应用MVC设计模式构建Web应用系统框架及其扩展

罗光春　卢显良　韩　宏　卢　军
（电子科技大学信息中心　成都610054）

The Web Application Frame and Expansion Developed from MVC

LUO Guang-Chun　LU Xian-Liang　HAN Hong　LU Jun
(Information Center of UEST of China ChengDu, China 610054)

Abstract　The MVC designed pattern was developed in Smalltalk-80 and was wildly used in the task of software design. This paper introduces MVC to Web application design, and describes a design pattern based MVC in detail. This design pattern not only detaches the transaction and expression form, but only can improve the maintainability, expandability and performance by using the module library, template library, message object and queue cache. In conclusion, this paper presents a novel Web application design pattern based on MVC.

Keywords　MVC, Designed pattern, Web application

1. 引言

Web应用系统是指利用各种动态Web技术开发的基于B(rowser)/S(erver)模式的事务处理系统。就目前来说，Web应用系统的开发模式和开发工具是业界探索的兴趣点之一，它直接影响到所开发出的应用系统的可维护性、可扩展性、稳定性、安全性等几个比较敏感的方面，本文所阐述的以MVC模式[1]进行Web应用系统开发的思想主要涉及前两个方面，即系统的可维护性和可扩展性，这也是目前Web应用系统开发中遇到的非常棘手的问题。在Web应用系统开发中引入MVC设计模式的思想，其重要意义在于将系统事务逻辑的实现与系统表现形式分离开来，使得这两方面的开发人员能尽可能地拥有各自的进度空间，极大地提高了项目开发的效率。

2. MVC设计模式的简单介绍

MVC包括三类对象。模型Model是应用对象，视图View是它在屏幕上的表示，控制器Control定义用户界面对用户输入的响应方式。不使用MVC，用户界面设计往往将这些对象混在一起，而MVC则将它们分离以提高灵活性和可复用性。MVC通过建立一个“订购/通知”协议来分离视图和模型。视图必须保证它的显示正确地反映了模型的状态。一旦模型的数据发生变化，模型将通知有关的视图，每个视图相应地得到刷新自己的机会。这种方法可以让你为一个模型提供不同的多个视图表现形式，也能够为一个模型创建新的视图而无需重写模型。

3. 引入MVC设计模式的Web应用系统及其改进方案

3.1　在Web应用系统开发中引入MVC的思想的尝试及其局限

尽管MVC的设计模式早在Smalltalk-80中就被提出并在此后得到业界的广泛接受，但在Web项目的开发中，引入MVC却是步履维艰。这主要基于以下原因：一是在Web项目的开发中，程序语言和HTML的分离一直是让人头疼的问题。早期的CGI程序以字符串输出的形式动态地生成HTML内容，这种方式本质上就是将HTML内容拆分之后，嵌入到CGI程序中。后来随着脚本语言的出现，前面的方式又被倒了过来，改成将脚本语言书写的程序嵌入至HTML内容中。这两种方式有一个相同的不足之处在于它们总是无法把这两者彻底地分离开来。因此，在此基础上实现MVC三层分离的模式也就变得极为困难；二是脚本语言的功能相对较弱，缺乏支持MVC设计模式的一些必要的技术基础，如消息机制和事件响应机制。这种情况直到JSP Model 2[4]问世时才得以改观。利用Java语言的强大功能，JSP Model 2在一定程度上实现了MVC的设计思想，如图1所示。其实现机制为：

·Model数据层：全部封装于Enterprise JavaBean之中；

·View表示层：由JSP负责处理页面的表示。比如：数据表示的格式，分页等等。

·Control控制层：Servlet接受用户在页面的输入以及提交动作，并根据动作指示，进行相应的事务处理（调用相应的EJB组件），然后根据处理结果交给相应的View表示层JSP程序，由它们负责表示。

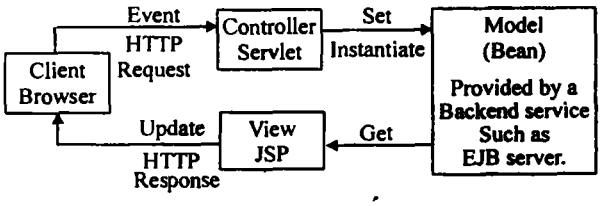

图1　JSP Model 2提出的Web应用系统框架

JSP Model 2的这种机制基本满足了开发者对于分离Web系统事务逻辑和用户界面的需要，但同时它也存在一些局限。例如，虽然Model 2实现了MVC设计模式的基本思想，但这三层（Servlet、JavaBean/EJB、JSP[5]）之间的联系却比较

罗光春　博士研究生，卢显良　教授，博士生导师，卢　军　博士研究生。

紧密，特别是作为 Controller 的 Servlet 这部分，它在整个系统中既控制着 Model 组件的选择和调用，又负责将 Model 组件提供的数据传递给特定的 JSP 处理，这样一来，Servlet 自身需要实现非常复杂的处理逻辑，对于系统的开发和维护来说都是一个棘手的问题。因此，在下面的 Web 应用系统模型中，本文首先考虑的就是要将 Controller 这一层所要实现的功能进一步地分解。

3.2 改进后的基于 MVC 模式的 Web 应用系统框架

如图2所示，这个系统模型包含三个从原始的 Controller 层分离出来的核心的控制对象/类（Sys Controller、Model Center、LPT Center，LPT 是指逻辑页面模板，关于它的概念将在后面介绍）以及它们各自对应的核心库（系统逻辑描述库、系统模型/组件库、逻辑页面模板库），下面就以这三个核心控制对象/类和核心库为线索，阐述该系统模型的设计思想和运行机制。

首先是系统控制器和系统逻辑描述库，这是整个系统的“感觉器官”，它负责接受用户请求，并根据请求查找相应的处理逻辑。系统逻辑描述库可以是一个 XML[2]文件，也可以是若干 XML 文件的集合，在这个库中，以树结构的形式逐一描述了不同的用户请求对应的处理逻辑，主要包括交互形式、数据源、差错处理机制等。位于前端的系统控制器负责检索该库的内容，并将取出的信息和用户请求组合在一起交由 Messenger 封装。系统控制器对该库的检索一般说来有两种方式，一种是使用 DOM 对象[3]，这种方式要求在这个应用系统启动时，将整个库的内容通过一个 DOM Parser 解析后以 DOM 树的形式加载至内存中；另一种方式是使用目前已经较为成熟的 SAX 接口。这两种方式各有其优缺点。使用 DOM 方式，可以实现结构化的查询，当库的规模较大时可以节省效率，但这种方式对资源的需求也相对较高；而 SAX 接口则以高效和较少的资源占用著称，唯一的不便就是某些信息的获取不像使用 DOM 那样方便。

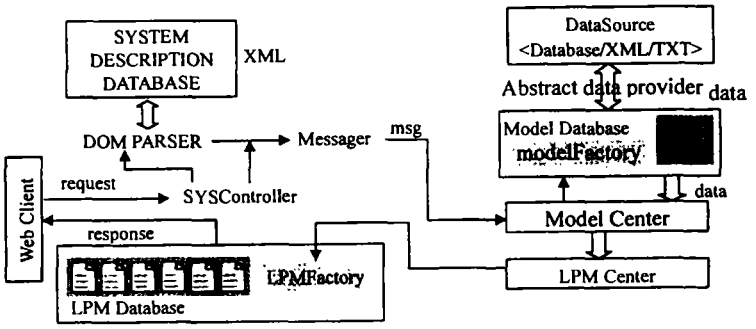

图2 基于 MVC 设计模式的 Web 应用系统框架的改进方案

其次是模型/组件中心和系统模型/组件库，这一层通过调用特定的组件实现对指定事务逻辑的处理。在模型/组件库中，包含着大量的对象或组件，这些对象或组件封装了对特定处理逻辑的响应方法，当模型中心从 Messenger 那里接收到处理消息（这部分模型的扩展内容我们将在下面讨论）时，它会根据消息中所封装的处理逻辑的不同，来选择合适的对象或组件，并将处理后得到的数据按一定格式封装后交给逻辑页面控制器。而对于每个 Model 对象或组件来说，它们只需要关心如何实现特定的处理逻辑以及如何取得它们需要的数据，而不必理会这些数据所需要的表现形式。

第三是逻辑页面模板（LPM）中心及其对应的 LPM 库。这里的逻辑页面模板是指为某一特定数据类型实现其表现形式的程序集，它既可以对应于一个磁盘文件，又可以是多个磁盘文件的集合。另一方面，用户请求的某一特定的 Web 页面既可以只包含一个逻辑页面，又可以由多个逻辑页面组成。对于一个逻辑页面模板来说，它所负责的功能非常简单，就是将特定类型的数据表示出来，它不需要被告知任何事务处理逻辑，也不需要关心数据的来源和具体内容，因此，在实际开发中，它们更多地使用诸如标签库（JSP）或数据绑定（.Net）等技术开发，而在 UI 设计阶段，这类型的 Server 端脚本可以很容易地被前台设计人员理解，并可以采用一些所见即所得（WYSIWYG）的开发工具进行开发。在这一部分中，LPC 中心的作用是接受来自模型/组件中心的数据，并针对其中指定的数据格式以及处理逻辑对 UI 的要求选择合适的逻辑页面模板，最后将逻辑页面模板的输出封装成 Http 响应并发往客户端。

这里需要指出的是，在系统的三个控制对象中，除了系统控制器外，其它两个都并不直接访问自己所控制的核心库，例如模型/组件中心并不直接调用组件库中的对象或组件，而是通过类 ModelFactory 来实现这种访问，如图2所示。

通过对该系统模型的分析我们可以看出，整个系统完全遵循 MVC 设计模式的框架，在系统开发期间，各部分开发人员之间不存在明显的制约关系，只要系统框架确定，各个模块的开发工作就可以完全并行地开展，这在以前的 Web 项目开发中是不可思议的。另一方面，由于将 MVC 模式中的 Control 一分为三，使得 MVC 三层之间的耦合变得更加宽松，从而在很大程度上提高了整个系统的可扩展性。下面我们就举一个简单的示例来说明如何通过在 MVC 的三层模型之间添加中间处理过程来扩展这个系统模型。

3.3 利用队列缓冲机制改善系统性能

前面提到系统控制器在获得与用户请求对应的处理逻辑之后，将会把这些内容封装成特定的消息发送给模型/组件中心，组件中心在收到消息后再根据其要求作具体的处理。这种一对一的模式结构简单，逻辑清晰，开发中也很容易实现，但根据模型系统实际运行的效果来看，这种简洁的方式只能适用于较小规模的应用，对于一个大型的 Web 应用系统（如企业级的网上协作平台）来说，这个环节必须进行优化。下面是一个讨论模型。

系统控制器在得到相应的处理逻辑之后，通过 Messenger 将这些内容封装成特定格式的消息，然后并不直接将消息交给 MC 处理，而是将消息放入系统的消息队列中（这里的系统消息队列并不是指操作系统的消息队列，而是一个由 Web 应用系统自行管理的一个全局的缓存区）。同时，模型/

（下转第164页）

F2: $\sum_{i=1}^{5} i\cos[(i+1)x_1+i]\cdot\sum_{i=1}^{5} i\cos[(i+1)x_2+i]$

其中 $-10\leqslant x_i\leqslant 10$ 当 $i=1,2$

这是一个二维多峰函数，函数有760个局部最小，其中18个是全局最小，其值为-186.73

F3: $100(x_1^2-x_2)^2+(1-x_1)^2$，$-2.048\leqslant x_i\leqslant 2.048$

这是一个非凸函数，虽然它是一个二维单峰函数，但它是病态的，难以进行全局极小化。它的全局极小值为0，极小点为(1,1)

F4: $-\sum_{i=1}^{n} x_i\sin(\sqrt{|x_i|})$ $(-500\leqslant x_i\leqslant 500)$

函数的次全局极小点远离全局极小点，函数的全局极小点$\{x_i=420.96875, i=1,\cdots,n\}$

对函数F2，F3各运行10次，函数值的计算保留小数点后14位有效数值。算法的参数的取值为N=20，b=10，因为T的取值越大，求得最优解的概率越大，求得最优解的精度也越高，所以实验测试连续运行算法10次，其解的T值较为精确。结果如下：

表1 二维函数计算的结果

函数	算法求得的最优函数值	迭代次数T
F2	0.00000000000000	15000
F3	186.73090883102394	500

对函数F4，取维数n=20，50，100，200，500各做1次实验，函数值的计算保留小数点后14位有效数值，算法的参数的取值为N=20，b=2，所得最优解向量的最小分量值，最大分量值，以及T的取值如下(T的取值更大的话，求解的精度也将更高)：

表2 高维函数计算的结果

维数	求得的最优点的最小分量值	求得的最优点的最大分量值	迭代次数T
20	420.96840375635936	420.96883072006694	35000
50	420.96691741297690	420.96942438125824	85000
100	420.96574323812429	420.96925839726208	170000
200	420.96458019978458	420.96946954489171	360000
500	420.96670311248499	420.96923486539667	1260000

对函数F4，若取参数N=20，b=5，函数的维数为20维，迭代次数T=35000，算得的最优解向量的最小分量值为-302.52493475233606，最大分量值为420.968747303 29939，算法收敛到局部最优点。若将T的值增大，T=70000，最优解向量的最小分量值为420.96874459692141，最大分量值为420.96874785510272，算法收敛到全局最优点。

结论 由以上运算实例可见，采用最优个体保留，每代用保留的最优个体替换当前群体的最差个体，只使用非均匀变异算子的单亲遗传算法，由于在前期能有效勘探出最优区域，在后期能对最优区域进行精细的搜索，从而能有效地防止遗传算法的过早收敛，又有较高的求解质量，算法的收敛速度也很快。

同其他的改进的遗传算法相比，PGA算法的参数较少，而且易于控制。算法的收敛速度和求解精度受终止代数及最大迭代次数T的控制，T越大解的质量越高，当然收敛速度也减慢，T的取值可由对解的质量要求来定。算法的参数b对算法收敛速度有影响，b增大，算法收敛速度会加快，但也容易出现算法的过早收敛，其原因是较大的b值，限制着非均匀算子的搜索步长，使前期的勘探不够充分，使个体进入最优点的吸引域的概率降低，从而造成算法最终收敛到局部最优值，当然如果加大迭代次数，也是可以收敛到全局最优点的。根据经验值，b的取值范围在1到10之间，对于简单低微的函数，可以取较大的b值，对于复杂的高函数，可以取较小的b值。

参考文献

1 徐宗本，高勇. 遗传算法过早收敛现象的特征分析及其预防[J]. 中国科学(E辑)，1996，26(4)：364～375

2 黄樟灿，陈思多，李亮. 演化计算中的种群隔离与自聚集. 软件学报，2002，13(4)：827～832

3 李海民，吴成柯. 自适应变异遗传算法及其性能分析. 电子学报，1999，27(5)：90～93

4 Janikow C，Michalewicz Z. An experimental comparison of binary and floating point representations in genetic algorithms. In Belew and Booker [21]，pages 151～157

(上接第130页)

组件中心负责扫描整个消息队列，发现未处理的消息并将其提取出来，解析其中内容，然后交给ModelFactory处理。

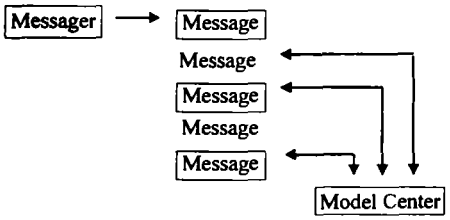

图3 采用消息队列进行缓冲处理

引入这种队列缓冲机制后，我们就可以根据需要对整个应用系统进行扩展以提高其性能。例如，为了提高系统的处理能力，可以通过编程使模型/组件中心支持多线程的工作方式，这样，系统对请求的处理能力就可以大幅度地提高；另外，我们看到，引入缓冲队列这一中间处理过程之后，我们只需要再附加少量的模块就可以把组件中心这个庞大的模块部署到多个服务器上去，在多个服务器之间实现分布式处理和负载平衡。

总结 综合上面对该系统模型的分析，我们可以看到，这个改进后的以MVC设计模式构建的Web应用系统框架，其结构较传统的框架清晰，系统各个模块之间的耦合相对宽松，特别是它在很大程度上解决了长期困扰Web开发人员如何将系统的事务逻辑和表现形式分离的难题，实现了开发工作的并行开展。并且，在引入了模型组件库和逻辑页面模板库之后，整个系统变得更为灵活，其可维护性和可扩展性得到了极大的提高。

参考文献

1 Gamma E. Design Patterns. U.S. Addison-Wesley Pub Co.

2 Martin D. Professional XML. U.K. Wrox Press

3 Mohr S T. Designing Distributed Applications with XML, ASP, IE5, LDAP & MSMQ. U.K. Wrox Press

4 Sun Microsystems Inc. JavaTM Servlet 2.3 and JavaServer Pages™1.2 Specificatios

5 Sun Microsystems Inc. JavaServer Pages™ Standard Tag Library Specification