El profiler

Para disponer del profiler en nuestra aplicación, si no lo tenemos ya, basta con instalarlo con Flex:

> *composer require profiler --dev*

Es una dependencia únicamente de desarrollo.

Secciones del profiler

Las secciones del profiler son:

- Request / Response
- Performance
- Validator
- Forms
- Exception
- Logs
- Events
- Routing
- Cache
- Translation
- Security
- Twig
- Doctrine
- E-Mails
- Debug
- Configuration

Cómo crear un Data Collector personalizado

El Profiler de Symfony obtiene la información utilizando unas clases especiales llamadas **data collectors**. Podemos crear nuestros propios data collectors para mostrar información en el Profiler.

Crear una clase que extienda de DataCollector

Un data collector es una clase PHP que implementa el interfaz **DataCollectorInterface**. Podemos implementar dicho interfaz o bien extender de la clase DataCollector, que ya implementa dicho interfaz y tiene algunas otras utilidades interesantes y la propiedad $this->data.

El siguiente ejemplo es un colector personalizado que guarda información sobre la request:

```
// src/DataCollector/RequestCollector.php
namespace App\DataCollector;

use Symfony\Component\HttpKernel\DataCollector\DataCollect
or;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
```

```
class RequestCollector extends DataCollector
{
    public function collect(Request $request, Response $re
sponse, \Exception $exception = null)
    {
        $this->data = array(
            'method' => $request->getMethod(),
            'acceptable_content_types' => $request->getAcc
eptableContentTypes(),
        );
    }

    public function reset()
    {
        $this->data = array();
    }

    public function getName()
    {
        return 'app.request_collector';
    }

    // ...
}
```

- el método collect()

Almacena en propiedades de la clase la información recogida. Si

extendemos de DataCollector ya tenemos la propiedad $this->data.

PRECAUCIÓN:
El profiler serializa la instancia del data collector, por lo que no deberíamos almacenar datos que no se puedan serializar.

- el método reset()

Este método es llamado entre dos peticiones. Se utiliza para resetear el estado del profiler borrando la información almacenada de la petición anterior.

- el método getName():

Devuelve el identificador del colector. Este identificador debe ser único en toda la aplicación.

Enabling Custom Data Collectors

Si tenemos activo el autoconfigure, Symfony configurará automáticamente nuestro collector. De no ser así, lo tendremos que configurar con el tag **data_collector**.

Adding Web Profiler Templates

La información recogida por nuestro colector puede ser mostrada tanto en la barra de depuración como en el profiler.

Para hacer esto, necesitamos crear una plantilla de Twig que extienda

de @WebProfiler/Profiler/layout.html.twig y que incluya algunos bloques específicos.

Antes de crear la plantilla de Twig, vamos a crear un par de métodos para acceder a lso datos recogidos.

```
// src/DataCollector/RequestCollector.php
namespace App\DataCollector;

use Symfony\Component\HttpKernel\DataCollector\DataCollect
or;

class RequestCollector extends DataCollector
{
    // ...

    public function getMethod()
    {
        return $this->data['method'];
    }

    public function getAcceptableContentTypes()
    {
        return $this->data['acceptable_content_types'];
    }
}
```

Para mostrar información en la barra de debug necesitamos definir el bloque **toolbar** y establecer los valores de dos variables llamadas **icon** y **text**.

```
{% extends '@WebProfiler/Profiler/layout.html.twig' %}

{% block toolbar %}
    {% set icon %}
        {# this is the content displayed as a panel in the
 toolbar #}
        <svg xmlns="http://www.w3.org/2000/svg"> ... </svg
>
        <span class="sf-toolbar-value">Request</span>
    {% endset %}

    {% set text %}
        {# this is the content displayed when hovering the
 mouse over
           the toolbar panel #}
        <div class="sf-toolbar-info-piece">
            <b>Method</b>
            <span>{{ collector.method }}</span>
        </div>

        <div class="sf-toolbar-info-piece">
            <b>Accepted content type</b>
            <span>{{ collector.acceptableContentTypes|join
```

```
(', ') }}</span>
        </div>
    {% endset %}

    {# the 'link' value set to 'false' means that this pan
el doesn't
       show a section in the web profiler #}
    {{ include('@WebProfiler/Profiler/toolbar_item.html.tw
ig', { link: false }) }}
{% endblock %}
```

Para crear una sección en el web profiler hay que definir los bloques **head** (opcional), **menú** y **panel**.

```
{% extends '@WebProfiler/Profiler/layout.html.twig' %}

{% block toolbar %}
    {% set icon %}
        {# ... #}
    {% endset %}

    {% set text %}
        <div class="sf-toolbar-info-piece">
            {# ... #}
        </div>
    {% endset %}
```

```
    {{ include('@WebProfiler/Profiler/toolbar_item.html.twig', { 'link': true }) }}
{% endblock %}

{% block head %}
    {# Optional. Here you can link to or define your own CSS and JS contents. #}
    {# Use {{ parent() }} to extend the default styles instead of overriding them. #}
{% endblock %}

{% block menu %}
    {# This left-hand menu appears when using the full-screen profiler. #}
    <span class="label">
        <span class="icon"><img src="..." alt=""/></span>
        <strong>Request</strong>
    </span>
{% endblock %}

{% block panel %}
    {# Optional, for showing the most details. #}
    <h2>Acceptable Content Types</h2>
    <table>
        <tr>
            <th>Content Type</th>
        </tr>
```

```
        {% for type in collector.acceptableContentTypes %}
        <tr>
            <td>{{ type }}</td>
        </tr>
        {% endfor %}
    </table>
{% endblock %}
```

Todos los bloques tienen acceso a la instancia del data collector.

Finalmente, hay que configurar nuestro servicio indicando una tac que contenga la plantilla.

```
# config/services.yaml
services:
    App\DataCollector\RequestCollector:
        tags:
            -
                name:     data_collector
                template: 'data_collector/template.html.tw
ig'
                # must match the value returned by the get
Name() method
                id:       'app.request_collector'
                # optional priority
                # priority: 300
        public: false
```

La posición de cada panel en la toolbar está definida por la prioridad que indiquemos.

Cómo acceder a los datos del profiler programáticamente

En la mayor parte de las ocasiones, la información accedida y analizada utilizando la visualización web. Sin embargo, también podemos acceder a esta información programáticamente gracias a los métodos del servicio **profiler**.

Cuando el objeto response está disponible, utiliza el método loadProfileFromResponse() para acceder a su profile asociado.

```
// ... $profiler es el servicio 'profiler'
$profile = $profiler->loadProfileFromResponse($response);
```

Cuando el profiler almacena datos de un request, también asocia un token. El token está disponible en el header HTTP X-Debug-Token de la respuesta. Con este token, podemos acceder al profile de cualquier respuesta anterior gracias al método loadProfile():

```
$token = $response->headers->get('X-Debug-Token');
$profile = $container->get('profiler')->loadProfile($token
);
```

Si está activado el profiler pero no la barra de debug, habría que utilizar las herramientas de desarrollo del navegador para conseguir el valor de la cabecera HTTP X-Debug-Token.

El profiler service también proporciona el método find() para buscar tokens basándose en algún criterio.

```
// obtiene los últimos 20 tokens
$tokens = $profiler->find('', '', 20, '', '', '');

// obtiene los últimos 20 tokens de todas las URL que cont
ienen /admin/
$tokens = $profiler->find('', '/admin/', 20, '', '', '');

// obtiene los últimos 20 tokens de peticiones realizadas
desde la ip 127.0.0.1
$tokens = $profiler->find('127.0.0.1', '', 20, 'POST', '',
 '');

// obtiene los últimos 20 tokens de peticiones que ocurrie
ron entre 2 fechas dadas
$tokens = $profiler->find('', '', 20, '', '4 days ago', '2
 days ago');
```

Finalmente, si queremos manipular datos del profiler desde una máquina diferente a la cual la información fue generada, tenemos un par de comandos que nos permiten importar y exportar dicha información entre máquinas.

```
# en la máquina de producción
$ php bin/console profiler:export > profile.data
```

```
# en la máquina de desarrollo
$ php bin/console profiler:import profile.data
```

Cómo deshabilitar el profiler en una acción concreta

En Symfony 4 se ha eliminado la posibilidad de habilitar/deshabilitar el Profiler mediante *matchers*.

Lo que sí que podemos hacer es deshabilitar el Profiler en una acción concreta.

```
use Symfony\Component\HttpKernel\Profiler\Profiler;
// ...

class DefaultController
{
    // ...

    public function someMethod(Profiler $profiler)
    {
        // for this particular controller action, the prof
iler is disabled
        $profiler->disable();

        // ...
    }
}
```

Para poder inyectar el profiler en nuestra acción, necesitamos crear un alias en nuestro services.yaml

```
# config/services.yaml
services:
    Symfony\Component\HttpKernel\Profiler\Profiler: '@prof
iler'
```

Cómo cambiar la ubicación del Profiler Storage

El profiler almacena los datos recogidos en el directorio **%kernel.cache_dir%/profiler/**.

Si queremos utilizar otro directorio para almacenar estos datos, hay que definir la opción **framework.profiler.dsn**:

```
# config/packages/dev/web_profiler.yaml
framework:
    profiler:
        dsn: 'file:/tmp/symfony/profiler'
```