コンポジット・アプリケーション連載: 第2回 QRコード を活用しよう(後編)

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

2008年 1月 25日

前編では、インターネット上で公開されているQRコード® 生成サイトを利用したコンポーネントを開発し、サンプルとして提供した別のコンポーネントと合わせて1つのコンポジット・アプリケーションに配置・ワイヤリングしました。今回は前編の続きのステップ2として、QRコードを表示するコンポーネントを改良していきます。

このシリーズの他の記事を見る

改良のポイント

第1回で作成したコンポーネントは埋め込まれたSWT Browserウィジェットを使って指定したURLの画像を表示するというものでした。 HTTPのGETメソッドで（パラメータを埋め込んだURLを渡すことで）、QRコードを画像として生成してくれる外部サービスの指定と、生成する画像に関する各種パラメータ（大きさ、データなど）を指定していました。

この形態で十分に動作はしますが、実用を考えたときに下記の欠点・課題が残ります。

1. **ネットワークに接続していないと使えない**：リモートのサービスを利用している以上、インターネット接続が可能な環境でした使えません。移動中など、ネットワークがない環境での作業中にデータを携帯電話に飛ばしたいといったこともあるでしょう。
2. **インターネット上をデータが流れる**：携帯電話に飛ばしたい情報を担当顧客の氏名と電話番号という可能性もあります。そのような情報を暗号化もせずにむやみにインターネット上に流すことは避けるべきです。

今回はこれらを解消すべく、コンポーネントを改良していきます。

ステップ2 : オフライン対応

改善策の検討

前回、コンポーネントの開発を行う前に、Kazuhiko Araseさんが公開されている"QRコードライブラリ for Java"について調査し、以下のことが分かりました。

- 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。

これらの情報を元に、上記問題の解消のためにとれる策を考えます。

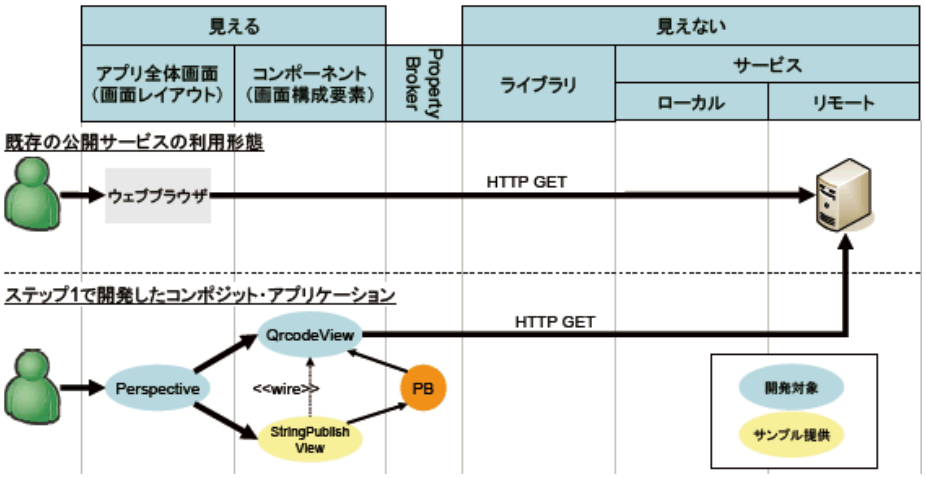
1番目の問題を解決には ネットワーク上ではないところ つまり端末のローカル環境でのQRコードの生成処理が必須です。これが実現すれば自動的に2番目も解消します。よって、ローカルで動かすための選択肢を検討することとします。

1つの方法として、ライブラリが提供するAPIを利用して生成されたQRコード画像を独自のGUIに埋め込むことが考えられます。 Eclipseベースの環境での標準UIライブラリはSWTです。SWTにはAWTで実装された部品を埋め込むことが可能ですので、実現性は十分にあるでしょう。通常、SWTでこうした異なる技術の埋め込み際にはビジュアルな面だけでなくイベントやスレッド処理に十分に注意する必要がありますが、今回は画像を表示するだけのシンプルなアプリケーションですのでその心配も不要です。ただ、この方法では新たに独自の画面（UI）を実装しなおさなければなりません。もっとよい方法はないものでしょうか。

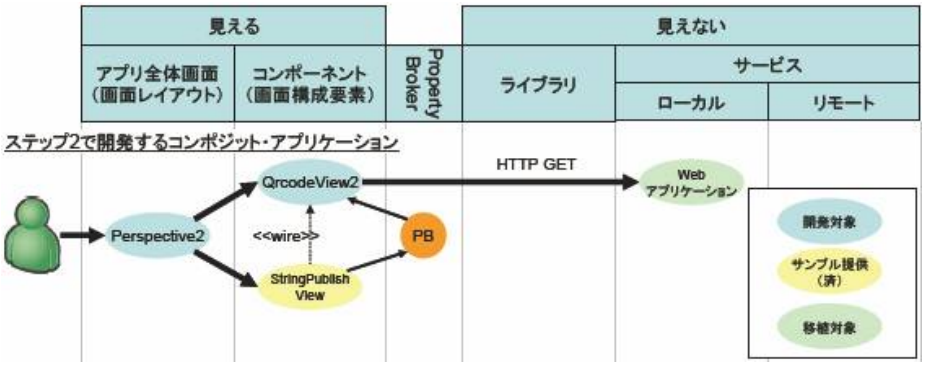
Lotus Expeditorでは、クライアント環境上で（ローカル）で動作するWebコンテナを提供していました。つまり、クライアント上でWebアプリケーションが動かすことができます。公開されているライブラリもWebアプリケーション(.war)としてパッケージ・提供されています。このWebアプリケーションをExpeditor上で動かすことで、指定URLを変更するだけで前回作成した画面を再利用できそうです。既存の資産を最大限再利用できますので今回はこの手法をとりたいと思います。

設計

オフラインで実行されたときの動作の仕組みを考えて見ましょう。まずは前回のモデルです。



そして、こちらがオフラインで利用可能となった場合のモデルです。ブラウザが見に行く先がインターネット上のサイトからローカルのWebアプリケーションに変更された以外、特に変更がないのがわかるかと思います。



ローカルWebコンテナについて

WebコンテナはWebコンテナでも、一般的なJavaEEサーバーのWebコンテナとExpeditorが提供するローカルWebコンテナには多少の差異があります。ここでは、事前にこれらについて簡単にまとまておきます。

アプリケーション開発者が知っておくべき重要な事項は下記の通りです。

- **Webコンテナが待ち受けるポート番号はデフォルトではランダム**：特定用途のために占有されるサーバー環境と異なり、パソコンではどのようなアプリケーションやサービスが稼動しているかわかりません。そのため、どの利用対象端末でも共通の特定ポートが必ず空いていると想定することは難しいでしょう。そこで、ExpeditorではWebコンテナはTCPのポート80番といった固定ポートではなく、クライアント環境が起動したときに空いているポートを使って待ち受けるのが標準の動作となっています。クライアント環境のJavaシステム変数を使ってポートを固定することも可能ですが、今回は標準の振る舞いを想定したいと思います。この振るまいで困るのは、ブラウザなどローカルのWebコンテナ上のWebアプリケーションを利用する側は事前にアクセスすべきURLを知ることができないという点です。対処方法は次のようになります。Webコンテナはアプリケーションが動作する前に必ず始動します。（これは、利用する側がWebコンテナへの依存関係を持つのでフレームワークが自動的に起動順序を制御してくれます）。起動したWebコンテナの動作ポートはフレームワーク経

由で取得することができます。したがって、ローカルのWebアプリケーションを利用する側は、アクセスする前にフレームワークから取得したWebコンテナの動作ポートを使ってURLを組み立ててればよいのです。
- パッケージング：JavaEE環境ではJavaEEのパッケージング方法が規定されているように、Eclipse（OSGi）にはEclipse(OSGi)のパッケージング方法があります。Webアプリケーションに関していうと、前者ではWAR（WebApplicationArchive)ファイルに、後者はWAB(WebApplicationBundle)という形式になります。ExpeditorToolkitでは両者の間の変換を行う機能が提供されていますので、これを利用することで詳細な差異を意識する必要はありませんが変換の主要ポイントは、1. OSGiバンドルとしての必要情報（マニフェストファイルなど）の追加、2. JSPのServletへのプリコンパイルとそれらのJSP用のServletMappingの生成です。

開発の流れ

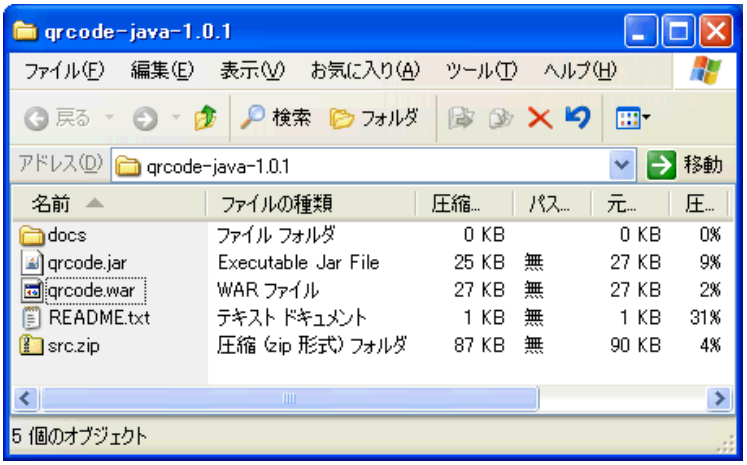
前述の事項を踏まえたアプリケーション開発の流れは以下のようになります。

1. 新規コンポーネントの作成
    - 1.1.ローカルWebアプリケーションの開発（既存Webアプリケーションの変換）
        - 1.1.1.WARファイルを環境にインポートし、動的Webプロジェクトqrcode.webを作成
        - 1.1.2.上記プロジェクトをClientServicesWebプロジェクトに変換し、プロジェクトdw.japan.ca_series.components.qrcode2.wabを作成
        - 1.1.3.コンテキスト・ルート、Servletにアクセスするためのパス（ServletMapping）の確認
        - 1.1.4 プラグインプロジェクトのビルド情報の修正
    - 1.2.新規コンポーネント（View）プラグインの作成
        - 1.2.1 新規Client Servicesプロジェクト"dw.japan.ca_series.components.qrcode2"を作成
        - 1.2.2 依存関係の設定
        - 1.2.3 BundleActivatorの修正
        - 1.2.4.QrcodeViewを継承した新規Viewdw.japan.ca_series.components.qrcode2.QrcodeView2を作成
        - 1.2.5 plugin.xmlへのViewの登録
        - 1.2.6 プロパティ・ブローカーに対して公開するインターフェース定義（WSDL)、ActionHandlerの作成
        - 1.2.7 plugin.xmlへのインターフェース定義（WSDL)、ActionHandlerの登録
    - 1.3.新規フィーチャー"dw.japan.ca_series.components.qrcode2.feature"の作成
2. 新規コンポジット・アプリケーションの作成
    - 2.1.プラグイン"dw.japan.ca_series.compositeapps.qrcode2"の作成
        - 2.1.1 新規Perspectiveへの2つのViewの配置
        - 2.1.2 plugin.xmlへのPerspectiveの登録、ランチャーへのPerspectiveの登録
    - 2.2. フィーチャー"dw.japan.ca_series.compositeapps.qrcode2.feature"の作成
3. 更新サイトの作成

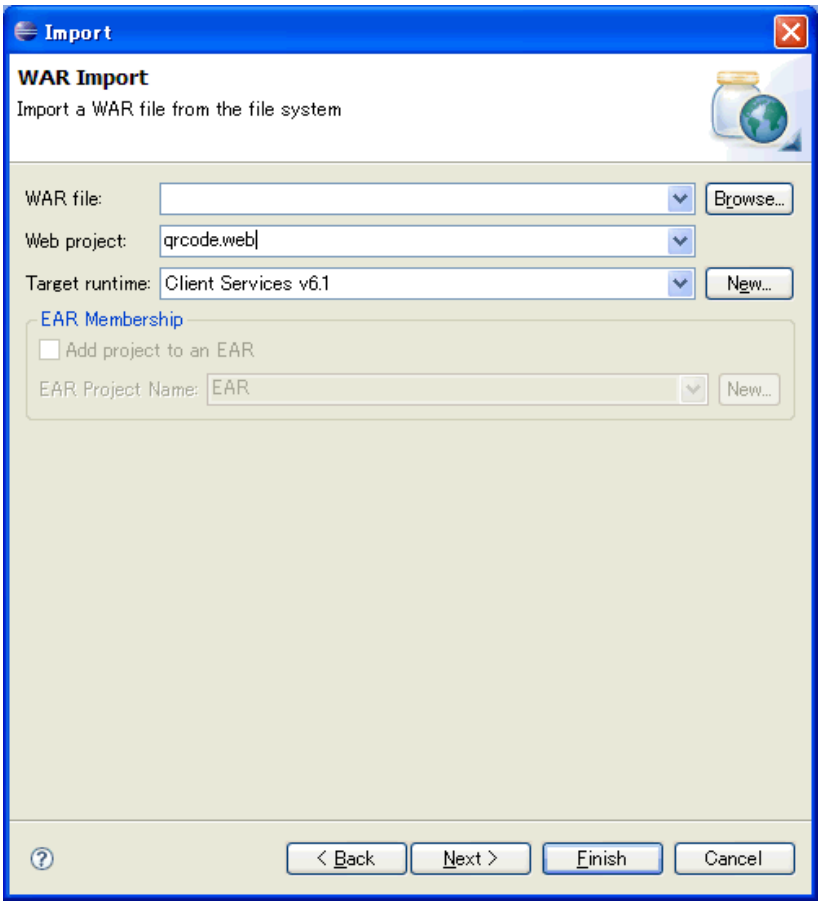
開発の手順

今回も上記のすべてではなく、ポイントとなる部分についてのみ詳細な解説を行います。より詳細な動作や実装コードについては本記事末尾より完成したプロジェクトをダウンロード・参照ください。

1. 新規コンポーネントの作成
    - 1.1.ローカルWebアプリケーションの開発（既存Webアプリケーションの変換）
        - 1.1.1.WARファイルを環境にインポートし、動的Webプロジェクトqrcode.webを作成
        インポート対象のファイルは、ダウンロードしたqrcode-java-1.0.1.zipに含まれているqrcode.warです。

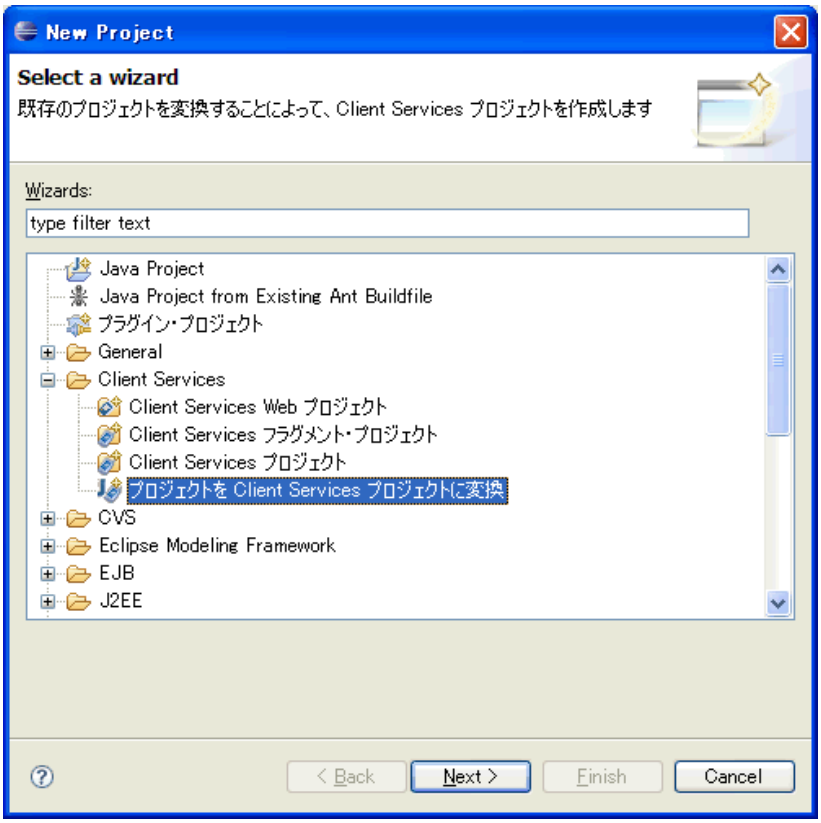


メニューの File -> Import　より下記ダイアログでWeb下の"WAR file"を指定し、ここでは以下のようにWeb project "qrcode.web"として"Finish"をクリックし、インポートします。

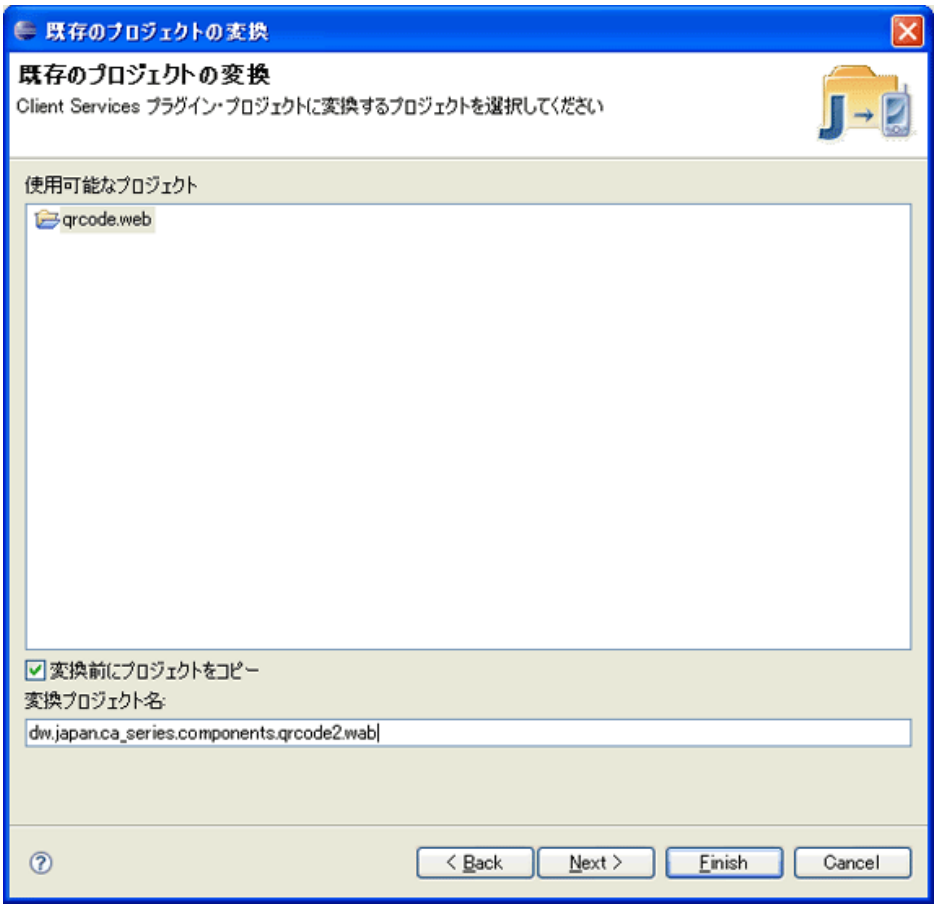


プロジェクト内のform.jspの構文が正しくないとエラーが出ますが、大きな問題ではありません。修正する、もしくは上書き保存すればここは進められます。

- 1.1.2.上記プロジェクトをClientServicesWebプロジェクトに変換し、プロジェクトdw.japan.ca_series.components.qrcode2.wabを作成
メニューの"File"->"New"->"Project"で表示されるダイアログの"Client Services"下の"プロジェクトを Client Servicesプロジェクトに変換"を選択します。



変換対象のプロジェクトとして、先ほど作成した"qrcode.web"を選択し、ここではコピーを"dw.japan.ca_series.components.qrcode2.wab"として作成して処理するように指定、"Finish"をクリックして変換します。



変換完了後に先程と同じようにform.jspのエラーが出るようでしたら、適当に修正します。本記事のアプリケーションではform.jspは使用しませんので、削除してしまってもかまわないでしょう。

- 1.1.3.コンテキスト・ルート、Servletにアクセスするためのパス（ServletMapping）の確認
  後ほど作成するViewではこのWebアプリケーションにアクセスします。その際、Webアプリケーションのコンテキスト・ルートや、QRコードを生成してくれるサーブレットのパスが必要となります。ここであらかじめ確認しておきましょう。下記のように、それぞれ "/qrcode.web", "/qrcode"　であることがわかります。

  コンテキスト・ルート（plugin.xmlより）

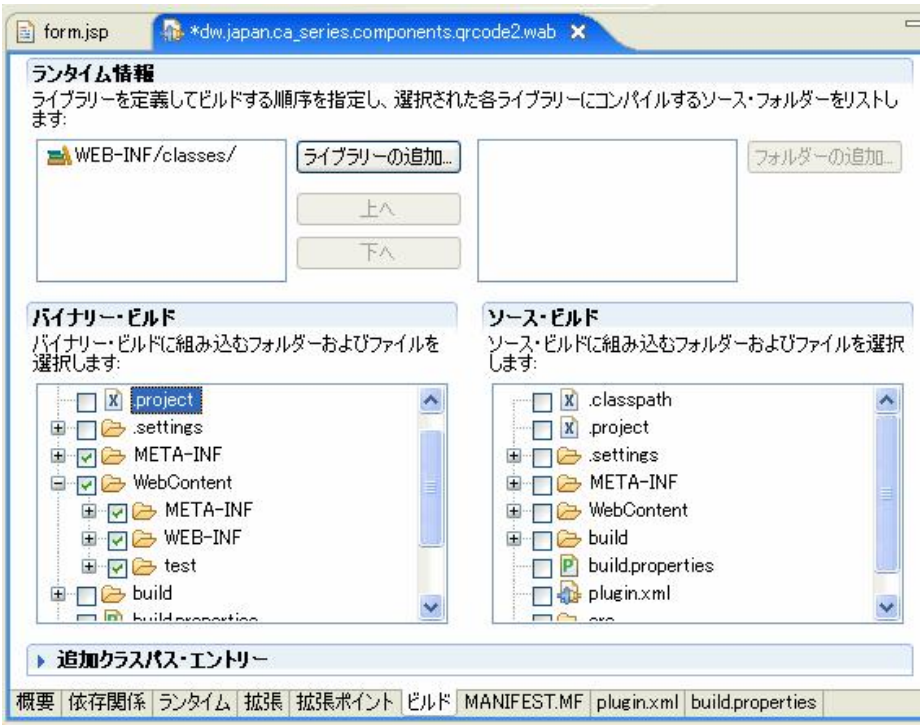
```
<extension point="com.ibm.pvc.webcontainer.application">
<contextRoot>/qrcode.web</contextRoot>
<contentLocation>WebContent</contentLocation>
</extension>
```

サーブレットのパス（ WEB-INF/web.xmlより ）

```
<servlet>
  <servlet-name>qrcode</servlet-name>
  <servlet-class>com.d_project.qrcode.web.QRCodeServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>qrcode</servlet-name>
  <url-pattern>/qrcode</url-pattern>
</servlet-mapping>
```

- 1.1.4 プラグインプロジェクトのビルド情報の修正
プラグインプロジェクトがビルドされた際、.jarに格納されるファイルを指定するのがマニフェスト・エディタの"ビルド"タブです。変換して作成したClient Service Webプロジェクトでは、デフォルトで重要なディレクトリのチェックが抜けています。下記のように、"WebContent"にチェックが入れ、ファイルを保管してください。この確認を怠ると、開発環境ではうまく動かないので、ビルド・配布したあとの実行環境では必要ファイルが欠けているために動作しない　といった問題が発生してしまいます。



以上で、QRコード生成機能を提供するローカルWebアプリケーションが開発できました。プログラムコードの記述なく、WARファイルのインポート、変換、パッケージングとほとんどGUI操作で行えることが体験いただけたかと思います。もちろん、複雑なWebアプリケーションの場合にはこれだけでは済まないかもしれません。JavaEEサーバー上で構成されたデータベースやキューといったリソースへJNDI経由でアクセスできることが想定されていれば、それに見合った移植作業も発生しうるからです。別の機会にこれらについて触れられればと思います。

- 1.2.新規コンポーネント（ View ）プラグインの作成

- 1.2.1 新規Client Servicesプロジェクト"dw.japan.ca_series.components.qrcode2"を作成します。
  プロジェクトの作成はメニューより File -> New -> Project と辿り、"Client Servicesプロジェクト"を指定して行います。前編同様、テンプレート "基本リッチクライアント"からはじめると分かりやすいでしょう。
- 1.2.2 依存関係の設定
  plugin.xmlをダブルクリックしてマニフェスト・エディタを開き、"依存関係"のタブで下記の4プラグインを依存関係に追加します。
  dw.japan.ca_series.components.qrcode：前回作成のViewを継承して新規Viewを作成するため
  dw.japan.ca_series.components.qrcode2.wab：先ほど作成したローカルWebアプリケーション
  com.ibm.rcp.propertybroker
  com.ibm.rcp.propertybroker.swt

- 1.2.3 BundleActivatorの修正
  前述のように、Webコンテナが待ち受けるポート番号は事前に知ることができません。そこで、この値はフレームワークより取得することとなります。プラグインの中で、フレームワークとのインターフェースを取るのはBundleActivatorですので、このクラスの中でその処理を書くのが良いでしょう。
  WebコンテナのInformation取得方法については、Lotus ExpeditorのInfoCenter内"Locating the Web Container ports using the HttpSettingListener Service"にも記載されてますので参考にしてください。
  例えば、下記のように記述することでActivatorはWebコンテナの待ちうけポート番号・ホスト名の通知を受け、インスタンス変数に保持することができます。この値をプラグイン内の各クラスが参照することで必要としている情報を取得可能となります。

```
package dw.japan.ca_series.components.qrcode2;

import java.util.Dictionary;
```

```
import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.ui.plugin.AbstractUIPlugin;
import org.osgi.framework.BundleContext;

import com.ibm.pvc.webcontainer.listeners.HttpSettingListener;

/**
 * The activator class controls the plug-in life cycle
 */
//Web##################HttpSettingListener######
public class Activator extends AbstractUIPlugin implements HttpSettingListener {

 // The plug-in ID
 public static final String PLUGIN_ID = "dw.japan.ca_series.components.qrcode2";

 // The shared instance
 private static Activator plugin;

 /**
  * ###################
  */
 private String hostname = null;

 /**
  * ####################
  */
 private int port = -1;

 ######

 public void start(BundleContext context) throws Exception {
  super.start(context);

  //Web##########################
  //###########Listener#########
  context.registerService(HttpSettingListener.class.getName(), this, null);
 }


 ######

 /**
  * Web##################
  * ####################
  */
 public void settingsModified(String PID, Dictionary settings) {
  String scheme = (String) settings.get(HttpSettingListener.SCHEME);
  if (HttpSettingListener.SCHEME_HTTP.equals(scheme)) {
   Object iPort = settings.get(HttpSettingListener.HTTP_PORT);
   if (iPort instanceof Integer[]) {
    port = ((Integer[]) iPort)[0].intValue();
   } else {
    port = ((Integer) iPort).intValue();
   }
   String sHost = (String) settings.get(HttpSettingListener.ADDRESS);
   if (HttpSettingListener.ALL_ADDRESSES.equals(sHost)) {
    hostname = "localhost";
   } else {
    hostname = sHost;
   }
  }
 }

 ######

 public int getPort() {
```

```
  return this.port;
 }

 public String getHostname() {
  return this.hostname;
 }

}
```

- 1.2.4.QrcodeViewを継承した新規View
  dw.japan.ca_series.components.qrcode2.QrcodeView2を作成
  今回作成するコンポーネントのViewの振る舞いは前編で作成したQrcodeViewとほぼ同じです。異なるのは、ブラウザにインターネット上のURLを指定する代わりに、ローカルで動かすWebアプリケーション内のサーブレットのURLになる点だけです。差分だけ開発すればよいよう、ここではQrcodeViewを継承する形で実装するとします。
  URLを生成するメソッド getQrcodeUrlPrefixは下記を参考にオーバーライドします。Activator経由でWebコンテナが待ち受けるホスト名・ポート番号を取得、また1.1.3.で確認したコンテキスト・ルート、パスを利用してURLを組み立てています。

```
package dw.japan.ca_series.components.qrcode2;

import dw.japan.ca_series.components.qrcode.QrcodeView;

public class QrcodeView2 extends QrcodeView {

 public static String ID = "dw.japan.ca_series.components.qrcode2.QrcodeView2";

 protected String getQrcodeUrlPrefix(){
  StringBuffer buff = new StringBuffer();
  buff.append("http://");
  String hostname = Activator.getDefault().getHostname();
  int port = Activator.getDefault().getPort();
  if (hostname != null && port != -1) {
   buff.append(hostname);
   buff.append(":");
   buff.append(port);
   buff.append("/qrcode.web");
  }
  buff.append("/qrcode");
  buff.append("?");
  return buff.toString();
 }
}
```

- 1.2.5 plugin.xmlへのViewの登録
  今回も慣例に従い、Viewのクラス名をViewIdとして使います。

```
<extension
         point="org.eclipse.ui.views">
          <view
                allowMultiple="true"
                class="dw.japan.ca_series.components.qrcode2.QrcodeView2"
                id="dw.japan.ca_series.components.qrcode2.QrcodeView2"
                name="%view.name">
      </view>
</extension>
```

- 1.2.6 プロパティ・ブローカーに対して公開するインターフェース定義（ WSDL)、ActionHandlerの作成

前編で作成したコンポーネントとインターフェースは同じですから、PropertyBroker.wsdlはそのままコピーして使ってよいでしょう。プロパティ・ブローカーに呼ばれたときの処理を行うActionHandlerについても同様です。ただし、Viewが異なってますので、下記の部分だけは修正します。

```
Display.getDefault().asyncExec(new Runnable() {
 public void run() {
  QrcodeView2 view = null;
  if (targetid != null && !"".equals(targetid)) {
   view = (QrcodeView2) SWTHelper.locateView(targetid);
  }
  if (view != null) {
   view.setQrcodeData((String) value.getValue());
  }
 }
});
```

- 1.2.7. plugin.xmlへのインターフェース定義（WSDL)、ActionHandlerの登録

```
<extension
         point="com.ibm.rcp.propertybroker.PropertyBrokerDefinitions">
      <handler
            class="dw.japan.ca_series.components.qrcode2.PBActionHandler"
            file="PropertyBroker.wsdl"
            type="SWT_ACTION"/>
</extension>
```

- 1.3.新規フィーチャー"dw.japan.ca_series.components.qrcode2.feature"の作成
作成したコンポーネントを配布するためのフィーチャーを作成します。メニューの "File" -> "New" -> "Project" より新規のFeatureプロジェクト"dw.japan.ca_series.components.qrcode2.feature"を作成し、2つのプラグイン "dw.japan.ca_series.components.qrcode2.wab", "dw.japan.ca_series.components.qrcode2"をそれぞれ登録しましょう。

以上でローカルのWebコンテナで動作するWebアプリケーションを利用するコンポーネントの開発ができました。

2. 新規コンポジット・アプリケーションの作成
コンポーネントがそろったところで、いよいよコンポジット・アプリケーションの組み立てです。今回、コンポーネントのインターフェースは前編とまったく変えていません。Viewを新規作成したため、配置したいViewのIDが異なるくらいでしょう。よって、ここでは詳細な手順については割愛させていただきます。前編の手順を真似てコンポジット・アプリケーションを開発にチャレンジしてみてください。

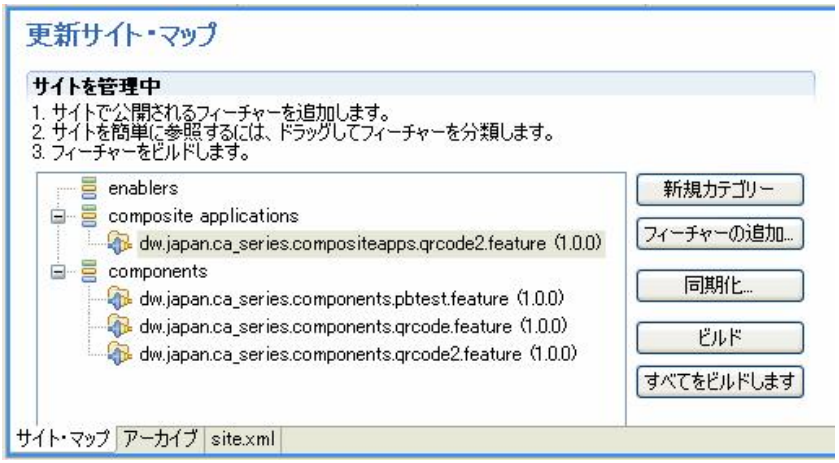
流れだけ確認しておきますと以下のようになります。

- 2.1.プラグイン"dw.japan.ca_series.compositeapps.qrcode2"の作成
  - 2.1.1 新規Perspectiveへの2つのViewの配置
  - 2.1.2 plugin.xmlへのPerspectiveの登録、ランチャーへのPerspectiveの登録
- 2.1.3 ワイヤリング
  - 2.2. フィーチャー"dw.japan.ca_series.compositeapps.qrcode2.feature"の作成

もちろん、前編で作成したものを修正するだけでもお試しいただけます。完成品もダウンロード可能としていますので参照ください。

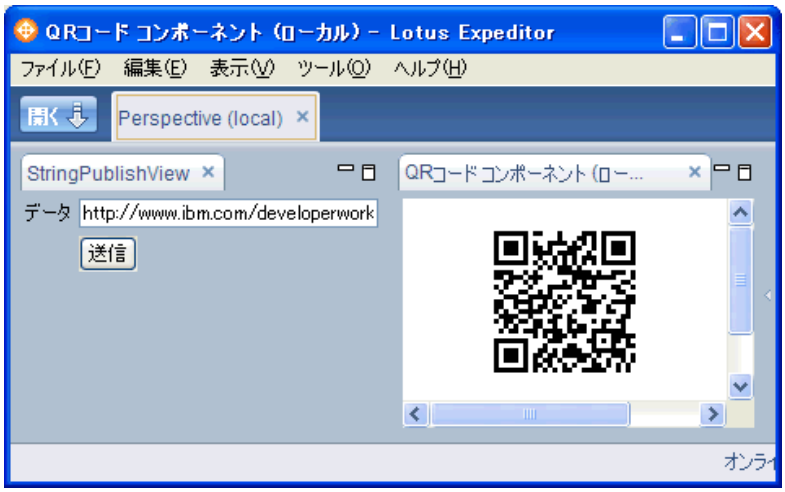
3. 更新サイトの作成

作成したフィチャーを配布するための更新サイトを作成します。ここでは後編（ステップ2）で開発したコンポジット・アプリケーションを動かすのに必要なすべてのフィーチャーを含めることとします。つまり、前編・後編両方のQRコード表示コンポーネント、テスト用のコンポーネント、コンポジット・アプリケーション　の4つのフィーチャーが該当します。



このプロジェクトをビルドした結果は以下のようになるはずです。zipして配布するなり、HTTPサーバー、Dominoサーバー上への配置など、ターゲットとするクライアントやインフラに応じた場所に配置するとよいでしょう。

以上でアプリケーション開発は完了しました。実際に動かしてみてください。見た目には前回と同様の結果が得られるはずです。ただし、今回の成果物では完全にオフラインでも利用可能です。これにより2つの問題・課題が解消され、より実用的になったのではないかと思います。



パッケージングについて考えてみる

さて、今回の記事を終える前に、より保守性の高いパッケージングについて考え方を整理し、それを踏まえて今回作成したアプリケーションをパッケージングしなおしてみたいと思います。

プラグインやフィーチャーの単位はどのように考えたらよいでしょうか？Eclipseのアプリケーション開発をはじめたばかりですと動かすことが最優先となりなかなかこの点を考慮していないかもしれません。しかし、これは保守性や配布のサイズや頻度に大きな影響がある重要なことなのです。以下、筆者が心がけている点についてまとめておきます。

プラグイン＝開発の単位、フィーチャー＝配布の単位　であることを改めて意識しましょう。どんなにプラグインを分割したところで、すべてが同じフィーチャーに含まれている限り、そのうちの1つだけを独立して更新　ということはできません。もしも、"これだけ切り出して他のアプリケーションでも利用する可能性がある"、あるいは　"ここだけ修正して入れ替えたくなることも考えられる"　と感じることがあれば、かならずプラグインだけではなく、フィーチャーも別にすることが重要です。

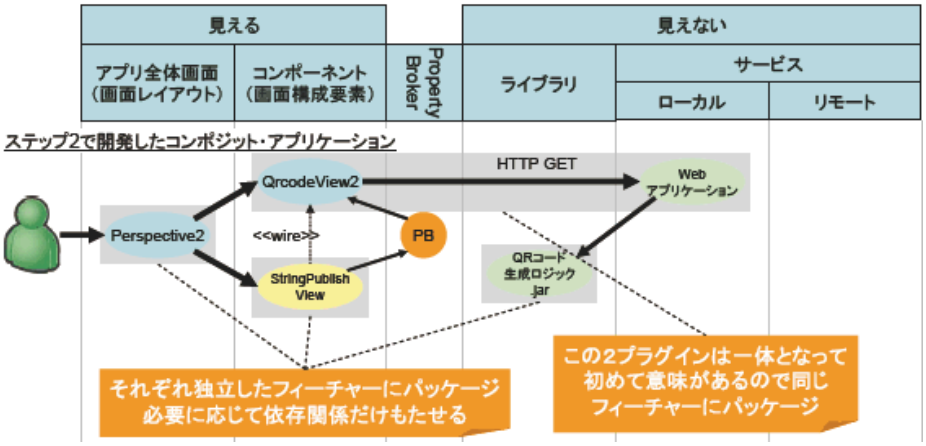
以上から今回のコンポジット・アプリケーションのパッケージングを考え直してみます。

Webアプリケーションにおいて、WEB-INF/lib以下に、参照される.jarファイルを数多く含めることはよくあります。この階層に格納された.jarは自動的にクラスパスに追加されますし、サーバーのシステムパスに同じライブラリの他のバージョンが含まれていてもこちらが優先されるので開発者にとっては都合のよい方法かもしれません。一方で、他のWebアプリケーションと共有することもできませんし、逆に特定のライブラリだけを最新のバージョンにあげたいと思ってもWebアプリケーションを丸々入れ替えなければならないといった問題も引き起こします。管理が1箇所に集中するサーバー環境ならいざしらず、モジュールの配布を意識しなければならないクライアント実行環境においてはあまり好ましくありません。

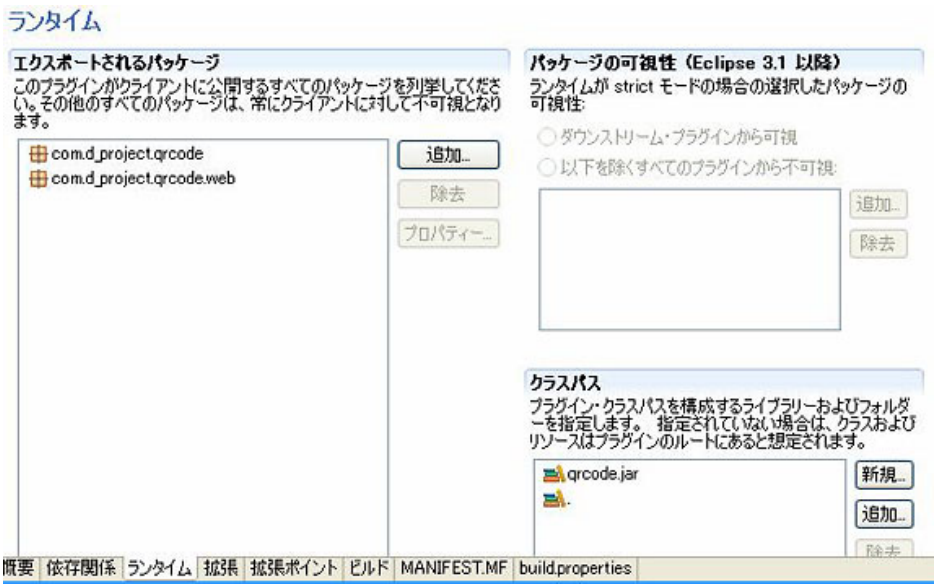
ちなみに、OSGiのWebアプリケーションでは、この階層は自動的にクラスパスに加えられることはなく、開発ツールでWARを変換する際に各.jarをバンドルのクラスパスに加えて（マニフェストファイル内）くれています。また、OSGiの世界では、同じバンドルの複数のバージョンが混在していてもかまいません。通常は最新のものが使われますが、依存関係にバージョン指定まで含めることで古いバージョンを使わせることも可能なのです。

こうなると、WEB-INF/lib下にむやみにライブラリを含める理由はもはやないでしょう。複数のコンポーネントから参照される可能性のあるライブラリはそれぞれ独立したプラグインとしてパッケージングし、利用する側は適切なプラグインを依存関係に含めておけばよいのです。ライブラリの単位で配布できなければなりませんので、ライブラリ毎に専用のフィーチャーも作成します。

今回作成したローカルWebアプリケーションについてこの考え方を提供すると以下のような形にできます。



Webアプリケーションで利用されているQRコード生成のコアロジックを提供するライブラリも外に切り出します。切り出したライブラリはプラグイン"qrcode.lib"とし、Kazuhiko Araseが公開されているqrcode.jarを保持すると同時にこのjarに含まれるすべてのクラスをエクスポートするように設定しました。Webアプリケーションはこの新しいプラグインに対する依存関係を持たせます。もちろん、このライブラリも専用のフィーチャーにパッケージします。ライブラリのプラグイン化の詳細な手順は示しませんが、興味ある方は本記事添付のプロジェクト"qrcode.lib"内のMETA-INF/MANIFEST.MFを参照ください。実際の設定操作はマニフェストエディタ（下図）で行いますが、このファイルにその結果が反映されているのが分かります。



これで万が一ライブラリがバージョンアップしたとしても、APIが変更されていない限り、ライブラリだけを更新すればよいでしょう。また、このライブラリを利用する他のプラグインを開発する場合にも、改めて同じ.jarを配布する必要がなくなりました。

Eclipseベースのアプリケーションを開発する際には是非パッケージングの入念な検討についても視野にいれておいていただければと思います。

まとめ

第1回、第2回と前後編に分けて、QRコードを生成するコンポーネント、そしてそのコンポーネントを含んだコンポジット・アプリケーションの開発を行いました。EclipseベースのLotus Expeditorのアプリケーション開発お手順、既存資産（Webアプリケーション、ライブラリ）の再利用方法・手順についてイメージがつかんでいただけていれば幸いです。

次回からは、どのように外部のWebシステムをコンポーネント化してコンポジット・アプリケーションの中で再利用できるのか取り上げたいと思います。お楽しみに。

ダウンロードファイルについて

最後に、第2回で取り扱った下記プロジェクトをまとめてzipファイルにまとめましたのでご利用ください。

プロジェクト名	内容
dw.japan.ca_series.components.pbtest	StringPublishViewコンポーネント（テスト用、前編と同じ）
dw.japan.ca_series.components.pbtest.feature	StringPublishViewコンポーネント配布用フィーチャー（前編と同じ）
dw.japan.ca_series.components.qrcode	QrcodeViewコンポーネント（前編と同じ）
dw.japan.ca_series.components.qrcode.feature	QrcodeViewコンポーネント配布用フィーチャー（前編と同じ）
dw.japan.ca_series.components.qrcode2	QrcodeView2コンポーネント（今回作成した、ローカルで完結して処理するコンポーネント）
dw.japan.ca_series.components.qrcode2.wab	上記コンポーネントが依存するローカルWebアプリケーション
dw.japan.ca_series.components.qrcode2.feature	QrcodeView2コンポーネント配布用フィーチャー
dw.japan.ca_series.compositeapps.qrcode2	コンポジット・アプリケーション
dw.japan.ca_series.compositeapps.qrcode2.feature	上記コンポジット・アプリケーション配布用のフィーチャー
qrcode.lib	Kazuhiko Araseさん提供.jarの共有ライブラリ化
qrcode.lib.feature	上記共有ライブラリ配布用のフィーチャー
dw.japan.ca_series.updatesite2	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
第2回で取り扱ったプロジェクトのサンプルファイル	dw_ca02_projects.zip	172B

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。