Alinous-Core 入門トレーニング
for version1.0

CROSSFIRE JAPAN INCORPORATED
http://alinous.org

INDEX

1. Alinous-Core とは

Alinous-Core は、Web とデータベースを使ったアプリケーションを作ることに特化した開発プラットフォームです。

1.1.開発の流れ

Alinous-Core には、運用のためのサーバーの他に、Eclipse プラグインで用意された開発環境があります。通常は、Eclipse プラグインで実装された開発環境でデバッグをしながら開発を行い、動作が確認できた段階で運用のためのサーバーに開発したコンテンツを移行します。Alinous-Core の開発環境には、デバッグ用の組込みサーバーが用意されているため、運用のためのサーバーは、開発中には特には必要ありません。

1.2.必要なシステム環境

Alinous-Core には、開発環境と運用環境がありますが、どちらも、Java の環境をベースとしています。

- JDK 1.5 以上
- Servlet コンテナ（運用環境のみ）
  (On using the Winstone, not necessary)
- JDK が存在する全ての OS で動作可能
  ie. Windows, Linux, AIX, FreeBSD .etc

対応しているデータベースとしては

- PostgreSQL
- MySQL
- Apache Derby

を利用することができます。

2.トレーニングの流れ

Alinous-Core には、開発環境と運用環境があります。今回のトレーニングでは、まず、最初に開発環境を学習したあとで運用環境に付いて学習するという流れになります。

2.1.開発環境のトレーニング

Alinous-Core の特徴として、開発環境が GUI デバッガを備えていると言う点があります。これは、トレーニングをする上で非常に大きなメリットがあります。実際にアプリケーションを動かしながら機能を試して学習することができるからです。ですので、トレーニングの流れとしては次のようになります。

1. Alinous-Core の開発環境のセットアップ
2. サンプルプロジェクトの作成
3. サンプルプロジェクトの実行及びデバッガの利用方法の実習
4. Alinous-Core プロジェクトの設定の概要説明
5. Alinous-Core の機能の説明
6. Alinous-Core の言語機能の詳細

ここまで学習することで、一人で Alinous-Core を利用して様々な Web とデータベースを利用したアプリケーションの開発を自由自在に行うことが出来るようになります。

2.2.運用環境のトレーニング

開発環境を使って作成したアプリケーションを実際にサーバー上で運用する際には、運用用の環境の構築が必要になります。運用環境では、パフォーマンスのチューニングなど、開発環境ではあまり関係のなかった実際に運用しているシステムならではの課題もあり、Alinous-Core の動作原理についてよく理解しておく必要があります。

3. まずは開発環境のセットアップ

まずは、開発環境のセットアップから行います。開発環境をセットアップすることで、Alinous-Core の開発を簡単に体験することができます。

3.1. Eclipse プラグインのインストール

まずは、Eclipse のプラグインをインストールします。
Eclipse プラグインのインストール手順を説明します。まず図 3.1.1.のように Eclipse のメニューにある「Help」のメニューから「Software Updates」 「Find and Install」を選択します。

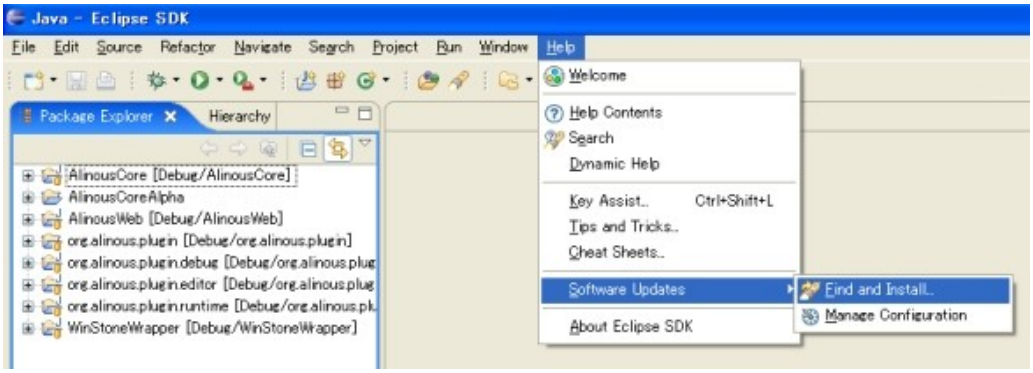

図 3.1.1. Eclipse のメニュー

次に「Search for new features to install」を選びます。

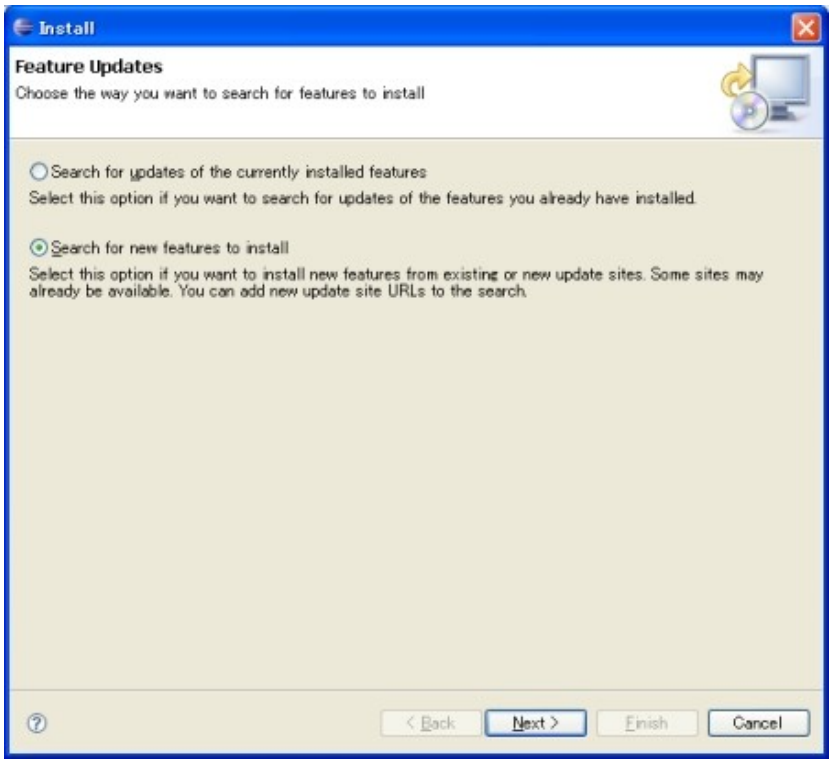

図 3.1.2. Eclipse のメニュー

すると図 3.1.3 のように、デフォルトで設定されている更新サイトの一覧が表示されます。現在、ここには Alinous-Core のための更新サイトはありませんので追加する必要があります。ダイアログの右上にある「New Remote Site」ボタンを押してください。

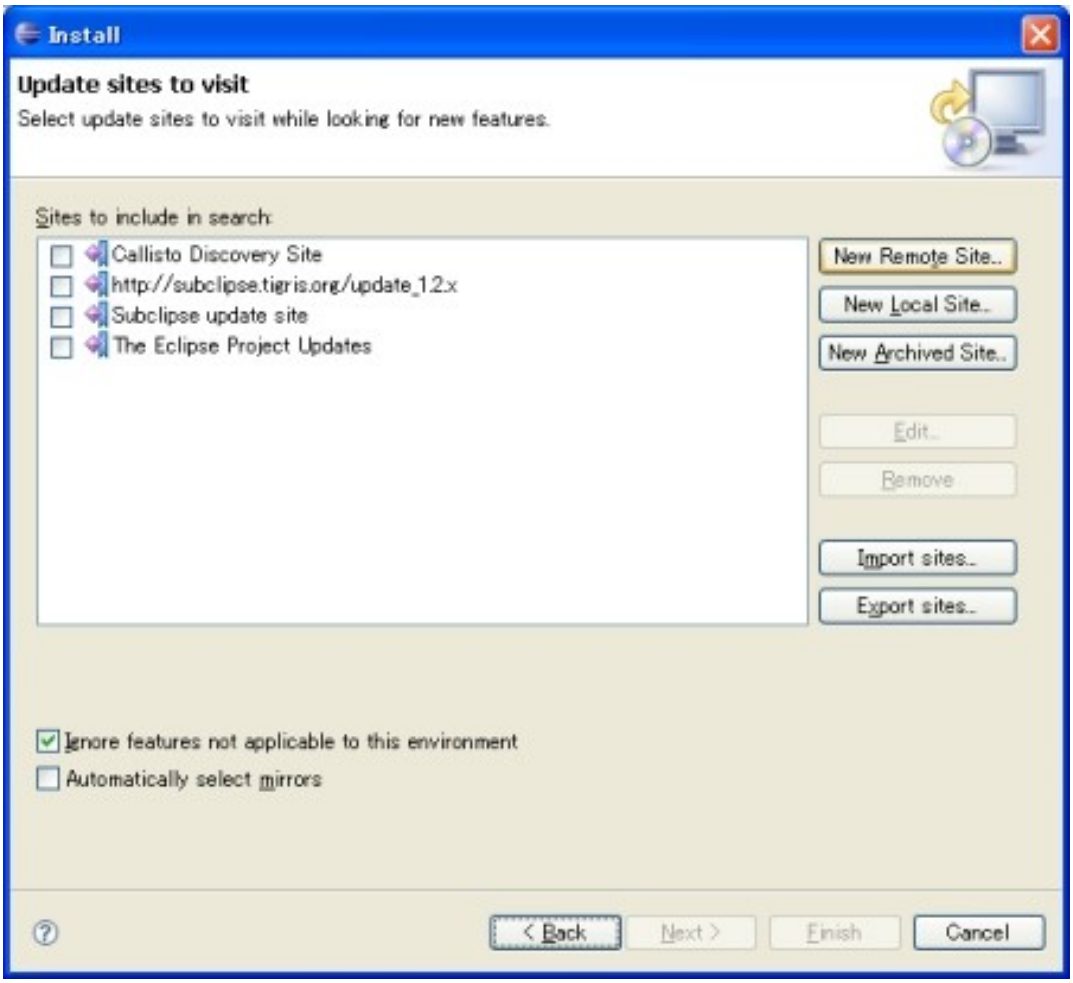

図 3.1.3. UPDATE サイト一覧のダイアログ

表示されたダイアログボックスに、図 3.1.4 にあるように更新サイト名と更新サイトアドレスを入力し、「OK」ボタンを押します。

図 3.1.4. 確認のダイアログ

選択リスト一覧の中に Alinous-Core の更新サイトが加わるので、図 2.1.5 のようにチェックボックスにチェックをいれて「Finish」ボタンを押してください。

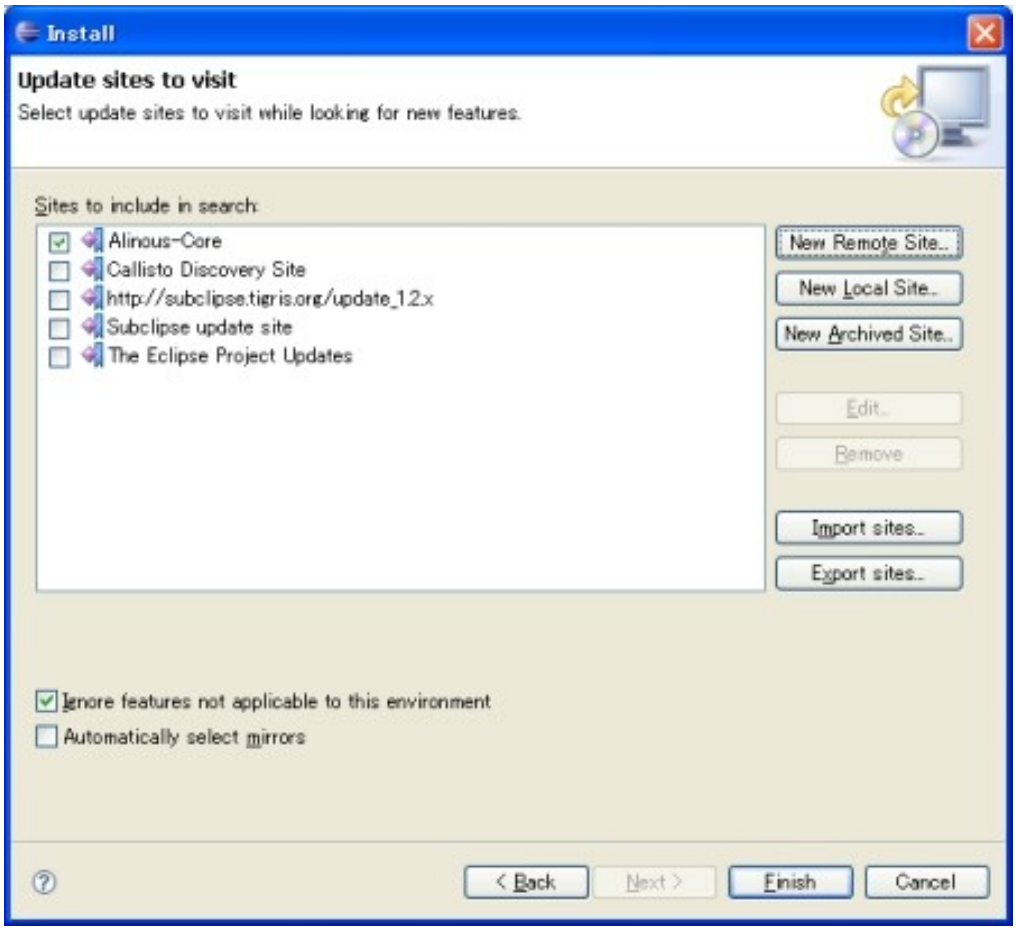

図3.1.5. 確認のダイアログ

ライセンスに関する確認に同意し、最後に実行すると、自動的にダウンロードとインストールが行われます。なお、インストール後にEclipseの再起動を行うことで、インストール作業が全て完了します。

3.2. サンプルプログラムの作成

次に、サンプルプログラムを作成します。このサンプルプログラムには、様々な Alinous-Core の機能が使われていますので、最初に Alinous-Core の機能を学習するのにものすごく役に立ちます。
まず、最初に Alinous-Core の Eclipse プラグインをインストールすること で、Alinous-Core のサンプルプロジェクトを新規プロジェクトウィザードから作成することができます。まずは、Java パースペクティブのパッケージエクスプローラー上で右クリックをし、図 3.2.1 のように「New」「Project」の順に選択します。

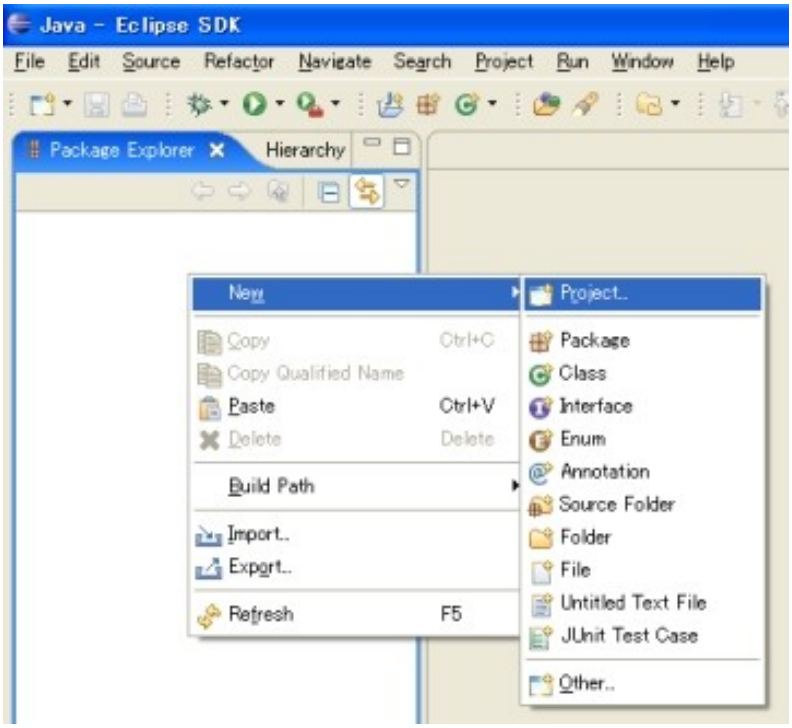

図 3.2.1. メニューから選択

図3.2.2のように新規プロジェクト作成用のウィザードが立ち上がります。
「Alinous-Core」というカテゴリーのフォルダが表示されるので、
「SampleProject」を選び、「Next」ボタンを押します。

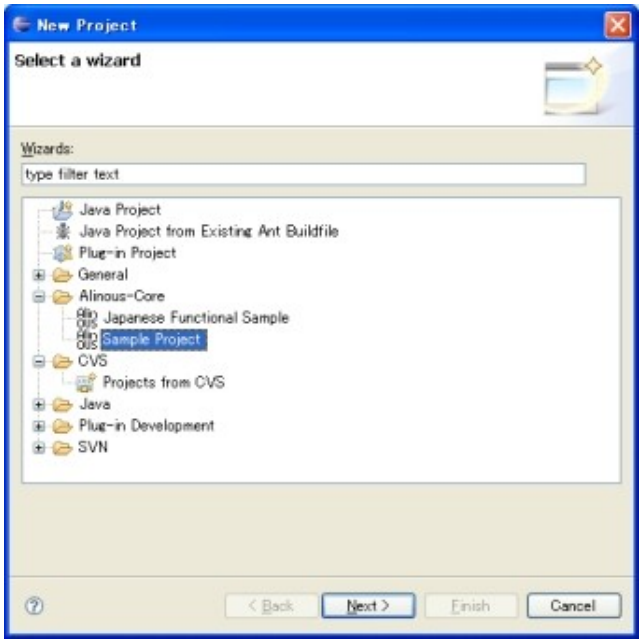

図3.2.2.プロジェクト選択ダイアログ

次に、図8のようなウィザードのページに遷移するので、プロジェクトの名前を適当に入力し、「Next」ボタンを押します。

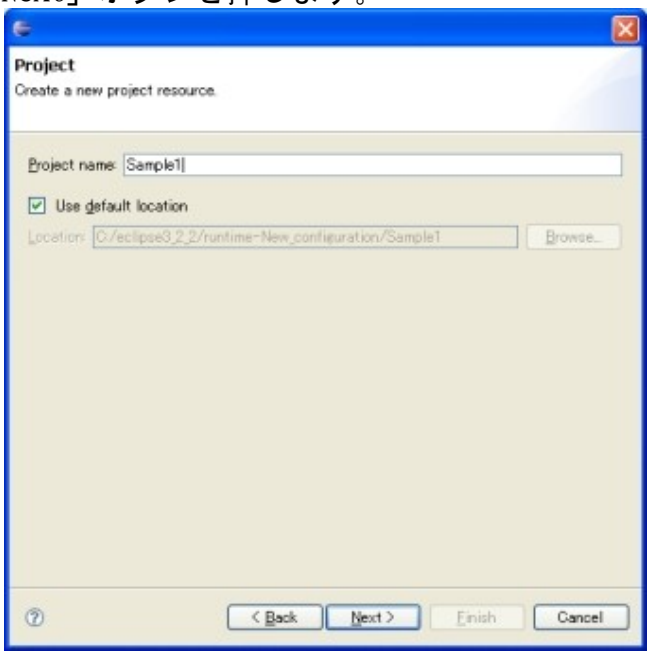

図3.2.3.プロジェクト名入力

プロジェクト名の入力後、図 3.2.4 のようなラジオボタンが表れます。ここでは、データベースとして組込み用の Apache Derby を利用するか、外部でセットアップされた PostgreSQL を使うかの選択をします。今回は、デフォルトの設定である Apache Derby をデータベースとして利用するので、このままの状態で「Finish」ボタンを押します。

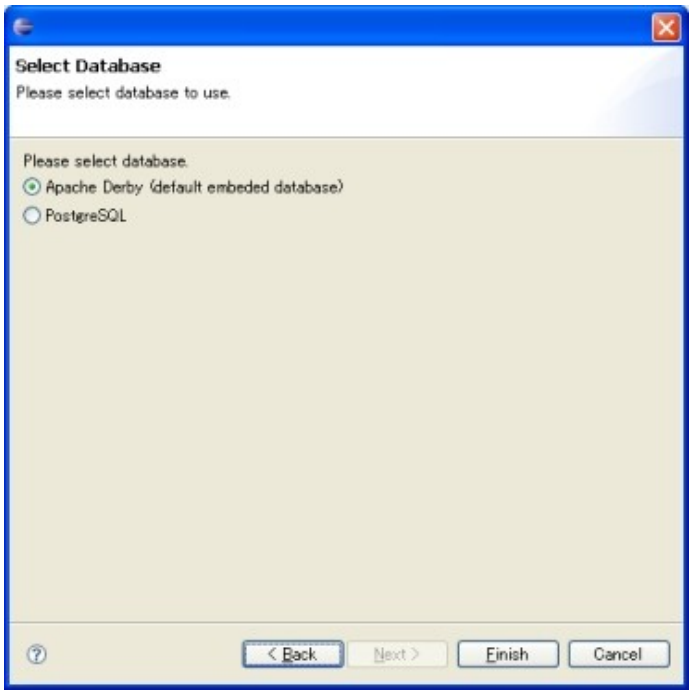

図 3.2.4.データベースの選択

プロジェクトの作成完了後には、パッケージエクスプローラー内に Alinous-Core のサンプルプロジェクトのフォルダ構成が表示されます。
以上の手続きで、実際に動かすことのできる Alinous-Core のプロ ジェクトを作成することができます。なお、このプロジェクトの中にある「alinous-config.xml」ファイルの設定や、サンプルプロジェクト が利用するアプリケーション用の初期データは、プロジェクト作成時に自動的に設定されています。

3.3. サンプルプログラムを動かす

次にサンプルプログラムを動かして、デバッガーで動作を確認してみましょう。Alinous-Core のプロジェクトを実行は、Alinous-Core のアプリケーションサーバを立ち上げ、ウェブブラウザからアクセスするという順番になります。その作業を一つ一つ解説していきます。

まずは、図 3.3.1 のように、Eclipse のメニューから「Run」　「Debug」の順で選択し、デバッグ実行ウィザードを立ち上げます。

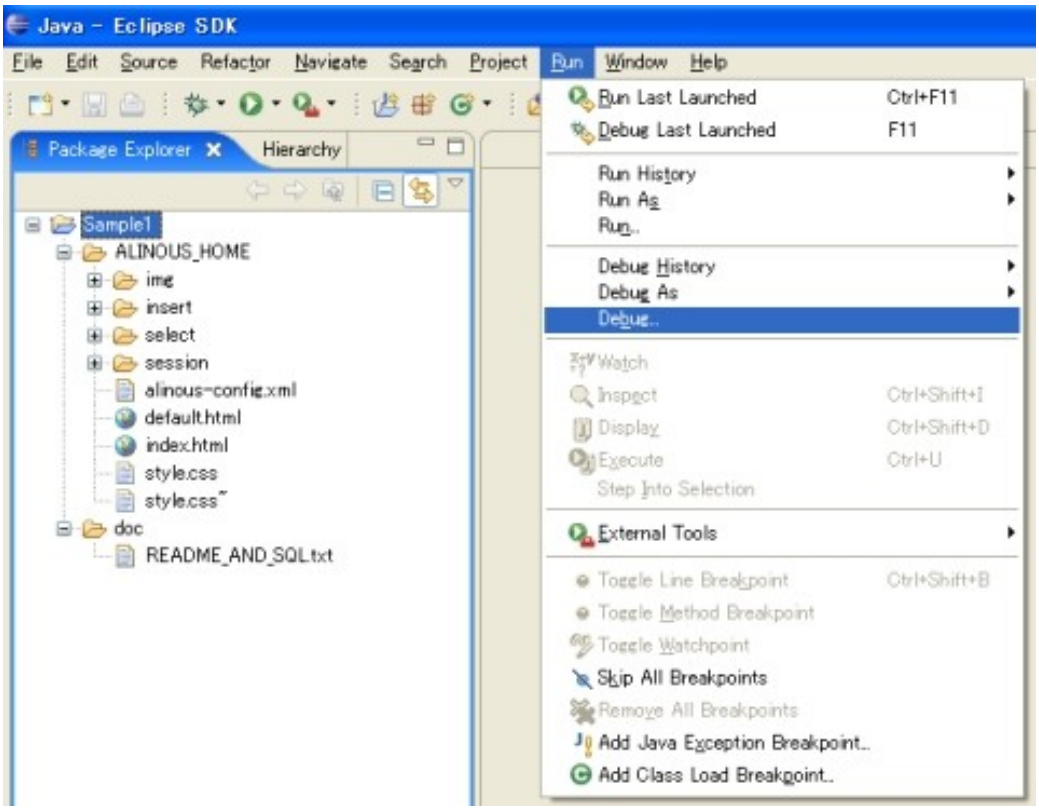

図 3.3.1. メニューからデバッグ用ウィザードの起動

デバッグウィザードが立ち上がった状態は、図3.3.2のようになっています。ここで左のリストボックスの中の一番上にある「Alinous」という項目をダブルクリックします。

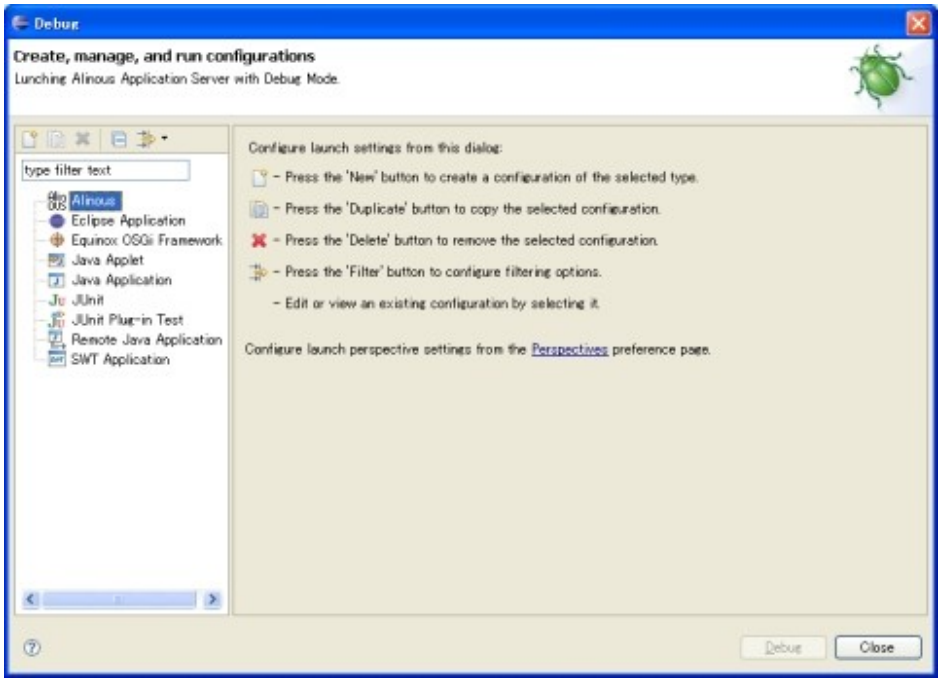

図3.3.2.デバッグ用ウィザードの起動

すると、図3.3.3のようにデバッグのコンテキスト「New_configuration」が、先ほどダブルクリックをした「Alinous」の項目の中に作られます。次に、画面の右側にある「Browse」ボタンを押してプロジェクトを選択します。

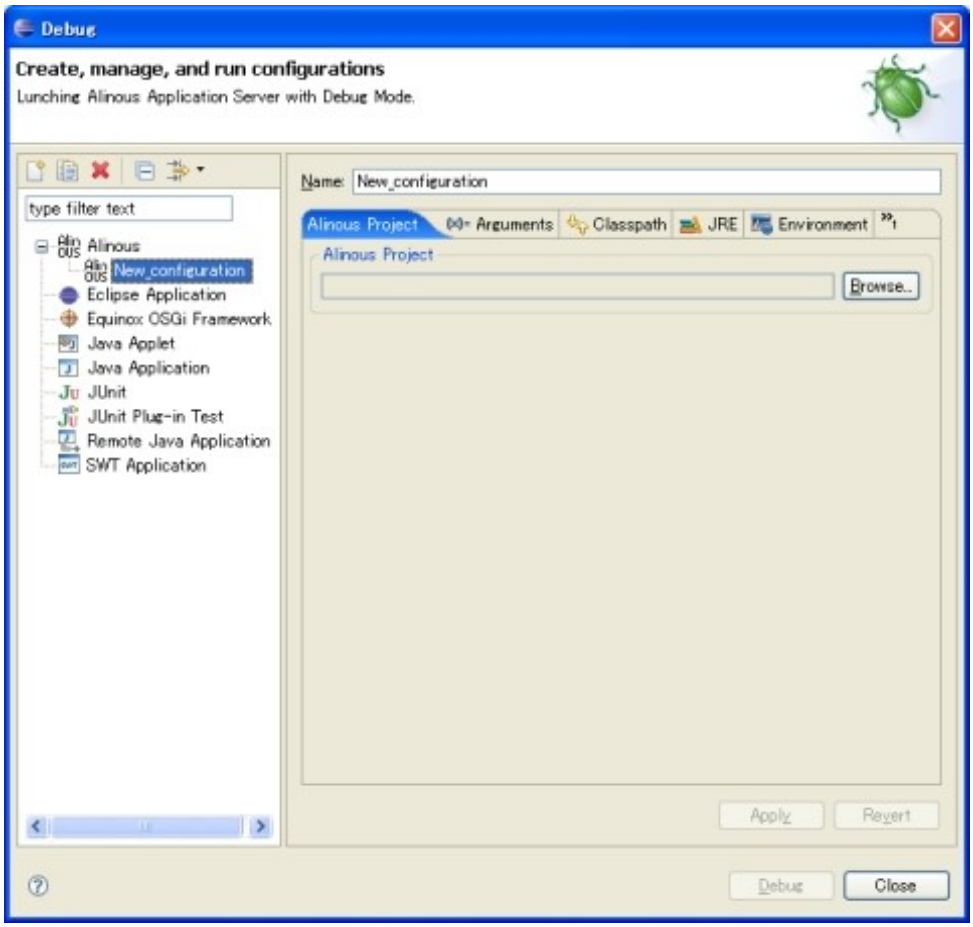

図 3.3.3. デバッグ用ウィザード

「Browse」ボタンを押すことで、図 3.3.4 のようなダイアログボックスが表示されるので、先ほど作成したプロジェクトを選択して「OK」ボタンを押します。

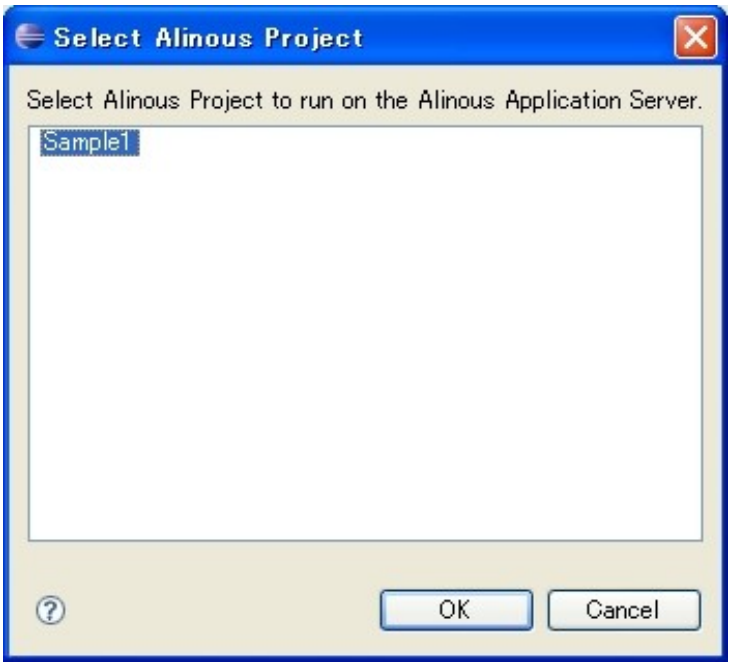

図 3.3.4.プロジェクトの選択

プロジェクト選択後、図 3.3.3 の中央の Alinous Project のところに先ほど選択したプロジェクト名が表示されるので、ウィザードの下のほうにある「Debug」ボタンを押します。このボタンを押す ことで、Alinous-Core のアプリケーションサーバと Apache Derby がネットワーク接続可能な DB として同時に立ち上がります。図 3.3.5 のように Eclipse のプロセスコンソールにログが出力されれば立ち上げ成 功です。

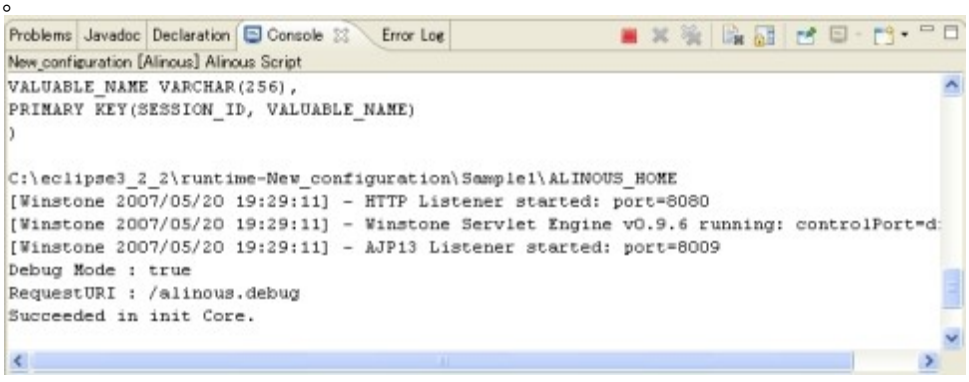

図 3.3.5.プロセスコンソール

実行を中止するときには、図 3.3.5 に表示されているプロセスコンソール上の赤い中止ボタンを押すことで、Alinous-Core アプリケーションサーバと Apache Derby のプロセスの両方を止めることができます。

それでは、Alinous-Core が動いていることを確認してみましょう。 ウェブブラウザから「http://localhost:8080/」にアクセスすることによって、図 3.3.6 のよう

に表示されます。このときに表示される内 容は、さきほど作成したサンプルプロジェクトにある「ALINOUS_HOME」ディレクトリの中の内容になります。

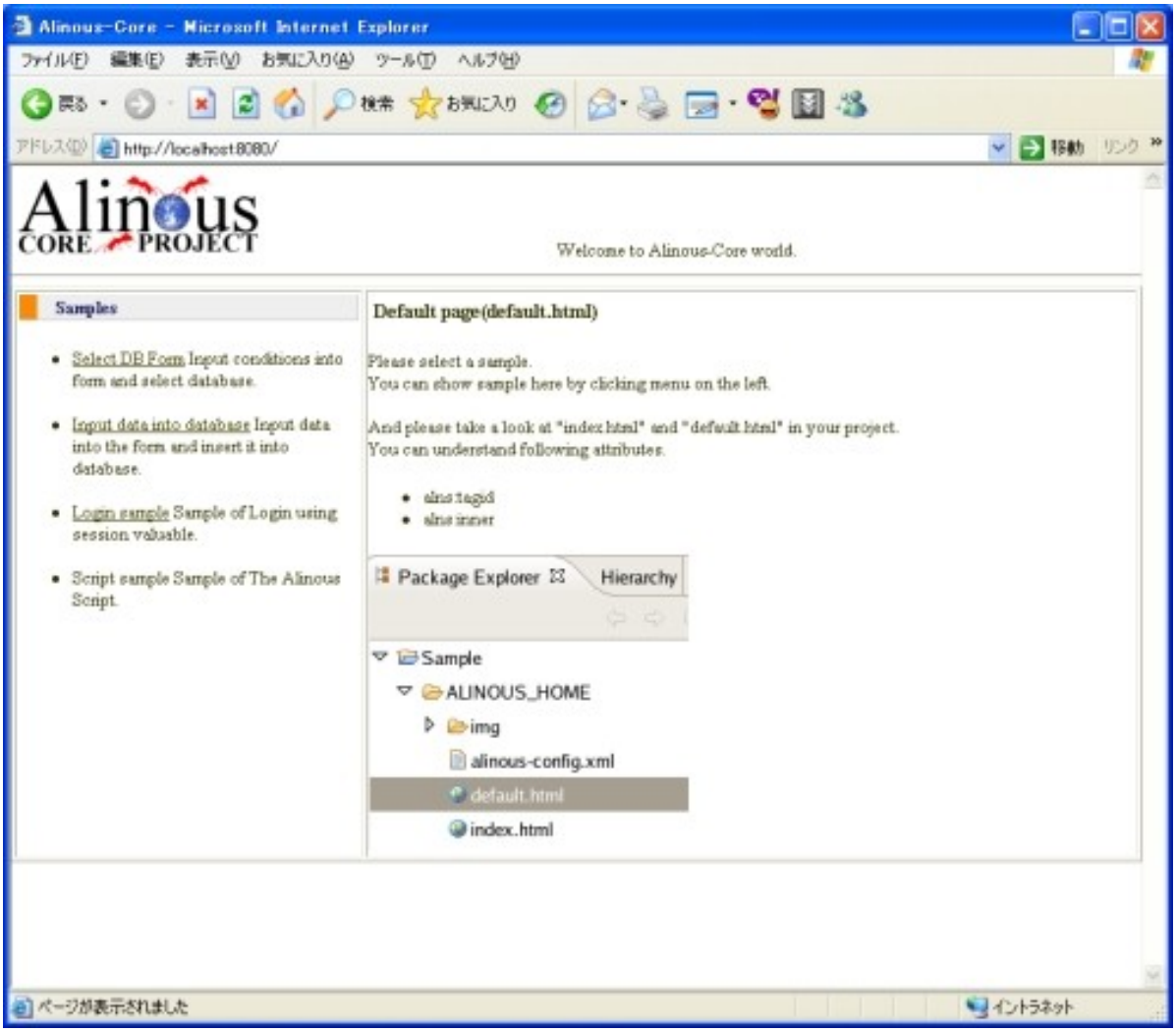

図 3.3.6.ブラウザからアクセスしたときの画面

次に、Alinous-Core の動作をデバッガーで追ってみます。まずは、「ALINOUS_HOME/select/」ディレクトリ内にある「show.alns」というファイルをダブルクリックしてみてください。図 3.3.7 のように専用エディタでファイルが開かれます。このとき、図 3.3.7 の左端に青い印が付いていると思いますが、この部分をダブルクリックするたびに、ブレイクポイ ントを ON/OFF させることができます。

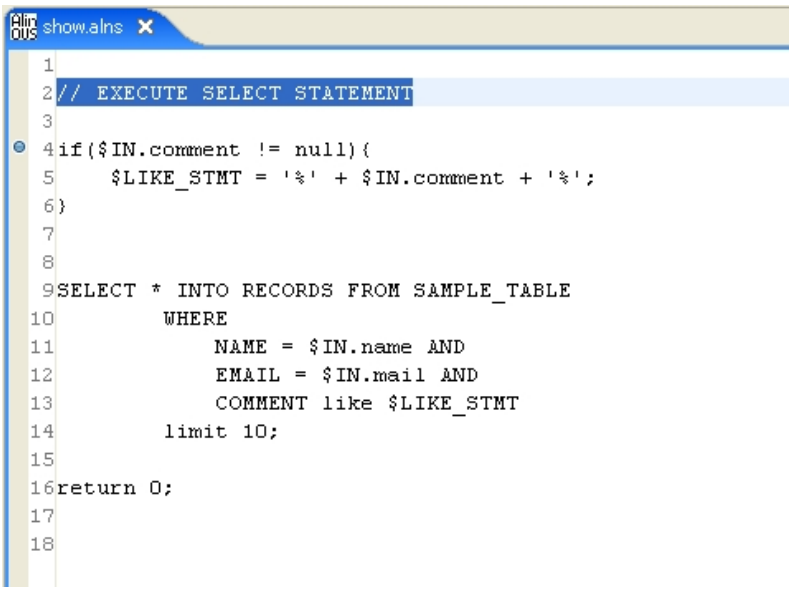

図 3.3.7.エディタとブレイクポイント

では、このブレイクポイントまで動作させてみましょう。Web ページ左 側のメニューの中の一番上にある「Select DBForm」というリンクをクリックします。フォームが表示されますが、とりあえずは何も入力せずにそのまま「Search」ボタンを押してみてくださ い。すると、図 3.3.8 のように、Eclipse のパースペクティブが Debug パースペクティブに切り替わり、さきほど設定したブレイクポイントのところで 止まりまることが確認できると思います。

このようにして、Java の場合と同様、ステップ実行しながらデバッグできることが理解できたかと思います。

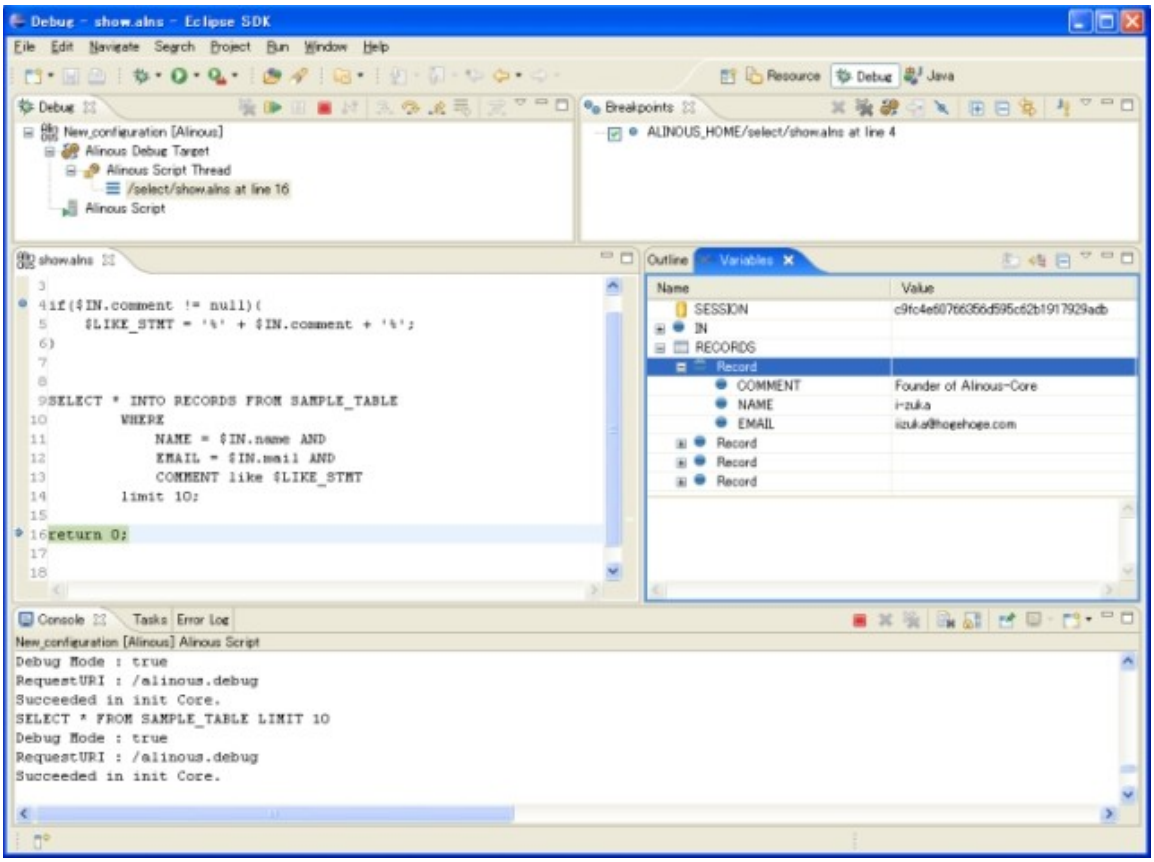

図 3.3.8.Eclipse とステップ実行

4.Alinous-Core の基本的な設定について

Alinous-Core のサンプルプログラムをさきほど動作させることが可能になりましたが、次に、Alinous-Core の動作設定について簡単に見ていくことにします。Alinous-Core の動作に関する設定は「alinous-config.xml」というファイルで行うことが出来ます。今回、セットアップした Alinous-Core のサンプルプロジェクトの「alinous-config.xml」について解説していきます。

4.1.最低限必要な設定

Alinous-Core は、データベースを利用した Web アプリケーションの開発に特化したプラットフォームです。Alinous-Core を動かすには最低、１つのデータベースのインスタンス（JDBC で接続できるデータベース）が必要です。このデータベースに対する設定は「alinous-config.xml」の中で行います。次に、「alinous-config.xml」の中で最低限必要な設定の部分を示します。

```
<?xml version="1.0" encoding="utf-8"?>
<alinous-config>
	<system>
		<system-datastore id="pgsrc" />
		<default-datastore id="pgapps" />
	</system>

	<datasources id="pgsrc" class="org.alinous.plugin.derby.DerbyDataSource">
		<connect>jdbc:derby://localhost/MyDB;create=true;</connect>
		<user></user>
		<pass></pass>
	</datasources>

	<datasources id="pgapps" class="org.alinous.plugin.derby.DerbyDataSource">
		<connect>jdbc:derby://localhost/MyDB;create=true;</connect>
		<user></user>
		<pass></pass>
	</datasources>
	  （中略）
</alinous-config>
```


4.2.利用するデータベースの設定

この設定では、システムのデータソースの指定と、２つのデータソース自体を定義しています。システムのデータソースの指定は<system>タグ内で行われています。<system-datastore>タグでは、Alinous-Core が内部的に利用するデータソースを指定しています。ここで指定されたデータソースの中には、Alinous-Core がセッション管理やポートレット機能（※後に説明します）を利用する際のデータを保存するためのテーブルを自動的に作って管理します。
<default-datastore>には、同様にしてアプリケーション用のテーブルを入れるためのデータソースを指定することが出来ます。これより、Alinous-Core では、セッション管理などのシステム用のデータベースとアプリケーション用のデータベースを個別に定義出来ることが分かります。

データソースの定義自体は、<datasources>タグで行われ、このタグの中で利用するデータベース、JDBC の URI、および DB にログインするパスワード、ユーザー ID を指定することができます。

4.3.Derby 以外のデータベースを利用する場合

今回の例では、Apache Derby をデータベースとして利用しています。Alinous-Core はこの他に、PostgreSQL と MySQL を利用することが出来ます。どの種類のデータベースを利用するかは、各<datasources>タグの class 属性で指定することができます。今回の例では”org.alinous.plugin.derby.DerbyDataSource”という Apache Derby を利用するためのデータソースクラスが指定されています。

4.3.1.PostgreSQL を利用する場合

PostgreSQL を利用する場合には、単純に class 属性に”
org.alinous.plugin.postgres.PostgreSQLDataSource”を指定します。

```
<datasources id="pgsrc" class="org.alinous.plugin.postgres.PostgreSQLDataSource">
        <connect>jdbc:postgresql://localhost:5432/alinous</connect>
        <user>postgres</user>
        <pass>password</pass>
</datasources>

<datasources id="pgapps" class="org.alinous.plugin.postgres.PostgreSQLDataSource">
        <connect>jdbc:postgresql://localhost:5432/alinous</connect>
        <user>postgres</user>
        <pass>password</pass>
</datasources>
```

4.3.2.MySQL を利用する場合

MySQL を利用する場合は、class 属性に”
org.alinous.plugin.mysql.MySQLDataSource”を指定します。

```
<datasources id="pgsrc" class="org.alinous.plugin.mysql.MySQLDataSource">
        <connect>jdbc:mysql://localhost:3306/alinous</connect>
        <user>root</user>
        <pass></pass>
</datasources>

<datasources id="pgapps" class="org.alinous.plugin.mysql.MySQLDataSource">
        <connect>jdbc:mysql://localhost:3306/alinous</connect>
        <user>root</user>
        <pass></pass>
</datasources>
```


また、MySQLを利用するためには、次の操作が必要です。

1. MySQLのJDBC Driverをダウンロードし、Alinous-Coreプロジェクトの"ALINOUS_HOME/lib/"フォルダーにコピーします。
2. もし、utf8 charsetを利用している場合には、my.cnfにキャラクターコードの設定をします。

5.Alinous-Core プログラミングの基本

では、次に、Alinous-Core のプログラミングについての基本的なことを説明していこうと思います。まだ、サンプルプロジェクトの内容を理解するのは難しいでしょうが、基本的なところから段階を追って学習していけば、他の言語などにくらべて Alinous-Core が非常にシンプルで簡単なことが分かっていただけるはずです。

5.1.ALINOUS_HOME フォルダーとは

まず、最初に覚えていただきたいことは、Alinous-Core は HTTP サーバの機能を提供しているということです。APACHE HTTP Server などは、「public_html」というドキュメントルートと呼ばれるフォルダーがあり、そのフォルダーの配下にあるファイルが HTTP の「http://localhost/」の下のアドレスにマッピングされます。

Alinous-Core の場合にも同様にして、「ALINOUS_HOME」と呼ばれるフォルダが存在し、このフォルダの配下にあるファイルが「http://localhost:8080/debug/alinous/」以下の HTTP のアドレスにマッピングされます。

例を上げると「ALINOUS_HOME/index.html」に HTTP（ブラウザ）からアクセスする場合には「http://localhost:8080/debug/alinous/index.html」と URL を指定することでアクセスすることが出来ます。

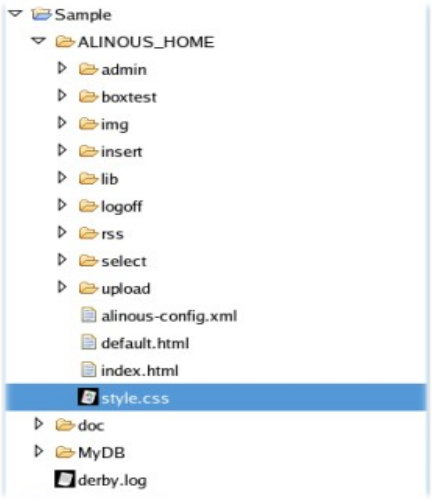

図 5.1.1.Alinous プロジェクトと ALINOUS_HOME

5.2.Alinous モジュールとは

Alinous-Core では、1 つの画面を表示するための単位を「Alinous-Core モジュール」と読んでいます。Alinous-Core モジュールは、次の要素で構成されています。

- 1 つの HTML ファイル
- Alinous スクリプトファイル（オプション）

簡単に言い替えれば、1 つの Alinous-Core モジュールは 1 枚の HTML と、それに関連するスクリプト（SQL と Alinous 独自スクリプト）で構成されます。

5.3.Alinous モジュールの例と実行されかた

select フォルダーの説明

6.Alinous-Core の基本機能

いよいよ、ここから、サンプルプロジェクトのコードを動かしながら、Alinous-Core の一つ一つの機能をみていきます。サンプルプロジェクトは次のようなフォルダ構成になっています。

図 5.1.サンプルプロジェクトのフォルダ

これらのフォルダーは、次のような内容になっています。

- ALINOUS_HOME フォルダ
  このサンプルプロジェクトのルートとなるフォルダです。
- ALINOUS_HOME/admin フォルダ
  フォーム認証のサンプルです。
- ALINOUS_HOME/img フォルダ
  各サンプルやデザインで共通の画像が入っているフォルダです。
- ALINOUS_HOME/insert フォルダ
  SQL でデータをインサートするサンプルが入っています。
- ALINOUS_HOME/lib フォルダ
  Java のライブラリである Jar ファイルを入れるためのフォルダです。ここには、Alinous-Core の標準関数のためのライブラリやシステムライブラリが入っています。
- ALINOUS_HOME/logoff フォルダ

一度ログオンしたあとで、ログオフするためのサンプルが入っています。

- ALINOUS_HOME/rss フォルダ
  rss の作成サンプルが入っています。
- ALINOUS_HOME/select フォルダ
  select 文を実行するサンプルが入っています。
- ALINOUS_HOME/upload フォルダ

6.1.ポートレット機能

まず最初に、このサンプルプロジェクトを見ていく上で最初に理解してもらう必要がある「ポートレット機能」について説明します。

6.1.1.サンプルプロジェクトのトップ画面

サンプルプロジェクトを実行するために、Alinous-Core のデバッグ用サーバを Eclipse から立ち上げ、

- http://localhost:8080/debug/alinous/

にアクセスすると次のような画面が表示されます。

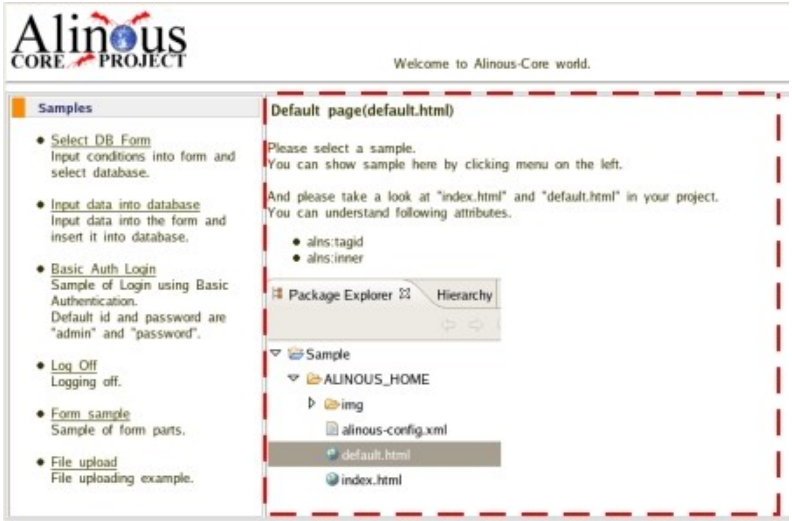

図 5.1.1.サンプルプロジェクトのトップ画面

ここで赤い破線で囲まれている部分に注目してください。この部分は、table タグ内の 1 つの td タグ（セル）にあたります。実は、この部分に他のページを埋め込

んでいます。実際に埋め込んでいるのは、「ALINOUS_HOME/default.html」です。そのため、

- http://localhost:8080/debug/alinous/default.html

にアクセスすると下図のようなページが表示されます。これは、まさに初期ページの td タグに埋め込まれていたものだということが分かると思います。

図 5.1.2.サンプルプロジェクトのトップに埋め込まれていたページ

また、サンプルプロジェクトの初期画面の左側にあるメニューにある 「Select DB Form」というリンクをクリックすると、その「default.html」が埋め込まれていた部分に別のページが入ることが分かると思います。そして、 そのページにあるフォームのボタンをクリックすると、上図の赤い破線に囲まれている td タグの領域内で画面遷移することが分かります

6.1.2.tdタグに別のページを組み込む方法

では、次にどのようにtdタグの中に別のページを組み込むかを具体的にHTMLのコードを見ながら説明していきます。まずは、「ALINOUS_HOME/index.html」の中のtdタグの部分を実際に見てみましょう。

```
<td alns:tagid="contents" alns:inner="/default.html"> </td>
```

tdタグに、alns:tagidとalns:innerというAlinous-Core用の属性が付いています。この2つの属性を利用することによって、初期状態の「index.html」のtdタグの部分に「/default.html」を埋め込むことができます。alns:tagidはtdタグの領域に名前を付けるために利用します。alns:innerは初期状態で埋め込まれるファイルを指定します。

では、初期画面の左側の「Select DB Form」というリンクをクリックしたときの動作はどのように実現されているのでしょうか？ 実際にそのリンクの部分のHTMLのコードを見てみます。

```
<a href="/select/" alns:target="contents">Select DB Form</a>
```

aタグにalns:targetという属性が付いています。この属性を付けることによって、ページ全体を遷移させるのではなく、名前を付けたtdタグの領域のみ遷移させることができます。また、このalns:targetにはthisという特殊な値を入れることができます。この設定値は、自分がどこかのHTMLに埋め込まれている場合には、埋め込まれているtdタグの中をhrefに指定されているURLに遷移させます。自分自身がトップの場合にはHTML全体を遷移します。

この様子を示すサンプルは、「Select DB Form」のリンクをクリックした後に表示される「ALINOUS_HOME/select/index.html」にあり、フォームの下に「TEST」というリンクがあります。ソースは次のようになっています

```
<a href="/" alns:target="this">TEST</a>
```


このａタグは今の自分の領域にトップページをそのまま、まるごと入れ込むことを意味しています。実際に、このリンクをクリックすると入れ子のような状態になりますが、これを複数回繰り返すと下図の ような状態になります。このような状態でもそれぞれの領域が独立した状態を持って画面遷移します。

このことは、入れ子になったページのリンクをクリックしてみると、そのクリックされた部分のページが入れ子になっていないときと同じ動きをすることで確認できます。

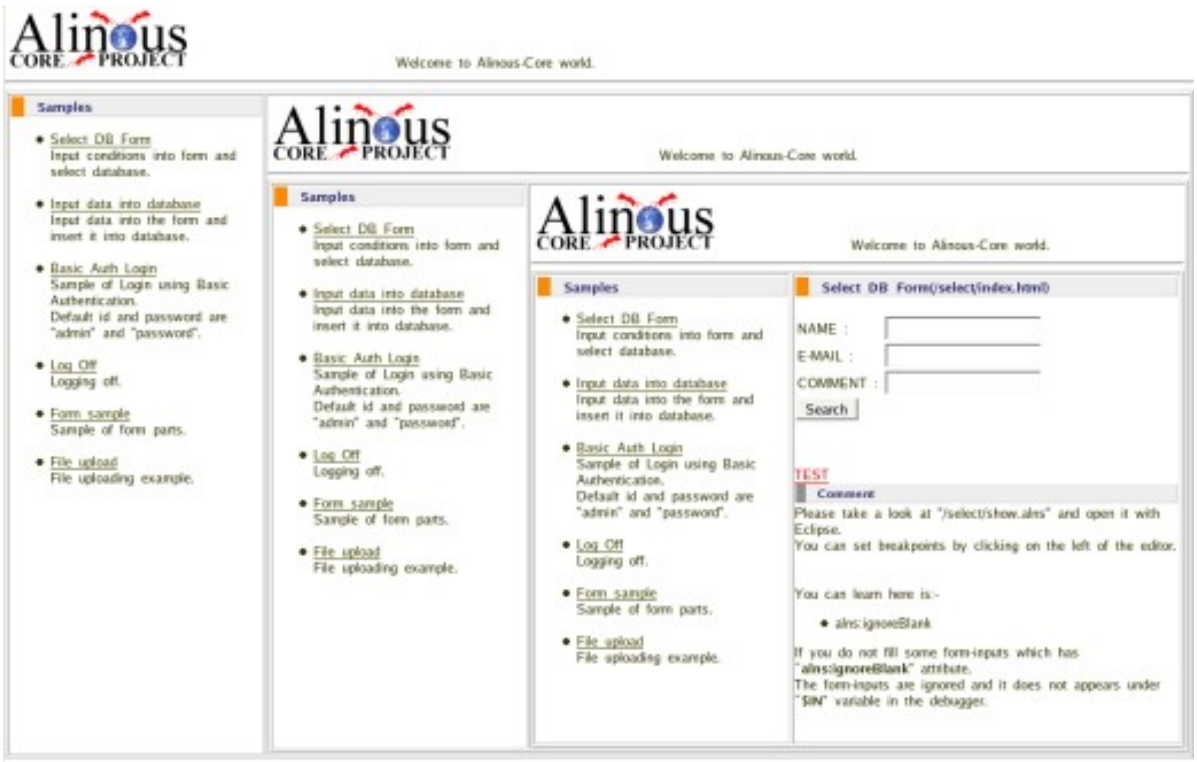

図 5.1.3. 入れ子状に画面を表示

6.1.3. span や div タグでもポートレットが可能

今回は、table タグを使ったレイアウトのときに利用される td タグで領域を指定してポートレット機能を実現しましたが、他に同じようなことを span タグおよび div タグで行うことが出来ます。スタイルシートを利用したレイアウトを行うときには div タグを多用しますが、そのような場合でもポートレット機能を利用することが出来ます。

6.2.フォーム関連機能

フォーム関連機能は、WebDB アプリケーションを作成する上で基本となる機能です。Alinous-Core では、フォームに対してカスタムのバリデータをデフォルトから付けられたり、戻るボタンの自動生成ができるなど、実際の開発の状況にあわせた仕様になっています。

6.2.1.フォームの種類と名前について

Alinous-Core でフォームを使うときには、1 つだけ命名ルールがあります。それは、フォームの値が配列の場合は名前が[]で終わるというルールです。フォームの種類を見るために「http://localhost:8080/boxtest/」にアクセスして「ALINOUS_HOME/boxtest/index.html」の内容を表示してみましょう。

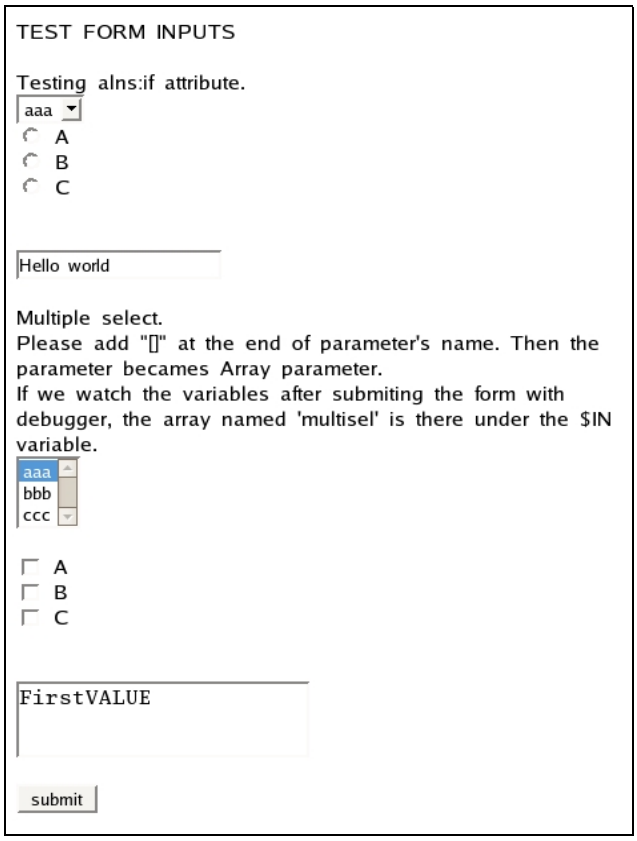

図 5.2.1.様々な種類のフォーム部品

フォームは上から、

- コンボボックス
- ラジオボタン
- テキスト
- リスト（複数選択）
- チェックボックス
- テキストエリア

になります。このうち、「リスト（複数選択）」「チェックボックス」が複数選択の対象になり、配列の値をとります。よって、これらを利用する場合にはそのインプットに[]で終わる名前を付ける必要があります。今回、例として、

- リスト（複数選択）：multisel[]
- チェックボックス：checktest[]

という名前がついています。複数選択のフォーム部品の周辺のソースコードは下のようになります。

```
<select name="multisel[]" alns:validate="custom" multiple="true">

  <option value="{$Record}"
          alns:iterate="@TEST" alns:variable="Record">
    {$Record}
  </option>

</select>

<br><br>
<input type="checkbox" name="checktest[]" value="A"
                              alns:validate="custom">A<br>
<input type="checkbox" name="checktest[]" value="B">B<br>
<input type="checkbox" name="checktest[]" value="C">C<br>
```


6.2.2.フォーム部品の動的描画

実際の開発では、フォームを動的に描画する機会もあると思いますが、Alinous-Core ではフォームの動的描画をサポートしています。先ほど示した「ALINOUS_HOME/boxtest/index.html の複数選択タグ周辺」のソースを見てください。

このソースでは、alns:iterate 属性が使われています。この属性は、HTML をレンダリングするまえに実行される AlinousScript で作成した配列を使って HTML のタグを展開したい場合に使われます。例では、@TEST という配列が AlinousScript で作られていて、その配列の個数だけ option タグが繰り返されます。実際に、配列を作っている AlinousScript は、「index.alns」というファイルになり、ソースコードは次のようになります。

```
$TEST[0] = "aaa";
$TEST[1] = "bbb";
$TEST[2] = "ccc";

return 0;
```

6.2.3.標準バリデータ

バリデータはとてもよく使われる機能で、Web フレームワークでは必須の機能といえます。Alinous-Core では、標準のバリデータとして次のものを用意しています。

名称	内容
notnull	空文字をはじくバリデータ。
int	数字フォーマット以外をはじくバリデータ。
regex	正規表現以外をはじくバリデータ。正規表現は alns:regex 属性で指定します。

標準バリデータを使っている例としては、「ALINOUS_HOME/insert/index.html」が分かりやすいと思います。ここでは notnull バリデータを利用しています。

「http://localhost:8080/debug/alinous/insert/」にアクセスして、何も入力しないまま「NEXT」ボタンを押すと、次のようにエラーが表示されて次のページに進みません。

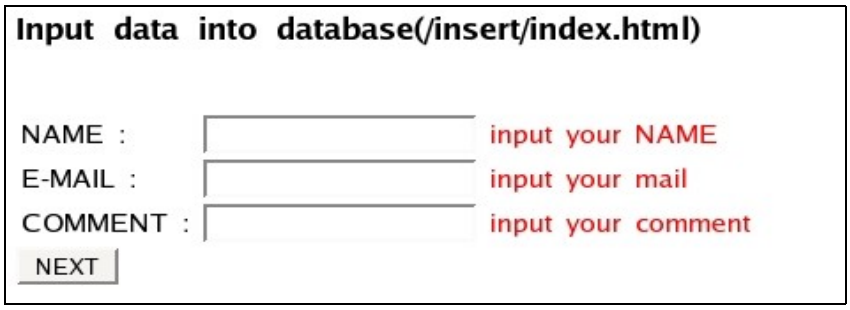

図 6.2.3.1.バリデーションとメッセージ

このときのフォーム全体のソースは次のようになります。


```
<form name="searchForm" action="confirm.html" method="POST"
      enctype="multipart/form-data">

  <table>
    <tr>
      <td>NAME : </td>
      <td>
        <input type="text" name="name" value=""
                          alns:validate="notnull">
        <span alns:msg="name" alns:form="searchForm"
                             alns:validate="notnull">
          <font color="#FF0000">input your NAME</font><br>
        </span>
      </td>
    </tr>
    <tr>
      <td>E-MAIL : </td>
      <td>
        <input type="text" name="mail" alns:validate="notnull">
        <span alns:msg="mail" alns:form="searchForm"
                             alns:validate="notnull">
          <font color="#FF0000">input your mail</font><br>
        </span>
      </td>
    </tr>
    <tr>
      <td>COMMENT : </td>
      <td>
        <input type="text" name="comment" alns:validate="notnull">
        <span alns:msg="comment" alns:form="searchForm"
                                 alns:validate="notnull">
          <font color="#FF0000">input your comment</font><br>
        </span>
      </td>
    </tr>
  </table>

  <input name="submit" type="submit" value="NEXT"/>
</form>
```

ここで、最初のテキストインプットに注目してみます。バリデータの設定仕方は、次のよ

うにHTML内で行うことができます。

```
<input type="text" name="name" value="" alns:validate="notnull">
  <span alns:msg="name" alns:form="searchForm" alns:validate="notnull">
    <font color="#FF0000">input your NAME</font><br>
  </span>
```

まずは、バリデータを設定する部分から説明します。inputタグの中にalns:validateという属性がありますが、この属性を設定することでバリデータを設定できます。notnullバリデータの場合はこれでバリデータの設定は終わりです。もし、regexバリデータを使う場合は、alns:validateにregexという値を設定し、さらにalns:regexという属性をinputタグの中に追加して正規表現を設定します。複数のバリデータを設定したい場合には、alns:validate="notnull,int"のようにカンマ区切りで複数指定できます。

つぎに、バリデーションに失敗したときに表示するメッセージを設定します。このメッセージはspanタグで設定できます。上のソースのspanタグにはalns:msg、alns:form、 alns:validateの3つの属性がありますが、これらを設定することでエラーメッセージを表示させることができます。alns:msgには、そのメッセージの対象になるinputタグの名前を指定します。alns:formには、inputタグが含まれるフォーム名を設定します。そして、alns:validateには、どのバリデータで失敗したときにこのエラーメッセージを表示するかを設定します。

以上のやり方だけで、標準バリデータを設定することができます。

6.2.4.カスタムバリデータ

Web系の開発をしていると、NULLチェックや正規表現のフォーマッ トチェックといった、データのフォーマットだけでバリデーションが完結するもの以外に、データベースの中身にアクセスしないとバリデーションができないよ うなものがあります。そのようなときもAlinous-Coreは簡単にカスタムのバリデーション処理を実装できます。カスタムバリデータの設定方法は、 標準のバリデータの設定とほぼ似ていますが、次の2点が違います。

- alns:validate属性の値にcustomを設定する
- validationのロジックを自分で実装する

では、実際にカスタムバリデータの動きを見てみましょう。カスタムバリデータのサンプルは、「ALINOUS_HOME/boxtest」フォル ダにさまざまなフォームを表示しているページがありますので、そこを参考にします。まずは、「ALINOUS_HOME/boxtest/index.html」の内容を見てみます。


```
<html>
  <head>
    <title>Alinous-Core</title>
  </head>

  <body>
  TEST FORM input<br>
  <br>
  Testing alns:if attribute.<br>
    <span alns:if="{BOOL($AA != null)}">
    $AA is not null.
    </span>

    <form name="testForm" action="show.html" method="POST">

      <select name="sel" alns:validate="custom">
        <option value="{$Record}"
                alns:iterate="@TEST" alns:variable="Record">
        {$Record}
        </option>
      </select>
      <br>
      <input type="radio" name="rtest" value="A"
                                      alns:validate="custom">A<br>
      <input type="radio" name="rtest" value="B">B<br>
      <input type="radio" name="rtest" value="C">C<br>
      <br><br>
      <input type="text" name="txt" value="Hello world"
                                   alns:validate="custom"><br>
      <br>
      Multiple select.<br>
      Please add "[]" at the end of parameter's name.
      Then the parameter becames Array parameter.<br>
      If we watch the variables after submiting the form with debugger,
      the array named 'multisel' is there under the $IN variable.
```



```
    <select name="multisel[]" alns:validate="custom" multiple="true">
      <option value="{$Record}"
              alns:iterate="@TEST" alns:variable="Record">
      {$Record}
      </option>
    </select>
    <br><br>
    <input type="checkbox" name="checktest[]" value="A"
                           alns:validate="custom">A<br>
    <input type="checkbox" name="checktest[]" value="B">B<br>
    <input type="checkbox" name="checktest[]" value="C">C<br>

    <br><br>

    <textarea name="txtArea" size="10" alns:validate="custom">
      FirstVALUE
    </textarea>

    <br><br>

    <input name="submit" type="submit" value="submit">
  </form>
 </body>
</html>
```

上のソースを見ると分かるとおり、HTMLの変更はalns:validate属性の値がcustomになっただけです。では、バリデーションのロジックの部分はどこに書くのでしょうか？ それは、このフォームがPOSTする先のページのAlinousScriptになります。上のソースのformタグでは、POSTする先のaction属性が「show.html」になっているので、「ALINOUS_HOME/boxtest/show.alns」を見てみましょう。


```
// please put breakpoint at the next line
return 0;

function validate($formName, $inputName, $value, $IN, $SESSION)
{
    if($value == ""){
        return "custom";
    }
    return 0;
}

function validateArray($formName, $inputName, @value, $IN, $SESSION)
{
    // Write validation logic for array parameters here
    return 0;
}
```

ここで、上のソースを見ると2つの関数があることが分かります。validate()とvalidateArray()関数がバリデーションのロジックになります。なぜ、2種類あるかというと、フォームのinputタグには、値が通常の変数のものと配列のものがあるからです。通常変数のバリデーションロジックは、validate()関数で処理され、配列のものはvalidateArray()関数で処理されます。

次に関数について説明します。どちらの関数も引数を5個とります。validate()とvalidateArray()関数での違いは、第3引数がvalidate()関数の方は普通の変数で、validateArray()関数の方は配列（変数名が@から始まる場合は配列を意味する）になっているところです。それぞれ引数の説明は次のようになります。

- $formName…入力されたフォームの値が入っています。
- $inputName…inputのname属性の値が入っています。これから評価する値がどのinputタグの値かを識別するためのものです。
- $value or @value…$inputNameで指定されたinputタグの値が入っています。

- $IN…入力パラメーターが入っています。
- $SESSION…セッションの値が入っています。このDOM変数にプロパティを足した場合は、セッションとして記憶されます（後述のセッションの操作参照）。

バリデーションの結果は、関数の戻り値で返します。もし、0をreturnした場合には、バリデーションOKということになります。任意の文字列を返した時には、バリデーションはNGで、ここで返した値がエラーコードになります。

エラーコードは、どこで使うのでしょうか？ それは、バリデーション失敗時のメッセージ表示の際に使います。上のコードでは、バリデーション失敗時にメッセージが表示されませんので、メッセージ表示のためのspanタグをこれから追加します。ラジオボタンの下にある「txt」という名前のテキストボックスがバリデーションに失敗したときのエラーメッセージ表示を付け加えてみます。

```
<span alns:msg="txt" alns:form="testForm" alns:validate="custom">
  <font color="#FF0000">テキストを入力してください</font><br>
</span>
```

このとき、alns:validate属性にエ ラーコードを入力します。これによりバリデーションチェックでエラーコードが返ってきた際、このエラーメッセージが表示されるようになります。実際のバリ デーションの動きは、バリデータの関数にブレイクポイントを仕掛けてデバッガで実際に動かしてみるとよく分かると思います。バリデータ関数は、カスタムバ リデータをセットしたinputタグの個数と同じ回数呼ばれ、どのinputタグの値を評価すべきかは、引数の$inputNameで判別します。

6.3.SQL の実行

Alinous-Core では SQL をそのままスクリプトに書き込んで実行で きます。SQL の仕様としては、PostgreSQL の仕様にあわせて作ってあります。基本的には SQL 92 の仕様に、Limit 句、Offset 句を加えた仕様になっています。現在のところ、UNION などの集合演算には対応していませんが、

- SELECT 文
    - DISTINCT 句
- エイリアス指定
- サブクエリー
- 各データベースの組み込み関数
- FROM 句
- INNER JOIN
- LEFT [OUTER] JOIN
- RIGHT [OUTER] JOIN
- CROSS JOIN
- WHERE 句条件文
- WHERE 句内部結合
- ORDER BY 句 ASC
- ORDER BY 句 DESC
- LIMIT 句
- OFFET 句
- INSERT 文
    - サブクエリー
- VALUES 句
- UPDATE 文
    - サブクエリー
- SET 句
- WHERE 句による条件指定
- DELETE 文
    - 全行削除
- WHERE 句による条件指定
- BEGIN 文
- COMMIT 文
- ROLLBACK 文

など、WebDB開発で一般的に使われるものはほぼカバーされています。Alinous-Coreでは、SQLとスクリプトの変数を組み合わせて使うことができるため、SQLが拡張されている部分があります。今回は主にその部分を解説していこうと思います

6.3.1.拡張SQLの例

拡張SQLの例として、1番分かりやすいのは「ALINOUS_HOME/select/show.alns」の部分です。ソースコードは次のようになります。

```
// EXECUTE SELECT STATEMENT
if($IN.comment != null){
    $LIKE_STMT = '%' + $IN.comment + '%';
}

  SELECT * INTO RECORDS
    FROM SAMPLE_table
   WHERE NAME = $IN.name
     AND EMAIL = $IN.mail
     AND COMMENT like $LIKE_STMT
ORDER BY NAME DESC
         limit 10;
```

このソースコードでは、SELECT文を実行していますが、そのときに、
AlinousScriptの変数をSQLの中で使っています。
$IN.name、$IN.mail、$LIKE_STMTがそれらの変数です。$IN.name、$IN.mailはフォームからPOSTされたフォーム変数で、$LIKE_STMTはSELECT文を実行する直前のスクリプトで作られる変数です。このソースを見て、誰もが「もし、その変数がNULLだったり存在しなかった場合はどうなるのだろう」と思うと思います。その部分についてはこれから説明します。

6.3.2.WHERE句の自動縮退機能

Alinous-Coreの拡張SQLでは、WHERE句に工夫がしてあり、 楽にプログラムを作ることができます。Alinous-Coreは、SQLの解析木を持っていてBNFというツリー形式（もしくはスタック形式）でデータを持っています。図にすると下図のようになります。

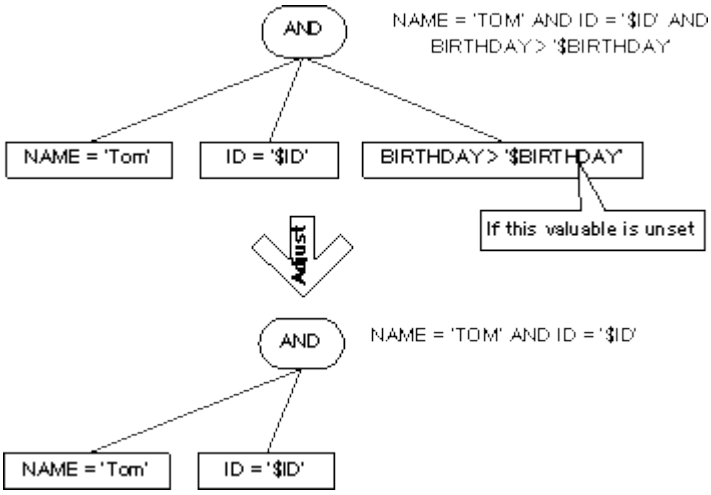

図 6.3.2.1.AND の縮退

この図では、ID と$BIRTHDAY という 2 つの変数が使われていて、$BIRTHDAY が NULL もしくは存在しないときにどのように処理されるかが表されています。この場合は、単純に$BIRTHDAY を含む部分の式が無視されます。ところで、AND 演算子は、2 つ以上式がないと成り立ちません。では、AND 演算子の下の式が 1 つになってしまったときはどのようになるのでしょうか？ そのときの処理の様子を下図で示します。

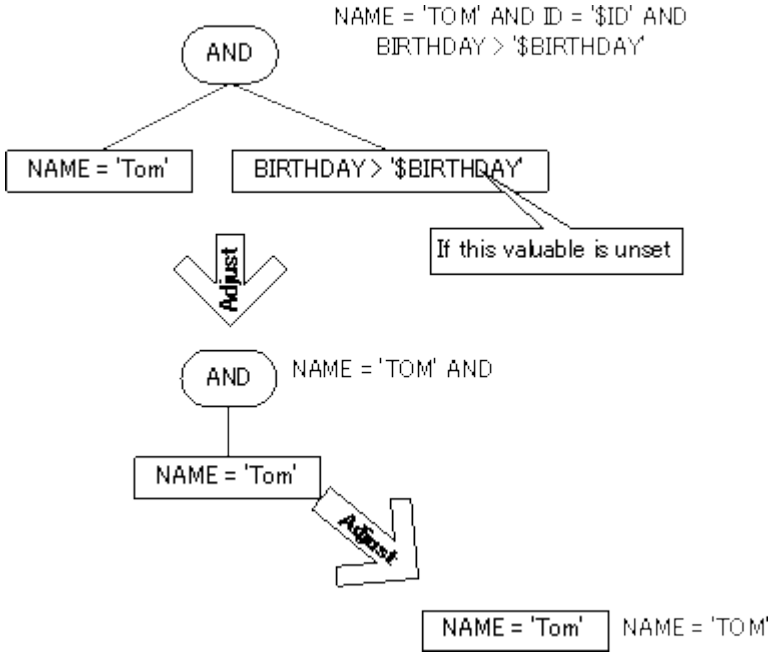

図 6.3.2.2.AND 自体の縮退

この場合には、AND の位置に、AND の下にぶら下がっている式が繰り上がってきます。また、AND の下に式が 1 つもないときは、AND 自体が消滅します。

今回の説明では AND 演算子 1 つだけですが、実際にはもっとたくさんの AND や OR 演算子が入れ子になった構造になります。そのときには、このような処理が、ツリーの下のほうの末端から上（ツリーのルート）に向かって再帰的に行われます。

6.3.3.WHERE 句の縮退機能を OFF にする

WHERE 句の縮退機能はとても便利な機能です。しかし、ときには NULL で あることをチェックしたいときもあり、縮退をしてほしくないこともあります。そのときには、SQL を拡張した文法で「ADJUST_WHERE 句」があ り、これを false に設定することにより、縮退を禁止できます。例としては次のように使います。

```
SELECT * INTO res
  FROM sample_table
 WHERE NAME = $NAME
ADJUST_WHERE = FALSE;

DELETE FROM sample_table
 WHERE NAME = $NAME
ADJUST_WHERE = FALSE;

UPDATE sample_table
   SET NAME = 'TEST'
 WHERE NAME = $NAME
ADJUST_WHERE = FALSE;
```


6.3.4.IN演算子とAlinousScript変数

IN演算子は、AlinousScriptとのコンビネーションでいろいろと使いかたがありますので、その例をいくつか示します。IN演算子は、通常は次のような使いかたをします。

```
SELECT * INTO RECORDS
  FROM sample_table
 WHERE NAME IN ("user1", "user2", $user3);
```

AlinousScriptの配列の変数とセットにして使うときには次のように使います。

```
$ARR[0] = "user1";
$ARR[1] = "user2";
$ARR[2] = "user3";
SELECT * INTO RECORDS FROM SAMPLE_table WHERE
NAME IN (@ARR);
```

また、普通のSQLのようにサブクエリーを使うこともできます。

```
SELECT * INTO RECORDS
  FROM SAMPLE_table
 WHERE NAME IN (SELECT NAME FROM SAMPLE_table);
```


6.3.5.SELECT後のレコードの編集

SELECT文でクエリーを実行したあとには、INTO句で指定した配列にレコードのDOM変数が格納されることになります。実際の開発では、このレコードをそのまま表示するのではなく、一部編集して出したいということが多くあると思います。例えば、<や>を&lt および&gt に変換したり、\nを
に変換したり、タイムスタンプをフォーマットしたりといった処理です。そのような場合は以下のようにして対応します。

```
SELECT * INTO RECORDS
  FROM SAMPLE_table
 WHERE NAME = $IN.name
   AND EMAIL = $IN.mail
   AND COMMENT like $LIKE_STMT
 ORDER BY NAME DESC
       limit 10;

// レコードの個数をとる
$CNT = Array.size(@RECORDS);
for($i = 0; $i < $CNT; $i++){
  // 一つ一つのレコードにたいして変換する
  $RECORDS[$i].NAME = String.replace($RECORDS[$i].NAME, "<", "&lt;");
  $RECORDS[$i].NAME = String.replace($RECORDS[$i].NAME, ">", "&gt;");
}
```

このように、SQLだけでなく、SQL実行後にスクリプトによってデータを編集できるのもAlinous-Coreの大きな特徴です。

6.4.HTTPセッションのハンドリング

セッションは、Webアプリケーションを作るときにおいて、最近では非常によく使われます。Cookieは最近はセキュリティの関係であまり推奨されないことが多いので、代わりにセッションで代用することも多いと思います。

Alinous-Coreのセッションの扱いは非常に簡単です。$SESSIONという予約変数があり、そのDOM変数のプロパティおよび、その子プロパティの子孫はセッション変数として保持されます。デバッガで表示したときに、変数表示は下図のようになっていると思います。

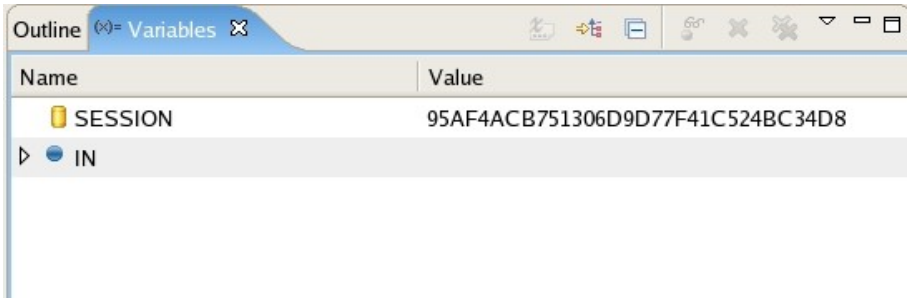

図6.4.1.セッション変数の表示

この$SESSION変数の子孫に値を追加する方法は、下のコードのようになります。

```
$SESSION.test = "aaa";
```

このコードを実行することにより、$SESSION変数に子プロパティが追加されます。デバッガで表示すると下図のようになります。

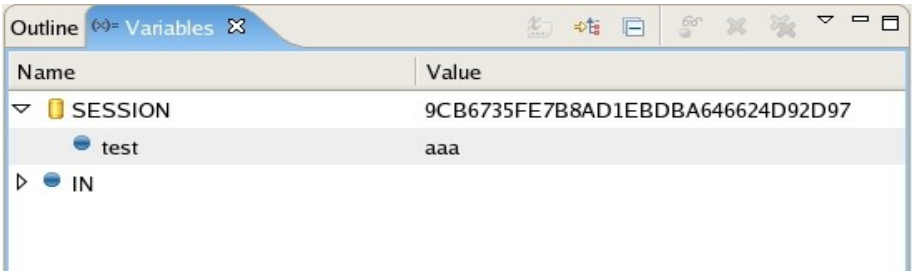

図6.4.2.追加されたセッション変数の表示

また、$SESSION変数の子プロパティ変数を開放したいときには、
Variable.release()関数を使います。具体的には下のようなコードを書きます。

```
Variable.release($SESSION.test);
```

6.5.ページのフォワード機能

サンプルの中のAlinousScriptは、どれも必ず値を返しています。ほとんどのものは0を 返していますが、ここで文字列を返すことによって、フォワードやリダイレクションを行うことができます。ここで返す値が「http://」から始まる場合 は、ページ全体がリダイレクションされます。返す値が、ALINOUS_HOMEの中のファイルの場合にはそのファイルにフォワードされます。

6.6.RSSの作成

Alinous-Coreでは、簡単にRSSの作成も行うことができます。こ のサンプルはサンプルプロジェクトの「ALINOUS_HOME/rss/」のフォルダの中にあります。RSS自体は、 「ALINOUS_HOME/rss/index.alns」と「ALINOUS_HOME/rss/index.rss」で構成されます。今までは、 HTMLファイルとALNSファイルのペアで1つのページが構成されていましたが、今回は、HTMLファイルの代わりにRSSファイルが使われます。 RSSの生成自体は非常に簡単なので、サンプルを見た方がよく分かると思います。

ALINOUS_HOME/rss/index.alns

```
SELECT * INTO RECORDS
  FROM SAMPLE_table
 ORDER BY NAME DESC
       limit 10;

$rdfAbout = "http://localhost:8080/";
$title = "Sample RSS";

return 0;
```


ALINOUS_HOME/rss/index.rss

```
<?xml version="1.0" encoding="utf-8" ?>
<rss version="2.0"
     xmlns:dc="http://purl.org/dc/elements/1.1/"
     xmlns:content="http://purl.org/rss/1.0/modules/content/"
     xml:lang="ja">

  <channel>
    <title>{$title}</title>
    <link>{$rdfAbout}</link>
    <description>Sample RSS</description>

    <item alns:iterate="@RECORDS" alns:variable="rec">
      <title>{$rec.NAME}</title>
      <link>http://localhost:8080/rss/show.html?name={$rec.NAME}
      </link>
      <description>{$rec.COMMENT}
          The email is {$rec.EMAIL}
      </description>
    </item>
  </channel>
</rss>
```

6.7.Basic認証、フォーム認証

Basic認証やフォーム認証は、WebDBアプリケーションを作る上で非常に重要な機能です。Alinous-Coreおける認証機能は、「alinous-config.xml」に設定することで動作させることができます。実際には以下のような記述をします。


```
<?xml version="1.0" encoding="utf-8"?>
<alinous-config>
  <system>
    <system-datastore id="pgsrc" />
    <default-datastore id="pgapps" />
  </system>

    (中略)

  <basic-auth>
    <relm>
      <datastore>pgsrc</datastore>
      <table>AUTH_TABLE</table>
      <users>USERS</users>
      <passwords>PASSWORDS</passwords>
      <roles>ROLES</roles>
    </relm>

    <zones>
      <zone>
        <area>/admin/</area>
        <roles>admin</roles>

        <form-auth>
          <login>/admin/login.html</login>
          <confirm>/admin/confirm.html</confirm>
        </form-auth>
      </zone>

      <zone>
        <area>/admin/system/</area>
        <roles>admin,owner</roles>
        <error-page>/admin/error.html</error-page>
      </zone>
    </zones>
  </basic-auth>
</alinous-config>
```


6.7.1. レルムの設定

レルムとは、「ユーザーの ID」「パスワード」「ロール」を管理するためのデータベースのことです。Alinous-Core では、どのテーブルのどのカラムをどのように割り当てるかを設定ファイルで決めることができ、<realm>タグの中に設定します。

datastore	どのデータベースを使うか
table	どのテーブルを使うか
users	ユーザー ID を格納するためのカラム
passwords	パスワードを格納するためのカラム
roles	ロールを格納するためのカラム

6.7.2. ゾーンの設定

ゾーンの設定は<zones>タグの中で行います。

area	指定したディレクトリ以下がそのゾーンの場所になります
roles	どのロールのユーザーを許可するか（コンマ区切りで複数可）
error-page	Basic 認証失敗時のエラーページ（省略化）
login	フォーム認証の際に利用するログインフォームページ（省略化）
confirm	ログイン処理の成功時、確認時に利用するページ（省略化）

サンプルの設定ファイルには、2 つのゾーンが設定されています。1 つの ゾーン「/admin/」はフォーム認証で管理され、もう 1 つのゾーン「/admin/system/」は、ベーシック認証で管理されます。ゾーンのマッ チングは、最長マッチングで処理されます。例を挙げると、「/admin/system/index.html」は、両方のゾーンに含まれますが、最長 マッチのルールを適用すると「/admin/system/」のゾーンの方にマッチし、こちらのゾーンの認証ルールが適用されます。

そのゾーンが Basic 認証になるか、フォーム認証になるかは<form-auth>タグに含まれる 2 つの設定項目、login と confirm の有無によって判断されます。Basic

認証の場合は、ログアウトがない代わりに、認証は設定ファイルのみで済ませることができます。一方、フォーム認証の場合は認証の処理を自分で書く必要があります。

6.7.3. フォーム認証のログイン

フォーム認証を設定した場合、そのゾーンに未認証でアクセスすると設定 したログインフォームが表示されます。また、認証をかける時に、そのログインフォームが POST する先は confirm で設定したページにします。この 2 つのページは、そのゾーンの中にありますが、認証のためのものなのでログインしていない状態でもアクセスすることができます。

ここで、ログインすることとは、どういうことかを説明します。Alinous-Core では、ログインするということは、セッションの中にログイン情報を生成することを意味します。セッション情報をどのような形式で生成するかは以下のようになります。

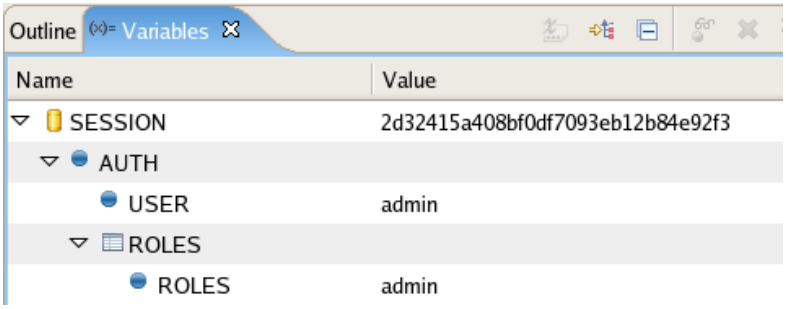

図 6.7.3.1. セッションのフォーマット

まずは、ログインするためのフォームを作らなくてはなりません。ログインフォームは下図のようなものを作成します。ログインフォームは設定ファイルの<login>タグで指定したページを使います。

図 6.7.3.2. ログインのためのフォーム

ソースは次のようになります。

```
<FORM name="login" action="confirm.html">

  <SPAN alns:msg="userId" alns:form="login" alns:validate="custom">
    <FONT color="#FF0000">
    Please input correct USER ID and PASSWORD.
    </FONT><BR>
  </SPAN>

  <TABLE>
    <TR>
      <TD>
      USER ID
      </TD>
      <TD>
        <INPUT type="text" name="userId"
            value="" alns:validate="custom">
      </TD>
    </TR>
    <TR>
      <TD>
      PASSWORD
      </TD>
      <TD>
      <INPUT type="password" name="password" value="">
      </TD>
    </TR>
  </TABLE>
  <INPUT type="submit" value="login">
</FORM>
```

ここでは、Alinous-Core のカスタムバリデータを利用しています。alns:validate という属性が最初の INPUT タ グについていますが、この属性を指定することでカスタムのバリデーションをかけることができます。カスタムのバリデーションが指定された場合、このフォー ムから POST したときにカスタムのチェック関数が呼ばれ、値が不正だと判断された場合には、再度このフォームに強制的に画面遷移します。このときに、<SPAN>タグの中身のエラーメッセージが表示されます。では、カ

スタムのバリデーションのロジックはどこにあるのでしょうか？ このフォームをPOSTする先の「confirm.html」のロジックに対応する「confirm.alns」を見てみましょう

```
// 通常のロジックはここからスタート
// return で0以外を返すとそのページにフォワードされる
return "/admin/";

// バリデーター用のコールバック関数
function validate($formName, $inputName, $value, $IN, $SESSION)
{
  $userId = $IN.userId;
  $pass = $IN.password;


  SELECT * INTO RECORDS FROM AUTH_TABLE
   WHERE     USERS = $userId AND
         PASSWORDS = $pass;

  $NumRoles = Array.size(@RECORDS);

  if($NumRoles == 0){
    return "custom";
  }else{
    $SESSION.AUTH.USER = $userId;
    for($i = 0; $i < $NumRoles; $i++){
      $SESSION.AUTH.ROLES[$i] = $RECORDS[$i].ROLES;
    }
  }

  return 0;
}
```

validate()という関数がありますが、カスタムのバリデーションのロジックはこの関数で実装します。この関数では、0を返したときは値が正常で、不正な値の場合は、不正な理由を文字列で返します。この不正な理由は、エラーメッセージを表示するための<SPAN>タグに設定されているalns:validate="custom"の 部分と連動しています。バリデーションが正常に行われると、通常通り、「confirm.alns」の先

頭からロジックが始まり、次に対応するデザインの 「confirm.html」が呼ばれます。今回はreturn文で0以外の値を返しているのでそのページにフォワードされます。

6.7.4.フォーム認証のログオフ

ログオフの作業はごく簡単です、ログオフするには、ログインの時にセッションに焼き付けたユーザー情報をシステム関数Valuable.release($VAL)で開放します。

```
Valuable.release($SESSION.AUTH);

return 0;
```

6.8.メールの高速大量送信

メールはWebとはもはや、切っても切り離せない関係です。しかも大量に、そして高速にメールを配信したいという需要は相当あるのではないかと思います。
Alinous-Coreでは単純にメールを送るだけではなく、高速にメールを送るための機能も備えています。

6.8.1.SMTPの設定

まずは、「ALINOUS_HOME/alinous-config.xml」にSMTPサーバの設定をする必要があります。


```
<mail>
  <server>mail.nanntoka.com</server>
  <port>25</port>

  <!-- If language encoding is necessary -->
  <lang-encode>ISO2022JP</lang-encode>

  <!-- If authentication is necessary -->
  <auth>
    <method>login</method>
    <user>user</user>
    <pass>pass</pass>
  </auth>
</mail>
```

日本語のメールを送りたい場合は、lang-encode を設定します。また、メール送信時に SMTP サーバの認証が必要な場合は、auth タグに含まれる内容を記述します。プロバイダによりますが、SMTP サーバからメール送信する際に認証が必要のない場合はこの部分は省略してください。

6.8.2.メールの送信

Alinous-Core で、メールを送信する場合には、Mail.send()関数を使います。下のサンプルコードを参考にしていただければ、使い方は分かるかと思います。
上のソースと、その前の通常のメール送信のプログラムで違うところは、$mail オブジェクトの to に配列を代入しているところです。このようにすることによって高速に SMTP プロトコルでメールを送信できます。今回は、to フィールドで例を示しましたが、cc、bcc も同様のことが可能です


```
// メールオブジェクト作成
$mail.from = "dummy@hogehoge.com";
$mail.to = "hoge@hogehoge.com";
$mail.cc = "hoge2@hogehoge.com";
$mail.bcc = "hoge2@hogehoge.com";
$mail.subject = "Test mail";

$mail.body = "こんにちは。

これは、テストメールです。

とりあえず、受信したら捨ててしまってかまいません。";

// Send
Mail.send($mail);
```

この例は、TO、CC、BCC のフィールドにそれぞれ 1 つずつアドレスを設定して送信している例です。では、大量送信したい場合にはどうしたらよいのでしょうか？実は、下の例のように、TO、CC、BCC のフィールドは配列型を代入できます。

```
$toArray[0] = "hoge@hogehoge.com";
$toArray[1] = "hoge1@hogehoge.com";
$toArray[2] = "hoge2@hogehoge.com";

// メールオブジェクト作成
$mail.from = "dummy@hogehoge.com";
@mail.to = @toArray;
$mail.cc = "hoge2@hogehoge.com";
$mail.bcc = "hoge2@hogehoge.com";
$mail.subject = "Test mail";

$mail.body = "こんにちは。

これは、テストメールです。
かなり高速に送信が可能ですので、スパムメール配信には絶対に利用しないでください。";

// Send
Mail.send($mail);
```


上のソースと、その前の通常のメール送信のプログラムで違うところは、$mail オブジェクトの to に配列を代入しているところです。このようにすることによって高速に SMTP プロトコルでメールを送信できます。今回は、to フィールドで例を示しましたが、cc、bcc も同様のことが可能です。

6.9. ファイルアップロード

Web アプリケーションでは、画像のアップロード機能が必要になることもあります。Alinous-Core では、画像やファイルのアップロードにも対応しており、業務系以外にもブログや SNS、CMS などの開発も可能になっています。

6.9.1. ファイルアップロード用のフォーム

ファイルアップロードのサンプルは「ALINOUS_HOME/upload」にあります。最初のファイルを入力するためのフォームは「ALINOUS_HOME/upload/index.html」にあります。まずは、入力フォームのソースコードを示します。


```
<form name="upload" action="upload.html" method="POST"
      enctype="multipart/form-data">

  <table>
  <tr>
    <td>FILE NAME</td>
    <td>
      <input type="text" name="filename" size="18"
             alns:validate="notNull">
      <span alns:msg="filename" alns:form="upload"
            alns:validate="notNull">
        <font color="#FF0000">Input FILE NAME.</font><br>
      </span>
    </td>
  </tr>

  <tr>
    <td>SELECT FILE</td>
    <td>
      <input type="file" name="file" size="18"
             alns:validate="custom">
      <span alns:msg="file" alns:form="upload"
            alns:validate="NO_NAME">
        <font color="#FF0000">Input FILE to upload.</font><br>
      </span>
      <span alns:msg="file" alns:form="upload"
            alns:validate="WRONG_FILE">
        <font color="#FF0000">File type is wrong.</font><br>
      </span>
    </td>
  </tr>

  <tr>
    <td></td>
    <td><input type="submit" name="submit" value="UPLOAD"></td>
  </tr>
  </table>
</form>
```

このフォームでは、method 属性を POST に設定し、enctype 属性の値を

multipart/form-dataに設定しています。そうすることにより、ファイルをマルチパートのフォームデータとしてサーバに送ることができます。逆に、この設定を忘れると、ファイルのアップロードができないので気を付けてください。ファイルの選択自体はtype属性がfileのフォームインプットで行います。具体的には次の部分になります。

```
<input type="file" name="file" size="18"  alns:validate="custom">
<span alns:msg="file" alns:form="upload" alns:validate="NO_NAME">
  <font color="#FF0000">Input FILE to upload.</font><br>
</span>
<span alns:msg="file" alns:form="upload" alns:validate="WRONG_FILE">
  <font color="#FF0000">File type is wrong.</font><br>
</span>
```

今回、抜き出した部分には、バリデータ用のメッセージも含まれています。file型のフォームインプットに対するバリデータの使いかたは後述します。

6.9.2.アップロードしたファイルの受け渡し

アップロードしたファイルの受け渡しは、「ALINOUS_HOME/tmp」フォルダにテンポラリーファイルを利用して行われます。テンポラリーファイルの名前は、

- [SESSION ID]_[FILE NAME]

というフォーマットになります。例を示しますと、「sample.jpg」というファイルをアップロードしたときに、現在のセッションの値が「A43B2C8C0408BE4F741C7B096113E22B」だったとします。この時のテンポラリファイルのパスは「ALINOUS_HOME/tmp/A43B2C8C0408BE4F741C7B096113E22B_sample.jpg」になります。

6.9.3.バリデーションとその後の処理

ファイルのアップロードに関しても、バリデータを使うことができます。ファイルのアップロードの場合は、custom型のバリデータを利用します。


```
$tmpFile = "/tmp/" + $SESSION + "_" + $IN.file;

File.makeDir("/upload/images/");
File.copy($tmpFile, "/upload/images/" + $IN.file);

return 0;

function validate($formName, $inputName, $value, $IN, $SESSION)
{
  if($inputName == "file"){
    return validateFile($value);
  }
}

function validateFile($value)
{
  if($value == ""){
    return "NO_NAME";
  }

  $value = String.toLowerCase($value);
  if(String.endsWith($value, ".gif") ||
     String.endsWith($value, ".jpg") ||
     String.endsWith($value, ".jpeg") ||
     String.endsWith($value, ".png")){
     return 0;
  }
return "WRONG_FILE";
}
```

ファイルアップロード用のインプットにcustom型のバリデータを設定した場合には、通常の場合と同じvalidate()関数が呼ばれます。このとき、$inputNameには、アップロードしたファイルの名前（パスからフォルダをのぞいた名前）が入ります。

バリデーション成功後のロジックは、ソースの先頭部分になります。ここでは、パラメーターの$IN.file（fileはファイルアップロードのフォームのname属性）から、テンポラリーファイルへのパスを作り、イメージを保存するフォルダにコピーしています。

7.運用環境のセットアップ

