2004－DD－46　(2)
2004／9／24

サーバーサイド Java を用いた動的 Web コンテンツの開発・修正支援手法

福田 健太郎

日本アイ・ビー・エム株式会社 東京基礎研究所
〒 242–8502 神奈川県大和市下鶴間 1623–14
Tel: 046–215–4659　Fax: 046–274–4282
E-mail: kentarou@jp.ibm.com

あらまし

Web アプリケーションの開発において，サーバーサイド Java が広く活用されている．JSP や Servlet などで記述された動的 Web コンテンツの変更や修正を行う際，出力された HTML のソースやブラウザ上で修正箇所を確認・指定しても，JSP・Servlet 中での修正箇所の特定が難しいという問題がある．本研究では，JSP や Servlet の出力処理に着目し，出力元となるコードの位置とその出力内容の関連付けを行うことで，動的 Web コンテンツの開発・修正を支援する手法を提案する．

Development and modification support method for Server-side Java based dynamic Web contents

Kentarou FUKUDA

Tokyo Research Laboratory, IBM Japan, Ltd.
1623–14 Shimotsuruma, Yamato-shi, Kanagawa 242-8502, Japan
Tel: +81 46 215 4659　Fax: +81 46 274 4282
E-mail: kentarou@jp.ibm.com

Abstract

In these days, Server-side Java, such as servlet and JSP, are becoming widely used for Web application platforms. In many cases, these dynamic Web contents are checked via HTML generated from them. As a result, web developers have to search through the Java code to find the corresponding positions where the HTML fragment was generated. To solve this issue, the method which narrows the search range down to a point is proposed in this work. At runtime of Server-side Java, the proposed method observes the JSP/servlet writer and creates a mapping between fragments of generated HTML and the corresponding code positions of the servlet. By using these mappings, developer can easily find the corresponding positions where the HTML fragment was generated.

1 はじめに

近年，World Wide Web(以下，Web) は，静的ページ (HTML) を中心とした情報発信のための手段にとどまらず，電子商取引・電子政府等に代表される種々の手続きのオンライン化や，個人の嗜好にあわせたポータルの提供など，動的ページを用いたアプリケーション提供手段として広く用いられている．

この様な動的 Web コンテンツの開発において，JSP[1]や Servlet を中心としたサーバーサイド Java が広く活用され，その開発を支援する為の統合開発環境も多数提供されている．また，担当業務や嗜好等，利用者の特性に応じた情報・アプリケーションを，その環境や好みに応じたデザインで提供するポータルサーバー [1] の利用も広まっている．さらに，Web アプリケーション開発時に，その機能テストを効率良く行う為の仕組みに関する研究 [2, 3] が多数行われるなど，動的コンテンツを含む Web サイトを開発・提供するのための仕組みは充実しつつある．

一方で，Web サイト構築後に様々な評価・検証を行い，Web サイトの修正・再構築を行う事が重要となってきている [4]．評価・検証の対象としては，例えば，ユーザビリティ，アクセシビリティの評価 [5-9] や，サイト全体のデザインの統一性の確認 [10] などが挙げられる．これらの評価の多くは，JSP などのサーバー側に存在する動的 Web コンテンツではなく，動的 Web コンテンツから生成される HTML を対象として行われている．その結果，提供される修正項目やガイドは，HTML やそのレンダリング後の表示情報を対象としたものとなる．

また，Web アプリケーションの機能テストや，開発時からサイト全体のデザインの統一を図る仕組みにおいても，最終的なチェックは HTML を対象に実施する事になり [3, 10]，その修正項目などは HTML に対するものとして与えられる．

これらの評価・検証結果に基づき，Web サイトの修正を行うためには，修正が必要な箇所を生成した JSP などの動的 Web コンテンツと，そのコンテンツ内での位置を特定する必要がある．しかしながら，動的 Web コンテンツの多くは複数のモジュールを組み合わせて構成されているため，修正箇所の特定は容易ではない．

例えば，JSP では JSTL[11] 等のタグライブラリやスクリプトレットの利用，他の JSP のインクルードが行われ，HTML 内での行数や特定のタグ名などから JSP 内での位置を特定する事は困難である．さらに，ポータル [1] 等の様にテンプレートを利用している場合，コンテンツの入れ子構造が複雑になっており，修正箇所の特定をさらに難しいものとしている．

本稿では，この様な問題を解決するため，JSP や Servlet の出力処理に着目し，出力元となるコードの位置と，その出力結果を関連付ける事で，HTML 内での位置から，該当箇所を生成した動的 Web コンテンツを特定する手法を提案する．提案手法を用いる事により，Web サイトの評価・検証結果から，修正対象となる動的 Web コンテンツを特定する事が容易となり，開発者の労力を軽減する事が期待される．

以下，2 で提案手法の詳細について説明し，3 で提案手法の適用例について述べる．最後に，4 で本稿についてまとめる．

2 出力処理に着目した修正位置特定手法

JSP や Servlet といったサーバーサイド Java を用いた動的 Web コンテンツの変更や修正を行う際に，生成された HTML 上で修正箇所が指定された場合，対応する箇所を生成した JSP 等の位置を特定する必要がある．

従来，この様な修正箇所特定の為の手法としては，生成された HTML 中での行数や，タグ・文字列などに基づいて検索を行ったり，対象となり得る JSP や Servlet 中に特定が容易な文字列を埋め込み，生成された HTML 中での位置を確認し，徐々に範囲を絞り込む手法などが用いられてきた．

[1] Java 及びすべての Java 関連の標語およびロゴは，米国 Sun Microsystems, Inc. の米国及びその他の国における商標または登録商標である．

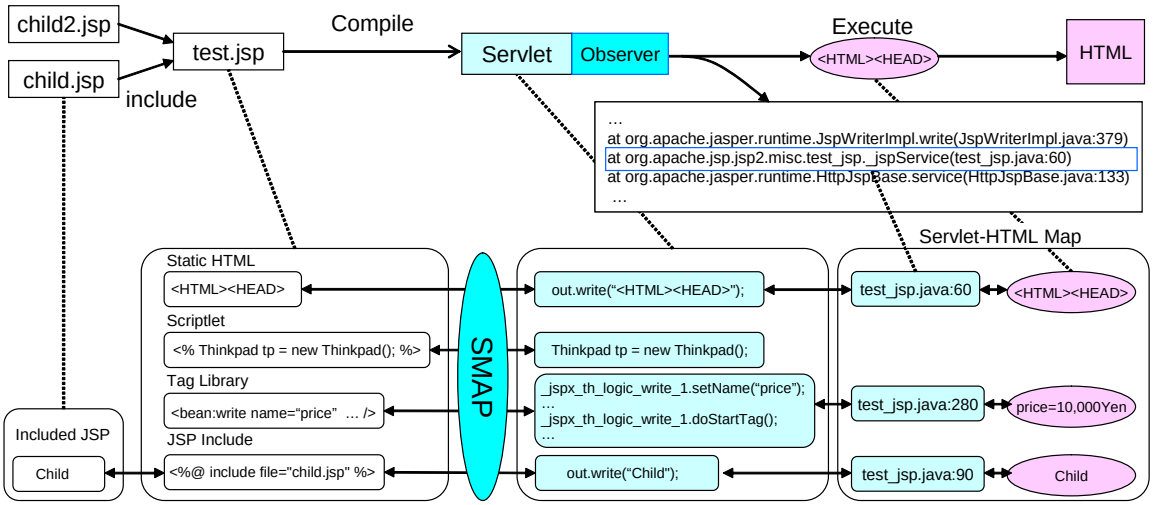

図 1: 出力処理の監視による動的 Web コンテンツと HTML の関連付け

しかし，動的 Web コンテンツ内でタグライブラリが用いられたり，ポータル [1] 等の様にコンテンツの入れ子構造が複雑になっていた場合，上記の様な手法で修正箇所を特定する事は困難である．他にも，Servlet をステップ実行して修正箇所を特定する手法なども存在するが，実際の修正作業を開始するまでに必要な作業量が非常に大きいという問題は同様である．

本稿では，この様な問題を解決するため，JSP や Servlet の出力処理に着目し，出力元となるコードの位置と，その出力結果との関連付けを行う事で，HTML 内での位置から，該当箇所を生成した動的 Web コンテンツを特定する手法を提案する．提案手法を用いる事により，Web サイトの評価・検証結果から，修正・変更の対象となる動的 Web コンテンツを特定する事が容易となり，開発者の労力を軽減する事が期待される．

2.1 修正位置特定手法

図 1 に JSP から HTML が生成される過程を示す．JSP を用いた動的コンテンツへのリクエストがサーバーに到着すると，まず JSP は Servlet にコンパイルされる．この際，JSP 内の各要素が Servlet 内のどの位置に対応するのかという情報が SMAP(Source Map)[12] として保存される．この SMAP の情報を用いる事で，Servlet 中の行番号から JSP 内での位置を知る事が可能となる．ただし，JSR-045[12] に対応していないサーバーでは，これらの情報は Servlet 内にコメントとして保存される事が多い．その場合，あらかじめ Servlet 内のコメントを解析し，SMAP に相当する関連付け情報を作成しておく必要がある．

次に，Servlet は Java の Class ファイルへとコンパイルされ，実行される．その結果，最終的な出力である HTML が得られる．本稿では，この Servlet の出力処理を監視する事で，出力元となるコードの位置と，その出力結果との関連付けを行う．具体的には，Servlet の出力処理を行う Writer クラスをラップして出力処理を監視する Observer を用意し，出力処理が呼ばれるたびに，出力された HTML の内容と，StackTrace 情報から得た Servlet 内での位置を Servlet-HTML Map として保存する (図 1)．

最後に，得られた Servlet-HTML Map と SMAP 情報に基づき，出力された HTML と出力元となるコードの位置の情報を含んだ HTML-Source Map を生成する．HTML-Source Map の生成に際しては，ServletContext や ServletRequest の持つ情報も利用して，JSP などが実際に存在するディレクトリの情報なども取得しておく．

図 2 に，HTML-Source Map を XML 形式で出力した例を示す．この HTML-Source

```
<?xml version="1.0" encoding="UTF-8" ?>
<SourceHtmlMap>
<fragment>
 <java>
  <rootdir>C:¥server¥webapps¥examples</rootdir>
  <servletpath>/test</servletpath>
  <realpath>
    C:¥server¥webapps¥examples¥snoop.jsp
  </realpath>
  <depth count="1">
   <class>
    <name>org.apache.jsp.snoop_jsp</name>
    <method>_jspService</method>
    <line>43</line>
   </class>
   <jsp>
    <name>snoop2.jsp</name>
    <actualfile>
      C:¥server¥webapps¥examples¥snoop2.jsp
    </actualfile>
    <line>1</line>
    </jsp>
  </depth>
 </java>
 <htmldata>
  <from line="2" pos="1" /><to line="3" pos="0" />
  <contents><![CDATA[<html>
]]></contents>
 </htmldata>
</fragment>
   ⋮
```

図 2: HTML-Source Map の例

Map を用いる事で，HTML 中の行数で指定された修正箇所から，実際に修正が必要な動的 Web コンテンツを知る事が可能となる．例えば，図 2 の例から，HTML 中の 2 行目 (“<html>”) を修正するためには，“c:\server\webapps\examples\snoop2.jsp” の 1 行目を修正すれば良い事がわかる．

ここで，リクエスト中の ServletPath に基づいて，最初にリクエストを処理すべく呼び出されるのは図中の <realpath> で与えられる “snoop.jsp” である．しかしながら，“snoop.jsp” 中で “snoop2.jsp” のインクルードが行われているため，実際に修正が必要なファイルは <actualfile> で与えられる “snoop2.jsp” となる．

以上のように，提案手法を用いる事で，HTML 内での位置から，該当箇所を生成した動的 Web コンテンツを特定する事が可能となる．

2.2 フィルターを用いた実装例

前節で述べた提案手法を，Filter クラスを用いて実装した例を図 3 に示す．この実装では，サーバーにリクエストが到着し，ServletRequest および ServletResponse が生成されると，Observer Filter を用いて ServletResponse をラップする．ここで生成された “Response with Observer” は，JSP やサーブレットから Writer の取得を要求されると，本来の Writer をラップし出力処理を監視する Observer を返す．

この様に Filter を用いる事で，Servlet/JSP コンテナに変更を加える事なく，提案手法を実装する事が可能となる．また，Filter を用いた実装の利点として，図 3 の様に，提案手法により得られた HTML-Source Map を HTML 文書にコメントとして付加することも可能となる．HTML-Source Map を HTML 文書内に持つことで，各種評価・検証の際に，修正が必要な動的 Web コンテンツを特定する事も可能となる．

3 提案手法の利用例

3.1 Source-HTML Map Viewer

図 4 に，Source-HTML Map の情報を用いて，生成された HTML とそのブラウザ上での表示，および対応箇所を生成した動的 Web コンテンツの情報を確認できる Viewer の例を示す．

図 4 中，左側のブラウザ上で任意の要素を選択すると，右上の HTML 表示領域内で対応する HTML 要素が反転表示される．同時に，右下の Source 情報表示領域に，選択箇所を生成した JSP 等の情報が表示される．同様に，HTML 表示領域内で任意の HTML 要素を選択すると，ブラウザ上で対応箇所がハイライトされ，選択要素を生成したコンテンツの情報が右下に表示される．

この様な Viewer を用いる事で，生成された HTML のブラウザ上での表示を確認しながら修正したい箇所を探索し，修正が必要な動的 Web コンテンツの位置を容易に把握する事が可能となるため，コンテンツの修正・変更作業の効率を向上できると考えられる．

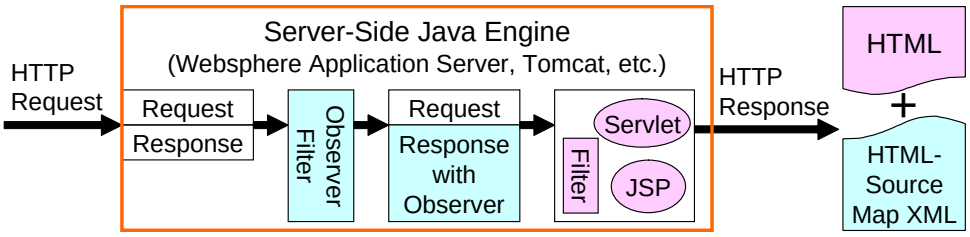

図 3: Filter を利用した Observer の実装例

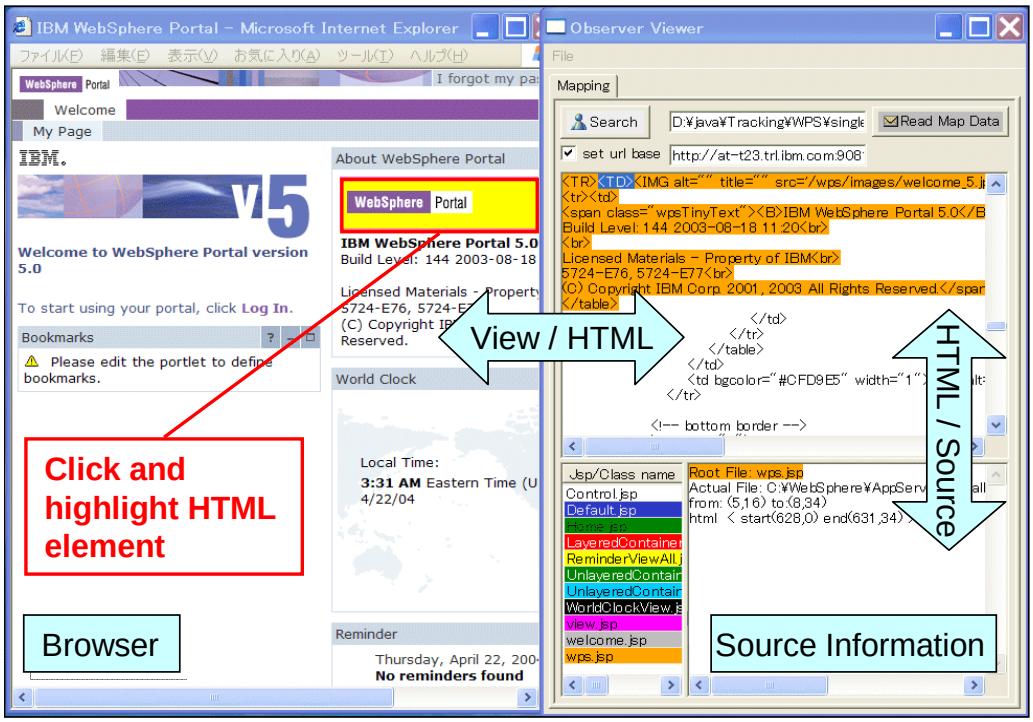

図 4: Source-HTML Map Viewer

3.2 アクセシビリティチェッカーとの連携

アクセシビリティチェッカーの多くは，HTML を対象にチェックを行い，その行数等の位置情報に基づいてチェック結果の出力を行っている．

図 5 は WebKing[13] と呼ばれるチェッカーのチェック結果の例である．このチェッカーを拡張し，HTML と同時に送信されている HTML-Source Map の情報から，HTML 中での位置に対応する動的 Web コンテンツの情報を取得するよう変更を行った．その結果，従来の行数や文字数などの位置情報 (図 5) に加えて，図 6 に示すような動的 Web コンテンツの情報がチェック結果に加えられ，チェック結果の確認や修正作業を効率良く行う事が可能となる．

```
<Error>
  <Page>./myportal/!ut/p/.cmd/cs/.ce/
    7_0_A/.s/7_0_75/_s.7_0_A/ 7_0_75
  </Page>
  <Line>98</Line>
  <Column>31</Column>
  <Severity>Severe Violation</Severity>
  <Message>Section 508 - 1194.22: Spacer
    image should have ALT attribute = "".
  </Message>
</Error>
```

図 5: 従来のチェック結果の例

```
<Source type="File">
  <Name>c:¥server¥installedApps¥test¥
    wps.ear¥wps.war¥themes¥html¥Default.jsp
  </Name>
  <Line>26</Line>
</Source>
```

図 6: 追加される情報の例

4 まとめ

本研究では、JSP や Servlet の出力処理に着目し、出力元となるコードの位置とその出力内容の関連付けを行うことで、サーバーサイド Java を用いた動的 Web コンテンツの開発・修正を支援する手法の提案を行った.

提案手法を用いる事により，Web サイトの評価・検証結果から，修正・変更の対象となる動的 Web コンテンツを特定する事が容易となり，開発者の労力を軽減する事が期待される.

また，提案手法を動的 Web コンテンツの依存解析手法 [14] などと組み合わせる事で，変数の値が設定された場所などを絞り込む事も可能になると考えられる．今後，さらなる検討を行っていきたい.

参考文献

[1] C. Wege, "Portal Server Technology," *IEEE Internet Computing*, vol. 6, pp. 73–77, May/June 2002.

[2] F. Ricca and P. Tonella, "Analysis and Testing of Web Applications," *Proceedings of ICSE 2001*, pp. 25–34, May 2001.

[3] S. Elbaum, S. Karre, and G. Rothermel, "Improving Web Application Testing with User Session Data," *Proceedings of ICSE 2003*, pp. 49–59, May 2003.

[4] Y. Deshpande, A. Chandrarathna, and A. Ginige, "Web Site Auditing: First Step Towards Re-engineering," *Proceedings of SEKE '02*, pp. 731–737, July 2002.

[5] L. Paganelli and F. Paternò, "Intelligent Analysis of User Interactions with Web Applications," *Proceedings of IUI '02*, pp. 111–117, January 2002.

[6] J. I. Hong, J. Heer, S. Waterson, and J. A. Landay, "WebQuilt: A Proxy-based Approach to Remote Web Usability Testing," *ACM Transactions on Information Systems*, vol. 19, pp. 263–285, July 2001.

[7] J. Vanderdonckt, A. Beirekdar, and M. Noirhomme-Fraiture, "Automated Evaluation of Web Usability and Accessibility by Guideline Review," *Proceedings of ICWE 2004*, pp. 17–30, July 2004.

[8] H. Takagi, C. Asakawa, K. Fukuda, and J. Maeda, "Accessibility Designer: Visualizing Usability for the Blind ," *to be appeared in Proceedings of ACM ASSETS 2004*, October 2004.

[9] J. Brewer, "Web Accessibility Highlights and Trends," *Proceedings of International Cross-Disciplinary Workshop on Web Accessibility 2004*, pp. 51–55, May 2004.

[10] B. Beier and M. W. Vaughan, "The Bull's-Eye: A Framework for Web Application User Interface Design Guidelines," *Proceedings of ACM CHI 2003*, pp. 489–495, April 2003.

[11] Sun Microsystems, "JSP Standard Tag Library (JSTL)." `http://java.sun.com/products/jsp/taglibraries.html`.

[12] Java Community Process, "Debugging Support for Other Languages Specification." `http://jcp.org/aboutJava/communityprocess/final/jsr045/`.

[13] PARASOFT, "WebKing." `http://www.parasoft.com/`.

[14] F. Ricca and P. Tonella, "Web Application Slicing," *Proceedings of IEEE ICSM '01*, pp. 148–157, November 2001.