リフレクションとオブジェクト資源の構造化を組み合わせたJavaの動的なモデル構成

秦野 亮†　　西山 裕之†

†東京理科大学理工学部

1　はじめに

近年発達したオブジェクト指向分野のソフトウェア開発手法に, DI(Dependency Injection) コンテナを用いることでプログラム実行時に目的とするシステムの処理モデルを組み立てる手法がある [1]. 旧来のコンテナを用いた手法と異なり, DI コンテナはリフレクション [2] と設定ファイルを利用する事でオブジェクトの生成や組み合わせの管理を行う. これは, 個々の部品の再利用性が上がるといった効果がある.

一方, DI コンテナは目的とするシステムを構成するために必要なオブジェクトの依存性の記述に, XML など静的に記述された設定ファイルを利用している. そのため, システムの状態は予め決められたライフサイクルに従ったものとなる. DI コンテナが普及した背景として, 近年高まるソフトウェアの需要がある事から, 今後システムをより早く構成し, 柔軟に拡張を可能としていく事はますます重要になると考えられる. しかし, そのためには予めシステムに対するあらゆる変更を見込んだ高度な設計が必要となる.

そこで, 本研究は DI コンテナが用いる処理モデル構築の仕組みを参考に, 稼動中のシステムに対して直接の処理モデルの構築・変更を可能とする手法を提案する. これにより, 予めシステムに対するあらゆる変更を見込んで設計を行わなくても稼働中のシステムを停止せずに処理モデルの変更を行う事が出来ると考えた.

2　設計

はじめに, 本研究の実証に用いたシステムの概要を図 1 に示す. 本システムは, 稼働中のプログラムのオブジェクトが持つフィールドを辿り, 辿った先のオブジェクトが持つメソッドの実行や値の書き換え等を対話的に可能にする環境を提供する. 実装には, 各種 OS 上で利用する事が可能な Java 言語を用いた. 図 1 にはシステムコール, ファイルシステム等の名称が並んでいるが, これらは OS のシステムを参考に構築を行ったためこのような名称がついている. 実際には Java 仮想マシン内でのみ用いられる仕組みの名称である.

仮想ファイルシステムは処理モデルを操作する際のオブジェクトの管理に用いられ, プロセス実行・管理機能はオブジェクトの操作に伴うスレッド等の管理に用いられる. また, これらの機能には静的領域に確保されたシステムコールを通じてクライアントからアクセスを行う. クライアントは, OS のシェルの様な対話型コンソールインターフェースや, 一度に多数の入力を扱えるバッチ処理を行うプログラムであり, システムコールへ適切な文字列の出力を行う. クライアントから出力された文字列は, システムコール・補助ライブラリを通じる過程で, 入力の文字列は適切なオブジェクトに変換される. このとき, 仮想的なファイルシステム上のオブジェクトやクラスパス上に存在するクラスの機能呼び出しを, ラッパーコマンドと呼ばれる機能によって組み込みの機能 (コマンド) と同様に扱うための処理が行われる. これにより, 本システムでは組み込みの機能と処理モデルを組み立てるための部品が持つ機能は同様な入力で利用が可能となる.

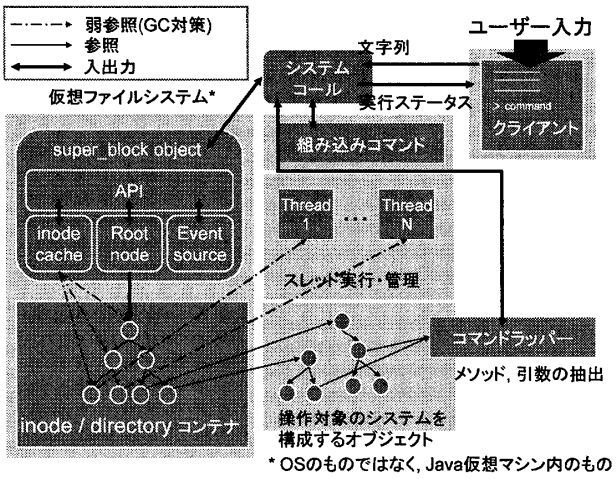

図 1: システムの概要

システムコールを通じた処理の出力は仮想ファイルシステム上のオブジェクトに対して行われ, それらの処理が成功したかどうかはステータス番号としてクライアントに出力される.

2.1　仮想ファイルシステム

本システムは, 処理モデル構築中のオブジェクトを管理するための仕組みとして, ツリー構造のインメモリの仮想的なファイルシステムを持つ. これにより, 処理モデルを操作する過程でのオブジェクトの生成・除去, フィールドを辿った際に得られた参照を階層構造として管理可能になり, URL の様なパスの指定によってアクセスすることが可能となる.

ツリーを構成するノードは, 処理モデルを構成するためのオブジェクトと関連情報を格納・操作するためのコンテナである. ノードとなるコンテナには, ツリーの末端を構成するオブジェクトへの参照を持つ inode コンテナと, オブジェクトへの参照を持たないものの複数の inode コンテナへの参照を持つ directory コンテナの 2 種類がある. なお, inode コンテナは参照するオブ

The dynamic model building of Java which combined reflection and structurized object resources.
Ryo Hatano†, Hiroyuki Nishiyama†
†Faculty of Sci. and Tech. Tokyo Univesity of Science

ジェクトに関係が深いフィールドなどのオブジェクトへの参照を格納した複数の inode コンテナを directory のように保持可能である．よって，通常のファイルシステムとは構造が異なりオブジェクトのフィールドを再帰的に辿る事が可能である．

2.2 ラッパーコマンド

本システムでは，処理モデルを操作するための組み込みの機能をコマンドとして実装している．全てのコマンドは，デザインパターンにおけるコマンドパターン [3] の仕組みを利用したものとなっており，共通のインターフェースを実装している．しかし，実際の用途では予め組み込んだ機能以外に，仮想ファイルシステムや Java 仮想マシンが管理するクラスパスから取得可能なオブジェクト (クラス等) が固有に持つ機能 (メソッド，コンストラクタ等) を利用したい場合がある．

そこで，クライアントからの入力文字列からリフレクションによってオブジェクトの実行対象メソッドを抽出し，引数にあたる文字列を仮想ファイルシステム上のオブジェクトや直接オブジェクトに変換するといった機能を持つラッパーコマンドを実装した．また，仮想ファイルシステム上の特定のオブジェクトが持つフィールドの読み書きを直接行うコマンドを同様に実装することによって，本システムは組み込みのコマンドと同様にオブジェクトの持つデータやメソッドを直接操作可能とした．

2.3 コマンドの実行

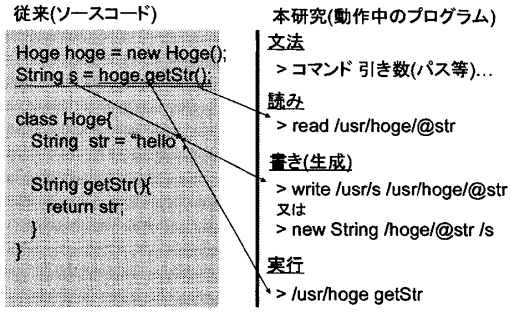

図 2: 入力例

クライアントはシステムコールに図 2 の右に示す様な入力文字列を渡すことでコマンドの実行を行う．入力文字列は空文字で分割され，第一引数はコマンドの識別に用いられる．残りの分割された文字列は引数として各コマンドに直接渡される．

第一引数が組み込みのコマンドを示していた場合は，共通のインターフェースを通じてコマンドを実行する．そうでない場合はオブジェクトの取得を仮想ファイルシステム，クラスパスの順に試み，オブジェクトが得られた場合はラッパーコマンドに取り込むことでコマンドとしての実行を試みる．実行されるコマンドはエラーによってコマンドを呼び出したスレッドが異常終了する事を防ぐため，全て新しいスレッドで行われる．コマンドを呼び出したスレッドは入力されたコマンドのオプションによって呼び出したコマンドの終了を待つか，そのまま並行動作させるかを選択することが出来る．

また，新たに生じたスレッドはコマンドの実行直前にプロセスと呼ぶオブジェクトに格納され，仮想ファイルシステム上に配置される．このオブジェクトを利用することで，コマンドを実行しているスレッド ID に対応するプロセスオブジェクトから，実行中のコマンドオブジェクトやスレッドそのものへアクセスし，制御に利用する事が可能となる．

3 実証実験

本研究の有効性を確認するために実証実験を行った．実験には TCP/IP で通信を行うクライアント・サーバープログラムを用いた．本システムを通じて通信中のクライアントプログラムの状態を直接書き換える事で接続先の変更を行い，その後の正常な通信を確認した．

実験ではサーバープログラム (以下，サーバー) を 2 つ起動し，それぞれ異なるアドレスを割り当てた．クライアントプログラム (以下，クライアント) は予め片方のサーバーに接続を行って待機をしている．そこで，本システムからクライアントの入り口となるオブジェクトを通じて接続先の情報を格納するオブジェクトまで辿り，もう片方のサーバーのアドレスを指すようにオブジェクトの書き換えを行った．その後，クライアントが実装している通信の切断・接続を行うメソッドを同様にオブジェクトを辿ることで起動し，接続先が正しく変更され，正常な通信が行われていることを確認した．

これにより，他のプログラムへの入り口となるオブジェクトを通じて，実行中の状態を直接操作し徐々に改善しながら開発を行うといった事が可能となる．

4 おわりに

本研究は，稼働中のシステムに対して直接の処理モデルの構築・変更を可能とするために，オブジェクトの管理をするための仮想的なファイルシステム，オブジェクトが固有に持つ機能を組み込みコマンドと同様に扱うラッパーコマンド，プロセスオブジェクトを用いたコマンドの実行と管理の仕組みを組み合わせた手法を提案し，実装した．

本システムは，稼働中のシステムを構成するオブジェクトのフィールドを辿ることで，各オブジェクトの読み書きや，予め本システムに組み込まれているわけではないオブジェクト固有の機能をコマンドとして呼び出すことが出来る．これは，DI コンテナが行っている基本的な処理モデルの構築に相当することを稼働中のシステムに対しても適用ができることを示している．この事から，稼働中のシステムの状態を確認しながら対話的に改善を行う開発や，デッドロックなど想定外の状態に陥った稼働中のシステムを停止せずにリカバリするといった用途などへの応用が考えられる．

また，本研究で提案したシステムは，オブジェクトに対して直接の操作を行う部分にリフレクションを利用した以外に，デバッガなどの特別な仕組みを一切用いていない．よって，オブジェクトやリフレクション相当の機能を持つ環境であれば，同様の仕組みを利用・実装可能であると考えられる．

参考文献

[1] Martin Fowler, "Inversion of Control Containers and the Dependency Injection pattern", http://martinfowler.com/articles/injection.html

[2] 千葉滋, 立堀道昭, 佐藤芳樹, 中川清志, "リフレクションの高速化技術", コンピュータソフトウェア, Vol.21, No.6, pp.5-15, 2004.

[3] Erich Gamma, Ralph Johnson, Richard Helm, John Vlissides, 本位田 真一 訳, 吉田 和樹 訳, "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", ソフトバンククリエイティブ, 1999.