# 简介

DWR是一个用于改善web页面与Java类交互的远程服务器端Ajax开源框架，DWR通过动态把Java类生成JavaScript文件给web浏览器使用，它允许web浏览器很轻松地调用Java代码，就像它在本地运行一样，但实际上服务器正在执行代码，DWR正在将数据编组回来。出于安全的考虑，WEB开发者必须适当地配置哪些Java类可以安全的被外部使用。

每个java类将会生成一个js文件，供前端引入并调用该类所配置的方法，引入格式：dwr/interface/xxxx.js（xxxx是类开启远程代理的别名）。与此同时，还需引入另外两个js：dwr/engine.js和dwr/util.js。

DWR既允许web浏览器调用服务器Java代码，又可以通过反向Ajax使服务器调用web浏览器的JS方法来动态地更新页面内容。

DWR是一个RPC库，它可以方便地从JavaScript调用Java函数，并从Java调用JavaScript函数。

## 服务器推送3种实现方式

DWR的逆向Ajax主要包括两种模式：主动模式和被动模式。其中主动模式包括polling和comet两种，被动模式只有piggyback（不会实时获取服务器的消息）这一种。这三种方式，都可以在dwr的配置文件中进行相关的配置，若不进行配置，则默认是下面的第一种piggyback。

1. piggyback方式

这是dwr默认的方式，消极Reverse Ajax模式。如果后台有什么内容需要推送到前台，是要等到那个页面进行下一次ajax请求的时候，将需要推送的内容附加在该次请求之后，传回到页面。只有等到下次请求页面主动发起了，中间的变化内容才传递回页面。

1. comet方式

当服务端建立和浏览器的连接，将页面内容发送到浏览器之后，对应的连接并不关闭，只是暂时挂起。如果后面有什么新的内容需要推送到客户端的时候直接通过前面挂起的连接再次传送数据。服务器所能提供的连接数目是一定的，在大量的挂起的连接没有关闭的情况下，可能造成新的连接请求不能接入，从而影响到服务质量。

comet方式就是一种长连接机制(long lived http)。同样是由Browser端主动发起请求，但是它允许服务器非常慢地开始响应浏览器的请求，并按照服务器所指示的时间表继续回答。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。

1. polling方式（轮询方式）

由浏览器定时向服务端发送ajax请求，询问后台是否有什么内容需要推送，有的话就会由服务端返回推送内容。这种方式和我们直接在页面通过定时器发送 ajax请求，然后查询后台是否有变化内容的实现是类似的。只不过用了dwr之后这部分工作由框架帮我们完成了。

每种方法都有一些好处。轮询方式很容易实现，但是它可以很容易地使服务器超载。相比之下，Comet方式是一种比较好的方式，在服务器上容易实现且对服务器造成的压力小，Comet也有非常低的延迟——没有必要等待下一次浏览器连接。轮询方式和comet方式都需要额外的网络连接，因此真正的低开销版本是piggyback方式，但这可能是非常高的延迟。

dwr实现了三种方式，只需要通过更改配置即可切换（piggyback方式不需要，默认模式）。

## Reverse Ajax模式

Reverse Ajax有两种基本工作模式，即消极模式（piggyback方式，dwr中不需要做任何配置）与积极模式。

开启Reverse Ajax积极模式，必须在WEB-INF/web.xml中增加以下参数（与springmvc集成时，可在springmvc配置文件中增加相应的配置）：

<init-param>

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

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

</init-param>

同时，在页面中还需增加以下两项：

//激活反ajax功能

dwr.engine.setActiveReverseAjax(true);

积极Reverse Ajax模式可以分为以下三种：

1. Early Closing模式（默认）

开启Reverse Ajax后，Early Closing模式不需要做任何配置。其中maxWaitAfterWrite（等待maxWaitAfterWrite时间，如果没有信息，则关闭连接。）参数默认设置维500ms，也可以通过以下修改增加连接存活时间。

<init-param>

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

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

</init-param>

1. Full Streaming模式（不支持IE，自动切换回Early Closing模式）

全流模式具有最快的响应特性，因为它每60秒关闭一次连接，以检查浏览器是否仍然处于活动状态。但因为该模式需要发送HTTP请求，由于外界因素导致出现问题（60秒返回一次信息）。

<init-param>

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

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

</init-param>

1. Polling模式

开启polling方式，在开启Reverse Ajax积极模式的基础上增加以下配置：

<init-param>

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

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

</init-param>

默认的轮询率为每5秒，可以通过以下配置项（单位：毫秒）修改：

<init-param>

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

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

</init-param>

## 长连接机制对Web服务器的要求

Reverse Ajax积极模式中的Early Closing模式和Full Streaming模式均采用长连接机制，随着大量连接被打开，线程可能会让应用服务器挤满。一些web服务器有一个特殊的模式，允许用户删除线程。同时DWR用户还具有创建和插件一个ServerLoadMonitor实现的能力，如果服务器负载过高，可以动态地修改断开连接的时间。

如使用tomcat服务器，则在web.xml中配置针对tomcat的servlet处理请求（针对comet方式），就可以由tomcat配置文件中配置的连接存活时间来控制连接的存活时间。

<servlet>

<display-name>DWR Servlet</display-name>

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

<servlet-class> org.directwebremoting.server.tomcat.DwrCometProcessor</servlet-class>

<init-param>

<param-name>debug</param-name>

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

</init-param>

<!-- 启动反ajax，必须启动，不然后台无法往前端发送消息 -->

<init-param>

<param-name>activeReverseAjaxEnabled</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>

其次，可以实现ServerLoadMonitor来动态地修改断开连接的时间。增加一个参数配置即可（servlet类还是使用org.directwebremoting.servlet.DwrServlet）。如下：

<init-param>

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

<param-value>com.example.MyCustomServerLoadMonitor</param-value>

</init-param>

## 应用场景

比如页面要实时获取用户的最新通知消息，如果客服端调用ajax在后台不断的查询服务器，看是否有关于自己的消息，如果有则查询返回。这种做法肯定会大量的占用系统的资源，是一种效率比较低的做法，不可取。

而现在DWR的反转AJAX功能，允许我们从服务器端来控制客服端，不需要客户端请求，服务器就可以自动把消息发给指定的客户端，还可以把消息发给一批在线用户。

# 快速应用

## 简单的web工程使用dwr（dwrtest1项目）

### 引入maven依赖

<dependency>

<groupId>org.directwebremoting</groupId>

<artifactId>dwr</artifactId>

<version>3.0.2-RELEASE</version>

</dependency>

<dependency>

<groupId>commons-logging</groupId>

<artifactId>commons-logging</artifactId>

<version>1.2</version>

</dependency>

### WEB-INF/web.xml加入dwr的servlet请求处理

<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>

<!-- 启动反ajax，必须启动，不然后台无法往前端发送消息 -->

<init-param>

<param-name>activeReverseAjaxEnabled</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>

### 添加WEB-INF/dwr.xml

DWR配置文件定义了DWR可以创建的类，以及Javascript可以使用的远程JAVA类。下面的例子中，定义了两个类，这些类是远程的，并在Javascript中给出类名。同时还可以指定类中哪些方法可以供Javascript调用或哪些方法不可以供Javascript调用。（可查官方文档的dwr.xml相关配置）

配置文件格式：

<!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>

<!-- Creator设定为new，表示使用Test1的无参构造函数生成对象，

javascript设定为test1，表示客戶端JavaScript程式可以

使用test1来调用的com.cctsort.dwrtest1.Test1的函数。 -->

<create creator=*"new"* javascript=*"test1"*>

<param name=*"class"* value=*"com.cctsort.dwrtest1.Test1"*/>

<exclude method=*"method5"*/><!-- 不供调用的方法 -->

<!-- <include method="method1"/> 可调用的方法 exclude与 include不能混合使用-->

</create>

<create creator=*"new"* javascript=*"test2"*>

<param name=*"class"* value=*"com.cctsort.dwrtest1.Test2"*/>

</create>

<!-- 必须配置声明返回值中的对象，这样在页面中可以直接通过类的属性取值

或者创建对象，把数据指定到相应的属性，最后把对象作为参数传递给方法

总之在页面中可以像在服务器端一样创建对象与获取对象的属性值

在页面中创建对象时，使用var people = new Object() -->

<convert converter=*"bean"* match=*"com.cctsort.dwrtest1.entities.People"*/>

</allow>

</dwr>

相关类的定义：

Date类系java的util下

Test1类内容：

**package** com.cctsort.dwrtest1;

**import** java.util.ArrayList;

**import** java.util.List;

**import** com.cctsort.dwrtest1.entities.People;

**public** **class** Test1 {

**public** **void** method() {

System.***out***.println("==method==无参数及返回值");

}

**public** String method1() {

System.***out***.println("==method1==");

**return** "==method1==";

}

**public** String method2(String msg) {

System.***out***.println("==method2==" + msg);

**return** "==method2==" + msg;

}

**public** String method3() {

System.***out***.println("==method3==");

**return** "==method3==";

}

**public** People method4() {

System.***out***.println("==method4==");

People people = **new** People();

people.setId(1);

people.setName("张三");

**return** people;

}

**public** String method5() {

System.***out***.println("==method5==");

**return** "==method5==";

}

**public** String method6(People people) {

System.***out***.println(people.getId() + "==method6==" + people.getName());

**return** people.getId() + "==method6==" + people.getName();

}

**public** List<People> method7() {

System.***out***.println("==method7==");

List<People> list = **new** ArrayList<People>();

People people = **new** People();

people.setId(10);

people.setName("王武");

list.add(people);

people = **new** People();

people.setId(11);

people.setName("陈七");

list.add(people);

**return** list;

}

}

Test2类内容：

**package** com.cctsort.dwrtest1;

**import** java.text.SimpleDateFormat;

**import** java.util.ArrayList;

**import** java.util.Collection;

**import** java.util.Date;

**import** java.util.List;

**import** java.util.Timer;

**import** java.util.TimerTask;

**import** org.directwebremoting.Browser;

**import** org.directwebremoting.ScriptBuffer;

**import** org.directwebremoting.ScriptSession;

**import** org.directwebremoting.ScriptSessionFilter;

**import** org.directwebremoting.WebContextFactory;

**public** **class** Test2 {

**public** **static** **final** String ***DEFAULT\_MARK*** = "userId";//会话ScriptSession中身份标识，用于区分不同用户ScriptSession

**public** **static** **final** String ***DEFAULT\_METHOD\_NAME*** = "showMessage";//javascript中接收信息的方法

**public** **static** SimpleDateFormat *format* = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

**public** **static** List<String> *list* = **new** ArrayList<String>();

**static** {

Timer timer = **new** Timer();

timer.schedule(**new** TimerTask() {

**public** **void** run() {

System.***out***.println("=======开始发送消息==========");

**for**(String userId: *list*) {

//发送给订阅该消息用户

**if**(userId != **null** && userId.startsWith("msgType@")) {

*sendMessageSingle*(userId, *format*.format(**new** Date()));

System.***out***.println("==发送==" + userId);

}

}

}

}, 10000, 1000);

}

/\*\*

\* 接收上线用户，需要维护

\*

\* **@param** userId

\*/

**public** **void** onPageLoad(String userId) {

//获取此次会话的ScriptSession对象，

//注意一个HttpSession会有多个ScriptSession，可以使用Map<HttpSessionId, ScriptSession>

//或Map<userId, ScriptSession>，最好使用后者，可允许同一用户打开不同页面接收不同的消息

//（即可以给一个用户推送不同的消息到不同的页面）

//可以实现ScriptSessionListener类，实现对ScriptSession的创建与销毁

//此处不维护ScriptSession，仅保存订阅用户userId到list中（图方便，不可取，

//如果用户退出，不会从list中移除）

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

scriptSession.setAttribute(***DEFAULT\_MARK***, userId);//打标记，

//记录在list中

*list*.add(userId);

System.***out***.println("==========订阅用户============" + userId);

}

/\*\*

\* 推送消息给单个用户

\*

\* **@param** userid

\* **@param** message

\*/

**public** **static** **void** sendMessageSingle(String userid, String message) {

**final** String userId = userid;

**final** String autoMessage = message;

Browser.*withAllSessionsFiltered*(**new** ScriptSessionFilter() {

//过滤，精准推送给指定用户

**public** **boolean** match(ScriptSession session) {

**if** (session.getAttribute(***DEFAULT\_MARK***) == **null**)//找不到身份标识，放弃

**return** **false**;

**else**

//如果找到相应的用户，则返回true

**return** (session.getAttribute(***DEFAULT\_MARK***)).equals(userId);

}

}, **new** Runnable() {

**private** ScriptBuffer script = **new** ScriptBuffer();

**public** **void** run() {

//调用javascript的showMessage方法，把数据传输给页面

script.appendCall(***DEFAULT\_METHOD\_NAME***, autoMessage);

//下面并不是发给所有用户，上面做了过滤

Collection<ScriptSession> sessions = Browser.*getTargetSessions*();

**for** (ScriptSession scriptSession : sessions) {

scriptSession.addScript(script);

}

}

});

}

}

People类内容：

**package** com.cctsort.dwrtest1.entities;

**public** **class** People {

**private** **int** id;

**private** String name;

**public** People() {

}

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

}

### 发布Web验证

发布完成后，访问如下地址，即可看到在dwr.xml中配置的java类，且点击某一个java类还可以看到可调用的方法，可以点击调用这些方法并获取返回结果。

http://localhost:8080/***[工程名称]***/dwr/

例如：http://localhost:8080/**dwrtest1**/dwr/

### 在页面中调用java方法

首先，需要引入DWR JavaScript 文件，其他每个类对应一个js文件，即以下的第一类js与dwr.xml配置的java类有关，配置几个就要引入几个，也可以引入其中某一个或几个。（下面系相对路径，可加[http://localhost:8080/***[工程名称]***/](http://localhost:8080/%5b工程名称%5d/)，需**注意引入顺序**。）

<script src='dwr/engine.js'></script>

<script src='dwr/util.js'></script>

<script src='dwr/interface/ [类引用，对应dwr.xml中的javascript项].js'></script>

<script src='dwr/interface/ [类引用，对应dwr.xml中的javascript项].js'></script>

其次，调用java类对应的js方法。如果调用的java方法有返回值，则需要传递回调函数来接收返回值。如：类引用.method(callBcakFunction);

例如：

<!DOCTYPE html>

<html>

<head>

<title>test1.html</title>

</head>

<body>

<label id=*"labelId1"*></label>

<label id=*"labelId2"*></label>

<label id=*"labelId3"*></label>

<label id=*"labelId4"*></label>

<label id=*"labelId6"*></label>

<label id=*"labelId7"*></label>

</body>

<script src=*'dwr/engine.js'*></script>

<script src=*'dwr/util.js'*></script>

<script src=*'dwr/interface/test1.js'*></script>

<script type=*"text/javascript"*>

//需要传递参数，也没有返回值，则不需要传递回调函数

test1.method();

test1.method1(**function**(data) {

document.getElementById('labelId1').innerHTML = data + "<br/>";

});

//传递参数和回调函数，注意按java方法参数排列顺序把参数排在回调函数前

test1.method2("前端调用java方法", **function**(data) {

document.getElementById('labelId2').innerHTML = data + "<br/>";

});

test1.method3(callBackFunc3);

//回调函数

**function** callBackFunc3(data) {

document.getElementById('labelId3').innerHTML = data + "<br/>";

}

test1.method4(**function**(data) {

document.getElementById('labelId4').innerHTML = data.id + "==" + data.name + "<br/>";

});

**var** people = **new** Object();

people.id = 2;

people.name = '李四';

test1.method6(people, **function**(data) {

document.getElementById('labelId6').innerHTML = data + "<br/>";

});

test1.method7(**function**(data) {

**var** str = "";

**for**(**var** i=0;i < data.length;i++) {

str = str + data[i].id + "==" + data[i].name + "======";

}

document.getElementById('labelId7').innerHTML = str + "<br/>";

});

</script>

</html>

### java后台调用javascript方法

<!DOCTYPE html>

<html>

<head>

<title>test2.html</title>

</head>

<body>

<label id=*"labelId1"*></label>

</body>

<script src=*'dwr/engine.js'*></script>

<script src=*'dwr/util.js'*></script>

<script src=*'dwr/interface/test2.js'*></script>

<script type=*"text/javascript"*>

//激活反ajax功能

dwr.engine.setActiveReverseAjax(**true**);

//默认为异步,

dwr.engine.setAsync(**true**);

//模拟用户id，用于区分不同的访问者

**function** guid() {

**function** S4() {

**return** (((1+Math.random())\*0x10000)|0).toString(16).substring(1);

}

**return** (S4()+S4()+"-"+S4()+"-"+S4()+"-"+S4()+"-"+S4()+S4()+S4());

}

//告知后台在线等待接收信息的用户，

//msgType可以是不同的值，用户告知后台需要获取的实时信息

test2.onPageLoad("msgType@" + guid());

//接收消息方法，java后台通过反ajax调用

**function** showMessage(sendMessages, clickEvent) {

document.getElementById('labelId1').innerHTML = "当前时间：" + sendMessages;;

}

//重写错误方法，异常处理

dwr.engine.\_errorHandler = **function**(message, ex) {

};

</script>

</html>

## springMVC与dwr整合（dwrtest2项目）

在2.1中，说到要自己管理ScriptSession，在DWR3.0中推出了 ScriptSessionListener用来监听ScriptSession的创建及销毁事件。这里将结合springmvc新建一个MyScriptSessionListener类实现ScriptSessionListener接口，由PushMessageCompont类提供方法给javascript访问，PushMessageUtil工具类将消息推送到用户，ScriptSessionManager类继承dwr的DefaultScriptSessionManager类来启动MyScriptSessionListener类监听dwr容器创建与销毁ScriptSession，从而来维护自己的ScriptSession。

### 引入maven依赖

这里忽略spring mvc相关的依赖。

<dependency>

<groupId>org.directwebremoting</groupId>

<artifactId>dwr</artifactId>

<version>3.0.2-RELEASE</version>

</dependency>

<dependency>

<groupId>commons-collections</groupId>

<artifactId>commons-collections</artifactId>

<version>3.2.2</version>

</dependency>

<dependency>

<groupId>commons-lang</groupId>

<artifactId>commons-lang</artifactId>

<version>2.6</version>

</dependency>

### WEB-INF/web.xml加入dwr的servlet请求处理

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app xmlns:javaee=*"http://java.sun.com/xml/ns/javaee"* xmlns:web=*"http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"*>

<javaee:display-name>dwrtest2</javaee:display-name>

<!-- 启动顺序：context-param====listener====filter====servlet -->

<!-- =================Spring配置================== -->

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:applicationContext.xml</param-value>

</context-param>

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<!-- 防止spring内存溢出监听器 -->

<listener>

<listener-class>org.springframework.web.util.IntrospectorCleanupListener</listener-class>

</listener>

<!-- ===============设置编码============== -->

<filter>

<filter-name>encodingFilter</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<async-supported>true</async-supported>

<init-param>

<param-name>encoding</param-name>

<param-value>UTF-8</param-value>

</init-param>

<init-param>

<param-name>forceEncoding</param-name>

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

</init-param>

</filter>

<filter-mapping>

<filter-name>encodingFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<!-- =============Spring MVC servlet=============== -->

<servlet>

<servlet-name>SpringMVCAop</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<!-- /WEB-INF/springmvc-servelet.xml或者不需要配置，但名称为dipatcher-servlet.xml -->

<param-value>classpath:springmvc-servelet.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

<async-supported>true</async-supported>

</servlet>

<servlet-mapping>

<servlet-name>SpringMVCAop</servlet-name>

<!-- 此处可以可以配置成\*.do等，对应struts的后缀习惯 -->

<url-pattern>\*.do</url-pattern>

</servlet-mapping>

<!-- DWR实时推送 dwr配置在springmvc中，从Spring MVC servlet进入，同时在springmvc配置文件中配置了处理相应请求的类-->

<servlet-mapping>

<servlet-name>SpringMVCAop</servlet-name>

<async-supported>true</async-supported>

<url-pattern>/dwr/\*</url-pattern>

</servlet-mapping>

</web-app>

### 在pringmvc-servelet.xml（springmvc配置文件）中加入dwr相关配置

在web.xml设置了dwr的相关请求从springmvc进入，则需要在springmvc配置文件中添加dwr相关配置来处理与dwr相关的请求。

<!-- DWR 配置 -->

<dwr:configuration />

<!-- 启用DWR 扫描，在类路径中检测 @RemoteProxy与@RemoteMethod注解的类 -->

<dwr:annotation-config id=*"dwr"*/>

<!-- 实体类转换@DataTransferObject -->

<dwr:annotation-scan base-package=*"com.cctsoft"* scanDataTransferObject=*"true"* scanRemoteProxy=*"true"*/>

<!-- 过滤请求，包含dwr的请求由dwrController处理 -->

<bean class=*"org.springframework.web.servlet.handler.SimpleUrlHandlerMapping"*>

<property name=*"alwaysUseFullPath"* value=*"true"*/>

<property name=*"mappings"*>

<props>

<prop key=*"/dwr/\*\*/\*"*>dwrController</prop>

</props>

</property>

</bean>

<dwr:controller id=*"dwrController"* debug=*"true"*>

<dwr:config-param name=*"allowScriptTagRemoting"* value=*"true"* />

<!-- 开启Reverse Ajax -->

<dwr:config-param name=*"activeReverseAjaxEnabled"* value=*"true"* />

<dwr:config-param name=*"classes"* value=*"java.lang.Object"*/>

<!-- 不允许跨域请求 -->

<dwr:config-param name=*"crossDomainSessionSecurity"* value=*"false"* />

<!-- 通知DWR 在应用程序启动时初始化 ReverseAjaxTracker -->

<dwr:config-param name=*"initApplicationScopeCreatorsAtStartup"* value=*"true"* />

<!-- ScriptSession管理 -->

<dwr:config-param name=*"org.directwebremoting.extend.ScriptSessionManager"* value=*"com.cctsoft.dwrtest.utils.pushmsg.ScriptSessionManager"* />

</dwr:controller>

### 维护ScriptSession

ScriptSession管理类：

package com.cctsoft.dwrtest.utils.pushmsg;

import java.util.Collection;

import org.directwebremoting.ScriptSession;

import org.directwebremoting.impl.DefaultScriptSessionManager;

public class ScriptSessionManager extends DefaultScriptSessionManager{

public ScriptSessionManager() {

this.addScriptSessionListener(new MyScriptSessionListener());

}

@Override

public Collection<ScriptSession> getAllScriptSessions() {

return MyScriptSessionListener.getAllScriptSessions();

}

}

侦听ScriptSession创建与销毁类：

**package** com.cctsoft.dwrtest.utils.pushmsg;

**import** java.util.ArrayList;

**import** java.util.Collection;

**import** java.util.HashMap;

**import** java.util.List;

**import** java.util.Map;

**import** org.apache.commons.lang.ObjectUtils;

**import** org.directwebremoting.ScriptSession;

**import** org.directwebremoting.event.ScriptSessionEvent;

**import** org.directwebremoting.event.ScriptSessionListener;

/\*\*

\* 重新实现ScriptSessionListener

\*

\*/

**public** **class** MyScriptSessionListener **implements** ScriptSessionListener{

//维护一个Map key为type@userid（type可以标识数据类型，userid为用户标识）， value为ScriptSession对象

**public** **static** **final** Map<String, ScriptSession> ***scriptSessionMap*** = **new** HashMap<String, ScriptSession>();

/\*\*

\* 创建Session

\* (non-Javadoc)

\* **@param** ev

\* **@see** org.directwebremoting.event.ScriptSessionListener#sessionCreated(org.directwebremoting.event.ScriptSessionEvent)

\*/

@Override

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

//PushMessageCompont类负责创建

}

/\*\*

\* 销毁Session

\* (non-Javadoc)

\* **@param** ev

\* **@see** org.directwebremoting.event.ScriptSessionListener#sessionDestroyed(org.directwebremoting.event.ScriptSessionEvent)

\*/

@Override

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

Object userId = ev.getSession().getAttribute(PushMessageUtil.***DEFAULT\_MARK***);

**if**(userId == **null**) {

}**else** {

@SuppressWarnings("unused")

ScriptSession scriptSession = ***scriptSessionMap***.remove(ObjectUtils.*toString*(userId));

}

}

/\*\*\*

\* 获取所有的ScriptSession

\* **@Title**: getAllScriptSessions

\* **@return**

\*/

**public** **static** Collection<ScriptSession> getAllScriptSessions() {

**return** ***scriptSessionMap***.values();

}

/\*\*

\* 获取所有的当前登录用户ID

\* **@Title**: getAllScriptSessionIds

\* **@return**

\*/

**public** **static** List<String> getAllScriptSessionIds() {

List<String> allUserIds = **new** ArrayList<String>(***scriptSessionMap***.keySet());

**return** allUserIds;

}

}

### 开放java类及方法供javascript调用

在PushMessageCompont类中，可以定义一些方法给javascript调用，用于告知服务器用户正在等待消息，需要服务器将实时的消息往页面推送，于此同时需要维护用户队列（即ScriptSession）。

需要使用@RemoteProxy注解相关的类，及使用@RemoteMethod来注解可供javascript访问的方法。

package com.cctsoft.dwrtest.utils.pushmsg;

import org.directwebremoting.ScriptSession;

import org.directwebremoting.WebContextFactory;

import org.directwebremoting.annotations.RemoteMethod;

import org.directwebremoting.annotations.RemoteProxy;

import org.directwebremoting.spring.SpringCreator;

@RemoteProxy(name="pushMessageCompont",creator=SpringCreator.class)

public class PushMessageCompont {

@RemoteMethod

public void onPageLoad(final String userId) {

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

scriptSession.setAttribute(PushMessageUtil.DEFAULT\_MARK, userId);

//保存在map中

MyScriptSessionListener.scriptSessionMap.put(userId, scriptSession);

}

}

### 推送消息工具

如果后台产生或收到消息，需要调用方法把消息推送给页面。

package com.cctsoft.dwrtest.utils.pushmsg;

import java.util.Collection;

import java.util.List;

import javax.servlet.http.HttpServletResponse;

import org.apache.commons.collections.CollectionUtils;

import org.apache.commons.lang.StringUtils;

import org.directwebremoting.Browser;

import org.directwebremoting.ScriptBuffer;

import org.directwebremoting.ScriptSession;

import org.directwebremoting.ScriptSessionFilter;

/\*\*

\* 消息推送.

\* @Description:

\* @ClassName: PushMessageUtil

\*/

public class PushMessageUtil{

// ScriptSession中用户的身份标识字段

public static final String DEFAULT\_MARK = "userId";

//调用javascript的方法，页面必须存在该方法来接收信息

public static final String DEFAULT\_METHOD\_NAME = "showMessage";

/\*\*

\* 信息回复

\* @Title: writeToscript

\* @param response

\* @param msg

\* @param username

\* @throws Exception

\*/

public void writeToscript(HttpServletResponse response,String msg,String username) throws Exception {

writeScript(response, "<script language='javascript'>parent.replyMsg('"+msg+"','"+username+"')</script>");

}

/\*\*

\* 写回浏览器

\* @param response

\* @param pw

\* @param script

\* @throws Exception

\*/

private void writeScript(javax.servlet.http.HttpServletResponse response,String script) throws Exception {

response.setContentType("text/html;charset=utf-8");

response.setCharacterEncoding("utf-8");

response.getWriter().write(script);

response.flushBuffer();

}

/\*\*

\* 向指定人推送消息

\* @Title: sendMessageToOne

\* @param userId

\* @param sendMessage 发送消息

\* @param scriptMethodName 方法名称

\*/

public static void sendMessageToOne(String userId,String sendMessage) {

sendMessageOneToOne(userId, DEFAULT\_METHOD\_NAME, sendMessage);

}

/\*\*

\* 向指定人推送消息

\* @Title: sendMessageToOne

\* @param userId

\* @param sendMessage 发送消息

\* @param scriptMethodName 方法名称

\*/

public static void sendMessageToOneCallBack(String userId,String sendMessage,String callBackName) {

sendMessageOneToOne(userId, DEFAULT\_METHOD\_NAME, sendMessage,callBackName);

}

/\*\*

\* 向指定人推送消息

\* @Title: sendMessageToOne

\* @param userId

\* @param sendMessage 发送消息

\* @param scriptMethodName 方法名称

\*/

public static void sendMessageToOne(String userId,String scriptMethodName,String sendMessage) {

sendMessageOneToOne(userId, scriptMethodName, sendMessage);

}

/\*\*

\* 向指定人推送消息 并调用回调

\* @Title: sendMessageToOne

\* @param userId

\* @param sendMessage 发送消息

\* @param callBackName 回调函数名称

\* @param scriptMethodName 方法名称

\*/

public static void sendMessageToOneCallBack(String userId,String scriptMethodName,String sendMessage,String callBackName) {

sendMessageOneToOne(userId, scriptMethodName, sendMessage, callBackName);

}

/\*\*

\* 向所有的人推送消息

\* @Title: sendMessageToAll

\* @param sendMessage 发送消息

\* @param methodName 方法名称

\*/

public static void sendMessageToAll(String methodName,String sendMessage) {

List<String> scriptSessions = MyScriptSessionListener.getAllScriptSessionIds();

sendMessageToAll(scriptSessions, methodName,sendMessage);

}

/\*\*

\* 向所有的人推送消息 回调函数

\* @Title: sendMessageToAllCallBack

\* @param sendMessage 发送的消息

\* @param callBackName 回调方法名称

\*/

public static void sendMessageToAllCallBack(String sendMessage,String callBackName) {

List<String> userIds = MyScriptSessionListener.getAllScriptSessionIds();

sendMessageToAll(userIds, DEFAULT\_METHOD\_NAME,sendMessage,callBackName);

}

/\*\*

\* 向所有的人推送消息 回调函数

\* @Title: sendMessageToAllCallBack

\* @param methodName 调用的方法名称

\* @param sendMessage 发送的消息

\* @param callBackName 回调方法名称

\*/

public static void sendMessageToAllCallBack(String methodName,String sendMessage,String callBackName) {

List<String> userIds = MyScriptSessionListener.getAllScriptSessionIds();

sendMessageToAll(userIds, methodName,sendMessage,callBackName);

}

/\*\*

\* 向所有的人推送消息 除了自己

\* 注意: 主要是由后台管理人员推送消息

\* @Title: sendMessageToAll

\* @param sendMessage 发送消息

\* @param methodName 方法名称

\*/

private static void sendMessageToAllNotContainCallBack(String methodName,Object ...sendMessage) {

String currentId = "11";

List<String> scriptSessions = MyScriptSessionListener.getAllScriptSessionIds();

scriptSessions.remove(currentId);

sendMessageToAll(scriptSessions, methodName,sendMessage);

}

/\*\*

\* 向所有的人推送消息 除了自己 回调

\* 注意: 主要是由后台管理人员推送消息

\* @Title: sendMessageToAll

\* @param sendMessage 发送消息

\* @param callBackName 回调方法名称

\*/

public static void sendMessageToAllNotContainCallBack(String sendMessage,String callBackName) {

sendMessageToAllNotContainCallBack(DEFAULT\_METHOD\_NAME,sendMessage,callBackName);

}

/\*\*

\* 向所有的人推送消息 除了自己

\* 注意: 主要是由后台管理人员推送消息

\* @Title: sendMessageToAll

\* @param sendMessage 发送消息

\*/

public static void sendMessageToAllNotContain(String sendMessage) {

sendMessageToAllNotContainCallBack(DEFAULT\_METHOD\_NAME,sendMessage);

}

/\*\*

\* 向所有的人推送消息 除了自己

\* 注意: 主要是由后台管理人员推送消息

\* @Title: sendMessageToAll

\* @param sendMessage 发送消息

\* @param methodName 方法名称

\*/

public static void sendMessageToAllNotContain(String methodName,String sendMessage) {

sendMessageToAllNotContainCallBack(methodName,sendMessage);

}

/\*\*

\* 向所有的人推送消息 除了指定人

\* @Title: sendMessageToAllNotContain

\* @param notContain 不包含

\* @param sendMessage 发送消息

\* @param methodName 前台接受方法名称

\*/

private static void sendMessageToAllNotContains(List<String> notContain,String methodName,Object ...sendMessage) {

List<String> scriptSessions = MyScriptSessionListener.getAllScriptSessionIds();

if(CollectionUtils.isNotEmpty(notContain)) {

scriptSessions.removeAll(notContain);

}

sendMessageToAll(scriptSessions, methodName,sendMessage);

}

/\*\*

\* 向所有的人推送消息 除了指定人

\* @Title: sendMessageToAllNotContain

\* @param notContain 不包含

\* @param sendMessage 发送消息

\*/

public static void sendMessageToAllNotContain(List<String> notContain,String sendMessage) {

sendMessageToAllNotContains(notContain, DEFAULT\_METHOD\_NAME,sendMessage);

}

/\*\*

\* 向所有的人推送消息 除了指定人

\* @Title: sendMessageToAllNotContain

\* @param notContain 不包含

\* @param sendMessage 发送消息

\* @param methodName 前台接受方法名称

\*/

public static void sendMessageToAllNotContain(List<String> notContain,String methodName,String sendMessage ) {

sendMessageToAllNotContains(notContain, methodName,sendMessage);

}

/\*\*

\* 向所有的人推送消息 除了指定人 并回调

\* @Title: sendMessageToAllNotContain

\* @param notContain 不包含

\* @param sendMessage 发送消息

\* @param callBackName 回调名称

\*/

public static void sendMessageToAllNotContainCallBack(List<String> notContain,String sendMessage,String callBackName) {

sendMessageToAllNotContains(notContain, DEFAULT\_METHOD\_NAME,sendMessage,callBackName);

}

/\*\*

\* 向所有的人推送消息 除了指定人 并回调

\* @Title: sendMessageToAllNotContain

\* @param notContain 不包含

\* @param sendMessage 发送消息

\* @param callBackName 回调名称

\* \* @param methodName 前台接受方法名称

\*/

public static void sendMessageToAllNotContainCallBack(List<String> notContain,String methodName,String sendMessage,String callBackName) {

sendMessageToAllNotContains(notContain, methodName,sendMessage,callBackName);

}

/\*\*

\* 向所有的人推送消息

\* @Title: sendMessageToAll

\* @param sendMessage 发送消息

\*/

public static void sendMessageToAll(String sendMessage) {

sendMessageToAll(DEFAULT\_METHOD\_NAME,sendMessage);

}

/\*\*

\* 向多个用户推送消息

\* @Title: sendMessageToAll

\* @param sendMessage 发送消息

\* @param methodName 前台页面方法名称

\*/

public static void sendMessageToMul(List<String> userIds,String methodName,String sendMessage) {

sendMessageToAll(userIds,methodName,sendMessage);

}

/\*\*

\* 向多个用户推送消息 并且回调

\* @Title: sendMessageToAll

\* @param sendMessage 发送消息

\* @param methodName 前台页面方法名称

\* @param callBackName 回调函数名称

\*/

public static void sendMessageToAllCallBack(List<String> userIds,String sendMessage,String callBack) {

sendMessageToAll(userIds,DEFAULT\_METHOD\_NAME,sendMessage,callBack);

}

/\*\*

\* 向多个用户推送消息

\* @Title: sendMessageToAll

\* @param sendMessage

\*/

public static void sendMessageToAll(List<String> userIds,String sendMessage) {

sendMessageToAll(userIds,DEFAULT\_METHOD\_NAME,sendMessage);

}

/\*\*

\* 精确推送

\* @Title: sendMessageToOne

\* @param mark

\* @param sendMessage

\*/

private static void sendMessageOneToOne(String userId,String scriptMethodName, Object ...sendMessage) {

final String tempUserId = userId;

final Object[] tempSendMessage = sendMessage;

final String tempScriptMethodName = scriptMethodName;

Browser.withAllSessionsFiltered(new ScriptSessionFilter() {

@Override

public boolean match(ScriptSession session) {

String getSessionUserId = (String) session.getAttribute(DEFAULT\_MARK);

if(StringUtils.isNotBlank(getSessionUserId)) {

return StringUtils.equals(getSessionUserId, tempUserId);

}

return false;

}

}, new Runnable() {

private ScriptBuffer scriptBuffer = new ScriptBuffer();

@Override

public void run() {

appendCurrentSession(scriptBuffer, tempScriptMethodName,tempSendMessage);

}

});

}

/\*\*

\* 向指定的多个用户推送消息

\* @Title: sendMessageToAll

\* @param sendMessage

\*/

private static void sendMessageToAll(List<String> userIds,String scriptMethodName,Object ...sendMessage) {

final List<String> tempUserIds = userIds;

final Object[] tempSendMessage = sendMessage;

final String tempScriptMethodName = scriptMethodName;

Browser.withAllSessionsFiltered(new ScriptSessionFilter() {

@Override

public boolean match(ScriptSession session) {

String getSessionUserId = (String) session.getAttribute(DEFAULT\_MARK);

if(StringUtils.isNotBlank(getSessionUserId) && CollectionUtils.isNotEmpty(tempUserIds) && tempUserIds.contains(getSessionUserId)) {

tempUserIds.remove(getSessionUserId);

return true;

}

return false;

}

}, new Runnable() {

private ScriptBuffer scriptBuffer = new ScriptBuffer();

@Override

public void run() {

appendCurrentSession(scriptBuffer,tempScriptMethodName,tempSendMessage);

}

});

}

/\*\*

\* 循环获取当前已登录的Session

\* @Title: appendCurrentSession

\* @param scriptBuffer

\* @param sendMessage

\* @param scriptMethodName

\*/

private static void appendCurrentSession(ScriptBuffer scriptBuffer, String scriptMethodName, Object ...sendMessage) {

scriptBuffer.appendCall(scriptMethodName, sendMessage);

Collection<ScriptSession> scriptSessions = Browser.getTargetSessions();//MyScriptSessionListener.getAllScriptSessions();

for(ScriptSession scriptSession : scriptSessions) {

scriptSession.addScript(scriptBuffer);

}

}

### 实时推送消息给页面

后台模拟消息产生并发送：

**package** com.cctsoft.dwrtest.service;

**import** java.text.SimpleDateFormat;

**import** java.util.Date;

**import** java.util.List;

**import** java.util.Timer;

**import** java.util.TimerTask;

**import** org.springframework.stereotype.Service;

**import** com.cctsoft.dwrtest.utils.pushmsg.MyScriptSessionListener;

**import** com.cctsoft.dwrtest.utils.pushmsg.PushMessageUtil;

@Service(value="systemMsgService")

**public** **class** SystemMsgService {

**static** {

SimpleDateFormat format = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

**try** {

Timer timer = **new** Timer();

timer.schedule(**new** TimerTask() {

**public** **void** run() {

//获取在线用户

System.***out***.println("=======开始发送消息==========");

List<String> allScriptSessionIds = MyScriptSessionListener.*getAllScriptSessionIds*();

**for**(String userId:allScriptSessionIds){

System.***out***.println(userId + "======在线用户");

//过滤，只推送给system@类型的用户

**if**(userId != **null** && userId.trim().length() >= 7 && "system@".equals(userId.substring(0, 7))){

System.***out***.println("======发送=====" + userId);

PushMessageUtil.*sendMessageToOne*(userId, format.format(**new** Date()));

}

}

}

}, 10000, 1000);

} **catch** (Exception e) {

e.printStackTrace();

}

}

}

页面：

<!DOCTYPE html>

<html>

<head>

<meta charset=*"utf-8"*>

<title>测试实例</title>

<script type=*"text/javascript"* src=*"dwr/engine.js"*></script>

<script type=*"text/javascript"* src=*"dwr/util.js"*></script>

<script type=*"text/javascript"* src=*"dwr/interface/pushMessageCompont.js"*></script>

</head>

<body>

<div id=*"app"*>

<label id=*"labelId1"*></label>

</div>

<script>

//加载dwr组件

onPageLoad();

dwr.engine.setActiveReverseAjax(**true**);

dwr.engine.setNotifyServerOnPageUnload(**true**, **true**);

**function** onPageLoad() {

pushMessageCompont.onPageLoad("system@123456");

}

**function** showMessage(sendMessages, clickEvent) {

document.getElementById('labelId1').innerHTML = "当前时间：" + sendMessages;;

}

//重写错误方法

dwr.engine.\_errorHandler = **function**(message, ex) {

};

</script>

</body>

</html>