

Super Strong-Typed プログラミングによるデータ変換の自動化の試み

菊池彰†

日本アイ・ビー・エム株式会社ソフトウェア開発研究所

キーワード: Java, Strongly-Typing, Introspection, Reflection, プログラミング方法論

要旨: データの構成変換は、実際のプログラムでは頻繁に行われる作業であるが、単調な作業であるが故に誤りも生じやすい。Java プログラムにおいて、強く型付けしたデータと Reflection の機能を用いることでこの作業を自動化することができた。

1. はじめに

ある種のサービスを利用する際、そのサービスが想定しているモデルと利用する側のドメイン・モデルとが一致していない場合、データの変換が必要になる。

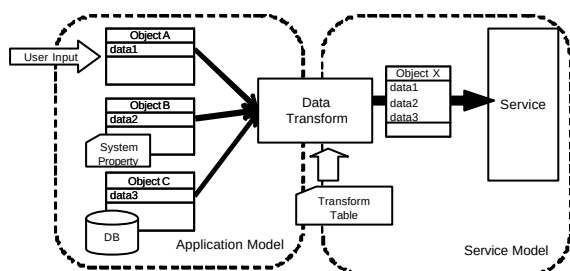


Figure 1 Data transformation for a service invocation

その例として Figure 1 において、Service 呼び出しのための Input である Object X は、ユーザ入力、システム・プロパティおよび Database のデータより構成されている。この類の変換は手作業で作られるか、別に定義された変換表を元に駆動するプログラムで処理されるが、強く型付けした変数と Java の機構を利用することにより、変換表を用いず自動化することが可能である。本稿では、上記変換作業の問題点と自動化手法について明らかにする。

2. データ変換の処理方法と問題点

Table 1 Example of Data Transformation

変換元	変換先 Object X		
	オブジェクト	Field	型
Object A		data1	String
Object B		data2	Date
Object C		data3	String

Table 1 のデータ変換を考えてみる。この表では、例えば変換先 ObjectX の data1 は、変換元 ObjectA の data1 から供給されるということを表現している。変換の処理方法としては、データの代入プログラムを手作業で記述する、もしくは変換表を用意して表に従い代入を行うという方法が考えられるが、どちらにも問題点が存在する。

An automatic method assigning data objects with super strong-typed programming model

†Akira Kikuchi, Software Development Laboratory - Yamato (YSL)

2.1. 手作業の問題点

Java は、コンパイル時に厳密な型分析をして、プログラミングエラーをキャッチできる[1]ため、Figure 2 の例のように Object X の String 型変数に Date 型である data2 を代入してしまうような誤りは、コンパイル時に発見することができる。しかし誤った代入であっても型が同じであれば、それをコンパイル時に発見することはできない。

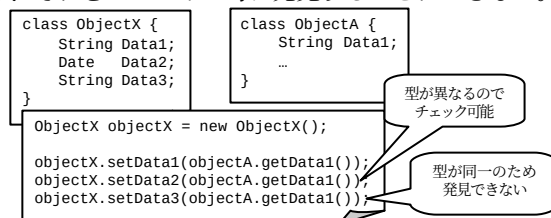


Figure 2 Type checking of illegal assignment

代入の数が少なければ十分注意してコーディングすることで誤りを回避することもできるが、数が多くなると手作業で誤りなく作業することが難しくなってしまう。

2.2. 変換表の問題点

変換表を用いる場合、データのマッピングがコードとは別のところに存在するため、変換表の妥当性をチェックする方法を別途必要とするという点が問題となる。またこの方法でも前節で問題とした、型が同じ誤った代入の検証を行うことができない。

3. Super Strong-Typed プログラミング

Super Strong-Typed プログラミングとは、String や Date などの汎用型を用いるのではなく、それぞれのデータ専用の型を用いてプログラムを記述する方法を指す我々の造語である。Figure 3 に例を示す。

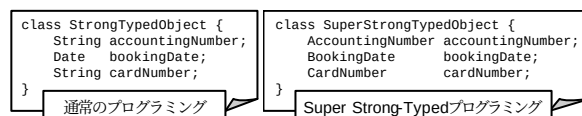


Figure 3 Super Strong-Typed Programming

Super Strong-Typed プログラミングでは、データの意味ごとにそれぞれ別の型を用意することになる。例えば同じユーザ ID を表す 14 桁の文字列であっても、一方は利用者 ID、他方は管理者 ID のように意味的に異なるものであるなら、別の型とする。

これにより、データが持つ意味同一性の問題をプログラミング言語の型検査という問題に置き換え、Java プログラムにおいて二つのデータが構文的に代入可能であるなら、両者の持つ意味が同じであることを保障する。

3.1. Super Strong-Typed クラスの作り方

Super Strong-Typed 型の実装方法としては格納されるデータ型を拡張するのが単純で良いと思われる。今回は Decorator パターン[2]を使うこととした。例えば Figure 3 の CardNumber クラスは、String を has-a 関係で持つ。

また、データの意味毎に大量のクラスが必要となるため、今回は IBM の alphaWorks[3]にて公開されている Design Pattern Toolkit (DPTK) [4]という Model Driven Architecture (MDA)を実現したツールを利用し、個々のクラスは宣言的な定義より自動生成した。

4. 自動データ変換

Super Strong-typed プログラミングにより、Figure 1 に示したデータ変換の問題は次の二つの型検索問題に分割できる。

1. ある型のデータを供給するオブジェクトはどれか
2. 変換先のデータはどの型を必要としているのか

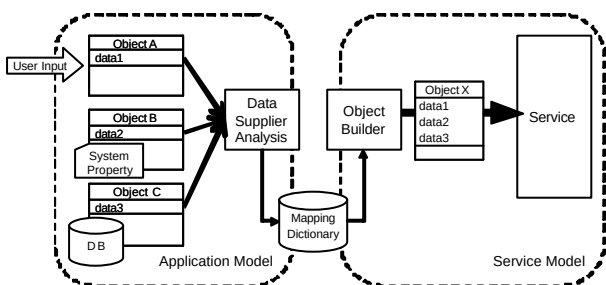


Figure 4 Automatic Data Transformation

Figure 1 の構成ではこれらは変換表として外部から与えられているが、Java 言語の Reflection[5]を使うことで Java プログラムとして記述することが可能となる。Figure 4 に、その構成を示す。

まず Data Supplier Analysis では、変換元の各オブジェクトのクラスが持つ get で始まるメソッドを検索し、その戻り型、メソッド名、変換元オブジェクトを辞書に登録する。

次に、Object Builder が変換先のオブジェクトのクラスが持つ set で始まるメソッドを検索し、そのパラメータ型を元に前述の辞書を用い、その型のデータを供給するオブジェクトとメソッドに関する情報を検索する。あとはそれぞれ getter メソッドと setter メソッドを Reflection により呼び出すことでデータを移行する。

5. 性能比較

比較方法は、m 個のデータを持つオブジェクト 2 個を結合して 2m 個データのオブジェクトを作成するデータ変換を 10000 回実行するのに必要な時間を直接代入する方式

と Super Strong-Typed プログラミングによる方式で測定した。m は [1, 120] の区間で変化させている。

測定環境は Pentium-M 1.3GHz PC 上で動作する WindowsXP である。JavaVM の Version は 1.4.2 で IBM 製のものを使用した。結果を Figure 5 に示す。

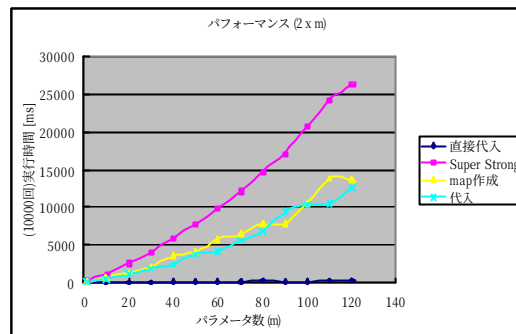


Figure 5 Performance Comparison

実行時間は常に 100 倍程度の差がある。また Super Strong-Typed プログラミングでは Supplier Map の作成とデータの代入の時間おおよそ 1:1 となっており、この比率は、型探索のために使用する java.lang.Class の getDeclaredMethods の使用比率と同じであった。

6. 結論

データの型データの持つ形式ではなくその意味を持たせる Super Strong-Typed プログラミングを行うことにより、データの持つ意味まで含めた処理が機械的に処理可能となる。また、Reflection API によりパフォーマンスは悪いが、手法的にソースコード量がデータ量に依存しない点と誤りが混入しないという点が利点として挙げられる。

当手法の本質はプログラムのセマンティクスをシンタックスの問題に変換できるという点にある。

強い型を持つデータを作るためには多数のクラスを作る必要があるが、DPTK のようなツールを活用することでプログラミング作業を大いに軽減できることがわかった。

謝辞

Super Strong-typed プログラミングの名称と自動変換のアイデアに関してソフトウェア開発研究所のメンバーに貴重な助言をいただきました。各位に感謝いたします。

参考文献

- 1) Guy L. Steele Jr.: Java 言語仕様設計上の考慮点, 情報処理, vol.39, No.4
- 2) Erich Gamma 他: オブジェクト指向における再利用のためのデザインパターン(改訂版), ソフトバンクパブリッシング, Japan (1999), pp187-196.
- 3) IBM alphaWorks, (<http://alphaworks.ibm.com/>)
- 4) Design Pattern Toolkit, (<http://alphaworks.ibm.com/tech/dptk>)
- 5) Reflection, Java 2 SDK, Standard Edition Documentation, (<http://java.sun.com/j2se/1.4.2/docs/guide/reflection/index.html>)