Fehlerbehandlung mit try und catch

LES

2023-05-23

1 Das Problem

Alle Programmzeilen, die auf externe Ressourcen (Dateien, Datenbanken…) zugreifen, können scheitern. In professionellen Programmen müssen diese Fehler abgefangen werden.

1.1 Beispiel: Lesen einer Zeile aus einer Datenbank

```php
<?php
$dbVerbindung = new mysqli(...);
$statement = $dbVerbindung->prepare('SELECT Name ... WHERE ... = ?');
$statement->bind_parameter('i', $schluessel);
$statement->execute();
$statement->bind_result($name);
$statement->fetch();
$statement->close();
$verbindung->close();
?>
```

Jede Zeile im Programmbeispiel kann scheitern. `mysqli`-Befehle liefern im Fehlerfall `null` zurück. Man muss dann noch unterscheiden, ob ein ernsthaftes Problem vorliegt (z.B. Datenbank beschädigt) oder ob es sich um einen harmlosen Zustand handelt (z.B. Artikel nicht gefunden).

Klassische Fehlerbehandlung mit `if`-Statements sieht wie folgt aus (Fehler bei `close()` werden oft ignoriert):

```php
<?php
if(!$dbVerbindung = new mysqli(...)) {
  echo 'Keine Verbindung'; die;
}else {
  if(!$statement = $dbVerbindung->prepare('SELECT Name FROM ...  = ?')) {
    echo 'Prepare gescheitert';
    $dbVerbindung->close();
    die;
  }else {
    if(!$statement->bind_parameter('i', $schluessel)) {
      echo 'bind_parameter gescheitert';
      $statement->close();
      $dbVerbindung->close();
      die;
    }else {
      if(!$statement->execute()) {
        echo 'execute gescheitert';
        $statement->close();
        $dbVerbindung->close();
        die;
      }else {
        if(!$statement->bind_result($name)) {
          echo 'bind_result gescheitert';
          $statement->close();
          $dbVerbindung->close();
          die;
      }else {
        if(!$statement->fetch($name)) {
          echo 'fetch gescheitert';
          $statement->close();
          $dbVerbindung->close();
          die;
        }
      }
    }
  }
}
$statement->close();
$verbindung->close();
?>
```

Probleme:

- unübersichtliches mehrfach verschachteltes `if`
- Der Fehlerbehandlungscode wiederholt sich zumindest teilweise.

2 Einsatz von try und catch

Die Lösung sieht so aus: Alle Anweisungen, die scheitern können, werden in einen `try`-Block eingeschlossen. Wenn eine Anweisung scheitert, wirft sie mit `throw` eine *Exception*. Die Programmausführung geht dann im `catch`-Block weiter. Wenn es keinen `catch`-Block gibt, bricht das Programm ab.

Wenn alle Anweisungen fehlerfrei durchlaufen, wird der `catch`-Block ignoriert.

Neben der Trennung von normalem Code und Fehlerbehandlung hat diese Lösung auch den Charme, dass ein Problem eindeutig identifiziert werden kann.

Erstes Beispiel: Wir akzeptieren die Standard-Exceptions von PHP.

```
<?php
// Exceptions scharf schalten:
mysqli_report(MYSQLI_REPORT_ERROR | MYSQLI_REPORT_STRICT);
try {
  $dbVerbindung = new mysqli(...);
  $statement = $dbVerbindung->prepare('SELECT Name ... WHERE ... = ?');
  $statement->bind_parameter('i', $schluessel);
  $statement->execute();
  $statement->bind_result($name);
  $statement->fetch();
  $statement->close();
  $verbindung->close();
}catch(Exception $e) {
  echo '<p>Problem beim Datenbankzugriff: ' . $e->getMessage() . '</p>';
  if($statement) {
    $statement->close();
  }
  if($dbVerbindung) {
    $dbVerbindung->close();
  }
  die;
}
```

Zweites Beispiel: Sie haben eine Funktion `getMaxIndex()` (die den höchsten Indexwert aus einer Tabelle liefert) geschrieben. Diese Funktion liefert `0` zurück. Am Rückgabewert kann nicht unterschieden werden, ob die Tabelle leer ist oder ob es einen Fehler bei der Ermittlung der höchsten Zahl gab. Wenn die Funktion einen Fehler kennzeichnet, indem sie eine Exception wirft, ist die Unterscheidung einfach. Wir definieren unsere eigene Exception und werfen sie mit speziellen Meldungen.

```php
<?php
class AbfrageException extends Exception
{
}
try {
  if(!$dbVerbindung = new mysqli(...)) {
    throw new AbfrageException('Keine Verbindung');
  }
  if(!$statement = $dbVerbindung->prepare('SELECT Name FROM ...  = ?')) {
    throw new AbfrageException('Prepare gescheitert');
  if(!$statement->bind_parameter('i', $schluessel)) {
    throw new AbfrageException('bind_parameter gescheitert');
  }
  if(!$statement->execute()) {
    throw new AbfrageException('execute gescheitert');
  }
  if(!$statement->bind_result($name)) {
    throw new AbfrageException('bind_result gescheitert');
  }
  if(!$statement->fetch($name)) {
    throw new AbfrageException('fetch gescheitert');
  }
}
catch (AbfrageException $e) {
  echo 'gescheitert mit ' . $e->getMessage();
  if($statement) {
    $statement->close();
  }
  if($dbVerbindung) {
    $dbVerbindung->close();
  }
  die;
}
$statement->close();
$dbVerbindung->close();
?>
```

Man kann auch mehrere `catch`-Blöcke verwenden, um verschiedenartige Exceptions abzufangen. Beispiel: Eine Exception-Klasse für Datenbankprobleme und eine für nicht gefundene Zeilen.

```php
<?php
class AbfrageException extends Exception
{
}
class SchluesselException extends Exception
{
}
try {
  ...
  if(!$statement->bind_result($name)) {
    throw new AbfrageException('bind_result gescheitert');
  }
  if(!$statement->fetch($name)) {
    throw new SchluesselException('Schluessel nicht gefunden', $id);
  }
  ...
}
catch(mysqli_sql_exception e){
  // SQL-Fehlernummer besorgen
  echo 'SQLstate: ' . e->getSqlState();
}
catch(AbfrageException $e) {
  ...
  // Wenn die Datenbankverbindung kaputt ist: aufgeben.
  die;
}
catch SchluesselException $e {
  echo 'Datensatz zu Schlüssel ' . $e->getCode()
      . 'nicht gefunden. Fehlertext: ' . $e->getMessage();
  // Programm kann weiter laufen.
}
```

2.1 Annotation

Dass eine Funktion eine Exception werfen kann, kann man in einem strukturierten Kommentar ankündigen. Das erleichtert die Verwendung der Funktion.

```php
<?php

/**
 * Lädt ein Produkt aus der Datenbank
 *
 * @param string produktnummer
 * @return Produkt
 *
 * @throws AbfrageException falls Datenbankproblem
 * @throws SchluesselException falls nichts gefunden
 */

 function ladeProdukt($nummer)
 {
 ...
 }
```

3 Aufgaben zu try/catch

1. Erklären Sie kurz, wofür try-catch verwendet wird.
2. »*try-catch* macht Programmcode besser verständlich.« Begründen Sie diese Aussage.
3. Formulieren Sie den folgenden Pseudocode so um, dass ein Fehler geworfen wird, wenn die Texte nicht übereinstimmen.

```
function pruefePasswort(string $eingabe) {
    global $passwort;
    if($eingabe != $passwort) {
        return false;
    }
    return true;
}
```

4. Zeigen Sie, wie der Funktionsaufruf geändert werden muss, wenn die Funktion pruefePasswort() einen Fehler wirft.

```
if(pruefePasswort($eingabe) == false) {
    echo 'Ihr Passwort ist falsch.';
    return;
}else {
    echo 'Sie sind angemeldet';
}
```

5. Zeigen Sie, wie der folgenden Programmcode geändert werden muss, damit *Erfolg* ausgegeben wird, wenn kein einziger Funktionsaufruf *null* zurückgeliefert hat.

```
if(($datei = oeffneDatei($name)) != null) {
    if(($inhalt = liesEineZeile($datei)) != null) {
        echo $inhalt;
    }else {
        echo 'Fehler!';
    }
    if(schliesseDatei($datei) == null)  {
        echo 'Fehler!';
    }
}else {
    echo 'Fehler!';
}
```

6. Formulieren Sie den Pseudocode aus dem vorigen Beispiel mit einem try-catch-Konstrukt um. Nehmen Sie dabei an, dass die Funktionen oeffneDatei(), liesEineZeile() und schliesseDatei() sich so umschalten lassen, dass sie einen Fehler werfen, wenn etwas schiefgeht (anstatt null zurück zu liefern).
7. Zeigen Sie, wie jetzt Aufgabe 5 gelöst werden kann.

3.1 Lösungen

1. try-catch-Konstrukte werden zur Fehlerbehandlung verwendet.
2. Es gibt mehrere Gründe:
   - Fehlerbehandlung wird von der normalen Programmlogik getrennt.
   - Verschachtelte oder verkettete if-Konstrukte fallen weg.
   - Code muss nicht dupliziert werden.
3. Ungefähr so:

```
1 function pruefePasswort(string $eingabe) {
2     global $passwort;
3     if($eingabe != $passwort) {
4         throw new Exception('Passwort ist falsch');
5     }
6     return true;
7 }
```

4. Die Erfolgsmeldung kann erst nach Zeile 9 gemacht werden. Dazu muss aber sichergestellt sein, dass sowohl Zeile 2 wie auch Zeile 7 erfolgreich waren. Das kann folgendermaßen gemacht werden:
   - In Zeile 3 wird eine Variable gesetzt. In Zeile 9 beginnt ein else-Teil zum if von Zeile 7, der die Variable prüft und dann gegebenenfalls die Erfolgsmeldung ausgibt.
   - Der Code zum Schließen der Datei wird verschoben, dupliziert und um einen else-Teil ergänzt:

```
1  if(($datei = oeffneDatei($name)) != null) {
2    if(($inhalt = liesEineZeile($datei)) != null) {
3      echo $inhalt;
4      if(schliesseDatei($datei) == null)  {
5        echo 'Fehler!';
6      }else {
7        echo 'Erfolg!';
8      }
9    }else {
10     echo 'Fehler!';
11     if(schliesseDatei($datei) == null)  {
12       echo 'Noch ein Fehler!';
13     }
14   }
15 }else {
16     echo 'Fehler!';
17 }
```

5. Etwa so:

```
1  try {
2    $datei = oeffneDatei($name));
3    $inhalt = liesEineZeile($datei);
4    echo $inhalt;
5    schliesseDatei($datei);
6  }catch (Exception e){
7    if($datei != null)
8      schliesseDatei($datei);
9    echo 'Fehler!';
10 }
```

6. Die Erfolgsmeldung wird nach Zeile 5 eingefügt.