84PHP 开源框架

入门指南

V1.2.1

目录

文档 2

综述 3

目录结构 4

准备工作及核心配置 5

模块 7

语法 8

模板 11

缓存及自动编译 14

报错机制 15

路由 16

安全规范 17

版权声明&联系方式 18

文档

除了本文档之外，在源码包内 Documents 目录下还有基础模块以及可选模块的说明文档，如有需要可查阅。

综述

感谢您选择 84PHP 开源框架!

请注意，一定不要以使用 MVC 模式开发框架的使用经验来理解 84PHP，不然你的大脑会宕机的。

此框架是一款力求在性能指标与简洁度之间达到平衡的开发框架,同时尽可能的在减少开发者使用框架时所花费的时间，为此我们引入了诸多新颖的设计理念:

->低耦合的可插拔模块设计

->简洁的文件结构

->可直接访问的编译缓存

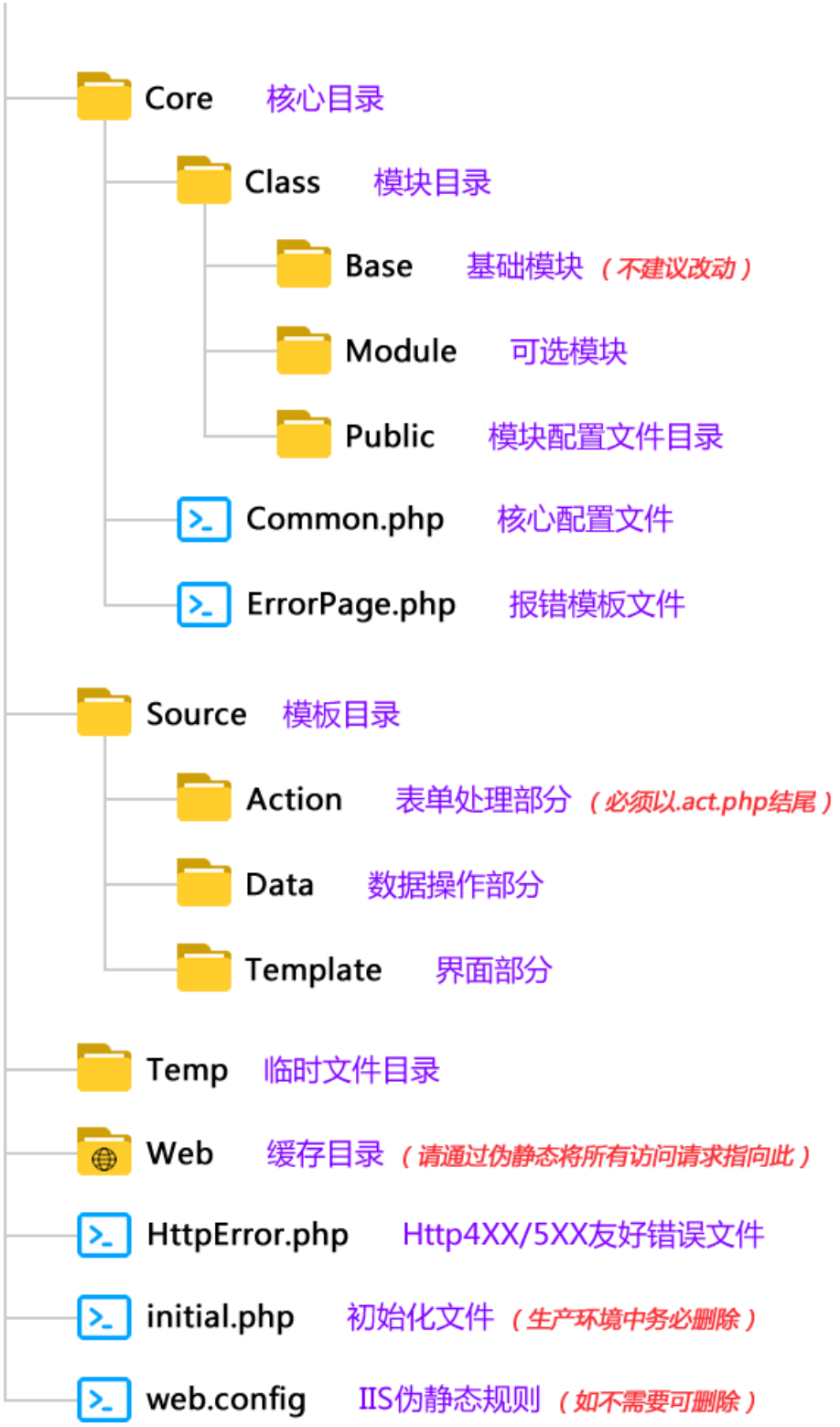
->完善详细的多语言报错信息

->与云服务相契合

本篇文档适合于对原生 PHP 已经有一定了解的开发者阅读（链接：*使用这个框架需要怎样的 PHP 编程水平？学习这个框架的时间平均是多久？*）。

准备好了吗？接下来，我们将为您详细介绍如何从零开始使用 84PHP 框架!

目录结构


Core 核心目录
Class 模块目录
Base 基础模块（不建议改动）
Module 可选模块
Public 模块配置文件目录
Common.php 核心配置文件
ErrorPage.php 报错模板文件
Source 模板目录
Action 表单处理部分（必须以.act.php结尾）
Data 数据操作部分
Template 界面部分
Temp 临时文件目录
Web 缓存目录（请通过伪静态将所有访问请求指向此）
HttpError.php Http4XX/5XX友好错误文件
initial.php 初始化文件（生产环境中务必删除）
web.config IIS伪静态规则（如不需要可删除）

准备工作及核心配置

在开始使用前，请通过伪静态将所有流量牵引至/Web 目录：

IIS 中，根目录下已附带 web.config 文件，在安装好 URL 重写模块之后，即可使用，也可根据需要自行调整。

nginx 中，.conf 文件的示例配置代码：

```
error_page   403 404 500 502 503 504    /HttpError.php;
location ~* ^\/(source|core|temp).* {
    rewrite "^/(.*)$" /Web/$1 last;
}
location /Web {
    if (-d $request_filename){
        rewrite "^/(.*)$" /$1/index.php last;
    }
    if (!-f $request_filename){
        rewrite "^/(.*)$" /$1.php last;
    }
}
location / {
    if (!-f $request_filename){
        rewrite "^/(.*)$" /Web/$1 last;
    }
}
```

以上规则，隐藏了.php 文件后缀，并屏蔽了对/Core、/Source、/Temp 路径的访问，以及当 HTTP 状态码为 4xx 或 5xx 时，展示友好的出错页面。

因此，当访问 yourdomain.com/a/b 时，将读取/Web/a/b.php 文件。

框架的核心配置文件位于/Core/Common.php，包含了所有的关键设置：

->DebugMode：调试模式（Bool），TRUE 代表开启、FALSE 代表关闭。当调试模式关闭时，报错等级将调整至最低（意味着某些不重要的错误将会被忽略），而当调试模式开启时，框架会自动对修改过的模板文件进行编译，并告知浏览器对所有页面都不进行缓存，同时在发生错误时显示详细的报错信息（可能包含文件路径、出错类型或数据库信息）。因此，在正式使用（以下称作生产环境）时，务必*关闭调试模式*。

->FrameworkHelpLanguage：框架帮助信息语言（String），“CN”代表中文、“EN”代表英语。框架会在显示帮助性文字（例如报错信息）时根据此设置显示指定的语言。

->RootPath：用以计算根目录路径，并非配置项，*禁止修改*。

->XPoweredBy：用以指定如何显示服务器端运行环境，默认为“ASP.NET”，可起到一定的伪装及安全防护作用，不建议修改。

自 1.1.2 版本起，加入了 ob_start()，开启了缓冲区控制。*在今后版本的文档中不再赘述。*

模块

再次提示：

一定不要以 MVC 模式开发框架的使用经验来理解 84PHP，不然你的大脑会宕机的。

在路径/Core/Class 中，有 Base、Module、Public 三个文件夹，其中包含了框架所有使用到的类（Class，每一个类均拥有各自的一类功能）。

但在今后所有关于 84PHP 的表述中，将把它称作“模块（Module）”：

Base 目录下的模块，是基础模块，在 Model 目录下的可选模块中可能会使用到这些基础模块。除非您完全了解框架的运行原理并参透了代码含义，*否则其中的任何一个文件都不能删除*。

Module 目录下的模块，是可选模块，除了在应用中被使用到的模块，可以删除其余的模块以精简代码。框架源码包（在线生成的除外）中已经包含了大部分常用模块，如果您需要下载这些模块，可以前往官网中的在线生成页面单独下载由官方团队编写的模块，或者您自行编写，还可以进入论坛中下载其他开发者编写的第三方模块。**使用第三方模块时，您需要对 Public 目录下的 Cache 模块的配置文件进行调整，才能够使得第三方模块生效。**

请注意，*Base 目录和 Module 目录下的模块名称不能够重复。*

Public 目录下**可能存在**与模块名相对应的配置文件，例如 Pay.php 对应的是 Pay 模块（提供在线支付功能，在 Module 目录下），其内存储着支付接口的通信密钥和其它 Pay 模块所需的配置项。对于基础模块的配置文件，建议*谨慎修改*。

特别提示：

如果您觉得在每个文件中分别配置信息过于繁琐，您可以将每个模块配置文件中的内容剪切并粘贴在核心配置文件 Common.php 中，或者也可以配置项存储在数据库中（不建议，因为需要自行修改代码逻辑）。但是，*任何对模块的修改，是基于您完全了解框架的运行原理并参透了代码含义的基础之上，否则将带来错误风险和安全隐患。*

语法

框架中有两种语法，一种是模块语法，另一种是前端语法。

模块语法，就是对模块进行调用的语法，它们应当出现在[Action、Data]目录中的模板文件里，完成模块对应的功能。

语法如下：

#[变量=]<模块名@方法名([参数 1, [参数 2, [...]]])>

在详述之前，请先看一个例子：

如果在模板文件中使用模块语法：

#$Post=<Receive@Post()>

那么，将会调用 Receive 模块中的 Post()方法对 POST 接收到的表单数据进行处理并将结果存储在$Post 变量中（$Post 可以是任意的变量名）。

来看一看编译之后的代码：

```
require_once (dirname(__FILE__)."/../Core/Class/Base/Receive.Class.php");
$ClassReceive=new Receive;
$Post=$GLOBALS['ClassReceive']->Post();
```

经过编译，框架自动请求了对应模块的**物理路径**，并**自动判断**了模块类别（Base 或 Module），同时**实例化**了模块（实际上是类）和调用指定的方法。

再来看模块语法：

#[变量=]<模块名@方法名([参数 1, [参数 2, [...]]])>

变量是可选的，如果填写了变量，务必要在变量后加上=，否则框架会报错。

如果模块中的方法没有返回值，或者您不需要模块的返回值，**可以留空**。例如在 Vcode 模块中使用 Base()方法生成一个验证码，由于在 Vcode 模块中 Base()方法是**没有返回值**的，那么：

#<Vcode@Base()>

就可以完成方法的使用。

特别注意：

框架之所以能够自动判断模块类别（Base 或 Module），是取决于 Cache 模块的配置文件，即/Core/Class/Public/Cache.php 中的**$ModuleArray** 变量的配置。当引用当

前版本框架的官方源码中所没有的模块，或第三方模块时，**请更新$ModuleArray 配置。**

前端语法，就是前端模板中的快捷语法，以 HTML 内嵌代码的形式实现 PHP 语言中的 echo、if...else if...else...等功能，并且可以嵌套结合使用。他们应当出现在[Template]目录中的模板文件里。

前端模板支持原生 PHP 语法，如需使用，请加入完整的 PHP 界定符<?php ... ?>。

和大多数框架类似，前端语法如下：

输出变量：

```
{变量}
```

例如，

```
{$Var}
```

等价于：

```
<?php echo $Var ?>
```

if...else if...else...语句：

```
{if (条件)}
    [code]
{else if (条件)}
    [code]
{else}
    [code]
{/if}
```

例如，

```
{if ($Var==1) }
    Yes{$Var}.
{else if ($Var==2) }
    Lucky{$Var}.
{else}
    No.
```

```
{/if}
```

等价于：

```
<?php if ($Var==1){ ?>
    Yes<?php echo $Var; ?>.
<?php else if ($Var==2){ ?>
    Lucky<?php echo $Var; ?>.
<?php }else{ ?>
    No.
<?php }?>
```

Foreach（也可以是 loop）语句，数组的元素键为$Key，对应的值为$Val：

```
{foreach 变量}
    [code]
{else}
    [code]
{/foreach}
```

例如，

```
{foreach $Array }
    The key is {$Key}, and the value is {$Val}.
{/foreach}
```

等价于：

```
<?php foreach ($Array as $Key => $Val){ ?>
    The key is <?php echo $Key; ?>, and the value is <?php echo $Val; ?>.
<?php }?>
```

特别注意：

前端语法的语法规则存储于**/Core/Class/Public/Cache.php**，您可以自行设计语法。

模板

框架模板文件的路径是**/Source**，包含 **Action、Data、Template** 三个目录。

浏览器访问时，**在没有设置额外伪静态规则的情况下，Web 目录下会有与之对应对应的实体文件**，即访问 yourdomain.com/a/test 时，/Web 目录下，目录 a 以及目录 a 中的 test.php 是**真实存在的**，而这些文件就是经过**编译处理之后**的文件。

Action 目录，是处理表单的代码存放的目录，文件名必须以.act.php 结尾。

Data 目录，是操作数据的代码存放的目录。

Template 目录，是前端模板存放的目录，一般来说尽可能的少包含原生 PHP 代码，以便提高模板的移植性。

三者之间的关系是：

1.假设 Template 目录有一个文件 A.php，如果 A.php 只是一个没有数据操作的页面，比如仅仅是从 Session 中输出用户名

```
{$_SESSION['username']}
```

，**而没有查询数据库这样需要用到模块的复杂操作**，则只需要建立 Template/A.php 即可，框架按照**前端语法**编译后将会在/Web 目录下生成 A. php。

2.如果 Template 目录中的 B.php 需要展示用户当前的余额$Count，而余额$Count 需要从数据库中查找最新数据，那么，

需要在 Data 目录下建立一个*相同名称的文件*，即建立 Data/B.php，并在其中编写例如：

```
#$Count=<Sql@Select(对应参数)>
```

的代码并保存，然后建立 Template/B.php，并在其中编写例如（假设余额存储在数组元素 Total 中）：

```
{$Count['Total']}
```

的代码并保存，框架会将 Data/B.php 中出现的**模块语法**编译（编译完成的部分称作 PartD）、将 Template/B.php 按照**前端语法**编译（编译完成的部分称作 PartT），随后将 PartD 在前、PartT 在后的规则，将这两部分编译后代码合并，最终在框架/Web 目录下生成 B. php。

本例编译后的 B.php 代码如下：

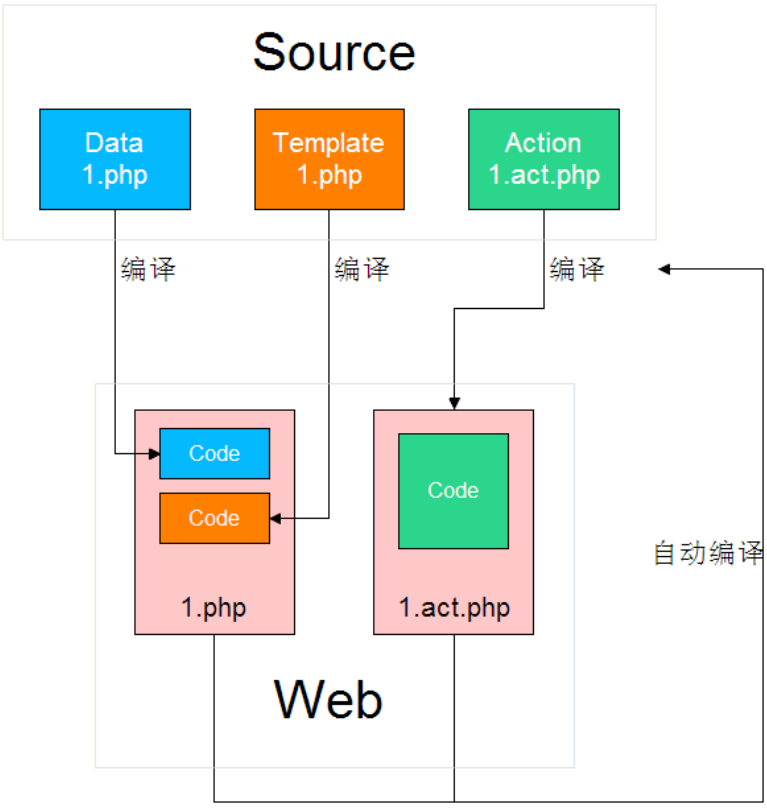
```php
require_once (dirname(__FILE__)."/../Core/Class/Base/Sql.Class.php");
$ClassSql=new Sql;

$Count=$GLOBALS['ClassSql']->Select(对应参数);


echo $Count['Total'];
?>
```

当仅仅需要操作数据，而不进行大量 HTML 输出时，比如判断用户是否登陆的代码，可以仅在 Data 目录下建立模板。

3.如果 Template 目录中的 C.php 有一个表单，那么就需要在 Action 目录下建立一个**对应前缀+.act+对应后缀的文件用以处理表单**，也就是 C.act.php，语法遵循**模块语法。**

*不应当在 Action 目录的模板中进行大量 HTML 输出！*因为这会降低模板的可移植性。一般来说用 JS 弹窗告知用户表单处理结果（如“注册成功！”）然后跳转至新页面，或跳转至结果页。

下面是三者关系图：



特别注意：

模板文件的文件类型为 php，不支持修改，为了防止模板文件被访问，除了通过服务器设置访问规则外，请务必在每一个模板文件加入

```
exit;#
```

，这六个字符不能多也不能少，独占一行。作用是当用户访问模板文件时，由于存在 exit;而停止运行。

这一行代码会在编译时被自动忽略。

Action、Data、Template 目录中模板文件的路径要**保持相同**，如 Template/Test/A.php 对应的文件是 Data/Test/A.php 和 Action/Test/A.act.php。

模板文件**相对于模板目录**（Action、Data、Template）的路径，也就是编译后文件相对于**/Web 目录**的路径。

缓存及自动编译

在/Web 下的所有文件均为经过编译的缓存文件。

当初次运行时，即/Web 目录下没有任何缓存文件时，请*访问 yourdomain.com/initial.php，运行站点根目录的 initial.php 进行初始化编译。*

框架具有自动编译机制，当调试模式开启时，所有模板文件中如果存在模板文件的修改时间晚于缓存文件的修改时间，或者是存在没有被编译的模板文件时，则框架会对该模板自动进行编译。因此如果您要修改代码，*切勿直接修改缓存文件！这将会导致自动编译失效。如遇此类情况，请修改模板文件使得模板文件的修改时间晚于缓存文件。*

另外，服务器进行时间调整时，或服务器时间异常时，可能会导致自动编译失效。框架会自动重置模板文件的修改时间来修复这个问题。如果仍然无法解决，请调整服务器时间，并清空/Web 目录下的所有文件，并运行站点根目录的 initial.php 进行初始化编译。

如果您的模板文件存在语法错误，例如出现两个;;符号，将会触发 php 系统级别的报错，且报错将会发生在自动编译模块运行之前。那么，您可以在修改完模板文件中的错误代码之后，访问/Web 目录下任意一个可以正常运行的文件（即您站点中的任意一个能够被正常显示的页面），或者运行站点根目录的 initial.php 来完成前述存在语法错误的缓存文件的更新。当您要让一个没有对应缓存文件的模板文件生成缓存时，也可以这么做。

特别注意：

Windows10+IIS10 环境下，会出现权限不足报错，我们正在努力解决这个问题。

报错机制

框架拥有完善的报错机制，报错信息的友好界面模板页位于/Core/ErrorPage.php 中，您可以自行替换，其中{$ErrorInfo}为报错信息显示的位置。

当调试模式开启时，将会显示详细的报错信息，否则将显示友好的报错信息。

当您开启云平台互联功能时，您的报错信息将会上传至 84PHP 云平台并进行离线存储以备您日后查看。

由于 PHP 语言的限制，有极少数在代码运行前的报错无法被捕获。此部分报错将由 PHP 系统显示。

如果您需要对您的应用添加自定义报错，请使用：

```
Wrong::Report(String 类型的出错信息);
```

上面的报错信息仅在调试模式开启时才会显示，如果您希望在调试模式关闭时也向访客显示出错信息，请使用：

```
Wrong::Report(String 类型的出错信息, TRUE);
```

而当您希望仅输出报错文本，而不输出友好报错界面的 HTML 时，请使用：

```
Wrong::Report(String 类型的出错信息, 是否显示详细报错, TRUE);
```

完成报错显示。

详情请参见基础模块说明。

路由

出于性能考虑，我们没有将框架设计为单一入口模式。

我们强烈建议页面使用原生的 URI 形式，即?Field1=value1&...这样的形式传递参数，因为目前的搜索引擎已经能够解析 URI 参数而不影响 SEO，且用户记忆 URI 中的参数的场景极少（通常是复制粘贴）。如果您依然需要对 URI 进行美化，请通过服务器的伪静态功能中的正则表达式完成操作，即使如此，服务器的操作效率通常比 PHP 高出 14%~23%（数据来自互联网）。

安全规范

养成良好的编码习惯对于应用安全性的提升至关重要!

以下是我们推荐的安全规范:

1. 必须对所有来自外部的输入通过 Receive 中的相应模块进行安全过滤! *禁止*直接使用 php://input、$_GET、$_POST、$_COOKIE 和直接输出用户上传的非二进制文件的内容。

2. 如果需要存储用户上传文件的原始文件名，请务必对文件名进安全过滤!

3. 尽可能的使用 SESSION 而非 COOKIE 来实现用户的登录状态保持。

4. 每一个模板文件必须添加 exit;#。

5. 在生产环境下，务必关闭调试模式，以及删除 initial.php。

6. 尽量不要将用户输入的内容作为 SQL 语句中的字段操作，也就是说，如果需要操作由用户确定的字段，则应尽可能的预先定义好字段，通过数字编号代替。否则，请单独建立数据库，并建立限制权限的单独的数据库用户。

7. 请通过服务器端的伪静态设置，将非/Web 目录以外的目录设置为禁止访问。

8. 请设置对应的 HTTP403、404、500 页面，而不显示服务器默认的错误页面。

9. 请*特别关注 HTTP403 列出目录*的安全隐患!

10. 请使用 CDN/WAF 完成服务器原始 IP 地址的隐藏及安全防护，同时建议在服务器上安装安全防护软件。

11. 在非开发阶段，关闭 Linux 服务器的 22 端口。

12. 对用户上传目录取消执行权限。

版权声明&联系方式

本框架为免费开源、遵循 Apache 开源协议的框架，但不得删除此文件的版权信息，违者必究。

This framework is free and open source, following the framework of Apache open source protocol, but the copyright information of this file is not allowed to be deleted,violators will be prosecuted to the maximum extent possible.

反馈 Bug，可进入云平台提交工单，或发送邮件至 84php@8-4.cn 进行反馈。