4 minutes

L'objet Response du composant HttpFoundation

L'objet `Response`

L'objet `Response` permet de créer, modifier et envoyer une réponse `HTTP`.

Créer une `Response`

Étudions son constructeur :

```
<?php

public function __construct(?string $content = '', int $status = 200, array $headers = [])
{
  $this->headers = new ResponseHeaderBag($headers);
  $this->setContent($content);
  $this->setStatusCode($status);
  $this->setProtocolVersion('1.0');
}
```

Copier

Remarquez que tous les arguments sont optionnels et qu'ils ont des valeurs par défaut.

Le premier argument est le contenu de la réponse `HTTP`, qui est vide par défaut.

Le second argument est le statut de la réponse `HTTP`, qui `200` pour `OK` par défaut.

Le troisième argument est un tableau associatif pour les en-têtes, qui est vide par défaut.

Pour créer une réponse vous pouvez donc par exemple faire :

```
<?php

$response = new Response(
  'Vous n\’êtes pas autorisé à voir cette page',
  Response::HTTP_FORBIDDEN,
  ['content-type' => 'text/html']
);
```

Copier

Remarquez que nous utilisons une constante pour le statut, tous les codes de statut sont disponibles sur la classe `Response` :

```
<?php
  public const HTTP_CONTINUE = 100;
  public const HTTP_SWITCHING_PROTOCOLS = 101;
  public const HTTP_PROCESSING = 102;            // RFC2518
  public const HTTP_EARLY_HINTS = 103;           // RFC8297
  public const HTTP_OK = 200;
  public const HTTP_CREATED = 201;
  public const HTTP_ACCEPTED = 202;
  public const HTTP_NON_AUTHORITATIVE_INFORMATION = 203;
  public const HTTP_NO_CONTENT = 204;
  public const HTTP_RESET_CONTENT = 205;
  public const HTTP_PARTIAL_CONTENT = 206;
  public const HTTP_MULTI_STATUS = 207;          // RFC4918
  public const HTTP_ALREADY_REPORTED = 208;      // RFC5842
  public const HTTP_IM_USED = 226;               // RFC3229
  public const HTTP_MULTIPLE_CHOICES = 300;
  public const HTTP_MOVED_PERMANENTLY = 301;
  public const HTTP_FOUND = 302;
  public const HTTP_SEE_OTHER = 303;
  public const HTTP_NOT_MODIFIED = 304;
  public const HTTP_USE_PROXY = 305;
  public const HTTP_RESERVED = 306;
  public const HTTP_TEMPORARY_REDIRECT = 307;
  public const HTTP_PERMANENTLY_REDIRECT = 308;  // RFC7238
  public const HTTP_BAD_REQUEST = 400;
```

```
    public const HTTP_UNAUTHORIZED = 401;
    public const HTTP_PAYMENT_REQUIRED = 402;
    public const HTTP_FORBIDDEN = 403;
    public const HTTP_NOT_FOUND = 404;
    public const HTTP_METHOD_NOT_ALLOWED = 405;
    public const HTTP_NOT_ACCEPTABLE = 406;
    public const HTTP_PROXY_AUTHENTICATION_REQUIRED = 407;
    public const HTTP_REQUEST_TIMEOUT = 408;
    public const HTTP_CONFLICT = 409;
    public const HTTP_GONE = 410;
    public const HTTP_LENGTH_REQUIRED = 411;
    public const HTTP_PRECONDITION_FAILED = 412;
    public const HTTP_REQUEST_ENTITY_TOO_LARGE = 413;
    public const HTTP_REQUEST_URI_TOO_LONG = 414;
    public const HTTP_UNSUPPORTED_MEDIA_TYPE = 415;
    public const HTTP_REQUESTED_RANGE_NOT_SATISFIABLE = 416;
    public const HTTP_EXPECTATION_FAILED = 417;
    public const HTTP_I_AM_A_TEAPOT = 418;                                    // RFC2324
    public const HTTP_MISDIRECTED_REQUEST = 421;                              // RFC7540
    public const HTTP_UNPROCESSABLE_ENTITY = 422;                             // RFC4918
    public const HTTP_LOCKED = 423;                                           // RFC4918
    public const HTTP_FAILED_DEPENDENCY = 424;                                // RFC4918
    public const HTTP_TOO_EARLY = 425;                                        // RFC-ietf-httpbis-replay-04
    public const HTTP_UPGRADE_REQUIRED = 426;                                 // RFC2817
    public const HTTP_PRECONDITION_REQUIRED = 428;                            // RFC6585
    public const HTTP_TOO_MANY_REQUESTS = 429;                                // RFC6585
    public const HTTP_REQUEST_HEADER_FIELDS_TOO_LARGE = 431;                  // RFC6585
    public const HTTP_UNAVAILABLE_FOR_LEGAL_REASONS = 451;
    public const HTTP_INTERNAL_SERVER_ERROR = 500;
    public const HTTP_NOT_IMPLEMENTED = 501;
    public const HTTP_BAD_GATEWAY = 502;
    public const HTTP_SERVICE_UNAVAILABLE = 503;
    public const HTTP_GATEWAY_TIMEOUT = 504;
    public const HTTP_VERSION_NOT_SUPPORTED = 505;
    public const HTTP_VARIANT_ALSO_NEGOTIATES_EXPERIMENTAL = 506;             // RFC2295
    public const HTTP_INSUFFICIENT_STORAGE = 507;                             // RFC4918
    public const HTTP_LOOP_DETECTED = 508;                                    // RFC5842
    public const HTTP_NOT_EXTENDED = 510;                                     // RFC2774
    public const HTTP_NETWORK_AUTHENTICATION_REQUIRED = 511;
```

Copier

Cela permet une meilleure lisibilité du code et de ne pas se tromper sur le statut.

Modifier une `Response`

La classe possède également des propriétés publiques qui sont modifiables avec des `setters`.

Il est ainsi possible de modifier un objet `Response` une fois que celui-ci a été créé.

Pour rappel, un `setter` est une méthode permettant de modifier une propriété sur un objet.

Par exemple, pour modifier le contenu de la réponse :

```
<?php
$response->setContent('Hello World !');
```

Copier

Même chose pour les en-têtes :

```
<?php
$response->headers->set('Content-Type', 'text/plain');
```

Copier

Et pour le code de statut :

```
<?php
$response->setStatusCode(Response::HTTP_NOT_FOUND);
```

Copier

Envoyer la `Response`

Une fois que l'objet `Response` est prêt à être envoyé au client, il suffit d'appeler la méthode **`send()`** :

```
<?php
$response->send();
```

Copier

Les réponses au format `JSON`

Bien qu'il soit possible d'utiliser l'objet `Response` pour envoyer une réponse au format `JSON`, ce n'est pas pratique car il faut manuellement définir l'entête `Content-Type` et encoder l'entité `PHP` au format `JSON`.

Le composant `HttpFoundation` a donc une classe spécifique qui permet de faire cela automatiquement pour vous : **`JsonResponse`**.

Vous pouvez ainsi directement faire :

```
use Symfony\Component\HttpFoundation\JsonResponse;

$response = new JsonResponse(['prenom' => 'Jean' ]);
```

Copier

Code de la vidéo : dossier `php`

Dans le fichier `index.php` mettez le code suivant :

```
<?php

use Symfony\Component\HttpFoundation\Response;

require __DIR__ . '/vendor/autoload.php';

$response = new Response('<h2>First !</h2><h1>Hello World ! </h1>');

$response->headers->set('salut', 'ca va');

$response->send();
```

Copier

Lancez ensuite le serveur de développement :

```
php -S localhost:3000
```

Copier

Rendez-vous sur `http://localhost:3000/`.

Code de la vidéo : dossier `dymaproject`

De retour sur le projet `Symfony`, nous modifions notre contrôleur pour récupérer l'objet `Request` créé par `Symfony` dans l'`action index` :

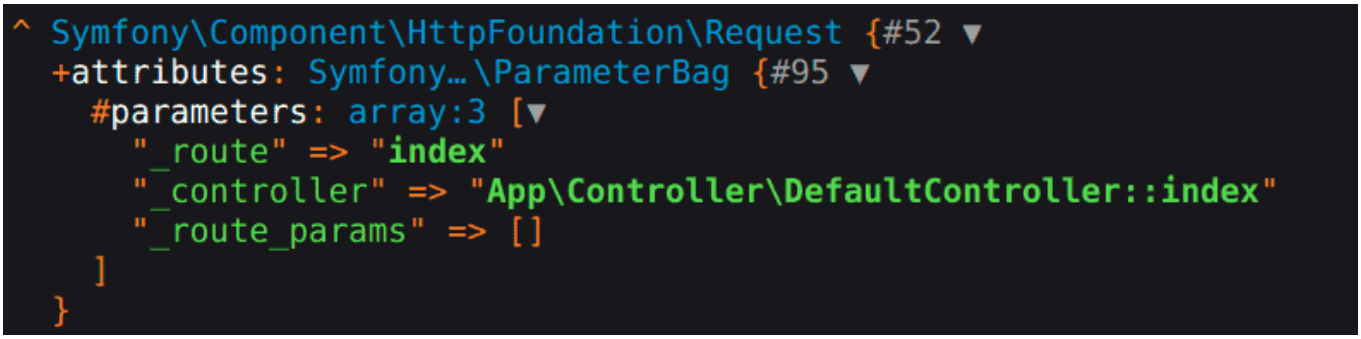
```
<?php

namespace App\Controller;

use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;

class DefaultController
{
  public function index(Request $request)
  {
    dd($request);
    return new Response('<h1>Hello World !</h1>');
  }
}
```

Copier

`Symfony` passe automatiquement l'objet `Request`, qu'il créée automatiquement, en argument des `actions`.

Pour rappel, les `actions` sont les méthodes sur les classes des contrôleurs qui sont exécutées lorsqu'une requête `HTTP` correspond à une route.

Notez bien la propriété `attributes` sur l'objet requête :



Comme nous l'avons vu, il s'agit d'une instance de la classe `ParamaterBag` qui contient des paires clé / valeur créées par l'application.

Dans ces paires nous retrouvons, comme prévu, le nom de la route dans `_route` et le chemin vers le contrôleur, et plus précisément son action, à exécuter dans `_controler`.