Javaにおけるクラス定義の動的変更可能システムの構築

4L−1　野上耕介　山口実靖　相田仁　齊藤忠夫

東京大学工学部

1　はじめに

現在のネットワーク管理システムにおいて、管理システム開始時に想定したサービスをより円滑にするため、その運用中に変更したり、新たな機能を追加したりする場合には、次のような問題がある。

- 管理システム全体が停止するため、変更対象以外のシステムに支障がでる。
- 再起動する際、データを外部に書き出す必要がある。

今回、システムがもつ機能をシステムを停止させることなく、動的に変更することが可能となるシステムを提案する。提案するシステムを使用すれば、アプリケーションの動的機能拡張という形式でのアップグレードといったことも可能になる。

2　Java言語における実装

Javaアプリケーションは、オブジェクト指向とよばれる概念に基づいて、モデルを設計し、クラスと呼ばれる単位で構成される。今回は、動的に機能を変更可能なシステムをJava言語で構築する。Java言語でシステムを構築すると言うことは、つまりシステムを構成するクラス定義を動的に変更することができるということである。

2.1　実現方法

クラス定義を変更する可能性のあるクラスのインスタンスを作成する際、提供されるラッパークラスを用いて作成する。このラッパークラスは、内部にメソッド情報を格納したテーブルを保持し、ユーザはそのラッパークラスに向かって、メッセージを投げる。ラッパークラス側では、その投げられたメッセージをメソッド情報テーブルと照合して、内包するインスタンスに対してメソッドを起動させる。クラスを交換する際には、メソッド情報格納テーブルを書き換えることにより、クラス定義の変更が可能になる。また、フィールド[1]の追加なども可能である。(図1)

2.2　提案システムとインタフェースの比較

インタフェースを利用しても、クラスの振舞いを動的に変更することは可能である。しかし、インタフェース自体の動的な変更はできないため、インタフェースに記述されていないメソッドの起動は不可能である。しかし、提案するシステムでは、クラス交換の際、新たなメソッドを追加することが可能である。

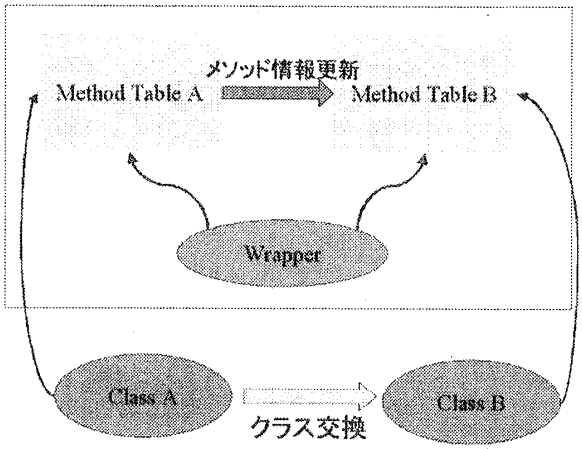

図 1: 概念図

2.3　実現できている機能

- フィールドの継承

  クラスの交換 (= インスタンスの交換) の際、インスタンスの持つフィールドを自動的に継承可能である。ただし、提供するライブラリの package の問題から public なクラス変数のみが継承可能となっている。

- イベント処理

  イベント処理に関しては、Delegation Event Model を採用する限りにおいては、リスナークラスの変更も可能となっている。

- スレッドに関する処理

  動作中のスレッドの振舞いを変更する際は、自動的にスレッドを停止し、交換した後、自動的に新たなクラス定義に従って、再起動する。ただし、ローカル変数などの情報は完全に失われる。

2.4　現段階における制約および性能、特徴

現在の実装段階で、アプリケーションを作成する際、以下のような制約がある。

- 交換するクラスは、そのクラスを継承する必要がある。これにより、変更したいメソッドの振舞いのみを変更するだけでよいという、オブジェクト

[1] C言語でのメンバ変数に等しい

Dynamic exchange of class definition with Java

Kousuke Nogami, Saneyasu Yamaguchi,
Aida Hitoshi,Tadao Saito
Faculty of Engineering,The University of Tokyo

	set メソッド	get メソッド
ノーマルシステム (ms)	8	7
提案システム (ms)	1974	1719

表 1: 計測環境:JRE 1.2 JIT Enterprise 450 (4CPU) Memory 1024M メソッド起動回数 100000 回

指向の恩恵を享受することができる。また、クラスを交換する際、交換前のクラスを継承していないとそのクラスの交換は出来ないように設計してある。

- メソッド呼出しの際、メソッド情報格納テーブルを検索し、リフレクション機能を利用しているため、通常のメソッド呼出しより時間を要する。このため、現段階では、速度を重視するようなアプリケーションには適していない。(表 1)
- メソッド呼出しに、リフレクション機能を利用しており、引数をオブジェクト型の配列にする必要があるなど、メソッド呼出しがシームレスではないため、プログラマに負担がかかる。また、返値が Object 型であるため、明示的にキャストする必要がある。
- インタフェースの概念の利用ができなくなっている。
- フィールドは継承できるが、ローカル変数は継承できない。

2.5 実装の具体的な中身

提案するシステムを構築するために作成したクラスを簡単に列挙する。

- TargetClass.class アプリケーションで作成されるインスタンスのクラスのラッパー機能を提供。
- EventClass.class イベントリスナーのインスタンスをこのクラスでラップする。
- ClassTable.class クラスの交換を行う際に必要となる情報を管理するテーブル。
- MethodTable.class TargetClass の中で利用されるクラスで、メソッド情報を格納する。
- Accede.class インスタンスごとのクラス変数の受渡しを行うための機能を提供。
- AccessServer.class 上記に挙げたクラスなどを包括し、プログラマが利用しやすいような API を提供。

2.6 サンプル

以下に、簡単なサンプルコードを載せる。クラス A があったとする。まず、クラス A を使ってアプリケーションを構築するが、それは、Sample のように、TargetClass.class のインスタンスとして、クラス A を作成する。このインスタンスに対して、メソッドを起動させる。クラス A をクラス B に交換させたいとすると、exchange メソッドを使用して交換します。それにより、deposite メソッドの振舞いが変更され、新たに追加された draw メソッドの利用も可能になる。

```
public class A{
    public int money = 10000;
    public void deposit(int plus){
        money += plus;
    }
}

public class B extends A{
    public void deposit(int plus){
        money = (money + plus) * 1.05;
    }
    public int draw(int minus){
        money -= minus;
        return money;
    }
}

public class Sample{
    public static void main(String args[]){
        AccessServer as = new AccessServer();
        TargetClass target = as.make("A");
        target.ex("deposit",1000);
        as.exchange("A","B");
        target.ex("deposit",2000);
        target.ex("draw",3000);
    }
}
```

2.7 実装予定の機能

今後、実装を予定している機能を述べる。

- 新しいクラスに交換した際のテストランのような機能の実装。また、その結果により、新しいクラスが使用できないと判明した場合に、新しいクラスを削除できるようにする機能の実装。
- 性能を向上させるために、交換予定がなくなったクラスを固定することで、通常メソッド呼出しと同等の速度が得られるようにする機能の実装。
- フィールドへ直接アクセスできないようになっているので、その点を改善する。

3 まとめ

本稿では、Java におけるクラス定義の動的変更を可能とするシステムの提案を行い、現在、実装できている機能などについて紹介した。性能面やシームレスでないなどの問題を抱えており、これらの問題に対処していきたい。また、実用的なサンプルアプリケーションを作成する予定である。