1 编写目的 3
2 整体要求 3
3 安全规范 3
3.1 包含文件 3
3.1.1 命名规则 3
3.1.2 存放规则 3
3.2 安全规则 3
3.3 一些针对 PHP 的规则 4
3.4 其它处理规则 4
3.4.1 对输入参数值进行转义处理 4
3.4.2 操作大 HTML 文本 4
4 编码规范 4
4.1 命名规范 4
4.1.1 变量命名 4
4.1.1.1 普通变量 5
4.1.1.2 静态变量 5
4.1.1.3 局部变量 5
4.1.1.4 全局变量 5
4.1.1.5 全局常量 5
4.1.1.6 session 变量 5
4.1.2 类 5
4.1.3 方法或函数 6
4.1.4 缩写词 6
4.1.5 数据库表名 6
4.1.6 数据库字段 6
4.2 书写规则 6
4.2.1 代码缩进 6
4.2.2 大括号{ }书写规则 6
4.2.3 小括号( )和函数、关键词等 7
4.2.4 = 符号书写 7
4.2.5 if else swith for while 等书写 7
4.2.6 类的构造函数 7
4.2.7 语句断行，每行控制在 80 个字符以内 7
4.2.8 使用 define 定义条件句中比较用的数字 8
4.2.9 true/false 和 0/1 判断 9
4.2.10 避免嵌入式赋值 9
4.2.11 错误返回检测规则 9
4.3 程序注释 9

4.3.1 程序头注释块 9
4.3.2 类的注释 10
4.3.3 函数和方法的注释 10
4.3.4 变量或者语句注释 11
4.4 其他规范（建议） 11
4.4.1 php 代码标记 11
4.4.2 程序文件名、目录名 11
4.4.3 PHP 项目通常的文件目录结构 11
4.4.4 PHP 和 HTML 代码的分离问题 12
4.4.5 PHP 项目开发中的程序逻辑结构 12
5 特定环境下 PHP 编码特殊规范 12
5.1 变量定义 12
5.2 引用的使用 12
5.3 变量的输入输出 13

1 编写目的

为了更好的提高技术部的工作效率，保证开发的有效性和合理性，并可最大程度的提高程序代码的可读性和可重复利用性，指定此规范。开发团队根据自己的实际情况，可以对本规范进行补充或裁减。

2 整体要求

本规范包含了 PHP 开发时程序编码中**命名规范、代码缩进规则、控制结构、函数调用、函数定义、注释、包含代码、PHP 标记、文件头的注释块、CVS 标记、URL 样例、常量命名**等方面的规则。

3 安全规范

3.1 包含文件

3.1.1 命名规则

提取出来具有通用函数的包含文件，文件后缀以.inc 来命名，表明这是一个包含文件。
如果有多个.inc 文件需要包含多页面，请把所有.inc 文件封装在一个文件里面，具体到页面只需要包换一个.inc 文件就可以了
如:xxx_session.inc
xxx_comm..inc
xxx_setting.inc
mysql_db.inc
把以上文件以一下方式，封装在 xxx.basic.inc 文件里面
require_once("xxx_session.inc");
require_once("xxx_comm.inc");
require_once("xxx_setting.inc");
require_once("mysql_db.inc");
注：是否需要封装到一个文件，视情况而定，如果每个 inc 的功能是分散到不同的页面使用的话，就不建议封装。

3.1.2 存放规则

一般包含文件不需要直接暴露给用户，所以应该放在 Web Server 访问不到的目录，避免因为配置问题而泄露设置信息。

3.2 安全规则

请参考产品安全检查表。
输入和输出
检查是否做了 HTML 代码的过滤
可能出现的问题 :如果有人输入恶意的 HTML 代码, 会导致窃取 cookie，产生恶意登录表单，和破坏网站
检查变量做数据库操作之前是否做了 escape
可能出现的问题 : 如果一个要写入查询语句的字符串变量包含了某些特殊的字符，比如引号(' ,")或者分号(;) 可能造成执行了预期之外的操作。

建议采用的方法：使用 mysql_escape_string() 或实现类似功能的函数。
检查输入数值的合法性
可能出现的问题：异常的数值会造成问题。如果对输入的数值不做检查会造成不合法的或者错误的数据存入 UDB、存入其它的数据库或者导致意料之外的程序操作发生。
举例：
如果程序以用户输入的参数值做为文件名，进行文件操作，恶意输入系统文件名会造成系统损毁。
核实对 cookie 的使用以及对用户数据的处理
可能出现的问题：不正确的 cookie 使用可能造成用户数据泄漏
访问控制
对内部使用的产品或者供合作方使用的产品，要考虑增加访问控制
logs
确保用户的保密信息没有记在 log 中(例如：用户的密码)
确保对关键的用户操作保存了完整的用户访问记录
https
对敏感数据的传输要采用 https

3.3 一些针对 PHP 的规则

设置 register_globals = off (Y!PHP 已经禁止了 register_globals，如果你使用 Y!PHP 可以不考虑这项设置)
设置 error_reporting = E_ALL (Y!PHP 的缺省设置)，并且要修正所有的 error 和 warning
将实际的操作放在被引用的文件中。把引用文件放到不可以被直接浏览的目录下

3.4 其它处理规则

3.4.1 对输入参数值进行转义处理

页面接到参数需要 SQL 操作，这时候需要做转义，尤其需要注意";"。
如：$a = ” Let's go ”；
$sql = “Insert into tmp(col) values('$a')”；
这种情况出现错误的不确定性。

3.4.2 操作大 HTML 文本

很多时候需要存放一大段 HTML 文本供页面使用，象用户定制页头页脚等。
需要剔除脚本标记，避免执行恶意 php 代码。
转换”<"">“号，保证代码完整。

4 编码规范

4.1 命名规范

制定统一的命名规范对于项目开发来说非常重要，不但可以养成程序员一个良好的开发习惯，还能增加程序的可读性、可移植性和可重用性，还能很好的提高项目开发的效率。

4.1.1 变量命名

变量命名分为普通变量、静态变量、局部变量、全局变量、Session 变量等方面的命名规则。

4.1.1.1 普通变量

普通变量命名遵循以下规则：
a． 所有字母都使用小写；
b． 对于一个变量使用多个单词的，使用'_'作为每个词的间隔。
例如：$base_dir、$red_rose_price 等

4.1.1.2 静态变量

静态变量命名遵循以下规则：
a． 静态变量使用小写的 s_开头；
b． 静态变量所有字母都使用小写；
c． 多个单词组成的变量名使用'_'作为每个词的间隔。
例子：$s_base_dir、$s_red_rose_prise 等。

4.1.1.3 局部变量

局部变量命名遵循以下规则：
a． 所有字母使用小写；
b． 变量使用'_'开头；
c． 多个单词组成的局部变量名使用'_'作为每个词间的间隔。
例子：$_base_dir、$_red_rose_price 等。

4.1.1.4 全局变量

全局变量应该带前缀'g',知道一个变量的作用域是非常重要的。
例如
global $gLOG_LEVEL;
global $gLOG_PATH;

4.1.1.5 全局常量

全局变量命名遵循以下规则：
a． 所有字母使用大写
b． 全局变量多个单词间使用'_'作为间隔。
例子：$BASE_DIR、$RED_ROSE_PRICE 等。

4.1.1.6 session 变量

session 变量命名遵循以下规则：
a． 所有字母使用大写；
b． session 变量名使用'S_'开头；
c． 多个单词间使用'_'间隔。
例子：$S_BASE_DIR、$S_RED_ROSE_PRICE 等。

4.1.2 类

php 中类命名遵循以下规则：
a． 以大写字母开头；
b． 多个单词组成的变量名，单词之间不用间隔，各个单词首字母大写。

例子：class MyClass 或 class DbOracle 等。

4.1.3 方法或函数

方法或函数命名遵循以下规则：
a． 首字母小写；
b． 多个单词间不使用间隔，除第一个单词外，其他单词首字母大写。
例子：function myFunction ()或 function myDbOracle ()等。

4.1.4 缩写词

当变量名或者其他命名中遇到缩写词时，参照具体的命名规则，而不采用缩写词原来的全部大写的方式。
例子：function myPear（不是 myPEAR） functio getHtmlSource（不是 getHTMLSource）。

4.1.5 数据库表名

数据库表名命名遵循以下规范：
a． 表名均使用小写字母；
b． 对于普通数据表，使用_t 结尾；
c． 对于视图，使用_v 结尾；
d． 对于多个单词组成的表名，使用_间隔；
例子：user_info_t 和 book_store_v 等

4.1.6 数据库字段

数据库字段命名遵循以下规范：
a． 全部使用小写；
b． 多个单词间使用_间隔。
例子：user_name、rose_price 等。

4.2 书写规则

书写规则是指在编写 php 程序时，代码书写的规则，包括缩进、结构控制等方面规范：

4.2.1 代码缩进

在书写代码的时候，必须注意代码的缩进规则，我们规定代码缩进规则如下：
a． 使用 4 个空格作为缩进，而不使用 tab 缩进（对于 ultraedit，可以进行预先设置）
例子：

```
for ( $i=0;$i<$count;$i++ )
{
echo "test";
}
```

4.2.2 大括号{ }书写规则

在程序中进行结构控制代码编写，如 if、for、while、switch 等结构，大括号传统的有两种书写习惯，分别如下：
a．{直接跟在控制语句之后，不换行，如

```
for ($i=0;$i<$count;$i++) {
```

```
echo "test";
}
```
b．{在控制语句下一行，如

```
for($i=0;$i<$count;$i++)
{
echo "test";
}
```

其中，a 是 PEAR 建议的方式，但是从实际书写中来讲，这并不影响程序的规范和影响用 phpdoc 实现文档，所以可以根据个人习惯来采用上面的两种方式，但是要求在同一个程序中，只使用其中一种，以免造成阅读的不方便。

4.2.3 小括号( )和函数、关键词等

小括号、关键词和函数遵循以下规则：

a． 不要把小括号和关键词紧贴在一起，要用一个空格间隔；如 if ($a<$b)；

b． 小括号和函数名间没有空格；如$test = date("ymdhis")；

c． 除非必要，不要在 Return 返回语句中使用小括号。 如 Return $a；

4.2.4 = 符号书写

在程序中=符号的书写遵循以下规则：

a． 在=符号的两侧，均需留出一个空格；如$a = $b 、if ($a = = $b)等；

b． 在一个申明块，或者实现同样功能的一个块中，要求=号尽量上下对其，左边可以为了保持对齐使用多个空格，而右边要求空一个空格；如下例：

```
$testa = $aaa;
$testaa = $bbb;
$testaaa = $ccc;
```

4.2.5 if else swith for while 等书写

对于控制结构的书写遵循以下规则：

a． 在 if 条件判断中，如果用到常量判断条件，将常量放在等号或不等号的左边，例如：if (6 == $errorNum)，因为如果你在等式中漏了一个等号，语法检查器会为你报错，可以很快找到错误位置，这样的写法要多注意；

b． switch 结构中必须要有 default 块；

c． 在 for 和 wiile 的循环使用中，要警惕 continue、break 的使用，避免产生类似 goto 的问题；

4.2.6 类的构造函数

如果要在类里面编写构造函数，必须遵循以下规则：

a． 不能在构造函数中有太多实际操作，顶多用来初始化一些值和变量；

b． 不能在构造函数中因为使用操作而返回 false 或者错误，因为在声明和实例化一个对象的时候，是不能返回错误的；

4.2.7 语句断行，每行控制在 80 个字符以内

在代码书写中，遵循以下原则：

a． 尽量保证程序语句一行就是一句，而不要让一行语句太长产生折行；

b． 尽量不要使一行的代码太长，一般控制在 80 个字符以内；
c． 如果一行代码太长，请使用类似 .= 的方式断行书写；
d． 对于执行数据库的 sql 语句操作，尽量不要在函数内写 sql 语句，而先用变量定义 sql 语句，然后在执行操作的函数中调用定义的变量；
例子：

```
$sql = "SELECT username,password,address,age,postcode FROM test_t ";
$sql .= " WHERE username='aaa'";
$res = mysql_query($sql);
```

4.2.8 使用 define 定义条件句中比较用的数字

一个在源代码中使用了的赤裸裸的数字是不可思议的数字，因为包括作者，在三个月内，没人它的含义。例如：

```
if       (22 == $foo)
{
start_thermo_nuclear_war();
}
else if (19 == $foo)
{
refund_lotso_money();
}
else
{
cry_cause_im_lost();
}
```

你应该用 define()来给你想表示某样东西的数值一个真正的名字，而不是采用赤裸裸的数字，例如：

```
define("PRESIDENT_WENT_CRAZY", "22");
define("WE_GOOFED", "19");
define("THEY_DIDNT_PAY", "16");

if ( PRESIDENT_WENT_CRAZY == $foo)
{
start_thermo_nuclear_war();
}
else if (WE_GOOFED == $foo)
{
refund_lotso_money();
}
else if (THEY_DIDNT_PAY == $foo)
{
infinite_loop();
}
else
{
```

```
happy_days_i_know_why_im_here();
}
```

4.2.9 true/false 和 0/1 判断

遵循以下规则：

a． 不能使用 0/1 代替 true/false，在 PHP 中，这是不相等的；

b． 不要使用非零的表达式、变量或者方法直接进行 true/false 判断，而必须使用严格的完整 true/false 判断；

如：不使用 if ($a) 或者 if (checka()) 而使用 if (FALSE != $a)或者 if (FALSE != check())

4.2.10 避免嵌入式赋值

在程序中避免下面例子中的嵌入式赋值：

不使用这样的方式：

```
while ($a != ($c = getchar()))
{
process the character
}
```

4.2.11 错误返回检测规则

检查所有的系统调用的错误信息，除非你要忽略错误。

为每条系统错误消息定义好系统错误文本，并记录错误 LOG。

4.3 程序注释

每个程序均必须提供必要的注释，书写注释要求规范，参照 PEAR 提供的注释要求，为今后利用 phpdoc 生成 php 文档做准备。程序注释的原则如下：

a． 注释中除了文件头的注释块外，其他地方都不使用//注释，而使用/* */的注释；

b． 注释内容必须写在被注释对象的前面，不写在一行或者后面；

4.3.1 程序头注释块

每个程序头部必须有统一的注释块，规则如下：

a． 必须包含本程序的描述；

b． 必须包含作者；

c． 必须包含书写日期；

d． 必须包含版本信息；

e． 必须包含项目名称；

f． 必须包含文件的名称；

g． 重要的使用说明，如类的调用方法、注意事项等；

参考例子如下：

```
<?php
//
// +----------------------------------------------------------+
// | PHP version 4.0                                          |
// +----------------------------------------------------------+
// | Copyright (c) 1997-2001 The PHP Group                    |
```

```
// +----------------------------------------------------------+
// | This source file is subject to   of the PHP license,      |
// | that is bundled with this packafile LICENSE, and is       |
// | available at through the world-web at                      |
// | http://www.php.net/license/2_02.txt.                       |
// | If you did not receive a copy of the   and are unable to |
// | obtain it through the world-wide-web,end a note to         |
// | license@php.net so we can mail you a immediately.          |
// +----------------------------------------------------------+
// | Authors: Stig Bakken <ssb@fast.no>                         |
// |              Tomas V.V.Cox <cox@idecnet.com>                |
// |                                                            |
// +----------------------------------------------------------+
//
// $Id: Common.php,v 1.8.2.3 2001/11/13 01:26:48 ssb Exp $
```

4.3.2 类的注释

类的注释采用里面的参考例子方式：

```
/**
* @ Purpose:
*  访问数据库的类，以 ODBC 作为通用访问接口
* @Package Name: Database
* @Author: Forrest Gump gump@crtvu.edu.cn
* @Modifications:
* No20020523-100:
* odbc_fetch_into()参数位置第二和第三个位置调换
* John Johnson John@crtvu.edu.cn
* @See: (参照)
*/
class Database
{
……
}
```

4.3.3 函数和方法的注释

函数和方法的注释写在函数和方法的前面，采用类似下面例子的规则：

```
/**
* @Purpose:
*  执行一次查询
* @Method Name: Query()
*
* @Param: string $queryStr SQL 查询字符串
* @Param: string $username  用户名
*
```

* @Author: Michael Lee
*
* @Return: mixed 查询返回值（结果集对象）
*/
function（$queryStr,$username）
{……}

4.3.4 变量或者语句注释

程序中变量或者语句的注释遵循以下原则：
a． 写在变量或者语句的前面一行，而不写在同行或者后面；
b． 注释采用/* */的方式；
c． 每个函数前面要包含一个注释块。内容包括函数功能简述，输入/输出参数，预期的返回值，出错代码定义。
d． 注释完整规范。
e． 把已经注释掉的代码删除，或者注明这些已经注释掉的代码仍然保留在源码中的特殊原因。
f．
例子：
/**
* @Purpose:
* 数据库连接用户名
* @Attribute/Variable Name: db_user_name
* @Type: string
*/
var db_user_name;

4.4 其他规范（建议）

4.4.1 php 代码标记

所有的 php 程序代码块标记均使用

4.4.2 程序文件名、目录名

程序文件名和目录名命名均采用有意义的英文方式命名，不使用拼音或无意义的字母，同时均必须使用小写字母，多个词间使用_间隔。

4.4.3 PHP 项目通常的文件目录结构

建议在开发规范的独立的 PHP 项目时，使用规范的文件目录结构，这有助于提高项目的逻辑结构合理性，对应扩展和合作，以及团队开发均有好处。
一个完整独立的 PHP 项目通常的文件和目录结构如下：
/ 项目根目录
/manage 后台管理文件存放目录
/css 文件存放目录
/doc 存放项目文档
/images 所有图片文件存放路径（在里面根据目录结构设立子目录）

/scripts 客户端 js 脚本存放目录
/tpl 网站所有 html 的模版文件存放目录
/error.php 错误处理文件（可以定义到 apache 的错误处理中）
以上目录结构是通常的目录结构，根据具体应用的具体情况，可以考虑不用完全遵循，但是尽量做到规范化。

4.4.4 PHP 和 HTML 代码的分离问题

对性能要求不是很高的项目和应用，我们建议不采用 PHP 和 HTML 代码直接混排的方式书写代码，而采用 PHP 和 HTML 代码分离的方式，即采用模版的方式处理，这样一方面对程序逻辑结构更加清晰有利，也有助于开发过程中人员的分工安排，同时还对日后项目的页面升级该版提供更多便利。
对于一些特殊情况，比如对性能要求很高的应用，可以不采用模版方式。

4.4.5 PHP 项目开发中的程序逻辑结构

对于 PHP 项目开发，尽量采用 OOP 的思想开发，尤其在 PHP5 以后，对于面向对象的开发功能大大提高。
在 PHP 项目中，我们建议将独立的功能模块尽量写成函数调用，对应一整块业务逻辑，我们建议封装成类，既可以提高代码可读性，也可以提高代码重用性。比如，我们通常将对数据库的接口封装成数据库类，有利于平台的移植。
重复的代码要做成公共的库。（除了我们在 plug-in 产品上遇到的情况，该产品系列有多个相类似的产品，为了尽可能地减少安装包尺寸，不适合将这些产品共用的所有函数做成公共的库）

5 特定环境下 PHP 编码特殊规范

5.1 变量定义

XXX 环境下的 php 代码编写要求所有的变量均需要先申明后使用，否则会有错误信息，对于数组，在使用一个不确定的 key 时，比如先进行 isset()的判断，然后再使用；比如下面的代码：

```
$array = array();
$var = isset($array[3]) ? $array[3] : “”；
```

5.2 引用的使用

引用在程序中使用比较多，为了公用同一个内存，而不需要另外进行复制，XXX 环境下的引用使用时，需要注意下面的情况；
在对函数的输入参数中使用引用时，不能在调用的时候在输入参数前加&来引用，而直接使用该变量即可，同时必须在函数定义的时候说明输入参数来自引用，比如下面的代码：

```
$a = 1；
function ab(&$var)
{
$var ++；
return $var；
}
$b = ab($a) // 注意，此处不能使用 $b = ab(&$a)的方式；
```

echo $b."/n"；
echo $a."/n"；
此时 $a 和$b 都是 2；
XXX 环境下对引用的特殊要求源自 php.ini 文件里面的 allow_call_time_pass_reference 项设置， 对外公开的版本是 On ，这样就可以支持&直接加到调用函数时变量前面进行引用，但是这一方法遭到抗议，并可能在将来版本的 PHP/Zend 里不再支持。受到鼓励的指定哪些参数按引用传递的方法是在函数声明里。你被鼓励尝试关闭这一选项（使用 off，XXX 的所有运行环境下都是 off）并确认你的脚本仍能正常工作，以保证在将来版本的语言里它们仍能工作。

5.3 变量的输入输出

在 XXX 环境下，对 web 通过 GET 或者 POST 方法传递来的参数均要求进行严格的过滤和合法性验证，不推荐使用直接的$_GET、$_POST 或者$_REQUEST 获取，而通过 XXX 的 XXX_yiv 模块提供的方法获取和过滤处理