初めに

MaaEngine は、Java＋Tomcat 上で動作する、小規模Web アプリケーション向けフレームワークです。構成として、MVC＋DI＋Dao となっており、JavaWeb アプリを OSS 上で開発する上で最低限の環境が搭載されています。

現在の OSS 上での Java 開発においては、下記のように、

- *Struts＋Spring＋Hibernate*
- *S2Struts＋S2Container＋S2Dao*

などの構成で行われることが多いと思いますが、幅広いニーズにも対応できるすばらしい環境の反面、汎用的であることから、新しい環境構築が大変であるし、開発当初において、フレームワーク自体に慣れる必要があり、敷居が高いと思ったのは、私だけではないと思います。

最近では、Ruby on Rails(以下 Rails)が開発効率の良さから、多くの Web アプリ開発者が、既存の言語から、Rails 環境に移行していますが、その流れに乗って、私自身もこの開発環境を少し使ってみました。

Rails を使ってみて思ったことは、

1. *ネーミングルールが強力で、外部定義をほとんど必要としない。*
2. *Rake コマンドにより、データベースも Rails の一部に思えるほどである。*
3. *Ruby 自体の言語特性から、プログラムを書く量が少なくて済む。*
4. *Rails を覚える＝Web アプリケーションを作れるようになる感じがする。*

のように、まとめると、『Web 開発において、Rails は合理的な開発環境』であると言えます。

私は思ったのですが、Rails のような合理的な環境を Java でも実現できないか？そう思って作ってみたのが、「MaaEngine」です。

MaaEngine は、基本的に、

- *mtable コマンドでテーブル作成及び、テーブルに合わせた DAO と Entry を作成する。*
- *ネーミングルールによって Action オブジェクトを呼び出すことができるので、別途定義条件などは、必要ない。*
- *Action,Dao,Service などのネーミングルールからの、DI 機能をサポート。*

これら３点の機能により構成されており、環境としては、Tomcat などの Servlet コンテナ上で、動作します。

ただし、基本動作環境として、１アクション間でのトランザクションしか想定しておらず、そのことから、１台のサーバ上で動作することしか考えていない作りであることから、MaaEngine は小規模 Web アプリケーション向けの環境を推奨します。

使い方

MaaEngine の使い方を説明します。

使い方の前に、以下の環境が必要です。

- Java５またはそれ以上の Version をインストール
- Tomcat５．５またはそれ以上の Version をインストール。
- MaaEngine がサポートしているデータベース(Postgre,MySQL,FireBird,Derby,HSql,H2 のどれか)をインストールする。
- インストールしたデータベースの JDBC ドライバーを用意する。

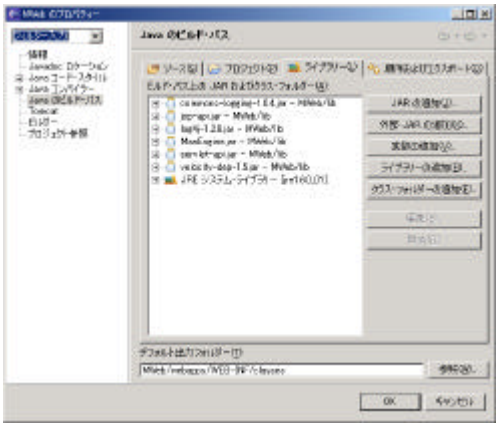
これらの環境を用意した上で、MaaEngine の開発環境を構築します。

※サンプルソースを動作させる場合、Action パッケージ=test.action、Form パッケージは、test.action.form、Dao+Entry パッケージ名=test.db としてください。

１．プロジェクトを作成（Eclipse を推奨)。

２．作成したプロジェクト以下に MaaEngine を解凍したディレクトリ以下の binary ディレクトリ以下の内容を、１で作成したプロジェクトディレクトリにコピー

３．プロジェクトに対して、２でコピーした/lib ディレクトリ以下の内容を、ライブラリ登録する(Eclipse の場合は、Project で右クリック[プロパティ]⇒[Java のビルド・パス]のライブラリで、[JAR の追加]ボタンを押して、登録)。



４．プロジェクトにおいて、Action パッケージ名、Form パッケージ名、Dao＋Entry パッケージ名を作成。(例:Action=test.action,Form=test.action.form,Dao+Entry=test.db)

５．４で作成した内容を webapp/WEB-INF/web.xml の以下の条件に登録.

```
18:<init-param>
19:     <param-name>actionPackage</param-name>
20:     <param-value>Action パッケージ名</param-value>
21:</init-param>
22:
```

```
23:<init-param>
24:     <param-name>formPackage</param-name>
25:     <param-value>Form パッケージ名</param-value>
26:</init-param>
```

6．４で作成した内容を webapp/WEB-INF/maaEngine.conf に登録。

```
33:# Entry 生成パッケージ名.
34:define_package="Dao＋Entry パッケージ名"
35:# Entry 生成ディレクトリ.
36:output_entry_java="プロジェクトソースディレクトリ名"
```

7．インストールしたデータベースの設定を行う。webapp/WEB-INF/maaEngine.conf に設定する。

```
06:# データベース定義.
07:[dbms]
08:adapter="データベースアダプタ名"
09:driver="JDBC ドライバー名"
10:url="JDBC 接続先 URL"
11:user="データベース接続ユーザ名"
12:passwd="データベース接続パスワード"
```

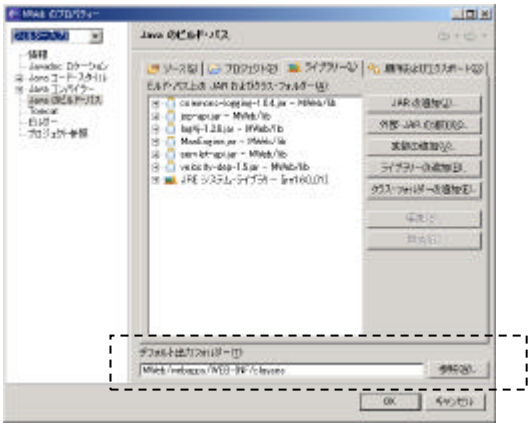
8．Tomcat に、このプロジェクトの内容を登録。

やり方として、${CATALINA_HOME}/conf/server.xml の<host>内に、以下の設定を行う。

```
<Context
  path="任意のコンテキストパス名"
  reloadable="true"
  docBase="プロジェクトのディレクトリ/webapp"
/>
```

9．用意した JDBC ドライバーを webapp/lib に配置する。

１０．クラスファイル出力先を webapp/WEB-INF/classes に変更する。



この部分をプロジェクトディレクトリ名/webapp/WEB-INF/classes に変更する。

これらの設定を行えば、基本的に mtable コマンドを実行することで、サンプルのテーブルとサンプルの DAO,Entry オブジェクトが作成されます。

Linux 系ならターミナル、Windows ならコマンドプロンプトを起動して、

<Linux の場合>

>cd プロジェクトディレクトリ名

>sh sh/mtable

<Windows の場合>

>cd プロジェクトディレクトリ名

>sh\mtable

と起動してみてください。

無事サンプルテーブル [Samples] と、Dao＋Entry パッケージ名のディレクトリ名/dao/SamplesDao.java, Dao＋Entry パッケージ名のディレクトリ名/entry/Samples.java が出来上がっているはずです。

mtable コマンド説明

mtable を起動させると、webapp/maaEngine.conf の内容を読み込みます。このとき、

```
25:[createTable]
26:
27:# Entry 出力
28:entry_out=”true/false”
29:
30:# テーブル生成ファイルディレクトリ.
31:define_directory=”./conf/table”
32:
33:# Entry 生成パッケージ名.
34:define_package=”Dao＋Entry パッケージ名”
35:
36:# Entry 生成ディレクトリ.
37:output_entry_java=”プロジェクトソースディレクトリ名”
```

の[createTable]セクション内の情報を読み込みますが、ここで重要なのは、***define_directory*** です。この定義は、指定ディレクトリ以下に存在する[*.conf]ファイル内容を全て読み込み対象とします。前項の最後に実行した時には、[samples.conf]しか存在しないので、そのファイルを読み込み対象となります。

この章では、テーブル生成 conf ファイルの仕様について、samples.conf を元に説明します。

＜samples.conf＞

```
01:# サンプルテーブル生成用テスト.
02:[Samples]
```

セクション名が、生成したいテーブル名となります。

```
03:
04:# ユーザ名.
05:name="char"
06:name="not_null"
07:name="key"
08:name="unique"
```

「カラム名=条件」で１つのカラムの内容を設定します。ここでは、[name]カラムに対して、たとえば、Postgre の場合、以下のカラム内容が発行されます。

＞name VARCHAR(255) UNIQUE NOT NULL

＞CONSTRAINT samples_pk PRIMARY KEY(name)

上記では、name カラムのタイプは、”char”ですが、他にも以下のカラムタイプがサポートされています。

項番	Mtable カラム名	Java カラムオブジェクト
1	boolean	java.lang.Boolean
2	int	java.lang.Integer
3	long	java.lang.Long
4	float	java.lang.Float
5	double	java.leng.Double
6	binary	byte[]
7	char	java.lang.String
8	text	java.lang.String
9	date	java.sql.Date
10	time	java.sql.Time
11	timestamp	java.sql.Timestamp

また、カラム以外に、上記 name では、not_null,key,unique 等が設定されていますが、全部で、以下の付加条件がサポートされています。

項番	定義名	説明
1	not_null	カラムを[not null]条件設定。
2	not null	「not_null」と同様
3	key	カラムを[key]とする。
4	unique	カラムを[unique]条件設定。
5	default	カラムのデフォルト値を設定する。記述方法としては、 columnName=”default 10”等と設定する。

mtable コマンドで生成されるテーブルには、必ずシーケンス ID が割り当てられる ID カラムが定義されます。逆に言えば、id と言う名前のカラム名は予約語なので、このコンフィグで設定した場合、エラーとなります。

また、１つの行を Insert した時や、Update したときに自動更新される予約されたカラム名が存在します。たとえば、[samples.conf]では、

```
14:# 生成日付.
15:create_time="timestamp"
16:
17:# 更新日付.
18:Update_time="timestamp"
```

のように定義することで、１つの行を追加や、更新した場合に、オートで条件に従って反映されます。ただし、ID とは違い,強制的に付加されるわけではないので、上記のように[samples]みたいに定義することで有効となります。また、上記２点のカラムタイプは、必ず[timestamp]にしてください。それ以外で定義した場合は、実行時にエラーとなります。

このような感じで、conf ファイルに定義することで、テーブル＋Dao+Entry が作成されます。また、ターゲットディレクトリ以下の*.conf ファイル名が読み込み対象となるので、１つ conf ファイル内に全てのテーブル定義を行っても良いし、１つのテーブルに対して、それぞれ１つの conf ファイルで登録するのも、自由です。

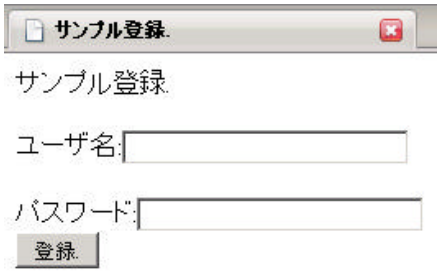
Web アプリを作ってみる

「使い方」の章で、とりあえず、最低限の動作環境ができました。この環境を使って、簡単なサンプルアプリを作ってみます。

サンプルソースは、binary/sample 以下に存在するので、その内容を作成した環境にそれぞれコピーします。

コピーした内容をコンパイルして、Tomcat を起動させた後 URL を指定、[http://localhost:(port)/(contextPath)/samples.jsp]して、Web アプリを起動します。

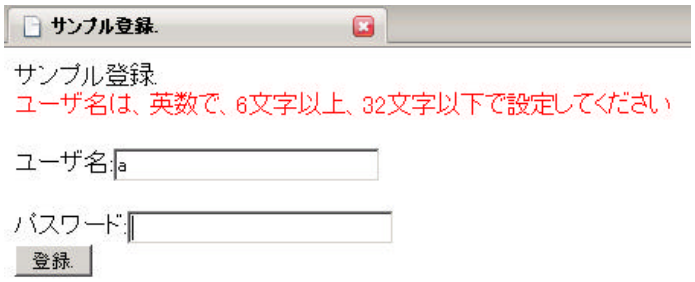
うまく動作した場合、下記のように画面がでます。



ためしに、ユーザ名、パスワードをそれぞれ入力してみます。

ユーザ名:a

パスワード:x

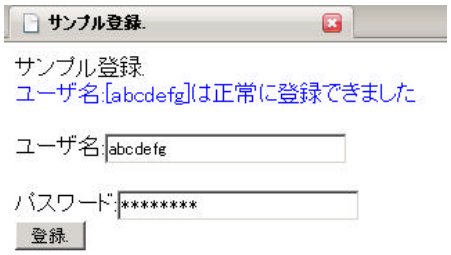


Validate によるエラーが表示されます。

今度は、Validate エラーが出ないように、任意の条件にしたがって、内容を設定します。

ユーザ名:abcdefg

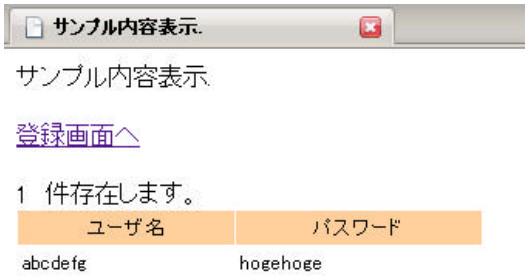
パスワード:hogehoge



これで、正常に登録できました。

次に、実際に登録されているかどうかをチェックしたいと思います。

リンク「登録一覧表示」をクリックしてみます。



先ほどの入力内容が、登録されていますね。

では、これらの実装内容を、サンプルソースを元に説明したいと思います。

＜samples.jsp＞

```
<%@ page contentType="text/html; charset=UTF8" %>
<%@ include file="/include/header.jsp"%>

<html>
<head>
<meta http-equiv="content-type" content="text/html;charset=utf-8">
<title>サンプル登録.</title>
</head>
<body>

サンプル登録.<BR>
<m:error/>
<m:message/>

<BR>
<form action="samples.do" method="post">
ユーザ名:<input name="name" size="32" type="input" value='<m:get name="name"/>'><BR>
<BR>
パスワード:<input name="passwd" size="32" type="password">
<BR>
<input value="登録." type="submit">
</form>

<BR>
<a href="samplesView.do">登録一覧表示</a>
<BR>

</body>
</html>
```

MaaEngine で作成する JSP ファイルは、全て UTF8 で作成してください。

webapp/include/には、基本ヘッダがあるので、これを加工して、利用します。

ResultMessage オブジェクトの結果内容を表示するタグ。

呼び出し対象 Action 名には、*.do で呼び出せます。また、ActionServlet. plus と、servlet-mapping を変更することで、別の拡張子でも利用することが可能です。

前回のページでリクエストされた内容を表示。

登録内容一覧表示用 Action

また、呼び出し対象 Action は、

(ActionServlet.actionPackage)+(servletPath から、拡張子を取った内容)+”Action”名のクラスファイルが実行対象となるので、このサンプルの POST 先は、test.action.SamplesAction となります。

＜test.action.SamplesAction.java＞

```
package test.action;

import org.maachang.engine.Action;
import org.maachang.engine.servlet.Parameter;
import org.maachang.engine.servlet.ResultMessage;
import org.maachang.engine.servlet.Validate;

import test.action.form.SamplesForm;
import test.db.dao.SamplesDao;

public class SamplesAction extends Action<SamplesForm> {

    private SamplesDao samplesDao = null ;
    public void setSamplesDao( SamplesDao samplesDao ) {
        this.samplesDao = samplesDao ;
    }

    public SamplesAction() {
        super( true ) ;// validate を有効に設定
    }

    @Override
    public String execution(ResultMessage result, SamplesForm fobj, Parameter params)
        throws Exception {
        samplesDao.save( fobj ) ;
        result.setSuccessMessage( "ユーザ名:["+fobj.getName() + "]は正常に登録できました" ) ;
        return "samples.jsp" ;
    }
```

必ず Action オブジェクトを継承してください。

DI コンテナから、Dao 情報を
取得するための準備です。
DI コンテナは、オブジェクトの名前の
最後に Action,Dao,Service がある場合、
対象となります。

validate メソッドを有効にする場合は、かならず、
この条件を追加しなければいけません。

入力内容を DB に登録。

正常メッセージ表示タグ<m:message/>
に表示するメッセージを設定。

正常終了時の、表示用（Forward 先）JSP 名を戻す。

```
    @Override
    public String validate(Validate validate,ResultMessage result,SamplesForm fobj,Parameter parameter)
            throws Exception {
        validate.setErrorForward( "samples.jsp" ) ;
        validate.append( "text","name",
        "ユーザ名は、英数で、6 文字以上、32 文字以下で設定してください","type=ascii","min=6","max=32" ).
           append( "text","passwd","パスワードは、英数で設定してください","type=ascii","min=1","max=32" ) ;
        if( validate.isValidate() == false ) {
            if( samplesDao.count( "where name=?",fobj.getName() ) > 0 ) {
                result.setErrorMessage( "既に同一名のユーザ名が存在します" ) ;
                return validate.getErrorFroward() ;
            }
        }
        return null ;
    }
}
```

Validate エラーのフォワード先は一番初めに定義しなければいけない

Validate 処理。

Validate エラーでない場合、指定ユーザが既に存在するかをチェックしている。

戻り値が[null]の場合、Validate でエラーではない場合は、Action 実行メソッド(execution)が呼び出される。

コンストラクタで、Validate 呼び出しを有効にしている場合は、Action.execution の前に、Action.validate メソッドが呼び出されます。

また、Validate を有効にしているにも関わらず、Action.validate メソッドをオーバライトしていない場合は、実行時に Exception が発生します。

Validate 処理は、Action.validate メソッドの第一引数の Validate オブジェクトを利用して入力内容をチェックします。入力チェック方法等は、「Validate 説明」の章で説明しますので、ここでは、省略します。

Validate 処理で、入力内容が正常である場合、Action.execution 処理が実行されます。ここでは、入力内容の SamplesForm(Samples を継承)を、SamplesDao を使って保存しています。Dao には、様々なデータベース操作メソッドがあるので、これらは、MaaEngine の JavaDoc を参照してください。

また、Action.execution メソッドで、フォワード先を[null]として、Action 内で、HttpServletResponse.getOutputStream()に内容をセットしない場合は、0 バイトの内容が返され、真っ白な画面が出力されるので、注意が必要です。

＜*samplesView.jsp*＞

```
<%@ page contentType="text/html; charset=UTF8" %>
<%@ include file="/include/header.jsp"%>

<html>
<head>
<meta http-equiv="content-type" content="text/html;charset=utf-8">
<title>サンプル内容表示.</title>
</head>
<body>
サンプル内容表示.<BR>
<BR>
<a href="samples.jsp">登録画面へ</a>
<BR><BR>

<m:size name="samplesList"/>  件存在します。<BR>

<table border="0" cellpadding="3" cellspacing="1" width="30%">
<tbody><tr>
<td align="center" bgcolor="#ffcc99" nowrap="nowrap" valign="middle"><font size="2">ユーザ名</font></td>
<td align="center" bgcolor="#ffcc99" nowrap="nowrap" valign="middle"><font size="2">パスワード</font></td>
</tr>
<m:list name="samplesList">
<tr>
<td bgcolor="#ffffff" nowrap="nowrap" valign="top"><font size="2">$(name)</font></td>
<td bgcolor="#ffffff" nowrap="nowrap" valign="top"><font size="2">$(passwd)</font></td>
</tr>
</m:list>
</tbody></table>
</body>
</html>
```

MaaEngine で作成する JSP ファイルは、全て UTF8 で作成してください。

webapp/include/には、基本ヘッダがあるので、これを加工して、利用します。

登録用 JSP

表示対象リスト数を表示するタグ

リスト内容を全て表示するタグ。
このタグで囲まれた内容を件数分表示する。

リスト内の表示したい要素名

リスト内容を全て表示する閉じタグ

ここでは、m:size,m:list タグを利用して、DB から取得した登録結果(samplesList)を表示している。

＜test.action.samplesViewAction.java＞

```
package test.action;

import org.maachang.engine.Action;
import org.maachang.engine.servlet.Parameter;
import org.maachang.engine.servlet.ResultMessage;

import test.db.dao.SamplesDao;

public class SamplesViewAction extends Action<Object> {

    private SamplesDao samplesDao = null ;
    public void setSamplesDao( SamplesDao samplesDao ) {
        this.samplesDao = samplesDao ;
    }

    @Override
    public String execution(ResultMessage result, Object fobj, Parameter params)
        throws Exception {
        params.setList( "samplesList",samplesDao.find( "order by name" ) ) ;
        return "samplesView.jsp" ;
    }
}
```

必ず Action オブジェクトを継承してください。

DI コンテナから、Dao 情報を
取得するための準備です。
DI コンテナは、オブジェクトの名前の
最後に Action,Dao,Service がある場合、
対象となります。

登録されている全データを名前ソートで、
[samplesList]名で登録する。

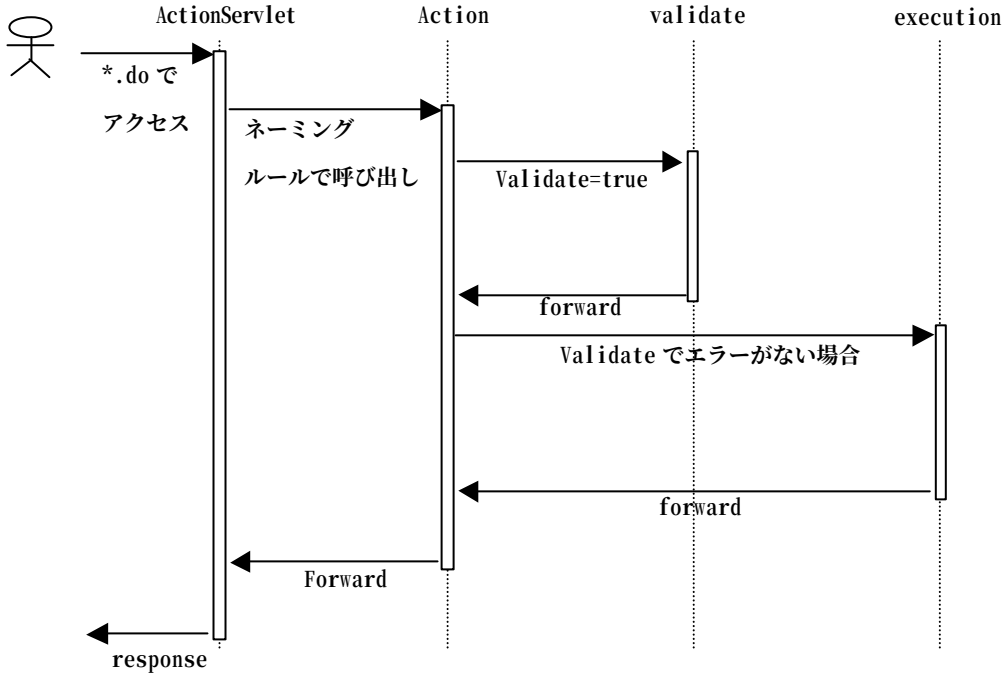
正常終了時の、表示用（Forward 先）JSP 名を戻す。

表示対象の[samplesList]の内容を samplesDao から、複数件取得して、設定しています。

Action 説明

「Web アプリを作ってみる」章で、説明した中の中核である、Action について詳しく説明したいと思います。

Action 処理は、web.xml に登録されている ActionServlet が呼び出されたときに、以下のような形で動作します。



シーケンス図っぽい書き方をすると、こんな感じの流れです。ActionServlet が呼び出された場合、ネーミングルールに則った形で、Action オブジェクトを探します。ここで検索対象となる条件は、以下の通りです。

1. web.xml の ActionServlet.actionPackage で定義されている内容。
2. １の内容と、URL の ServerPath(HttpServletRequest.getServerPath()の値)を結合し、[/]情報を[.]情報に変換した後の値と、拡張子を外して、[Action]を結合し、ファイル名となる部分に対して[name.substring(0,1).toUpperCase()+name.substring(1)]に変換した名前を Action オブジェクトとして生成しています。

例として、ActionServlet.actionPackage=test.action で、ServerPath=hoge.do の場合は、上記内容で処理した場合、[test.action.HogeAction]となります。これを、現在実行中のクラスローダから、動的にローディングしています。

なので、たとえば、ServerPath=a/b/hoge.do が階層となっているとしても、その場合は、[test.action.a.b.HogeAction]が呼び出されるようになります。

このようにして、ネーミングルールに則った形で、Action が実行されますが、これと同様に、ActionForm も呼び出されます。

たとえば、web.xml で定義されている ActionServlet.formPackage=test.action.form として、この内容を上記説明で使った[test.action.HogeAction]で割り当てたい場合は、パッケージ名[test.action.form.HogeForm]と言う名前で、定義することで、この Form オブジェクトが、Action 実行時に、動的に割り当てられます。

ただし、上記説明のような、Action と Form の関係は、絶対ではなく、たとえば、Entry を Form として割り当てたい Action があったとして、この場合、下記のように、Action コンストラクタで、Form オブジェクトの Class オブジェクトを割り当てることで、対応できます。

```
public class TestAction extends Action<Samples> {
    public TestAction() {
        super( Samples.class ) ;
    }
・・・・
}
```

ちなみに、Form オブジェクトは、POST や、GET で渡されたパラメータの内、Form オブジェクト内の Bean 形式(getter/setter)に対して、同一名の内容が割り当てられます。

次に、DI 機能について説明します。MaaEngine には単純な DI コンテナを保持しており、この機能を使って、オブジェクト名の最後が Action,Dao,Service の場合、Action で定義している setter に割り当てます。

「Web アプリを作ってみる」章の、サンプルプログラムでも利用していますが、そこでは、SamplesDao が、DI 機能を使って、Action 内に割り当てられてます。

```
private SamplesDao samplesDao = null ;
public void setSamplesDao( SamplesDao samplesDao ) {
    this.samplesDao = samplesDao ;
}
```

ただ、これら、Action,Dao,Service は、それぞれ、シングルトンとして、管理しているので、これらのオブジェクトをプログラミングする場合、少し注意が必要です。

たとえば、下記のような定義をしているサービスオブジェクトがあるとして、

```
public class HogeService {
    private String hoge = null ;
    public void setHoge( String hoge ) {
        this.hoge = hoge ;
    }
    public String getHoge() {
        return hoge ;
    }
}
```

このオブジェクト(HogeService)がシングルトンの場合、中身の(String hoge)は、シングルスレッドから見た場合、値の中身が保証されません。要は、シングルトンにおいて、シングルスレッドでしか、動作を保証しないような実装を施した場合、思わぬ障害や、セキュリティーホールの原因となるので、注意が必要です。

次にValidateについて説明をします。Validateの詳細は、「Validate説明」章で、詳しく説明しますが、ここでは、Actionに対してのvalidateメソッドの振舞いについて説明します。

初めに、validateメソッドを有効にする場合は、コンストラクタにおいて、

```
super( true )
super( true,Object.class )
```

のどちらかを利用して、Validateを許可する必要があります。

そして、validateメソッドをオーバライトして、Validate処理を実装することで、Validate処理がはじめて有効となります。

たとえば、「Webアプリを作ってみる」章で、SamplesActionオブジェクト内で、以下のように、

```
public String validate(Validate validate,ResultMessage result,SamplesForm fobj,Parameter parameter)
    throws Exception {
     validate.setErrorForward( "samples.jsp" ) ;
     validate.append( "text","name",
    "ユーザ名は、英数で、6文字以上、32文字以下で設定してください","type=ascii","min=6","max=32" ).
       append( "text","passwd","パスワードは、英数で設定してください","type=ascii","min=1","max=32" );
    ・・・・・・
```

Validateオブジェクトのappendメソッドを使って、パラメータチェックを行います。

最後に、Action オブジェクトで、提供される、メソッドについて、簡単に説明します。

・ ***String execution(ResultMessage result,T formObject,Parameter parameter)***
Action 実行処理。パラメータとして、result メッセージ出力オブジェクトと、割り当てられた formObject、HttpServletRequest.getAttribute()を I/O するパラメータオブジェクトが設定されます。

・ ***String validate(Validate validate,ResultMessage result,T formObject,Parameter parameter)***
Validate 処理。パラメータとして、validate チェックを行うためのオブジェクトと、result メッセージ出力オブジェクトと、割り当てられた formObject、HttpServletRequest.getAttribute()を I/O するパラメータオブジェクトが設定されます。
また、Validate オブジェクトを利用する場合、初めに[validate.setErrorForward]で、フォワード先を設定する必要があります。

Action オブジェクトには、Servlet からの提供内容や、データベースコネクション用オブジェクトや、MaaEngne.conf 内容を管理している内容が、以下のメソッドによって、提供されます。

・ ***HttpServletRequest request()***
・ ***HttpServletResponse response()***
・ ***Record record()***
・ ***Config getConfig()***

MaaEngine では、ページ内セッションがサポートされており、それを開始する場合、JSP にタグライブラリ<m:startPage/>や、<m:toPage/>を form タグ内に定義することで、Action.pageSession()に、その範囲でのみ有効なページ内セッションが利用できます。

・ ***PageSession pageSession()***

その他 Action メソッドについては、JavaDoc を参照してください。

Validate 説明

「Web アプリを作ってみる」章で、説明した中に、Validate 処理がありましたが、この章では、Validate 処理の詳細説明をおこないたいと思います。

基本的に Validate は、Action の Validate を有効にして、validate メソッドをオーバライトした上で、validate メソッド引数の Validate オブジェクトを利用して、

validate.append(…).append(…) … ;

のようにして、Validate 処理を実装します。

Validate.append()メソッドは、以下のような形式となっており、

append(String name,String target,String message) ;

append(String name,String target,String message,String... args) ;

第一引数が、Validate 対象名、第二引数が、チェック対象ターゲット、第三引数が、エラー時のメッセージ、第四引数からは、Validate パラメータ引数です。

Validate 対象名と、Validate パラメータは、以下の内容がサポートされています。

項番	名前	説明	引数	引数	引数
1	text	テキスト内容をチェックします。 type:num　数値だけかチェック type:alph　英字だけかチェック type:ascii　Ascii コードだけかチェック type:kana　カタカナだけかチェック type:hira　ひらがなだけかチェック min:size　最小文字数をチェック。 max;size　最大文字数をチェック。	type	min	max
2	match	正規表現でチェックします。 正規表現は Apache oro の Perl5Util を利用。	正規表現	-	-
3	date	日付フォーマットでチェックします。 日付判別は、SimpleDateFormat 形式。	format	-	-
4	ip	IPV4 形式でチェック。	-	-	-
5	url	URL 形式でチェック。	-	-	-
6	email	EMAIL 形式でチェック。	-	-	-
7	zip	郵便番号形式でチェック。	-	-	-
8	isbn	ISBN 形式でチェック。	-	-	-

引数が複数ある Validate.append()では、たとえば、text 等がそうですが、この場合は、

>validate.append(“text”,”ほげほげ”,”おかしいほげほげ”,
”type=hira”,”max=10”) ;

のように、Key=value のような形式で設定します。

また、逆に、引数が一つしかない場合では、date を例に取ると、

>validate.append(“date”,”2007/10/27”,”日付がおかしい”,”yyyy/MM/dd”) ;

のように、直接要素を設定できます。

また、「Web アプリを作ってみる」章のサンプルのように、Validate で正常であるが、データベース内に既に同一名の内容が存在する場合は、エラーとしたい場合などは、

```
if( validate.isValidate() == false ) {
    if( samplesDao.count( "where name=?",fobj.getName() ) > 0 ) {
        result.setErrorMessage( "既に同一名のユーザ名が存在します" ) ;
        return validate.getErrorFroward() ;
    }
}
```

のように、「validate 結果が正常であった場合に、こうする」みたいな処理で、対応します。

最後に、独自で Validate を提供する場合は、[org.maachang.engine.validate]パッケージの、ValidateElement インターフェイスと、AbstractValidateElement オブジェクトを継承したオブジェクトを実装し、MaaEngine.conf 内の[validate]セクションの package キーに、パッケージ名+オブジェクト名を登録することで、Validate.append()上で、利用可能になります。

また、このとき、[public String getName()]で戻す名前が、Validate.append()の第一引数で呼び出す名前になるので、他の Validate と同一名でないものを設定してください。

Dao 説明

MaaEngine には、単純な Dao(Database-Access-Object:以降は Dao)を提供しています。ただ、現在の Version(1.00)では、Join がサポートされていないので、自前で、Join 的なことを実装する必要があります。これについては、早めに対応する予定です。

Dao が提供しているメソッドは、以下の通りです。

```
public T save( T o ) ;
public void update( String set,Object... params )
public void delete( T o )
public void delete( String where,Object... params )
public List<T> find( String where,Object... params )
public List<T> find( String where,int offset,int limit,Object... params )
public T findFirst()
public T findFirst( String where,Object... params )
public List<T> findAll()
public List<T> findAll( int offset,int limit )
public int count()
public int count( String where,Object... params )
```

これらについての利用方法等は、JavaDoc を参照してください。

タグライブラリ説明

MaaEngine が提供するタグライブラリについて説明します。

現在 MaaEngine が提供しているタグライブラリは、以下の通りです。

項番	名前	説明
	out	HttpServletRequest.getAttribute()の内容を表示する。 このタグライブラリでは、格納内容を HTML で表示可能な形式に変換します。 改行、タグ表記、シングル・ダブルコーテーション等。
	get	HttpServletRequest.getAttribute()の内容を表示する。 このタグライブラリは、out とは違い、変換せずに表示する。
	error	ResultMessage.getErrorMessage()の内容を表示する。
	message	ResultMessage.getSuccessMessage()の内容を表示する。
	list	HttpServletRequest.getAttribute()の内容が List<Bean>の場合、その内容を全表示するためのタグライブラリ。 このタグで囲まれた範囲が有効となるが、その中の１件のパラメータを表示させる場合は、$(name)のように定義することで、Bean.getName()の内容が表示できる。
	size	HttpServletRequest.getAttribute()の内容が List<Bean>の場合、その件数(List<Bean>.size())を表示するタグライブラリ。
	velocity	HttpServletRequest.getAttribute()の内容を Velocity を使って表示するためのタグライブラリ。このタグで囲まれた範囲が有効となる。
	startPage	ページ内セッションを開始する場合のタグライブラリ。また、このタグライブラリは、<form>タグの中で定義しなければ、有効になりません。
	toPage	ページ内セッションを継続する場合のタグライブラリ。また、このタグライブラリは、<form>タグの中で定義しなければ、有効になりません。 このタグライブラリは、このページより前に startPage タグライブラリを定義していないと、有効になりません。

最後に

現在（Version1.00）では、MaaEngine は、本当に最低限の実装しかしていないので、広く開発者を募集しています。

また、何か質問とか、ご意見がございましたら、ms@y-ys.com にまで、メールをください。

皆様の開発が楽になれば、幸いと思い、このプロジェクトをすすめていきたいと思います。