**日志接口**

本文档描述了日志类库的通用接口。

主要目标是让类库获得一个Psr\Log\LoggerInterface对象并能通过简单通用的方式来写日志。有自定义需求的框架和CMS可以根据情况扩展这个接口，但推荐保持和该文档的兼容性，以确保应用中使用到的第三方库能将日志集中写到应用日志里。

[RFC 2119](http://tools.ietf.org/html/rfc2119)中的必须(MUST)，不可(MUST NOT)，建议(SHOULD)，不建议(SHOULD NOT)，可以/可能(MAY)等关键词将在本节用来做一些解释性的描述。

关键词实现者在这个文档被解释为：在日志相关的库或框架实现LoggerInterface接口的开发人员。用这些实现者开发出来的类库的人都被称作用户。

**1. 规范**

**1.1 基础**

* LoggerInterface暴露八个接口用来记录八个等级(debug, info, notice, warning, error, critical, alert, emergency)的日志。
* 第九个方法是log，接受日志等级作为第一个参数。用一个日志等级常量来调用这个方法必须和直接调用指定等级方法的结果一致。用一个本规范中未定义且不为具体实现所知的日志等级来调用该方法必须抛出一个Psr\Log\InvalidArgumentException。不推荐使用自定义的日志等级，除非你非常确定当前类库对其有所支持。

**1.2 消息**

* 每个方法都接受一个字符串，或者一个有\_\_toString方法的对象作为message参数。实现者 可以对传入的对象有特殊的处理。如果没有，实现者 必须将它转换成字符串。
* message参数中可能包含一些可以被context参数的数值所替换的占位符。

占位符名字必须和context数组类型参数的键名对应。

占位符名字必须使用一对花括号来作为分隔符。在占位符和分隔符之间不能有任何空格。

占位符名字应该只能由A-Z，a-z，0-9，下划线\_和句号.组成。其它的字符作为以后占位符规范的保留字。

实现者 可以使用占位符来实现不同的转义和翻译日志成文。因为用户并不知道上下文数据会是什么，所以不推荐提前转义占位符。

下面提供一个占位符替换的例子，仅作为参考：

<?php

/\*\*

\* Interpolates context values into the message placeholders.

\*/

function interpolate($message, array $context = array())

{

// build a replacement array with braces around the context keys

$replace = array();

foreach ($context as $key => $val) {

$replace['{' . $key . '}'] = $val;

}

// interpolate replacement values into the message and return

return strtr($message, $replace);

}

// a message with brace-delimited placeholder names

$message = "User {username} created";

// a context array of placeholder names => replacement values

$context = array('username' => 'bolivar');

// echoes "Username bolivar created"

echo interpolate($message, $context);

**1.3 上下文**

* 每个方法接受一个数组作为context参数，用来存储不适合在字符串中填充的信息。数组可以包括任何东西。实现者必须确保他们尽可能包容的对context参数进行处理。一个context参数的给定值不可导致抛出异常，也不可产生任何PHP错误，警告或者提醒。
* 如果在context参数中传入了一个异常对象，它必须以exception作为键名。记录异常轨迹是通用的模式，并且可以在日志系统支持的情况下从异常中提取出整个调用栈。实现者在将exception当做异常对象来使用之前必须去验证它是不是一个异常对象，因为它可能包含着任何东西。

**1.4 助手类和接口**

* Psr\Log\AbstractLogger类可以让你通过继承它并实现通用的log方法来方便的实现LoggerInterface接口。而其他八个方法将会把消息和上下文转发给log方法。
* 类似的，使用Psr\Log\LoggerTrait只需要你实现通用的log方法。注意特性是不能用来实现接口的，所以你依然需要在你的类中implement LoggerInterface。
* Psr\Log\NullLogger是和接口一起提供的。它在没有可用的日志记录器时，可以为使用日志接口的用户们提供一个后备的“黑洞”。但是，当context参数的构建非常耗时的时候，直接判断是否需要记录日志可能是个更好的选择。
* Psr\Log\LoggerAwareInterface只有一个setLogger(LoggerInterface $logger)方法，它可以在框架中用来随意设置一个日志记录器。
* Psr\Log\LoggerAwareTrait特性可以被用来在各个类中轻松实现相同的接口。通过它可以访问到$this->logger。
* Psr\Log\LogLevel类拥有八个日志等级的常量。

**2. 包**

[psr/log](https://packagist.org/packages/psr/log)中提供了上文描述过的接口和类，以及相关的异常类，还有一组用来验证你的实现的单元测试。

**3. Psr\Log\LoggerInterface**

<?php

namespace Psr\Log;

/\*\*

\* Describes a logger instance

\*

\* The message MUST be a string or object implementing \_\_toString().

\*

\* The message MAY contain placeholders in the form: {foo} where foo

\* will be replaced by the context data in key "foo".

\*

\* The context array can contain arbitrary data, the only assumption that

\* can be made by implementors is that if an Exception instance is given

\* to produce a stack trace, it MUST be in a key named "exception".

\*

\* See https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-3-logger-interface.md

\* for the full interface specification.

\*/

interface LoggerInterface

{

/\*\*

\* System is unusable.

\*

\* @param string $message

\* @param array $context

\* @return null

\*/

public function emergency($message, array $context = array());

/\*\*

\* Action must be taken immediately.

\*

\* Example: Entire website down, database unavailable, etc. This should

\* trigger the SMS alerts and wake you up.

\*

\* @param string $message

\* @param array $context

\* @return null

\*/

public function alert($message, array $context = array());

/\*\*

\* Critical conditions.

\*

\* Example: Application component unavailable, unexpected exception.

\*

\* @param string $message

\* @param array $context

\* @return null

\*/

public function critical($message, array $context = array());

/\*\*

\* Runtime errors that do not require immediate action but should typically

\* be logged and monitored.

\*

\* @param string $message

\* @param array $context

\* @return null

\*/

public function error($message, array $context = array());

/\*\*

\* Exceptional occurrences that are not errors.

\*

\* Example: Use of deprecated APIs, poor use of an API, undesirable things

\* that are not necessarily wrong.

\*

\* @param string $message

\* @param array $context

\* @return null

\*/

public function warning($message, array $context = array());

/\*\*

\* Normal but significant events.

\*

\* @param string $message

\* @param array $context

\* @return null

\*/

public function notice($message, array $context = array());

/\*\*

\* Interesting events.

\*

\* Example: User logs in, SQL logs.

\*

\* @param string $message

\* @param array $context

\* @return null

\*/

public function info($message, array $context = array());

/\*\*

\* Detailed debug information.

\*

\* @param string $message

\* @param array $context

\* @return null

\*/

public function debug($message, array $context = array());

/\*\*

\* Logs with an arbitrary level.

\*

\* @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;

/\*\*

\* Describes a logger-aware instance

\*/

interface LoggerAwareInterface

{

/\*\*

\* Sets a logger instance on the object

\*

\* @param LoggerInterface $logger

\* @return null

\*/

public function setLogger(LoggerInterface $logger);

}

**5. Psr\Log\LogLevel**

<?php

namespace Psr\Log;

/\*\*

\* Describes log levels

\*/

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';

}