

# メディア情報応用 第6週

Advanced Topics in Media Informatics Week 06

金沢工業大学 情報フロンティア学部  
メディア情報学科

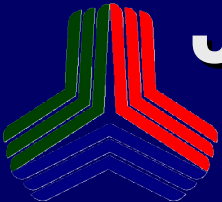
山岸 芳夫

Yoshio Yamagishi

Dept. of Media Informatics

Col. of Informatics and Human Communication

Kanazawa Inst. Tech.



# Javaサーブレット(Servlet)

- サーバサイドで動作する Java プログラム。実行環境としては**サーブレットコンテナ**と呼ばれるサーバが必要となる

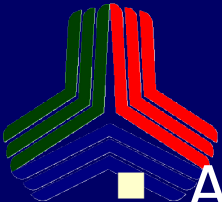
A server-side Java program. The server called servlet container is required to execute a servlet program

- 信頼性が高く、業務用アプリケーションなどミッションクリティカルな状況にも対応できる

Highly reliable and available for mission-critical use.

- データベースとの連携や、WebSocketにも対応可能

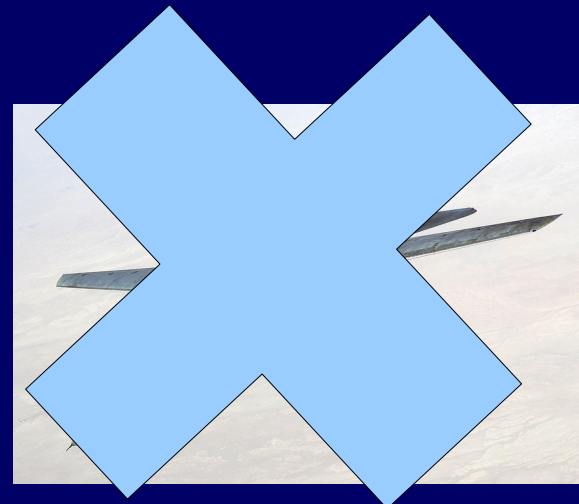
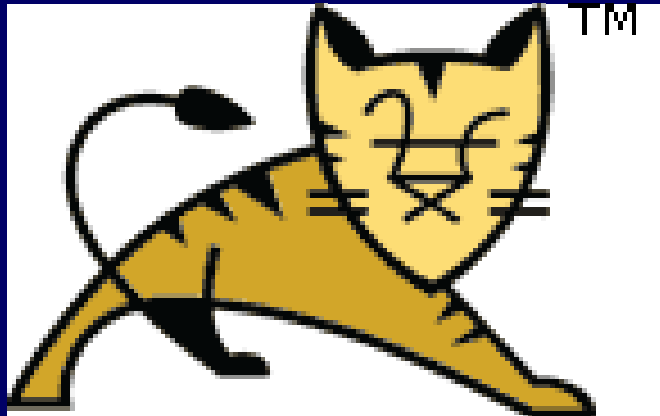
Database connection and WebSocket extension are also implemented.

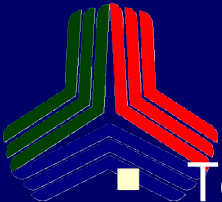


# Apache Tomcat

- Apache Software Foundation によるオープンソースのサーブレットコンテナ。サーブレットやJSPの実行を可能にする。それ自体 Web サーバの機能も持っている（グラマンF-14とは関係無し）

An open-source servlet container built by Apache Software Foundation, which provides execution environment of servlet and JSP programs. It also have a web server function(No relation to Grumman F-14!)





# Apache Tomcat

- Tomcatの実行にはJavaソフトウェアが必須。本来はOracle純正が望ましいとされているが、今回はCentOS純正の OpenJDK バージョンを用いる。

Some kind of Java software is required to execute Tomcat. It is often said that the Oracle genuine Java software is preferable as the backend of Tomcat. However, we use OpenJDK Java software, which is the default package of CentOS.

- 今回の演習環境では既にポート8080にてTomcatが動作している。<http://localhost:8080> にアクセスして確認してみよう

Currently Tomcat server is already running on the virtual machine. Access <http://localhost:8080> to check running status of Tomcat.

- Apache httpd の `mod_proxy_ajp` を使えば httpd との連携も可能だが、今回はやらない

Tomcat can be combined into httpd if `mod_proxy_ajp` module is included in the httpd. However, we omit that at this time.

# Tomcatの設定

## Tomcat Configuration

- /usr/share/tomcat/conf/tomcat.confの以下の行を書き換えて日本語対応にして再起動しよう。

Add below string to /usr/share/tomcat/conf/tomcat.conf to change locale of Tomcat to Japanese.

# You can change your tomcat locale here

#LANG="en\_US"

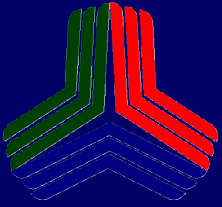
LANG="ja\_JP"

- 再起動は To restart Tomcat:

systemctl restart tomcat

# サーブレットのコンパイル

## Compilation of the Servlet

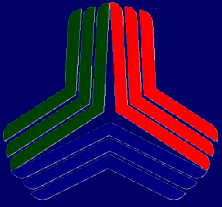


- サーブレットは基本的にJavaプログラムなので、コンパイルするためにはJavaの開発環境が必要。ただしクラスパスにサーブレットAPIを含める必要がある。今回の環境では以下のようにしてコンパイルを行う  
Since servlet is a Java program, a Java SDK is required to compile servlet. Note that the classpath must include the servlet API. In this VM environment, the servlet compilation will be done with below command:

```
javac -classpath /usr/share/tomcat/lib/tomcat-servlet-3.0-api.jar (Java source file name)
```

# web.xml の作成

## Creation of web.xml

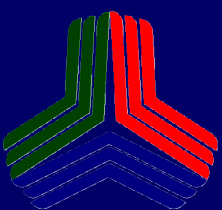


- URLとクラスファイル名の設定をするファイル。  
今回は最低限の設定を行っている

web.xml is a configuration file for the description of the relation between URLs and class files. At this environment, the web.xml includes minimum essential configuration only.

# サーブレットの配置

## Deployment of Servlet



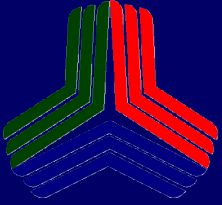
- サーブレットは tomcat ホーム (tomcatのインストールされたディレクトリ。この環境では /usr/share/tomcat ) の下の **webapps** ディレクトリ以下に配置する。**ディレクトリ**と**ファイル**の配置は以下の通り。サービス名、クラスファイル以外のファイル名、ディレクトリ名は原則的に変更できない  
The servlet executable must be placed in the webapps/ directory under the "tomcat home"( the directory which Tomcat is installed. In this environment, the "tomcat home" is /usr/share/tomcat). The deployment of the directories and files are following. The **file** and **directory** names except service name and class-file name cannot be changed.

**webapps/** - (service name) /- **WEB-INF/** - **classes/** - (class file name)  
- **web.xml**



# Tomcat webアプリケーションマネージャー

## Tomcat Web Application Manager

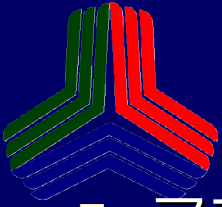


- <http://localhost:8080/manager> にアクセスすると、現在利用可能なアプリケーションが一覧できる。起動、停止、再ロード、配備解除もこの画面で可能である

In this environment, the URL of Tomcat web application manager is <http://localhost:8080/manager> . It has a list of the applications available in the environment and function of application management(i.e., start, stop, reload and undeploy)

# Tomcat webアプリケーションマネージャー

## Tomcat Web Application Manager



- アプリケーションマネージャーの画面を開くにはログインが必要であり、今回は

ユーザ名 : **admin**

パスワード : **adminadmin**

となっている（これは/usr/share/tomcat/conf/tomcat-user.confに書いてあるので、自分で好きなものに変更しても良い。変更したらsystemctlで再起動すること）

The user login is required to open the application manager. In this environment:

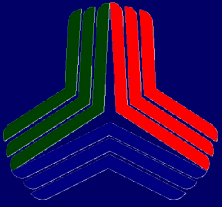
username: **admin**

password: **adminadmin**

The account is described in /usr/share/tomcat/conf/tomcat-user.conf . This can be customized by changing the description in this file. After customization, restart tomcat with systemctl command.

# サーブレットの実行

## Execution of Servlet



- `http://localhost:8080/ (サービス名) / (web.xml で設定したURL) /` にアクセスすると、サービスが実行できる。
  - Access `http://localhost:8080/(service name)/(URL configured in web.xml)/` to execute servlet.

# chat サーブレット(1)

## chat servlet(1)

- まず chat サーブレットをインストールするディレクトリを作成する

Firstly, make directories to install chat servlet.

```
mkdir /usr/share/tomcat/webapps/chat
```

```
mkdir /usr/share/tomcat/webapps/chat/WEB-INF
```

```
mkdir /usr/share/tomcat/webapps/chat/WEB-INF/classes
```

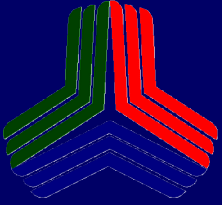
# chat サーブレット(2)

## chat servlet(2)

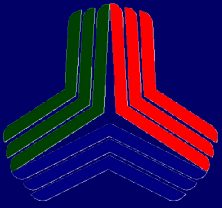
- Moodle の第6週から week06.tar.gz をダウンロードして展開し、できたweek06/ディレクトリの中から chat.java を/usr/share/tomcat/webapps/chat/WEB-INF/classes に、web.xml を/usr/share/tomcat/webapps/chat/WEB-INF にそれぞれコピーする  
Download "week06.tar.gz" from Topic 6 in Moodle and extract it. Then, copy chat.java and web.xml from week06/ directory which is created by week06.tar.gz extraction into /usr/share/tomcat/webapps/chat/WEB-INF/classes and /usr/share/tomcat/webapps/chat/WEB-INF respectively.

# chat サーブレット(3)

## chat servlet(3)

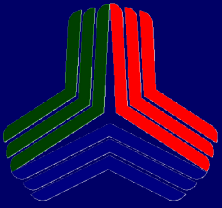


- chat.java をコンパイルしてclassファイルを作る。  
Compile chat.java to make class file.  
**javac -classpath /usr/share/tomcat/lib/tomcat-servlet-3.0-api.jar chat.java**
- あとは <http://localhost:8080/chat/chat> にアクセスすればチャット画面が表示される。ソースを変更しコンパイルし直した際は、manager 画面で再ロードした方が良い。  
Chat program appears when you access <http://localhost:8080/chat/chat> . You should better to reload the application with application manager after you modify and re-compile the source.



# JSP(Java Server Pages)

- Tomcatなどのサーブレットコンテナで利用可能な、PHPのように**スクリプト言語的に記述できるJavaプログラム**。HTMLの中にJavaのソースコードの記述が可能（<% と %>というタグに挟んだ部分）  
Script-like Java program available with servlet container. JSP source code can be directly written in the part between start tag "<%" and end tag "%>" in the HTML source.



# JSP(Java Server Pages)

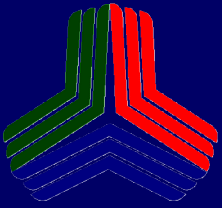
- JSPはサーブレットと異なりコンパイルする必要はなく、web.xmlなども用意する必要はないが、実は裏でTomcatがソースをサーブレットJavaソースに変換しコンパイルを行っている。そのため初回の起動が若干遅い。

Unlike servlet, JSP requires no compilation nor web.xml. However, Tomcat converts the JSP to servlet source and then compile it. Therefore, the response is relatively slow at the first time to access the JSP.

- このように、基本的にサーブレットと同じものなので、サーブレットでできる事はほとんどJSPでも可能でしかもお手軽。サーブレットと連携して使うことも可能

Since JSP is basically equivalent to servlet as shown above, whatever can be done with servlet can be mostly done with JSP. The collaboration between JSP and servlet is also available.





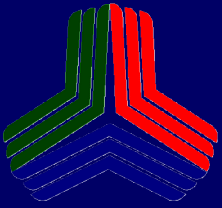
# JSP Chat

- まず JSP チャットをインストールするディレクトリを作成する

Firstly, make a directory to install JSP chat.

```
mkdir /usr/share/tomcat/webapps/jspchat
```


- week06/ ディレクトリの中の chat.jsp  
を /usr/share/tomcat/webapps/jspchat/ にコピーする  
Copy chat.jsp from the week06/ directory into  
/usr/share/tomcat/webapps/jspchat/



# JSP Chat

- あとは `http://localhost:8080/jspchat/chat.jsp` にアクセスすればチャット画面が表示される。ソースを変更した場合もmanager 画面で再ロードしなくてよい

Chat program appears when you access `http://localhost:8080/jspchat/chat.jsp` . Unlike the case of servlet, you need not reload the application with application manager after you modify and re-compile the source.



# バリデーション(Validation)

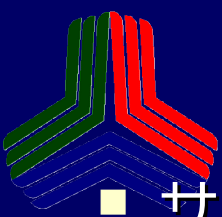
- 入力されたデータが正しいかどうかを判定する

Validate input data

- 入力ミス防止 / To prevent from input error
- バッファオーバーランやSQLインジェクションなどの不正アクセス防止 / To prevent unauthorized access like buffer-over-run or SQL injection

- 今回は**正規表現**によるバリデーションを行う。正規表現は新しい言語ならばほぼ使えるが、それぞれの言語で若干仕様が異なるものの、考え方は同じである。今回はJavascriptの正規表現を使う

At this time we use **regular expression** for validation. The regular expression is mostly available in modern programming languages but the detail of the specification is slightly different from each other. Here we take the regular expression of Javascript.



# バリデーション(Validation)

- サーバ側にデータを送ってサーバ側でバリデーションを行ってもよいが、Javascript を使えばサーバにデータを送る前にバリデーションを済ませることができる（水際作戦）

While it is also available to validate the data on the server-side, we can **complete the validation before data transmission** with Javascript(shoreline operation).

- week06/ ディレクトリにある removeTag.html を /var/www/html にコピーし、 http://localhost/removeTag.html にアクセスしてみよう。HTML タグを除去することができる。  
copy removeTag.html from week06/ directory into /var/www/html/ , then access http://localhost/removeTag.html . Confirm that HTML tag is completely removed.

# 正規表現

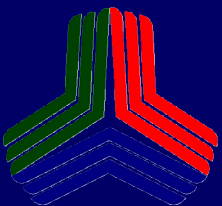
## Regular Expression:RegExp

- パターンマッチによる文字列検索に用いられる  
To find string with pattern-matching

Ex :

- “Javascript”で始まる行を検索したい  
To find lines start with "Javascript"
- “Javascript” または “Java script” のどちらかが含まれる行を検索したい  
To find lines include "Javascript" or "Java script"
- メールアドレスの形式(....@.....)の文字列が含まれる行を検索したい  
To find lines include the format of a mail address(....@.....)

# 正規表現の例



行頭にマッチ

- “Javascript” で始まる行に一致

⇒ `/^Javascript/`

直前の文字の0または1回の繰り返し

- “Javascript” または “Java script” に一致

⇒ `/Java ?script/`

行末にマッチ

- メールアドレスの形式に一致

⇒ `/^([\w\.-]+)@([\w\.-]+)$/`

直前の文字の1回以上の繰り返し

ピリオドにマッチ

英数字と“\_”にマッチ

ハイフンにマッチ



# Example of RegExp

matches the start position of the line

- matches lines start with "Javascript"

⇒ **`/^Javascript/`**

matches preceding character  
zero or one time

- matches "Javascript" or "Java script"

⇒ **`/Java ?script/`**

matches the end  
of the line

- matches the format of mail address

⇒ **`/^([\w\.-]+)@([\w\.-]+)$/`**

matches preceding character  
one or more times


matches "."

matches alphanumeric  
character and "\_"

matches "-"

# 正規表現関係のメソッド

## Methods of RegExp

- 
- `match()`  
正規表現のパターンにマッチした文字列を返す  
Returns string matches the pattern.  
使い方： `文字列.match(パターン)`  
Syntax: `(string).match(pattern)`
  - `search()`  
正規表現のパターンにマッチした文字列が先頭から何文字目にあったかを返す  
Returns the position of the string matches the pattern  
使い方： `文字列.search(パターン)`  
Syntax: `(string).search(pattern)`
  - `replace()`  
正規表現のパターンにマッチした文字列を別な文字列に置き換える  
Replace the string matches the pattern to new string  
使い方： `文字列.replace(パターン, 置き換える文字列)`  
Syntax: `(string).replace(pattern, other string)`



# マッチした文字列の利用

## Use of the Strings matches the Pattern

- グループ化することで、マッチした文字列を部分的に利用することができる。

Strings matches the pattern can be used partially by Grouping

例：メールアドレスにマッチするパターン

Ex. a pattern that matches a mail address

```
mailRegex = /^(([w\.-]+))@(([w\.-]+))$/  
                RegExp.$1  RegExp.$2
```

“yamagisi@neptune.kanazawa-it.ac.jp”.match(mailRegex);  
とすると、 / will be

“yamagisi” -> RegExp.\$1

”neptune.kanazawa-it.ac.jp” -> RegExp.\$2

# 最長一致と最短一致

## Greedy and Lazy Quantifier

- デフォルトでは正規表現は最長一致（その行の一番最後に見つかった文字に一致）

The default Quantifier of Javascript RegExp is "Greedy"

例： / Ex:

HTMLタグを表す正規表現... /<.+>/ とすると、 / This pattern matches

<h1>Javascript入門</h1>

がマッチしてしまう（本当は"<h1>", "</h1>"にマッチしてほしい）！  
while we think it should be mached with "<h1>", "</h1>"...

こういう場合は、最短一致（一番最初に見つかった文字に一致）を表すキャラクター"?"を追加して、 / In such case, we use a "Lazy Quantifier"

/<.+?>/

とすればよい。