# 日志接口规范

本文制定了日志类库的通用接口规范。

本规范的主要目的，是为了让日志类库以简单通用的方式，通过接收一个 Psr\Log\LoggerInterface 对象，来记录日志信息。 框架以及CMS内容管理系统如有需要，**可以**对此接口进行扩展，但需遵循本规范， 这才能保证在使用第三方的类库文件时，日志接口仍能正常对接。

关键词 “必须”("MUST")、“一定不可/一定不能”("MUST NOT")、“需要”("REQUIRED")、 “将会”("SHALL")、“不会”("SHALL NOT")、“应该”("SHOULD")、“不该”("SHOULD NOT")、 “推荐”("RECOMMENDED")、“可以”("MAY")和”可选“("OPTIONAL")的详细描述可参见 [RFC 2119](http://tools.ietf.org/html/rfc2119) 。

本文中的 实现者 指的是实现了 LoggerInterface 接口的类库或者框架，反过来讲，他们就是 LoggerInterface 的 使用者。

## 1. 规范说明

### 1.1 基本规范

* LoggerInterface 接口对外定义了八个方法，分别用来记录 [RFC 5424](http://tools.ietf.org/html/rfc5424) 中定义的八个等级的日志：debug、 info、 notice、 warning、 error、 critical、 alert 以及 emergency 。
* 第九个方法 —— log，其第一个参数为记录的等级。可使用一个预先定义的等级常量作为参数来调用此方法，**必须**与直接调用以上八个方法具有相同的效果。如果传入的等级常量参数没有预先定义，则**必须**抛出 Psr\Log\InvalidArgumentException 类型的异常。在不确定的情况下，使用者**不该**使用未支持的等级常量来调用此方法。

### 1.2 记录信息

* 以上每个方法都接受一个字符串类型或者是有 \_\_toString() 方法的对象作为记录信息参数，这样，实现者就能把它当成字符串来处理，否则实现者**必须**自己把它转换成字符串。
* 记录信息参数**可以**携带占位符，实现者**可以**根据上下文将其它替换成相应的值。

其中占位符**必须**与上下文数组中的键名保持一致。

占位符的名称**必须**由一个左花括号 { 以及一个右括号 } 包含。但花括号与名称之间**一定不能**有空格符。

占位符的名称**应该**只由 A-Z、 a-z,0-9、下划线 \_、以及英文的句号 .组成，其它字符作为将来占位符规范的保留。

实现者**可以**通过对占位符采用不同的转义和转换策略，来生成最终的日志。 而使用者在不知道上下文的前提下，**不该**提前转义占位符。

以下是一个占位符使用的例子：

```php /\*\* \* 用上下文信息替换记录信息中的占位符 \*/ function interpolate($message, array $context = array()) { // 构建一个花括号包含的键名的替换数组 context as $key => $val) { $replace['{' . $key . '}'] = $val; }

// 替换记录信息中的占位符，最后返回修改后的记录信息。  
 return strtr($message, $replace);

}

// 含有带花括号占位符的记录信息。 $message = "User {username} created";

// 带有替换信息的上下文数组，键名为占位符名称，键值为替换值。 $context = array('username' => 'bolivar');

// 输出 "Username bolivar created" echo interpolate($message, $context); ```

### 1.3 上下文

* 每个记录函数都接受一个上下文数组参数，用来装载字符串类型无法表示的信息。它**可以**装载任何信息，所以实现者**必须**确保能正确处理其装载的信息，对于其装载的数据，**一定不能** 抛出异常，或产生PHP出错、警告或提醒信息（error、warning、notice）。
* 如需通过上下文参数传入了一个 Exception 对象， **必须**以 'exception' 作为键名。 记录异常信息是很普遍的，所以如果它能够在记录类库的底层实现，就能够让实现者从异常信息中抽丝剥茧。 当然，实现者在使用它时，**必须**确保键名为 'exception' 的键值是否真的是一个 Exception，毕竟它**可以**装载任何信息。

### 1.4 助手类和接口

* Psr\Log\AbstractLogger 类使得只需继承它和实现其中的 log 方法，就能够很轻易地实现 LoggerInterface 接口，而另外八个方法就能够把记录信息和上下文信息传给它。
* 同样地，使用 Psr\Log\LoggerTrait 也只需实现其中的 log 方法。不过，需要特别注意的是，在traits可复用代码块还不能实现接口前，还需要 implement LoggerInterface。
* 在没有可用的日志记录器时， Psr\Log\NullLogger 接口**可以**为使用者提供一个备用的日志“黑洞”。不过，当上下文的构建非常消耗资源时，带条件检查的日志记录或许是更好的办法。
* Psr\Log\LoggerAwareInterface 接口仅包括一个 setLogger(LoggerInterface $logger) 方法，框架可以使用它实现自动连接任意的日志记录实例。
* Psr\Log\LoggerAwareTrait trait可复用代码块可以在任何的类里面使用，只需通过它提供的 $this->logger，就可以轻松地实现等同的接口。
* Psr\Log\LogLevel 类装载了八个记录等级常量。

## 2. 包

上述的接口、类和相关的异常类，以及一系列的实现检测文件，都包含在 [psr/log](https://packagist.org/packages/psr/log) 文件包中。

## 3. Psr\Log\LoggerInterface

<?php  
  
namespace Psr\Log;  
  
/\*\*  
 \* 日志记录实例  
 \*  
 \* 日志信息变量 —— message， \*\*必须\*\*是一个字符串或是实现了 \_\_toString() 方法的对象。  
 \*  
 \* 日志信息变量中\*\*可以\*\*包含格式如 “{foo}” (代表foo) 的占位符，  
 \* 它将会由上下文数组中键名为 "foo" 的键值替代。  
 \*  
 \* 上下文数组可以携带任意的数据，唯一的限制是，当它携带的是一个 exception 对象时，它的键名 必须 是 "exception"。  
 \*  
 \* 详情可参阅： https://github.com/PizzaLiu/PHP-FIG/blob/master/PSR-3-logger-interface-cn.md  
 \*/  
interface LoggerInterface  
{  
 /\*\*  
 \* 系统不可用  
 \*  
 \* @param string $message  
 \* @param array $context  
 \* @return null  
 \*/  
 public function emergency($message, array $context = array());  
  
 /\*\*  
 \* \*\*必须\*\*立刻采取行动  
 \*  
 \* 例如：在整个网站都垮掉了、数据库不可用了或者其他的情况下，\*\*应该\*\*发送一条警报短信把你叫醒。  
 \*  
 \* @param string $message  
 \* @param array $context  
 \* @return null  
 \*/  
 public function alert($message, array $context = array());  
  
 /\*\*  
 \* 紧急情况  
 \*  
 \* 例如：程序组件不可用或者出现非预期的异常。  
 \*  
 \* @param string $message  
 \* @param array $context  
 \* @return null  
 \*/  
 public function critical($message, array $context = array());  
  
 /\*\*  
 \* 运行时出现的错误，不需要立刻采取行动，但必须记录下来以备检测。  
 \*  
 \* @param string $message  
 \* @param array $context  
 \* @return null  
 \*/  
 public function error($message, array $context = array());  
  
 /\*\*  
 \* 出现非错误性的异常。  
 \*  
 \* 例如：使用了被弃用的API、错误地使用了API或者非预想的不必要错误。  
 \*  
 \* @param string $message  
 \* @param array $context  
 \* @return null  
 \*/  
 public function warning($message, array $context = array());  
  
 /\*\*  
 \* 一般性重要的事件。  
 \*  
 \* @param string $message  
 \* @param array $context  
 \* @return null  
 \*/  
 public function notice($message, array $context = array());  
  
 /\*\*  
 \* 重要事件  
 \*  
 \* 例如：用户登录和SQL记录。  
 \*  
 \* @param string $message  
 \* @param array $context  
 \* @return null  
 \*/  
 public function info($message, array $context = array());  
  
 /\*\*  
 \* debug 详情  
 \*  
 \* @param string $message  
 \* @param array $context  
 \* @return null  
 \*/  
 public function debug($message, array $context = array());  
  
 /\*\*  
 \* 任意等级的日志记录  
 \*  
 \* @param mixed $level  
 \* @param string $message  
 \* @param array $context  
 \* @return null  
 \*/  
 public function log($level, $message, array $context = array());  
}

## 4. Psr\Log\LoggerAwareInterface

<?php  
  
namespace Psr\Log;  
  
/\*\*  
 \* logger-aware 定义实例  
 \*/  
interface LoggerAwareInterface  
{  
 /\*\*  
 \* 设置一个日志记录实例  
 \*  
 \* @param LoggerInterface $logger  
 \* @return null  
 \*/  
 public function setLogger(LoggerInterface $logger);  
}

## 5. Psr\Log\LogLevel

<?php  
  
namespace Psr\Log;  
  
/\*\*  
 \* 日志等级常量定义  
 \*/  
class LogLevel  
{  
 const EMERGENCY = 'emergency';  
 const ALERT = 'alert';  
 const CRITICAL = 'critical';  
 const ERROR = 'error';  
 const WARNING = 'warning';  
 const NOTICE = 'notice';  
 const INFO = 'info';  
 const DEBUG = 'debug';  
}