**Phalcon 3.4**

Phalcon是一个功能强大、松耦合的、专为高性能全栈优化的全栈PHP框架

革命性的架构使得Phalcon成为了史上最快的PHP框架

1. 通用

* 简介

欢迎

欢迎来到Phalcon框架

任务

我们的任务是为开发着提供一个高级的功能强大的工具允许我们开发高性能的应用程序，但是却仅占用较少的服务器资源。

Phalcon是什么？

Phalcon是一个开源的、作为PHP的C语言扩展的全栈PHP 框架。Phalcon 转为高性能而优化，他独有的框架允许框架常驻内存，无需传统PHP框架那样昂贵的文件读取开销就能够提供他的功能。

使用

开发无需了解C语言就能使用Phalcon，Phalcon提供的功能以在Phalcon命名空间下的类来提供。

Phalcon是松耦合的，允许开发者仅仅使用那些他们需要的对象来作为组建应用的组件。

* 发布日志
* 资助
* FAQ

Phalcon是什么？

Phalcon是一个开源的以C语言扩展实现的全栈PHP框架。

Phalcon如何工作？

Phalcon不是一个PHP加速器也不是一个用PHP语言实现的项目。

PHalcon是一个使用相对低级语言级别的C语言实现其所有功能的框架，C语言扩展是将PHP代码一同编译，能够使你的应用程序更加快速同事降低应用程序的开销。

Phalcon通过一下途径实现上述功能：

* 利用本地编辑通过提供可执行的二进制代理，这些二进制代码能够被处理器直接理解和处理。因此不需要在虚拟机中执行字节码。
* 通过针对面向指定C语言编译器例如GCC/CLANG/VCC的专门的优化来减少内存占用。这些平台级别的提高了框架性能。
* Phalcon具有将变量和数据存储在栈中的能力，通常这种方式具有较高的本地访问。
* 分支预测将会更加简单，因为这些操作符是直接在用户代码级别而不是在VM的实现执行，*Mystical在Stack Overflow上面整理了很好的解释。*
* 直接访问内部结构体和方法将减少整个过程的计算量。
* 在现有的执行描述文件的基础上使用Profile Guided Optimization （PGO）进一步提高性能。

PGO是一个编译器优化技术，使用Profiling来提高程序的运行时性能。

Phalcon 依赖一系列的PHP内部设计，例如内存管理、垃圾回收以及其内部结构体。这一系列的改进对于PHP和Phalcon的性能都有着积极的效果。

1. 贡献

* 概览

为Phalcon做出贡献

Phalcon是一个高度依赖志愿者的开源的项目。我们欢迎来自全球的贡献者们。

为了是贡献过程更加简单并且影响其他人，请花一点时间阅读本文档。

跟随一下指引，将加强沟通、加速问题的解决最终促使此项目不断向前。

贡献

向Phalcon作出贡献应该从向GitHub发去Pull Reqeust开始，每一个Pull Request会被具有合并Pull Request权限的核心贡献者审核。根据Pull Request的类型和内容，这些Pull Request会被合并、挂起或者被拒绝。

问题和支持

Bug提交列表

* 在提交Bug之前先确认您当前使用Phalcon的最新版本，老版本中的Bug将不会被核心团队关注和重视。
* 当你发现Bug时，一个基本的操作时，提供中肯的信息来时Bug重现。能够重现的Bug会显著减少我们调查和修复此Bug的时间。这些信息应该通过一小段代码或者一个小程序甚至是一个失败测试的案例，请参阅[Subimt Reproducible Test](https://github.com/phalcon/cphalcon/wiki/Submit-Reproducible-Test) 获取更多信息。

生成回溯

有时候因为分段错误的原因Phalcon会使服务器进程中断，请在您的But报告中加上崩溃的调用堆栈来帮助我们查找该问题。

请参照一下指导来了解如何生成回溯。

* [Generating a gdb backtrace](https://bugs.php.net/bugs-generating-backtrace.php)
* [Generating a backtrace, with a compiler, on Win32](http://bugs.php.net/bugs-generating-backtrace-win32.php)
* [Debugging Symbols](https://github.com/oerdnj/deb.sury.org/wiki/Debugging-symbols)
* [Building PHP](http://www.phpinternalsbook.com/build_system/building_php.html)

Pull Request 检查表

* 请勿在Master分之发起Pull Request，在提交您的Pull Request 之前请在需要的分枝上拉取新分支，如果有需要，请将您的Pull Request 衍和到合适的分支。如果您的合并不够干净，您可能会被要求执行衍和操作。
* 除非您的Pull Request 计划与其他分支合并，否则请勿提交子模块的更新、composer.lock 等文件。
* 增减针对Bug修复或者新功能的适当的测试用例，参照我们测试指引以获取更多信息。
* Phalcon使用Zephir编写，请不要提交一些修改直接生成的C文件或者一些C元氏县的功能修复。
* 确保宁的PHP代码符合通用的风格以及编码规范。
* 提交之前请移除所有针对ext/kernel, \*.zep.c、\*.zep.h的更改

在提交新功能之前，请在Github上以新问题的形式打开NFR来讨论增加此功能带来的影响，或者对核心扩展的更改。一旦您的功能被批准，请确保您的PR包含一下内容：

* CHANGELOG.md
* 单元测试
* 文档或者使用案例

获得支持

如果您对使用Phalcon有任何问题，请查看[支持页面](https://phalconphp.com/support)。

希望添加新功能

如果您脑海里有一些改动或者新功能，请填写一个NFR

感谢

Phalcon团队

* 新Pull Request
* NFR
* 生成回溯
* 可在现测试
* 测试指引

1. 起步

* 安装

依赖

Phalcon需要依赖PHP来执行，他是松耦合的，被设计成允许开发者无需其他附加扩展来安装Phalcon和使用他的所有功能。一些必要的组件已经通过其他扩展来引用。举例来说，使用数据库连接需要依赖php\_pdo扩招，如果你的关系型数据库管理系统是MySql/MariaDB或者Aurora 数据库那么你将还需要依赖php\_mysqlnd扩展。同样的使用PostgreSql你讲需要php\_dgsql扩展。

硬件

Phalcon被设计成使用尽可能少的资源的同时提供较高的性能。尽管我们在各种很低的系统环境下面比如说0.25GB内存、0.5CPU测试过了Phalcon。最终硬件还是要以你的应用程序的需求来决定。

软件

* PHP >=5.5

Phalcon至少需要一下扩展来运行

* Curl
* Gettext
* Gd2（为了使用Phalcon\Image\Adapter\Gd这个类）
* Libpcre3-dev（Debian/Ubuntn），Pcre-devel（CentOS），pcre（Macos）
* Json
* Mbstring
* Pdo\_\*
* Fileinfo
* Openssl

根据您的应用可选下面扩展

* PDO
* OpenSSL
* Mbstring
* Memcache，Memcached

安装

因为Phalcon被编译成PHP扩展，所以他的安装与其他传统的PHP框架有所不同，Phalcon需要安装在服务器上面并且作为一个模块来加载。

Linux

在Linux上面安装Phalcon，您需要把我们的源装在分发包中然后在安装Phalcon。

基于BED的分发版（Debian，Ubuntu等）

安装源

将下列元安装到您的分发包：

稳定版

Curl-s https://packagecloud.io/install/repositories/phalcon/stable/script.deb.sh | sudo bash

或者

最新版

curl-s https://packagecloud.io/install/repositories/phalcon/nightly/script.deb.sh | sudo bash

安装Phalcon

在您的终端执行一下命令

PHP 5.X

sudo apt-get update

sudo apt-get install php5-phalcon

PHP 7

sudo apt-get update

sudo apt-get install php7.0-phalcon

基于RPM的分发版（CentOS，Fedora等）

安装源

稳定版

curl -s https://packagecloud.io/install/repositories/phalcon/stable/script.rpm.sh | sudo bash

最新版

curl -s https://packagecloud.io/install/repositories/phalcon/nightly/script.rpm.sh | sudo bash

安装Phalcon

在终端执行一下命令

PHP 5.X

sudo yum update

sudo yum install php56u-phalcon

PHP 7

sudo yum update

sudo yum install php70u-phalcon

其他RPM

Remi Clooet给RPM分发版维护了一个非常不错的源，你可以在[这里](https://blog.remirepo.net/pages/Config-en)找到如何启用这些。

在此之后安装Phalcon是非常简单的，就想这样：

yum install php56-php-phalcon3

FreeBSD

FreeBSD有一个可用的端口，来安装Phalcon，使用下列命令来安装：

Pkg\_add

pkg\_add -r phalcon

Source

export CFLAGS="-O2 --fvisibility=hidden"

cd /usr/ports/www/phalcon

make install clea

MacOS

在MacOS上面你可以使用brew或者macports来编译安装这个扩展或者源码。

依赖：

* PHP 5.5.x/5.6.x/7.0.x/7.1.x development resources
* XCode

Brew

因为homebrew/php标签标记为过期并且正在移除中，所以有人创建了一个自定义Phalcon源。

PHP5.6没有被导入到homebrew因此也不存在了，因此7.2被创建了出来。

brew tap tigerstrikemedia/homebrew-phalconphp

brew install php56-phalcon

brew install php70-phalcon

brew install php71-phalcon

brew install php72-phalcon

MacPorts

sudo port install php55-phalcon

sudo port install php56-phalcon

编辑您的PHP.ini文件，在末尾添加php\_phalcon.so

重启Web服务器

Windows

在Windows平台使用Phalcon你需要下载安装phalcon.dll，我们为一系列的windows平台都编译了dll，您可以在我们的[下载界面](https://phalconphp.com/en/download/windows)下载到这些dll。

源码安装略

* Web服务器设置

为了使用Phalcon应用成功工作，你需要设置您的Web服务器的重定向属性，以下为流行的Web服务器设置：

PHP-FPM

PHP-FPM（FastCGI 进程管理器）通常用来允许服务器之执行php文件，目前来说PHP-FPM已经默认打包在所有的以Linux为基础的发新版PHP中。

在Windows系统上面PHP-FPM通过PHP分发包中的php-fpm.exe来提供，你可以使用下面的脚本来帮助启动他。Windows平台不支持Unix套接字所以这个脚本将在TCP模式使用9000端口来启动fast-cgi。

创建文件php-cgi.bat文件并写入一下内容：

@ECHO OFF

ECHO Starting PHP FastCGI...

set PATH=C:\PHP;%PATH%

c:\bin\RunHiddenConsole.exe C:\PHP\php-cgi.exe -b 127.0.0.1:9000

PHP为开发人员准备的内置Web服务器

为了让你的应用在开发环境下快速的运行去来，最简单的办法就是使用PHP内置的Web服务器，请不要在生产环境下使用此服务器。

配置Phalcon

你可以不需要使用Apache或者Nginx仅通过一个路由文件.htrouter.php就可以启用动态Url重写。

如果你使用Phalcon-Devtools创建了你的应用，那么这个文件已经在路径的更目录下面，你用以使用一下命令来启动服务器：

$(which php) -S localhost:8000 -t public .htrouter.php

我们来解析一下上面的命令：

* $(which php) - 将插入你的PHP二进制文件路径
* -S localhost:8000 – 使用您提供的端口调起服务器
* -t public – 定义服务器的跟路径，对PHP必要的路径
* .htrouter.php 处理每一个请求的入口文件

到此为止在浏览器中输入<http://localhoust:8000>来检查是否工作正常。

Nginx

Nginx是一个免费的、开源的、高性能的HTTP服务器，同时也可以作为IMAP/POP3代理服务器使用。不像传统的服务器，Nginx不依赖线程来处理请求，而是更像一个可扩展的异步事件驱动的架构。这种架构将占用更小的更重要的更小的内存招用。

安装Nginx

Phalcon 配置

使用下面的配置文件在Nginx中配置Phalcon。

略

启动Nginx

Apache 也是一个众所周知的流行的多平台的Web服务区。

Phalcon配置

以下你在Apache中设置Phalcon可能使用的东西，这些笔记主要关注mod\_rewrite模块，使得能够使用更加友好的Url和路由组件。一般来说应用程序具有一下目录结构：

test/

app/

controllers/

models/

views/

public/

css/

img/

js/

index.php

文档根目录：

这是最常见的情况，应用程序在安装在跟文档根目录的任意路径下。在这种情况下，我们使用.htaccess文件将对所有的请求隐藏应用程序的真实路径。

请注意，要使用.htaccess文件要求你安装的Apache包含所有的AlowOverride的所有设置项。

<IfModule mod\_rewrite.c>

RewriteEngine on

RewriteRule ^$ public/ [L]

RewriteRule ((?s).\*) public/$1 [L]

</IfModule>

第二个.htaccess文件在public文件夹下，这将会把所有的URI订单到public/index.php文件：

# test/public/.htaccess

<IfModule mod\_rewrite.c>

RewriteEngine On

RewriteCond %{REQUEST\_FILENAME} !-d

RewriteCond %{REQUEST\_FILENAME} !-f

RewriteRule ^((?s).\*)$ index.php?\_url=/$1 [QSA,L]

</IfModule>

对用户来说在URI就是在URI参数中使用Persian字符，这在mod\_reweite中是一个问题。为了使得匹配能够在英文字符中生效你需要像下面一样更改你的.htaccess文件：

# test/public/.htaccess

<IfModule mod\_rewrite.c>

RewriteEngine On

RewriteCond %{REQUEST\_FILENAME} !-d

RewriteCond %{REQUEST\_FILENAME} !-f

RewriteRule ^([0-9A-Za-z\x7f-\xff]\*)$ index.php?params=$1 [L]

</IfModule>

如果你的URI包含一些非英文字符，你可能需要调整上面的顺序是的mod\_rewrite来精确匹配你的路由。

Apcahe配置

如果你不想使用.htaccess文件你可以把这些配置项移动到Apache的主配置文件中：

<IfModule mod\_rewrite.c>

<Directory "/var/www/test">

RewriteEngine on

RewriteRule ^$ public/ [L]

RewriteRule ((?s).\*) public/$1 [L]

</Directory>

<Directory "/var/www/test/public">

RewriteEngine On

RewriteCond %{REQUEST\_FILENAME} !-d

RewriteCond %{REQUEST\_FILENAME} !-f

RewriteRule ^((?s).\*)$ index.php?\_url=/$1 [QSA,L]

</Directory>

</IfModule>

虚拟站点

这个第二个配置中允许你将Phalcon应用程序安装在一个虚拟站点中：

<VirtualHost \*:80>

ServerAdmin admin@example.host

DocumentRoot "/var/vhosts/test/public"

DirectoryIndex index.php

ServerName example.host

ServerAlias www.example.host

<Directory "/var/vhosts/test/public">

Options All

AllowOverride All

Require all granted

</Directory>

</VirtualHost>

Cherokee

Cherokee是一个非常快、灵活的并且容易配置的高性能的Web服务器。

略

* WAMP

总揽

Wamp服务器是一个Windows下面的网页开发环境，他允许我们使用Apache2、PHP、MySQL数据库创建网页应用程序。一下是关于如何在Windows上面的WampServer上面安装Phalcon的详细介绍。最好使用最新版的WampServer。

下载正确版本的Phalcon

WAMP包含32位和64位两个版本，在下载部分你可以下载适合您安装的WAMP服务器的Phalcon DLL。

下载完成之后你将有一个压缩包，解压这个压缩包，你回看到Phalcon的DLL。

复制php\_phalcon.dll文件到PHP的扩展文件目录，比如说如果WAMP安装在C：wamp文件夹，那么这个扩展的位置需要放在C：wamp\bin\php\php5.5.12\ext（假设你安装的WAMP安装的是PHP5.5）。

编辑位于C：wamp\bin\php\php5.5.12\下面的php.ini文件。这个文件可以使用记事本或者类似的程序打开，我们推荐使用Notepad++来避免出现一些问题。在文件末尾增加下面的内容并保存：

extension=php\_phalcon.dll

同样打开C:\wamp\bin\apache\apache2.4.9\bin\php.ini这个文件并且在末尾添加一下内容然后保存：

extension=php\_phalcon.dll

重启Apache服务器之后，在系统状态栏下面淡季Wamp服务器的图标，在弹出的菜单中选择“重启所有服务“，状态栏中的小图标会在此变成绿色。

* XAMPP

1. 教程

* 基础教程

通过本次教程，我们将带你概览一遍关于创建一个带有注册表单的简单应用程序。以下教程目标是想你介绍Phalcon的设计理念。

本次教程覆盖了一个简单MVC程序的实现，展示了Phalcon是如何简单和快速的做到这一点的。本次教程将帮助起步和帮助你创建一个你可以在任何地方扩展使用的应用程序，本次教程里面的代码还可以当作学习Phalcon其他方面的训练场。

如果你只是想如何开始一个应用，你可以跳过本章，然后通过使用Phalcon-Devtools自动创建一个项目（如果当你卡住的时候并且你没有足够的经验搞定问题的时候，你最好是这样做）。

学习本章最好的方式就是一步一步跟着做并且试着保持兴趣。你还可以在这里获得本章的完整代码。如果过程中你被卡住了，请访问我们的论坛。

文件结构

Phalcon的一个主要特性是他是松耦合的，你可以以一种对你的应用来说很方便的文件目录来构建Phalcon项目。意思就是说一些一致性的标准对于与其他人合作是有帮助的。所以本教程将使用一种你很熟悉的和你之前工作过的MVC框架相似的“标准”的目录结构。

└── tutorial

├── app

│   ├── controllers

│   │   ├── IndexController.php

│   │   └── SignupController.php

│   ├── models

│   │   └── Users.php

│   └── views

└── public

├── css

├── img

├── index.php

└── js

如果这个结构对你来说是全新的，那么你最好是安装一下Phalcon Devtools因为这个工具改变了PHP内置的Web服务器，让你在不在你的文档根目录添加添加.htrouter文件的情况下是你的App运行起来。

此外如果你希望使用Nginx你可以查看[此设置向导](https://docs.phalconphp.com/en/3.4/webserver-setup#nginx)。

Apache也可以使用，这里是[设置向导](https://docs.phalconphp.com/en/3.4/webserver-setup#apache)。

最后如果你偏向于 Cherokee，你可以使用[本向导](https://docs.phalconphp.com/en/3.4/webserver-setup#cherokee)。

Bootstrap

首先你要创建的第一个文件是bootstrap文件。这个文件扮演着程序的入口点以及配置者的角色。在这个文件中你可以实现程序组件和行为的实例化。

这个文件处理以下3件事情：

* 注册自动加载的组件
* 配置Services并把他们注册到依赖注入上下文中
* 解析应用程序的HTTP请求

自动加载

自动加载器通过一个符合PSR-4规范的问价加载器，通常需要被添加到自动加载器里面的是你的控制器和模型。你可以把文件夹，然后他们就将在应用成的命名空间类自动查找并加载。如果想查看一些使用加载器的其他方式请点击[这里](https://docs.phalconphp.com/en/3.4/loader#overview)。

开始之前，我们先把我们App的控制器和模型的目录注册进去，别忘了从Phalcon\Loader命名控空间下引入加载器。

<?php

use Phalcon\Loader;

// Define some absolute path constants to aid in locating resources

define('BASE\_PATH', dirname(\_\_DIR\_\_));

define('APP\_PATH', BASE\_PATH . '/app');

// ...

$loader = new Loader();

$loader->registerDirs(

[

APP\_PATH . '/controllers/',

APP\_PATH . '/models/',

]

);

$loader->register();

依赖管理

既然Phalcon是松耦合的，服务是通过框架的依赖管理器注册的，那么这些服务可以被自动注入而且被打包在IoC容器中。你经常回听到我们依赖注入称为DI。依赖注入和控制反转听起来像是一个很复杂的东西，但是在Phalcon里面他们用起来非常简单。Phalcon的依赖反转容器有以下几个方面构成：

* 服务容器：一个贯穿整个应用程序需要的全局存储包
* 服务和组件：数据处理对象将被注入到组件

框架每次需要一个组件或者服务的时候，都会使用一个约定好的名称去尝试获取它。不要忘了包含Phalcon\Di来设置好服务容器。

默认工厂

Phalcon\Di\FactoryDefault 是一个Phalcon\Di的变体。为了让一切变得简单，他会自动注册Phalcon自带的很多组件。我们建议你手动注册你的所有服务，但是默认工厂的目标是帮你降低难度，让你快速的开始使用依赖管理。稍后，一旦你熟悉这一切之后，你可以随时指定一个新的管理器。

服务可以通过多种方式去注册，在本教程里面我们将使用匿名函数来注册。

public/index.php

<?php

use Phalcon\Di\FactoryDefault;

// ...

// Create a DI

$di = new FactoryDefault();

在下一节中，我们注册了一个View服务来知名框架可以在那个目录下找到试图文件。因为试图不一定是一个雷文件，他们无法被自动加载加载。

public/index.php

<?php

use Phalcon\Mvc\View;

// ...

// Setup the view component

$di->set(

'view',

function () {

$view = new View();

$view->setViewsDir(APP\_PATH . '/views/');

return $view;

}

);

接下来，我们注册了一个基础URI，因此所有被Phalcon生成的URI都被匹配到应用程序的“/”目录。稍后当我们使用Phalcon\Tag这个类来生成超链接的时候这将会变得重要起来。

public/index.php

<?php

use Phalcon\Mvc\Url as UrlProvider;

// ...

// Setup a base URI

$di->set(

'url',

function () {

$url = new UrlProvider();

$url->setBaseUri('/');

return $url;

}

);

处理应用程序请求

在这个文件最后一部分，我们找到了Phalcon\Mvc\Appplication。它的目的是初始化请求环境，路由收到的请求，然后分发发现的Action；当处理完成之后，他将收集返回值并返回。

public/index.php

<?php

use Phalcon\Mvc\Application;

// ...

$application = new Application($di);

$response = $application->handle();

$response->send();

汇总

tutorial/public/index.php应该像这样

public/index.php

<?php

use Phalcon\Loader;

use Phalcon\Mvc\View;

use Phalcon\Mvc\Application;

use Phalcon\Di\FactoryDefault;

use Phalcon\Mvc\Url as UrlProvider;

// Define some absolute path constants to aid in locating resources

define('BASE\_PATH', dirname(\_\_DIR\_\_));

define('APP\_PATH', BASE\_PATH . '/app');

// Register an autoloader

$loader = new Loader();

$loader->registerDirs(

[

APP\_PATH . '/controllers/',

APP\_PATH . '/models/',

]

);

$loader->register();

// Create a DI

$di = new FactoryDefault();

// Setup the view component

$di->set(

'view',

function () {

$view = new View();

$view->setViewsDir(APP\_PATH . '/views/');

return $view;

}

);

// Setup a base URI

$di->set(

'url',

function () {

$url = new UrlProvider();

$url->setBaseUri('/');

return $url;

}

);

$application = new Application($di);

try {

// Handle the request

$response = $application->handle();

$response->send();

} catch (\Exception $e) {

echo 'Exception: ', $e->getMessage();

}

你可以看到，启动问价非常简单无需包含一些附加文件。不超过30行代码你就就能配置一个灵活的MVC应用程序。

创建控制器

默认情况下，当没有在请求中指明控制器或者操作时，Phalcon将会查找一个叫做IndexController的控制器，比如说<http://localhost:8000/>。一个Index控制器和Index操作就像这样：

app/controllers/IndexController.php

<?php

use Phalcon\Mvc\Controller;

class IndexController extends Controller

{

public function indexAction()

{

echo '<h1>Hello!</h1>';

}

}

控制器的类名必须以Controller结尾，控制器方法必须以Action结尾。如果你在浏览器访问这个地址，你会看到下面的内容。

略

恭喜，Phalcon将会带你飞。

输出内容到视图

从控制器中发送内容到屏幕有时候时必要的，但是正如MVC社区里面一些纯粹主义者证明的一样这不是最合适的。任何数据必须传递到负责在屏幕上显示内容的视图。Phalcon会用最近一次执行的控制名称名称对应的文件夹下面需寻找一个以动作为文件名的视图文件。本例中（app/views/index/index.phtml）

app/views/index/index.phtml

<?php echo "<h1>Hello!</h1>";

现在我们的控制器里面定义了一个空的动作：

<?php

use Phalcon\Mvc\Controller;

class IndexController extends Controller

{

public function indexAction()

{

}

}

此时浏览器应该是输出同样的东西，因为Phalcon\Mvc\View静态组件会在动作执行结束之后自动创建。您可以在[这里](https://docs.phalconphp.com/en/3.4/views)了解到更多关于试图的使用信息。

设计一个注册表单

现在，我们将调整index.phtml文件，添加一个链接到一个名为“signup”的控制器的链接。目标是允许用户登陆在门的应用程序上注册。

app/views/index/index.phtml

<?php

echo "<h1>Hello!</h1>";

echo PHP\_EOL;

echo PHP\_EOL;

echo $this->tag->linkTo(

'signup',

'Sign Up Here!'

);

生成的HTML代码将显示一个链接到另外一个Controller的a标签：

app/views/index/index.phtml (rendered)

<h1>Hello!</h1>

<a href="/signup">Sign Up Here!</a>

我们使用Phalcon\Tag来生成标签。这个工具类允许我们构建框架预设好的HTML标签。同时这个类还可以也是一个在DI中注册过的服务，所以我们可以使用$this->tag来访问他。

[在这里](https://docs.phalconphp.com/en/3.4/tag)你可以找到更多关于渲染HTML的文章。

下面是Signup控制器：

app/controllers/SignupController.php

<?php

use Phalcon\Mvc\Controller;

class SignupController extends Controller

{

public function indexAction()

{

}

}

这个空的动作指明一条干净的道路通向我们的表单定义页面。

app/views/signup/index.phtml

<h2>Sign up using this form</h2>

<?php echo $this->tag->form("signup/register"); ?>

<p>

<label for="name">Name</label>

<?php echo $this->tag->textField("name"); ?>

</p>

<p>

<label for="email">E-Mail</label>

<?php echo $this->tag->textField("email"); ?>

</p>

<p>

<?php echo $this->tag->submitButton("Register"); ?>

</p>

</form>

在浏览器中查看这个表单大概会是这个样子：

略

Phalcon\Tag还提供了很多悠悠的方法来构建表单元素。

Phalcon\Tag::form() 方法接受一个对象参数，一个指向一个控制器\动作的一个相对URL。

通过点击“发送”按钮，你会注意到框架将抛出一个异常，表示我们的signup控制器中缺少一个register的动作。Public/index.php文件将抛出这个错误：

Exception: Action "register" was not found on handler "signup"

实现这个方法之后这个错误就会消失：

app/controllers/SignupController.php

<?php

use Phalcon\Mvc\Controller;

class SignupController extends Controller

{

public function indexAction()

{

}

public function registerAction()

{

}

}

当你在此点击“发送”按钮，你会看到一个空白页，用户在姓名和邮箱输入的内容应该存储在数据库中。根据MVC的指引手册，数据库操作必须通过模型来完成，这样我们可以保证我们的代码是完全面向对象的。

创建模型

Phalcon为PHP带来了第一个使用C语言编写的ORM框架。我们并没有增加开发的复杂度，我们甚至还简化了这个过程。

在创建我们第一个模型之前，我们需要在外部创建一个数据库表，好让Phalcon来映射他。一个简单的记录已经注册过的用户的表大概像这样：

create\_users\_table.sql

CREATE TABLE `users` (

`id` int(10) unsigned NOT NULL AUTO\_INCREMENT,

`name` varchar(70) NOT NULL,

`email` varchar(70) NOT NULL,

PRIMARY KEY (`id`)

);

模型文件应该位于app/model路径下。以下是映射到users表的模型：

app/models/Users.php

<?php

use Phalcon\Mvc\Model;

class Users extends Model

{

public $id;

public $name;

public $email;

}

设置数据库连接

为了使用通过模型来连接数据然后访问数据，我们在我们启动流程中来指定一些信息。数据库连接仅仅是我们所有可以使用的服务组件之一。

public/index.php

<?php

use Phalcon\Db\Adapter\Pdo\Mysql as DbAdapter;

// Setup the database service

$di->set(

'db',

function () {

return new DbAdapter(

[

'host' => '127.0.0.1',

'username' => 'root',

'password' => 'secret',

'dbname' => 'tutorial1',

]

);

}

);

配置了正确的数据库参数之后，我们的模型就已经做好了与数据库打交道的准备。

使用模型存储数据

app/controllers/SignupController.php

<?php

use Phalcon\Mvc\Controller;

class SignupController extends Controller

{

public function indexAction()

{

}

public function registerAction()

{

$user = new Users();

// Store and check for errors

$success = $user->save(

$this->request->getPost(),

[

"name",

"email",

]

);

if ($success) {

echo "Thanks for registering!";

} else {

echo "Sorry, the following problems were generated: ";

$messages = $user->getMessages();

foreach ($messages as $message) {

echo $message->getMessage(), "<br/>";

}

}

$this->view->disable();

}

}

在注册动作的开头，我门从Users类，创建了一个管理User表记录的空的Users对象。类的公开属性对应了数据Users表的字段。设置这个对象的相关属性，然后调用Save()方法就会爆这条数据保存在数据库中。Save()方法返回一个布尔值代码数据是否保存成功。

Phalcon的ORM框架会自动处理输入太防止SQL注入，所以我们只需要将输入直接传入input方法里面就可以了。

设置为非空的字段验证附加的验证会自动执行。如果我们不输入任何内容那么页面将显示错误信息：

略

结论

你可以看到，使用Phalcon构建应用程序是非常简单的。事实上，Phalcon运行在一个能够显示降低项目足迹的扩展下面，同时能提供显著的性能提升。

如果你已经准备好了，可以点击[下一节教程Rest](https://docs.phalconphp.com/en/3.4/tutorial-rest)来了解更多 。

* REST

教程：创建一个简单的REST API

在本章教程，我们将介绍如何创建一个简单的使用不同HTTM方法来提供RESTful风格的Api应用程序。

* GET 获取和搜索数据
* POST 添加数据
* PUT 更新数据
* DELETE 删除数据

定义API

API将有一下方法构成：

| **Method** | **URL** | **Action** |
| --- | --- | --- |
| GET | /api/robots | Retrieves all robots |
| GET | /api/robots/search/Astro | Searches for robots with 'Astro' in their name |
| GET | /api/robots/2 | Retrieves robots based on primary key |
| POST | /api/robots | Adds a new robot |
| PUT | /api/robots/2 | Updates robots based on primary key |
| DELETE | /api/robots/2 | Deletes robots based on primary key |

创建应用程序

因为这个程序。分简单，我们不会实现全部的MVC环境来开发它。这个例子中，我们将使用一个[微程序](https://docs.phalconphp.com/en/3.4/application-micro)来实现我们的目标；

下面的结构将就足够了：

my-rest-api/

models/

Robots.php

index.php

.htaccess

首先我们需要一个包含到达所有重定向规则的 .htaccess的文件来重定向到达index.php文件的请求（也就是应用程序的入口点）。

<IfModule mod\_rewrite.c>

RewriteEngine On

RewriteCond %{REQUEST\_FILENAME} !-f

RewriteRule ^((?s).\*)$ index.php?\_url=/$1 [QSA,L]

</IfModule>

我们大多数的代码会写在index.php文件里面。这个文件创建出来应该是这个样子：

<?php

use Phalcon\Mvc\Micro;

$app = new Micro();

// Define the routes here

$app->handle();

接下来我们需要创建路由下面的路由：

<?php

use Phalcon\Mvc\Micro;

$app = new Micro();

// Retrieves all robots

$app->get(

'/api/robots',

function () {

// Operation to fetch all the robots

}

);

// Searches for robots with $name in their name

$app->get(

'/api/robots/search/{name}',

function ($name) {

// Operation to fetch robot with name $name

}

);

// Retrieves robots based on primary key

$app->get(

'/api/robots/{id:[0-9]+}',

function ($id) {

// Operation to fetch robot with id $id

}

);

// Adds a new robot

$app->post(

'/api/robots',

function () {

// Operation to create a fresh robot

}

);

// Updates robots based on primary key

$app->put(

'/api/robots/{id:[0-9]+}',

function ($id) {

// Operation to update a robot with id $id

}

);

// Deletes robots based on primary key

$app->delete(

'/api/robots/{id:[0-9]+}',

function ($id) {

// Operation to delete the robot with id $id

}

);

$app->handle();

每个路由的名称定义都对应了一个HTTP方法，作为第一个参数，我们传递了一个路由模式，后面更这一个处理程序。在这个例子中，我们传入的处理程序是一个匿名函数。下面这个路由/api/robots/{id:[0-9]+}里面，希望接受一个必须为整数的id参数。

当路由规则匹配了当前请求URI之后程序就会执行相应的执行程序。

创建模型

我们的API提供了关于robots的信息，这些信息被保存在数据库中。下面的模型，允许我们以面向对象的方式访问数据库表。我们使用内置的验证器和简单验证器实现了一些业务规则。通过这种方式我们可以安心的保存数据。模型文件应该放在Models下面。

<?php

namespace Store\Toys;

use Phalcon\Mvc\Model;

use Phalcon\Mvc\Model\Message;

use Phalcon\Mvc\Model\Validator\Uniqueness;

use Phalcon\Mvc\Model\Validator\InclusionIn;

class Robots extends Model

{

public function validation()

{

// Type must be: droid, mechanical or virtual

$this->validate(

new InclusionIn(

[

'field' => 'type',

'domain' => [

'droid',

'mechanical',

'virtual',

],

]

)

);

// Robot name must be unique

$this->validate(

new Uniqueness(

[

'field' => 'name',

'message' => 'The robot name must be unique',

]

)

);

// Year cannot be less than zero

if ($this->year < 0) {

$this->appendMessage(

new Message('The year cannot be less than zero')

);

}

// Check if any messages have been produced

if ($this->validationHasFailed() === true) {

return false;

}

}

}

接下来我们必须设置一个模型需要使用的连接，然后在我们的App文件中加载它。

<?php

use Phalcon\Loader;

use Phalcon\Mvc\Micro;

use Phalcon\Di\FactoryDefault;

use Phalcon\Db\Adapter\Pdo\Mysql as PdoMysql;

// Use Loader() to autoload our model

$loader = new Loader();

$loader->registerNamespaces(

[

'Store\Toys' => \_\_DIR\_\_ . '/models/',

]

);

$loader->register();

$di = new FactoryDefault();

// Set up the database service

$di->set(

'db',

function () {

return new PdoMysql(

[

'host' => 'localhost',

'username' => 'asimov',

'password' => 'zeroth',

'dbname' => 'robotics',

]

);

}

);

// Create and bind the DI to the application

$app = new Micro($di);

获取数据

在我们将要实现的第一个Handler是通过GET方法返回所有可用的robots。让我们使用PHQL来执行这个简单查询用JSON格式返回：

<?php

// Retrieves all robots

$app->get(

'/api/robots',

function () use ($app) {

$phql = 'SELECT \* FROM Store\Toys\Robots ORDER BY name';

$robots = $app->modelsManager->executeQuery($phql);

$data = [];

foreach ($robots as $robot) {

$data[] = [

'id' => $robot->id,

'name' => $robot->name,

];

}

echo json\_encode($data);

}

);

PHQL可以让我们以一种高级的、面向对象的方式书写查询，然后在其内部按照不通的目标数据库系统转换成正确的SQL语句。其中匿名方法中use子句允许我们很方便的把全局的变量传递到本地区域。

按名称查询的处理程序会是这样：

<?php

// Searches for robots with $name in their name

$app->get(

'/api/robots/search/{name}',

function ($name) use ($app) {

$phql = 'SELECT \* FROM Store\Toys\Robots WHERE name LIKE :name: ORDER BY name';

$robots = $app->modelsManager->executeQuery(

$phql,

[

'name' => '%' . $name . '%'

]

);

$data = [];

foreach ($robots as $robot) {

$data[] = [

'id' => $robot->id,

'name' => $robot->name,

];

}

echo json\_encode($data);

}

);

按照ID字段查询也是一样，在这个例子中我们还注意到，我们还能知道robots是否找到。

<?php

use Phalcon\Http\Response;

// Retrieves robots based on primary key

$app->get(

'/api/robots/{id:[0-9]+}',

function ($id) use ($app) {

$phql = 'SELECT \* FROM Store\Toys\Robots WHERE id = :id:';

$robot = $app->modelsManager->executeQuery(

$phql,

[

'id' => $id,

]

)->getFirst();

// Create a response

$response = new Response();

if ($robot === false) {

$response->setJsonContent(

[

'status' => 'NOT-FOUND'

]

);

} else {

$response->setJsonContent(

[

'status' => 'FOUND',

'data' => [

'id' => $robot->id,

'name' => $robot->name

]

]

);

}

return $response;

}

);

插入数据

我们还可以将请求体中传入的数据插入到数据库中，我们也可以使用PHQL来执行这个插入操作。

<?php

use Phalcon\Http\Response;

// Adds a new robot

$app->post(

'/api/robots',

function () use ($app) {

$robot = $app->request->getJsonRawBody();

$phql = 'INSERT INTO Store\Toys\Robots (name, type, year) VALUES (:name:, :type:, :year:)';

$status = $app->modelsManager->executeQuery(

$phql,

[

'name' => $robot->name,

'type' => $robot->type,

'year' => $robot->year,

]

);

// Create a response

$response = new Response();

// Check if the insertion was successful

if ($status->success() === true) {

// Change the HTTP status

$response->setStatusCode(201, 'Created');

$robot->id = $status->getModel()->id;

$response->setJsonContent(

[

'status' => 'OK',

'data' => $robot,

]

);

} else {

// Change the HTTP status

$response->setStatusCode(409, 'Conflict');

// Send errors to the client

$errors = [];

foreach ($status->getMessages() as $message) {

$errors[] = $message->getMessage();

}

$response->setJsonContent(

[

'status' => 'ERROR',

'messages' => $errors,

]

);

}

return $response;

}

);

更新数据

数据更新和插入差不多，传入的ID参数指明了需要更新的数据是哪一条。

<?php

use Phalcon\Http\Response;

// Updates robots based on primary key

$app->put(

'/api/robots/{id:[0-9]+}',

function ($id) use ($app) {

$robot = $app->request->getJsonRawBody();

$phql = 'UPDATE Store\Toys\Robots SET name = :name:, type = :type:, year = :year: WHERE id = :id:';

$status = $app->modelsManager->executeQuery(

$phql,

[

'id' => $id,

'name' => $robot->name,

'type' => $robot->type,

'year' => $robot->year,

]

);

// Create a response

$response = new Response();

// Check if the insertion was successful

if ($status->success() === true) {

$response->setJsonContent(

[

'status' => 'OK'

]

);

} else {

// Change the HTTP status

$response->setStatusCode(409, 'Conflict');

$errors = [];

foreach ($status->getMessages() as $message) {

$errors[] = $message->getMessage();

}

$response->setJsonContent(

[

'status' => 'ERROR',

'messages' => $errors,

]

);

}

return $response;

}

);

删除数据

删除操作与更新操作类似，传入的ID参数指明需要删除的是哪一条数据。

<?php

use Phalcon\Http\Response;

// Deletes robots based on primary key

$app->delete(

'/api/robots/{id:[0-9]+}',

function ($id) use ($app) {

$phql = 'DELETE FROM Store\Toys\Robots WHERE id = :id:';

$status = $app->modelsManager->executeQuery(

$phql,

[

'id' => $id,

]

);

// Create a response

$response = new Response();

if ($status->success() === true) {

$response->setJsonContent(

[

'status' => 'OK'

]

);

} else {

// Change the HTTP status

$response->setStatusCode(409, 'Conflict');

$errors = [];

foreach ($status->getMessages() as $message) {

$errors[] = $message->getMessage();

}

$response->setJsonContent(

[

'status' => 'ERROR',

'messages' => $errors,

]

);

}

return $response;

}

);

测试我们应用程序

我们将使用curl来测试我们程序的每一个路由，看他们是否工作良好。

略

结论

我们可以看到，使用Phalcon的micro组件和PHQL来创建一个RESTful的Api是十分简单的。

* Vokuro

Vokuro是另外一个你可以用来了解Phalcon的简单的程序。Vokuro是一个了解如何实现安全性功能和管用用户以及用户权限的小网站。你可以在[GitHub](https://github.com/phalcon/vokuro)上面克隆它的代码。

项目结构

vokuro/

app/

config/

controllers/

forms/

library/

models/

views/

cache/

public/

css/

img/

schemas/

这个项目按照一个非常类似于INVO的目录结构组织文件，当你在浏览器中打开<http://localhost/vokuro>的时候将会显示一下内容：

略

这个程序分为两部分，前台允许客户注册服务，后台允许管理员管理已经注册的会员。前后端被整合在一个模块中。

依赖与类加载

本项目使用Phalcon\Loader来加载控制器、模型、表单等。同时使用composer来加载项目依赖。所以为了运行Vokuro首先需要使用Composer安装它的依赖。如果正确安装之后，在控制台键入以下命令：

Cd vokuro

Composer install

Vokuro使用Swift发送邮件到用户来确认注册信息。Composer.json文件应该像这样：

{

"require" : {

"php": ">=5.5.0",

"ext-phalcon": ">=3.0.0",

"swiftmailer/swiftmailer": "^5.4",

"amazonwebservices/aws-sdk-for-php": "~1.0"

}

}

到目前为止，在app\config目录下会有一个loader.php，所有的自动加载设置都将在这个文件中完成。在这个文件的末尾你可以看到composer的自动加载器已经被包含，以此来将下载好的依赖全部加载进来。

<php

//Use compser autoloader to load verdor class

Require\_once BASE\_PATH. ‘vender/autoload.php’

此外，Vokuro与INVO不同，它为控制器和模型使用名称空间，这是构建项目的推荐做法。这样的话，自动装载器看起来和我们之前看到的有点不同。

<?php

use Phalcon\Loader;

$loader = new Loader();

$loader->registerNamespaces(

[

'Vokuro\Models' => $config->application->modelsDir,

'Vokuro\Controllers' => $config->application->controllersDir,

'Vokuro\Forms' => $config->application->formsDir,

'Vokuro' => $config->application->libraryDir,

]

);

$loader->register();

注册

首先，我们先检测一下用户是否已经注册。当用户点击创建账号的按钮的时候，控制器SessionController被调用，然后signup动作被执行：

<?php

namespace Vokuro\Controllers;

use Vokuro\Forms\SignUpForm;

class SessionController extends ControllerBase

{

public function signupAction()

{

$form = new SignUpForm();

// ...

$this->view->form = $form;

}

}

这个动作仅仅把一个SignUpForm的实例传到了视图，然后就能被渲染出一个表单让用户输入登录的相关信息：

{{ form('class': 'form-search') }}

<h2>

Sign Up

</h2>

<p>{{ form.label('name') }}</p>

<p>

{{ form.render('name') }}

{{ form.messages('name') }}

</p>

<p>{{ form.label('email') }}</p>

<p>

{{ form.render('email') }}

{{ form.messages('email') }}

</p>

<p>{{ form.label('password') }}</p>

<p>

{{ form.render('password') }}

{{ form.messages('password') }}

</p>

<p>{{ form.label('confirmPassword') }}</p>

<p>

{{ form.render('confirmPassword') }}

{{ form.messages('confirmPassword') }}

</p>

<p>

{{ form.render('terms') }} {{ form.label('terms') }}

{{ form.messages('terms') }}

</p>

<p>{{ form.render('Sign Up') }}</p>

{{ form.render('csrf', ['value': security.getToken()]) }}

{{ form.messages('csrf') }}

<hr>

{{ endForm() }}

* 性能

提高性能

想要获得更高的性能有很多方面的要求：服务器、客户端、网络、数据库、网页服务器、静态资源等等。在本章中，我们着重分析了一些你能提高性能的地方，同时也指出了如何找出你的应用程序中真正运行缓慢的地方。

在服务器上面执行分析

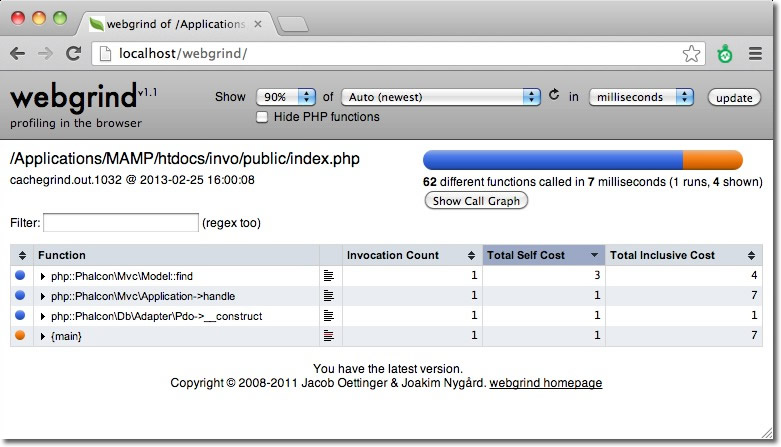
每个程序都是不同的，为了找到程序能够被优化的地方，一个恒定的分析是非常重要的。执行分析能够给我们一个清晰的画面展示，那些地方是真正缓慢的而那些地方是不缓慢的。分析在不同请求下面可能是不同的，所以在下定论之前一定要执行足够多的测量。

使用XDebug来分析

XDebug提供了一个更加简单的方式来分析PHP程序，你只需要安装扩展，然后在php.ini文件中启用分析即可。

Xdebug.profiler\_enabl = on

使用蕾西Webgrind 你可以看到那些方法执行的比其他方法更慢：



使用Xhprof分析

Xhprof是另外一个有趣的用来分析PHP程序的扩展，在启动文件的开头一行中增加下面这一行：

Xhprof\_enabl(XHPORF\_FLAGS\_CPU + XHPROF\_FLAGS\_MEMORY)

然后在文件结尾保存分析结果数据

<?php

$xhprof\_data = xhprof\_disable('/tmp');

$XHPROF\_ROOT = '/var/www/xhprof/';

include\_once $XHPROF\_ROOT . '/xhprof\_lib/utils/xhprof\_lib.php';

include\_once $XHPROF\_ROOT . '/xhprof\_lib/utils/xhprof\_runs.php';

$xhprof\_runs = new XHProfRuns\_Default();

$run\_id = $xhprof\_runs->save\_run($xhprof\_data, 'xhprof\_testing');

echo "http://localhost/xhprof/xhprof\_html/index.php?run={$run\_id}&source=xhprof\_testing\n";

Xhporf提供了一个哪住的HTML浏览器来分析结束数据：

略

分析SQL语句

大多数数据库系统都提供了定位SQL慢查询语句的工具，检测和修复慢查询在提高程序性能方面是十分重要的。通过分析慢查询你可以知道那些时间比实际预料的时间要长。

Log-slow-queries= /ver/log/slow-querise.log

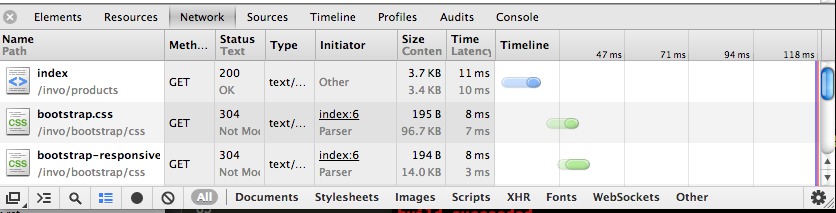
Long\_query\_time = 1.5

分析客户端

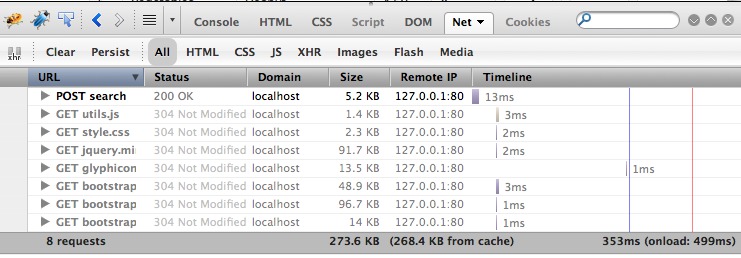
有些时候我们需要提高加载类似图片、javascript、css等这些静态资源的速度。下面一些工具在检测在检测前端瓶颈方面很有用：

使用Chrome/Firefox分析

很多现代的浏览器都带有检测页面加载时间的工具，在Chrome中我们使用Web指示器来知晓一个页面中要求的各种资源的加载时间：

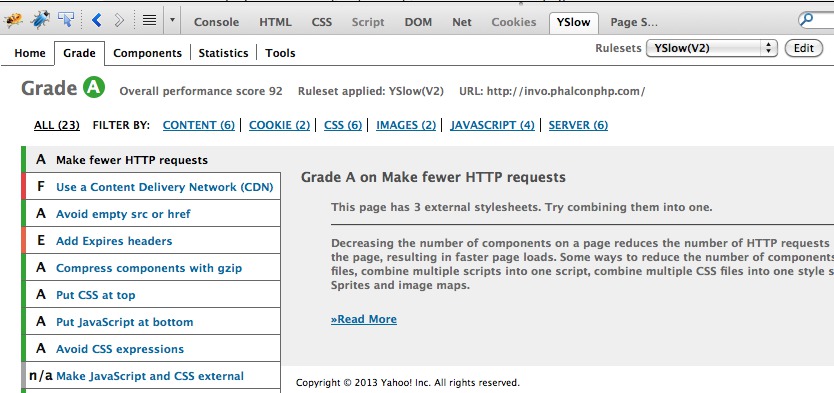


Firebug也提供了相似的功能：



雅虎的YSlow

YSlow遵守“高性能Web应用程序规范”来分析Web页面，然后给出提高性能的一些建议。



使用SpeedTrace分析

SeedTracer是一个用来帮助我们定位和解决Web应用程序当中的性能问题。在你程序运行的同时，这个工具可以把Web程序中一些零散的数据以可视化图表的方式展现。SpeedTracer可以在任何支持Chrome扩展的平台上面使用。

这个工具非常有用，因为他可以帮你分析加载整个HTML页面的过程中，HTML渲染、Javascript执行、CSS样式应用分别真实的耗费了多少时间。

使用最新版本的PHP

PHP每天都在变快，因此使用最新版本的PHP能够提高你的程序性能，Phalcon也是一样。

使用PHP 二进制缓存

APC想起他二进制代码缓存一样，也是一个用来降低每一次请求中读取、解析PHP文件的时间。当你安装这个扩展之后，你可以使用一下设置来启用APC：

Apc.enabled = on

将耗时较多的工作放在后台执行

解析视屏、发送邮件、压缩文件或者图片这些工作因为过于缓慢，因此必须放在后台任务中执行。 现在有很多工具提供了与PHP功能良好的队列或者消息系统。

Beanstalkd

Redis

RabbitMQ

Resque

Gearman

ZeroMQ

Google Page Speed

mod\_pagespeed你的网站速度和减少页面加载时间。这个开源的Apache HTTP服务器模块(nginx也可以用ngx\_pagespeed的名字)自动将web性能最佳实践应用到页面和相关资产(CSS、JavaScript、图像)上，而不需要修改现有的内容或工作流。

* Invo

INVO教程

在这第二个教程中，我们将解释一个更加完成的应用程序来进一步深入的理解如何使用Phalcon开发。INVO是一个我们创建的另一个应用程序。它是一个允许用户创建发票或者管理商品和客户的小网站。你可以在此处克隆[代码](https://github.com/phalcon/invo)。

INVO使用前端框架来创建，尽管他无法创建真实的发票，但他仍然可以作为一个展示框架如何工作的例子。

项目结构

当你可克隆他的代码到你的王党根目录下面之后，你将看到下面的结构。

invo/

app/

config/

controllers/

forms/

library/

logs/

models/

plugins/

views/

cache/

volt/

docs/

public/

css/

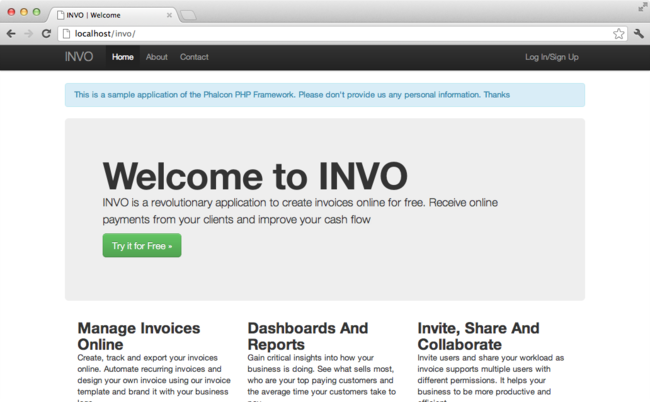
fonts/

js/

schemas/

正如你所知，开发Phalcon应用程序不强求一个特定的文件结构。这个项目拥有一个简单的MVC结构以及一个公开的文档路径。

当你在浏览器中打开：<http://localhost/invo>的时候，你会看到胰腺癌内容：



这个程序分为前台和后台两部分。前端是一块公开的部分，游客可以在公开部门获取到关于INVO的信息和请求获取联系人信息。后台是只有具有管理员权限的能够访问的区域，注册的用户可以管理他们的商品和客户。

路由

INVO使用内置的Router组件定义的提供的标准组件。这些路由匹配一下规则：/:controller/:action/:params。这意味着URI的前一部分是控制器，第二部分是动作，剩下的是参数。

/session/register将执行SessionController里面的registerAction.

配置

INVO有一个包含应用程序所有通用配置的文件配置文件。这个文件位于app/congif/config.ini，这个文件在应用程序引导文件的最开始的几行被加载：

<?php

use Phalcon\Config\Adapter\Ini as ConfigIni;

// ...

// Read the configuration

$config = new ConfigIni(

APP\_PATH . 'app/config/config.ini'

);

Phalcon配置（Phalcon\Config）允许我们已面向对象的方式操作配置文件。举例来说，我们使用ini文件来配置项目，但是Phalcon有用一些列适配器（adapters）来访问其他文件类型的配置文件。本例的配置文件包含一下设置项目：

[database]

host = localhost

username = root

password = secret

name = invo

[application]

controllersDir = app/controllers/

modelsDir = app/models/

viewsDir = app/views/

pluginsDir = app/plugins/

formsDir = app/forms/

libraryDir = app/library/

baseUri = /invo/

Phalcon没有任何预定义的配置习惯。配置节帮助我们适当的组织配置项目。稍后我们将使用这个文件中Application和Database两个配置节/

自动加载

引导文件的第二部分出现的是自动加载部分：

<?php

/\*\*

\* Auto-loader configuration

\*/

require APP\_PATH . 'app/config/loader.php';

自动加载注册了一系列文件路径，应用程序将在这些路径下寻找我们常用的类文件。

<?php

$loader = new Phalcon\Loader();

// We're a registering a set of directories taken from the configuration file

$loader->registerDirs(

[

APP\_PATH . $config->application->controllersDir,

APP\_PATH . $config->application->pluginsDir,

APP\_PATH . $config->application->libraryDir,

APP\_PATH . $config->application->modelsDir,

APP\_PATH . $config->application->formsDir,

]

);

$loader->register();

注意到，上面的代码注册了我们在配置文件中定义的文件路径，我们唯一没有注册时的viewDir，因为这个目录下包含了HTML和PHP文件但是没有类。此外，还要注意到我们使用了一个名为APP\_PATH的常量，这个常量在应用程序引导文件中定义，来让我们拿到一个指向我们项目根目录的地址：

<?php

// ...

define(

'APP\_PATH',

realpath('..') . '/'

);

服务注册包含在一些闭包中，以完成必须组件的懒加载：

<?php

use Phalcon\Mvc\Url as UrlProvider;

// ...

/\*\*

\* The URL component is used to generate all kind of URLs in the application

\*/

$di->set(

'url',

function () use ($config) {

$url = new UrlProvider();

$url->setBaseUri(

$config->application->baseUri

);

return $url;

}

);

我们稍后会深入的讨论这个文件。

处理请求

如果我们直接跳到public/index.php的最末尾，我们可以发现所有请求最终被Phalcon\MVC\Application处理，这个类将初始化和执行使程序运行起来的一切。

<?php

use Phalcon\Mvc\Application;

// ...

$application = new Application($di);

$response = $application->handle();

$response->send();

依赖注入

上面代码的第一行，Application类的构造器收到了一个$di变量作为参数。这个变量的目的是什么呢？Phalcon是一个高度解耦合的框架，所以我们需要一个扮演胶水角色的组件来让一切合力工作。这个组件就是Phalcon\Di。他也是一个服务容器，同时完成了依赖注入以及服务定位，实例化所有程序需要的组件等任务。

在容器中有很多种注册服务的方法。在INVO中，大多数服务已经通过匿名函数或者闭包注册过了。多亏了这种方法，对象都是以一种懒加载的方式实例化出来，因此降低了程序所需要的资源。

举例来说，下面的代码希望session服务被提前注册。这个匿名函数只有在程序需要访问session数据的时候才会被调用：

<?php

use Phalcon\Session\Adapter\Files as Session;

// ...

// Start the session the first time a component requests the session service

$di->set(

'session',

function () {

$session = new Session();

$session->start();

return $session;

}

);

到此，我们可以自由的改变适配器来执行附加的实例化操作。清除一，服务是使用session这个名次来注册进去的。这将使得，框架在服务容器中定位服务变得方便。

一次请求kennel会使用到很多服务，单独注册每一个服务将是一个笨重的任务。因此，框架提供了一个Phalcon\Di的变体，叫做Phalcon\Di\FactoryDefault，这个文件的目的就是注册所有的服务来提供一个全栈的框架。

<?php

use Phalcon\Di\FactoryDefault;

// ...

// The FactoryDefault Dependency Injector automatically registers the

// right services providing a full-stack framework

$di = new FactoryDefault();

这个文件作为标准注册了框架提供的大多数服务。如果我们需要重写一些服务的定义我们只需要在di中在设置一次，就像我们上面设置session和url一样。这就是$di这个变量存在的意义。

应用程序日志记录

一个日志功能将允许我们在控制器内操作。后端控制器和前端控制器的分离是合理的。所有的控制都唯一同一个路径中（app/controller）.

用户必须要用一个合法的用户名和密码才能进入系统。用户信息是被存在invo数据库的users表中。

在开始一个会话之前，我们需要在应用程序中配置数据库连接。一个带有数据库连接的名为db的服务被设置到服务容器内部。通过自动加载，我们再次把参数从配置文件中提取出来来配置此服务：

<?php

use Phalcon\Db\Adapter\Pdo\Mysql as DbAdapter;

// ...

// Database connection is created based on parameters defined in the configuration file

$di->set(

'db',

function () use ($config) {

return new DbAdapter(

[

'host' => $config->database->host,

'username' => $config->database->username,

'password' => $config->database->password,

'dbname' => $config->database->name,

]

);

}

);

这里，我们返回了一个MySQL连接的是配置。如有有需要，你可以做一些附加操作，比如说加入一个日志记录器，或者按照你的需要改变这个适配器。

下面的是一个来自（app/view/session/index.volt）例子需要登录信息。我们移除了一些HTML代码让例子更简洁。

{{ form('session/start') }}

<fieldset>

<div>

<label for='email'>

Username/Email

</label>

<div>

{{ text\_field('email') }}

</div>

</div>

<div>

<label for='password'>

Password

</label>

<div>

{{ password\_field('password') }}

</div>

</div>

<div>

{{ submit\_button('Login') }}

</div>

</fieldset>

{{ endForm() }}

我们开始使用Volt而不是像之前一样使用原始的php代码。这是一个内置的模版引擎，这个引擎收到Jinja\_启发。提供更简单又好的语法来创建模版。开始使用Volt不会需要太多时间。

SessionController里面的startAction方法完成了对来自表单提交的数据的验证，包括验证用户名是否存在于数据库：

<?php

class SessionController extends ControllerBase

{

// ...

private function \_registerSession($user)

{

$this->session->set(

'auth',

[

'id' => $user->id,

'name' => $user->name,

]

);

}

/\*\*

\* This action authenticate and logs a user into the application

\*/

public function startAction()

{

if ($this->request->isPost()) {

// Get the data from the user

$email = $this->request->getPost('email');

$password = $this->request->getPost('password');

// Find the user in the database

$user = Users::findFirst(

[

"(email = :email: OR username = :email:) AND password = :password: AND active = 'Y'",

'bind' => [

'email' => $email,

'password' => sha1($password),

]

]

);

if ($user !== false) {

$this->\_registerSession($user);

$this->flash->success(

'Welcome ' . $user->name

);

// Forward to the 'invoices' controller if the user is valid

return $this->dispatcher->forward(

[

'controller' => 'invoices',

'action' => 'index',

]

);

}

$this->flash->error(

'Wrong email/password'

);

}

// Forward to the login form again

return $this->dispatcher->forward(

[

'controller' => 'session',

'action' => 'index',

]

);

}

}

为了简便的目的，我们使用sha1来将密码hash之后存储在数据库中。但是在实际的应用程序中不推荐使用这个算法，请使用bcrypt代替。

注意观察这个控制器里面涉及到的一些公用的属性，这些都是我们先前注册到服务容器里面的服务。当我们首次访问这个属性时，他们会作为控制器的一部分注入到控制器内。这个服务时共享的，这意味着不管我们在哪里访问都是同一个对象。比如说，我们在这里访问绘画服务然后我们把用户标志存储在auth变量中：

<?php

$this->session->set(

'auth',

[

'id' => $user->id,

'name' => $user->name,

]

);

这一章另一个重要的方面是，用户是否有效是如何验证的，首先我们验证了请求是否是通过POST方式提交过来的。

<?php

if ($this->request->isPost()) {

// ...

}

然后我们收到了从表单中提交的参数：

<?php

$email = $this->request->getPost('email');

$password = $this->request->getPost('password');

然后我们需要检查数据库里面是否有相同的邮箱和密码：

略

请注意，这些称之为绑定参数是一些真实参数的占位符。这样将安全的将一些非法的字段名称移除，避免SQL注入的风险。

如果用户是有效的，那么用户将被引导到仪表板。

如果用户不存在，我们会将用户引导到先前的注册页面。

后台的安全保障

后台是一块只有注册用户才能访问的私有区域，因此，检测只有注册用户才能访问这些控制器是有必要的。如果你没有登录进系统，那么你见看到下面的内容。

略

在每次有人请求任何控制器和动作的时候，应用程序将在session验证，是否能够访问，否则它将显示一条消息，然后将流程退回到首页。

以下内容未完成。

1. 环境

* Phalcon构成（Docker）
* Nanobox
* Phalcon Box（Vagrant）

1. 开发工具

* 安装

安装Phalcon开发工具

以下步骤将带领你完成在各种平台安装Phalcon开发工具

前期准备

运行Phalcon开发工具需要安装Phalcon扩展。

安装

你可以在我们的下载界面下载到，包含开发工具的跨平台的包。

Linux

打开终端键入下面的命令：

git clone git://github.com/phalcon/phalcon-devtools.git

然后进入指定文件夹，然后

* 使用
* 调试

1. 核心
2. 服务
3. 数据库
4. 前端
5. 业务逻辑
6. HTTP
7. 安全性
8. i18n