PHP 开发规范

公司	美博雅（MEIBOYA）
项目	美肤志（meifuzhi.com）
编者	曲显平
审核	赵楠
发布日期	2012-07-11
版本	V1r4

编码规范

一、 命名规则

1. 名副其实

在为任何一个变量、函数、类命名前，首先要明确地想清楚它是做什么的，然后再为它取一个直达其意的名字。如果在看到一个命名的五秒钟内，你还是想不起来它是做什么的话，那么这个命名就是糟糕的；如果还需要查手册才能明白它的含义时，这个命名的糟糕程度就与#$%^&&%*￥（*无异。

坏例子： $u = ...

好例子： $userName = ...

2. 变量名

（1） 统一使用驼峰式，首字母小写，例如：$fileName

（2） 如果是类的保护或私有变量，应在变量标记$后以(_)下划线开头，例如：$_fileName

（3） 全局常量或静态变量，一般使用全大写，(_)下划线分隔的命名方式，例如：$FILE_NAME

3. 方法和函数名

（1） 类中的方法必须显示声明（public/private/protected）

（2） 统一使用驼峰式，首字母小写，例如：

```
function findByPrimaryKey()
```

（3） 通常每个方法和函数都是执行一个动作的，所以对它们以动宾结构进行命名会更清楚地说明它们是做什么的：比如用 checkForErrors()代替 errorCheck()，用

dumpDataToFile()代替 dataFile(),这么做也可以使功能和数据变得更易区分

常用动词参考定义：set、get、check、save、remove、create、update、find、open、close、show、view、add、delete

（4） 保护或私有方法，总是以 _ 下划线开头，这样可以让人清楚地了解其作用域

例如：

```
class NameOneTwo
{
    public function doIt() {
    };
    private function _handleError() {
    };
}
```

4. 类名

（1） 统一使用驼峰式，首字母大写，例如：class DiaryFactory

（2） 类名应该是名词或名词短语， 如 Customer、WikiPage、Account，避免使用 Manager、Processor、Data 或 Info 这样不明含义的类名 ,类名也不应该是动词。

5. 缩写词

缩写词应该使用首字母大写，其余字母小写的方式来书写命名。

例如：使用 GetHtmlStatistics，而不是用 GetHTMLStatistics

6. 关键词

public、protected、private 这类关键词应统一使用小写

7. 文件与文件夹

（1） 文件名与类名保持一致，增加 .php 后缀

（2） 文件夹名统一使用小写字母

二、 格式化

8. 大括号 {}

（1） 类声明之后的 {} 统一在下一行起首使用，例如：

```
class UserMessage extends Message
{
……
}
```

（2） 函数声明之后的 {} 统一在下一行起首使用，例如：

```
public static function model($className=__CLASS__)
{
……
}
```

（3） If、else、foreach、while、for 等关键词后的 {} 统一在关键词后空一格使用，例如：

```
if (false === $abc) {
    ;
} else {
    ……
}
```

（4） 关键词后的大括号不能被省略，例如：

```
if (false === $a) {
    do();
}
```

而不是

```
if (false === $a) do();
```

9. 排版（缩进、制表符、空格）

（1） 缩进统一使用 TAB，一个 TAB 使用 4 个空格宽度，变量内部（如多行的数组元素）的对齐使用空格

（2） 变量赋值尽量保持优雅美观，“=“和”=>”可尽量对齐，例如：

```
$abcdef = ‘123’;
$abc    = ‘1’;
$a      = ‘1234567’;
$arr    = array(
    ‘abc’   => ‘123’,
    ‘b’     => ‘1’,
    ‘cdefg’ => ‘123456’,
);
```

（3） Vi 可使用“行数==”这种命令进行排版，其它 IDE 同理

（4） 函数内部缩进层数应控制在<=3 层，>3 层应该考虑拆分代码

（5） 文件必须是 linux 格式，换行符为\n，而不是\r 或\n\r（尤其注意在 windows 下的 IDE 编辑器经常会出现^M 这种行结束符，需要消灭掉，否则下次 vi 上编辑该文件可能会引起错误），文件编码必须是 UTF-8

10. 小括号、关键词和函数

（1） 小括号里面的内容和括号之间不要留有空格

（2） 关键词后的小括号应加一空格，只有函数名才是紧贴的，这样好区分；

例如：

```
if ($abc > 100) {
```

和

```
public function abc($abc)
```

（3） 函数如果具有多个参数，参数之间使用一个逗号+空格分隔

例如：

```
function abc($a, $b, $c)
```

（4） 不要在 return 返回的语句中使用小括号（return 是语法结构，不是函数，使用()实际上是进行了一次运算，再把运算结果进行返回，这种情况应拆分成两条语句）

11. ==和===

（1） 常量放在等号/不等号左边，例如 if ('something' == $errorNum) 。以防少写了 = 等号的错误无法被及时发现（可以直接被语法检查发现）

（2） 尽量使用===，能有助于你了解变量的值，否则==用多了，你根本分辨不出 false、0、null、空字符串 等的区别

12. 关于 ?:

（1） 把 ? 前的条件放到括号内，和其它代码分离

（2） : 冒号前后的动作尽量简单，复杂的应该封装成函数后使用

（3） 太长的可以将其分成三行展示

三、 字符串引用

（1） 纯字符串使用单引号界定（单引号比双引号效率高）

（2） 字符串中引用变量时，变量用 {} 界定，例如：{$abc} （可以避免 "$abc" 到底是 $a.'bc' 还是 $ab.'c' 的疑问）

四、 忠告

13. 避免魔鬼数字

既无注释，又未定义的赤裸裸的数字就是魔鬼数字，很多人写完之后一个月就不记得他

的意义是什么，例如：

```
if (22 === $foo) {
    do();
} else if (19 === $foo) {
    ….
```

应该使用 define()、CONST、或者至少用一个有意义的变量名称来表示；

14. 关于 else if

（1） 不要使用 elseif ，而必须使用 else if（有空格的），他们两个在不使用 {} 的情况下会有区别，因此强制使用 else if（有空格的）也是保证使用{}的一种方式

（2） else / else if 和上个判断的 } 在同一行，并且之间有空格，例如：

```
} else if (false === f()) {
```

15. 关于 for 和 while

少用 for 和 while 语法，因为 PHP 提供了 foreach，基本取代了 for，很大程度上也解决了 while 的问题，即使你十分不得已要使用这两个语法，请一定注意异常判断和死循环；

16. 关于 continue 和 break

continue 和 break 是变相的 goto 方法，在循环中应慎用，更切忌一起使用

五、 注释

（1） 注释的作用：1).意图的描述；2).警醒后人（风险或 TODO）

（2） 注释的使用（注意单行注释避免使用/* … */，块注释的正文部分每行前应保留*）

a) 文档注释

```
/**
 * ……
 */
```

b) 单行注释

```
// ……
```

c) 块注释（>=2 行）

```
/*
 * ......
 */
```

（3） 注释要准确，胡言乱语的注释会让程序员发疯（经常看到胡乱粘贴过来的各种驴唇不对马嘴的注释）

（4） 注释要及时维护，过期的注释也是程序员的噩梦

（5） 如果代码存在改进的空间的，需要用 “@todo 2012-03-06 tony XXXXX” 的形式来标识出 todo 的内容，这种标识可以在后续统一找到

（6） 代码无用请立刻删除，不应该出现注释掉的代码（后人不知如何处理）

（7） 注释仅仅是注释，它不能掩饰糟糕的代码

17. 文件注释

文件注释是对一个文件的整体概括描述，应该至少包含如下几项：

（1） 用途描述

（2） 所在文件夹/包路径

（3） copyright 版权信息

（4） @file 文件名

（5） @since 创建日期

（6） @author 作者

（7） @version Id 开启 svn 的$Id 模式（自动更新最后修改时间和修改人）

```
/**
 * meifuzhi - controller - 用户消息控制类 *
```

```
* ======================================
* Copyright 2011-2012 meifuzhi.com
* ======================================
*
* @file MessageController.php
* @since 2012-07-10
* @author vincent.qu@meifuzhi.com
* @version $Id: MessageController.php 455 2012-07-10 06:40:34Z quxianping $
*
*/
```

18. 类注释

由于在我们的 PHP 编码规范中，要求每个文件只能包含一个类，那么文件注释其实也就可以作为类注释，类不必单独维护注释

19. 函数或方法注释

函数或方法注释是对一个函数或方法的描述，应该至少包含如下几项：

（1） 用途描述

（2） @param 参数列表（标明变量类型、变量描述）

（变量类型的表示采用下列语法（与 PHP 语法中定义保持一致）：

int 整型

bool 布尔型

string 字符串

array 数组

mixed 上述类型都有可能

（3） @return 返回值列表（标明返回值类型、返回值描述）

（4） @since 创建时间

（5） @author 作者

（6） @lastupdate 最后修改者和最后修改时间

```
/**
 * 获取用户的消息列表
 * @param int $uid 用户ID
 * @return array $arrayName 用户名称列表
 * @since 2012-07-10
 * @author vincent.qu@meifuzhi.com
 * @lastupdate 2012-07-12 vincent.qu
 */
```

六、 其它

20. PHP 文件

（1） 统一使用<?php 开头，前面不应有任何多余字符，且不要使用 <? 这种简易方式（这种方式在 PHP 配置中已经禁用），否则可能和某些其它语言的文件混淆（如 xml）

（2） 文件结尾不要使用 ?> ，以避免文件末尾带有其它字符造成的文件解析或代码无法执行等问题

21. 异常处理

（1） try/catch 使用在可能抛出异常的代码段外层，原则上不同类型的多个异常应该分别使用 try/catch 捕获，同类型的多个异常可以使用同一组 try/catch 捕获，尽量避免多层的 try/catch 嵌套，很难追查定位问题

（2） 对于可能抛出异常的系统调用、数据库操作、缓存操作等必须有 try/catch 错失，以免给用户保留系统错误信息

（3） 捕获异常的类型统一使用 CException

设计风格

七、 类

（1） 一个 php 文件只包含一个类，且类名与文件名保持一致

（2） 避免在类的构造方法中做真实的工作，在构造方法中只初始化变量或做任何不会有失误的事情，因为构造方法不能返回错误

（3） 方法应与类直接相关，切忌把不相关的方法放到类中（新建一个类更好一些）

八、 函数或方法

（1） 一个函数的长度在 20 行以内为佳 ,如果超过了 50 行 ,那么它几乎一定应该被重构，别人难以忍受

（2） 确保一个函数或方法只做一件事情

（3） 参数应尽量少，参数超过 5 个就很难能令人看懂了

（4） 不可容忍出现重复的代码逻辑，如果出现了重复代码逻辑，就应该及时抽象成统一函数或方法封装使用，否则后患无穷

九、 系统调用

（1） 所有系统调用的返回值都应该检查，除非你确认要忽略错误

（2） 为每条系统错误消息定义好错误输出的文本

十、 不要采用缺省方法测试非零值

使用 `if (false !== f())` 而不是 `if (f())`

十一、 布尔逻辑类型

大部分函数在执行失败时返回 0，执行正常可能返回其它非 0 值，所以不要用 1（TRUE）的等式来检查返回值，应该用 0（FALSE）的不等式来检查

十二、 避免嵌入式的赋值

```
while ($a != ($c = getchar())) {
        …
```

甚至各种 ++/-- 齐用，谁都受不了

老老实实地改成：

```
$c = getchar();
while ($a != $c) {
    …
```

十三、 面向对象

（1） 是否使用了设计模式？

很多人都觉得过分追求设计模式反而会影响开发效率，但作为过来人，我可以直白地说，不使用优秀的设计模式，软件可能会死得很快

代码范例

```
<?php
/**
 * meifuzhi - models - diary - 用户日志类
 *
 * ========================================
 * Copyright 2011-2012 meifuzhi.com
 * ========================================
 *
 * @file UserDiary.php
 * @since 2012-07-06
```

```
 * @author vincent.qu@meifuzhi.com
 * @version $Id: UserDiary.php 455 2012-07-10 06:40:34Z quxianping $
 *
 */

define('SAVE_DIARY_ERROR_CODE', -1);

class UserDiary extends DaoModel
{
    private static $TABLE_NAME = 'usr_diary';
    protected static $_id = 0;

    /**
     * 日志类构造函数
     * @param int $diaryId 日志ID
     * @since 2012-07-06
     * @author vincent.qu@meifuzhi.com
     * @lastupdate 2012-07-11 vincent.qu
     */
    public function __construct($diaryId = 0)
    {
        $this->_id = $diaryId;
    }

    /**
     * 查看日志详情
     * @return array $diaryDetail 日志详情
     * @since 2012-07-06
     * @author vincent.qu@meifuzhi.com
     * @lastupdate 2012-07-11 vincent.qu
     */
    public function viewDetail()
    {
        $db = $this->getDbConnection();
        $table = self::$TABLE_NAME;
        $sql = "SELECT * FROM `{$table}` WHERE `id` = :ID";
        $command = $db->createCommand($sql);
        $command->bindValue(':ID', $this->_id, PDO::PARAM_INT);
        $diaryDetail = $command->queryRow();
        return $diaryDetail;
    }

    /**
     * 保存日志
```

```
     * @param array $diaryDetail 日志内容
     * @return int $flag 执行成功或失败
     * @since 2012-07-06
     * @author vincent.qu@meifuzhi.com
     * @lastupdate 2012-07-11 vincent.qu
     */
    public function saveDiary($diaryDetail)
    {
        try {
            $db = $this->getDbConnection();
            //开启事务
            $db->beginTransaction();
            $this->_saveIndex($diaryDetail);
            $this->_saveContent($diaryDetail['content']);
            $db->commit();
        } catch (CException $e) {
            Log::file($e->getMessage(), 'error', 'system');
            $db->rollback();
            return SAVE_DIARY_ERROR_CODE;
        }
        return 0;
    }

    /**
     * 保存日志索引
     * @param array $diaryDetail 日志详情
     * @return int $flag 执行成功或失败
     * @since 2012-07-10
     * @author vincent.qu@meifuzhi.com
     * @lastupdate 2012-07-11 vincent.qu
     */
    protected function _saveIndex($diaryDetail)
    {
        if (true === $diaryDetail['is_draft']) {
            ;
        } else if (true === $diaryDetail['is_private']) {
            ;
        } else {
            ;
        }
        return 0;
    }

    /**
```

```
     * 保存日志正文
     * @param string $diaryContent 日志正文
     * @return int $flag 执行成功或失败
     * @since 2012-07-10
     * @author vincent.qu@meifuzhi.com
     * @lastupdate 2012-07-11 vincent.qu
     */
    protected function _saveContent($diaryContent)
    {
        //....
        return 0;
    }
}
```

数据库设计

十四、 存储引擎

存储引擎一般情况下使用 InnoDB，在某些情况下可使用 MyISAM；

在如下几类情况下必须使用 InnoDB：

（1） 需要支持事务（MyISAM 不支持）

（2） 需要外键（MyISAM 不支持）

（3） 高并发写（MyISAM 更新为表级锁，InnoDB 为行级锁，超高并发读写时 MyISAM 锁竞争比较严重）

在如下几种情况较适用于 MyISAM：

（1） 读多写少（MyISAM 性能高）

（2） 需要经常备份和恢复全表数据（MyISAM 只要 cp 数据文件即可）

十五、 表设计

（1） 表名，统一使用小写字母，单词之间下划线分隔，如 usr_diary（注意：MySQL 库表

大小写敏感）

（2） 字段名称，统一使用小写字母，单词直接下划线分隔，主键一般使用 id，设置自增（注意：MySQL 字段名大小写不敏感）

（3） 库、表、字段避免用保留字命名。

（4） 表与表的对应关系，关联 id 字段一般使用 :tablename_id，例如：有一个表是 usr_diary，里面主键为 id，代表日志 id，usr_diary_image 表中存有日志对应的图片，那么在图片表中关联的日志 id 字段应起名作 usr_diary_id，代表 usr_diary 表的主键 id

（5） 尽量避免使用外键关联。外键关联当然有很多优点，例如可以保证数据的一致性；但是当外键关系过多时，一个是会严重影响 MySQL 执行效率，造成大规模锁表；另外，MySQL 本身也处理不好已经有的各种外键嵌套关系，严重地会造成死锁或者 dump 出的数据无法再导回 MySQL，应该尽量避免使用

（6） 数据库里尽量不存图片或文件，存储效率低

（7） 字符集统一使用 UTF8，校对规则 utf8_general_ci

十六、 字段类型

（1） INT/ENUM 远好于 CHAR，无论是存储空间、建索引速度、查询效率，所以如果有可能可尽量用 INT/ENUM/SET 取代 CHAR

（2） 避免使用 NULL 字段，它是索引的噩梦，性能差

（3） TEXT 类型性能远低于 VARCHAR，建议拆分到单独的表存储

十七、 SQL 编写

（1） 避免 SQL 注入，外界传入的参数一律不可信，统一使用 bindValue 或者 AR 类赋值方

式与数据库交互，切忌直接拼写 SQL 查询

（2） SQL 中的所有库、表、字段名，需使用 `` （小撇）界定，可以避免造成关键词冲突的问题。例如：order 是 SQL 中的关键字，如果你写一个 SQL： SELECT * FROM order WHERE order > 1 ORDER BY order DESC LIMIT 1; 那么 MySQL 就疯了；正确的方式：SELECT * FROM `order` WHERE `order` > 1 ORDER BY `order` DESC LIMIT 1;

（3） 所有 SQL 关键字都要使用大写

（4） SELECT * 请少用。所有人都知道服务器性能瓶颈一般都在 IO，因为 IO 速度 << 内存速度 << CPU 寄存器，并且网络带宽也应该尽量地节省

（5） DISTINCT、ORDER BY 和 GROUP BY 相当消耗资源，慎用

（6） 编写 SQL 时，应尽量避免大型 SQL，减少多表连接操作。简单 SQL 的好处：缓存命中率高，能用多核 CPU，减少锁表时间，有利于高并发

（7） 尽可能命中覆盖索引（高阶题目请自学）

（8） 关于 OR。同一字段用 OR，使用 IN 代替（OR 效率 O(n)，IN 效率 O(logn)，IN 后面的数目建议小于 200）；不同字段用 OR，使用 UNION 代替

（9） 避免负向查询（NOT、!=、<>、!<、!>、NOT EXISTS、NOTIN、NOT LIKE 等）

（10） 避免使用 % 前缀模糊查询，因为：无法使用索引，只能全表扫描

（11） COUNT(*)资源开销大，尽量少用

（12） LIMIT 10000,10，偏移量越大越慢

（13） 大批量插入数据，使用批量 INSERT，而不是 FOR 循环

（14） 避免使用大事务，不主动对数据库加锁

（15） 尽量避免子查询，一般用 WHERE IN 或 JOIN 替代

十八、 查询方式

（1） 尽量不在数据库做运算，虽然你懂得各种复杂的 SQL 语法，但是你随便查查可以，如果要面对超高并发的流量，数据库会死得很惨

（2） 拒绝 3Big：大 SQL、大事务、大批量

（3） 使用 Yii 框架原生的 AR 操作效率较低，一般适用于单条数据的查询或写入，批量数据的查询或写入一般不应使用 AR 操作，直接使用 CDbCommand 的方式

（4） Yii 框架原生 AR 消耗 PHP 内存巨大，大批量数据时千万慎用！

十九、 表规模

（1） 纯 INT 表控制在千万级

（2） 含 CHAR 表控制在百万级

（3） 单表不超 20 个 CHAR(10)字段，不超 50 个纯 INT 字段

开发测试流程

二十、 开发

22. 开始

（1） 是否有了 svn 权限？不会用的，请尽快去学习吧。

（2） 没有其它人在并行开发你的模块，请使用主干开发；若有其它人也在并行开发你的模块，且和你又不是一个版本，请使用分支开发；

（3） 当前开发环境是 linux+ngnix+mysql+php+memcached+yii+smarty…

（4） Linux 开发机是大家共用，所以有覆盖/删除文件或者消耗服务器性能的操作时，请在不影响其它同学的前提下慎重操作；

（5） 开发环境不允许连接线上数据库/缓存等；

23. 进阶

（6） SVN 提交代码时必须认真填写版本描述（-m）

（7） 系统生成数据（包括上传文件、图片等）统一存放在 htdocs 下的 data 目录中

（8） 系统日志（用户操作、系统自身）统一存放在与 htdocs 平级的 logs 目录中

（9） 注意所有用户的增删改操作，应都留有操作日志

二十一、 测试

（1） 代码 review 了吗？你可以请他人帮你 review 一下

（2） 如果有时间，推荐进行单元测试

（3） 正式测试：开发者自测+产品设计师+用户试用

二十二、 上线

（1） 你的模块是否有自动上线脚本？没有的话请赶快写一个

（2） 配置文件的修改千万注意，一失足可能千古恨

（3） 上线完成后注意线上检查，保证原有功能和新功能 OK

文档

二十三、 文档命名

文档名称示例：《PHP 开发规范_20120710V1r2-draft_tony_comments》

注解：

（1） 20120710：创建大版本号的日期

（2） V1r2：V 为 version，表示大版本号，从 1 开始；r 为 revision，表示小版本号，从 1 开始增加，每次文档发给别人时，该数字+1

（3） -draft：表明为草案，在正式发布时，这个去掉即可

（4） _tony_comments：任何的说明，可以表示谁谁谁做了什么，空格用_代替