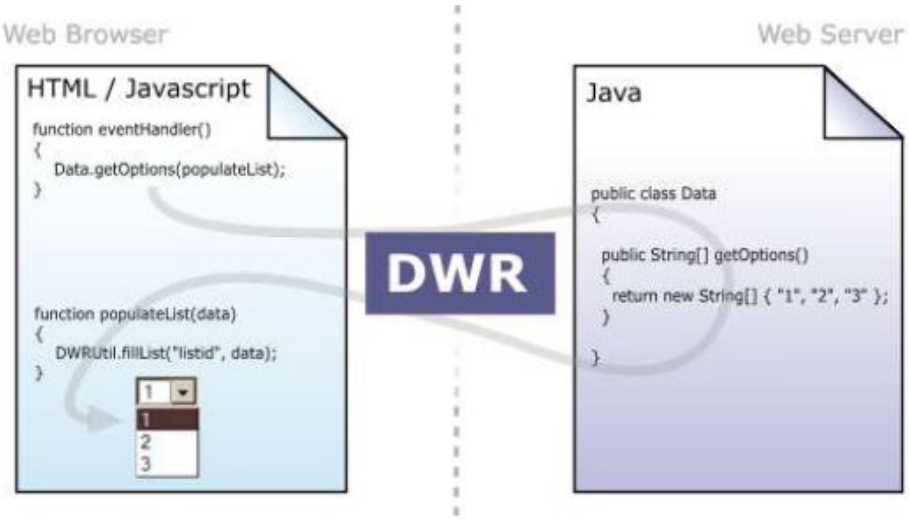
DWR – DIRECTED WEB REMOTING

Vamos ver nesse artigo um pouco sobre o Frameworks Ajax para Java, o DWR. Vamos ver seus conceitos, utilidades, vantagens, algumas práticas e fazer o passo a passo para baixar, configurar e executar algumas de suas funcionalidades. Na primeira parte vamos abordar todo o seu conceito, instalação, configuração e um exemplo simples, na segunda vamos aumentar mais o nível de complexidade dos exemplos, com acesso ao banco, montagem de combos e etc, na terceira parte vamos mostrar algumas possíveis soluções de otimização e funções de tratamento de exceção. Espero que esse conteúdo seja de utilidade a todos.

Parte 1 – Conceitos, Instalação, Configuração e Exemplo.

Primeiro vamos saber o que é o DWR.

DWR nada mais é que um frameworks Ajax para Java, com funcionalidades que facilitam a sua vida na hora de desenvolver alguma aplicação que precise utilizar Ajax para executar seus métodos de sua classe Java e tem além de suas funções, uma que se destaca e é sua principal utilidade: Integração com as classes Java de dentro do próprio JavaScript(JSP/JS ←→ JAVA). Essa é sua principal importância, fazer com que você execute métodos das suas classes de dentro do próprio JavaScript com apenas uma única linha.



(Imagen 1)

Vantagens de se utilizar o DWR.

1.Fácil integração entre Java e JavaScript

Com apenas uma simples configuração no xml, você define qual classe vai ser utilizada como um "objeto" Javascript.

2.Integração Spring Beans;

Simples integração com componentes Spring Beans.

3.Creators configuráveis: new, session, spring static;

Você pode escolher o tipo de objeto JavaScript que você quer criar para suas classes.

4. Debugging no console;

O DWR tem a opção de ativar a exibição de suas chamadas para exibir no console do Tomcat ou do Eclipse

5. Real possibilidade de manter o seu padrão MVC;

Se você esta utilizando na sua arquitetura o padrão MVC o DWR pode facilmente se integrar e não prejudicar a sua camada. Criando apenas mais uma camada, que seria do JS.

6. Integração com os principais Frameworks Java;

O DWR vem com integração para: Spring, Struts, JSF, WebWorks, Hibernate, Pageflow – Beehive / Weblogic, Servlet Objects.

Desvantagens ?

Bem na minha opinião é que a sua única desvantagem(que pra mim não influi) é que ele só funciona dentro da plataforma Java.

Vamos primeiro criar um projeto web simples e baixar o dwr.jar.

Entendendo a configuração no contexto global.

Para se iniciar com DWR você deverá seguir alguns passos simples.

1. *Copiar o dwr.jar para a sua pasta WEB-INF/lib da sua aplicação;*
2. *Configurar o seu web.xml para a sua aplicação "enxergar" o DWR;*
3. *OPCIONALMENTE criar o dwr.xml;*
4. *Incluir as bibliotecas .js do DWR nas suas páginas JSP ou HTML.*

Com o nosso projeto WEB criado vamos agora "instalar" o DWR na aplicação.

Vamos baixar o dwr.jar(http://getahead.ltd.uk/dwr/download), e coloca-lo na pasta lib do WEB-INF do seu projeto. Estamos utilizando a versão 1.1.3 para esse artigo.

Copie o dwr.jar para dentro da pasta lib do seu WEB-INF que NO MEU CASO como estou utilizando o MyEcplise então vai estar localizado em: *[?]:\[SeuWorkSpace]\PaletrasDWR\web\WEB-INF\lib, se você utiliza outro plugin é só coloca-lo na pasta lib do seu WEB-INF.*

Pronto agora clique em cima do seu projeto dentro do eclipse e aperte F5 para atualizar o seu projeto, o MyEclipse vai atualizar automaticamente a sua lib, mais caso isso não aconteça, ainda com o seu projeto selecionado clique com o botão direito em cima dele e selecione Properties, depois vá em *Java Build Path*, e na aba *Libaries*, depois selecione *Add JARs...* e selecione o dwr.jar e dê OK.

Agora vamos configurar o **WEB.XML.**

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
    xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">


<display-name>DWR - Ajax Frameworks for Java. Por: Handerson Frota</display-name>
  <description>
     Sistema para o artigo: DWR - Ajax Frameworks for Java, para o Portal Java Magazine. Por: Handerson Frota
  </description>

  <!-- Servlet de invocação dwr -->
    <servlet>
        <servlet-name>dwr-invoker</servlet-name>
        <servlet-class>uk.ltd.getahead.dwr.DWRServlet</servlet-class>
        <init-param>
            <param-name>debug</param-name>
            <param-value>true</param-value>
        </init-param>
    </servlet>

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

</web-app>
```

(Imagem 2)

<u>Explicando as tags do XML:</u>

Primeiro você vai apontar o Servlet do DWR e criar um servlet name para ele, para que possamos mapear.

```
13  <!-- Servlet de invocação dwr -->
14    <servlet>
15        <servlet-name>dwr-invoker</servlet-name>
16        <servlet-class>uk.ltd.getahead.dwr.DWRServlet</servlet-class>
17        <init-param>
18            <param-name>debug</param-name>
19            <param-value>true</param-value>
20        </init-param>
21    </servlet>
```

(Imagem 3)

Agora vamos mapear o servlet:

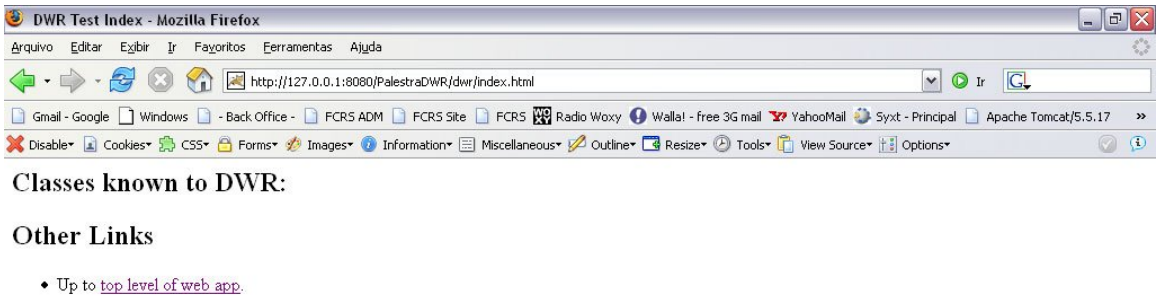
```
23    <!-- Mapeamento dwr -->
24    <servlet-mapping>
25        <servlet-name>dwr-invoker</servlet-name>
26        <url-pattern>/dwr/*</url-pattern>
27    </servlet-mapping>
28
29 </web-app>
```

(Imagem 4)

Agora o DWR já esta instalado na nossa aplicação e já esta pronto para o uso. Vamos entender que antes de tudo o DWR é um frameworks Ajax , ou seja, ele vai trabalhar com JavaScripts, isso quer dizer que, se você não quiser utiliza-lo para referenciar classes Java ele vai poder ser usado da mesma forma, só que claro sem a integração com Java, mais você poderá utilizar alguns de seus recursos como: montar grid, montar uma combo, popular divs, span, tbody e etc, isso explica o OPICIONAL para a criação do dwr.xml que iremos ver mais na frente.

Inicie o Tomcat para podermos rodar a nossa aplicação.

Entre com a url para acessar a sua aplicação: http://127.0.0.1:8080/PalestraDWR/dwr/, se o DWR foi instalado corretamente a seguinte página será carregada.



(Imagem 5)

Nesse momento, se você quiser já poderá utilizar o DWR nos seus JSP's ou HTML's, bastando apenas importar as bibliotecas JS nas respectivas páginas (ver imagem 13).

Agora vamos ver como utilizar o DWR para acessar uma classe JAVA.

Criando uma classe JAVA.

O nosso sistema de exemplos vai utilizar uma conexão com o banco de dados MySql, e vamos ter duas tabelas: PESSOA(nome, endereço, cidade, telefone) e USUARIO(id, login, senha). Então além de uma classe com os nossos métodos vamos criar também um Bean que represente cada tabela e suas propriedades. Mais para esse primeiro artigo ainda não vamos utilizar essa tabela no banco, fica apenas como adiantamento e justificativa para a criação desses Beans.

FacadeAjax

Estamos criando uma classe simples, inicialmente com um método que vai retornar uma String.

FacadeAjax.java

```
1  package com.celulajava.palestradwr.facadeAjax;
2
3
4  public class FacadeAjax {
5
6      public String getMensagem() {
7
8          String msg = "Paletra Ajax e DWR,<br><font color=#ff0000>Por: Handerson Frota.</font>";
9
10         return msg;
11     }
12
13
14 }
15
```

(Imagem 6)

Foi criada uma classe simples, dentro da estrutura de pacotes do nosso projeto: com.celulajava.palestradwr.facadeAjax. Nos vamos nomear essa classe de FacadeAjax, pois ela fará o papel de Facade do Java para acessar as outras camadas da sua aplicação. Então vai funcionar da seguinte estrutura.

JSP ou HTML/JS ←→ FacadeAjax ←→ Camada de negocio ou o seu Controller←→ Persistência(DAOs)←→ SGBD

Agora vamos criar um Bean para utilizar-mos na nossa aplicação.

Bean: Pessoa

O nosso bean Pessoa vai representar a tabela Pessoa, vamos criá-lo dentro da seguinte estrutura de pacotes: `com.celulajava.palestradwr.bean`

Pessoa.java

```java
package com.celulajava.palestradwr.bean;

/**
 * Bean Armario Classe que representa um objeto da tabela ARMARIOS, que é
 * equivalente aos Armarios.
 *
 */
public class Pessoa {

    private String nome;

    private String endereco;

    private String cidade;

    private String telefone;

    public String getCidade() {
        return cidade;
    }

    public void setCidade(String cidade) {
        this.cidade = cidade;
    }

    public String getEndereco() {
        return endereco;
    }

    public void setEndereco(String endereco) {
        this.endereco = endereco;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getTelefone() {
        return telefone;
    }

    public void setTelefone(String telefone) {
        this.telefone = telefone;
    }


}
```

(Imagem 7)

Bean: Usuário

O nosso bean Usuario vai representar a tabela Usuário, vamos criá-lo dentro da seguinte estrutura de pacotes: `com.celulajava.palestradwr.bean`

Usuario.java

```
1  package com.celulajava.palestradwr.bean;
2
3  public class Usuario {
4
5      private int id;
6
7      private String login;
8
9      private String senha;
10
11     public int getId() {
12         return id;
13     }
14
15     public void setId(int id) {
16         this.id = id;
17     }
18
19     public String getLogin() {
20         return login;
21     }
22
23     public void setLogin(String login) {
24         this.login = login;
25     }
26
27     public String getSenha() {
28         return senha;
29     }
30
31     public void setSenha(String senha) {
32         this.senha = senha;
33     }
```

(Imagem 8)

Com a classe FacadeAjax e os Beans criados, vamos agora criar o objeto Java para o JavaScript, com isso precisaremos criar o arquivo dwr.xml dentro da pasta WEB-INF no mesmo nível que esta o web.xml.

Esse arquivo dwr.xml vai servir para mapearmos as classes Java para que elas possam ser reconhecidas e utilizadas no JavaScript. Veja abaixo:

dwr.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE dwr PUBLIC
    "-//GetAhead Limited//DTD Direct Web Remoting 1.0//EN"
    "http://www.getahead.ltd.uk/dwr/dwr10.dtd">
<dwr>
    <allow>
        <create creator="new" javascript="FacadeAjax">
            <param name="class" value="com.celulajava.palestradwr.facadeAjax.FacadeAjax" />
        </create>
        <convert converter="bean" match="com.celulajava.palestradwr.bean.*"/>
    </allow>
</dwr>
```

(Imagem 9)

Explicando as tags:

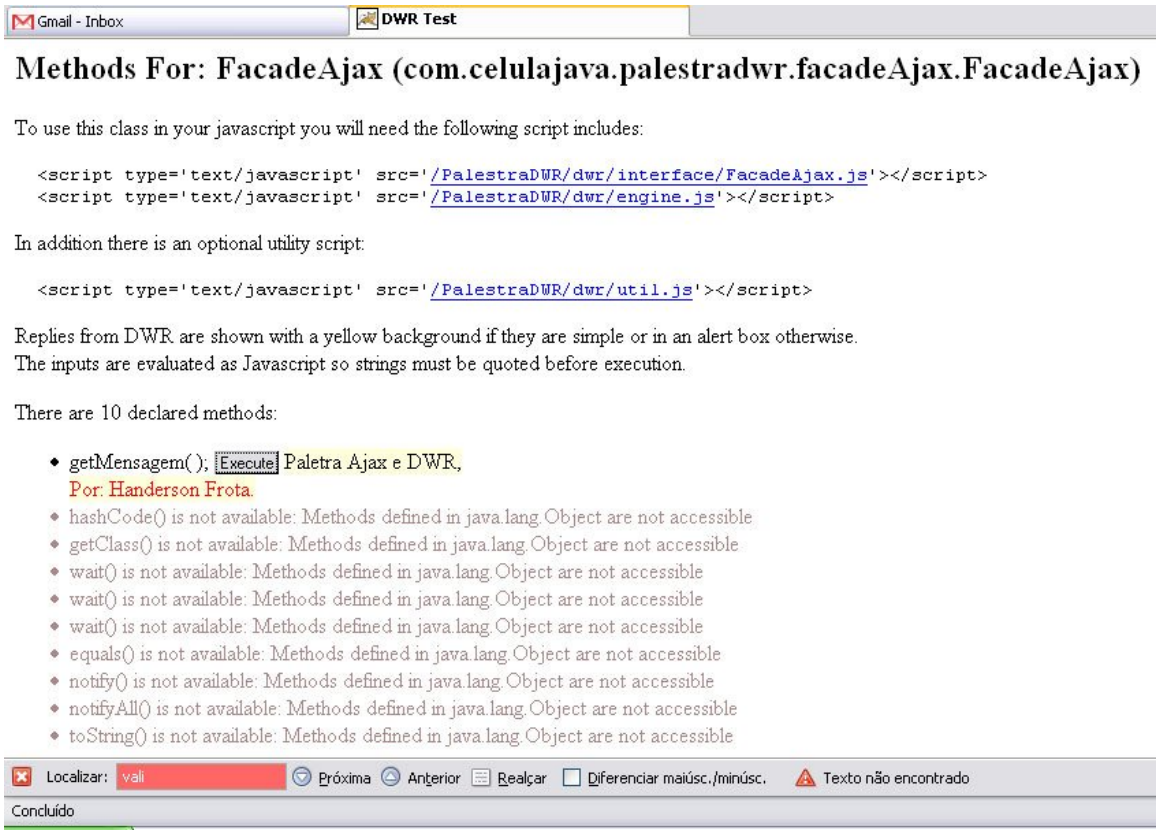
1. **Allow<allow>:** Esta tag é onde você define qual classe você vai mapear para o JS.
2. **Create<create>:** Nesta tag você vai definir o tipo de creator(new, session, struts, spring e etc) e o nome do objeto JavaScript.
    a. **creator –** *Tipo de creator do Objeto JS/Java.*
    b. **javascript –** *Nome do Objeto JS/Java.*
3. **Param<param>:** É onde você especifica a classe que você quer utilizar.
    a. **name –** *Tipo.*
    b. **value –** *Endereço completo da sua Classe.*
4. **Convert<convert>:** Nessa tag você define quem serão os seus beans.
    a. **converter –** *Tipo.*
    b. **Match –** *Endereço completo do pacote do seu bean ou o próprio bean.*

Inicie novamente o Tomcat para que as novas configurações do xml sejam reconhecidas.

Acesse novamente a url do seu projeto: http://127.0.0.1:8080/PalestraDWR/dwr/

(Imagem 10)

Agora o DWR já reconhece a sua classe Java. Se você clicar no link da sua classe você vai visualizar todos os métodos dela, e isso é muito útil, pois assim você já pode testar o seu método antes mesmo de criar seu código JavaScript. Veja abaixo:



(Imagem 11)

Se você clicar em EXECUTE o DWR vai executar o seu método. Pronto seu JavaScript já esta acessando a sua classe Java.

Vamos agora criar o nosso arquivo Java Script, vamos chamá-lo de ***exemplo.js*** e vamos criar uma pasta chamada JS onde iremos salva-lo.

Exemplo.js

Vamos criar uma função para acessarmos o método da nossa classe.

exemplo.js

```
1  function getMsg(){
2      FacadeAjax.getMensagem(exibirMsg);
3  }
4
5  function exibirMsg(mensagem){
6      DWRUtil.setValue("divMensagem", mensagem);
7  }
8
9  function apagaMsg(){
10     DWRUtil.setValue("divMensagem", "");
11 }
```

(Imagem 12)

Eu criei três funções no meu js, uma que vai acessar o meu método da minha classe Java, outra que vai exibir esse retorno dentro do meu JSP ou HTML e a última que apaga a minha mensagem da tela.

Antes de mostrarmos o nosso JSP vamos dá um breve resumo da primeira e segunda função.

Observem que na minha função getMsg() eu chamo o meu Objeto JS/Java que faz a chamada do meu método getMensagem(), da minha classe FacadeAjax.

Mais então porque estou passando parâmetro?
Primeiro o que eu estou passando não é um parâmetro e sim uma função que vai ser a função de retorno do meu método.
Simples, no Js essa é uma utilização bem comum, pode ser chamada de *Função de Callback,* o que ela faz exatamente é equivalente a você fazer o seguinte trecho.

DWRUtil.setValue(“divMensagem”, FacadeAjax.getMensagem());
ou
var msg = FacadeAjax.getMensagem();
exibirMsg(msg);

Só que no JS isso praticamente não existe, então uma saída que temos é usar o método de callback, que consiste fazer a chamada da minha função de retorno, direto na assinatura da minha função principal, por isso o nome *callback.*

E se o meu método estiver esperando um parâmetro ?
Essa metodologia se resume a:

Função(callback, parametro1); → Função espera um parâmetro.
Função(callback, parametro1, paremetro2); → Função espera dois parâmetros.

Se a sua função espera um parâmetro, basta você inseri-lo logo após a sua função de callback. Lembrando que existem outras formas de callback.
Bem acho que deu para entender, então vamos adiante para não sairmos do foco.

Resumindo você já configurou o web.xml, dwr.xml, criou a classe FacadeAjax, os beans Pessoa e Usuário e o JS exemplo.js.

Falta agora o JSP para chamarmos a nossa função e rodar esse exemplo.

Criando um JSP

Index.jsp

Vamos criar um JSP chamado index.jsp e vamos salva-lo na raiz do nosso projeto.
Veja abaixo:

index.jsp

```
1  <html>
2      <head>
3          <script type="text/javascript" src='<%=request.getContextPath() %>/dwr/interface/FacadeAjax.js'></script>
4          <script type="text/javascript" src='<%=request.getContextPath() %>/dwr/engine.js'></script>
5          <script type="text/javascript" src='<%=request.getContextPath() %>/dwr/util.js'></script>
6          <script type="text/javascript" src='js/exemplo.js'></script>
7          <title>DWR: Ajax for Java - Handerson Frota - Portal Java Magazine</title>
8      </head>
9
10     <body>
11         Executando o exemplo.getMsg(), FacadeAjax.getMensagem().
12
13         <a href="#" onclick="javascript:getMsg();">Exibir mensagem</a> |
14         <a href="#" onclick="javascript:apagaMsg();">Apaga Mensagem</a>
15         <br>
16         <div id="divMensagem"></div>
17     </body>
18 </html>
```

(Imagem 13)

Explicando o código:

```
1. <%=request.getContextPath() %>
```

Um script que vai nos retornar o nome do nosso contexto web, no nosso caso vai retornar a string: /PalestraDWR, para montar o link do import do DWR. Você poderá colocar na “mão” caso deseje.

```
2. <script type="text/javascript" src='<%=request.getContextPath()
%>/dwr/interface/FacadeAjax.js'></script>
```

Nessa linha você esta informando o seu objeto JavaScript que você mapeou no dwr.xml para representar a sua classe Java. Observe que temos o seguinte estrutura de endereço: /Contexto/dwr/**interface**/seuObjetoJS.js

```
3. <script type="text/javascript" src='<%=request.getContextPath()
%>/dwr/engine.js'></script>
```

Agora estamos importando a biblioteca engine.js(http://getahead.ltd.uk/dwr/browser/engine) do DWR. Observe que ela não possui na sua url o endereço **interface**. Pois o **interface** é só no caso de Objetos JS/Java que você mapeou no dwr.xml.

```
4. <script type="text/javascript" src='<%=request.getContextPath()
%>/dwr/util.js'></script>
```

Importamos agora a biblioteca que você irá mais utilizar, a util.js(http://getahead.ltd.uk/dwr/browser/util). É dela que usamos a função DWRUtil.setValue(); e outras que veremos nos próximos artigos.

```
5. <script type="text/javascript" src='js/exemplo.js'></script>
```

Fazemos um import do nosso JS onde se encontra as nossas funções.

```
6. <a href="#" onclick="javascript:getMsg();">Exibir mensagem</a> |
   <a href="#" onclick="javascript:apagaMsg();">Apaga Mensagem</a>
```

Criamos dois links que ao serem clicados vão executar no ***onClick*** a nossa respectiva função. Lembrando que essa chamada pode ser feita dentro de um <*input*>, <*link*> e onde for possível fazer a chamada com JavaScript.

```
7. <div id="divMensagem"></div>
```

É nessa tag **<*div*>** onde dizemos onde o DWR(Ajax) irá colocar a sua mensagem. Para isso basta colocar no setValue o nome do ID do seu DIV. Podemos também usar a maioria das tag do HTML: span, div, table, tbody e etc.

Vamos ver esse sistema rodando, acesse: http://127.0.0.1:8080/PalestraDWR/ ou http://localhost:8080/PalestraDWR/.

(Imagem 14)

(Imagem 15)

Conclusão

O DWR não é um bicho de sete cabeças, ele é bem simples como vocês poderam observar, ele é fácil de usar, fácil de instalar, fácil de configurar e dar manutenção.

Vimos na primeira parte desse artigo como criar, configurar um exemplo simples do DWR. No próximo artigo vamos ver alguns exemplos com acesso ao banco de dados, com novas funções e exemplos.

Espero que tenham gostado deste artigo e que ele tenha sido de utilidade.

Até o próximo.

Handerson Frota
(handersonbf@gmail.com)