PHP Document 代码注释规范

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。老的版本是 phpdoc。

1. 什么是 phpDocumentor ?

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。老的版本是 phpdoc，从 1.3.0 开始，更名为 phpDocumentor，新的版本加上了对 php5 语法的支持，同时，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。

PHPDocumentor 工作时，会扫描指定目录下面的 php 源代码，扫描其中的关键字，截取需要分析的注释，然后分析注释中的专用的 tag，生成 xml 文件，接着根据已经分析完的类和模块的信息，建立相应的索引，生成 xml 文件，对于生成的 xml 文件，使用定制的模板输出为指定格式的文件。

2. 安装 phpDocumentor

和其他 pear 下的模块一样，phpDocumentor 的安装也分为自动安装和手动安装两种方式，两种方式都非常方便：

a. 通过 pear 自动安装

在命令行下输入

pear install PhpDocumentor

b. 手动安装

在 http://manual.phpdoc.org/下载最新版本的 PhpDocumentor（现在是 1.4.0），把内容解压即可。

3. 怎样使用 PhpDocumentor 生成文档

命令行方式：

在 phpDocumentor 所在目录下，输入

Php –h

会得到一个详细的参数表，其中几个重要的参数如下：

-f 要进行分析的文件名，多个文件用逗号隔开

-d 要分析的目录，多个目录用逗号分割

-t 生成的文档的存放路径

-o 输出的文档格式，结构为输出格式：转换器名：模板目录。

例如：phpdoc -o HTML:frames:earthli -f test.php -t docs

Web 界面生成

在新的 phpdoc 中，除了在命令行下生成文档外，还可以在客户端浏览器上操作生成文档，具体方法是先把 PhpDocumentor 的内容放在 apache 目录下使得通过浏览器可以访问到，访问后显示如下的界面：

点击 files 按钮，选择要处理的 php 文件或文件夹，还可以通过该指定该界面下的 Files to ignore 来忽略对某些文件的处理。
然后点击 output 按钮来选择生成文档的存放路径和格式.

最后点击 create，phpdocumentor 就会自动开始生成文档了，最下方会显示生成的进度及状态，如果成功，会显示

Total Documentation Time: 1 seconds

done

Operation Completed!!

然后，我们就可以通过查看生成的文档了，如果是 pdf 格式的，名字默认为 documentation.pdf。

4．给 php 代码添加规范的注释

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，PHPdoc 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，同时多多少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 phpdoc 分析的关键字，例如 class，var 等，具体的可参加附录 1.

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 phpdoc 所分析，也不会出现在你产生的 api 文当中。

3.2 如何书写文档性注释:

所有的文档性注释都是由/**开始的一个多行注释，在 phpDocumentor 里称为 DocBlock, DocBlock 是指软件开发人员编写的关于某个关键字的帮助信息，使得其他人能够通过它知道这个关键字的具体用途，如何使用。 PhpDocumentor 规定一个 DocBlock 包含如下信息：

1. 功能简述区

2. 详细说明区

3. 标记 tag

文档性注释的第一行是功能描述区，正文一般是简明扼要地说明这个类，方法或者函数的功能，功能简述的正文在生成的文档中将显示在索引区。功能描述区的内容可以通过一个空行或者 . 来结束

在功能描述区后是一个空行，接着是详细说明区,. 这部分主要是详细说明你的 API 的功能，用途，如果可能，也可以有用法举例等等。在这部分，你应该着重阐明你的 API 函数或者方法的通常的用途，用法，并且指明是否是跨平台的（如果涉及到），对于和平台相关的信息，你要和那些通用的信息区别对待，通常的做法是另起一行，然后写出在某个特定平台上的注意事项或者是特别的信息，这些信息应该足够，以便你的读者能够编写相应的测试信息，比如边界条件，参数范围，断点等等。

之后同样是一个空白行，然后是文档的标记 tag，指明一些技术上的信息，主要是最主要的是调用参数类型，返回值极其类型，继承关系，相关方法/函数等等。

关于文档标记，详细的请参考第四节:文档标记。

文档注释中还可以使用例如<b> <code>这样的标签，详细介绍请参考附录二。

下面是一个文档注释的例子

以下为引用的内容：

```
/**
* 函数 add,实现两个数的加法
*
* 一个简单的加法计算，函数接受两个数 a、b，返回他们的和 c
*
* @param int 加数
* @param int 被加数
* @return integer
*/
function Add($a, $b) {
return $a+$b;
}
```

生成文档如下：

Add

integer Add(int $a, int $b)

[line 45]

函数 add,实现两个数的加法

Constants 一个简单的加法计算，函数接受两个数 a、b，返回他们的和 c

Parameters

? int $a - 加数

? int $b - 被加数

5. 文档标记：

文档标记的使用范围是指该标记可以用来修饰的关键字，或其他文档标记。

所有的文档标记都是在每一行的 * 后面以@开头。如果在一段话的中间出来@的标记，这个标记将会被当做普通内容而被忽略掉。

@access

使用范围：class,function,var,define,module

该标记用于指明关键字的存取权限：private、public 或 proteced

@author

指明作者

@copyright

使用范围：class，function，var，define，module，use

指明版权信息

@deprecated

使用范围：class，function，var，define，module，constent，global，include

指明不用或者废弃的关键字

@example

该标记用于解析一段文件内容，并将他们高亮显示。Phpdoc 会试图从该标记给的文件路径中读取文件内容

@const

使用范围：define

用来指明 php 中 define 的常量

@final

使用范围：class,function,var

指明关键字是一个最终的类、方法、属性，禁止派生、修改。

@filesource

和 example 类似，只不过该标记将直接读取当前解析的 php 文件的内容并显示。

@global

指明在此函数中引用的全局变量

@ingore

用于在文档中忽略指定的关键字

@license

相当于 html 标签中的<a>,首先是 URL，接着是要显示的内容

例如百度

可以写作 @license http://www.baidu.com 百度

@link

类似于 license

但还可以通过 link 指到文档中的任何一个关键字

@name

为关键字指定一个别名。

@package

使用范围：页面级别的-> define，function，include

类级别的->class，var，methods

用于逻辑上将一个或几个关键字分到一组。

@abstrcut

说明当前类是一个抽象类

@param

指明一个函数的参数

@return

指明一个方法或函数的返回指

@static

指明关建字是静态的。

@var

指明变量类型

@version

指明版本信息

@todo

指明应该改进或没有实现的地方

@throws

指明此函数可能抛出的错误异常,极其发生的情况

上面提到过，普通的文档标记标记必须在每行的开头以@标记，除此之外，还有一种标记叫做 inline tag,用{@}表示，具体包括以下几种：

{@link}

用法同@link

{@source}
显示一段函数或方法的内容
6. 一些注释规范
a.注释必须是
/**
* XXXXXXX
*/
的形式
b.对于引用了全局变量的函数，必须使用 glboal 标记。
c.对于变量，必须用 var 标记其类型（int,string,bool...）
d.函数必须通过 param 和 return 标记指明其参数和返回值
e.对于出现两次或两次以上的关键字，要通过 ingore 忽略掉多余的，只保留一个即可
f.调用了其他函数或类的地方，要使用 link 或其他标记链接到相应的部分，便于文档的阅读。
g.必要的地方使用非文档性注释，提高代码易读性。
h.描述性内容尽量简明扼要，尽可能使用短语而非句子。
i.全局变量，静态变量和常量必须用相应标记说明
7. 总结
phpDocumentor 是一个非常强大的文档自动生成工具，利用它可以帮助我们编写规范的注释，生成易于理解，结构清晰的文档，对我们的代码升级，维护,移交等都有非常大的帮助。
关于 phpDocumentor 更为详细的说明，可以到它的官方网站
http://manual.phpdoc.org/查阅
8. 附录
附录 1:
能够被 phpdoc 识别的关键字：
Include
Require
include_once
require_once
define
function
global
class
附录 2
文档中可以使用的标签
<b>
<code>

<kdb>

<pre>

<samp>
<var>

附录三：

一段含有规范注释的 php 代码 ：

以下为引用的内容：

```
<?php
/**
* Sample File 2, phpDocumentor Quickstart
*
* This file demonstrates the rich information that can be included in
* in-code documentation through DocBlocks and tags.
* @author Greg Beaver <cellog@php.net>
* @version 1.0
* @package sample
*/
// sample file #1
/**
* Dummy include value, to demonstrate the parsing power of phpDocumentor
*/
include_once 'sample3.php';
/**
* Special global variable declaration DocBlock
* @global integer $GLOBALS['_myvar']
* @name $_myvar
*/
$GLOBALS['_myvar'] = 6;
/**
* Constants
*/
/**
* first constant
*/
define('testing', 6);
/**
* second constant
*/
define('anotherconstant', strlen('hello'));
/**
* A sample function docblock
* @global string document the fact that this function uses $_myvar
* @staticvar integer $staticvar this is actually what is returned
* @param string $param1 name to declare
```

```
* @param string $param2 value of the name
* @return integer
*/
function firstFunc($param1, $param2 = 'optional') {
static $staticvar = 7;
global $_myvar;
return $staticvar;
}
/**
* The first example class, this is in the same package as the
* procedural stuff in the start of the file
* @package sample
* @subpackage classes
*/
class myclass {
/**
* A sample private variable, this can be hidden with the --parseprivate
* option
* @accessprivate
* @var integer|string
*/
var $firstvar = 6;
/**
* @link http://www.example.com Example link
* @see myclass()
* @uses testing, anotherconstant
* @var array
*/
var $secondvar =
array(
    'stuff' =>
                array(
                                6,
                                17,
                                'armadillo'
                        ),
    testing => anotherconstant
);
/**
* Constructor sets up {@link $firstvar}
*/
function myclass() {
$this->firstvar = 7;
```

```
}
/**
* Return a thingie based on $paramie
* @param boolean $paramie
* @return integer|babyclass
*/
function parentfunc($paramie) {
if ($paramie) {
return 6;
} else {
return new babyclass;
}
}
}
/**
* @package sample1
*/
class babyclass extends myclass {
/**
* The answer to Life, the Universe and Everything
* @var integer
*/
var $secondvar = 42;
/**
* Configuration values
* @var array
*/
var $thirdvar;
/**
* Calls parent constructor, then increments {@link $firstvar}
*/
function babyclass() {
parent::myclass();
$this->firstvar++;
}
/**
* This always returns a myclass
* @param ignored $paramie
* @return myclass
*/
function parentfunc($paramie) {
return new myclass;
}
```

```
}
?>
```