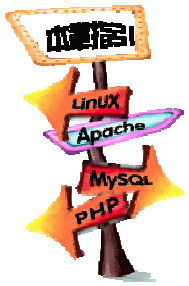
附录 A 编码规范



本书提供的编码规范是以 Zend 官方资料为依据，根据编程原则融合并提炼了多家软件公司长时间积累下来的成熟模式。目的是让软件项目遵守公共一致的标准，帮助那些刚刚入行的 PHP 新手和正在使 PHP 开发的项目组形成良好一致的编程风格，使参与者可以很快的适应环境，减少编码出错的机会，在团队协作开发和项目后期维护中有更高的效率，以达到事半功倍的效果，使项目长远健康的发展。防止部分参与者出于节省时间的需要，自创一套风格并养成终生的习惯，导致其它人在阅读时浪费过多的时间和精力。

文件状态： ［ ］草稿 ［√］正式发布 ［ ］正在修改	文件标识：	编码规范
	当前版本：	1.0
	作　　者：	兄弟连
	完成日期：	2011-07-18

A1 绪论

A1.1 适用范围

本文档提供的代码规则和文档指南不仅适用于《细说 PHP》中的每个实例，也适用于所有 PHP 项目，意在帮助新手在编程风格上养成良好的习惯，也可以作为部分软件公司中项目团队的参考文档，根据自己公司团队的特点进行部分修改。

A1.2 目标

能够遵守公共一致的编码标准对任何开发项目都很重要，特别是在多人的开发团队中，编码标准能帮助确保代码的质量、减少 Bug 和容易维护。目标如下：

- 新人可以很快的适应环境，方便的融入到项目团队中
- 在一致的环境下，团队协作中有更高的效率，团队的成员可以减少犯错的机会
- 程序员可以方便的了解其他人的代码，弄清程序的状况，就和看自己的代码一样
- 防止接触 PHP 的新人自创一套风格并养成终生的习惯，一次次的犯同样的错误

A1.3 开发工具

PHP 的开发工具很多，常用的代码编辑工具有 Zend Studio、UltraEdit、EditPlus、PHPEdit、Eclipse、 Dreamweaver 和 vim 等，每种开发工具都有其各自的优势。在编写程序时，一款好的编辑工具会使程序员的编写过程更加轻松、有效和快捷，达到事半功倍的效果。对于一款好的代码编辑工具，除了具备最基本的代码编辑功能外，一个必备的功能就是语法的高亮显示、代码提示和代码补全。另外，一款好的代码编辑工具应具备格式排版功能，格式排版功能可以使程序代码的组织结构清晰易懂，并且易于程序员进行程序调试，排除程序中的错误异常。每程序员都可以根据自己的需求有选择性地使用开发工具，但开发工具种类之多，也给程序员在选择上代来困惑。本规范对开发工具的使用有如下建议：

- 一个项目团队尽量要使用统一的开发工具，并且要统一版本
- 项目团队中的每个成员要在所使用的工具中设置统一的字符编码，例如 UTF-8，以避免因为编码不统一，在项目整合时部分页面出现乱码
- 项目开发中常见的是使用缩进，缩进由制表符 tab 组成，目的是让代码组织结构和层次清晰易懂。需每个参与项目的开发人员在编辑器中进行强制设定，每个缩进的单位约定是一个 TAB(8 个空白字符宽度)， 以防在编写代码时遗忘而造成格式上的不规范。本缩进规范适用于 PHP、JavaScript 中的函数、类、逻辑结构、循环等。
- 如果有必要每行代码的字符数也不宜过多，具体控制每行字符数量也需要在工具中设定，80 字符以内是比较合适，但最多一行也不要超过 120 个字符
- 行结束标志在 Windows 中是“\r\n”,而 Unix/Linux 中则是“ \n”。要在开发工具中设定需要遵循 Unix/Linux 文本文件的约定，使用“\n”结束，不要使用 Windows 的回车换行组合

A2 PHP 的文件格式

A2.1 PHP 开始和结束标记

当脚本中带有 PHP 代码，可以使用<?php ?>、<? ?>或<% %>等标记来界定 PHP 代码，在 HTML 页面中嵌入纯变量时，还可以使用<?=$variablename ?>这样的形式。为了防止短标记<? ?>和 ASP 风格的<% %>与一些技术发生冲突，有时需要在 PHP 配置文件中将其关闭，因而导致这样的标记不总是可用。所以在编写 PHP 脚本时不允许使用短标记，所有脚本全部使用完整的、标准的、PHP 定界标签<?php ?>作为 PHP 开始和结束标记。

对于只包含有 PHP 代码的文件，结束标志（"?>"）是不允许存在的，PHP 自身不需要（"?>"）。 这样做, 可以防止它的末尾被意外地注入，从而导致像使用 Header()、setCookie()和 session_start()等设置头信息的函数时发生失败。

A2.2 注释规范

注释是对于那些容易忘记作用的代码添加简短的介绍性内容，可以在 PHP 脚本中使用以“/* *”开始和以“*/”作为结束的多行文档注释、还有普通多行注释“/* */”，以及单行

注释符号“//”。所有文档块建议和 phpDocumentor 格式兼容。PHPDocumentor 是一个用 PHP 写的工具，对于有规范注释的 php 程序，它能够快速生成具有相互参照,索引等功能的 API 文档。可以通过在客户端浏览器上操作生成文档，文档可以转换为 PDF,HTML,CHM 几种形式，非常的方便。PHPDocument 是从你的源代码的注释中生成文档，因此在给你的程序做注释的过程，也就是你编制文档的过程。从这一点上讲，PHPDocumentor 促使你要养成良好的编程习惯，尽量使用规范，清晰文字为你的程序做注释，同时多少也避免了事后编制文档和文档的更新不同步的一些问题。 在 PHPDocumentor 中，注释分为文档性注释和非文档性注释。 所谓文档性注释，是那些放在特定关键字前面的多行注释，特定关键字是指能够被 PHPDocumentor 分析的关键字，例如 class，var 等。 那些没有在关键字前面或者不规范的注释就称作非文档性注释，这些注释将不会被 PHPDocumentor 所分析，也不会出现在你产生的 api 文档中。如下所示：

```
894     /**
895      * Add config directory(s)
896      *
897      * @param string|array $config_dir directory(s) of config sources
898      * @param string key of the array element to assign the config dir to
899      * @return Smarty current Smarty instance for chaining
900      */
901     public function addConfigDir($config_dir, $key=null) {
902         ......
903         return $this;
904     }
```

在程序开发中难免留下一些临时代码和调试代码，以免日后遗忘，此类代码必须添加注释。所有临时性、调试性、试验性的代码，都可以添加统一的注释标记,例如“//debug”并后跟完整的注释信息，这样可以方便在程序发布和最终调试前批量检查程序中是否还存在有疑问的代码。

```
$flag = $_GET[“page”];          //debug 这里不能确定是否需要赋值
```

建议添加注释的地方：

- 在每个文件首部添加注释说明；
- 每个函数或每个方法上方添加注释说明；
- 对各变量的功能、范围、缺省条件等加上注释；
- 对使用的逻辑算法加上注释；

A2.3 空行和空白

一般来说，空白符（包括空格、Tab 制表符、换行）在 PHP 中无关紧要，会被 PHP 引擎忽略。可以将一个语句展开成任意行，或者将语句紧缩在一行。空格与空行的合理运用（通过排列分配、缩进等）可以增强程序代码的清晰性与可读性，如果不合理运用，便会适得其反。空行将逻辑相关的代码段分隔开，以提高可读性。任何情况下，PHP 程序中不能出现空白的带有 TAB 或空格的行，即：这类空白行应当不包含任何 TAB 或空格。同时，任何程序行尾也不能出现多余的 TAB 或空格。多数编辑器具有自动去除行尾空格的功能，如果习惯养成不好，可临时使用它，避免多余空格产生；

A2.3.1 空行的使用时机

每段较大的程序体，上、下应当加入一个空白行，下列情况应该总是使用一个空行，禁

止使用多行：

- 两个函数声明之间。
- 函数内的局部变量和函数的第一条语句之间。
- 块注释或单行注释之前。
- 一个函数内的两个逻辑代码段之间，用以提高可读性。

A2.3.2 空格的使用时机

空格的应用规则是可以通过代码的缩进提高可读性。

- 空格一般应用于关键字与左括号“(”之间，不过需要注意的是，函数名称与左括号之间不应该用空格分开。右括号“)”除后面是“)”或者“.”以外，其他一律用空格隔开它们；
- 一般在函数的参数列表中的逗号后面插入空格。
- 数学算式的操作数与运算符之间应该添加空格（特例是二进制运算与一元运算除外，字符连接运算符号两边不加空格）。
- for 语句中的表达式应该用逗号分开，后面添加空格。
- 强制类型转换语句中的强制类型的右括号与表达式之间应该用逗号隔开，添加空格。
- 除字符串中特意需要，一般情况下，在程序以及 HTML 中不出现两个连续的空格；
- 说明或显示部分中，内容如含有中文、数字、英文单词混杂，应当在数字或者英文单词的前后加入空格。

正确的书写格式

```php
<?php
    $num = 10;
    $int = 20;
    $sum = (($num + 1) * 6 / 2 + $int).'Abc';
    $page = isset($_GET["page"]) ? $_GET["page"] : 1;

    function myFun($arg1, $arg2, $arg3) {

        //statements more lines

    }
```

A2.4 字符串的使用

在程序开发中字符串的使用机率是最高的，字符串的声明可以使用双引号，也可以使用单引号。而在 PHP 中单引号和双引号具有不同的含义，最大的几项区别如下：

- 单引号中，任何变量($var)、特殊转义字符(如“\t \r \n”等)都不会被解析，因此 PHP 的解析速度更快，转义字符仅仅支持“\’”和“\\”这样对单引号和反斜杠本身的转义；
- 双引号中，变量($var)值会代入字符串中，特殊转义字符也会被解析成特定的单个字符，还有一些专门针对上述两项特性的特殊功能性转义，例如“\$”和“{$array[‘key’]}。这样虽然程序编写更加方便，但同时 PHP 的解析也很慢；

A2.4.2 使用单引号声明字符串

单引号不需要去解析变量，也不需要解析全部的转义字符，所以解析的速度快。因些在绝大多数可以使用单引号的场合，禁止使用双引号。依据上述分析，可以或必须使用单引号的情况如下（但不限于此）：

- 字符串为固定值，不包含“\t”等特殊转义字符

```
$html = '<input type="text" name="username" value="admin" />';
```

- 当字符串是不包含变量的文字，应当用单引号来括起来

```
$var  =  'value';
```

- 关联数组的下标数组中，如果下标不是整型，而是字符串类型，请务必用单引号将下标括起，正确的写法为$array['key']，而不是$array[key]，因为不正确的写法会使 PHP 解析器认为 key 是一个常量，进而先判断常量是否存在，不存在时才以“key”作为下标带入表达式中，同时出发错误事件，产生一条 Notice 级错误。

```
$array['key'];
```

- 数据库 SQL 语句中，所有数据必须加单引号，无论数值还是字串，以避免可能的注入漏洞和 SQL 错误。

```
UPDATE users SET name='admin', age='22', height='178.5' where id='1'
```

A2.4.2 使用双引号声明字符串

- 字符串中的变量需要替换时：

```
$var= "hello {$world}";
```

- 当文字字符串包含单引号，那么字符串就用双引号括起来，主要针对于 sql 语句

```
$sql = "SELECT * FROM `table` WHERE id='{$id }'";
```

- 在正则表达式(用于 preg_系列函数和 ereg 系列函数)中，建议全部使用双引号，这是为了人工分析和编写的方便，并保持正则表达式的统一，减少不必要的分析混淆。

注意：所有数据在插入数据库之前，均需要进行 addslashes()处理，以免特殊字符未经转义在插入数据库的时候出现错误。

A2.5 命名原则

就一般约定而言，文件、目录、类、函数、变量和常量的名字，应该让代码阅读者能够

容易的知道这些代码的作用。命名的原则就是以最少的字母达到最容易理解的意义。命名是程序规划的核心，只有了解系统的程序员才能为系统取出最合适的名字。如果所有的命名都与其自然相适合，则关系清晰，含义就可以推导得出。形式越简单、越有规则，就越容易让人感知和理解，应该避免使用不标准的命名。

A2.5.1 文件名

所有包含 PHP 代码的程序文件或半程序文件，应以小写.php 作为扩展名，而不要使用.phtml、.php3、.inc、.class 等作为扩展名。文件名称一定要有意义，应具有描述性，让人看到文件名就可以大概猜到文件中的内容。不允许使用拼音、不直观的单词简写和缩写。文件名包括数字字母和下划线字符，允许但不鼓励使用数字，不允许使用其他字符。如果文件名包括多个单词，单词全部小写，使用下划线进行连接。

- 能够被 URL 直接调用的程序，直接使用程序名+.php 的方式命名

login.php、index.php	普通程序

- 类库程序只能被其他程序引用，而不能独立运行。其中不能包含任何流程性的、不属于任何类的程序代码，类文件的后缀统一为：**.class.php**

goods.class.php	文件中声明一个 Product 的类

- 函数库也只能被其他程序引用，不能独立运行。其中不能包含任何流程性的、不属于任何函数程序代码。函数文件的后缀统一为： **.func.php**

common.func.php	文件中声明一些通用的函数

- 只能被其他程序引用，而不能独立运行。其中不能包含任何函数或类代码的程序代码。文件后缀统一为： **.inc.php**

config.inc.php	文件中声明一些项目的配置内容

A2.5.2 目录命名

目录命名也一定要有描述性的意义，在可能的情况下，多以复数形式出现，如./templates、./images 等。由于目录数量较少，因此目录命名大多是一些习惯和约定俗成，开发人员如需新建目录，应与项目组成员进行磋商，达成一致后方可实施。

另外，要在所有不包含普通程序(即能够被 URL 直接调用的程序)的目录中放置一个 1 字节的 index.htm 文件，内容为一个空格。几乎除根目录以外，所有目录都属于这一类型，因此开发者需要在这些目录全部放入空 index.htm 文件，以避免当 http 服务器的 Directory 容器中 Options Indexes 打开时，服务器文件被索引和列表。

A2.5.3 类名

一个文件中声明一个类，文件名中必须包含类名字符串，这些不仅容易查找，也有利于实现在程序中自动加载类。

- 类名应有描述性，杜绝一切拼音、或拼音英文混杂的命名方式
- 类名包括字母字符，不允许使用数字和其他字符
- 如果类名包括多个单词,应使用驼峰式命名方式,每个单词的第一个字母必须大写，不允许连续大写。

例如：**AaaBbbCcc** （如果类名由 aaa，bbb，ccc 三个单词组成的）

A2.5.4 函数和方法名

- 函数名应具有描述性，杜绝一切拼音、或拼音英文混杂的命名方式
- 函数名包括字母字符，不允许使用数字和其他字符。
- 函数名首字母小写，当包含多个单词时，后面的每个单词的首字母大写.

例如：**aaaBbbCcc** （如果函数名由 aaa，bbb，ccc 三个单词组成的）

- 函数名应带有‘get’，‘set’等动作性描述。

```
function getUser(){
        //函数内容
}
```

- 可以声明像函数名前带有下划线的形式，表示该函数为该类的私有方法，外部不允许进行访问。

```
function _func(){
        //函数内容
}
```

A2.5.5 变量名

- 变量也应具有描述性，杜绝一切拼音、或拼音英文混杂的命名方式
- 变量包含数字、字母和下划线字符，不允许使用其他字符，变量命名最好使用项目中有据可查的英文缩写方式， 尽可以要使用一目了然容易理解的形式；
- 变量以字母开头，如果变量包含多个单词，首字母小写，当包含多个单词时，后面的每个单词的首字母大写.

例如：$**aaaBbbCcc** （如果变量名由 aaa，bbb，ccc 三个单词组成的）

- 可以合理的对过长的命名进行缩写，例如$bio($biography)，$tpp($threadsPerPage)，前提是英文中有这样既有的缩写形式，或字母符合英文缩写规范；
- 必须清楚所使用英文单词的词性，在权限相关的范围内，大多使用$allow***或$is***的形式，前者后面接动词，后者后面接形容词。

例如：$**allowInsert**，$**isInt**等

- 变量除了在循环体(for，foreach，while)中，其他位置允许但不鼓励使用没有描述意义的字母作为变量名。例如：$i,$j。

A2.5.6 常量名

- 常量名应具有描述性，杜绝一切拼音、或拼音英文混杂的命名方式
- 常量名包含字母字符和下划线，不允许使用数字和其他字符。
- 常量名所有字母必须大写，少数特别必要的情况下，可使用划线来分隔单词。
  例如： define('AAA_BBB_CCC', 'true'); (如果常量名由 aaa, bbb, ccc 三个单词组成的)
- PHP 的内建值 TRUE、FALSE 和 NULL 必须全部采用大写字母书写。

A2.6 语言结构

A2.6.1 if/else/else if

- if 结构中，前花括号必须和条件语句在同一行，后花括号单独在最后一行，其中内容使用缩进。else 和 else if 与前后两个大括号同行，左右各一个空格。另外，即便 if 后只有一行语句，仍然需要加入大括号，以保证结构清晰;
- 首括号与关键词同行，尾括号与关键字同列;
- if 中条件语句的圆括号前后必须有一个空格;
- 括号内的条件语句中操作符必须用空格分开;
- 允许但强烈不鼓励 elseif 写法，鼓励使用 else if 的写法;

```
if ($one == '1' ) {

} else if ($one == '2') {

}
```

- 在条件语句中存在多个运算符的时候，使用括号强制说明优先级，避免开发人员因为运算符优先级概念混乱造成的逻辑错误。

```
$one = $two == 1 || $two == 2 && $three > 3;        //错误的
$one = (($two == 1) || ($two == 2 && $three > 3)); //正确的
```

- 在判断条件中明确判断内容的变量类型，使用正确的类型，不允许在判断结果为 true 的时候使用 1，反之亦然。

A2.6.2 switch

- switch 在条件语句的圆括号前后必须都有一个空格。
- switch 中的代码使用缩进，case 内的代码再进行缩进
- switch 结构中， break 的位置视程序逻辑，与 case 同在一行，或新起一行均可，但同一 switch 体中，break 的位置格式应当保持一致。
- switch 语句应当有 default 语句作为结束。
- 允许但不鼓励使用两个及两个以上的 case 条件对应一个 break 语句，如果有这样的情况出现，必须要注明当时的情况。

```
switch ($var) {
    case 1:     echo 'var is 1'; break;
    case 2:     echo 'var is 2'; break;
    default:    echo 'var is neither 1 or 2'; break;
}


switch ($str) {
    case 'abc':
        $result = 'abc';
        break;
    default:
        $result = 'unknown';
        break;
}
```

A2.6.3 数组声明

- 数字索引数组索引不能为负数，如果不存在定义索引，建议索引以 0 开始。
- 数组中逗号后面间隔一个空格，提高可读性。

```
array('one_value', 'two_value');
```

- 如果数组元素过多需要换行显示时，在每个连续行要用缩进将开头对齐。

```
array('one_value',
      'two_value',
      'three_value'
);
```

- 如果使用 key/value 的形式进行关联数组声明的话，鼓励把数组分成多行，提高可读性。

```
array(
       'one_key'=>'one_value',
       'two_key'=>'two_value',
       'three_key'=>'three_value'
);
```

A2.6.4 类的声明

- 开始的左大括号与类的定义为同一行，中间加一个空格，不要另起一行；
- 每个类必须有一个符合 PHPDocumentor 标准的文档块；
- 类中的代码必须使用缩进；
- 每个 PHP 文件中只有一个类,在文件名中包含类名 ，例如“类名.class.php”；
- 不建议将其他代码放到类文件里。

```
/**
* Documentation Block Here
*/
class SampleClass {
    //类的所有内容
}
```

A2.6.5 类中成员属性和变量的声明

- 类中成员属性的声明必须放到类的顶部，也就是在方法上面声明，而且需要使用合适的权限限制外部访问级别；
- 任何变量在进行累加、直接显示或存储前必需进行初使化。因为 PHP 中的变量不并不像强类型语言那样需要事先声明，而 PHP 解释器会在第一次使用时自动创建他们，同样类型也不需要指定，解释器会根据上下文环境自动确定。从开发人员的角度来看，这无疑是一种极其方便的处理方法。一个变量被创建了，就可以在程序中的任何地方使用。这导致的结果就是开发人员经常不注意初始化变量。因此，为

了提高程序的安全性，我们不能相信任何没有明确定义的变量。所有的变量在定义使用前要初使化以防止恶意构造提交的变量覆盖程序中使用的变量。

```
$number = 0;                    //数值型初始化
$string = ‘’;                   //字符串初始化
$array = array();               //数组初始化
```

- 判断一个无法确定（不知道是否已被赋值）的变量时，可用 empty()或 isset()，而不要直接使用 if($switch)的形式，除非你确切的知道此变量一定已经被初始化并赋值。
- 判断一个变量是否为数组，请使用 is_array()，这种判断尤其适用于对数组进行遍历的操作，例如 foreach()，因为如果不事先判断，foreach()会对非数组类型的变量报错；
- 判断一个数组元素是否存在，可使用 isset($array[‘key’])，也可使用 empty();

A2.6.6 函数的定义与使用

- 声明函数时参数的名字和变量的命名规范一致；
- 函数定义中的左小括号，与函数名紧挨，中间无需空格；
- 开始的左大括号与函数定义为同一行，中间加一个空格，不要另起一行；
- 如果使用具有默认值的参数，应该位于参数列表的后面；
- 函数不管在调用还是在声明的时候，参数与参数之间都要加入一个空格；
- 必须仔细检查并切实杜绝函数起始缩进位置与结束缩进位置不同的现象。
- 建议不要使用全局函数；

```
function authcode($string, $operation, $key = '') {
    //函数体
}
```

- 在使用系统函数时，除非必要，否则不要使用 PHP 扩展模块中的函数。如果使用，也应当加入必要的判断，这样当服务器在环境不支持此函数的时候，也可以进行必要的处理。还应该在文档和程序中的功能说明中，加上一些兼容性说明。

A2.7 其它规范细节

A2.7.1 代码重用和包含调用

- 在任何时候都不要在同一个程序中出现两段或更多的相似代码或相同代码，即便在不同程序中，也应尽力避免。只要超过 3 行实现相同功能的程序代码，就切勿在同一程序中多次出现，这是无法容忍和回避的问题；
- 代码的有效重用不仅可以减少效率的损失与资源的浪费，将更多的精力用在新技术的应用和新功能的创新开发上面，而且有利于代码的修改和更新。因为只要修改或更新重用的代码，就会改变所有使用这些重用代码的形为。
- 重用代码的调用可以使用 require 和 include 两个系统指令，require 语句语句通常放在 PHP 脚本程序的最前面。PHP 程序在执行前，就会先读入 require 语句所引入的文件，使它变成 PHP 脚本文件的一部分。include 语句的使用方法和 require 语句一样，而这个语句一般是放在流程控制的处理区段中。PHP 脚本文件在读到 include 语句时，才将它包含的文件读进来。这种方式，可以把程序执行时的流程简单化。

```
require 'config.php';          //使用 require 语句包含并执行 config.php 文件

if ($condition) {              //在流程控制中使用 include 语句
    include 'file.txt';        //使用 include 语句包含并执行 file.txt 文件
} else {                       //条件不成立则包含下面的文件
    include 'other.php';       //使用 include 语句包含并执行 other.php 文件
}

require 'somefile.txt';        //使用 require 语句包含并执行 somefile.txt 文件
```

A2.7.2 错误报告级别

- 在软件开发和调试阶段，请在全局文件中使用 error_reporting(E_ALL)，作为默认的错误报告级别，此级别最为严格，能够报告程序中所有的错误、警告和提示信息，以帮助开发者检查和核对代码，避免大多数安全性问题和逻辑错误、拼写错误。
- 在软件发布时，请使用 error_reporting(E_ERROR | E_WARNING | E_PARSE)，作为默认的错误报告级别，以利于用户使用并将无谓错误提示信息降至最低。

A3MySQL 设计规范

A3.1 数据表的设计

A3.1.1 数据库表名

- 表名应具有描述性，杜绝一切拼音、或拼音英文混杂的命名方式；
- 表名允许使用字母，数字和下划线，不允许使用其他字符。表名使用单词开头，不允许使用数字和下划线开头；
- 表名一律有统一前缀，前缀与表名之间以下划线连接。使用前缀可以让同一个项目在一个库中安装多个；
- 表名单词一律小写，单词之间使用下划线连接；
- 表名长度不能超过 64 个字符；
- 所有数据表名称，只要其名称是可数名词，则建议以复数方式命名，例如：xs_users(用户表)、xs_articles(文章表)；
- 表名要回避 MySQL 的保留字（保留字见 附录 B）；

A3.1.2 数据表字段名

- 字段名应具有描述性，杜绝一切拼音、或拼音英文混杂的命名方式
- 字段名允许使用字母，数字和下划线，不允许使用其他字符。字段名鼓励使用与所在表的内容相关单词开头，允许但不鼓励使用数字和其他字符开头。
- 字段名一律小写，单词之间使用下划线连接。
- 字段名长度不能超过 64 个字符。
- 字段类型和长度在不同数据表中必须保证一致性，不允许出现同一字段在一个表中为整型但另外一个表中为字符型的情况出现。
- 当几个表间的字段有关连时，要注意表与表之间关联字段命名的统一，如 xs_orders 表中的 uid 与 xs_carts 表中的 uid, 都保存有 xs_users 表中的 id。
- 存储多项内容的字段，或代表数量的字段，也应当以复数方式命名，例如： views(查看次数)；
- 每个表建议都要有一个代表 id 自增量的字段，可以使用全称的形式，也可只将其命名为 id。

A3.1.3 字段索引名称

- 索引名称允许使用字母，数字和下划线，不允许使用其他字符。
- 对任何外键采用非成组索引。
- 不要索引 text/blob 类型的字段，不索引字符过多的字段。
- 根据业务需求建立组合索引。
- 索引长度不能超过 64 个字符。

A3.1.4 字段结构

进行表结构设计时，应当做到恰到好处，反复推敲，从而实现最优的数据存储体系。

- NULL 值的字段，数据库在进行比较操作时，会先判断其是否为 NULL，非 NULL 时才进行值的比对。因此基于效率的考虑，所有字段均不能为空，即全部使用 NOT NULL 的属性修饰字段；
- 如果不会使用存储非负数的字段时（如各项 id、访问数等），必须设置为 UNSIGNED 类型，能获得范围大一倍数值存储空间；
- 任何类型的数据表，字段空间应当本着足够用，不浪费的原则
- 个别字段类型在数据结构设计的时候需要注意：enum 枚举类型由 tinyint 类型代替；
- 包含任何 varchar、text 等变长字段的数据表，即为变长表，反之则为定长表。在设计表结构时如果能够使用定长数据类型尽量用定长的，因为定长表的查询、检索、更新速度都很快。必要时可以把部分关键的、承担频繁访问的表拆分，例如定长数据一个表，非定长数据一个表。
- 更小的字段类型永远比更大的字段类型处理要快得多。对于字符串，其处理时间与字符串长度直接相关。一般情况下，较小的表处理更快。对于定长表，应该选择最小的类型，只要能存储所需范围的值即可。例如，如果 mediumint 够用，就不要选择 bigint。对于可变长类型，也仍然能够节省空间。一个 TEXT 类型的值用 2 字节记录值的长度，而一个 LONGTEXT 则用 4 字节记录其值的长度。如果存储的值长度永远不会超过 64KB，使用 TEXT 将使每个值节省 2 字节。
- 数值运算一般比字符串运算更快。例如比较运算，可在单一运算中对数进行比较。而串运算涉及几个逐字节的比较，如果串更长的话，这种比较还要多。如果字符串列的数值数目有限，应该利用普通整型来获得数值运算的优越性。

A3.2 索引设计原则

MySQL 索引，常用的有 PRIMARY KEY、INDEX、UNIQUE 几种。通常，在单表数据值不重复的情况下，PRIMARY KEY 和 UNIQUE 索引比 INDEX 更快，要酌情使用。

索引能加快查询速度，而索引优化和查询优化是相辅相成的，既可以依据查询对索引进行优化，也可以依据现有索引对查询进行优化，这取决于修改查询或索引，哪个对现有产品架构和效率的影响最小。根据产品的实际运行和被访问情况，找出哪些 SQL 语句是最常被执行的。最常被执行和最常出现在程序中是完全不同的概念。最常被执行的 SQL 语句，又可被划分为对大表(数据条目多的)和对小表(数据条目少的)的操作。无论大表或小表，有可分为读多、写多或读写都多的操作。

事实上，索引是将条件查询、排序的读操作资源消耗，分布到了写操作中，索引越多，耗费磁盘空间越大，写操作越慢。因此，索引决不能盲目添加。对字段索引与否，最根本的出发点，依次仍然是 SQL 语句执行的概率、表的大小和写操作的频繁程度。

A3.3 SQL 语句设计

- 所有 SQL 语句中，除了表名、字段名称以外，全部语句和函数均需大写，应当杜绝小写方式或大小写混杂的写法；

例如 **select * from xs_users;**是不符合规范的写法

- 字段列表不要使用“*”，需要查询的字段就在字段列表中给出,同样插入数据时也要给出字段列表。这样不仅可以提高查询效率，也可以得到自己想要的字段列表顺序；

例如：**SELECT name,age,sex FROM users;**

- 在使用字段值时，不管什么类型的数据，都要使用单引起来

例如：**INSERT INTO users(name, age, sex) VALUES(‘admin’, ‘10’, ‘男’);**

- 通常情况下，在对多表进行操作时，要根据不同表名称，对每个表指定一个 1~2 个字母的缩写，以利于语句简洁和可读性。

```
$query = $db->query("SELECT s.*, u.*
    FROM {$tablepre}sessions s, {$tablepre}users u
    WHERE u.uid=s.uid AND s.sid='$sid');
```

A4 模板设计

- 所有模板文件建议使用小写.htm 作为扩展名。
- HTML 代码标记一律采用小写字母形式，杜绝任何使用大写字母的方式
- 所有 HTML 标记参数赋值需使用双引号包含，例如，应当使用<input type=”text” name=”username” value=”admin” />, 而绝对不能使用 <input type=text name=username value=admin />。
- 在任何情况下，产品中的模板文件必须采用手写 HTML 代码的方式，而绝对不能使用 DreamWeaver、FrontPage 等自动网页制作工具进行撰写或修改。
- 建议自定义模板文件的位置，还有模板文件被编译后自动生成目标程序的位置，以及缓存文件位置，可以根据项目的需求去设置这些目录的位置，但都要使用绝路径去设置。

```
$tpl=new Smarty;
define("PATH", "/usr/local/www/");        //先定义一个绝对路径
$tpl->setTemplateDir(PATH.'templates');  //模板文件位置
$tpl->setCompileDir(PATH.'templates_c'); //编译后自动生成目标程序位置
$tpl->setConfigDir(PATH.'configs');       //模板配置文件位置
$tpl->setCacheDir(PATH.'caches');          //缓存文件位置
```

- Smarty 模板中默认的定界符号“ {}”，会和模板中使用的 css 和 javaScript 中的大括号发生冲冲突， 所以一定要换掉，建议改写成“<{}>”符号作为定界符号。

```
$tpl=new Smarty;
```

```
$tpl->left_delimiter='<{';        //左定界符号
$tpl->right_delimiter='}>';       //右定界符号
```

- 模板中的注释建议使用 Smarty 提供的注释 “<{*注释的内容*}>”
- 在 Smarty 的*.htm 模板文件中，由于具备逻辑结构，故不考虑任何 HTML 本身的缩进，所有缩进均意为着逻辑上的缩进结构。缩进采用 TAB 方式，不使用空格作为缩进符号，仅需适当断行即可。

```
<{section loop=$data name=”item”}>
    <table cellspacing=”0” cellspadding=”0” border=”1”>
          <tr><td>$data[item].id</td></tr>
    </table>
<{/section}
```