

Mar 07 Construindo aplicações Flex usando AMFPHP

Postado em : Sábado, 7 de Março de 2009 | Autor: Paulo Teixeira

Estou escrevendo um sistema em Flex + PHP, nesse sistema estou fazendo VO das classes PHP com as classes AS3 usando o AMFPHP para isso.

Com isso, resolvi escrever uma basicão de como fazer essa integração pois vejo muitos usuários de PHP interessados em Flex, mas sem saber por onde começar. Então vamos lá:

Primeiro postarei a estrutura de pastas que estou usando no Flex e no PHP:

FLEX:

- **c:/FlexProjects/root/src**
 - **controllers**
 - **models**
 - **remote**
 - **views**
 - **vo**

Detalhando a estrutura do Flex, eu divido meus projetos nessas quatro partes:

Controllers - Onde eu mantenho minhas classes de interação, onde manipulam alguma coisa ou faz a troca de mensagens com o PHP.

Models - Onde eu mantenho as classes de estrutura, essas classes não fazem acesso com minhas classes remotas, elas servem como base para as classes de interação, é através dessas classes que os controllers recebem as informações para manipular os itens de tela.

Remote - No package remote eu tenho as classes que fazem o acesso as classes do PHP. São reponsaveis pelo envio das mensagens para o PHP e fazem o Responder.

Views - Na camada view eu tenho os meus componentes Flex, seja em action script ou mxml.

VO - Onde eu tenho meus **VO's**, classes que espelham as classes de **VO** do PHP.

PHP

- **c:/Apache/htdocs/root**
 - **controllers**
 - **models**
 - **vo**

No PHP a estrutura é meio diferente, pois eu uso um Framework que eu desenvolvi para abstração de dados. Esse framework me ajudar a escrever sistemas com diferentes bases de dados sem ter que escrever mais de uma versão de softwares. O mesmo sistema funciona em MySQL, Oracle, MSSql e PGSql, por isso minha estrutura é meio diferente.

Controllers - Eu tenho as classes que serão acessadas pelas classes do package remote do Flex.

Models - Na camada model eu tenho o meu framework, não mostrarei detalhes do meu framework pois não vem ao caso, nas minhas classes controllers verão que eu não uso comandos SQL's, somente chamarei métodos que já me retornarão os **QueryResults**.

VO - Onde eu tenho os **VO's** do PHP que servirão como ponte de informação com as classes **VO's** do PHP.

A estrutura de desenvolvimento é essa acima, agora vamos aos códigos:

Nesse exemplo eu demonstrarei como buscar os dados numa base para preencher um datagrid: Começando pelo PHP, temos no package VO a classe notificationVO.php

vo/notificationVO.php

```
<?php
class notificationVO
{
    public $UserId;
    public $NotificationId;
    public $NotificationTitle;
    public $NotificationFrom;
    public $NotificationDate;

    public $_explicitType = "vo.notificationVO";
}
?>
```

Uma coisa que deverá ser notada é que o tipo das propriedades retornadas no PHP devem ter o mesmo

tipo das propriedades recebidas no PHP, nesse caso, eu retorno datas do SQL formatadas e com o tipo String para isso como não preciso filtrar nada por data, caso tenha que fazer, temos que retornar como data mesmo.

Agora faremos a classe que executará uma consulta no banco e retornará um array com os dados recuperado.

controllers/notification.php

```
<?php
require_once('../fwQueryBuilder.class.php');
require_once('Notification.class.php');

/**
 * Classe de notificação de usuários
 * @name Notification
 * @todo Faz a notificação automática dos usuários
 * @version 1.0
 * @author Infinitum Technologies
 */
class Notification
{
    /* PRIVATES VARIABLES */
    private $Query;
    private $Notificacao;
    private $tableData;
    private $rsData;
    /* PUBLIC VARIABLES */
    public $UserId;
    public $AuthenticationId;
    public $NotificationId;
    public $NotificationTitle;
    public $NotificationFrom;
    public $NotificationDate;

    public function __construct()
    {
        $this->Query = new queryBuilder(); // Notificacao acessa uma
        classe que faz ligação com o meu framework e efetua as consultas no banco
        // esse acesso você pode substituir por uma consulta ao banco
        $this->Notificacao = new Notificacao();
    }

    /**
     * Notificações de usuários
     * @name GetUserNotification
     * @access Public
     * @param Array ArrayData
     * @return Query
     */
    public function GetUserNotification($arrDados)
    {
        // instancia as propriedades de VO
        $dados = new notificationVO();
        $dados->UserId = $arrDados["UserId"];

        // executa o método de consulta que retorna os dados do framework
        $this->rsData = $this->Notificacao->GetNotificationUser($dados->UserId);

        // distruo os dados recebidos como parametro
        unset($dados);
        // crio um index e populo as propriedades com os dados retornados
        $idx = 0;
        while($dadosResult = $this->Query->Fetch($this->rsData))
        {
            $dados[$idx] = new notificationVO();
            $dados[$idx]->NotificationTitle = $dadosResult['notification_title'];
            $dados[$idx]->NotificationId = $dadosResult['notification_id'];
            $dados[$idx]->NotificationDate = $dadosResult['notification_date'];
            $dados[$idx++]->NotificationFrom = $dadosResult['notification_from'];
        }

        // em caso de não ter retorno do banco crio uma instancia de
        notificationVO para não retornar algo em branco
        if(!isset($dados))
            $dados = new notificationVO();

        return $dados;
    }
}
```

```
}  
?>
```

Bom o trabalho do PHP está finalizado, lembrando que estou usando AMFPHP, então no meu servidor está devidamente configurado e meu código PHP está na pasta service do amf.

Vamos agora ao Flex:

Começando pela VO do Flex que terá o mesmo nome e as mesmas propriedades do VO do PHP. A diferença é que no VO do Flex implementaremos os Getters and Setters das propriedades, vejamos:

vo/notificationVO.as

```
package vo{  
    [Bindable] [RemoteClass(alias="vo.notificationVO")]  
  
    public class notificationVO  
    {  
  
        private var _UserId:int;  
        private var _NotificationId:int;  
        private var _NotificationTitle:String;  
        private var _NotificationFrom:String;  
        private var _NotificationDate:String;  
  
        public function get UserId() : int  
        {  
            return _UserId;  
        }  
        public function set UserId(value : int) : void  
        {  
            _UserId = value;  
        }  
  
        public function get NotificationId() : int  
        {  
            return _NotificationId;  
        }  
        public function set NotificationId(value : int) : void  
        {  
            _NotificationId = value;  
        }  
  
        public function get NotificationTitle() : String  
        {  
            return _NotificationTitle;  
        }  
        public function set NotificationTitle(value : String) : void  
        {  
            _NotificationTitle = value;  
        }  
  
        public function get NotificationFrom() : String  
        {  
            return _NotificationFrom;  
        }  
        public function set NotificationFrom(value : String) : void  
        {  
            _NotificationFrom = value;  
        }  
  
        public function get NotificationDate() : String  
        {  
            return _NotificationDate;  
        }  
        public function set NotificationDate(value : String) : void  
        {  
            _NotificationDate = value;  
        }  
    }  
}
```

remote/notificationRemote.as

```
package remote{  
    import mx.controls.Alert;  
    import mx.rpc.AsyncToken;
```

```

import mx.rpc.responder,
import mx.rpc.events.FaultEvent;
import mx.rpc.remoting.mx.xml.RemoteObject;

import vo.notificationVO;

public class NotificationRemote
{

    private var notificationRO:RemoteObject;

    private static var instance:NotificationRemote;
    public static function getInstance():NotificationRemote
    {
        if(instance == null){
            instance = new NotificationRemote();
        }
        return instance;
    }

    public function GetNotificationsUser(dados:notificationVO,
result:Function):void
    {
        var async:AsyncToken = notificationRO.GetUserNotification(dados);
        async.addResponder(new Responder(result, defaultFaultHandler));
    }

    private function defaultFaultHandler(e:FaultEvent):void{
        Alert.show(e.fault.faultDetail, e.fault.faultString);
    }

    public function NotificationRemote(){
        notificationRO = new RemoteObject();
        notificationRO.showBusyCursor = true;
        notificationRO.destination = 'amfphp';
        notificationRO.source = 'root._controllers.Notification';
    }

}
}

```

Com a classe e o método acima acessamos a classe PHP e recebemos os dados recuperados no banco de dados. Agora mostrarei a classe e o métodos que executa o chamado ao remoteObject acima para receber os dados.

views/telaNotificacao.mxml

```

<?xml version="1.0" encoding="utf-8"?>
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="790" height="582"
creationComplete="init();" >
    <mx:Script>
        <![CDATA[
            import mx.containers.Grid;
            import mx.core.Application;
            import mx.events.FlexEvent;
            import mx.rpc.events.ResultEvent;
            import mx.collections.ArrayCollection;

            import remote.NotificationRemote;

            private var oRemote:NotificationRemote =
NotificationRemote.getInstance();

            [Bindable] private var arrProvider:ArrayCollection;
            [Bindable] private var arrNotifications:Array = new Array();

            private function init():void
            {
                // passar o id do usuário logado como parametro
                this.GetNotification(int(1));
            }

            // notificação =====
            private function GetNotification(UserId:int):void
            {
                var dados:notificationVO = new notificationVO();
                dados.UserId = UserId;
                oRemote.GetNotificationsUser(dados,notificationHandler);
            }
        ]]]>
    </mx:Script>
</mx:Canvas>

```

```

        private function notificationhandler(e:ResultEvent):void
        {
            arrNotifications = e.result as Array;
            arrProvider = new ArrayCollection(arrNotifications);
            dgNotification.dataProvider = arrProvider;

            txtNumNotification.text = ("+"arrNotifications.length+"");
            notificacaoRO = null;
        }
    ]]>
</mx:Script>
<!-- grid de notificações -->
<mx:Canvas x="10" y="68" width="770" height="174"
borderStyle="solid" borderColor="#E3E3E3">
    <mx:DataGrid id="dgNotification" x="10" y="13" width="750"
height="131" alternatingItemColors="[#F8EA50, #FCFDAB]">
        <mx:columns>
            <mx:DataGridColumn headerText="Id"
dataField="NotificationId" width="30"/>
            <mx:DataGridColumn headerText="Título"
dataField="NotificationTitle"/>
            <mx:DataGridColumn headerText="Remente"
dataField="NotificationFrom" width="120"/>
            <mx:DataGridColumn headerText="Data"
dataField="NotificationDate" width="80"/>
        </mx:columns>
    </mx:DataGrid>
</mx:Canvas>
<!-- barra do título -->
<mx:ApplicationControlBar x="10" y="42" width="770"
id="appbarNotifications">
    <mx:Text id="txtNotificationtittle" text="Notificações
Automáticas"/>
    <mx:Text id="txtNumNotification" text="(0)"/>
</mx:ApplicationControlBar>
</mx:Canvas>

```

Nesse caso eu não precisei usar a classe de interação e nem usar algum método do model no flex, pois fiz diretamente do componente para facilitar. Mas poderia implementar os métodos numa classe do package model e chamá-las do componente.

Mas o post já está ficando grande de mais. E como a ideia era apenas mostrar um pouco dessa integração com o PHP fiz de um modo mais rápido.

Bom gente, não testei e não sei se está faltando alguma coisa, caso tenha algum problema favor postem um comentário ou mandem-me um e-mail que eu arrumo.

[].s.