Slim avancé (I)

Container de dépendances et gestion des erreurs dans Slim

rappel

- Slim utilise un ***conteneur de dépendances*** pour stocker sa configuration :
  - variables diverses, services

```
$config = [ 'settings' =>
                    ['production' => true ] ];

$c = new \Slim\Container($config);
$app = new \Slim\App($c);


// utiliser le conteneur :
$app->get('/hello/{name}',function ($rq,$rs, $args) {
        ...
        $p = $this[ 'settings' ]['production'] ;
    }
);
```

- on peut y stocker toutes nos variables de configuration :

```
$conf = [
    'settings' => [
          'production' => true ,
          'db' => ['driver'=>'mysql,
                   'host'=>'localhost,
                   'database'=> 'lbs']];

$app = new \Slim\App(new \Slim\Container($conf));

$app->get('/hello/{name}',function ($rq,$rs, $args) {
          ...
          $dbconf = $this[ 'settings' ]['db'] ;
          $dsn= $dbconf['driver'].$dbconf['host'];

      }
);
```

- **la valeur d'une entrée dans le conteneur peut être une fonction qui est exécutée lors de l'accès à cette entrée**

```
$app = new \Slim\App(['settings' => [ … ] ]) ;

$c = $app->getContainer();
$c['db'] = function( $c ) {
            return parse_ini_file('src/conf/db.ini');
          };

$app->get('/hello/{name}',function ($rq,$rs, $args) {

      /* la fonction est exécutée ici : */
       $dbconf = $this['db'] ;
       $dsn= $dbconf['driver'].$dbconf['host'];

    }
);
```

- **utilisation** : pour enregistrer un service (1 fonctionnalité exécutable) dans le conteneur, enregistrer une fonction qui ***fabrique*** ce service et renvoie un objet :

```
$conf = [
  'logger' = function( $c ) {
      return new \totolog\Logger() ;
}] ;

$app = new \Slim\App(new \Slim\Container($conf));

$app->get('/hello/{name}',function ($rq,$rs, $args) {

        $logger = $this->logger ; // $this['logger'] ;
        $logger->log('message de log') ;
      }
);
```

- le service retourné par le conteneur peut-être une fonction :

```
$conf = [
 'formater' = function( $c ) {
     return function($rs, string $m) {
        return "Log.app: $m " . date('d-m-Y') ;
      };
}] ;


$c = new \Slim\Container($conf);
$app = new \Slim\App($c);

$app->get('/hello/{name}',function ($rq,$rs, $args) {

       $f = $this->formatter; // $this['formatter'] ;
       print $f('message de log') ;
    }
);
```

- **Configurer une application** : enregistrer les services dont elle dépend dans le conteneur de dépendances
- la création d'un service peut se baser sur un autre service enregistré dans le conteneur :

```php
$container = $app->getContainer() ;

$container['logger'] = function( $c ) {
      $logger= new \totolog\Logger() ;
      $logger->setFormatter( $c['formatter] ) ;
      return $logger ;
};

$container['formatter'] = function( $c ) {
      return function($rs, string $m) {
         return "Log.app: $m " . date('d-m-Y') ;
       };
};
```

la gestion d'erreurs en slim

- les erreurs slim sont prises en charge par des gestionnaires d'erreur,
- les gestionnaires d'erreurs sont des **services** enregistrés dans le **conteneur de dépendances**
- un **handler** d'erreur est une fonction retournée par le conteneur de dépendances et appelée pour traiter un cas d'erreur
- bonne nouvelle : **on peut donc les modifier** !

la gestion des erreurs slim

- le framework gère certaines erreurs :
  - **route non trouvée : Not Found** – correspond au cas la requête courante ne correspond à aucune route déclarée - **`notFoundHandler`**
  - **méthode non permise : Not Allowed** – correspond au cas où l'url de la requête correspond à une route existante mais pas la méthode HTTP - **`notAllowedHandler`**
  - **erreur exception PHP 7 : PHP Runtime Error –** erreur d'exécution en PHP 7 - **`phpErrorHandler`**
  - **autres erreurs : Default Error** – cas d'erreur par défaut, appelé en cas d'erreur d'exécution - **`errorHandler`**

programmer ses handler d'erreurs

- erreur NotFound : aucune route
- le handler est une fonction qui reçoit 1 requête et 1 réponse en paramètre
- retourne 1 réponse

```
$c = $app->getContainer() ;
$c[ 'notFoundHandler' ]= function( $c ) {

  return function( $req, $resp ) {

    $resp= $resp->withStatus( 400 ) ;
    $resp->getBody()->write( 'URI non traitée' ) ;
    return $resp ;
    } ;
```

- méthode non permise : notAllowedHandler
- le handler est une fonction recevant la requête, 1 réponse et un tableau de méthodes autorisées
- retourne 1 réponse

```
$c[ 'notAllowedHandler' ]= function( $c ) {

  return function( $req, $resp , $methods ) {

    $resp= $resp
      ->withStatus( 405 )
      ->withHeader('Allow', implode(',', $methods) );
    $resp->getBody()
       ->write( 'méthode permises :' .
                          implode(',', $methods) ) ;
    return $resp ;
    };
```

- erreurs PHP : phpErrorHandler
- le handler est 1 fonction qui reçoit la requête, la réponse et l'exception à l'origine de l'erreur
- retourne 1 réponse
- Ne pas l'utiliser en mode développement

```
$conf = [ 'phpErrorHandler' => function( $c ) {

   return function( $req, $resp , $e ) {

     $resp= $resp->withStatus( 500 ) ;
      $resp->getBody()
        ->write( 'error :' . $e->getMessage() )
        ->write( 'file : ' . $e->getFile() )
        ->write( 'line : ' . $e->getLine() ) ;
     return $resp ;
     }
] ;
```

les gestionnaires par défaut

- gestionnaires par défaut :
  - notFoundHandler => 404 + text/html
  - notAllowedHandler => 405 + text/html
  - errorHandler => 500 + text/html
    - settings => ['displayErrorDetail'=>true] affiche un message d'erreur étendu, à utiliser en dev

squelette de l'application

```php
<?php
require_once  __DIR__ . '/../src/vendor/autoload.php';

use \Psr\Http\Message\ServerRequestInterface as Request ;
use \Psr\Http\Message\ResponseInterface as Response ;
use \lbs\common\bootstrap\Eloquent ;
$settings = require_once __DIR__ . '/.../settings.php';
$errors   = require_once __DIR__ . '/.../errors.php';
$dependencies= require_once __DIR__ . '/.../deps.php';

$app_config = array_merge($settings, $errors, $dependencies);
$app = new \Slim\App( new \Slim\Container($app_config) );

Eloquent::start(($app->getContainer())->settings['dbconf']);

$app->get( '/categories[/]', function($rq, $rs, $args) {..}) ;

/* si les routes sont nombreuses :          */
require_once __DIR__ . '/../src/api/routes.php';

$app->run() ;
```