

データ変換及びアクセスライブラリ

NetConscious, Inc.

2006 年 1 月 31 日

データ変換及びアクセスライブラリの概要

データ変換及びアクセスライブラリは、Daruma の基本機能アクセスツールやデータ構造変換ツールが使用しているライブラリ群である。

1 テクニカルアーキテクチャ

MISP 基本機能アクセスツールは、拡張性を考慮し、各コマンドに対応したコマンド・アドオンと、それを実行するコマンドランタイムの二つで構成されている。また、コマンド・アドオンは、Message インターフェース、Payload インターフェース、Processor インターフェースの 3 種類の Java インタフェースを基本としている。

1.1 ライブラリのディレクトリ構成

本ツールのディレクトリ構成は、大きく分けて、コンポーネントプラグインのための components、各種設定ファイルを格納する etc、コマンド・ランタイムのライブラリを格納する lib、コマンド・アドオン本体及びライブラリを格納する work で構成されている。以下にディレクトリ構成を示す。

<HOME>

components <-機能拡張を行う場合のコンポーネントを格納するディレクトリ

etc <-各コマンドの設定ファイルを格納するディレクトリ

lib <-コマンド実行環境の基本ライブラリファイルを格納するディレクトリ

work <-コマンド実行環境の作業用ディレクトリ

beans <-コマンド実行環境をカスタマイズする場合に使用するディレクトリ

classes <-拡張クラスを格納するためのディレクトリ

components <-各コンポーネントが使用する作業用ディレクトリ

lib <-コマンド実行環境の拡張ライブラリファイルを格納するディレクトリ

log <-ログフォルダ

samples <-サンプルデータディレクトリ

sysprops <-各コマンドのパラメータをカスタマイズする場合に使用するディレクトリ

xsl <-各コマンドが使用する XSLT ファイルを格納するディレクトリ

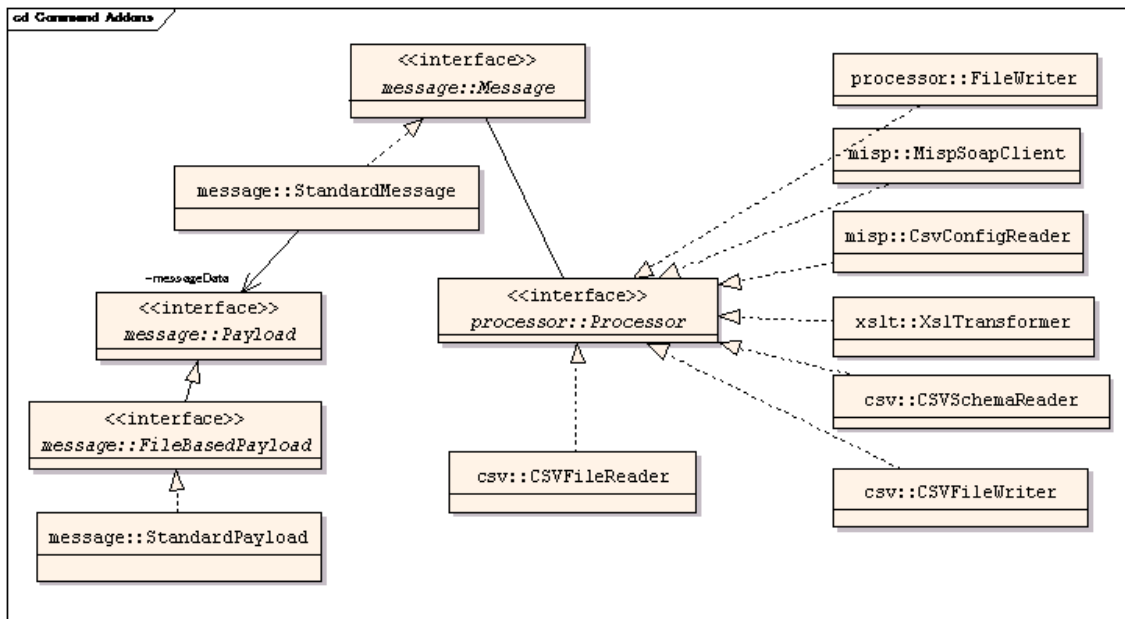


図1 クラスダイアグラム

1.2 コマンド・ランタイム (アプリケーションプラットフォーム)

MISP 基本機能アクセスツールは、NetConscious が LGPL として開発しているアプリケーションプラットフォーム^{*1}を基に開発されている。

このアプリケーションプラットフォームは、コンポーネントプラグインアーキテクチャと DI コンテナを基本技術として実装しており、コマンドラインツールからサーバアプリケーションまでを実装することができる。今回、コンポーネントは使用していないが、ServletContainer コンポーネントを追加することにより、容易にサーバ機能を追加実装することができる。

1.3 コマンド・アドオン (アドオンプログラム)

各コマンドは、コマンド・アドオンとして、コマンド・ランタイム上で動作するアドオンプログラムとして開発されている。共通のランタイム部分とコマンドの処理部分を分離することにより、設計の柔軟性と Java クラスの可搬性を高めている。

1.3.1 オブジェクトモデル

各コマンドの基本処理フローは、データを入力し、処理し、出力するという共通のフローを持っており、これらは3つの基本的なインターフェースによってオブジェクトモデルが形成されている。データ本体を扱う Payload インターフェース、処理を行う Processor インタフェース、Processor と Payload を関連付ける Message インターフェースである。

^{*1} NetConscious BMX (BlueMeme/Basis eXpress Edition)

Payload インターフェース Payload インターフェースは、データ本体をストリーム形式で扱うためのインターフェースである。本ツールのコマンドは、このインターフェースを派生した `FileBasedPayload` の実装クラスである `StandardPayload` クラスを使用しており、すべてのデータはファイルを主体として処理される。以下に、`StandardPayload` クラスの使用例を示す。

```
// メッセージオブジェクトの生成
StandardMessage message = new StandardMessage();
StandardPayload payload = new StandardPayload();

// リクエストファイルのアタッチ
payload.attach(new File("request.xml"));
message.setPayload(payload);
```

Processor インターフェース Processor インターフェースは、`Message` オブジェクトが保持する `Payload` オブジェクトを処理するためのインターフェースである。本コマンドで開発している `Processor` インターフェースを実装したクラスは、`Payload` オブジェクトを処理するために、システムプロパティと `Message` オブジェクトが保持しているプロパティの双方を参照し、処理を実行している。以下に `Processor` の実装クラスである `FileWriter` の使用例を示す。

```
// メッセージオブジェクトの生成
StandardMessage message = new StandardMessage();
StandardPayload payload = new StandardPayload();

// リクエストファイルのアタッチ
payload.attach(new File("request.xml"));
message.setPayload(payload);

// プロパティの設定
message.setProperties(new Properties());
message.getProperties().setProperty(FileWriter.PROPS_OUTPUT_PATH, "result.txt");

// 処理の実行
FileWriter fileWriter = new FileWriter();
fileWriter.process(message);
```

Message インターフェース Message インターフェースは、`Payload` オブジェクトと複数の `Processor` オブジェクトを扱うためのインターフェースである。Message インターフェースを使用することにより、様々な変換処理を行う複数の `Processor` オブジェクトを組み合わせ、一つの処理を実行することができる。ま

た、Processor の処理順序とプロパティを指定できるため、複雑な多段階変換処理を実現することができる。本ツールのコマンドは、このインターフェースを実装した StandardMessage クラスを使用している。以下に StandardMessage クラスの使用例を示す。

```
// メッセージオブジェクトの生成
StandardMessage message = new StandardMessage();
StandardPayload payload = new StandardPayload();

// リクエストファイルのアタッチ
payload.attach(new File("request.xml"));
message.setPayload(payload);

// プロパティの設定
message.setProperties(new Properties());
message.getProperties().setProperty(FileWriter.PROPS_OUTPUT_PATH, "result.txt");
message.getProperties().setProperty(XslTransformer.PROPS_XSLT_FILE, "sample.xslt");

// 処理の実行
MispSoapClient soapClient = new MispSoapClient();
XslTransformer transformer = new XslTransformer();
FileWriter fileWriter = new FileWriter();

// プロセッサの登録 （追加した順序で実行される）
message.setProcessorList(new ArrayList());
message.getProcessorList().add(soapClient);
message.getProcessorList().add(transformer);
message.getProcessorList().add(fileWriter);

// 処理の実行
message.send();
```

1.4 コマンドの実行

本ツールのコマンドは、すべてスクリプトを使用して実行することが可能であるが、Java コマンドを使用して起動することができる。各コマンドは、全てコマンド・ランタイム上で動作するコマンド・アドオンであるため、コマンド・ランタイムを起動後、コマンド・アドオンをロードして実行する必要がある。

本ツールは、起動された JavaVM のシステムプロパティから JAVA_HOME の設定値を取得するため、JAVA_HOME の設定は必須ではない。

1.4.1 コマンド・ランタイムの起動

コマンド・ランタイムを起動するには、ランタイムのホームディレクトリをシステムプロパティで指定し、startup.jar を -jar オプションを指定して Java コマンドを実行する必要がある。ランタイムのホームディレクトリを省略した場合は、Java VM を起動したカレントディレクトリをホームとして設定する。以下に、コマンド・ランタイムの起動方法と示す。

```
set DARUMA_HOME=c:\darumatool
java -Dapplication.home="%DARUMA_HOME:\=/" -jar startup.jar
```

コマンドを実行するカレントディレクトリが本ツールのホームである場合は、-Dapplication.home は省略可能である。

1.4.2 コマンド・アドオンのロード

コマンド・ランタイム単体を起動した場合、etc ディレクトリ以下の default.system.properties.xml を読み込み標準の StandardApplication クラス（標準のコマンド・アドオン）が起動する。特定のコマンド・アドオンをロードするには、コマンド・ランタイムの起動時に、システムプロパティ system.properties.xml に対して、ロードしたいコマンド・アドオンが記述されたプロパティファイルを設定する必要がある。以下に、コマンド・アドオンのロード方法を示す。

```
set DARUMA_HOME=c:\darumatool
java -Dapplication.home="%DARUMA_HOME:\=/"
-Dsystem.properties.xml="./etc/properties.xml" -jar startup.jar
```

コマンドを実行するカレントディレクトリが本ツールのホームである場合は、-Dapplication.home は省略可能である。

1.4.3 コマンド・アドオンの設定ファイル

コマンドアドオンの設定ファイルは、etc ディレクトリの各コマンド名のディレクトリ以下に格納されている。設定ファイルには、system.properties.xml と system.context.xml の2種類の XML ファイルがある。

system.properties.xml ファイル system.properties.xml は、コマンド・ランタイム及びコマンド・アドオンの動作を設定するものであり、以下の system.context.xml のパスや、起動するメインクラスを記述する。サーバーモードの切り替えもこのファイルを行うことができる。メインクラスを変更することによって、様々なコマンド・アドオンを同一のコマンド・ランタイム上で起動することができる。以下に system.properties.xml の設定例を示す。

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<!--モード切替 -->
<entry key="system.server.mode">false</entry>
<entry key="system.silent.mode">true</entry>
<!-- システムパラメータ -->
<entry key="system.context.cfg.xml">
${application.home}/etc/darumaclient/system.context.xml
</entry>
<!-- ランタイムパラメータ -->
<entry key="application.name">DarumaClient for MISP</entry>
<entry key="application.description">
2006 AIST DaRuMa Project - Daruma Client for MISP
</entry>
<entry key="application.main.class">daruma.main.DarumaClient</entry>
<!-- アドオンパラメータ -->
<entry key="mispsoapclient.xslt.copy.template">
${application.home}/work/xsl/standard_copy.xslt
</entry>
<entry key="darumaclient.retrieve.soap.body.xslt.template">
${application.home}/work/xsl/retrieve_soap_body.xslt
</entry>
</properties>

```

system.context.xml ファイル system.context.xml は、DI コンテナで処理される Bean 定義ファイルである。本ツールでは、コマンドラインのオプション及びパラメータの定義に使用している。以下に system.context.xml の設定例を示す。

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">

<beans>
<bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
</bean>
<bean id="commandLineHelper" class="application.runtime.CommandLineHandler">
<property name="commandName"
value="ShowFeatureTypes [-debug] [-out OUTPUT_FILE] -host HOSTNAME"/>
<property name="parser">
<bean class="org.apache.commons.cli.BasicParser"/>
</property>
<property name="optionList">
<list>
<bean class="application.runtime.CommandLineOption">
<property name="optionName" value="debug"/>
<property name="description" value="中間ファイルを削除しません。"/>
<property name="hasValue" value="false"/>
<property name="required" value="false"/>
<property name="argName" value=""/>
<property name="argCount" value="0"/>
<property name="optionalArg" value="false"/>
</bean>
<!-- Server Parameters -->
<bean class="application.runtime.CommandLineOption">
<property name="optionName" value="host"/>
<property name="description" value="サーバのホスト名又は IP アドレス"/>
<property name="hasValue" value="true"/>
<property name="required" value="true"/>
<property name="argName" value="HOSTNAME"/>
<property name="argCount" value="1"/>
<property name="optionalArg" value="false"/>
</bean>
<bean class="application.runtime.CommandLineOption">
<property name="optionName" value="out"/>
<property name="description" value="出力ファイル名 (XML ファイル)"/>
<property name="hasValue" value="true"/>
<property name="required" value="false"/>
<property name="argName" value="OUTPUT_FILE"/>
<property name="argCount" value="1"/>
<property name="optionalArg" value="false"/>
</bean>
</list>
</property>
</bean>
</beans>

```

1.5 システムプロパティによるパラメータ設定

各 Processor オブジェクトやコマンドは、システムプロパティによって、動作が決定される。システムプロパティを設定する方法は、コマンド・ランタイム及びコマンド・アドオンの双方で用意されており、適用される優先度によって使い分けることができる。

1. Java VM の-D オプションによる設定値
2. etc ディレクトリ以下の default.system.properties.xml ファイルの値
3. etc ディレクトリ以下の各コマンド用ディレクトリの system.properties.xml ファイルの値
4. work/sysprops ディレクトリ以下の拡張子が xml であるファイルの値
5. プログラム内で System オブジェクトに対して設定した値

2 プロセッサクラス

プロセッサクラスは、Processor インターフェースを実装したクラスである。本ツールは、複数のプロセッサクラスを組み合わせることによって、コマンド処理を実現している。

2.1 FileWriter プロセッサ

FileWriter は、Payload オブジェクトの内容を指定したファイルまたは標準出力に出力するプロセッサである。以下に、パラメータを示す。

パラメータ

filewriter.input.encoding 入力ファイルのエンコーディング
filewriter.output.encoding 出力ファイルのエンコーディング
filewriter.output.path 出力ファイル名 (filewriter.value.output.path.stdout を指定すると標準出力)

2.2 CsvFileReader プロセッサ

CsvFileReader プロセッサは、CSV を、CSVX 形式 (CSV の内部フォーマット) に変換するプロセッサである。以下に、パラメータを示す。

パラメータ

csvfilereader.csvfile.encoding 入力ファイルのエンコーディング
csvfilereader.csvx.encoding 出力ファイルのエンコーディング
csvfilereader.csvfile.comment CSV ファイルのコメントの文字
csvfilereader.csvfile.separator CSV ファイルの値のデリミタ文字
csvfilereader.csvfile.parenthesis CSV ファイルの値の括弧文字
csvfilereader.xslt.copy.template コピー用 XSLT ファイル

2.3 CsvFileWriter

CsvFileWriter は、CSVX 形式を、CSV に変換するプロセッサである。以下に、パラメータを示す。

パラメータ

csvfilewriter.output.encoding 入力ファイルのエンコーディング
csvfilewriter.output.path 出力ファイルのエンコーディング
csvfilewriter.xslt.csvx2csv.template CSVX 形式から CSV に変換するための XSLT ファイル

2.4 CSVSchemaReader

CSVSchemaReader は、CSV 形式を、スキーマ定義に変換するプロセッサである。以下に、パラメータを示す。

パラメータ

csvschemareader.csvfile.encoding 入力ファイルのエンコーディング
csvschemareader.csvschema.encoding 出力ファイルのエンコーディング
csvschemareader.csvfile.comment CSV ファイルのコメントの文字
csvschemareader.csvfile.separator CSV ファイルの値のデリミタ文字
csvschemareader.csvfile.parenthesis CSV ファイルの値の括弧文字

2.5 BuiltinFilter

BuiltinFilter は、GeometoryPropertyType(LineString/Point/Polygon) 部分を、MISP 形式または CSV 形式に変換するプロセッサである。MISP 形式と CSV 形式の変換方向は、BuiltinFilter のコンストラクタで決定される。

```
// CSV 形式から MISP 形式へ変換を行う場合
BuiltinFilter builtinFilter = new BuiltinFilter(BuiltinFilter.MODE_CSV2XML);
// MISP 形式から CSV 形式に変換を行う場合
BuiltinFilter builtinFilter = new BuiltinFilter(BuiltinFilter.MODE_XML2CSV);
```

以下に、パラメータを示す。

パラメータ

なし

2.5.1 GeometoryPropertyType の表現例

基本的にデータ値と CSV のカラムは、一対一でマッピングされるが、データ型が GeometoryPropertyType (LineString/Point/Polygon) の場合は、BuiltinFilter プロセッサクラスによって、特定のルールに従って変

換することが出来る。以下に変換例を示す。

MISP による LineString の表現

```
<gml:Point>
  <gml:coordinates>1.2345,6.7890</gml:coordinates>
</gml:Point>
```

CSV による LineString の表現

```
LINESTRING(1.2345 6.7890,1.1111 2.2222,3.3333 4.4444)
```

MISP による Point の表現

```
<gml:Point>
  <gml:coordinates>1.2345,6.7890</gml:coordinates>
</gml:Point>
```

CSV による Point の表現

```
POINT(1.2345 6.7890)
```

MISP による Point の表現

```
<gml:Polygon>
  <gml:outerBoundaryIs>
    <gml:LinearRing>
      <gml:coordinates>1.2,6.7 1.1,2.2 3.3,4.4</gml:coordinates>
    </gml:LinearRing>
  </gml:outerBoundaryIs>
  <gml:innerBoundaryIs>
    <gml:LinearRing>
      <gml:coordinates>1.2,6.7 5.6,7.7 8.8,9.9</gml:coordinates>
    </gml:LinearRing>
  </gml:innerBoundaryIs>
  <gml:innerBoundaryIs>
    <gml:LinearRing>
      <gml:coordinates>1.2,6.7 1.1,2.2 3.3,4.4</gml:coordinates>
    </gml:LinearRing>
  </gml:innerBoundaryIs>
</gml:Polygon>
```

CSV による Point の表現

```
POLYGON((1.2 6.7,1.1 2.2,3.3 4.4),(1.2 6.7,5.6 7.7,8.8 9.9),(1.2 6.7,1.1 2.2,3.3 4.4))
```

2.6 CsvConfigReader

CsvConfigReader は、CSV 変換ルール定義ファイルから CsvGetFeature 用 XSLT または CsvInsert 用 XSLT を生成するプロセッサである。

パラメータ

csvxsltgenerator.xslt.type 生成する XSLT の種類を指定する。CsvGetFeature であれば csvxsltgenerator.xslt.type.getfeature、CsvInsert であれば csvxsltgenerator.xslt.type.insert を指定する。
csvxsltgenerator.xslt.getfeature.csv.template CsvGetFeature 用 XSLT 生成用 XSLT
csvxsltgenerator.xslt.insert.csv.template CsvInsert 用 XSLT 生成用 XSLT
csvxsltgenerator.output.encoding 出力ファイルのエンコーディング

2.6.1 CSV 変換ルール定義

CSV 変換は、CSV 変換ルール定義ファイルを使用して、CsvFileReader プロセッサクラスと、CsvConfigReader プロセッサクラスを使用して変換を行う。CsvFileReader は、CSV 形式から内部形式である CSVX 形式に変換し、CsvConfigReader は、CSV 変換ルール定義ファイルより、CSVX と MISP 間の変換用の XSLT を動的に生成する。CSV 変換ルール定義ファイルには、CSV のカラムと MISP の XML の XPath 形式を一对で記述する。以下に記述例を示す。

```
<?xml version="1.0" encoding="UTF-8"?>
<cx:Config xmlns:cx="http://staff.aist.go.jp/i.noda/Standard/2005/CsvXml">
<cx:xml>
<xsd:schema id="BuildingInfo.xsd">
... スキーマ定義 (省略)
</xsd:schema>
</cx:xml>
<!-- CSV の変換ルールを定義 -->
<cx:csv element="BuildingFireInfo" cs="," rs="\n" quoteChar="'" commentPrefix="#">
<cx:columnList>
<!-- XPath に MISP の構造、name に CSV の構造を定義する -->
<cx:column xpath="id" name="id"/>
<cx:column xpath="damage/level" name="damagelevel"/>
<cx:column xpath="time/begin" name="beginntime"/>
<cx:column xpath="time/end" name="endtime"/>
</cx:columnList>
</cx:csv>
</cx:Config>
```

2.7 FeatureTypeFilter

FeatureTypeFilter は、スキーマ定義を、XPath 形式に変換するプロセッサである。

パラメータ

featuretypefilter.output.encoding 出力ファイルのエンコーディング
featuretypefilter.separator XPath とデータ型のセパレータ文字

2.8 GetFeatureFilterReader

GetFeatureFilterReader は、様々なフィルタを、GetFeature リクエストのフィルタに変換するプロセッサである。^{*2}

パラメータ

mispssoapclient.filter.type フィルタの種類を指定する。XML 形式であれば mispssoapclient.filter.xml、S 式であれば mispssoapclient.filter.s.expression を指定する。

2.9 MispSoapClient

MispSoapClient は、Payload オブジェクトの内容をサーバへ送信し、レスポンスを Payload オブジェクトに格納するプロセッサである。

パラメータ

mispssoapclient.server.host サーバのホスト名または IP アドレス
mispssoapclient.server.port サーバのポート
mispssoapclient.xslt.copy.template コピー用 XSLT ファイル

2.10 XpathFilter

XpathFilter は、XML 形式を XPath 形式に変換するプロセッサである。

パラメータ

xpathfilter.output.encoding 出力ファイルのエンコーディング
xpathfilter.separator XPath を値のセパレータ文字列

2.11 XslTransformer

XslTransformer は、XML ファイルを指定された XSLT ファイルを用いて別の形式に変換するプロセッサである。

^{*2} 現在、MISP の XML 形式のみ対応

パラメータ

xslttransformer.xml.encoding 出力ファイルのエンコーディング

xslttransformer.xslt.file XSLT ファイル

xslttransformer.omit.xml.declaration XML 宣言を出力するかどうかのフラグ (yes/no)

xslttransformer.indent インデントを出力するかどうかのフラグ (yes/no)