@NagaokaKenichi 2018年12月04日に更新

アルス・ウェアー Advent Calendar 2018 | 4日目

try-finallyよりもtry-with-resourcesを使おう

Java java8 EffectiveJava 新人プログラマ応援

⚠ この記事は最終更新日から1年以上が経過しています。

try-finallyでのリソースクローズ

tryブロックの中で何らかのリソース（InputStream、OutputStream、BufferedReader等）を扱う場合、Java7以前はfinallyブロックでcloseメソッドを呼び出すことで、tryブロック内の処理が正常終了したか異常終了したかに関わらずリソースが確実に閉じられることを保証していました。以下のような形です。

```
protected void copy(String src, String dest) throws IOException {
    InputStream in = new FileInputStream(src);
    try {
        OutputStream out = new FileOutputStream(dest);
        try {
            byte[] buf = new byte[100];
            int n;
            while ((n = in.read(buf)) >= 0) {
                out.write(buf, 0, n);
            }
        } finally {
            out.close();
        }
    } finally {
        in.close();
    }
}
```

また、finallyブロックで例外が発生しないよう更にチェックしたり等もありますよね。

```
} finally {
    if (br != null)
        try {
            br.close();
        } catch (IOException e) {
            // 例外処理
        }
}
```

ただし、これらの書き方にはいくつか問題点があります。

try-finallyでのリソースクローズの問題点

- close処理を色々な箇所で行ったりリソースのnullチェックを行ったり等コードが冗長となる
- tryブロックの中とfinallyブロックの中の両方で例外が発生した場合、finallyブロックの中で発生した例外がスローされる（本来tryブロックの中で発生した例外が知りたいはずだが、その例外は抑制される）

こういった問題に対して、Java7でtry-with-resources構文が導入されました。try-with-resources構文によりこれらの問題は一挙に解決されます。

try-with-resourcesでのリソースクローズ

tryのすぐ後ろにクローズの対象となるリソースの生成処理を記述します。クローズすべきリソースが複数ある場合はセミコロンで区切ります。

```
protected void copy(String src, String dest) throws IOException {
    try (InputStream inputStream = new FileInputStream(src); OutputStream outputStream = new
        byte[] buf = new byte[100];
        int n;
        while ((n = in.read(buf)) >= 0) {
            out.write(buf, 0, n);
        }
    }
}
```

このように書くことで以下のような挙動になります。

- finallyブロックでcloseメソッドを呼び出さなくても自動でcloseをしてくれるようになる（ソースが簡潔になり可読性が向上する）
- close処理内で例外が発生してもtryスコープの中の例外がスローされる（本来知りたいはずの例外がスローされる）

これにより「try-finallyでのリソースクローズの問題点」で挙げた問題点が解決されます。

教訓

クローズしなければならないリソースを扱う場合は、try-finallyよりもtry-with-resourcesを使おう

参考

- Oracle Technology Network
- Effective Java

長岡 賢一 @NagaokaKenichi
https://roku-sui.hatenablog.com
フォロー

コメント

この記事にコメントはありません。