EBook Gratis

APRENDIZAJE symfony

Free unaffiliated eBook created from
Stack Overflow contributors.

#symfony

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: symfony

It is an unofficial and free symfony ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official symfony.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con Symfony

Observaciones

Symfony es un conjunto de componentes PHP reutilizables, que se pueden usar por separado o como parte de Symfony Framework.

Como la mayoría de los frameworks, Symfony resuelve problemas técnicos recurrentes para ti (como autenticación, enrutamiento, etc.) para que puedas concentrar tu tiempo en los problemas de negocios reales que estás tratando de resolver.

Sin embargo, al contrario de otros marcos, los componentes de Symfony están desacoplados entre sí, lo que le permite seleccionar los que necesita. En lugar de tener que adaptar su aplicación a su marco, puede adaptar el marco a sus necesidades.

Esto es lo que hace que Symfony sea muy popular y permite que otros proyectos y marcos (incluidos Laravel, Drupal, Magento y Composer) utilicen los componentes sin tener que usar el marco completo.

Fuente abierta

Symfony es un proyecto de código abierto. Vea cómo puede contribuir .

Documentacion oficial

La documentación oficial de Symfony se puede encontrar en el sitio web de Symfony.

Versiones

Symfony 3

Versión	Fin de la vida	Fecha de lanzamiento
3.3	07/2018	2017-05-29
3.2	01/2018	2016-11-30
3.1	07/2017	2016-05-30
3.0	01/2017	2015-11-30

Symfony 2

Versión	Fin de la vida	Fecha de lanzamiento
2.8	11/2019	2015-11-30
2.7	05/2019	2015-05-30
2.6	01/2016	2014-11-28
2.5	07/2015	2014-05-31
2.4	01/2015	2013-12-03
2.3	05/2017	2013-06-03
2.2	05/2014	2013-03-01
2.1	11/2013	2012-09-06
2.0	09/2013	2011-07-28

Examples

Creando un nuevo proyecto de Symfony usando el instalador de Symfony

El instalador de Symfony es una herramienta de línea de comandos que te ayuda a crear nuevas aplicaciones de Symfony. Requiere PHP 5.4 o superior.

Descarga e instalación del instalador de Symfony en Linux / MacOS

Abre un terminal y ejecuta los siguientes comandos:

```
sudo mkdir -p /usr/local/bin
sudo curl -LsS https://symfony.com/installer -o /usr/local/bin/symfony
sudo chmod a+x /usr/local/bin/symfony
```

Esto crea un ejecutable global de `symfony` que puede ser llamado desde cualquier lugar. Solo debes hacer esto una vez: ahora puedes crear tantos proyectos Symfony con él como quieras.

Creando un nuevo proyecto con la última versión de Symfony.

Una vez que se instala el instalador, puedes usarlo para crear un nuevo proyecto de Symfony. Ejecuta el siguiente comando:

```
symfony new my_project_name
```

Este comando creará un nuevo directorio (llamado `my_project_name`) que contiene la versión más reciente de la edición estándar de Symfony . También instalará todas sus dependencias (incluidos los componentes reales de Symfony) usando Composer.

Creando un nuevo proyecto usando una versión específica de Symfony

Si quieres seleccionar una versión específica de Symfony en lugar de la última, puedes usar el segundo argumento opcional del `new` comando.

Para seleccionar una versión menor:

```
symfony new my_project_name 3.2
```

Para seleccionar una versión de parche:

```
symfony new my_project_name 3.2.9
```

Para seleccionar una versión beta o lanzar un candidato:

```
symfony new my_project 2.7.0-BETA1
symfony new my_project 2.7.0-RC1
```

Para seleccionar la versión más reciente de Soporte a largo plazo (LTS):

```
symfony new my_project_name lts
```

Creando un nuevo proyecto de Symfony usando Composer

Si, por alguna razón, usar el instalador de Symfony no es una opción, también puedes crear un nuevo proyecto usando Composer. En primer lugar, asegúrese de haber instalado Composer .

A continuación, puede usar el comando `create-project` para crear un nuevo proyecto:

```
composer create-project symfony/framework-standard-edition my_project_name
```

Al igual que el instalador de Symfony, esto instalará la última versión de Symfony Standard Edition en un directorio llamado `my_project_name` y luego instalará sus dependencias (incluidos los componentes de Symfony).

Instalando una versión específica de Symfony

Al igual que con el instalador de Symfony, puedes seleccionar una versión específica de Symfony proporcionando un tercer argumento opcional:

```
composer create-project symfony/framework-standard-edition my_project_name "2.8.*"
```

Sin embargo, tenga en cuenta que no todos los alias de versión (como `lts` por ejemplo) están disponibles aquí.

Ejecutando la aplicación Symfony usando el servidor web incorporado de PHP

Después de crear una nueva aplicación Symfony , puedes usar el comando `server:run` para iniciar un servidor web PHP simple, para que puedas acceder a tu nueva aplicación desde tu navegador web:

```
cd my_project_name/
php bin/console server:run
```

Ahora puede visitar http: // localhost: 8000 / para ver la página de bienvenida de Symfony.

Importante: al utilizar el servidor web incorporado es ideal para el desarrollo, **no se** debe utilizar en la producción. Utilice un servidor web completo como Apache o Nginx en su lugar.

Lea Empezando con Symfony en línea: https://riptutorial.com/es/symfony/topic/9448/empezando-con-symfony

Capítulo 2: Contenedor de servicio

Introducción

Una aplicación de Symfony suele estar compuesta por una gran cantidad de objetos que realizan diferentes tareas, como repositorios, controladores, correos, etc. En Symfony, estos objetos se denominan **servicios** y se definen en `app/config/services.yml` o en uno de Los paquetes instalados.

El **contenedor de servicios** sabe cómo crear una instancia de estos servicios, y mantiene una referencia de ellos para que no tengan que ser instanciados dos veces. Si un servicio tiene dependencias, también las instanciará.

Examples

Recuperar un servicio del contenedor.

```
$logger = $container->get('logger');
```

Esto recuperará el servicio con el identificador de servicio "logger" del contenedor, un objeto que implementa `Psr\Log\LoggerInterface` .

Lea Contenedor de servicio en línea: https://riptutorial.com/es/symfony/topic/10183/contenedor-de-servicio

Capítulo 3: Controladores

Introducción

Un controlador en Symfony es un PHP ejecutable (una función, un método en un objeto o un cierre) que recibe una solicitud HTTP y devuelve una respuesta HTTP. Una respuesta HTTP puede contener cualquier cosa: una página HTML, una cadena JSON, una descarga de archivos, etc.

Para decirle a Symfony qué controlador debe manejar una determinada solicitud, necesitas configurar una ruta .

Sintaxis

- $ this-> generateUrl ('route_name', ['placeholder' => 'value']); // genera una URL para la ruta `route_name` con un marcador de posición
- $ this-> render ('template.html.twig'); // representa una plantilla de Twig y devuelve un objeto de respuesta
- $ this-> render ('template.html.twig', ['parámetro' => $ valor]); // representa una plantilla de Twig con un parámetro
- lanzar $ this-> createNotFoundException ('Mensaje'); // lanza una NotFoundHttpException que hará que Symfony devuelva una respuesta 404

Observaciones

Los controladores deben ser pequeños y centrarse en el manejo de las solicitudes HTTP: la lógica de negocios real de su aplicación debe delegarse en diferentes partes de su aplicación, por ejemplo, su modelo de dominio.

Examples

Una clase de controlador simple

```
// src/AppBundle/Controller/HelloWorldController.php
namespace AppBundle\Controller;

use Symfony\Component\HttpFoundation\Response;

class HelloWorldController
{
    public function helloWorldAction()
    {
        return new Response(
            '<html><body>Hello World!</body></html>'
        );
    }
}
```

Renderizando una plantilla de ramita

La mayoría de las veces, deseará representar las respuestas HTML de una plantilla en lugar de codificar el código HTML en su controlador. Además, sus plantillas no serán estáticas, sino que contendrán marcadores de posición para los datos de la aplicación. Por defecto, Symfony viene con Twig, un poderoso lenguaje de creación de plantillas.

Para usar Twig en tu controlador, extiende la clase de `Controller` base de Symfony:

```
// src/AppBundle/Controller/HelloWorldController.php
namespace AppBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\HttpFoundation\Response;

class HelloWorldController extends Controller
{
    public function helloWorldAction()
    {
        $text = 'Hello World!';

        return $this->render('hello-world.html.twig', ['text' => $text]);
    }
}
```

Cree la plantilla Twig (ubicada en `app/Resources/views/hello-world.html.twig`):

```
<html><body>{{ text }}</body></html>
```

Twig reemplazará automáticamente el marcador de posición `{{ text }}` con el valor del parámetro de `text` , pasado por el controlador. Esto representará la siguiente salida HTML:

```
<html><body>Hello World!</body></html>
```

Devolviendo una página 404 (No encontrada)

A veces desea devolver una respuesta 404 (No encontrado) porque el recurso solicitado no existe. Symfony te permite hacerlo lanzando una `NotFoundHttpException` .

El controlador básico de Symfony expone un método `createNotFoundException` que crea la excepción para ti:

```
public function indexAction()
{
    // retrieve the object from database
    $product = ...;

    if (!$product) {
        throw $this->createNotFoundException('The product does not exist');
    }

    // continue with the normal flow if no exception is thrown
```

```
    return $this->render(...);
}
```

Usando datos del objeto Request

Si necesita acceder al objeto de `Request` (por ejemplo, para leer los parámetros de consulta, leer un encabezado HTTP o procesar un archivo cargado), puede recibir la solicitud como un argumento de método agregando un argumento de tipo insinuado:

```
use Symfony\Component\HttpFoundation\Request;

public function indexAction(Request $request)
{
    $queryParam = $request->query->get('param');

    // ...
}
```

Symfony reconocerá la sugerencia de tipo y agregará el argumento de solicitud a la llamada del controlador.

Lea Controladores en línea: https://riptutorial.com/es/symfony/topic/10085/controladores

Capítulo 4: Enrutamiento

Introducción

El enrutamiento es el proceso de asignar una URL a un controlador . Symfony tiene un potente componente de enrutamiento que te permite definir rutas.

El componente de enrutamiento admite varios formatos de configuración: anotaciones, YAML, XML y PHP en bruto.

Parámetros

Parámetro	Detalles
nombre	El nombre de la ruta. Ejemplo: `book_show`
camino	La ruta (puede contener comodines). Ejemplo: `/book/{isbn}`
por defecto	Valores por defecto de los parámetros.

Examples

Rutas sencillas

Usando YAML:

```
# app/config/routing.yml
blog_list:
    path:     /blog
    defaults: { _controller: AppBundle:Blog:list }
```

Uso de anotaciones:

```
// src/AppBundle/Controller/BlogController.php
namespace AppBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;

class BlogController extends Controller
{
    /**
     * @Route("/blog", name="blog_list")
     */
    public function listAction()
    {
        // ...
    }
```

```
}
```

Una solicitud para la URL de `/blog` será manejada por el método `listAction()` del `BlogController` dentro de `AppBundle` .

Rutas con marcadores de posición.

Usando YAML:

```
# app/config/routing.yml
blog_show:
    path:     /blog/{slug}
    defaults: { _controller: AppBundle:Blog:show }
```

Uso de anotaciones:

```
// src/AppBundle/Controller/BlogController.php
namespace AppBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;

class BlogController extends Controller
{
    /**
     * @Route("/blog/{slug}", name="blog_show")
     */
    public function showAction($slug)
    {
        // ...
    }
}
```

Cualquier solicitud con una URL coincidente `/blog/*` será manejada por el método `showAction()` del `BlogController` dentro de `AppBundle` . La acción del controlador recibirá el valor del marcador de posición como un argumento de método.

Por ejemplo, una solicitud de `/blog/my-post` activará una llamada a `showAction()` con un argumento `$slug` contiene el valor `my-post` . Usando ese argumento, la acción del controlador puede cambiar la respuesta dependiendo del valor del marcador de posición, por ejemplo, recuperando la publicación del blog con el slug `my-post` de la base de datos.

Valores por defecto para marcadores de posición

Si desea tener un marcador de posición que se puede omitir, puede asignarle un valor predeterminado:

Usando YAML:

```
# app/config/routing.yml
blog_list:
    path:      /blog/{page}
```

```
    defaults:  { _controller: AppBundle:Blog:list, page: 1 }
    requirements:
        page: '\d+'
```

Uso de anotaciones:

```
// src/AppBundle/Controller/BlogController.php
namespace AppBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;

class BlogController extends Controller
{
    /**
     * @Route("/blog/{page}", name="blog_list", requirements={"page": "\d+"})
     */
    public function listAction($page = 1)
    {
        // ...
    }
}
```

En este ejemplo, las URL `/blog` y `/blog/1` coincidirán con la ruta `blog_list` y serán manejadas por el método `listAction()` . En el caso de `/blog` , `listAction()` aún recibirá el argumento `$page` , con el valor predeterminado `1` .

Lea Enrutamiento en línea: https://riptutorial.com/es/symfony/topic/10084/enrutamiento

Capítulo 5: La solicitud

Introducción

La clase Request de Symfony es una representación orientada a objetos de la solicitud HTTP. Contiene información como la URL, la cadena de consulta, los archivos cargados, las cookies y otros encabezados que provienen del navegador.

Sintaxis

- $ solicitud-> getPathInfo (); // devuelve la ruta (parte local de la URL) que se solicita (pero sin la cadena de consulta). Es decir, cuando visite https://example.com/foo/bar?key=value , esto contendrá / foo / bar
- $ solicitud-> consulta-> obtener ('id'); // devuelve un parámetro de cadena de consulta ($ _GET)
- $ solicitud-> consulta-> obtener ('id', 1); // devuelve un parámetro de cadena de consulta con un valor predeterminado
- $ solicitud-> solicitud-> obtener ('nombre'); // devuelve una variable de cuerpo de solicitud ($ _POST)
- $ solicitud-> archivos-> obtener ('archivo adjunto'); // devuelve una instancia de UploadedFile identificada por "archivo adjunto"
- $ request-> cookies-> get ('PHPSESSID'); // devuelve el valor de una cookie ($ _COOKIE)
- $ request-> headers-> get ('content_type'); // devuelve un encabezado de solicitud HTTP
- $ solicitud-> getMethod (); // devuelve el método de solicitud HTTP (GET, POST, PUT, DELETE, etc.)
- $ request-> getLanguages (); // devuelve una matriz de idiomas que el cliente acepta

Examples

Obtener un parámetro de cadena de consulta

Digamos que queremos construir una lista de productos paginada, donde el número de la página se pasa como un parámetro de cadena de consulta. Por ejemplo, para obtener la tercera página, iría a:

```
http://example.com/products?page=3
```

La solicitud HTTP sin formato se vería así:

```
GET /products?page=3 HTTP/1.1
Host: example.com
Accept: text/html
User-Agent: Mozilla/5.0 (Macintosh)
```

Para obtener el número de página del objeto de solicitud, puede acceder a la propiedad de `query` :

```
$page = $request->query->get('page'); // 3
```

En el caso de un parámetro de `page` , es probable que desee pasar un valor predeterminado en caso de que el parámetro de cadena de consulta no esté establecido:

```
$page = $request->query->get('page', 1);
```

Esto significa que cuando alguien visita http://example.com/products (tenga en cuenta la ausencia de la cadena de consulta), la variable `$page` contendrá el valor predeterminado `1` .

Creando un objeto de solicitud a partir de variables globales

PHP expone una serie de *variables globales* que contienen información sobre la solicitud HTTP, como `$_POST` , `$_GET` , `$_FILES` , `$_SESSION` , etc. La clase `Request` contiene un `createFromGlobals()` estático `createFromGlobals()` para instanciar una solicitud Objeto basado en estas variables:

```
use Symfony\Component\HttpFoundation\Request;

$request = Request::createFromGlobals();
```

Cuando uses el framework Symfony, no deberías crear una instancia del objeto de solicitud. En su lugar, debe usar el objeto del que se `app.php` instancia cuando el marco se reinicia en `app.php` / `app_dev.php` . Por ejemplo, por tipo insinuando el objeto de solicitud en su controlador .

Accediendo a una variable POST

Para obtener el contenido de un formulario que se envía con `method="post"` , use la propiedad `post` :

```
$name = $request->request->get('name');
```

Accediendo a los contenidos de una cookie.

```
$id = $request->cookies->get('PHPSESSID');
```

Esto devolverá el valor de la cookie 'PHPSESSID' enviada por el navegador.

Lea La solicitud en línea: https://riptutorial.com/es/symfony/topic/10097/la-solicitud

Creditos

S. No	Capítulos	Contributors
1	Empezando con Symfony	Braiam, Code Lover, Community, Nic Wortel, Paul Crovella
2	Contenedor de servicio	Nic Wortel
3	Controladores	Nic Wortel
4	Enrutamiento	Nic Wortel
5	La solicitud	Nic Wortel