開発者必読！

“もしも”のときのデバッグ技法

.NETアプリケーション障害解析

株式会社NTTデータ
木村 佳織
KIMURA, Kaori
Supported by 飯山 教史

第９回

リフレクションでバグを摘み取る

はじめに

リリースしたシステムの品質が低いという場合は「テスト不足」がその原因として挙げられる。仕様どおりに動作することをリリース前に確認するためにはテストを実施するしかないからだ。ところが開発工期の短期化に伴い、十分にテストされないままプログラムがリリースされることも珍しくはない。

最近言われている「テストの自動化」はこの問題を解決するためのテスト手法である。この手法ではテスト用のプログラムをあらかじめ実装しておく。このプログラムを利用して、テスト実施者が手動でテストを実行する手間を省き、テストにかかる時間を削減する。

.NETでこの手法を利用できるように開発されたのが「リフレクションクラス」である。リフレクションを利用すれば実装したクラスのメソッドやフィールドに動的にアクセスできるので、開発者は容易にテストプログラムを開発できるようになる。

今回は、リフレクションを利用してテストプログラムを記述する手法について説明する。

また、単体テストツールNUnitを利用した効率的なテスト方法についても簡単に触れることにする。

Level

1 2 3 4 (5)

Technology Tools

- ◯ Visual Basic
- ✓ Visual C#
- ◯ Visual C++
- ◯ SQL Server
- ◯ Oracle
- ◯ Access
- ◯ ASP.NET
- ◯ Other: ↓ NUnit

単体テストとは

「要件定義」「設計」「製造」「単体テスト」「結合テスト」「総合テスト」という一連のシステム開発の工程の中で、製造工程終了後、一番最初に行なわれるテストが“単体テスト”である。単体テストとは、個々のモジュールのみを対象としたテストであり、モジュールが仕様書に規定されている機能を満たしているかどうか、例外データに対してきちんと対処できるかどうかなどを検証するテストである。

もっと具体的に言うと、単体テストでは主に以下のチェックを行なう。

- 実行したメソッドの戻り値が仕様で定義された値と一致するかどうか
- メソッドを実行した後のオブジェクトの内部状態が仕様で定義された値と一致するかどうか

単体テストではメソッド単位で仕様を満たしているかどうかを確認できる

ので、バグが発生した場合にもメソッド単位での修正が可能になる。そのため、複数のモジュールを組み合わせて行なう結合テスト時にバグを修正するよりも少ない稼動で修正できる。これらの理由から、必ず十分に時間をかけて単体テストを行なわなくてはならない。

開発現場での状況

ところが、開発現場では試験工程に十分な時間を割けられないのが現状である。というのも、以前に比べプロジェクトは短期間化しているため、試験工程に割り当てられる工数が少なくなったからだ。その上、クライアントの度重なる仕様変更で製造工程が膨らみ、試験工程はさらなるしわ寄せを受けることになる。そうなると、手作業で膨大なテストケースを実行しなくてはならないテスト作業は多くの稼動をとられるという理由から品質を確保できないとわかりつつも省略されてしまうのである。

では最初のテストである単体テストを省略するとどうなるだろうか。本来単体テスト時に検出されなくてはならないバグが結合テストで発見されたり、もっとひどい場合にはリリース後に発見されたりする。後工程のテストになるほどモジュールの結合度は高くなるわけだから、これではバグの箇所の特定は困難になり、修正にも多くの時間を費やしてしまう。その結果本来の作業中の工程で行なうはずのテストが十分に実施できなくなり、さらに品質を確保できなくなってしまうのである。

では面倒で多くの作業時間を要するテスト作業を、限られた時間内で効果的に実施するにはどうしたらよいのだろうか。この目的を解決するための手法が「テストの自動化」である。そして、テストの自動化を実現してくれるのが「テストコード」である。メソッドを呼び出し、戻り値を確認する機能を実装したプログラムコードを前もって用意しておけば、テスト対象クラスのメソッドの呼び出しとチェックを自動でしてくれる。そのため手動でテストを実行する手間が省けて時間を短縮できるだけでなく、手動の場合に発生しがちな操作ミスもなくなるのである。

リフレクションを利用した実装

テストコードを開発すればよいことはわかったが、実はこのタイプのコードの作成は手間がかかる。構造化プログラミングの場合は問題ないのだが、オブジェクト指向プログラミングの場合は他クラスのprivateなメソッドにはアクセスできないためオブジェクトの中身を確認するコードの記述方法が難しいからだ。もちろん、単体テストではすべてのメソッドの戻り値を検証する必要があるのでこれらのメソッドにもアクセスして戻り値をチェックしなくてはならない。このアクセス権の制約を解決し、privateなメソッドにもアクセスできるコードを実現してくれるのがリフレクションである。

リフレクションクラスは実行時にクラスの型情報を取得する機能を持ち、クラスのコンストラクタ、フィールド、メソッドに関する型情報を取得できる。この取得した型情報で該当クラスをインスタンス化し、非publicなメソッドやフィールドにもアクセスできるため今まで困難であった、

・非publicなメソッド呼び出し
・非publicなフィールドへのアクセス（フィールドの設定と取得）

が可能になる。

以降では、リフレクションクラスの使用方法を示すため、クラスのprivateなメンバにアクセスするコードを実装する。

▶メソッド呼び出し

単体テストでは実行したメソッドの戻り値が仕様で定義された値と一致するかどうかを確認するため、他クラスのprivateなメソッドにもアクセスする必要がある。そこでprivateなメソッドにアクセスする例として、数値と文字列を引数にとってそれらを連結して表示するテスト対象クラスをリスト1に、そのクラスのメソッドを呼び出すコードをリスト2に示す。

privateなメソッドにアクセスしたい場合は、Reflection名前空間を通してTypeオブジェクトのInvokeMemberメソッドを使用し、引数を以下のように設定する。

・第1引数：呼び出すメンバの文字列
・第2引数：検索の実行方法を指定するひとつ以上のBindingFlagsからなるビットマスク（表1）
・第3引数：一連のプロパティを定義