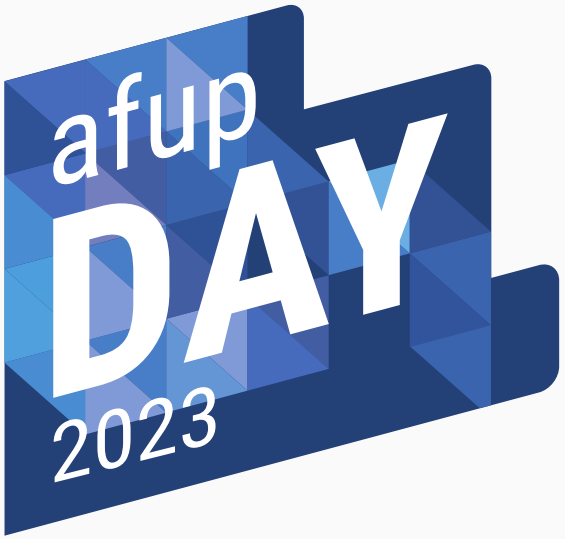
La gestion d’erreur dans toutes ses couleurs

George Peter Banyard

12 Mai 2023

The PHP Foundation

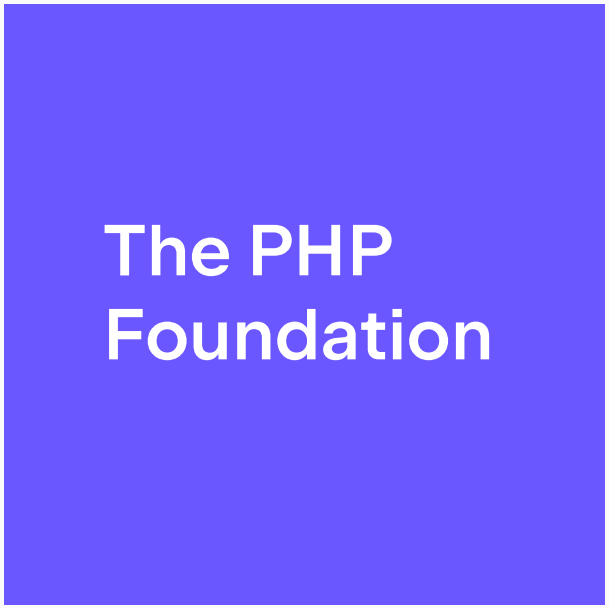


À propos de moi

Mastodon: @Girgias@phpc.social

GitHub: Girgias

Site: *https://gpb.moe*



- A étudié les maths pures
- Core dev de PHP financé à temps partiel par The PHP Foundation
- Mainteneur principal de la documentation FR de PHP
- Se concerne trop des sémantiques de PHP [9] [8] [7] [5] [3]
- Aime les systèmes de type [6] [4] [2] [1]

Idéalement

```hack
function fait_quelquechose(
    A $a,
    B $b,
    C $c,
    /* ... */
): T {
    /* ... */
    return $truc_utile;
}

$truc_utile = fait_quelquechose($a, $b, $c);
/* Faire des choses avec $truc_utile */
```

Signalisation des erreurs via un code d’erreur

Via une variable globale

```c
int main(void) {
    errno = 0;
    printf("log(-1.0) = %f\n", log(-1.0));
    if (errno)
        printf("%s\n",strerror(errno));
    errno = 0;
    printf("log(1.0)  = %f\n", log(1.0));
    if (errno)
        printf("%s\n",strerror(errno));
}
```

```
log(-1.0) = nan
Numerical argument out of domain
log(1.0)  = 0.0000
```

Via une variable globale

```c
int main(void) {
    errno = 0;
    printf("log(-1.0) = %f\n", log(-1.0));
    if (errno)
        printf("%s\n",strerror(errno));
    //errno = 0;
    printf("log(1.0)  = %f\n", log(1.0));
    if (errno)
        printf("%s\n",strerror(errno));
}
```

```
log(-1.0) = nan
Numerical argument out of domain
log(1.0)  = 0.0000
Numerical argument out of domain
```

Via une variable globale

```
switch (errno) {
    case EDOM:
        // domain error
        break;
    case ERANGE:
        // pole or range error
        break;
    case 0:
        // no error
        break;
}
```

Via la valeur de retour

```php
// mb_parse_str(string $string, array &$result): bool
$status = mb_parse_str($url_encoded_string, $data);
if ($status === false) {
    /* Error */
}
/* Faire qqch avec $data */
```

Via une valeur de retour du même type

```c
int main(void) {
    // Signature : time(time_t *arg): time_t
    time_t result = time(NULL);
    if (result != (time_t)(-1)) {
        printf(
            "The current time is %s\n",
            asctime(gmtime(&result))
        );
    }
}
```

Via une valeur de retour de type différent

```php
// json_encode(mixed $value, /* ... */): string|false
$value = json_encode($i);
if ($value === false) {
    match (json_last_error()) {
        JSON_ERROR_DEPTH => handle_depth_err(),
        JSON_ERROR_UTF8 => handle_invalid_utf8(),
        /* ... */
    };
}
/* Faire qqch avec $value */
```

Via une valeur de retour de type différent

```php
// json_encode(mixed $value, /* ... */): string|false
$value = json_encode($i);
match (json_last_error()) {
    JSON_ERROR_NONE => faire_qqch($value),
    JSON_ERROR_DEPTH => handle_depth_err(),
    JSON_ERROR_UTF8 => handle_invalid_utf8(),
    /* ... */
};
```

Via un out-param

```php
function foo(Input $i, int &$err): Output|null

$value = foo($i);
if ($err) {
    /* Error */
}
assert($value instanceof Output);
/* Faire qqch avec $value */
```

Récap

- On sait que la fonction peut échouer
- La vérification se fait où la fonction est appelé
- Facile d'oublier de vérifier le statut
- Le code d'erreur doit être décodé
- Impossible de savoir si toutes les erreurs sont couvertes
- ”Happy Path” est obstrué par de la gestion d'erreur

Signalisation des erreurs via une exception

Retour sur log()

```
function log(float $v): float {
    if ($v <= 0.0) {
        throw new DomainError(
            "Value must be greater than 0"
        );
    }
    return actual_log($v);
}
```

Retour sur log()

```
try {
    log(-1.5);
} catch (\DomainError) {
    /* Gérer l'erreur */
}
```

Retour sur log()

```
log(-1.5);
/* Aucune indication que l'appel peut échouer
 * Ni si l'erreur est gérer */
```

Localité de la gestion d'erreur

```php
function foo(float $v): float {
    $v = log($v);
    /* ... */
    return $v;
}

try {
    foo(-1.5);
} catch (\DomainError) {
    /* Gérer l'erreur */
}
```

Localité de la gestion d'erreur

```
try {
    foo(-1.5);
    autre_fonction_qui_lance_DomainError($input);
} catch (\DomainError) {
    /* Sans savoir s'occupe de foo() */
}
```

Localité de la gestion d'erreur

```
try {
    $db = db_connect(/* ... */);
    $q  = db_prepare_query($db, /* ... */);
    db_execute_query($db, $q);
} catch (\DataBaseError) {
    /* S'occupe des trois fonctions */
}
```

Localité de la gestion d'erreur

```
try {
    $db = db_connect(/* ... */);
    $q  = db_prepare_query($db, /* ... */);
    db_execute_query($db, $q);
} catch (\DataBaseConnectionError) {
    /* ... */
} catch (\DataBaseQueryError) {
    /* ... */
}
```

Les exceptions ça coûte cher !

À chaque exception lancée, la pile d'appel doit être générée.

```php
$retries = 3;
while (true) {
    try {
        $r = fonction_qui_echou_souvent(/* ... */);
        break;
    } catch (\Exception $e) {
        if (!$retries) throw $e;
        --$retries;
    }
}
```

Checked or Unchecked?

Checked :

```php
/** @throws DomainError */
function log(float $v): float { /* ... */ }
```

Unchecked :

```php
function log(float $v): float { /* ... */ }
```

Aparté

Hiérarchie des exceptions en PHP :

- Throwable
  - Error
  - Exception

Récap

- Erreurs descriptives sans décodage
- Possibilité de gérer plusieurs erreurs différentes directement
- Location approximative de la gestion d'erreur
- Des exceptions peuvent apparaitre de nulle part si elles ne sont pas checked
- Coûte cher à cause de la génération de la pile d'appel
- "Happy Path" clair

Signalisation des erreurs via des valeurs de retour multiple

Imitation Golang

```php
/* @return list{T, Throwable} */
function foo(Input $i) { /* ... */ }

[$value, $err] = foo($v);
if ($err) {
    /* Error */
}
/* Faire qqch avec $value */
```

Peut-on faire mieux ?

Objectifs

- Erreurs descriptives sans décodage
- Être assuré de gérer *tous* les cas d'erreurs
- Possibilité de gérer plusieurs erreurs différentes au même endroit
- Ne pas avoir le coût d'une exception
- "Happy Path" clair

Exemple : ouvrir un fichier

Exemple : ouvrir un fichier

Quelles erreurs ?

Exemple : ouvrir un fichier

Quelles erreurs ?

- Fichier inexistant
- Permission insuffisante
- Le fichier est un dossier

Exemple : ouvrir un fichier

```
enum OpenFileErrors {
    case FileDoesNotExist;
    case AccessDenied;
    case IsDirectory;
}

function open_file(string $path): File|OpenFileErrors
```

Exemple : ouvrir un fichier

```php
$data = open_file($path);
if ($data instanceof OpenFileErrors) {
    match ($data) {
        OpenFileErrors::FileDoesNotExist => /* ... */,
        OpenFileErrors::AccessDenied => /* ... */,
        OpenFileErrors::IsDirectory => /* ... */,
    };
}
/* $data est une instance de File */
```

Exemple : ouvrir un fichier

```
$data = open_file($path);
$result = match ($data::class) {
    OpenFileErrors::class => match ($data) {
        OpenFileErrors::FileDoesNotExist => /* ... */,
        OpenFileErrors::AccessDenied => /* ... */,
        OpenFileErrors::IsDirectory => /* ... */,
    },
    File::class => use_file($data),
};
```

Exemple : ouvrir un fichier

```php
$data = open_file($path);
$result = match ($data::class) {
    OpenFileErrors::class => match ($data) {
        OpenFileErrors::FileDoesNotExist => /* ... */,
        OpenFileErrors::AccessDenied => /* ... */,
        OpenFileErrors::IsDirectory => /* ... */,
    },
    File::class => match (($x = use_file($data))::class)
        UseErrors::class => match ($x) {
            /* ... */
        },
        Output::class => $x
    },
};
```

Objectifs

- Erreurs descriptives sans décodage ✓
- Être assuré de gérer *tous* les cas d'erreurs ✓
- Possibilité de gérer plusieurs erreurs différentes au même endroit
- Ne pas avoir le coût d'une exception ✓
- ”Happy Path” clair

En utilisant un type somme!

Type Somme bancal en PHP

```php
/** @template T1
 * @template E1 */
class Wrapper {
  /** @param T1 $value
   * @param E1 $err */
    public function __construct(
        public mixed $value,
        public mixed $err,
        public bool $isErr = false,
    ) {}
    /* ... */
}
```

Rendons le utile

```php
/**
 * @template T2
 * @template E2
 * @param callable(T1): Wrapper<T2, E2> $f
 * @return Wrapper<T2, E1|E2>
 */
public function bind($f): Wrapper {
    if ($this->isErr) {
        /** @var Wrapper<T2, E1> Shut up Psalm */
        return $this;
    }
    return $f($this->value);
}
```

Changeons nos définitions

```php
/** @return Wrapper<File, OpenFileErrors> */
function open_file(string $path): Wrapper {
    return new Wrapper(
        new File(),
        OpenFileErrors::FileDoesNotExist
    );
}

/** @return Wrapper<UnparsedOutput, GetContentErrors> */
function get_content_file(File $f): Wrapper {
    return new Wrapper(
        new UnparsedOutput(),
        GetContentErrors::E1
    );
}
```

Résultat

```php
$data = open_file('')
    ->bind(get_content_file(...));
$result = match ($data->isErr) {
    true => match ($data->err) {
        OpenFileErrors::FileDoesNotExist => /* ... */,
        OpenFileErrors::AccessDenied => /* ... */,
        OpenFileErrors::IsDirectory => /* ... */,
        GetContentErrors::E1 => /* ... */,
        GetContentErrors::E2 => /* ... */,
    },
    false => $data->value,
};
```

Bind à l'infinie

```php
$data = open_file('')
    ->bind(get_content_file(...))
    ->bind(parse_content(...));
$result = match ($data->isErr) {
    true => match ($data->err) {
        OpenFileErrors::FileDoesNotExist => /* ... */,
        OpenFileErrors::AccessDenied => /* ... */,
        OpenFileErrors::IsDirectory => /* ... */,
        GetContentErrors::E1 => /* ... */,
        GetContentErrors::E2 => /* ... */,
        ParseContentErrors::E1 => /* ... */,
        ParseContentErrors::E2 => /* ... */,
    },
    false => $data->value,
};
```

Tester en ligne !

Via le site de Psalm : *https://psalm.dev/r/dd8e8f59fe*

Objectifs

- Erreurs descriptives sans décodage ✓
- Être assuré de gérer *tous* les cas d'erreurs ✓
- Possibilité de gérer plusieurs erreurs différentes au même endroit ✓
- Ne pas avoir le coût d'une exception ✓
- ”Happy Path” clair (à peu près)

Mais ce Wrapper, c’est quoi au fait?

Unraveling the Wrapper

```php
/** @template T1
 * @template E1 */
class Wrapper {
  /** @param T1 $value
   * @param E1 $err */
    public function __construct(
        public mixed $value,
        public mixed $err,
        public bool $isErr = false,
    ) {}
    /* ... */
}
```

Unraveling the Wrapper

```php
/** @template T1
 * @template E1 */
class Either {
  /** @param T1 $left
   * @param E1 $right */
    public function __construct(
        public mixed $left,
        public mixed $right,
        public bool $isRight = false,
    ) {}
    /* ... */
}
```

Le contrat d'une monade

```php
/**
 * @template T1
 */
interface Monad {
    /** @property T1 $value */
    /**
     * @template T2
     * @param callable(T1): Monad<T2> $f
     * @return Monad<T2>
     */
    public function bind($f): Monad;
}
```

Monade Maybe

```php
/** @template T1 */
class Maybe implements Monad {
    /** @param T1 $value */
    public function __construct(
        public mixed $value,
        public bool $isErr = false,
    ) {}
    /* ... */
}
```

Monade Maybe

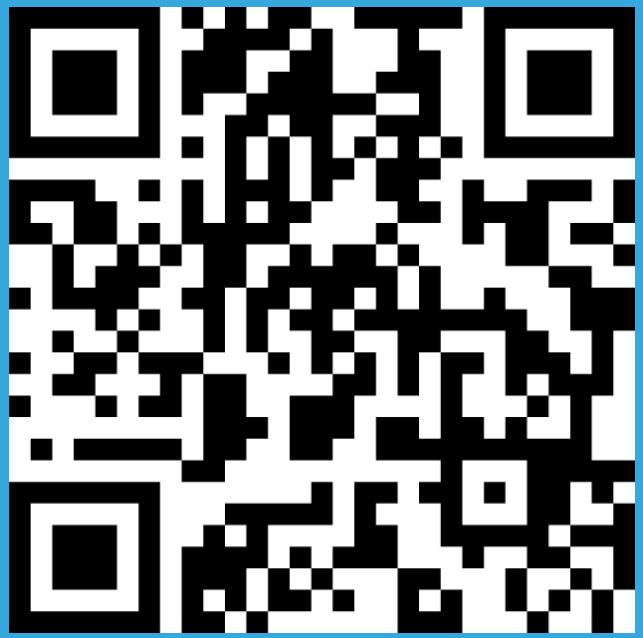
```php
/**
 * @template T2
 * @param callable(T1): Monad<T2> $f
 * @return Monad<T2>
 */
public function bind($f): Monad {
    if ($this->isErr) {
        /** @var Monad<T2> Shut up Psalm */
        return $this;
    }
    return $f($this->value);
}
```

Merci beaucoup !

Mastodon: @Girgias@phpc.social

GitHub: Girgias

Site: `https://gpb.moe`

Feedback: 

References

[1] George Peter Banyard. *Add true type*. English. Apr. 2022. URL: *https://wiki.php.net/rfc/true-type* (visited on 10/10/2022).

[2] George Peter Banyard. *Allow null and false as stand-alone types*. English. Feb. 2022. URL: *https://wiki.php.net/rfc/null-false-standalone-types* (visited on 10/10/2022).

[3] George Peter Banyard. *Define proper semantics for range() function*. Mar. 2023. URL: *https://wiki.php.net/rfc/proper-range-semantics* (visited on 05/11/2023).

Références ii

[4] George Peter Banyard. *Disjunctive Normal Form Types*. English. Apr. 2022. URL: https://wiki.php.net/rfc/dnf_types (visited on 10/10/2022).

[5] George Peter Banyard. *Path to Saner Increment/Decrement operators*. English. Nov. 2022. URL: https://wiki.php.net/rfc/saner-inc-dec-operators (visited on 02/15/2023).

[6] George Peter Banyard. *Pure intersection types*. English. Mar. 2021. URL: https://wiki.php.net/rfc/pure-intersection-types (visited on 10/10/2022).

[7] George Peter Banyard. *Saner array_(sum|product)()*. English. Jan. 2023. URL: https://wiki.php.net/rfc/saner-array-sum-product (visited on 02/15/2023).

Références iii

[8] George Peter Banyard. *Saner numeric strings*. English. June 2020. URL: *https://wiki.php.net/rfc/saner-numeric-strings* (visited on 10/10/2022).

[9] George Peter Banyard and Máté Kocsis. *Locale-independent float to string cast*. English. Mar. 2020. URL: *https://wiki.php.net/rfc/locale_independent_float_to_string* (visited on 02/19/2023).