# 言語間で比較するエラーの通知と処理

nsfisis (いまむら)

第 154 回 PHP 勉強会@東京

## 自己紹介



nsfisis (いまむら)

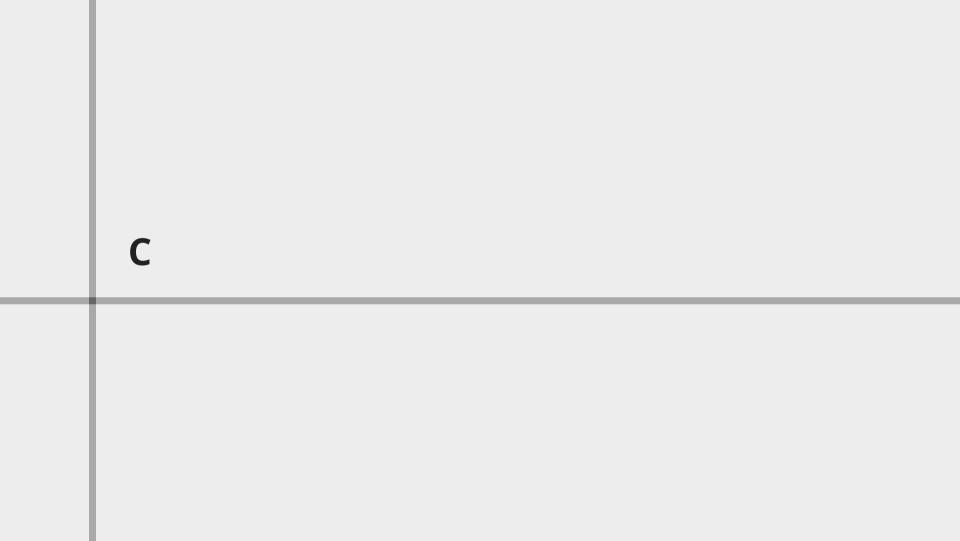
@ デジタルサーカス株式会社

## 話すこと、話さないこと

- 話すこと
  - 。様々な言語におけるエラー通知の方法
- 話さないこと
  - 。君たちはどう生きるか (PHP におけるベストプラクティス)

# 取り上げる言語

- (
- C++
- Java
- Go
- Rust



#### C: 成功 / 失敗を戻り値で表す

```
#include <stdbool.h>
bool do something() {
    // ...
    if (success) {
        return true;
    } else {
        return false;
```

PHPではmkdir()など

#### C: 成功 / 失敗を戻り値で表す

- Bad 処理結果があるときに使えない
- Bad エラーに情報を載せられない
- Bad エラーハンドリングを省略できる
  - 。Bad 省略したとき、後続の処理が一切止まらない

#### C: 引数で処理結果を受け取る

```
#include <stdbool.h>
bool do something(something* result) {
    if (success) {
        *result = ...:
        return true;
    } else {
        return false;
```

PHPではpreg\_match()など

#### C: 引数で処理結果を受け取る

- Good 処理結果があるときに使える
- Good エラーに情報を載せられる
- Bad エラーハンドリングを省略できる
  - 。Good 省略したとき、処理結果を使おうとすると止まる
  - 。Bad 省略したとき、後続の処理が進みうる

#### C: 処理結果または失敗を返す

```
something* do something() {
   // ...
   if (success) {
        return result;
    } else {
        return NULL;
```

PHPではfopen()など

#### C: 処理結果または失敗を返す

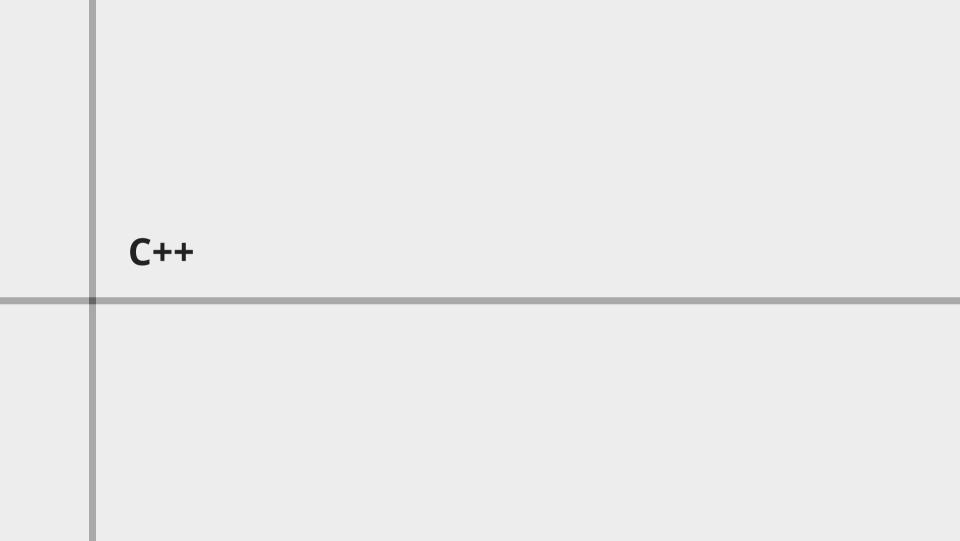
- Good 処理結果があるときに使える
- Bad エラーに情報を載せられない
  - 。注:動的型付き言語の場合はその限りでない
- Bad エラーハンドリングを省略できる
  - 。Good 省略したとき、処理結果を使おうとすると止まる
  - 。Bad 省略したとき、後続の処理が進みうる

## C: グローバル状態から通知する

```
some error error;
void do something() {
    // ...
   if (success) {
        error = 0;
    } else {
        error = 42;
some error get error() {
    return error;
```

## C: グローバル状態から通知する

- Good 処理結果があるときに使える
- Good エラーに情報を載せられる
- Bad エラーハンドリングを省略できる
  - 。Bad 省略したとき、後続の処理が一切止まらない



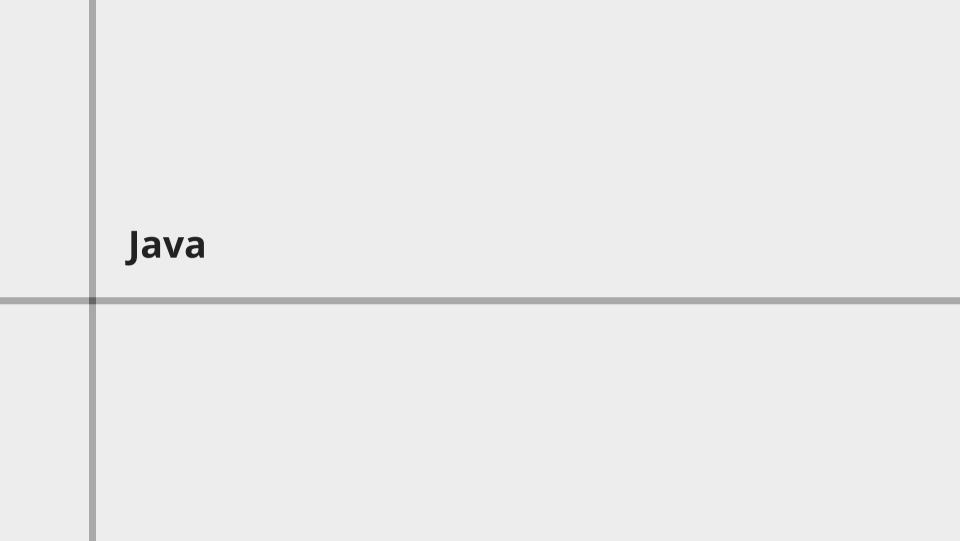
#### C++: 例外を投げる

```
#include <stdexcept>
void do something() {
    if (!success) {
        throw std::runtime error{"..."};
```

PHPではDateTimeImmutable:: construct()など

## C++: 例外を投げる

- Good 処理結果があるときに使える
- Good エラーに情報を載せられる
- Bad エラーハンドリングを省略できる
  - 。Good 省略したとき、そこで止まる



# Java: 検査例外

```
void doSomething() throws SomeException {
    // ...
    if (!success) {
        throw new SomeException(...);
    }
}
```

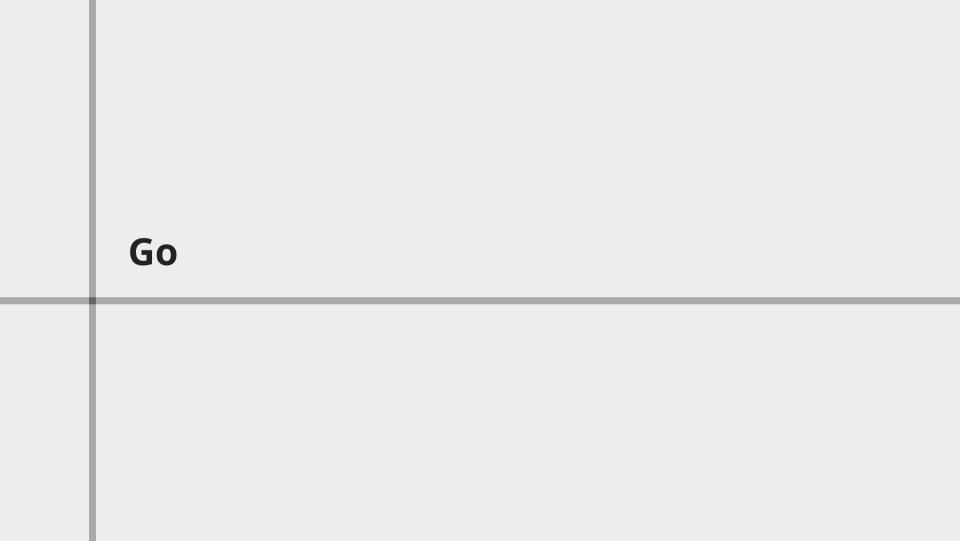
PHP では@throws が (一応) 近い

# Java: 検査例外

- Good 処理結果があるときに使える
- Good エラーに情報を載せられる
- Good エラーハンドリングを省略するとコンパイルエラーになる
  - 。Good 省略ができない

# Java: 検査例外

- Good 処理結果があるときに使える
- Good エラーに情報を載せられる
- Good エラーハンドリングを省略するとコンパイルエラーになるGood 省略ができない
- Bad 大域脱出は処理の動きが複雑になる
- Bad 例外クラスの継承ツリー設計が困難
- Bad 回復不能なエラーまで例外で表される



# Go: 多值返却

```
func doSomething() (int, error) {
    // . . .
    if success {
        return 42, nil
    } else {
        return 0, errors.New("...")
result, err := doSomething()
if err != nil {
   // . . .
```

# Go: 多值返却

```
<?php
function do something(): array {
    if ($success) {
        return [42, null];
    } else {
        return [null, new SomeError(...)];
[$result, $err] = do something();
if (isset($err)) {
    // ...
```

#### Go: 多值返却

- Good 処理結果があるときに使える
- Good エラーに情報を載せられる
- Bad エラーハンドリングを省略できる
  - 。Good linter での検知は可能

# Go: panic

```
func doSomething() int {
    // ...
    if !success {
        panic("...")
    }
    return 42
}
```

回復不能なエラー。(基本的には)捕まえられない

# Go: 多値返却 + panic

- Good 処理結果があるときに使える
- Good エラーに情報を載せられる
- Bad エラーハンドリングを省略できる
  - 。Good linter での検知は可能
- Good 特殊な構文なし、大域脱出なし
- Good エラー値のカテゴライズに継承が使われない
- Good 回復可能なエラーと回復不能なエラーを区別できる
- Bad 静的型付き言語以外だと、C と同レベルの安全性になる



# Rust: 代数的データ型

```
fn do something() -> Result<i32, SomeError> {
    // ...
   if success {
        0k(42)
    } else {
        Err(SomeError::new())
```

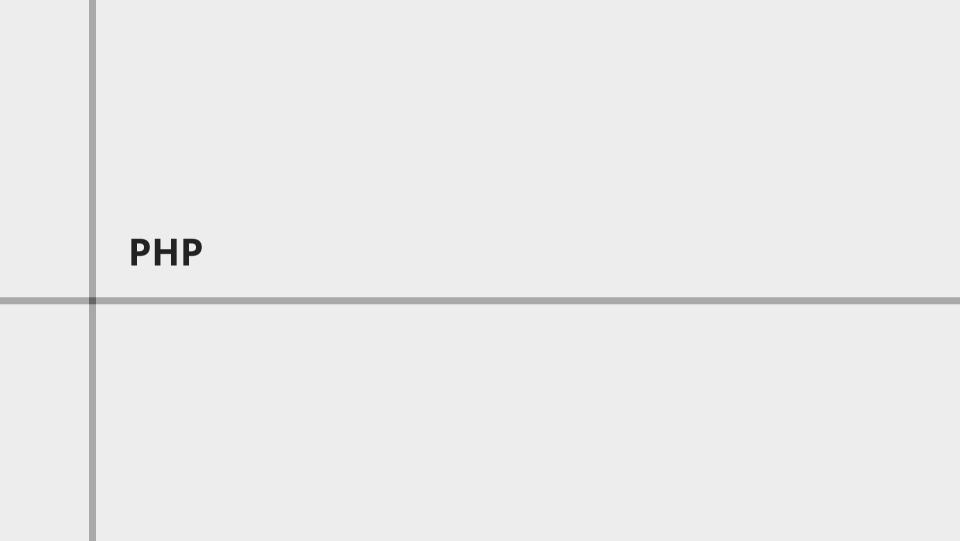
PHP では union 型が一番近い

## Rust: 代数的データ型 + panic

- Good 処理結果があるときに使える
- Good エラーに情報を載せられる
- Good エラーハンドリングを省略するとコンパイルエラーになるGood 処理結果がない場合でも、linter での検知は可能
- Good 特殊な構文なし、大域脱出なし
- Good エラー値のカテゴライズに継承が使われない
- Good 回復可能なエラーと回復不能なエラーを区別できる

## Rust: 代数的データ型 + panic

- Good 処理結果があるときに使える
- Good エラーに情報を載せられる
- Good エラーハンドリングを省略するとコンパイルエラーになるGood 処理結果がない場合でも、linter での検知は可能
- Good 特殊な構文なし、大域脱出なし
- Good エラー値のカテゴライズに継承が使われない
- Good 回復可能なエラーと回復不能なエラーを区別できる
- Bad 静的型付き言語以外だと、C と同レベルの安全性になる
- Bad 言語側にそれ相応のシンタックスシュガーがないと書きづらい



## 我らが PHP は ...

C 言語の素朴な値による通知と例外をミックスしたキメラ どう設計すべきかは今回のスコープ外

#### 我らが PHP は ...

C言語の素朴な値による通知と例外をミックスしたキメラ どう設計すべきかは今回のスコープ外 君たちはどう生きるか