Mind Open Source Project

MosPフレームワーク
手順書
Version 1.2.0

平成21年9月7日

株式会社マインド

目次

はじめに

当ドキュメントでは、MosP フレームワークを用いて Java による Web アプリケーションを開発する手順を、サンプルプログラムを通して紹介します。

当ドキュメントには、理解するために Web アプリケーションや Java の知識が必要になる箇所があります。

当ドキュメントでは、開発環境として OS に Windows XP、開発言語に Java（Sun Java 1.5 以上）、IDE に Eclipse、RDBMS に PostgreSQL（或いは MySQL）、サーブレットコンテナに Tomcat を利用した場合を例にして、手順を説明しています（これらの環境が整っていることを前提としています）。

1 準備

プログラムを書き始める前に、データベースや設定ファイルの準備を行います。

1.1 サンプルデータベースの準備

PostgreSQL にログインして、サンプルデータベースを作成します。

```
CREATE DATABASE sample;
\c sample;
CREATE TABLE sample (
      code character varying(4) NOT NULL,
      name character varying(16) NOT NULL
);
ALTER TABLE ONLY sample
      ADD CONSTRAINT sample_pkey PRIMARY KEY (code);
```

MySQL の場合も同様に、ログインしてサンプルデータベースを作成します。

```
CREATE DATABASE sample CHARACTER SET cp932;

GRANT insert, update, delete, select ON sample.* TO usermosp@localhost
IDENTIFIED BY 'passmosp' WITH GRANT OPTION;

USE sample;

CREATE TABLE `SAMPLE` (
  `CODE` varchar(4) NOT NULL default '',
  `NAME` varchar(16) NOT NULL default '',
  PRIMARY KEY  (`CODE`)
) DEFAULT CHARSET=cp932;
```

1.2 サンプルプロジェクトの準備

Eclipse を起動し、プロジェクトを作成します。
ここでは、Eclipse のプラグイン tomcatPlugin を利用して、Tomcat プロジェクトを作成します。
(「Window」→「設定」→「Tomcat」で Eclipse に Tomcat の設定を行っておく必要があります。)
Eclipse のメニューバーから「ファイル」→「新規」→「プロジェクト」→「Tomcat プロジェクト」と選択し(図 1-1)、プロジェクト「sample」を作成します。
その際、「コンテキスト定義の更新を可能にする」ようにしておくと(図 1-2)、Tomcat へサンプルプログラムを登録する手間が省けます。

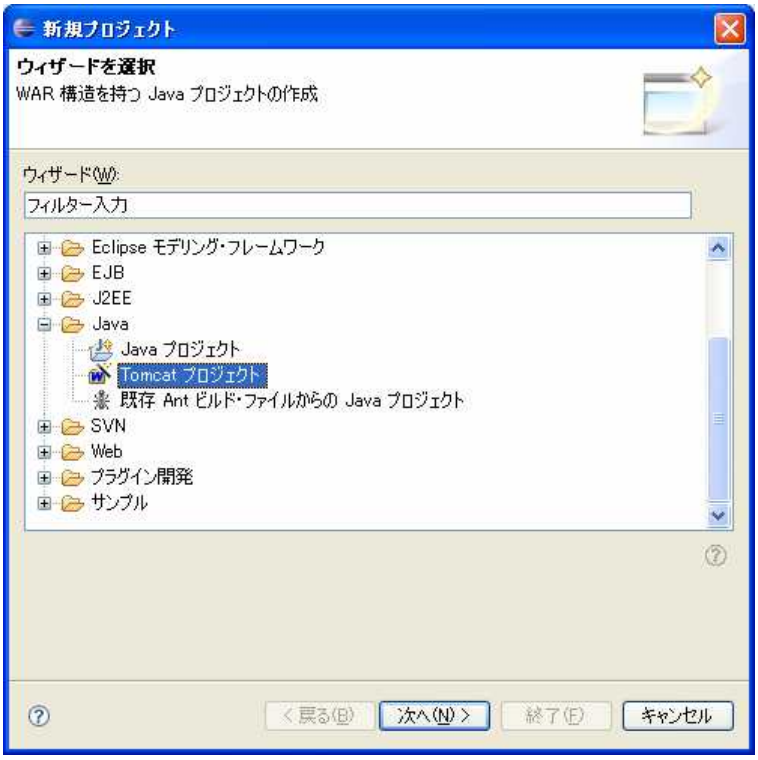


図 1-1 新規プロジェクト画面

図 1-2 新規 Tomcat プロジェクト画面

1.3 フレームワークの準備

MosP フレームワークをプロジェクトに登録します。

プロジェクトディレクトリを確認してください。Eclipse のメニューバーから「プロジェクト」→「プロパティー」で、ロケーションを確認することができます。

ダウンロードしたサンプルの中にある mosp.jar を、プロジェクトディレクトリの「/WEB-INF/lib/」に配置します。

「F5」キーを押して、パッケージの更新を行います。

Eclipse のメニューバーから「プロジェクト」→「プロパティー」→「Java のビルド・パス」と選択し、

「JAR の追加」から mosp.jar をビルド・パスに追加します（図 1-3）。

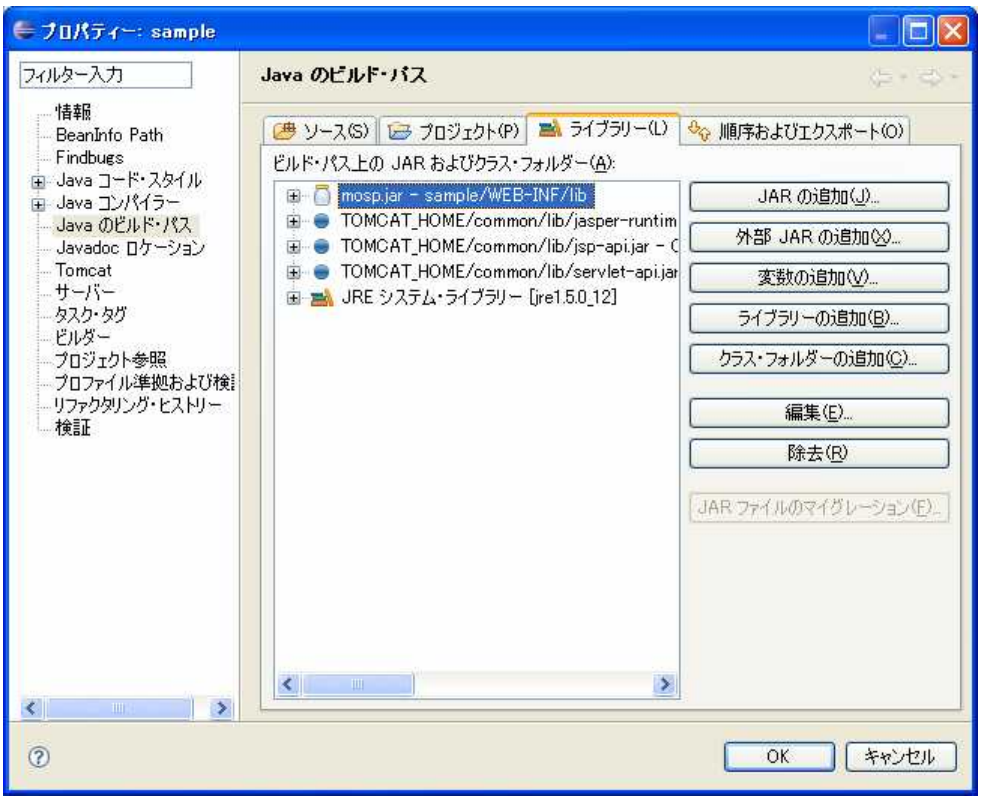


図　1-3　Java のビルド・パス画面

1.4 設定ファイル等の準備

MosP フレームワークで利用する設定ファイル等をプロジェクトに追加します。

ダウンロードしたサンプルの中から表 1-1 のファイルをプロジェクトディレクトリに配置し、「F5」キーを押して、パッケージの更新を行います。

Eclipse のパッケージエクスプローラーで、確認します（図 1-4）。

表 1-1 設定ファイル等

ファイル	説明
/WEB-INF/ mosp.properties	MosP アプリケーションに関する設定をするファイル。
/WEB-INF/ controller.properties	コマンドとアクション（後述）を紐付けるファイル。
/WEB-INF/ message.properties	MosP アプリケーションにおけるメッセージを管理するファイル。
/WEB-INF/ naming.properties	MosP アプリケーションにおける文言を管理するファイル。
/WEB-INF/ hot.properties	上記設定ファイルの読込タイミングを指定するファイル。
/WEB-INF/web.xml	Web アプリケーション配備記述子。
/pub/common/js/mosp.js	MosP フレームワークを利用するための JavaScript ファイル。
/pub/common/css/mosp.css	MosP アプリケーション共通の CSS ファイル。
/pub/common/error.html	MosP アプリケーションでエラーが起こった際に表示される HTML。
/pub/common/index.html	MosP アプリケーションにアクセスする際に用いる HTML。

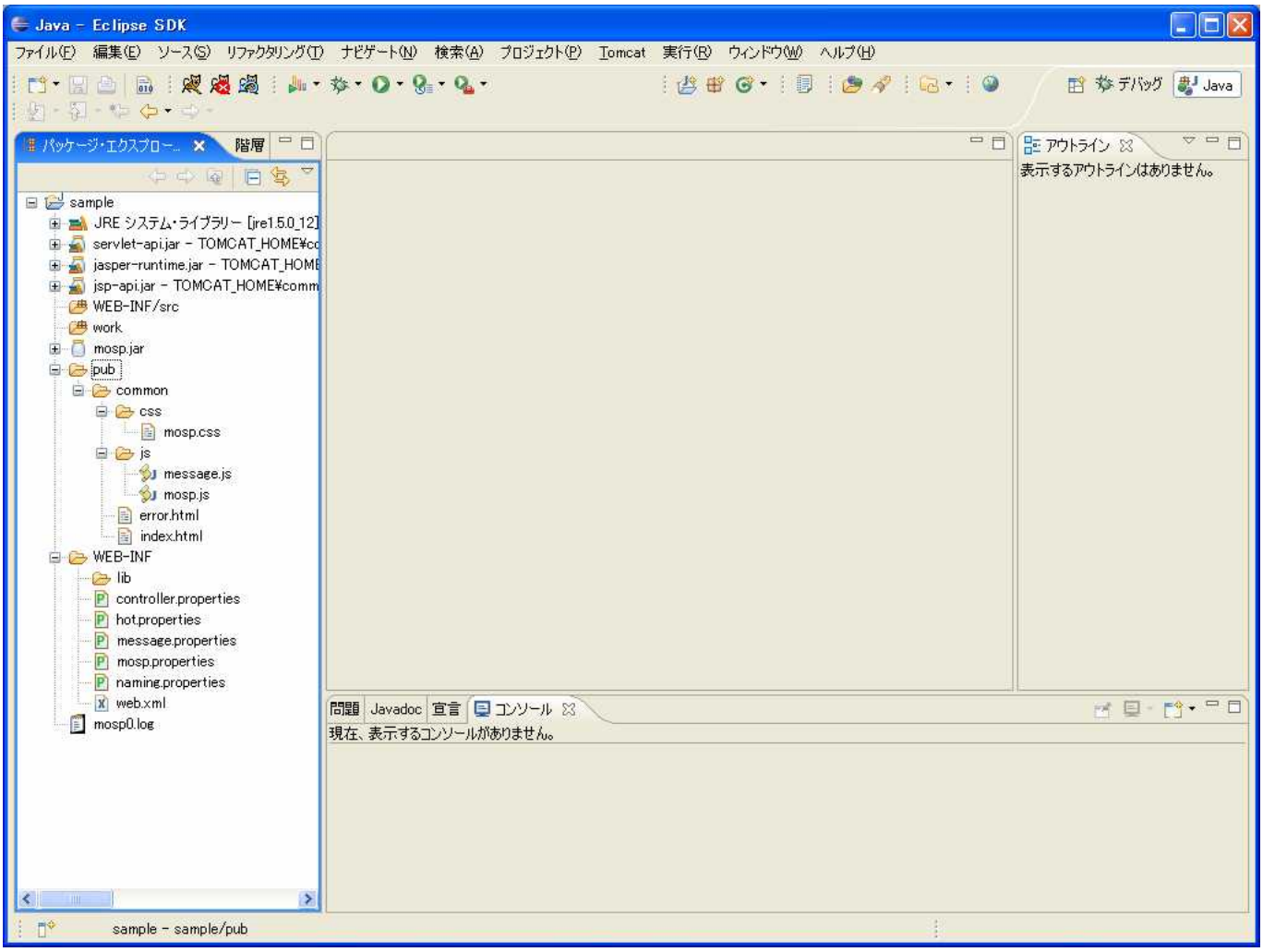


図 1-4 パッケージエクスプローラー画面

続いて、データベースへ接続するためのライブラリを追加します。
ダウンロードしたサンプルの中から「/WEB-INF/lib/postgresql-8.3-605.jdbc3.jar」及び
「/WEB-INF/lib/mysql-connector-java-5.0.4-bin.jar」をプロジェクトディレクトリに配置し、
「F5」キーを押して、パッケージの更新を行います。

データベース接続情報を確認します。
テキストエディタで「/WEB-INF/ mosp.properties」を開き、内容を確認します。

```
        <entry key="asp.dbdriver">org.postgresql.Driver</entry>
        <entry key="asp.dburl">jdbc:postgresql://localhost:5432/sample</entry>
        <entry key="asp.dbuser">postgres</entry>
        <entry key="asp.dbpass">postgres</entry>
        <!--
        <entry key="asp.dbdriver">org.gjt.mm.mysql.Driver</entry>
        <entry
key="asp.dburl">jdbc:mysql://localhost:3306/sample?useUnicode=true&amp;characterEncoding=MS932&
amp;zeroDateTimeBehavior=convertToNull</entry>
        <entry key="asp.dbuser">usermosp</entry>
        <entry key="asp.dbpass">passmosp</entry>
        -->
```

ユーザー(asp.dbuser)、パスワード(asp.dbpass)等は、接続できるものに変更してください。
MySQL の場合は、コメントアウトされている方の設定をご利用ください。

補足 1:
MosP のソースは、文字コード Shift-JIS で書かれています。
Eclipse の設定によっては、Eclipse 上で上記ファイルを編集した場合に文字化けを起こす可能性があります。
Eclipse のメニューバーから「ウィンドウ」→「設定」→「一般」→「コンテンツタイプ」で「デフォルト・エンコード」を
確認するか、メモ帳等のテキストエディターで編集してください。

補足 2:
データベース等を変更した場合、「/WEB-INF/mosp.properties」を編集し、データベースの接続先を
設定し直す必要があります。

2 登録画面の作成

MosP フレームワークを用いて、サンプルデータベースに情報を登録する画面を作ります。
また、サンプルアプリケーションの初期ページとして、アクセス時に仮ログインする機能を、実装します。

2.1 登録画面の概要

前述 1.1 サンプルデータベース準備で作成した、「SAMPLE」テーブルに対して、データを書き込みます。
既に存在するコードに対する登録要求があった場合は、更新をします。
登録画面には、表 2-1 の項目を準備します。

表 2-1 登録画面の項目

区分	項目	補足
入力	コード	コードを入力する。必須。英数字以外は入力不可。
入力	名称	名称を入力する。
ボタン	登録	入力内容を登録する。
ボタン	一覧へ	一覧画面へ遷移する。

2.2 ユーザーインターフェースの作成 1

MosP フレームワークにおいて、ユーザーインターフェース（UI）は表 2-2 のファイルで構成します。

表 2-2 ユーザーインターフェースを構成するファイル

ファイル	説明
Vo クラス(java ファイル)	ユーザーインターフェースと Java プログラムとの橋渡しをするクラス。
jsp ファイル	Vo クラスと連携し、ユーザーインターフェースの表示項目等を記載するファイル。
js ファイル	ユーザーインターフェースに動的な表現を加えるためのファイル。
css ファイル	ユーザーインターフェースのスタイルを定義するファイル。

まず、Vo クラスを作成します。
パッケージ「jp.mosp.sample.vo」を作成し、その中にクラス「SampleCardVo」を作成してください。
Eclipse のメニューバーから「ファイル」→「新規」→「パッケージ」で、パッケージを作成できます。
Eclipse 上に複数プロジェクトがある場合は、パッケージエクスプローラー上で sample プロジェクトを
右クリックすることで、同様にパッケージを作成できます。
パッケージエクスプローラーに作成したパッケージが表示されますので、そのパッケージを右クリックし、
「新規」→「クラス」からクラスを作成することができます。

以下、クラス「SampleCardVo」について説明します。
まず、MosP フレームワークにおいて、Vo クラスは「jp.mosp.common.common.BaseVo」を継承する必要があります。

```
package jp.mosp.sample.vo;
import javax.servlet.http.HttpServletRequest;
import jp.mosp.common.common.BaseVo;
import jp.mosp.common.common.MospException;
public class SampleCardVo extends BaseVo {
```

続いてクラス内にフィールド（コード、名称）を定義します。

```
	// パラメータ
	public static final String PRM_TXT_CODE = "txtCode";
	public static final String PRM_TXT_NAME = "txtName";
	// 項目長
	public static final byte LEN_CODE = 4;
	public static final byte LEN_NAME = 16;
	// 項目名
	public String NAM_CODE;
	public String NAM_NAME;
	// フィールド
	private String txtCode;
	private String txtName;
```

フィールドには、UI からの入力値が格納されます。
項目長及び項目名は、UI における表示及び入力確認に用いられます。
パラメータは、HTTP リクエストから入力値を取得する際に用います。
パラメータ、項目長、項目名は、定数名にプレフィックス（PRM_、LEN_、NAM、）を付けて定義することで、JavaScript でも利用できるようにすることができます。

コンストラクタを定義します。

```
	public SampleCardVo() {
		super();
		setViewPath(getClassName());
		needDirectJs = true;
	}
```

メソッド「setViewPath(getClassName())」は、パッケージ、クラス名から jsp、js、css ファイルパスを生成し設定します。
「needDirectJs」で、前述のパラメータ等を JavaScript でも利用できるようにするか否かを設定します。

名称等を設定するメソッドを実装します。

```
	protected void setSubTitle() {
		super.setSubTitle();
		subTitle = getName("S_CARD_TITLE");
	}
	protected void setFieldsName() {
		super.setFieldsName();
		NAM_CODE = getName("S_CODE");
		NAM_NAME = getName("S_NAME");
	}
```

「getName(String)」メソッドで、「/WEB-INF/naming.propaties」から文言を取得します。

HTTP リクエストからパラメータを取得するメソッドを実装します。

```
	public void setParams(HttpServletRequest request) {
		txtCode = request.getParameter(PRM_TXT_CODE);
		txtName = request.getParameter(PRM_TXT_NAME);
	}
```

前述のパラメータ「PRM_」を利用してパラメータを取得し、フィールドに設定します。

入力値確認メソッドを実装します。

```
	public void validate() throws MospException {
		// 確認開始時処理
		startValidation();
		// 必須確認
		checkRequired(txtCode, NAM_CODE, PRM_TXT_CODE);
		// 文字タイプ確認
		checkCode    (txtCode, NAM_CODE, PRM_TXT_CODE);
		// 文字列長確認
		checkLength  (txtCode, LEN_CODE, NAM_CODE, PRM_TXT_CODE);
		checkLength  (txtName, LEN_NAME, NAM_NAME, PRM_TXT_NAME);
		// 確認終了時処理
		endValidation();
	}
```

最後に、アクセサメソッドを実装します。

```
	public String getTxtCode() {
		return txtCode;
	}
	public String getEscTxtCode() {
		return escapeHTML(txtCode);
	}
	public void setTxtCode(String txtCode) {
		this.txtCode = txtCode;
	}
	public String getTxtName() {
		return txtName;
	}
	public String getEscTxtName() {
		return escapeHTML(txtName);
	}
	public void setTxtName(String txtName) {
		this.txtName = txtName;
	}
```

2.3 ユーザーインターフェースの作成 2

ここでは、jsp、js、css ファイルを作成します。

「/jsp/sample/sampleCard.jsp」、「/pub/sample/js/sampleCard.js」、「/pub/sample/css/sampleCard.css」を作成します (図 2-1) 。

Eclipse のメニューバーから「ファイル」→「新規」→「フォルダ」で、フォルダを作成できます。

また、「ファイル」→「新規」→「ファイル」で、ファイルを作成できます。

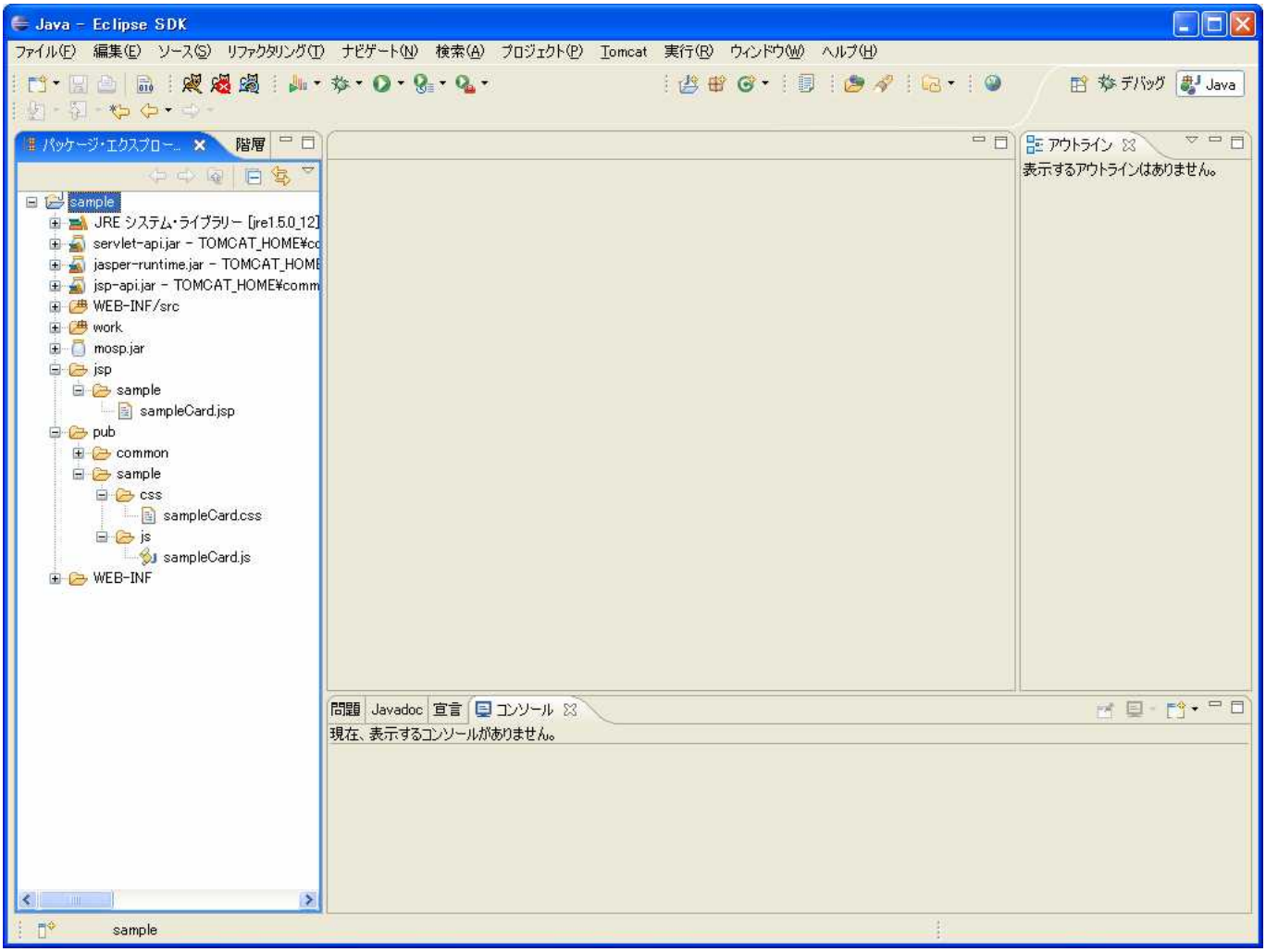


図 2-1 パッケージエクスプローラー画面

jsp ファイルには、表示項目等を記載します。

```
<%@ page
language     = "java"
pageEncoding = "Windows-31J"
%><%@ page
import = "jp.mosp.common.common.MospConst"
import = "jp.mosp.common.common.HtmlUtility"
import = "jp.mosp.sample.vo.SampleCardVo"
%><%SampleCardVo vo = (SampleCardVo)request.getAttribute(MospConst.ATT_VO);%>
<%=HtmlUtility.getHeader(vo)%>
<div class="Body">
	<div class="Header">
	<div class="SubTitleBar"><%= vo.getSubTitle() %></div></div>
	<%=HtmlUtility.getMessages(vo)%>
	<div class="Card">
		<table><tbody>
			<tr>
				<td class="TitleLabel"><%= vo.NAM_CODE +
vo.getName("S_SEPARATOR") %></td>
				<td><input type="text" class="CodeTextBox" id="<%=
SampleCardVo.PRM_TXT_CODE %>" name="<%= SampleCardVo.PRM_TXT_CODE %>" value="<%=
vo.getEscTxtCode() %>" /></td>
			</tr>
			<tr>
				<td class="TitleLabel"><%= vo.NAM_NAME +
vo.getName("S_SEPARATOR") %></td>
				<td><input type="text" class="TextBox" id="<%=
SampleCardVo.PRM_TXT_NAME %>" name="<%= SampleCardVo.PRM_TXT_NAME %>" value="<%=
vo.getEscTxtName() %>" /></td>
			</tr>
		</tbody></table>
	</div>
	<button type="button" class="ExecuteButton" onclick="submitForm(document.form)"><%=
vo.getName("S_REGIST") %></button>
</div>
```

「request.getAttribute(MospConst.ATT_VO)」で、Vo インスタンスを取得します。
以下、ここで取得した Vo に格納されている値を埋め込んでいくことになります。
ヘッダー、フッター及びメッセージ領域は、HtmlUtility クラスを利用して作成します。
項目名を Vo のフィールド「NAM_」から、input タグの id、name を Vo のフィールド「PRM_」から取得します。
また、input タグの value も Vo から取得します。
ボタンを押した際の動作を記述します。動作の詳細は、js ファイルに記述します。

続いて、js ファイルを作成します。
まず、画面読込時の追加処理を記述します。この関数は、HTML の Body ロード時に実行されます。

```
function onLoadExtra() {
	//項目背景色設定
	setFieldsBgColor(ARY_FIELDS_ID,     COLOR_FIELD_NORMAL);
	setFieldsBgColor(ARY_ERR_FIELDS_ID, COLOR_FIELD_ERROR);
	//項目長設定
	setMaxLength(PRM_TXT_CODE, LEN_CODE);
	setMaxLength(PRM_TXT_NAME, LEN_NAME);
	//フォーカス設定
	setFocus(PRM_TXT_CODE);
}
```

「PRM_」、「LEN_」は、Vo で記述した値が、そのまま利用できるようになっています。

サーバー通信時の処理を記述します。
登録ボタンを押した時にこの関数が呼ばれるように、jsp で記述しました。

```
function submitForm(objForm) {
	if (validate()) {
		doSubmitMessage(objForm, 'S1003', 'QC0001', '');
	}
}
```

入力値確認処理を記述します。

先述の「submitForm(objForm)」関数内で、利用されます。

```
function validate() {
	//フィールド背景色設定
	setFieldsBgColor(ARY_FIELDS_ID, COLOR_FIELD_NORMAL);
	//入力チェック準備
	var aryMessage = new Array();
	//入力チェック
	checkRequired(PRM_TXT_CODE, NAM_CODE, aryMessage);
	checkCode    (PRM_TXT_CODE, NAM_CODE, aryMessage);
	//チェック内容確認
	if (aryMessage.length == 0) {
		return true;
	} else {
		showMessage(aryMessage);
		return false;
	}
}
```

「NAM_」は、Vo で記述した値が、そのまま利用できるようになっています。

最後に css ファイルを作成します。

```
.Card table {
	width: 300px;
}
.Card .TitleLabel {
	width: 100px;
}
.Card .input {
	width: 180px;
}
```

ここでは、入力域の幅を定義しています。

2.4 アクションクラスの作成 1

MosP フレームワークにおいて、Action クラスはビジネスロジックの実行、UI の選択等を行います。

登録画面の表示、登録等の処理を行う Action クラスを作成します。
パッケージ「jp.mosp.sample.action」を作成し、その中にクラス「SampleCardAction」を作成してください。

以下、クラス「SampleCardAction」について説明します。
まず、MosP フレームワークにおいて、Action クラスは「**jp.mosp.common.common.BaseAction**」を
継承する必要があります。

```
package jp.mosp.sample.action;
import jp.mosp.common.common.BaseAction;
import jp.mosp.common.common.BaseVo;
import jp.mosp.common.common.MospConst;
import jp.mosp.common.common.MospException;
import jp.mosp.common.common.MospUtility;
import jp.mosp.common.dto.CmAspUserDto;
import jp.mosp.common.dto.CmUserDto;
import jp.mosp.sample.dao.SampleDao;
import jp.mosp.sample.dto.SampleDto;
import jp.mosp.sample.vo.SampleCardVo;
public class SampleCardAction extends BaseAction {
```

続いて、クラス内にコマンドを定義します。

```
    public static final String CMD_SHOW   = "S1001";
    public static final String CMD_SELECT = "S1002";
    public static final String CMD_REGIST = "S1003";
```

UI からリクエストされたコマンドに従い、Action クラスが処理を実行します。
コマンドと実行 Action クラスは、「/WEB-INF/controller.properties」で関連付けられます。

コンストラクタを定義します。

```
public SampleCardAction() {
        super();
        setNeedAspUser(false);
        setNeedUser(false);
        setCheckProcSeq(false);
        setCheckAuth(false);
}
```

ここでは、Action 共通の確認設定を行っています（サンプル用に各種確認をしないようにしています）。

主に利用する Vo クラスを指定します。

```
protected BaseVo getSpecificVo() {
        return new SampleCardVo();
}
```

当メソッドは、後述の「prepareVo()」メソッド内で利用されます。

action メソッドを実装します。

```
	public void action() throws Exception {
		// 仮ログイン処理
		login();
		// VO 取得及び設定
		prepareVo();
		// コマンド毎の処理
		if (this.cmd.equals(MospConst.CMD_INDEX)) {
			// 画面表示
			show();
		} else if (this.cmd.equals(CMD_SHOW)) {
			// 画面表示
			show();
		} else if (this.cmd.equals(CMD_SELECT)) {
			// 選択処理
			select();
		} else if (this.cmd.equals(CMD_REGIST)) {
			// 登録処理
			regist();
		} else {
			throw new MospException(MospConst.EX_CMD_INVALID);
		}
	}
```

コマンドにより呼び出された Action は、共通処理の後この action メソッドを実行します。
ここでは、「prepareVo()」により Vo を取得した後、コマンド毎に処理を分岐させています。

当サンプルアプリケーションではログイン画面は設けないため、コマンド「MospConst.CMD_INDEX」
がリクエストされた場合に当 Action で仮ログインする機能を実装します。

```
	private void login() {
		aspUser = new CmAspUserDto();
		aspUser.setAspUserId(cfg.getProperty(MospConst.PPT_ASP_ASPUSER ));
		aspUser.setAspName  (cfg.getProperty(MospConst.PPT_ASP_ASPNAME ));
		aspUser.setDbDriver (cfg.getProperty(MospConst.PPT_ASP_DBDRIVER));
		aspUser.setDbUrl    (cfg.getProperty(MospConst.PPT_ASP_DBURL   ));
		aspUser.setDbUser   (cfg.getProperty(MospConst.PPT_ASP_DBUSER  ));
		aspUser.setDbPass   (cfg.getProperty(MospConst.PPT_ASP_DBPASS  ));
		session.setAttribute(MospConst.ATT_ASPUSER, aspUser);
		user = new CmUserDto();
		user.setUserId("sample");
		session.setAttribute(MospConst.ATT_USER, user);
	}
```

ここでは、「/WEB-INF/mosp.properties」からデータベース接続情報を取得するとともに、仮のユーザーを設定しています。

続いて、画面表示処理を実装します。

```
	private void show() {
		// VO 取得
		SampleCardVo vo = (SampleCardVo)getVo();
		// VO フィールド設定
		vo.setTxtCode("");
		vo.setTxtName("");
	}
```

ここでは、「prepareVo()」で取得した Vo を再取得して、コード及び名称のフィールドを空白にしています。

選択、登録処理については、定義だけしておきます。

```
	private void select() throws Exception {
	}
	private void regist() throws Exception {
	}
```

ここまで作成すると、画面を表示することができるようになります。

2.5 サンプルアプリケーションへのアクセス

ビルドして、Tomcat を起動し、Web ブラウザからサンプルプルグラムにアクセスしてみます。

Eclipse のメニューバーから「プロジェクト」→「プロジェクトのビルド」で、ビルドできます。
「自動的にビルド」するように設定されている場合、ファイルを保存すると自動でビルドされます。
Eclipse のプラグイン tomcatPlugin を利用している場合、Eclipse のメニューバーから
「Tomcat」→「Tomcat 起動」で、Tomcat を起動できます。
Web ブラウザを起動し、URL (http://localhost:8080/sample/srv/) を入力します。

図 2-2 のような画面が表示されれば、成功です。

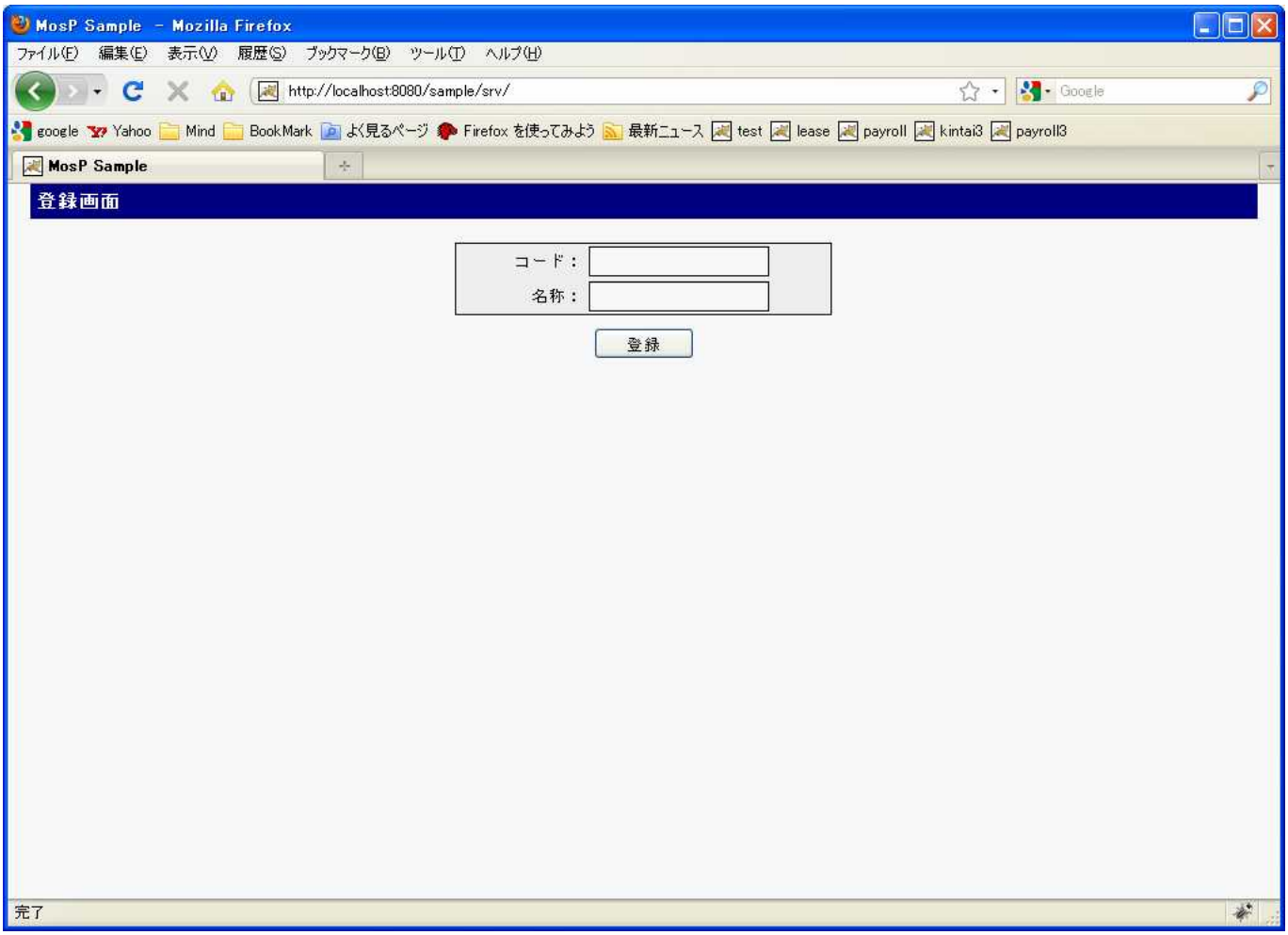


図 2-2 登録画面

2.6 データ操作クラスの作成

MosP フレームワークにおいて、データベースへのアクセスは Dao クラスを介して行われます。
また、レコードは Dto クラスを介して扱われます。

まず、SAMPLE テーブルのレコードを扱う Dto クラスを作成します。
パッケージ「jp.mosp.sample.dto」を作成し、その中にクラス「SampleDto」を作成してください。

以下、クラス「SampleDto」について説明します。
まず、「**jp.mosp.common.common.BaseDto**」を継承します。

```
package jp.mosp.sample.dto;
import jp.mosp.common.common.BaseDto;
public class SampleDto extends BaseDto {
```

続いて、クラス内にフィールドを定義します。

```
	private String code = "";
	private String name = "";
```

ここでは、単純に SAMPLE テーブル内の列を格納するフィールドを準備します。

コンストラクタ及びアクセサメソッドを定義します。

```
	public SampleDto() {
		super();
	}
	public String getCode() {return code; }
	public String getName() {return name; }
	public void setCode (String code) { this.code = code; }
	public void setName (String name) { this.name = name; }
```

Dto クラスは、基本的にテーブルの 1 レコードを格納する機能のみを実装します。

次に、SAMPLE テーブルにアクセスするための Dao クラスを作成します。
パッケージ「jp.mosp.sample.dao」を作成し、その中にクラス「SampleDao」を作成してください。

以下、クラス「SampleDao」について説明します。
MosP フレームワークにおいて、Dao クラスは「**jp.mosp.common.common.BaseDao**」を継承する必要があります。

```
package jp.mosp.sample.dao;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.Properties;
import jp.mosp.common.common.BaseDao;
import jp.mosp.common.common.MospException;
import jp.mosp.common.dto.CmAspUserDto;
import jp.mosp.common.dto.CmUserDto;
import jp.mosp.sample.dto.SampleDto;
public class SampleDao extends BaseDao {
```

続いて、クラス内に定数を定義します。

```
	// テーブル名
	public static final String TABLE = "SAMPLE";
	// キー
	public static final String KEY_1 = "CODE";
	// 列名
	public static final String COL_CODE = "CODE";
	public static final String COL_NAME = "NAME";
```

ここでは、テーブル名、プライマリーキー、列名を定義しています。
プライマリーキーには「KEY_」、列名には「COL_」を、プレフィックスとして付ける必要があります。

コンストラクタを定義します。

```
	public SampleDao(
			Properties	cfg,
			String		cmd,
			CmAspUserDto	aspUser,
			CmUserDto	user,
			Connection	conn
	) {
		super(cfg, cmd, aspUser, user, conn);
		INSERT_DATE = "";
		INSERT_USER = "";
		UPDATE_DATE = "";
		UPDATE_USER = "";
	}
```

ここでは、データベースとのコネクション等を設定しています。
「INSERT_DATE」～「UPDATE_USER」は、作成日～更新者の列名を表します。
SAMPLE テーブルではこれらは無いため、空白を設定しています。

検索結果を Dto として取得するメソッドを実装します。

```
	private SampleDto mapping() throws SQLException {
		SampleDto dto = new SampleDto();
		dto.setCode(rs.getString(COL_CODE));
		dto.setName(rs.getString(COL_NAME));
		mappingCommonInfo(dto);
		return dto;
	}
```

プライマリーキーで検索してレコードを取得するメソッドを実装します。

```
	public SampleDto findForKey (
			String code
	) throws SQLException, IllegalAccessException, NoSuchFieldException {
		try {
			index = 1;
			prepareStatement(getSelectQuery(getClass()) +
getConditionForKey(getClass()));
			setParam(index++, code);
			executeQuery();
			SampleDto dto = null;
			if (rs.next()) {
				dto = mapping();
			}
			return dto;
		} catch (SQLException e) {
			throw e;
		} finally {
			releaseResultSet();
			releasePreparedStatement();
		}
	}
```

同様に、行ロックを取得するメソッドを実装します。

```
	public SampleDto findForUpdate (
			String code
	) throws SQLException, IllegalAccessException, NoSuchFieldException {
		try {
			index = 1;
			prepareStatement(getSelectQuery(getClass()) +
getConditionForKey(getClass()) + getForUpdate());
			setParam(index++, code);
			executeQuery();
			SampleDto dto = null;
			if (rs.next()) {
				dto = mapping();
			}
			return dto;
		} catch (SQLException e) {
			throw e;
		} finally {
			releaseResultSet();
			releasePreparedStatement();
		}
	}
```

挿入するためのメソッドを作成します。

```
	public int insert(
			SampleDto dto
	) throws SQLException, IllegalAccessException, NoSuchFieldException, MospException {
		try {
			index = 1;
			prepareStatement(getInsertQuery(getClass()));
			setParams(dto, true);
			executeUpdate();
			chkInsert(1);
			return cnt;
		} catch (SQLException e) {
			throw e;
		} finally {
			releaseResultSet();
			releasePreparedStatement();
		}
	}
```

更新するためのメソッドを作成します。

```
	public int update(
			SampleDto dto
	) throws SQLException, IllegalAccessException, NoSuchFieldException, MospException {
		try {
			index = 1;
			prepareStatement(getUpdateQuery(getClass()));
			setParams(dto, false);
			setParam(index++, dto.getCode());
			executeUpdate();
			chkUpdate(1);
			return cnt;
		} catch (SQLException e) {
			throw e;
		} finally {
			releaseResultSet();
			releasePreparedStatement();
		}
	}
```

パラメータを設定するためのメソッドを作成します。

```
	private void setParams(SampleDto dto, boolean isInsert) throws SQLException {
		setParam(index++, dto.getCode());
		setParam(index++, dto.getName());
		setCommonParams(isInsert);
	}
```

2.7 アクションクラスの作成 2

アクションクラスに、登録、更新機能を実装します。

先ほど定義した SampleCardAction クラスのメソッド「regist()」に、登録処理を実装します。

```
	private void regist() throws Exception {
		// コネクション取得
		getConnection();
		// VO 取得
		SampleCardVo vo = (SampleCardVo)getVo();
		// パラメータ取得
		vo.setParams(request);
		// 入力値確認
		vo.validate();
		// データ存在確認
		SampleDao dao = new SampleDao(cfg, cmd, aspUser, user, conn);
		SampleDto dto = dao.findForUpdate(vo.getTxtCode());
		if (dto == null) {
			// 登録処理
			dto = new SampleDto();
			dto.setCode(vo.getTxtCode());
			dto.setName(vo.getTxtName());
			dao.insert(dto);
			// メッセージ設定
			setMessage(MospUtility.getMessage(msg, "IC0001", ""));
		} else {
			// 更新処理
			dto.setName(vo.getTxtName());
			dao.update(dto);
			// メッセージ設定
			setMessage(MospUtility.getMessage(msg, "IC0002", ""));
		}
		// コミット
		commit();
	}
```

ここではまず、「getConnection()」メソッドでデータベースとのコネクションを取得します。
続いて、リクエストから入力値を取得し、入力値の確認を行います。
その後、Dao、Dto を用いて登録、或いは更新処理を行い、最後にコミットします。
これで、データの登録、更新が行えるようになります。

2.8 登録画面の確認

ビルドして、Tomcat を再起動し、Web ブラウザからサンプルプルグラムにアクセスしてみます。

内容の入力後登録ボタンを押し、図 2-3 のような画面が表示されれば、成功です。

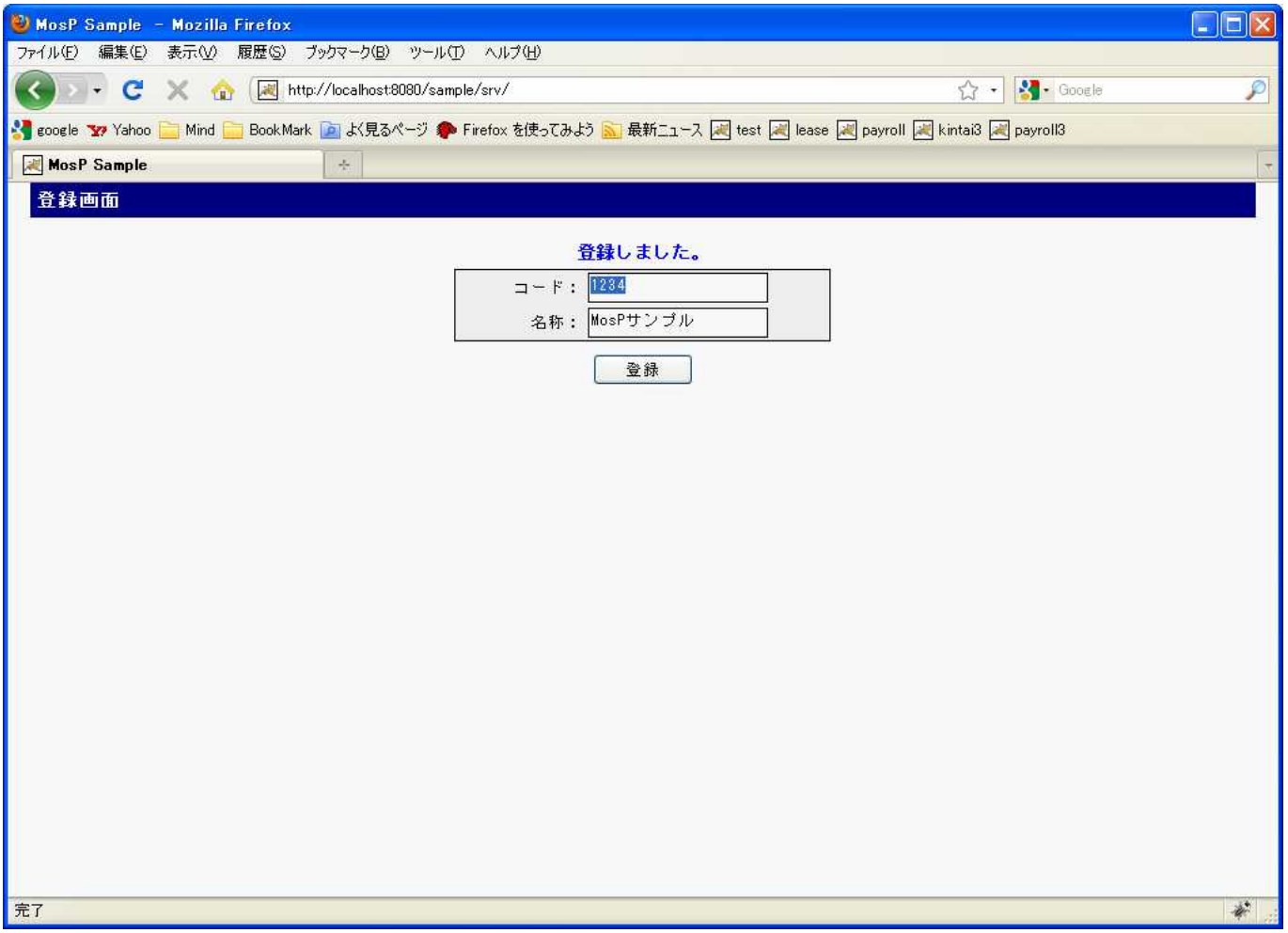


図 2-3 登録画面

3 一覧画面の作成

登録した内容を一覧表示する画面を作成します（図 3-1）。
基本的なつくりは、登録画面と同様です。詳細はサンプルプログラムソースを参照してください。

サンプルプログラムソースは MosP Developer's Community 内に有りますので、是非ご覧ください。
http://www.mosp.jp/d_user/index.html
＊MosP Developer's Community は無料で参加いただけます。

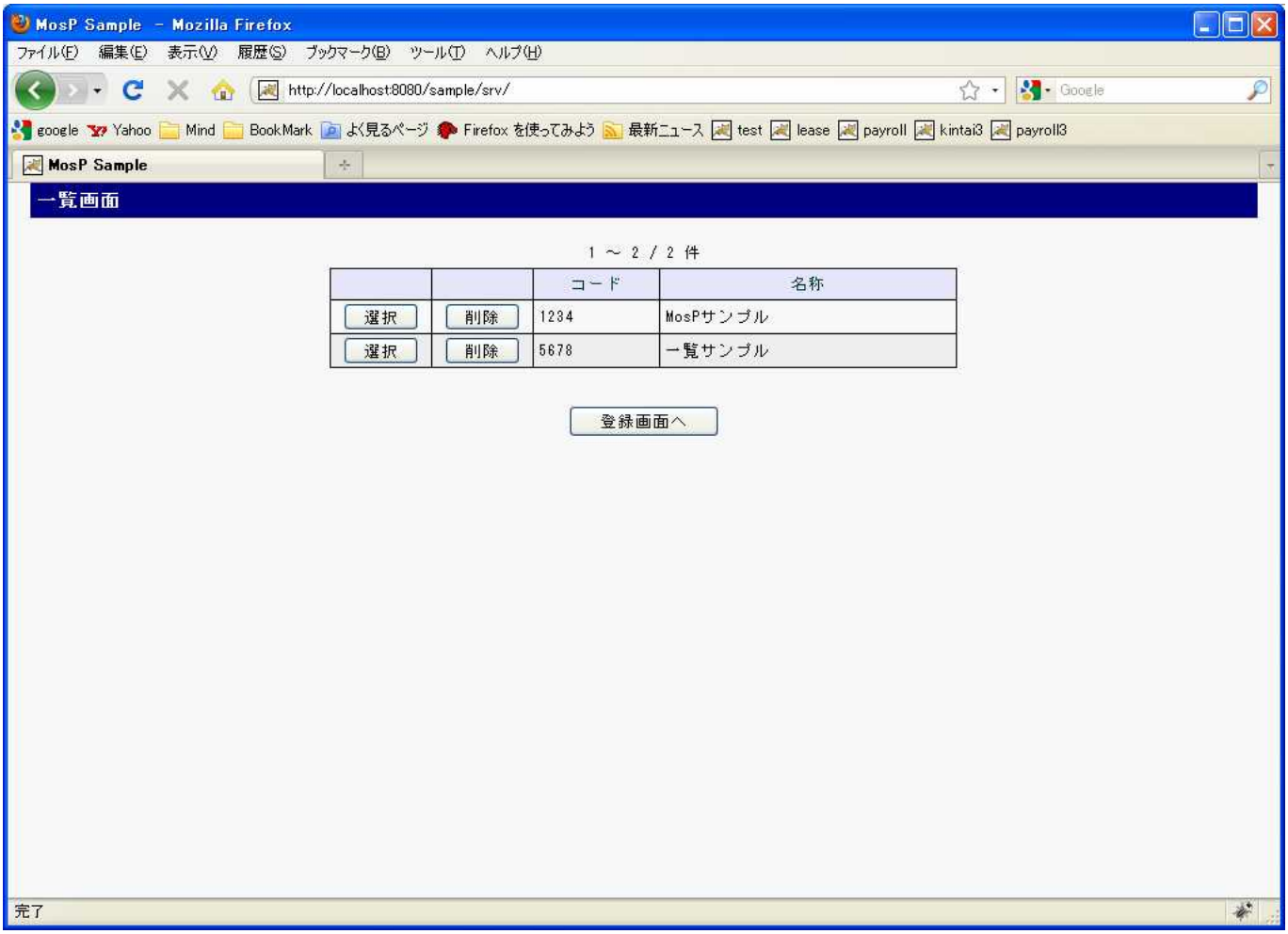
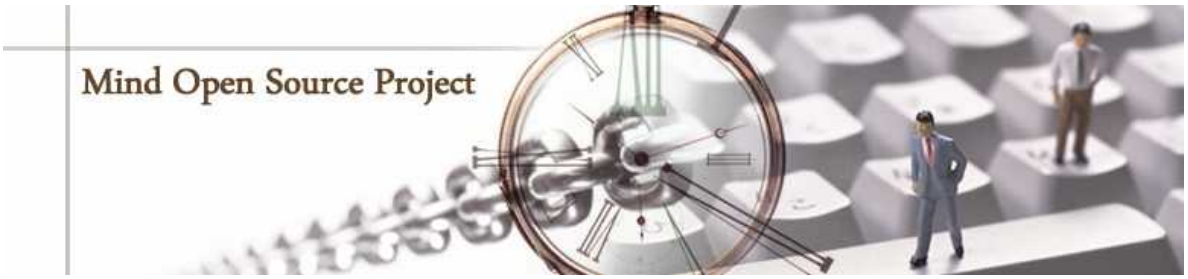


図 3-1 一覧画面



オープンソースソフトウェアに携わる世界中の開発者の皆様に感謝しております。

＝ お問い合わせ先 ＝
株式会社マインド 営業部 宛
E-Mail：sales@e-mind.co.jp
TEL：044-272-9093 FAX：044-272-9094
（受け付け時間：平日 9 時～17 時まで）
〒210-0005 神奈川県川崎市川崎区東田町６－２ ミヤダイビル 8F
URL:http://www.e-mind.co.jp