# Php开发规范

## 一、命名规范

1. 文件夹名小写
2. 文件名小写
3. 命名空间小写
4. 类文件和类名按大驼峰法命名 如：SiteController.php
5. 类方法和变量按小驼峰法命名 如：$this->getUserName(), $userName,
6. 面向过程的函数按下划线法命名 如：file\_get\_contents()
7. 常量以大写字母和下划线命名，例如：YII\_DEBUG

## 二、代码规范

1. controller调service调model

业务逻辑尽量在service层实现，controller通过service得到业务数据，service调model得到基础数据。

service和model一样，要根据所属模块分目录。

2.函数重用性

所有是相同功能的代码块被两处及两处以上调用，则把改代码块封装成函数放到特定类中；如果调用处有局部差异，则可以通过参数区别。

1. 精准归类

一个类里放的属性和方法应当属于该类。一般来说，一张表对应一个model对应一个service对应一个controller；若多张表属于个逻辑体，则多张表对应多个model对应一个service对应一个controller。

1. 输入检查

所有方法体，都必须对参数做基本验证（空验证，类型一致），参数不合法应立即return。对象在使用前，必须判空，特别是对于model对象。

1. 注释

所有php文件加头注释，所有方法加参数说明注释。

6.模型化和参数化

对于单条数据读写的情况，需要尽量把数据进行对象化处理（模型化）。基于Model去做数据操作，会方便后期维护，也能防止注入攻击。

对于批量数据读写的情况，更新内容尽量参数化，可以防止注入攻击。

7.每个service都要预留systemKey = null这个参数（用来识别不同的客户端，做一些定制）

如：public function test($a, $b, $systemKey = null)

## 三、php高效编程建议(网上搜集)

1.尽量静态化：

如果一个方法能被静态，那就声明它为静态的，速度可提高1/4，甚至我测试的时候，这个提高了近三倍。

当然了，这个测试方法需要在十万级以上次执行，效果才明显。

其实静态方法和非静态方法的效率主要区别在内存：静态方法在程序开始时生成内存,实例方法在程序运行中生成内存，所以静态方法可以直接调用,实例方法要先成生实例,通过实例调用方法，静态速度很快，但是多了会占内存。

任何语言都是对内存和磁盘的操作,至于是否面向对象,只是软件层的问题,底层都是一样的,只是实现方法不同。静态内存是连续的,因为是在程序开始时就生成了,而实例申请的是离散的空间,所以当然没有静态方法快。

静态方法始终调用同一块内存，其缺点就是不能自动进行销毁，而是实例化可以销毁。

2.echo的效率高于print,因为echo没有返回值，print返回一个整型;

测试：

Echo

0.000929 - 0.001255 s (平均 0.001092 seconds)

Print

0.000980 - 0.001396 seconds (平均 0.001188 seconds)

相差8%左右，总体上echo是比较快的。

注意，echo大字符串的时候，如果没有做调整就严重影响性能。使用打开apached的mod\_deflate进行压缩或者打开ob\_start先将内容放进缓冲区。

3.在循环之前设置循环的最大次数，而非在在循环中;

傻子都明白的道理。

4.销毁变量去释放内存，特别是大的数组;

数组和对象在php特别占内存的，这个由于php的底层的zend引擎引起的，

一般来说，PHP数组的内存利用率只有 1/10, 也就是说，一个在C语言里面100M 内存的数组，在PHP里面就要1G。

特别是在PHP作为后台服务器的系统中，经常会出现内存耗费太大的问题。

5.避免使用像\_\_get, \_\_set, \_\_autoload等魔术方法;

对于\_\_开头的函数就命名为魔术函数，此类函数都在特定的条件下初访的。总得来说，有下面几个魔术函数

\_\_construct()，\_\_destruct()，\_\_get()，\_\_set()，\_\_unset()，\_\_call()，\_\_callStatic()，\_\_sleep()，\_\_wakeup()，\_\_toString()，\_\_set\_state()，\_\_clone()，\_\_autoload()

其实，如果\_\_autoload不能高效的将类名与实际的磁盘文件(注意，这里指实际的磁盘文件，而不仅仅是文件名)对应起来，系统将不得不做大量的文件是 否存在(需要在每个include path中包含的路径中去寻找)的判断，而判断文件是否存在需要做磁盘I/O操作，众所周知磁盘I/O操作的效率很低，因此这才是使得autoload机制效率降低的原因。

因此，我们在系统设计时，需要定义一套清晰的将类名与实际磁盘文件映射的机制。这个规则越简单越明确，autoload机制的效率就越高。

结论：autoload机制并不是天然的效率低下，只有滥用autoload，设计不好的自动装载函数才会导致其效率的降低.

所以说尽量避免使用\_\_autoload魔术方法，有待商榷。

6.requiere\_once()比较耗资源;

这是因为requiere\_once需要判断该文件是否被引用过),所以能不用尽量不用。常用require/include方法避免。

7.在includes和requires中使用绝对路径。

如果包含相对路径，PHP会在include\_path里面遍历查找文件。

用绝对路径就会避免此类问题，因此解析操作系统路径所需的时间会更少。

8.如果你需要得到脚本执行时的时间，$\_SERVER['REQUSET\_TIME']优于time();

可以想象。一个是现成就可以直接用，一个还需要函数得出的结果。

9.能用PHP内部字符串操作函数的情况下，尽量用他们，不要用正则表达式； 因为其效率高于正则;

没得说，正则最耗性能。

有没有你漏掉的好用的函数？例如：strpbrk()strncasecmp()strpos()/strrpos()/stripos()/strripos()加速 strtr如果需要转换的全是单个字符的时候，

用字符串而不是数组来做 strtr：

< ?php

$addr = strtr($addr, "abcd", "efgh"); // good

$addr = strtr($addr, array('a' => 'e', )); // bad

?>

效率提升：10 倍。

10.str\_replace字符替换比正则替换preg\_replace快，但strtr比str\_replace又快1/4;

另外不要做无谓的替换即使没有替换，str\_replace 也会为其参数分配内存。很慢！解决办法：

用 strpos 先查找(非常快)，看是否需要替换，如果需要，再替换效率：- 如果需要替换：效率几乎相等，差别在 0.1% 左右。

如果不需要替换：用 strpos 快 200%。

11.参数为字符串

如果一个函数既能接受数组又能接受简单字符做为参数，例如字符替换函数，并且参数列表不是太长，可以考虑额外写一段替换代码，使得每次传递参数都是一 个字符，而不是接受数组做为查找和替换参数。大事化小，1+1>2;

12.最好不用@，用@掩盖错误会降低脚本运行速度;

用@实际上后台有很多操作。用@比起不用@，效率差距：3 倍。特别不要在循环中使用@，在 5 次循环的测试中，即使是先用 error\_reporting(0) 关掉错误，在循环完成后再打开，都比用@快。

13.$row['id']比$row[id]速度快7倍

建议养成数组键加引号的习惯;

14.在循环里别用函数

例如For($x=0; $x < count($array); $x), count()函数在外面先计算;原因你懂的。

16.在类的方法里建立局部变量速度最快，几乎和在方法里调用局部变量一样快;

17.建立一个全局变量要比局部变量要慢2倍;

由于局部变量是存在栈中的，当一个函数占用的栈空间不是很大的时候，这部分内存很有可能全部命中cache，这时候CPU访问的效率是很高的。

相反，如果一个函数里既使用了全局变量又使用了局部变量，那么当这两段地址相差较大时，cpu cache需要来回切换，那么效率会下降。

(我理解啊)

18.建立一个对象属性(类里面的变量)例如($this->prop++)比局部变量要慢3倍;

19.建立一个未声明的局部变量要比一个已经定义过的局部变量慢9-10倍

20.声明一个未被任何一个函数使用过的全局变量也会使性能降低(和声明相同数量的局部变量一样)。

PHP可能去检查这个全局变量是否存在;

21.方法的性能和在一个类里面定义的方法的数目没有关系

因为我添加10个或多个方法到测试的类里面(这些方法在测试方法的前后)后性能没什么差异;

22.在子类里方法的性能优于在基类中;

23.只调用一个参数并且函数体为空的函数运行花费的时间等于7-8次$localvar++运算，而一个类似的方法(类里的函数)运行等于大约15次$localvar++运算;

24 用单引号代替双引号来包含字符串，这样做会更快一些。

因为PHP会在双引号包围的字符串中搜寻变量，单引号则不会。

PHP 引擎允许使用单引号和双引号来封装字符串变量，但是这个是有很大的差别的！使用双引号的字符串告诉 PHP 引擎首先去读取字符串内容，查找其中的变 量，并改为变量对应的值。一般来说字符串是没有变量的，所以使用双引号会导致性能不佳。最好是使用字

符串连接而不是双引号字符串。

BAD:

$output = "This is a plain string";

GOOD:

$output = 'This is a plain string';

BAD:

$type = "mixed";

$output = "This is a $type string";

GOOD:

$type = 'mixed';

$output = 'This is a ' . $type .' string';

25.当echo字符串时用逗号代替点连接符更快些。

echo一种可以把多个字符串当作参数的“函数”（译注：PHP手册中说echo是语言结构，不是真正的函数，故把函数加上了双引号）。

例如echo $str1,$str2。

26.Apache解析一个PHP脚本的时间要比解析一个静态HTML页面慢2至10倍。

尽量多用静态HTML页面，少用脚本。

28.尽量使用缓存，建议用memcached。

高性能的分布式内存对象缓存系统，提高动态网络应用程序性能，减轻数据库的负担;

也对运算码 (OP code)的缓存很有用，使得脚本不必为每个请求做重新编译。

29.使用ip2long()和long2ip()函数把IP地址转成整型存放进数据库而非字符型。

这几乎能降低1/4的存储空间。同时可以很容易对地址进行排序和快速查找;

30.使用checkdnsrr()通过域名存在性来确认部分email地址的有效性

这个内置函数能保证每一个的域名对应一个IP地址;

31.使用mysql\_\*的改良函数mysqli\_\*;

32.试着喜欢使用三元运算符(?：);

33.是否需要PEAR

在你想在彻底重做你的项目前，看看PEAR有没有你需要的。PEAR是个巨大的资源库，很多php开发者都知道;

35.使用error\_reporting(0)函数来预防潜在的敏感信息显示给用户。

理想的错误报告应该被完全禁用在php.ini文件里。可是如果你在用一个共享的虚拟主机，php.ini你不能修改，那么你最好添加error\_reporting(0)函数，放在每个脚本文件的第一行(或用

require\_once()来加载)这能有效的保护敏感的SQL查询和路径在出错时不被显示;

36.使用 gzcompress() 和gzuncompress()对容量大的字符串进行压缩(解压)在存进(取出)数据库时。

这种内置的函数使用gzip算法能压缩到90%;

37.通过参数变量地址得引用来使一个函数有多个返回值。

你可以在变量前加个“&”来表示按地址传递而非按值传递;

38. 完全理解魔术引用和SQL注入的危险。

Fully understand “magic quotes” and the dangers of SQL injection. I’m hoping that most developers reading this are already familiar with SQL injection. However, I list it here because it’s absolutely critical to understand. If you’ve never heard the term before, spend the entire rest of the day googling and reading.

39.某些地方使用isset代替strlen

当操作字符串并需要检验其长度是否满足某种要求时，你想当然地会使用strlen()函数。此函数执行起来相当快，因为它不做任何计算，只返回在zval 结构（C的内置数据结构，用于存储PHP变量）中存储的已知字符串长度。但是，由于strlen()是函数，多多少少会有些慢，因为函数调用会经过诸多步骤，如字母小写化（译注：指函数名小写化，PHP不区分函数名大小写）、哈希查找，会跟随被调用的函数一起执行。在某些情况下，你可以使用isset() 技巧加速执行你的代码。（举例如下）

if (strlen($foo) < 5) { echo “Foo is too short”$$ }

（与下面的技巧做比较）

if (!isset($foo{5})) { echo “Foo is too short”$$ }

调用isset()恰巧比strlen()快，因为与后者不同的是，isset()作为一种语言结构，意味着它的执行不需要函数查找和字母小写化。也就是说，实际上在检验字符串长度的顶层代码中你没有花太多开销。

40.使用++$i递增

When incrementing or decrementing the value of the variable $i++ happens to be a tad slower then ++$i. This is something PHP specific and does not apply to other languages, so don’t go modifying your C or Java code thinking it’ll suddenly become faster, it won’t. ++$i happens to be faster in PHP because instead of 4 opcodes used for $i++ you only need 3. Post incrementation actually causes in the creation of a temporary var that is then incremented. While preincrementation increases the original value directly. This is one of the optimization that opcode optimized like Zend’s PHP optimizer. It is a still a good idea to keep in mind since not all opcode optimizers perform this optimization and there are plenty of ISPs and servers running without an opcode optimizer.

当执行变量$i的递增或递减时，$i++会比++$i慢一些。这种差异是PHP特有的，并不适用于其他语言，所以请不要修改你的C或Java代码并指望它们能立即变快，没用的。++$i更快是因为它只需要3条指令(opcodes)，$i++则需要4条指令。后置递增实际上会产生一个临时变量，这个临时变量随后被递增。而前置递增直接在原值上递增。这是最优化处理的一种，正如Zend的PHP优化器所作的那样。牢记这个优化处理不失为一个好主意，因为并不是所有的指令优化器都会做同样的优化处理，并且存在大量没有装配指令优化器的互联网服务

提供商（ISPs）和服务器。

40. 不要随便就复制变量

有时候为了使 PHP 代码更加整洁，一些 PHP 新手（包括我）会把预定义好的变量复制到一个名字更简短的变量中，其实这样做的结果是增加了一倍的内存消耗，只会使程序更加慢。试想一下，在下面的例子中，如果用户恶意插入 512KB 字节的文字到文本输入框中，这样就会导致 1MB 的内存被消耗！

BAD:

$description = $\_POST['description'];

echo $description;

GOOD:

echo $\_POST['description'];

41 使用选择分支语句

switch case好于使用多个if，else if语句,并且代码更加容易阅读和维护。

42.在可以用file\_get\_contents替代file、fopen、feof、fgets

在可以用file\_get\_contents替代file、fopen、feof、fgets等系列方法的情况下，尽量用 file\_get\_contents，因为他的效率高得多！但是要注意file\_get\_contents在打开一个URL文件时候的PHP版本问题；

43.尽量的少进行文件操作，虽然PHP的文件操作效率也不低的；

44.优化Select SQL语句，在可能的情况下尽量少的进行Insert、Update操作(在update上，我被恶批过)；

45.尽可能的使用PHP内部函数

46.循环内部不要声明变量，尤其是大变量：对象

(这好像不只是PHP里面要注意的问题吧？)；

47.多维数组尽量不要循环嵌套赋值；

48.foreach效率更高，尽量用foreach代替while和for循环；

49.“用i+=1代替i=i+1。符合c/c++的习惯，效率还高”；

50.对global变量，应该用完就unset()掉；

51 并不是事必面向对象(OOP)，面向对象往往开销很大，每个方法和对象调用都会消耗很多内存。

52 不要把方法细分得过多，仔细想想你真正打算重用的是哪些代码？

53 如果在代码中存在大量耗时的函数，你可以考虑用C扩展的方式实现它们。

54、打开apache的mod\_deflate模块，可以提高网页的浏览速度。

（提到过echo 大变量的问题）

55、数据库连接当使用完毕时应关掉，不要用长连接。

56、split比exploade快

split()

0.001813 - 0.002271 seconds (avg 0.002042 seconds)

explode()

0.001678 - 0.003626 seconds (avg 0.002652 seconds)

Split can take regular expressions as delimiters, and runs faster too. ~23% on average.