GluonJ を用いたビジネスロジックからのデータベースアクセスの分離

石川 零　　千葉 滋

東京工業大学大学院 情報理工学研究科
数理・計算科学専攻
{rei, chiba}@csg.is.titech.ac.jp

要 旨

本稿は GluonJ がデータベースアクセスのコードをビジネスロジックから分離し、シンプルな設計にできることを示す。GluonJ はコンポーネント間の依存関係を弱めることを目的とした AOP 言語である。GluonJ のアスペクトは設定ファイルのようなものであり、コンポーネント間の結びつき方とメッセージのやり取りの仕方を記述するために用いられる。本稿では GluonJ を用いて設計したデータベースアクセスを伴うプログラムと AspectJ を用いて設計したプログラムを紹介し、GluonJ を用いたほうが再利用性の高いプログラムの記述ができることを示す。

1　はじめに

Web アプリケーションは複雑化しており、それにともないアプリケーションのコンポーネント化が進んでいる。多くの Web アプリケーションはデータベースアクセスをともなう。データベースへのアクセス処理は利用するデータベースの仕様に依存するため、データアクセスオブジェクト (以下 DAO) という 1 つのコンポーネントにまとめて隠蔽化するという手法が取られる。

アスペクト指向プログラミング (以下 AOP) は、DAO のメソッドを呼び出すコードを Web アプリケーションのビジネスロジック [1] から分離することを助ける。多くの AOP システムはプログラム中のある実行点で新たなコードを実行する機能を提供する。DAO のメソッドを呼び出すコードを分離することで、データベースアクセスに依存しないビジネスロジックを作ることができる。これによりビジネスロジックの保守性は向上する。AOP を使わずにビジネスロジックと DAO の依存関係を弱める技術として、Dependency Injection コンテナ [3] (以下 DI コンテナ) がある。開発者は DI コンテナを使うことでビジネスロジックから DAO の生成を分離することができるが、DAO のメソッド呼び出しを分離することはできない。

本稿では我々の開発した AOP 言語である GluonJ を用いて DAO へのアクセスを分離し、シンプルになったプログラムを示す。GluonJ はコンポーネント間の依存関係を弱めるための AOP 言語である。GluonJ のアスペクトは設定ファイルのようなものであり、コンポーネント間の結びつき方やメッセージのやり取りの仕方を記述するために用いる。我々はこのアスペクトを *glue* と呼んでいる。GluonJ ではビジネスロジックと DAO の依存関係を全て glue の中に記述する。そのため DAO がビジネスロジックに依存する必要がなく、DAO の再利用性は高くなる。AspectJ [6] [7] などアスペクトをコンポーネントとして扱う[2]AOP 言語では DAO をアスペクトとして記述するため、DAO に含まれるアドバイスがビジネスロジックに依存してしまう。

以後、2 章で AOP が DAO へアクセスするコードの分離を助けることを示す。3 章で GluonJ を用いることで分離したプログラムがシンプルになることを示す。最後に 4 章で本稿をまとめる。

2　AOP を用いたデータベースアクセスコードの分離

この章では AOP を用いることでビジネスロジックから DAO を呼び出すコードを分離できることを示す。まず例として用いるユーザ登録プログラムについて説明し、DI コンテナではこのプログラムか

[1] ビジネスロジックとは Web アプリケーション中でおこないたい業務処理のことを指す。

[2] アスペクトをコンポーネントとして扱うとは、アスペクトをインスタンス化して使うことと同じ意味である。

らDAOのメソッドを呼び出すコードを分離できないことを示す。次にAOPを使うことでどのようにDAOのメソッド呼び出しを分離できるかについて説明する。そしてAspectJなどのAOP言語でアスペクトをコンポーネントとして扱うことの問題点を示す。

2.1 例：RegisterUser プログラム

データベースアクセスをおこなうプログラム、RegisterUserについて説明する。RegisterUserはユーザの名前と住所をデータベースに登録する機能を持つ。登録は2つのWebページを使っておこなう。最初のページで名前の登録を、次のページで住所の登録をおこなう。これら2つの処理は一連のトランザクションとして管理される。2つの処理が同じコンピュータからのものかどうかはセッションIDが同一かどうかで判断する。

2.2 DI コンテナの限界

DIコンテナはオブジェクトを生成する処理をプログラムの外側に分離して記述することを支援するフレームワークである。DIコンテナを利用することでビジネスロジックからDAOを生成する処理を分離して記述することができる。代表的なDIコンテナにはSpring Framework [5]やPico Container [2]、Seaser [8]がある。以下はDIコンテナを用いてRegisterUserプログラムを実装した例である。NameSubmitServiceは名前の登録をおこなうクラスである。submitNameメソッドは名前に含まれる英字を大文字に変換し、daoフィールドのpostNameメソッドを使ってデータベースへ登録する。

```
public class NameSubmitService {
  private int sessionId;
  private DataAccessObject dao;
  public NameSubmitService(int s) {
    sessionId = s;
  }
  public setDAO(DataAccessObject d) {
    dao = d;
  }
  public void submitName(
      String firstname,
      String lastname) {
    firstname = firstname.toUpperCase();
    lastname  = lastname.toUpperCase();
    dao.postName(firstname, lastname);
  }
}
```

このプログラム内ではsubmitNameメソッドを実行する前にdaoフィールドを初期化する必要はない。なぜならDIコンテナがDataAccessObject型のオブジェクトを生成し、setDAOを呼んでフィールドを初期化するからである。DataAccessObject型のオブジェクトの生成に使うコンストラクタや初期化に利用するオブジェクトは設定ファイルを用いて記述する。設定ファイルは以下のようになる。

```
<beans>
  <bean id = "nameSubmit"
        class = "NameSubmitService">
    <property name = "dao">
      <ref local = "dataAccessObject"/>
    </property>
  </bean>
  <bean id = "dataAccessObject"
        class = "DataAccessObjectImpl">
    <!-- データベースのURL などを指定 -->
  </bean>
</beans>
```

二つあるbeanタグのうち上のタグが、NameSubmitServiceのdaoフィールドにDataAccessObject型のオブジェクトを与えるという意味を持つ。下のbeanタグは特定のデータベースへのアクセスするためのDataAccessObjectを生成する記述である。

しかしDIコンテナではDAOのメソッド呼び出しを分離することはできない。NameSubmitServiceクラスのsubmitNameメソッドはデータベースに名前を登録するために、daoフィールドのpostNameメソッドを明示的に呼び出す必要がある。

2.3 AOP による理想的な分離

AOPを使うことでDAOのメソッド呼び出しをビジネスロジックから分離することができる。以下はAOPを利用して記述したRegisterUserプログラムのNameSubmitServiceクラスである。

```
public class NameSubmitService {
  private int sessionId;
  private String firstname;
  private String lastname;
  public NameSubmitService(int s) {
    sessionId = s;
  }
  public void submitName(
      String first, String last) {
    firstname = first.toUpperCase();
    lastname  = last.toUpperCase();
  }
}
```

DI コンテナを使った場合と違い、NameSubmitService クラスに DAO へアクセスするコードは含まれない。DAO の生成とデータの登録はアスペクトがおこなう。以下はその TransactionDAOAspect アスペクトである。このアスペクトは AspectJ 風の抽象的な AOP 言語を用いて記述した。

```
aspect TransactionDAOAspect {
  TransactionDAO dao;
  advice after (
      Service オブジェクトの生成時) {
    dao = new TransactionDAO();
  }
  advice after (
      NameSubmitService.submitName(
      String, String) の実行時) {
    dao.postName(firstname, lastname);
  }
  advice after (
      AddressSubmitService.
      submitAddress(String) の実行時) {
    dao.postAddress(address);
  }
}
```

TransactionDAOAspect アスペクトは 1 つのフィールドと 3 つのアドバイスから構成される。1 番目のアドバイスは NameSubmitService オブジェクトが生成されたときに DAO を生成してフィールドに格納する。Service クラスはデータ登録をおこなうクラスに共通の親クラスである。2 番目のアドバイスは submitName メソッドの実行終了時に DAO を使って名前の登録をおこなう。3 番目のアドバイスは submitAddress メソッドの実行終了時に DAO を使って住所の登録をおこなう。AddressSubmitService クラスは住所を登録するためのクラスである。

このアスペクトはオブジェクトと複雑に結びつける必要があり、AspectJ で書くのは難しい。複数のサービスにまたがったトランザクション管理をおこなうため、同一のセッション ID を持つ Service オブジェクトは同じ DAO を使わなければならない。DAO はアスペクトが管理するため、それらの Service オブジェクトに対して同じアスペクトを結びつける必要がある。AspectJ ではオブジェクトとアスペクトの結びつけ方をあらかじめ決められた指定子の中から選んで使う。例えば全てのオブジェクトに対して 1 つのアスペクトを結びつける issingleton、各オブジェクトに対して 1 つのアスペクトを結びつける perThis, perTarget などがある。ところがこれらの指定子では上述したアスペクトの結びつけ方を実現することはできない。そのため AspectJ の既存の指定子を使ってこのような結びつけの仕組みを実装する必要があるが、それではプログラムの設計が複雑になってしまう。これについては 3.2 節で複雑になった実例を示す。

3 GluonJ による設計の改善

我々の開発した AOP 言語、GluonJ はビジネスロジックから DAO の呼び出しを分離でき、AspectJ を利用した場合よりもアプリケーションの設計をよりシンプルにすることができる。この章では、まず GluonJ を用いて書いたアスペクトについて説明し、次に AspectJ を用いて書いたアスペクトを説明する。そして GluonJ を用いたほうが DAO の再利用性が高いことを示す。

3.1 GluonJ で書いたアスペクト

GluonJ は AspectJ のポイントカット・アドバイスモデルを利用した Java 用の AOP 言語である。GluonJ ではアスペクトを用いてコンポーネント間の結びつきを記述する。このアスペクトを glue と呼ぶ。以下は GluonJ を用いて記述した RegisterUser プログラムの TransactionDAOAspect アスペクトである。glue は XML を用いて記述する。

```
<aspect>
  <injection>
    TransactionDAO Service.aspect =
      TransactionDAO.factory(
      this.sessionId);
  </injection>
  <advice>
    <param>
      <name>n</name>
      <type>NameSubmitService</type>
    </param>
    <pointcut>
      execution(void NameSubmitService.
      submitName(String, String)) AND
      this(n)
    </pointcut>
    <after>
      TransactionDAO.aspectOf(n).
        postName(n.firstname, n.lastname);
    </after>
  </advice>
  <advice>
    <param>
      <name>n</name>
      <type>AddressSubmitService</type>
    </param>
    <pointcut>
      execution(void AddressSubmitService.
```

```
      submitAddresss(String)) AND this(n)
    </pointcut>
    <after>
      TransactionDAO.aspectOf(n).
        postAddress(n.address);
    </after>
  </advice>
</aspect>
```

Glue はどのコンポーネントとどのコンポーネントを結びつけるか、イベントでどのメソッドを呼ぶかを指定する。この TransactionDAOAspect は、トランザクション管理機能を持つ TransactionDAO クラスと Service クラスを結びつける。

glue を用いることで DAO の再利用性を高めることができる。この例で、TransactionDAO クラスは Service クラスなどビジネスロジックのコードを参照する必要はない。なぜなら TransactionDAO クラスが Service クラスに依存しなければならない部分の処理は、全て glue の中に書かれているからである。そのためビジネスロジックに変更があったとしても変更する必要があるのは glue のみである。例えば Service クラス以外の別なクラスが TransactionDAO クラスを利用したいときも、変更する必要があるのは TransactionDAOAspect アスペクトだけである。TransactionDAO クラスを変更する必要はない。

以下で glue の各タグの役割について説明する。開発者は injection タグ内にオブジェクト同士を結びつけるための記述を書くことができる。上述した例の injection タグに書かれた記述は、Service オブジェクトが生成したときにファクトリを使って TransactionDAO オブジェクトを生成し結びつけるという意味である。

advice タグ内にはポイントカットとアドバイスの組を記述する。ポイントカットを記述するために、GluonJ は AspectJ と同様の文法を用意している。アドバイスでは Java の文法に従ったコードのほかに aspectOf という特別なメソッドを使うことができる。これは injection タグで結び付けられたオブジェクトを取り出すためのゲッターメソッドである。TransactionDAO.aspectOf(n) は引数 n に結びついた TransactionDAO オブジェクトを返す。

次に glue の中で利用している TransactionDAO クラスについて説明する。このクラスはトランザクション管理機能を持つ DAO の簡易版である。

```
public class TransactionDAO {
  DBContext db = getDBContext();
  public void postName(
      String first, String last) {
    try {
      db.query("INSERT (" + first +
               " , " + last + ")");
    } catch(DBException e) {
      db.rollback();
    }
  }
  public void postAddress(String a) {
    try {
      db.query("INSERT " + a);
      db.commit();
    } catch(DBException e) {
      db.rollback();
    }
  }
  private DBContext getDBContext() {
    // データベースへのアクセサを返す
  }

  // ファクトリメソッドのための記述
  private static HashMap daoMap =
      new HashMap();
  private static TransactionDAO factory
      (int sessionId) {
    Integer i = new Integer(sessionId);
    TransactionDAO dao =
        (TransactionDAO) daoMap.get(i);
    if (dao == null) {
        dao = new TransactinoDAO(sessionId);
        daoMap.put(i, dao);
    }
    return dao;
  }
}
```

postName、postAddress メソッドはトランザクション管理を考慮しつつ、データをデータベースへ登録するメソッドである。factory メソッドは同一セッション ID に対して同じ TransactionDAO オブジェクトを返す。図 1 は GluonJ を用いて記述した RegisterUser プログラムを表している。

3.2 AspectJ で書いたアスペクト

GluonJ との比較のために、AspectJ を用いて記述した RegisterUser プログラムの TransactionDAOAspect アスペクトを以下に示す。これは GluonJ を用いて記述したプログラムの TransactionDAO クラスに相当する。

```
privileged aspect TransactionDAOAspect {
  // インタータイプ宣言
  private DBContext Service.db;
  after(Service n) :
      execution(Service+.new(..)) &&
      this(n) {
    n.db = factory(n.sessionId);
  }
```

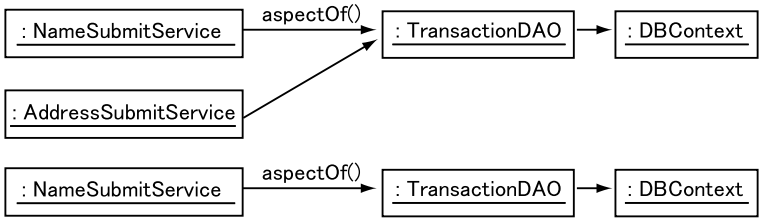

図 1: GluonJ を用いて記述した RegisterUser プログラム

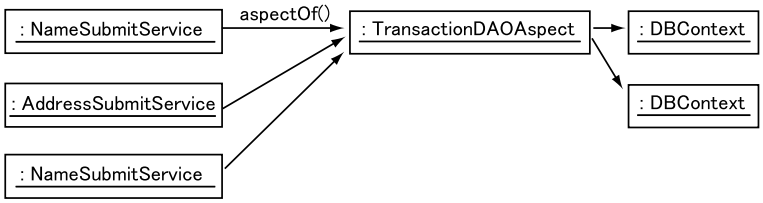

図 2: AspectJ を用いて記述した RegisterUser プログラム

```
  after(NameSubmitService n,
        String first, String last) :
      execution(void NameSubmitService.
      submitName(String, String)) &&
      this(n) && args(first, last) {
    try {
      db.query("INSERT (" + first +
               " , " + last + ")");
    } catch(DBException e) {
      db.rollback();
    }
  }
  after(AddressSubmitService n) :
      execution(void AddressSubmitService.
      submitAddress(String)) && this(n) {
    try {
      db.query("INSERT " + address);
      db.commit();
    } catch(DBException e) {
      db.rollback();
    }
  }

  // ファクトリメソッドのための記述
  private static HashMap dbMap =
      new HashMap();
  private static DBContext factory
      (int sessionId) {
    Integer i = new Integer(sessionId);
    DBContext db = (DBContext) dbMap.get(i);
    if (db == null) {
        db = getDBContext();
        dbMap.put(i, dao);
    }
    return db;
  }
  private static DBContext getDBContext() {
    // データベースへのアクセサを返す
  }
}
```

1 番目のアドバイスは Service クラスのサブクラスのオブジェクトとトランザクションの状態を管理する DBContext オブジェクトとを結びつける。db フィールドはインタータイプ宣言を用いて Service に追加されたフィールドである。2 番目と 3 番目のアドバイスはそれぞれ名前の登録、住所の登録をおこなうアドバイスである。

AspectJ を用いた ResisterUser プログラムに含まれる TransactionDAOAspect アスペクトの再利用性は、GluonJ を用いた ResisterUser プログラムの TransactionDAO クラスに比べて低い。なぜなら TransactionDAOAspect アスペクトがビジネスロジックに含まれる特定のクラスに依存しているからである。例えば TransactionDAOAspect のアドバイスは Service クラスや NameSubmitService クラスのメソッド実行をポイントカットすることで、それぞれのクラスに依存している。このため、例えばビジネスロジックに含まれる別のクラスが TransactionDAOAspect を利用したいときには TransactionDAO クラスを修正する必要が出てくる。たとえば Service クラスを継承していない MailTransfer クラスに TransactionDAOAspect を利用させたいとき、TransactionDAOAspect に新たなアドバイスを追加しなければならない。

4 まとめと今後の課題

本稿は GluonJ がデータベースアクセスのコードをビジネスロジックから分離し、シンプルな設計にできることを示した。GluonJ は AspectJ と似た AOP 言語であるが、アスペクトのインスタンス化をしない点やアスペクトのスコープがジョインポイントのスコープと等しいという点で AspectJ と異なる。これらの違いにより、GluonJ はユーザ登録をおこなう RegisterUser プログラムを AspectJ に比べてよりシンプルな設計にできる。

コンポーネントフレームワークのための AOP 言語として JBoss AOP [4] や AspectWerkz [1] がある。これらの言語ではオブジェクトとアスペクトの結びつけをいくつかの指定子から選んで指定する。アスペクトのインスタンスは指定子に基づいて作られるため、AspectJ と同様の問題が生じる。

また AspectJ の実装を拡張して新たに指定子を導入し、可能な結びつけ方を増やそうという研究として association aspects [9] がある。association aspects はオブジェクトの組にアスペクトインスタンスを結びつけるための指定子を AspectJ に導入する。この研究は AspectJ が提供する指定子だけでは十分でないことを示している。

GluonJ はオブジェクトとオブジェクトの結びつけ方を柔軟に定義でき、実現できる結びつけ方は association aspects よりも豊富である。

GluonJ は AOP を用いてコンポーネントの分離を助け、再利用性を高めることを目標にしている。AOP は関心事の分離を促進するための技術であり、コンポーネント同士の依存関係を分離することに利用できる。しかし現状の AspectJ ではこのような依存関係をうまく分離することができない。また我々は GluonJ を用いた応用技術としてコンポーネントフレームワークの設計と実装を進めている。

参考文献

[1] Jonas Boner and Alexandre Vasseur. Aspectwerkz 1.0. `http://aspectwerkz.codehaus.org/`.

[2] Codehaus. Pico Container. `http://www.picocontainer.org/`.

[3] Martin Fowler. Inversion of control containers and the dependency injection pattern. `http://www.martinfowler.com/articles/injection.html`.

[4] JBoss Inc. Jboss aop 1.0.0 final. `http://www.jboss.org/`.

[5] Rod Johnson and Juergen Hoeller. *Expert One-on-One J2EE Development without EJB*. Wrox, 2004.

[6] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. An overview of aspectj. In Jørgen Lindskov Knudsen, editor, *ECOOP*, volume 2072 of *Lecture Notes in Computer Science*, pages 327–353. Springer, 2001.

[7] AspectJ Project. Aspectj. `http://eclipse.org/aspectj/`.

[8] The Seasar Project. Seasar. `http://homepage3.nifty.com/seasar/`.

[9] Kouhei Sakurai, Hidehiko Masuhara, Naoyasu Ubayashi, Saeko Matsuura, and Seiichi Kimoya. Association aspects. In *Aspect-Oriented Software Development*, pages 16–25, Mar. 2004.