

# DWRでAjaxしてみよう

## ～ リッチクライアントへの古くて新しい挑戦 ～

### Ajaxって何?の前にリッチクライアントについて

**突然** ですがキーワードです。Ajax。“えいじゃくす”と読むのでしょうか?ともかく最近のリッチなUIを持つWebアプリケーションに関わるキーワードのようです。

でも、まずは“リッチクライアント”という言葉についておさらいしておきましょう。

近年、業務システムを中心によく使われていたサーバ・クライアントシステムが急速にWebアプリケーション<sup>1</sup>にリプレイスされていきました。この急激な変化の理由はなんだったのでしょうか。

Webアプリケーションというフレームワークは、これまでのサーバ・クライアントシステムにあった厄介な問題を解決してくれました。たとえばサーバ・クライアントシステムでは、クライアントソフトウェアのバージョンを管理する必要があり、システムのバージョンアップをしたら、サーバだけでなく、各クライアントのソフトウェアについてもバージョンアップ作業を行わなければなりません。こうした管理コストを考えると、Webアプリケーション化し、クライアント側の管理コストを低減させるというのは魅力的な選択肢でした。また、Webアプリケーションは開発コストを比較的抑えやすいという点も大きな原動力となったと言えるかもしれません。

もちろんWebアプリケーションは“銀の弾丸”ではありません。管理コストや開発コストの低減により、フレームワークとしては進化を遂げたわけですが、退化した部分ももちろんありました。

Webアプリケーションは、基本的なHTMLのみを用いてシステムを構築するため、HTMLで表現できない機能については処理することができません。また、同一ページ内で動的に内容を更新・変更するような仕組みを実現するに

1. クライアントとしてWebブラウザを使用し、Webサーバとの通信を行いながら処理を行っていくシステム。ここでは特にJavaアプレット・JavaScript等のブラウザの機能を使用しない、基本的なHTMLのみを用いるシステムとします。

は、毎回ポストバック<sup>2</sup>を行う必要があります。この処理は大抵秒単位の時間が掛かるため、ユーザにとってはストレスを引き起こす原因ともなります。

Webアプリケーション化によって、ユーザの操作性・利便性が犠牲となり、特に業務アプリケーションにおいてはWebブラウザが標準で持つ入力系機能の弱さが問題となってきました。そこで新たに考案されたのがリッチクライアントあるいはRIA(リッチインターネットアプリケーション)というアイデアです。

リッチクライアントは、Webアプリケーションシステムでありながら、かつてのサーバ・クライアントシステムで実現していたような“リッチな”操作性を実現する技術です。たとえばファンクションキーを使った入力や、待ち時間を感じさせない画面遷移を実現することができます。リッチクライアントの大半はWebブラウザ上のプラグインで実現されています。

しかし、そうしたリッチクライアントにも弱点があります。最大の弱点は、「Webブラウザほどに普及していない」という点です。リッチクライアントを使用するには、前述のプラグイン等の追加のソフトウェアをインストールする必要がありますし、そのソフトウェアが対応するOS・ブラウザを使用していなければなりません。対象ユーザの環境にある程度絞り込むことのできる社内システム等であればクライアントの管理コストが掛かるという弱点はあるものの対応は可能ですが、インターネットに公開するシステムの場合、ユーザ全員にソフトウェアのインストールを強いるというのは大きな弱点となってしまいます<sup>3</sup>。クライアント管理コストを低減するためのWebアプリケーションシステムで、クライアントに関する悩みが増えるという

2. WebブラウザがWebサーバに情報を送り、再度HTMLを取得して画面を描画しなおすこと。  
3. なお、Macromedia FlexやOpen LaszloのようにリッチクライアントとしてFlash Playerを用いる、という解法もあります。Flash Playerのインストール率は非常に高い(Macromediaによればインターネットユーザの97.4%がインストールしているとのこと)ため、普及度の点で大きな強みを持っています。実際リッチクライアントとしてFlashを用いるケースは大変増えています。

のは困ったものです。Ajax はこうした問題に対する一つの解となるアイデア、技術です。

## “Asynchronous JavaScript + XML” ～ Ajax ～ えいじゃっくす？

では Ajax はどんな技術を用いてリッチクライアントに必要な操作性を実現しているのでしょうか？ 発想は実にシンプルで、「ほぼすべての Web ブラウザに共通して組み込まれている機能だけを使い、追加のクライアントソフトウェア無しでリッチクライアントを実現しよう」というものです。したがって、鍵となるのは、略号の元の単語となっている “JavaScript” と “XML” という二つの技術、そして “Asynchronous” という手法です。

JavaScript と XML については説明の必要もないでしょう。今日のほとんどの Web ブラウザで追加のソフトウェア無しに使用することができ、また将来的にもサポートされ続けるだろう、という観点からしても、普及率という問題を解決しながら、標準の HTML 以上の表現力を利用することができます。

そして “Asynchronous” (非同期的な) という手法です。Web アプリケーションを使う際にイライラさせられるのは、何か操作する度にポストバックが発生して Web サーバとの通信が行われ、その間ユーザが待っていなければならない、という点です。JavaScript を用いたとしても、サーバから情報を取得する際にユーザを待たせてしまうのならば操作性の改善とは言えません。

Ajax では、XMLHttpRequest という JavaScript のオブジェクトを使用してこの問題を解決します。このオブジェクトは Web サーバとの XML ベースの通信を行います。通信終了を待たずに処理を返す非同期処理を行うことができます。通信終了時にコールバック関数が呼ばれますので、そのコールバック関数で通信後の処理を記述することで、「ボタンを押すとしばらく何もできずに待たされてしまう」ということが無くなります。

したがって Ajax を簡単にまとめると「各 Web ブラウザで共通に使用できる JavaScript を使い、ユーザの操作に応

4. 冷静にこの文章を読むと、Ajax が革新的なアイデアや新しいアーキテクチャでは無いことが分かります。むしろ非常にシンプルな、多くの開発者が自然に活用してきたベストプラクティスを再定義しただけである、とも言えます。本記事のサブタイトル「古くて新しい」にはそんな含みがあります。が、この言葉が定義されることでブームとなり、様々なソフトウェアが開発されてきたことを考えるならば、再定義に過ぎないとしても価値のある言葉ではないでしょうか。

じて Web サーバと XML ベースの非同期通信を用いて情報をやり取りし、画面全体の再描画を避けることで操作性を向上させる技術である」といえます<sup>4</sup>。

しかし、JavaScript そして XMLHttpRequest というソフトウェア部品だけを用いて、サーバと複雑な連携を行う、Ajax システムとするのはなかなか大変です。そのため、Ajax の実装を助けてくれるライブラリが幾つか開発されています。今回取り扱う DWR もその一つです。

## DWR って？

DWR は Direct Web Rendering の略で、クライアントサイドの JavaScript とサーバサイドの Java を連携させるためのオープンソースライブラリです (<http://www.getahead.ltd.uk/dwr/index.html>)。Servlet コンテナに搭載して使用します。

DWR の現バージョンである 0.6 は以下のブラウザに対応しています。一部のブラウザ (Mac OS X 上の Firefox、Safari) では非 ASCII 文字列に関する不具合があるようです。

- Firefox 1.0 (Windows) 対応
- Internet Explorer 6.0 / 5.5 (Windows) 対応
- Mozilla 1.7 対応
- Opera 8.0 / 7.5.4 (Windows) 対応
- Safari 1.3 / 1.2 (Mac) 対応

ブラウザ同士の JavaScript の非互換部分に関しては、DWR で使用される JavaScript コード内で吸収しています。リスト 1 は XMLHttpRequest を作成するコード例です。

### リスト 1 : DWR で使用される engine.js の内容 (一部)

```
if (window.XMLHttpRequest)
{
    call.req = new XMLHttpRequest(); //
    InternetExplorer 以外のブラウザでの処理
}
// IE5 for the mac claims to support
window.ActiveXObject, but throws an
error when it's used
else if (window.ActiveXObject &&
!(navigator.userAgent.indexOf('Mac') >=
0 && navigator.userAgent.indexOf("MSIE")
>= 0))
{
    call.req = new
ActiveXObject("Microsoft.XMLHTTP"); //
InternetExplorer での処理
}
```

Internet Explorer とそれ以外のブラウザで処理が異なります。では早速 DWR を用いて Ajax な Web アプリケーションを試してみましょう。

### DWR の組み込み方～サーバサイド

ここでは、サーバ側 DWR コンポーネントの Web アプリケーションへの組み込み方法を説明します。お使いの Servlet コンテナに合わせ、あらかじめ Web アプリケーションを作成しておいてください。筆者は Windows XP Professional 上の Tomcat 5.0 を使用してテストを行いました。Tomcat のポート番号は 8080、Web アプリケーション名は dwr-test とします。

#### 1. JAR ファイルのコピー

前述の <http://www.getahead.ltd.uk/dwr/index.html> にアクセスし、右側の "Download" リンクから辿って dwr.jar をダウンロードします。この JAR ファイルを組込先の Web アプリケーションフォルダの WEB-INF/lib フォルダにコピーします。

#### 2. web.xml の編集

Web アプリケーションフォルダの WEB-INF/web.xml を編集し、ルートの webapp 要素に以下の記述を追加してください。以下の記述により、/dwr/ 以下の URL については、DWR 側で処理されることとなります (リスト 2)。

#### リスト 2：web.xml への追加内容

```
<servlet>
  <servlet-name>dwr-invoker</servlet-name>
  <display-name>DWR Servlet</display-name>
  <description>Direct Web Remoter Servlet</description>
  <servlet-class>uk.ltd.getahead.dwr.DWRServlet</servlet-class>
  <init-param>
    <param-name>debug</param-name>
    <param-value>>true</param-value>
  </init-param>
</servlet>

<servlet-mapping>
  <servlet-name>dwr-invoker</servlet-name>
  <url-pattern>/dwr/*</url-pattern>
</servlet-mapping>
```

### 3. サーバ側クラスの作成

Web ブラウザから呼び出したいクラスを作成し、WEB-INF/src 以下に保存してください。特別な作法は必要なく、通常の Java コードを記述可能です。ただし、以下の 2 点に注意してください。

- ・ 引数無しのコンストラクタ (デフォルトコンストラクタ) を public とする。
- ・ 関数名に JavaScript の予約語を使用しない。基本的には Java の予約語と同じですが、Java には無い delete という予約語がありますので、使用しないように注意してください。

今回はリスト 3 のような 2 つのメソッドを定義したクラスを作成しました。この DWRTTest クラスは "DWR Test" という文字列を返す getText メソッドと、引数の値に応じて都道府県名の文字列配列を返す getPrefs メソッドを持っています。これらのメソッドを Web ブラウザから呼び出し、getText の返値を表示したり getPrefs の返値を用いてドロップダウンの内容を書き換えたりする、というのが今回のサンプルです。

### 4. dwr.xml の作成

Web アプリケーションフォルダの WEB-INF フォルダにリスト 4 のような内容の dwr.xml を作成します。これによ

#### リスト 3：サンプルクラス DWRTTest.java

```
public class DWRTTest {

  public String getText() {
    return "DWR Test";
  }

  // 都道府県リストを返すメソッド 北海道と東北のみ実装
  public String[] getPrefs(int no) {
    switch(no) {
      case 0: // 北海道だったら
        return new String[]{"北海道"};
      case 1: // 東北だったら
        return new String[]{"青森","秋田","福島","山形","岩手"};
      default: // それ以外
        return new String[]{""};
    }
  }
}
```

## リスト 4 : dwr.xml の内容

```
<dwr>
  <allow>
    <create
      creator="new"
      javascript="DWRTest"
    class="DWRTest" />
  </allow>
</dwr>
```

り、サーバサイドの Java クラスを JavaScript に対して公開することができます。

create 要素でクラスの定義を行います。class 属性に対象クラスの (パッケージ名を含んだ) 名前を、javascript 属性に JavaScript 上で扱う際の名前 (パッケージが入ると長くなりますので、任意の名前で再定義します。) を記述します。creator 属性は new ないしは session を記述することができ、new の場合は Web ブラウザからのアクセスごとに Java クラスが生成され、session の場合は Web ブラウザのセッションごとに Java クラスが生成されます。

今回は先ほど作成した DWRTest クラスを、アクセスごとに新規作成するという設定で、DWRTest という名前で JavaScript に対して公開します。

## 5. 公開されたクラスの確認

以上までの記述が済めば、JavaScript から公開された Java クラスを呼び出すことができます。まずは標準のサンプルを見てみましょう。

http://localhost:8080/dwr-test/dwr にアクセスしてみてください。図 1 のような画面が表示されます。上に公開されている Java クラスの一覧が表示されます。

DWRTest をクリックすると図 2 の画面が表示されます。DWRTest クラスの公開されているメソッドの一覧と、その呼び出し方法が表示されます。

興味深いのは、このページ自体が DWR を用いて書かれており、各メソッドをここから呼び出すことができる、という点です。たとえば、getText() の右の [Execute] ボタンを押すと、Web サーバとの通信が行われ、図 3 のように表示が変化します。

以上でサーバサイドの設定は終わりです。

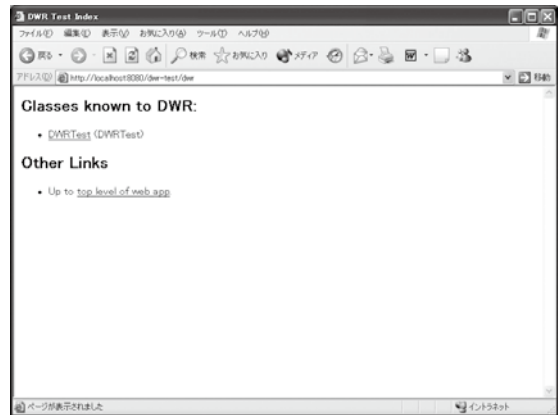


図 1 : 公開されている Java クラスの一覧

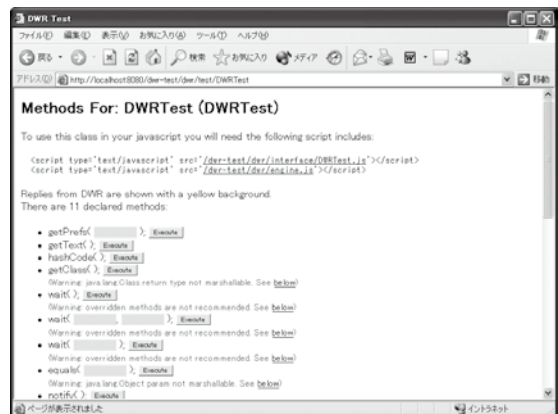


図 2 : DWRTest クラスの公開されているメソッド一覧

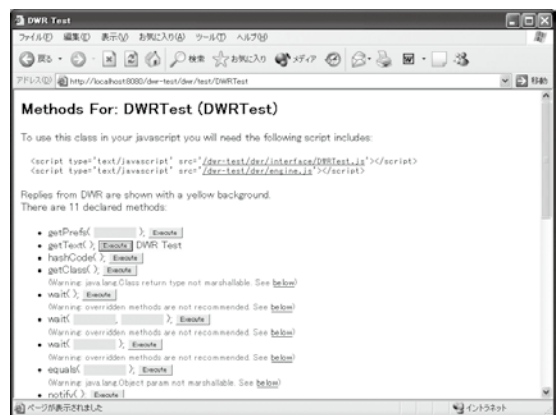


図 3 : getText() 実行後 "DWR Test" という文字列が表示されていることに注目

## DWRの組み込み方～クライアントサイド

### 1. HTML ファイルの作成

まず、DWRを組み込むためのHTMLファイルを作成します。今回はtest.htmlというHTMLファイルを作成しました。

### 2. DWR スクリプトの組み込み

DWRのライブラリをインポートします。先ほど見たメソッド一覧画面にも表示されていましたが、

```
<script type='text/javascript' src='/dwr-test/dwr/interface/DWRTest.js'></script>
<script type='text/javascript' src='/dwr-test/dwr/engine.js'></script>
```

の2行、そして以下の行をHTMLファイルに追加します。

```
<script type='text/javascript' src='/dwr-test/dwr/util.js'></script>
```

3行目は後述のDWRUtilクラスを使用するための記述です。

### 3. DWR スクリプトの呼び出し

HTML上に適宜ボタンや表示するためのコントロールを配置し、ボタンを押した時のイベントハンドラにDWRスクリプトを呼び出すようにします。では、サーバサイドで定義したgetTextメソッドとgetPrefsメソッドを呼び出してみましょう。

まずgetTextですが、サーバから取得した文字列をtextTestというIDを持つspanタグ内に表示させることにします。以下のようにボタンとspanタグを配置します。

```
<input type='button' onclick='UpdateText()' value='ExecuteText' />
<span id='textTest'></span>
```

ボタンを押した時のイベントハンドラUpdateTextは以下のように定義します。

```
<script type='text/javascript'>
function UpdateText()
{
    DWRTest.getText(setText);
}
</script>
```

DWRTestは、dwr.xmlのjavascript属性で定義したクラス名です。通常のクラスのメソッド呼び出しに見えますが、引数を持たないはずのgetTextメソッドが、setTextという

引数付きで呼び出されていることに注目してください。これは、非同期通信が終了した時にコールバック関数として呼び出されるJavaScriptのメソッド名です。そのsetTextメソッドは以下のように定義しています。

```
<script type='text/javascript'>
function setText(data)
{
    DWRUtil.setValue("textTest",data);
}
</script>
```

引数のdataにはサーバから取得したデータ(ここでは文字列)が入っています。DWRUtilはutils.jsで定義されたDWRのユーティリティクラスで、ここではtextTestというIDを持つタグにdataをセットする操作を行っています。

以上のように記述することで、サーバ側のgetTextメソッドを呼び出してそれをブラウザ上に表示させることができます。

次に、getPrefsメソッドを呼び出してみましょう。getPrefsメソッドは地方に対応した都道府県リストを返すメソッドです。地方を選択するためのドロップダウン(ID='district')と、都道府県を表示するためのドロップダウン(ID='selectTest')、そしてボタンを配置し、以下のように記述します。

```
<input type='button'
onclick='UpdateSelect()'
value='ExecuteSelect' />
<select id='district'>
<option value='0'>北海道</option>
<option value='1'>東北</option>
</select>
<select id='selectTest' />
```

ボタンを押した時のイベントハンドラUpdateSelectは以下のように定義します。

```
<script type='text/javascript'>
function UpdateSelect()
{
    DWRTest.getPrefs(setSelect,district.
value);
}
</script>
```

DWRTestクラスのgetPrefsメソッドを呼び出していますが、今回もサーバ側と引数が異なります。第一引数はコールバック関数定義、第二引数以降がサーバ側で定義した引数となります。

したがって、コールバック関数はsetSelectメソッド、サー

バの `getPrefs` メソッドへの引数は `district` ドロップダウンの値になります。`setSelect` は以下のように定義します。

```
<script type='text/javascript'>
  function setSelect(data)
  {
    DWRUtil.fillList("selectTest", data);
  }
</script>
```

引数の `data` にはサーバから取得したデータ(ここでは文字列配列)が入っています。通常であれば配列をループしながら `option` タグを作成する必要がありますが、`DWRUtil` クラスには配列から `option` タグを生成して `select` タグにセットする `fillList` というメソッドがあるので、それを使用しています。ここでは `selectTest` という ID を持つ `select` タグに `data` をセットしています。

`getText`・`getPrefs` メソッドを実行すると、それぞれ以下のように画面表示が変わります(図4~6)。

以上のような動作を、画面の再読み込み無しに、非同期に行うことができます。非常に簡潔な記述でサーバとの連携を行うことができる `DWR` のおかげですね。

## まとめ

`DWR` はまだバージョンも 0.6 で、ドキュメントもそれほど揃っていません。特に日本語の情報源はほとんど無いため、サンプルやソースコードを見ながら試行錯誤が必要な段階です。

`DWR` を用いることで直接 `JavaScript` から `XMLHttpRequest` を使う場合よりもずっと簡潔な記述とすることができます。とはいえ、`DWR` はサーバサイド `Java` への基本的なインタフェースを提供するのみであり、リッチクライアントに必要な様々な機能、たとえば `ASP.NET` で実現されているような豊富なバリデーション機能をすぐに使用できるわけではありません。今後、`DWR` を基として、サーバ技術とさらに連携を図ることができるタグライブラリなどが登場して、`Ajax` 実装環境がさらに整備されていくことを期待しましょう。

`DWR` 公式サイト<sup>5</sup>(<http://www.getahead.ltd.uk/dwr/index.html>)には、サンプルも含め有益な情報が多く掲載されて

5. `DWR` 公式サイトでは、この頃流行りの `Spring Framework` と `DWR` を連携させるための設定についてのドキュメント等もあります。`Spring` と `DWR` を連携させると `XML` ベースで色んなことができる、かなり興味深い取り組みです。

いますので、`DWR` を使用する際には目を通されることをおすすめします。ではみなさま、快適な `Ajax` ライフを `DWR` でお楽しみください!

(土井 毅)

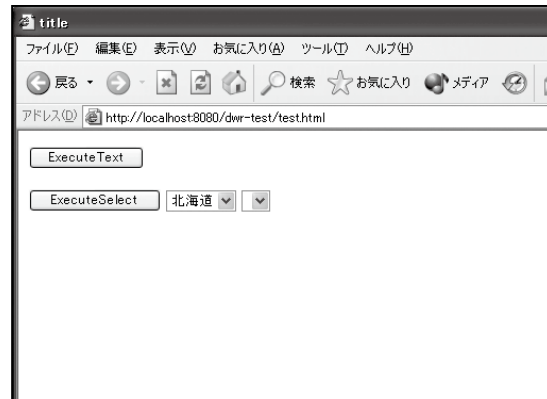


図4：実行前の画面

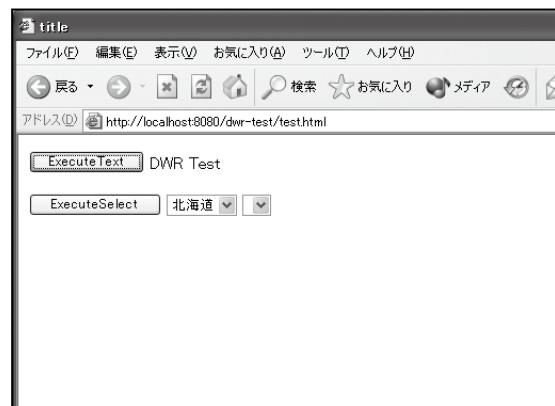


図5：`getText` メソッド実行後 “DWR Test” が表示されている



図6：“東北”を選択して `getPrefs` メソッド実行 都道府県がドロップダウンにセットされている