技術レポート

繰り返し型開発におけるテスト技法
－JUnit について－

IT ソリューション＆マーケティング推進本部　先端技術研究室　田中佳美

１．繰り返し型開発におけるテスト

開発プロセスの１つとして、ウォーターフォール・モデルがある。これは、システムを開発するための作業を、複数の工程に分けて順番に進めていくやり方である（図１参照）。

現在、ウォーターフォール・モデルは最もよく用いられている開発手法である。しかし、情報技術が加速度的に変化している現在の状況には、このモデルはあまりそぐわないようだ。次々と最新技術が登場する状況では、技術的課題に対する解決策を短期間で提示することが求められる。しかし、ウォーターフォール・モデルは、先進的な技術に即応するようには考慮されていない。つまり、既知の技術を基本とするシステムの開発には適しているのだが、新しい技術を盛り込んだシステム開発には対応していないのである。

このような、採用経験のない技術に柔軟に対応すべく考え出された手法が、繰り返し型開発である。繰り返し型開発は、分析から統合テストまでの工程を、いく度も繰り返しながら作業を進めていく開発プロセスである。現在、オブジェクト指向の世界で取り上げられている RUP（Rational Unified Process）や XP（eXtreme Programming）は、繰り返し型開発の代表的なものだ。RUP、および XP については、本誌の別記事[*1]で詳しく紹介されているので参照されたい。RUP の流れを図２に示す。

この作業において、繰り返し実行されるテストを、特に回帰テストと呼ぶ。回帰テストはシステムに何らかの手を加えた際に、以前と同じ機能をシステムが保持しているかチェックするために行う。このテストは大切なものであるが、手作業で行うには困難を伴う。同じテスト内容で繰り返し実施する必要があるため、テスターのモティベーションは著しく低下し、テスト結果の見落としや誤ったテストの実行などが起こり易い。

そこで、この回帰テスト実行の自動化を実現するツールである「JUnit」を紹介する。本稿では JUnit の概要と使い方、および有用性について概要を述べ、それから Application Facades を用いた Web アプリケーションへの適用に関して考察する。

２．JUnit とは

JUnit とは、Erich Gamma 氏と Kent Beck 氏が作成した、ソフトウェアをテストするためのプログラムである。

図１　ウォーターフォール・モデル

＊１）『SOFTECHS』本号 p.89～p.94「繰り返し型開発プロセス概説－RUP、XP を中心に－」（金融システム第七事業部　今井雅之）を参照。

図2　RUP の流れ

このプログラムは、XP の思想より生まれた Testing Framework という考えを、Java 言語用に実装したものだ。Testing Framework には、他にも Smalltalk や C++、Visual Basic 版も用意されており、フリーで使用することができる。

Testing Framework では、図3のように、テストをフレームワークとして扱う。

TestCase クラスが Testing Framework の基礎となる。一般的なテストは、準備、実行、後処理という手順で行う。これらの機能のインタフェースを TestCase が提供する。テストを実行した結果を集めるのが TestResult クラスである。TestResult には、実行したテストや発生したエラーの数、最終的に失敗したテストのリストなどの情報が記録される。

TestSuite は複数のテストを統一的に扱うクラスである。この TestSuite により、いくつかのテストケースをひとまとまりのテストとして実行することが可能になる。

プログラマは繰り返し型開発において、基本的に TestCase を継承するテストクラスを作成していく。図3での AppTestCase クラスに相当するものである。テストケースに応じて testXX というメソッドを定義していき、新たにテストコードを書き入れ、必要な場合は setUp メソッドと tearDown メソッドをオーバーライドする。setUp はテストで使うリソースを割り当てるメソッドで、tearDown はそのリソースを解放するものである。テストコード中では assert メソッドを呼び出してテストの実行結果の判断を行う。これは assertion という手法で、例えば以下のように記述する。

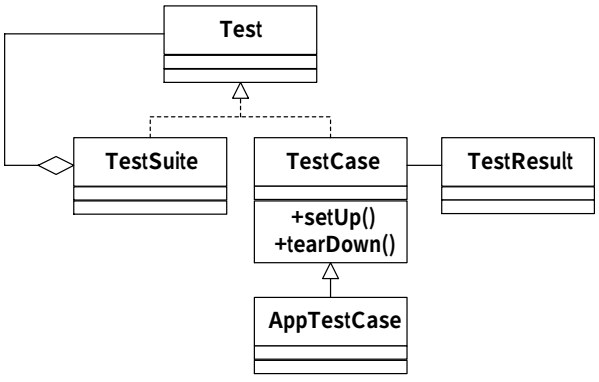

図3　Testing Framework

・assert（expected == result）

テスト対象のプログラムが正常に稼動しているときは expected オブジェクトと result オブジェクトは同じ値を持っている、ということを意味している。すなわち、テストを走らせて expected == result を評価し、true であればプログラムに問題はないと見なされる。false だった場合は、あらかじめ定義してあったアクションが取られる。JUnit では、TestResult に assertion failure が記録されることになる。この assert メソッドを使って、引き続きさまざまなケースのテストを行っていくのである。

テストを実行すると、実行時間や発生したエラーなど、TestResult に集められた情報が表示される。JUnit にはテスト結果を表示する GUI（Graphical User Interface）が付いており、それを使用した場合の結果表示は図4のようになる。

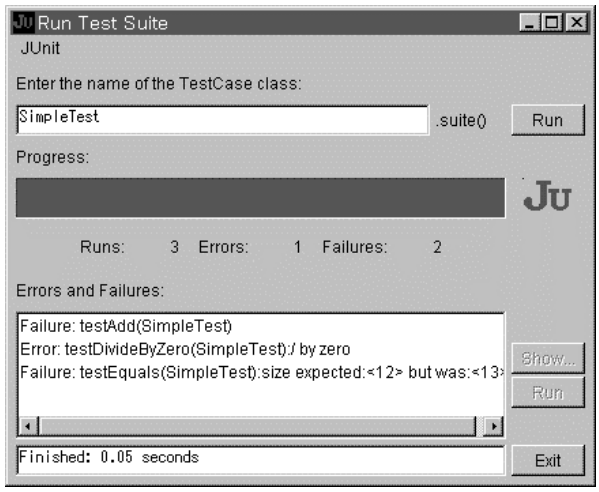

図4　JUnit の GUI

3．JUnit の使用方法

3.1　JUnit の簡単な使用例

テストクラスの例（JUnit のチュートリアルに記載）を作例1に示す。

TestCase クラスのコンストラクタは、引数に文字列を必要とする。テストが成功しなかったとき、どのテストで失敗したかを特定することに用いられる。

次に setUp メソッドを、テスト対象となるクラスを生成するようにオーバーライドしている。テストコードは“12 YEN”という Money オブジェクトに“14YEN Money”を

〈テスト対象のクラス〉

```
class Money {
    private int fAmount;
    private String fCurrency;

    public Money(int amount, String currency) {
        fAmount = amount;
        fCurrency = currency;
    }

    public Money add(Money m) {
        return new Money(amount() + m.amount(), currency());
    }

    public int amount() {
        return fAmount;
    }

    public String currency() {
        return fCurrency;
    }

    public boolean equals(Object anObject) {
        if (!(anObject instanceof Money)) return false;
        Money aMoney = (Money)anObject;
        return aMoney.currency().equals(currency()) && aMoney.amount() == amount();
    }
}
```

〈テストクラス〉

```
import junit.framework.*;

public class MoneyTest extends TestCase {
    protected Money f12YEN;
    protected Money f14YEN;

    public MoneyTest(String name) {     // コンストラクタ
        super(name);
    }

    protected void setUp() {     // テストで使うリソースを割り当てるメソッド
        f12YEN = new Money(12, "YEN");
        f14YEN = new Money(14, "YEN");
    }

    public void testSimpleAdd() {     // テストコード
        Money expected = new Money(26, "YEN");
        Money result = f12YEN.add(f14YEN);
        assertEquals(expected, result);
    }

    public static Test suite() {     // 実行時に最初に呼び出されるメソッド
        TestSuite suite = new TestSuite();
        suite.addTest(new MoneyTest("testSimpleAdd"));
        return suite;
    }
}
```

作例1　テストクラスの作例

足したものが“26YEN Money”と等しくなっているかどうか、というものだ。

最後の suite メソッドは Java 言語における Main メソッドのようなもので、JUnit でテストクラスを走らせた際に自動的に呼び出されるものである。TestSuite の addTest メソッドで実行するテストを指定する。

3.2 優れている点と、それに伴うトレードオフ

全体的に JUnit は、繰り返し実行する単体テストにとても有用なツールであるといえる。

作成しているクラスにメソッドを１つ書き加えるなど、開発しているプログラムに何らかの変更を施したとき、果たしてそのメソッドはきちんと動くのか、また他の部分に影響がおよんでいないか、JUnit を実行することで即座に明らかにできる。

実行した結果は、JUnit の GUI に assert メソッドの利用によって、“OK”であれば緑のプログレス・バーで、“NG”であれば赤のプログレス・バーとエラーメッセージで示される。画面や帳票、ファイルに出力されたプログラムの実行結果を、いちいちチェックする必要がない。

さらに startUp メソッドと tearDown メソッドが testXX メソッドごとに呼び出されるので、testXX メソッドとして記述したさまざまなテストケースは、お互いに全く関わりを持たずに個別に実行される。１つのテストケースで起こったエラーは、他のケースの実行には影響しない。開始したテストは途中でアベンドしてしまうことなく最後まで実行される。

また Testing Framework では、テストを頻繁に実施することを推奨している。これによりテストをパスしたコードと、それに対して手を加えたコードとの差異が小さくなるため、エラーが発見された場合でも原因を特定しやすい。かといって無理にテストの実行を心掛けなくても、JUnit はテストを何度も行うほどに、その利便性が発揮されるツールなので、利用者は自然にテストの間隔が短くなっていく。

ただし、JUnit にも注意して用いるべきポイントがある。まず、基本的に JUnit を使ったテストは、プログラマの技量に大きく依存する、ということである。例えば、assertion を間違えて定義すると、そのプログラムが正しく動いていなくてもテスト結果が“OK”になってしまう可能性がある。

また、コーディングの他にテストコードも書かなければならない。プログラムを書いたらテストも書くことによってアプリケーションの品質や生産性が向上する、ということは周知の事実である。しかし、時間に追われておろそかになってしまうのも現実である。ましてや納期間近に「コードを少し、テストを少し、コードを少し、テストを少し･･･(code a little, test a little, code a little, test a little)[*2]」という JUnit の手法を忠実に実行することには、とても抵抗を覚えるだろう。このため、JUnit を使用するにはプログラマの意識の変革はもちろん、システム開発のプロセスやプロジェクトの運営方法の見直しなどが必要である。

４．Web アプリケーションに対する用い方

Web アプリケーションへの適用の可能性について考察する。前述のように、JUnit は繰り返し型開発において大きな効力を発揮できるツールである。しかし、Web アプリケーションの開発に用いるには、乗り越えなければならない課題が存在する。

図５のような、注文を入力する簡単なサンプル・プログラムを対象として考える。

このプログラム作成に際し、考慮しなければならないポイントがある。Web アプリケーションへアクセスするには、一般的にブラウザが用いられる。しかし、JUnit ではブラウザを扱うことができない、ということである。

JUnit は、メソッドを直接呼び出すことの可能なプログラムをターゲットとしている。これでは、JUnit 自体がどれほど有用でも、GUI アプリケーションの開発においては、その実力を示すことができない。この問題は、Application Facades という考え方を取り入れることで解決できる。

4.1 Application Facades

Application Facades は、Martin Fowler 氏によって提示された。３層アーキテクチャの内のユーザー・インタフェース層を、プレゼンテーション層とアプリケーション・ファサード層に分けるという手法である。Application Facades の階層を図６に示す。

ユーザー・インタフェース層は、ドメインのオブジェクトを操作し、受け取ったデータを表示するレイヤである。このユーザー・インタフェース層を、機能に特化した２つの層に分割する。プレゼンテーション層は、画面や帳票上の項目を制御する。アプリケーション・ファサード層は、ドメインからデータを取得して、画面や帳票のフォーマットに合った形に加工する役目を担う。

このようにして、ユーザー・インタフェースとその裏側のドメインの結合度を弱めることにより、GUI をバイパスしてドメインのテストを実施することが可能になる。GUI を扱うテストはスクリプトの作成や編集に手間がかかり、かなり厄介である。この問題をアプリケーション・

＊２）code a little, test a little･･･：Web サイト「JUnit, Testing Resources for Extreme Programming」(http：//www.junit.org/)を参照。

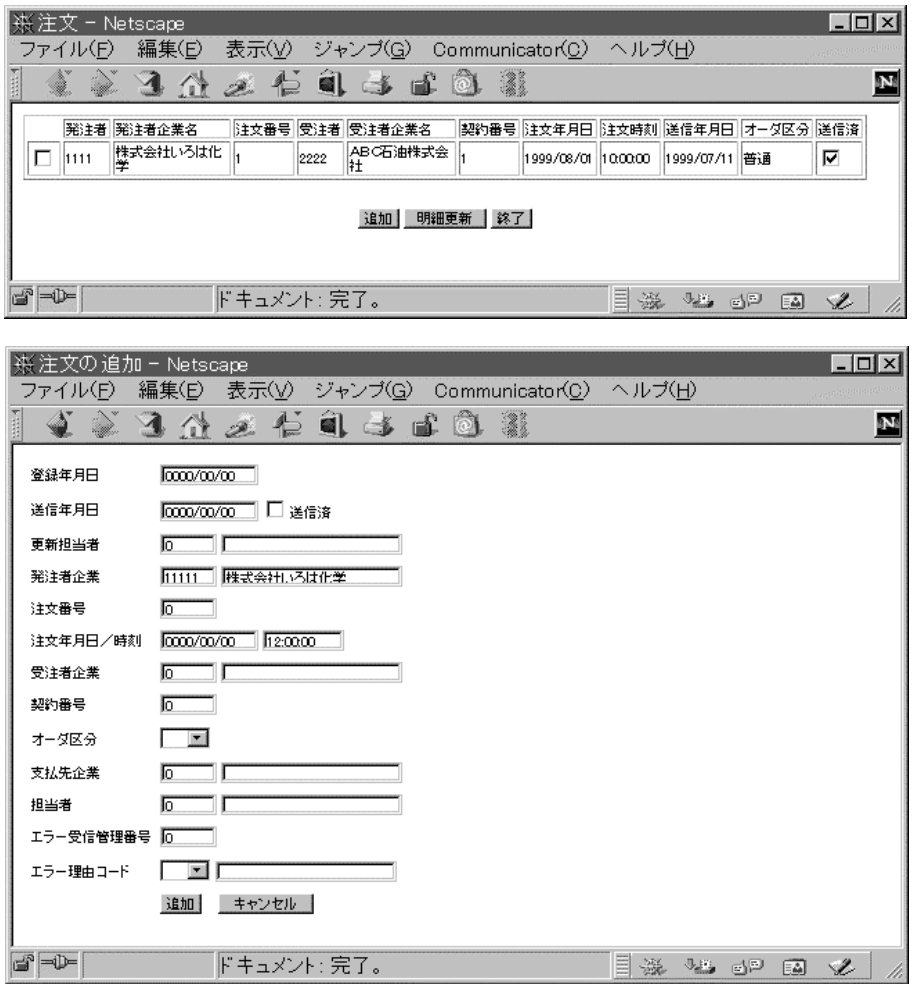

図5　注文を入力するサンプル・プログラム

ファサード層へテストを行うことで解決できる。GUIのレイアウトまではチェックできないが、アプリケーションが正しく動くかどうかはファサードに対するテストでこと足りるのである。しかも、GUIから切り離されているので手軽にテストでき、自動化も容易である。GUIに対して行わなければならないテストも、デザインなどのフォーマットに関するものが残るだけとなり、その絶対量は格段に減少する。

4.2　サンプルプログラムの構築

Webアプリケーションにファサードを適用してJUnitでテストする、という方法を検証するために図7のような構成でサンプル・プログラムを作成した。

プレゼンテーションとドメインのやり取りをファサードに仲介させて、ユーザー・インタフェースに関する機能とドメインに関するものが完全に分かれるようにした。

そして、ファサードに対してテストを作成してみたところ、このテストクラスでドメインに関するテストを網羅でき得ることが分かった。

もちろん、プレゼンテーションを別途テストしなければならない。しかし、JUnitを使わない場合と比べると、テストの手間には大きな差がある。ちょっとしたデバッグでもJUnitを利用しないと、そのテストは、いちいちブラウザを立ち上げてWebページを表示させ、フィールドのひとつひとつにデータを手入力しなければならない。JUnit上で走らせることのできるテストクラスがあれば、そのクラスを動かすコマンドをキーインするだけでテストが完了する。

5．今後に残された課題

今回の検証は、非常に簡易なサンプル・プログラムまでにとどまった。JUnitが本当に使えるものであるかどうかは、実際のプロジェクトで用いてみて、その適用性を把握する必要がある。その際のポイントとして以下が挙げられる。

(1)　他のテスト技法や開発ツールとの併用方法

これまで述べてきたように、JUnitは単体の、しかも直

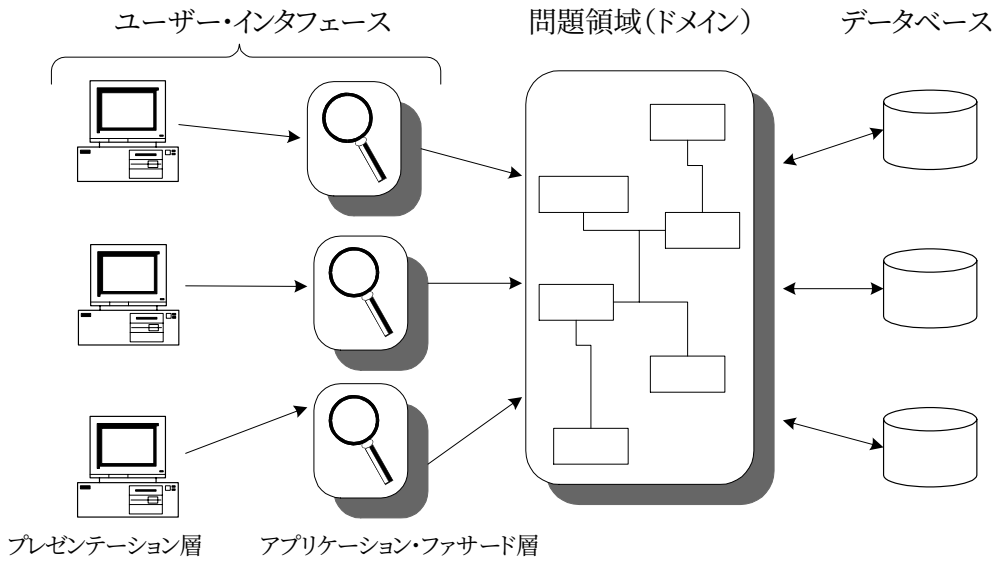

図6　Application Facades

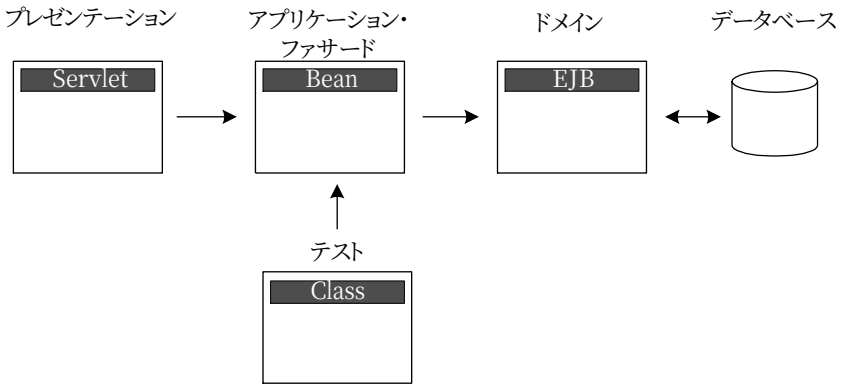

図7　サンプル・プログラムの構成

接的に実行できるJava言語のプログラムを対象としている。したがって、プレゼンテーションのテストは別途行う必要があり、さらに統合テストの方法も考慮しなければならない。単体テストの総まとめが、必ずしも統合テストにはならないからである。

また、一般的なシステム開発においては、何らかの統合開発ツールが用いられているのが現状である。そのため、これらのツールとJUnitを協調させながら、開発を効率的に進める方法を考え出す必要がある。

⑵ JUnitに適したアーキテクチャの提示

今回のサンプル・プログラムは、Application Facadesを採用することでJUnitの適合性が高くなったが、別のやり方が必要とされる場合もあるだろう。そこで、JUnitを実行するだけでアプリケーションのすべての機能をテストできるアーキテクチャを、はっきり示すことが求められる。

⑶ テストクラスの管理方法

プロジェクトのメンバーによって作成される多数のテストクラスは、テスト対象のクラスと同等の管理が必要である。アプリケーションのクラスを変更した場合にはテストクラスにも同じように手を加えなければならないし、パッケージングも正しくする必要がある。きちんと管理されていないテストクラスは、開発や保守の妨げとなってしまう。

また、テストクラス群が管理されていれば、１日１回の実施を推奨している回帰テストもスムーズに運ぶため、アプリケーションの品質向上にも貢献する。

６．おわりに

JUnit、およびWebアプリケーション開発へのJUnitの適用／検証方法について報告してきた。ご参考いただけただろうか。

なお、JUnitと協調してWebページをテストするため

のツールとして「HttpUnit[*3]」がリリースされている。詳しい調査はこれからだが、ブラウザ機能の何割かをエミュレートできるようであり、Web サイト全体の自動テストの実現が期待される。

今後は、JUnit をより大規模なシステム開発に用いながら、HttpUnit の調査も含めて有用性や回帰テストの実用性について検証を進めていきたい。

〈参考文献〉

1．I. ヤコブソン，M. クリスターソン，P. ジョンソン，G. ウーバガード著：西岡利博，渡邊克弘，梶原清彦監訳：『オブジェクト指向ソフトウェア工学 OOSE』，トッパン（1995）

2．石井勝著：『Kent Beck Testing Framework 入門』
http : / / member.nifty.ne.jp / masarl / article / testing-framework.html

3．Shel Siegel 著：古宮誠一，廣田豊彦監訳：『オブジェクト指向ソフトウェアテスト技法リスク管理への技術的アプローチ』，共立出版 bit 別冊（1999）

4．M. ファウラー著：堀内　一監訳：児玉公信，友野晶夫，大脇文雄訳：『アナリシスパターン再利用可能なオブジェクトモデル』，アジソン・ウェスレイ・パブリッシャーズ・ジャパン（1998）

5．Robert V.Binder 著：『Testing Object-Oriented Systems : Models, Patterns, and Tools』，Addison-Wesley（1999）

6．Martin Fowler, Kent Beck, John Brant, William Opdyke, Don Roberts 著：『Refactoring : Improving the Design of Existing Code』，Addison-Wesley（1999）

7．Martin Fowler 著：『Application Facades』
http : / / www.martinfowler.com / apsupp / appfacades.pdf

8．樋口節夫著：『サーバ・サイド Java プログラムのチューニング技法「月刊 JavaWORLD」2000年 9 月号』，IDG ジャパン

9．“ エクストリームプログラミング ”
http://ObjectClub.esm.co.jp/eXtremeProgramming/

10. “ Testing Framework ”
http : / / c2.com / cgi / wiki?TestingFramework

＊3）Http Unit : Web サイト「HttpUnit」（http : / / httpunit.sourceforge.net / ）を参照。