

Symfony - Bases - 4 - Profiler

⌚ Cours	Symfony
⌚ Type	Guide
📎 Supports	https://symfony.com/doc/current/profiler.html
⌚ Difficulté	Moyen



Le Profiler, c'est un outil ultra complet vous permettant d'analyser votre application web en tant réel et de déboguer votre programme. De quoi bien éviter les prises de tête !

Pour commencer, tapez la commande suivante : `composer require symfony/debug-pack`

i Cette commande va vous importer pleins de nouveaux composants trop cool (ça se passe dans votre fichier composer.json, dans le dossier vendor et dans votre dossier config mais pour le moment, il vaut mieux éviter de se perdre là-dedans. C'est un peu comme quand vous installez un programme, vous l'utilisez sans regarder ce qui se passe et ça ne vous empêche pas le comprendre).

Actualisez maintenant votre page pour voir apparaître une super-barre 🚀 en bas !



En survolant l'onglet en bas à gauche @app_quelquechose, vous pouvez apercevoir le Controller responsable de l'affichage de la page (dans notre cas c'est toujours testController) suivi d'une info sur le temps que la page a mis à charger, des infos mémoires, d'un onglet réservé aux erreurs et au code déprécié ainsi qu'un onglet 🌱 réservé à vos Twig.

A droite, vous avez les infos sur le serveur et les infos sur les versions de vos différents outils.

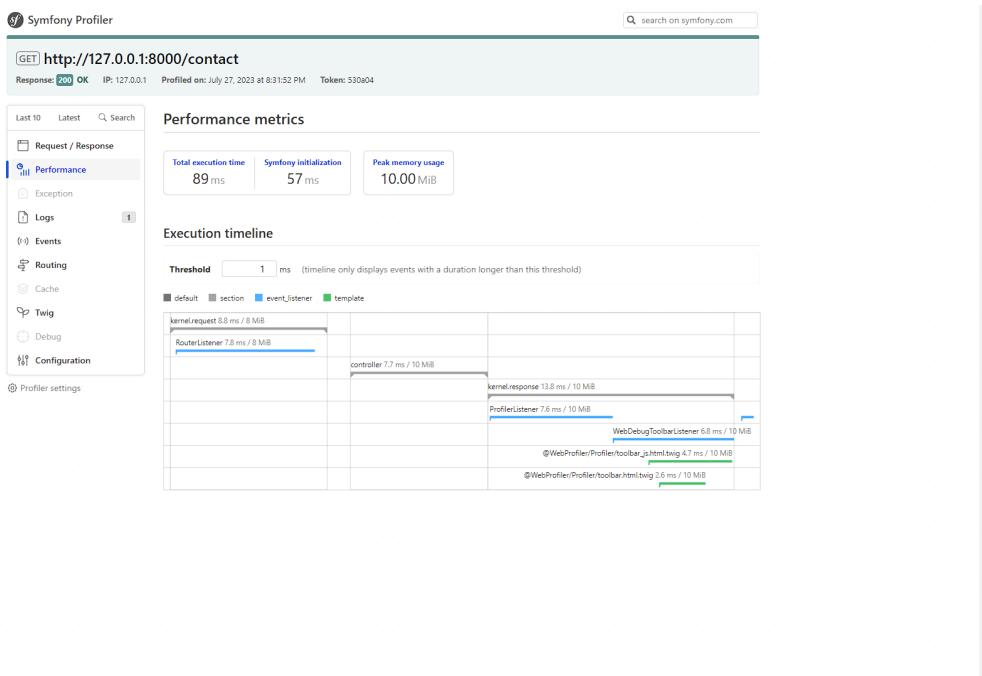
Ouvrons le Profiler !

Et pour ça, rien de plus simple, il vous suffit de cliquer sur l'icône de votre choix dans la barre.

The screenshot shows the Symfony Profiler interface for a request to `/contact`. The top bar displays the status as `200 OK`, the IP as `127.0.0.1`, and the profiling date as `July 27, 2023 at 8:31:52 PM`. The main area is titled `testController :: showContact`. It includes sections for `Request / Response`, `Logs`, `Events`, `Routing`, `Cache`, `Twig`, `Debug`, and `Configuration`. The `Request / Response` section is expanded, showing `GET Parameters` (none), `POST Parameters` (none), and `Uploaded Files` (none). The `Request Attributes` table lists parameters like `_controller`, `_route`, `_route_params`, and `_stopwatch_token`. The `Request Headers` table lists headers such as `accept`, `accept-encoding`, `accept-language`, `cache-control`, `connection`, `content-length`, `content-type`, `host`, and `mod-rewrite`. A message at the bottom says `Bienvenue sur votre Profiler perso.`.

Sur la gauche, vous pouvez trouver tout un tas d'info sur votre application web. Les Routes utilisées, votre configurations, vos pages twig et le nombres de fois que vous les avez ouvertes mais aussi et surtout l'onglet **PERFORMANCE** 🚗

C'est là-dedans qu'on va s'amuser un peu.



Si votre application bugue, met du temps à charger ou fait des trucs étranges de temps en temps, c'est ici que vous allez trouver une réponse.

Les 3 blocs du haut sont des indicateurs de performance. Dans l'ordre :

- Le temps total qui s'est écoulé pour exécuter le chargement de la page.
- Le temps total qui s'est écoulé pour lancer l'ensemble des librairies Symfony de votre projet.
- Le pic de mémoire utilisé pendant le chargement.

Pour ces 3 blocs, votre objectif est de toujours chercher à avoir ces indicateurs au plus bas possible.

Et c'est grâce au super graphique juste en dessous que vous allez y parvenir ! Ce qu'il vous montre, c'est le temps d'exécution de chaque composant de votre page avec leurs dépendances (vos scripts, vos events, vos chargements de twig, vos requêtes aux BDD etc.).

🔍 Le controller du screen à pris 7.7 millisecondes à se charger.

i "Et le Threshold la ça sert à quoi ?"

Je ne peux que vous proposer d'essayer de le mettre à 0 pour comprendre par vous-mêmes ! 😊

Le fameux var_dump()

Vous vous souvenez de `var_dump()` ? Cette merveilleuse ligne de code vous permettant d'afficher ce que vous voulez ?

Symfony vous a réservé une petite surprise avec ce debug-pack ! Essayez donc `dd($variableQueVousVoulez)` dans votre code ! (je vous laisse deviner où il faut l'écrire pour que ça fonctionne bien sûr).

i dd() pour var_dump & die !

L'avantage que vous offre ce dd, c'est que vous pouvez aussi afficher des objets complexes et toutes leurs propriétés ! Essayez le sur à peu près tout et n'importe quoi !

```
*@appController.php on line 43)
App\Controller\testController (@178 ▶
  controller: App\Controller\testController (@178)
    container: Container_App_KernelDevDebugContainer (@76 ->)
      factories: array(4)
        0 => "services"
        1 => "privates"
        2 => "getHttpKernelService"
        3 => false
      parameter_bag => array(4) [▼
        0 => "parameters"
        1 => "parameter_bag"
        2 => "getParameterBagService"
        3 => false
      ]
      request_stack => array(4) [▼
        0 => "request"
        1 => "request_stack"
        2 => "getRequestStackService"
        3 => false
      ]
      router => array(4) [▼
        0 => "services"
        1 => "router"
        2 => "getRouterService"
        3 => false
      ]
      twig => array(4) [▼
        0 => "privates"
        1 => "twig"
        2 => "getTwigService"
        3 => false
      ]
    ]
  loading: []
  factory: Container_App_KernelDevDebugContainer::getService(string $registry, string $id, string $method, string $bool $load): mixed (@18 ▶
    returnType: "mixed"
    class: Container_App_KernelDevDebugContainer (@76 ->)
  )
  class: Container_App_KernelDevDebugContainer (@76 ->)
)▶
serviceMap: array(5) [▼
  0 => "services"
  1 => "privates"
  2 => "getHttpKernelService"
  3 => false
  4 => "parameter_bag"
]▶
parameter_bag => array(4) [▼
  0 => "parameters"
  1 => "parameter_bag"
  2 => "getParameterBagService"
  3 => false
]
request_stack => array(4) [▼
  0 => "request"
  1 => "request_stack"
  2 => "getRequestStackService"
  3 => false
]
router => array(4) [▼
  0 => "services"
  1 => "router"
  2 => "getRouterService"
  3 => false
]
twig => array(4) [▼
  0 => "privates"
  1 => "twig"
  2 => "getTwigService"
  3 => false
]
]▶
servicetypes: array(1) [▼
  "http_kernel" => 3
]
```

Un simple `dd($this)` peut donner tellement d'informations.

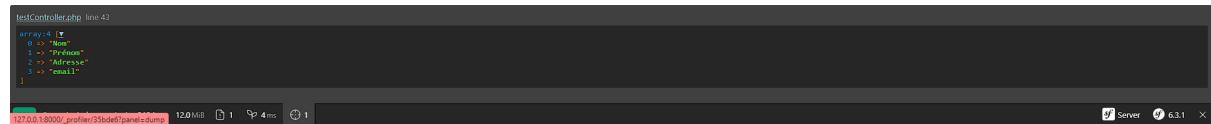
Il existe encore mieux : `dump()`

L'avantage avec cette méthode, c'est que le script ne sera pas arrêté pour autant !

Ce qui vous permet de surveiller plusieurs informations en même temps que votre page tourne.

Vous trouverez ces infos dans l'onglet en bas qui est apparu à l'occasion :

Contact : Nom Prénom Adresse email



A screenshot of the Symfony Profiler interface. The main area shows a dump() output from a variable named \$form. The output is an array with four elements: 'Nom', 'Prénom', 'Adresse', and 'email'. Below the dump output, there's a status bar showing '127.0.0.1:8000/_profiler/259de67panel?dump' and performance metrics: 12.0 MiB, 1 request, 4 ms average. At the bottom right, it says 'Server 6.3.1'.

```
testController.php line 43
array:4 [▼
  0 => "Nom"
  1 => "Prénom"
  2 => "Adresse"
  3 => "email"
]
127.0.0.1:8000/_profiler/259de67panel?dump 12.0 MiB 1 4 ms Server 6.3.1
```

i “Et si je veux faire ça dans mon Twig ?”

Bah, tu peux aussi ! 😊 : {{ dump() }}

Laisse-le vide, sans préciser d'arguments pour afficher toutes les variables accessibles.

Au passage, comme le Twig est une vue, le dump se fera aussi dans la vue.

Le Profiler permet de faire encore pleins d'autres choses très techniques et peut rendre la vie d'un développeur tel que vous bien plus facile. Avant d'en arriver là, il faut apprendre beaucoup de choses. C'est pour cette raison que je souhaite d'abord vous inviter à faire vos propres recherches sur le système de Garbage Collector sans vous donner aucun lien.

Lorsque vous avez le moindre soucis sur votre site, prenez le réflexe d'ouvrir votre profiler, puis d'utiliser les dump() et les dd() à la pelle pour connaître le cheminement de votre code.

On reverra le profiler un peu plus tard de notre côté, pour apprendre encore plus de choses.

Si vous tenez à aller voir dès maintenant ce dont il est capable, voici [un petit lien](#).