DWR – DIRECTED WEB REMOTING

Trabalhando com Bean's

Uma das facilidades de se lhe dar com o DWR é a forma como ele trabalha com bean's, coisas que você antes se preocuparia e muito, agora é simples de se fazer.

Antes de iniciar-mos o artigo vamos esclarecer para o leitor algumas informações.

No inicio do artigo resolvi adotar o DWR 1.1.4 que é a versão estável atualmente do DWR, mas essa versão não daria suporte para alguns dos recursos que queríamos demonstrar, então resolvi utilizar o DWR 2.0 M2, que é a versão ainda em desenvolvimento do DWR, porém das versões Milestone é a mais estável e que utilizo já em alguns projetos em produção, por isso você não terá problemas, pelo menos até agora eu não tive nenhum e ela já está em 4 projetos grandes e já faz algum tempo.

PS1: A versão que está para download no site do DWR atualmente é a versão DWR2.0 RC2 que esta NÃO é indicada no momento pois ainda apresenta alguns problemas e bugs, claro que por motivos óbvios, ela ainda esta em desenvolvimento.

PS2: Vamos utilizar nomes de funções, métodos, parâmetros e variáveis em inglês, pois esse artigo usará o mesmo projeto para a vídeo aula da Java Magazine e para a mesma esse padrão em inglês tem que ser seguido.

Preparando o ambiente

Aproveitando a estrutura dos artigos anteriores e do projeto de algumas vídeo aulas da JavaMagazine vamos aproveitar o mesmo projeto e vamos fazer nesse artigo alguns exemplos de como utilizar e trabalhar com Bean's com o DWR.

Antes vamos citar algumas diferenças e configurações para se utilizar o DWR2.0 no nosso projeto, mas caso prefira baixe o projeto aqui.

Versão DWR2.0M2 download

Definição do WEB.XML
Você fará uma única alteração no seu web.xml, alterando apenas o Servlet do DWR.

Para versões 1.x.x

```
<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>
```

Listagem 01. Trecho do mapeamento do DWR 1.x.

Para versões 2.x.x

```
<servlet>
      <servlet-name>dwr-invoker</servlet-name>
      <servlet-class>org.directwebremoting.servlet.DwrServlet</servlet-
class>
  <init-param>
     <param-name>debug</param-name>
     <param-value>true</param-value>
  </init-param>
</servlet>
```

Listagem 02. Trecho do mapeamento do DWR 2.x

Definição do DWR.XML
Agora no seu dwr.xml

Para versões 1.x.x

```
<?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">
```

Listagem 03. Inicio da declaração do dwr.xml versão 1.x

Para versões 2.x.x

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE dwr PUBLIC "-//GetAhead Limited//DTD Direct Web Remoting
2.0//EN" "http://www.getahead.ltd.uk/dwr/dwr20.dtd">
```

Listagem 04. Inicio da declaração do dwr.xml versão 2.x

IDE e ferramentas utilizadas
Vamos utilizar o MyEclipse versão 5.1.1 GA, Aptana e Tomcat 5.5.17

Criando o JSP

Na Listagem 05 temos o nosso JSP(indexBean.jsp), será um JSP simples que no decorrer do artigo vamos incrementando novas funcionalidades.

```
<%@ page language="java" import="java.util.*" pageEncoding="ISO-8859-
1"%>
<html>
  <head>
    <title>DWR Working with bean</title>
      <meta http-equiv="pragma" content="no-cache">
      <meta http-equiv="cache-control" content="no-cache">
      <meta http-equiv="expires" content="0">
      <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
      <meta http-equiv="description" content="This is my page">

<!– Import do FaçadeAjax -->
<script type='text/javascript'
src='/DWRProject/dwr/interface/FacadeAjax.js'></script>

<!– Import do DWREngine -->
<script type='text/javascript'
src='/DWRProject/dwr/engine.js'></script>

<!– Import do DWRUtil -->
<script type='text/javascript' src='/DWRProject/dwr/util.js'></script>

<!– Import do arquivo JS que será utilizado para os exemplos -->
<script type='text/javascript' src='js/formBeans.js'></script>

  </head>
<body>
</body>
</html>
```

Listagem 05. JSP

Criaremos um arquivo JS(JavaScript) chamado formBeans.js é nele que iremos trabalhar os exemplos, e na Listagem 05 fizemos o import do mesmo.

Preparando o FacadeAjax e os Bean's JAVA

BeanPeople e BeanPeopleCharacteristics

Para demonstrar a real funcionalidade deste artigo vamos criar Beans Java para demonstrarmos como ele trabalha do JS para o Java e suas facilidades.

BeanPeople

```
private int id = 0;
private String name = "";
private String telephone = "";
private String address = "";
private BeanPeopleCharacteristics peopleCharacteristics = null;
private List<BeanPeopleCharacteristics> listPeopleCharacteristics = new
ArrayList<BeanPeopleCharacteristics>();

public String getXXX()...
public void setXXX(XXX)...
...
```

Listagem 06. BeanPeople com seus gets e sets

BeanPeopleCharacteristics

```
private int idCharacteristics = 0;
private String characteristics = "";

public String getXXX()...
public void setXXX(XXX)...
...
```

Listagem 07. BeanPeopleCharacteristics com seus gets e sets

Métodos para utilizar os Beans no JAVA.
Vamos utilizar dois métodos para trabalharmos com os Beans que estamos criando no JS e veremos como o DWR faz esse parse.

Classe FacadeAjax

```
public BeanPeople setBean(BeanPeople beanPeople){
            System.out.println("Bean");

            System.out.println(beanPeople.getId());
            System.out.println(beanPeople.getName());
            System.out.println(beanPeople.getTelephone());
            System.out.println(beanPeople.getAddress());

            return beanPeople;
      }
...
```

Listagem 08. Método setBean.

Na Listagem 08 vimos um exemplo de como receber esse bean na camada JAVA, como podem ver é um método simples que recebe como parâmetro um BeanPeople e apenas dá um System.out.println() em cada propriedade.

```
public BeanPeople setBeanList(BeanPeople beanPeople){
      System.out.println("Bean");

      System.out.println(beanPeople.getId());
      System.out.println(beanPeople.getName());
      System.out.println(beanPeople.getTelephone());
      System.out.println(beanPeople.getAddress());

List<BeanPeopleCharacteristics> list =
beanPeople.getListPeopleCharacteristics();

      for(int i=0;i<list.size();i++){
            System.out.println(list.get(i).getCharacteristics());
      }

      return beanPeople;
}
...
```
Listagem 09. Método setBeanList.

Na Listagem 09 vimos um exemplo bem semelhante ao da Listagem 08 e claro poderíamos ter utilizado o mesmo, mas para efeito de teste e explicação eu criei outro método que faz a mesma coisa da Listagem 08, só que agora eu executo um loop no List de Beans que é uma propriedade do BeanPeople

Mapeamento dwr.xml

Agora vamos mapear o FacadeAjax, não vamos entrar em detalhes do mapeamento pois o mesmo já foi explicado no artigo DWR – Directed Web Remoting – Parte I.I.

Dwr.xml

```
<dwr>
  <allow>
      <create creator="new" javascript="FacadeAjax" scope="request">
            <param name="class" value="com.dwr.facade.FacadeAjax" />
      </create>

      <convert converter="bean" match="com.dwr.bean.BeanPeople" />
      <convert converter="bean"
match="com.dwr.bean.BeanPeopleCharacteristics" />
  </allow>
</dwr>
```
Listagem 10. DWR.xml mapeando os Beans e a Classe FacadeAjax

Iniciando os Exemplos

Iniciaremos com exemplos simples e vamos aumentando o nível no decorrer do artigo. Vamos exemplificar as principais maneiras de se trabalhar com um bean, e algumas facilidades, lembrando que vamos focar sempre de como trabalhar com Beans no JS para o Java.

A versão 1.1.4 do DWR trabalha perfeitamente e sem problemas com Bean's e list de Beans do Java para o JS, porém tem algumas limitações do JS para o Java, por isso a escolha da versão 2.0 M2.

Criando um Simples Bean no JavaScript

Primeiramente vamos saber como criar um Bean no JavaScript para que ao passar para o java esse mesmo "*popule*" o "real" bean.

Para você criar um bean e o DWR "enxergar" o mesmo no java e fazer a conversão, você precisa seguir um "padrão", veja Listagem 11.

```
var bean = {
      id:1,
      name:"Handerson Frota",
      telephone:"3333333",
      address:"Mr Hull"
}
```

Listagem 11. BeanPeople sendo criado no JS.

Observe que as propriedades do nosso beanJS(vamos chama-lo assim) são AS MESMAS do nosso beanJava, quando você utiliza o DWR para enviar esse beanJS para a sua classe java e essa mesma espera um Bean que possui as mesmas propriedades ele faz um parse dos valores e "popula" o beanJava.

E para dar um "get" nos valores do Bean é mais simples ainda.

```
bean.id;
bean.name;
bean.telephone;
bean.address;
```

Listagem 12. Pegando os valores do beanJS dentro do JavaScript.

Vamos ao um exemplo mais prático.

Criemos a seguinte função javascript no nosso formBean.js. Ver Listagem 13 e Listagem 14.

```
function setBean(){
      var bean = {
            id:1,
            name:"Handerson Frota",
            telephone:"3333333",
            address:"Mr Hull"
      }

      FacadeAjax.setBean(renderBean, bean);
}
```

Listagem 13. Função setBean que cria um beanJS e envia o mesmo para um método da classe java(setBean ver Listagem 08).

```
function renderBean(bean){
      DWRUtil.setValue("renderSetBean", "Id:" + bean.id + "<br>" +
      "Name: " + bean.name + "<br>" +
      "Telephone: " + bean.telephone + "<br>" + "Address: " +
      bean.address + "<br>");
}
```

Listagem 14. Função que “renderiza” o retorno do método setBean(ver Listagem 08) e monta o resultado na tela.

Na Listagem 14 o parâmetro do retorno será um BeanJava “populado” por um BeanJS.

Agora vamos criar a chamada para executar essa função e o div que vai renderizar o resultado. Ver Listagem 15.

```
Method <a href="javascript:setBean()">setBean()</a>
<div id="renderSetBean"></div>
```

Listagem 15. Chamada para a função setBean(ver Listagem 13) do JS

O resultado da execução desta operação esta logo abaixo. Ver Figura 01 e Figura 02.

Figura 01

Figura 02

Imagine a seguinte situação: Você tem um formulário e quer popular o mesmo com um bean que esta vindo de um JS qualquer ou do DWR, não importa, e fazer isso com apenas uma linha como você faria ?

Simples, veja Listagem 16 a 18.

```
Method <a href="javascript:setBeanForm()">setBeanForm()</a>
<form action="" method="get" id="formBean" name="formBean">
<table>
  <tr>
      <td>
      Id: <input type="text" name="id" id="id" value=""/><br>
      Name: <input type="text" name="name" id="name" value=""/><br>
      Telephone: <input type="text" name="telephone" id="telephone"
value=""/><br>
      Address: <input type="text" name="address" id="address"
value=""/><br>
      </td>
            </tr>
      </table>
 </form>
```

Listagem 16. Formulário criado para o nosso exemplo.

Na Listagem 16 observe que criamos um formulário simples e básico, a única coisa que devemos nos atentar é para o ID de cada INPUT, pois é com ele que o DWR vai se identificar para preencher os valores respectivos. Claro que o nome dos ID's tem que ser o mesmo do seu beanJS e beanJava.

Vamos agora criar a função que irá preencher esses dados no formulário.

```
function setBeanForm(){
      var bean = {
                  id:2,
                  name:"Handerson Frota",
                  telephone:"3333333",
                  address:"Mr Hull"
            }

      FacadeAjax.setBean(renderBeanForm, bean);
}
```

Listagem 17. Função setBeanForm que cria um beanJS e envia o mesmo para um método da classe java(setBean ver Listagem 08).

```
function renderBeanForm(bean){
      DWRUtil.setValues(bean);
}
```

Listagem 18. Função que "renderiza" o retorno do método setBean(ver Listagem 08) só que agora utilizamos a função DWRUtil.setValues().

Na Listagem 17 é basicamente é a mesma função que usamos anteriormente, Listagem 13, com a diferença que agora a função de retorno(Listagem 18) ao invés de pegar propriedade por propriedade utiliza o DWRUtil.setValues() para "renderizar" automaticamente os valores, essa função é muito útil e simples de utilizar, vamos entende-la melhor.

DWRUtil.setValues();

Essa função vai receber como parâmetro: propriedades ou beans.
Quando você utilizar essa propriedade como assim foi feito na Listagem 18 a mesma vai percorrer na página procurando pelos ID's os nomes dessas propriedades e colocar os valores respectivos. Ver resultado nas figuras: 03 e 04.

Figura 03

Figura 04

Vamos agra utilizar o DWRUtil.getValues() para pegar os valores de um formulário. Imagine que você tem um formulário de “N” campos e você quer enviar esses valores do seu formulário para a sua camada de negócio, um cadastro, por exemplo, veja como é simples.

Adicionamos um botão no formulário, ver listagem 19.

```
<input type="button" name="submit" value="Method SendBeanForm"
onclick="sendBeanForm()"/>
```

Listagem 19. Botão que executa o sendBeanForm(ver Listagem 20).

Função JS que vai pegar esses valores do formulário e enviar para a classe java, ver Listagem 20.

```
function sendBeanForm(){
      var bean = DWRUtil.getValues("formBean");
      FacadeAjax.setBean(renderBean, bean);
}
```

Listagem 20. Função que utiliza o DWRUtil.getValues(“nomeDoFormulario”) para enviar um BeanPeople já “populado” para a método setBean(ver Listagem 08) e como retorno chama a função renderBean(ver Listagem 14) para renderizar o resultado.

DWRUtil.getValues();

Essa função vai receber como parâmetro: nome do formulário.
Com uma única linha, DWRUtil.getValues(), você consegue pegar todos os campos do formulário que possuem ID. Se os id’s forem iguais as propriedades de um determinado Bean, por exemplo o BeanPeople(ver Listagem 06) então automaticamente quando você passar esses valores para o DWR ele irá fazer o parse e você terá um Bean populado com apenas uma única linha.Ver Listagem 21.

```
var bean = DWRUtil.getValues("formBean");
```

Listagem 21. Pegando valores de um formulário e criando um beanJS para ser enviado para o java, "populando” um beanJava correspondente.

Veja nas figuras **05** e **06** o resultado deste exemplo.

Figura 05

Figura 06

Exemplos mais Avançados

Bem ate agora vimos como criar, enviar, pegar e setar(DWRUtil.getValues(), DWRUtil.setValues()) valores em um bean. Mais e um bean que possui uma propriedade que é outro bean ?

Na Listagem 07 vimos o BeanPeopleCharacteristics que é um Bean que identifica as características de uma pessoa(BeanPeople ver Listagem 06).

Na Listagem 06 temos o BeanPeople(ver Listagem 06) que possui uma propriedade peopleCharacteristics que é do tipo BeanPeopleCharacteristics.(Ver Listagem 06 e Listagem 22)

```
private BeanPeopleCharacteristics peopleCharacteristics = null;
```

Listagem 22. Trecho do BeanPeople(Ver Listagem 06) onde se define uma propriedade do tipo BeanPeopleCharacteristics(Ver Listagem 07) .

Para executarmos nosso exemplo vamos adicionar um trecho logo abaixo da nossa tabela do nosso formulário. Ver Listagem 23.

```
<br>
<br>
Method <a
href="javascript:setBeanCharacteristics()">setBeanCharacteristics()</a>
<div id="renderSetBeanCharacteristics"></div>
```

Listagem 23. Trecho no JPS para a chamada da função setBeanCharacteristics(Ver Listagem 24)definindo um div onde será renderizado o resultado.

```
function setBeanCharacteristics(){
      var bean = {
            id:1,
            name:"Handerson Frota",
            telephone:"3333333",
            address:"Mr Hull",

            peopleCharacteristics:{idCharacteristics:2,characteristics:
      "Black"}
      }
      FacadeAjax.setBean(renderSetBeanCharacteristics, bean);
}
```

Listagem 24. Função que cria um beanJS de BeanPeople(ver Listagem 06) e adiciona um segundo beanJs agora de características(ver Listagem 07) na propriedade peopleCharacteristics.

```
function renderSetBeanCharacteristics(bean){
      DWRUtil.setValue("renderSetBeanCharacteristics", "Id:" + bean.id
      + "<br>" + "Name: " + bean.name + "<br>" +
      "Telephone: " + bean.telephone + "<br>" + "Address: " +
      bean.address + "<br>" +
      "<strong>BeanCharacteristicsId: " +
      bean.peopleCharacteristics.idCharacteristics  + "<br>" +
      "BeanCharacteristics: " +
      bean.peopleCharacteristics.characteristics + "</strong>");
}
```

Listagem 25. Função que renderiza o resultado do método setBean(ver Listagem 08).

Vimos que na Listagem 24 dizemos que a propriedade **peopleCharacteristics** recebe um bean, claro que com as mesmas propriedades do BeanPeopleCharacteristics(Ver Listagem 07). Poderíamos fazer também da seguinte maneira obtendo o mesmo resultado (ver Listagem 26).

```
function setBeanCharacteristics(){
      var beanCharacteristics = {
            idCharacteristics:2,
            characteristics:"Black"
       }

      var bean = {
            id:1,
            name:"Handerson Frota",
            telephone:"3333333",
            address:"Mr Hull",
            peopleCharacteristics:beanCharacteristics
      }
      FacadeAjax.setBean(renderSetBeanCharacteristics, bean);
}
```

Listagem 26. Uma segunda forma de fazer o que a Listagem 24 fez.

Veja agora o resultado deste exemplo nas figuras 07 e 08.

Figura 07

Figura 08

Vamos ver a seguir o mesmo exemplo, porém pegando os valores de um formulário, que é o que mais interessante mesmo.

Adicionamos no JSP o seguinte trecho abaixo em negrito.

```
CharacteristId: <input type="text" name="idCharacteristics"
id="idCharacteristics" value=""/>
<br>
Characterist: <input type="text" name="characteristics"
id="characteristics" value=""/>
<br>
<input type="button" name="submit" value="Method SendBeanForm"
onclick="sendBeanForm()"/>
<br>
<input type="button" name="submit" value="Method
sendBeanFormCharacteristics"
onclick="sendBeanFormCharacteristics()"/><br>
```

Listagem 27. Trecho em negrito adicionado no JSP que contém o formulário de exemplo. Adicionando mais dois campos input e um botão.

```
function sendBeanFormCharacteristics(){
	var bean = DWRUtil.getValues("formBean");

	var beanCharacteristics = {
			idCharacteristics:bean.idCharacteristics,
			characteristics:bean.characteristics
	};

	bean.peopleCharacteristics = beanCharacteristics;

	FacadeAjax.setBean(renderSetBeanCharacteristics, bean);
}
```

Listagem 28. Função sendBeanFormCharacteristics que “popula” um bean a partir dos dados de um formulário e renderiza na tela com o método renderSetBeanCharacteristics(Ver Listagem 25)

Na Listagem 28 estamos novamente utilizando o DWRUtil.getValues() para recuperar os dados do formulário e coloca-los em um beanJS, mas porque então o trecho da Listagem 29 ?

```
	var beanCharacteristics = {
			idCharacteristics:bean.idCharacteristics,
			characteristics:bean.characteristics
	};

	bean.peopleCharacteristics = beanCharacteristics;
```

Listagem 29. Trecho que recupera os dados do formulário e individualmente os dados referentes ao BeanPeopleCharacteristics(ver Listagem 07)

Nas versões 1.x o DWR não consegue como já foi dito trabalhar adequadamente com um list de beans e beans aninhados, já na versão 2.0M2 ele trabalha sem problemas, mas ainda não esta perfeito, pois ainda possui determinadas funcionalidade que ele

ainda não implementa, no nosso caso popular automaticamente um bean aninhado através de um formulário, assim como ele faz com um simples bean.

Mas quando o bean possui uma propriedade que é outro bean, ele ainda não converte automaticamente com o getValues() para o bean correspondente, fazendo com que você seja obrigado a criar "*na mão*" este bean(ver Listagem 28 e Listagem 29) e logo após adicionar esse beanJS na propriedade correspondente e assim o DWR consegue fazer o parse corretamente.

Porem já na versão RC2 essa funcionalidade já esta sendo implementada, mas como a versão RCX ainda possui vários bug's, eu não vou comentar e ou citar exemplos, pois a mesma ainda esta em fase beta teste e a maneira que esta atualmente poderá ser alterada.

O resultado do exemplo anterior pode ser visto nas figuras 09 e 10.

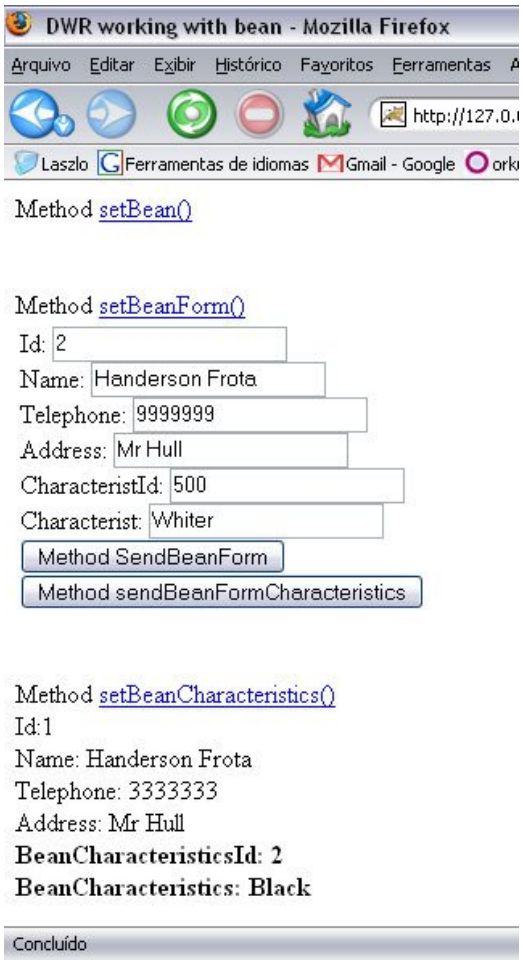

Figura 09

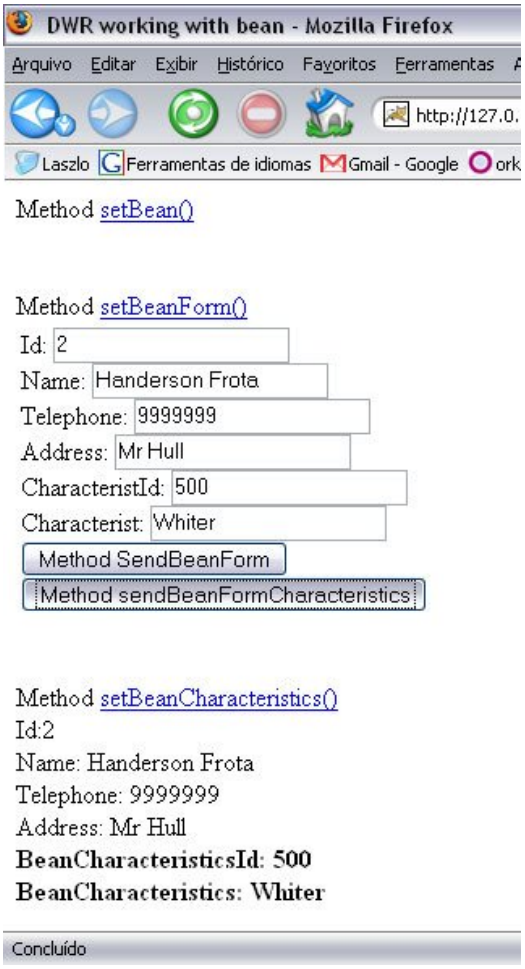

Figura 10

Para finalizar nosso artigo vamos agora demonstrar como fazer o mesmo processo dos exemplos anteriores, só que agora utilizaremos um List de Beans.

Na Listagem 06 temos o BeanPeople(ver Listagem 06) que possui uma propriedade listPeopleCharacteristics que é do tipo List<BeanPeopleCharacteristics>, ou seja, um list de beanPeopleCharacteristics.(Ver Listagem 06 e 30)

```
private List<BeanPeopleCharacteristics> listPeopleCharacteristics = new ArrayList<BeanPeopleCharacteristics>();
```

Listagem 30. Trecho do BeanPeople(Ver Listagem 06) onde se define uma propriedade do tipo List<BeanPeopleCharacteristics>(Ver Listagem 06 e 07) .

No nosso JSP vamos agora fazer algumas mudanças.

Vamos adicionar o trecho de código da Listagem 22 na Listagem 23

```
Method <a href="javascript:setBeanCharacteristics()">setBeanCharacteristics()</a>
    <div id="renderSetBeanCharacteristics"></div>
    <ol id="olCharacteristics"></ol>
```

Listagem 31. Adicionando OL para lista numeradas.

Vamos utilizar na Listagem 31 uma lista numerada, e utilizar outras funcionalidades do DWR, que é o DWRUtil.addOptions() e DWRUtil.removeAllOptions(), esses últimos não vamos entrar em detalhes pois um artigo já foi elaborado para este assunto e esta no link a seguir: DWRUtil.addOptions().

Logo após a Listagem 31 vamos adicionar o trecho de código da Listagem 32.

```
<br>
<br>
<input type="checkbox" id="type" name="types" value="Black"/>Black
<input type="checkbox" id="type" name="types" value="White"/>White
<input type="checkbox" id="type" name="types" value="Indian"/>Indian
<input type="checkbox" id="type" name="types" value="Mexican"/>Mexican
<br>
<input type="button" name="submit" value="Method sendBeanList" onclick="sendBeanList()"/><br>
```

Listagem 32. Trecho referente ao conjunto de Características que o usuário poderá selecionar, sendo um ou várias características.

```
DWRUtil.removeAllOptions("olCharacteristics");
DWRUtil.addOptions("olCharacteristics", bean.listPeopleCharacteristics, "characteristics");
```

Listagem 33. Trecho adicionado ao código da Listagem 25 para ser possível montar um list de beans.

```
function sendBeanList(){
      1.var bean = DWRUtil.getValues("formBean");

      2.var beanCharacteristics = {
                  idCharacteristics:bean.idCharacteristics,
                  characteristics:bean.characteristics
      };

      3.bean.peopleCharacteristics = beanCharacteristics;

      4.var array = document.getElementsByName("types");
      5.var listBean = [];
      var x = 0;

      6.for(var i=0;i<array.length;i++){

            7.if(array[i].checked){
                  8.var beanCheck = {
                        9.characteristics:array[i].value
                  };
                  10.listBean[x] = beanCheck;
                  x++;
            }

      }

      11.bean.listPeopleCharacteristics = listBean;

      12.FacadeAjax.setBeanList(bean,renderSetBeanCharacteristics);
}
```

Listagem 34. Função que recupera os valores de um formulário e popula um bean especifico, enviando o retorno para a função renderSetBeanCharacteristics (ver Listagem 25 e Listagem 33).

1. Recuperando todos os dados do formulário e adicionando no beanJS.
2. Criando “na mão” um beanJS para BeanPeopleCharacteristics, já que nessa versão o getValues não consegue já preencher automático o bean com uma propriedade que é um bean ou um list de beans.
3. Após a criação deste bean na linha 2, e com os valores já preenchidos, esse bean é adicionado na propriedade que é um BeanPeopleCharacteristics.
4. Recupera-se os elementos com o nome: “types”, que na verdade são checkbox e adicionados em um array.
5. É criando um Array que será adicionado a propriedade que é um list de beans
6. Inicia-se um For para percorrer o array de checkbox
7. É verificado se o checkbox está “ticado”(selecionado)
8. É inicializado um bean do tipo BeanPeopleCharacteristics
9. O valor que foi selecionado no checkbox é adicionado na propriedade do beanJS criando na linha **8**.
10. Esse beanJS gerado nas linhas de 8 a 9 é adicionado dentro do listBean, gerado na linha **5**.
11. Por fim, esse listBean que possui vários beanJS de BeanPeopleCharacteristics é adicionado a propriedade `listPeopleCharacteristics` do BeanPeople.

O resultado deste último exemplo você confere nas figuras **11** e **12.**

Figura 11

Figura 12

Conclusão

A maneira de trabalhar com um bean no JAVA e no JavaScript(DWR ou javascript puro) são muito parecidas, e intuitivas, TODOS os exemplos que utilizam o FacadeAjax.metodo(), passam o beanJS criado para o java, com os system.out.println() você poderá ver no console os valores do bean no lado Java e poder comprovar que o bean que você criou no JS popula o bean que você tem no Java.

A simplicidade, facilidade, usabilidade e robustez que você adquire trabalhando com bean no DWR é notável. Temos várias possibilidades de uso, fica a cargo da sua necessidade.

Você viu também como é simples pegar *N* campos de um formulário e como é simples popular esses mesmos *N* campos, utilizando o getValues e setValues, lembrando que para utilizar o setValues e getValues você não é obrigado a utilizar um beanJava, você tem que entender que ele é um objeto JavaScript que contem propriedades e que quando esse objeto vai para o Java através do DWR o mesmo faz os parses necessários para “converte-lo” em um BeanJava.

Espero que eu tenha conseguido passar uma visão geral e mais prática de algumas maneiras de se usar um bean com o DWR e que você tenha visto e entendido como é simples e fácil de trabalhar.

A versão FINAL do DWR 2.0 promete muita coisa boa, muito mais integração, simplicidade e facilidade de uso, vale a pena ficar no aguardo e enquanto a versão 2.0 não sai, ficamos com a versão 2.0M2 que funciona sem problemas.

Para baixar o projeto deste artigo e sql do banco clique no link abaixo.

OBS: As informações sobre a tabela do banco esta na pasta BD dentro do projeto, caso queria utilizar dados do banco.

Baixar projeto aqui.

Obrigado e Boa Sorte.

Até o próximo artigo.

Handerson Frota
(handersonbf@gmail.com).