新一代Web应用框架JSF

朱庆生　葛　亮
（重庆大学计算机学院　重庆400044）

摘　要　本文介绍新一代Web应用框架JavaServer Faces(JSF)。JSF的最终标准于2004年5月颁布，它代表了Web应用技术的发展趋势。本文分析JSF与现有相关技术的关系，描述JSF的组成、处理流程和开发Web应用程序的过程，归纳JSF的主要优点。同时，针对JSF标准中渲染组件存在的不足，提出JSF与JSP、XML相结合的改进应用方案。

关键词　Web应用框架，JSF，渲染组件

JSF:A New Generation of Web Application Framework

ZHU Qing-Sheng　GE Liang
(College of Computer Science, Chongqing University,Chongqing400044)

Abstract　This paper introduces a new generation of Web application framework, which named JavaServer Faces (JSF). The final release of JSF's specification is issued in May 2004, which represents the trend of Web application technology's evolution. The paper analyzes the relation between JSF and interrelated technologies,describes the components and handling flow of JSF and the process of developing Web application with JSF and concludes the main merits of JSF. It also mends the limitations of component rendering in present JSF specification through the combination of JSF, JSP and XML.

Keywords　Web application framework,JavaServer faces,Component rendering

1　引言

由于互联网技术自身的众多优点，如今越来越多的信息化解决方案都采用B/S体系结构。早期的服务器端应用程序是通过程序生成标准的HTML页面返回给客户端，客户端再使用浏览器将它们显示给用户。随后出现的Servlet[1]/JSP[2]、Struts等技术为创建Web应用提供了一个强大的实现模型，可以更好地控制业务逻辑和表现逻辑。尽管如此，这些技术规范却没有定义相关的API来创建客户端的图形用户界面，这使得开发人员不得不手工管理页面中控件的状态并处理相关的事件，增大了Web应用开发的复杂度且容易出错，不便于项目的快速开发和实施。因此，创建一个标准的图形用户界面组件框架，简化Web应用中图形用户界面的开发，势在必行。JSF(JavaServer Faces)[3]就是在这种背景下，由Ed Burns、Craig R. McClanahan等人于2001年5月提出意向，并在2004年5月27日最终形成规范标准。

2　JSF概述

JSF是一个基于Java的Web应用框架，侧重于用户界面的创建和管理。它与现有的技术相互补充，相得益彰。Java Servlet是JSF的基础，它定义了如何用服务器端组件(servlet)来封装和实现Web应用程序；JavaServer Pages(JSP)在Servlet的基础上提供了页面模板创建文本内容(如HTML)的机制，它能与JSF很好地集成；JSP标准标签库(JSTL)则定义了一套标准的JSP操作(被称之为标签)，简化了JSP页面的开发，它能与JSF的标签库相互补充。图1描述了这几种技术之间的关系。

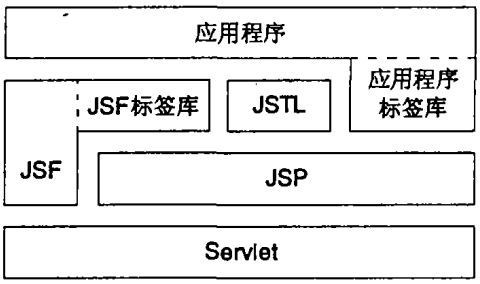

图1　JSF与现有技术之间的关系

JSF主要包括两大方面的内容：一个API集合，用来表示用户接口组件、管理组件的状态、处理事件和验证输入、定义页面间的导航以及支持国际化；一个JSP的自定义标签库，用来在JSP页面中显示JSF接口。其中API集合是JSF的核心组成部分。可以把JSF看成是Struts与Swing(基于Java的桌面应用程序的标准用户接口)的组合：JSF像Struts那样遵循MVC体系结构，通过一个控制器servlet来管理应用程序的生命周期；而像Swing那样提供了一个可扩展的组件模型，此组件模型还包含了一个重要功能——渲染组件(component rendering)。所谓渲染组件就是把组件所包含的信息转换成客户端能够接受的表达形式，比如对于Web浏览器而言就是HTML页面，而对于手机等无线移动设备所使用的WAP浏览器而言就是WML页面，每个组件都具有各自的渲染组件功能。

JSF的提出大大促进了Web应用的开发效率。其一，利用JSF所提供的组件模型，结合所定义的JSP自定义标签库，

朱庆生　教授，博士生导师，主要研究方向：多媒体技术，软件工程；葛　亮　硕士研究生，主要研究方向：电子商务，网络信息系统及软件开发环境。

可以方便地创建用户接口；其二，组件中封装了事件处理、输入验证等操作，利用 JSF 所提供的运行环境可以很容易地将客户端事件与服务器端的事件处理程序绑定；其三，JSF 所提供的框架具有很好的可扩展性，能够使开发人员实现可重用的自定义组件；其四，JSF 的提出进一步细化了开发人员的分工，在页面设计人员和代码开发人员的基础上引入了组件开发人员——他们负责创建可重用的用户接口对象，这有利于项目的团队开发，加快开发进度，此外组件开发人员可以进一步演化成第三方的组件开发提供商，专门提供高质量的组件，提高行业的整体开发水平。

3 JSF 的组成及处理流程

3.1 用户接口组件模型

JSF 将 Web 应用程序用户界面抽象成用户接口组件模型，而将用户界面中的元素抽象成可被容器管理的用户接口组件。一个用户接口组件代表了用户接口中一个可配置、可重用的元素，它既可以表示一个简单的控件（比如按钮或者文本输入框），也可以表示一个组合控件（比如一张表）。所有的用户接口组件都是 javax.faces.component.UIComponent 抽象类的子类，该抽象类通过 Java 的 API 函数定义了组件基本的状态信息和行为模式，为 JSF 管理组件提供了统一的界面。JSF 的用户接口组件模型包括以下几个方面的内容：

首先通过 UIComponent 抽象基类定义了组件的基本属性和方法。基本属性包含两个方面：一个是组件标志符、组件类型等通用属性；一个是存放与特定组件相关的基本属性（如渲染特定组件时所需的额外信息）的属性列表，该属性列表以属性名为键值进行索引，可以让特定组件很方便地存放额外的基本信息。而基本方法则主要包括：存取基本属性的方法以及跟 JSF 处理流程相关的方法（比如处理事件、渲染响应等）。

然后通过组件行为接口扩展 UIComponent 抽象类所定义的组件基本行为。这些组件行为接口有：ActionSource 接口、StateHolder 接口、ValueHolder 接口、EditableValueHolder 接口等。其中 ActionSource 接口定义了实现该接口的组件是 ActionEvent 事件的事件源，组件可以指定处理 ActionEvent 事件的事件监听器及其调用的方法，这为 JSF 与 Web 应用中的应用逻辑层集成提供了途径；StateHolder 接口提供了保存和恢复组件状态的属性和方法，如果组件需要在多个请求之间保存状态，则必须实现这个接口；而 ValueHolder 接口和 EditableValueHolder 接口则为组件提供了存取本地值的能力（ValueHolder 接口只能读取属性值），它们通过 converter 转换器类自动将请求中的属性值与组件中对应的属性值进行类型转换，此外 EditableValueHolder 接口还通过 validator 验证器类对发生改变的属性值进行验证，这就简化明晰了请求响应与 JSF 组件模型之间数据传输的操作。

最后用户接口组件模型将每个组件都看成是组件的容器，也就是说每个组件都可以包含任意多个子组件，以构成 Web 应用中所要求的页面。这种包含关系有两种形式：最为常见的是父子关系，即包含与被包含的组件之间是父与子的关系，子组件是父组件的组成部分之一，并且这种关系可以递归下去，从而形成树形结构，其树根被规定为 UIViewRoot 组件。这是 JSF 组件间关系的主要形式。另一种关系是修饰关系，即包含与被包含的组件之间是被修饰与修饰的关系。在这种形式中，被包含的组件并不是包含组件的组成部分，而是对包含组件的某方面进行说明修饰。比如对一张表格而言，表格与组成表格的行构成了父子关系，而表头是说明表格用途的，所以表格与表头之间构成了修饰关系。对于这两种形式的包含关系，在 UIComponent 抽象类中都定义了相应的方法，以便组合和遍历相应的组件。

由上所述，我们可以方便地定义出满足需要的 JSF 组件。当然 JSF 在 UIComponent 的基础上定义了一套标准组件集合，可以满足常见的 Web 应用需求，这些标准组件的具体描述可以参看 JSF 规范[3]。

3.2 请求处理生命周期

对于处理含有 JSF 组件的请求，JSF 定义了一个标准的处理流程，该处理流程被称作请求处理生命周期，它主要由六个不同的阶段组成，它们分别是：

1）重建请求树阶段。其任务是创建请求页面的组件树。如果以前显示过请求的页面，则页面以前的状态信息会被 JSF 自动恢复到组件树中。这就解决了 Web 应用程序的一次会话过程中多次请求之间状态保持的问题。

2）应用请求值阶段。其任务是从请求中提取有关信息并存储到组件中。JSF 将根据请求中的信息，从组件树的根开始递归更新组件所包含的数据。其中，对于数据输入型组件而言，把请求中的数据赋值给组件将自动调用相应的 converter 转换器类，并把数据转换失败的信息存放到 JSF 上下文中；而对于是 ActionEvent 事件源的组件，如果该组件处于激活状态，将产生 ActionEvent 事件，并在后面适当的处理事件阶段被处理。

3）验证有效性阶段。该阶段就是遍历组件树，将需要验证其值的组件用组件的 validate()方法或者相应的 validator 验证器类验证，确保组件的值符合语义的要求。

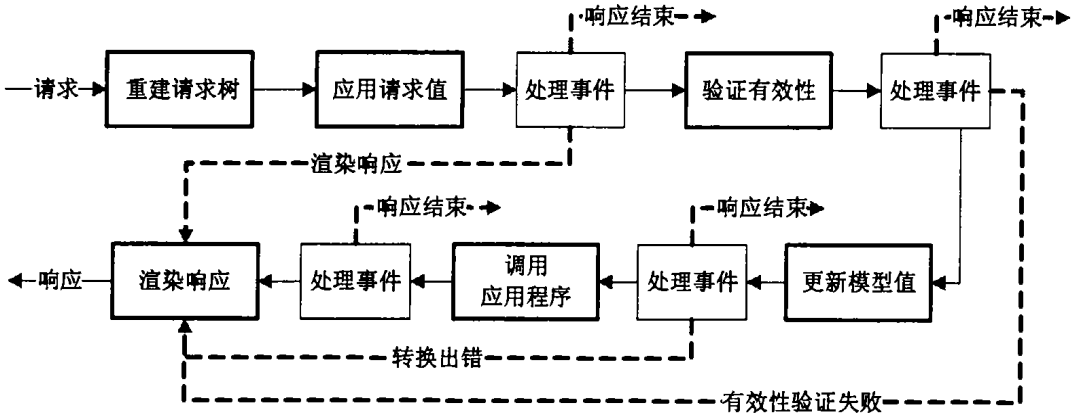

图2 JSF 的请求处理生命周期

4)更新模型值阶段和调用应用程序阶段。这两个阶段可以看作是联系 Web 应用程序中应用逻辑层的桥梁与纽带。在更新模型值阶段,JSF 用组件的值更新相应的数据模型(通常是应用逻辑层中的 JavaBean)的值,而在调用应用程序阶段,JSF 则调用特定的方法来处理这些数据,完成应用逻辑层的任务。处理的结果连同 Web 应用中关于页面导航的配置信息,一起决定了响应所渲染的页面。

5)在上述几个阶段的后面都会跟随一个处理事件阶段,该阶段就是将前面阶段所产生的事件送给相应的事件监听器进行处理。事件中的 phaseId 属性决定该事件在哪一个处理事件阶段被处理。如果事件处理中调用了上下文的 responseComplete()方法,将导致响应结束;而如果调用了上下文的 renderResponse()方法,将会使流程处理直接转向渲染响应阶段。

6)渲染响应阶段。其任务是将请求处理的结果转换成客户端可以接受的形式传给客户端,并且保存响应状态,以便处理后续请求。其主要操作就是调用页面中各组件所提供的渲染组件功能,把组件中的信息转换成客户端所显示的内容。

这些阶段按照图2所示的次序构成了 JSF 的请求处理生命周期。图中的实线箭头指示了正常的处理流程,而虚线箭头则指示了例外情况下的处理流程。

4 JSF 开发 Web 应用程序的过程

前面我们已经提到,在用 JSF 开发 Web 应用程序时,项目开发人员可以有几种角色:页面设计人员、代码开发人员和组件开发人员。这几种角色可以相互配合、并行工作,从而提高项目的开发效率。下面就分角色描述 JSF 开发 Web 应用程序的过程(在 JSF 集成到 JSP 的环境下)。

对于页面设计人员而言,其任务就是在页面中放置 JSF 组件、将这些组件与应用层的 JavaBean 绑定、设置 validator 验证器类和 converter 转换器类等。为了完成这些任务,需要注意以下几点:

·在每个 JSP 页面的头部写入下面的语句:

〈%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %〉

〈%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %〉

这样就可以在页面中以标签的方式使用 JSF 提供的 JSP 自定义标签库,以及 JSF 所定义的标准组件。

·〈f:view〉标签代表组件树的根 UIViewRoot 组件,因此页面中所有的 JSF 组件都应放在〈f:view〉标签中,以便 JSF 正确地生成页面所对应的组件树。

·可以用值绑定表达式和方法绑定表达式来把 JSF 组件与应用层的 JavaBean、验证器类、转换器类等对象联系起来。值绑定表达式和方法绑定表达式的形式是"#{…}",其中省略号代表绑定的对象。

对于代码开发人员而言,除了编写应用程序代码以外,一个重要的任务就是配置应用程序。在 JSF 中,配置应用程序主要包括两个方面:一是配置 Web 应用程序的部署描述符文件 web.xml。在这里需要指定 FacesServlet 和 FacesServlet 映射关系。FacesServlet 就是 JSF 的前端控制器,它处理所有与 JSF 相关的请求;而 FacesServlet 映射关系则定义了一个规则,以便让含有 JSF 请求的页面由 FacesServlet 来处理。另外一个需要配置的文件是 faces-config.xml,它作为 JSF 特定的配置信息文件,主要包含两方面的内容:ModelBean 的声明和页面间的导航规则。其中 ModelBean 可以看作是应用层的 JavaBean,它们封装了程序的应用逻辑,能够保存属性值,可以与页面中的值绑定表达式和方法绑定表达式配合使用;而页面间的导航规则定义了如何把分散的页面组成有机的整体,它主要是以页面为单位,分别对每个源页面定义一些条件和对应的目标页面,以形成导航规则。这样当某个条件满足时,JSF 就可以从源页面转到该条件对应的目标页面。通过这种方式,JSF 就将 Web 应用程序的流程控制集中化了。

如果 JSF 提供的标准组件库不能满足应用的要求,这时就需要组件开发人员来创建自定义组件。组件开发人员可以在 UIComponent 抽象基类或者标准组件的基础上进行开发,在这里就不再赘述。关于 JSF 开发 Web 应用程序的具体示例可以参见《An Introduction to JavaServer Faces》[4]一文。

5 JSF 渲染组件的改进

如前所述,渲染组件就是生成客户端能够显示的信息,对于 Web 浏览器而言,这些信息一般包括 HTML、CSS 和 JavaScript 等。JSF 提供了两种方式来渲染组件:一种是由组件自己来实现渲染操作,这使得如果不改变组件的代码就不可能修改渲染的样式;一种是将渲染操作分发给 renderer 渲染器类来完成,组件可以注册特定的渲染器类。这种方式虽然可以不用修改组件的代码,但仍然需要修改渲染器类的代码来实现样式的改变。因此,这两种方式都没能把样式与代码分离开来,使得样式的修改需要由组件开发人员来完成。

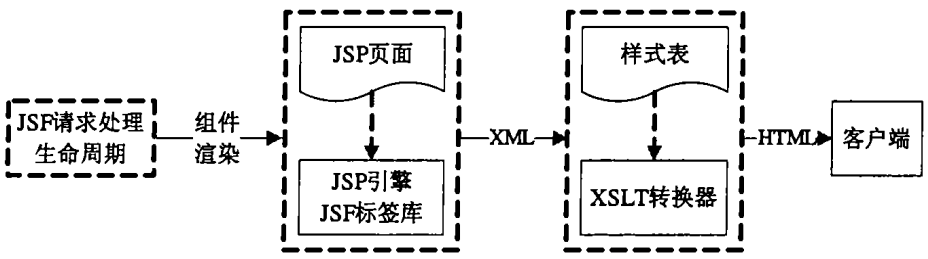

图3 改进后的组件渲染流程

为了解决这个问题,本文提出了一个 JSF 与 JSP、XML 相结合的应用方案。众所周知,XML 独立于平台和表现形式,只反映数据及其相互关系,用 XML 表示 JSF 组件所产生的数据,正好满足了组件要与表现样式相分离的需求,同时利用 XSLT 技术,同样的 XML 数据可以有形式多样的表现形式;而 JSP 被设计用来产生任何形式的文本数据,除了生成 HTML 文档以外,也可以生成 XML 文档,并且 JSF 可以很好地与 JSP 集成。这样,JSF 组件在渲染时就可以通过 JSP 生成独立于表现形式的 XML 数据。而这些 XML 数据则通过样式转换语言 XSLT 的处理,加入数据的表现样式,形成最终供

客户端显示的数据。该应用方案的处理流程如图3所示。

通过改进后的应用方案，我们可以将渲染组件中的样式处理集中化，可以不用修改代码就可以改变显示的样式，增强了渲染组件的可扩展性，使渲染操作的条理更加明晰。

结束语 JSF 作为新一代的 Web 应用框架，通过引入用户接口组件模型及其相应的处理流程，从而使 Web 应用程序的开发具有了与桌面应用程序的开发相似的特点和便利。此外，JSF 的定义均基于 Java API，并提供了 JSP 自定义标签库，为 JSF 与 JSP、Servlet 等现有技术实现很好的集成提供了条件。在实际的应用中，针对 JSF 现有标准在渲染组件方面的一点不足，本文还提出了一个 JSF 与 JSP、XML 相结合的解决方案，从而充分发挥了 JSF 在用户界面显示方面的可扩展性和多样性。

JSF 自身诸多的优点使包括 SUN、IBM 等大公司在内的越来越多的人投身到 JSF 的发展中，这让我们有理由相信 JSF 将成为多层 Web 应用程序开发的主流技术。

参 考 文 献

1 Coward D, Yoshida Y. Java Servlet Specification 2.4[S/OL]. http://jcp.org/aboutJava/communityprocess/final/jsr154/, 2003-11-24/2004-8-2

2 Roth M, Pelegrí-Llopart E. JavaServer Pages Specification 2.0[S/OL]. http://jcp.org/aboutJava/communityprocess/final/jsr152/, 2003-11-24/2004-8-2

3 McClanahan C, Burns E, Kitain R. JavaServer Faces Specification 1.1[S/OL]. http://java.sun.com/j2ee/javaserverfaces/, 2004-5-27/2004-8-2

4 Burns E, Horwat J. An Introduction to JavaServer Faces[Z/OL]. http://java.sun.com/j2ee/javaserverfaces/jsfintro.html, 2004-6-28/2004-8-2

（上接第223页）

间的计算性能差距会随着规模的增大而增大。我们取节点平均计算时间的比例来确定 δ。

4.2 BWMM 计算性能试验

因为 BWMM 的状态空间数为 (n^k) 个，因此我们用动态规划法来代替枚举法，分别考虑进程数从10到30时，子任务数为5，10，15时的计算时间。

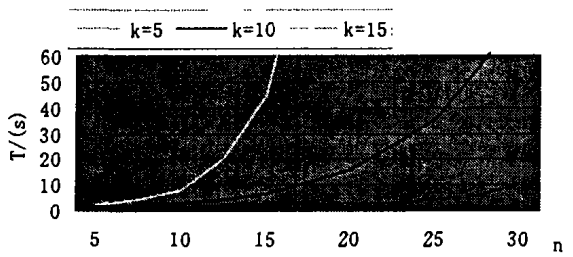

图2 BWMM 计算时间

从图2中可以看到，随着规划的子任务数数量增多，使得计算时间发生突变的进程数越小。如果我们考虑1分钟内结果都是可接受的，那么当进程数达到20，子任务数为10时性能最好。

工作小结及将来工作 本文针对现有实时分布式系统中事件驱动任务流资源分配策略的问题提出了一种通用的任务流均衡负载算术模型，BWMM，并且实现了动态任务规划的必须需求：

(1)子任务大小的动态划分。BWMM 以面向对象的方式对任务 J 进行子任务划分，将 J 初始化为不可再分的以类为单位的子任务，因此每个子任务的执行实际上是这个实例化了的这个类的对象，而我们根据网络中节点的性能差异允许部分子过程进行串行计算，这实际上是将多个子任务合并成为上级子任务 j'，这样就实现了 J 的子任务大小不定的动态划分。

(2)子任务执行进程的动态选择。因为网络中参与节点的环境异构，因此导致了节点间计算性能的差异。BWMM 中用 $\delta(p,v)$ 来表示进程 p 所在节点 v 的性能参数，选择各种情况中计算时间最短的一种为子任务分配策略。当子任务的标准计算时间发生变化时（例如类中新增添了代码），任务分配策略也动态地发生变化。

(3)计算时间总能最优。虽然 BWMM 解决的是一个 NP 问题，但最终计算结果中总能找到一种任务分布情况为计算时间最优。

但 BWMM 针对的是事件驱动的任务流类型，即相对于数据驱动的任务流来说，忽略了子任务间数据的关联（例如 j3 的执行必须要 j1的计算结果）。另外，节点的性能参数准确性的提高以及改进问题算法也是下一步要进行的工作。

参 考 文 献

1 Kafil M, Ahmad I. Optimal Task Assignment in heterogeneous Distributed Computing Systems. IEEE Concurrency on Complex Distributed Systems, 1998, 6(3): 42～51

2 Jaroodi AI, Jiang Hong, Swanson D. A Comparative Study of Parallel and Distributed Java Projects for Heterogeneous Systems. In: Proc. of the 16th Intl. Parallel and Distributed Processing Symposium, 2002

3 Georgiadis L, Nikolaou C, et al. A fair workload allocation policy for heterogeneous systems. Journal of Parallel and Distributed Computing, 2004, 64(1): 507～519

4 Rotaru T, Nageli H-H. Dynamic load balancing by diffusion in heterogeneous systems. Journal of Parallel and Distributed Computing, 2004, 64(4): 481～497

5 Ali S, Kim J-K, etc. Greedy Heuristics for Resource Allocation in Dynamic Distributed Real-Time Heterogeneous Computing Systems. 2002 International Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA'02), Las Vegas, NV, 2002. 519～530

6 Ravindran B, Li P, et al. Proactive resource allocation for asynchronous real-time distributed systems in the presence of processor failures. Journal of Parallel and Distributed Computing, 2003, 63(12): 1219～1242

7 Ranaweera S, Agrawal D P. A Task Duplication Based Scheduling Algorithm for Heterogeneous Systems. IPDPS 2002. 445～450

8 Boloni L, Marinescu D C. Robust Scheduling of Metaprograms. Journal of Scheduling, 2002, 5(5): 395～412

9 Ali S, Maciejewski, et al. Measuring the Robustness of a Resource Allocation. IEEE Transaction on Parallel and Distributed Systems, 2004, 15(7): 630～641

10 王永炎，王强，等. 基于优先级别表的实时调度算法及其实现. 软件学报, 2004, 15(3): 360～370

11 Bajaj R, Agrawal D P. Improving Scheduling of Tasks in a Heterogeneous Environment. IEEE Transaction on Parallel and Distributed Systems, 2004, 15(2): 107～118

12 Bansal S, Kumar P, et al. An Improved Duplication Strategy for Scheduling Precedence Constrained Graphs in Multiprocessor Systems. IEEE Transaction on Parallel and Distributed Systems, 2003, 14(6): 533～544