コンポジット・アプリケーション連載: 第1回 QRコードを活用しよう (前編)

森谷 直哉
ソフトウェア事業 Lotusテクニカルセールス
IBM Japan

2007年 12月 21日

第0回ではコンポジット・アプリケーションを学ぶ上で有用な背景知識を整理しました。今回からはプログラムコードを交えて、実際のアプリケーション開発のステップの例を紹介していきます。第1回から前編・後編の2回にわたって、最近広告や名刺などでよく目にするQRコード® を利用したコンポーネントの開発を扱います。

このシリーズの他の記事を見る

なぜQRコード®なのか？

QRコードをご存知ない方のために、少し事前に解説しておきたいと思います。QRコードとは2次元コード（バーコード）の一種です。例えば、右のQRコードをお持ちの携帯電話のリーダーで読み込んでみてください。IBM developerWorks JapanのURLが取得できるはずです。 2次元であることにより、従来の1次元のいわゆるバーコードよりも格段に多くの情報を小さな面積に表現することが可能となりました。2次元コードにはいくつも規格が存在しますが、QRコードはその中でも広く普及しており、皆さんがお持ちの多くの携帯電話でもサポートしています。QRコードをデバイスのカメラで認識することにより、URLやメールアドレスのようなちょっとした情報を簡単に取り込むことができます。携帯電話にならメールでいいのでは？と思うかもしれませんが、スパムメール対策のため知らないドメインからのメールの受信には手間がかかったりします。また、携帯電話にメールを届けるためには、インターネットという言わば公共の（必ずしもセキュアとはいえない）インフラを介すことになります。その点、紙面、広告、スクリーン上から直接デバイスに情報を転送できるQRコードを利用する利点は手軽さ以上にあるのではないかと思います。

そこで、今回は外部から受けたデータをQRコードとして表示するコンポーネントの開発に取り組んでみたいと思います。こうしたコンポーネントにどのような情報を渡して活用するかはアイディア次第です。例えば、ウェブブラウザの現在のURLをいつでも転送できるようにする、アドレス帳のデータを転送する、といった利用例が考えられるでしょう。

対象プラットフォームは、クライアント・アプリケーションの基盤であるLotus Expeditorとし、開発環境としてはEclipse3.2.2 + Lotus Expeditor Toolkit 6.1.1を使います。完成したコンポーネントは他にもLotus Notes 8.xでも使えるはずです。

再利用可能なコードを見つけよう

コンポジット・アプリケーションを利用する一つの大きな目的は既存コードの再利用です。つまり、書かなくてよいコードは書かないで済ますことが大事であり、生産性の向上につながります。

インターネット上で公開されているQRコードに関する情報を検索したところ、 Kazuhiko Araseさんが公開されている "QRコードライブラリ for Java"が見つかりました。こちらをベースに、コンポジット・アプリケーションの部品として利用できるコンポーネントを作ってみることとします。

中身を確認しよう

既存の資産を利用する際、その事前調査がとても大事です。自分で開発したコードではないですからその中身については分かりません。提供されているドキュメントやコードから的確な事前分析を行うことから始まります。こうすることにより、そのライブラリを活かした適切な実装が見えてくるはずです。

ライセンス

実はコードの中身より大事な調査事項があります。それはライセンスです。特に企業での利用を検討する場合、ライセンス形態を十分に理解せずにコードの再利用した結果、思わぬ結果（システム全体のコードを公開しなければならない、等）を引き起こすことがあります。オープンソース・ソフトウェアを利用する際での1つの肝ですので十分にご注意ください。幸い、添付のREADME.txtを読んだ限り、"QRコードライブラリ for Java" を使うことに問題はなさそうです。ライブラリを見つけた際に事前にその提供者にメールなどでコンタクトしてみるのもよいでしょう。

ライブラリの構成・仕様・稼動環境など

続いて、ライブラリの中身です。ドキュメント等から以下のことがわかりました。

- JavaSEの標準GUIライブラリであるAWTを使ってグラフィック処理を実装している
- 必要なクラスファイルをまとめた.jarファイルが提供されている
- Webコンテナ上ですぐに使えるWebアプリケーション(.war)としても提供されている。 つまり、サーバーさえあれば、ブラウザベースのユーザー・インターフェースが用意されている。
- 上記のため、JavaSEのクラスライブラリだけでなく、JavaEEのJSP/Servlet APIにも依存しているクラスもある。
- WebアプリケーションのサーブレットはHTTPのGETメソッドでパラメータを渡すことにより画像（GIF,JPG等）を返してくれる。
- ライブラリ公開サイトのリンクをたどると、このWebアプリケーションを実際に動かしているサーバーがインターネット上にある。

いろいろと分かりましたね。これらの情報を踏まえて、コンポーネントの設計をしていきます。

設計のコツ

コンポーネントの設計といっても、どのような機能をどのような単位に切り分けるとよいのか、はじめは戸惑うかもしれません。筆者はよく以下のように分類し、再利用される可能性のある単位でプラグインとしてまとめています。

- 見える
  - アプリケーション画面全体（画面レイアウト）
  - コンポーネント（画面構成要素）
- 見えない
  - ライブラリ
  - サービス
    - ローカル（実行環境上）
    - リモート（ネットワーク上）

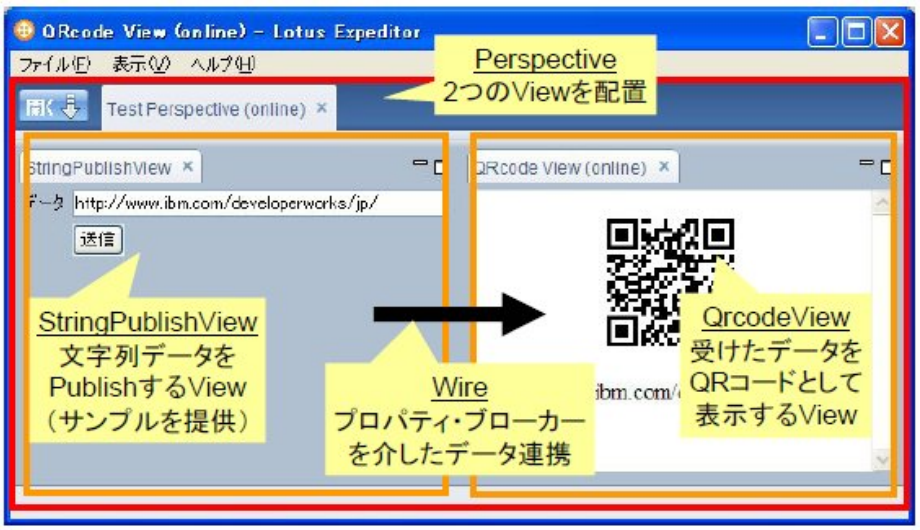
コンポーネントとコンポーネントとの間の連携を支援するためのプロパティ･ブローカーは上記の見える・見えないの間に位置づけて整理すると分かりやすいのではないでしょうか。後ほどこの分類に基づいて機能とその配置を整理していきたいと思います。

ステップ１：既存Webサイトをラップする

ライブラリを使わなくとも、この機能を提供しているインターネット上のサイトがありました。幸い、Eclipseが提供するGUIライブラリSWTにはBrowserウィジェット(Windows環境であればデフォルトでインタネット・エクスプローラをラップ）が提供されています。ウィジェットにはブラウザを操作するための各種APIも用意されています。そこで、ステップ１としては"既存のWebサイトをラップする形でコンポーネント化”したいと思います。

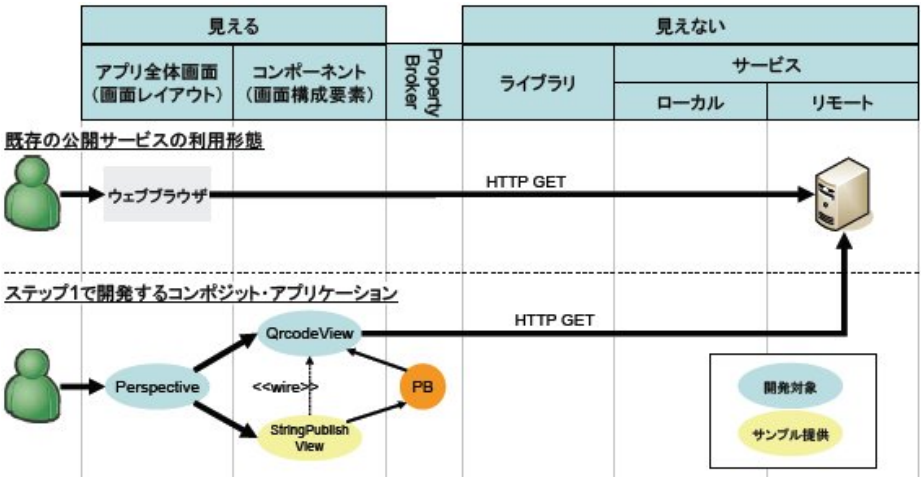
以下、完成イメージです。開発対象のコンポーネントQrcodeView（右側）はワイヤ定義に基き、他のコンポーネント（左側は後述のサンプル・コンポーネント）が送出したデータをプロパティ・ブローカー経由で受け取ります。そしてそのデータから生成されたQRコードを表示します。

図. ステップ１ 完成イメージ



そして、前述の考え方のもとに部品の配置箇所を簡単に描いたのが次の図になります

図. 開発コードの役割と配置



開発するプラグインは以下の2つとなります。

1. dw.japan.ca_series.components.qrcode
   QrcodeViewというViewを提供するコンポーネントのプラグイン。
2. dw.japan.ca_series.compositeapps.qrcode
   Perspective(Eclipse RCPにおいて画面レイアウトを司るクラス）とワイヤ情報を持つコンポジット・アプリケーションのプラグイン。このPerspective上に上記Viewと次に説明するテスト用のViewを配します。 また、2つのコンポーネント間の連携も定義します。

テスト用のコンポーネント

QrcodeViewと名づけたViewの開発がステップ１のメインとなります。ただ、コンポーネントが1つあっただけでは外部との連携を試すことができません。そこで、今回は実装済みのコンポーネントを使用してコンポーネント間の連携をテストを行いたいと思います。このコンポーネントの開発の詳細については特に触れません。興味のある方はコードを参照ください。

ここに機能概要だけまとめておきます。

表. テスト用コンポーネント概要

機能概要		ボタンをクリックするとテキストフィールドの値を文字列型のプロパティ"data"としてプロパティ・ブローカーに投げる
実装技術		SWT
View	id	dw.japan.ca_series.components.pbtest.StringPublishView

	名前	StringPublishView
プロパティ	名前	data
	名前空間	http://www.w3.org/2001/XMLSchema
	データ型	string
パッケージング	プラグイン	dw.japan.ca_series.components.pbtest
	フィーチャー	dw.japan.ca_series.components.pbtest.feature

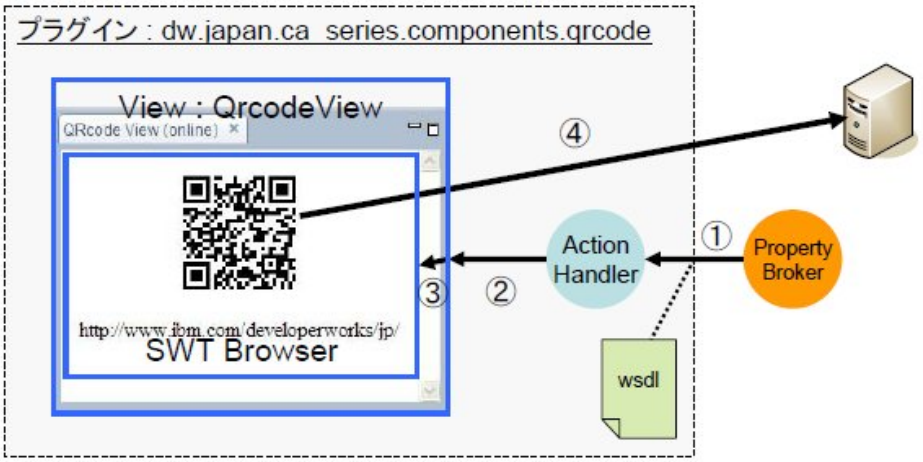
このような汎用的なコンポーネントはコンポジット・アプリケーション開発のテストで重宝します。 Lotus Notes 8.xをお使いの方であればdeveloperWorks USで公開されている"Composite Application Component Library"中にもそのようなツールがいくつかありますので是非一度参照ください。

今回は、ダウンロードいただけるprojects.zipから上記プラグインとフィーチャーをインポートしてお使いください。

Qrcodeコンポーネントの開発の流れ

いよいよQrcodeViewの開発です。まずはコンポーネントがプロパティ・ブローカーからデータを受信した時の処理の流れを確認しましょう。

図. 処理の流れ



1. コンポーネントがプロパティ・ブローカーに呼ばれます。コンポーネントのインターフェースはwsdlファイルにて定義してあり、ツール/フレームワークはその定義に合致した呼び出しのみをつなぎます。今回の例では、QRコードとして表現したいデータ文字列のみを受けられるようにします。
2. ActionHandlerは受けたデータを元に、Viewに働きかけます。
3. Viewの中にはSWTのBrowserウィジェットが埋め込まれています。 データを受けたViewは適切なHTMLをウィジェットにセットします。
4. ブラウザが受けたHTMLコンテンツにはimgタグでQRコード画像のURLが指定してあります。ブラウザは指定されたサイトにHTTPでリクエストを送り、返された画像を表示します。

このような機能を提供するコンポーネントをプラグインdw.japan.ca_series.components.qrcodeとして開発します。手順は以下のようになります。

1. コンポーネント プラグインのための新規"Client Servicesプロジェクト"の作成
2. ウィジェットBrowser(org.eclipse.swt.browser.Browser)を埋め込んだView(dw.ca.sample.qrcode.BrowserView)の作成
3. 上記Viewに、受けたパラメータからQRコード画像のURLを生成してブラウザにセットするメソッドを追加
4. コンポーネントが提供するインターフェースの定義（ワイヤリング・プロパティ）の作成
5. プロパティ・ブローカーからイベントを受けたときに呼ばれ、処理を実行する ActionHandler の作成
6. 上記2,4,5で作成したものをつなぎ合わせるプラグイン定義ファイル(plugin.xml/MANIFEST.MF)の記述追加
7. Eclipseの世界での配布単位はプラグインではなく、それをまとめたフィーチャーなので、このプラグインのための新規フィーチャープロジェクトの作成とプラグインの登録

ツールをはじめて使われる方は "IBM Lotus Expeditor 技術情報 - Japan" を参照ください。本連載の中ではエッセンスの部分の解説にフォーカスしたいと思います。サンプルとしてインストール可能な完成版とその実装コードを提供しておりますので、あわせて参照ください。

1. 新規プロジェクト作成

プラグインの器となるプロジェクトを作成します。種類としては"Client Services プロジェクト"を選択します。これはLotus Expeditorベースのプラットフォームの開発にあわせて Eclipseのプラグイン開発環境PDEが提供する"プラグイン・プロジェクト"を拡張したものです。プロジェクト名はプラグインのidと一致させておくと分かりやすいでしょう。以下の表を参考に作成ください。特に指定のないところはデフォルト値で結構です。

プロジェクトタイプ	Client Services プロジェクト
プロジェクト名	dw.japan.ca_series.components.qrcode
プラグインID	dw.japan.ca_series.components.qrcode
プラグイン名	dw.japan.ca_series.components.qrcode
バージョン	1.0.0
アクティーベーター	dw.japan.ca_series.components.qrcode.Activator
プラグインにUIを追加	はい（チェック入れる）
テンプレート	基本リッチ・アプリケーション（基本的な要素を生成してくれます）
Javaパッケージ名	dw.japan.ca_series.components.qrcode
アプリケーション・クラス	QrcodeView

2. Viewの追加

前手順でテンプレートからQrcodeViewを生成しましたのでこれを修正していきます。生成されたコードの中にはGroupやButtonといったウィジェットがすでに配置されていますが、ここでは必要ありませんのではじめに削除してしまった方がよいでしょう。新規にクラスを作成する場合にはorg.eclipse.ui.part.ViewPartを継承します。

ここで必要としている画面は非常にシンプルです。以下を参考にViewの矩形領域一杯に1つのBrowserウィジェットを配置しましょう。

開発環境にEclipseが提供するWYSIWYGのGUI開発ツール "Visual Editor"を追加している場合には、コードを書かずにGUI操作だけでこのような画面が作れますので便利です。

```
public static String ID = "dw.japan.ca_series.components.qrcode.QrcodeView";

protected Composite top = null;
protected Browser browser = null;

public void createPartControl(Composite parent) {
 GridLayout gridLayout = new GridLayout();
 top = new Composite(parent, SWT.NONE);
 createBrowser();
 top.setLayout(gridLayout);
}
protected void createBrowser() {
 GridData gridData = new GridData();
 gridData.grabExcessHorizontalSpace = true;
 gridData.grabExcessVerticalSpace = true;
 gridData.horizontalAlignment = GridData.FILL;
 gridData.verticalAlignment = GridData.FILL;
 browser = new Browser(top, SWT.NONE);
 browser.setLayoutData(gridData);
}
```

3. データからブラウザコンテンツを生成・セット

リモートのサービスはHTTPのGETメソッドでパラメータを投げれば画像を返してくれるものでした。指定可能なパラメータについてはHTMLフォームを見ると分かります。以下がその一覧と今回使う値になります。簡略化のため、データのためのパラメータ"d"以外は事前に決めた固定値を使うこととします。

HTTP GETメソッドで送信するパラメータ

パラメータ名	用途	指定可能な値	今回使用する値
o	出力形式	text/plain image/gif image/jpeg image/png	どのブラウザも対応している image/gif
d	データ	文字列	動的に生成
e	誤り訂正レベル	L(7%) M(15%) Q(25%) H(30%)	L。パソコンの画面で表示する場合は劣化の可能性が低いため
t	種別	0～10 0が自動	0
m	余白	0～32	5
s	セルサイズ	1-4	3

この情報を元に下記のようなメソッドをQrcodeViewに追加します。ここでは、画像のURLを直接ブラウザにセットするのではなく、レイアウトの容易性からHTMLコンテンツを生成してその中にimgタグで画像を埋め込んでみました。

```
/**
 * QR#################
 */
```

```
public void setQrcodeData(String data) {
 //###################
 this.browser.setText(this.getHtmlContent(data));
}

/**
 * #####HTML########
 */
public String getHtmlContent(String data) {
 StringBuffer buff = new StringBuffer();
 buff.append("<html>");
 buff.append("<body>");
 buff.append("<center>");
 buff.append("<img ");
 buff.append(" src=\"" + getQrcodeUrl(data, true) + "\"");
 buff.append(" alt=\"" + data + "\"");
 buff.append("/>");
 buff.append("<br/><br/>");
 buff.append("<div>" + data + "</div>");
 buff.append("</center>");
 buff.append("</body></html>");
 return buff.toString();
}

protected String getQrcodeUrlPrefix() {
 //####URL###############
 return "http://XXXXXXXXXXXXX/jsp/qrcode?";
}

private String getQrcodeUrl(String data, boolean encode) {
 return this.getQrcodeUrlPrefix() + this.getParamString(data, encode);
}

/**
 * GET###################
 * ###################
 * @param data QR###########
 * @param encode HTML#################
 */
private String getParamString(String data, boolean encode) {
 StringBuffer buff = new StringBuffer();
 buff.append(this.getQrcodeUrlPrefix());

 String SLASH = "/";
 String AMP = "&";
 if (encode) {
  SLASH = "%2F";
  AMP = "&";
 }

 // #### text/plain, image/gif, image/jpeg, image/png
 buff.append("o=");
 buff.append("image" + SLASH + "gif");

 try {
  // ###
  String encodeddata = URLEncoder.encode(data, "Shift_JIS");
  // String encodeddata = URLEncoder.encode(data, "UTF-8");
  buff.append(AMP);
  buff.append("d=");
  buff.append(encodeddata);
 } catch (UnsupportedEncodingException e) {
  e.printStackTrace();
 }

 // ####### L(7%),M(15%),Q(25%),H(30%)
 buff.append(AMP);
```

```
 buff.append("e=L");

 // ## 0-10, 0###
 buff.append(AMP);
 buff.append("t=0");

 // ## 0-32
 buff.append(AMP);
 buff.append("m=5");

 // ##### 1-4
 buff.append(AMP);
 buff.append("s=3");

 return buff.toString();
}
```

4. インターフェース定義の追加

コンポーネント間をワイヤで結ぶためには、ツールやフレームワークにそのコンポーネントがどのようなインターフェース（入出力）を持っているのか伝えなければなりません。このインターフェースはXML形式の定義ファイル（WSDL)として記述します。ただし、その詳細な記述方法（XMLタグセット）を覚える必要も見る必要もありません。Lotus Expeditor Toolkitでは専用のエディタを提供していますので、必要項目を埋めるだけで作成できます。

以下を参考にメニューより新規の"Wiring Properties"を作成します。

表. インターフェース定義ファイル

保存場所	プロジェクト直下
ファイル名	PropertyBroker.wsdl（デフォルト）

作成後、Wiring Propertiesエディタで下記のプロパティとアクションを追加します。

コンポーネントのインターフェース詳細

プロパティ	名前	data
	名前空間	http://www.w3.org/2001/XMLSchema
	データ型	string
	その他	"公開を許可"のチェックをはずします
アクション	名前	consumeQrcodeData
	タイトル	QRコードデータ
	説明	QRコードに変換するデータ
	入力パラメータ	data

5. ActionHandlerの追加

コンポジット・アプリケーションでは、コンポーネントが他のコンポーネントからデータを受け取る際にプロパティ・ブローカーを介します。コンポーネント内でプロパティ・ブローカーから呼ばれるのがActionHandlerです。このクラスにイベント・データを受けた時の処理を実装していきます。

インターフェースorg.eclipse.core.commands.IHandlerを実装したクラスPBActionHandlerを新規作成します。実際にロジックを書かなければならないメソッドはexecuteのみです。

以下、サンプル内の実装コードのコメントの形で解説します。

```
public Object execute(ExecutionEvent event) throws ExecutionException {
final Object eventTrigger = event.getTrigger();

if (eventTrigger instanceof PropertyChangeEvent) {
 final PropertyChangeEvent pce = (PropertyChangeEvent) eventTrigger;
 final PropertyValue value = pce.getPropertyValue();

 // ##############
 Wire wire = pce.getWireDefinition();

 // ###############QrcodeView##ID###ID#######
 // ################### #######ID#SecondaryID######
 // ####View#ID#Secondary ID#########ID#########
 // Secondary#################Notes8####
 // ####################
 // #######ID#####Wire##########
 final String targetid = wire.getTargetEntityId();

 Display.getDefault().asyncExec(new Runnable() {
  public void run() {
   // ##############ID##View##########
   QrcodeView view = (QrcodeView) SWTHelper.locateView(targetid);
   if (view != null) {
    // QrcodeView#################
    view.setQrcodeData((String) value.getValue());
   }
  }
 });
}
// ##null##### (IHandler#ApiDoc###
return null;
}
```

以上がプロパティ・ブローカーに呼ばれたActionHandlerからのViewの呼び出し, データの引き渡しになります。

6. プラグイン定義ファイル(plugin.xml/MANIFEST.MF)

作成した部品をつなぎあわせるステップです。Eclipseというプラットフォームはプラグインによって機能を拡張します。今回作成した機能もプラグインとしてプラットフォームに追加します。この、実行環境に対して開発したコードを登録する作業はプラグインの定義ファイルであるplugin.xml/MANIFEST.MF を通して行います。

MANIFEST.MFにはプラグインの名前やバージョン、依存関係など OSGi バンドルとしての属性が記録されています。ここでは解説を割愛しますが、コードを参照くだされば と思います。

plugin.xmlに記述を加えることでコンポーネントとしての情報を実行環境に登録しています。以下、そのソースコードのコメントとして解説します。

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.2"?>
<plugin>
<extension point="org.eclipse.ui.views">
```

```
 <!--
 View###################ID######
 ####View######ID#########
 name#################################
 ################plugin_XX.properties##########
 ### #########plugin_ja.properties######
 ######################
 ##View###################allowMultiple#true#####
 #################################
 ##true##############
 -->
 <view
  allowMultiple="true"
  class="dw.japan.ca_series.components.qrcode.QrcodeView"
  id="dw.japan.ca_series.components.qrcode.QrcodeView"
  name="%view.name">
 </view>
</extension>
<extension
 point="com.ibm.rcp.propertybroker.PropertyBrokerDefinitions">
 <!--
 Lotus Expeditor#################
 ActionHandler#########################
 ##############################
 ############################
 -->
 <handler
  class="dw.japan.ca_series.components.qrcode.PBActionHandler"
  file="PropertyBroker.wsdl"
  type="SWT_ACTION" />
</extension>
</plugin>
```

7. フィーチャーの追加

ここまででコンポーネントはほぼ完成しました。あとは配布するのに適切なパッケージングです。 Eclipse RCPの世界ではプラグインは機能の単位あるいは開発の単位です。実際に配布する際には1つ以上のプラグインをフィーチャーとしてまとめる必要があります。今回作成したコンポーネントは1つのプラグインで構成されますので、これだけを含んだフィーチャーをdw.japan.ca_series.components.qrcode.featureという名前で作成します。

作成方法の説明はここでは割愛します。 第0回でご紹介している資料などを参考にしてみてください。もちろんサンプルコードを参考にしていただいてもよいです。

以上で、コンポーネントの開発が完了しました。流れに沿って要素要素を作成し、plugin.xmlを介して実行環境につなぎこんでいく。 Eclipse RCPでの開発、そしてコンポジット・アプリケーションのコンポーネントの開発のイメージがなんとなくでもつかめていれば幸いです。

コンポジット・アプリケーションの作成

コンポーネントが2つそろったところでいよいよ連携のテストを行いたいところですが、もう1つ作らなければならないものがあります。2つのコンポーネントを使ったコンポジット・アプリケーションです。

Lotus Notes 8 や WebSphere Portal で管理されたクライアントでは、このようなコンポジット・アプリケーションのコードやプラグインをアプリ開発者が用意する必要はありません。コンポーネントを組み合わせてアプリケーションを組み立てるためのツールや設定画面が用意されているか

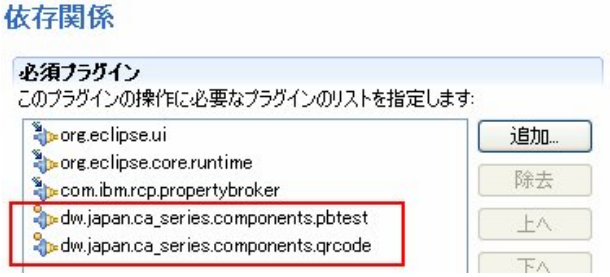
らです。今回はLotus Expeditorをターゲット環境としているため、この部分をプログラムとして開発します。前述のような環境を対象としている場合にも、この手順は知っておいて損はないと思います。見えないところで同様の処理が自動的になされているからです。各ツール内で出てくるキーワードへの理解がしやすくなるはずです。

手順は以下のようになります。

1. プラグイン用の新規"Client Servicesプロジェクト"の作成。ここでは、プロジェクト名・プラグインIDともに"dw.japan.ca_series.compositeapps.qrcode"としましょう。
2. このコンポジット・アプリケーションが利用するコンポーネントはすべて外部のプラグインです。動作する上でこれらに依存しますので、プラグインの依存関係を適切に設定します。今回の場合は、作成した"dw.japan.ca_series.components.qrcode"とテスト用コンポーネント"dw.japan.ca_series.components.pbtest"がそれにあたります。
3. Perspectiveクラスの実装。画面のレイアウト、つまりどのコンポーネントをどこにどのように配置するかこのクラスの中に記述します。
4. 上記PerspectiveをEclipseの拡張ポイントを使用して登録します。(plugin.xml)
5. クライアント実行環境のランチャーから起動できるよう、アプリケーション・ランチャー登録用の拡張ポイントを使って上記Perspectiveを登録します。(plugin.xml)
6. 配置した2つのコンポーネント間の連携（Wire）を専用の拡張ポイントを介して登録します。(plugin.xml)
7. このプラグインを配布できる単位にパッケージするためのフィーチャー（新規フィーチャープロジェクト）を作成し、プラグインを追加します。ここでは、プロジェクト名・フィーチャーIDとも "dw.japan.ca_series.compositeapps.qrcode.feature" としましょう。
8. 上記フィーチャーと必要な他のフィーチャーをすべて1つの新規更新サイトに加え、ビルドします。

これらのうち、主要なステップについて解説します。

依存関係



画面にあるように、"プラグインのマニフェスト・エディター"でplugin.xmlかMANIFEST.MFを開き、依存関係のタブで2つのコンポーネントのプラグインを追加します

Perspectiveの実装・登録・ランチャー登録（上記3～5）

以下がPerspective実装クラスの中で実際にレイアウトを行っている部分です。ここではでは各ViewのIDをstaticな変数として宣言し、変数で2つのコンポーネントのIDを指定してそれぞれ配置しています。

```
public void createInitialLayout(IPageLayout layout) {
 layout.setEditorAreaVisible(false);
 layout.addView(StringPublishView.ID, IPageLayout.LEFT, 0.5f, layout.getEditorArea());
 layout.addView(QrcodeView.ID, IPageLayout.RIGHT, 0.5f, layout.getEditorArea());
}
```

そして、このクラスを名前をつけて登録するのがplugin.xml内の以下の記述です。

```
<extension point="org.eclipse.ui.perspectives">
 <!--
 ####################
 ##########ID#####
 -->
 <perspective
  name="%perspective.name"
  class="dw.japan.ca_series.compositeapps.qrcode.Perspective"
  id="dw.japan.ca_series.compositeapps.qrcode.Perspective">
 </perspective>
</extension>
```

さらに、下記の記述により、ランチャーへの登録を行います。上記でPerspectiveにつけたIDを使用します。

```
<extension
 id="app1"
 point="com.ibm.eswe.workbench.WctApplication">
 <!--
 ######################Perspective###
 ############################
 ########################
 -->
 <Application
  DisplayName="%application.name"
  PerspectiveId="dw.japan.ca_series.compositeapps.qrcode.Perspective">
 </Application>
</extension>
```

この記述の追加だけでワイヤの登録も完了です。ツール内でこの拡張ポイントを使う際に依存するプラグインを依存関係のリストに加えるか聞いてくるダイアログが現れますが"はい"を選択して進めてください。依存関係にPropertyBrokerのプラグインが追加されたことが確認できます。

更新サイトの作成

ここまでで2つのコンポーネント、1つのコンポジット・アプリケーションをプラグインとして用意できました。そしてそれぞれ用のフィーチャーも準備済みです。あとはインストール可能な形にビルドするだけです。更新サイトプロジェクトの詳細な作成手順は前回ご紹介している関連資料を参照ください。更新サイトに単純に3つのフィーチャーを追加してもよいのですが、画面にあるように、カテゴリ分けすると分かりやすいでしょう。

最後に、"すべてをビルドします"ボタンをクリックします。正常に終了すれば開発は完了です。

動かしてみよう

クライアント環境(Lotus ExpeditorまたはLotus Notes 8)に実際にアプリケーションを導入して動作を確認してみてください。アプリケーション・ランチャーからコンポジット・アプリケーションを起動し、最初のご紹介した完成イメージの通りに並べられたコンポーネント間でデータが連携されれば成功です。

まとめ

今回は、既存のWebサイトが提供している画像生成機能をラップし、コンポジット・アプリケーションの一構成要素（コンポーネント）として利用できるようにしました。しかしながら、この形態ではネットワークに依存していますし、インターネット上にむやみに情報を流すことにもなります。

次回はステップ２として、コンポーネントのインターフェースを変えることなくローカルで完結して動作するコンポーネントに作り変えていきます。また、筆者の経験からプラグインとしてまとめる単位、パッケージングの単位の決め方についても触れていく予定です。

サンプルについて

最後に、サンプルの.zipファイルの中身について簡単にまとめておきます。中身を参照される際に参考にしてください。

完成版の更新サイト:updatesite.zip

まずは動かしてみたいという方はLotus Expeditor / Lotus Notes 8でこのzipファイルを更新サイトに指定し、インストールください

プロジェクトファイル:projects.zip

開発環境にインポートしてお使いいただけるプロジェクトのアーカイブです。各プロジェクトの内容は以下の通りです。

表. projects.zipの内容

プロジェクト名	内容

dw.japan.ca_series.components.qrcode	QrcodeViewコンポーネント
dw.japan.ca_series.components.qrcode.feature	上記コンポーネント用フィーチャー
dw.japan.ca_series.components.pbtest	StringPublishViewコンポーネント
dw.japan.ca_series.components.pbtest.feature	上記コンポーネント用フィーチャー
dw.japan.ca_series.compositeapps.qrcode	コンポジット・アプリケーション
dw.japan.ca_series.compositeapps.qrcode.feature	上記用のフィーチャー
dw.japan.ca_series.updatesite	全フィーチャーを含む更新サイト

ダウンロード

内容	ファイル名	サイズ
プロジェクトファイル	projects.zip	61KB
完成版の更新サイト	updatesite.zip	21KB

著者について

森谷 直哉

森谷 直哉
2002 年頃よりパーベイシブ・コンピューティング分野（音声、モバイル、etc.）のソフトウェア製品の技術支援に従事。ここ数年はEclipseベースのリッチクライアントテクノロジーであり、Notes8のベース技術ともなっているLotus Expeditorを中心にLotusブランドにてテクニカル・セールスとして活動。2008年からはIBM Mashup Centerという新製品でエンタープライズ・マッシュアップのエリアで奮闘中。