php

点滴学习，随时记录

array_search 和 in_array 函数效率问题

问题

array_search 查找数组中的元素的 key 时，效率随着数组变大，耗时增加。特别是多次查找大数组时，非常耗时。在函数 in_array 也有这个问题。

解决办法

采用 array_flip （array_flip — 交换数组中的键和值）翻转后，用 isset 代替 in_array 函数，用 $array[key] 替代 array_search, 这样能解决大数组超时耗时问题.

注意：

1、这种优化只适用于无重复数据的大数组且多次查找时有效，原因见下面。
2、array_search、in_array、array_flip 原理都是数组遍历。array_search、in_array是发现便终止，array_flip遍历整个数组
3、因array_flip遍历整个数组，相比array_search和in_array，array_flip更耗时，只有多次使用array_search和in_array时才能显示出array_flip效果
4、isset 走的 hash 表,查找时间复杂度为 O(1),更高效。
5、使用场景不多，基本用处不大 只是提供一种思路

实例：

```php
<?php
$array = array();

for ($i=0; $i<200000; $i++){
    ##随机字符串
    $array[$i] = get_rand().$i;
}

$str = $array[150000];
$time1 = microtime(true);
```

```
array_search($str, $array);
$time2 = microtime(true);
echo '原始方法:'.($time2-$time1)."\n";

$time3 =  microtime(true);
$new_array = array_flip($array);
isset($new_array[$str]);
$time4 = microtime(true);
echo '新方法:'.($time4-$time3);


$array = array();

for ($i=0; $i<200000; $i++){
    ##随机字符串
    $array[$i] = get_rand().$i;
}

$str = $array[199999];
$time1 = microtime(true);
for ($i=0; $i<5000; $i++){
    array_search($str, $array);
}
$time2 = microtime(true);

echo '原始方法:'.($time2-$time1)."\n";

$time3 =  microtime(true);
$new_array = array_flip($array);
for ($i=0; $i<5000; $i++){
    isset($new_array[$str]);
}
$time4 = microtime(true);

echo '新方法:'.($time4-$time3);
```

转自： https://learnku.com/laravel/t/34826

phpredis 手册

目录（点击下面链接查看详情）

转自GitHub，地址： https://github.com/phpredis/phpredis

PHPMailer 在本地发送成功 阿里云上发送失败原因

阿里云默认不允许访问SMTP 25端口，可申请解封25端口或换ssl加密方式连接和端口

已腾讯企业邮箱为例：

```
    $mail = new PHPMailer();
$mail->IsSMTP();
# $mail->Host = 'smtp.exmail.qq.com';  # 本地可直接使用
$mail->Host = 'ssl://smtp.exmail.qq.com'; # 阿里云线上使用ssl加密方式
$mail->Port = 465; # ssl方式 用465端口
$mail->SMTPAuth = true;            // 打开SMTP认证
$mail->Username = 'xxx@xxx.com';
$mail->Password = 'yyyyyyy';
$mail->From = 'xxx@xxx.com';
$mail->FromName = iconv('utf-8', 'GBK', "xxx");  // 发件人

$mail->CharSet = 'GB2312';
$mail->Encoding = "base64";
$mail->IsHTML(true);
// 邮件主题
$mail->Subject = iconv('utf-8', 'GBK', $subject);
// 邮件内容
$mail->Body = iconv('utf-8', 'GBK', $message);
$mail->AltBody = "text/html";
if ($mail->Send()) {
    exit('success');
} else {
    exit($mail->ErrorInfo);
}
```

urldecode & rawurldecode 说明

区别：

urlencode把空格编码为 '+'， rawurlencode()把空格编码为 '%20'

urldecode() 会把 '+' 破解为空格，rawurldecode() 不会

注意

因为'+' 号是 base64的编码字符，当urldecode与base64_decode配合使用时，要用rawurldecode()

推荐在PHP中使用用rawurlencode。弃用urlencode;
大部分使用场景下都适合使用rawurlencode()

url中base64之后的参数 一定要rawurlencode, 因为base64之后的字符串会包含"+" "/" 等特殊字符(此时base64可以替换为urlbase64，也可以解决这个问题)

超全局变量 $*GET 和 $*REQUEST 已经被解码了。对 $*GET 或 $*REQUEST 里的元素使用 urldecode() 将会导致不可预计和危险的结果，即接受urlencode转义的请求时不需要urldecode转义。其他的如$_POST等请求则需要自己转义

PHP在linux上执行系统命令

方法一：用PHP提供的专门函数（四个）：
1）exec():

```
string exec ( string $command [, array &$output [, int &$return_var ]
)
```

说明: exec执行系统外部命令时不会输出结果，而是返回结果的最后一行。如果想得到结果，可以使用第二个参数，让其输出到指定的数组。此数组一个记录代表输出的一行。即如果输出结果有20行，则这个数组就有20条记录，所以如果需要反复输出调用不同系统外部命令的结果，最好在输出每一条系统外部命令结果时清空这个数组unset($output)，以防混乱。第三个参数用来取得命令执行的状态码，通常**执行成功都是返回0**。

```
<?php
    // 输出运行中的 php/httpd 进程的创建者用户名
    //  (在可以执行 "whoami" 命令的系统上)
    echo exec('whoami');
?>
```

2）system():

```
string system ( string $command [, int &$return_var ] )
```

说明: system和exec的区别在于，system在执行系统外部命令时，它执行给定的命令，输出和返回结果。
成功
则返回命令输出的最后一行， 失败则返回 FALSE。第二个参数是可选的，用来得到命令执行后的状态码。

```php
<?php
    $res = system("pwd",$result);
    print $result;//输出命令的结果状态码
    print $res;//输出命令输出的最后一行

?>
```

关于第二个参数结果状态码的简单介绍：
如果**返回0是运行成功**，
在Bash中，当错误发生在致命信号时，bash会返回128+signal number做为返回值。
如果找不到命令，将会返回127。
如果命令找到了，但该命令是不可执行的，将返回126。
除此以外，Bash本身会返回最後一个指令的返回值。
若是执行中发生错误，将会返回一个非零的值。
Fatal Signal : 128 + signo
Can't not find command : 127
Can't not execute : 126
Shell script successfully executed : return the last command exit status
Fatal during execution : return non-zero

3）passthru():

```
void passthru ( string $command [, int &$return_var ] )
```

说明: passthru与system的区别，passthru直接将结果输出到游览器，不返回任何值，且其可以输出二进制，比如图像数据。第二个参数可选，是状态码。

```
同 exec() 函数类似， passthru() 函数 也是用来执行外部命令（command）的。 当所执行的 Unix 命令输出二进制数据， 并且需要直接传送到浏览器的时候， 需要用此函数来替代 exec() 或 system() 函数。 常用来执行诸如 pbmplus 之类的可以直接输出图像流的命令。 通过设置 Content-type 为 image/gif， 然后调用 pbmplus 程序输出 gif 文件， 就可以从 PHP 脚本中直接输出图像到浏览器。
```

```php
<?php
    header("Content-type:image/gif");
    passthru("/usr/bin/ppm2tiff  /usr/share/tk8.4/demos/images/teapot.ppm");
?>
```

4）shell_exec():

```
string shell_exec ( string $cmd )
```

说明: 直接执行命令$cmd，将完整的命令输出以字符串的方式返回。如果执行过程中发生错误或者进程不产生
输出，则返回 NULL。所以，使用本函数无法通过返回值检测进程是否成功执行。 如果需要检查进程执行的退
出码，请使用 exec() 函数。

```php
<?php
    $output = shell_exec('ls -lart');
    echo "
<pre>$output</pre>";
?>
```

方法二：反撇号：
原型: 反撇号`（和~在同一个键）执行系统外部命令，相当于 shell_exec
说明: 在使用这种方法执行系统外部命令时，要确保shell_exec函数可用，否则是无法使用这种反撇号执行系统外部命令的。

```php
<?php
    echo `dir`;
```

```
?>
```

用户自定义输入命令转义：
若命令需要用户输入，此时为了安全应该使用以下方法对用户输入命令或参数进行转义

1、shell 元字符转义

```
string escapeshellcmd ( string $command )

参数说明：
command    要转义的命令。
返回        转义后的字符串。
```

除去了字符串中的特殊符号，可以防止使用者耍花招来破解该服务器系统。
escapeshellcmd() 对字符串中可能会欺骗 shell 命令执行任意命令的字符进行转义。 此函数保证用户输入的数据在传送到 exec() 或 system() 函数，或者 执行操作符 之前进行转义。

反斜线（\）会在以下字符之前插入： #&;`|*?~<>^()[]{}$, \x0A 和 \xFF。 ' 和 " 仅在不配对儿的时候被转义。 在 Windows 平台上，所有这些字符以及 % 都会被空格代替。

实例：

```
<?php
    // 我们故意允许任意数量的参数
    $command = './configure '.$_POST['configure_options'];

    $escaped_command = escapeshellcmd($command);

    system($escaped_command);
?>
```

escapeshellcmd() 应被用在完整的命令字符串上。 即使如此，攻击者还是可以传入任意数量的参数。 请使用 escapeshellarg() 函数 对单个参数进行转义。

2、shell 参数转义

```
string escapeshellarg ( string $arg )

arg ：需要被转码的参数。

返回值：转换之后字符串。
```

escapeshellarg() 将给字符串增加一个单引号并且能引用或者转码任何已经存在的单引号，这样以确保能够直接将一个字符串传入 shell 函数，并且还是确保安全的。对于用户输入的部分参数就应该使用这个函数。shell 函数包含 exec(), system() 执行运算符 。
实例：

```
<?php
    system('ls '.escapeshellarg($dir));
?>
```

php运算符优先级

https://www.php.net/manual/zh/language.operators.precedence.php

依赖管理Composer

人生苦短，我用 Composer

在 PHP 生态中，Composer 是最先进的依赖管理方案。我们推荐 PHP: The Right Way 中关于 依赖管理的完整章节。
如果你没有使用 Composer 来管理应用的依赖，最终（hopefully later but most likely sooner）会导致应用里某个依赖会严重过时，然后老旧版本中的漏洞会被利用于计算机犯罪。
重要： 开发软件时，时常记得保持依赖的更新。幸运地，这只需一行命令：

```
composer update
```

如果你正在使用某些专业的，需要使用 PHP 扩展（C 语言编写），那你不能使用 Composer 管理，而需要 PECL 。

面向对象特性与设计原则

三大特性是：封装、继承、多态

所谓封装，也就是把客观事物封装成抽象的类，并且类可以把自己的数据和方法只让可信的类或者对象操作，对不可信的进行信息隐藏。

封装是面向对象的特征之一，是对象和类概念的主要特性。 简单的说，一个类就是一个封装了数据以及操作这些数据的代码的逻辑实体。在一个对象内部，某些代码或某些数据可以是私有的，不能被外界访问。通过这种方式，对象对内部数据提供了不同级别的保护，以防止程序中无关的部分意外的改变或错误的使用了对象的私有部分。

所谓继承是指可以让某个类型的对象获得另一个类型的对象的属性的方法,它支持按级分类的概念。

继承是指这样一种能力：它可以使用现有类的所有功能，并在无需重新编写原来的类的情况下对这些功能进行扩展。 通过继承创建的新类称为“子类”或“派生类”，被继承的类称为“基类”、“父类”或“超类”。继承的过程，就是从一般到特殊的过程。要实现继承，可以通过“继承”（Inheritance）和“组合”（Composition）来实现。继承概念的实现方式有二类：实现继承与接口继承。实现继承是指直接使用基类的属性和方法而无需额外编码的能力；接口继承是指仅使用属性和方法的名称、但是子类必须提供实现的能力；

所谓多态就是指一个类实例的相同方法在不同情形有不同表现形式。

多态机制使具有不同内部结构的对象可以共享相同的外部接口。这意味着，虽然针对不同对象的具体操作不同，但通过一个公共的类，它们（那些操作）可以通过相同的方式予以调用。

五大设计原则

单一职责原则 (The Single Responsibility Principle)

一个类的功能要单一，不能包罗万象；
修改某个类的理由应该只有一个，如果超过一个，说明类承担不止一个职责，要视情况拆分。

开放封闭原则 (The Open Closed Principle)

软件实体应该对扩展开放，对修改封闭。一般不要直接修改类库源码（即使你有源代码），通过继承等方式扩展。

里氏替代原则（The Liskov Substitution Principle）

当一个子类的实例能够被替换成任何超类的实例时，它们之间才是真正的 is-a 关系。
即子类应当可以替换父类并出现在父类能够出现的任何地方。

依赖倒置原则（The Dependency Inversion Principle）

高层模块不应该依赖于底层模块，二者都应该依赖于抽象。换句话说，依赖于抽象，不要依赖于具体实现。比方说，你不会把电器电源线焊死在室内电源接口处，而是用标准的插头插在标准的插座（抽象）上。

接口分离原则（The Interface Segregation Principle）

模块间要通过抽象接口隔离开，而不是通过具体的类强耦合起来

不要强迫用户去依赖它们不使用的接口。换句话说，使用多个专门的接口比使用单一的大而全接口要好。

生成二维码

qr-code项目

github 地址为：https://github.com/endroid/qr-code

可设置参数：
`setSize` – 二维码大小 px
`setWriterByName` – 写入文件的后缀名
`setMargin` – 二维码内容相对于整张图片的外边距
`setEncoding` – 编码类型
`setErrorCorrectionLevel` – 容错等级，分为L、M、Q、H四级
`setForegroundColor` – 前景色
`setBackgroundColor` – 背景色
`setLabel` – 二维码标签
`setLogoPath` – 二维码logo路径
`setLogoWidth` – 二维码logo大小 px

使用说明： https://juejin.im/entry/5a7bc1976fb9a0634a390122?utm_medium=be&utm_source=weixinqun

反斜杠过滤

```
# 过滤小于等于两个反斜杠
$str = stripslashes($str);

#过滤不限数量反斜杠
$str = stripslashes(trim(implode("", explode("\\", $str))));
```

时间格式判断

```php
# 判断所给时间数据格式是否为`m-d`的形式
# 注意这种方式只能判断当年的，若判断往年的还得考虑平、闰年的情况
$date = '02-03';
if (date('m-d', strtotime(date('Y') . '-' . $date)) === $date) {
    echo '是';
} else {
    echo '否';
}
```

php获取用户和服务器ip及其地理位置详解

浏览器访问获取用户ip：

```
/**
 * php获取用户真实 IP
 * 注意这种方式只适用于浏览器访问时
 */
function getIP()
{
    if (isset($_SERVER)){
        if (isset($_SERVER["HTTP_X_FORWARDED_FOR"])){
            $realip = $_SERVER["HTTP_X_FORWARDED_FOR"];
        } else if (isset($_SERVER["HTTP_CLIENT_IP"])) {
            $realip = $_SERVER["HTTP_CLIENT_IP"];
        } else {
            $realip = $_SERVER["REMOTE_ADDR"];
        }
    } else {
        if (getenv("HTTP_X_FORWARDED_FOR")){
            $realip = getenv("HTTP_X_FORWARDED_FOR");
        } else if (getenv("HTTP_CLIENT_IP")) {
            $realip = getenv("HTTP_CLIENT_IP");
        } else {
            $realip = getenv("REMOTE_ADDR");
        }
    }
    return $realip;
}
```

注意：

1、以上方式只适用于用浏览器访问后台服务时可用

2、以浏览器访问和在后台直接执行php脚本所生成的 $_SERVER 变量是不同的

后台脚本执行获取服务器ip：

```php
<?php
/**
*方法一：使用 gethostbyname() 方法,此方法获取的是内网ip
*/
$realip =  gethostbyname(gethostname());
print_r($realip);

echo "\n";

/**
*方法二：php执行linux系统命令 ifconfig
* 利用正则表达式获取 ip
*/

$output = shell_exec('ifconfig');
preg_match("/\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3}/",$output,$realip );
print_r($realip [0]);

/**
*方法三：php执行linux系统命令 ifconfig ，此方法获取外网ip
* 利用grep获取 ip
*/
$shell = "/sbin/ifconfig | grep -oP '(?<=addr:).*(?=\s+B)' | sed '1d'";
$output = shell_exec($shell);
$shell = "/sbin/ifconfig | grep -oP '(?<=addr:).*(?=\s+B)'";
$output = exec($shell);
return $output;

?>
```

注意：

1、gethostname() 获取的是 eth0 的ip，虚拟机下linux没有eth0项，所以当在虚拟机下执行该方法时返回 127.0.0.1

其他方法解释：

```
gethostbyname(string $hostname) : 获取对应主机名的一个ipv4地址

gethostbyaddr ( string $ip_address ) : 获取指定的IP地址对应的主机名

gethostbynamel ( string $hostname ) : 获取对应主机名的一系列所以ipv4地址
```

php获取ip所属城市：

```
/**
 * php获取 IP  地理位置
 * 淘宝IP接口
 * @Return: array
 */
function getCity($ip = '')
{
    if($ip == ''){
        $url = "http://int.dpool.sina.com.cn/iplookup/iplookup.php?format=json";
        $ip=json_decode(file_get_contents($url),true);
        $data = $ip;
    }else{
        $url="http://ip.taobao.com/service/getIpInfo.php?ip=".$ip;
        $ip=json_decode(file_get_contents($url));
        if((string)$ip->code=='1'){
            return false;
        }
        $data = (array)$ip->data;
    }

    return $data['city'];
}
```

lumen框架解决非简单请求 cors 跨域问题

Lumen在做前后端分离项目时，在浏览器中访问，若前后端域名不一致会导致跨域问题，简单跨域好解决。若发送的是非简单跨域，此时浏览器会先发送option请求进行预检，预检通过才发送真正的请求。此时服务端要实现option请求的接收。服务端代码实现如下：

添加如下中间件：

```php
<?php

namespace App\Http\Middleware;

use Closure;

class CrossHttp
{
    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @return mixed
     */
    public function handle($request, Closure $next)
    {
        if($request->getMethod() == "OPTIONS") {
            $allowOrigin = [
                'http://192.168.1.47',
                'http://localhost',
            ];
            $Origin = $request->header("Origin");
            if(in_array($Origin, $allowOrigin)){
                return response()->json('ok', 200, [
                    # 下面参数视request中header而定
                    'Access-Control-Allow-Origin' => $Origin,
                    'Access-Control-Allow-Headers' => 'x-token',
                    'Access-Control-Allow-Methods' => 'GET,POST,OPTI
```

```
    ONS']);
30              } else {
31                  return response()->json('fail', 405);
32              }
33          }
34
35          $response = $next($request);
36          $response->header('Access-Control-Allow-Origin', '*');
37          return $response;
38      }
39  }
```

在 boostrap/app.php 里注册一下全局中间件即可完成

```
1 $app->middleware([
2       \App\Http\Middleware\CrossHttp::class,
3 ]);
```