スクリプティング Java － Grails/Groovy による開発 －

上原潤二 †

JVM 上のスクリプト言語および Web アプリケーションフレークワークである Groovy/Grails を開発に適用することで，信頼性を犠牲にすることなく 2～3 割の生産性向上を達成することができた．本稿では事例適用を通じて得られた効果について分析し，直面した課題と対処について説明する．Groovy/Grails は，他のスクリプト言語と比して Java との親和性が非常に高く，Java 開発での経験やノウハウその他有形無形の Java 資産をそのまま活かすことができることを示した．

Scripting Java with Groovy And Grails

Uehara, Junji

We get 20% to 30% improvement of productivity without sacrificing of reliability through applying Groovy and Grails, which is a script language on JVM and web application framework, to our software development projects. We show the analysis to our case studies and issues we confronted. Compared with other languages or frameworks that runs on the JVM, Groovy and Grails have a high affinity for Java, and it arrows developer to get maximum leverage out of assets includes know-hows and skills on Java.

1. はじめに

NTT ソフトウェアは Groovy/Grails についての技術蓄積・普及活動を，書籍執筆やコミュニティ活動，社内外での研修等を通じて取り組んで来ており，2012 年 1 月から Grails 推進室を設置し取り組みをさらに本格化させた．本稿では Groovy/Grails について簡単に説明し，開発事例を紹介した上で適用のメリットと課題を示す．

2. Groovy について

Groovy[1]は Java 仮想機械上で動作する動的言語であり，文法が Java のほぼ上位互換である．冗長な記述(型指定，クラス定義，チェック例外記述等)を省略できること，メソッドを実行時に追加・解釈する動的言語機能，および強力な言語機能(Closure や Java 標準ライブラリへのメソッド単位での機能追加，AST 変換等)により，Javaと比較して 2～10 分の 1 程度のコード量での記述を可能とする生産性の高い言語である．

Java を主要な開発言語として使用してきた企業にとっては，既存 Java 資産(Java コードやライブラリ,AP フレームワーク,開発・運用ノウハウ)がそのまま利用可能であることが利点である．特に Java 経験者にとっては，学習コストが僅少で済むことが，他の JVM 上言語(Scala, JRuby, Clojure 等)と比しての利点となる．

Groovy は，高い生産性とともに，高い信頼性を同時に達成できる言語であり，信頼性向上にも寄与する以下のような機能・特徴を持つ．

- オプショナルな型付け(後述)
- AST 変換によるデザインパターンのライブラリ化
- 静的解析ツール Codenarc
- Groovy 2.0 の静的型チェック機能
- JVM のセキュリティモデルや信頼性を享受

3. Grails について

Grails[2]は，Groovy を使って開発する OSS の Web アプリケーションフレームワークである．Groovy と Grails の関係は，Ruby と Ruby on Rails の関係と同様である．Grails は Groovy のメリット(簡潔記述，内部 DSL††，動的性質)を活用するフレームワークである．

Grails は，Web アプリケーション開発に必要なフレームワークやライブラリを数多く組合せたフルスタックフレームワークであり，以下のような機能を持つ．

- 規定されたアプリケーション構造・設計
- DB をアクセスするための簡易 GUI を自動生成（Scaffold）
- プラグイン機構による多数（公式プラグイン 817 件）の既存コンポーネントの再利用
- Java EE(Java 基盤)が持つ安定性，スケーラビリティ，資産

† NTT ソフトウェア株式会社 Grails 推進室所属

†† 言語自身をカスタマイズして，処理対象概念を記述するための最適な記述ができるようにする．

をそのまま引き継ぐ

4. 適用事例と評価

2011 年度の弊社における Grails 適用事例を示す.

項番	プロジェクト名	顧客	工数削減効果	コード量比較(対 Java)††	**規模**
1	A システム	自社	詳細設計～結合テスト工程が, Java の約 1/2～1/5 の期間で開発	Java の約 1/10	50K L
2	B システム	自社	詳細設計～プログラム作成工程が, Java の約半分の工数で開発(開発全体で工数 2 割減)	未測定	未測定
3	C システム	受託	詳細設計～結合テスト工程まで, Java の約 1/7～1/10 の期間で開発	Java の約 1/10	20K L

上記適用を通じての評価結果は以下の通り.

- 全体としては2～3割の工数削減が見られる.
- 概要定義まで, および総合テスト工程以降の工数は従来と大きく変わらない. これは開発プロセスとして従来と同様のウォーターフォール開発を適用したためである.
- 結合テストまでの工程における削減効果は大きい.
- 詳細設計工程についての工数削減の理由は, アプリケーションの基本的な構造についての設計が不要になること.
- 少なくとも「Groovy/Grails に適したプロジェクトで採用すれば効果は大きい」と言える. 適したプロジェクトとは
  - データベースのレコード内容をCRUD(作成・表示・変更・削除)するための多数画面を含む
  - 開発前にすべての要件が十分に確定できず, インクリメンタルな機能拡張や変更が多い

以下に, 項番 2 のプロジェクトメンバによる評価を示す.

- 習熟が容易であった. Grails はリファレンスドキュメントがまとまっており, 基本的に Groovy/Grails の知識のみで開発を行うことができる. 従来であれば JavaEE 開発で必要となる複数の要素技術(Spring, JavaEE, Hibernate, RDMBS)すべてに習熟していないと開発が開始できなかった.
- Java スキルが活かせた. Java と言語仕様が共通しており, Java 部分の開発と並行して作業しても違和感が少ない.
- プロトタイプ開発に向いている. 画面・ロジック・DB 間の I/F は Grails で規定されており, これらに関する設計工数を削減できるとともに, 開発開始直後の段階からこれらを連携動作させることができる.

5. 課題

業務適用を通じて直面した課題は以下のとおり.

- IDE の成熟度が低い. 一般にはスクリプト言語では IDE の重要性は低いが「Java開発からの移行」を重視する場合 IDE の適用は必須となる. 代表的な IDE である Eclipse に対し VMware 社が Groovy/Grails サポートを提供している[3]が, Java に対するものと比べると完成度がやや低い. 代替として IntelliJ IDEA(Grails サポート機能に関して有償, Groovy サポートのみであれば無償利用可)を利用したが, 高機能かつ完成度が高かった.
- 型チェックの問題. 一般に動的型言語では変数に型がなく, また動的言語では動的クラス拡張(メソッド追加)が可能であるため, IDEでの自動補完やコンパイル時チェックがしにくいことが問題となる(特に大規模多人数開発時). スクリプト言語一般には, 単体テストを充実させることがこの問題に対する対処となり, Groovy/Grails でも試験サポート機能を多く持つ. 加えて Groovy では, Java と同様に型宣言を行うこともでき(「オプショナルな型付け」), コーディング規約で型情報を宣言することを決めることで, 多くの場合型エラーを回避できる††.
- 性能の問題. 実行性能は問題とはならなかったが, 起動時間は遅く低速マシンでは開発効率を阻害した.
- 「フルスタック」であることによる問題解析の困難さ.
- 日本語情報の不足. 多くの Java ベースの先進的なフレームワークについて共通するが, 日本語情報が少ない. Groovy については[4]などがあるが Grails については最新版に追随できる日本語書籍がない.

6. まとめ

JVM は安定した高性能プラットフォームであるが, Java 言語については冗長性によって発生する「変更の難しさ」が迅速な開発の支障となっている. かといって他の言語へ移行することは, Java 関連の資産(人的資源を含む)を捨てることを意味し大きな痛みを伴う. Groovy/Grails を用いることで, 漸進的な生産性の向上と Java 技術で培ってきた高信頼性の維持を同時に見込むことができる.

参考文献

[1] http://groovy.codehaus.org/

[2] http://www.grails.org/

[3] Groovy/Grails Tool Suite http://www.springsource.org/node/3594

[4] 関谷和愛,上原潤二,須江信洋,中野靖治, プログラミング Groovy, 2011, 技術評論社

†削減効果・コード量の比較対象は, 当社の見積りシステムによる予測

††2012 年 6 月にリリースされた Groovy 2.0 は静的型チェック(および静的結合ベースのコンパイル機能)を採用し, Java と同等のコンパイルチェックと性能向上が可能となり, IDE での補完も改善されることが期待される.