例外処理

目次

1. 例外処理とは
2. 例外処理の書き方`try~catch`
3. `Error`クラス
4. 例外処理とコードの可読性
5. 非同期処理での例外処理

1.例外処理

Step1(外部サーバーのファイルにアクセス)→Step2(データを取得し値を加工)→Step3(Step1 ので取得したデータをもとに別のファイルにアクセス)と順番に処理を進めるプログラムがあったとする。Step2 の加工部分で失敗した場合、処理を中断し、通常のルートから外れてエラーが出たことを知らせる必要がある。その際の通常ルートではないコード部分を例外処理と呼ぶ。

2.例外処理の書き方try~catch

- コードの概要

```
try {
  //例外が発生する可能性がある処理
  throw 例外を作成;
} catch (エラーを受け取る変数) {
  //例外発生時の処理
} finally {
  // try~catch実行後の処理(省略可能)
}
```

- コード例

```
try {
  if (true) {
    //正常処理
  } else {
    throw new Error("サーバーアクセス失敗");
  }
} catch (e) {
  console.log(e.message);
}
```

- サンプルコード：4_am_samplecode_3.ts
- ログ結果

```
非同期処理に失敗しました
```

Errorクラス

- 例外処理で問題が発生したときの情報伝達に使う
- newすることでオブジェクトとなる
- Errorクラスは作成時にメッセージの文字列を受け取れる
- name属性にはクラス名、messageには作成時にコンストラクタに渡した文字列が格納される

例外処理とコードの可読性

例外処理は、処理が細かく分岐するため、コードが煩雑になる可能性がある。以下のようなポイントを抑えてコードを記述するとよい

- try節はなるべく狭くする

  tryが広すぎると、例外がどこで発生するか判りずらくなるため。実際に実行時に例外が起きうる(デバッグしても例外を制御できない)ポイントは、外部の通信などごく一部のため、たくさん例外処理を書く必要がない。 **例外処理を使わず、エラー処理ができる箇所はそのように書く(if 文での分岐)**

- Error以外をthrowしない

  catchの引数はErrorという前提で書くことで、型定義などよりシンプルに書ける。

非同期処理での例外処理

Promiseの場合catch()節でエラーをキャッチする。

- コード例

```
{
  function PromisFunc(msg: string, time: number) {
    return new Promise<void>((resolve, reject) => {
      setTimeout(() => {
        if (time < 6000) {
          console.log(msg);
          resolve();
        } else {
          reject(`${msg}の非同期で失敗しました`);
        }
      }, time);
    });
  }

  PromisFunc("1回目の非同期", 2000)
    .then(() => PromisFunc("2回目の非同期", 300))
    .then(() => PromisFunc("3回目の非同期", 3000))
    .then(() => PromisFunc("4回目の非同期", 6000))
    .then(() => PromisFunc("5回目の非同期", 1000))
    .catch((err) => console.log(err));
}
```

- サンプルコード：4_am_samplecode_0-5.ts
- ログ結果

```
1回目の非同期
2回目の非同期
3回目の非同期
4回目の非同期の非同期で失敗しました
```

async/awaitの場合は、try~catchでエラー処理を書く

- コード例

```
function promise(time: number, msg: string) {
  return new Promise<string>((resolve, reject) => {
    setTimeout(() => {
      if (time < 6000) {
        console.log(msg);
        resolve(`${msg}成功`);
      } else {
        reject(`${msg}が失敗しました。`);
      }
    }, 3000);
  });
}
async function async(num: number) {
  try {
    if (num % 2 === 0) {
      const result1: string = await promise(5000, "1回目の非同期処理");
      const result2: string = await promise(4000, `${result1}→2回目の非同期処
理`);
      const result3: string = await promise(2000, `${result1}→3回目の非同期処
理`);
    } else {
      throw new Error("渡されてきた引数が2で割り切れません");
    }
  } catch (error: unknown) {
    if (error instanceof Error) {
      console.log(error.message);
    }
  }
}

async(35)
  .then(() => console.log("全ての非同期処理が完了しました"))
  .catch((err) => console.log(err));
```

- サンプルコード：4_am_samplecode_2-3.ts
- ログ結果

```
渡されてきた引数が2で割り切れません;
全ての非同期処理が完了しました;
```

- ポイント
  - returnの代わりに例外や値をthrowするとreject(失敗)とみなされる

- エラーの場合処理は、`catch()`の方へ入る
- `err.message`で`throw`したメッセージを受け取れる