PHP 7.4 中的预加载

作者：henryspace
原文链接：https://ld246.com/article/1618466404753
来源网站：链滴

在PHP 7.4中，添加了对预加载的支持，这是一个可以显著提高代码性能的特性。

简而言之，这是它的工作方式：

- 为了预加载文件，您需要编写一个自定义PHP脚本
- 该脚本在服务器启动时执行一次
- 所有预加载的文件在内存中都可用于所有请求
- 在重新启动服务器之前，对预加载文件所做的更改不会产生任何影响

让我们深入了解它。

###Opcache

虽然预加载是建立在opcache之上的，但它并不是完全一样的。Opcache将获取您的PHP源文件，将编译为“ opcodes”，然后将这些编译后的文件存储在磁盘上。

您可以将操作码看作是代码的底层表示，在运行时很容易解释。因此，opcache会跳过源文件和PHP释器在运行时实际需要之间的转换步骤。巨大的胜利！

但我们还有更多的收获。Opcached文件不知道其他文件。如果类a是从类B扩展而来的，那么仍然需在运行时将它们链接在一起。此外，opcache执行检查以查看源文件是否被修改，并将基于此使其缓失效。

因此，这就是预加载发挥作用的地方:它不仅将源文件编译为操作码，而且还将相关的类、特征和接口接在一起。然后，它将这个“已编译”的可运行代码blob(即:PHP解释器可以使用的代码)保存在内存。

现在，当请求到达服务器时，它可以使用已经加载到内存中的部分代码库，而不会产生任何开销。

那么，我们所说的“代码库的一部分”是什么呢?

###实践中的预加载

为了进行预加载，开发人员必须告知服务器要加载哪些文件。这是用一个简单的PHP脚本完成的，确没有什么困难。

规则很简单：

- 您提供一个预加载脚本，并使用opcache.preload命令将其链接到您的php.ini文件中。
- 您要预加载的每个PHP文件都应该传递到opcache_compile_file()，或者在预加载脚本中只需要一。

假设您想要预加载一个框架，例如Laravel。您的脚本必须遍历vendor/laravel目录中的所有PHP文件并将它们一个接一个地添加。

在php.ini中链接到此脚本的方法如下：

opcache.preload=/path/to/project/preload.php

这是一个虚拟的实现：

```
$files = /* An array of files you want to preload */;

foreach ($files as $file) {

    opcache_compile_file($file);

}
```

＃警告：无法预加载未链接的类

等等，有一个警告！为了预加载文件，还必须预加载它们的依赖项（接口，特征和父类）。

如果类依赖项有任何问题，则会在服务器启动时通知您：

```
Can't preload unlinked class

Illuminate\Database\Query\JoinClause:

Unknown parent

Illuminate\Database\Query\Builder
```

看，opcache_compile_file()将解析一个文件，但不执行它。这意味着如果一个类有未预加载的依赖，它本身也不能预加载。

这不是一个致命的问题，您的服务器可以正常工作。但你不会得到所有你想要的预加载文件。

幸运的是，还有一种确保链接文件也被加载的方法：您可以使用require_once代替opcache_compile_ile，让已注册的autoloader(可能是composer的)负责其余的工作。

```
$files = /* All files in eg. vendor/laravel */;

foreach ($files as $file) {

    require_once($file);

}
```

还有一些需要注意的地方。例如，如果您试图预加载Laravel，那么框架中的一些类依赖于其他尚不存的类。例如，文件系统缓存类\ lighting \ filesystem \ cache依赖于\League\Flysystem\Cached\Stoage\AbstractCache，如果您从未使用过文件系统缓存，则可能无法将其安装到您的项目中。

尝试预加载所有内容时，您可能会遇到“class not found”错误。幸运的是，在默认的Laravel安装，只有少数这些类，可以轻易忽略。为了方便起见，我编写了一个小小的preloader类，以使忽略文更容易，如下所示：

```
class Preloader

{

    private array $ignores = [];
```

```
private static int $count = 0;
private array $paths;
private array $fileMap;
public function __construct(string ...$paths)
{
    $this->paths = $paths;
    // We'll use composer's classmap
    // to easily find which classes to autoload,
    // based on their filename
    $classMap = require __DIR__ . '/vendor/composer/autoload_classmap.php';
    $this->fileMap = array_flip($classMap);
}

public function paths(string ...$paths): Preloader
{
    $this->paths = array_merge(
        $this->paths,
        $paths
    );
    return $this;
}
public function ignore(string ...$names): Preloader
{
    $this->ignores = array_merge(
        $this->ignores,
        $names
    );
```

```
        return $this;
    }
    public function load(): void
    {
        // We'll loop over all registered paths
        // and load them one by one
        foreach ($this->paths as $path) {
            $this->loadPath(rtrim($path, '/'));
        }
        $count = self::$count;
        echo "[Preloader] Preloaded {$count} classes" . PHP_EOL;
    }
    private function loadPath(string $path): void
    {
        // If the current path is a directory,
        // we'll load all files in it
        if (is_dir($path)) {
            $this->loadDir($path);
            return;
        }
        // Otherwise we'll just load this one file
        $this->loadFile($path);
    }
    private function loadDir(string $path): void
    {
        $handle = opendir($path);
        // We'll loop over all files and directories
```

```
        // in the current path,
        // and load them one by one
        while ($file = readdir($handle)) {
            if (in_array($file, ['.', '..'])) {
                continue;
            }
            $this->loadPath("{$path}/{$file}");
        }
        closedir($handle);
    }
    private function loadFile(string $path): void
    {
        // We resolve the classname from composer's autoload mapping
        $class = $this->fileMap[$path] ?? null;
        // And use it to make sure the class shouldn't be ignored
        if ($this->shouldIgnore($class)) {
            return;
        }
        // Finally we require the path,
        // causing all its dependencies to be loaded as well
        require_once($path);
        self::$count++;
        echo "[Preloader] Preloaded `{$class}`" . PHP_EOL;
    }
    private function shouldIgnore(?string $name): bool
    {
        if ($name === null) {
```

```
            return true;
        }
        foreach ($this->ignores as $ignore) {
            if (strpos($name, $ignore) === 0) {
                return true;
            }
        }
        return false;
    }
}
```

通过在相同的预加载脚本中添加此类，我们现在可以像这样加载整个Laravel框架：

```
// ...
(new Preloader())
    ->paths(__DIR__ . '/vendor/laravel')
    ->ignore(
        \Illuminate\Filesystem\Cache::class,
        \Illuminate\Log\LogManager::class,
        \Illuminate\Http\Testing\File::class,
        \Illuminate\Http\UploadedFile::class,
        \Illuminate\Support\Carbon::class,
    )
    ->load();
```

有效吗?

这当然是最重要的问题:所有文件都正确加载了吗?您可以简单地通过重新启动服务器来测试它，然后将pcache_get_status()的输出转储到PHP脚本中。您将看到它有一个名为preload_statistics的键，它列出所有预加载的函数、类和脚本;以及预加载文件消耗的内存。

###Composer支持

一个很有前途的特性可能是基于composer的自动预加载解决方案，它已经被大多数现代PHP项目所用。人们正在努力在composer.json中添加预加载配置选项，它将为您生成预加载文件!目前，此功能在开发中，但您可以在此处关注。

＃服务器要求

在使用预加载时，关于devops方面还有两件更重要的事情需要提及。

您已经知道，需要在php.ini中指定一个条目才能进行预加载。这意味着如果您使用共享主机，您将无自由地配置PHP。实际上，您需要一个专用的(虚拟)服务器，以便能够为单个项目优化预加载的文件记住这一点。

还要记住，每次需要重新加载内存文件时，都需要重新启动服务器(如果使用php-fpm就足够了)。这大多数人来说似乎是显而易见的，但仍然值得一提。

###性能

现在到最重要的问题：预加载真的能提高性能吗?

答案是肯定的：Ben Morel分享了一些基准测试，可以在之前链接的相同的composer问题中找到。

有趣的是，您可以决定仅预加载“hot classes”，它们是代码库中经常使用的类。Ben的基准测试显，只加载大约100个热门类，实际上可以获得比预加载所有类更好的性能收益。这是性能提升13%和1%的区别。

当然，应该预加载哪些类取决于您的特定项目。明智的做法是在开始时尽可能多地预加载。如果您确需要少量的百分比增长，您将不得不在运行时监视您的代码。

当然，所有这些工作都可以自动化，将来可能会实现。

现在，最重要的是要记住composer将添加支持，这样您就不必自己制作预加载文件，并且只要您完控制了此功能，就可以在服务器上轻松设置此功能。