プログラミング言語 MixJuice による HTTP server のモジュール化

Modularization of HTTP server with programming language MixJuice

田中 哲 †
Akira TANAKA
akr@m17n.org

一杉 裕志 †‡
Yuuji ICHISUGI
y-ichisugi@aist.go.jp

† 産業技術総合研究所
National Institute of Advanced Industrial Science and Technology
‡ 科学技術振興事業団 さきがけ研究 21
PRESTO Japan Science and Technology Corporation

従来のオブジェクト指向言語ではクラスという構文がモジュールの役割を兼ねているため、モジュール化の自由度が制限されるという問題がある。我々はこの問題を解決するため、クラスではなく差分をモジュールとして扱うプログラミング言語 MixJuice を提案した。本論文では、 Java 言語で書かれた Jasper というプログラムを MixJuice を用いて書き直すことにより、 MixJuice のモジュール機構の有効性を評価する。Jasper は Java 言語で書かれた約 1 万行のオープンソースの HTTP server であり、もともと高い拡張性と小さいサイズという特徴を有している。Jasper を MixJuice に移植し、機能選択機構を付加することにより、拡張性と必要最小サイズという特徴がオリジナルよりも向上することを示す。

1 はじめに

従来、オブジェクト指向プログラミング言語においてはクラスがモジュールとして使われてきた [1]。本稿ではこれを**クラスベースモジュール機構** *(class-based module system)* と呼ぶ。本来、クラスはひとつのデータ構造に関連するコードをまとめるという役割を担っている。しかし、データ構造よりも機能という観点からコードをまとめたほうが適切な場合もある。そのような場合、クラス機構を使って機能に関連するコードをまとめるにはデザインパターン [2] などの技巧を凝らす必要がある。しかし、そのような技巧を凝らすと、技巧を凝らしただけコードが複雑になり、記述性・保守性が低下する。

我々はそのような問題を持つクラスベースモジュール機構のかわりに**差分ベースモジュール機構** *(difference-based module system)* を提案しており、Java に差分ベースモジュール機構を導入した MixJuice[3] という言語を開発している。差分とはシステムに加える変更点をまとめたものであり、例えば既存のクラスへメソッドを追加することなどができる。

差分ベースモジュール機構では技巧を凝らさなくてもモジュール化が可能なため、同じような分割を行なう場合にもクラスベースモジュール機構にくらべ技巧が必要ないだけ理解しやすく、保守しやすいシステムを構築できる。また、クラスベースモジュール機構よりも細かくモジュールを分割することによって、再利用性を高めることができる。MixJuice ではこの差分ベースモジュール機構により、モジュールはクラスとは直交しており、クラスに縛られずにモジュール化を行なえる。

また、MixJuice ではモジュールを選択することによってさまざまな機能構成のアプリケーションを生成することができる。このようにユーザが機能構成をカスタマイズするための機構を**機能選択機構**と呼ぶ。機能選択機構は cpp の `#ifdef` や Strategy パターンなどでも実現可能であるが、MixJuice の差分ベースモジュール機構を使うとより容易に実現できる。

我々は差分ベースモジュール機構の有効性を示すため、Jasper[4] という Java で記述された HTTP server を MixJuice に移植し、機能選択機構を付加した。その結果、最低限のモジュールだけを選択してアプリケーション (サーバ) を構成することにより、必要最小サイズが 1/2 に減少するという有効性が確

認できた。

以下、2 節で MixJuice のモジュールについて概説し、3 節で Jasper のアーキテクチャについて述べる。4 節で MixJuice によって Jasper に機能選択機構を追加したことを述べ、5 節でその結果を評価する。6 節では機能選択機構を実装するための他の手法について述べ、7 節では差分ベースモジュール機構の欠点について述べる。8 節ではまとめを述べる。

2 MixJuice のモジュール

MixJuice は Java をベースとして、Java のクラスベースモジュール機構のかわりに差分ベースモジュール機構を導入した言語であり、次の特徴を持つ。

- クラスとは直交した**差分** *(difference)* を単位とするモジュール機構を提供
- リンク時ないしはプログラム起動時に、使用するモジュールを自由に選択可能
- 名前の衝突を完全に防止
- モジュール毎に型検査・分割コンパイル可能

差分ベースモジュール機構ではモジュールはクラスではなく差分である。MixJuice で実現されている差分ベースモジュール機構の場合、差分 (モジュール) の追加によってシステムに以下の変更を加えることができる。

- 新しいクラス・インターフェースを導入する
- 既存のクラス・インターフェースに新しいフィールドを導入する
- 既存のクラス・インターフェースに新しいメソッドを導入する
- 既存のクラスの既存のメソッドを拡張する
  (既存の実装は `original()` で呼び出し可能)

例えば、図 1 はモジュール A とモジュール B を結合することにより、クラスやメソッドが追加された S というシステムを構成できることを示している。

3 Jasper のアーキテクチャ

Jasper はすべてのコンテンツを servlet によって提供するように設計された servlet ベースのシンプルで軽量・コンパクトなオープンソースの HTTP server である。また Java により約 1 万行で記述されており、

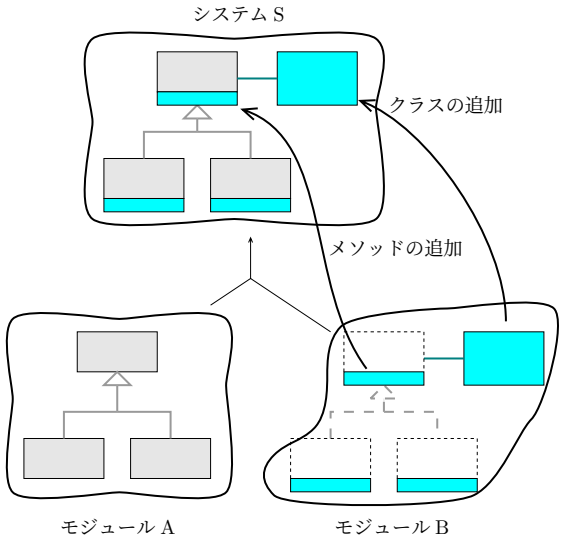

図 1: MixJuice によるモジュールの追加

サーブレットライブラリ[1]を利用している。Jasper は拡張性を意識して設計されており、設定の指定法、ログの取り方を用途に応じて選択することができる。設定の指定としては XML による指定とプロパティによる指定が提供されており、ログの取り方としてはファイルに記録するものと標準出力に出力するものが提供されている。この は選択機構は Dynamic Loading 機構 (`Class.forName`) を利用しているため、再コンパイル無しに他の機能を実装し、利用することもできる。

ただし、細かい機能まで含めて全ての機能を選択可能なように実装するのは現実的ではない。これは選択機構自体の実装に手間がかかる上、ソースが複雑化するためである。そのため、Jasper では利点の大きなもののみをいくつか選んで選択可能にしており、servlet 機構、BASE64 デコーダなど、Jasper に組み込まれて取り外せない機能も多数存在する。

4 MixJuice による機能選択機構の追加

オリジナルの Jasper のコンテンツは servlet によって提供される。我々はコンテンツ提供機構を選択可能とすることにより、servlet 以外の選択肢も選べるようにした。

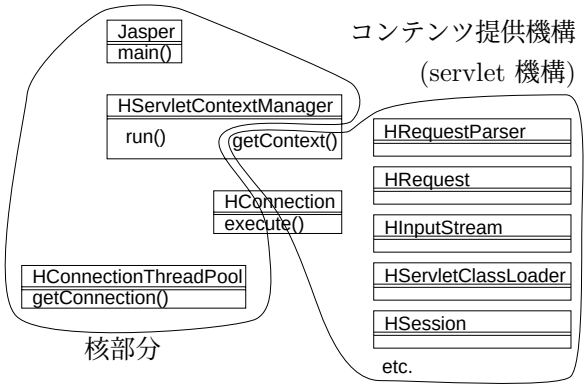

図 2: サーバの核部分と servlet 機構のクラス

4.1 Jasper の servlet 機構

Jasper は図 2 のような構造になっており、accept loop を実装する HTTP サーバの核部分とコンテンツを提供する servlet 機構が 2 つのクラスで混在している。つまり、コンテンツ提供機構は核部分と分離できていない。Jasper の servlet 機構を分離し、コンテンツ提供機構を選択可能とするためには HServletContextManager というクラスが問題になる。このクラスにはクライアントからのコネクションを受け付けて個々の処理を起動する accept loop を実装した run() と、servlet の設定情報を得る getContext() が混在している。しかし、これは Java で記述する限り妥当な設計である。なぜなら、Jasper の設定において accept loop の設定とその accept loop から起動する servlet の設定はまとめて記述されるため、データ構造の観点から見るとひとつのクラスに含めることが自然だからである。

4.2 MixJuice による servlet 機構の分離

MixJuice の差分ベースモジュール機構ではモジュールはクラスと直交しているため、クラス構造は変えずにモジュール分割を行なえる。

Jasper の核部分と servlet 機構のコラボレーション図を図 3 に示す。つまり、核部分の処理と servlet の処理は機能として別れており、機能の観点からモジュール分割を行なうと servlet 機構を分離できる。そして、MixJuice ではモジュールとクラスは直交しているため、図 3 の構造をそのまま表現できる。つまり、核部分と servlet 機構を別個のモジュールとして定義しつつ HServletContextManager と HConnection

[1] servlet-2.2.jar: http://www.euronet.nl/ pauls/java/servlet/

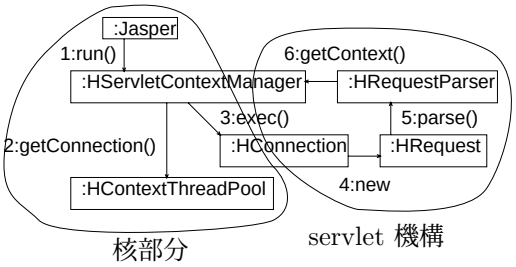

図 3: サーバの核部分と servlet 機構

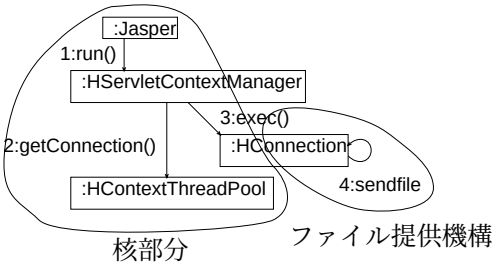

図 4: サーバの核部分とファイルを提供する機構

というクラスは両方のモジュールにまたがって定義し、個々のメソッドをそれぞれが属するモジュールで定義できる。

servlet 機構をモジュールに分離すると、コンテンツ提供機構が選択可能になる。例えば、ファイルを静的なコンテンツとして提供するモジュールを実装することができる。そのようなファイル提供機構と核部分を組み合わせた例を図 4 に示す。この例ではファイルを提供するコードはすべて HConnection 内で実装され、新しいクラスは導入していない。

5 評価

Jasper を MixJuice に移植して servlet 機構を分離し、コンテンツ提供機構を選択可能とした。また servlet 機構の代わりとなるコンテンツ提供機構として、ファイル提供機構を実装した。そして 3 種類の構成についてコンパイル結果のクラスファイルのサイズを測定した。

なお、MixJuice に移植するベースは Jasper 0.4.0.b2 を用いた。また、Jasper 自身にも Dynamic Loading 機構を利用した機能選択機構があるが、ここでは XML による設定、ファイルに対するログ出力を選択した。

Java 版: オリジナルの Jasper は約 280KB になる。

MixJuice 版 (servlet 機構): Jasper を MixJuice

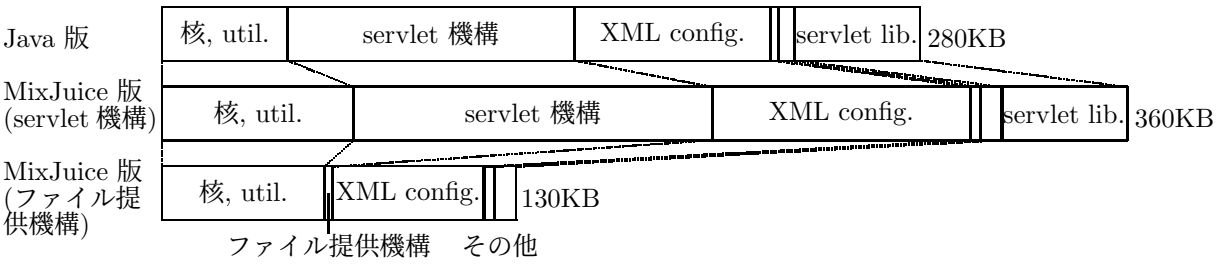

図 5: MixJuice 化による Jasper のクラスファイルのサイズの変化

に移植し、コンテンツ提供機構のための機能選択機構を実装した場合、約 360KB になる。Java 版よりサイズが増加しているのは、主に MixJuice のコンパイラが多くのクラスを生成し、個々のクラス毎にコンスタントプールが大きなサイズを占めるためである。

MixJuice 版 (ファイル提供機構): コンテンツ提供機構として servlet 機構の代わりにファイル提供機構を選択した場合、約 130KB になる。このとき、servlet 機構がなくなるだけでなく、サーブレットライブラリが不要になっている。

クラスファイルのサイズの測定結果をモジュール毎に集計したものを図 5 に示す。ファイル提供機構を選択したサーバはオリジナルの Jasper の 1/2 のサイズで構成できている。

コンテンツ提供機構としてファイル提供機構を選択した場合、servlet 機構を選択した場合よりも、核部分、XML の設定ファイルを解釈する部分のサイズが減少している。これは、それぞれが複数のモジュールから構成されており、依存関係にしたがって必要最小限のモジュールだけがリンクされるためである。例えば、XML の設定を解釈するモジュールのうち servlet 機構のための設定項目を扱うモジュールはファイル提供機構を選択した場合にはリンクされない。MixJuice では依存関係が単純になるため、このような減少が顕著に起きる。

6 他の機能選択機構の実現手法との比較

機能選択機構は差分ベースモジュール機構で容易に実現できるが、それ以外の技術でも実現可能である。本節ではそれらについて述べる。表 1 はそれらをまとめたものである。

6.1 プリプロセッサ

C 言語の cpp のようなプリプロセッサは条件コンパイル機能 (`#if, #ifdef`) があり、これは機能選択機構として利用できる。しかし、機能選択はコンパイル時に単一ファイルの中で行なわれるため、選択肢を追加するにはソースに手を加える必要があり、選択をやり直すだけでも再コンパイルが必要になる。

6.2 デザインパターン

デザインパターンとは OOP において頻繁に利用可能な再利用性の高い設計のことである。デザインパターンは経験豊富な設計者が行なう再利用性の高い設計のやりかたを文書化したものであり、初心者が再利用性の高い設計を行なうことを支援する。

デザインパターンによる機能選択機構は、例えば Strategy パターンによって実現されるが、動的に選択が行なわれるため、静的な型検査ができずに down cast が必要になる場合がある。また、Dynamic Loading 機構を併用しない限り、新たな選択肢を追加するにはソースの変更と再コンパイルが必要になる。

6.3 AOP

AOP (Aspect Oriented Programming)[5] はプログラムを複数の観点からみた側面 (アスペクト) に分割し、それぞれに関する記述を行ない、それらの記述を合成することによってプログラムを構成するというパラダイムである。AOP を支援するプログラミング言語としては Java をベースとした AspectJ などがある。

AspectJ ではプログラムをコンパイルするときに本来のコンパイル対象のクラスに加えて組み合わせるアスペクトを指定でき、コンパイル結果はそれらが統合されたものとなる。また、各アスペクトは実行中に有効・無効を切替えることができる。

表 1: 機能選択機構の実現手法

機能選択機構	cpp (プリプロセッサ)	Strategy パターン (デザインパターン)	AspectJ (AOP)	MixJuice (差分ベース モジュール機構)
機能選択	コンパイル時	実行時	コンパイル時・実行時	リンク時
静的な型・整合性検査	○:可能	△:要 down cast	○:可能	○:可能
機能毎の分割コンパイル	×:不可能	○:可能	×:不可能	○:可能
不要なコードのリンク	○:されない	×:される	○:されない	○:されない
バイナリへの選択肢追加	×:不可能	○:DL により可能	△:再コンパイルが必要	◎:容易
実行効率	◎	△:動的な dispatch	△:実行時の制御は遅い	○

AspectJ ではアスペクトはクラスの一種として記述する。そのため、アスペクト間でコードを移動する場合には this の調整など細かな変更を行なわなければならない。このため、モジュールの切り分けを繰り返し試行するという用途には向いていない。

7 差分ベースモジュールの欠点

差分ベースモジュールによる記述では、1 つのクラスのコードが複数のモジュールに分散するため、再利用性・拡張性と引き換えにソースコードの可読性が低下する場合がある。同様の現象は従来のオブジェクト指向言語においても起きる。つまり、1 つの手続きが複数のクラスに分散したり、1 つのオブジェクトの振舞の記述がスーパークラスとサブクラスに分離すると、可読性が低下する。従来のオブジェクト指向言語と同様に MixJuice においても、可読性の低下を UML などを使った外部ドキュメントで補うことが重要である。

もう 1 つの問題は、複数の差分を組み合わせたときの衝突の問題である。今回は全体を把握する 1 人のプログラマが選択可能なモジュールを実現したために問題は起きなかったが、複数のプログラマが独立にモジュールを実装する場合、それらを組み合わせた時に正しく動作しない可能性がある。この問題については、契約主導設計 [6] の徹底と表明検査機構による実行時検査により、対処しなければならない考えている。

8 まとめ

Jasper を MixJuice に移植したことにより、容易に機能選択機構を追加することができた。これにより、用途に必要な機能だけをもった HTTP server を構成できるようになった。例えば、小規模システムのためにコードサイズを最小にするように構成した場合は 130KB になり、これは Jasper 本来の 1/2 のサイズである。

このようにコードサイズが減少することはソフトウェアの頑健性の向上の保証も意味する。なぜなら、不要なモジュールはリンクされないため、それらのモジュールに混入している間違いが影響しないことが保証されるためである。また、差分ベースモジュール機構によって観点に即したモジュール化が可能となり、個々のモジュールを小さくできる。このため、プログラマがモジュールの挙動を把握しやすくなり、間違いが入り込む可能性が減少してソフトウェアの頑健性が向上する。

参考文献

[1] Szyperski, C. A.: ”Import is not Inheritance – Why We Need Both: Modules and Classes.”, In Proc. of ECOOP’92, LNCS 615, pp.19–32, 1992.

[2] Gamma, E., Helm, R., Johnson, R. and Vlissides, J.: *Design patterns: elements of reusable object-oriented software*, Addison-Wesley professional computing series, Addison-Wesley, Reading, MA, USA, 1995.

[3] 一杉裕志: シンプルかつ強力なモジュール機構を有するオブジェクト指向言語 MixJuice の提案, **日本ソフトウェア科学会第 *17* 回全国大会論文集**, `http://staff.aist.go.jp/y-ichisugi/mj/`, 日本ソフトウェア科学会, September 2000.

[4] OpenJE: Jasper, http://www.openje.org/jasper/

[5] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C. V., Loingtier, J.M. and Irwin, J.: ”Aspect-Oriented Programming.” Invited Talk. In Proc. of ECOOP’97, LNCS 1241, pp.220-242, 1997.

[6] Meyer, B.: *Object-Oriented Software Construction, 2nd Ed*, Prentice-Hall, Englewood Cliffs, NJ 07632, USA, 1997.