

CACTUSCREW DEVELOPPEMENT WEB

Classe de gestion des erreurs

Je dépose une ici mon dernier fignoloage en matière de gestion des erreurs.

Présentation

J'ai developpé cette classe au fur et à mesure de mes différents developpement, et je l'ai doucement améliorée.

Elle permet de gérer des modes de comportement tel que DEV / PROD, et, ma dernière trouvaille, gère automatiquement le rendu de sortie TEXTE / HTML. Pratique si on désire travailler en console.

Elle est basée sur le design pattern singleton, car il est indispensable de centraliser la gestion de toutes les erreurs en un seul point, donc une seule instance de cette classe est autorisée.

Cette classe peut bien sûr être grandement améliorée, au niveau de la gestion des erreurs en prod par exemple.

Utilisation

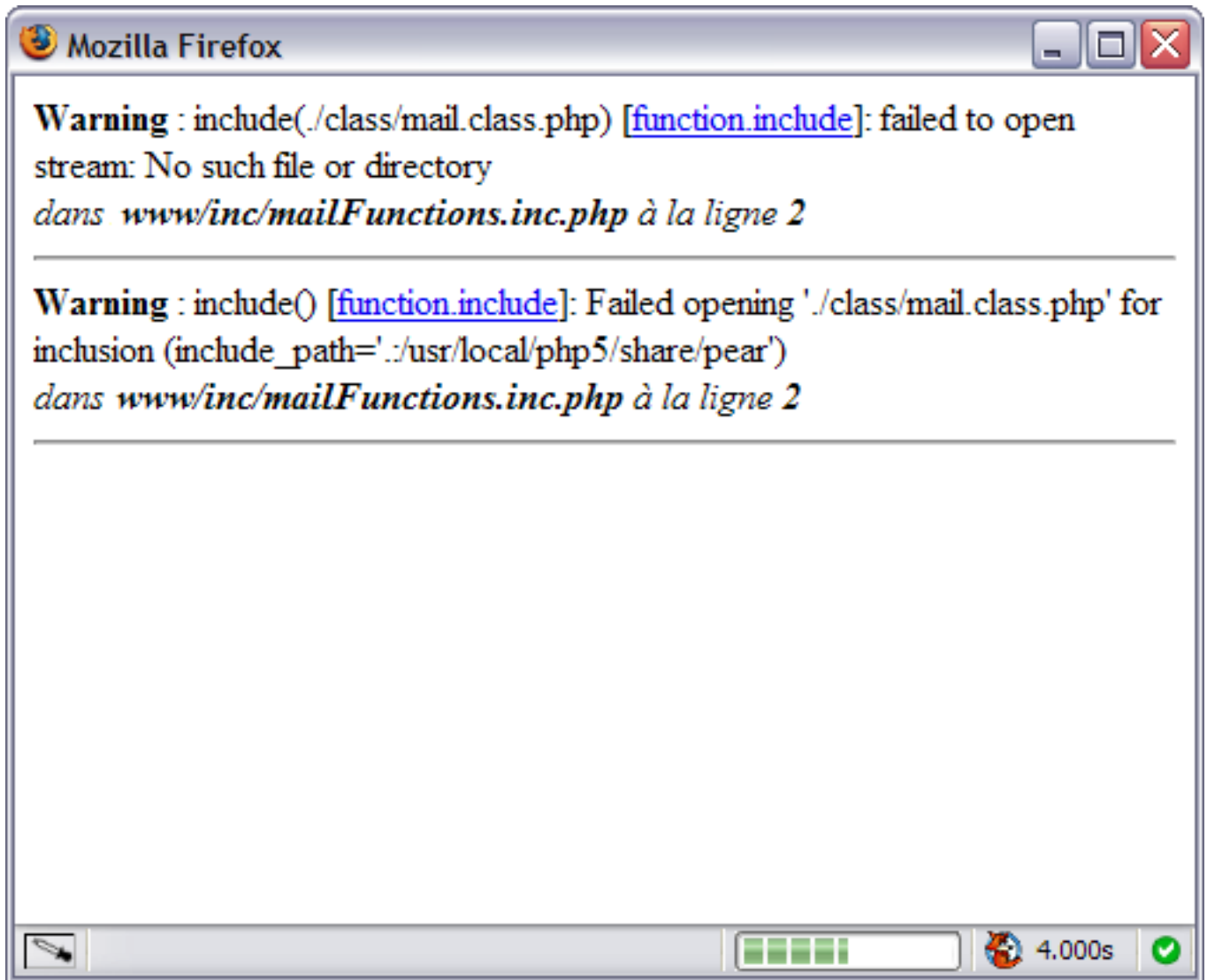
```
// récupérer l'instance
$_handler = ErrorHandler::getInstance();

// Test sur l'ip, permettant de basculer du mode DEV à PROD
if ($_SERVER['REMOTE_ADDR'] != 'xxx.xxx.xxx.xxx') {
    $_handler->setMode(ErrorHandler::MODE_PROD);
} else {
    $_handler->setMode(ErrorHandler::MODE_DEV);
}
```

Et c'est tout. Les messages d'erreurs sont "catchés" par la classe. Puis lors de la fin de l'exécution du script, le traitement s'effectue dans le destructeur de la classe. Très simple, somme toute.

Affichage des erreurs en production

La popup :



Le code source :

```
<?php
/**
 * CLASSE DE GESTION DES ERREURS
 *
 */
class ErrorHandler {

/**
 * mode production (pas d'affichage d'erreur)
 *
 */
const MODE_PROD = 'prod';

/**
 * mode developpement (affichage en popup)
 *
 */
const MODE_DEV = 'dev';
```

```

/**
 * type de sortie html
 *
 */
const OUTPUT_HTML = 'html';

/**
 * type de sortie texte
 *
 */
const OUTPUT_TXT = 'txt';

/**
 * instance du singleton de la classe de gestion d'erreur
 *
 * @var ErrorHandler
 */
private static $O_errorHandler;

/**
 * liste des messages d'erreur
 *
 * @var array
 */
private $A_msg = array();

/**
 * mode de gestion (dev / prod)
 *
 * @var string
 */
private $S_mode = "";

/**
 * gestion de la sortie (HTML / TEXTE)
 *
 * @var string
 */
private $S_output = "";

/**
 * Constructeur
 *
 */
private function __construct() {
// initialisation de l'error handler
set_error_handler(array($this, 'catchError'));

// detecte le mode de sortie du texte selon que le soit en Web, ou en console (HTML / TXT)
if (isset($_SERVER['HTTP_HOST'])) {
$this->S_output = self::OUTPUT_HTML ;
}
}

```

```

} else {
    $this->S_output = self::OUTPUT_TXT ;

}

}

/**
 * retourne l'instance de la gestion d'erreur
 * Singleton
 *
 * @return ErrorHandler
 */
public static function getInstance() {
    // si on a pas d'instance existante, on la cr e
    if (!isset(self::$O_errorHandler)) {
        self::$O_errorHandler = new self();
    }
    return self::$O_errorHandler;
}

/**
 * R cupere les erreurs
 *
 * @param int $errno num ro d'erreur
 * @param string $errstr message d'erreur
 * @param string $errfile fichier
 * @param int $errline ligne
 */
public function catchError($errno, $errstr, $errfile, $errline) {
    $S_msg = "";
    // selon le num ro d'erreur
    switch ($errno) {
        case E_ERROR:
            $S_msg .= '<strong style="color:#FF0000">Error </strong>';
            break;
        case E_USER_ERROR:
            $S_msg .= '<strong style="color:#FF0000">User Error </strong>';
            break;

        case E_WARNING:
            $S_msg .= "<strong>Warning </strong>";
            break;
        case E_USER_WARNING:
            $S_msg .= "<strong>User Warning </strong>";
            break;

        case E_NOTICE:
            $S_msg .= "<em>notice </em>";
            break;
    }
}

```

```

case E_USER_NOTICE:
    $S_msg .= "<em>User notice </em>";
    break;

case E_STRICT:
    $S_msg .= "<em>Strict error</em>";
    break;
default:
    $S_msg .= "<strong>Erreur n° $errno</strong>";

}
$S_msg .= " : {$errstr}<br/><em>dans <strong>{$errfile}</strong> Ã la ligne
<strong>{$errline}</strong></em><br/>";

// on ajoute ce message d'erreur dans la liste des messages
$this->A_msg[] = $S_msg;

}

/**
 * applique le mode (DEV ou PROD)
 *
 * @param string $S_mode
 */
public function setMode($S_mode) {
    $this->S_mode = $S_mode;
}

/**
 * Retourne le mode actuel
 *
 * @return unknown
 */
public function getMode() {
    return $this->S_mode;
}

/**
 * Formate les messages pour le mode HTML
 *
 * @return string
 */
private function _outHtml() {
    $S_msg = stripslashes(implode('<hr/>', $this->A_msg)).'<hr/>';
    return <<< JS
<script type="text/javascript">
    popup = open("", 'debug',
'width=500,toolbar=0,location=0,directories=0,status=1,menubar=0,scrollbars=1,resizable=1,copyhis
tory=0');
    popup.document.write("{ $S_msg }");
</script>

```

```

JS;

}

/**
 * formate les messages pour le mode TXT
 *
 * @return unknown
 */
private function _outTxt() {
    return strip_tags(str_replace('<br/>', "\n", stripslashes(implode("\n", $this->A_msg))));
}

/**
 * destructeur (appelé automatiquement lors de la fin de l'exécution du script)
 *
 */
public function __destruct() {
    // si on a enregistré des erreurs
    if (sizeof($this->A_msg) > 0) {
        // selon le mode
        switch ($this->getMode()) {
            // DEVELOPPEMENT
            case self::MODE_DEV :
                // selon les mode de sortie
                switch ($this->S_output) {
                    // HTML
                    case self::OUTPUT_HTML :
                        $S_out = $this->_outHtml();
                        break;
                    // PAR DEFALT
                    default:
                        $S_out = $this->_outTxt();
                        break;
                }

                echo $S_out;
                break;

            default:
                /** TODO : enregistre les erreur, envoi de mail.. ? */
                break;
        }
    }
}
}
}
}

```

