Java servlet におけるクラス間の依存情報抽出

A dependence extraction between classes for Java servlet

石垣 一[†] 松井 藤五郎[††] 大和田 勇人[††]
Hajime Ishigaki[†] Tohgoroh Matsui[††] Hayato Ohwada[††]

[†] 東京理科大学 大学院理工学研究科 経営工学専攻
Department of Industrial Administration, Graduate School of Science and Technology, Tokyo University of Science

[††] 東京理科大学 理工学部 経営工学科
Department of Industrial Administration, Faculty of Science and Technology, Tokyo University of Science

j7404605@ed.noda.tus.ac.jp {matsui, ohwada}@ia.noda.tus.ac.jp

Java servlet を用いて開発された Web アプリケーションのソースコードは非常に膨大になってしまう傾向にあり，開発者はプログラムソースからアプリケーションの全体像を理解する事が困難になるという現状がある．そこで，本論文では，Java servlet に対して，バイトコード埋め込み方式による動的解析とバイトコードに対する静的解析を行い，プログラム理解に有効な情報の抽出方法の提案する．また，本手法の有効性に関して実際に大規模な Web アプリケーションに本手法を適用した結果に基づいて考察を行い，有効性を示す．

1 はじめに

近年，Java servlet は大規模な Web アプリケーションの開発に広く用いられている．これは，Java servlet の Web との親和性と，オブジェクト指向言語の側面が評価されたためである．これにより，Java servlet を用いることによって，容易に大規模な Web アプリケーションが実現できるようになった．

その一方で，他人が記述した膨大な Java servlet プログラムを理解することは，その複雑さから困難である．そこで，本論文では，Java servlet の複雑さの要因を明らかにし，その複雑さを解消するために，Java servlet プログラムに対して，動的・静的な解析手法を適用する．そして，Java servlet プログラムから開発に有効な情報の抽出を試みる．最後において，本手法を実際の Web アプリケーションの解析に適用し，その有効性について考察する．

2 関連研究

オブジェクト指向言語の静的・動的解析に関する研究には [2] がある．この手法では，本研究と同様に，プログラムを静的・動的に解析する．静的な解析ではプログラムに含まれる命令文の間の依存関係に着目している．また，動的な解析は仮装マシン（VM）を機能拡張することによって実現している．

しかしながら，本研究で対象としている Java servlet は，サーブレット・コンテナ上で動作するため，この手法による動的な解析は困難である．また，プログラムと Web が連動して振る舞うため，この手法による静的な解析も困難である．

また，バイト・コードを埋め込むことによって動的に解析する手法 [3] が提案されているが，ユーザが挿入するバイト・コードを直接記述する必要があり，特別な知識を必要とする．本研究では，Java コード最適化フレームワークの Soot [1] を用いて自動的に解析コードを埋め込むことによって，Java servlet の振る舞いを動的に解析する．

3 Java servlet の複雑さ

本章では，Java servlet による Web アプリケーションの振る舞いを理解するのに必要な情報について述べる．また，これらの情報を取得することを困難にしている Java servlet の複雑さについて，オブジェクト指向言語と Web ベースシステムの性質という二つの観点から検討し，複雑さを解消するために静的・動的な解析が必要であることを示す．

3.1 振る舞いの理解に必要な情報

Java servlet の開発において，有効な情報は，Web アプリケーション全体の動作に関わる情報である．それは，次の 3 つである．

1. Web アプリケーションの実行に必要なクラス群
2. Web アプリケーションのサイト構造の情報
3. セッション関連の情報

これらの情報は，関連するクラス間に依存する情報であり，これらを理解し，正しいことを確認することにより，開発者はシステムの動作が妥当であると知ることができる．

3.2 オブジェクト指向の側面

オブジェクト指向言語としての Java servlet は，オブジェクト指向言語特有の性質である実行時決定要素を数多く含んでいる．さらに，Java servlet は特有の機能として，セッション管理というクライアントとの一連の通信を管理するを備えている．そのため，通信において関連するクラスの依存関係を理解するためには，実際の動作中の情報が必要となる．

また，スタンドアローンで動作する Java プログラムは，通常最初に main(String[]) メソッドがコールされ，最終的に同メソッドが return することによりプログラムは終了する．しかしながら，Java servlet では，通常とは異なり，クライアントからの接続方式が，HTTP プロトコルの GET メソッドによるものであれば，doGet メソッドがコールされ，POST メソッドによるものであれば，doPost メソッドがコールされ，各々のメソッドによる return によりプログラムは終了する．そのため，単純にプログラムをレビューしただけでは，実行系が二通り存在し，通常の Java プログラムの理解より困難である．

3.3 Web ベースシステムの性質という側面

Web ベースシステムという観点から，Java servlet を捉えたとき，複雑さはさらに増大する．Web ベースシステムは，特有の性質として，直接 URL を指定することにより任意のロケーションより，サービスを開始することが可能である．そのため，どのクラスの doPost，doGet がコールされることにより，サービスが開始されるか一意に定まらない．また，一連のサービスに関連するどのようなクラスがコールされ，どのクラスにおいてクライアントがサービスを終了させるのかは，サーバサイドで明示的に指定することは困難である．

すなわち，Web ベースシステムとしての Java servlet は，サービスの開始から終了までの実行系が，通常のプログラムよりも複雑であるという特徴を持つ．

3.4 複雑さの解消

上述の複雑さをまとめると以下のようになる．

1. 一つのクラスに実行系が二つ存在する

2. サービスの開始・終了の曖昧さによる複雑さ

しかしながら，上記の複雑さは，プログラムを動的に解析することにより解消される．

なぜならば，実際に Web アプリケーションを実行し，実行した系に従うことにより，システムの開始・終了を明瞭化できるためである．また，任意のクラスにおいてで doPost,doGet のいずれが呼ばれたかを知ることができるためである．

しかしながら，実行した系でなければ，解析できないのでは，結果の信頼度が低い．そこで，本研究においては，静的解析も行い，動的な解析結果以外の実行系の存在可能性を示せるようにする．

4 依存関係情報抽出手法

4.1 方針

情報抽出対象は，Java servlet である．本論文では，静的・動的解析を行う．しかし，ソースコードを直接改変することにより，動的な解析を行うことは，Java の構文規則などの制約により困難である [2]．そこで，本稿では Java servlet に対して直接解析を試みるものではなく．プログラムをコンパイルした後に生成されるバイトコード，つまりは，class ファイルに対して静的，動的な解析を行う．

抽出する情報は，前章で述べた開発に有効な情報である．

1. Web アプリケーションの実行に必要なクラス群

   (静的) 呼ばれる可能性のあるメソッド情報

2. Web アプリケーションのサイト構造情報

   (静的) encodeURL と encodeRedirectURL の引数

   (動的) encodeURL と encodeRedirectURL の返り値，そして doPost か doGet による出力

3. セッション関連の情報

   (静的) getParameter と setAttribute の引数

   (動的) getParameter と setAttribute の返り値

以上の方針に基づいて，動的，静的解析手法の詳細に関して示す．

```
input：バイトコード B(クラスファイル)
output： 解析結果 R
local variable：
  B：バイトコード (クラスファイル)
  M：invoker されるメソッド名
  eu：encodeURL の引数
  eru：encodeRedirectURL の引数
  gp：getParameter の引数
  sa：setAttribute の引数
  U, STMT：一時的なリスト
  u, stmt：一時変数
1   begin
2    U ← B のメソッド単位のリスト
3    while U ≠ φ do begin
4     u ← U から一要素を取り出す
5     R ← u のメソッド名 M
6     STMT ← u をバイトコード単位のリスト
7     while STMT ≠ φ do begin
8      stmt ← STMT から一要素を取り出す
9      if stmt =invoker ステイトメント
10       then ifstmt = “invoke getParameter”
11        then R ← gp
12       else_if stmt =”invoke setAttribute”
13        then R ← sa
14       else_if stmt =”invoke encodeURL”
15        then R ← eu
16       else_if stmt =”invoke encodeRed.”
17        then R ← eru
18       end_if
19      end_if
20     end
21    end
22   end
```

図 1: 静的解析アルゴリズム

```
input：バイトコード B(クラスファイル)
output： 解析コードを付加したバイトコード B′
local variable：
  B：バイトコード (クラスファイル)
  P：POST を出力バイトコード
  G：GET を出力するバイトコード
  eu：encodeURL の返値を取得するバイトコード
  eru：encodeRed. の返値を取得するバイトコード
  gp：getParameter の返値を取得するバイトコード
  sa：setAttribute の返値を取得するバイトコード
  U, STMT：一時的なリスト
  u, stmt：一時変数
1   begin
2    U ← B のメソッド単位のリスト
3    while U ≠ φ do begin
4     u ← U から一要素を取り出す
5     STMT ← u をバイトコード単位のリスト
6     while STMT ≠ φ do begin
7      stmt ← STMT から一要素を取り出す
8      if stmt =”invoker ステイトメント”
9       then ifstmt =”invoke getParameter”
10       then B に gp を加える
11      else_if stmt =”invoke setAttribute”
12       then B に sa を加える
13      else_if stmt =”invoke encodeURL”
14       then B に eu を加える
15      else_if stmt =”invoke encodeRed.”
16       then B に ern を加える
17      else_if stmt =”invoke println”
18       then B に (P,G) を加える
19      end_if
20     end_if
21    end
22   end
23  end
```

図 2: 動的解析アルゴリズム

4.2 静的解析

静的解析のアルゴリズムを図 1 に示す.

このアルゴリズムの特徴は，バイトコードをステイトメント単位で処理を行っている点である．また，ある程度まとまった量で処理を行うために，メソッドごとに処理を行っている．そして，メソッドの invoker ステイトメントに着目し，プログラムの実行によって呼ばれる可能性のあるすべてのメソッドを取得する．また，そのようなメソッドに対して，特に前節で触れたメソッドに関して，その引数を取得する．

Web アプリケーションを構成するすべてのクラスに対して，図 1 を適用する必要がある．

4.3 動的解析

動的な解析アルゴリズムを図 2 に示す.

動的解析のアルゴリズムは，正確には解析アルゴリズムではなく，解析のポイントとなる情報を取得するために，クラスファイルに対してバイトコードを挿入するためのアルゴリズムである．

動的な解析結果は，図 2 により生成されたクラスファイルを実際に Web アプリケーションとして実行することによって，取得できる．

静的な解析アルゴリズムと同様に，Web アプリケーションを構成するすべてのクラスに対して図 2 を適用する必要がある．

5 実装

アルゴリズムを実装するために Soot[1] を用いた．Soot は，Java コード最適化のフレームワークであり，Java コードの解析や変換の目的に沿ったいくつかの中間言語を生成し，その上でのコードの挿入や，削除などの基本的な操作を行う環境を提供する．

本稿においては，Soot が生成する中間言語の一つである型付けされた 3-address 表現で，人間の理解が容易な Jimple という言語に，一度 class ファイルを変換し，解析コードを追加，また，それと同時に静的な解析を行い，その後，コードが追加された Jimple 言語を再度 class ファイルへ変換する．

これにより，実現したシステムによって，1class ファ

イル当たり平均 2 秒程度で，静的な解析およびコードの挿入が行える．また，コードが挿入された class ファイルは，コード挿入前と同様の Web アプリケーションとして正常に動作をし，動的な解析結果を出力するような変更を加えたにもかかわらず，処理が特に重くなるようなこともなかった．

6 手法の有効性

6.1 静的解析の有効性

本手法を，クラス数 2560，総プログラム行数 43 万からなる Web アプリケーションに適用し，以下の情報に関して全部で 39 万個の情報を抽出した．

1. 呼ばれる可能性のあるメソッド情報
2. encodeURL と encodeRedirectURL の引数
3. getParameter と setAttribute の引数

かなり情報量が多いが，その原因は呼ばれる可能性のあるメソッド情報の重複を許可しているためである．呼ばれる可能性のあるメソッド情報を除いた情報数は，1 万 7 千個であった．1 クラス当たりでは，平均 7 個の情報数である．

少なくとも，2. と 3. の情報が分かれば，Web アプリケーションの実行系を理解できる．よって，直接ソースコードに目を通すより容易に，Web アプリケーションの理解が可能となったといえる．

6.2 動的解析の有効性

動的な解析は，対象システムの主要な部分に関してのみしか検証を行っていないが，本手法により実際に以下の項目に関してわかったことを述べる．

1. encodeURL と encodeRedirectURL の返り値
2. doPost か doGet による出力
3. getParameter と setAttribute の返り値

まず，セッションの管理がクッキーを用いて行われているのか，また，サーバサイドで行われているのかを，encodeURL と encodeRedirectURL の返り値を取得することによって理解した．

そして，doPost か doGet による出力結果を動的に取得したことにより，いずれのメソッドによりサーバが処理したかわかるようにした．また，アクセスのたびに，過去の出力を保持するようにしたので，エラーと過去の正しい出力の比較ができる．

実際， getParameter と setAttribute の返り値により，本システムは見つからなかったが，セッションに関する null ポインターの発見に有効であった．

6.3 手法の課題

本手法において，Web アプリケーションシステムの全体像を理解する上で重要なセッション関連の情報と，システムの状態遷移関係 (リンク情報) などを取得することができた．

しかしながら，構造を理解するためには，視覚的な表現を用いてシステムの全体像を表現する必要があると考えられる．

7 おわりに

本論文では，開発者が Java servlet プログラムを理解する上で有効な情報として，1) Web アプリケーションの実行に必要なクラス情報，2) Web アプリケーションのサイト構造情報，3) セッション関連の情報という，関連するクラス間の依存情報を抽出した．

Java servlet は，1) 一つのクラスに実行系が二つ存在する，2) Web アプリケーションの開始・終了の時点が複数存在，という実行系が定まらない問題を解決するために動的なプログラム解析を行った．さらに，動的な解析ではすべての実行系の解析結果を取得できないので，合わせて静的な解析も行った．

実際に，大規模な Web アプリケーションに対して本手法を適用し， 開発に有効な情報の抽出を行い，その情報を用いてシステムの理解と検査ができることを示した．

今後の課題は，抽出した情報をより理解しやすいように視覚的に表現するなどの工夫が必要である．

参考文献

[1] Raja Vallee-Rai, Laurie Hendren, Vijay Sundaresan, Patrick Lam, Etienne Gagnon,and Phong Co. : Soot- a java optimization framework. *In Proceedings of CASCON 1999*, pp. 125-135, 1999.

[2] 誉田, 大畑, 井上: Java バイトコードにおけるデータ依存解析手法の提案と実現, コンピュータソフトウェア, Vol.18, No.3, pp.40-44, 2001.

[3] H.B.LEE, B.G.Zorn: BIT: A Tool for Instrumenting Java Bytecodes, *In Proceedings of USENIX Symposium on Internet Technologies and Systems*, pp.73-83, 1997.