快速デバッグ技法

脱！休日出勤

.NETアプリケーション障害解析の極意

新連載

エラー処理の書き方
－プログラミング工程－

株式会社NTTデータ
飯山 教史
IIYAMA, Takashi

デバッグのポイントを押さえる

「開発の稼動の約7割はデバッグ」と言われて久しい。1995年、Windows 95が出荷されて以来IT産業は隆盛し、多くのプログラマが現場で開発作業を行なうようになった。それからXPだTDDだと開発者向けに「楽しく」「高品質に」「体系立てて」開発する手法が登場した。しかし、どのような開発プロセスであっても障害を見つけたらデバッグしなければ、問題は解決できない。

つまり稼動削減の一手法として「高度なデバッグ手法」はもっと注目を集めても良い分野のはずだが、デバッグを体系立てて説明している本は少ないのが現状である。

この理由としてデバッグには高度な技術力が求められる点が挙げられる。現在、現場にいる多くの開発者たちは（たとえ理系の大学院出身であっても）学生時代にOSなどデバッグに必要なコンピュータのアーキテクチャを詳細に学んだ人は少ない。そして開発者となった今日でも、業務の忙しさに圧倒されてこの知識を身につける余裕などないのである。しかも書籍などで紹介されているデバッグのノウハウはこれらの高度な知識を前提にしているので現場の開発者たちに読まれることはない。これら書籍の著者は「デバッグ＝修行」とし、読者にストイックな姿勢を求めているからだ。このような状況により、高度なノウハウが展開されず、その結果また品質の悪いコードが出る、という悪いスパイラルに陥っている。

そこでこの連載では「とりあえず動くものは書ける」状態から、一段スキルアップしたい開発者を対象とし、高度な知識を前提とせずにデバッグの「手順」と「その結果からわかること」を説明していきたい。また、毎回説明するトピックをスムーズに理解してもらえるように、本連載では実際の開発工程に沿って解説を進めることにする（図1）。

その意味で本連載の目的は「プロフェッショナル」になることではなく「作業を早く終わらせる」ことである。作業とは、デバッグであれば、「問題部を切り分け、解決はその部分を担当する開発者に任せるスキームを作る」ところまでである（よく言われる「障害の切り分け」である）。よって、かなり技術レベルの低いトピックを説明することもあると思うが、私はこれこそ現場のニーズであると考えている。

この記事が皆さんを現場のトラブルからいち早く解放する手助けとなることを切に願う。トラブルはさっさと片付けて、空いた時間を有効に使ってもらえれば幸いである。

我々の合言葉は以下のとおり。

レベル >>> Level

❶ ❷ ❸ ④ ⑤

言語 >>> Language

- Visual Basic

ツール >>> Tool

- Visual Studio 2005 Team Edition for Software Testers

A worker is a loser.（働いたら負け！）

では早速はじめましょう。

エラー処理で仕様を決めよう！

今回は「プログラミング工程」である。この作業は人任せにできないので一生懸命にやろう。後工程でエラーが出るといろいろと面倒なので、気づく限りケアするコードを書かなくてはならない。いわゆる「エラー処理」だ。

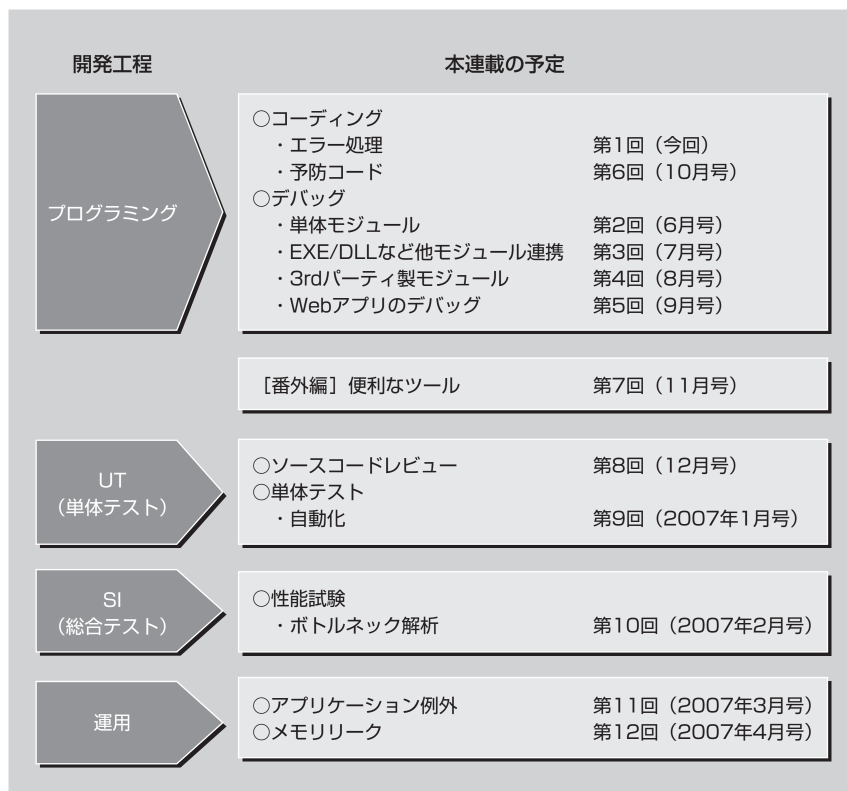
開発者は忘れがちであるが、それぞれのエラー処理は仕様とからむ。なぜなら故障が起こったときの後処理はシステム全体の方針に影響を与えるからだ。そのため、開発者は必ず開発する部位で発生する故障の原因を洗い出し、その後処理を早い段階[注1]で決めなくてはならない。後処理を決める主な観点は以下のとおり。

①連続運転させるか？

業務アプリケーションなら、「落ちない」を優先させる傾向がある。この場合は例外が発生してもログに残して再試行する。対して、MRIなどの医療検査用システムなら、故障発生時に運転を続けるよりも処理しないほうが良いので処理を中断する。これらのように、アプリケーションやシステムに合った後処理を定義する。

注1）業務フローがわかればエラーと最終出力との関連付けができる。これがわかれば、あとは最終出力へのエラー情報受け渡し方針を決めるだけである。

図1：開発工程とデバッグのポイント



②ログに記録する内容

開発者向けのデバッグログとして何を記録するかを決める。たとえばエラーが発生した場所、原因（エラーを起こした関数に渡されたパラメータ）など。同時に、システムの利用者向けのログ（エラーの通知、エラー時の後処理のナビゲートなど）と、運用者向けのログ（発生したエラータイプと時間など）も決める。

③後処理はどこがやるか？

エラーの後処理の方針を決める。自前のコードで処理（リソースの解放、再試行の実行など）を行なうか、上位へ通知（例外／エラーコードを返す）するかを決めること[注2]。

これらがあいまいなとき、決して「これでいいや」と勝手にコードを書いてはいけない。開発者は仕様どおりに動くコードを書くのが仕事であって、自由に仕様を変更してはいけないからだ。故障発生時の後処理が不明瞭な場合は、必ず責任者へ連絡して仕様を定義させなくてはならない。

エラー処理の種類

ここからは打率を表示する、簡単なサンプルコードを使って説明していこう。ありがちなエラーは「打席数を0と入力する」だろう。このとき皆さんな

注2）この取り決めを徹底しておかないとモジュール間の結合試験の際に多くのエラーが報告されることになる。