Web アプリをユーザー毎にカスタマイズ可能にする AOP フレームワーク

戸部 敦[†1] 千葉 滋[†1]

Web アプリをユーザー毎にカスタマイズ可能にすることは, ユーザーを引き付ける上で重要である. しかし, そのような Web アプリの開発にはプログラミングの知識や事前の入念な設計が必要であり, 通常の Web アプリに比べて格段の開発コストがかかってしまう. そこで本稿では, ユーザー毎にカスタマイズ可能な Web アプリを容易に開発できるようにするための AOP フレームワークを提案する. カスタマイズ部分を AOP 言語によって記述することで, カスタマイズ機構を予め組み込んでおく必要がなくなる. 本フレームワークを実現するために, 我々は per-session weaving を提案する. per-session weaving は, セッションからユーザーを特定し, ユーザー毎に異なるアスペクトを Web アプリのプログラムに weave する技術である.

An AOP Framework for User-Customizable Web Apps

Atsushi Tobe[†1] and Shigeru Chiba[†1]

Making a web application customizable by users is important but needs extra development costs. This paper presents an AOP framework for developing user-customizable web applications. Since the users describe customization by aspects, the web application itself does not include a dedicated customization mechanism. For this framework, we developed a new technique called *per-session weaving*, which weaves different aspects with programs for every user.

1. はじめに

現在, ユーザー毎にカスタマイズ可能な Web アプリが多く普及している. 例えば多くのブログ・サイトでは, 公開されているプラグインを選択することで, カレンダーやカテゴリの表示など, 好きなコンテンツをブログ内に配置できる. さらに, 自分のブログの色やスタイル, テンプレートまで変更できるものが多い.

これらのカスタマイズは, ユーザーの独自性を尊重している. ユーザーの創造力をくすぐり, 他のユーザーと差をつけるべく競争心をあおる. Web アプリをカスタマイズ可能にすることは, Web サイトにユーザーを引き付ける重要な要因である.

このようなカスタマイズ機能を Web アプリに組み込むのは容易ではない. そのような Web アプリを実現するには, まず, カスタマイズ対象の機能を仕様から抽出しなければならないが, これを設計段階でおこなうのは容易ではない. 将来の利用状況を想像して, Web アプリのどの部分をどのようにカスタマイズできるようにするかを決めなければならないからである.

†1 東京工業大学
Tokyo Institute of Technology

```
interface Style {
  String table(String[][] data); // 表を作成
  String title(String s);        // 表題を作成
}
```

図 1 Style インターフェース

これを怠ると, 完成後に後からカスタマイズできる部分を追加・変更しなければならないが, 以下で述べるように, それは大幅なプログラムの書き換えを必要とする.

例えば, 今, Web ページの中の表の表現を自由に変更できるようにしたいとする. その場合, プログラム中で表の html コードを生成している部分を修正し, 生成ロジックを直接埋め込むのではなく, 別なオブジェクトのメソッド (例えば table) として分離し, それを呼ぶようにする必要がある. つまり,

```
String html = webApp.style.table(data);
```

のように修正する. ここで data は表の中身のデータ, style は図 1 に示す Style インタフェースをもつオブジェクトである. style がさすオブジェクトは, 単純に new 演算子で生成してはいけない. 初期化時に設定ファイル等にしたがって生成しなければならない. カスタマイズする際は, まず新しいロジックを実行する

クラスを Style インタフェースを実装するクラスとして定義する．次に, 設定ファイルの中で, 生成するオブジェクトのクラスの指定を新しいものに変えることで, 元のプログラムを改変せずにカスタマイズをおこなうことができる．このようなカスタマイズ処理は, いわゆるプラグイン機構の中核であるので, 汎用のフレームワークを流用して実装することもできる．例えば DI (Dependency Injection) コンテナは, 自動的に設定ファイルが指定するクラスのオブジェクトを生成し, 指定されたオブジェクトのフィールド (例えば style) をそれで初期化する．

ところが, ユーザ毎に Web アプリの一部のロジックもカスタマイズ可能にしようとすると, このような汎用のフレームワークでは不十分である．style フィールドがさすオブジェクトを, ユーザごとに変えなければならないからである．つまり,

```
String html
  = webApp.getStyle(uid).table(data);
```

のように修正しなければならない．ここで uid はユーザの識別子である．getStyle メソッドは, uid ごとに対応する Style オブジェクトを返す．このようなユーザごとにオブジェクトを切り替えるような機能は一般的なフレームワークでは提供されない．

あるいは, Web アプリ全体で 1 つの Style オブジェクトを共有し, そのオブジェクト内部でユーザによって実行するロジックを切り替える方法もある． この方法では, プラグイン機構を提供する既存のフレームワークを流用できる．しかし, この方法では, あるユーザ向けにだけ表の表現を変更しようとすると, その共有オブジェクトのクラス自体を修正しなければならない．元のクラスはそのままに, そのユーザ用のロジックだけを追加実装して, 変更を実現することはできない．これではユーザごとに柔軟にカスタマイズ可能とは言い難い。

2. 提案するフレームワーク

本稿では, ユーザー毎にカスタマイズ可能な Web アプリを容易に開発できるようにするための AOP フレームワークを提案する．ユーザーは AOP 言語を用いて, カスタマイズ部分をアスペクトとして記述する．カスタマイズされたロジックは, AOP 言語処理系によって元のプログラムに組み込まれるので, Web アプリ作成者がカスタマイズ機能を独自に実装する必要はない．これにより Web アプリ開発が容易になる。また, AOP 言語は既存のプログラムを修正することなく, 既存クラスの挙動を変えることができるので, 設計段階でカスタマイズ対象の機能を決める必要がない．後からプログラムの任意の部分をカスタマイズすることができる．

また我々は本フレームワークのために, *per-session weaving* と呼ぶ技術を開発した．本フレームワークは, セッションに保存されたユーザーの ID からユーザーを特定し, ユーザごとに異なるアスペクトを weave した Web アプリを各セッションごとにロードする．この技術により, ユーザごとに異なるアスペクトを実行中に weave できるようになるので, Web アプリ全体を再起動することなく, ユーザごとにカスタマイズされた web アプリを提供することができる．

2.1 AOP 言語の使用

AOP は, オブジェクト指向ではモジュール化できない関心事を, アスペクトという新たなモジュールを導入することによりモジュール化するプログラミング技法である．アスペクトは, 既存のプログラムを直接変更せずに, プログラムの挙動を修正・拡張することにも使える．本稿ではこれを応用して, Web アプリのカスタマイズに用いているのが特徴である．なお, 言語処理系がアスペクトを実際に元のプログラムに適用して, 挙動を修正・拡張することを「アスペクトを weave する」という．

我々のフレームワークは AOP 言語を使うので, Web アプリ作成者が独自のカスタマイズ機構を用意していなくても, 後からプログラムの任意の部分のロジックをカスタマイズできる．そのとき元のプログラムを変更する必要はない．カスタマイズ対象のロジックが独立したメソッドにさえなっていればよい．

例えば前章の例で, webApp.style フィールドがさすオブジェクトのクラスが HtmlRenderer であったとする．我々のフレームワークでは, Style インタフェースなど定義されていなくても, HtmlRenderer クラスの定義をユーザごとに直接修正・拡張することができる．図 2 は, 我々が開発した AOP 言語である GluonJ[1] によるカスタマイズの記述の例である．このアスペクトを特定のユーザに対してだけ weave するように指示すれば, そのユーザ向けのカスタマイズが実現される．このアスペクトは HtmlRenderer クラスの table メソッドの呼び出しを横取りして, アスペクトに記述されたロジックで html コードを作成し, それを元の table メソッドの戻り値とする．

2.2 per-session weaving

アスペクトを weave するタイミングとして, 実行前にアスペクトを weave する static weaving や, クラスのロード時にアスペクトを weave する load-time

```
@Glue class Customizer {
  @Around("{return (html コードの作成); }")
  Pointcut pc
    = Pcd.call("HtmlRenderer#table(..)");
}
```

図 2 GluonJ によるカスタマイズ例

weaving[2], 実行中に動的にアスペクトを weave する dynamic weaving[3] が存在する. 本稿では, load-time weaving を拡張した per-session weaving を提案する.

Java プログラムを実行する JVM は, 実行に必要なクラスを必要になったときにロードする仕組みになっている. クラスのロードにはクラスローダーが使用され, クラスローダーは親子関係を持つ. クラスローダーは, 要求されたクラスが既に親のクラスローダーにロードされている場合には, 親のクラスローダーでロードされているクラスを JVM に渡す. 要求されたクラスが親のクラスローダーにロードされていない場合には, 子のクラスローダーがロードを行うこととなる. 子のクラスローダーを作成することで, 新たなクラスをロードする直前にアスペクトを weave することができ, load-time weaving を実現できる.

Per-session weaving は load-time weaving を拡張し, セッション毎に異なるアスペクトを weave する技術である. JVM が Web アプリのクラスをロードする直前に, セッションを見てユーザーを特定し, ユーザーが設定したアスペクトを Web アプリのクラスに weave してロードする. 同一のクラスローダーを用いてこれを実現すると, 重複したクラスを定義しようとしたことでクラスローダーがエラーを投げる. これを回避するために, per-session weaving ではユーザー毎に異なるクラスローダーを使用する.

ユーザーが登録したアスペクトが, 他のユーザー Web アプリに weave されると, 他のユーザーのアスペクトが weave された Web アプリは, ユーザー自身のカスタマイズを上書きしてしまうため, ユーザー毎にカスタマイズ可能な Web アプリが実現できない. JVM は親子関係を持たない複数のクラスローダーがロードしたクラスを, 名前空間が異なるものとみなすので, これらのクラスは互いに影響を与えることはない. Per-session weaving によって, Web アプリやアスペクトはユーザー毎に異なるクラスローダー上でロードされるため, ユーザーによるカスタマイズが他のクラスに影響を与えることはない. したがって per-session weaving を用いると, ユーザー毎に異なる挙動をする Web アプリを安全に実現できる.

```
└── System
    └── Common
        ├── Catalina
        └── Shared
            ├── WebApp1
            :
            └── WebAppN
```

図 3 Tomcat のクラスローダー

Per-session weaving には, load-time weaving と同様の実行時オーバーヘッドが存在する. しかし, load-time weaving は JBoss AS や Seasar 等で既に利用されている技術であり, オーバーヘッドが実用的な許容範囲にあることが検証されている.

2.3 実　　装

提案するフレームワークは, AOP 言語に GluonJ, サーバープログラムに Tomcat を用いて実装されている. Tomcat は, 専用の階層構造を持ったクラスローダーを使用している. Tomcat のクラスローダーの階層構造を **図 3** に示す. 図 3 の階層構造を見るとわかるように, Tomcat は Web アプリ毎に異なるクラスローダーを使用している. 本稿ではこれを拡張し, セッション毎に異なったクラスローダーを作成して使用するよう変更した.

作成されたクラスローダーは, ユーザーによって設定されたアスペクトやクラスをロードし, さらに, ロードされたアスペクトを Web アプリを構成するクラスに weave する. こうして, ユーザーによるカスタマイズを適用した Web アプリがクラスローダーにロードされる.

フレームワークは, ユーザー毎にアスペクトやヘルパークラスを登録・削除する API を提供している. Web アプリ作成者は, この API を通じてユーザーのアスペクトを登録すればよい.

2.4 実　　例

フレームワークの動作を確認するために, サンプルアプリを作成した. このサンプルアプリを構成するページは, ログイン画面 (セッションにユーザー ID を保存するページ), 設定画面 (Weave するアスペクトの指定とクラスファイルのアップロードを行うページ. セッションに保存されたユーザー ID とアスペクトを関連付ける), サンプルページ (文章や HTML をユーザーが自由にカスタマイズできるページ), の 3 つである. このアプリに対して異なる 2 人のユーザーとしてログインをし, 異なるアスペクトを設定した. **図 4** はそのときのサンプルページの表示結果である. 全て

図 4 サンプルアプリ

異なる結果が表示されていることから, 他のユーザーのアプリに影響がないことがわかる.

3. 関 連 研 究

本稿で提案した per-session weaving の他に, static weaving や load-time weaving, dynamic weaving などの技術が研究されている. Java を拡張した AOP 言語 AspectJ[4] は, static weaving を使用している. static weaving は実行前に予めアスペクトを weave しておく技術である. しかし提案するフレームワークは, ユーザーによって異なるアスペクトを weave するため, 実行前に weave するアスペクトを特定することはできない.

Load-time weaving は, クラスローダーがクラスをロードするときにアスペクトを weave する技術である. しかし load-time weaving のみで本フレームワークを実現しようとした場合, 全てのユーザーが同一のクラスローダーを使用することとなる. この場合, 2.2 に示したように, クラスの多重定義によりエラーとなってしまう. 本フレームワークでは, セッションを見てユーザー毎にクラスローダーを変更する技術が必要となる.

Nacoara ら[3] の提案する Dynamic weaving は, プログラム実行中に weave するアスペクトを差し替える技術である. しかし dynamic weaving は, 標準の JVM の制限により, 新たなメソッドやフィールドを加えることができないため, アスペクトによる拡張が制限されてしまう. これは JVM を修正することによって防げるが, Web アプリの JVM を修正することによって回避不可能なエラーが生じてしまう可能性もあり現実的ではない.[5]

4. ま と め

本稿では, ユーザー毎にカスタマイズ可能な Web アプリを実現する AOP フレームワークを提案した. カスタマイズ部分を AOP 言語で記述することで, 設計段階からカスタマイズ機構を組み込む必要がないことを示した. また, 本稿で提案した per-session weaving を使用することで, フレームワークが実現可能であることと, ユーザーによるカスタマイズが他のユーザーに影響を与えないことを示した. さらに, 簡単なサンプルアプリによってこれらの技術が実現可能なことを確かめた.

今後の課題は実行時性能の実用性の検証およびそれが十分でない場合の改善である. また, 現在の実装では, ユーザーによって悪意のあるカスタマイズも可能となってしまう. 例えば, ユーザーがサーバープログラムを強制終了させるようなアスペクトを記述した場合には, ページの閲覧時にフレームワークがそのアスペクトを Web アプリに weave し, サーバープログラムが終了してしまう. この問題は, Java Applet のように, Java の SecurityManager を用いてサンドボックスを実現することで解決できると予想されるが, 今後の課題である.

参 考 文 献

1) : `GluonJ Home Page`, http://www.csg.is.titech.ac.jp/projects/gluonj/.
2) Chiba, S.: Load-time structural reflection in Java, *In Proceedings of the 15th European Conference on Object Oriented Programming (ECOOP 2001)*, LNCS 2072, ECOOP 2001, Springer-Verlag, pp.313–336 (2000).
3) Nicoara, A. and Alonso, G.: Dynamic AOP with PROSE, *Proc. of 1st International Workshop on Adaptive and Self-Managing Enterprise Applications*, pp.125–138 (2005).
4) : `The AspectJ Project`, http://www.eclipse.org/aspectj/.
5) Sato, Y., Chiba, S. and Michiaki, T.: A Selective, Just-in-Time Aspect Weaver, *Proc. of 2nd Int'l Conf. on Generative Programming and Component Engineering (GPCE '03)*, pp. 189–208 (2003).