平成14年度学士論文

安全性の高いセッション管理方式のServletへの導入

東京工業大学 理学部 情報科学科
学籍番号 99-2270-6
松沼 正浩

指導教官
千葉 滋 助教授

平成15年2月28日

概要

現在のインターネット社会において、多くの Web サーバが静的な情報だけでなく動的な情報を扱い、それをクライアントへ配信している。Web サーバに動的な情報の配信を可能にしたのは、サーバーサイドプログラミングに代表されるサーバーサイドテクノロジーと呼ばれる技術の進歩によるところが大きい。サーバーサイドプログラミングの代表的なものとしては Perl、ASP、Servlet・JSP などが挙げられるが、いずれもクライアントからの入力をリクエストと共にサーバ側で受け取り、特定のプログラムを起動し、入力を反映させた HTML ファイルを作成・送信するという動作に変わりはない。

Web サーバが動的なページを扱うようになり、クライアントによって送信する情報を変化させるようになった事で、今度はサーバ側にクライアントの識別を行う必要性が生じてきた。このクライアント識別の為に現在多くのサーバーサイドプログラミングで採用されているのがセッション管理と呼ばれる概念である。セッション管理の一般的な方法として知られているのは、セッション ID を用いた管理方法である。これはクライアントが Web サーバにログインした際に、クライアント毎に異なるセッション ID と呼ばれる情報を発行し、その情報を元にクライアントの識別・管理をサーバ側が行うという方法である。

現在、多くのサーバーサイドプログラミングでは、上記のセッション ID を格納する方式として、cookie に格納する方式、もしくは encodeURL してリクエスト URL に埋め込む方式を採用している。しかしそのどちらの方式も、それぞれ問題点（セキュリティ上の問題点や、使用上の問題点など）を抱えているといった実情である。

本研究では、サーバーサイドプログラミングの一つであり Java 言語を用いた Servlet・JSP、実行環境である Tomcat に焦点を当てて研究し、セッション管理のセキュリティを向上させた Tomcat 改を提案する。Tomcat 改ではセッション管理に用いる ID の格納方式として上述の二種類に加え、新たにセキュリティ性の高い Form の hiddenParameter フィールド

を追加した。また状況に応じて三つの方式を使い分ける事を可能にする為、プログラマーが管理方式を選択出来るようにした。なお実験においては、Tomcat 改使用時におけるサーバ側のスループットを計測することで、Tomcat との性能比較を行った。

謝辞

本研究を進めるにあたり、研究の方向付けや論文の組立て方についての助言をしていただいた指導教官の千葉先生に感謝いたします。

同研究室所属の佐藤芳樹氏には、論文の組み立てからシステムの設計・実装に至るまでの様々な助言を頂き大変感謝致します。

また筑波大学の横田大輔氏、東京工業大学大学院の西沢無我氏、中川清志氏、栗田亮氏には多くの助言を頂きました。東京大学の光来健一氏には論文のスタイルファイルを作って頂きました。東京工業大学大学院の宇崎央泰氏には実験で使用したベンチマークソフトの使用について多くの助言を頂きました。以上の方々に重ねて御礼を申し上げます。

目 次

図 目 次

表 目 次

第1章　はじめに

インターネットが一般に普及してから10年程しか経っていないが、利便性の良さから爆発的に日常生活へと定着してきた。普及当初、インターネットはサーバ側に置かれた静的な情報のやり取りの為に用いられており、現在では当たり前に行っている情報の検索・オンラインショッピングに代表される動的な情報を扱う事は出来なかった。インターネットの普及を支えてきたのは、これらの動的な情報を扱えるようになった事にあると言っても過言ではない。動的な情報を扱えるようになった主な原因は、ASPやServlet・JSPのようなサーバーサイドプログラミングに代表されるテクノロジーの発展にある。この発展によってサーバーサイドでプログラムを実行させる事が可能となり、クライアントの入力を解析し、動的なページの作成を行なえるようになったという訳である。

1.1　インターネットの実情

インターネットは元々アメリカにおいて、東西の緊張関係が高まっていた時代の1963年、軍事関係のシンクタンク、ランドコーポレーションの国防総省への提案をきっかけとして誕生したものである。当初の目的は国家の中枢地域が壊滅しても国家機能と軍事情報網を切断させない為に、コンピュータネットワーク上に情報基地を分散させて、個々の基地を結び全体として機能するシステムを作成するというものであった。70年代になり、このシステムが学術分野に採用され、ネットワークで結んだ大学間で情報の共有や交換が行われるようになった。インターネットが商用として用いられるようになったのは1991年、情報スーパーウェイ構想に端を発している。93年には、より操作が簡単になった検索ソフトウェアが登場し、インターネットの急速な普及が実現した。

元々インターネットは、サーバがクライアントからのリクエストを受けて要求されたファイルを返すという仕組みのものであった為、元々サーバ側で作成されて保存されているHTMLファイルのような静的な情報し

かやり取り出来なかった。これを解決し、動的な情報についても扱う事を可能にする為に作り出されたのがサーバーサイドテクノロジーと呼ばれる仕組みである。

まずインターネットの仕組みについて説明し、続いて動的ページ作成の仕組みについて説明する。

1.1.1　インターネットの仕組み

インターネットとは、TCP/IP というネットワークプロトコルを基盤にした、世界規模のネットワークであり、WWW(World Wide Web) の仕組みを利用して情報のやり取りが行われている。

- **WWW**
  WWW は、1989 年に CERN(European Particle Physics Laboratory) によって開発された分散型情報共有方式であり、Http サーバと呼ばれるサーバプログラムと、Web ブラウザと呼ばれるクライアントプログラムによって実現されるクライアント/サーバ方式のシステムである。WWW を用いて参照する事の出来る情報は、基本的にタグと呼ばれるマークを用いる HTML(Hyper Text Markup Language) 形式で作成された文書ファイルである。HTML ファイルと音や画像などのデータを WWW サーバに格納する事で、その情報を WWW クライアントに提供する事が可能になる。

1.2　動的 Web ページの作成

現在サーバが動的な情報をやり取りする為の手法として用いられているのは、クライアントサイド技術とサーバーサイド技術の二つである。

- **クライアントサイド技術**
  クライアントサイド技術とはクライアント側で、特定のプログラムを実行させて動的なページを扱う為の技術であり、JavaScript や VBScript などがこれに当たる。

- **サーバーサイド技術**
  サーバーサイド技術とはサーバ側で、特定のプログラムを実行させて動的な情報を扱う為の技術であり Servlet・JSP、サーバーサイドスクリプト言語などがこれに当たる。

1.2.1　サーバーサイドスクリプト

サーバーサイドスクリプトとは、Web サーバが Web ブラウザからの要求に応じて、サーバ側でスクリプト言語を起動する為の仕組みである。元々、リクエストに応じて静的なファイルを送信するだけであった Web サーバがサーバーサイドスクリプトを用いる事で、スクリプトの処理結果に基づいて動的にファイルを作成し送信する事が可能となったのである。サーバーサイドスクリプト言語の代表的なものとして ASP、PHP、Perl などが挙げられる。

- **Perl**
  Perl とは、プログラマの作業効率を重視したインタープリタ型のスクリプト言語であり、テキスト処理に重点が置かれて開発された言語である。Perl は時代が進むにつれ、言語仕様が拡張され、ファイルそのものの操作を始め、システムプログラミングやネットワークプログラミングなども可能な多くの機能を備えるようになった。最近では WWW においても、クライアントからの入力を受け取り動的なページを作成する為のシステムが、CGI と Perl スクリプトを利用して構築されている。また、最新 Version の Perl-Ver.5 ではオブジェクト指向的な拡張が取り入れられている。

- **ASP**
  ASP(Active Server Pages) とは、Windows NT/2000 等で利用可能なスクリプトを Web サーバ側で動作させ、HTML ファイルを作成して表示させるプログラムである。利用可能なスクリプト言語は、標準的なものでは VBScript、JScript などが挙げられ、Perl の追加も可能となっている。

- **PHP**
  PHP(Hypertext Preprocessor) とは、Html ファイル埋め込み型のサーバーサイドスクリプト言語であり、Perl で書かれた小さな CGI ラッパーを C 言語で書き直しを図ったのが始まりとなっている。PHP はコンパイルを必要とせずにソースを修正してすぐに実行可能となっている。この為、生産性は高くなっているが、スクリプト言語であるので実行の度に文法解析を行う事になる点は免れない。

1.2.2　Servlet・JSP

Servlet・JSP とは、Web サーバ上で実行されるモジュール化された Java プログラムである。Servlet・JSP は Java で記述されている為、特定の OS やハードウェアに依存する事が無く、ServletAPI を実装したあらゆる Web サーバで稼動させる事が可能である。CGI などのサーバサイドプログラミングとは異なり、一度呼び出されるとメモリに常駐する為、高速な処理が可能となっている。また、データを永続的に扱える為、複数のユーザ間で情報を共有する事も可能となっている。

1.3　現状の問題点と解決策

以上の様に、サーバーサイドテクノロジーが発展し、動的なページを含む多くの情報をサーバ側が扱うようになった事で、今度はクライアントからの入力情報を漏洩させない為の情報管理に関するセキュリティ性を向上させる必要性が生じてきたのである。クライアントからの入力情報の管理法として、セッション ID と呼ばれる ID 番号を用いてクライアントを管理する方法が一般的であるが、多くのサーバーサイドプログラミング言語ではその方法にセキュリティ上の問題点を含んでいる。

Servlet・JSP は数多くあるサーバーサイドプログラミング言語の一つであり、他の言語同様にセッション管理法のセキュリティ問題点が存在する。しかし、Servlet・JSP は Java を用いたプログラミング言語である為、高い可搬性とパフォーマンス性を保持している。これらの点から Servlet・JSP は今後、サーバーサイド分野において更に発展する言語であると考えられる。そこで本研究では、Servlet・JSP のセッション管理法の問題点を解決し、そのセキュリティの向上を目指す事を目標とした。

Servlet・JSP では、サーバ側でクライアントの動的な情報を維持する為にセッション管理を行っているが、そのセッション管理法は大きく分けて二種類存在する。一つ目は cookie 又はリクエスト URL 書き換えを利用しての ID 管理法によるものであり、二つ目は Form や cookie を用いて直接データを管理する方法である。しかし、そのどちらもが以下に挙げるセキュリティ上、又は使用上における脆弱な部分を持っている。

- cookie の脆弱性
  クロスサイトスクリプティング [6] による cookie 情報の第三者への漏洩。

- リクエスト URL 書き換えの脆弱性
  Referer 機能 [5] による URL 内容の他サーバへの漏洩。

- Form の問題点
  情報送信を Form タグを使用して行う為、リンクタグを使用出来なくなる。

この為、セキュリティに関心を持たないプログラマーがサーバを立ち上げ、セッション管理を行った場合には、クライアントの情報が悪意を持つ第三者に漏洩する可能性が生じる。

そこで本研究では Servlet・JSP におけるセッション管理法の一つである ID 管理法に、セキュリティ性の高い Form の hiddenParameter 方式を加え、且つその利便性向上させることを目標とした。これによってサーバ作成者がセキュリティに関心を置かずに設計しても、セッション管理時に高いセキュリティを維持できるようにする事が目的　である。
以下ではまず次章で、サーバーサイドプログラミングの一つである Servlet・JSP のセッション管理の問題点について述べる。つづいて 3 章では、2 章で挙げた問題点の解決法を提案し、4 章でその解決法を実装したプログラムの性能についての実験を行う。最後に 5 章で本論文をまとめる。

第2章　Servlet・JSPにおけるセッション管理の問題点

2.1　セッション管理

WWWサービスで使用されるHTTPプロトコルでは、クライアントからWebサーバへのリクエストがある度に、クライアントとサーバ間でコネクションが成立し、リクエストに対するレスポンスが行われ、コネクションは切断される。その為Webサーバに対する複数リクエストが同一クライアントから送信されたものであるかどうかを判断することは不可能である。

一般的なWebブラウジングでは特に問題にはならないが、電子商取引サイトのようにWebサーバにユーザがログインしてからログアウトするまで、ログイン情報を保持したままページを移管するには、このままでは問題がある。そこで、クライアントとサーバ間でその情報を保持し、アクセス制御を一つの集合体として管理する仕組みが必要となる。この仕組みをセッション管理と呼んでいる。

このセッション管理を実現する一般的な方法に、Webサーバが発行するセッションIDを利用するものがある。これは、クライアントがWebサーバにログインした際、セッションIDを発行し、以後、その情報を基にクライアントとWebサーバのセッション管理を行う方法である。

2.2　Servlet・JSP

この章を用いてサーバーサイド技術の一つであるServlet・JSPと、それを使用可能なサーバであるTomcatについて詳しく説明を行う。

2.2.1　Servlet・JSP の利点

Servlet・JSP は Java を用いて作られているサーバーサイド技術である為に、動作が遅いと言われている。しかし、同時に数百以上の多くのリクエストが来るような場合においては、Perl などに比べて数倍の性能を発揮すると言われている。

今後更に発展すると考えられるこの分野において、多数のリクエスト時における性能は重要な要因であると考えられる。その為、本研究では Servlet・JSP を研究対象とした訳である。

2.2.2　Servlet とは

Servlet とは、Web サーバと連携して Java のプログラムを動作させる為の仕組みであり、サーバサイドでの Java 実行形態の一つである。サーバ側で Java を動作させるだけならば、通常の Java Application を動作させるだけでも良い訳であるが、Servlet は主に Web アプリケーションでの使用を想定し、Web サーバと連携してクライアントと情報のやり取りを行いながら、処理していくという点で優れているわけである。

通常の Web ページは、Web サーバ上のファイルとして格納されていて、クライアントからのリクエストに応じ、Web サーバが指定されたファイルの内容をクライアントに送信する仕組みになっている。その為 Web サーバ単独でページの提供を行った場合、その内容は固定された静的な物のみになってしまう。それを解決し動的なページを提供する為に、クライアントからの入力をサーバ側で Java を用いて処理し、その入力情報を用いて動的なページを生成し送信する為の仕組みが Servlet という訳である。

Servlet の他に Java 言語をベースにして動的ページを生成する為のシステムとして、Sun Microsystems 社と Netscape Communications 社が開発した JavaScript というスクリプト言語がある。Servlet と JavaScript との違いは、JavaScript は Java と似た文法を用いて HTML 中にコードを記述して動的な処理を行わせるものであるが、実行は Web ブラウザで行われるという事が最大の違いである。Servlet は Web ブラウザ上で実行されるのでは無く、Web サーバ側で Java のコードが実行され、その実行結果を HTML に反映させクライアントに送る訳である。この為に JavaScript で書かれたページは、ページのソース上に JavaScript コード表示されるが、Servlet で書かれたページのソースを見ても Servlet のコードは一切表示されることは無い。

次に Servlet で作成した簡単なページのソースファイルを紹介し、それについて簡単に説明を行う。

- **Servlet の記述**
  一般的な Servlet は javax.servlet.GenericServlet クラスとして定義されており、HTTP を処理する為に javax.servlet.http.HttpServlet という抽象クラスを継承して作成する必要がある。
  以下に、”HelloWorld”と出力させる簡単な Servlet のソースコードを表示する。

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {
  public void doGet(HttpServletRequest request,HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String doctype = "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
                     "Transitional//EN\">\n";
    out.println(doctype +
                "<HTML>\n" +
                "<HEAD><TITLE>HelloWorld</TITLE></HEAD>\n" +
                "<BODY>\n" +
                "<H1>HelloWorld</H1>\n" +
                "</BODY></HTML>");
  }
}
```

HttpServlet クラスは抽象クラスであるので、少なくともメソッドの一つをオーバーライドする必要があり、処理したい HTTP リクエストに応じてメソッドを選択する。上の例では Http の GET リクエストを処理する為に、doGet メソッドをオーバーライドしている。ここで仮に POST リクエストにも対応可能にする為には、doPost メソッドをオーバーライドさせ、その中で doGet メソッドを呼び出すようにするのである。

続いて、Servlet の拡張とも言える JSP についての説明をおこなう。

表 2.1: HttpServlet クラスのオーバーライドする為のメソッド

メソッド	内容
init	サーブレットの初期化
destroy	サーブレットの終了処理
getServletInfo	サーブレットに関する情報を伝える
doGet	HTTP の GET リクエストの処理
dePost	HTTP の POST リクエストの処理
doPut	HTTP の PUT リクエストの処理
doDelete	HTTP の DELETE リクエストの処理
doOptions	HTTP の OPTIONS リクエストの処理
doTrace	HTTP の TRACE リクエストの処理

2.2.3　JSP とは

JSP(Java Server Pages) とは Servlet ファイルから HTML 部分を分離させ、編集を容易なものにする為に開発されたものである。その為 JSP と Servlet との間で実行できる機能に差は無い。JSP の動作原理を説明すると、クライアントが JSP ファイルを呼び出す際には、HTML ファイル同様に「***.JSP」というファイル名を指定した形式でリクエストが Web サーバに向け送信され、JSP に対応した Web サーバが受信したリクエストの URL を解析し、「.jsp」という URL 指定がなされていた場合には、Web サーバ自身は処理をせずに JSP プロセッサに処理をまかせるのである。すると JSP プロセッサは指定された JSP ファイルを解析し、実行して結果をクライアントに送信するという動作を行うのである。

またこの JSP プロセッサの具体的な動作手順は、指定された JSP ファイルを解析し、これを同じ動作を行う Servlet ファイルに書き換えてコンパイルし、出来たクラスをロード、実行という処理を行っているのである。つまり、全ての JSP ファイルは Web サーバ側で Servlet ファイルに書き換えられ、Servlet ファイルとして実行されているのである。そのため JSP は、初めてそのファイルが呼び出される場合にはソースファイルの生成、コンパイル、実行とを行う為に処理が遅くなる事が生じるのである。その為、処理時間が大きくなると考えられるが、二度目のリクエストからは、始めに作成された Servlet クラスのファイルを呼び出すだけになる為、通常の Servlet ファイルとほとんど変わらない処理速度となる

事が可能となっている。
　次に、JSP で作成する簡単なページのソースファイルを紹介し、それについて簡単に説明を行う。

- **JSP の記述**
  JSP のソースコードは、HTML ファイルの中に特別なタグを用いて記述するという形式で記述する。
  以下に Java で九九の掛け算を実行させ、その結果を HTML ファイルにする JSP ソースコードを表示する。

```
<%@ page contentType="text/html; charset=euc-jp" %>
<%! int i,j;%>
<HTML>
<HEAD>
<TITLE>9 × 9 JSP</TITLE>
</HEAD>

<TABLE BORDER=1>
<% for(i=1;i<10;i++) { %> <TR>
  <% for(j=1;j<10;j++) { %>
    <TD> <%= i %> × <%= j %> = <%= i * j %></TD>
  <% } %> </TR>
<% } %>
</HTML>
```

このコード中に、HTML ファイルに見かけないタグは以下の四種類である。

- **<%@ … %>**
  このタグは指令タグと呼ばれ、コンテナに対しての JSP の動作設定などを行う為に、JSP のコード中に指令を記述する為のものである。指令はリクエストの内容とは関係無く, 主にコンパイル (トランスレート) 時に効果を発揮する。このコードの場合には page 指令が書かれている。page 指令とはページ内容に関する属性を設定する為の指令内容であり、この場合はこのページが contentType 属性を使用し、ドキュメントは HTML であることをブラウザ側に送信する事を示し、また送信文字コードは EUC-jp とすることを設定する事を示している。この page 指令の他に、他のファイルを JSP に挿入する為に include 指令や、独自のタグライブラリを使用する為の taglib 指令などがある。

- <%! … %>
  このタグは宣言タグを呼ばれ、そのページの Java 言語部分で使用する変数や関数の宣言を行う。ここでの記述には基本的に Java の記述を用いる。JSP ページが JSP コンテナによって初期化される時に一緒に実行される。つまり、ページがクライアントから呼び出されたかどうかには関係無く、ページ中の他の部分に先立って一番早くに実行される事になる。よってこの部分でクライアントへの出力を行うような処理をする事は出来ない。

- <%= … %>
  このタグは式タグと呼ばれ、このタグ内に書かれた「式」を評価、計算して文字列に変換して HTML 上に表示させるというものである。仮にここで文字列に変換出来ないような処理を持つ式を記述した場合には、実行時に例外 ClassCastException が発生してしまう事でなる。このタグに書く内容も基本的に Java の記述を用いる。

- <% … %>
  このタグはスクリプトレットと呼ばれ、Java によるスクリプトを記述する。この部分には自由に Java のコードを記述する事が可能であり、基本的に上から下へと順番に実行されていく。スクリプトレットのコードは一つのタグ内で完結している必要は無く、全てのタグを併せて完結していれば良いのである。よってタグの途中で HTML を出力したい場合には、途中で一旦タグを中断して HTML や式などを記述する事が可能である。

2.2.4　Tomcat とは

Tomcat とは、JSP･Servlet を使用する事のできるオープンソースの Web サーバ兼 Servlet エンジンであり、Jakarta と呼ばれるプロジェクトの中の一つのサブプロジェクトで開発されている。Tomcat の他にも JServ、JRun、JSWDK などが Servlet エンジンとして挙げられるが、現段階においては、設定ファイル自体が XML で書かれているなど最新の技術を駆使している点、各種起動パラメータの設定なども容易である点、Apache との連携が容易な点などから Tomcat が今後の Web サーバの有力な候補とみなされている。また Jakarta プロジェクトでは、独自のタグライブラリやコードのビルド用プログラム (Ant) などが含まれ、Tomcat では JSP1.1

と Servlet2.2 をインプリメントしている。

以下に Tomcat の、クライアントからのリクエストを受けてから HTML を送信するまで流れを図を用いて示す。

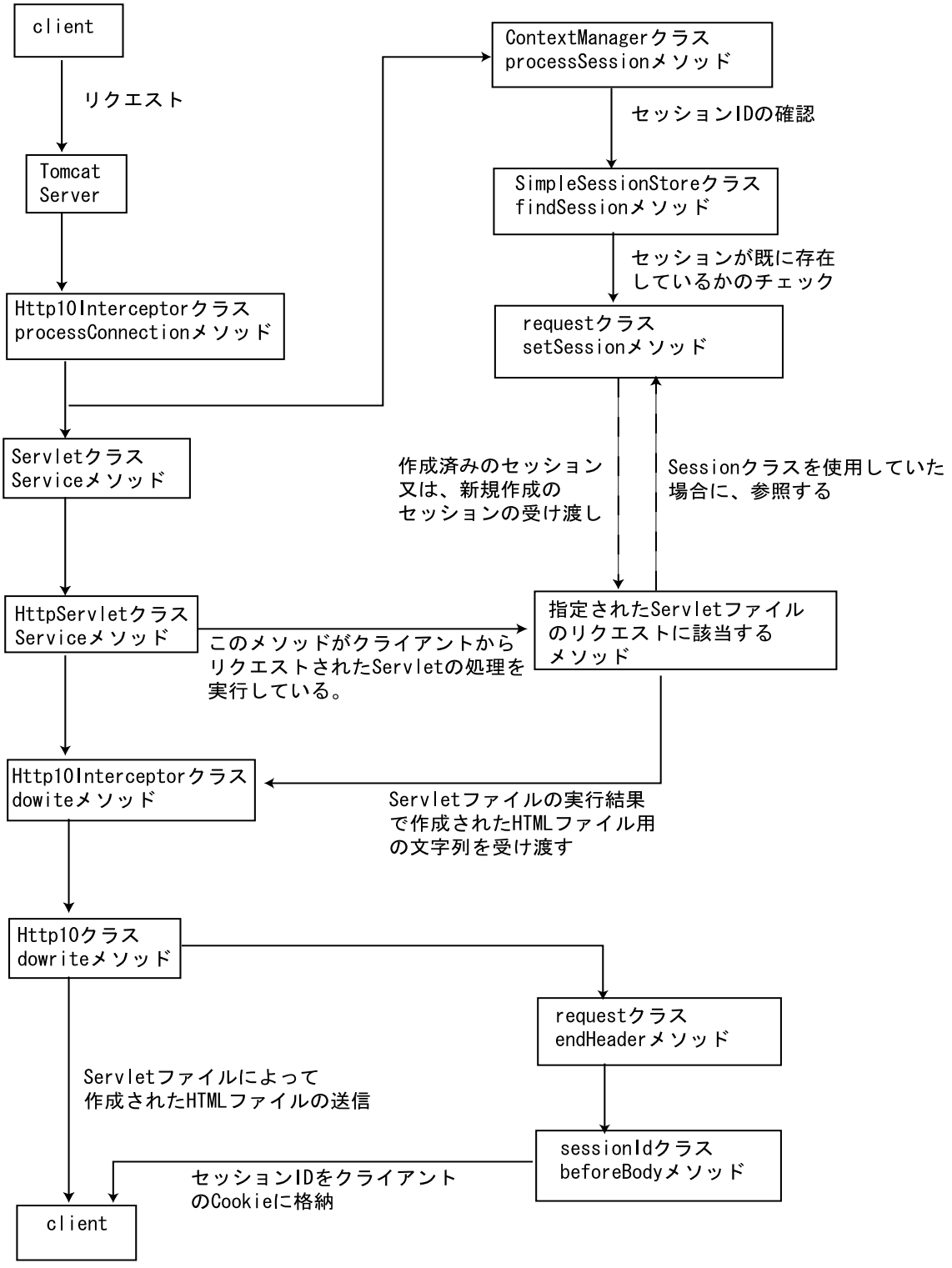

図 2.1: Tomcat の動作説明図

2.3　現在の Servlet のセッション管理方法

現在の Servlet では、セッション管理の方法として、主に二種類の方法が存在する。一つ目の方法は、Session クラスを用いて情報のやり取りを行う方法 (具体的にはセッション ID と呼ばれる、情報を引き出す為の鍵をやり取りする方法) であり、二つ目の方法は Form などの、入力情報をサーバに送信する仕組みを利用して、情報を直接受け渡す方法である。

- **Session クラスを用いる方法**
  Session クラスを用いる場合には、セッション ID をページ間でやり取りする必要がある。このやり取りを行う為の方式には、cookie 方式と encodeURL 方式の二種類が採用されており、Tomcat のデフォルトの設定では cookie 方式が採用されている。encodeURL 方式は、ソースコードでオプション指定をしていて、且つクライアントの Web ブラウザで cookie が使用不可である場合にのみ切り替わる事になっている。

- **情報を直接渡す方法**
  直接情報を受け渡す方法としては、Form の hiddenParameter に情報を入れて渡す方法と、cookie に情報を入れて渡す方法などが良く知られている。これらの方法は特に Servlet 内の特別なクラス (メソッド) を使用する事は無く、プログラマーが自分でセッション管理の為の記述をする事になっている。

以下でそれぞれの方法についての説明と、それぞれの特徴について説明する。

2.4　Session クラスを用いる場合

ここで、Session クラスを用いる場合の、二種類の方式 (cookie 方式と encodeURL 方式) についてそれぞれ説明し、その特徴と問題点を指摘する。

2.4.1　cookie 方式

cookie 方式とは SessionID をクライアントの cookie に

```
jsessionid=**********
```

という形で保存する仕組みである。
この方式は、次に説明するクロスサイトスクリプティングを用いての攻撃に弱いという性質がある。

- **cookie とは**
  cookie とは Web サイト提供者が、Web ブラウザを通じてサイト訪問者のコンピュータに一時的にデータを書き込んで保存させる仕組みの為の物である。cookie には、ユーザーに関する情報や最後にサイトを訪れた日時、そのサイトの訪問回数などの訪問者固有の情報などを記録しておく事が出来る。また、cookie はユーザーの識別にも使われており、認証システムや、WWW によるサービスをユーザーごとにカスタマイズする為などのパーソナライズシステムの要素技術として利用されている。

  cookie 自体の始まりは Netscape Communications 社が同社の Web ブラウザに組み込んだのが始まりとなっており、標準化団体によって正式に規格化されているわけではない。しかし、多くの Web ブラウザは cookie を標準的にサポートしている為、事実上の世界標準となっている。

  一つの cookie には最大で 4096 バイトのデータを記録する事ができ、最大で 300 の cookie を保存できる。また一台のサーバが同じコンピュータに対して発行する事ができる数は 20 個までと制限がある。cookie はそれぞれ有効期限が設定されており、その期限を切れた cookie は消滅する仕組みになっている。cookie は、Web ブラウザを終了しても、継続可能な唯一のセッション管理法である。

- **cookie 方式の場合のプログラム例**
  以下に、cookie 方式を用いてログイン名とパスワードを Session クラスに収めて、次のページに引き渡す Servlet のソースコードを示す。

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SAMPLE1 extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
    throws ServletException, IOException {
```

```
        String log,pass;
        log = request.getParameter("logname");
        pass = request.getParameter("passwd");
              //FORM からのログイン名とパスワードの取得
        HttpSession session = request.getSession(true);
              //セッション作成
        session.setAttribute("LOGNAME" ,log);
        session.setAttribute("PASSWORD",pass);
               //セッションへのデータの格納。
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println( … 省略 … );
      }
    }
```

2.4.2　クロスサイトスクリプティングによる cookie の脆弱性

この章ではクロスサイトスクリプティング [6][1] による、cookie 格納データの漏洩について説明する。

- **クロスサイトスクリプティングとは**
  クロスサイトスクリプティングとは 2000 年 2 月に CERT CoorinationCenter[2] や, Microsoft[3] から発表された問題であり、クライアントからの入力に対して動的に HTML ページを作成するアプリケーションが対象となるサーバへの攻撃方法で、攻撃対象者に任意のスクリプトを書き込みさせて送信することでそのスクリプトをブラウザ上で実行させる方法である。

  例えば、入力フォームを持つページとその入力だけをそのまま出力するページがあったとする。

  この入力フォームに

  ```
  <SCRIPT>window.open()<SCRIPT>
  ```

  のようなスクリプトを含む内容の書き込みを行うのである。すると書き込み内容を表示するページでは、このスクリプトを含む HTML をクライアントに送信する。するとスクリプトを含む HTML を受けた Web ブラウザは、そのスクリプトをサーバが送信したもので

図 2.2: 入力 Form ページ

あると判断し、実行してしまう事になる。このスクリプトの例であれば、クライアントの意思に関係なく新しいブラウザウィンドウを開いてしまう事になる訳である。

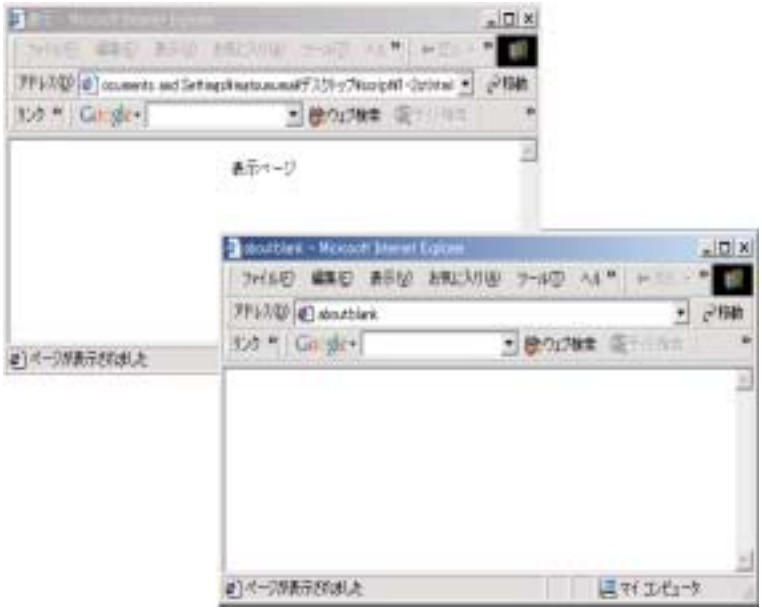

図 2.3: スクリプトが実行され二つ目のブラウザが自動で作成された様子

- **クロスサイトスクリプティングの具体的な方法**
  クロスサイトスクリプティングの具体的な方法をここで説明する。
  - 始めに攻撃するサーバの管理するページで, クライアントからの入力内容をチェックせずに出力するようなアプリケーションのあるページを探す。
    * 仮に入力フォームのあるページのアドレス
      `http://www.*******.co.jp/nyuuryoku.html` とし
      入力内容を表示するページのアドレス
      `http:www.*******.co.jp/hyouji.jsp` とする
  - 次に掲示板などの適当なページに URL のあとに送信するスクリプト命令を含んだリンクを置いておく。
    `http://www.*******.co.jp/nyuuryoku.html?nyuuryoku="`スクリプト命令`"`
  - 第三者が、このリンクをクリックする。リンクをクリックした第三者の Web ブラウザがスクリプトの実行を許可していた場合、サーバからスクリプト命令を含んだ HTML が送信され、画面上で実行される。

以上の動作によって第三者のブラウザ上でスクリプト命令を実行させるのである。

- **クロスサイトスクリプティングによる実際の cookie データ取得方法**
  第三者には漏洩する事は無いと考えられていた cookie データが、クロスサイトスクリプティングでどのように漏洩するのかをここで説明する。

  一般的な Web ブラウザには JavaScript 言語の実行系が搭載されているが、この実行系には cookie データを読み書きする機能が存在している。具体的な方法は HTML ページに
  `var COOKIE = document.cookie;`
  のように記述することで、変数 `COOKIE` に cookie データを読みだすことが可能になる。しかし、通常はこの方法で読み出し可能な cookie データは、この記述をした HTML に対して発行した cookie だけであるので、cookie データが第三者に漏洩することはないと考えられていた。

  しかし、ここで上で説明したクロスサイトスクリプティングを用いる事によって cookie データが第三者に漏れるという事が判明したのである。例えば先に挙げた Form ページで、入力するデータを
  `<SCRIPT>document.write(document.cookie)</SCRIPT>`
  にした場合、表示ページではそのページに対して発行した cookie がブラウザ上に表示される事になる。

図 2.4: cookie に格納していたセッション ID がブラウザ上に表示された様子

しかし、これだけではこの cookie データが第三者に漏洩するわけではない。そこで上で説明したように掲示板などにリンクを貼り、スクリプト命令を

```
<SCRIPT>window.open("http://第三者のサーバ/save.jsp
                    ?+escape(document.cookie)")</SCRIPT>
```

のようにした場合第三者のサーバに、リンクのページで発行された cookie が漏洩することになるわけである。

2.4.3　encodeURL 方式

セッション管理に、「cookie はセキュリティ的に不安がある」「元々使用している Web ブラウザが cookie をサポートしていない」などの理由で cookie 方式を使えない場合がある。そのような場合でも、Session クラスを扱えるようにする為に用いられている方式であり、cookie に sessionid を保存する代わりに、FORM の Get 方式に似た手順で Servlet へのリクエスト URL の後ろに sessionid を後付けする方法である。

例: `http://www.*******.co.jp/myservlet;jsessionid=**********`

Tomcat ではリクエスト URL に”;”文字が含まれているかをチェックし、含まれていた場合には;jsessionid=に続く文字を sessionid として取得して、その部分を切り取って URL を、新たにリクエスト URL とするように設

計されている。

ServletAPI2.2 以降では sessionid を直接 Servlet 内から参照することは推奨されていなく、HttpServletRequest クラスの endodeURL メソッドを用いて、URL の書き換えは Servlet コンテナに実行させる形で行う。

またServletAPI の仕様上、encodeURL メソッドを呼んでいても、使用している Web ブラウザが cookie を使える状態の場合は cookie に sessionid が格納されてしまいます。またこの方式では、セッション管理内に自己サーバ管理外へのリンクを置いた場合に、後述する Referer 機能 [5] によって情報が漏れるという可能性が有る。

- **encodeURL 方式の場合のプログラム例**
  以下に、encodeURL 方式を用いてログイン名とパスワードを Session クラスに収めて、次のページ (./SAMPLE3) に引き渡す Servlet のソースコードを示す。

```
public class SAMPLE2 extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
    throws ServletException, IOException {

    String log,pass,newURL;
    log = request.getParameter("logname");
    pass = request.getParameter("passwd");

    HttpSession session = request.getSession(true);
    session.setAttribute("LOGNAME" ,log);
    session.setAttribute("PASSWORD",pass);
    newURL = response.encodeURL("./SAMPLE3");
     //URL の書き換えを実行し、この新 URL をリンク等に用いる。
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println( … 省略 … );
  }
}
```

2.4.4　Referer 機能による encodeURL 方式の脆弱性

この章では Referer 機能による、encodeURL して URL に格納されたデータの漏洩の方法について説明する。

- **Referer 機能とは**
  Referer とは、多くの Web ブラウザに設定されている機能で、リンクをクリックした時にリンク先 URL のサーバに対して出力されるリクエストヘッダの一種で、いわゆる「リンク元」の URL 情報である。この情報を見ることでサーバ側はどのページからデータを引き出そうとしているのかが判断できるというわけである。具体的には Referer 情報を参照して特定のページ以外からのアクセスを制限するなどの方法に使用されている (直接的なリンクの防止等)。しかし、送信する Referer 情報の偽造は簡単に可能である為に、それほどの効果は無いとされている。

- **Referer 機能の使い方**
  Referer 情報はリクエストヘッダに格納されて送信されてくるので、Servlet で Referer 情報を参照するためには request クラスの getHeader メソッドを利用する。具体的な参照方法は、Servlet ファイル内で
  `String referer = request.getHeader("Referer")`
  と記述して参照する。

- **Referer 機能による実際の encodeURL データの漏洩**
  セッション ID を encodeURL 方式を用いてやり取りしていた場合に、データの漏洩が発生する。具体的にはセッション管理中のページに、他のサーバの管理するページへのリンクが置いて有る場合に漏洩する可能性がある。そのリンクをクリックした場合、そのページを管理しているサーバへとセッション中の URL 情報が送信されます。その URL 情報にはそこでのセッションに用いているセッション ID が含まれている為、漏洩するというわけである。

この他にも encodeURL 方式では、URL に情報が表示されてしまう事になっているので、ネットカフェなどの不特定多数の人間がいる場所などでは (URL を直接見る事で) 情報が漏洩する可能性もある。

2.5　Session クラスを用いず直接データを渡す場合

2.5.1　hiddenParameter を用いる場合

セッション管理内においての全てのページに Form タグを埋め込み、Form の hiddenParameter にデータを直接書き込んで、ページ間のやり取りを行う方法である。

hiddenParameter とは

Form の hidden フィールドとは、データを入力を取り扱う Form でブラウザ表面上には表示されることのないフィールドの事で、
<INPUT TYPE="HIDDEN" NAME="***" VALUE="###">
のように記述して用います。情報は Web ブラウザ上には表示される事はないが、HTML ソース上には表示されている。

- **hiddenParameter 使用時のコード例と表示例**

```
<html>
<center><head>送信ページ</head></center>
<title>Form</title>

<Body>
<center>
<form name="a" action="¥hyouji.html">
<Input type="text" name="txt" value="">
<Input type="hidden" name="hidden" value="hidden">
<input type="submit" name"ccc" value="送信">
</form>
</center>
</Body>
</HTML>
```

図 2.5: Tomcat 改によって作成されるページ

hiddenParameter の特性

hiddenParameter はセキュリティ上、情報の漏れる可能性が大変低く、情報は Form と同じ形で送信されるので、上述したクロスサイトスクリプティングや Referer 機能によって情報が漏洩する事はない。

しかし、セキュリティが高い反面、使い勝手が悪いという点も指摘されている。

- **リンクタグの使用不可**
  hiddenParameter を用いて情報を引き渡す場合、Form の形式で情報を渡す為、ページの移動にはリンクタグを使用する事が出来なくなる (SUBMIT ボタンのみになる)。その為、見栄えが悪くなるなどの問題が存在する。

- **既存のページからの書き換えの面倒さ**
  Session クラスを用いてセッション管理していた既存のページを、hiddenParametr を用いてセッション管理するページに書き換える場合、Session クラスに格納していた情報を全て hiddenParamter に入れ替え、ページ内にある全てのリンクをその hiddenParameter を含んだ Form タグに書き換える必要性がある。

以上の様に hiddenParameter を用いてセッション管理を行うページは、他のセッション管理との間の互換性が少ないという点がある。

2.5.2　cookie を用いる場合

この方法は, ページ間で受け渡す情報を全て cookie に書き込む方法である。

cookie を用いる場合の特性

この方式では、cookie のブラウザを閉じても情報を保存出来るという特性を生かす事が可能である。例えば、一度ログインしたページにおいて、cookie にログイン名と、パスワードを保存しておき次からの認証においては cookie を参照することで認証をさせないという事などが可能となっている。

しかし、この方式では先に挙げたクロスサイトスクリプティングの被害をもろに受けることになるのは明らかであるし、hiddenParameter 同様に他のセッション管理方式との間の互換性が少ないという点がある。

- **cookie を用いる場合のプログラム例**
  以下に、cookie に直接ログイン名とパスワードを格納して次のページに引き渡すページのソースコードを示す。

```
public class SetCookie extends HttpServlet {
  public void doGet(HttpServletRequest request,HttpServletResponse response)
      throws ServletException, IOException {

      String log,pass;
      log = request.getParameter("logname");
      pass = request.getParameter("passwd");

      Cookie cookie = new Cookie("logname",log);
      response.addCookie(cookie);
       //Cookie への情報のセット
      cookie = new Cookie("passwd",pass);
      response.addCookie(cookie);


      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      String title = "Setting Cookies";
      out.println( … 省略 … );
  }
}
```

第3章　システムの設計と実装

3.1　セキュリティ向上の為の手法提案

前章で挙げた様に、Servlet・JSP のセッション管理に関するセキュリティには、Session クラスの使用時・不使用時のそれぞれに場合についてセキュリティ上・使用上における問題点が存在する。

そこで本研究においては、Session クラスを用いる場合におけるセキュリティレベルの問題点を解決する為に、第三の sessionid 格納場所として Form の hiddenParameter フィールドを付け加える事を提案する。hiddenParameter フィールドの特性はセキュリティ面では高い性能を持っているが、使い方・互換性における問題点が存在するというものであった。そこで、使い方・互換性の面でも問題点を解決しつつ、Session クラスで用いる事が出来れば、セッション管理に関する 2 種類の問題を同時に解決できると考えたのである。

この他にも、cookie のクロスサイトスクリプティングへの対応性を上げる為に cookie からデータを引き出す際に、なんらかの認証を必要とするようなメソッドを作成するという事でセキュリティ性を向上させる事を提案する。

3.2　システムの設計

- sessionid の格納場所に hiddenParameter を加える為のシステム設計
  - Session クラスのセッション判別時に使用する sessionid の参照場所に Form の hiddenParameter フィールドを追加する。
  - Session クラスの sessionid の格納先に hiddenParameter フィールドを加え、セッション作成時にオプションを設定する事で選択可能にする。
  - Tomcat では、Session クラス作成時に、格納方式によらず cookie

に sessionid を格納していたが、この方式ではクロスサイトスクリプティングの脆弱性を改善できない。その為、この方式を中止し、三つの方式を完全に独立させ、選択可能にする。(encodeURL 方式を採用した場合には、cookie に sessionid を格納しないようにする)

- hiddenParameter の使い安さを向上させる為のシステム設計
  - Servlet・JSP ソースコードの書き換えをしなくても、sessionid の格納方式を変更可能にする。
    - HTML データの送信前に、HTML データ途中で拾い出し、特定のタグの存在をチェックする。
    - sessionid の格納先に hiddenParameter を選択していて、HTML データ内に特定のタグが存在する場合、タグ自体の書き換えを行う。
  - hiddenParameter 方式採用時にもリンクタグの使用を可能にする為のシステム設計
    - リンクタグの書き換え時に、Form 送信命令 JavaScript である submit() を作用させる事で、リンクタグの使用を可能にする。
- cookie 自体のセキュリティ向上の為のシステム設計
  - cookie に格納されたデータを取得する時に、特定の認証を必要なように cookie の仕様を変更する。

3.3　実装方法

- Session クラスの、sessionid 参照フィールドに request クラスの getParameter メソッドを用いて、Form の hiddenParameter を加える。
- オプションを設定する事で格納場所を cookie、encodeURL に変更することも可能に変更。ここで Tomcat では、Web ブラウザが cookie をサポートしていた場合には encodeURL を指定していた場合でも cookie に格納させていたのだが、これではセッション管理法の使い分けをサーバ側で確定させられない為、サーバ側が選択した方法にできるように、選択したものに格納させるように変更。

表 3.1: sessionid の格納先を変更する為の response クラスのメソッド

メソッド	内容
encodecookie	sessionid の格納先に cookie を指定する
encodeCookie	同上
encodeurl	sessionid の格納先に URL を指定する
encodeURL	同上
encodehidden	sessionid の格納先に hiddenParameter を指定する
encodeHidden	同上

- セキュリティを高める為に、sessionid 参照のデフォルトフィールドを既存の cookie から hiddenParamter に変更。
- Session クラスが呼ばれ、且つ hiddenParameter 方式を選択 (デフォルト) されていた場合には、クライアントへ最終的に HTML データを送信する Http10 クラスが、送信データの書き換えを行う。

  以下に簡単な、Session クラス使用時における Tomcat と Tomcat 改の動作比較図を載せる。

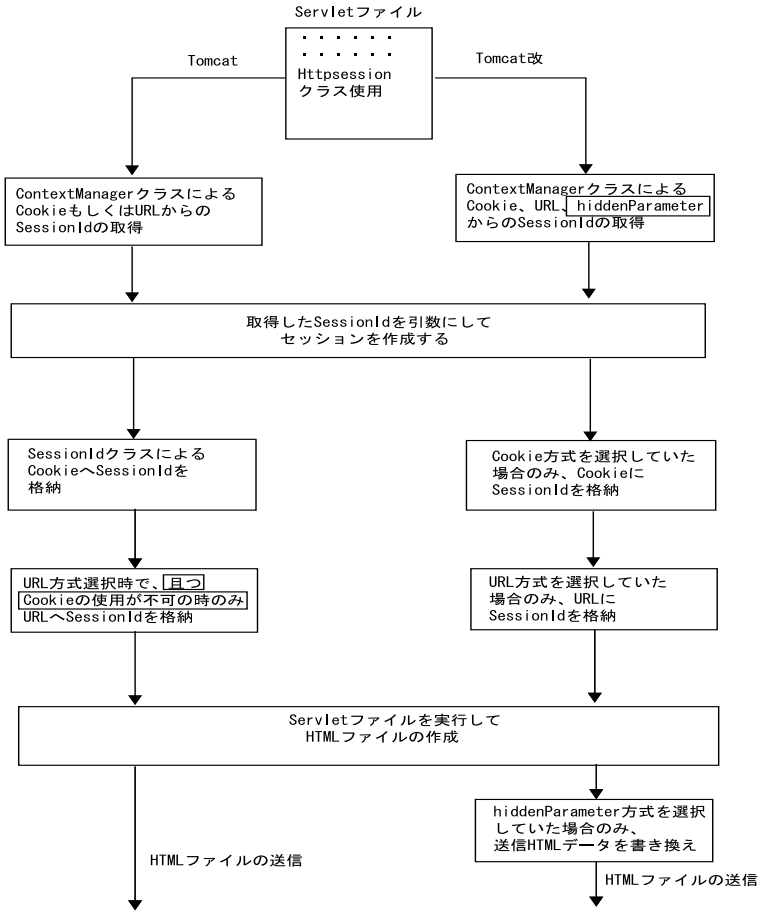

図 3.1: Tomcat と Tomcat 改、それぞれについての Session クラス使用時の動作説明図

書き換え手順

- HTML データに <Form> タグの存在をチェックして、存在した場合 hiddenParameter のタグをその <Form> 内に埋め込む。
- HTML データに <A> タグの存在をチェックし、存在した場合、hiddenParameter 送信用の JavaScript の埋め込みと <A> タグの書き換えを行う。

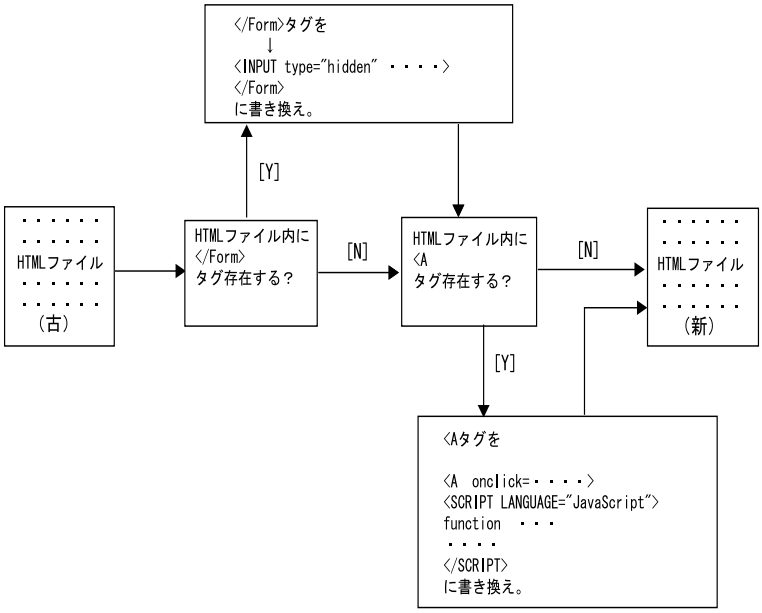

図 3.2: HTML データ書き換えの手順

- 第三者が cookie の内容を取得し、自己の cookie にセットして送信してきてもセッションジャックされない為に cookie の値参照に認証を必要とするようなセキュリティの高い値取得メソッドを用意。

次に実装した Tomcat 改のクライアントからのリクエストを受けてから HTML を送信するまで流れを図を用いて示す。

3.3.1　Tomcat 改の動作説明図

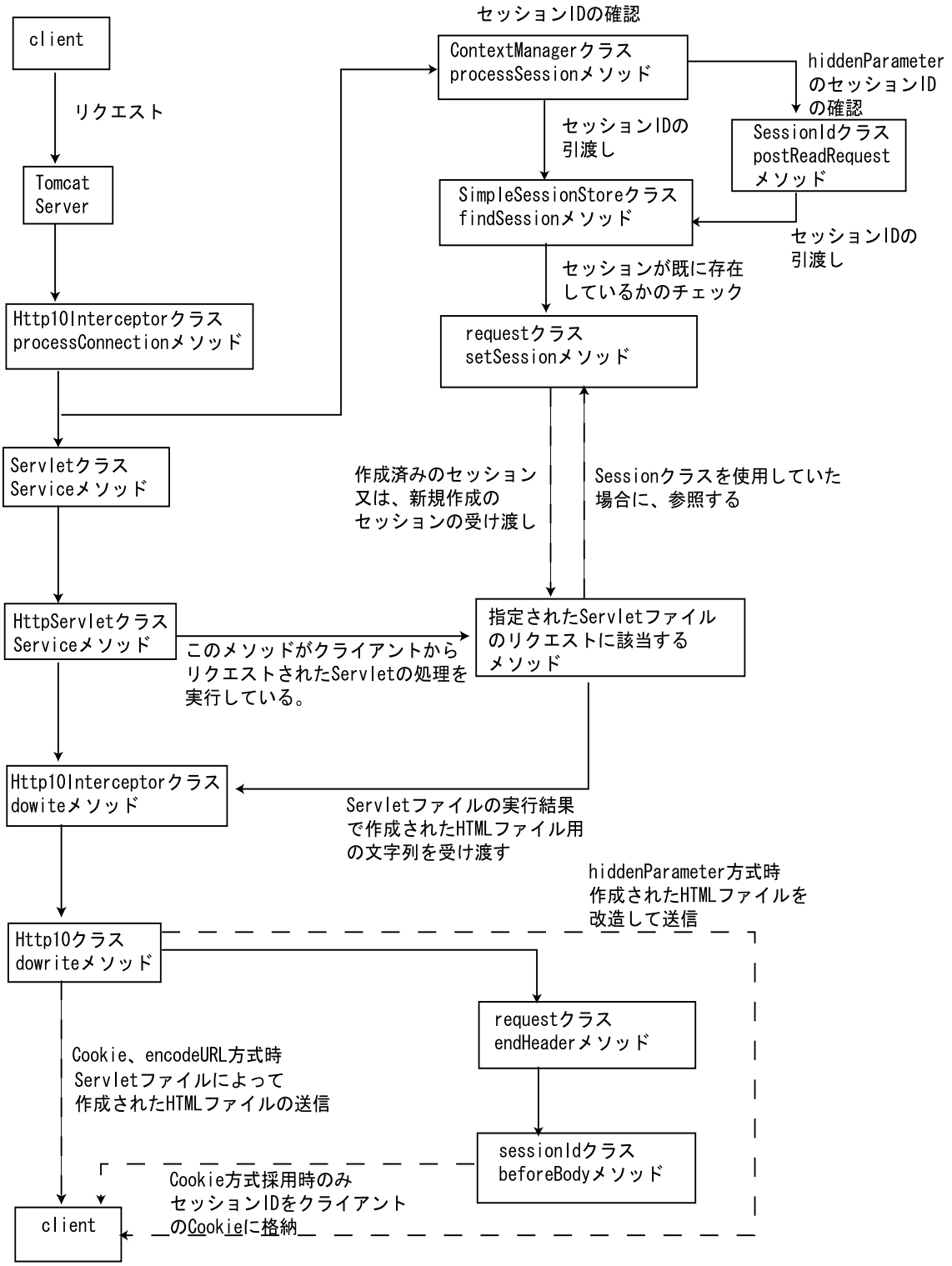

図 3.3: Tomcat の動作説明図

3.4 Tomcat 改での送信 HTML ファイルの書き換え例

この章で、Session クラスを使用する Servlet のファイルと、そのファイルが Tomcat 上で実行され作成される HTML ファイル、Tomcat 改上で実行され作成される HTML ファイルを、それぞれ並べて表示する事で、どうのように書き換えが行われているのかを示す。

- **Session クラスを使用する Servlet ソースコード**

```
public class hidden extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    HttpSession session = request.getSession(true);
    session.setAttribute("passwd","12345");

    String docType ="<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0
                                          Transitional//EN\">\n";
    out.println(docType +
                "<HTML>\n" +
                "<HEAD><TITLE>Hidden ! </TITLE></HEAD>\n" +
                "<HEAD>\n" +
                "<A href=\"hidden2\"> next page </A>"+
                "</BODY></HTML>");
  }
}
```

- **Tomcat で作成された HTML ファイル**

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD><TITLE>Hidden ! </TITLE></HEAD>
<HEAD>
<A href="hidden2"> next page </A>
</BODY>
</HTML>
```

- **Tomcat 改で作成された HTML ファイル**

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD><TITLE>Hidden ! </TITLE></HEAD>
<HEAD>
<A onclick="formwindow(href);return false" href="hidden2">
                                next page </A></BODY></HTML>

<SCRIPT LANGUAGE="JavaScript">
function formwindow(href) {
document.write("<FORM NAME=\"NINNSHOU\" METHOD=\"POST\"
                                        ACTION=\""+href+"\">");
document.write("<INPUT TYPE=\"HIDDEN\" NAME=\"jsessionid\"
                                       VALUE=\"e117c7upm1\"></FORM>");
document.write("<"+"SCRIPT>document.NINNSHOU.submit()</SCRIPT"+">");}
</SCRIPT>
```

- **Tomcat 改による HTML ページ作成の様子**
  図より Form データの送信を、JavaScript に実行させる為、ページの外観を変えるという Form 特有の弱点を克服した事が確認出来る。

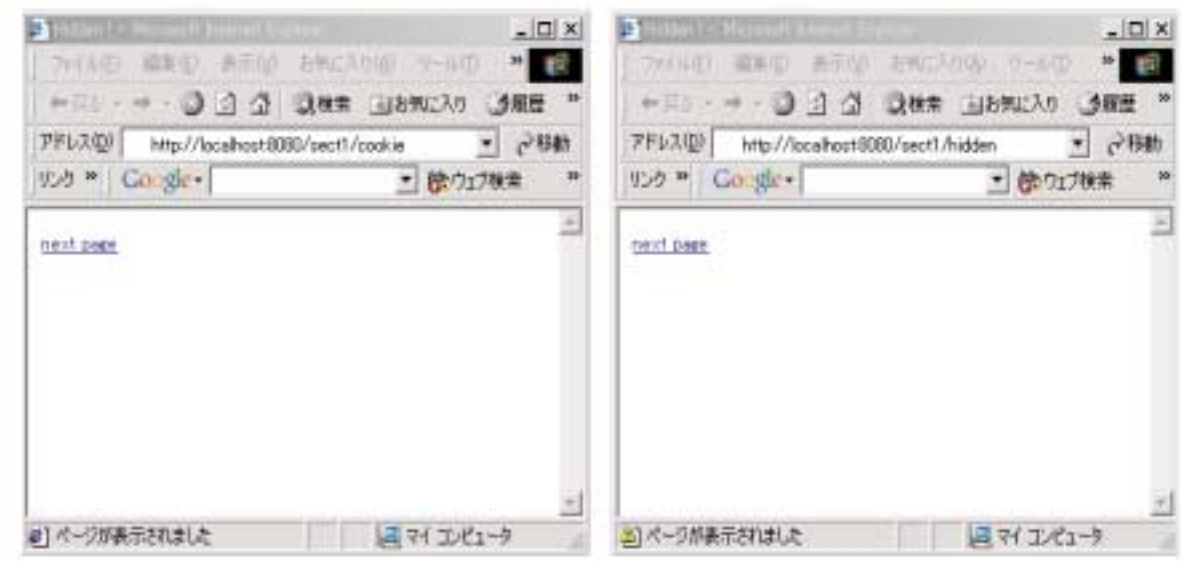

図 3.4: 左図：cookie 方式を使用し、Tomcat で作成した HTML ページ
　　右図：hiddenParameter 方式を使用し、Tomcat 改の作成した HTML ページ

3.5　cookie の安全性を高めた値取得メソッド

Tomcat では他人の cookie データをクロスサイトスクリプティングによって取得した者が、自分の Web ブラウザの cookie にセットし、サーバにアクセスして来た場合、その cookie によってセッションジャックされる事が多々ある。そこで Tomcat 改ではそのようなセッションジャック法を防ぐ為の手法として、サーバ側がクライアントの cookie データを取得する際に、リクエストの Authorization ヘッダを用いる認証を必要とするような値取得メソッド・secureGetValue() を追加作成した。サーバ側がこのメソッドを用いて cookie データの取得を行っていれば、仮に第三者に cookie データが漏洩したとしても、アクセスしてきた場合に認証を必要とするのでセッションジャックされる事はなくなるのである。

表 3.2: Tomcat 改における、cookie データ取得用のメソッド

メソッド	説明
getValue()	cookie の値を取得する為の既存のメソッド
secureGetValue()	値取得に認証を必要とするような新メソッド

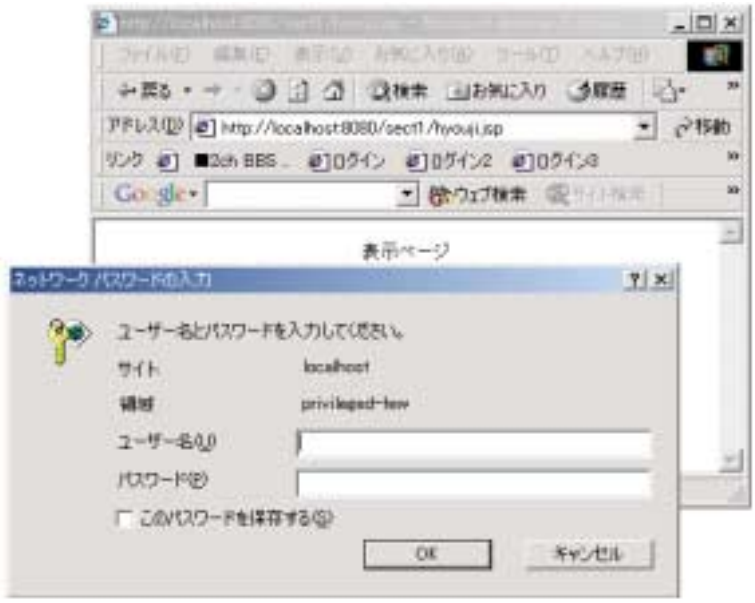

図 3.5: secureGetValue メソッドで値を取得する時の様子

使い方は getValue メソッドと全く同じ方法で使用可能である。新システム Authorization ヘッダを用いているので、認証を行うのも始めの一回のみですむようになっている。cookie からデータを取得する時に、この新メソッドを使用している場合、仮に cookie データが他人に漏洩したとしても、サーバ側でデータを取得する時に認証を行う為にセッションジャックを防ぐ事が可能となる。

第4章　実験

Tomcat 改の性能評価の為に、送信するデータ種類別のスループットを測定した。この測定には、Web サーバ用ベンチマークである Mindcraft 社の WebStone の Version2.5[4] を使用した。なお、測定に使用したマシンは
CPU:Pentium　733MHz, メモリ:128MB,OS:linux 2.2.14-5.0, ネットワーク:100BaseTX
のマシン 13 台。この内一台をサーバマシンとし、残りの 12 台をクライアントマシンとして実験を行った。

4.1　HTML ファイル送信時のスループット

書き換えを全く行わない HTML ファイルを送信する時のサーバのスループットを計測する為に HTML ファイルのサイズを変えて実験を行った。

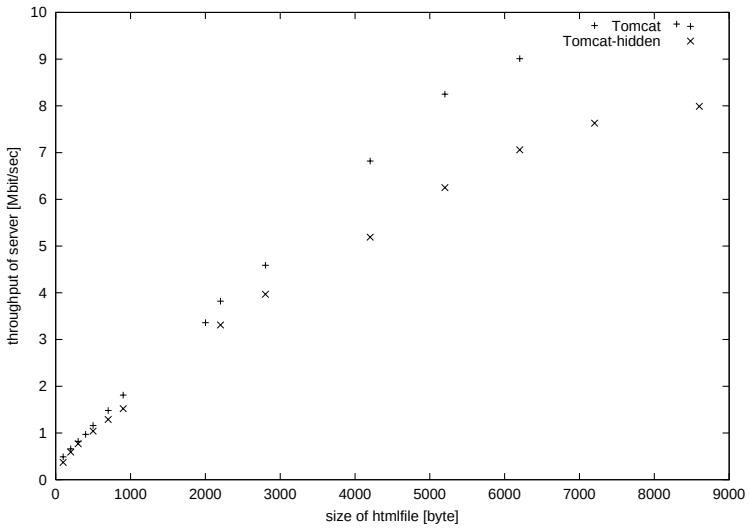

図 4.1: HTML ファイル送信時のスループット

HTML ファイルの送信において、スループットの差の原因として考えられるのは、Tomcat 改ではデータをクライアントへ送信する前に hiddeParameter の使用をしているかどうかのチェックを行い、送信データを一度 String にし、再度バイトに戻し、各種変数の初期化を実行している。この為 Tomcat に比べて性能が 2 割程度減少していると考えられる。

4.2　hiddenParameterへの変換を行わないJSPファイル送信時のスループット

JSP ファイルの内、hiddenParameter への書き換えを行わない場合におけるサーバのスループットを計測する為に JSP ファイルのサイズを変えて実験を行った。

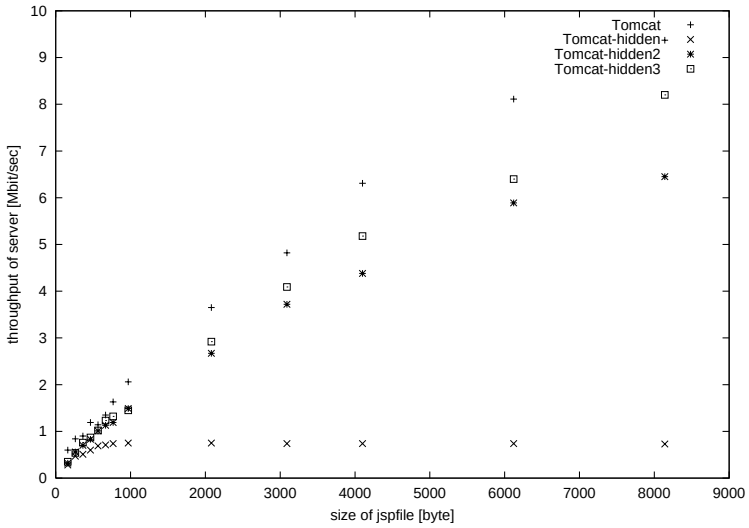

図 4.2: hiddenParameter への変換を行わない JSP ファイル送信時のスループット
(Tomcat-hidden2 は Tomcat-hidden からタグチェックを外したものであり、Tomcat-hidden3 は Tomcat-hidden から送信データの変更を全て外したものである)

このグラフから Tomcat 改の動作において、最も時間の掛かっている処理は、送信するデータに特定のタグ (<Form> タグや <A> タグ) をチェックする為正規表現を用いて検索している場所である事がわかる。

また、変換を行わない JSP がこのようにスループットが低下している原因は JSP が Servlet に変換される際に、自動的に Session クラスを作成してしまうという事が挙げられる。この為 Tomcat 改では最も時間の掛かるタグの探索を行ってしまう為、この用にスループットが低下しているものと考えられる。

4.3　hiddenParameterへの変換を行うJSPファイル送信時のスループット

JSP ファイルの内、hiddenParameter への書き換えを行う場合におけるサーバのスループットを計測する為に、書き換えを行う数を変えて実験を行った。

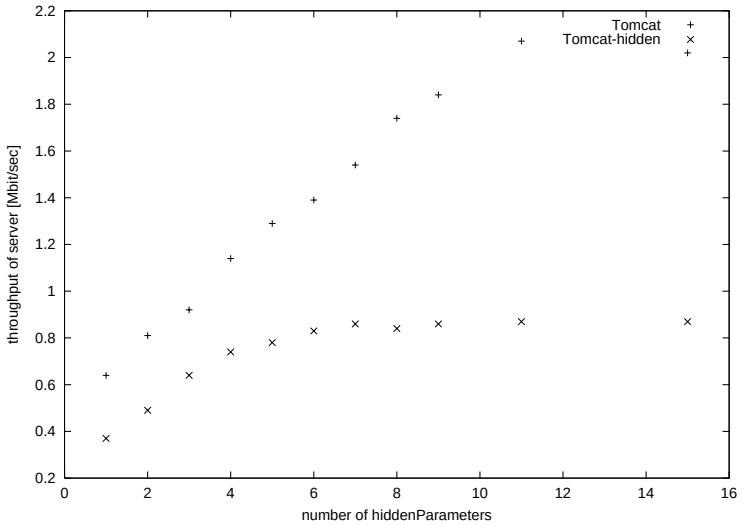

図 4.3: hiddenParameter への変換を行う JSP ファイル送信時のスループット

このグラフから hiddenParameter に変換するタグの個数は、スループットにそれ程影響しないことが分かる。

4.4　hiddenParameterへの変換を行わないServletファイル送信時のスループット

Servlet ファイルの内、hiddenParameter への書き換えを行わない場合におけるサーバのスループットを計測する為に Servlet ファイルのサイズを変えて実験を行った。

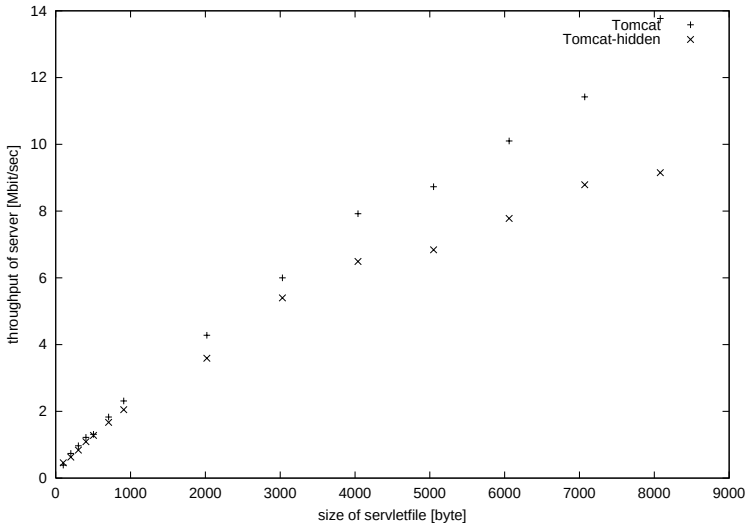

図 4.4: hiddenParameter への変換を行わない Servlet ファイル送信時のスループット

このグラフから、hiddenParameter への変換を行わない場合にはタグの検索を行わない為、スループットはそれ程低下していないという事が分かる。

4.5　hiddenParameter への変換を行う Servlet ファイル送信時のスループット

Servlet ファイルの内、hiddenParameter への書き換えを行う場合におけるサーバのスループットを計測する為に、書き換えを行う数を変えて実験を行った。

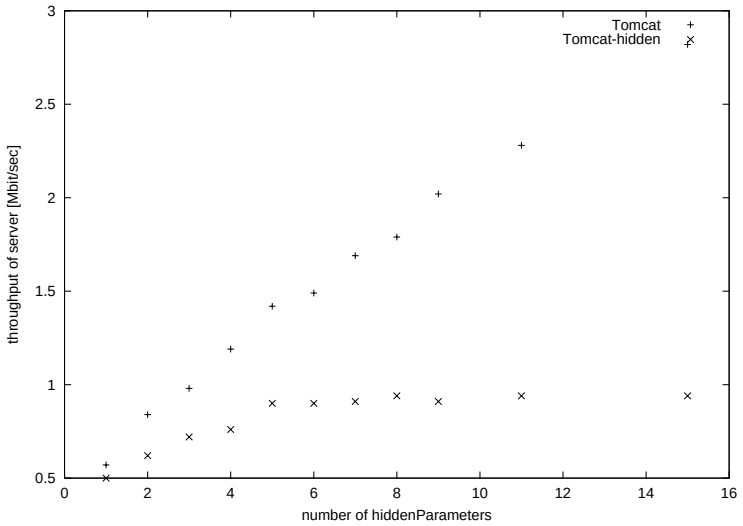

図 4.5: hiddenParameter への変換を行う Servlet ファイル送信時のスループット

このグラフから、「hiddenParameter への変換を行う JSP ファイル送信時のスループット」の実験同様に、変換する数はそれほど影響しないという事が分かる。

第5章　まとめ

本研究で提案したTomcat改ではServlet・JSPのセッション管理に関してのセキュリティを向上させる為にSessionクラス使用時に用いるsessionidの格納場所にFormのhiddenParameterフィールドを加えた。更にcookie自体のセキュリティを向上させる為にcookieデータ取得に認証を必要とする新たなメソッドを加えた。

hiddenParameterフィールドの追加によって、2章で挙げたServlet・JSPのセッション管理における問題点であったクロスサイトスクリプティング・Referer機能による脆弱性を克服する事が出来た。またcookie自体のセキュリティを向上させた事で、仮にcookieデータが漏洩したとしてもセッションジャックされないようなWebページを作成する事が容易となった。

本研究に置いて重要視した点の一つは、hiddenParameterをID格納場所に加える時、如何に元コードを書き換えずに済むかという点である。Tomcat改では、Tomcatにおいて使用していたSessionクラス使わないServletソースコードは、書き換えずに使用する事が出来る。またSessionクラスを使用する場合においても、書き換える必要があるのは、sessionidの格納場所をcookieにする場合にのみとなっている。これはセキュリティ性を向上させるのに、デフォルトのsessionid格納場所がcookieからhiddenPrameterへと変更した為に生じた書き換えである。

もう一つは、hiddenParameter方式使用時におけるページ外観の変更を如何に少なく済ませるかという点であった。Tomcat改ではhiddenParameter方式を使用していても、JavaScriptを用いてページの外観自体を変えない方式を採用した。

5.1　今後の課題

Tomcat改では、HTMLデータをクライアントに送信する前に、特定のタグが存在するかをチェックしている為、セキュリティは向上したが、

サーバ自体のパフォーマンスが低下するという事になった。その為今後の課題としては、このタグチェック部分に掛かる時間を減少させパフォーマンスを上げる事を目標としている。

参考文献

[1] apache: Cross Site Scripting Info, http://httpd.apache.org/info/css-security/index.html.

[2] Jason Rafail, C. C. C.: Cross-Site Scripting Vulnerabilities, http://www.cert.org/archive/pdf/cross_site_scripting.pdf.

[3] Microsoft: クロスサイトスクリプティングの脆弱性の問題, http://support.microsoft.com/default.aspx?scid=http://www.microsoft.com%/japan/support/kb/articles/252/9/85.asp.

[4] Mindcraft: WebStone2.x Benchmark Description, http://www.mindcraft.com/webstone/ws201-descr.html.

[5] 高木浩光: Cookie を使用せず URL に埋め込む ID に頼ったセッション管理方式の脆弱性 (1), http://securit.etl.go.jp/SecurIT/advisory/webmail-1/.

[6] 高木浩光, 関口智嗣, 大蒔和仁: クロスサイトスクリプティング攻撃に対する電子商取引サイトの脆弱さの実体とその対策, 情報処理学会コンピュータセキュリティ研究会 and 第 4 回コンピュータセキュリティシンポジウム (2001).

付 録A　実験に用いた、プログラム

5 章の実験に用いた、HTML ファイル、変更しない JSP ファイル、変更する JSP ファイル、変更しない Servlet ファイル、変更する Servlet ファイルの 5 種類を載せる。各ファイルは単調な繰り返しを含んでいる為、省略する。

- **HTML ファイル**

```
<html>
<title>aaa</title>
<body>
aaaaa …
 … 省略 …
 … aaaaa
</body>
</html>
```

- **変更しない JSP ファイル**

```
<%@ page import="java.lang.*" contentType="text/html; charset=Shift_JIS" %>
<HTML>
<TITLE>aaaaa</TITLE>
<BODY>
aaaaa …
 … 省略 …
 … aaaaa
</BODY>
</HTML>
```

- **変更する JSP ファイル**

```
<%@ page import="java.lang.*" contentType="text/html; charset=Shift_JIS" %>
<%
String strPass="yes";
String bolMember="no";

HttpSession session2 = request.getSession(true);
session2.setAttribute("MEMBERFLG",bolMember);
%>
<HTML>
<TITLE>aaa</TITLE>
 <BODY>

 <FORM METHOD="POST" ACTION="news2.jsp">
 <INPUT TYPE="SUBMIT" VALUE="GO">
 </FORM>
 <FORM METHOD="POST" ACTION="news2.jsp">
 <INPUT TYPE="SUBMIT" VALUE="GO">
 </FORM>

  … 省略 …

 <FORM METHOD="POST" ACTION="news2.jsp">
 <INPUT TYPE="SUBMIT" VALUE="GO">
 </FORM>

 </BODY>
</HTML>
```

- **変更しない Servlet ファイル**

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class h2 extends HttpServlet {
  public void doGet(HttpServletRequest request,HttpServletResponse response)
    throws ServletException, IOException {

    PrintWriter out = response.getWriter();
    out.println("aaaaa … 省略 … aaaaaa");
  }
}
```

- **変更する Servlet ファイル**

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class a2 extends HttpServlet {
  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    HttpSession session1 = request.getSession(true);
      session1.setAttribute("aaa" ,"AAA");
      String aaa = (String)session1.getAttribute("aaa");

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0
                 Transitional//EN\\">\n"+
                "<HTML>" +
                "<BODY>\n" +
                "<H1>Hello WWW3</H1>\n" +
                "<FORM METHOD=\"POST\" ACTION=\"pack.HelloWWW3\">
                 <INPUT TYPE=\"SUBMIT\" VALUE=\"GO\"> </FORM>"+
                "<FORM METHOD=\"POST\" ACTION=\"pack.HelloWWW3\">
                 <INPUT TYPE=\"SUBMIT\" VALUE=\"GO\"> </FORM>"+

               … 省略 …

              "<FORM METHOD=\"POST\" ACTION=\"pack.HelloWWW3\">
                 <INPUT TYPE=\"SUBMIT\" VALUE=\"GO\"> </FORM>"+
                "</BODY></HTML>");
  }

  public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
      doGet(request,response);
  }
}
```