# **[DWR入门教程](http://www.cnblogs.com/cyjch/archive/2012/02/16/2353758.html)**

|  |
| --- |
| DWR(Direct Web Remoting)是一个WEB远程调用框架.利用这个框架可以让AJAX开发变得很简单.利用DWR可以在客户端利用JavaScript直接调用服务端的Java方法并返回值给JavaScript就好像直接本地客户端调用一样(DWR根据Java类来动态生成JavaScrip代码).它的最新版本DWR0.6添加许多特性如:支持Dom Trees的自动配置,支持Spring(JavaScript远程调用spring bean),更好浏览器支持,还支持一个可选的commons-logging日记操作.    以上摘自open-open，它通过反射，将java翻译成javascript，然后利用回调机制，轻松实现了javascript调用Java代码。    其大概开发过程如下：  1.编写业务代码，该代码是和dwr无关的。  2.确认业务代码中哪些类、哪些方法是要由javascript直接访问的。  3.编写dwr组件，对步骤2的方法进行封装。  4.配置dwr组件到dwr.xml文件中，如果有必要，配置convert，进行java和javascript类型互转。  5.通过反射机制，dwr将步骤4的类转换成javascript代码，提供给前台页面调用。  5.编写网页，调用步骤5的javascript中的相关方法（间接调用服务器端的相关类的方法），执行业务逻辑，将执行结果利用回调函数返回。  6.在回调函数中，得到执行结果后，可以继续编写业务逻辑的相关javascript代码。    下面以用户注册的例子，来说明其使用。（注意，本次例子只是用于演示，说明DWR的使用，类设计并不是最优的）。    1.先介绍下相关的Java类      User: 用户类，    public class User {  //登陆ID，主键唯一  private String id;  //姓名  private String name;  //口令  private String password;  //电子邮件  private String email;    //以下包含getXXX和setXXX方法  .......    }      UserDAO：实现User的数据库访问，这里作为一个演示，编写测试代码    public class UserDAO {      //存放保存的数据      private static Map dataMap = new HashMap();        //持久用户      public boolean save(User user) {        if (dataMap.containsKey(user.getId()))          return false;        System.out.println("下面开始保存用户");        System.out.println("id："+user.getId());        System.out.println("password："+user.getPassword());        System.out.println("name："+user.getName());        System.out.println("email："+user.getEmail());        dataMap.put(user.getId(), user);        System.out.println("用户保存结束");        return true;      }        //查找用户      public User find(String id) {        return (User)dataMap.get(id);      }  }      DWRUserAccess：DWR组件，提供给javascript访问的。      public class DWRUserAccess {          UserDAO userDAO = new UserDAO();          public boolean save(User user) {          return userDAO.save(user);        }          public User find(String id) {          return userDAO.find(id);        }    }        下面说明下程序执行的流程      1.用户在页面上输入相关注册信息，id、name、password、email，点击“提交”按钮    2.javascript代码开始执行，根据用户填写相关信息，通过dwr提供的DWRUserAccess.js里save的方法，调用服务器端的DWRUserAccess类save方法，将注册信息保存。    3.通过DWRUserAccess.jsp里的find方法，调用服务器端DWRUserAccess类里的find方法，执行用户信息查找。      注意，在以上的执行过程中，DWRUserAccess是供DWR调用的，是DWR组件，因此需要将DWRUserAccess类配置到dwr中。      接下来讲解本次dwr测试环境的配置。      1.新建一个webapp，命名为testApp    2.将dwr.jar拷贝到testApp的WEB-INF的lib目录下    3.编译上面的User，UserDAO，DWRUserAccess类，放到classes目录下    4.在web.xml中配置servlet,适配路径到dwr目录下,如下所示      <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>      <init-param>        <param-name>scriptCompressed</param-name>        <param-value>false</param-value>      </init-param>      <load-on-startup>1</load-on-startup>    </servlet>      <servlet-mapping>      <servlet-name>dwr-invoker</servlet-name>      <url-pattern>/dwr/\*</url-pattern>    </servlet-mapping>      以上的配置可以拦截testApp下所有指向dwr的请求，关于这个拦截器，我们会在后面介绍。      5.WEB-INF下新建一个dwr.xml文件，内容如下：    < 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="DWRUserAccess">        <param name="class" value="test.DWRUserAccess"/>      </create>  <convert converter="bean" match="test.User"/>    </allow>  </dwr>      这里我们把DWRUserAccess配置到了dwr中，create元素中，creater="new"表示每调用一次DWRUserAccess时，需要new一个这样的类；javascript="DWRUserAccess"，表示提供给前台页面调用的javascirpt文件是DWRUserAccess.js。      convert元素用于数据类型转换，即java类和javascript之间相互转换，因为和前台交换的是User对象，因此需要对此使用bean转换，我们将在后面介绍这个类。      4.编写测试的HTML页面 test.html     <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">  <HTML>  <HEAD>  <TITLE>DWR测试</TITLE>  <meta http-equiv=Content-Type content="text/html; charset=gb2312">  <script src="/oblog312/dwr/engine.js"></script>  <script src="/oblog312/dwr/util.js"></script>  <script src="/oblog312/dwr/interface/DWRUserAccess.js"></script>  </HEAD>  <BODY>  <B>用户注册</B><br>  ------------------------------------------------  <Br>  <form name="regForm">  登陆ID：<input type="text" name="id"><br>  口  令：<input type="password" name="password"><br>  姓  名：<input type="text" name="name"><br>  电子邮件：<input type="text" name="email"><br>  <input type="button" name="submitBtn" value="提交" onclick="OnSave()"><br>      </form>    <br>  <br><B>用户查询</B><br>  ------------------------------------------------  <Br>  <form name="queryForm">  登陆ID：<input type="text" name="id"><br>  <input type="button" name="submitBtn" value="提交" onclick="OnFind()"><br>  </form>  <br>  </BODY>  </HTML>  <SCRIPT LANGUAGE="JavaScript">  <!--  function saveFun(data) {  if (data) {    alert("注册成功！");  } else {    alert("登陆ID已经存在！");  }  }    function OnSave() {  var userMap = {};  userMap.id = regForm.id.value;  userMap.password = regForm.password.value;  userMap.name = regForm.name.value;  userMap.email = regForm.email.value;  DWRUserAccess.save(userMap, saveFun);  }    function findFun(data) {  if (data == null) {    alert("无法找到用户："+queryForm.id.value);    return;  }    alert("找到用户，nid："+data.id+"，npassword："+data.password+"，nname："+data.name+"，nemail："+data.email);    }    function OnFind() {  DWRUserAccess.find(queryForm.id.value, findFun);  }  //-->  </SCRIPT>      以下对页面的javascript进行解释    <script src="/oblog312/dwr/engine.js"></script>  <script src="/oblog312/dwr/util.js"></script>  这两个是dwr提供的，用户可以不必关心，只需要导入即可    <script src="/oblog312/dwr/interface/DWRUserAccess.js"></script>  是我们编写的DWRUserAccess类，经dwr反射后，生成的javascript代码，它和DWRUserAccess.java是对应的，供用户调用，实际上我们就是通过这个js文件去调用服务器端的DWRUserAccess类的。    <SCRIPT LANGUAGE="JavaScript">  <!--  function saveFun(data) {  if (data) {    alert("注册成功！");  } else {    alert("用户名已经存在！");  }  }    function OnSave() {  var userMap = {};  userMap.id = regForm.id.value;  userMap.password = regForm.password.value;  userMap.name = regForm.name.value;  userMap.email = regForm.email.value;  DWRUserAccess.save(userMap, saveFun);  }    function findFun(data) {  if (data == null) {    alert("无法找到用户："+queryForm.id.value);    return;  }    alert("找到用户，nid："+data.id+"，npassword："+data.password+"，nname："+data.name+"，nemail："+data.email);    }    function OnFind() {  DWRUserAccess.find(queryForm.id.value, findFun);  }  //-->  </SCRIPT>    这段javascirpt代码，我们来看下OnSave函数，首先它构造一个map，将表单数据都设置到map中，然后调用DWRUserAccess.save(userMap, saveFun)，执行save操作。大家可以注意到，服务器端的DWRUserAccess中的save方法是这样的：boolean save(User user)，其参数是一个User对象，返回一个boolean值；而客户端的方法是这样的：save(userMap,saveFun)，第一个参数userMap是javascirpt中的map对象，在这里相当于服务器端的User对象（在服务器端执行时，会通过convert转换成User对象），前面我们提到dwr是利用回调函数来返回执行结果的，第二个参数saveFun即是一个回调函数。在函数function saveFun(data)中，data是执行结果，这里是一个bool值，非常简单的，我们通过判断data是否为真，可以知道用户名是否重复，用户是否注册成功。    看一下OnFind查找函数，执行结果在回调函数findFun(data)中，因为服务器端返回的是一个User对象，通过convert，将会转换成javascript的一个map对象，  于是在findFun中，通过data.id、data.name、data.password、data.email我们可以轻松的访问到这个User对象。      好了配置完毕，启动服务器，在目录中打入localhost/testApp/test.html。    1.在“用户注册”表单中，id框中输入admin，password中输入123456，name中输入chenbug，email中输入chenbug@zj.com，点击提交按钮，弹出对话框：“注册成功”，在服务器后台可以看到信息如下：    下面开始保存用户  id：admin  password：123456  name：chenbug  email：chenbug@zj.com  用户保存结束    再次点击提交按钮，弹出对话框“登陆ID已经存在”。    2.在“用户查询”对话框中，输入登陆ID为admin，点击提交按钮，提示找到用户，并显示相关信息，输入admin123，点击提交按钮，提示无法找到用户。    至此，测试结束。      后续：  1。拦截器 uk.ltd.getahead.dwr.DWRServlet  该类拦截所有指向dwr目录下的请求，并调用Processor的handler方法进行处理，在uk.ltd.getahead.dwr.impl.DefaultProcessor下，我们可以看到详细的处理过程。  if (pathInfo.length() == 0 ||              pathInfo.equals(HtmlConstants.PATH\_ROOT) ||              pathInfo.equals(req.getContextPath()))          {              resp.sendRedirect(req.getContextPath() + servletPath + HtmlConstants.FILE\_INDEX);          }          else if (pathInfo.startsWith(HtmlConstants.FILE\_INDEX))          {              index.handle(req, resp);          }          else if (pathInfo.startsWith(HtmlConstants.PATH\_TEST))          {              test.handle(req, resp);          }          else if (pathInfo.startsWith(HtmlConstants.PATH\_INTERFACE))          {              iface.handle(req, resp);          }          else if (pathInfo.startsWith(HtmlConstants.PATH\_EXEC))          {              exec.handle(req, resp);          }          else if (pathInfo.equalsIgnoreCase(HtmlConstants.FILE\_ENGINE))          {              file.doFile(req, resp, HtmlConstants.FILE\_ENGINE, HtmlConstants.MIME\_JS);          }          else if (pathInfo.equalsIgnoreCase(HtmlConstants.FILE\_UTIL))          {              file.doFile(req, resp, HtmlConstants.FILE\_UTIL, HtmlConstants.MIME\_JS);          }          else if (pathInfo.equalsIgnoreCase(HtmlConstants.FILE\_DEPRECATED))          {              file.doFile(req, resp, HtmlConstants.FILE\_DEPRECATED, HtmlConstants.MIME\_JS);          }          else          {              log.warn("Page not found (" + pathInfo + "). In debug/test mode try viewing /[WEB-APP]/dwr/"); //$NON-NLS-1$ //$NON-NLS-2$              resp.sendError(HttpServletResponse.SC\_NOT\_FOUND);          }    通过判断request请求的servlet路径，进行处理，大家可以自己去参看，这里不详细讨论。      2.bean转换器，<convert converter="bean" match="test.User"/>  将dwr.jar解压缩，在路径ukltdgetaheaddwr下可以看到dwr.xml，这里配置了系统默认的一些转换器，  <converter id="bean" class="uk.ltd.getahead.dwr.convert.BeanConverter"/>即是刚才用到User类的转换器，进入代码我们来看看它是如何在javascript和java间进行转换的。    打开BeanConverter代码，定位到函数    public Object convertInbound(Class paramType, InboundVariable iv, InboundContext inctx) throws ConversionException    即是将javascript对象转换成java对象的，其中  paramType即Class类型，在上面的例子中是test.User，  InboundVariable iv，是传入的值，通过iv.getValue可以得到传入的javascript值串  InboundContext inctx，是入口参数上下文，用于保存转换的后java对象。    因为前台传入的是一个javascript的map类型，而map肯定是以{开始和以}结束的，于是在这个函数一开始进行了判断  if (!value.startsWith(ConversionConstants.INBOUND\_MAP\_START))          {              throw new IllegalArgumentException(Messages.getString("BeanConverter.MissingOpener", ConversionConstants.INBOUND\_MAP\_START)); //$NON-NLS-1$          }            if (!value.endsWith(ConversionConstants.INBOUND\_MAP\_END))          {              throw new IllegalArgumentException(Messages.getString("BeanConverter.MissingCloser", ConversionConstants.INBOUND\_MAP\_START)); //$NON-NLS-1$          }    javascript中，map里各个项是用逗号连接的，如var userMap = {id:'admin',password:'123456',name:'chenbug',email:'chenbug@zj.com'};而每个项的键值对是用冒号连接的，  在convertInbound函数的接下来的处理中，即是通过分析map字串，通过paramType构造java实例（即User类），然后通过反射，将这些键值对设置到java实例中，并返回。  这样就完成了javascript到java的转换。      另一个函数  public String convertOutbound(Object data, String varname, OutboundContext outctx) throws ConversionException    即是将java对象转换为javascript对象（其实是声明和赋值语句）。  Object data ，是待转换的java对象  String varname，是javascript中的该对象的变量名  OutboundContext outctx，传出参数上下文，用于保存转换后的javascript值    StringBuffer buffer = new StringBuffer();          buffer.append("var "); //$NON-NLS-1$          buffer.append(varname);          buffer.append("={};"); //$NON-NLS-1$  这里声明了map类型的变量。    即下来来的代码即是通过反射进行变量赋值，如下    buffer.append(varname);                      buffer.append('.');                      buffer.append(name);                      buffer.append('=');                      buffer.append(nested.getAssignCode());                      buffer.append(';');  大家可以自己去参看更多的代码。    3.dwr本身提供了一个测试环境，大家在配置完后，可以在IE中输入地址http://localhost/testApp/dwr/index.html，看到配置的各DWR组件，并进行相关测试。 |