一个基于 J2EE 的通用问题跟踪平台

A Common Issue Tracking Platform Based on J2EE

宗 薇 （外交学院计算机中心 北京 100037）

摘要：对现实中各种问题跟踪系统进行抽象，建立一个通用问题跟踪平台，可以有效地降低由于重复开发带来的成本，大大地提高效率。文章中介绍了利用 J2EE 相关的 Struts 技术和 Hibernate 技术开发的一个通用问题跟踪平台的整体架构和设计思路，并对平台中用到的相关技术进行了分析。

关键词：问题跟踪 MVC Struts Hibernate

1 问题跟踪平台

在实际工作和学习中会碰到各种问题，于是会有反映问题、解决问题的一套流程，这套流程可能是在公司内部，也可能涉及多个公司。针对不同性质的问题一般有不同的专有系统，比如 bug 系统主要是针对软件开发过程的软件质量控制，customer service 系统主要是针对产品的用户，helpdesk 系统一般用于公司内部的 MIS，还有其他类似的各种系统。

市场上有各种相关产品，比如在软件开发领域被广泛应用的 bug 管理系统 bugzilla、jira 等。各个公司可能会各自开发一些系统，功能类似于 customer service 系统和 helpdesk 系统。这些系统虽然是针对不同的领域，但是其本质都是以问题为中心的，针对问题进行跟踪管理。问题都有一个创建、指派、解决、关闭的生命周期，而其生命周期的各个动作都由具备相关角色的人员去完成。

基于上述情况对这些系统进行抽象，设计一个通用问题跟踪平台，在实际使用中，这个平台可以根据需求再进行定制，然后成为一个特定的问题跟踪系统。

这里所说的问题跟踪平台实质上是一个电子化流程工具，是为用户与支持之间提供一个沟通的平台，用户可利用这个平台反馈各种问题，支持能快速解决问题并将结果反馈给用户。

通过分析，这个通用问题跟踪平台应该有如下特点：

（1）以问题（后面称为 issue）为中心，issue 具有各种属性，如：优先级、状态等，还有各种有用信息，如主题、内容、时间等；

（2）有如下几种关键角色（role）：reporter（问题报告人）、solver（问题处理人）、reporterManager（问题报告人的 manager）、solverManager（问题处理人的 manager），另外还有 admin（管理员）、guest 等角色。其中 reporter、solver 是和 issue 紧密关联的双方，reporterManager、solverManager 则是 issue 的关注，在某些系统中可能仅有关注的权限（比如可能会收到相关 mail），在某些系统中可能还会有问题的分发处理的权限（如 bug 系统）；

（3）issue 在 role 之间流转，issue 的流转形成问题流（issueFlow）。issueFlow 的起点是 reporter（创建 issue），终点也是 reporter（关闭、删除、无效 issue）。issueFlow 实际上是完成了 issue 整个生命周期。

可以看到各种不同的问题系统之间的差异主要体现在 issueFlow 上，因此这个问题解决平台的 issueFlow 是可以定制的，这样才具有通用性。另外，问题的各种属性也是可以定义的。

2 J2EE 技术介绍

J2EE（Java 2 Platform Enterprise Edition）是 Sun 公司提出的一种基于 Java 技术企业级应用模型，它提供了多层分布式的应用模型、组件复用、一致化的安全模型以及灵活的事务控制。在此基础上开发的系统具有较高的可用性、安全性、可扩展性和可移植性等优点。

传统的 Client/Server 模式是二层化应用，Server 端一般提供数据库服务，而 Client 端负责数据访问、业务

逻辑实现、结果展示，其缺点是结构不清晰、维护困难、扩展性差，特别是对于复杂环境的大型应用，使用成本很高。

J2EE 框架采用 Browser/Server 结构，Server 端定义了一套标准化组件，将企业应用分为多个层次，每个层次专注于特定的任务，这种多层结构使企业应用具有了较强的灵活性和可伸缩性。

J2EE 技术内容庞杂，包括 JSP、Servlet、JDBC、JNDI、EJB、RMI、XML、JTS、JTA、JAF、JavaMail、JMS、Java IDL 等技术。

一个典型的 J2EE 应用是采用三层结构：

- 表示层（Presentation）：负责提供用户界面
- 业务逻辑层（Business Logic）：负责实现业务逻辑
- 数据持久层（Data Persistence）：负责业务数据的存储

3 基于 J2EE 的通用问题跟踪平台

3.1 软件架构

如前面所述，通用问题跟踪平台是多种系统的抽象，可能应用于较复杂的环境。基于 J2EE 的各种优点和开发优势，因此采用 J2EE 技术实现这个平台。

架构上主要采用 Struts 技术和 hibernate 技术相结合的方案。

3.1.1 整体结构采用 struts

Struts 是一个开源的应用框架，是当今流行的 MVC 模式（Model/View/Control）的一个实现。利用 Struts 对通用问题跟踪平台进行如下分解：

（1）Model 层。相当于 J2EE 三层结构的业务逻辑层，作用是建立业务模型，完成业务数据的处理。

将 issue 在 issueFlow 中的行为分解成很多元操作：creat（创建）、save（保存）、reply（答复）、assign（指派）、close（关闭）、delete（删除）。这些元操作在商业逻辑类里面都表现为一个个独立的 method。

不同的应用，业务流程不同，业务流程是上述元操作的组合。

另外还有一些类实现 issueFlow 无关的操作，比如查询、统计、邮件通知。

这些具体的业务逻辑类主要是封装了各种业务数据和业务状态。Model 层还有各种 Action 类，Action 类主要是调用这些业务逻辑类，去完成特定的操作。

比如 issueCreateAction 会去调用 creat、save 这两个元操作，去完成 issue 的创建，根据需要，可能会调用邮件通知方法发送邮件。

（2）View 层。负责数据的展示，即用户界面。实现方式主要是 jsp 和 servlet 及一些静态页面，在 jsp 文件里面用到一些 struts 标签库。View 层对控制器一无所知，当更改模型时，view 层会自动得到通知。

在该平台中主要是 issue 的上报、处理以及统计查询等页面，是一些 jsp 文件。在 jsp 文件里大量使用 struts 的标签（主要是 html、bean、logic 这三个 taglibs），通过这些标签库来控制输出。由于这些标签库的使用，屏蔽了业务数据，所以 jsp 开发人员无需了解业务，只需专注于外观设计。

在实际使用中，可以很容易修改外观，而不会影响整个系统，同时如果业务上有修改，也不用修改 jsp 文件。比如，在创建 issue 时，需要有一个 priority（优先级）的下拉式选择，只需在 jsp 文件中使用 html：optionsCollection 标签即可，以后 priority 的数量及名字若有调整，只需要修改业务数据。

（3）Control 层。负责 View 和 Model 之间的流程控制，由 Struts 提供的中心控制器（ActionServlet）来完成。

用户的所有 http 请求，都会送给 controller，controller 结合配置文件 struts – config. xml 对这些请求进行分析和处理，最后将请求分发给相应的 action 类。controller 还将来自 http 请求的数据以 ActionForm 类的形式传送给 Action 类，同时 controller 还会执行一些简单的数据校验（比如表单必须不为空）、reset 操作（比如将表单清空）。

比如当 reporter 点击 issueCreate. do 来创建 issue 时，http 请求发给 controller，controller 读取 struts – config. xml 后，将请求参数填写在 issueCreateForm 里，然后将 issueCreateForm 及请求分发给 issueCreateAction。issueCreateAction 调用具体的 issue 业务逻辑完成相应的操作，最后返回操作结果给 controller，controller 根据操作操作结果，调用合适的 jsp 页面给 browser。其结构大致如图 1 所示。

MVC 模式减弱了业务逻辑层和数据层之间的耦合，同时能让视图层更富于变化。这样我们的通用问

题跟踪平台可以根据具体应用定制各种不同的问题跟踪系统出来。

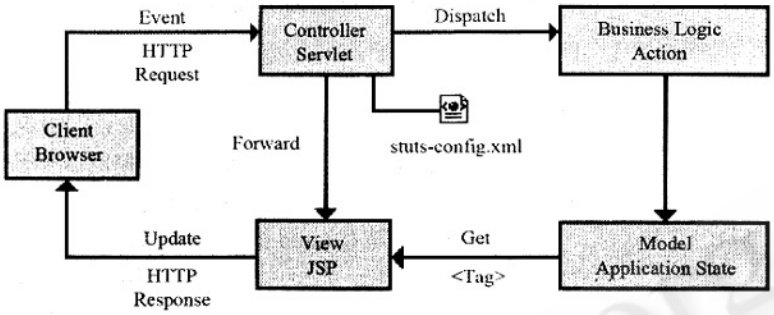

图 1

3.1.2 平台的数据持久层部分用 Hibernate 来实现。

Hibernate 是一个面向 Java 环境的开源的 O/R(对象/关系数据库)映射工具,其作用是对 java 类与数据表进行映射,完成数据的持久化,同时也提供数据的查询、一致性检查、事务管理等机制。

与直接调用 JDBC 操作数据库相比,hibernate 可以大幅度减少开发时人工使用 SQL 和 JDBC 处理数据的时间。程序员能够完全用 OO 的思维去设计系统,将全部精力集中在对象编程中,而不用关心数据库的连接、事务、并发性等问题。

另外,由于在编程中不会有特定数据库相关的代码,所以我们可以很容易地移植数据库而不用修改 JAVA 代码(仅需要修改 database schema 及 O/R 映射文件),大大提高了系统的灵活性。

Hibernate 的一些高级特性也会大大降低开发工作量。比如创建 issue 时,会同时写两个表:issue、issuememo,issue 表存储基本信息,issuememo 存储各种往来信息。由于 issue 与 issuememo 之间建立了 one – to – many(一对多)的关联,并且设置了 cascade(级联)特性,所以在创建保存的时候,只需保存 issue 对象即可,相应的 issuememo 对象会自动获取 issue ID 并自动保存。

3.2 设计思路

3.2.1 关系数据建模

平台数据库使用 mysql,可以选用一个合适的建模工具(如 PowerDesigner、Rose、Together 等)进行数据建模,这样就会生成一个基于 mysql 的 schema,如果未来需要使用其他数据库,可以在建模环境里稍加修改,即可生成其他相应数据库的 schema。大致包括这些内容:

(1) 表——问题(issue)表、各种属性表、角色表、用户表、几个 issue 扩展表(issue 详细来往信息、issue 关联信息等)、日志表;

(2) 索引——主要是对 issue 表建立索引;

(3) 数据完整性——实体完整性体现在表的主键上,比如 issue 表的 IssueID;域完整性主要是非空约束,比如 issue 表的 title 字段为 not null;参照完整性主要是以 issue 为中心,issue 表及扩展表对其他表的外键引用。

3.2.2 域建模及 O/R map

完成数据建模后,利用 hibernate 自带的工具 hibernate tools 生成 POJO 及 O/R 映射文件。

这些 POJO(Plain Ordinary Java Objects)实际上是 Java Bean,属性通过 getter 和 setter 方法访问,对外隐藏了内部实现的细节。POJO 和数据表之间是一一对应关系,如果数据模型比较完善,自动生成的 POJO 基本上勿需修改。也可以根据实际需要做些调整,如 User. java 与 Issue. java(用户与问题)之间应该是一对多的关系,在 User. java 里应该有一个属性对应 Issue 对象,这个属性可以设置成一个合适的集合(比如可设置为 Set 类型)。

O/R 映射文件是一个 xml 文件,文件里包含了对象/关系映射所需的元数据。元数据包含了 POJO 的声明和把它与其属性映射到数据库表的信息(属性作为值或者是指向其他实体的关联)。

映射文件里列出了持久化类的各个属性,以 Issue 表为例,主要有 IssueID 属性、title 属性、userid 属性、status 及其他相关属性等等。IssueID 是一个标识属性,对应数据库表主键字段,在映射文件里用一个 id 元素表示,可在 id 元素内设置其 generator 为 increment,这是 mysql 数据库自增型主键的生成方式。Issue 表的其他属性在 property 元素里表示。

可以在 id、property 元素内以 java 类型或 hibernate 类型来表示 Issue 对象的各个属性。

持久化类之间的关系(一对一、多对一)通过 one-to-one、many-to-one 元素来定义。比如 Issue 类与 User 类是多对一关系,Issue 类与 Status 类之间也是多对一关系。

3.2.3 实现商业逻辑类

完成建模和 O/R map 后,即可编码实现具体的业务类。

主要是两大部分:

(1) 实现 IssueService 类,里面是各种元数据。

(2) 实现 IssueAction 类,里面包含各种 method,对应来自 browser 的各种 http request。

在实现 IssueAction 类的同时,完成 IssueForm 类。IssueForm 类与 IssueAction 类是对应的,用于 View 层与 Controller 层之间数据传送。

3.2.4 系统认证

在不同的企业、不同的应用中,可能有不同的认证方式,比如常见的 LDAP 认证、NTLM 认证、UNIX 系统认证等,因此本系统需要考虑对这些常见认证方式的支持,同时还需要具备扩展性,以对未来新的认证方式提供快速支持。

这一部分的设计可以借鉴设计模式中的一种常见模式:工厂模式。

可以设计一个抽象认证类 Auth. java,针对不同的认证方式有不同的认证类,比如 AuthLdap. java 用于 LDAP 认证,AuthUnix 用于 Unix 系统帐号认证。另外有一个认证工厂类 AuthFact. java。以后需要增加新的认证时,可以增加一个新的认证类,只要这个新的认证类覆盖了 Auth. java 的相应方法即可,整个系统的其他部分勿需做任何修改。通过这种设计,平台的认证模块具有了很强的扩展性。

3.2.5 邮件功能

邮件通知功能在许多系统中常用。可以使用 JavaMail API,在 JavaMail API 的 javax. mail 和 javax. mail. internet 包中定义了会话、邮件、地址、认证、传输和存储等基本邮件管理对象,可以很容易地实现邮件对象的封装、发送、身份认证和接收等功能。

3.2.6 本地化与国际化

考虑到可能会有不同语言的用户,这个平台应该具备支持多语言的功能。Struts 框架很好的提供了实现 web 应用国际化的机制。

简单的讲,struts 是通过资源文件来实现的,我们可以针对不同的语言准备不同的资源文件。同时,在程序中做些处理,对来自不同语言的用户,调用不同的资源文件。

3.2.7 日志

平台的编码中使用了 Log4j 日志包,一方面用于程序开发过程中的调试,另一方面可以针对一些要求,在使用中输出特定条件、特定格式的日志信息,日志信息的输出目标也可以灵活配置。

Log4j 是一个开源的项目,是一个优秀的 java 日志工具包,通过 Log4j 可以在不修改代码的情况下,方便、灵活地控制任意粒度的日志信息的开启或关闭,然后使用定制的格式,把日志信息输出到一个或多个需要的地方。

最终我们通过 Struts 和 Hibernate 完成了一个基于 J2EE 的通用问题跟踪平台,该平台可以根据需要,定制为一个 bug 系统、一个客户支持系统、一个 help-desk 系统或者其他问题相关的跟踪系统。

系统具有如下特点:

(1) 基于 B/S 结构,系统的使用者操作简单;

(2) 基于 J2EE 技术,部署容易,可以部署在任何符合标准 J2EE 规范的容器里,如 tomcat、jboss 等,同时可安装在各种操作系统下使用;

(3) 使用开放型数据库设计,可以很容易支持各种数据库;

(4) 多用途,可用做各种问题相关的跟踪管理系统;

(5) 支持国际化及本地化

4 结束语

使用 J2EE 技术作为实现平台,采用 Struts 技术进行系统架构,可以用相对简单的方式实现通用问题跟踪平台,充分利用 J2EE 技术的特点,以较低的成本完成该系统的开发,并有利于系统的进一步扩展。

参考文献

1 http://struts. apache. org/

2 http://www. hibernate. org/hib_docs/reference/zh-cn/html/

3 http://logging. apache. org/log4j/docs/