XWork2 特徴

http://wiki.opensymphony.com/display/XW/XWork+Features

XWork について

XWork は、コマンドパターンのフレームワークで、Struts2 の コアとして利用されている。

・設定方法が、シンプルであるため、柔軟でカスタマイズし易い。設定を XML に記述することや、プログラムに記述することもできる。
・コアコマンドパターンフレームワークは、どんな request/response にも、interceptors を利用して、カスタマイズや拡張が可能。
・組み込みの型変換や、action プロパティのバリデーションは、OGNL を利用する。
・強力なバリデーションフレームワークは、ランタイムの属性や、validation interceptor によって実現される。

概観

Xwork は、Action インターフェースを集めた、コマンドパターンフレームワーク。
Action インターフェースを実装したアクションクラスを定義すると、XWork は、それらをセットアップし、実行する。XWork は、Webwork と呼ばれる web MVC フレームワークとして広く知られている。しかしながら XWork は、それ自身単独で使用できるため、XWork レイヤーおよび、どのようにアクションがセットアップされ実行されるのかを理解することが重要である。

・Action インターフェース
・ActionProxy インターフェース
・ActionInvocation インターフェース
・ActionContext
・簡単な例

```
ActionProxy ------- ActionInvocation ------> Action
```

Actions

Action は、実行の基本単位

Action インターフェース

基本インターフェースである、すべての XWork アクションは実装され、いくつかの標準の結果 (SUCCESS、INPUT など) を提供する。またひとつのメソッドのみを持つ。
Action.java

```
public interface Action {
    public static final String SUCCESS = "success";
    public static final String NONE = "none";
    public static final String ERROR = "error";
    public static final String INPUT = "input";
    public static final String LOGIN = "login";

    public String execute() throws Exception;
```

たいていは、Action クラスは、単に com.opensymphony.xwork.ActionSupport を継承すればよい。

ActionSupport クラスは Action インターフェースを実装し、多くの Action に共通な基本的な振る舞いを提供する。

ActionProxy

Action のライフサイクルは、ActionProxy を通じて、保守される。ActionProxy は Xwork API の最上位のレイヤーで、アクションのセットアップや、実行の開始点となる。XWork は、ActionProxy をインスタンス化するためのエントリポイントとなるファクトリを提供する。それぞれの xwork レイヤーの多くの実装は、インターフェースの背後に隠されており、規定の実装をオーバーライドすることにより、とても簡単にカスタマイズすることができる。

デフォルトの ActionProxy を取得する例

```
ActionProxyFactory.getFactory().createActionProxy("", "viewBook", objectMap);
```

自分自身で実装した ActionProxywo 登録する必要があれば、factory で、次のように。

```
class CustomizedActionProxyFactory extends DefaultActionProxyFactory{
  createActionProxy(...){ return new CustomizedActionProxy(...); }
}


ActionProxyFactory.setFactory(new CustomizedActionProxyFactory());
ActionProxy proxy = ActionProxyFactory.getFactory().createActionProxy(...);
```

ActionInvocation

ActionProxy レイヤーの下層には、AvctionInvocation インターフェースがあります。ActionInvocation はアクションのインスタンスのが保持するアクションの状態とともに、処理の前後にラップされた、interceptors を表現しています。

簡単な例

最初に、単純な JavaBean による例

```
public class Book {
    String id;
    String title;
    Set authors;
    public void setId(id){ this.id = id; }
    public void setTitle(String title){ this.title = title; }
    public void setAuthors(Set authors){ this.authors = authors; }
    public String getId(){ }
    public String getTitle{ }
    public Set getAuthors{ }
}
```

データソースから book オブジェクトを参照し、呼び出し元に返す必要がある場合、これを取り扱うアクションを書くことができる。xwork のアクションは概してとても単純なクラスとなる。Action インターフェースを実装する要件はたった一つである。execute メソッド (バリデーション、型変換などが分離された) をもった、javabean と同じくらい単純なアクションとなります。アクションを実行する目的はたいてい、データへのアクセスや、操作を提供することでしょう。アクションの実行結果は、アクションの実行後の状態を表す、単純な文字列表現です。なので、結果は success 文字列、failure 文字列、forward 文字列などとなります。以下の例では、ある book オブジェクトがアクションによりみつけられたら、"sucess "、見つけられなければ、"notFoud" が返されます。これにより、book が見つからない場合、別の目録へリクエストをフォワードしたり、book

オブジェクトを返すためにセットアップしたりといったコントロールが簡単にできます。

com.opensymphony.xwork.example.ViewBookAction

```
public class ViewBookAction  implements Action{
    Book book;
    String id;

    public String execute() throws Exception{

        // lets pretend we have a data access object that will return a book from storage
        book = bookDAO.findById(id, Book.class);
        if(book != null) return "success";
        return "notFound";
    }
    public Book getBook(){ return this.book; }
    public setId(String id){this.id = id; }
}
```

簡単なモデルを定義するアクションを定義し、アクションプロキシをセットしアクションを実行してみましょう。
XWork をセットアップし、アクションを実行します。

```
Map paramMap = new HashMap();
paramMap.put("id", "0123456789");

// set the ActionContext parameters
Map context = new HashMap();
context.put(ActionContext.PARAMETERS, paramMap);

// create an action proxy with no namespace, action alias (defined in xwork.xml), and a map of the
context info
ActionProxy proxy = ActionProxyFactory.getFactory().createActionProxy("","viewBook", context);

// we have the action proxy instance, lets execute it and retrieve the action
String result = proxy.execute();
if ("success".equals(result)){
   ViewBookAction action = (ViewBookAction) proxy.getAction();

   // return info back to caller or just print to screen for this example
   System.out.println(action.getBook().getTitle());
} else if("notFound".equals(result){
   // forward to another inventory source
} else {
   throw new RuntimeException("Im lazy");
}
```

これだけで終わりではありません。いくつかの設定を xwork.xml にする必要があります。そうすることで、XWork は、createActionProxy(...) メソッドで、利用するエイリアスに対応する適切なクラスを見つけ出すことができます。

```
<xwork>
    <include file="xwork-default.xml"/>
    <package name="default" extends="xwork-default">
       <action name="viewBook" class="com.opensymphony.xwork.example.ViewBookAction"/>
    </package>
</xwork>
```
