will be fast, however, if 10,000 terms are added 6 months later, that dropdown is going to take a very long time to generate.

So always add limits to your queries, even if you don’t think they’re needed. Place an unrealistically high number you never expect to hit them, e.g. 100 or 1000.

Post Queries

Post Queries

Post queries retrieve posts from the database so that they can be processed or displayed on the frontend. This section covers some vital concepts, and methods of generating these queries.

The Main Loop

Every page displayed by WordPress has a main query. This query grabs posts from the database, and is used to determine what template should be loaded.

Once that template is loaded, the main loop begins, allowing the theme to display the posts found by the main query. Here is an example main loop:

```
if ( have_posts() ) {
    while ( have_posts() ) {
        the_post();
        // display post
    }
} else {
    // no posts were found
}
```

The Main Query and Query Variables

The main query is created using the URL, and is represented by a `WP_Query` object.

This object is told what to fetch using Query Variables. These values are passed into the query object at the start, and must be part of a list of valid query variables.

For example, the query variable ‘p’ is used to fetch a specific post type, e.g.

```
$posts = get_posts( 'p=12' );
```

Fetches the post with ID 12. The full list of options are available on the `WP_Query` codex entry.

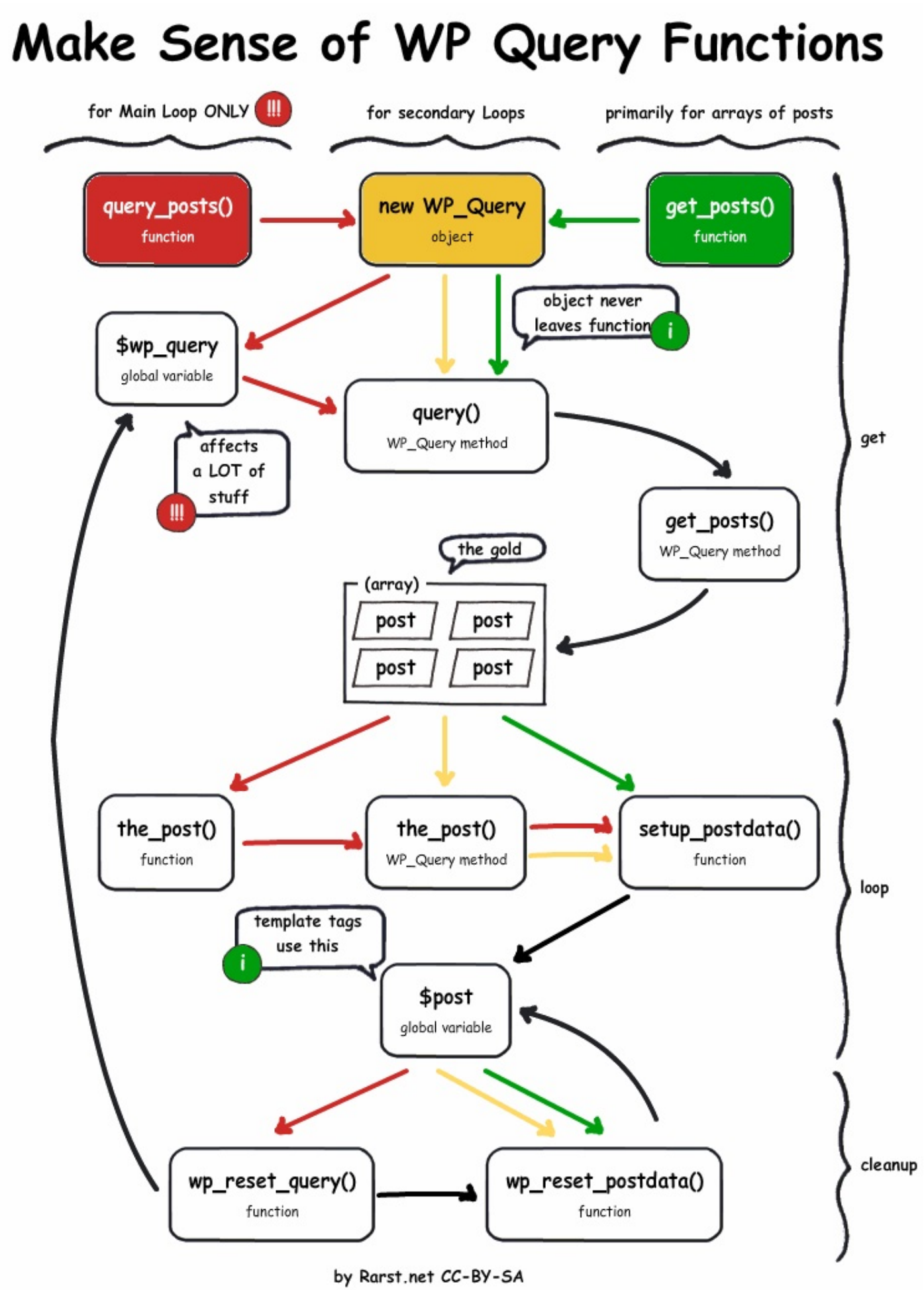
Making a Query

To retrieve posts from the Database, you need to make a post query. All methods of getting posts are layers on top of the `WP_Query` object.

There are 3 ways to do this:

- `WP_Query`
- `get_posts`
- `query_posts`

This diagram explains what happens in each method:


Make Sense of WP Query Functions
for Main Loop ONLY !!!
for secondary Loops
primarily for arrays of posts
query_posts()
function
new WP_Query
object
get_posts()
function
object never leaves function
$wp_query
global variable
query()
WP_Query method
affects a LOT of stuff !!!
get_posts()
WP_Query method
the gold
(array)
post
post
post
post
get
the_post()
function
the_post()
WP_Query method
setup_postdata()
function
loop
template tags use this
$post
global variable
wp_reset_query()
function
wp_reset_postdata()
function
cleanup
by Rarst.net CC-BY-SA

`WP_Query`

```
$query = new WP_Query( $arguments );
```

All post queries are wrappers around `WP_Query` objects. A `WP_Query` object represents a query, e.g. the main query, and has helpful methods such as:

```
$query->have_posts();
$query->the_post();
```

etc. The functions `have_posts();` and `the_post();` found in most themes are wrappers around the main query object:

```
function have_posts() {
    global $wp_query;

    return $wp_query->have_posts();
}
```

`get_posts`

```
$posts = get_posts( $arguments );
```

`get_posts` is similar to `WP_Query`, and takes the same arguments, but it returns an array containing the requested posts in full. You shouldn't use `get_posts` if you're intending to create a post loop.

While `get_posts` is conceptually simpler than `WP_Query` for novice programmers to understand, it does have a downside. `get_posts` doesn't make extensive use of the object cache in the way that `WP_Query` does, and may not be as performant.

Don't use `query_posts`

`query_posts` is an overly simplistic and problematic way to modify the main query of a page by replacing it with new instance of the query.

It is inefficient (re-runs SQL queries) and will outright fail in some circumstances (especially often when dealing with posts pagination). Any modern WordPress code should use more reliable methods, such as making use of the `pre_get_posts` hook, for this purpose. Do not use `query_posts()`.

Meta Queries and Performance

When performing a query involving meta keys, there can be performance issues. This is because there is no index on the post meta tables. As a result post meta queries have the potential to be very expensive. Queries involving both post meta keys and post

meta values can be even more expensive.

`NOT IN` Queries

Queries looking for posts that are not in a category or don't have a post meta key can be very expensive, and should be avoided. The nature of the query means that they are expensive, as the database has to figure out which posts do have the term/meta key, then subtract those results from the full list of posts. These queries don't scale and are resource intensive.

Cleaning up after Queries

`wp_reset_postdata`

When using `WP_Query` or `get_posts`, you may set the current post object, using `the_post` or `setup_postdata`. If you do, you need to clean up after yourself when you finish your while loop. Do this by calling `wp_reset_postdata`.

A common mistake is to call `wp_reset_postdata` after the if statement. This is incorrect, as the post hasn't changed if the if statement is false, leading to potentially unexpected behavior. Always call the function before the closing brace, not after, e.g.

```
1. if ( $q->have_posts() ) {
2.     while( $q->have_posts() ) {
3.         $q->the_post();
4.     }
5.     wp_reset_postdata();
6. }
```

`wp_reset_query`

When you call `query_posts`, you will need to restore the main query after you've done your work. Failure to do so can lead to a large number of issues and unexpected behavior. You can do this with `wp_reset_query`. Always do this after calling `query_posts`, and only do it when necessary.

The `pre_get_posts` Filter

If you need to change the main query and display something else on the page, you should use the `pre_get_posts` filter.

Many people will want to use this for things such as removing posts from an archive, changing the post types for search, excluding categories, and others

Here is the Codex example for searching only for posts:

```php
function search_filter($query) {
    if ( !is_admin() && $query->is_main_query() ) {
        if ($query->is_search) {
            $query->set('post_type', 'post');
        }
    }
}

add_action( 'pre_get_posts', 'search_filter' );
```

These filters can go in a themes `functions.php`, or in a plugin.

Further Reading

- You don’t know query, a talk by Andrew Nacin
- When you should use WP_Query vs query_posts, Andrei Savchenko/Rarst

Taxonomy and Term Queries

Taxonomy and Term Queries

When dealing with taxonomies (including post categories and tags), it’s safer to rely on the generic APIs rather than the legacy helper APIs. These include:

- `get_taxonomies`
- `get_terms`
- `get_term_by`
- `get_taxonomy`
- `wp_get_object_terms`
- `wp_set_object_terms`

It’s easier to learn one set of APIs, and think of categories and tags as just another taxonomy, rather than mixing and matching older functions such as `get_category` etc.

Comment Queries

Comment Queries

You can retrieve comments using the `WP_Comment_Query` class. When WordPress tries to load a single post, it constructs one of these objects in order to retrieve the number of comments it has, ready for when it’s displayed later on.

This is a basic comment query:

```
$args = array(
   // args here
);

// The Query
$comments_query = new WP_Comment_Query();
$comments = $comments_query->query( $args );

// Comment Loop
if ( $comments ) {
    foreach ( $comments as $comment ) {
        echo '<p>' . $comment->comment_content . '</p>';
    }
} else {
    echo 'No comments found.';
}
```

Comment queries can find comments of different types across multiple or single posts. Using a comment query can be faster than a raw SQL command thanks to the built cache system.

User Queries

User Queries

Similar to comment queries, user queries can be used to find individual users, users with specific roles, and other parameters.

Here is a basic User query:

```php
$args = array(
    //
);

// The Query
$user_query = new WP_User_Query( $args );

// User Loop
if ( ! empty( $user_query->results ) ) {
    foreach ( $user_query->results as $user ) {
        echo '<p>' . $user->display_name . '</p>';
    }
} else {
    echo 'No users found.';
}
```

Note that the user query class may not be available yet if your code runs very early.

SQL

SQL

WPDB

It can be tempting for the uninformed to resort to a raw SQL query to grab posts. Only do this as a last resort.

But if you have to make an SQL query, use `WPDB` objects.

dbDelta and Table Creation

The `dbDelta` function examines the current table structure, compares it to the desired table structure, and either adds or modifies the table as necessary, so it can be very handy for updates.

The `dbDelta` function is rather picky, however. For instance:

- You must put each field on its own line in your SQL statement.
- You must have two spaces between the words `PRIMARY KEY` and the definition of your primary key.
- You must use the key word `KEY` rather than its synonym `INDEX` and you must include at least one KEY.
- You must not use any apostrophes or backticks around field names.
- `CREATE TABLE` must be capitalised.

With those caveats, here are the next lines in our function, which will actually create or update the table. You'll need to substitute your own table structure in the $sql variable.

Further Reading

- Creating Tables With Plugins - Codex

Routing

Routing

Common questions asked by new developers revolve around a misconception. They believe that single.php is what made WordPress load a single post, and talk about making WordPress load archive.php instead so that they can view multiple posts rather than an individual post.

That viewpoint is confusing, the truth is that such a viewpoint is completely upside down. The template does not determine the content. The content determines the template used.

This chapter will go into more depth regarding how WordPress breaks down a URL, creates a query, then figures out which template to load.

- An explanation of how rewrite rules generate a query, which loads a template, which displays a page
- Custom Query variables & routing
- Adding a rewrite rule
- Flushing rewrite rules
- Debugging rewrite rules
- Clashes & slugs

The Main Loop & Template Loading

The Main Loop & Template Loading

- The main query is a WP_Query object
- The main loop is using that main query
- The main query is done before the template is even loaded
- The template is loaded based on what the main query is
- The main query is determined by parameters called query variables
- Which template is loaded when is shown on the template hierarchy diagram
- All templates are just custom ways of showing the main post loop

What are Query Variables and Where do They Come From?

What Are Query Variables and Where Do They Come From?

Query variables are what gets passed into the main `WP_Query` as arguments. For example:

```
$query = new WP_Query ( [
    'posts_per_page' => 5,
    'post_type'      => 'page',
]);
```

Here, `posts_per_page` and `post_type` are query variables.

Query Variables Work In URLs Too

You can append query variables on to a URL in WordPress, e.g.

- https://example.com/category/wordpress/
- https://example.com/category/wordpress/?posts_per_page=5

This is because all URLs are made out of query variables. Rewrite rules take a pretty URL, and map it on to an ugly URL made entirely out of query variables. You can see this in action by browsing a WordPress site with permalinks turned off.

Because all URLs in WordPress are actually query variables on an `index.php` call, these variables are passed directly to the main query. They determine:

- Which posts get loaded
- If the main query is for an archive or a single page

This means we can control what page WordPress thinks it’s on, or modify what it does in the database via the `pre_get_posts` filter.

Query Variables Determine Which Template Gets Loaded

There’s a misunderstanding that’s common amongst those new to WordPress theming, that the template determines what gets loaded. You might see questions such as “How do I load XYZ with `single.php`?”.

The answer is that WordPress grabs posts from the database long before it figures out which theme template to load. By the time it decides to load `single.php`, it’s already

retrieved the post. This is where the template hierarchy comes in.

Based on the main query, the template loader asks questions to figure out if we're in an archive, and what kind of archive or post. It then loads the most specific template it can find, falling back to more generic options until it reaches `index.php` .

If you install Query Monitor, you can see the query variables for the main query, and which templates it tried to load and in which order.

Adding New Query Variables

Now that we know how query variables work, we might want to add our own query variable to act as a tag or flag we can watch for in a rewrite rule.

Query variables are whitelisted for security reasons, but we can add extra query variables to the whitelist using the `query_vars` filter:

```
1. function wpd_query_vars( $query_vars ){
2.     $query_vars[] = 'my-api';
3.     return $query_vars;
4. }
5. add_filter( 'query_vars', 'wpd_query_vars' );
```

Rewrite Rules

Rewrite Rules

- Rewrite rules are based on regular expressions
- Regular expressions map nice URLs on to uglier query var based URLs that can be parsed
- Rewrite rules have priority/order
- Rewrite rules are generated then stored in the database
- Rebuilding rewrite rules is expensive

Clashes, Slugs, & Debugging

Clashes, Slugs, & Debugging

- Slugs must be unique
- 1 URL can’t be 2 things, there must be no ambiguity
- Showing different things depending on where the user has been before is terrible for caching
- Monkey rewrite tools plugin for debugging

Templates

Templates

Loading templates via `get_template_part`

When including templates in your theme, it’s tempting to use code such as this:

```
1. include( 'customloop.php' );
```

However, doing this breaks support for child themes. Instead using `get_template_part` will do the job better, while giving extra flexibility. For example:

```
1. get_template_part( 'custom', 'loop' );
```

This way WordPress will attempt to load `custom-loop.php` . If the file does not exist, it will load `custom.php` , and if a child theme exists, it will load the child theme version of the file.

This allows fallback templates and specialised templates based on post meta and other data such as post type. For example:

```
1. get_template_part( 'loop', get_post_type() );
```

This will load `loop.php` , but if a custom version of the template exists for that post type, it will load that instead. e.g. `loop-page.php`

Internally, `get_template_part` uses the `locate_template` function. This function finds the appropriate file, and returns its name. This is useful for finding a template, without loading it.

`locate_template is also useful for plugin theming. For example:

```
1. // if the theme has a custom template for my plugin
2. if ( locate_template( 'mycustomplugin.php') != '' ) {
3.     // load the custom template for my plugin from the theme
4.     get_template_part( 'mycustomplugin.php' );
5. } else {
6.     // fallback to the plugins default theme
7.     include( 'defaulttemplates/mycustomplugin.php' );
8. }
```

If you wish to provide such a system in your plugin though, it’s advised you use the `template_include` filter. Scroll down for a more in depth look at the `template_include`

filter.

How Templates are Chosen, and The Template Hierarchy

- Notes on how a template is chosen using the main query. How templates are chosen and loaded, how child themes are involved. Show the template hierarchy diagram

Functions.php and Plugins

`functions.php` is a file in your theme that gets loaded prior to any templates. If your theme has non-template functionality, such as changing the length of excerpts, adding stylesheets and scripts, etc, this is where that code would go.

Because of the way functions.php is loaded, it can be considered a plugin, as there is no difference between `functions.php` and plugin development. However, there is a difference in how it's loaded.

If your theme registers post types and taxonomies, shortcodes, or widgets, this data is no longer available to the user when they change theme. This is a large problem for data portability, and can cause a persons site to become non-functional or broken.

Post types, taxonomies, shortcodes, and widgets, should be implemented in a separate plugin so that the users data remains portable, and their site is not broken when themes change. To do otherwise is irresponsible.

Loading Stylesheets

- enquing stylesheets properly

Templates and Plugins

- `template_include` filter

Forms

- Forms that submit to a separate standalone PHP file in your theme are bad. An example of how to handle a basic form submission on a page template

Virtual Pages

- For when you need a page/URL that doesn't have an associated post or archive, e.g. a shopping cart or an API endpoint.

Further Reading

- link to template diagram
- interactive template diagram

JavaScript

JavaScript

Javascript is the future of WordPress, but there are a number of things to keep in mind.

While there's a lot of things that should always be done, there are three approaches to using WordPress javascript the right way.

- **The Wrong Way** - Sending AJAX requests to files in your theme or page templates, then including them in your header with a manually coded tag
- **The Old Way** - Using the WP AJAX API for requests
- **The Best Way** - Building your admin UI in Javascript instead of PHP, and powering it with the REST API.

At the time of writing, the REST API and the content endpoints are the future, and admin UIs need to prepare for a fully JS powered admin UI. This means:

- Clean, portable, cachable Data APIs
- Enforced, and simplified built in security in your endpoints
- A standardised system to work in
- More secure interfaces by avoiding the need for escaping with Javascript templating and reactive libraries
- Faster admin screens whose sole job is to bootstrap the JS UI

While information on these are compiled, information is preserved below.

Registering and Enqueueing

WordPress comes with dependency management and enqueueing for JavaScript files. Don't use raw `<script>` tags to embed JavaScript.

JavaScript files should be registered. Registering makes the dependency manager aware of the script. To embed a script onto a page, it must be enqueued.

Let's register and enqueue a script.

```
1. // Use the wp_enqueue_scripts function for registering and enqueueing scripts on the front end.
2. add_action( 'wp_enqueue_scripts', 'register_and_enqueue_a_script' );
3. function register_and_enqueue_a_script() {
4.     // Register a script with a handle of `my-script`
5.     //  + that lives inside the theme folder,
6.     //  + which has a dependency on jQuery,
7.     //  + where the UNIX timestamp of the last file change gets used as version number
8.     //    to prevent hardcore caching in browsers - helps with updates and during dev
```

```
    //  + which gets loaded in the footer
    wp_register_script(
        'my-script',
        get_template_directory_uri().'/js/functions.js',
        array( 'jquery' ),
        filemtime( get_template_directory().'/js/functions.js' ),
        true
    );
    // Enqueue the script.
    wp_enqueue_script( 'my-script' );
}
```

Scripts should only be enqueued when necessary; wrap conditionals around `wp_enqueue_script()` calls appropriately.

When enqueueing javascript in the admin interface, use the `admin_enqueue_scripts` hook.

When adding scripts to the login screen, use the `login_enqueue_scripts` hook.

Localizing

Localizing a script allows you to pass variables from PHP into JS. This is typically used for internationalization of strings (hence localization), but there are plenty of other uses for this technique.

From a technical side, localizing a script means that there will be a new `<script>` tag added right before your registered script, that contains a *global* JavaScript object with the name you specified during localizing (the 2nd argument). This also means that if you add another script later on, that has this script as dependency, then you will be able to use the global object there as well. WordPress resolves chained dependencies just fine.

Let’s localize a script.

```
add_action( 'wp_enqueue_scripts', 'register_localize_and_enqueue_a_script' );
function register_localize_and_enqueue_a_script() {
    wp_register_script(
        'my-script',
        get_template_directory_uri().'/js/functions.js',
        array( 'jquery' ),
        filemtime( get_template_directory().'/js/functions.js' ),
        true
    );
    wp_localize_script(
        'my-script',
        'scriptData',
        // This is the data, which gets sent in localized data to the script.
        array(
            'alertText' => 'Are you sure you want to do this?',
        )
```

```
    );
    wp_enqueue_script( 'my-script' );
}
```

In the javascript file, the data is available in the object name specified while localizing.

```
( function( $, plugin ) {
    alert( plugin.alertText );
} )( jQuery, scriptData || {} );
```

Deregister / Dequeueing

Scripts can be deregistered and dequeued via `wp_deregister_script()` and `wp_dequeue_script()` .

AJAX

WordPress offers an easy server-side endpoint for AJAX calls, located in `wp-admin/admin-ajax.php` .

Let’s set up a server-side AJAX handler.

```
// Triggered for users that are logged in.
add_action( 'wp_ajax_create_new_post', 'wp_ajax_create_new_post_handler' );
// Triggered for users that are not logged in.
add_action( 'wp_ajax_nopriv_create_new_post', 'wp_ajax_create_new_post_handler' );

function wp_ajax_create_new_post_handler() {
    // This is unfiltered, not validated and non-sanitized data.
    // Prepare everything and trust no input
    $data = $_POST['data'];

    // Do things here.
    // For example: Insert or update a post
    $post_id = wp_insert_post( array(
        'post_title' => $data['title'],
    ) );

    // If everything worked out, pass in any data required for your JS callback.
    // In this example, wp_insert_post() returned the ID of the newly created post
    // This adds an `exit`/`die` by itself, so no need to call it.
    if ( ! is_wp_error( $post_id ) ) {
        wp_send_json_success( array(
            'post_id' => $post_id,
        ) );
    }

    // If something went wrong, the last part will be bypassed and this part can execute:
    wp_send_json_error( array(
```

```
        'post_id' => $post_id,
    ) );
}


add_action( 'wp_enqueue_scripts', 'register_localize_and_enqueue_a_script' );
function register_localize_and_enqueue_a_script() {
    wp_register_script(
        'my-script',
        get_template_directory_uri().'/js/functions.js',
        array( 'jquery' ),
        filemtime( get_template_directory().'/js/functions.js' ),
        true
    );
    // Send in localized data to the script.
    wp_localize_script(
        'my-script',
        'scriptData',
        array(
            'ajax_url' => admin_url( 'admin-ajax.php' ),
        )
    );
    wp_enqueue_script( 'my-script' );
}
```

And the accompanying JavaScript:

```
( function( $, plugin ) {
    $( document ).ready( function() {
        $.post(
            // Localized variable, see example below.
            plugin.ajax_url,
            {
                // The action name specified here triggers
                // the corresponding wp_ajax_* and wp_ajax_nopriv_* hooks server-side.
                action : 'create_new_post',
                // Wrap up any data required server-side in an object.
                data   : {
                    title : 'Hello World'
                }
            },
            function( response ) {
                // wp_send_json_success() sets the success property to true.
                if ( response.success ) {
                    // Any data that passed to wp_send_json_success() is available in the data property
                    alert( 'A post was created with an ID of ' + response.data.post_id );

                // wp_send_json_error() sets the success property to false.
                } else {
                    alert( 'There was a problem creating a new post.' );
                }
            }
```

```
        );
    } );
} )( jQuery, scriptData || {} );
```

`ajax_url` represents the admin AJAX endpoint, which is automatically defined in admin interface page loads, but not on the front-end.

Let’s localize our script to include the admin URL:

```
add_action( 'wp_enqueue_scripts', 'register_localize_and_enqueue_a_script' );
function register_localize_and_enqueue_a_script() {
    wp_register_script( 'my-script', get_template_directory_uri() . '/js/functions.js', array( 'jquery' ) );
    // Send in localized data to the script.
    $data_for_script = array( 'ajax_url' => admin_url( 'admin-ajax.php' ) );
    wp_localize_script( 'my-script', 'scriptData', $data_for_script );
    wp_enqueue_script( 'my-script' );
}
```

The JavaScript side of WP AJAX

There are several ways to go on this. The most common is to use `$.ajax()` . Of course, there are shortcuts available like `$.post()` and `$.getJSON()` .

Here’s the default example.

```
/*globals jQuery, $, scriptData */
( function( $, plugin ) {
    "use strict";

    // Alternate solution: jQuery.ajax()
    // One can use $.post(), $.getJSON() as well
    // I prefer defered loading & promises as shown above
    $.ajax( {
         url  : plugin.ajaxurl,
         data : {
            action      : plugin.action,
            _ajax_nonce : plugin._ajax_nonce,
            // WordPress JS-global
            // Only set in admin
            postType     : typenow,
         },
         beforeSend : function( d ) {
             console.log( 'Before send', d );
         }
    } )
        .done( function( response, textStatus, jqXHR ) {
            console.log( 'AJAX done', textStatus, jqXHR, jqXHR.getAllResponseHeaders() );
        } )
        .fail( function( jqXHR, textStatus, errorThrown ) {
            console.log( 'AJAX failed', jqXHR.getAllResponseHeaders(), textStatus, errorThrown );
```

```
        } )
        .then( function( jqXHR, textStatus, errorThrown ) {
            console.log( 'AJAX after finished', jqXHR, textStatus, errorThrown );
        } );
} )( jQuery, scriptData || {} );
```

Note that above example uses `_ajax_nonce` to verify the NONCE value, which you will have to set by yourself when localizing the script. Just add `'_ajax_nonce' => wp_create_nonce( "some_value" ),` to your data array. You can then add a referrer check to your PHP callback that looks like `check_ajax_referer( "some_value" )` .

AJAX on click

Actually it's pretty simple to execute an AJAX request when some clicks (or does some other user interaction) on some element. Just wrap up your `$.ajax()` (or similar) call. You can even add a delay like you might be used to.

```
$( '#' + plugin.element_name ).on( 'keyup', function( event ) {
    $.ajax( { ... etc ... } )
        .done( function( ... ) { etc }
        .fail( function( ... ) { etc }

} )
    .delay( 500 );
```

Multiple callbacks for a single AJAX request

You might come into a situation where multiple things have to happen after an AJAX request finished. Gladly jQuery returns an object, where you can attach all of your callbacks.

```
/*globals jQuery, $, scriptData */
( function( $, plugin ) {
    "use strict";

    // Alternate solution: jQuery.ajax()
    // One can use $.post(), $.getJSON() as well
    // I prefer defered loading & promises as shown above
    var request = $.ajax( {
         url  : plugin.ajaxurl,
         data : {
            action      : plugin.action,
            _ajax_nonce : plugin._ajax_nonce,
            // WordPress JS-global
            // Only set in admin
            postType     : typenow,
         },
         beforeSend : function( d ) {
```

```
            console.log( 'Before send', d );
        }
    } );

    request.done( function( response, textStatus, jqXHR ) {
        console.log( 'AJAX callback #1 executed' );
    } );

    request.done( function( response, textStatus, jqXHR ) {
        console.log( 'AJAX callback #2 executed' );
    } );

    request.done( function( response, textStatus, jqXHR ) {
        console.log( 'AJAX callback #3 executed' );
    } )
} )( jQuery, scriptData || {} );
```

Chaining callbacks

A common scenario (regarding how often it is needed and how easy it then is to hit the mine trap), is chaining callbacks when an AJAX request finished.

About the problem first:

> *AJAX callback (A) executes*
> *AJAX Callback (B) doesn't know that it has to wait for (A)*
> *You can't see the problem in your local install as (A) is finished too fast.*

The interesting question is how to wait until A is finished to then start B and its processing.

The answer is "deferred" loading and "promises", also known as "futures".

Here's an example:

```
( function( $, plugin ) {
    "use strict";

    $.when(
        $.ajax( {
            url :  pluginURl,
            data : { /* ... */ }
        } )
           .done( function( data ) {
                // 2nd call finished
           } )
           .fail( function( reason ) {
               console.info( reason );
           } );
    )
    // Again, you could leverage .done() as well. See jQuery docs.
    .then(
```

```
        // Success
        function( response ) {
            // Has been successful
            // In case of more then one request, both have to be successful
        },
        // Fail
        function( resons ) {
            // Has thrown an error
            // in case of multiple errors, it throws the first one
        },
    );
    //.then( /* and so on */ );
} )( jQuery, scriptData || {} );
```

Source: WordPress.StackExchange / Kaiser

Widgets

JavaScript

Enqueing a script

For the widget form in the admin area

A quick note on how to do it, and a note on running the JS, so that it doesn't get ran on the html used to create new widget forms, only those in the sidebars on the right.

For the frontend

How to enqueue a widgets scripts and styles, but only if the widget is on the page

Events

Running code when:

A New Widget is Added, or Re-ordered

- Make use of the ajaxStop event to process javascript when a widget is added or re-ordered

```
jQuery( document ).ready( function( $ ) {
    function doWidgetStuff() {
        var found = $( '#widgets-right .mywidgetelement' );
        found.each( function( index, value ) {
          // process elements
        } );
    }

    window.counter = 1;

    doWidgetStuff();

    $( document ).ajaxStop( function() {
        doWidgetStuff();
    } );
} );
```

The widget form opens

- Some js to show how to do things when the form opens and closes

Further Reading

- Executing javascript when a widget is added

I18n

I18n

When talking about I18n here, we're going to talk about translation strings in user interfaces and on the frontend. For content in multiple languages or language pickers for users, you will need to install a plugin to provide the editing tools for posts and other content types.

At any point, you can manually set the language WordPress uses by user or overriding the `WPLANG` option. Older tutorials will recommend the `WP_LANG` constant, but this has been deprecated

However you will need to make sure the necessary language files are in place in your `wp-content/languages` folder before the change takes full effect.

Translation work falls under the Polyglots group at contributor days. If you're interested in translating WordPress Core, you should read the official translators handbook to find out how

Securing Language Files

Language files have 2 major attack vectors:

- Unescaped translation strings containing javascript tags
- n-plurals

Embedded Security Risks

It's important to use the escaping functions with the translation API to verify that dangerous content isn't inserted. It's possible to do by placing language files for a particular translation domain inside a WordPress install

You can save typing out `echo esc_html( __( '','' ) )` by using the helper functions:

- `esc_html__` instead of `__`
- `esc_attr__` instead of `__`
- etc

There are also an extended set of functions that simplify this further by adding `e` to the function name:

- `esc_html_e`
- `esc_attr_e`

These will output on their own, so an `echo` statement isn't needed.

Translation API Abuse

The `esc_html_e` helper functions are sometimes misused as a susbtitute for `echo esc_html` . Always use the second parameter that sets the translation domain. If you don't it could have unanticipated side effects as your strings are mistranslated:

```
1. // Good:
2. echo esc_html('date');
3. // Great:
4. esc_html_e( 'date', 'mytheme' );
5. // Bad:
6. esc_html_e( 'date' );
```

n-plurals

A relatively unknown part of the translation format is the `n-plurals` field. This determines the way plural forms work in a language for a particular file.

Because of its complexity, and for performance reasons, WordPress loads this field as a string, wraps it in a function, and passes the result to `eval` . Because of this, it's very easy to craft a language file with a primitive PHP shell.

The only way to mitigate this is code review/manual inspection.

Setting the Admin language

On installation WordPress asks you to select a language, but you may want to set different languages for the front end back end. For example a German website ran by an English speaker may want the admin area to be in their native language.

In order to do this, set your sites language to German using the `WP_LANG` constant mentioned earlier, and add this code to set the admin area language to english:

```
1. add_filter('locale', 'wpse27056_setLocale');
2. function wpse27056_setLocale($locale) {
3.     if ( is_admin() ) {
4.         return 'en_US';
5.     }
6.
7.     return $locale;
8. }
```

Foreign Twitter Embeds

Sometimes oembeds come back in an unexpected foreign language, this is because the service being used is looking at your servers request and tracing it back to its

origin to determine it’s country. For example, an English website hosted on a German server may result in German twitter embeds.

Further Reading

- Different languages for front and back ends

Multisite

Multisite

Grabbing Data From Another Blog in a Network

Getting data from another blog on the same multisite install can be done. Some people use SQL commands to do this, but this can be slow, and error prone.

Although it's an inherently expensive operation, you can make use of `switch_to_blog` and `restore_current_blog` to make it easier, while using the standard WordPress APIs.

```
1. switch_to_blog( $blog_id );
2. // Do something
3. restore_current_blog();
```

`restore_current_blog` undos the last call to `switch_to_blog`, but only by one step, the calls are not nestable, so always call `restore_current_blog` before calling `switch_to_blog` again.

Listing Blogs in a Network

Listing blogs in a network is possible, but it's an expensive thing to do.

It can be done using the `get_sites( $args)` function, available since version 4.6 of WordPress. The `get_sites( $args)` function accepts an array or query string of site query parameters specifying the kind of sites you are looking for. For versions 3.7 to 4.6 a similar function `wp_get_sites( $args )` was available.

See the codex entry for `get_sites` for more details.

Domain Mapping

Domain mapping allows a blog on a multisite install to serve from any domain name. This way a blog does not have to be a subdirectory of the main install, or a subdomain. The WordPress Default supports Domain Mapping without Alias. Add the Domain in the blog-settings to the blog of the Network administration area.

Often is it helpful - but not necessary, to set the `COOKIE_DOMAIN` constant to an empty string in your `wp-config.php` :

`define('COOKIE_DOMAIN', '');`

Otherwise WordPress will always set it to your network's `$current_site->domain`, which

could cause issues in some situations.

WordPress Core hopes to provide Domain Alias Mapping in the future, but until then you can make use of one of the following plugins:

- Mercator - WordPress multisite domain mapping for the modern era.
- WordPress MU Domain Mapping - Map any blog/site on a WordPressMU or WordPress 3.X network to an external domain.

Testing

Testing

It’s important that you test your code and your themes, but that takes time. Luckily there are tools and methods of simplifying and automating these things. This chapter is going to cover basic preventative testing, and tools to help catch bugs and things you may have missed.

WP Test & Theme Test Data

- Good for testing content
- It’s a content export file
- Contains lots of posts and categories of varying types to test as many possible combinations as possible
- Useful for testing themes and unhandled scenarios such as posts without titles, giant nav menus, or very long tag names.

Theme review tester plugin

- Good for testing theme completion
- A plugin that runs several automated tests on the current theme
- Checks for things the theme review team checks for when submitting themes to wordpress.org
- Includes things such as comment forms, showing tags and categories, displaying author names, etc

Integration vs unit vs behavioural testing

- Good for testing code and as a development methodology
- Automated testing
- Explain the difference between the three
- Mention they’re covered in more depth in sub-chapters

Unit Testing

Unit Testing

Unit testing tests individual components. Each test runs in isolation, and tests only a single item, such as a function or method. If a unit test involves multiple interacting objects, then you have written an integration test.

For example, if I have this function:

```
1. function add( $a, $b ) {
2.   return $a + $b;
3. }
```

A unit test might check:

- If 5+5 = 10
- That 2+3 is not 7
- That 0+0 does not fail

Tools for Unit Testing

- PHPUnit
- PHPSpec

Helpful Projects and Further Reading

- John P Blochs WP Unit Test Starter project
- WP Mock
- Writing Unit Tests for WordPress

Behaviour Testing

Behaviour Testing

Also known as Behaviour Driven Development, this kind of testing tests the entire stack. For example a behavioral test may start by visiting a webpage, clicking a button, and checking that an expected string was found.

BDD is good for testing business requirements. It generally falls into 2 types, story based testing, and code-based testing. An example of story based testing would be Behat, which uses a human readable format so that clients can read the tests in plain English (with support for other languages included).

A major benefit of these types of tests is that the tests themselves do not need to load the WordPress PHP environment. A test site can be put up on a server, and the tests can be pointed at the test site. Tools such as Behat then run as if they were a user controlling a browser (which is exactly how most Behat Mink tests work). This makes it one of the easiest ways to introduce testing, and the easiest to learn first

Tools for Behaviour Testing

- http://behat.org/Behat
- SpecBDD

Test Driven Development

Test Driven Development

Explain theory and idea behind TDD

WP_UnitTestCase

WP_UnitTestCase

Explain the WP_UnitTestCase

Further Reading

- http://taylorlovett.com/2014/07/04/wp_unittestcase-the-hidden-api/

Servers And Deployment

Servers And Deployment

Test Your Changes

Always test changes on a local environment before copying them to your production server (see *Getting Started > Local Development Environment*.)

Make sure the development server has error-reporting turned on so you catch anything that would be invisible on the live site.

Make sure your local environment is as similar to your production server as possible (see also *Migrations*.)

- are you running the same PHP version?
- do you have the same PHP.ini settings?
- do you have the same version of MySQL?
- do you have the same Apache or Nginx version & configuration?
- do you have the same version of WordPress with the same plugins enabled?

Consider using a staging server to help with this.

If you're copying a database from development to production (or vice-versa), you'll need to change the URLs. See the *Migrations* section in this chapter.

Use Version Control

If you use a version control or source code management system such as Git, you'll be able to roll back your changes when (not if) you make a mistake. You can 'push' your changes with a single command and updated files will first be copied to a temporary area before being deployed simultaneously. This avoids the site ever being left in a broken state if you have a slow connection.

A good server host provides SSH access, but if your hosting provider only allows SFTP access, consider using git-ftp, so you can minimise the time it takes to update the site and the chance of forgetting to upload any new files. You'll still benefit from version control locally or if you're working with other developers.

Avoid using file editors on control panels like CPanel.

Built-in Editors

WordPress has theme and plugin editors built into the admin area.

Avoid using them.

- The editor is a simple HTML textarea - you get none of the code highlighting, formatting or syntax checking of an IDE or basic text editor, it might seem quicker but it’s also much easier to make mistakes.
- there’s no version control, you don’t get a list of what you changed and the only protection is your own backups (if you remembered to make any).
- A significant error might break WordPress in such a way that you can no longer access the editor itself.

The theme/plugin editor is also a potential security risk: if someone gains access to an administrator account they can edit sensitive files on the server.

You can turn off file editing completely by adding this line to wp-config.php

```
1. define( 'DISALLOW_FILE_EDIT', true );
```

WP CLI

WP CLI

WP CLI is a command line tool maintained by numerous experienced WordPress developers and core contributors. It’s similar to Drupals Drush.

Deploying a New Install

To download WordPress into a folder on the command line, use this command:

```
1. wp core download
```

This will download the latest version of WordPress into the current directory. Next you’ll need to create your `wp-config.php` :

```
1. wp core config --dbname=testing --dbuser=wp --dbpass=securepswd
```

Finally, run the install command to set up the database:

```
1. wp core install --url="example.com" --title="Example Site" --admin_user="exampleadmin" --
   admin_password="changeme" --admin_email="example@example.com"
```

You should now have a fresh new WordPress install ready to log in to.

Multisite

If you want to create a multisite install, use the `wp core multisite-convert` command:

```
1. wp core multisite-convert --title="My New Network" --base="example.com"
```

Importing Content

You may have content you want to pre-add or migrate to your new install. For this WP CLI provides the content import and export commands. These commands accept or create standard wxr files, the same format used by the WordPress export and import plugin in the admin interface.

Use this command to import:

```
1. wp import content.wxr
```

Use this command to export:

```
1. wp export
```

Migrations

Migrations

There are a number of things to take note of when moving sites from server to server, and when changing their URLs.

Since server moves and domain changes are a large topic, we're going to cover only the most important things.

Imports and Exports

When performing imports and exports, there are a lot of pitfalls as your site increases in size. To avoid problems, do the following:

- **Use WP CLI** to run imports and exports. The admin UI is limited by the PHP time limits, if your import or export doesn't finish within the available time, it can fail. Running in a terminal using WP CLI gives you unlimited time to do it
- **Ask the exporter to generate in 5MB chunks**. This reduces the memory requirements of each individual import, and gives a lot more flexibility
- **Disable image resizing**. This speeds up importing of images, letting you manually resize in bulk once the content is imported.

Server Moves

On a new server, the environment may not match the old environment, and so you should look out for:

- Older PHP versions. Using newer PHP features on a server, then moving to an older version could cause your code to Fatal error. Check before hand what version of PHP is used and make sure it's the same or greater.
    - Run `php -v` or use the `phpversion()` command if you don't have access to the server.
    - PHPStorm uses: use the *PHP Language Level* setting to check for errors automatically.
- File system changes. Not every server puts your site at /srv/www, some use /var/www, and you should make sure that any hardcoded paths are changed to match. You can normally substitute `$_SERVER['DOCUMENT_ROOT']` instead.

URL changes

If you're changing your sites URL, you may or may not be moving server. If you do change URL however, it's not enough to change the DNS and expect things to work.

WordPress stores data in the database that contains your sites URL.

A new user may decide to use a small SQL command to search for all instances of the old URL, and replace them with the new URL. This will not work.

The reason for this is that some data is stored in serialised PHP data structures. These serialised strings contain the length of the URL, and if your URLs length changes, the data structures are no longer valid. This causes issues when you attempt to load your site.

To get around this, a number of tools are available that can look inside the data structures and modify them correctly. We recommend using WP-CLI's search-replace command, but other solutions exist.

Security

Security

Salts

Your passwords and cookies are stored with salts applied. Salts are strings of data that are kept secret, and hashed together with important data so that it’s harder to guess. This way a hacker can’t just run through every password and generate a rainbow table of all possible results and brute force every website. Instead they need to generate a new table for every site they target after acquiring the secret salts used. There is an API to provide salts and secret keys at wordpress.org, which you can then copy paste into your `wp-config.php` .

Escaping

When outputting data, you should escape it. For example, if you output a css class, you should use `esc_attr` , otherwise, an attacker could sneak in the value `classname"><script>alert('hello');</script><span` and run arbitrary code on your site.

An important part of escaping however, is to escape as late as possible. If you escape a variable once, then use it 5 times, that variable may be modified at any point between escaping and output, so always escape at the moment of output.

- Sanitise early
- Escape Late
- Escape Often

Nonces

In the days of MySpace, a user could add an image to their profile, and set the `src` tag as `/logout.php` . Any user who visited their profile would be immediatley logged out. This is an example of a CSRF attack or Cross Site Reference attack.

In order to get around this, we use nonces. Nonces are small tokens that can be passed around to validate an action. For example, a form may contain a nonce, which is then checked for when processed. This makes sure that all form submissions came from the form, and not a malicious or unintended script.

@todo: Add notes on how to use nonces effectively

Note: In the United Kingdom, a nonce is a name for a child sex offender, be careful of using the word out of context

The Location of `wp-config`

- You can move it one level up so it’s not in a web accessible location

Table prefixes

- Don’t use the default wp_
- Notes on automated attacks

User ID 1

- Don’t call it ‘admin’
- Don’t give it administrator priviledges

Roles and Capabilities

- What they are

Removing vs Hiding Settings Pages

- Hiding things with CSS doesn’t make it secure
- People have dev tools too
- Automated tools ignore CSS
- how to remove admin menus and change the capabilities needed to do things

Custom Password Reset Code

- Some people write their own password reset facilities. This is bad
- If you really must, make it a forgotten password link, don’t make it actually show your password

timthumb.php

- Don’t use it
- There’s an image API for that
- timthumb was disowned by its creators and is officially no longer supported
- Banned on a number of managed WordPress hosts

SSL

All big clients deserve an SSL certificate. If you’re running an e-commerce site, this is especially true, and your entire site should be using SSL for all logged in users.

- A note on public wifi, unsecured wifi, and snooping
- Maybe mention firesheep?

Admin Only SSL

- If your site isn’t an ecommerce site, but you have users who visit the backend, their logged in sessions should be sent over an https connection.
- Explain how

Myths

There are a lot of feel good security fixes that float around, that do nothing to help your security, waste your time, and sometimes increase the risk. Here are a few:

Hiding the Admin and Login URLs

- Some people try to change the admin and login URLs in hopes it will fool attackers and automated tools
- WordPress adds in /admin/ and /login/ rewrite rules in the newer versions so moving the files is pointless
- It can break some functionality in code without necessary care
- Trying to go to the admin URLs will redirect you to the changed login URL anyway, and if you fix that then the modal box that shows in the admin screen when your session expires will be broken too

Deactivated Plugins & Themes

- Because of how PHP works, deactivated plugins can still be hit from a users web browser
- Badly written plugins might do things if the right URL is loaded, even if they’re not activated. This is especially true of plugins with their own AJAX endpoints that don’t use the WP AJAX API.

Recovering From Attacks

- Take and use regular backups
- Download a fresh copy of WordPress and extract it over the top of your existing install to make sure that WP Core is unmodified
- Check your plugins and code against version control

Community

Community

WordCamps

WordCamps are short, 1-2 day conferences that focus on everything WordPress. They are designed to have both a general focus (on blogging, writing content, marketing websites and the business surrounding WordPress) and a technical focus (aimed at developers writing plugins and themes).

There are over 50 WordCamps every year held all over the world in over 40 countries, and are a great way to explore the community. Many WordCamp sessions are placed onto WordPress.tv, and is a great opportunity to help the WordPress community, either by speaking, volunteering, or sponsoring.

You can find out all upcoming WordCamps at WordCamp Central.

Contributor Days

Contributor days, usually held after WordCamps (but can be independent), are events that are set up to help you contribute to WordPress. The benefits to contributing to WordPress are numerous, both for a business looking to get more exposure in and amongst the WordPress community, to lone developers looking to grow their skills working in a team on a massive project.

You do not need to be code proficient to contribute to WordPress. They are looking for a wide range of skills, like support, theme review, as well as accessibility.

Look at your local WordCamp if they are holding a Contributor day. Alternatively, if you’re based in the United Kingdom, you can find the latest WordPress Contributor day at http://www.wpcontributorday.com/.

- Local User Groups
- .org Support Forums
- IRC Channels
- WordPress Stack Exchange
- WordPress Slack

Credits

Credits

Many people have contributed to WordPress The Right Way, and they've done so using GitHub. You can view the full list of contributors here, and you can fork and submit your own Pull Request to join them!

日文版

はじめに

WordPress The Right Way

この本はWordPress開発者のためのWordPress開発者によるベストプラクティスを凝縮した資料です。

これは生きているドキュメントで、役に立つ情報やサンプルなどで常に更新され続けます。

参加協力の方法

協力してくださる方はGitHubへお願いします。 メインのリポジトリが変更されると、Gitbook.ioに自動的にプッシュされます。

この本の編集は、テキストエディタでマークダウンファイルのアップデートするか、Gitbook desktop appでリポジトリを開いて編集することで可能です。デスクトップアプリではライブプレビューも可能です。

ライセンス

さあ、始めよう

さあ、始めよう

PHPの基礎

この本では、PHPの基礎的な知識があることを前提としています。その知識には次の項目が含まれます:

- 関数
- 配列
- 変数
- ループと条件
- クラスとオブジェクト
- クラスの継承
- ポリモーフィズム
- `POST` と `GET`
- 変数のスコープ

これらの概念の十分な理解がない場合、先に進む前にしっかりと理解しておいたほうがいいでしょう。

また、PHPシンタックスハイライト機能を持つコードエディターを持っていることも前提としています。次も訳に立ちます:

- 自動インデント
- 自動補完
- ブレスマッチング
- 構文チェック

ローカル開発環境

ローカルの開発環境を持つことも重要です。PHPファイルを変更し、本番サーバーのそれをアップデートして無事を祈るという昔の日々は去りました。

ローカルの開発環境を使えば、より速く作業でき、ファイルのアップロードやダウンロードが必要無くなり、不安定なインターネット接続に翻弄されることもなく、ウェブページの読み込みを待たされることもなくなります。ローカルのサーバスタックを使えば、Wifiや携帯電話の電波のないトンネルに入った列車の中でも作業できますし、本番サーバにデプロイ前にテストもできます。

ローカルの開発環境の構築にはいくつか方法がありますが、大別すると2つのカテゴリーになります:

- バーチャルマシーン
- ネイティブのサーバスタック

1つ目のタイプの環境は、Vagrantなどのプロジェクトを通常は含んでいて、標準化された一貫性のある仮想マシンを利用します。

2つ目のタイプは、自分のオペレーティングシステムにサーバーソフトウェアを直接インストールするタイプです。これを簡単に行うための様々なツールがありますが、その環境は独自のものになってしまうためデバッグが難しくなります。これらはLAMP(Linux Apache MySQL PHPの頭文字)スタックと呼ばれることもあります。

IIS

Microsoft Internet Information Services はWindowsベースのサーバを動かすサーバーソフトウェアです。Windowsに付属していて、インストールするコンポーネントによって様々な変種があります。WordPressのコミュニティではIISのセットアップに関する知見は多くありません。たいていのリモートサーバーはApacheもしくはNginxで動いていて、開発者の知見もそちらの方に集中しています。

IISの選択は難しいものとなるでしょう。

バージョン管理

チームでの作業にはバージョン管理は必須です。バージョン管理システムは長期間に渡って変更を追跡し、開発者が共同で作業をしたり、変更を元に戻したりできるようにします。

Git

Linuxの作者、リーナス・トーバルズによって作られたGitは人気のある分散型のシステムです。GitHubをお使いであれば、すでにご存知かと思います。

Subversion

svnとしても知られていて、集中管理型のバージョン管理システムです。WordPress.org上のプラグインとテーマのリポジトリに使われています。

コーディングスタイルガイド

コーディングスタイルガイド

きれいなコード

コードは読みやすく、メンテナンスしやすい状態に維持することが重要です。

インデント

WordPressではインデントはタブを使い、見た目は半角スペース4個分になるようにします。インデントはコードの読みやすさにとって重要で、各命令文はそれぞれの行に置くべきです。インデントがないと動作を理解するのがとても難しくなり、ミスも起こりやすくなります。また、フォーラムやStack Excahngeでも答えを得るのが難しくなるでしょう。

良いエディターは自動インデント機能を持っていて、たいていは修正が必要なコードがあればファイルの再インデントが可能です。

チームの全員がEditor Configを利用して同じスタイルを使用するようにすると確実でしょう。これには各種エディターのプラグインが含まれていて、各自で自分の好みのエディターを使うことができます。

例えば、以下の `.editorconfig` ファイルは半角スペース4文字分の幅のタブをインデントとして使用する、上記のルールを強制的に適用します。

```
[*.php]
indent_style = tabs
indent_size = 4
```

PHPタグスパム

`<?php` と `?>` タグは控えめに使うべきです。例えば:

```
<?php while( have_posts() ) { ?>
    <?php the_post(); ?>
    <?php the_title(); ?>
    <strong><?php the_date(); ?></strong>
    <?php the_content(); ?>
<?php } ?>
```

は次のようにしたほうが読みやすいでしょう:

```
<?php
while( have_posts() ) {
    the_post();
```

```
    the_title();
    ?>
    <strong><?php the_date(); ?></strong>
    <?php
    the_content();
} ?>
```

良いガイドラインは何を表示すべきかで判断することです。そして2つを混ぜるのではなく、1つのタグ内ですべてを表示させるようにします。

リント

多くのエディターはシンタックスチェックをサポートもしくはビルトインで持っています。これらはリンターと呼ばれています。優れたエディターを使えばシンタックスエラーが強調表示されたり指摘されたりします。

例えば、PHPStormではシンタックスエラーは赤の下線が付けられます。

コーディング規約

WordPressでは独自のコーディング規約に従っています。これはPSRの規約とは違いがあります。例えば、空白スペースではなくタブを使い、開始ブラケットは同じ行に置きます。

このコーディング規約についてはWordPress貢献者ハンドブックで詳述されています。

- HTMLコーディング規約
- PHPコーディング規約
- JavaScriptコーディング規約
- CSSコーディング規約

PHPコードスニッファーとPHP CSフィクサー

PHPコードスニッファーはコーディング規約の違反を見つけるツールです。これは多くのエディターでサポートされていて、そうした違反を自動的に修正する2番めのツールのサポートも含まれているます。

これを使うにはWordPressのコーディング規約定義が必要です。 PHPStorm用の説明とともにここで見つけることができます。 .

デバッギング

デバッグ

WordPressの開発時には自分のコードが動作することは重要ですが、動作に失敗した時は、その失敗が非常に見つけにくいことがあります。そうした失敗を見つけられるようにする必要があります。

この章では以下扱います:

- どのようなエラーが発生しているのかを見つける
- どのようにその問題をデバッグするか
- 助けになるプラグインやツール
- デバッグを容易にするWordPressの機能
- 問題の発生を未然に防ぐ方法とミスを見つけるための簡単な自動化されたツール

先にいく前に画面が真っ白になった時の対応を。

画面が真っ白になったとき

WordPressの新人開発者が出くわすよくある問題の1つが、画面が真っ白になって動かなくなることでしょう。これはPHPで致命的なエラー(fatal error)が発生した時に起こります。多くの新人WordPress開発者は変更することによって対応し、この問題が何処かへ言ってしまうことを望むことでしょう。しかし、この事態に対応するにはもっといい方法があります。

PHPではエラーが発生するとどこかにそれを記録し、それにより何が原因で、どこでそれが発生しているのかを見つけることができます。

開発者にとっての良い出発点は `WP_DEBUG` を有効にすることです。

エラーロギング

エラーの記録

エラーの記録にはいくつか種類がありますが、主なものは以下になります:

- 表側にエラーを表示する
- ログファイルにエラーを書き込む
- 何も表示させない

本番環境ではエラーはログファイルに書き込んだほうがいいでしょう。

警告(Warning) vs エラー(Error)

PHPがどのように設定されているかにもよりますが、警告(Warning)が表示されることもあります。警告(Warning)はPHPを止めてしまうわけではありませんが、問題となり得ることを示します。例えば:

```
1. $my_array = array(
2.     'alice' => 5,
3.     'bob' => 6
4. );
5. echo $my_array['eve'];
```

ここでは `$my_array` 内の'eve'エントリーをエコーしようとしているのですが、そのようなエントリーはありません。PHPは空の値を作り警告を記録します。警告(Warning)はバグやミスを指し示すのです。

PHPエラーレポーティング

`php.ini` に何が定義されていのかによりますが、PHPにはエラーレポーティングレベルというものがあります。そのレベル以下のものはすべて無視されるか単なる警告とみなされます。これはサーバーによって様々です。

`@` エラー制御演算子

エラー制御演算子 `@` は決して使ってはいけません。これはコードのエラーと警告を隠しますが、普通の人が望むような動作はしません。

`@` はコマンド上のエラー報告レベル設定によって動作するので、エラーが記録されません。これは、普通の人が望むような、エラーの発生を防止するものではありません。これは致命的なエラーが捉えられないか、記録されないようにすることを意味します。エラー制御演算子 `@` の使用は避け、このインスタンスはすべて疑いをもって扱うようにします。

エラーの扱い

エラーの取り扱い

返り値

`is_wp_error`

`WP_Error`

ツール

ツール

デバッグ用ツールは2つのカテゴリーに分かれます:

- 何か発生した時にそれを診断し、問題点を明らかにするためのツール
- ミスやエラーの発生自身を未然に防ぐツール

古の格言はここでも当てはまります: 予防は治療に勝る

デバッグ用ツール / プラグイン

WordPressのリポジトリからDeveloperプラグインをインストールすると、たくさんの種類のデバッグ用ツールに簡単にアクセスできるようになります:

- Log Deprecated Noticesは、deprecated(非推奨)の関数の利用のログを取ります。
- Debug BarはPHPのNotices/Warnings/Errorsのデバッギング、SQLクエリーのレビュー、キャッシュ動作の分析、その他多くのためのインターフェイスを提供します。また、プラグインで拡張可能です。
- 例えば、Debug Consoleはとても便利です。

クエリーモニター

XDebug

PHPデバッガー

ブラウザーのウェブインスペクター

予防

PHP Mess Detector

SCheck

wp-config.php

`wp-config.php`の内容

`wp-config.php` ファイル上にはPHPのいくつかの定数が今のところ、WordPressのコードを改善したりデバッグの助けになったりします。

`WP_DEBUG`

これはWordPress バージョン 2.3.1で含まれるようになったオプションです。

デフォルトでは `false` にセットされていて、警告やエラーを表示しないようになっていますが、すべての**WordPress**の開発者はこのオプションを有効にするべきです。

ログの有効化

```
1. define( 'WP_DEBUG', true );
```

ログの無効化

```
1. define( 'WP_DEBUG', false );
```

この値は文字列ではなく真偽値でなくてはなりません。

WordPressバージョン 2.3.2であとからマイナーなパッチが取り込まれ、データベースのエラーログに対するより粒度の細かいコントロールを可能になりました。

さらにその後、バージョン2.5でエラーレポーティングのレベルがE_ALLに引き上げられました。これによりNotices(注意)とDeprecation(非推奨)メッセージを表示するようになりました。

メモ:

このオプションを有効にするとAJAXリクエストで問題が発生するかもしれません。この問題はAJAXレスポンスの出力にNoticeが表示されてしまい、**XML**と**JSON**を壊してしまうことに関連します。

`WP_DEBUG_LOG`

`WP_DEBUG` を使い、この定数を `true` にセットすると、NoticeやWarningのログをファイルに記録します。

`WP_DEBUG_DISPLAY`

`WP_DEBUG` を使いこの定数を `true` にセットするとNoticeやWarningをスクリーンに表示するかどうかを選択できます。

メモ:

もしこれらの変数が期待していた出力を産み出さないのであれば、Codexのロギングセットアップに関するセク

ション(英語)（日本語)を読むといいでしょう。

`SCRIPT_DEBUG`

ミニファイされたバージョンのCSSやJavaScriptのファイルをデフォルトでプラグインやテーマに持たせるのはよくありません!

開発用とミニファイされたバージョンのファイルを作成するというWordPressの考えに従うのはよい方法で、自分のプラグインには両方のファイルを持たせるべきです。その変数をベースにすればどちらかをエンキューさせることができます。

デフォルトではこの定数は `false` にセットされていて、WordPressからのCSSやJavaScriptをデバッグしたいときはこれを `true` にするといいでしょう。

ログ取得の有効化

```
1. define( 'SCRIPT_DEBUG', true );
```

この値は文字列ではなく真偽値でなくてはなりません。

`true` にセットすると `wp-includes` と `wp-admin` にあるWordPressのデフォルトファイルは開発バージョンになります。

`CONCATENATE_SCRIPTS`

WordPressの管理画面では、依存性とエンキューの優先度に応じてすべてのJavaScriptファイルが1つのリクエストに連結されます。

この機能を無効にするにはこの定数を `false` に設定します。

```
1. define( 'CONCATENATE_SCRIPTS', false );
```

`SAVEQUERIES`

データベースを扱うときには、プラグインやテーマ内で起こっていることをデバッグできるように、クエリーを保存したいと考えることでしょう。

Make `$wpdb` save Queries

```
1. define( 'SAVEQUERIES', true );
```

メモ: これ*true*にすると*WordPress*が遅くなります

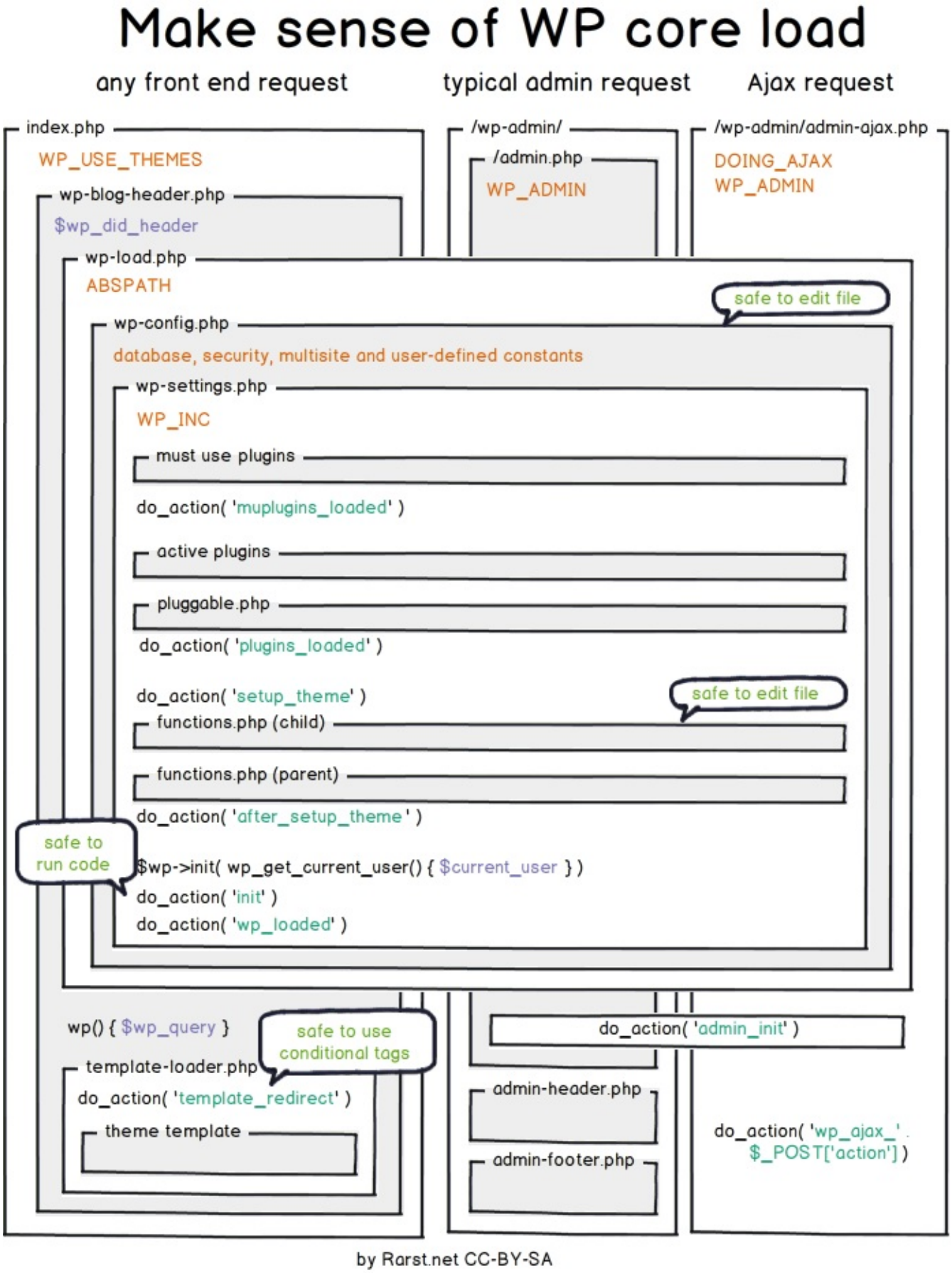
コア

コア

WordPressコアとは、WordPress自身を動かしているコードのことです。wordpress.orgからWordPressをダウンロードすると取得できるものからテーマとプラグインを取り除いた分になります。

読み込みプロセス

todo: コアの読み込みプロセスに関するメモとRarstsのブログ記事への参照



jQueryの登録解除

コアに同梱されているjQueryの登録を解除して自分のコピーやGoogle CDNのものを追加しようとするプラグイン

やテーマの開発者がたくさんいますが、互換性の問題を引き起こす可能性があるので、それはやめましょう。

その代わり、WordPressに同梱されているjQueryのコピーを使い、テスト時には最新のWordPressで使われているバージョンを目標にしましょう。これにより、プラグイン間の互換性を最大化します。

コアファイルの変更

何かを追加したり取り除いたりするためにコアの一部を修正したい誘惑に駆られるかもしれませんが、それは絶対にダメです。WordPressがアップデートされるとその変更はすべて失われます。

その代わり、フック/アクションとフィルターを使ってコアの挙動を変更しましょう。

.orgでのコアの開発作業

- trunk
- develop
- grunt builds develop into trunk
- Trac
- code freeze
- release

データ

データ

WordPressのデータモデルの説明です:

投稿タイプ

投稿、ページ、添付、そしてメニュー、これらはすべて違う種類の投稿タイプです。独自の投稿タイプを登録することもできます。投稿タイプを変更した際はパーマリンク設定で(変更がなくても)「変更を保存」をクリックしてください。こうしないと404(ページがみつかりません)になってしまいます。

コメント

コメントは独自のテーブルを持ち、メタデータの保存も可能です。開発者がこれを利用することはまれですが、面白いことが可能です。

タームとタクソノミー

タグとカテゴリーはタクソノミーの一種です。それぞれのタグとカテゴリーはタームと呼ばれます。独自のタクソノミーを登録することも可能です。

例えば、色のタクソノミーを考えてみましょう。紫色は色タクソノミーに含まれる一つのタームです。タクソノミーの登録を変更したら忘れずにパーマリンクを更新してください。

タクソノミーのタームにはオブジェクトIDが付きます。これらのIDは通常は投稿IDですが、これは純粋に慣習です。ユーザーやコメントのタクソノミーも可能です。ユーザータクソノミーはユーザーを場所や役割に応じてグループ分けするときに便利でしょう。

メタ

投稿、コメント、ユーザーはメタを持ちます。

オプション

オプションは専用のテーブルにキーバリューのペアとして保存されます。いくつかのオプションは自動読み込みフラグがセットされていて、クエリーの数を減らすために各ページの読み込み時に自動的に読み込まれます。

トランシエント

トランシエント(Transients)はオプションとして保存され、一時的なキャッシュとして使用されます。

オブジェクトキャッシュ

デフォルトでは、WordPressはメモリー内のキャッシュを利用しますが、これはページの読み込みと読み込みの間では存続しません。APCやmemcachなどを利用してこれを存続させるプラグインがいくつか有ります。

クエリー

クエリー

この章では2つの種類のクエリーについて説明します。ポストクエリー、タクソノミークエリー、コメントクエリー、ユーザークエリー、そして通常のSQLクエリーです。

ポストクエリー

メインループ

WordPressによって表示されるすべてのページにはメインクエリーがあります。このクエリーはデータベースから投稿を取得し、どのテンプレートが読み込まれるべきなのかを決定するのにも使われます。

テンプレートが読み込まれるとメインループが開始され、メインクエリーによって見つけられた投稿をテーマが表示します。

```
1. if ( have_posts() ) {
2.     while ( have_posts() ) {
3.         the_post();
4.         // display post
5.     }
6. } else {
7.     // no posts were found
8. }
```

メインクエリーとクエリー変数

メインクエリーはURLがを使って生成され、 `WP_Query` オブジェクトによって表示されます。

このオブジェクトはクエリー変数を使って何を取得するのかを命令されます。これらの変数は最初にクエリーオブジェクトに渡され、有効なクエリー変数の一部となります。

例えば、クエリー変数 'p' は特定の投稿タイプを取得するのに次ように使われます:

```
1. $posts = get_posts( 'p=12' );
```

これはID12の投稿を取得します。完全なオプションの一覧はCodexの `WP_Query` のエントリーで参照できます。

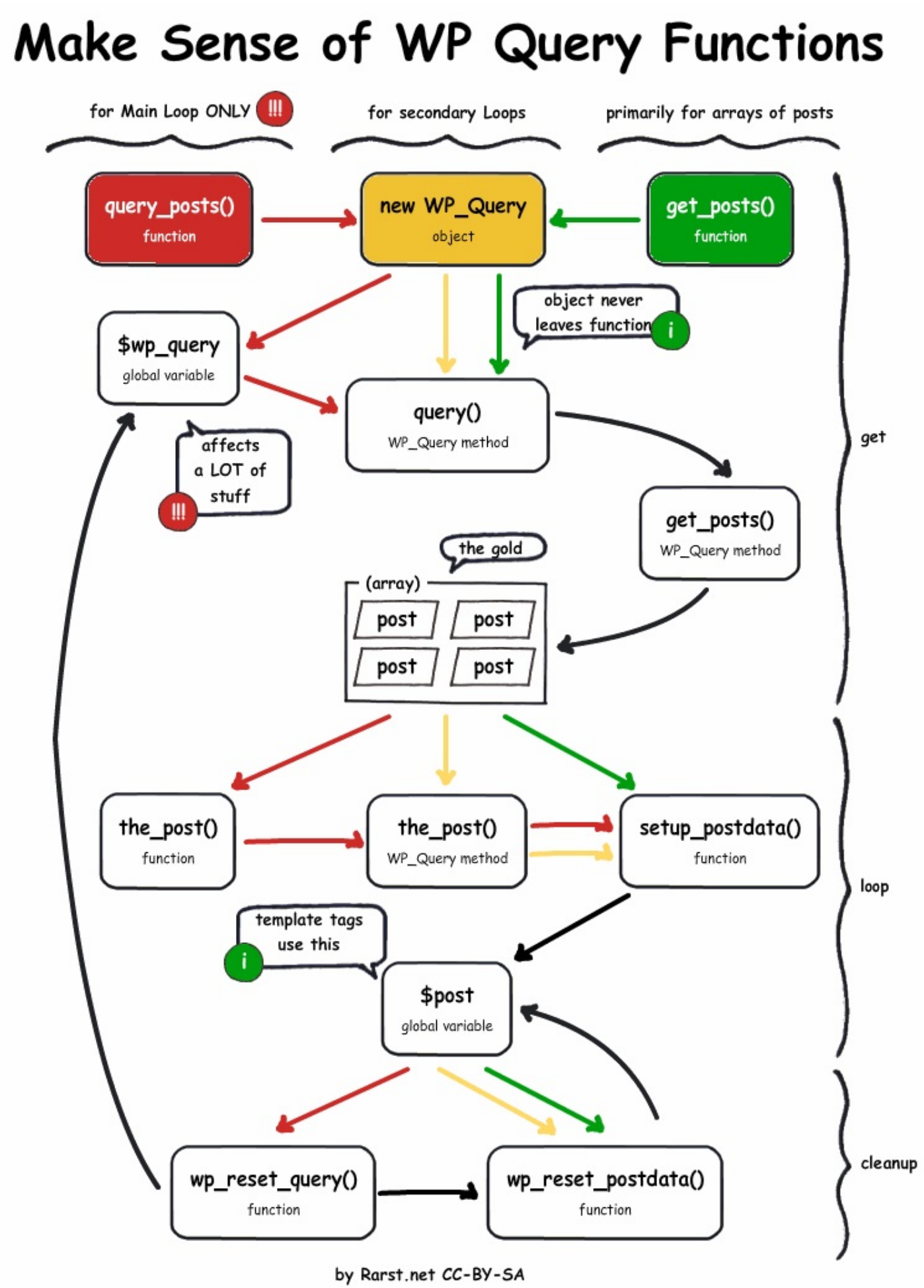
クエリー発行

データベースから投稿を取得するには投稿クエリーを作成する必要があります。投稿取得のすべてのメソッドは `WP_Query` オブジェクトの上に層になっています。

これを行うには3つの方法があります：

- `WP_Query`
- `get_posts`
- `query_posts`

このダイアグラムは各メソッドで何が行われているのかを説明しています：


Make Sense of WP Query Functions
for Main Loop ONLY !!!
for secondary Loops
primarily for arrays of posts
query_posts()
function
new WP_Query
object
get_posts()
function
object never
leaves function
$wp_query
global variable
query()
WP_Query method
affects
a LOT of
stuff
!!!
get_posts()
WP_Query method
the gold
(array)
post
post
post
post
get
the_post()
function
the_post()
WP_Query method
setup_postdata()
function
loop
template tags
use this
$post
global variable
wp_reset_query()
function
wp_reset_postdata()
function
cleanup
by Rarst.net CC-BY-SA

`WP_Query`

```
1. $query = new WP_Query( $arguments );
```

すべての投稿クエリーは `WP_Query` オブジェクトのラッパーです。 `WP_Query` オブジェクトはクエリー(例えばメインクエリー)を表していて、次のような便利なメソッドを持っています:

```
1. $query->have_posts();
2. $query->the_post();
```

たいていのテーマにみられる関数の `have_posts();` と `the_post();` はメインクエリーオブジェクトのラッパーです:

```
1. function have_posts() {
2.     global $wp_query;
3.
4.     return $wp_query->have_posts();
5. }
```

`get_posts`

```
1. $posts = get_posts( $arguments );
```

`get_posts` は `WP_Query` と似ていて同じ引き数を取りますが、リクエストされた投稿のすべてを含んだ配列を返します。投稿ループの作成を意図していないのであれば `get_posts` は使うべきはありません。

`query_posts` は使わない

`query_posts` は極端に単純化されていて、ページのメインクエリーを変更する方法としては、クエリーの新しいインスタンスでそれを置き換えるので問題のある方法です。

これは非効率(SQLクエリーを再度走らせます)で、特定の状況(特にページングを扱うときに)で完全に動作しなくなります。この用途では `pre_get_posts` フックの利用など、モダンなWordPressのコードにはより信頼性のあるメソッドを使うべきです。 `query_posts()` を使ってはいけません。

クエリー発行後の後始末

`wp_reset_postdata`

`WP_Query` もしくは `get_posts` を使うとき、 `the_post` もしくは `setup_postdata` を使ってカレントの投稿オブジェクトをセットできます。これを行ったら、ループが終わった時点で後始末をする必要があります。 `wp_reset_postdata` を呼び出すことでこれを行います。

`wp_reset_query`

`query_posts` を呼び出した時、その動作が終わったらメインクエリーを戻す必要があります。これは `wp_reset_query` で行えます。

`pre_get_posts` フィルター

メインクエリーを変更して、そのページに何か別のものを表示させる必要がある場合には `pre_get_posts` を使うといいでしょう。

アーカイブから投稿を取り除いたり、検索時の投稿タイプを変更したり、カテゴリーを除外したりなどなどのためにこれはよく利用されます。

以下は投稿のみを検索対象にするためのCodexに記載されているサンプルです:

```
function search_filter($query) {
    if ( !is_admin() && $query->is_main_query() ) {
        if ($query->is_search) {
            $query->set('post_type', 'post');
        }
    }
}

add_action( 'pre_get_posts', 'search_filter' );
```

このフィルターはテーマの `functions.php` もしくはプラグイン内で使えます。

タクソノミークエリー

タクソノミー(投稿のカテゴリーやタグも含む)を扱うときは古いヘルパーAPIよりも包括的な以下の様なAPIに頼るほうが安全です:

- `get_taxonomies`
- `get_terms`
- `get_term_by`
- `get_taxonomy`
- `wp_get_object_terms`
- `wp_set_object_terms`

APIのひとセットを学ぶほうがより簡単ですし、カテゴリーとタグは単なるタクソノミーの一種として考え、 `get_category` などの古い関数の組み合わせとマッチングとは考えないほうがいいでしょう。

コメントクエリー

```
$args = array(
   // args here
);

// The Query
$comments_query = new WP_Comment_Query;
$comments = $comments_query->query( $args );

// Comment Loop
if ( $comments ) {
    foreach ( $comments as $comment ) {
```

```
        echo '<p>' . $comment->comment_content . '</p>';
    }
} else {
    echo 'No comments found.';
}
```

ユーザークエリー

```
$args = array(
    //
);

// The Query
$user_query = new WP_User_Query( $args );

// User Loop
if ( ! empty( $user_query->results ) ) {
    foreach ( $user_query->results as $user ) {
        echo '<p>' . $user->display_name . '</p>';
    }
} else {
    echo 'No users found.';
}
```

SQL

WPDB

よく知らない方は、投稿記事を取得するために生のSQLクエリーに頼ろうとする誘惑に駆られることでしょう。しかし、それは最後の手段です。

SQLクエリーを作る必要のある場合は `WPDb` オブジェクトを使いましょう。

dbDelta とテーブル作成

`dbDelta` 関数は現行のテーブル構造を調べ、望ましいテーブル構造を比較し、必要に応じてテーブルを追加もしくは変更します。そのため、アップデートにはとても便利な時があります。

dbDelta関数は好き嫌いがあるかもしれません。例えば:

- 各フィールドはSQLステートメント内の各行に置く必要があります。
- `PRIMARY KEY` とプライマリーキーの定義の間には半角スペースを2つ入れる必要があります。
- 同義語の `INDEX` ではなくキーワードの `KEY` を使う必要があり、最低でも一つのキーを含める必要があります。
- フィールド名の周りではアポストロフィとバッククォートは使えません。
- `CREATE TABLE` は大文字にする必要があります。

こうした注意事項を念頭に置いて、実際にテーブルを作成もしくは更新する関数を作ります。$sql変数内で独自のテーブル構造を置き換える必要がでてくるでしょう。

さらに詳しくは

- You don’t know query, a talk by Andrew Nacin
- When you should use WP_Query vs query_posts, Andrei Savchenko/Rarst
- Creating Tables With Plugins - Codex
- pre_get_posts - Codex

ルーティング

ルーティング

- An explanation of how rewrite rules generate a query, which loads a template, which displays a page
- Custom Query variables & routing
- Adding a rewrite rule
- Flushing rewrite rules
- Debugging rewrite rules
- Clashes & slugs
- テンプレートを読み込んだりページを表示したりするリライトルールがどのようにクエリーを生成するのかの説明
- リライトルールの追加
- リライトルールの更新
- リライトルールのデバッグ
-

テンプレート

テンプレート

`get_template_part`経由でテンプレートを読み込む

`locate_template`関数について書く。

テンプレートがどのように選択されるかとテンプレート階層について

メインクエリーを使ってテンプレートがどのように選択されるのかの記述。テンプレートがどのように選択されるのか、子テーマがどのように読み込まれるのかについて。テンプレート階層図を表示。

functions.phpとプラグイン

スタイルシートの読み込み

スタイルシートを正しく読み込ませる

テンプレートとプラグイン

`template_include`フィルター

フォーム

テーマ内で別のスタンドアローンのPHPファイルをサブミットするフォームはよくありません。ページテンプレート上で基本的なフォームサブミッションをどのように扱うかの例

下層ページ

投稿やアーカイブと関連付けられていないページ/URL、例えばショッピングカートやAPIのエンドポイントが必要なとき。

さらに詳しくは

- テンプレートダイアログへのリンク
- インタラクティブなテンプレートダイアログ

JavaScript

JavaScript

WordPressにはJavaScriptのための依存マネージャーとエンキューの機能が組み込まれています。そのままのJavaScriptを埋め込む `<script>` タグは使ってはいけません。

登録とエンキュー

JavaScriptファイルは登録するようにします。登録により依存マネージャーにスクリプトがあることを知らせます。ページにスクリプトを埋め込むにはな必ずエンキューさせます。

ではスクリプトの登録とエンキューをしてみましょう。

```
1.  // Use the wp_enqueue_scripts function for registering and enqueueing scripts on the front end.
2.  add_action( 'wp_enqueue_scripts', 'register_and_enqueue_a_script' );
3.  function register_and_enqueue_a_script() {
4.      // Register a script with a handle of `my-script`
5.      //  + that lives inside the theme folder,
6.      //  + which has a dependency on jQuery,
7.      //  + where the UNIX timestamp of the last file change gets used as version number
8.      //    to prevent hardcore caching in browsers - helps with updates and during dev
9.      //  + which gets loaded in the footer
10.     wp_register_script(
11.         'my-script',
12.         get_template_directory_uri().'/js/functions.js',
13.         array( 'jquery' ),
14.         filemtime( get_template_directory().'/js/functions.js',
15.         true
16.     );
17.     // Enqueue the script.
18.     wp_enqueue_script( 'my-script' );
19. }
```

スクリプトは必要なときだけエンキューするようにしましょう。 `wp_enqueue_script()` の呼び出しを適切に条件分岐でラップしましょう。

ローカライズ

スクリプトをローカライズすることにより、PHPからJSに変数を渡すことができるようになります。これは文字列の国際化(つまりローカライゼーション)によく利用されますが、他にもたくさんの使い道があります。

技術的な面では、スクリプトをローカライズするということは登録したスクリプトの直前に新しい `<script>` タグが追加されることを意味していて、ローカライズしているときに指定した名称(2番めの引き数)とともに グローバル なJavaScriptのオブジェクトを含んでいることを意味します。これはまた、別のスクリプトをあとから追加

したら依存関係にしたがってこのスクリプトを持つということであり、同じようにグローバルなオブジェクトを利用できるということでもあります。WordPressはこうしたチェーンされた依存関係もちゃんと解決します。

ではスクリプトをローカライズしてみましょう。

```
add_action( 'wp_enqueue_scripts', 'register_localize_and_enqueue_a_script' );
function register_localize_and_enqueue_a_script() {
    wp_register_script(
        'my-script',
        get_template_directory_uri().'/js/functions.js',
        array( 'jquery' ),
        filemtime( get_template_directory().'/js/functions.js' ),
        true
    );
    wp_localize_script(
        'my-script',
        'scriptData',
        // This is the data, which gets sent in localized data to the script.
        array(
            'alertText' => 'Are you sure you want to do this?',
        )
    );
    wp_enqueue_script( 'my-script' );
}
```

このJavaScriptファイルの中ではデータは

```
( function( $, plugin ) {
    alert( plugin.alertText );
} )( jQuery, scriptData || {} );
```

登録解除 / キューの解除

スクリプトは `wp_deregister_script()` と `wp_dequeue_script()` によって登録の解除とキューの解除ができます。

AJAX

WordPressでは、 `wp-admin/admin-ajax.php` にある、AJAX呼び出しのための簡単なサーバーサイドのエンドポイント提供しています。

ではサーバーサイドのAJAXハンドラーをセットアップしてみましょう。

```
// Triggered for users that are logged in.
add_action( 'wp_ajax_create_new_post', 'wp_ajax_create_new_post_handler' );
// Triggered for users that are not logged in.
add_action( 'wp_ajax_nopriv_create_new_post', 'wp_ajax_create_new_post_handler' );

function wp_ajax_create_new_post_handler() {
```

```
    // This is unfiltered, not validated and non-sanitized data.
    // Prepare everything and trust no input
    $data = $_POST['data'];

    // Do things here.
    // For example: Insert or update a post
    $post_id = wp_insert_post( array(
        'post_title' => $data['title'],
    ) );

    // If everything worked out, pass in any data required for your JS callback.
    // In this example, wp_insert_post() returned the ID of the newly created post
    // This adds an `exit`/`die` by itself, so no need to call it.
    if ( ! is_wp_error( $post_id ) ) {
        wp_send_json_success( array(
            'post_id' => $post_id,
        ) );
    }

    // If something went wrong, the last part will be bypassed and this part can execute:
    wp_send_json_error( array(
        'post_id' => $post_id,
    ) );
}


add_action( 'wp_enqueue_scripts', 'register_localize_and_enqueue_a_script' );
function register_localize_and_enqueue_a_script() {
    wp_register_script(
        'my-script',
        get_template_directory_uri().'/js/functions.js',
        array( 'jquery' ),
        filemtime( get_template_directory().'/js/functions.js' ),
        true
    );
    // Send in localized data to the script.
    wp_localize_script(
        'my-script',
        'scriptData',
        array(
            'ajax_url' => admin_url( 'admin-ajax.php' ),
        )
    );
    wp_enqueue_script( 'my-script' );
}
```

そしてJavaScriptは以下のようになります：

```
( function( $, plugin ) {
    $( document ).ready( function() {
        $.post(
            // Localized variable, see example below.
```

```
            plugin.ajax_url,
            {
                // The action name specified here triggers
                // the corresponding wp_ajax_* and wp_ajax_nopriv_* hooks server-side.
                action : 'create_new_post',
                // Wrap up any data required server-side in an object.
                data   : {
                    title : 'Hello World'
                }
            },
            function( response ) {
                // wp_send_json_success() sets the success property to true.
                if ( response.success ) {
                    // Any data that passed to wp_send_json_success() is available in the data property
                    alert( 'A post was created with an ID of ' + response.data.post_id );

                // wp_send_json_error() sets the success property to false.
                } else {
                    alert( 'There was a problem creating a new post.' );
                }
            }
        );
    } );
} )( jQuery, scriptData || {} );
```

`ajax_url` は管理画面のAJAXエンドポイントを表していて、管理画面のインターフェースページが読み込まれると自動的に定義されます。

次に、管理画面のURLを含んだスクリプトをローカライズしてみましょう：

```
add_action( 'wp_enqueue_scripts', 'register_localize_and_enqueue_a_script' );
function register_localize_and_enqueue_a_script() {
    wp_register_script( 'my-script', get_template_directory_uri() . '/js/functions.js', array( 'jquery' ) );
    // Send in localized data to the script.
    $data_for_script = array( 'ajax_url' => admin_url( 'admin-ajax.php' ) );
    wp_localize_script( 'my-script', 'scriptData', $data_for_script );
    wp_enqueue_script( 'my-script' );
}
```

WP AJAX のJavaScriptサイド

これを行うにはいくつか方法があります。もっとも一般的なのは `$.ajax()` を使う方法です。もちろん `$.post()` や `$.getJSON()` などのショートカットも利用可能です。

以下はデフォルトの例です。

```
/*globals jQuery, $, scriptData */
( function( $, plugin ) {
    "use strict";

```

```
    // Alternate solution: jQuery.ajax()
    // One can use $.post(), $.getJSON() as well
    // I prefer defered loading & promises as shown above
    $.ajax( {
         url  : plugin.ajaxurl,
         data : {
            action      : plugin.action,
            _ajax_nonce : plugin._ajax_nonce,
            // WordPress JS-global
            // Only set in admin
            postType     : typenow,
         },
         beforeSend : function( d ) {
             console.log( 'Before send', d );
         }
    } )
        .done( function( response, textStatus, jqXHR ) {
            console.log( 'AJAX done', textStatus, jqXHR, jqXHR.getAllResponseHeaders() );
        } )
        .fail( function( jqXHR, textStatus, errorThrown ) {
            console.log( 'AJAX failed', jqXHR.getAllResponseHeaders(), textStatus, errorThrown );
        } )
        .then( function( jqXHR, textStatus, errorThrown ) {
            console.log( 'AJAX after finished', jqXHR, textStatus, errorThrown );
        } );
} )( jQuery, scriptData || {} );
```

上の例ではNONCE値の検証のため `_ajax_nonce` を使ってることに注意してください。これはスクリプトをローカライズする際に自分でセットする必要があります。データ配列に `'_ajax_nonce' => wp_create_nonce( "some_value" ),` を追加するだけです。すると、PHPコールバックに次のようなリファラーチェックを追加できます: `check_ajax_referer( "some_value" )`

クリック時のAJAX

特定の要素に対するクリック時(もしくはその他のユーザーインタラクション時)にAJAXリクエストを実行するのは、実際のところとても簡単です。単に `$.ajax()` (もしくは類似のものの)呼び出しをラップするだけです。また、ディレイも追加することができます。

```
$( '#' + plugin.element_name ).on( 'keyup', function( event ) {
    $.ajax( { ... etc ... } )
        .done( function( ... ) { etc }
        .fail( function( ... ) { etc }

} )
    .delay( 500 );
```

シングルのAJAXリクエストへの複数コールバック

AJAXリクエスト完了後に複数のことをする必要があることがあります。幸いなことにjQueryではオブジェクトを

返しますので、すべてのコールバックをアタッチすることができます。

```
/*globals jQuery, $, scriptData */
( function( $, plugin ) {
    "use strict";

    // Alternate solution: jQuery.ajax()
    // One can use $.post(), $.getJSON() as well
    // I prefer defered loading & promises as shown above
    var request = $.ajax( {
         url  : plugin.ajaxurl,
         data : {
            action      : plugin.action,
            _ajax_nonce : plugin._ajax_nonce,
            // WordPress JS-global
            // Only set in admin
            postType     : typenow,
         },
         beforeSend : function( d ) {
             console.log( 'Before send', d );
         }
    } );

    request.done( function( response, textStatus, jqXHR ) {
        console.log( 'AJAX callback #1 executed' );
    } );

    request.done( function( response, textStatus, jqXHR ) {
        console.log( 'AJAX callback #2 executed' );
    } );

    request.done( function( response, textStatus, jqXHR ) {
        console.log( 'AJAX callback #3 executed' );
    } )
} )( jQuery, scriptData || {} );
```

コールバックの連鎖

よくある状況としては(どのくらい頻繁に必要とされるか、メイントラップにどのくらい簡単にひっとするかによりますが)、AJAXリクエストが完了した時のコールバックの連鎖です。

最初の問題:

> *AJAX*コールバック*(A)*が実行され
> *AJAX*コールバック*(B)*が*(A)*を待たなくてはならないことを知らない
> *(A)*の終了が早すぎてローカルでの問題が見えない

Aが終了するまでどのように待ち、Bがどのようにスタートして処理するのかは、興味深い質問です。

答えは「遅延」読み込みと「futures」としても知られる「promises(日本語の解説)」です。
以下はその例です:

```
( function( $, plugin ) {
    "use strict";

    $.when(
        $.ajax( {
            url :  pluginURl,
            data : { /* ... */ }
        } )
           .done( function( data ) {
                // 2nd call finished
           } )
           .fail( function( reason ) {
               console.info( reason );
           } );
    )
    // Again, you could leverage .done() as well. See jQuery docs.
    .then(
        // Success
        function( response ) {
            // Has been successful
            // In case of more then one request, both have to be successful
        },
        // Fail
        function( resons ) {
            // Has thrown an error
            // in case of multiple errors, it throws the first one
        },
    );
    //.then( /* and so on */ );
} )( jQuery, scriptData || {} );
```

Source: WordPress.StackExchange / Kaiser

ウィジェット

ウィジェット

A very brief description of what widgets and sidebars are

基本的なウィジェット

基本的なウィジェット

ウィジェットとは

一番簡単なウィジェット

```
1. class My_Widget extends WP_Widget {
2. 
3.     public function __construct() {
4.         parent::__construct(
5.             'my_widget', // Base ID
6.             __('My Widget', 'text_domain'), // Name
7.             array( 'description' => __( 'A my widget', 'text_domain' ), ) // Args
8.         );
9.     }
10. 
11.     public function widget( $args, $instance ) {
12.         echo 'hello world';
13.     }
14. 
15.     public function form( $instance ) { }
16. 
17.     public function update( $new_instance, $old_instance ) {
18.         return array();
19.     }
20. }
21. 
22. // make WordPress aware of this widget:
23. add_action( 'widgets_init', function(){
24.      register_widget( 'My_Widget' );
25. });
```

- `__construct`
- `widget`
- `form`
- `update`

ウィジェットフィールドの追加

バックエンドにフォームフィールドを追加し、フロントエンドのそのフィールドにアクセスする

`the_widget`

サイドバーなしでウィジェットを表示させるには

```
1. the_widget( $widget, $instance, $args );
```

これを行う必要があるとのなら、やり方を再考すべきでしょう。

JavaScript

JavaScript

スクリプトのエンキュー

管理画面でのウィジェットのフォーム

A quick note on how to do it, and a note on running the JS, so that it doesn’t get ran on the html used to create new widget forms, only those in the sidebars on the right

For the frontend

How to enqueue a widgets scripts and styles, but only if the widget is on the page

Events

Running code when:

A new widget is added

The widget form opens

Widgets are re-ordered

I18n

I18n

- 翻訳ファイルの作成
- 翻訳ファイルの読み込み
- コアの翻訳
- 投稿タイプのスラッグ
- 管理画面の言語の設定
- 外部Twitterの埋め込み

マルチサイト

マルチサイト

Networkネットワーク内の他のブログからデータを取得する

可能だが、高くつく

```
switch_to_blog( $blog_id );
// Do something
restore_current_blog();
```

`restore_current_blog` は `switch_to_blog` への最後の呼び出しを取り消しますが、これは1ステップのみによってなので、 `switch_to_blog` を再度呼びだす前に `restore_current_blog` をつねに呼び出します。

ネットワーク内のブログを一覧表示する

可能だが、大きなネットワークではスケールせず、高くつく

ドメインマッピング

ドメインマッピングのプラグインのメモ

テスト

Testing

- WP Test & Theme Test Data
- Theme review tester plugin
- Integration vs unit vs behavioural testing

テストタイプ

テストタイプ

ユニットテスト

ユニットテスト

統合テスト

統合テスト

エンドツーエンドテスト

エンドツーエンドテスト

挙動テスト

挙動テスト

テスト駆動開発

テスト駆動開発

WP_UnitTestCase

WP_UnitTestCase

ユニットテストの例

ユニットテストの例

サーバーとデプロイ

Servers And Deployment

- WP CLI
- Composer
- Search Replace

セキュリティ

セキュリティ

ソルト

Nonces

`wp-config`の場所

テーブルプリフィックス

User ID 1

権限

設定ページを取り除くか隠すか

カスタムなパスワードリセットコード

timthumb.php

SSL

管理画面のみのSSL

迷信

管理画面とログインのURLを隠すことについて

無効化されたプラグインとテーマ

攻撃からの復旧

リソース

Resources

コミュニティ

コミュニティ

WordCamp

WordCampは1日もしくは2日間のWordPressのすべてに関するカンファレンスです。一般的な項目(ブログ、コンテンツの作り方、ウェブサイトのマーケティングやWordPressを取りますビジネス)と技術的な項目(プラグインやテーマ開発者向けのもの)の両方を扱うよう企画されます。

現在では世界40カ国以上で毎年50ものWordCampが開催されていて、コミュニティのことを知るのにはとてもいい機会になっています。WordCampの多くのセッションがWordPress.tvで公開されていて、講演したり、ボランティアをしたり、スポンサーになったりと、WordPressのコミュニティに参加協力するとてもいい機会にもなっています。

WordCampの開催予定はすべてWordCamp Centralで参照することができます。

コントリビューター・デイ

Contributor days, usually held after WordCamps (but can be independant), are events that are set up to help you contribute to WordPress. The benefits to contributing to WordPress are numerous, both for a business looking to get more exposure in and amongst the WordPress community, to a lone developer looking to grow their skills working in a team on a massive project.

You do not need to be code proficient to contribute to WordPress. They are looking for a wide range of skills, like support, theme review as well as accessibility.

Look at your local WordCamp if they are holding a Contributor day. Alternatively, if you're based in the United Kingdom, you can find the latest WordPress Contributor day at http://www.wpcontributorday.com/.

- Local User Groups
- .org Support Forums
- IRC Channels
- WordPress Stack Exchange

クレジット

Credits

Here are some of the people who contributed to this book, names are linked to their Github accounts, or failing that, twitter

- Tom J Nowell

葡萄牙语版

Introdução

WordPress Do Jeito Certo

Este livro possui fontes sólidas, das melhores práticas já feita pela comunidade WordPress para seus desenvolvedores, com a intenção de ajudar a depurar os erros mais comuns e dolorosos feito por nós meros mortais.

Este documento continuará sendo atualizado, com mais informações e exemplos úteis sempre que forem disponíveis.

Como contribuir?

Você pode contribuir em nosso repositório no Github. As mudanças serão feitas no Gitbook.io automaticamente quando o repositório principal mudar.

Você pode atualizar este livro, tanto pelo editor de arquivos markdown quanto abrindo o repositório na aplicação para desktop do Gitbook. O aplicativo para desktop, dará a você opções de `live preview` .

Licença

Por onde começar?

Por onde começar?

PHP Básico

Estamos partindo da ídeia de que você já possui um conhecimento básico de PHP. Incluindo alguns tópicos básicos como:

- Functions
- Arrays
- Variáveis
- Loops e Condições
- Classes e Objetos
- Herança de Classes
- Polimorfismo
- POST and GET
- Escopo de variáveis

Tenha certeza de que você possui conhecimentos sólidos nos tópicos abordados acima antes de continuar, também estamos assumindo que você possui um editor de texto que suporte o PHP, recomenda-se que seu editor possua as seguintes “features”.

- Idêntação automática
- Preenchimento automático
- Fechamento automático de parênteses
- Verificador de sintaxe

Ambientes de Desenvolvimento Local (Localhost)

É importante possuir um `localhost` . A época em que tinhamos de fazer `upload` da aplicação em produção, esperando que o melhor aconteça já é passado, hoje felizmente possuimos algo chamado `Local Web Server` ou Ambiente de Desenvolvimento Local.

Com ambientes locais você pode trabalhar mais rapidamente, sem ficar fazendo `download e upload` de arquivos, ficando refém de uma internet problemática, ou esperar por todas as páginas carregarem. Com um `localhost` , você pode trabalhar de um trêm, ou de um túnel, sem estar conectado a `Wifi ou Plano de Dados` , e testar a aplicação antes de fazer `deploy` em produção.

Aqui estão algumas opções para configurar seu `localhost` . As duas opções mais populares são as/os:

- VMs (Máquinas Virtuais)

- LAMPs (Linux, Apache, MySQL e PHP)

O primeiro tipo de ambiente, geralmente envolve terceiros, tais como Vagrant, que oferece uma “máquina virtual” pré-configurada e consistente para trabalhar.

O segundo tipo instala um programa de servidor diretamente no sistema operacional. Existem várias ferramentas para tornar isto mais fácil, tais como `WAMP, MAMP, XAMPP e ou Vertrigo`. Mas seu ambiente será único e mais difícil de depurar. Estes ambientes são chamados de `LAMP`, que traz consigo `Linux, Apache, MySQL e PHP`.

IIS

Microsoft Internet Information Services é um programa de servidor que roda em servidores Windows. Variantes desse sistema vem com o Windows se você instalar os componentes apropriadamente, mas o conhecimento em desenvolvimento IIS com WordPress na comunidade é raro. A maioria dos servidores hoje, rodam com Apache ou Nginx, e o conhecimento dos desenvolvedores tendem a ser direcionados para tais ferramentas. A curva de aprendizado com IIS é maior, e talvez não traga grandes resultados.

Controladores de Versão

Uma parte vital do trabalho em grupo, organização de tarefas e projetos “Open Source”, é o uso de um Controlador de Versão. Ele é sistema que monitora as mudanças do seu código em tempo real, e permite, aos desenvolvedores (de sua equipe ou comunidade) a colaborarem e aprimorarem sua aplicação.

Git

Criado por Linus Torvalds o criador do Linux, Git é um dos controladores de versão mais populares hoje em dia.

Subversion

Também conhecido como SVN é um outro controlador de versão muito popular, ele é usado nos plugins e repositórios do WordPress.

Guia de estilo de código

Guia de estilo de código

É importante manter o código legível e de fácil manutenção. Isso previne que pequenos erros, tornem-se críticos a ponto de você perder horas de trabalho, para perceber que esqueceu de adicionar um “;” na Linha X. É recomendável usar o mesmo “Code Standard” ao decorrer de toda a sua aplicação. Mas se você se sente mais confortável, usando o padrão PSR, não há problema algum nisso, com tanto que a use de maneira consistente.

Indentação

Indentação no WordPress é feito usando “tabs”, representado por 4 espaços. Indentação é muito importante para um código mais legível, e cada declaração deve estar em sua própria linha. Sem a indentação, fica mais díficil de entender o código, e os erros tornam-se mais frequentes.

Dê uma olhada em Editor Config. Ele é uma ótima maneira de assegurar de que todos os membros do time, estão seguindo UM mesmo padrão.

O seguinte arquivo `.editorconfig` reforça as regras acima, com indentação de largura de 4 espaços.

```
[*.php]
indent_style = tabs
indent_size = 4
```

Tag Spam do PHP

As tags `<?php` and `?>` devem ser usadas separadamente. Por exemplo:

```
<?php // bad ?>
<?php while( have_posts() ) { ?>
    <?php the_post(); ?>
    <?php the_title(); ?>
    <strong><?php the_date(); ?></strong>
    <?php the_content(); ?>
<?php } ?>
```

Seria mais fácil de ler:

```
<?php
// good
while( have_posts() ) {
    the_post();
```

```
    the_title();
    ?>
    <strong><?php the_date(); ?></strong>
    <?php
    the_content();
} ?>
```

Linting

Muitos editores suportam ou já possuem um verificador de sintaxes. Esses verificadores são chamados de Linters, quando usado com um bom editor, erros de sintaxe são destacados e depurados mais facilmente. Por exemplo, no PHPStorm, é dado a um erro de sintaxe um destaque vermelho.

Padrões do WordPress

WordPress segue um conjunto de padrões. Esses padrões são diferentes do padrão PSR. Por exemplo, WordPress usa tabs ao invés de espaços, e abre os parentêses em uma nova linha.

O Manual do WordPress para contribuidores, aborda os padrões de código, mais detalhadamente. Abaixo estão algumas referências para estudo.

- HTML Coding Standards
- PHP Coding Standards
- JavaScript Coding Standards
- CSS Coding Standards

PHP Code Sniffer & PHP CS Fixer

O PHP Code Sniffer é uma ferramente responsável por verificar violações no código. Atualmente muitos editores, trazem com eles essa função. E o PHP CS Fixer, corrigi automaticamente essas violações para você.

Para usar essas ferramentas, você precisará da definição dos `Padrões de Código` do WordPress. Você pode encontra-los aqui, junto com as instruções do PHPStorm

Debugging

Logs

Tratamento de Erros

Ferramentas

wp-config.php

Core

Core

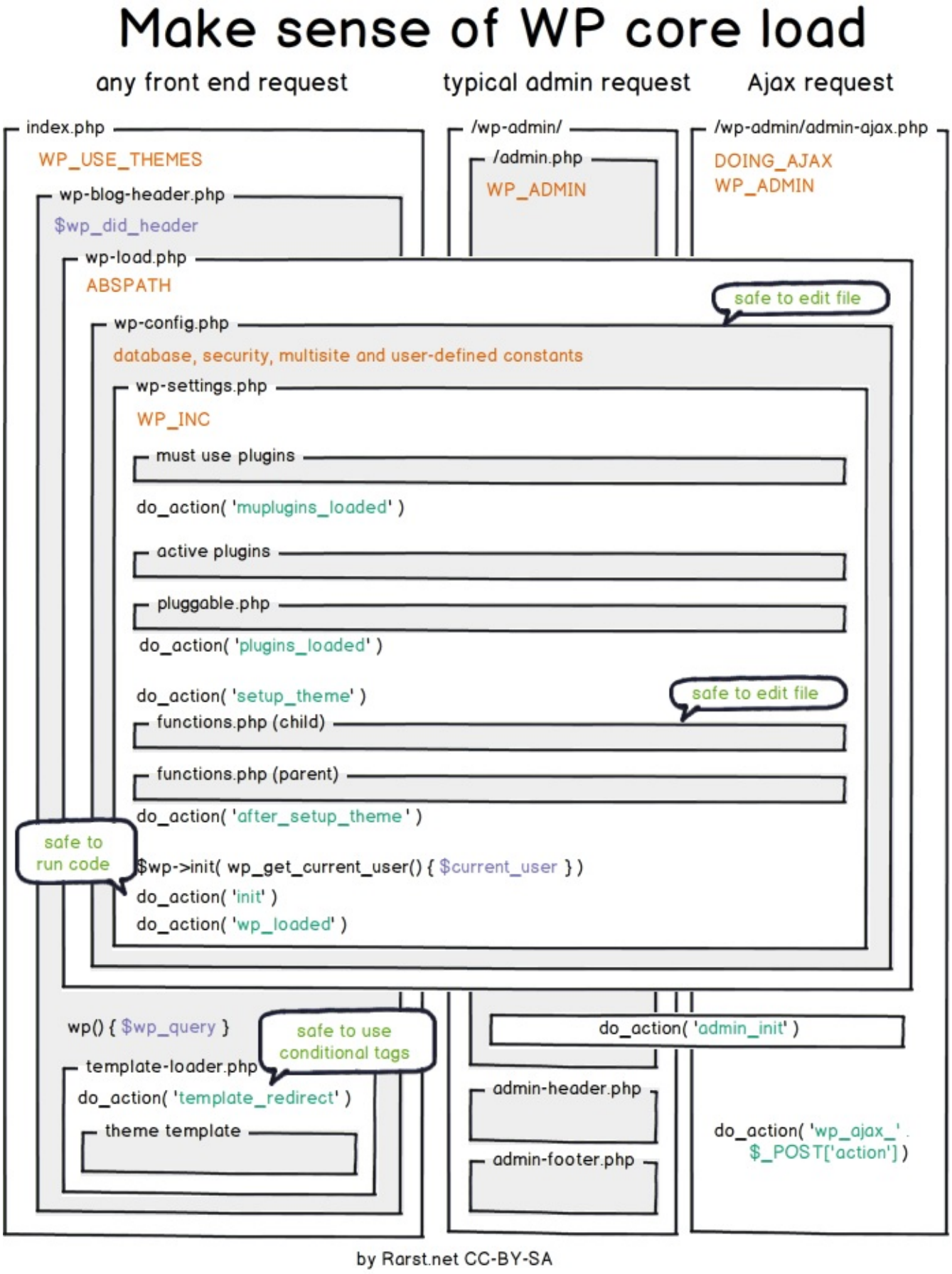
O `Core` do WordPress é o código que faz com ele funcione. São os primeiros arquivos que você tem logo após fazer o `download` do site oficial, exceto os temas e os plugins.

Carregamento de Processos

De maneira simples, o `core` do WordPress segue o seguinte padrão de processos:

- MU Plugins ou Must-Use Plugins
- Plugins Ativados
- Carrega as funções do Tema (functions.php)
- Roda o hook inicial
- Carrega a main query
- Carrega o template

Administração e requisições AJAX segue um processo similar porém mais leve. O diagrama abaixo aborda este processo com mais detalhes:



jQuery

Muitos desenvolvedores de plugins e temas preferem remover o jQuery que vem por padrão

com o WordPress, e acabam por adicionar sua própria versão do mesmo, normalmente é o jQuery do Google CDN. Não faça isso! Caso contrário poderá causar problemas de compatibilidade. Use a versão do jQuery que vem junto com o WordPress. Isso garante maior compatibilidade dentre os plugins.

Modificando o Core

É tentador modificar partes do `Core`, removendo ou adicionando funcionalidades, mas isso NUNCA deve ser feito. Quando você realizar a atualização do WordPress, todas as suas modificações serão perdidas. Ao invés de mudar o `Core`, use os famosos `Hooks, Actions e Filters` para modificar as funcionalidades da sua aplicação.

Mais Informações

- Entendo o carregamento do Core

Dados

Queries

Post Queries

Taxonomias e Term Queries

Comentários Queries

User Queries

SQL

Routing

The Main Loop & Template Loading

Where Query Variables Come From

Rewrite Rules

Clashes, Slugs, & Debugging

Templates

JavaScript

JavaScript

O WordPress possui um gerenciador de dependência, que lhe permite controlar os *imports* do JavaScript. Não use a tag `<script>` para fazer *embeds* de scripts, ao invés disso, crie um arquivo separado e o importe.

Registrando e Importando

Os *scripts* devem ser registrados, isso facilita o trabalho do gerenciador de dependência, pois quando você registra o arquivo, você declara a existência do *script* para o *WordPress*.

Vamos registrar e importar um script.

```
1. // Usa a função wp_enqueue_scripts para registrar e importar os scripts no Front-End.
2. add_action( 'wp_enqueue_scripts', 'register_and_enqueue_a_script' );
3. function register_and_enqueue_a_script() {
4.    // Associa uma ID chamada `my-script` para este script em específico.
5.    //  + este arquivo está localizado na raiz do tema,
6.    //  + que possui como dependência o o jQuery,
7.    //  + Um *timestamp* é adicionado toda vez que o arquivo é modificado, isso previne o cache do arquivo durante o desenvolvimento,
8.    //  + ele é posicionado no final da página (footer)
9.    wp_register_script(
10.      'my-script',
11.      get_template_directory_uri().'/js/functions.js',
12.      array( 'jquery' ),
13.      filemtime( get_template_directory().'/js/functions.js',
14.      true
15.   );
16.   // Importamos o script.
17.   wp_enqueue_script( 'my-script' );
18. }
```

Pro-Tips:

- Os scripts devem ser importados apenas quando necessário; Envolvendo a função `wp_enqueue_script()` usando condicionais para controle do mesmo.
- Quando você for importar seus *scripts* no Painel do Admin, use o *hook* `admin_enqueue_scripts`.
- Se for adicionar scripts para a tela de login, use o *hook* `login_enqueue_scripts`.

Localização

A *localização* é um script que lhe permite passar variáveis(dados) do PHP para JavaScript. Isso normalmente é usado para internacionalização de *strings* (tradução), mas existem muitas outras meneiras de usarmos esta técnica.

De um ponto de vista técnico, localizar um script significa que haverá uma nova tag `<script>` adicionada logo antes dos scripts registrados, que contém o objeto `_global_` JavaScript com o nome que você especificou durante a localização no segundo argumento. Isso significa que, se você adicionar um outro *script* que possui esse mesmo *script* como dependência, você poderá usar o mesmo objeto `_global_` no novo *script* sem problemas. O WordPress trabalha muito bem com encadeamento de dependências.

Vamos *localizar* um script.

```
1. add_action( 'wp_enqueue_scripts', 'register_localize_and_enqueue_a_script' );
2. function register_localize_and_enqueue_a_script() {
3.    wp_register_script(
4.       'my-script',
5.       get_template_directory_uri().'/js/functions.js',
6.       array( 'jquery' ),
7.       filemtime( get_template_directory().'/js/functions.js' ),
8.       true
9.    );
10.   wp_localize_script(
11.      'my-script',
12.      'scriptData',
13.      // Estes são os dados, que irão ser mandados para o arquivo JavaScript.
14.      array(
15.         'alertText' => 'Are you sure you want to do this?',
16.      )
17.   );
18.   wp_enqueue_script( 'my-script' );
19. }
```

No arquivo de javascript, os dados estão disponíveis através do objeto especificado durante o processo de *localização*.

```
1. ( function( $, plugin ) {
2.    alert( plugin.alertText );
3. } )( jQuery, scriptData || {} );
```

Remover os importes e registros padrão do WordPress

Você pode remover um registro ou um importe através das funções `wp_deregister_script()` e `wp_dequeue_script()` .

AJAX

WordPress oferece *endpoints* no servidor para requisições **AJAX**, localizado em `wp-admin/admin-ajax.php` . Vamos configurar um *endpoint* no servidor para manipulação AJAX.

```
1.  // É acionado quando o usuário está logado no Painel.
2.  add_action( array( 'wp_ajax_nopriv_create_new_post', 'wp_ajax_create_new_post' ),
    'wp_ajax_create_new_post_handler' );
3.  function wp_ajax_create_new_post_handler() {
4.     // Este é um dado que não foi filtrado, validado e tratado.
5.     $data = $_POST['data'];
6.
7.     // Faz coisas aqui.
8.     // Por exemplo: Insere ou atualiza um post.
9.     $post_id = wp_insert_post( array(
10.       'post_title' => $data['title'],
11.    ) );
12.
13.    // Se tudo der certo, insira qualquer tipo de dado para o callback do JavaScript.
14.    // Neste exemplo, wp_insert_post() retorna a ID de último post criado.
15.    // Isso adiciona um `exit`/`die` por si só, então não há necessidade de ser chamada.
16.    if ( ! is_wp_error( $post_id ) ) {
17.       wp_send_json_success( array(
18.          'post_id' => $post_id,
19.       ) );
20.    }
21.
22.    // Se alguma coisa der errado, a última parte será passada e só então executada:
23.    wp_send_json_error( array(
24.       'post_id' => $post_id,
25.    ) );
26. }
27.
28.
29. add_action( 'wp_enqueue_scripts', 'register_localize_and_enqueue_a_script' );
30. function register_localize_and_enqueue_a_script() {
31.     wp_register_script(
32.       'my-script',
33.       get_template_directory_uri().'/js/functions.js',
34.       array( 'jquery' ),
35.       filemtime( get_template_directory().'/js/functions.js' ),
36.       true
37.     );
38.
39.     // Manda os dados do PHP através de variáveis para o JavaScript.
40.     wp_localize_script(
41.       'my-script',
42.       'scriptData',
43.       array(
44.          'ajax_url' => admin_url( 'admin-ajax.php' ),
45.       )
46.     );
```

```
47.
48.     wp_enqueue_script( 'my-script' );
49. }
```

Em seguida vem o JavaScript:

```
1. ( function( $, plugin ) {
2.    $( document ).ready( function() {
3.       $.post(
4.
5.          plugin.ajax_url,
6.          {
7.             // Os nomes especificados são os `triggers` correspondentes aos hookers
8.             // wp_ajax_* e wp_ajax_nopriv_* do lado do servidor.
9.             action : 'create_new_post',
10.
11.            // Encapsula todos os dados do lado do servidor em um objeto.
12.            data : {
13.               title : 'Hello World'
14.            }
15.         },
16.
17.         function( response ) {
18.            // wp_send_json_success() define a propriedade "sucsess" para verdadeiro
19.            if ( response.success ) {
20.               // Qualquer dado que passou pela função wp_send_json_sucesess(), está disponível na propriedade
   "data".
21.               alert( 'A post was created with an ID of ' + response.data.post_id );
22.
23.            // wp_send_json_error() define a propriedade "sucsess" para falso.
24.            } else {
25.               alert( 'There was a problem creating a new post.' );
26.            }
27.         }
28.
29.      );
30.   } );
31. } )( jQuery, scriptData || {} );
```

`ajax_url` representa o *endpoint* do admin via AJAX, que é automaticamente definida na página do Admin, mas não no front-end.

Vamos localizar nosso script para incluir a URL do admin:

```
1. add_action( 'wp_enqueue_scripts', 'register_localize_and_enqueue_a_script' );
2. function register_localize_and_enqueue_a_script() {
3.    wp_register_script( 'my-script', get_template_directory_uri() . '/js/functions.js', array( 'jquery' ) );
4.    // Manda os dados do PHP através de variáveis para o JavaScript.
5.    $data_for_script = array( 'ajax_url' => admin_url( 'admin-ajax.php' ) );
6.    wp_localize_script( 'my-script', 'scriptData', $data_for_script );
7.    wp_enqueue_script( 'my-script' );
8. }
```

JavaScript com WP AJAX

Existe várias maneiras de fazermos isto. A maneira mais comum é usar `$.ajax()` . É claro que existem atalhos disponíveis como `$.post()` e `$.getJSON()` .

Aqui esta um exemplo padrão.

```
/*globals jQuery, $, scriptData */
( function( $, plugin ) {
    "use strict";

    // Variação do jQuery.ajax()
    // Você também pode usar o $.post(), $.getJSON() se quiser
    // Eu prefiro declarar explicitamente o carregamento & as promessas descritas abaixo
    $.ajax( {
        url  : plugin.ajaxurl,
        data : {
          action      : plugin.action,
          _ajax_nonce : plugin._ajax_nonce,
          // Objeto-Global do WordPress
          // É mostrado apenas no "admin"
          postType     : typenow,
        },
        beforeSend : function( d ) {
          console.log( 'Before send', d );
        }
    } )
       .done( function( response, textStatus, jqXHR ) {
          console.log( 'AJAX done', textStatus, jqXHR, jqXHR.getAllResponseHeaders() );
       } )
       .fail( function( jqXHR, textStatus, errorThrown ) {
          console.log( 'AJAX failed', jqXHR.getAllResponseHeaders(), textStatus, errorThrown );
       } )
       .then( function( jqXHR, textStatus, errorThrown ) {
          console.log( 'AJAX after finished', jqXHR, textStatus, errorThrown );
       } );
} )( jQuery, scriptData || {} );
```

Perceba que o exemplo acima usa `_ajax_nonce` para verificar o valor NONCE, o qual você terá de definir sozinho, quando for *localizar* o script. É só adicionar `'_ajax_nonce' => wp_create_nonce( "some_value" ),` no objeto *array*. Você então poderá adicionar um marcador nos seus *callbacks* do PHP, que se parecem um pouco com isso `check_ajax_referer( "some_value" )` .

Trabalhando com AJAX e Eventos

Na verdade é muito simples executar uma requisição AJAX por cliques (ou qualquer outro tipo de interação do usuário) em algum elemento. Apenas envolva a sua chamada

`$.ajax()` (ou algum de seu similares). Você pode ainda, adicionar um delay na requisição.

```
$( '#' + plugin.element_name ).on( 'keyup', function( event ) {
   $.ajax( { ... etc ... } )
      .done( function( ... ) { etc }
      .fail( function( ... ) { etc }

} )
   .delay( 500 );
```

Multiplos callbacks para uma única requisição AJAX

Você pode acabar caindo em uma situação da qual multiplas tarefas acontecem depois de que uma requisição AJAX é finalizada. Felizmente jQuery retorna um objeto, onde você pode anexar todos os seus callbacks.

```
/*globals jQuery, $, scriptData */
( function( $, plugin ) {
   "use strict";

   // Variação do jQuery.ajax()
   // Você também pode usar o $.post(), $.getJSON() se quiser
   // Eu prefiro declarar explicitamente o carregamento & as promessas
   var request = $.ajax( {
       url  : plugin.ajaxurl,
       data : {
         action      : plugin.action,
         _ajax_nonce : plugin._ajax_nonce,
         // Objeto-Global do WordPress
         // É mostrado apenas no "admin"
         postType     : typenow,
       },
       beforeSend : function( d ) {
         console.log( 'Before send', d );
       }
   } );

   request.done( function( response, textStatus, jqXHR ) {
      console.log( 'AJAX callback #1 executed' );
   } );

   request.done( function( response, textStatus, jqXHR ) {
      console.log( 'AJAX callback #2 executed' );
   } );

   request.done( function( response, textStatus, jqXHR ) {
      console.log( 'AJAX callback #3 executed' );
   } )
```

```
} )( jQuery, scriptData || {} );
```

Encadeamento de Callbacks

Um cenário comum (do qual muitas vezes um *callback* é necessário e como ele é simples de ser utilizado), é o encadeamento de callbacks quando uma requisição AJAX é finalizada.

Vamos dar uma olhada no problema primeiro:

> *O callback (A) é executado.*
> *O callback (B) não sabe que deve esperar por (A).*
> *Você não pode ver o problema na sua instalação local se (A) terminar rapidamente.*

A grande questão é, quando nós devemos esperar (A) finalizar, para só então, inicializarmos o processo de (B).

A resposta é que o processo é “adiado” carregando suas “promessas”, também conhecido como “futuros”.

Veja um exemplo:

```
( function( $, plugin ) {
    "use strict";

    $.when(
        $.ajax( {
            url :  pluginURl,
            data : { /* ... */ }
        } )
           .done( function( data ) {
                // 2nd call finished
           } )
           .fail( function( reason ) {
               console.info( reason );
           } );
    )
    // Novamente, você poderia utliziar o método .done() se quisesse. Veja a documentação do jQuery.
    .then(
        // Sucesso
        function( response ) {
            // Has been successful
            // In case of more then one request, both have to be successful
            // Sucesso! Em caso de mais de uma requisição, ambos deverão ser bem sucedidos.
        },
        // Falhou
        function( resons ) {
            // É jogado um erro, em caso de multiplos erros, é o primeiro que é cuspido.
        },
    );
    //.then( /* and so on */ );
```

```
30. } )( jQuery, scriptData || {} );
```

Fonte: WordPress.StackExchange / Kaiser

Widgets

I18n

I18n

Quando nós falamos sobre I18n, nós estamos falando sobre traduções de `strings` em interfaces de usuários e ou no frontend. Para apresentarmos conteúdo em multiplos idiomas ou para gerar um sistema, que seleciona o idioma com base no país do usuário, você precisará instalar um `plugin` que possui ferramentas de edição linguística para `posts` e ou qualquer outro tipo de conteúdo como: Páginas, Categorias, etc.

A qualquer momento, você pode alterar manualmente o idioma do WordPress, mudando a constante `WP_LANG` no `wp-config.php`, e.g:

```
1. define ('WPLANG', 'zh_CN');
```

No entanto, você precisa ter certeza de que o arquivo do idioma foi colocado corretamente no diretório `wp-content/languages` antes de fazer tal alteração.

As traduções nunca funcionam, dentre os grupos de poliglotas durante os “Dias de Contribuidor”. Se você esta interessado em traduzir o `Core` do WordPress, você deveria dar uma olhada no HandBook oficial para tradutores e descobrir como ajudar.

Configurando o Idioma do Admin

No processo de instalação do WordPress, um dos passos é a seleção do idioma, porém você talvez queira definir idiomas diferentes para o frontend e outro para o backend. Por exemplo um site alemão que é administrado por um falante da lingua inglesa, talvez queira a area do admin em sua lingua nativa.

Para fazer isso, defina o idioma para Alemão na constante `WP-LANG` mencionada anteriormente, e adicione este código para deixar a area do admin em ingles:

```
1. add_filter('locale', 'wpse27056_setLocale');
2. function wpse27056_setLocale($locale) {
3.     if ( is_admin() ) {
4.         return 'en_US';
5.     }
6. 
7.     return $locale;
8. }
```

Embeds do Twitter para Estrangeiros

As vezes os `embeds` aparecem inesperadamente em uma lingua estrangeira, isso porque tal serviço que esta hospedado em seu servidor, esta realizando uma `request` do servidor de seu país de origem. Por exemplo, um site em ingles hospedado em um servidor Alemão, resultará em um embed do Twitter em alemão.

Mais detalhes

- Idiomas diferentes para frontend e backend

Multisite

Testes

Testes Unitários

Teste Funcionais/Comportamentais

Test Driven Development(TDD)

WP_UnitTestCase

Servidores e Deploy da Aplicação

WP CLI

Composer

Migrations

Segurança

Comunidade

Comunidade

WordCamps

Os WordCamps são curtos, em média de 1-2 dias de conferência totalmente focados em WordPress. Os encontros dão atenção tanto para (blog, formatação de conteúdo, marketing de websites e empreendedorismo envolvendo WordPress) quanto as partes técnicas, de como desenvolver temas e plugins.

Existem hoje mais de 50 WordCamps acontecendo em mais de 40 países por todo o mundo, e através deles você poderá explorar/conhecer a comunidade e as novas “Trends” em WordPress. Muitas das sessões do WordCamp são transmitidas através do WordPress.tv, e é uma ótima oportunidade para ajudar a Comunidade WordPress, tanto ao falar, ao voluntariado ou patrocinío.

Você pode dar uma olhada nas próximas conferências através do WordCamp Central.

Dias do Contribuidor

Os “Dias de Contribuidor”, acontecem normalmente após os WordCamps (mas eles podem ser independentes), são eventos criados para ajudar você a contribuir mais para o WordPress. São vários os benefícios de contribuir com o WordPress, tanto para o lado empreendedor, expondo seu nome pela comunidade, quanto para desenvolvedores sozinhos, buscando melhorar suas habilidades de trabalho em equipe e massificar seus projetos.

Você não precisa ser um desenvolvedor profissional para contribuir para o WordPress. Eles procuram por uma grande variedade de habilidades, como suporte, review de temas e acessibilidade web.

De uma olhada em um WordCamp local, e veja se eles estão realizando o “Dia do Contribuidor”. Em caso de você viver no Reino Unido, você pode acompanhar os “Dias do Contribuidor” mais recentes através do http://www.wpcontributorday.com/.

- Grupos de Usuários Locais
- .org Foruns de Suporte
- Canais IRC
- WordPress Stack Exchange
- WordPress Slack

Créditos

Créditos

Muitas pessoas tem contrubuido para o WordPress The Right Way através do Github. Você pode ver a lista completa de contribuidores aqui, e você pode `Forkar` e enviar seus próprios `Pull Requests` e juntar-se a eles!