文章编号:1001 - 9081(2009)07 - 1764 - 03

基于 JSF 定制生命周期的 Ajax 组件开发

彭 胜[1,2],刘卫国[1]
(1. 中南大学 信息科学与工程学院,长沙 410083; 2. 吉首大学 数学与计算机科学学院,湖南 吉首 416000)
(pengjiaying@163.com)

摘 要:为提高 JSF 组件的客户端交互能力,在分析 JSF 定制生命周期实现机制的基础上,提出了一个用于开发 Ajax 组件的支持框架。该框架通过设计 AjaxEvent 类和定制相关的生命周期对象,使 Ajax 请求的处理过程能像普通的 Faces 请求那样封装在 JSF 组件内部。实际应用表明,该框架能充分利用 JSF 和 Ajax 各自的优点,高效快捷地开发具有良好交互能力的 Web 用户界面。

关键词:JSF;Ajax;定制生命周期;组件
中图分类号: TP311 **文献标志码**:A

Development of Ajax component based on JSF custom lifecycle

PENG Sheng[1,2], LIU Wei-guo[1]
(1. *School of Information Science and Engineering, Central South University, Changsha Hunan* 410083, *China*;
2. *School of Mathematics and Computer Science, Jishou University, Jishou Hunan* 416000, *China*)

Abstract: In order to improve the clients' interactivity of the JSF component, a framework was presented to develop Ajax component on the basis of analyzing the implement mechanism of the custom lifecycle in JSF. By designing the Ajax event class and introducing the related custom lifecycle object in this framework, the processing of the Ajax event could be encapsulated inside the JSF component, just like the normal Faces event. The practical applications indicate that this framework can take the respective advantage of JSF and Ajax and Web user interface can be developed to be more rapid and efficient.

Key words: JSF; Ajax; custom lifecycle; component

0 引言

JSF 作为面向 J2EE 的新一代的标准 Web 界面开发框架,是一个以组件为中心且采用事件驱动机制的开发框架,页面事件由组件发出,事件处理方法由组件指定,整个页面由组件构成,因而组件开发是 JSF 构造 Web 界面的基础性环节。然而,受当前浏览器技术的限制,JSF 组件普遍存在着交互能力不强的问题,而 Ajax 技术作为提升客户交互能力方面的成熟技术,也存在着解决问题的层面较低,单独使用会遇到诸多不便等缺点。因此,如何使两者有机结合,是大家普遍关注的课题[1-3]。

在 JSF 中,用户请求的处理具体是由请求处理生命周期对象(以下简称生命周期对象)来负责的,JSF1.2 专家组成员 Jacob Hookon 引入了一种新方法,能够针对不同 Servlet 映射配置不同的 lifecycleId,使得同一应用程序的不同类型的请求可以由不同的生命周期对象去处理。Jacob 的想法是让 init-param 元素还包含 lifecycleId,从而就可以为处理 Ajax 请求而专门定制生命周期来实现[4]349。本文通过分析 JSF 定制生命周期的具体实现机制,在 J2EE5 自带的 JSF 实现基础之上,设计并实现了一个专门用于开发具备 Ajax 功能的 JSF 组件(以下简称 Ajax 组件)的支持框架,可以在开发 Ajax 组件过程中更多地利用到 JSF 提供的各种便利条件,有效地提高 Ajax 组件的开发效率。

1 JSF 生命周期及其事件处理机制

1.1 JSF 应用程序结构

JSF 是一个以组件为中心,建立在 Servlet 基础之上,体现 MVC 思想的开发框架,它提供了对 Model、View、Controller 的明确定义[5-6]:

1) Model。模型部分代表着业务逻辑和数据集合。由托管 Bean 工具来负责配置管理。

2) View。视图部分的基本组成单位是用户界面组件,它是一个横跨了客户端和服务器端的用户可配置的综合实体。

3) Controller。主要包括 FacesServlet、相关配置文件和动作处理器。是系统中视图部分与模型部分联系的中介,同时负责维护视图组件服务器端与客户端内容的一致性。

1.2 请求处理生命周期

面对日益复杂的 Web 页面请求,前端控制器 FacesServlet 的功能发生了分化,现在只负责请求的转发,而真正的处理工作则交由请求生命周期来负责。针对请求处理的一般过程,请求处理生命周期被划分为 6 个阶段:新建或恢复视图、应用请求值 、处理验证、更新模型值、调用应用程序、呈现响应。每个阶段完成特定的处理工作,配合适当的事件处理机制,可以较大降低请求处理的工作难度,有效地提高系统开发效率[4,7-8]。其工作过程如图 1 所示,图中①~⑥分别对应于 6 个处理阶段。

从图 1 可以看出,在整个请求处理生命周期工作过程中存

收稿日期:2008 - 12 - 22;**修回日期**:2009 - 02 - 27。

作者简介:彭胜(1974 -),男(土家族),湖北潜江人,讲师,硕士研究生,主要研究方向:计算机网络应用、信息系统集成; 刘卫国(1963 -),男,湖南祁阳人,教授,博士,主要研究方向:网络信息安全、数据库技术。

在两种数据流动循环:第 1 阶段与第 4 阶段之间的循环体现了视图部分与模型部分之间的交互,第 2 阶段与第 6 阶段之间的循环体现了视图部分客户端与服务器端之间内容同步。

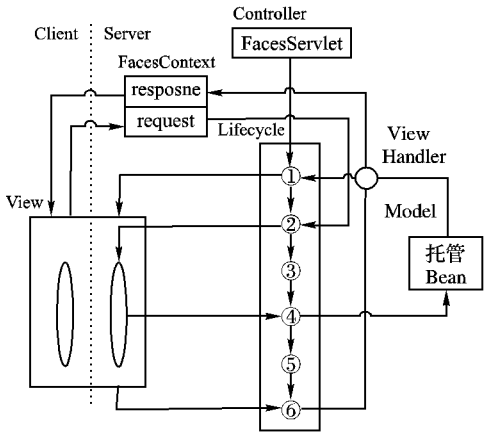

图 1　请求处理生命周期工作过程

1.3　Faces 事件响应及处理实现机制

JSF 中用户界面组件发出的事件称为 Faces 事件[4]135-141,包括动作事件和值改变事件,两者响应和处理机制是一致的。首先通过组件的标签处理器将监听器注册到组件上,当页面事件发生时,组件的客户端部分捕获事件并提交给服务器,然后服务器记录下该事件,具体就是创建 FacesEvent 对象并进入 Faces 事件队列等待处理,该对象的主要作用就是记录下事件来源对象和事件监听器宿主对象,最后在事件处理时便可以通过 FaceEvent 对象的 getComponent 方法获取事件监听器宿主对象,从而进一步找到并运行相应的事件处理代码。其一般的响应和处理实现机制如图 2 所示。

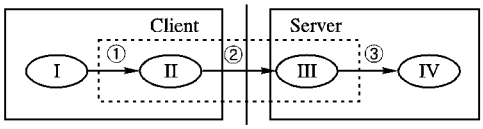

图 2　Faces 事件响应和处理实现机制

图 2 中,I 至 IV 分别代表 JavaScript 事件源、事件捕获及提交代码、事件记录代码和事件处理代码。①至③则为各部分间传递的数据,分别是:JavaScript 事件对象、Http 请求数据和 FacesEvent 对象。其中 II 和 III 这两部分要封装在同一个组件中,并且在组织 Http 参数时要带上该组件的 clientId,以方便服务器定位记录事件代码。

1.4　定制生命周期注册方法

所谓定制生命周期,是指通过自定义生命周期,让特定的 Faces 按照用户指定的请求处理路径来处理。要做到这一点,除了提供特定的生命周期实现外,还必须做好以下工作[4]348-349:

1) 提供新的生命周期工厂类,让其事先注册好定制的生命周期对象,并与特定的 lifecycleId 值相关联,然后在 faces-config.xml 中注册该工厂以替代默认值。

2) 在 Servlet 映射配置的 init-param 元素中设置好相关的 lifecycleId 值,以便于当不同 URL 模式的页面请求到来时,前端控制器能根据其值自动加以区分,从而委托不同的生命周期对象去处理。

2　Ajax 组件支持框架

本文提出一个支持框架,可以方便地开发同时支持 Faces 事件和 Ajax 事件的用户界面组件(即 Ajax 组件),其基本工作原理如图 3 所示。

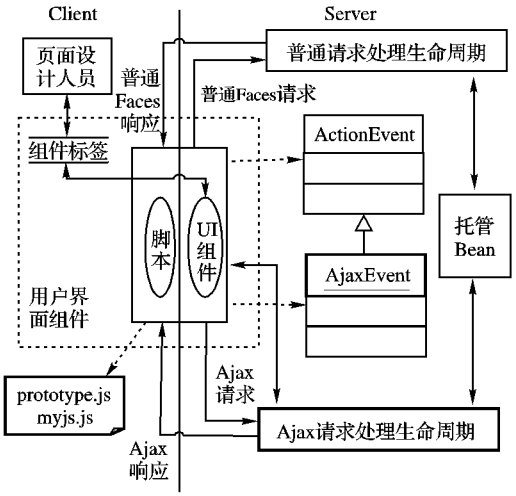

图 3　Ajax 组件工作原理

在图 3 中,虚线框部分是组件的基本内容。而粗线框部分则属于本支持框架的组成要素,大致上包括 3 个部分:第一部分提供定制的 Ajax 请求处理生命周期,为使其发挥作用,须提供相关的工厂类并做好配置;第二部分提供一个自定义的 JavaScript 函数库,方便组件客户端提交 Ajax 请求以及接收服务器响应;第三部分设计一个新类 AjaxEvent,便于系统在服务器端记录 Ajax 事件,该类继承于 ActionEvent,但其事件源属性和事件监听器宿主属性可以不是同一个组件。为提高开发效率,在框架实现上,大量采用继承等机制,尽可能重用 JSF 实现中的已有代码。

2.1　Ajax 请求生命周期支持模块

1)定制 Ajax 请求处理生命周期类。

首先继承默认生命周期实现类,然后用反射技术将默认的呈现响应阶段类动态替换成定制的呈现响应阶段类,其余阶段类则可重用 JSF 实现已有代码。

2) Ajax 呈现响应类。

继承默认呈现响应阶段类,然后覆盖其中的 execute 方法:首先从 RequestMap 中获取刷新元素 ID 列表,然后依次找到其对应的 UI 组件,利用其 encode 系列函数将每个页面元素要呈现的内容以合适形式加以组织呈现给客户端,这里选择 XML 方式。

3)响应格式的组织。

为了方便客户端脚本的分析和处理,将 Ajax 呈现响应内容按以下格式封装:

```
<?xml version = "1.0" encoding = "utf-8"? >
<items >
<item >
<name > 页面元素 ID </name >
<content > 页面元素内容 </content >
</item >
...
</items >
```

4)定制生命周期工厂类。

利用装饰模式将默认生命工厂类包装起来,再在其带参构造函数中将 Ajax 请求处理生命周期对象注册并让其与特定的 lifecycleId 值相关联。然后,将普通 JSF 请求和 Ajax 请求都映射到前端控制器 FacesServlet(显然,两者映射 URL 模式要不同),并且在 Ajax 请求映射配置的 init-param 元素添设 Ajax 定制生命周期对象对应的 lifecycleId 配置,这样,

FacesServlet 就可自动区分两类不同类型的请求而委托不同的生命周期对象去处理。

2.2 自定义的函数库 myjs.js

这里主要提供了两个 JavaScript 函数:一个主要负责提交页面请求;另一个则负责接收服务器响应,分析处理后完成页面的局部刷新。为提高开发效率,同时避免因浏览器不兼容所引起的麻烦,在编码时利用了 Prototype 框架提供的一些函数[9],因此要事先引入其函数库文件 prototype.js。

1)请求提交函数 send(url,params)。

该函数的主要作用是提交客户端的事件请求。其中,参数 url 要与 Ajax 请求映射中的设置相匹配,否则前端控制器 FacesServlet 无法找到 Ajax 请求处理生命周期对象;第二个参数 params 则包含了为确保请求在服务器端正确处理而必须要提供的一些信息,具体有:组件 clientId(方便服务器定位事件记录代码),刷新元素 ID 列表(局部呈现响应类该利用它组织响应内容,这需要 UI 组件从标签处理器中获取)以及目标组件所在表单参数(无此项信息,则会被 JSF 认为是非回传请求,也就不会沿着正确的轨道去处理该请求了[4]294)。很显然,这些参数的具体值均与组件使用环境有关,因此需要在开发组件时动态注入并运行获取这些参数的代码。

该函数的功能实现上主要是利用 Prototype 提供的 Ajax.Request 类来完成请求的提交,使用时,只要创建好一个 Ajax.Request 对象,即已完成一次 Ajax 交互。其基本的使用方式类似于:

```
var xhr = new Ajax.Request(url, {...params, ..., onResp,...});
```

2)响应分析函数 onResp(xhr)。

该函数的主要作用是从服务器端获取 XML 形式的响应内容,然后进行解析,针对其中每一个 item 元素,分别析取对应的页面元素的 ID 值和页面元素内容后,利用 Prototype 的 Element.replace()方法来实现各个元素的局部刷新。该函数要跟前面的提交函数配合使用,其形式参数 xhr 即代表前面函数中创建的 Ajax.Request 对象。

2.3 AjaxEvent 类的设计与实现

该类的主要作用是记录事件的来源以及事件监听器的宿主组件。这里具体参照 ActionEvent 类的实现(有时也可以考虑 ValueChangeEvent),先继承 ActionEvent,然后在构造函数注入事件来源组件和事件监听器宿主组件:

```
public AjaxEvent(sourceComp, hostComp){
  super(sourceComp);
  this.hostComponent = hostComp;
  }
```

然后覆盖 ActionEvent 类的 getComponent 方法,让其返回事件监听器宿主组件,以便系统在事件处理时顺利找到事件监听器:

```
public UIComponent getComponent( ){
  return hostComponent;
  }
```

3 开发实例

在前面介绍的框架基础上,开发一个 Ajax 使能组件,该组件能够方便地给其他普通组件添加 Ajax 事件支持。

根据 JSF 规范,一个组件主要由 3 个部分组成:标签及标签处理器,负责传递网页开发人员意图,并且在标签处理时注册事件监听器;UI 组件类,代表组件的核心行为,主要包括负责客户端资源的呈现以及接收客户的请求并记录;客户端资源,包括页面显示元素及相关 JavaScript 代码,这些代码负责客户端相关事件的捕获、提交,对 Ajax 组件而言还得负责服务器端响应的解析处理工作。

3.1 标签及标签处理器

根据需要,该组件标签设置了 3 个值表达式属性和 1 个方法表达式属性。其中,值表达式属性 source、event 和 targetIds 分别代表目标组件(事件源)、事件名称和事件处理后需要刷新的元素列表,方法表达式属性 ajaxListener 则用于指定事件监听方法。

然后,在标签处理器类的 setProperties 方法中将它们的值传递给对应的 UI 组件。其中值表达式利用 UI 组件的 setValueExpression 方法即可。而对于方法表达式,则需要先要创建监听器对象,然后将其注册到对应组件中去:

```
MethodExpressionActionListener listener =
  new MethodExpressionActionListener(ajaxListener);
  ((AjaxAction)comp).addActionListener(listener);
```

这样,在事件处理时,只要找到该 UI 组件,也就能找到该动作监听器对象。

3.2 UI 组件类 AjaxAction

UI 组件类首先继承 UICommand 类,然后分别覆盖其 encodeBegin 方法和 decode 方法。

1)encodeBegin 方法负责呈现资源:先从当前 FacesContext 对象中获取 ResponseWriter 对象,再利用它将组件客户端的资源呈现给页面,这种方式能使资源随组件的引用而加载。当然,也要注意防止资源重复加载,否则会引起客户端错误[4]260。

2)decode 方法主要负责接收从本组件客户端发来的事件请求并记录该事件,记录的方法是通过创建 AjaxEvent 对象来记录该事件的事件源和事件监听器宿主,前者要根据标签传过来的值获取,而后者就是 AjaxAction 自身:

```
AjaxEvent aEvent = new AjaxEvent(sourceComp, this);
```

事件对象创建好后将其放入事件队列中去,等待系统在合适时侯去处理:

```
this.queueEvent(aEvent);
```

3.3 客户端资源

这里包括静态资源和动态资源,静态资源即前面介绍的两个脚本函数库:prototype.js 和 myjs.js,动态资源负责特定事件的监听、准备相关 URL 参数后提交请求,这些内容与组件使用环境有关,因此必须动态获取。

1)准备事件监听函数名。

由于该函数与组件使用环境有关,函数名每次使用时都要有所不同,这里将其组件被引用时的 clientId 信息添加到函数名中:

```
String CompS = this.getClientId(context).replace(":", "_");
mycode += "function send" + CompS + "Req( ) {";
```

2)准备上传参数。

```
mycode += " var uri = " + getAjaxURI(context) + "; ";
mycode += " var params = " + getAjaxParams(context) + ";";
```

3)注入提交代码。

注入提交代码,完成动态监听函数构造。

```
myCode += "send(uri, params);}";
```

4)注入监听的执行代码。

事件监听函数构造好后,利用 Prototype 提供的 Event.observe()方法去给监听目标组件的特定事件指定监听函数。

(下转第 1770 页)

通过分析表1和图4,可以看出,随着构件数量的增加,构件检索方法R3在检索效率上优于方法R1和R2,这种优势在构件库规模大时更明显,更适应于大规模构件库的检索。

表1 三种构件检索方法的执行时间 ms

构件数量	R1执行时间	R2执行时间	R3执行时间
20	139	118	252
50	171	147	284
100	192	162	293
150	263	230	308
200	385	350	352
300	820	692	423
400	1135	1064	497
500	1728	1638	564

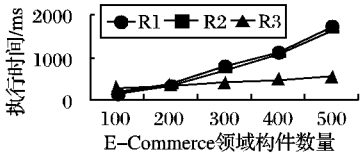

图4 三种构件检索方法检索时间的比较

6 结语

程序挖掘是主动服务的实现机制,本文以主动服务和程序挖掘概念为基础,通过建立程序挖掘知识库、构件索引库、构件实体库、领域本体知识库和用户兴趣模型,设计了基于多智能代理运行平台的程序挖掘系统框架,利用多智能代理完成对用户需求的分析、功能提取,构件的搜索获取、组装以及验证,从而实现按需服务。为了实现程序挖掘,本文将刻面、关键字、领域本体和用户兴趣模型相结合,设计并实现了基于多知识库的构件检索方法。首先给出结合本体和特征领域模型的本体特征领域模型的定义,然后在本体特征领域模型的基础上给出构件描述模型CDM-OF,该模型既便于用户理解又可以被机器自动处理,同时建立了用户兴趣模型,在此基础上,设计了基于多知识库的构件检索框架,并给出一个检索关联构件的检索算法。

最后,通过实验,以E-commerce领域内构件资源为例,对该检索方法的有效性进行了分析,并与常用的基于关键词和刻面的检索方法相比较,验证了该检索方法在一定程度上能够提高某领域内构件检索的查全率、查准率和效率,尤其适用于大规模构件库的检索。

参考文献:

[1] 张尧学,方存好. 主动服务——概念、结构与实现[M]. 北京:科学出版社,2005:43-61.

[2] 窦郁宏. 程序挖掘中构件描述和检索的研究[D]. 长沙:中南大学,2002.

[3] XU KE-GANG, ZHANG YAO-XUE, WEI ZI-ZHONG. PMMAP: A mobile agents platform for program mining[C]// Computer Networks and Mobile Computing. Los Alamitos, CA: IEEE Computer Society, 2001:347-355.

[4] MAARTEN B, BART D W, WOUTER J, *et al*. Ontology-based discovery of data-driven services [C]// SOSE'06: Service-Oriented System Engineering. Shanghai: IEEE Computer Society, 2006:175-178.

[5] ZHONG L, NIE L. Design of program mining system framework for active service[C]// 2008 International Conference on Computer Science and Software Engineering. Wuhan: IEEE Computer Society, 2008:436-439.

[6] 陈刚,陆汝钤,金芝. 基于领域知识重用的虚拟领域本体构造[J]. 软件学报,2003,14(3):350-355.

[7] 张彦,张永奎. 基于层次概念的用户兴趣模型研究[J]. 计算机工程与设计,2008,29(1):181-183.

[8] PODGURSKI A, PIEREE L. Retrieving reusable software by sampling behavior [J]. ACM Transactions on Software Engineering and Methodology, 1993, 2(3):286-303.

[9] 王渊峰,张涌,任洪敏,等. 基于刻面描述的构件检索[J]. 软件学报,2002,13(8):1546-1551.

(上接第1766页)

```
myCode + = " Event. observe( " + getForId( context) + "," + getEvent
(context) + ", send" + curCompS + "Req, false); ";
```

这些动态代码准备好后,即可交由AjaxAction中的encodeBegin方法呈现到客户端。

3.4 组件的使用

至此,该Ajax使能组件即可投入使用。使用方式与普通的JSF定制组件类似:事先在页面顶端做好相关tld声明,然后,利用该组件对应标签在页面适当位置引用组件,并做好相关属性设置,最后在托管Bean中编写好事件处理代码。

下面以一个具体的实例来说明该组件的使用。在原始页面中,有一个Id为userName的输入组件,其值与托管Bean中的enterName属性绑定在一起,另外还有两个Id分别为info和sysmsg的标签组件,这两个标签组件的值都用el表达式设置为托管Bean的valiadUserMsg属性。

在页面中添加以下代码来为输入组件添加Ajax事件支持:

```
< ajax: action source = "userName" targetset = "info, sysmsg"
event = "blur" actionListener = "#{ userBean. validUser} "/ >
```

然后,在托管Bean的validUser方法中设置事件处理代码:

```
public void validUser( ActionEvent event) {
  this. validUserMsg = this. enterName; }
```

最后,该输入组件就具备相应的Ajax事件处理能力了。

4 结语

本文利用定制请求处理生命周期的思想,为开发支持Ajax的JSF组件提供了一个支撑框架,使得开发出来的组件可以同时支持普通Faces请求和Ajax请求。由于组件采用JSF组件封装机制封装,使用起来非常方便,同时有效避免了单独使用Ajax技术时常见的诸多麻烦和不便。实际应用表明,使用该框架开发Ajax组件,开发强度低,开发效率高,可以充分融合JSF和Ajax技术,使它们优势互补,十分适合Web用户界面的快速构建。

参考文献:

[1] 左学明,张力. 一种新的基于JSF技术的Web用户界面开发方法[J]. 计算机应用,2005,25(1):215-217.

[2] 刘高原,刘觉夫,张国平. 结合Ajax和JSF技术开发Web应用[J]. 微计算机信息,2007,23(12):252-253.

[3] 吴鹏,尹思良,张学杰. 一种基于Ajax的JSF应用改进[J]. 云南大学学报:自然科学版,2007,29(s2):244-248.

[4] SCHALK C, BURNS E. JavaServer Faces完全参考手册[M]. 张猛,译. 北京:人民邮电出版社,2007.

[5] 邝文清,郭跟成. 基于JSF框架Web应用开发的研究[J]. 计算机应用研究,2007,24(12):272-275.

[6] 王峰,江勤绕,俞欢军. 基于JSF框架的信息管理系统的设计与实现[J]. 计算机工程与设计,2007,28(21): 5211-5224.

[7] DUDOEY B, LEHR J, WILLIS B, *et al*. Mastering JavaServer™ Faces[M]. Indianapolis: John Wiley and Sons, 2004.

[8] JACOBI J, FALLOWS J R. Pro JSF and Ajax: Building Rich Internet Components[M]. New York: Springer-Verlag, 2006.

[9] 施伟伟,张蓓. 征服Ajax——Dojo、Prototype、script. aculo. us框架解析与实例[M]. 北京:人民邮电出版社,2007.