エラーハンドリング勉強会（2011-09-04）

エラーハンドリングモデル考察

導入

- 自分なりにエラーハンドリングまわりについていろいろ考察した内容をまとめてみました。
- エラーハンドリングを制することは間違いなくプログラミングを制するのに必要不可欠な要素ですので、エラーハンドリングそのものについての理解を深めていきましょう！

前回の発表

- 以前、「Boost.勉強会 #3 関西」でも「エラーハンドリング」というタイトルで発表しました。主張が変わってるところなどがありますが、よろしかったらそちらもご参照ください。
  - PPTX版: http://www.trickpalace.net/paper/error.handling.pptx
  - PDF版: http://www.trickpalace.net/paper/error.handling.pdf
  - slideshare http://www.slideshare.net/wraith13/ss-7987895

トピック

- エラーとは？
- エラー情報モデル
- 人間系エラーハンドリング
- 未検知エラー

この発表で「エラー」について

- 全般的に抽象的な話なので、抽象的且つ広義な意味でのエラーをさします。

エラーハンドリングモデル考察

エラーとは？

そもそもエラーとは？

そもそもエラーとは？

- 「なんらかの不整合」

エラーハンドリングの目的

- エラーが「なんらかの不整合」であるとするとエラーハンドリングの目的は次のようなものだと言える。
  - 不整合を正す、あるいは正された状態に近づける。
  - 不整合を記録あるいは通知する。
    - 人間系をシステムの一部と見なす場合、これは目的ではなく前項を果たす為の手段と言える。

エラーハンドリングモデル考察

エラー情報モデル

エラー情報モデル

- どのようにエラー情報を扱うかはエラーハンドリングの肝の一つです。
- さまざまなエラー情報モデルについて考えてみましょう。

エラー情報モデル

- voidエラーモデル
- unitエラーモデル
- singleエラーモデル
- stackエラーモデル/nestエラーモデル
- listエラーモデル
- treeエラーモデル

エラー情報モデル

- voidエラーモデル
  - エラーをそもそも検知すらしないモデル。
  - 結果として問題が発生した場合には暴走などといった事態を招きやすい。
  - 実例：Z8oアセンブラ

エラー情報モデル

- unitエラーモデル
  - エラーを検知はするがその情報を扱わないモデル。
  - 検知により処理を中断するのみ。
  - 実例：Icon言語

エラー情報モデル

- singleエラーモデル
  - エラーを検知しその情報を同時にひとつのエラーについてのみ扱うモデル。
  - 現状、もっとも一般的だと思われるモデル。
  - 実例：86系アセンブラ、Win32API

エラー情報モデル

- stackエラーモデル/nestエラーモデル
  - エラーを検知しその情報を必要に応じて入れ子にして扱うモデル。
  - singleエラーモデルを拡張したものと言え、一般的に言語/システムの機能ではなくライブラリで実現される。
  - 実例：Java,C#,C++11

エラー情報モデル

- listエラーモデル
  - エラーを検知し必要に応じてその情報をリストで扱う。
  - エラーが発生したままの状態でも処理の継続が可能なモデル。
  - 実例：各種コンパイラ

エラー情報モデル

- treeエラーモデル
  - エラーを検知しその情報を必要に応じてtreeで扱うモデル。
  - stackエラーモデル/nestエラーモデルとlistエラーモデルのハイブリッド。
  - 実例：trickerr.h (C++)

エラーハンドリングモデル考察

人間系エラーハンドリング

人間系エラーハンドリング

- 仮に人間系もシステムの一部と見なした場合、それはどのようなものであるべきか考えてみましょう？

人間系エラーハンドリング

- システム的な視点での人間の意味は次のようなものとなります。
  - 大本の呼び出し元関数。
  - 割り込みイベントの発生元のひとつ。
  - 末端の呼び出し先関数。

  - →関数およびイベントと見なせる。

人間系エラーハンドリング

- 人間系の関数の特徴
  - 非常に動作が重い。
  - グローバル変数(リソース)にアクセスしまくり。
  - 失敗がすることがある。(誤操作、誤認、誤判断)
  - →関数としては最悪ですね。

人間系エラーハンドリング

- 人間系が呼び出し元関数であるならばエラー情報を受け付けるべき。
  - プログラム全体もひとつの関数であるという視点に立って、内部の関数がそうであるようにエラーをありのまま返すべき。

人間系エラーハンドリング

- 人間系が末端の呼び出し先関数であるならばエラー情報を返せるべき。
  - 人間系もひとつの関数であるという視点に立って、プログラム内の関数がそうであるようにエラーを返し得るという前提に立つべき。

人間系エラーハンドリング

- 実際に人間系関数はどのようなエラーを返すのか？
- 例：
  - キャンセル
  - タイムアウト
  - 一時停止
  - サスペンド
  - 終了
  - ヘルプ
  - バグレポート
  - サポートリクエスト

人間系エラーハンドリング

- これらのエラーは常に人間が返せるようになっているべき。
- 特にバグレポートやサポートリクエストはシステムとして用意しておくことで、より詳細な情報取得が可能になり、デバッグやサポートの役に立つことでしょう。

エラーハンドリングモデル考察

未検知エラー

未検知エラー

- システム的にエラーはエラーとして検知されて初めてエラーとなります。
- これは言い換えると本来エラーであるべき状態であってもエラーを検知するコードが不十分な為にエラーとして認識されない状態があることになります。

未検知エラー

- 実害があるとは限らないですが、未検知エラーは全てバグと言えます。
- 「疑わしきはエラー」とする視点に立つとプログラムは常にエラー状態にあるとも言えます。

未検知エラー

- 未検知エラーへの対処として人間系関数がエラーを返すことで問題を軽減できる可能性がある。
  - →未検知エラーへの対処の一環として、本来、システム系しか返すことの無いハズのエラーも人間系関数から返せるようにするのも面白いかも？

エラーハンドリングモデル考察

質疑応答

エラーハンドリングモデル考察

ご清聴ありがとうございました。