DWR框架介绍及使用方法

—— giser062 2014-12-04

# DWR介绍

DWR是一个RPC类库，可以很容易的用Javascript调用Java方法，也可以用Java调用Javascript方法（也称为Reverse Ajax）。

主页：http://directwebremoting.org/dwr/index.html

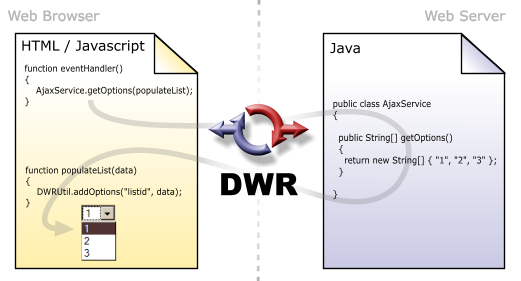
下载链接：

http://java.net/projects/dwr/downloads/download/DWR\_3.x/3.0\_RC3/dwr.jar

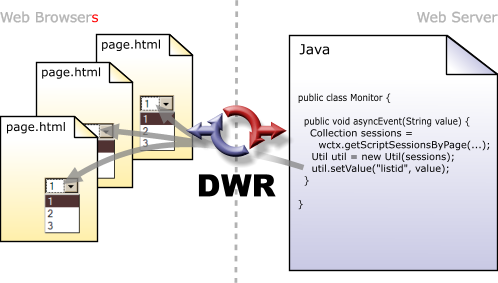
官方文档：

<http://directwebremoting.org/dwr/documentation/index.html>

## Javascript调用Java



## Java调用Javascript（Reverse Ajax）



# 快速使用指南

## 下载安装dwr JAR 文件

下载dwr.jar文件，并且复制到Web工程的WEB-INF/lib目录里

## 下载安装Commons Logging JAR文件

dwr.jar依赖commons-logging.jar，下载该commons-logging.jar，并且复制到Web工程的WEB-INF/lib目录里

## 配置DWR servlet

将下面的Servlet配置内容添加到Web应用程序的WEB-INF/web.xml里。其中<servlet>部分放在web.xml中最后一个<servlet>的后面，<servlet-mapping>同样放在web.xml中最后一个<servlet-mapping>的后面。

 XML Code

|  |  |  |
| --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 |  | <servlet>   <display-name>DWR Servlet</display-name>   <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>  <servlet-mapping>   <servlet-name>dwr-invoker</servlet-name>   <url-pattern>/dwr/\*</url-pattern> </servlet-mapping> |

## 创建DWR的配置文件(dwr.xml)

在WEB-INF目录里创建一个新的配置文件dwr.xml，详细的dwr.xml文档地址：

<http://directwebremoting.org/dwr/documentation/server/configuration/dwrxml/index.html>

简单示例：

 XML Code

|  |  |  |
| --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 |  | <!DOCTYPE dwr PUBLIC "-//GetAhead Limited//DTD Direct Web Remoting 3.0//EN" "http://getahead.org/dwr/dwr30.dtd"> <dwr>   <allow>     <create creator="new" javascript="JDate">       <param name="class" value="java.util.Date" />     </create>     <create creator="new" javascript="Demo">       <param name="class" value="your.java.Bean" />     </create>   </allow> </dwr> |

这个DWR配置文件定义了DWR能创建哪些类，并且能被JavaScript远程调用。上面的例子中我们定义了2个能被JavaScript远程调用的类，并且给出了在JavaScript中的类名。

其中上面用到的new creator，要求所有用到的Java Beans都必须有一个**无参的公共构造方法**。

DWR还有个限制：

**避免使用JavaScript的保留字**，因为用保留字命名的方法会自动被执行。大多数的JavaScript保留字和Java保留字相同，所以不能定义一个“try()”方法，然而有一个最常用的方法是“delete()”，这个方法在JavaScript中有特殊含义，但在Java中没有。

## 获取DWR的测试驱动

启动Web服务器，访问下面的URL：

http://localhost:8080/[YOUR-WEBAPP-CONTEXT]/dwr/

你应该看到一个页面显示在上一步中配置的类。点击一个链接，应该能看到所有能被调用的方法列表，可以输入需要的参数（如果需要参数的话），然后点击“执行”按钮。如果设置了DWR的debug init-param为true的话，这个页面会触发后台Java的Debugging工具。

## 如何在Web应用程序中使用

1、打开[http://localhost:8080/[YOUR-WEBAPP-CONTEXT]/dwr/](http://localhost:8080/%5bYOUR-WEBAPP-CONTEXT%5d/dwr/)，点击你的类

2、把页面上的javascript标签内容添加到需要的页面上

 XML Code

|  |  |  |
| --- | --- | --- |
| 1 2 3 |  | <script type='text/javascript' src='/[YOUR-WEBAPP-CONTEXT]/dwr/engine.js'></script> <script type='text/javascript' src='/[YOUR-WEBAPP-CONTEXT]/dwr/util.js'></script> <script type='text/javascript' src='/[YOUR-WEBAPP-CONTEXT]/dwr/interface/[YOUR-SCRIPT].js'></script> |

# JavaScript调用Java方法

## 使用回调函数处理异步请求

有两种建议的方式

### Call Options Object

建议的语法是使用Call Options Object方式，明确指定callback函数和其它选项。

假设我们有一个Java方法，如下：

public class Remote {

public String getData(int index) { return “data is ” + index }

}

可以使用下面的javascript调用方式：

<script type="text/javascript"

src="*[WEBAPP]*/dwr/interface/Remote.js"> </script>

<script type="text/javascript"

src="*[WEBAPP]*/dwr/engine.js"> </script>

...

Remote.getData(42, {

callback:function(str) {

alert(str);

}

});

‘42’是传递给Java方法getData()的参数。str是getData()方法的返回值。

**指定timeout和错误处理**

Remote.getData(42, {

callback:function(str) { alert(str); },

timeout:5000,

errorHandler:function(message) { alert("Oops: " + message); }

});

### Simple Callback Functions

另一个方法是简单地把一个回调函数参数列表：

function handleGetData(str) {

alert(str);

}

Remote.getData(42, handleGetData);

或者你可以都写在一行：

Remote.getData(42, function(str) { alert(str); });

## 创建Javascript Object和Java Object映射

假设有下面的Java方法，有一个Person类型的Java Bean参数：

public class Remote {

public void doSomethingWithPerson(Person p) {

// Some Java code that does something with Person p.

}

}

Person类如下：

public Person {

private String name;

private int age;

private Date[] appointments;

// getters and setters ...

}

你可以像下面一样调用Javascript：

var myJSPerson = {

name:"Fred Bloggs",

age:42,

appointments:[ new Date(), new Date("1 Jan 2008") ]

};

Remote.doSomethingWithPerson(myJSPerson);

# Reverse Ajax技术

Ajax技术是JavaScript直接调用服务器端Java代码，Reverse Ajax技术即可以实现服务器端Java代码直接调用浏览器中的JavaScript代码，和传统Ajax技术正好相反。Reverse Ajax技术是从DWR2.0版本开始。

Web的访问机制天生是设计用来 pull数据的，也就是只允许 Browser端主动发起请求，server是被动的响应，不允许Server向 Browser发出一个 connection请求 ，所以需要使用一些变通的方式完成类似的功能。DWR支持3种方法来向Browser推送数据：Piggyback，Polling，Comet。

## 服务器推送实现方式

### Polling（轮询）

Polling（轮询）是最明显的解决方案，就是让浏览器以一个固定的间隔或频率向Server发送请求，比如3秒钟，来看看这个页面是否有更新。一般是在HTML文件中加入 <META HTTP-RQUIV="Refresh" CONTENT=10>，实际上就是 HTTP 头标告知浏览器每 10 秒更新一次文档。

**优点** ：不需要服务器端的配置。

**缺点** ：

a) 糟糕的用户体验

b) 对服务器的压力很大，并且造成带宽的极大浪费。

**应用**： 这是一项非常常见的技术，例如，大多数 webmail应用程序就是通过这种技术在电子邮件到达时显示电子邮件的。

### Comet（长连接）

Comet方式通俗的说就是一种长连接机制(long lived http)。同样是由Browser端主动发起请求，但是Server端以一种似乎非常慢的响应方式给出回答。这样在这个期间内，服务器端可以使用同一个connection把要更新的数据主动发送给Browser。因此请求可能等待较长的时间，期间没有任何数据返回，但是一旦有了新的数据，它将立即被发送到客户机。Comet又有很多种实现方式，但是总的来说对Server端的负载都会有增加.虽然对于单位操作来说，每次只需要建议一次connection,但是由于connection是保持较长时间的,对于 server端的资源的占用要有所增加。

**优点**：实时性好（消息延时小）；性能好（能支持大量用户）

**缺点**：长期占用连接，丧失了无状态高并发的特点。

**应用**：股票系统、实时通讯。

### Piggyback

使用piggyback选项，当Server有数据要发送时，并不立即发送数据，而是等待Browser下一次建立连接时，才将两次连接时间段内的更新数据发送出去。

**优点**：不使用长连接，只发送变化的数据

**缺点**：实时性略差，服务端潜在风险大

**应用**：

### 比较

三种方式都有各自的优点，Polling方式是最简单的，但是容易使server过载。相反的Comet需要很大的开发量，但是server端更简单点。Comet的加分点也在具有非常低的延迟，因为是长连接，所以无需等待下一次的浏览器连接。Polling和Comet都需要额外的网络连接，所以真正的低开销的是piggyback方式，但可能会有非常高的延迟。

以上三种方式DWR都支持，仅仅是通过修改配置文件就可以切换实现方式。

## 积极的和消极的Reverse Ajax

如果需要更快的响应速度，并且可以接受额外的服务器负载，可以配置DWR使用Coment或Polling方式。这种模式称作积极的Reverse Ajax。**默认情况下DWR不是以积极Reverse Ajax模式启动**，仅仅允许piggyback传输模式。

## 启用积极Reverse Ajax模式

使用消极的Reverse Ajax模式（piggyback方式），不需要任何配置，如果要启用积极的Reverse Ajax模式，配置步骤如下

### 修改服务器配置

修改服务器WEB-INF/web.xml配置文件，添加一个初始参数，具体如下：

<servlet>

<servlet-name>dwr-invoker</servlet-name>

<servlet-class>org.directwebremoting.servlet.DwrServlet</servlet-class>

**<init-param>**

**<param-name>activeReverseAjaxEnabled</param-name>**

**<param-value>true</param-value>**

**</init-param>**

...

</servlet>

### 修改web页面

在使用Reverse Ajax的web页面的onload事件中添加下面的方法调用，用来开始Comet/Polling周期。

dwr.engine.setActiveReverseAjax(true);

## 配置Reverse Ajax

Reverse Ajax有两种基本工作模式，消极和积极，积极Reverse Ajax模式又可以拆分成下面三种模式：

* Early Closing模式（默认）
* Full Streaming模式（IE浏览器不可用）
* Polling模式

### 通用配置

要想使用上面的三种模式，必须要先启用积极Reverse Ajax模式，启用方法见4.3节内容

### Early Closing模式

使用Early Closing模式和使用Full Streaming一样，DWR都会保持连接的打开。一旦向Browser推送完数据，DWR会等待一个配置的时间（maxWaitAfterWrite）后自动关闭连接，强制代理通过任何消息。使用Early Closing模式，不需要任何配置，因为该模式是默认的。maxWaitAfterWrite参数默认设置为500ms，可以通过init-param参数修改：

<init-param>

<param-name>maxWaitAfterWrite</param-name>

<param-value>1000</param-value>

</init-param>

Early Closing模式的缺点是，当服务器高速率的输出时，会导致大量的命中计数。这种情况可以通过设置maxWaitAfterWrite=1000或更大来增加连接存活时间。

### Full Streaming模式（IE不可用）

Full Streaming模式有最快速的响应特性，因为如果没有数据输出它能最长保持60秒连接打开。由于Internet Explorer用户无法使用全Full Streaming模式。所有Internet Explorer的用户将被自动转到Early Closing模式。

开启Full Streaming模式，设置maxWaitAfterWrite参数值为-1：

<init-param>

<param-name>maxWaitAfterWrite</param-name>

<param-value>-1</param-value>

</init-param>

### Polling模式

如果觉得保持长连接是不明智的，DWR可以使用Polling模式，除了需要设置activeReverseAjaxEnabled=true，还需要设置PollingServerLoadMonitor：

<init-param>

<param-name>org.directwebremoting.extend.ServerLoadMonitor</param-name>

<param-value>org.directwebremoting.impl.PollingServerLoadMonitor</param-value>

</init-param>

Polling模式下，默认的poll频率是每5秒钟，这个值可以通过下面的参数自定义：

<init-param>

<param-name>disconnectedTime</param-name>

<param-value>60000</param-value>

</init-param>

上例中是每60秒poll一次，这对于大多数的应用60秒都是足够了，这也可以允许web服务器保持一个非常大数量的连接。

## 消极Reverse Ajax模式

消极模式是默认模式，Browser不需要维持一个额外的到web server的连接，DWR会在server上用一个队列按照顺序来保存要发给客户端的数据，以便下次Browser建立连接时一次性发给客户端。

# Reverse Ajax提示和技巧

## ScriptSession生命周期

ScriptSession在/dwr/engine.js被页面加载是被创建。默认情况下，ScriptSession的生命周期由org.directwebremoting.impl.DefaultScriptSessionManager类来管理。

如果你在web页面中调用了下面的javascript函数：

dwr.engine.setNotifyServerOnPageUnload(true);

当web页面unloaded时，一个远程的DWR调用会通知ScriptSessionManager类来让当前ScriptSession失效。默认情况下，DWR会采用同步调用方式以保证该调用的成功，但是同步调用特性可能会导致关闭Browser时有一点延迟。如果这不是你想要的，你可以通过设置可选的第二个参数，如下：

dwr.engine.setNotifyServerOnPageUnload(true, true);

第二个参数告诉DWR以异步的方式调用unloader方法，这回导致Browser关闭连接和unloader方法被调用产生不一致。

如果dwr.engine.setNotifyServerOnPageUnload没有设置为true，或者调用失败，ScriptSessions会使用timeout策略以自动失效，server上的ScriptSession超过**5分钟**不活动就会被自动失效。

## 在non-dwr线程中获取request信息

由于non-dwr线程中无法获取dwr线程的引用（reference），所以WebContextFactory().get().getScriptSession()将会返回null，因此需要从DWR线程向non-dwr线程传送数据。

## 在non-dwr线程中获得ScriptSessionManager对象

Container container = ServerContextFactory.get().getContainer();

ScriptSessionManager manager = container.getBean(ScriptSessionManager.class);

## 如何获取指定ScriptSessions

DWR的Browser API为更新Browser提供了一系列有用的方法。大多数Browser方法都有一个ScriptSessionFilter类型参数，用来通过attributes获取指定的ScriptSessions。

使用实例：利用Browserc API的ScriptSessionFilter来区分用户

### 实现ScriptSessionFilterc接口

public class TestScriptSessionFilter implements ScriptSessionFilter

{

private String attributeName;

public TestScriptSessionFilter(String attributeName)

{

this.attributeName = attributeName;

}

/\* (non-Javadoc)

\* @see org.directwebremoting.ScriptSessionFilter#match(org.directwebremoting.ScriptSession)

\*/

public boolean match(ScriptSession session)

{

Object check = session.getAttribute(attributeName);

return (check != null && check.equals(Boolean.TRUE));

}

}

### 在ScriptSession中设置attribute

// Add the attribute into the ScriptSession sometime before using the Filter.

ScriptSession scriptSession = WebContextFactory.get().getScriptSession();

String attributeName = "attr";

scriptSession.setAttribute(attributeName, true);

注意：这些必须在DWR初始化时设置。

### 使用ScriptSessionFilter

1、直接设置web页面中div元素的值

ScriptSessionFilter filter = new TestScriptSessionFilter(attributeName);

Browser.withPageFiltered(page, filter, new Runnable()

{

public void run()

{

// Call a method on DWR's Util class which sets the value on an element on your HTML page with a id of "divID".

Util.setValue("divID", "value of div");

}

});

2、直接调用web页面中javascript方法

ScriptSessionFilter filter = new TestScriptSessionFilter(attributeName);

Browser.withPageFiltered(page, filter, new Runnable()

{

public void run()

{

// Call a named function from your html page. Note - The ScriptsSessions.addFunctionCall will only

// send the function call to ScriptSessions matching TestScriptSessionFilter. ScriptSessions.addFunctionCall("yourJavaScriptFunctionName", arg1, arg2, etc.);

}

});

3、在web页面中添加任意脚本

ScriptSessionFilter filter = new TestScriptSessionFilter(attributeName);

Browser.withPageFiltered(page, filter, new Runnable()

{

public void run()

{

// Add script which will modify the document.title. on your html page.

ScriptSessions.addScript(("document.title = 'My new title, from DWR reverse AJAX!';"));

}

});

**严重注意**：有一些Browser方法需要来自DWR线程的WebContext请求，目前这些方法包括：withCurrentPageFiltered, withCurrentPage, and getTargetSessions，其它所有的方法都可以在non-dwr线程中安全调用。

### 在ScriptSession中设置不同的attribute

一个用来区分访问同一个页面的不同用户的最常见的方式是通过在ScriptSession中设置attribute，并在Reverse Ajax线程中获取这些attribute。由于ScriptSession直到web页面加载了engine.js之后才被创建，所以这些事情没法在MVC控制器中或JSP页面中完成。

目前有两种最好的做法在ScriptSession中设置attribute。

#### 调用远程DWR方法

一般是在web页面的onload方法中，通过DWR远程调用下面的方法，方法中的属性值可以冲session中获取，也可以从页面中传递过来。

/\*\*

\* This method should be remoted via DWR and generally called before reverse ajax is initialized.

\* You may choose to call this method when your page which uses reverse AJAX is initialized, then

\* in your callback function you may initialize reverse AJAX (dwr.engine.setActiveReverseAjax(true);)

\* and be certain that the

\*/

public void remoteMethod() {

String value = "someValue"; // this may come from the HttpSession for example

ScriptSession scriptSession = WebContextFactory.get().getScriptSession();

scriptSession.setAttribute("key", value);

}

#### 使用ScriptSessionListener监听

ScriptSessionListener会在ScriptSession被创建和销毁时收到通知，示例代码如下：

/\*\*

\* When a ScriptSession is created set a "userId" attribute on the ScriptSession.

\* In this case userId is an attribute you have set on the HttpSession that uniquely

\* identifies the user.

\*/

public void sessionCreated(ScriptSessionEvent ev) {

HttpSession session = WebContextFactory.get().getSession();

String userId = (String) session.getAttribute("userId");

ev.getSession().setAttribute("userId", userId);

}

一旦向ScriptSession中设置了attribute之后，Reverse Ajax线程就可以通过Browser API接口，并且使用ScriptSessionFilter去匹配到指定的ScriptSession或者获得到ScriptSession里的attribute。

## 通过request参数区分ScriptSession

另外一种区分访问同一页面的不同用户的方法是通过request参数。为了使用该功能，需要设置normalizeIncludesQueryString参数init-param值为true。启用这个参数后，ScriptSessionManager会包含request的详细参数，因此可以通过参数来区分：

scriptSessionManager.getScriptSessionsByPage("/page?foo=bar")

## ScriptSessionListeners

ScriptSessionListeners可以监听ScriptSession的创建和销毁。配置ScriptSessionListeners，需要类似下面的代码：

Container container = ServerContextFactory.get().getContainer();

ScriptSessionManager manager = container.getBean(ScriptSessionManager.class);

ScriptSessionListener listener = new ScriptSessionListener() {

public void sessionCreated(ScriptSessionEvent ev) {

HttpSession session = WebContextFactory.get().getSession();

String userId = (String) session.getAttribute("userId");

ev.getSession().setAttribute("userId", userId);

}

public void sessionDestroyed(ScriptSessionEvent ev) { }

};

manager.addScriptSessionListener(listener);

严重注意：ScriptSessionListeners必须在DWR初始化之后再添加，通常有两种方式：

### 扩展原DWR servlet

扩展DwrServlet，在DWR servlet初始化完成之后在调用上面的方法：

示例如下：

**public** **class** AlarmDwrServlet **extends** DwrServlet {

/\*\*

\*

\*/

**private** **static** **final** **long** *serialVersionUID* = -2101972807775729198L;

@Override

**public** **void** init(ServletConfig config) **throws** ServletException {

// 正常初始化DWR

**super**.init(config);

**this**.initListener();

}

/\*\*

\* 注册监听，DwrServlet初始化后调用 <br>

\*

\* Author: liujianliang<br>

\* Create Date: 2014-9-15 上午11:23:35<br>

\* History: 2014-9-15 上午11:23:35 liujianliang Created.<br>

\*

\* **@throws** ServletException

\*

\*/

**private** **void** initListener() **throws** ServletException {

Container container = ServerContextFactory.*get*().getContainer();

ScriptSessionManager manager = container.getBean(ScriptSessionManager.**class**);

ScriptSessionListener listener = **new** ScriptSessionListener() {

**public** **void** sessionCreated(ScriptSessionEvent ev) {

HttpServletRequest request = WebContextFactory.*get*().getHttpServletRequest();

SysUserInfo userInfo = UserUtil.*getUserInfo*(request);

**if** (userInfo != **null**) {

System.*out*.println("====== scriptSession [" + ev.getSession().getPage()

+ "] is created for user [" + userInfo.getSysUserName() + "] from ["

+ request.getRemoteHost() + "] !");

ev.getSession().setAttribute("sysUserName", userInfo.getSysUserName());

ev.getSession().setAttribute("userHost", request.getRemoteHost());

}

}

**public** **void** sessionDestroyed(ScriptSessionEvent ev) {

System.*out*.println("\*\*\*\*\*\* scriptSession [" + ev.getSession().getPage() + "] is distroyed");

}

};

// 为当前页面添加监听

manager.addScriptSessionListener(listener);

}

}

### 自定义一个servlet

自定义一个servlet，在初始化方法中调用上面的代码。最后在WEB-INF/web.xml文件中设置新servlet的<load-on-startup />值比DWR servlet大，即要比DWR servlet晚加载和初始化。

<load-on-startup>large than DWR servlet</load-on-startup>

示例代码如下：

**public** **class** AlarmDwrServlet **extends** HttpServlet {

/\*\*

\*

\*/

**private** **static** **final** **long** *serialVersionUID* = -2101972807775729198L;

@Override

**public** **void** init(ServletConfig config) **throws** ServletException {

**super**.init(config);

Container container = ServerContextFactory.*get*().getContainer();

ScriptSessionManager manager = container.getBean(ScriptSessionManager.**class**);

ScriptSessionListener listener = **new** ScriptSessionListener() {

**public** **void** sessionCreated(ScriptSessionEvent ev) {

HttpServletRequest request = WebContextFactory.*get*().getHttpServletRequest();

SysUserInfo userInfo = UserUtil.*getUserInfo*(request);

**if** (userInfo != **null**) {

System.*out*.println("====== scriptSession [" + ev.getSession().getPage()

+ "] is created for user [" + userInfo.getSysUserName() + "] from ["

+ request.getRemoteHost() + "] !");

ev.getSession().setAttribute("sysUserName", userInfo.getSysUserName());

ev.getSession().setAttribute("userHost", request.getRemoteHost());

}

}

**public** **void** sessionDestroyed(ScriptSessionEvent ev) {

System.*out*.println("\*\*\*\*\*\* scriptSession [" + ev.getSession().getPage() + "] is distroyed");

}

};

// 为当前页面添加监听

manager.addScriptSessionListener(listener);

}

}

# 附：Browser API

browser api地址：

<http://directwebremoting.org/dwr/javadoc/org/directwebremoting/Browser.html>

|  |  |
| --- | --- |
| **Constructor Summary** | |
| [**Browser**](http://directwebremoting.org/dwr/javadoc/org/directwebremoting/Browser.html#Browser())() |  |

|  |  |
| --- | --- |
| **Method Summary** | |
| static void | [**close**](http://directwebremoting.org/dwr/javadoc/org/directwebremoting/Browser.html#close(java.lang.Object))(java.lang.Object proxy)            If a browser passes an object or function to DWR then DWR may need to release the reference that DWR keeps. |
| static java.util.Collection<[ScriptSession](http://directwebremoting.org/dwr/javadoc/org/directwebremoting/ScriptSession.html)> | [**getTargetSessions**](http://directwebremoting.org/dwr/javadoc/org/directwebremoting/Browser.html#getTargetSessions())()            This method discovers the sessions that are currently being targeted by browser updates. |
| static void | [**withAllSessions**](http://directwebremoting.org/dwr/javadoc/org/directwebremoting/Browser.html#withAllSessions(java.lang.Runnable))(java.lang.Runnable task)            Execute some task (represented by a Runnable) and aim the output at all browser window open at all pages in this web application. |
| static void | [**withAllSessions**](http://directwebremoting.org/dwr/javadoc/org/directwebremoting/Browser.html#withAllSessions(org.directwebremoting.ServerContext, java.lang.Runnable))([ServerContext](http://directwebremoting.org/dwr/javadoc/org/directwebremoting/ServerContext.html) serverContext, java.lang.Runnable task)            As [withAllSessions(Runnable)](http://directwebremoting.org/dwr/javadoc/org/directwebremoting/Browser.html#withAllSessions(java.lang.Runnable)), but for use when there is more than one copy of DWR in the ServletContext. |
| static void | [**withAllSessionsFiltered**](http://directwebremoting.org/dwr/javadoc/org/directwebremoting/Browser.html#withAllSessionsFiltered(org.directwebremoting.ScriptSessionFilter, java.lang.Runnable))([ScriptSessionFilter](http://directwebremoting.org/dwr/javadoc/org/directwebremoting/ScriptSessionFilter.html) filter, java.lang.Runnable task)            Execute a task an send the output to a subset of the total list of users. |
| static void | [**withAllSessionsFiltered**](http://directwebremoting.org/dwr/javadoc/org/directwebremoting/Browser.html#withAllSessionsFiltered(org.directwebremoting.ServerContext, org.directwebremoting.ScriptSessionFilter, java.lang.Runnable))([ServerContext](http://directwebremoting.org/dwr/javadoc/org/directwebremoting/ServerContext.html) serverContext, [ScriptSessionFilter](http://directwebremoting.org/dwr/javadoc/org/directwebremoting/ScriptSessionFilter.html) filter, java.lang.Runnable task)            As [withAllSessionsFiltered(ScriptSessionFilter, Runnable)](http://directwebremoting.org/dwr/javadoc/org/directwebremoting/Browser.html#withAllSessionsFiltered(org.directwebremoting.ScriptSessionFilter, java.lang.Runnable)), but for use when there is more than one copy of DWR in the ServletContext. |
| static void | [**withCurrentPage**](http://directwebremoting.org/dwr/javadoc/org/directwebremoting/Browser.html#withCurrentPage(java.lang.Runnable))(java.lang.Runnable task)            Execute a task and aim the output at all the browser windows open at the same page as the current request. |
| static void | [**withCurrentPageFiltered**](http://directwebremoting.org/dwr/javadoc/org/directwebremoting/Browser.html#withCurrentPageFiltered(org.directwebremoting.ScriptSessionFilter, java.lang.Runnable))([ScriptSessionFilter](http://directwebremoting.org/dwr/javadoc/org/directwebremoting/ScriptSessionFilter.html) filter, java.lang.Runnable task)            Execute a task and aim the output at a subset of the browser windows open at the same page as the current request. |
| static void | [**withPage**](http://directwebremoting.org/dwr/javadoc/org/directwebremoting/Browser.html#withPage(org.directwebremoting.ServerContext, java.lang.String, java.lang.Runnable))([ServerContext](http://directwebremoting.org/dwr/javadoc/org/directwebremoting/ServerContext.html) serverContext, java.lang.String page, java.lang.Runnable task)            As [withPage(String, Runnable)](http://directwebremoting.org/dwr/javadoc/org/directwebremoting/Browser.html#withPage(java.lang.String, java.lang.Runnable)), but for use when there is more than one copy of DWR in the ServletContext. |
| static void | [**withPage**](http://directwebremoting.org/dwr/javadoc/org/directwebremoting/Browser.html#withPage(java.lang.String, java.lang.Runnable))(java.lang.String page, java.lang.Runnable task)            Execute a task and aim the output at all the browser windows open at a given page in this web application. |
| static void | [**withPageFiltered**](http://directwebremoting.org/dwr/javadoc/org/directwebremoting/Browser.html#withPageFiltered(org.directwebremoting.ServerContext, java.lang.String, org.directwebremoting.ScriptSessionFilter, java.lang.Runnable))([ServerContext](http://directwebremoting.org/dwr/javadoc/org/directwebremoting/ServerContext.html) serverContext, java.lang.String page, [ScriptSessionFilter](http://directwebremoting.org/dwr/javadoc/org/directwebremoting/ScriptSessionFilter.html) filter, java.lang.Runnable task)            As [withPageFiltered(java.lang.String, org.directwebremoting.ScriptSessionFilter, java.lang.Runnable)](http://directwebremoting.org/dwr/javadoc/org/directwebremoting/Browser.html#withPageFiltered(java.lang.String, org.directwebremoting.ScriptSessionFilter, java.lang.Runnable)), but for use when there is more than one copy of DWR in a ServletContext. |
| static void | [**withPageFiltered**](http://directwebremoting.org/dwr/javadoc/org/directwebremoting/Browser.html#withPageFiltered(java.lang.String, org.directwebremoting.ScriptSessionFilter, java.lang.Runnable))(java.lang.String page, [ScriptSessionFilter](http://directwebremoting.org/dwr/javadoc/org/directwebremoting/ScriptSessionFilter.html) filter, java.lang.Runnable task)            Execute a task and aim the output at a subset of the users on a page. |
| static void | [**withSession**](http://directwebremoting.org/dwr/javadoc/org/directwebremoting/Browser.html#withSession(org.directwebremoting.ServerContext, java.lang.String, java.lang.Runnable))([ServerContext](http://directwebremoting.org/dwr/javadoc/org/directwebremoting/ServerContext.html) serverContext, java.lang.String sessionId, java.lang.Runnable task)            As [withSession(String, Runnable)](http://directwebremoting.org/dwr/javadoc/org/directwebremoting/Browser.html#withSession(java.lang.String, java.lang.Runnable)), but for use when there is more than one copy of DWR in a ServletContext. |
| static void | [**withSession**](http://directwebremoting.org/dwr/javadoc/org/directwebremoting/Browser.html#withSession(java.lang.String, java.lang.Runnable))(java.lang.String sessionId, java.lang.Runnable task)            Execute a task and aim the output at a specific script session. |

|  |
| --- |
| **Methods inherited from class java.lang.Object** |
| equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait |