Logs

Symfony viene por defecto con una librería externa (Monolog) que permite crear logs que pueden ser almacenados en varios lugares diferentes.

Instalación

> *composer require symfony/monolog-bundle*

Escribir mensajes de log

Para escribir un mensaje en el log, primero hay obtener el servicio *logger* del contenedor de servicios:

```
use Psr\Log\LoggerInterface;

public function indexAction(LoggerInterface $logger)
{
    // alternative way of getting the logger
    // $logger = $this->get('logger');

    $logger->info('I just got the logger');
    $logger->error('An error occurred');

    $logger->critical('I left the oven on!', array(
        // include extra "context" info in your logs
        'cause' => 'in_hurry',
    ));

    // ...
}
```

Monolog cumple el estándar PSR-3, que obliga a implementar el interfaz *LoggerInterface*.

```php
<?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 = a
rray());

    /**
     * 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, unexpec
```

```
ted exception.
     *
     * @param string $message
     * @param array $context
     * @return null
     */
    public function critical($message, array $context = ar
ray());

    /**
     * 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 = arr
ay());

    /**
     * Normal but significant events.
     *
     * @param string $message
     * @param array $context
     * @return null
     */
    public function notice($message, array $context = arra
y());

    /**
     * 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());
}
```

Por lo tanto, tenemos los siguientes métodos, que se corresponden con nivel de
criticidad:

- debug($message, $context)
- info($message, $context)
- notice($message, $context)
- warning($message, $context)
- error($message, $context)
- critical($message, $context)
- alert($message, $context)
- emergency($message, $context)

Y un método genérico, con un parámetro que indica en nivel:

- log($level, $message, $context);

```php
<?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';
}
```

Cómo utilizar el servicio logger dentro de otro servicio

Para utilizar el servicio logger dentro de otro servicio, basta con inyectarlo en el constructor del servicio:

```
use Psr\Log\LoggerInterface;
class MiServicio {

  public function __construct(LoggerInterface $logger) {
    $this->logger = $logger;
  }
}
```

Configuración

El comportamiento de los logs es uno de los que es significativamente distinto en producción que en desarrollo. Así que encontraremos diferentes configuraciones en desarrollo y en producción.

Configuración por defecto de producción:

```
//prod
monolog:
    handlers:
        main:
            type: fingers_crossed
            action_level: error
            handler: nested
            excluded_404s:
                # regex: exclude all 404 errors from the l
ogs
                - ^/
        nested:
            type: stream
            path: "%kernel.logs_dir%/%kernel.environment%.
log"
            level: debug
        console:
            type:   console
            process_psr_3_messages: false
```

```
            channels: ["!event", "!doctrine"]
```

Configuración por defecto de desarrollo:

```
//dev
monolog:
    handlers:
        main:
            type: stream
            path: "%kernel.logs_dir%/%kernel.environment%.
log"
            level: debug
            channels: ["!event"]
        # uncomment to get logging in your browser
        # you may have to allow bigger header sizes in you
r Web server configuration
        #firephp:
        #    type: firephp
        #    level: info
        #chromephp:
        #    type: chromephp
        #    level: info
        console:
            type:   console
            process_psr_3_messages: false
            channels: ["!event", "!doctrine", "!console"]
```

Opciones de configuración más usuales:

- type: Indica el tipo de Handler que se quiere utilizar
- level: Indica el nivel mínimo que debe tener el mensaje para ser loggeado.

Lista de manejadores

La lista de manejadores de log y sus opciones se puede encontrar en el siguiente enlace:

https://github.com/symfony/monolog-bundle/blob/master/DependencyInjection/Configuration.php#L25

El manejador especial fingers_crossed

Este manejador es especial. Guarda en memoria todos los mensajes de log, sean del nivel que sean, y si en algún momento de la petición alguno de los mensajes alcanza al menos el nivel indicado en *action_level*, entonces envía TODOS los logs al manejador indicado en *handler*.

De esta forma, en caso de error grave, la traza loggeada será completa, con los mensajes de todos los niveles.

```
monolog:
    handlers:
        main:
            type: fingers_crossed
            action_level: error
            handler: nested
        nested:
            type: stream
            path: '%kernel.logs_dir%/%kernel.environment%.
log'
            level: debug
        console:
            type: console
            level: debug
```

Configuración de log con ficheros rotativos

Configurar Monolog para generar ficheros de log rotativos es muy sencillo:

```
monolog:
    handlers:
        main:
            type:  rotating_file
            path:  '%kernel.logs_dir%/%kernel.environment%
.log'
            level: debug
            # número máximo de ficheros que guardar
            # por defecto, 0, que significa infinitos fich
eros
            max_files: 10
```

Cómo configurar Monolog para enviar correos

Monolog puede ser configurado para enviar un correo cuando ocurre un error en la aplicación. La configuración para esto requiere un para de handlers anidados para evitar recibir demasiados correos. Esta configuración parece complicada al principio pero cada handler es bastante intuitivo cuando se miran por separado.

```
monolog:
    handlers:
        mail:
            type:         fingers_crossed
            action_level: critical
            handler:      deduplicated
        deduplicated:
            type:    deduplication
            time:    60
            handler: swift
        swift:
            type:       swift_mailer
            from_email: 'error@example.com'
            to_email:   'error@example.com'
            # or list of recipients
            # to_email:   ['dev1@example.com', 'dev2@examp
le.com', ...]
```

```
            subject:    'An Error Occurred! %%message%%'
            level:      debug
            formatter:  monolog.formatter.html
            content_type: text/html
```

El handler *mail* es un handler fingers_crossed, lo que significa que solamente se “disparará” cuando el nivel sea en este caso, *critical*.

Por defecto el nivel *critical* solamente se alcanza en errores HTTP 5xx. Si se alcanza este nivel el handle fingers_crossed pasará todos los logs de dicha petición al handler *deduplicated*.

El handler deduplicated simplemente elimina los mensajes repetidos durante el periodo de tiempo especificado y se los pasa al handler swift. De esta forma se reducen los correos que pueden llegar en momentos de fallo crítico.

El handler swift es el que se encarga finalmente de mandar el correo con el mensaje.

El manejador especial group

El handler *group* hace que un mensaje sea manejado por más de un handler. Aquí podemos ver un ejemplo en el que los mensajes que pasan el filtro del handler fingers_crossed al mismo tiempo escritos en un fichero y enviados por correo.

```
monolog:
    handlers:
        main:
            type:         fingers_crossed
            action_level: critical
            handler:      grouped
        grouped:
            type:    group
            members: [streamed, deduplicated]
        streamed:
            type:  stream
            path:  '%kernel.logs_dir%/%kernel.environment%
.log'
            level: debug
        deduplicated:
            type:    deduplication
            handler: swift
        swift:
            type:       swift_mailer
            from_email: 'error@example.com'
```

```
            to_email:   'error@example.com'
            subject:    'An Error Occurred! %%message%%'
            level:      debug
            formatter:  monolog.formatter.html
            content_type: text/html
```

Cómo separar logs en distintos ficheros según el canal

Symfony organiza los mensajes de log en canales: doctrine, event, security, request... El canal aparece en el mensaje de log y puede ser utilizado para configurar distintos *handlers* según el canal.

```
monolog:
    handlers:
        security:
            level:    debug
            type:     stream
            path:     '%kernel.logs_dir%/security.log'
            channels: [security]

        main:
            level: debug
            type: stream
            path: '%kernel.logs_dir%/%kernel.environment%.
log'
            channels: ['!security']
```

```
channels: ~    # Incluye todos los canales

channels: security  # Incluye solamente el canal 'security
'
```

```
channels: '!security' # Incluye todos los canales menos se
curity

channels: [security, events]   # Incluye solamente los can
ales 'security' y 'events'
channels: ['!security', '!events'] # Incluye todos los can
ales menos 'security' y 'events'
```

Cómo crear canales

```
monolog:
    channels: ['micanal','otrocanal']
    handlers:
        main:
            type: stream
            path: '%kernel.logs_dir%/%kernel.environment%.
log'
            level: debug
```

```
      $loggerMiCanal = $this->get('monolog.logger.micanal'
);
      $loggerMiCanal->info('Este es un mensaje con nivel i
nfo');

      $loggerOtroCanal = $this->get('monolog.logger.otroca
nal');
      $loggerOtroCanal->info('Este es un mensaje con nivel
```

```
info');
```

Cómo deshabilitar la precisión de microsegundos para ganar rapidez

Poniendo el parámetro *use_microseconds* a *false*, evitamos que se llame a la función *microtime(true)* y a su posterior parseado. Si tenemos una aplicación que genera una gran cantidad de mensajes de log, se puede llegar a notar la mejora de eficiencia.

```
monolog:
    use_microseconds: false
    handlers:
        main:
            type: stream
            path: '%kernel.logs_dir%/%kernel.environment%.
log'
            level: debug
```