



性感的PHP

——现代化高性能的PHP开发























商业产品事业部

主讲：谈腾

主要内容

- 命名空间
- PSR规范
- 闭包和匿名函数
- Trait
- Composer包管理器
- 语法新特性
- PHP 7性能
- Laravel框架介绍

全面的PHP技术堆栈

PHP				HTTP
前端				HTML/CSS
数据库缓存				
服务器				
版本控制				
调试工具				

.....

命名空间

命名空间在 PHP 5.3.0 中引入，其作用是按照一种虚拟的层次结构组织 PHP 代码，这种层次结构类似操作系统中文件系统的目录结构。命名空间是现代 PHP 组件生态的基础，现代的 PHP 组件框架代码都是放在各自全局唯一的厂商命名空间中，以免和其他厂商使用的常见类名冲突。

```
<?php

namespace App\Http\Controllers;

use App\Models\Destination;
use App\Models\Travel;
use GrahamCampbell\Markdown\Facades\Markdown;
use Illuminate\Support\Facades\Cache;
```


命名空间

没有命名空间之前，使用冗长的命名避免冲突：

```
class  
Zend_Cloud_DocumentService_Adapter_WindowsAzure_Query
```

关于命名空间我们要掌握的：

- 命名空间的声明
- 命名空间导入和别名
- 命名空间和自动加载的关系

PSR标准规范

PSR 1	基础编码规范
PSR 2	编程风格规范
PSR 3	日志接口规范
PSR 4	自动加载规范
PSR 6	缓存接口规范
PSR 7	HTTP消息接口规范

注：PSR 0自动加载已废弃，PSR 4取代，PSR 5 PHP doc规范还在起草，以及PSR 8，9，10，11，12，13

PSR 1 基础编码规范

- PHP代码文件必须以 `<?php` 或 `<?='` 标签开始
- PHP代码文件必须使用不带 BOM 的 UTF-8 编码
- PHP代码中应该只定义类、函数、常量等声明，或其他会产生副作用的操作
(如：生成文件输出以及修改 .ini 配置文件等)，二者只能选其一
- 命名空间以及类必须符合 PSR 的自动加载规范：PSR-4 中的一个
- 类的命名必须遵循 StudlyCaps 大写开头的驼峰命名规范
- 类中的常量所有字母都必须大写，单词间用下划线分隔
- 方法名称必须符合 camelCase 式的小写开头驼峰命名规范

PSR 2 编程风格规范

- 代码 必须 使用4个空格符而不是「Tab 键」进行缩进。
- 每行的字符数 应该 软性保持在 80 个之内，理论上 一定不可 多于 120 个，但 一定不可有硬性限制。
- 每个 namespace 命名空间声明语句和 use 声明语句块后面，必须 插入一个空白行。
- 类、方法、控制结构的开始花括号 ({) 和 结束花括号 (}) 也 必须 写在函数主体后自成一行。
- 类的属性和方法 必须 添加访问修饰符 (private、protected 以及 public) ， abstract 以及 final 必须 声明在访问修饰符之前，而 static 必须 声明在访问修饰符之后。
- 控制结构的关键字后 必须 要有一个空格符，而调用方法或函数时则 一定不可 有。
- 控制结构的开始左括号后和结束右括号前，都一定不可有空格符。

PSR 3 日志接口规范

八个等级的日志：debug、info、notice、warning、error、critical、alert 以及 emergency。

emergency	系统不可用
alert	必须立即采取行动
critical	紧急情况
error	运行时出错，但不会挂掉
warning	非错误性的异常
notice	一般重要
info	重要事件，记录sql，用户登录等
debug	用于调试

PSR 4 自动加载规范

一个完整的类名：\<命名空间>(\<子命名空间>)*\<类名>

```
namespace App\Http\Controllers;  
class TestController extends Controller  
{  
}
```

App\Http\Controller\TestController

完整类名	命名空间前缀	文件基目录	文件路径
\Aura\Web \Response>Status	Aura\Web	/path/to/aura-web/ src/	/path/to/aura-web/ src/Response/ Status.php
\Symfony\Core \Request	Symfony\Core	./vendor/Symfony/ Core/	./vendor/Symfony/ Core/Request.php
\Zend\Acl	Zend	/usr/includes/ Zend/	/usr/includes/ Zend/Acl.php

PSR 6 缓存接口规范

缓存有如下几种方式，除了APC和OPcache是扩展，其他的都应该实现统一的缓存接口，如设置缓存，获取缓存，设置过期时间等。

- Database
- File
- Memcache
- Redis
- Array
- APC
- OPcache (PHP 7性能优化第一条方法就是开启Opcache)

PSR 7 HTTP消息接口规范

HTTP 消息是 Web 技术发展的基础。浏览器或 HTTP 客户端如 curl 生成发送 HTTP 请求消息到 Web 服务器，Web 服务器响应 HTTP 请求。服务端的代码接受 HTTP 请求消息后返回 HTTP 响应消息。

HTTP Request消息格式	HTTP Response消息格式
POST /path HTTP/1.1 Host: example.com foo=bar&baz=bat	HTTP/1.1 200 OK Content-Type: text/plain

PSR 7 HTTP消息接口规范

PSR 7 规范定义了HTTP请求，响应，数据流的接口，以MessageInterface接口为例，响应和请求接口同时要继承这个接口。

```
<?php
namespace Psr\Http\Message;

interface MessageInterface
{
    public function getProtocolVersion();
    public function withProtocolVersion($version);
    public function getHeaders();
    public function hasHeader($name);
    public function getHeader($name);
    public function getHeaderLine($name);
    public function withHeader($name, $value);
    public function withAddedHeader($name, $value);
    public function withoutHeader($name);
    public function getBody();
    public function withBody(StreamInterface $body);
}
```

闭包和匿名函数

在PHP中，闭包和匿名函数是一个概念，闭包就是匿名函数。闭包是指在创建时封装周围状态的函数，即使闭包所在的环境的不存在了，闭包中封装的状态依然存在。

匿名函数其实就是没有名称的函数，匿名函数可以赋值给变量，还能像其他任何PHP函数对象那样传递。不过匿名函数仍然是函数，因此可以调用，还可以传入参数，适合作为函数或方法的回调。之所以能调用变量是闭包函数实现了__invoke魔术函数。

```
<?php  
  
$xunlei = function ($name) {  
    return sprintf("Hello %s\r\n", $name);  
};  
  
echo $xunlei('www.xunlei.com');
```


匿名函数实例

这是在Laravel中使用匿名函数的例子，使用匿名函数使代码逻辑非常简洁，富有条理。以下示例是从缓存中取数据：

```
/**
 * 全部游记
 * @return \Illuminate\Contracts\View\Factory|\Illuminate\View\View
 */
public function latest()
{
    $data = Cache::remember('latest', self::CACHE_TIME, function () {
        $travelList = $this->travel->latest('date')->paginate(20);
        return [
            'travelList' => $travelList,
        ];
    });
    $travelList = $data['travelList'];
    return view('travel.latest', compact('travelList'));
}
```

闭包bindTo方法

在闭包中使用外部变量可以使用use关键字引入，闭包也可以通过bindTo方法将指定的类引入。

```
<?php
class App {
    protected $routes = [];
    protected $responseStatus = '200 OK';
    protected $responseContentType = 'text/html';
    protected $responseBody = '迅雷';

    public function addRoute($routePath, $routeCallback) {
        $this->routes[$routePath] = $routeCallback->bindTo($this, __CLASS__);
    }

    public function dispatch()
    {
        //TODO
    }
}
```

闭包bindTo方法

实例化类，调用方法addRoute，第二个参数传入一个闭包，在闭包内部可以访问绑定类的属性和方法，这一点跟引入外部变量类似，可以理解为引入一个类。

```
$app = new App();  
$app->addRoute('user/nonfu', function(){  
    $this->responseContentType = 'application/json;charset=utf8';  
    $this->responseBody = '{"name":"xunlei"}';  
});  
$app->dispatch('user/nonfu');
```


新特性之Trait

Trait 是 PHP 5.4 引入的新概念，看上去既像类又像接口，其实都不是，Trait 可以看做类的部分实现，可以混入一个或多个现有的PHP类中，其作用有两个：表明类可以做什么；提供模块化实现。Trait是一种代码复用技术，为PHP的单继承限制提供了一套灵活的代码复用机制。

```
<?php
namespace Illuminate\Database\Eloquent;

trait SoftDeletes
{
    protected $forceDeleting = false;
    public function forceDelete(){}
    protected function runSoftDelete(){}
    public function restore(){}
    public static function restored($callback){}
    public function getDeletedAtColumn(){}
}
```

Trait的使用

如果说继承是纵向扩展，那么 Trait 实现了类的横向扩展，而且更加灵活，需要用到这个 Trait 的功能的时候就用，就像即插即用的工具一样。

```
<?php
namespace Xunlei\Web;

use Xunlei\Database\SoftDeletes;

class Foo
{
    public function one(){}

    public function two(){}

    $this->forceDelete();
}
```



走进新时代

PHP组件

<https://packagist.org/>

Packagist *The PHP Package Repository*

[Browse](#)

[Submit](#)

[tanteng](#)



Packagist is the main [Composer](#) repository. It aggregates public PHP packages installable with Composer.

Getting Started

Define Your Dependencies

Put a file named `composer.json` at the root of your project, containing your project dependencies:

```
{
  "require": {
    "vendor/package": "1.3.2",
    "vendor/package2": "1.*",
    "vendor/package3": "^2.0.3"
  }
}
```

For more information about packages versions usage, see the [composer documentation](#).

Install Composer In Your Project

Run this in your command line:

```
curl -sS https://getcomposer.org/installer | php
```

Publishing Packages

Define Your Package

Put a file named `composer.json` at the root of your package, containing this information:

```
{
  "name": "your-vendor-name/package-name",
  "description": "A short description of what your package does",
  "require": {
    "php": "^5.3.3 || ^7.0",
    "another-vendor/package": "1.*"
  }
}
```

This is the strictly minimal information you have to give.

For more details about package naming and the fields you can use to document your package better, see the [about](#) page.

Commit The File

You surely don't need help with that.

PHP组件特点

PHP组件具备以下特点：

- 作用单一：专注于解决一个问题，而且使用简单的接口封装功能
- 小型：小巧玲珑，只包含解决某个问题所需的最少代码
- 合作：PHP组件之间可以良好合作，组合在一起实现大型项目
- 测试良好：本身提供测试，而且有充足的测试覆盖度
- 文档完善：应该提供完善的文档，能让开发者轻易安装、理解和使用

PHP组件列举

PHP组件非常多，组件一般都是针对某一特定功能的，并且遵循同样的规范，这样就可以复用。

- [zizaco/entrust](#) 基于角色的权限管理
- [predis/predis](#) 灵活的Redis API接口
- [mews/captcha](#) 验证码生成和校验
- [guzzlehttp/guzzle](#) HTTP客户端
- [nesbot/carbon](#) 时间日期人性化的处理
- [barryvdh/laravel-debugbar](#)
- [laravel/laravel](#)

PHP组件依赖定义

```
{
  "name": "laravel/laravel",
  "description": "The Laravel Framework.",
  "keywords": ["framework", "laravel"],
  "license": "MIT",
  "type": "project",
  "require": {
    "php": ">=5.5.9",
    "laravel/framework": "5.2.*",
    "zglhdh/qiniu-laravel-storage": "^0.3.0",
    "zizaco/entrust": "dev-laravel-5",
    "predis/predis": "~1.0.1",
  },
  "autoload": {
    "classmap": [
      "database"
    ],
    "psr-4": {
      "App\\": "app/"
    },
    "files": [
      "app/Http/helpers.php"
    ]
  }
}
```

composer.json

PHP组件依赖定义

```
"require": {  
    "php": ">=5.5.9",  
    "laravel/framework": "5.2.*",  
    "barryvdh/laravel-ide-helper": "^2.2",  
    "erusev/parsedown": "^1.6",  
    "doctrine/dbal": "^2.5",  
    "cyvelnet/laravel5-fractal": "^1.3",  
    "vinkla/hashids": "^2.3",  
    "cviebrock/eloquent-sluggable": "^4.0",  
    "league/flysystem-aws-s3-v3": "^1.0",  
    "spatie/laravel-medialibrary": "^3.18",  
},  
"require-dev": {  
    "fzaninotto/faker": "~1.4",  
    "mockery/mockery": "0.9.*",  
    "phpunit/phpunit": "~4.0",  
    "symfony/css-selector": "2.8.*|3.0.*",  
    "symfony/dom-crawler": "2.8.*|3.0.*"  
},
```

PHP组件脚本定义

一个脚本，在 Composer 中，可以是一个 PHP 回调（定义为静态方法）或任何命令行可执行的命令。脚本对于在 Composer 运行过程中，执行一个资源包的自定义代码或包专用命令是非常有用的。

```
"scripts": {
    "post-root-package-install": [
        "php -r \"copy('.env.example', '.env');\""
    ],
    "post-create-project-cmd": [
        "php artisan key:generate"
    ],
    "post-install-cmd": [
        "Illuminate\\Foundation\\ComposerScripts::postInstall",
        "php artisan optimize"
    ],
    "post-update-cmd": [
        "Illuminate\\Foundation\\ComposerScripts::postUpdate",
        "php artisan optimize"
    ]
},
```

PHP组件脚本事件

Composer 在运行过程中将会触发以下事件：

事件名称	说明	事件名称	说明
pre-install-cmd	在 install 命令执行前触发	pre-package-install	在资源包安装前触发
post-install-cmd	在 install 命令执行后触发	post-package-install	在资源包安装后触发
pre-update-cmd	在 update 命令执行前触发	pre-package-update	在资源包更新前触发
post-update-cmd	在 update 命令执行后触发	post-package-update	在资源包更新后触发
pre-status-cmd	在 status 命令执行前触发	pre-package-uninstall	在资源包被卸载前触发
post-status-cmd	在 status 命令执行后触发	post-package-uninstall	在资源包被卸载后触发

PHP组件安装和更新

推荐安装方式：

```
curl -sS https://getcomposer.org/  
installer | php  
mv composer.phar /usr/local/bin/  
composer
```

设置中国镜像：

```
composer config repo.packagist  
composer https://  
packagist.phpcomposer.com
```

Composer几个命令：

```
composer install  
composer update  
composer dump-autoload
```

```
- Installing doctrine/inflector (v1.1.0)  
  Loading from cache  
  
- Installing anahkiasen/underscore-php (2.0.0)  
  Downloading: 100%  
  
- Installing spatie/string (2.1.0)  
  Downloading: 100%  
  
- Installing spatie/pdf-to-image (1.2.0)  
  Downloading: 100%  
  
- Installing symfony/polyfill-mbstring (v1.2.0)  
  Loading from cache  
  
- Installing symfony/http-foundation (v3.0.9)  
  Downloading: 100%  
  
- Installing symfony/event-dispatcher (v3.1.3)  
  Downloading: 100%  
  
- Installing psr/log (1.0.0)  
  Loading from cache  
  
- Installing symfony/debug (v3.0.9)  
  Downloading: 100%  
  
- Installing symfony/http-kernel (v3.0.9)  
  Downloading: 100%
```


PHP组件使用

使用PHP组件，首先在composer.json定义依赖，并安装依赖，在文件中声明USE的命名空间，以下是GuzzleHttp请求接口的示例：

```
use GuzzleHttp\Client;

/**
 * Github上提交版本历史
 * @method POST
 * @return string
 */
public function gitCommitHistory()
{
    $client = new Client([
        'base_uri' => 'https://api.github.com/',
    ]);
    $uri = 'repos/tanteng/tanteng.me/commits';
    $query['client_id'] = Config::get('github.client_id');
    $query['client_secret'] = Config::get('github.client_secret');
    $result = $client->get($uri, $query)->getBody()->getContents();
    $result = json_decode($result, true);
    $commitHistory = [];
    foreach ($result as $item) {
        $commitHistory[] = [
            'message' => $item['commit']['message'],
            'datetime' => date('Y-m-d H:i:s', strtotime($item['commit']['committer']['date']))
        ];
    }
}
```

PHP组件自动加载

使用composer方式安装PHP组件，会自动产生一个vendor目录放第三方组件，并且生成自动加载机制。

在vendor目录下，可以看到一个autoload.php文件，以及composer文件夹，里面定义了自动加载的映射。

本质还是通过PHP的spl_autoload_register自动加载函数实现，按照composer定义的自动加载规范。

PHP语法新特性

1. 运算符

```
$a = $_GET['a'] ?? $_GET['a']:1;
```

2. 函数指定返回类型

```
function arraysSum(array ...$arrays): array
{
    return array_map(function (array $array): int {
        return array_sum($array);
    }, $arrays);
}
```

```
print_r(arraysSum([1, 2, 3], [4, 5, 6], [7, 8, 9]));
```

3. 类型和标量约束

4. 操作符<=> （比较两个数大小，返回-1、0或1）

5. 延迟静态绑定static

新特性之类型约束

PHP 5 可以使用类型约束。函数的参数可以指定必须为对象（在函数原型里面指定类的名字），接口，数组（PHP 5.1 起）或者 callable（PHP 5.4 起）。

两种类型约束：参数，返回值。PHP 7 支持 int , string 等参数的类型约束。

```
public function withoutGlobalScopes(array $scopes = null)
{
    if (is_array($scopes)) {
        foreach ($scopes as $scope) {
            $this->withoutGlobalScope($scope);
        }
    } else {
        $this->scopes = [];
    }

    return $this;
}
```


PHP 7 性能

什么是HHVM?

2010年 Facebook 开发了HHVM, HHVM 使用了 Just-In-Time (JIT) 编译方式将 PHP 代码转换成某种字节码。接下来把字节码再转换成机器码并进行优化, 让它尽可能快的运行。

PHP 7 性能

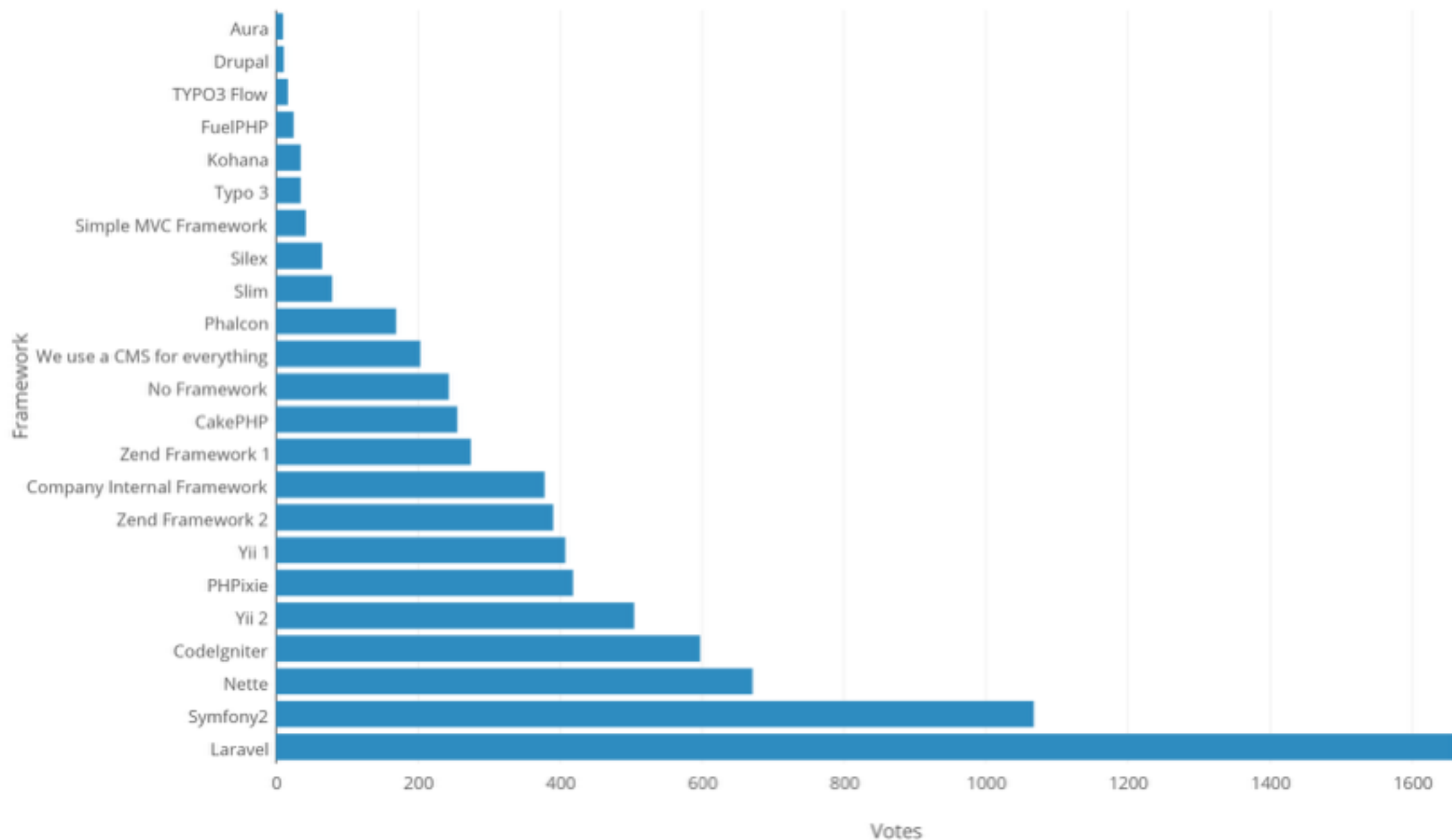
很多的测试表明 PHP 7 性能比 PHP 5 性能翻倍, 跟 HHVM 差不多, 而且还可以进一步优化 PHP 7 性能, 开启 Opcache 缓存字节码, 它是一个 PHP 扩展, PHP 5.5 以上就可以使用。



学习现代化PHP开发的最好方式是
学习现代化PHP开发框架

国外PHP框架排行

PHP Framework Popularity at Work - SitePoint, 2015




学习现代化框架的理由

- 充分利用PHP新特性
- 遵循PSR规范
- 优雅的代码风格
- 设计模式
- 依赖注入和容器
- 通过composer使用丰富的PHP组件

“The only way to do great work is to love what
you do”

–Steve Jobs



谢谢

欢迎交流

主页 <http://tanteng.me>

博客 <http://blog.tanteng.me>

邮箱 tanteng@xunlei.com