### index.php

```php <?php /\**\* 这个文件只用来加载 '/wp-blog-header.php' \* \* @package WordPress*/

/\**\* 声明一个全局变量，用来判断是否加载主题 \* \* @var bool*/ define('WP\_USE\_THEMES', true);

/\* *加载WordPress环境和模板* / require **DIR** . '/wp-blog-header.php'; ```

### wp-blog-header.php

入口文件很明显只定义了WP\_USE\_THEMES常量，表示是否启用主题功能，不启用的话则显示空站点，require\* \_\_DIR\_\_ . '/wp-blog-header.php'; 表示引入主题的文件，所以打开wp-blog-header.php

``` php <?php /\**\* 加载WordPress环境和模板 \* \* @package WordPress*/

if (!isset($wp\_did\_header)) { // 表示我们第一次进入此文件时要设置的值以及引入的PHP加载文件 $wp\_did\_header = true;

// 加载WordPress库  
require\_once \_\_DIR\_\_ . '/wp-load.php';  
  
// 初始化WordPress请求  
wp();  
  
// 加载主题模板  
require\_once ABSPATH . WPINC . '/template-loader.php';

} ```

初始时没有wp()函数，wp()函数一定在wp-load.php中被声明或者通过其他PHP文件引入，所以以下查看wp-load.php中执行的代码（定义或者声明的代码用不到的先不看）

### wp-load.php

``` php <?php /\**\* 设置ABSPATH常量的驱动文件 \* 加载 wp-config.php 文件. \* wp-config.php会加载wp-settings.php file, \* wp-settings.php会加载wp-config.php中定义的常量 \* \* 如果找不到wp-config.php文件，会尝试抛出error \* \* \* 会尝试在WordPress的父目录中寻找wp-config.php文件 \* 以便保持WordPress目录不变 \* \* @package WordPress*/

/\* *定义ABSPATH 为本文件的文件夹目录* / if (!defined('ABSPATH')) { define('ABSPATH', **DIR** . '/'); }

/*\* error\_reporting() 函数可以在php.ini中取消.*/ if (function\_exists('error\_reporting')) { /*\* error\_reporting() 函数用于设置错误报告的级别。 \* 不同的错误报告级别可以用来报告不同类型的错误 \* \* This will be adapted in wp\_debug\_mode() located in wp-includes/load.php based on WP\_DEBUG. \* @see https://www.php.net/manual/en/errorfunc.constants.php List of known error levels.*/ error\_reporting(E\_CORE\_ERROR | E\_CORE\_WARNING | E\_COMPILE\_ERROR | E\_ERROR | E\_WARNING | E\_PARSE | E\_USER\_ERROR | E\_USER\_WARNING | E\_RECOVERABLE\_ERROR); }

/*\* 首先，代码检查wp-config.php文件是否存在。 \* 如果存在，则直接包含该文件。 \* 如果wp-config.php文件不存在，代码会检查父目录中是否存在wp-config.php文件，并且父目录中不存在wp-settings.php文件。 \* 如果满足这些条件，则包含父目录中的wp-config.php文件。 \* 如果以上两个条件都不满足，代码会尝试自动创建wp-config.php文件。 \* 首先，它会检查PHP版本和MySQL扩展或数据库驱动程序是否符合要求。 \* 然后，它会标准化$\_SERVER变量，设置WP\_CONTENT\_DIR常量，包含functions.php文件，并尝试猜测网站URL。 \* 最后，它会跳转到setup-config.php页面，提示用户手动创建wp-config.php文件。 \* 如果wp-config.php文件仍然不存在，代码会输出错误信息，并提示用户通过Web界面创建wp-config.php文件，或者手动创建该文件。*/ if (file\_exists(ABSPATH . 'wp-config.php')) { /\* *Config配置文件* / require\_once ABSPATH . 'wp-config.php'; } ```

在这里我们先查看存在Config配置文件的情况，因为配置好站点后会进入这个选项，下面查看wp-config.php

这段代码是WordPress的配置文件wp-config.php的基础模板。它包含了数据库设置、密钥、数据库表前缀和ABSPATH等配置。

1. 数据库设置（DB\_NAME、DB\_USER、DB\_PASSWORD、DB\_HOST、DB\_CHARSET和DB\_COLLATE）: 这些设置用于连接到数据库。DB\_NAME是数据库名称，DB\_USER是数据库用户名，DB\_PASSWORD是数据库密码，DB\_HOST是数据库主机名，DB\_CHARSET是数据库字符集，DB\_COLLATE是数据库排序规则。
2. 密钥（AUTH\_KEY、SECURE\_AUTH\_KEY、LOGGED\_IN\_KEY、NONCE\_KEY、AUTH\_SALT、SECURE\_AUTH\_SALT、LOGGED\_IN\_SALT和NONCE\_SALT）: 这些密钥用于加密和安全性。
3. 数据库表前缀（$table\_prefix）: 不同的表前缀来避免表名冲突。
4. WP\_DEBUG: 这个设置用于开发环境中的调试。如果设置为true，将在开发过程中显示通知。
5. ABSPATH: 这个常量指向WordPress的根目录。
6. wp-settings.php: 这个文件包含了WordPress的核心设置。

### wp-config.php

```php <?php

// \*\* 数据库参数定义 \*\* // /\* *The name of the database for WordPress* / define('DB\_NAME', 'wordpress');

/\* *Database username* / define('DB\_USER', 'root');

/\* *Database password* / define('DB\_PASSWORD', '123456');

/\* *Database hostname* / define('DB\_HOST', 'localhost');

/\* *Database charset to use in creating database tables.* / define('DB\_CHARSET', 'utf8mb4');

/\* *The database collate type. Don't change this if in doubt.* / define('DB\_COLLATE', '');

define('WP\_ENV', 'development');

define('AUTH\_KEY', '#Wb>fA2tY/^boB:/G:}bx|;kvG-([4x~Kk(dspC6rcb#{$C3?7a&|dic,PSFv!O'); define('SECURE\_AUTH\_KEY', '??,Y\_RrLb&IG<$ju^IULT>UVlvWtf:il)z9ic &\_kzI8irciizM%#Wh[r%xZ8%'); define('LOGGED\_IN\_KEY', 'W=l6OiJbc^Bg6HYU.aWr{.zlD)7$:kTjj |XuMJiOEZlfE1}9zm7Bx\*tw leJ\*N?'); define('NONCE\_KEY', 'aO}9eT4ydg/ME 9@u8I-wFT)bZC|L,|zdo?Z?b;F5:{:61KM)bv.?4mXk!w0$Vv'); define('AUTH\_SALT', '>t$*@5xK4}/m~<bzvuQb(7Va&xo#:71^aiAc|FwrdYk75vu:4?DT}Ule\_4{ve'); define('SECURE\_AUTH\_SALT', 'Y|5+lLR.b2fsPFenR{fCL5!B>{AV4Ta,%6RQz>5s (4jZ-(;$RG/dY)lHH6Db(U'); define('LOGGED\_IN\_SALT', 'a<?%~P%SLQrU5sZA-#-F~b?Xc( Cb 3!O$|*>uj:Ys}[LN>5<wT.dmRGE3].,nG'); define('NONCE\_SALT', 'tC(6Z|&m#T%^1p3w^()DSQ]hO8wOo,T.T*$%p=&c[})JhS&W!d*!c601R)Kc/(c');

$table\_prefix = 'wp\_';

define('WP\_DEBUG', true);

/\* *重新定义ABSPATH常量* / if (!defined('ABSPATH')) { define('ABSPATH', **DIR** . '/'); }

/\* *引入wp-settings.php* / require\_once ABSPATH . 'wp-settings.php';

```

接着我们查看wp-settings.php文件

这个文件引入的内容比较多，但是大部分思想都是一样的

### wp-settings.php

```php <?php /\**\* WP库文件夹*/ define( 'WPINC', 'wp-includes' );

/**\* global 用于提前声明全局变量，之后赋值 \*/ global $wp\_version, $wp\_db\_version, $tinymce\_version, $required\_php\_version, $required\_mysql\_version, $wp\_local\_package; /**\* version.php：这个文件包含了WordPress的版本信息，如主版本号、次版本号、修订号等。通过这个文件，可以获取到当前WordPress的版本信息。 \* compat.php：这个文件包含了WordPress与其他软件或库的兼容性处理。例如，处理不同版本的PHP和MySQL之间的兼容性问题。 \* load.php：这个文件包含了WordPress的核心功能，如初始化数据库、加载插件和主题等。通过这个文件，可以实现WordPress的基本功能。 \*/ require ABSPATH . WPINC . '/version.php'; require ABSPATH . WPINC . '/compat.php'; require ABSPATH . WPINC . '/load.php'; ```

这里依次查看 这仨文件

#### version.php

```php <?php /\**\* WordPress版本.*/ $wp\_version = '6.6.1';

/\**\* 用于存储WordPress数据库的版本号*/ $wp\_db\_version = 57155;

/\**\* TinyMCE版本要求.*/ $tinymce\_version = '49110-20201110';

/\**\* PHP版本要求*/ $required\_php\_version = '7.2.24';

/\**\* MYSQL版本要求*/ $required\_mysql\_version = '5.5.5'; ```

compat.php只存在两个定义的常量和其他声明的函数

#### compat.php

``` php // IMAGETYPE\_AVIF constant is only defined in PHP 8.x or later. if ( ! defined( 'IMAGETYPE\_AVIF' ) ) { define( 'IMAGETYPE\_AVIF', 19 ); }

// IMG\_AVIF constant is only defined in PHP 8.x or later. if ( ! defined( 'IMG\_AVIF' ) ) { define( 'IMG\_AVIF', IMAGETYPE\_AVIF ); } ```

load.php中只有声明的函数，并没有执行任何工作

所以接着往下看代码

php // 检查PHP和MySQL版本是否满足WordPress的要求 wp\_check\_php\_mysql\_versions();

这个是load.php文件中的函数，观看细节：

```php /\**\* 这段PHP代码主要用于检查WordPress运行所需的PHP和MySQL版本。 \* 定义全局变量分别用于存储WordPress运行所需的PHP版本和WordPress版本。 \* 获取当前PHP版本$php\_version。 \* 使用version\_compare()函数比较当前PHP版本与至少要求的PHP版本。 \* 如果当前PHP版本低于WordPress所需的PHP版本，则提示并退出 \* 检查是否安装了MySQLi扩展，如果没有安装，且没有找到wp-content目录下的db.php文件，则提示并退出：*/ function wp\_check\_php\_mysql\_versions() { global $required\_php\_version, $wp\_version;

$php\_version = PHP\_VERSION;  
  
if (version\_compare($required\_php\_version, $php\_version, '>')) {  
 $protocol = wp\_get\_server\_protocol();  
 header(sprintf('%s 500 Internal Server Error', $protocol), true, 500);  
 header('Content-Type: text/html; charset=utf-8');  
 printf(  
 'Your server is running PHP version %1$s but WordPress %2$s requires at least %3$s.',  
 $php\_version,  
 $wp\_version,  
 $required\_php\_version  
 );  
 exit(1);  
}  
  
// 这段代码在定义之前运行，所以需要先进行检查  
$wp\_content\_dir = defined('WP\_CONTENT\_DIR') ? WP\_CONTENT\_DIR : ABSPATH . 'wp-content';  
  
if (!function\_exists('mysqli\_connect')  
 && !file\_exists($wp\_content\_dir . '/db.php')  
) {  
 require\_once ABSPATH . WPINC . '/functions.php';  
 wp\_load\_translations\_early();  
  
 $message = '<p>' . \_\_('Your PHP installation appears to be missing the MySQL extension which is required by WordPress.') . " '<code>\n";  
  
 $message .= '<p>' . sprintf(  
 /\* translators: %s: mysqli. \*/  
 \_\_('Please check that the %s PHP extension is installed and enabled.'),  
 '<code>mysqli . sprintf(  
 /\* translators: %s: Support forums URL. \*/  
 \_\_('If you are unsure what these terms mean you should probably contact your host. If you still need help you can always visit the <a href="%s">'  
 ) . "\* translators: %s: Support forums URL. \*/  
 \_\_('If you are unsure what these terms mean you should probably contact your host. If you still need help you can always visit the <a href="%s">\n";  
  
 $message .= '<p>' . sprintf(  
 /\* translators: %s: Support forums URL. \*/  
 \_\_('If you are unsure what these terms mean you should probably contact your host. If you still need help you can always visit the <a href="%s">WordPress support forums),  
 $args  
 );  
 exit(1);  
}

} ```

.'), \_\_('https://wordpress.org/support/forums/') ) . "exit' =>\n";

$args = array(  
 'exit' => false,  
 'code' => 'mysql\_not\_found',  
 );  
 wp\_die(  
 $message,  
 \_\_('Requirements Not Met'),  
 $args  
 );  
 exit(1);  
}

} ```

version\_compare() 函数返回一个整数，表示两个版本字符串的大小关系。如果第一个版本小于第二个版本，返回 -1；如果第一个版本大于第二个版本，返回 1；如果两个版本相等，返回 0。

这里的代码主要用于检查，所以我们还是回到wp-settings.php中继续向下面看

php require ABSPATH . WPINC . '/class-wp-paused-extensions-storage.php'; require ABSPATH . WPINC . '/class-wp-fatal-error-handler.php'; require ABSPATH . WPINC . '/class-wp-recovery-mode-cookie-service.php'; require ABSPATH . WPINC . '/class-wp-recovery-mode-key-service.php'; require ABSPATH . WPINC . '/class-wp-recovery-mode-link-service.php'; require ABSPATH . WPINC . '/class-wp-recovery-mode-email-service.php'; require ABSPATH . WPINC . '/class-wp-recovery-mode.php'; require ABSPATH . WPINC . '/error-protection.php'; require ABSPATH . WPINC . '/default-constants.php'; require\_once ABSPATH . WPINC . '/plugin.php';

这些文件其实都类似，我们取第一个解释

#### class-wp-paused-extensions-storage.php

```php <?php /\**\* WP\_Paused\_Extensions\_Storage的类，主要用于存储暂停的插件或主题。 \* 这个类主要用于处理错误保护API \* 确保在遇到错误时能够正确地存储和检索插件或主题的错误信息。*/

/\**\* 存储已暂停插件的核心类 \* \* @since 5.2.0*/

# [AllowDynamicProperties]

class WP\_Paused\_Extensions\_Storage ```

这里的 #[AllowDynamicProperties] 是 PHP 中的一个属性，用于允许在类中使用动态属性。

当在类中使用 #[AllowDynamicProperties] 属性时，可以允许在类中使用未定义的属性。这样，即使属性名不是有效的标识符，也可以正常设置属性值。

例如，以下代码定义了一个名为 MyClass 的类，并使用 #[AllowDynamicProperties] 属性允许动态属性：

```php <?php

# [AllowDynamicProperties]

class MyClass { public $prop1 = 'Hello, world!'; public $prop2 = 'PHP is fun!'; }

$obj = new MyClass(); $obj->dynamicProp = 'Dynamic properties are cool!';

echo $obj->prop1; // 输出 "Hello, world!" echo $obj->prop2; // 输出 "PHP is fun!" echo $obj->dynamicProp; // 输出 "Dynamic properties are cool!" ```

这段代码定义了一个名为 MyClass 的类，并使用 #[AllowDynamicProperties] 属性允许动态属性。然后，创建了一个 MyClass 对象 $obj，并设置了动态属性 dynamicProp。最后，输出了 $obj 的属性值。

需要注意的是，虽然 #[AllowDynamicProperties] 属性允许在类中使用动态属性，但建议尽量使用已定义的属性，以提高代码的可读性和可维护性。

接着继续读代码

```php class WP\_Paused\_Extensions\_Storage { /\**\* 主题类型*/ protected $type;

/\*\*  
 \* 接收一个参数$extension\_type，表示插件或主题的类型，可以是'plugin'或'theme'。  
 \*  
 \*/  
public function \_\_construct($extension\_type)  
{  
 $this->type = $extension\_type;  
}  
  
/\*\*  
 \* 用于记录插件或主题的错误信息。如果已经记录了相同的错误，则覆盖之前的错误。  
 \*/  
public function set($extension, $error)  
{  
 if (!$this->is\_api\_loaded()) {  
 return false;  
 }  
  
 $option\_name = $this->get\_option\_name();  
  
 if (!$option\_name) {  
 return false;  
 }  
  
 $paused\_extensions = (array) get\_option($option\_name, array());  
  
 // Do not update if the error is already stored.  
 if (isset($paused\_extensions[$this->type][$extension]) && $paused\_extensions[$this->type][$extension] === $error) {  
 return true;  
 }  
  
 $paused\_extensions[$this->type][$extension] = $error;  
  
 return update\_option($option\_name, $paused\_extensions);  
}  
  
/\*\*  
 \* 用于删除之前记录的插件或主题的错误信息。  
 \*/  
public function delete($extension)  
{  
 if (!$this->is\_api\_loaded()) {  
 return false;  
 }  
  
 $option\_name = $this->get\_option\_name();  
  
 if (!$option\_name) {  
 return false;  
 }  
  
 $paused\_extensions = (array) get\_option($option\_name, array());  
  
 // Do not delete if no error is stored.  
 if (!isset($paused\_extensions[$this->type][$extension])) {  
 return true;  
 }  
  
 unset($paused\_extensions[$this->type][$extension]);  
  
 if (empty($paused\_extensions[$this->type])) {  
 unset($paused\_extensions[$this->type]);  
 }  
  
 // Clean up the entire option if we're removing the only error.  
 if (!$paused\_extensions) {  
 return delete\_option($option\_name);  
 }  
  
 return update\_option($option\_name, $paused\_extensions);  
}  
  
/\*\*  
 \* 用于获取指定插件或主题的错误信息。  
 \*  
 \* @since 5.2.0  
 \*  
 \* @param string $extension Plugin or theme directory name.  
 \* @return array|null Error that is stored, or null if the extension is not paused.  
 \*/  
public function get($extension)  
{  
 if (!$this->is\_api\_loaded()) {  
 return null;  
 }  
  
 $paused\_extensions = $this->get\_all();  
  
 if (!isset($paused\_extensions[$extension])) {  
 return null;  
 }  
  
 return $paused\_extensions[$extension];  
}  
  
/\*\*  
 \* 用于获取所有暂停的插件或主题及其错误信息。  
 \*/  
public function get\_all()  
{  
 if (!$this->is\_api\_loaded()) {  
 return array();  
 }  
  
 $option\_name = $this->get\_option\_name();  
  
 if (!$option\_name) {  
 return array();  
 }  
  
 $paused\_extensions = (array) get\_option($option\_name, array());  
  
 return isset($paused\_extensions[$this->type]) ? $paused\_extensions[$this->type] : array();  
}  
  
/\*\*  
 \* 移除所有已经暂停的插件.  
 \*/  
public function delete\_all()  
{  
 if (!$this->is\_api\_loaded()) {  
 return false;  
 }  
  
 $option\_name = $this->get\_option\_name();  
  
 if (!$option\_name) {  
 return false;  
 }  
  
 $paused\_extensions = (array) get\_option($option\_name, array());  
  
 unset($paused\_extensions[$this->type]);  
  
 if (!$paused\_extensions) {  
 return delete\_option($option\_name);  
 }  
  
 return update\_option($option\_name, $paused\_extensions);  
}  
  
/\*\*  
 \* 用于检查是否已经加载了存储暂停插件或主题的底层API。  
 \*/  
protected function is\_api\_loaded()  
{  
 return function\_exists('get\_option');  
}  
  
/\*\*  
 \* 用于获取用于存储暂停插件或主题的选项名称。  
 \*/  
protected function get\_option\_name()  
{  
 if (!wp\_recovery\_mode()->is\_active()) {  
 return '';  
 }  
  
 $session\_id = wp\_recovery\_mode()->get\_session\_id();  
 if (empty($session\_id)) {  
 return '';  
 }  
  
 return "{$session\_id}\_paused\_extensions";  
}

}

```

由此可揣测，这些文件是WP内部用于加载一些任务或事件的，并不需要过多关注，这些类并没有立即执行，对我们理解WP入口进入主题不能起到很大的作用

接着看wp-settings.php

php /\*\* \* $blog\_id用于标识当前WordPress站点（博客）的ID。 \* 在单站点配置中，如果尚未配置$blog\_id，则默认为1。 \* 在多站点配置中，它将被ms-settings.php文件中的默认设置覆盖 \*/ global $blog\_id; // 初始化常量包括 WP\_MEMORY\_LIMIT, WP\_MAX\_MEMORY\_LIMIT, WP\_DEBUG, SCRIPT\_DEBUG, WP\_CONTENT\_DIR and WP\_CACHE. wp\_initial\_constants();

wp\_initial\_constants中，基本都是定义常量的，我们取其中一部分查看

php define( 'KB\_IN\_BYTES', 1024 ); define( 'MB\_IN\_BYTES', 1024 \* KB\_IN\_BYTES ); define( 'GB\_IN\_BYTES', 1024 \* MB\_IN\_BYTES ); define( 'TB\_IN\_BYTES', 1024 \* GB\_IN\_BYTES ); define( 'PB\_IN\_BYTES', 1024 \* TB\_IN\_BYTES ); define( 'EB\_IN\_BYTES', 1024 \* PB\_IN\_BYTES ); define( 'ZB\_IN\_BYTES', 1024 \* EB\_IN\_BYTES ); define( 'YB\_IN\_BYTES', 1024 \* ZB\_IN\_BYTES );

其实这里定义的是存储容量单位大小的常量

类似地，定义的还有：

1. 定义了一些表示数据大小的常量，如KB\_IN\_BYTES、MB\_IN\_BYTES、GB\_IN\_BYTES等，方便后续的计算。
2. 定义了WP\_START\_TIMESTAMP，表示程序运行的开始时间。
3. 获取并定义了memory\_limit的值，这是PHP的一个配置选项，表示PHP程序可以使用的最大内存量。
4. 定义了WP\_MEMORY\_LIMIT和WP\_MAX\_MEMORY\_LIMIT两个常量，分别表示WordPress程序的内存限制和最大内存限制。
5. 设置了memory\_limit的值，如果当前的memory\_limit设置不可更改，或者当前的WP\_MEMORY\_LIMIT大于memory\_limit，那么将memory\_limit设置为WP\_MEMORY\_LIMIT。
6. 如果blog\_id未定义，则将其设置为1。
7. 定义了WP\_CONTENT\_DIR常量，表示wp-content目录的路径。
8. 定义了WP\_DEVELOPMENT\_MODE常量，表示WordPress的开发模式。
9. 定义了WP\_DEBUG常量，表示是否显示错误信息。
10. 定义了WP\_DEBUG\_DISPLAY常量，表示是否在浏览器中显示错误信息。
11. 定义了WP\_DEBUG\_LOG常量，表示是否将错误信息记录到日志文件中。
12. 定义了WP\_CACHE常量，表示是否启用缓存。
13. 定义了SCRIPT\_DEBUG常量，表示是否加载未压缩的脚本和样式表。
14. 定义了一些表示时间间隔的常量，如MINUTE\_IN\_SECONDS、HOUR\_IN\_SECONDS、DAY\_IN\_SECONDS等，方便后续的计算。

接着是处理器函数：

php /\* 这个函数会注册一个关闭处理器，当程序发生致命错误时， \* 这个处理器会被触发，从而执行一些清理工作 \* 例如保存错误信息到数据库或者发送错误报告到指定的邮箱等。 \*/ wp\_register\_fatal\_error\_handler();

这里笔者个人认为还是有必要观察这个代码细节的，因为错误处理在开发中是比较常用的

php /\*\* \* 注册错误处理器 \*/ function wp\_register\_fatal\_error\_handler() { // 检查致命错误处理器是否开启 if (!wp\_is\_fatal\_error\_handler\_enabled()) { return; } $handler = null; if (defined('WP\_CONTENT\_DIR') && is\_readable(WP\_CONTENT\_DIR . '/fatal-error-handler.php')) { $handler = include WP\_CONTENT\_DIR . '/fatal-error-handler.php'; } /\* \* 检查$handler是否为对象，并且是否具有handle方法。 \* 如果不是，则创建一个新的WP\_Fatal\_Error\_Handler对象， \* 并将其赋值给$handler变量。 \*/ if (!is\_object($handler) || !is\_callable(array($handler, 'handle'))) { $handler = new WP\_Fatal\_Error\_Handler(); } /\* register\_shutdown\_function函数将handle方法注册为关闭函数， 当PHP脚本执行完毕时，这个方法会被调用。\*/ register\_shutdown\_function(array($handler, 'handle')); }

当PHP脚本遇到一个无法恢复的错误时，如内存不足、文件无法打开等，这个处理器会被触发。

运行时一般会new WP\_Fatal\_Error\_Handler();

```php public function handle() { if (defined('WP\_SANDBOX\_SCRAPING') && WP\_SANDBOX\_SCRAPING) { return; }

// 检查当前是否处于维护模式。如果是，则直接返回，不执行后续操作。  
 if (wp\_is\_maintenance\_mode()) {  
 return;  
 }  
  
 try {  
 // 调用detect\_error()方法检测是否存在错误。如果不存在错误，则直接返回  
 $error = $this->detect\_error();  
 if (!$error) {  
 return;  
 }  
  
  
 if (!isset($GLOBALS['wp\_locale']) && function\_exists('load\_default\_textdomain')) {  
 load\_default\_textdomain();  
 }  
  
 $handled = false;  
 /\*  
 如果不是多站点且已初始化恢复模式，  
 则调用wp\_recovery\_mode()->handle\_error()方法处理错误，  
 并将处理结果赋值给$handled。  
 \*/  
 if (!is\_multisite() && wp\_recovery\_mode()->is\_initialized()) {  
 $handled = wp\_recovery\_mode()->handle\_error($error);  
 }  
  
 /\*\*  
 \* 如果是在后台或响应头未发送，  
 \* 则调用display\_error\_template()方法显示错误模板。  
 \*/  
 if (is\_admin() || !headers\_sent()) {  
 $this->display\_error\_template($error, $handled);  
 }  
 } catch (Exception $e) {  
 // Catch exceptions and remain silent.  
 }  
}

```

总之，这段代码的作用是注册一个致命错误处理器，当遇到无法恢复的错误时，可以自动处理错误并给出友好的错误提示。

讲到这里，require和include（\_once）的区别附上如下：

#### require、include、require\_once和include\_once的区别

require在包含文件时如果文件不存在，会抛出一个致命错误

include会给出一个警告，但仍然会继续执行后续代码

* require\_once 和 include\_once 用于确保包含的文件只被执行一次，会创建一个新的作用域

接着我们继续讲解settings.php

```php // 将默认时区设置为UTC。这是因为WordPress计算偏移量时使用的是UTC时间。 date\_default\_timezone\_set('UTC');

//标准化$\_SERVER变量。这可以确保在不同环境下，$\_SERVER变量的值是一致的。 wp\_fix\_server\_vars();

// 检查网站是否处于维护模式。维护模式中只有管理员可以访问网站，其他用户将被重定向到维护页面。 wp\_maintenance();

// 开始加载计时器，测量加载页面的事件 timer\_start();

// 是否启用Debug，启用的话会在后台显示一些调试信息 wp\_debug\_mode();

if (WP\_CACHE && apply\_filters('enable\_loading\_advanced\_cache\_dropin', true) && file\_exists(WP\_CONTENT\_DIR . '/advanced-cache.php')) { // 用于高级缓存插件，以提高网站的性能 include WP\_CONTENT\_DIR . '/advanced-cache.php';

// 如果advanced-cache.php文件中手动添加了任何钩子，则重新初始化这些钩子  
if ($wp\_filter) {  
 $wp\_filter = WP\_Hook::build\_preinitialized\_hooks($wp\_filter);  
}

}

// 没有定义WP\_LANG\_DIR则重新定义 wp\_set\_lang\_dir(); ```

php require ABSPATH . WPINC . '/class-wp-list-util.php'; require ABSPATH . WPINC . '/class-wp-token-map.php'; require ABSPATH . WPINC . '/formatting.php'; require ABSPATH . WPINC . '/meta.php'; require ABSPATH . WPINC . '/functions.php'; require ABSPATH . WPINC . '/class-wp-meta-query.php'; require ABSPATH . WPINC . '/class-wp-matchesmapregex.php'; require ABSPATH . WPINC . '/class-wp.php'; require ABSPATH . WPINC . '/class-wp-error.php'; require ABSPATH . WPINC . '/pomo/mo.php'; require ABSPATH . WPINC . '/l10n/class-wp-translation-controller.php'; require ABSPATH . WPINC . '/l10n/class-wp-translations.php'; require ABSPATH . WPINC . '/l10n/class-wp-translation-file.php'; require ABSPATH . WPINC . '/l10n/class-wp-translation-file-mo.php'; require ABSPATH . WPINC . '/l10n/class-wp-translation-file-php.php';

接下来说明以下每个文件的作用：

1. class-wp-list-util.php：定义了WP\_List\_Util类，用于处理列表相关的操作。
2. class-wp-token-map.php：定义了WP\_Token\_Map类，用于处理令牌映射。
3. formatting.php：包含了一些用于格式化文本的函数。
4. meta.php：包含了一些与文章元数据相关的函数。
5. functions.php：包含了一些通用的函数，如处理URL、文件等。
6. class-wp-meta-query.php：定义了WP\_Meta\_Query类，用于处理元数据查询。
7. class-wp-matchesmapregex.php：定义了WP\_MatchesMapRegex类，用于处理匹配映射正则表达式。
8. class-wp.php：定义了WP类，用于处理全局变量、数据库连接等。
9. class-wp-error.php：定义了WP\_Error类，用于处理错误和异常。
10. pomo/mo.php：包含了一些与MO文件（翻译文件）相关的函数。
11. class-wp-translation-controller.php：定义了WP\_Translation\_Controller类，用于处理翻译相关的操作。
12. class-wp-translations.php：定义了WP\_Translations类，用于处理翻译文件。
13. class-wp-translation-file.php：定义了WP\_Translation\_File类，用于处理翻译文件。
14. class-wp-translation-file-mo.php：定义了WP\_Translation\_File\_MO类，用于处理MO翻译文件。
15. class-wp-translation-file-php.php：定义了WP\_Translation\_File\_PHP类，用于处理PHP翻译文件。

接着看：

```php /\**\* @global wpdb $wpdb WordPress数据库抽象类实例*/ global $wpdb; // 导入wpdb类，用于与数据库进行交互 require\_wp\_db();

/\**\* 表前缀*/ $GLOBALS['table\_prefix'] = $table\_prefix;

// 设置数据库表前缀和数据库表列的格式规范 wp\_set\_wpdb\_vars();

// 开始WordPress对象缓存，或者如果存在drop-in，则开始外部对象缓存 wp\_start\_object\_cache();

// 引入默认过滤器。 require ABSPATH . WPINC . '/default-filters.php';

// 如果启用了多站点 if (is\_multisite()) { require ABSPATH . WPINC . '/class-wp-site-query.php'; require ABSPATH . WPINC . '/class-wp-network-query.php'; require ABSPATH . WPINC . '/ms-blogs.php'; require ABSPATH . WPINC . '/ms-settings.php'; } elseif (!defined('MULTISITE')) { define('MULTISITE', false); }

// 注册关闭函数 register\_shutdown\_function('shutdown\_action\_hook');

// 如果启用了SHORTINIT，则停止加载WordPress的大部分功能 if (SHORTINIT) { return false; }

// 加载L10n库 require\_once ABSPATH . WPINC . '/l10n.php'; require\_once ABSPATH . WPINC . '/class-wp-textdomain-registry.php'; require\_once ABSPATH . WPINC . '/class-wp-locale.php'; require\_once ABSPATH . WPINC . '/class-wp-locale-switcher.php';

// 如果没有安装则进行WP安装，里面会重定向 wp\_not\_installed();

// 加载库 require ABSPATH . WPINC . '/class-wp-walker.php'; require ABSPATH . WPINC . '/class-wp-ajax-response.php'; require ABSPATH . WPINC . '/capabilities.php'; require ABSPATH . WPINC . '/class-wp-roles.php'; require ABSPATH . WPINC . '/class-wp-role.php'; require ABSPATH . WPINC . '/class-wp-user.php'; ...... ```

接下来就是加载几乎所有的WP库,包含主题相关的文件等等

```php /\**\* 一个用于wp\_script\_modules()类的add\_hooks方法， \* 另一个用于wp\_interactivity()类的add\_hooks方法。 \* 这两个钩子将在主题设置后执行*/ add\_action('after\_setup\_theme', array(wp\_script\_modules(), 'add\_hooks')); add\_action('after\_setup\_theme', array(wp\_interactivity(), 'add\_hooks'));

/\**\* 创建一个新的WP\_Embed对象，并将其赋值给$wp\_embed全局变量*/ $GLOBALS['wp\_embed'] = new WP\_Embed();

/\**\* 创建一个新的WP\_Textdomain\_Registry对象，并将其赋值给$wp\_textdomain\_registry全局变量。 \* WP\_Textdomain\_Registry类用于支持在需要时为手动加载的文本域提供即时翻译。*/ $GLOBALS['wp\_textdomain\_registry'] = new WP\_Textdomain\_Registry(); $GLOBALS['wp\_textdomain\_registry']->init();

// 如果启用多站点功能，则引入以下文件 if (is\_multisite()) { require ABSPATH . WPINC . '/ms-functions.php'; require ABSPATH . WPINC . '/ms-default-filters.php'; require ABSPATH . WPINC . '/ms-deprecated.php'; } ```

紧接着开始定义插件目录

php // 它定义了一个必须使用的插件目录常量 PLUGIN\_DIR， // 它的默认值是 WP\_PLUGIN\_DIR wp\_plugin\_directory\_constants();

#### wp\_plugin\_directory\_constants函数

```php function wp\_plugin\_directory\_constants() { // 确保定义了WP\_CONTENT目录常量 if (!defined('WP\_CONTENT\_URL')) { define('WP\_CONTENT\_URL', get\_option('siteurl') . '/wp-content'); // Full URL - WP\_CONTENT\_DIR is defined further up. }

/\*\*  
 \* 插件的完整目录  
 \*/  
if (!defined('WP\_PLUGIN\_DIR')) {  
 define('WP\_PLUGIN\_DIR', WP\_CONTENT\_DIR . '/plugins'); // Full path, no trailing slash.  
}  
// 插件的完整URL  
if (!defined('WP\_PLUGIN\_URL')) {  
 define('WP\_PLUGIN\_URL', WP\_CONTENT\_URL . '/plugins'); // Full URL, no trailing slash.  
}  
  
/\*\*  
 \* 如果尚未定义，则定义为wp-content/plugins，  
 \* 这是一个相对路径，用于向后兼容  
 \*/  
  
if (!defined('PLUGINDIR')) {  
 define('PLUGINDIR', 'wp-content/plugins'); // Relative to ABSPATH. For back compat.  
}  
  
/\*\*  
 \* mu-插件的完整路径  
 \*/  
if (!defined('WPMU\_PLUGIN\_DIR')) {  
 define('WPMU\_PLUGIN\_DIR', WP\_CONTENT\_DIR . '/mu-plugins'); // Full path, no trailing slash.  
}  
  
/\*\*  
 \* mu-插件的完整URL  
 \*/  
if (!defined('WPMU\_PLUGIN\_URL')) {  
 define('WPMU\_PLUGIN\_URL', WP\_CONTENT\_URL . '/mu-plugins'); // Full URL, no trailing slash.  
}  
  
/\*\*  
 \* mu-插件的相对路径  
 \*/  
if (!defined('MUPLUGINDIR')) {  
 define('MUPLUGINDIR', 'wp-content/mu-plugins'); // Relative to ABSPATH. For back compat.  
}

} ```

接着看wp-settings.php，这里将会依次引入插件

```php /\**\* 用于存储已经加载过的插件的路径*/ $GLOBALS['wp\_plugin\_paths'] = array();

// 使用wp\_get\_mu\_plugins()函数获取所有必须使用的插件的路径,并遍历 foreach (wp\_get\_mu\_plugins() as $mu\_plugin) { $\_wp\_plugin\_file = $mu\_plugin; include\_once $mu\_plugin; //在插件的主文件被包含之后，将$mu\_plugin变量的值恢复为原始值，以避免在插件中修改变量值导致的问题。 $mu\_plugin = $\_wp\_plugin\_file;

/\*\*  
 \* 触发mu\_plugin\_loaded钩子，通知其他插件该插件已经加载完毕  
 \* 这个钩子可以用于在插件加载后执行一些操作。  
 \*/  
do\_action('mu\_plugin\_loaded', $mu\_plugin);

} //释放$mu\_plugin和$\_wp\_plugin\_file变量，避免内存泄漏。 unset($mu\_plugin, $\_wp\_plugin\_file); ```

接下来是用于加载多站点网络激活插件的代码

```php // 加载多站点网络激活的插件 if (is\_multisite()) { foreach (wp\_get\_active\_network\_plugins() as $network\_plugin) { wp\_register\_plugin\_realpath($network\_plugin);

$\_wp\_plugin\_file = $network\_plugin;  
 include\_once $network\_plugin;  
 $network\_plugin = $\_wp\_plugin\_file;  
  
 do\_action('network\_plugin\_loaded', $network\_plugin);  
}  
unset($network\_plugin, $\_wp\_plugin\_file);

} ```

和上面的加载方法基本相同

接着看

```php /\**\* 加载所有插件后执行的动作*/ do\_action('muplugins\_loaded');

if (is\_multisite()) { ms\_cookie\_constants(); }

// 定义常量 wp\_cookie\_constants();

// SSL相关常量定义 wp\_ssl\_constants();

// 引用公共变量库，简单变量 require ABSPATH . WPINC . '/vars.php';

// 在WordPress初始化时创建初始的分类法和文章类型 create\_initial\_taxonomies(); create\_initial\_post\_types(); ```

接着是加载已激活并有效的插件

```php // 在编辑文件时检测到错误，并将错误信息存储起来 wp\_start\_scraping\_edited\_file\_errors();

// 这个函数用于注册默认的主题目录根 register\_theme\_directory(get\_theme\_root());

if (!is\_multisite() && wp\_is\_fatal\_error\_handler\_enabled()) { // 检查当前是否是多站点模式，并且是否启用了致命错误处理器 // 在用户请求恢复模式链接并启动恢复模式时进行处理 wp\_recovery\_mode()->initialize(); }

// 遍历所有已激活且有效的插件 foreach (wp\_get\_active\_and\_valid\_plugins() as $plugin) { // 将插件的绝对路径注册到WordPress中，以便WordPress可以找到和管理插件 wp\_register\_plugin\_realpath($plugin);

$\_wp\_plugin\_file = $plugin;  
include\_once $plugin;  
$plugin = $\_wp\_plugin\_file;  
  
// 在插件加载完成后执行  
do\_action('plugin\_loaded', $plugin);

} // 释放内存 unset($plugin, $\_wp\_plugin\_file); ```

接着是可插拔函数：

```php // 加载可插拔函数 require ABSPATH . WPINC . '/pluggable.php'; require ABSPATH . WPINC . '/pluggable-deprecated.php';

// 设置内部编码 wp\_set\_internal\_encoding();

// 如果启用了对象缓存并存在wp\_cache\_postload()函数，则运行该函数 if (WP\_CACHE && function\_exists('wp\_cache\_postload')) { wp\_cache\_postload(); }

/\**\* 触发plugins\_loaded动作 \* 此时，可插拔函数已经加载，插件可以执行一些操作*/ do\_action('plugins\_loaded');

// 定义一些影响功能性的常量 wp\_functionality\_constants();

// 用于处理用户输入，防止SQL注入等安全问题 wp\_magic\_quotes();

/\**\* 在清理评论cookie时触发*/ do\_action('sanitize\_comment\_cookies'); ```

后边就是定义全局变量以及初始化主题

```php /\**\* WordPress Query 对象*/ $GLOBALS['wp\_the\_query'] = new WP\_Query(); $GLOBALS['wp\_query'] = $GLOBALS['wp\_the\_query'];

/\**\* URL重写对象*/ $GLOBALS['wp\_rewrite'] = new WP\_Rewrite();

/\**\* WordPress环境实例*/ $GLOBALS['wp'] = new WP();

/\**\* 小工具工厂对象*/ $GLOBALS['wp\_widget\_factory'] = new WP\_Widget\_Factory();

/\**\* 用户角色对象*/ $GLOBALS['wp\_roles'] = new WP\_Roles();

/\**\* 设置主题*/ do\_action('setup\_theme');

// 定义模板相关的常量和全局变量 wp\_templating\_constants(); wp\_set\_template\_globals();

// 加载默认的文本本地化域 load\_default\_textdomain();

$locale = get\_locale(); $locale\_file = WP\_LANG\_DIR . "/$locale.php"; if ((0 === validate\_file($locale)) && is\_readable($locale\_file)) { require $locale\_file; } unset($locale\_file);

/\**\* WP\_Locale对象*/ $GLOBALS['wp\_locale'] = new WP\_Locale();

/\**\* 用于切换本地化*/ $GLOBALS['wp\_locale\_switcher'] = new WP\_Locale\_Switcher(); $GLOBALS['wp\_locale\_switcher']->init();

// 遍历加载活动主题和有效主题的functions.php文件 foreach (wp\_get\_active\_and\_valid\_themes() as $theme) { if (file\_exists($theme . '/functions.php')) { include $theme . '/functions.php'; } } unset($theme);

/\**\* 用于在主题加载后执行一些操作*/ do\_action('after\_setup\_theme');

// 创建WP\_Site\_Health类实例 if (!class\_exists('WP\_Site\_Health')) { require\_once ABSPATH . 'wp-admin/includes/class-wp-site-health.php'; } WP\_Site\_Health::get\_instance();

// 初始化当前用户 $GLOBALS['wp']->init(); ```

完成站点构建

```php /\**\* 用于初始化各种插件和小工具*/ do\_action('init');

// 检查站点状态 if (is\_multisite()) { $file = ms\_site\_check(); if (true !== $file) { require $file; die(); } unset($file); }

/\**\* 触发wp\_loaded动作，通常用于在WordPress完全加载后执行一些操作*/ do\_action('wp\_loaded');

```

至此站点加载完毕