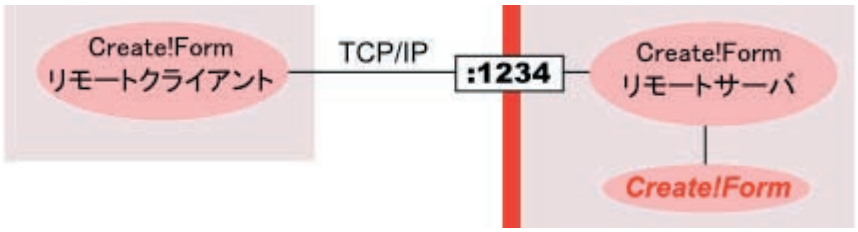
FormRemoteObject

1. 概要

Create!FormRemoteObject（以下、FormRemoteObject）は、ネットワーク上にある Create!Form ランタイムを別のマシンから実行可能にする帳票サーバ分散環境を実現するためのオプション製品です。

図：Create!FormRemoteObject



動作環境やインストール方法については、Design マネージャのメニュー［ヘルプ］-［オンラインマニュアル］から「1. インストール」をご覧ください。

FormRemoteObject はサーバ製品とクライアント API ライブラリから構成されます。

サーバ製品
サーバ製品は、常駐型のサービスとして動作しサーバ上の特定のポートを監視します。クライアントからのリクエストを受け、サーバ上で Create!Form ランタイムを実行し帳票出力を行います。

クライアント製品
クライアント製品は、Java の API ライブラリとして提供されます。サーバへの接続、帳票出力の実行、出力結果の取得などの操作を API から行うことができます。

2. サーバの設定

FormRemoteObject サーバの設定は、FormRemoteObject 導入ディレクトリ内の「conf/cfro-server.properties」の内容を編集することで行います。

2-1. ポート

最低限設定が必要な項目は使用するポート番号を指定する「port」です。以下のように任意のポート番号を指定してください。

```
port = 56789
```

<< 注意 >>
ポート番号を指定していない場合、空いているポート番号が自動で割り当てられます。

2-2. ログ

サーバの稼働状況はログとして出力することができます。ログには通常のサービス稼動ログ、アクセスログ、エラーログの 3 種類があり、アクセスログはクライアントからのリクエストが発生したときのみ出力され、エラーログはエラーが発生したときのみ出力されます。ログファイルは、初期設定では FormRemoteObject 導入ディレクトリ内の「var/log」ディレクトリに出力されます。

ログの出力設定を行う場合は、以下の手順で行います。

1. FormDesign マネージャのメニュー [設定]-[ログ設定] からログ設定ダイアログを起動します。

2. RemoteObject 製品を選択し、[ログ出力する] のチェックを有効にします。

3. サービスの再起動時にログをクリアしたくない場合は [追記モードでログを記録する] のチェックを有効にします。

4. [サービス稼動ログファイル名][アクセスログファイル名] 欄にファイル名を入力します。

5. [OK] ボタンで設定を確定します。

ログの出力設定の詳細については Design マネージャのメニュー [ヘルプ]-[オンラインマニュアル] から「1. インストール」-「1.3 導入環境の設定」-「ログファイルの出力設定」をご覧ください。

2-3. 再実行のための設定

FormRemoteObject 経由で実行された帳票出力ジョブは入力データや実行オプションなどの情報がサーバ上に一定期間保管されるため、その情報を利用して同じジョブを再実行することができます。最初に実行されてから何日間までの間であれば再実行を許可するかを「job.available」で指定することができます。

```
job.available = 1
```

ここでは日数を指定します。「-1」を指定すると、明示的にジョブを削除するまでいつまでも再実行が可能になります。

再実行の詳細については「4-5. 再実行」をご覧ください。

3. サービスの開始・停止

FormRemoteObject はサービスとして動作します。FormRemoteObject サービスの開始・停止は管理用のコマンドラインツールから行うことができます。

3-1. 管理コマンドからの操作

管理コマンドは FormRemoteObject 導入ディレクトリ内の「bin」ディレクトリに含まれる「cfro」コマンドです。FormRemoteObject サービスを開始するには「cfro」コマンドを「start」という引数を指定して実行します。

```
cfro start &
```

サービスを停止するには「stop」という引数を指定して実行します。

```
cfro stop
```

サービスが開始しているか停止しているかを確認するためには「status」を指定します。

```
cfro status
```

サービスが開始している場合は「active」と表示され、停止している場合は「stop」と表示されます。エラーが発生して停止している場合は「stop(error)」と表示され、詳細なエラー内容が表示されます。

3-2. 自動起動の設定

Windows

Windows 版では、FormRemoteObject は Windows サービスとしてインストールされます。サービス名は「cfrod10」、表示名は「Create! FormRemoteObject V10」です。

Windows の起動時に FormRemoteObject サービスを自動的に開始させるには、以下の手順でサービスの設定を変更してください。

1. Windows のコントロールパネルから [管理ツール]-[サービス] を実行します。

2. 「Create! FormRemoteObject V10」サービスのプロパティを表示します。

3. [スタートアップの種類] を [自動] に変更します。

Linux

Linux 版では、FormRemoteObject をサービスとして自動実行するための起動スクリプトが付属しています。この起動スクリプトを「/etc/rc.d/init.d」に登録することで、自動起動を有効にすることができます。

Linux の起動時に FormRemoteObject サービスを自動的に開始させるには、以下の手順を行ってください。ここでは、FormRemoteObject の導入ディレクトリを「/opt/createv10」として説明します。

1. FormRemoteObject 導入ディレクトリ内の「bin」ディレクトリに含まれている「cfrod」スクリプトを、「/etc/rc.d/init.d」へコピーします。

```
# cp /opt/createv10/bin/cfrod /etc/rc.d/init.d/cfrod
```

2. コピーした「cfrod」スクリプトをエディタで開きます。

```
# vi /etc/rc.d/init.d/cfrod
```

3. FormRemoteObject 製品の環境変数のパス、Create!Form ランタイム製品の環境変数のパス、Java の環境変数のパスを変更します。例えば、Create!Form ランタイムとして FormCast を「/opt/createv10」に導入し、Java を「/usr/java」に導入している場合は、以下のように変更します。

（変更前）

```
export CDIR_RO=/home/createform
export CDIR_CAST=/home/createform
export JAVA_HOME=/home/java
```

（変更後）

```
export CDIR_RO=/opt/createv10
export CDIR_CAST=/opt/createv10
export JAVA_HOME=/usr/java
```

4.「/etc/rc.d/init.d」へ移動し、「cfrod」スクリプトの実行権限を変更します。

```
# cd /etc/rc.d/init.d
# chmod 755 cfrod
```

5. 以下のコマンドを実行してサービスを登録します。

```
# chkconfig --add cfrod
# chkconfig cfrod on
```

以上の手順により、FormRemoteObject サービスの自動起動が有効になります。

なお、FormRemoteObject サービスの自動起動を解除したい場合は以下のコマンドを実行してください。

```
# chkconfig --del cfrod
```

4. クライアント API

クライアント API を使用した実行方法を説明します。

4-1. クライアント環境の準備

クライアント API ライブラリは、製品 DVD-ROM の「FormRemoteObject/client/cfroclient1000.zip」に含まれています。ファイルを展開すると「lib」「doc」「sample」という 3 つのディレクトリが展開されます。

クライアント API ライブラリは「lib」ディレクトリに含まれる「cfro-client.jar」です。このファイルをご利用の環境にコピーもしくはクラスパスを設定してご利用ください。

動作に最低限必要なものは「cfro-client.jar」ライブラリのみですが、設定ファイルを用意してライブラリの動作を制御することも可能です。設定ファイルについては「5. クライアント設定ファイル」をご覧ください。

クライアント API のクラスライブラリは以下のパッケージに含まれます。

```
net.createform.ro
```

以下の説明では上記パッケージ名は省略しています。
各 API の詳細については付属の API リファレンス（doc/api/index.html）をご覧ください。

4-2. サーバへの接続

まずは CreateForm オブジェクトを使用してサーバとの接続を確立します。

```
CreateForm server = CreateForm.getInstance("servername", 56789);
```

getInstance メソッドを使用して CreateForm オブジェクトのインスタンスを取得します。その際、引数として接続サーバ名とポート番号を指定します。

取得したインスタンスに対して isAvailable メソッドを呼ぶと指定したサーバに接続可能かどうかを確認することができます。

```
boolean connected = server.isAvailable();
```

false が返ってくる場合は接続に失敗しています。接続先サーバ名、ポート番号が正しいか、アクセス可能なポート番号かどうかを確認してください。また、サーバが起動しサービスが開始しているかを確認してください。

接続確認には sonar メソッドを使用することもできます。

```
try {
    server.sonar();
}
catch (RemoteObjectException e) {
    e.printStackTrace();
}
```

sonar メソッドは接続に失敗すると例外を発生させます。これにより接続エラーの内容を例外として取得することができます。

4-3. 帳票出力の実行

帳票出力を実行する場合は、Job オブジェクトを使用します。Job オブジェクトは、CreateForm オブジェクトの newJob メソッドにより取得します。

```
Job job = server.newJob();
```

次にどの出力ランタイムを使用するかを Job オブジェクトの setMode メソッドで指定します。

```
job.setMode(CreateFormMode.CAST);
```

setMode で指定できる引数は、CreateFormMode という列挙型のオブジェクトです。以下のものが指定できます。

- CAST - FormCast を実行します。
- COLLECT - FormCollect を実行します。
- PRINT - FormPrint を実行します。
- PRINTSTAGE - FormPrintStage を実行します。
- PRINTSTAGE_WEB - FormPrintStage Web を実行します。

Create!Form ランタイムの実行オプションを指定するには addOpt メソッドを使用します。

```
job.addOpt("D", "C:/CreateV10/sample/remoteobject/formfiles");
job.addOpt("s", "sheet");
```

"-D" オプションで指定するパスは「サーバ側の作業ディレクトリ」のパスを指定します。

実行オプションを削除する場合は removeOpt メソッドを使用します。

```
job.removeOpt("o");
```

次に入力データファイルを指定します。

```
job.getSource().addFile("sheet_p1.csv");
```

ここでのファイルはクライアント側にあるファイル名を指定します。

ここまでで実行パラメータのセットアップは完了しました。execute メソッドを実行して帳票出力を実行します。

```
JobResult result = job.execute();
```

execute メソッドを実行するとサーバへ情報が送信され、サーバ上で帳票出力が実行されます。そして実行結果がサーバから送信されます。

実行結果は JobResult オブジェクトとして取得されます。成功したかどうかは getErrorCode メソッドの値で確認できます。

```
int code = result.getErrorCode();
if (code == 0) {
    // 正常終了
}
else {
    // エラー発生
}
```

実行に成功した場合、JobResult オブジェクトの publish メソッドによって、サーバ上に生成された帳票データをクライアント側で OutputStream に書き出すことができます。

```
OutputStream out = new FileOutputStream("result.pdf");
result.publish(out);
out.close();
```

また、ページ数取得オプション（-P、-Pn）が指定されていた場合は、JobResult オブジェクトの getPages メソッドでページ数を取得することができます。

```
job.addOpt("P");
JobResult result = job.execute();

int pages = result.getPages();
```

4-4. 非同期実行

execute メソッドはサーバ上で帳票生成処理が終了するまで処理をブロックします。処理をブロックさせずに一旦処理を返し、結果は後から任意のタイミングで取得する場合は、execute メソッドの代わりに post メソッドを使用します。

```
job.post();
```

JobStatus オブジェクトを取得することで、ジョブの実行状態を確認することができます。

```
JobStatus status = job.getStatus();
if (status.done()) {
    // ジョブの実行が正常に完了している
}
else if (status.fail()) {
    // ジョブの実行中にエラーが発生している
}
else {
    // ジョブを実行中
}
```

done または fail メソッドが true を返す場合は、ジョブの実行が完了しているため実行結果を取得することができます。結果を取得するには getResult メソッドを使用します。

```
JobResult result = job.getResult();
```

また、ジョブはそれぞれユニークな ID（ジョブ ID）を持ちます。この ID を保持しておいて、後からその ID を使ってそのジョブを再取得することもできます。

```
String jobId = job.getId();
...
Job job = server.lookupJob(jobId);
```

これによって、post メソッドの後で一旦クライアントのプロセスが終了したとしても、ジョブ ID さえ分かっていれば別のプロセスから Job オブジェクトにアクセスして結果を受け取ることができます。

非同期実行機能を使用するためにはサーバ側で再実行のための設定「job.available」に「1」以上の値が設定されている必要があります。もしくは次のコードのように setExpiration メソッドで有効期限を指定してから post メソッドを実行する必要があります。

```
job.setExpiration(1);
job.post();
```

ジョブ ID に任意の文字列を指定する

ジョブ ID は基本的にサーバ側で自動的に割り振られますが、Job オブジェクトを生成する際に以下のように newJob メソッドの代わりに newJobWithId メソッドを使用することでジョブ ID を任意の文字列にすることができます。

```
Job job = server.newJobWithId("myjob0001");
```

指定されたジョブ ID のジョブがすでにサーバ上に存在する場合はエラーとなり、null が返されます。

エラーとせずに既存のジョブを上書きさせたい場合は、サーバ側の設定ファイルに以下の記述を追加します。

```
job.id.overridable = true
```

4-5. 再実行

一度実行された帳票出力ジョブは再実行することができます。

再実行機能を使用するためにはサーバ側で再実行のための設定「job.available」に「1」以上の値が設定されている必要があります。もしくは Job オブジェクトの setExpiration メソッドで有効期限を設定する必要があります。

「4-4. 非同期実行」で説明したようにジョブはそれぞれユニークな ID（ジョブ ID）を持っています。このジョブ ID を指定することで実行済みのジョブに再度アクセスすることができます。

```
Job job = server.lookupJob(jobId);
```

再取得した Job オブジェクトに対しては、実行時と同じように execute メソッドや post メソッドを実行することができます。実行オプションを追加したり出力ランタイム種別を変えて実行することも可能です。

```
job.addOpt("Pse", "3"); // 3 ページ目だけ出力
job.setMode(CreateForm.PRINTSTAGE); // FormPrintStage に変更
JobResult result = job.execute();
```

<< 注意 >>
有効期限が切れたジョブを取得した場合、null が返されます。

ジョブ ID リストの取得

再実行可能なジョブ ID の一覧をサーバから取得するには、CreateForm オブジェクトの listJobs メソッドを使用します。

```
List<JobDescriptor> jobs = server.listJobs();
```

JobDescriptor オブジェクトのリストが取得できるので、JobDescriptor オブジェクトからジョブ ID を取得できます。

```
for (Iterator<JobDescriptor> it = jobs.iterator(); it.hasNext(); ) {
    JobDescriptor descriptor = it.next();
    String jobId = descriptor.getId();
}
```

<< 注意 >>
有効期限が切れたジョブはリストで取得することはできません。

実行回数の取得

そのジョブが再実行を含めて何回実行されたかを取得するには、JobDescriptor オブジェクトの getTimes メソッドを使用します。

```
JobDescriptor descriptor = job.getDescriptor();
int times = descriptor.getTimes();
```

出力ランタイムごとの実行回数を取得することもできます。

```
int timesOfCast = descriptor.getTimes(CreateFormMode.CAST);
int timesOfPrintST = descriptor.getTimes(CreateFormMode.PRINTSTAGE);
```

この場合、FormCast ランタイムと FormPrintStage ランタイムの実行回数がそれぞれ取得できます。

ジョブの削除

有効期限内のジョブを再実行可能リストから削除するためには Job オブジェクトの remove メソッドを使用します。

```
job.remove();
```

<< 注意 >>
削除したジョブは復元できません。

5. クライアント設定ファイル

クライアント側は、cfro-client.jar モジュールが配置されていれば動作しますが、「createform.properties」という設定ファイルを用意して API の動作を制御することもできます。「createform.properties」は、CLASSPATH が設定された場所に配置することで有効になります。

※設定ファイルのサンプルは、FormRemoteObject 導入ディレクトリ内の「sample¥remoteobject¥createform.properties」として配置されています。

設定ファイルは以下のような「キー = 値」のプロパティリスト形式で記述されるファイルです。

```
remote.hostname = localhost
remote.port = 56789
```

remote.hostname
接続先のホスト名を指定します。

remote.port
接続先のポート番号を指定します。

「remote.hostname」と「remote.port」が指定されている場合、クライアント API でホスト名とポート番号の指定を省略することができます。

```
CreateForm server = CreateForm.getInstance();
//CreateForm server = CreateForm.getInstance("localhost", 56789); // 通常
```

6. プリセットコンテキスト

実行オプションの指定などはサーバ側であらかじめ指定したものを用意しておくことができます。例えば、出力ランタイム種別と「-D」オプション、「-s」オプションで指定する内容をあらかじめサーバ側で設定しておけば、クライアント側からは次のコード例のようにデータファイルのみを送信して帳票出力を実行することができます。

```
// あらかじめ設定されている "sample" の設定を使用する
Job job = server.newJob("sample");
job.getSource().addFile("sheet_p1_sjis_dos.csv");
JobResult result = job.execute();
```

この機能を利用する場合は、FormRemoteObject サーバの導入ディレクトリ内にある「context」ディレクトリに設定ファイルを配置します。設定ファイルは以下のような XML ファイルです。

```
<context enable="true">
  <properties>
    <property key="cf.mode" value="FormCast"/>
    <property key="cf.opt.D" value="${cfro.home}/sample/remoteobject/formfiles"/>
    <property key="cf.opt.s" value="sheet"/>
  </properties>
</context>
```

「/context/properties/property」要素の「key」属性に指定するプロパティ名を、「value」属性には指定するプロパティ値を記述します。
上記ファイル例のように「value」に「${cfro.home}」と記述するとその部分は RemoteObject 導入ディレクトリのパスに展開されます。

※プリセットコンテキストファイルのサンプルは、FormRemoteObject 導入ディレクトリ内の「context¥sample.xml」、プリセットコンテキストを利用したコードサンプルは、「sample¥remoteobject¥Sample.java」として配置されています。

6-1. プロパティー覧

設定ファイルで指定可能な主なプロパティには以下のものがあります。

cf.mode（実行モード）
実行モード（FormCast、FormCollect、FormPrint、FormPrintStage、FormPrintStage Web）を表します。

cf.opt.D（作業ディレクトリ）
作業ディレクトリ指定オプションの内容を表します。

cf.opt.s（ジョブファイル名）
ジョブファイル名指定オプションの内容を表します。

cf.opt.#（プリンタ）
出力プリンタ名指定オプションの内容を表します。

cf.opt.nc（部数）
印刷部数指定オプションの内容を表します。

cf.opt.q（QDF ファイル名）
QDF ファイル指定オプションの内容を表します。

cf.opt.j（スプールファイル名）
スプールファイル名指定オプションの内容を表します。

cf.opts（拡張オプション）
拡張実行オプション文字列の内容を表します。
例）「-pse3 -P」

その他、Create!Form で使用可能な実行オプションは全て「cf.opt.＜オプションレター＞」形式で指定可能です。例えば､PDF セキュリティのマスターパスワードを指定する｢-Xm｣オプションを指定する場合は「cf.opt.Xm」となります。

7. エラー発生時

帳票出力ジョブの実行でエラーが発生すると、サーバ側では以下の処理が行われます。

1. エラーログをサービス稼動ログに出力する
2. エラー情報をエラーログに出力する

サービス稼動ログには以下のようにエラーログが出力されます。

```
2013/03/15,10:00:00,ERROR,13d1f952a1b,0,0,LICENSE invalid.
```

エラーログにはさらに詳細な情報が出力されます。

8. コーディング例

FormRemoteObject API を使用したコーディング例を紹介します。ここで紹介しているコーディング例は、FormRemoteObject 導入ディレクトリ内の「sample¥remoteobject¥api」に配置されています。

1. FormCast を実行する (Windows)

```
import java.io.*;
import net.createform.ro.*;                                          // (1)

public class Sample {
  public static void main(String[] args) throws Exception {
    CreateForm server = CreateForm.getInstance("server", 56789);// (2)
    if (!server.isAvailable()) {                                     // (3)
      return;
    }
    Job job = server.newJob();                                       // (4)
    job.setMode(CreateFormMode.CAST);                                // (5)
    job.addOpt("D", "C:¥¥serverpath¥¥to¥¥work-directory");           // (6)
    job.addOpt("s", "sheet.sty");                                    // (7)
    job.getSource().addFile("C:¥¥clientpath¥¥to¥¥sheet_p1.csv");// (8)
    JobResult result = job.execute();                                // (9)
    if (result.getErrorCode() != 0) {                                // (10)
      return;
    }
    OutputStream out = new FileOutputStream("result.pdf");
    result.publish(out);                                             // (11)
    out.close();
  }
}
```

(1) クライアント API のパッケージは「net.createform.ro」です。
(2) サーバへ接続します。
(3) isAvailable メソッドにより接続の確認を行います。
(4) newJob メソッドにより新しいジョブを作成します。
(5) 実行モードを FormCast に指定します。
(6) addOpt により「-D」オプションを指定します。(サーバ側)
(7) addOpt により「-s」オプションを指定します。
(8) 入力データファイルを指定します。(クライアント側)
(9) ジョブを実行し結果を result に受け取ります。
(10) 結果コードを確認します。
(11) 出力結果を OutputStream へ書き出します。

2. FormCollect を実行する (Linux)

```
import java.io.*;
import net.createform.ro.*;                                          // (1)

public class Sample {
  public static void main(String[] args) throws Exception {
    CreateForm server = CreateForm.getInstance("server", 56789);// (2)
    if (!server.isAvailable()) {                                     // (3)
      return;
    }
    Job job = server.newJob();                                       // (4)
    job.setMode(CreateFormMode.COLLECT);                             // (5)
    job.addOpt("D", "/serverpath/to/work-directory");                // (6)
    job.addOpt("s", "sheet.sty");                                    // (7)
    job.getSource().addFile("/clientpath/to/sheet_p1.csv");          // (8)
    JobResult result = job.execute();                                // (9)
    if (result.getErrorCode() != 0) {                                // (10)
      return;
    }
    OutputStream out = new FileOutputStream("result.pdf");
    result.publish(out);                                             // (11)
    out.close();
  }
}
```

(1) クライアント API のパッケージは「net.createform.ro」です。
(2) サーバへ接続します。
(3) isAvailable メソッドにより接続の確認を行います。
(4) newJob メソッドにより新しいジョブを作成します。
(5) 実行モードを FormCollect に指定します。
(6) addOpt により「-D」オプションを指定します。(サーバ側)
(7) addOpt により「-s」オプションを指定します。
(8) 入力データファイルを指定します。(クライアント側)
(9) ジョブを実行し結果を result に受け取ります。
(10) 結果コードを確認します。
(11) 出力結果を OutputStream へ書き出します。

3. FormPrintStage を非同期実行する (Windows)

```
import net.createform.ro.*;

public class Sample {
  public static void main(String[] args) throws Exception {
    Sample stage = new Sample();
    String jobId = stage.postJob();
    if (jobId.equals("")) {
      // エラー処理
    }
    stage.getJob(jobId);
  }

  /**
   * 帳票出力ジョブを非同期実行します
   */
  private String postJob() throws RemoteObjectException {
    CreateForm server = CreateForm.getInstance("server", 56789);
    String jobId = "";
    if (!server.isAvailable()) {
      return jobId;
    }
    Job job = server.newJob();
    jobId = job.getId();                                          // (1)
    job.setMode(CreateFormMode.PRINTSTAGE);
    job.addOpt("D", "C:¥¥serverpath¥¥to¥¥work-directory");
    job.addOpt("s", "sheet.sty");
    job.addOpt("#", "PRT2");
    job.getSource().addFile("C:¥¥clientpath¥¥to¥¥sheet_p1.csv");
    job.post();                                                   // (2)
    return jobId;
  }

  /**
   * 帳票出力ジョブを取得して実行結果を確認します
   */
  private void getJob(String jobId) throws RemoteObjectException {
    CreateForm server = CreateForm.getInstance("server", 56789);
    if (!server.isAvailable()) {
      return;
    }
    Job job = server.lookupJob(jobId);                            // (3)
    JobStatus status = job.getStatus();                           // (4)
    if (status.done() || status.fail()) {                         // (5)
      JobResult result = job.getResult();                         // (6)
    }
    else {
```

```
        JobResult result = job.waitResult();                    // (7)
      }
    }
  }
```

(1) ジョブ ID を取得します。
(2) 非同期で実行します。
(3) (1) で取得したジョブ ID を使ってジョブを再取得します。
(4) JobStatus オブジェクトを取得します。
(5) ジョブの実行が終了しているかどうかを確認します。
(6) 終了している場合は getResult で結果を取得できます。
(7) waitResult を使うとジョブが終了するまで待ちます。

4. ジョブの実行オプションを変更して再実行する

```
import net.createform.ro.*;

public class Sample {
  public static void main(String[] args) throws Exception {
    CreateForm server = CreateForm.getInstance("server", 56789);
    Job job = server.lookupJob("123abc45def");                  // (1)
    job.execute();                                              // (2)

    job.addOpt("#", "PRINTER NAME");                            // (3)
    job.addOpt("pse", "3");
    job.setExpiration(30);                                      // (4)
    job.execute();
  }
}
```

(1) ジョブ ID を指定して再実行するジョブを取得します。
(2) Job オブジェクトの execute メソッドもしくは post メソッドを使用して再実行します。
(3) 実行オプションの値を変更します。プリンタ名は直接プリンタ名を指定することができます。なお、既に指定されている実行オプションを再指定した場合、実行オプションの値は上書きされます。
(4) ジョブの有効期限を 30 日間にします。

5. ジョブの一覧を取得する

```
import java.util.*;

import net.createform.ro.*;

public class Sample {
  public static void main(String[] args) throws Exception {
    CreateForm server = CreateForm.getInstance("server", 56789);
    List<JobDescriptor> descriptors = server.listJobs();        // (1)
    for (JobDescriptor descriptor : descriptors) {
      System.out.println("id: " + descriptor.getId());          // (2)
      System.out.println("times: " + descriptor.getTimes());    // (3)

      Job job = server.lookupJob(descriptor.getId());           // (4)
      job.remove();                                             // (5)
    }
  }
}
```

(1) ジョブの一覧を取得します。
(2) JobDescriptor オブジェクトからジョブ ID を取得して標準出力へ表示します。
(3) JobDescriptor オブジェクトからジョブの実行回数を取得して標準出力へ表示します。
(4) ジョブ ID を指定してジョブを取得します。
(5) ジョブを削除します。

9. FormMagicfolder/FormRemoteObject 連携

9-1. 概要

FormMagicfolder と FormRemoteObject は連携して動作させることが可能です。両製品を連携させることで以下のような仕組みを実現することができます。

・FormMagicfolder で処理したジョブの履歴リストを FormRemoteObject で取得する。
・FormMagicfolder で処理したジョブを FormRemoteObject で再実行（再印刷）する。

なお、連携にはデータ保存期間とストレージパスの設定が必要です。以下では、連携方法についての詳細を説明します。

データ保存期間

FormMagicfolder および FormRemoteObject のジョブのデータは、指定された「保存期間」に従ってストレージに保存されます。ストレージに保存された各ジョブは、この保存期間中は FormRemoteObject API によってアクセスすることが可能になります。

両製品では、初期設定としてジョブの保存期間は「0」になっており、ジョブは保存されません。ジョブを保存するには、保存期間の設定を変更する必要があります。変更方法については、「9-2. 準備」で説明します。

ストレージ

FormMagicfolder と FormRemoteObject のいずれも、それぞれローカルストレージ（データベース）を内部で保持し、ジョブの処理データをストレージに保存することができます。両製品ではストレージを個別に保持していますが、一つのストレージを共有することもできます。ストレージを共有することで、FormMagicfolder と FormRemoteObject でジョブデータを共有することができるようになります。

FormMagicfolder と FormRemoteObject を同一マシンかつ同一パスにインストールした場合は、初期設定により既にストレージは共有されています。そのため、特にストレージパスの設定を変更する必要はありません。FormMagicfolder と FormRemoteObject が別のマシンまたは別のパスにインストールされている場合は、設定を変更する必要があります。変更方法については、「9-2. 準備」で説明します。

9-2. 準備

FormMagicfolder の設定

保存期間
FormMagicfolder では初期設定としてジョブの保存期間は「0」になっています。
FormRemoteObject から FormMagicfolder のジョブを参照可能にするためにはこの保存期間を変更しておく必要があります。

FormMagicfolder の設定は導入ディレクトリ内の「conf¥cfmf-server.properties」を編集す

ることで行います。「cfmf-server.properties」をテキストエディタで開き、以下の 1 行を追加してください。

```
job.available = 30
```

ここで指定している「30」は「30 日間ジョブデータを保存する」ということを意味しています。つまりこの設定では、FormMagicfolder の監視フォルダにデータファイルを投入してから 30 日間は FormRemoteObject API で参照可能となり、31 日目以降はストレージから削除されデータは参照不可能になります。

データを自動削除せずに恒久的に保存しておきたいという場合は、「job.available = -1」を指定してください。その場合は、FormRemoteObject API で明示的に削除指示を出さない限り削除されません。

ストレージパス

FormMagicfolder では初期設定として、ストレージパスは導入ディレクトリ内の「var¥storage¥01」になっています。FormMagicfolder と FormRemoteObject を別のマシンや別のディレクトリにインストールした場合は、このストレージパスの設定を両製品で同一のものとなるように変更する必要があります。

「cfmf-server.properties」をテキストエディタで開き、以下の 1 行を追加してください。

```
storage.path = C:/path/to/storage
```

「C:/path/to/storage」の部分をストレージパスとして指定したいパスに設定してください。

```
<< 注意 >>
ストレージパスが不正なパスの場合、FormMagicfolder は正常に動作しません。
```

また、ここではパスセパレータ（区切り文字）として「/」を使用していますが、「C:¥path¥to¥storage」のように「¥」を使用すると動作しません。「/」もしくは「¥¥」を使用してください。

ネットワーク上のディスクを指定する場合は、UNC パスで指定してください。

```
storage.path = //other_server/path/to/storage
```

「Z:/path/to/storage」のようにドライブレターを割り当てた形式ではアクセスできませんのでご注意ください。また、「FormMagicfolder サービスの起動ユーザ」からアクセス可能となるように権限設定を確認してください。

FormRemoteObject の設定

保存期間

FormRemoteObject では初期設定としてジョブの保存期間は「0」になっています。ただし、これは「FormRemoteObject 経由で生成されるジョブの保存期間」ですので、FormMagicfolder で生成されたジョブを FormRemoteObject で実行する、という場合には保存期間を変更する必要はありません。FormRemoteObject で生成したジョブも再実行可能にするためには、この保存期間を変更しておく必要があります。

FormRemoteObject の設定は導入ディレクトリ内の「conf¥cfro-server.properties」を編集することで行います。「cfro-server.properties」をテキストエディタで開き、以下の 1 行を追加してください。

```
job.available = 30
```

ここで指定している「30」は「30 日間ジョブデータを保存する」ということを意味しています。つまりこの設定では、FormRemoteObject でジョブを実行してから 30 日間は FormRemoteObject API で参照可能となり、31 日目以降はストレージから削除されデータは参照不可能になります。

データを自動削除せずに恒久的に保存しておきたいという場合は、「job.available = -1」を指定してください。その場合は、FormRemoteObject API で明示的に削除指示を出さない限り削除されません。

ストレージパス

FormRemoteObject では初期設定としてストレージパスは導入ディレクトリ内の「var¥storage¥01」になっています。FormMagicfolder と FormRemoteObject を別のマシンや別のディレクトリにインストールした場合は、このストレージパスの設定を両製品で同一のものとなるように変更する必要があります。

「cfro-server.properties」をテキストエディタで開き以下の 1 行を追加してください。

```
storage.path = C:/path/to/storage
```

「C:/path/to/storage」の部分をストレージパスとして指定したいパスに設定してください。

<< 注意 >>
ストレージパスが不正なパスの場合、FormRemoteObject は正常に動作しません。

また、ここではパスセパレータ（区切り文字）として「/」を使用していますが、「C:¥path¥to¥storage」のように「¥」を使用すると動作しません。「/」もしくは「¥¥」を使用してください。

ネットワーク上のディスクを指定する場合は、UNC パスで指定してください。

```
storage.path = //other_server/path/to/storage
```

「Z:/path/to/storage」のようにドライブレターを割り当てた形式ではアクセスできませんのでご注意ください。また、「FormRemoteObject サービスの起動ユーザ」からアクセス可能となるように権限設定を確認してください。

サービスの再起動

FormMagicfolder および FormRemoteObject の設定を変更した場合は、必ず各サービスを再起動してください。サービスを再起動するまで設定は反映されません。サービスの再起動方法については、以下のマニュアルをご覧ください。

FormMagicfolder -「4. サービスの開始・停止」
FormRemoteObject -「3. サービスの開始・停止」

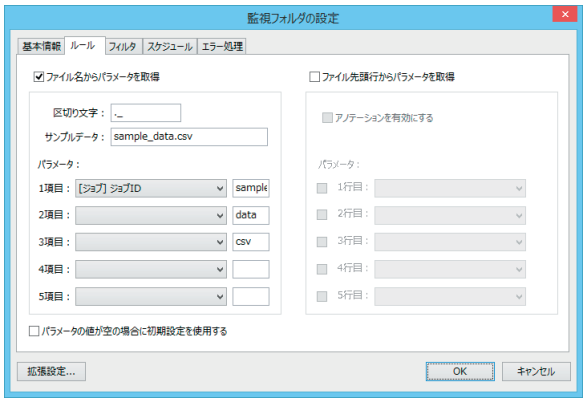
9-3. 連携

ジョブの生成と再実行

連携のための特殊な操作やAPIは必要ありません。前述の連携の準備の設定が適切にされていれば、FormMagicfolderで実行したジョブデータがFormRemoteObjectのAPIで取得できるようになります。

まず、FormMagicfolderで任意のジョブIDを割り当てるための設定を行います。FormMagicfolderの「パラメータ取得ルール」機能を使用し、ファイル名の「1項目」を「ジョブID」に設定します。

図：ルール設定



この設定により、ファイル名の「1項目」が「ジョブID」として認識されます。例えば、「abc123_data.csv」というデータファイルを監視フォルダへ投入した場合、「abc123」としてジョブIDが割り当てられ、ジョブが生成されます。このジョブIDをFormRemoteObject APIで指定してジョブを取得し、再実行します。

```
CreateForm server = CreateForm.getInstance("localhost", 56789);
Job job = server.lookupJob("abc123"); // ジョブIDが「abc123」のジョブを取得
JobResult result = job.execute(); // ジョブを再実行
```

<< 注意 >>
任意のジョブIDを割り当てずにジョブを生成した場合、ジョブIDに一意な値が自動で割り当てられます。そのため、CreateFormオブジェクトのlookupJobメソッドを使用して直接ジョブIDを指定してジョブを取得することができなくなります。

このように、FormRemoteObject APIで取得したFormMagicfolderのジョブは、そのままFormRemoteObject APIで実行することができます。これは通常のFormRemoteObjectでの再実行の処理と同じです。

「パラメータ取得ルール」機能の詳細については、FormMagicfolder「7. パラメータ取得ルール機能」をご覧ください。

ジョブ一覧の取得

ジョブ ID を指定せず、ジョブ一覧からストレージに保存されているすべてのジョブを再実行することもできます。

```
CreateForm server = CreateForm.getInstance("localhost", 56789);
for (JobDescriptor jobDesc : server.listJobs()) {
    // ジョブを一件ずつ処理
    Job job = server.lookupJob(jobDesc.getId()); // ジョブを取得
    job.execute(); // ジョブを再実行
}
```

ジョブの設定変更と再実行

印刷ページや部数、ランタイム種別など、ジョブの設定を変更して再実行することもできます。

```
CreateForm server = CreateForm.getInstance("localhost", 56789);
Job job = server.lookupJob("abc123"); // ジョブ ID が「abc123」のジョブを取得
job.addOpt("pse", "2"); // 2 ページ目だけを再印刷
job.addOpt("nc", "3"); // 3 部印刷
job.setMode(CreateFormMode.PRINTSTAGE); // FormPrintStage に変更
JobResult result = job.execute(); // ジョブを再実行
```

ジョブプロパティ

FormRemoteObject にはジョブプロパティという機能があり、これを利用することで FormMagicfolder と FormRemoteObject で値の受け渡しが可能です。

例えば、「mf.directory」というジョブプロパティには、そのジョブが生成された監視フォルダのパスが格納されています。FormRemoteObject API でこの値にアクセスすることができます。

```
String path = job.getProperty("mf.directory");
```

これにより、監視フォルダが複数ある場合も、そのジョブがどの監視フォルダから生成されたものであるかを API により判別することができます。それ以外にも、アクセス可能なジョブプロパティがいくつか存在します。ジョブプロパティの詳細については、FormMagicfolder「11. ジョブプロパティ」をご覧ください。

フリーパラメータ

FormMagicfolder でアクセス可能なジョブプロパティの中に「user.1」～「user.5」という「フリーパラメータ」があります。このフリーパラメータは、自由に値を格納して使用することができます。これを活用すると、ジョブに固有の ID を埋め込んで FormRemoteObject API で追跡するようなことも実現できます。

例えば、監視フォルダの「ルール」設定で下記のような設定にしておきます。

- ファイル名から取得
- 区切り文字：「_.」
- 1 項目： フリーパラメータ 1 (user.1)

この監視フォルダに「abc123_sheet.csv」のようなファイル名でデータを投入します。すると「abc123」の部分が「user.1」に割り当てられます。この値はFormRemoteObject APIで以下のように取得することができます。

```
String value = job.getProperty("user.1");
//=> "abc123"
```

さらにFormMagicfolderのアノテーション機能を利用すると、独自のジョブプロパティを定義することもできます。例えば、「your.original.property」というジョブプロパティは存在しないため、通常はFormMagicfolderのルール設定で定義することはできません。また、FormRemoteObject APIで取得しようとしても「null」が返されます。

```
String value = job.getProperty("your.original.property");
//=> null
```

しかし、アノテーションとして入力データファイルに記述するとそのジョブプロパティが有効になります。

```
<cf:mf-annotation>
your.original.property = ABC123
</cf:mf-annotation>
```

このように、アノテーション機能を利用してジョブプロパティを記述することで、FormRemoteObject APIで独自のジョブプロパティを取得できます。

```
String value = job.getProperty("your.original.property");
//=> "ABC123"
```

このように、フリーパラメータやアノテーション機能を利用することによって、システムで使用する識別コードなどの独自のメタデータをジョブに埋め込み、システム連携のパラメータとして活用することができます。

10. ストレージの移行

10-1. 概要

サーバマシンの入れ替えなどにより、Create!Form 製品を再インストールする必要があり、なおかつ移行元で実行したジョブを移行先でも再実行するような場合、ストレージの移行が必要になります。（サーバマシンはそのままで、製品を別のディレクトリへ再インストールする場合も該当します）

ストレージを移行することで、移行元のジョブを移行先でも利用できるようになります。ただし、ストレージを移行する場合、移行先のストレージのデータはすべて削除する必要がありますので、注意してください。

以下に、移行手順を説明します。

10-2. 移行手順

1. サービスの停止

ストレージを移行する前に、移行元と移行先の FormMagicfolder サービス、FormRemoteObject サービスを停止してください。サービスが稼動した状態ではストレージを正しく移行することができません。また、ストレージの移行中はサービスを稼動させないでください。

サービスの停止の詳細については、以下のマニュアルをご覧ください。

FormMagicfolder -「4. サービスの開始・停止」
FormRemoteObject -「3. サービスの開始・停止」

2. ストレージパスの確認

移行元の FormRemoteObject 導入ディレクトリ内の「conf¥cfro-server.properties」をテキストエディタで開き、ストレージパス「storage.path」を確認します。ストレージパスが設定されている場合は、そのストレージパスを使用して移行を進めます。ストレージパスが未設定の場合は、導入ディレクトリ内の「var¥storage¥01」をストレージパスとして移行を進めます。また、同様の手順で、移行先のストレージパスも確認してください。

※FormRemoteObject のストレージパスは初期設定として、導入ディレクトリ内の「var¥storage¥01」に設定されています。例えば、FormRemoteObject を「C:¥CreateV10」に導入している場合、ストレージパスは「C:¥CreateV10¥var¥storage¥01」となります。

3. 移行先のストレージの退避

移行前に、移行先に存在するストレージを以下の手順に従って予めすべて退避します。

Windows 環境

Windows のコマンドプロンプトを起動し、以下の「move」コマンドを実行します。「move」コマンド内の「＜ストレージパス＞」は移行先のストレージパスを指定してください。ここで、誤っ

て移行元のストレージパスを指定しないように注意してください。

```
move ＜ストレージパス＞＜ストレージパス＞_bak
```

移行先のストレージパスが「C:¥CreateV10¥var¥storage¥01」の場合、以下のように指定して実行します。

（移行先での操作）
```
move C:¥CreateV10¥var¥storage¥01 C:¥CreateV10¥var¥storage¥01_bak
```

実行後、ストレージが「01_bak」としてリネームされ、退避されます。なお、ストレージの移行後、退避した「01_bak」が不要な場合は削除してください。

Linux 環境

Linux ターミナルより、以下の「mv」コマンドを実行します。「mv」コマンド内の「＜ストレージパス＞」は移行先のストレージパスを指定してください。ここで、誤って移行元のストレージパスを指定しないように注意してください。

（移行先での操作）
```
$ mv ＜ストレージパス＞＜ストレージパス＞_bak
```

実行後、ストレージが「01_bak」としてリネームされ、退避されます。なお、ストレージの移行後、退避した「01_bak」が不要な場合は削除してください。

4. ストレージのコピー

Windows 環境

Windows のコマンドプロンプトを起動し、以下の「xcopy」コマンドを実行します。「xcopy」コマンドは、ストレージパス内のファイルとディレクトリ構造をそのままコピーするための Windows 標準のコマンドです。「xcopy」コマンド内の「＜ストレージパス＞」は移行元のストレージパスを指定し、「＜コピー先＞」は移行先のストレージパスまたは任意の一時ディレクトリへのパスを指定します。

```
xcopy /E /V /I /F /H /K /X /Y ＜ストレージパス＞＜コピー先＞
```

既に移行先の環境へ FormRemoteObject 製品が導入されており、移行元から移行先のストレージパスが参照可能な場合は、移行先のストレージパスへ直接ストレージをコピーすることができます。例えば、移行元のストレージパスは「C:¥CreateV10¥var¥storage¥01」、移行先のストレージパスも「C:¥CreateV10¥var¥storage¥01」、移行先のマシン名が「servername」であり、「C:¥CreateV10」が共有ディレクトリとして移行元から参照可能な場合、以下のように指定して実行します。

（移行元での操作）
```
xcopy /E /V /I /F /H /K /X /Y
 C:¥CreateV10¥var¥storage¥01 ¥¥servername¥var¥storage¥01
```

※ 2 行に分けて記述していますが、実際は「xcopy」コマンドは 1 行で実行してください。

このように実行することで、移行元のストレージパス「C:¥CreateV10¥var¥storage¥01」から移行先のマシンにあるストレージパス「C:¥CreateV10¥var¥storage¥01」へストレージがコピーされます。

移行元から移行先へ直接コピーではなく、任意の一時ディレクトリへストレージをコピーした場合は、コピー後に移行先のストレージパスへ再び「xcopy」コマンドを実行してコピーしてください。例えば、一時ディレクトリを同一ネットワーク上の共有ディレクトリ「¥¥serveername¥temp」とした場合、以下のように指定して実行します。

（移行元での操作）
```
xcopy /E /V /I /F /H /K /X /Y
 C:¥CreateV10¥var¥storage¥01 ¥¥servername¥temp
```

（移行先での操作）
```
xcopy /E /V /I /F /H /K /X /Y
 ¥¥servername¥temp C:¥CreateV10¥var¥storage¥01
```

※2行に分けて記述していますが、実際は「xcopy」コマンドは1行で実行してください。

このように実行することで、移行元のストレージパス「C:¥CreateV10¥var¥storage¥01」から一時ディレクトリへストレージがコピーされ、その後、一時ディレクトリから移行先のマシンにあるストレージパス「C:¥CreateV10¥var¥storage¥01」へストレージがコピーされます。

<< 注意 >>
「xcopy」コマンドを使用せず、通常の「copy」コマンドや、エクスプローラでコピーを行うと、ストレージが正しくコピーされません。

Linux 環境

Linux 環境ではストレージを tar アーカイブにまとめてからストレージのコピーを行います。Linux ターミナルより、ストレージパスへ移動し、以下の「tar」コマンドを実行します。「tar」コマンドは、ストレージパス内のファイルとディレクトリ構造をそのまま1つのファイルにまとめるための Linux 標準のコマンドです。「tar」コマンドによってストレージを1つのファイルにまとめた後は、その tar アーカイブを移行先のストレージパスへ配置し、tar アーカイブを展開します。

```
$ tar cvf storage.tar v2
```

例えば、移行元のストレージパスが「/opt/createv10/var/storage/01」、移行先のストレージパスが「/opt/createv10/var/storage/01」の場合、以下のように指定して実行します。

（移行元での操作）
```
$ cd /opt/createv10/var/storage/01
$ tar cvf storage.tar v2
```

tar アーカイブ「storage.tar」が移行元のストレージパス内に作成されますので、FTP などを使用して移行先のストレージパスへ配置します。配置後、移行先で tar アーカイブを展開します。

（移行先での操作）

```
$ cd /opt/createv10/var/storage/01
$ tar xvf storage.tar
```

5. 作業ディレクトリのコピー

Create!Form ランタイムの実行に必要な作業ディレクトリを移行元から移行先へコピーします。移行元と移行先で作業ディレクトリを配置するパスが異なる場合、そのままでは移行先でジョブの再実行を行うことができません。ジョブの再実行を行うためには、作業ディレクトリのパスの再設定が必要になります。詳しくは、「10-3. 作業ディレクトリのパスの再設定」をご覧ください。

10-3. 作業ディレクトリのパスの再設定

ストレージを移行するにあたり、FormRemoteObject API で実行したジョブに設定されている作業ディレクトリ（ジョブプロパティ「cf.opt.D」）が移行先に存在しない場合、移行先でジョブを参照することはできますが、ジョブの再実行を行うことはできません。これは、ジョブの再実行の際に、ジョブに設定されている作業ディレクトリのパスを参照しているためです。ジョブの再実行を行うためには、移行先に存在する作業ディレクトリのパスを再設定する必要があります。

※移行元と移行先の作業ディレクトリのパスが同一の場合は、再設定は必要ありません。

作業ディレクトリのパスの再設定は、FormRemoteObject API の Job オブジェクトの removeOpt と addOpt メソッドを使用して行います。例えば、移行元で作業ディレクトリを「C:/path/to/formfiles」に配置しており、移行先では「D:/change/path/to/formfiles」に配置している場合、以下のように作業ディレクトリを再設定して実行します。

```
Job job = server.lookupJob(jobId);
job.removeOpt("D");
job.addOpt("D", "D:/change/path/to/formfiles");
JobResult result = job.execute();
```

Create!Form

FormRemoteObject 第 5 版

発行日 2015 年 2 月
発行者 インフォテック株式会社
〒 160-0023 東京都新宿区西新宿 7-5-25