書籍学習【基礎からのサーブレット/JSP】

ノートブック:	210_メモ		
作成:	2015/12/19 20:45	**更新:**	2019/07/19 16:06
作成者:	Tsutsui tomoaki		
URL:	about:blank		

タイトル： 基礎からのサーブレット/JSP

	chapter 01【サーブレット/JSPって何？】

01.サーブレットとは

Java Servletは、「**サーバ上で動作するJavaプログラム**」の事
＊特徴
・**動的な処理**：プログラム処理を行い動的な結果を返す
・**Webを対象**：ブラウザなどのWebクライアントに対するサービスを提供する
・**アプリケーションサーバ**：アプリケーションサーバ上で動作する

・**動的な処理**：プログラム処理を行い動的な結果を返す

○ Coulmn:サーブレットとCGI

CGI：サーブレットと同じようにブラウザからのリクエストに応じてサーバ側でプログラムを動作させる仕組み
サーブレットも広い意味ではCGIの一種ともいえるが、通常CGIといった場合は、PerlやPHPなどのスクリプト言語で記述。

～サーブレットの利点～
○パフォーマンス
CGI：リクエストの度に新たなプロセスを起動して結果を返す。
サーブレット：先にJavaプロセスを起動し、プロセスの中のスレッド単位でリクエストを処理する。
--速度とメモリ消費の点でCGIより優れている
○ポータビリティ(移植性)
CGIは特定サーバへの依存度が高い

・**Webを対象**：ブラウザなどのWebクライアントに対するサービスを提供する
サーブレットはHTTPプロトコル上で動作するプログラムといえる。
SMTPやFTPで動作するサーブレットも作成できなくはないが、今回開発するのはHTTPサーブレットの機能

・**アプリケーションサーバ**：アプリケーションサーバ上で動作する
サーブレットが動作するには、アプリケーションサーバが必要！！
サーブレットコンテナ：サーブレットを呼び出して実行する環境
ブラウザからの***HTTP*リクエスト**はまず、**Web(HTTP)サーバ**で受け付けられた後、**サーブレットコンテナ**に渡される。
サーブレットは**サーブレットコンテナ**から呼び出される。

02.JSPとは

HTMLページを出力するためのサーブレットの拡張技術
JSPは裏で自動コンパイルされる仕組み

03.JavaEEとは

Javaのエディションの説明

04.サーブレット/JSPの位置づけ

MVCモデルにのっとって設計していくべし。
◆プレゼンテーション
V:JSP
C:サーブレット
◆ビジネスロジック
M:Javaクラス
◆データアクセス

05.XXXXX

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 02【開発環境の準備】

01.本書で利用するツール類

・JDK
・Tomcat
・HSQLDB
・JSTL
・Eclipse

02.インストール

～説明～

03.環境変数の設定

JAVA_HOME
Path

04.Tomcatを起動してみる

◆Tomcatとは
Apache Software Foundationで開発されているオープンソースのアプリケーションサーバ
内部にHTTPサーバを持っています。
apache-tomcat
|-bin ----------起動、停止などのスクリプト
|-conf ----------設定ファイル系
|-lib ----------クラス、Jarファイルを配置
|-logs ----------ログファイルが出力される
|-temp ----------JVMの一時ファイルが出力される
|-webapps ----------Webアプリケーションを配置
└work ----------Webアプリケーションの一時ファイルが出力

05.XXXXX

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 03【Webアプリケーションの動作原理を知る】

01.Webアプリケーションのディレクトリ構造

～説明～

02.Webアプリケーションの作成

Webappsの下にディレクトリをきってアプリケーションの作成

03.404 NOT FOUND

※ディレクトリを指定した場合、404 NOT FOUND を表示するのではなく、ディレクトリ構造を表示させる方法
/conf/web.xml

```
<init-param>
    <param-name>listing</param-name>
    <param-value>true</param-value>
</init-param>
```

04.ウェルカムファイル

tomcatでは以下のようなファイルがウェルカムファイルとして登録されている

・index.html
・index.htm
・index.jsp
デフォルトファイル、インデックスファイルなどと呼ばれることもある

05.相対パスと絶対パス

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 04【はじめてのサーブレット】

01.まずは動かしてみよう

◆サーブレットJava
・javax.servlet.http.HttpServletを継承している
・doGet()メソッドをオーバーライドしている

◆Web.xmlの記述

```
<web-app>

    <servlet>
        <servlet-name>HelloServlet</servlet-name>
        <servlet-class>HelloServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>HelloServlet</servlet-name>
        <url-pattern>/HelloServlet</url-pattern>
    </servlet-mapping>
</web-app>
```

02.サーブレットクラスの内容

◆HttpServletクラスを継承している
◆doGet()メソッドをオーバーライドしている
ブラウザからサーブレットを呼び出した時にこのメソッドが呼ばれる
○戻り値
void です
○引数
Javax.servlet.http.HttpServletRequest,
Javax.servlet.http.HttpServletResponse
○例外
java.io.Exception
javax.servlet.ServletException
レスポンスを書き込む際に発生するIOExceptionをそのままスローし、その他の例外はServletExceptionにラップしてスローすることが出来ます。

03.web.xmlの役割

web.xmlは配備記述子(Deproyment Descriptor)
◆サーブレットの登録
◆サーブレットのマッピング

04.コンパイル用バッチファイルの作成

[compile.bat]
@set CLASSPATH=C:¥servletbook¥apache-tomcat¥lib¥servlet-api.jar
javac -sourcepath src -d classes src/%1

-sourcepath : パッケージの異なる2つのクラスをコンパイルする際に、依存するクラスも同時にコンパイルしてくれる

05.tomcatのオートリロード

META-INF直下の
context.xmlの中に下記を記載する
<Context reloadable="true"/>

06.パッケージ付きのサーブレット

対象のJavaファイルでパッケージ宣言を行い、
コンパイル時に-dオプションで出力フォルダを指定していれば、パッケージフォルダが勝手に作成される。

```
<web-app>

    <servlet>
        <servlet-name>BarServlet</servlet-name>
        <servlet-class>foo.BarServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>BarServlet</servlet-name>
        <url-pattern>/BarServlet</url-pattern>
    </servlet-mapping>
</web-app>
```

07.他のクラスを使うサーブレット

08.纏め

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 05【日本語の表示とコンテンツタイプ】

01.HTMLを出力するサーブレット

```
response,setContentType("text/html; charset=Windows-31J");
PrintWriter out = response.getWriter();
```

out.ptintln("<HTML>");
…

・setContentType()メソッドでコンテンツタイプを指定している
・HTMLタグも出力している
◆setContentType
・MINEタイプ
response.setContentType("text/html")
response.setContentType("text/plain")
MINEタイプとは、インターネット上でやり取りされるデータの種類のこと

・文字エンコーディング
出力する文字のエンコーディングの種類のこと
サーブレットからブラウザに文字列を送るときには、エンコード、デコードという処理が必要になります。
-エンコード(符号化)　：文字列を2進数のデータに変換する
-デコード(復号化)　　：2進数のデータを文字列に変換する

02.プレーンテキストを表示するサーブレット

03.XXXXX

04.XXXXX

05.XXXXX

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 06【web.xmlの基本】

01.XML入門

◆XML宣言
先頭一行目はXML宣言と呼ばれる物で、
<? ～ ?>という記号を使用して記述します。
XML内に日本語を記述する際は encoding属性を指定します。(省略する場合は必ずUTF-8で保存します。)
XML宣言は必須ではないが、多くの場合は記述する。
version：必須　xmlのバージョン。現状1.0と1.1が指定できる。1.1の場合はXML宣言は必須
encoding：任意　ファイルのエン
コーディングを指定する。省略した場合はデフォルトでUTF-8
standalone：任意　スタンドアロン文書かどうかをyes,noで示す。スタンドアロン文書とは外部ファイルの参照が必要かどうかを示す。
◆ルート要素
ルート要素は単一である必要がある！

◆スキーマ
XMLの構造を定義したものはスキーマと呼ばれ、スキーマを使って文法チェック(妥当性チェック)を行うことができます。
例)書籍情報を交換するためのデータ
・ルート要素はbooks
・booksの下に適当な数のbook要素を書く

・book要素にはisbn属性を必ずつける
・book要素の下にtitle要素を書く
--books.dtd
<!ELEMENT books (book*)>
<!ELEMENT book (title)>
<!ATTLIST book
    isbn CDATA #REQUIRED
>

<XMLスキーマの種類>
・DTD(Dcument Type Definition) 昔からあるスキーマ言語で、記述が凄く簡単なので広く利用されています。複雑なデータ型やルール定義が出来ない
・XMLスキーマ
・RELAX_NG

◆整形式(Well-Formed)と妥当(Valid)
整形式のXML　：ルート要素が一つ、開始タグと閉じタグが存在するというXMLの基本的な文法を満たしたXML
妥当なXML　　：アプリケーションスキーマ定義が守られている文書

02.web.xmlの構造

先頭、
＞XML宣言
＞スキーマ宣言

03.web.xmlにスキーマ定義をつける

04.XXXXX

05.XXXXX

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 07【サーブレットを簡単に作る】

01.invokerサーブレットの設定

/conf/web.xml
org.apache.catalina.servlets.InvokerServlet

/conf/context.xml
privileged="true"
を有効にする

invokerサーブレットを使用すると、
<サーバのURL>/<アプリケーション名>/servlet/完全クラス名
というURLでサーブレットを呼び出すことができる

◆invokerサーブレットの注意点

・基本的にはサーブレットはweb.xmlに登録しないと実行出来ない
・invokerサーブレットはTomcat独自の機能である
・invokerサーブレットはセキュリティ上の問題がある

02.XXXXX

03.XXXXX

04.XXXXX

05.XXXXX

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 08【はじめてのJSP】

01.まずは作ってみよう

1+1 = <%= 1 + 1 %>
この<%= ～ %>というタグで囲まれた部分は「JSPの式(Expression)」と呼ばれる物でタグの中に記述したJava式の結果が出力されます。
なお、Java式の中には区切り文字(;)はつかないので注意する
1+1 = <%= 1 + 1 ; %>

◆JSPはサーブレット
JSPは実はサーブレットで、実行されるまでは変換、コンパイルという処理が行われ、サーブレットとして実行されます。
JSPのソース・ファイル
↓
サーブレットのソース・ファイル
↓
サーブレットのクラスファイル
↓
サーブレットのインスタンス
これらの処理は最初にリクエストが会った時に行われるので、一回目にJSPにアクセスするときには、レスポンスが帰ってくるまで若干待たされます。

○ Coulmn:中間ファイルの場所

デフォルトで中間ファイルであるJAVAファイルは
workディレクトリ以下に作成されます。
JSPの挙動がおかしい時はこのディレクトリのファイルを削除してみると良い。

02.スクリプトレットを使ってみよう

```
<%
int count = 0;
for(int i = 0; i < 10; i++){
    count += i;
}
%>
```

<%～%>というタグが使われています
この部分はスクリプトレットと呼ばれ、内部にJavaコード(複数の分)を記述出来ます。

そして、JSPの式で
count = <%=count%>
画面に表示することが出来ます。

03.出力用の変数out

```
<%
int count = 0;
for(int i = 0; i < 10; i++){
    count += i;
}
out.println("count=" + count);
%>
```

outはJSPの暗黙のオブジェクト(Implicit Object)と呼ばれる物
◆補足
・out変数はJspWriterというクラスのインスタンスです。

04.JSPのコメント

```
<%-- JSP comment --%>
```

05.コンテンツタイプの指定

◆JSPにおけるコンテンツタイプの指定はpageディレクティブを使用する。
<%@page contentType="text/html; charset=Windows-31J" %>

#Servletの時はdoGet()メソッド中に
response.setContentType("text/html; windows-31J");
で記述した！
<%@ ～ %>で記述されるタグはディレクティブと呼ばれJSPの各種設定を行います。
<%@<ディレクティブ名> <属性> = <値> %>
contentType="<MINEタイプ> ; charset=<文字エンコーディング>"

06.import宣言

<%@page import="java.util.Date"%>
◆複数記述する場合
<%@page import="java.util.Date , java.io.PrintWriter"%>
◆補足
javax.servlet, javax.servlet.http, javax.servlet.jspパッケージはデフォルトでインポートされているので記述する必要はない

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 09【コンテキストパスを理解する】

01.XXXXX

02.XXXXX

03.XXXXX

04.XXXXX

05.XXXXX

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 10【入力パラメータの取得】

01.フォームデータを受け取り

02.日本語を入力する

・URLエンコード

03.GETとPOST

リクエストメソッド	サーブレットのメソッド	説明
GET	doGet()	コンテンツを取得する。情報を送信する
POST	doPost()	情報を送信する
HEAD	doHead()	ヘッダー部分を取得する
PUT	doPut()	コンテンツを作成、更新する
DELETE	doDelete()	コンテンツを削除する
OPTIONS	doOptions()	使用可能なオプションの一覧を返す
TRACE	doTrace()	ループバックを起動する

サーブレットの開発者としてはdoGet()もしくはdoPost()の両方をオーバーライドして使用する
・doGetはURLに受け渡しパラメータが埋め込まれる
・PostはURLではなくボディに情報が埋め込まれる

04.XXXXX

05.XXXXX

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 11【XXXXXXXXXXXXX】

01.XXXXX

02.XXXXX

03.XXXXX

04.XXXXX

05.XXXXX

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 12【フィルタの利用】

01.XXXXX

02.XXXXX

03.XXXXX

04.XXXXX

05.XXXXX

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 13【画面遷移を行う】

01.リクエストのディスパッチ

・フォワード　：転送先のリソースに出力処理を任せる
・インクルード：転送先のリソースに出力結果をインクルードする

```
//RequestDispatcherオブジェクトの取得
RequestDispatcher dispacher = request.getRequestDispatcher("/servlet/dispatch.ForwardServletB");
//Forwardの実行
dispatcher.forward(request, response);
```

02.JSPにフォワードする

```
//JSPページにフォワードする！
RequestDispatcher dispatcher = request.getRequestDispatcher("/dispatch/forward_ok.jsp");
dispatcher.forward(request, response);
```

JSPにフォワードする処理はWebアプリケーションではよく行われる処理なので覚えておく！！！
確かにこれで指定したJSPに遷移することが可能。。＊

03.フォーワード先を動的に変える

04.フォワードとインクルード

フォワード前のサーブレットでresponse.getWriter();
を使用してレスポンスに対する処理をしてもフォワード時に「レスポンスバッファがクリアされる」ので書き込みは反映されません。

05.リダイレクトについて

リダイレクトのメリット
・Webアプリケーション外のページに遷移させるための手法
同じアプリケーションないであれば、処理時間がかかることや、スコープの範囲外になってしまう関係からあまり使用されない。

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 14【オブジェクトのスコープとリクエスト属性】

01.オブジェクトのスコープ

■リクエストスコープ
データ有効範囲は1つのリクエスト内
リクエストをまたいでデータは保持されません

■セッションスコープ
データ有効範囲は同じクライアント

■アプリケーションスコープ
Webアプリケーション内であれば、クライアントが異なっても、サーブレットが異なってもデータを共有出来ます。

スコープ	クラス
リクエスト	javax.servlet.http.HttpServletRequestインターフェース
セッション	javax.servlet.http.HttpSessionインターフェース
アプリケーション	javax.servlet.ServletContextインターフェース

02.サーブレット間でデータを渡す

03.XXXXX

04.XXXXX

05.XXXXX

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 15【セッション属性を使う】

01.セッションの基本

02.セッションを一覧する

03.セッションタイムアウト

Web.xmlで設定可能
以下の例では5分に設定している

```
<session-config>
    <session-timeout>5</session-timeout>
</session-config>
```

なお、Tomcat6ではデフォルトで30分に設定されている

Column:セッションオブジェクトとSerializable
セッションに格納するデータは、通常はメモリ上に存在するので、直列化可能である必要はありません。
ただしセッションに格納されたデータをリモート転送したり、ハードディスクに書き込む場合は直列化しておく必要があります。

04.XXXXX

05.XXXXX

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 16【サーブレットクラスの詳細】

01.アプリケーションのライフサイクル

02.アプリケーションのスコープ

```
ServletContext application = getServletContext();
Integer count = (Integer) application.getAttribute("count");
```

03.シングルインスタンスマルチスレッドモデル

いろんな議論があってあんまり考えたくない。。今はサラッと読み流す。
基本的にフレームワークが制御してくれる？

04.XXXXX

05.XXXXX

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 17【外部ファイルから値を読み込む】

01.サーブレットの初期化パラメーター

フィルターでも学習したが、初期化パラメータをWeb.xmlに記載すれば
javax.servlet.ServletConfigインターフェースを使用してアクセスできる。
ServletConfigオブジェクトを取得するには、HttpServletクラスのgetServletConfig()メソッドを使用します。
＊＊サーブレットごと
--web.xml

```
<servlet>
    <servlet-name>InitParamServlet</servlet-name>
    <servlet-class>init.InitParamServlet</servlet-class>
    <init-param>
        <param-name>key1</param-name>
        <param-value>value1</param-value>
    </init-param>
</servlet>
```

--java
```
ServletConfig config = getServletConfig();
Enumeration<String> names = config.getInitParameterNames();
while(names.hasMoreElements()){
    String name = names.nextElements();
    String value = config.getInitParameter(name);
    out.println(name + "=" + value);
}
```

02.アプリケーションの初期化パラメータ

＊＊アプリケーション全体
```
<context-param>
    <param-name>appkey1</param-name>
    <param-value>appvalue1</param-value>
</context-param>

<context-param>
    <param-name>appkey2</param-name>
    <param-value>appvalue2</param-value>
</context-param>
```

--java
```
ServletConfig context = getServletContext();
Enumeration<String> names = context.getInitParameterNames();
while(names.hasMoreElements()){
    String name = names.nextElements();
    String value = context.getInitParameter(name);
    out.println(name + "=" + value);
}
```

03.初期化パラメータのその他の話題

○init()メソッド
パラメータを利用して一度だけ何か処理をしたい場合はサーブレットのinit()メソッドを利用することのほうが多い
public void init()
public void init(ServletConfig config)
※注意※
ServletConfigを引数に取るメソッドをオーバーライドする場合は、必ずsuper.init(config)を呼び出す必要があります！！！！

○web.xmlの<load-on-startup>要素
init()メソッドが呼ばれるタイミングは最初にサーブレットがインスタンス化された時で、通常これは「最初にリクエストがあった」とき
初期化処理等を先に行っておきたい場合、Web.xmlの<servlet>要素の子要素として<load-on-startup>要素を指定すると、Webアプリケーションの起動時にサーブレットもロードされます。
要素に指定する値は0以上の数字で数字の小さいほうが先にロードされます。

```
<servlet>
    <servlet-name>InitParamServlet</servlet-name>
    <servlet-class>init.InitParamServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
```

04.物理パスの取得

```
ServletContext context = getServletContext();
String path = context.getRealPath("/WEB-INF/init.properties");
```

・Webアプリケーション上の特定の場所においた設定ファイルを読みこんだり、データを書き込んだりすることができる
・データベースを利用しないちょっとしたWeb

05.クラスパスからファイルを読む

```
URL url = ClasspathFileServlet.class.getResource("init.properties");
InputStream in = url.openStream();

Properties prop = new Properties();
prop.load(in);
in.close();
```

getResourceメソッドだと、クラス階層を元にファイルを探すので、

ファイル名だけ指定すればよい！

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 18【HTTPリクエストとレスポンス】

01.HTTPリクエスト

02.HTTPレスポンス

ステータスコード	説明
100-199	お知らせ
200-299	リクエストの成功
300-399	ファイルの移動
400-499	クライアントのエラー
500-599	サーバー側のエラー

ステータスコードの詳細		
ステータスコード(名前)	定数	説明
200(OK)	SC_OK	問題なし。通常のステータスコード
301(Move Permanently)	SC_MOVED_PERMANENTLY	指定されたドキュメントが他の場所にあることを示す。場所はLOCATIONヘッダで与えられている。
302(Found)	SC_MOVED_TEMPORARITY	301と似ているが、LOCATIONヘッダで与えられている場所は一時的な場所であることを示す。
304(Not Modified)	SC_NOT_MODIFIED	クライアントがドキュメントをキャッシュしていて、If-Modified-Sinceリクエストヘッダが送られていた時に、変更
404(Not Found)	SC_NOT_FOUND	指定されたページが存在しない
500(Internal Server Error)	SC_INTERNAL_SERVER_ERORR	CGI全般のエラー。サーブレットから例外を投げた時もこれ。HTTPDと連携して連携部分でエラーになった時

○ヘッダ
リダイレクトが可能

```
response.setContentType("text/html");
response.setStatus(HttpServletResponse.SC_MOVED_TEMPORARILY);
response.setHeader("Location","/");
```

03.XXXXX

04.XXXXX

05.XXXXX

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 19【クッキーの利用】

01.くっきーの基礎知識

クッキーはクライアント側に保存されるテキストデータです。
ユーザーID、パスワード、カートに入れた商品などはクッキーに保存しておくことでこのような仕組みが実現できます。

◯クッキーの制約
クッキーはサーバからのレスポンスやJavascriptなどによって制御されます。
不当なアクセスが行われないように次のようなセキュリティ上の制約がある。
・自分で発行したクッキーにしかアクセス出来ない
・サイズや数の制限
クッキーは全部で300個まで
1個のクッキーのサイズは4Kバイトまで
サーバ、ドメインごとに20個まで
・クライアントでオフにできる

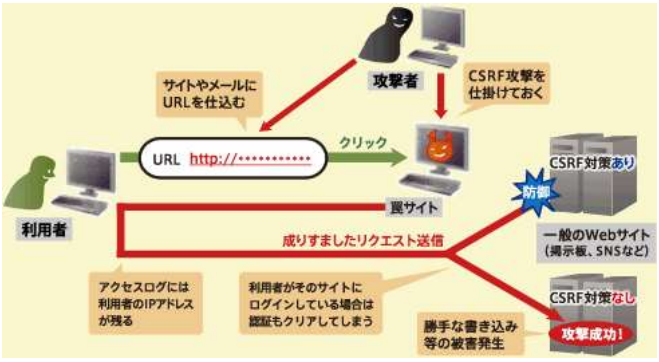
・HTTPヘッダにより送信される
クッキーはHTTPヘッダにより送受信されます。
通信が暗号化されていない場合(ただのHTTP通信の場合)、途中で盗聴される危険性はゼロではありません。
また脆弱性のあるサイトでは、CSS(クロスサイトスクリプティング)、CSRF(クロスサイトリクエストフォージェリ)と言った攻撃でセキュリティ・ホールをつかれ、
ユーザーのクッキーが盗まれたり意図せぬスクリプトを実行させられてしまうこともあります。
■CSS
http://www.atmarkit.co.jp/ait/articles/0211/09/news002.html

■CSRF
https://www.scutum.jp/information/web_security_primer/csrf.html

攻撃の仕組み

攻撃の仕組みの大まかな流れは、以下のようになります。

【1】 攻撃者が罠を仕掛けたWebサイトを用意する
【2】 被害者が罠サイトを閲覧する
【3】 罠サイトを閲覧した被害者のPCから、被害者がそのときアクセスするつもりのないWebサイトの掲示板などに強制的にアクセスさせられ、コメントなどが書き込まれる
【4】 書き込まれたWebサイトの接続ログを調べると、被害者のPCからのアクセス情報（IPアドレスなど）が記録されている



○サーブレットのクッキーAPI
javax.servlet.http.Cookieクラスで操作される

getCookie()メソッド

public Cookie[] getCookies()

02.クッキーを利用するサンプル

03.クッキーに関するその他の話題

○保存されたクッキー
IE
ツール＞インターネットオプション＞全般タブ＞閲覧の履歴＞設定＞ファイルの表示
FireFox
ツール＞オプション＞プライバシー＞Cookieを個別に削除
sqliteというデータベースに保存されている。
上記でCookieの情報を確認することが可能

○URLリライティング
クライアントでクッキーが使用できない環境の場合
URLリライティングでHTMLの次のリンク部分にjssessionidというパラメータを不可することで実現
encodeURL(String url)

○日本語を保存する場合

```
String keyParam = request.getParameter("key");
String valueParam = request.getParameter("value");
if(keyParam != null && keyParam.length() > 0){

    keyParam = URLEncoder.encode(keyParam, "Windows-31J");
    valueParam = URLEncoder.encode(valueParam, "Windows-31J");

    Cookie cookie = new Cookie(keyParam, valueParam);
    cookie.setMaxAge(60 * 60 * 24 * 5);
    response.addCookie(cookie);
}

Cookie[] cookies = request.getCookies();
if (cookies != null) {
    for (int i = 0; i < cookies.length; i++) {
        Cookie cookie = cookies[i];
        String key = cookie.getName();
        String value = cookie.getValue();

        key = URLDecoder.decode(key, "Windows-31J");
        value = URLDecoder.decode(value, "Windows-31J");

        out.println(key + "=" + value + "<br>");
    }
}
```

04.XXXXX

05.XXXXX

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 20【WARファイルとデプロイ】

01.標準のデプロイ順序

①必要なファイルの作成
②パッケージング
③デプロイ

02.Managerアプリケーション経由のデプロイ

○Manageアプリケーションの準備

%TOMCAT_HOME%/conf/tomcat-users.xml

```
<tomcat-users>
<!--
  NOTE:  By default, no user is included in the "manager" role required
  to operate the "/manager" web application.  If you wish to use this app,
  you must define such a user - the username and password are arbitrary.
-->
<!--
  NOTE:  The sample user and role entries below are wrapped in a comment
  and thus are ignored when reading this file. Do not forget to remove
  <!.. ..> that surrounds them.
-->
<!--
  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <user username="tomcat" password="tomcat" roles="tomcat"/>
  <user username="both" password="tomcat" roles="tomcat,role1"/>
  <user username="role1" password="tomcat" roles="role1"/>
  <user username="admin" password="admin" roles="manager" />
-->
</tomcat-users>
```

○パッケージング
jar cvf depapp.war .

Column ： Ant
Apacheプロジェクトが開発提供しているビルドツール
アントでは多数のタスクが標準で用意されており、ユーザーはタスクを利用すればたいていの処理をすることが出来ます。

```
  <role rolename="manager"/>
  <user username="admin" password="admin" roles="manager"/>
```

○デプロイ

http://localhost/manager/html
admin
admin
ログイン後、
画面下部の配備！
配備に成功すると、
/depappというアプリケーションが追加される

この方法で、下記URLでアプリケーションにアクセス可能となる。
http://localhost:8080/depapp

この方法であればリモートのサーバーに対してデプロイすることも可能
サーバを起動したままでアプリケーションのデプロイを行うことは「ホットデプロイ」と呼ばれます。

○無駄にアプリケーションを増やすとtomcatの動作が遅くなってしまうので、depadppはアンデプロイしておく
「配備解除」を押下すると完了！

このように標準の手順はWARにパッケージングしてデプロイツールでデプロイという手順であることを覚えておくこと
ただしJSPを１行直しただけで毎回WARにしてデプロイすることは効率的ではないので、多くのアプリケーション・サーバーが
Tomcatの簡易デプロイと同等の仕組みが提供されていまs。

03.Tomcatのコンテキストファイル

web.xml　：　サーブレット/JSP標準の設定を行うもの
コンテキストファイル　：　tomcat独自の設定を行うもの

○コンテキストファイルの場所
①<tomcatディレクトリ>/conf/Catalina/localhost/xxx.xml(xxxは任意の名前)
②<webアプリケーション>/META-INF/context.xml
①は②のファイルが存在しなかった場合のみ有効になります。
C:¥servletbook¥apache-tomcat¥conf¥Catalina¥localhost
xxxはプロジェクト名が入る。コンテキスト定義の行進をするとここのファイルが最新化される仕組み

○コンテキストファイルを利用したデプロイ
TomcatではコンテキストファイルのdocBase属性を使って、任意の場所に作成したWebアプリケーションを
webappsの下にコピーすることなしに実行出来ます。

▼Context要素の属性
path　　:コンテキストパス
docBase　:Webアプリケーションを配置しているディレクトリのパス
reloadable :オートリロードの設定。trueを指定するとオン、デフォルトはオフ
workDir　:JSPサーブレットなどが出力される作業ディレクトリのパス
<Context path="/mores" reloadable="true" docBase="C:¥servletbook¥workspace¥mores"
workDir="C:¥servletbook¥workspace¥mores¥work" />

○server.xmlとコンテキストファイル
Tomcat5より前では、server.xmlにContext要素の記述を行っていました。
Tomcat5以降もserver.xmlにContext要素を記述出来ますが、server.xmlではなくコンテキストファイルを利用することが推奨されています。

04.XXXXX

05.XXXXX

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 21【アクションとディレクティブ】

01.アクションによるインクルード

<jsp:include page="header.jsp" />
<jsp:~ page="~" />というタグはJSPのアクションと呼ばれる

○解説
JSPでは<jsp:include>タグで他のリソースをインクルードします。
include_action.jspでheader.jspをインクルードします。
この<jsp:include>タグはサーブレットのRequestDispatcher#incluede()メソッドと同じ動作をします。

▼includeアクションの属性
page　:インクルードするページ。相対パスか、"/"で始まるコンテキストルートからのパスを指定する
flush :インクルード前にレスポンスバッファをクリアするかどうかの指定。trueを指定した場合はクリアする

また、includeアクションは子要素にパラメータを指定可能
<jsp:include page="header.jsp">
　　<jsp:param name="user" value="john">
</ jsp:include>
パラメータを指定した場合は、インクルードされるJSPからはgetParameter()メソッドで取得出来ます。
<% String value = request.getParameter("user"); %>

02.ディレクティブによるインクルード

もう一つJSPをインクルードする方法がある。
includeアクションではなく、<@include>というディレクティブを利用する方法

***include_directive.jsp
<%@include file="common.jsp"%>
<html>
<body>
<%=new Date()%>
</body>
</html>
***

***common.jsp
<%@page import="java.util.Date"%>

***

▼includeディレクティブとアクション
includeディレクティブは静的なファイルをインクルードを行うので、
このサンプルのようにimport文の記述を取り込むと言った使い方が可能です。
一方includeディレクティブではimport文を取り込むことは不可能
・インクルード先のファイルを更新して、インクルード元のJSPにアクセスしても変更内容が更新されない
・fileは物理的なパスを指定するので、URLマッピングしたサーブレットなどは読み込めない

03.アクションとディレクティブ

includeディレクティブはファイルが結合される
includeアクションはRequestDispatcher#include()メソッドに変換される

04.XXXXX

05.XXXXX

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

01.JSPの要素

・コア要素
・ディレクティブ
・アクション

02.コア要素

・スクリプトレット
・式
・宣言
・コメント
・EL式
①スクリプトレット(scriptlet)

```
<%
for(int i=0; i<10 ; i++ ){
    out.println("count:" + i + "<br>");
}
%>
```

②式(expression)
Javaの式なので文の区切り ";"はつけません(つけるとコンパイルエラーになります)

```
<%="hello" %>
<%= 3 * 4 * 5%>
```

③宣言(declaration)
JSPでも、フィールドやメソッドを宣言することが出来ます。

```
<%! ～  %>
<%!
    String message;
%>
<%!
    int add(int a, int b){
        return a + b;
    }
%>
```

Javaの「フィールド変数」になります。
フィールド変数はスレッドセーフではないので注意が必要
単にスクリプトレット内で利用する変数を定義したい場合は、スクリプトレットで定義すればよいでしょう。
スクリプトレットで変数を定義した場合、「ローカル変数」となります。

④コメント
JSPのコメントは<%-- ～ --%>で記述します。

⑤EL式
簡潔な表現でオブジェクトにアクセスする為の式言語です。JSP2.0で導入される
＄{～}で記述する。

03.ディレクティブ

▼ディレクティブの種類

名前	説明
page	JSPページ全体の設定
include	他のファイルの静的なインクルード
taglib	カスタムタグの定義
tag	タグファイルにおけるpageディレクティブのようなもの
attribute	カスタムタグの属性を宣言する
variable	EL式の変数を定義する

○pageディレクティブ

属性	説明	デフォルト
language	JSPページの言語。JSPの実装ベンダーがJAVA以外の言語を実装した場合の為にこの属性が用意	Java

	されているが、通常Javaである。	
extends	JSPページの親クラス。一般的開発者はextendsよりカスタムタグを利用したほうがよい。	
import	クラスのimport宣言。複数記述して意味を成すのはimportのみ	
session	trueの場合、JSPが呼ばれたときにセッションオブジェクトを作成し、暗黙オブジェクトsessionに代入する	TRUE
buffer	レスポンスバッファ・サイズ	
autoFlush	レスポンスバッファの自動フラッシュの有無	TRUE
isTreadSafe	JSPページがスレッドセーフかどうかの指定。Trueの場合はJSPがマルチスレッドで動作する。	TRUE
errorPage	キャッチしなかったエラーがスローされた場合の遷移先URL	text/html; ISO-8859-1
contentType	MIMEタイプと文字エンコーディングの指定。サーブレットのHttpServletRequest#setContentType()メソッドと同じ	FALSE
isErrorPage	trueの場合、JSPが呼ばれたときにセッションオブジェクトを作成し、暗黙オブジェクトsessionに代入する	
pageEncoding	JSPファイル自体の文字エンコーディング。省略した場合はContentTypeで指定した文字コード	ISO-8859-1
isELIgnored	ページ中のEL式を無視するかどうかを指定する。Trueの場合は無視する	FALSE
info	JSPページの追記情報を指定する。指定した値は、getServerInfo()の返り値として利用される	
deferredSyntaxAllowedAsLitera	#{~}という文字列リテラルを許可するか。Trueの時許可する	FALSE
trimDirectiveWhitespaces	ディレクティブ行の空白をJSP出力から除去するか。Tureの場合は除去する	FALSE

04.アクション

▼標準アクション一覧
＊＊＊＊＊＊＊＊＊＊

05.暗黙オブジェクト

▼暗黙オブジェクト一覧
＊＊＊＊＊＊＊＊＊＊＊

06.4つのスコープ

ページ＜リクエスト＜セッション＜アプリケーション
ページスコープの暗黙オブジェクトは"page"ではなく"pageContext"です

07.JSPのページの一括設定

web.xmlの<jsp-pproperty-group>要素で複数のJSPを一括で設定出来ます。

ちょっとここでは覚えきれないので、とばす。

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 23【JavaBeansとEL式】

01.EL式とは

EL(Expression Language:式言語)はJSPの式をより簡潔に表現する記述方法です。
JSPの式では<%=～%>で変数へアクセスしますが、EL式では${～}という書き方で変数へアクセスします。

```
<%= ((UserBean)request.getAttribute("user")).getName()%>
${user.name}
```

JSPの式は、標準アクションやカスタムタグの属性としてもよく利用されます。
アクションの属性に指揮を指定すると、次のような記述になります。

```
<jsp:include page="<%=((PageController)request.getAttribute("ctrl")).getHeader()%>" />

<jsp:include page="${ctrl.header}" />
```

02.EL式を使ってみよう

○オブジェクトへのアクセス
スコープ変数の検索
ページ＞リクエスト＞セッション＞アプリケーションの順番で検索され先に見つかったものから取得されます。
○オブジェクトのプロパティへのアクセス
▼JavaBeansとプロパティ
ビジネスロジックとデータをカプセル化した再利用可能なコンポーネントのことです。
***access.jsp

```
<%@page contentType="text/html; charset=Windows-31J"%>
<%@page import="el.*"%>

<html>
<body>

<h2>ELのテスト</h2>

<%
  //準備
  User user = new User();
  user.setName("太郎");
  user.setMale(true);
  request.setAttribute("user", user);
%>

<pre>
名前は ${user.name}

男ですか？ ${user.male}

ちなみに ${user}
</pre>
</body>
</html>
***
```

***user.java

```
package el;

public class User {

    private String name;
    private boolean male;
    private User friend;


    public User(){
    }
    public User(String name, boolean male){
        this.name = name;
        this.male = male;
    }


    public boolean isMale() {
        return male;
    }

    public void setMale(boolean male) {
        this.male = male;
    }
```

```
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public User getFriend() {
        return friend;
    }

    public void setFriend(User friend) {
        this.friend = friend;
    }
}
```

${user} はuserオブエジェクトのtoString()メソッドが返却される

○集合や配列の要素へのアクセス

```
<%@page contentType="text/html; charset=Windows-31J"%>
<%@page import="java.util.*"%>

<html>
<body>

<h2>ELのテスト</h2>

<%
  List list = new ArrayList();
  list.add("list0");
  list.add("list1");
  list.add("list2");

  String[] array = new String[]{
    "array0",
    "array1",
    "array2"
  };

  Map map = new HashMap();
  map.put("keyA", "mapA");
  map.put("keyB", "mapB");
  map.put("keyC", "mapC");

  request.setAttribute("list", list);
  request.setAttribute("array", array);
  request.setAttribute("map", map);
%>


<p>リストから ${list[2]}

<p>配列から ${array[2]}

<p>マップから ${map["keyA"]} / これもOK ${map.keyA}


</body>
</html>
```

03.演算子

簡単な演算をすることもできる。！

```
<%@page contentType="text/html; charset=Windows-31J"%>
<%@page import="java.util.*"%>

<html>
<body>

<h2>ELのテスト</h2>

<%
 //準備
 List list = new ArrayList();
 request.setAttribute("list", list);
%>


<pre>

${ 1 + 2 }

${ 1 < 2 }

${ true && false }
```

```
${ 1 < 2 ? 2 : 1 }

${ empty list }

</pre>


</body>
</html>
```

04.XXXXX

05.XXXXX

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

	chapter 24【カスタムタグとJSTL】

01.カスタムタグとJSTLについて

JSP標準アクションは、よく利用される処理をタグの形で提供したものです。例えば、インクルード処理を行う場合、
スクリプトレットでは次のようなコードになります。

```
<%
request.getRequestDispatcher("/page1.jsp").include(request, response);
%>
```

これを標準アクションを利用することで、次のように簡略化することができます。

```
<jsp:include page="/page1.jsp">
```

カスタムタグはこのようなタグによる処理の記述をユーザーが独自に定義できるようにしたものです。
例えば標準アクションにはない以下のようなループ処理を

```
<%
int count  = 10;
for (int i=0; i<count; i++){
    //...
}
%>
```

カスタムタグにすることにより、以下の様な記述が可能になります。

```
<mytag:loop count="10">
...
</mytag>
```

**誰もがよく行う処理を「サード・パーティのライブラリ」から一歩進んで、
「Javaの標準ライブラリー」に近い位置づけで利用できるようにしたのがJSTL(JavaServer Pages Standard Tag Library)**

02.JSPから利用する

```
<%page contentType="text/html; charset=Windows-31J"%>
<%taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<html>
<body>

<c:out value＝"こんにちわ" />

</body>
</html>
```

○JSTLだけでなく、カスタムタグをJSPで利用する場合は
・taglibディレクティブの宣言を行う
・taglibディレクティブで宣言したprefixでタグを利用する

JSTLのCoreタグは慣習的にcを利用します。

Column：JSTLのバージョン
JSTL1.2:JSP2.1以上(Tomcat6)
JSTL1.1:JSP2.0以上(Tomcat5)
JSTL1.0:JSP1.2以上(Tomcat4)
また、JSTLの１．０と１．１以降では指定するURIが異なっているので注意
Coreタグの場合は以下のようになる。
JSTL1.1: http://java.sun.com/jsp/jstl/core
JSTL1.0: http://java.sun.com/jstl/core

03.Coreタグの利用

JSTLは単一のカスタムタグライブラリとしてではなく、以下のように複数の機能のカスタムタグの集まりとして提供されています。
▼JSTLの種類

種類	説明	Wiki
Core	スコープ変数の操作、繰り返し、条件分岐などの基本的なタグ	
XML processing	XML文書の操作を行うタグ	
I１８N(インターナショナライゼイション)	数値や日付などのフォーマットを行うタグ。国際化機能をサポートするタグとしてi18nタグと呼ばれる	ソフトウェアの多言 めの設計や仕様な うにしたり、メニュー 離したりといった改 国際化されたソフ （ローカライゼーシ 応を組み込み、利 きるようにすること M17N）という。 関連用語
SQL	JSP上でデータベースアクセスを行うタグ。非常に簡単にSQL発行が記述できる	
Functions	文字列の加工などの関数を提供している。カスタムタグのFunction	

▼JSTLのURIとPrefix

種類	URI	Prefix
Core	http://java.sun.com.jsp/jstl/core	c
XML	http://java.sun.com.jsp/jstl/xml	x
I18N	http://java.sun.com.jsp/jstl/fmt	fmt
SQL	http://java.sun.com.jsp/jstl/sql	sql
Functions	http://java.sun.com.jsp/jstl/functions	fn

この中でも利用頻度の高いCoreタグに付いて説明

▼Coreタグの一覧

機能	タグ	説明
変数	set	スコープ変数をセットする
	remove	スコープ変数を削除する
フロー制御	if	条件分岐
	choose	yes/noより複数の条件分岐
	forEach	ループ
	forTokens	文字列の走査
URL管理	import	<jsp:include>に似ているが、サーバー外のリソースもインクルードできる
	redirect	リダイレクトを行う
	url	リンクなどのURLを生成する。パラメータの指定やURLエンコードが可能
その他	catch	エラーをキャッチする
	out	出力を行う

○変数の操作
<set>タグの属性

名前	必須	リクエスト時	型	説明
var	FALSE	FALSE	java.lang.String	スコープ変数の名前
value	FALSE	TRUE	java.lang.String	セットする値
target	FALSE	TRUE	java.lang.String	スコープ変数の名前。プロパティを利用する場合は、targetを指定する。対象オブジェクトはJavaBeanのセッタかMapのキー
property	FALSE	TRUE	java.lang.String	プロパティ名。Targetとともに指定する
scope	FALSE	FALSE	java.lang.String	変数のスコープをpage,request,sesion,applicationで指定する。省略した場合はpageスコープ

※「リクエスト時」がTRUEというのは、JSPに直接記述する固定の文字列だけでなく、リクエスト時に動的に変わる値を指定できるという意味です。

targetとproperty属性はスコープ変数に値をセットするのではなく、スコープ変数のオブジェクトのプロパティに値を設定します。この場合スコープ変数はSetterを持つオブジェクトかMapである必要があります。

```
<c:set target="${user}" property="name" value="john" />
```

<remove>タグ

名前	必須	リクエスト時	型	説明
var	TRUE	FALSE	java.lang.String	削除する変数の名前
scope	FALSE	FALSE	java.lang.String	変数のスコープ

○フロー制御
<if>タグ
条件分岐によってこ要素を評価するかしないかを決めるタグです。

名前	必須	リクエスト時	型	説明
test	TRUE	TRUE	boolean	子要素を評価するかどうかの条件
var	FALSE	FALSE	java.lang.String	条件の結果を格納するスコープ変数の名前。スコープ変数はBoolean
scope	FALSE	FALSE	java.lang.String	変数のスコープ

```
<c:set var="age" value="19" />

<c:if test="${age < 20}">
20未満です。
</c:if>
```

trueの時に子要素が評価されるので、条件によってこ要素を表示したい時に便利かもね。

<choose>タグ

```
<c:choose>
    <c:when test="${age >= 20}">
        成人
   </c:when>
    <c:when test="${age >= 10}">
        10代
    </c:when>
    <c:otherwise>
        子ども
    </c:otherwise>
</c:choose>
```

<forEach>タグ

名前	必須	リクエスト時	型	説明
items	FALSE	TRUE	java.lang.String	ループを行うコレクション
begin	FALSE	TRUE	int	取り出す先頭要素のインデックス。最初は０。省略した場合は０
end	FALSE	TRUE	int	取り出す最後の要素のインデックス。省略した場合は最後の要素
step	FALSE	TRUE	int	1ループですすめる要素数
var	FALSE	FALSE	java.lang.String	取り出した要素を格納する変数の名前。子要素からアクセスできる
varStatus	FALSE	FALSE	java.lang.String	現在のループの状態を示す変数(ステータス変数)の名前。子要素からアクセスできる

```
<c:forEach var="item" items="${list}" varStatus="status">
   ${status.index}
   ${item}
</c:forEach>
```

▼ステータス変数のプロパティ

名前	説明
current	var属性が示す現在のオブジェクト
index	０から始まるループカウント
count	１から始まるループカウント
first	現在がループの銭湯であればtrue
last	現在がループの最後であればtrue
begin	begin属性に指定した値
end	end属性に指定した値
step	step属性に指定した値

<forTokens>タグ
<forEach>タグの対象は配列や集合でしたが、<forTokens>タグの対象は「文字列」です。
▼<forTokens>タグの属性

名前	必須	リクエスト時	型	説明
items	TRUE	TRUE	java.lang.String	ループ対象の文字列
delims	TRUE	TRUE	java.lang.String	区切り文字
begin	FALSE	TRUE	int	ループ開始のインデックス。最初は０
end	FALSE	TRUE	int	ループを終了するインデックス
step	FALSE	TRUE	int	１ループですすめる要素数
var	FALSE	FALSE	java.lang.String	取り出した要素を格納する変数名。変数は子要素からアクセスできる
varStatus	FALSE	FALSE	java.lang.String	現在のループの状態を示す変数の名前。小他所からアクセスできる。

```
<%
String text = "aaa,bbb,ccc";
request.setAttribute("text",text);
%>
<c:forTokens var="item" items="${text}" delims=",">
   ${item}
</c:forTokens>
```

○URL管理
<import>タグ

名前	必須	リクエスト時	型	説明
url	TRUE	TRUE	java.lang.String	インポートするリソースのURL
var	FALSE	FALSE	java.lang.String	インポートした結果を格納するスコープ変数
scope	FALSE	FALSE	java.lang.String	varのスコープ
varReader	FALSE	FALSE	java.lang.String	インポートしたリソースを読みだすReaderの変数名
context	FALSE	TRUE	java.lang.String	外部コンテキストからインポートする場合のコンテキストの名前
charEncoding	FALSE	TRUE	java.lang.String	インポートするリソースの文字エンコーディング

<redirect>タグ
<c:redirect url="http://yahoo.co.jp">

<url>タグ
URLを作成するタグです。主要な用途は、クッキーが使えない場合の
セッションIDや日本語パラメータなどをURLに含めた、パラメータ月のURLを作成します。

名前	必須	リクエスト時	型	説明
var	FALSE	FALSE	java.lang.String	作成したURLを格納する為のスコープ変数の名前
scope	FALSE	FALSE	java.lang.String	varのスコープ
value	FALSE	TRUE	java.lang.String	処理するもととなるURL
context	FALSE	TRUE	java.lang.String	外部コンテキストからの相対パスを指定する場合のコンテキスト名

```
<c:url var="url" value="/jstl/hello.jsp">
    <c:param name="name1" value="ほげ" />
</c:url>

<a href="${url}">link</a>
```

これによりスコープ変数(デフォルトはpage)には、以下のようなURLの値が格納されます。
/morej/jstl/hello.jsp?name1=%82%d9%82%b0

○その他
<catch>タグ
<out>タグ

04.TLDとURIのマッピング

05.XXXXX

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 25【データベースの基礎知識 】

01.データベースの基礎知識

```
CREATE TABLE ACCOUNT (
    ID INTEGER PRIMARY KEY,
    NAME VARCHAR(100),
    MONEY INTEGER
);

DROP TABLE ACCOUNT

INSERT INTO ACCOUNT VALUES(1, 'ボブ', 1000)
INSERT INTO ACCOUNT VALUES(2, 'パトリック', 2000)
INSERT INTO ACCOUNT VALUES(3, 'サンディー', 3000)

SELECT * FROM ACCOUNT;

-- CTRL + ENTERでsql実行できる!!
```

02.HSQLDBのインストール

03.XXXXX

04.XXXXX

05.XXXXX

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 26【JDBCを使う】

01.JDBCを利用したプログラムの全体像

1. Class.forName()メソッドでJDBCドライバをロードする(classは最初にロードされるときstaticイニシャライザが実行される！)
2. DriverManager#getConnection()メソッドでConnectionオブジェクト(DBとの接続) を取得する
3. Connection#createStatement()メソッドでStatementオブジェクト(SQLの実行口)を取得する
4. 更新の場合、Statement#executeUpadate()メソッド、検索の場合はStatement#executeQuery()メソッドを実行する

Class.forName()メソッドの利点
クラスパスに該当のhsqldb.jarを含めずにJavaの標準クラスだけでデータベースに接続することができる点

02.データベースの更新

03.XXXXX

04.XXXXX

05.XXXXX

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 27【Webアプリケーションとデータベース】

01.Webアプリケーションでのデータベースのりよう

リクエストごとのコネクション
・コネクションはリクエストのたびに開放する！！
「コネクションの数は有限」で特にWebアプリケーションの場合
、セッションなどでコネクションを保持するトすぐに足りなくなってしまうからです。
HTTPは「根本的にステートレス(状態がない)である」ということ
セッションという仕組みは有りますが、セッションが提供するのは擬似的な状態の維持
サーバ側からはクライアントが処理が終わったのか考え中なのかトラブルでシャットダウンしたのかを知ることが出来ません。
セッションタイムアウトのように一定時間アクセスが内ことで「接続が切れた」とみなすことは出来ますが、
その一定期間まで、使われないセッションは多数残ります。
Javaインスタンスが占めるメモリリソースも有限ですが、データベースのコネクションのほうがはるかに早く枯渇します。。

02.XXXXX

03.XXXXX

04.XXXXX

05.XXXXX

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 28【データベースアクセスの改良】

Column：デザインパタンとJava EEパターン
過去にうまく行ったノウハウをまとめものをデザインパターンという
ソフトウェア開発に於ける「おばあちゃんの知恵袋」的な存在
有名なのはGofのデザインパターンとJava EEパターンがアリます。

01.XXXXX

02.XXXXX

03.XXXXX

04.XXXXX

05.XXXXX

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 29【データ・ソースの利用】

01.データソースの概要

○コネクションプール
コネクションの取得や解放はコストの高い処理なので、この問題を解消する為に「コネクションプール」という方法がとられる
予めコネクションを複数取得しておき、実際に使用するときには新たに接続に行くのではなく、プールの中から割り当てるという方法

○データ・ソース
コネクションプールなどの機能を提供する技術は「データ・ソース」と呼ばれます。
データソースは、javax.sql.DataSourceインターフェースとして標準化されています。
アプリケーション内でNewして使用することも出来ますが、通常は後述の「JNDI経由でルックアップ」という方法で取得します。

○JNDI(Java Naming and Directory Interface)
ネーミングサービス及びディレクトリサービスを利用するためのAPIです。
ネーミングサービスとは、セッション属性のマップの高機能版のようなものです。
ネーミングサービスはアプリケーション・サーバーにセットで付いていることが多いです。！
ディレクトリサービスもネーミングサービスと似ていますが、階層構造でオブジェクトを管理します。

ネーミングサービスに登録されたオブジェクトをアプリケーションから検索して、取得する処理はルックアップと呼ばれまス

02.Tomcatでのデータ・ソースの利用

前準備
・JDBCドライバを適切な場所に設置
・コンテキストファイルの編集

○JDBCドライバの配置
JDBCを直接利用する場合はWebアプリケーションのクラスパス(WEB-INF/libなど)にJDBCドライバを配置しましたが、
データ・ソースを利用する場合は、アプリケーション・サーバーが利用するクラスパスにJDBCドライバを配置する必要があります。
<tomcatをインストールしたディレクトリ>/lib

○コンテキストファイルの編集
Tomcatでデータ・ソースを登録する方法はいくつかあります。
一番簡単なのはコンテキストファイルに<Resource>要素を登録する方法です。
WebアプリケーションのMETA-INFディレクトリのしたにcontext.xmlという名前で作成する！
※注意※
EclipseのTomocatプラグインを使用している場合は、META-INF/context.xmlではなく、<Tomcatディレクトリ>/conf/Catalina/localhost以下にあるファイル
を編集してください！

```context.xml

<Context>
    <Resource name="jdbc/myds" auth="Container"
        type="javax.sql.DataSource"
        username="sa" password=""
        driverClassName="org.hsqldb.jdbcDriver"
        url="jdbc:hsqldb:hsql://localhost" />
</Context>

```


```
package websample;

import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;

import javax.naming.InitialContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.sql.DataSource;



public class DataSourceTest extends HttpServlet{


   public void doGet(HttpServletRequest request,
         HttpServletResponse response)
         throws ServletException, IOException {

      response.setContentType("text/plain; charset=Windows-31J");
      PrintWriter out = response.getWriter();
```

```
        try{
            InitialContext ctx = new InitialContext();
            DataSource ds = (DataSource)ctx.lookup("java:comp/env/jdbc/myds");
            Connection con = ds.getConnection();
            out.println("con=" + con);

        }catch(Exception e){
            throw new ServletException(e);
        }

    }
}
```

1. InitialContextオブジェクトを取得
2. InitialContextからDataSourceオブジェクトを取得
3. DataSourceからConnectionオブジェクトを取得

javax.naming.InitialContextはJNDI経由でリソースを取得する場合の窓口となるクラスです。

03.XXXXX

04.XXXXX

05.XXXXX

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn: