基于双矩阵访问的控制技术①

杨光豹[1], 郑慧锦[2]

[1](浙江广播电视大学 青田学院, 青田 323900)
[2](浙江青田职业技术学校, 青田 323900)

摘 要: 在企业级 web 系统的应用中, 传统的用户权限管理的方法是按功能模块进行粗粒度分配, 首先将系统中各个模块的操作权限赋予特定的角色,然后再将这些角色赋予某些用户. 这种基于角色的权限管理缺少灵活性, 不能做到“量身定制”; 对于具有多种角色权限的用户来说, 使用系统时会存在诸多不便, 同时在系统扩展时权限管理与业务功能会相互影响, 它会给程序员带来额外工作量. 为了改进上述缺陷, 本文提出利用 Struts2、Spring3、Hibernate4 等进行整合开发 Web 系统, 通过权限管理拦截器对每个用户请求进行权限验证, 使系统能够采用细粒度方式管理用户权限, 增强权限管理的灵活性与系统的可扩展性.

关键词: 基于角色的访问控制; 访问控制矩阵; 拦截器; Spring;Struts2; Hibernate

Access Control Technology Based on Multiple Matrix

YANG Guang-Bao[1], ZHENG Hui-Jin[2]

[1](School of Qingtian, Zhejiang Radio & Television University, Qingtian 323900, China)
[2](Zhejiang Qingtian Vocational and Technical School, Qingtian 323900, China)

Abstract: In the enterprise web application system, the access powers of user are used to be assigned on the basis of the function module of system in traditional methods. First, we used to assign the power of every module to different roles, and then these roles were assigned to some particular users. This access control technology based on roles lacks flexibility and cannot make to measure; it makes it very inconvenient for the user who has multiple roles in web application system. Furthermore, it will bring extra work in system expansion due to interaction effect of authority manage and business function. To improve the above- mentioned defect, this thesis puts forward methods of developing web system by integrating frameworks of Struts2, Spring3 and Hibernate4. It validates the log-in and permission of access for every request by the permission-interceptor. In this way, the web system can control user access powers in fine grain and Enhance the flexibility and expandability of system.

Key words: RBAC; access control matrices; interceptor; spring; Struts2; Hibernate

随着电子商务的发展, 基于 Internet 的 web 应用服务及其信息资源呈爆炸式增长, 使得人们对访问控制技术要求越来越高. 如何对信息系统进行安全, 有效的访问控制是所有信息系统的共同要求. 当前, 在企业信息系统中对资源的访问控制技术一般有三种: 自主访问控(DAC)、强制访问控制(MAC)和基于角色的访问控制[1](RBAC).

DAC 与 MAC 都是主体和访问客体直接发生关系, 根据主体与客体的所属关系来决定主体对客体的访问权, 它的优点是管理集中, 但其实现工作量大、不便于管理, 不适用于主体或客体经常更新的应用环境; RBAC 是一种可扩展的访问控制模型, 通过引入角色对主体与客体进行解耦, 简化了授权操作和安全管理, 它是目前公认的解决大型企业的统一资源访问控制的有效方法. 但它仍然存在以下三个方面的缺陷: 1.无法做到对特定客体分配的“量身定制”和“微调”. 由于主

① 收稿时间:2013-09-06;收到修改稿时间:2013-09-29

体对客体的访问权限完全由角色决定, 在不定义新的角色的条件下, 我们无法做到对某个主体授予特定的权限, 或从某些主体收回对特定客体的访问权限; 而如果重新定义角色, 则会引起拥有该角色的所有主体权限发生连锁效应. 2.多角色用户使用系统不方便. 对于具有多角色的主体来说, 要想同时进行不同角色的工作时, 就必须要不断地切换角色, 特别是这些角色权限存在相互排斥时, 有时候甚至要不断地退出系统与登录系统. 3.不便于系统功能扩展. 当给系统增加新的功能时, 除了实现新功能模块外, 还需要对角色验证逻辑进行重新定义, 当有大量角色时, 系统的复杂性会大增, 给开发人员带来额外工作量.

鉴于以上访问控制技术存在的缺陷. 本文提出基于双矩阵访问控制技术, 通过两个访问矩阵使得系统既能进行粗粒度访问控制管理, 又能进行细粒度访问控制管理. 很好地解决以上的技术缺陷. 增强权限管理的灵活性与系统的可扩展性.

1　双矩阵访问控制模型

本文提出采用双矩阵访问控制的思想, 它结合了以上三种技术的优点. 在权限分配时采用角色授权与主体直接授权相结合的方式, 在权限验证时采用角色权限矩阵与主体特权矩阵相结合的方式. 如图 1 所示:

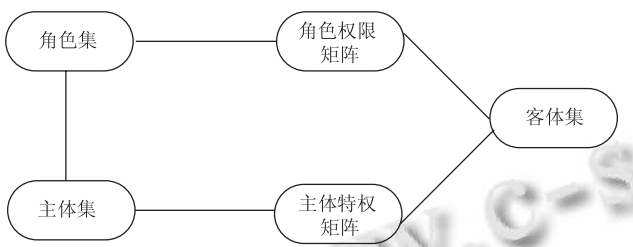

图 1　访问控制模型

主体特权矩阵(见图 2)中的行标代表主体集(用户), 列标代表客体集, 对应坐标点表示权限, 其值有 true, false, 空等三种状态, 分别表示许可、拒绝、未定义. 角色权限矩阵(见图 3)的行标代表角色集, 列标代表客体集, 对应的坐标点表示权限, 其值有 true, 空 两种状态, 分别表示许可、拒绝. 在权限验证时, 主体特权矩阵的优先级要高于角色权限矩阵,系统首先从主体特权矩阵中查找相关权限规则, 如果存在权限值 true 或 false, 则按这个权限规则执行, 而不再去理会角色权限矩阵中的权限值, 如果没有相关权限规则(记录为空), 则从角色权限矩阵中查找相关权限记录, 如果在角色权限矩阵对应的数据表中能找到相应记录, 则表示权限许可, 否则系统将拒绝主体对客体的访问请求. 一个主体如果拥有多个角色, 则以多个角色的权限矩阵的并集来验证是否允许访问. 只要该主体所拥有的角色中有一角色拥有可访问权限, 则权限验证结果就给予许可授权.

	P1	P2	P3	……	Pm
user1	true	false	--	……	false
user2	--	--	true	……	--
use3	true	--	true	……	--
……	……	……	……	……	……
useN	true	--	true	……	true

图 2　主题特权矩阵

	P1	P2	P3	……	Pm
role1	true	--	--	……	true
role2	--	--	true	……	--
role3	--	true	true	……	true
……	……	……	……	……	……
roleN	true	--	true	……	true

图 3　角色权限矩阵

在权限分配时, 首先对系统的角色集进行权限分配, 将各个角色赋予相应的权限, 生成角色权限矩阵, 然后将角色分配给各个主体. 另外, 如果对特定用户需要进行权限“微调”, 同时却不改变角色权限, 则直接在主体特权矩阵中对用户进行授权或权限回收.

通过这种双矩阵模型, 就能解决 RBAC 所存在的前两方面的缺陷. 为了解决第三方面的缺陷, 本文通过将权限管理与系统其它业务逻辑完全分离的方法加以解决, 两者的逻辑互相独立, 不存在任何依赖关系. 这样以来, 在系统进行功能扩展时, 就不会影响权限模块, 而改变权限模块也不会影响业务模块.

2　访问控制模块的分析设计

2.1 权限矩阵表的分析设计

在 Web 系统中, 每一次的访问请求都是一次对 Action 对象的方法调用, 所有的 Action 方法构成了客体集. 依据前述模型, 在权限矩阵中我们将 Action 与 Method 名作为列标, 将用户名或角色名作为行标. 在

用户与客体集之间创建一张特权表以存放特权矩阵的权限值, 其中行标由用户集构成, 列标由客体集构成; 在角色集与客体集之间通过多对多的关系建立角色权限表用以存放角色权限值, 其中行标由角色集构成, 列标由客体集构成.

本文采用 Struts2.3.4, Spring3.2, Hibernate4.1 技术框架[2] [3] 进行 WEB 开发. 由 Struts2 工作原理[4] 知道, Struts2 的前置拦截器位于所有应用层, 业务层, 持久层的最前面. 任何请求首先都要通过 Struts2 的前置拦截器, 通过之后才能继续调用内部应用层的 Action 方法; 如果一个请求被拦截器拒绝, 则它无法进入到系统的内层. 据此, 我们利用 Struts2 框架配置一个权限验证拦截器[5],它充当权限矩阵规则的执行者. 拦截器与系统其它业务模块之间是相互独立的, 不存在依赖关系. 当访问矩阵中权限规则许可, 则拦截器放行, 否则就拒绝执行相应 Action 方法, 从而实现访问控制矩阵对业务功能操作的控制.

2.2 权限分配与验证过程分析

在进行权限分配时, 采用角色权限分配与用户特权分配的混合模式. 角色作为权限组合的概念参与权限分配, 它是粗粒度的管理模式. 用户特权分配是细粒度管理模式, 它的最小分配单位是每一个 Action 方法的执行权限. 在系统运行期间, 采用这种混合模式, 大大提高了系统权限分配的灵活性. 进行粗粒度权限分配时, 可以采用角色权限分配模式, 在进行权限微调或细粒度管理时可采用用户特权分配方式.

在进行权限验证时, 所有对系统的访问请求, 除了注册与登录请求之外, 其它都必须经过权限验证拦截器(PermissionInterceptor)进行拦截验证, 只有拥有合法权限的用户才允许访问系统相关页面. 该拦截器位于系统所有拦截器的最前端, 这样可以把非法请求拦截在系统的最外层, 只有合法请求被放行后才能进入到系统, 再经过其它的一系列默认拦截器后才能到达系统的核心 Action.

2.3 访问时序图分析

访问时序图见图 4, 它通过拦截器与 Struts2 框架的 ActionInvocation 对象的递归调用来实现的: 1.当用户请求到达服务器时, Web 容器将请求转给 Struts2 框架, 框架会创建一个 ActionInvocation 对象, 并利用它来调用权限验证拦截器(PermissionInterceptor). 2.权限验证拦截器向权限验证器对象 PowerValidate 发送消息, 要求进行权限验证, PowerValidate 首先检查用户是否

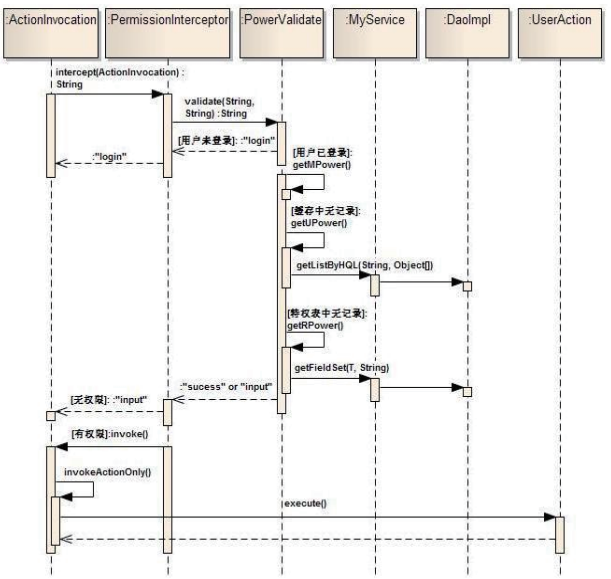

图 4 访问拦截时序图

已经登录, 如果未登录, 则向拦截器返回“login”, 表示用户未登录; 如果用户已登录, 则权限验证器调用自身的 getMPower()方法, 从缓存中查找相应权限, 如果找到, 则返回权限规则结果(“success”表示允许授权, “input”表示拒绝授权), 否则调用 getListUPower()方法, 从用户特权表中查找权限值, 如果找到则返回相应权限值. 如果用户特权矩阵中无相应权限记录, 则接着调用自身的方法 getRPower(), 从角色权限矩阵表中查找相应权限的记录, 如果找到则表示允许访问, 否则表示拒绝访问. 3.访问控制拦截器依据 PowerValidate 对象返回的字符串进行访问控制许可判断, 并向 ActionInvacation 返回相应字符串, 如果是拒绝访问, 则拦截器向 ActionInvocation 返回“input”, 让页面返回请求前的页面, 并在页面中显示提示信息. 如果 PowerValidate 验证结果是允许访问, 则拦截器向 ActionInvocation 发送消息,调用回调方法 invoke(). 4.ActionInvocation 调用 interceptor 方法处理其它拦截器(如果还有待执行的拦截器,图中未列出). 5.ActionInvocation 调用 invokeActionOnly()方法, 执行用户请求的 Action 的方法. 并执行相关业务逻辑.

3 系统框架的整合实施

3.1 准备工作

开发环境采用开源的 Eclipse 集成开发环境, 下载

Struts2, Hibernate, Spring 等相关类包, 创建 Web 工程项目, 把 Struts2、Spring、Hibernate 的相关类库加入到工程的 lib 目录中.

3.2 Web 与 Struts2, Spring 框架的整合

为了在Web 应用中使用 Struts2 来处理请求, 同时启动 Spring 框架的各项服务需要修改 web.xml 文件, 加入如下代码(图 5 所示)

```
<context-param>
<param-name>contextConfigLocation</param-name>
<param-value>WEB-INF/classes/applicationContext.xml
</param-value>
</context-param>
<listener>
<listener-class>
org.springframework.web.context.ContextLoaderListener
</listener-class>
</listener>
<filter>
<filter-name>struts2</filter-name>
<filter-class>
org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecut
eFilter
</filter-class>
</filter>
<filter-mapping>
<filter-name>struts2</filter-name>
<url-pattern>/*</url-pattern>
</filter-mapping>
<listener>
<listener-class>
org.springframework.web.context.request.RequestContextListener
</listener-class>
</listener>
```

图 5　web.xml 时序

3.3 整合 Hibernate 与 Spring

在 WEB 系统开发中, 应用层逻辑与数据持久层要尽量互相独立, 不要混杂在一起. 应用层一些复杂的业务逻辑可以放到专门的处理类中来. 这些用来专门处理应用层复杂逻辑的类我们称之为服务层. 当服务层需要访问持久层的数据时, 由 Spring 来提供相应的数据访问对象, 再由该对象调用 Hibernate 框架来进行数据库访问的底层操作.

Spring 与 Hibernate 整合时, 它们的配置文件可以合二为一, 也可以各自独立存在. 本文采用后者, 所以需要在配置文件 applicationContext.xml 中加入 ygbDataSource 与 sessionFactory 两个 Bean, 它可以确定数据源与 Hibernate 配置文件的位置. 在系统中, 持久层的 SessonFactory、daoImpl、服务层的 Service、SpringContext、以及拦截器 pessioninterceptor 等都是单例类[6], 即在系统运行期间只有一个实例(在应用系统启动时实例化), 所以必须把它们设置成无状态的类, 不能在其中存在有状态的成员属性. 而权限验证类 PowerValidate 由于它存放着登录用户的权限信息, 故我们设置 scope="session", 而各种 Action 类是有状态的, 它们在 Spring 配置文件中必须设置成 scope="prototype". 该文件配置内容较多, 在此不再详细举例. applicationContext.xml 部分代码见(图 6 所示).

```
<bean id="ygbDataSource"
class="org.apache.commons.dbcp.BasicDataSource"destrov-
<property name="driverClassName" value="com.mysql.jdbc.Driver" />
<property name="url" value="jdbc:mysql://localhost:3306/qtdd" />
<property name="username" value="root" />
<property name="password" value="881227" /></bean>
<bean id="sessionFactory"
class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
<property name="dataSource" ref="ygbDataSource"/>
<property name="configLocation"
value="classpath:hibernate.cfg.xml"/></bean>
<bean name="springContext" class="com.business.SpringContext"></bean>
<bean name="daoImpl" class="com.dao.DaoImpl">
<property name="sessionFactory" ref="sessionFactory" /></bean>
<bean name="myService" class="com.business.MyService">
<property name="daoImpl" ref="daoImpl" /></bean>
<bean name="permissionInterceptor"
class="com.interceptor.PermissionInterceptor"></bean>
<bean name="powerValidate" class="com.interceptor.PowerValidate"
scope="session">
<property name="myService" ref="myService" /></bean>
<bean name="pageResult" class="com.business.PageResult"
scope="prototype">
<property name="myService" ref="myService" /></bean>
<bean name="roleAction" class="com.action.RoleAction"
parent="pageResult"scope="prototype"></bean>
<bean name="powerAction" class="com.action.PowerAction"
parent="pageResult" scope="prototype"></bean>
<bean name="userAction" class="com.action.UserAction"
parent="pageResult" scope="prototype"></bean>
<bean name="loginAction" class="com.action.LoginAction"
parent="pageResult" scope="prototype"></bean>
</beans>
```

图 6　application 部分代码

Hibernate 框架实现了模型中对象与数据库表之间的映射, 它通过映射文件*.hbm.xml 来确定, 并由 SessionFactory 对象来管理. 而 SessionFactory 对象需要在 hibernate.cfg.xml 文件中进行配置, 它也可以在 applicationContext.xml 文件中配置; 本文将数据源放在 applicationContext.xml 中配置, 而其它则在 hibernate.cfg.xml 中配置, hibernate.cfg.xml 配置代码见图 7.

```
<hibernate-configuration>
<session-factory>
<property name="hibernate.dialect">
org.hibernate.dialect.MySQLDialect</property>
<property name="hibernate.show_sql">false</property>
<property name="hibernate.hbm2ddl.auto">update</property>
<property name="current_session_context_class">thread</property>
<property name="hibernate.format_sql">true</property>
<mapping resource="com/persist/entity/User.hbm.xml" />
<mapping resource="com/persist/entity/Role.hbm.xml" />
<mapping resource="com/persist/entity/Power.hbm.xml" />
<mapping resource="com/persist/entity/Privilege.hbm.xml" />
</session-factory>
</hibernate-configuration>
```

图 7　hibernate.cfg.xml 配置代码

3.4 拦截器配置[7]

在 struts.xml 文件中, 先配置用于对用户请求进行

拦截处理的拦截器(PermissionInterceptor), 再自定义一个拦截器堆栈, 然后将它设置成默认, 最后不要忘了将登录与注册模块设置成不被拦截(只需将它们各自的 Action 配置成框架提供的默认拦截器堆栈即可). 配置代码如图 8 所示.

```
<interceptors>
<interceptor name="permissionInterceptor" class="permissionInterceptor" />
<interceptor-stack name="myDefaultInterceptorStack">
<interceptor-ref name="permissionInterceptor" />
<interceptor-ref name="defaultStack"></interceptor-ref>
 </interceptor-stack>
</interceptors>
<default-interceptor-ref name="myDefaultInterceptorStack" />
<global-results>
 <result name="error">/error.jsp</result>
<result name="login">/login.jsp</result>
</global-results>
<action name="loginAction" class="loginAction">
<result name="success">index.jsp</result>
<result name="input">login.jsp</result>
<result name="invalid.token">login.jsp</result>
<interceptor-ref name="defaultStack"></interceptor-ref>
</action>
```

图 8 struts.xml 部分代码

4 数据库表的结构及关联关系说明

与访问控制相关的表有: USER, ROLE, POWER, PRIVILEGE, USER_ROLE, ROLE_POWER 等五个数据表, 其中 USER 用于存放所有用户, ROLE 用于存放系统中的所有角色, POWER 用于存放系统中所有功能权限字符串(Action 方法名), PRIVILEGE 用于存放所有用户的特权矩阵值, USER_ROLE 用于存放用户与角色的对应关系, ROLE_POWER 用于存放角色与权限的对应关系. 数据库表的字段设计以及字段类型见如图 9 所示.

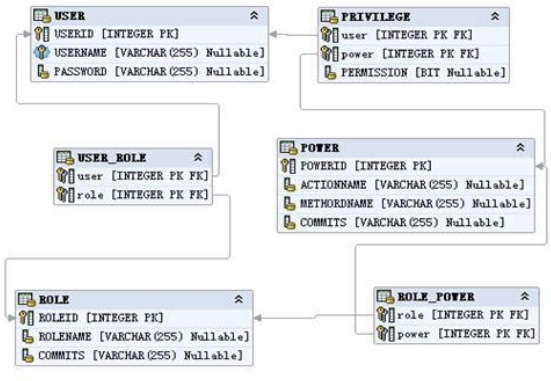

图 9 权限管理所涉数据表

其中 ROLE 表与 POWER 表中的 COMMITS 字段表示备注, 可以为空, 其它字段均不能为空, USERID, ROLEID, POWERID? 三个字段为自增的整型, 它作为记录的标识字段. USER_ROLE 表决定所有用户与所有角色之间的对应关系; ROLE_POWER 决定所有角色与所有权限资源的对应关系; Privilege 与 User、Power 之间都是多对一的关系. 权限资源 Power 表中字段 actionName 与 methordName 分别用来存放系统中 Action 类的全名与方法名,不能为空. Privilege 表中存放着用户特权值, 它以 User 与 Power 的主健作为联合主键, 字段 permission 值为 true 表示允许访问, 为 false 时, 表示拒绝访问.

用户的权限请求首先要在 PRIVILEGE 中查找权限, 如果找到则按这个规则执行, 不再访问 ROLE_POWER 表; 如果找不到记录, 则到该用户所属的角色集所对应的权限表 ROLE_POWER 中找(通过 USER_ROLE 与 ROLE_POWER 两表的内连接进行查找), 在 ROLE_POWER 中如果存在与用户及访问权限相应记录则表示该用户拥有相应权限, 如果不存在则表示无相应权限.

5 系统总体框架

系统类图依赖关系如图 10 所示. 本系统表现层由 JSP 页构成; 应用层包括访问控制拦截器 PermissionInterceptor、系统框架中其它拦截器以及实现不同功能的 Action 类; 服务层包括 MyService、SpringContext、PowerValidate、PageResult; 数据持久层: daoImpl、SessionFactory; 与访问控制相关的持久类有: 用户类(User)、角色类(Role)、客体类(Power)、特权类(Privilege)等; 最底层是数据库表. 当请求到来时数据访问顺序依次自上向下: 表现层、应用层、服务层、持久层、数据库. 第一层与第二层之间由 Struts2

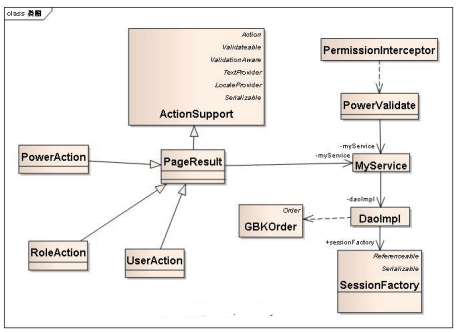

图 10 权限管理主要类图

框架负责协调关联, 由配置文件 struts.xml 文件决定,

第二层到第四层之间通过 Spring 框架建立关联, 由 spring 配置文件决定, 第四层与数据库之间由 Hibernate 框架的映射文件*.hbm.xml 决定. 系统中应用层依赖于服务层, 而服务层要依赖持久层, 各层之间的调用要求分层次进行.

6 系统特点分析

6.1 系统优点

本系统相比采用传统单一访问控制技术的 web 系统而言, 由于采用了基于角色访问控制与基于主体特权访问控制的双矩阵混合模式, 大幅度提高了系统权限管理的灵活性与机动性. 同时由于它把系统权限管理验证与业务功能相分离, 通过客体名(actionName 与 methodName), 把两者关联起来. 从而做到了让系统的功能操作与权限管理真正各自独立. 从而把权限管理从系统其它业务逻辑中解藕出来, 增强了系统的可扩展性. 我们甚至可以在系统还没开发前就事先约定好这个权限字符串, 让程序员各自遵守约定去开发不同的模块. 本系统能有效解决本文开头提出的 RBAC 所存在的三大技术缺陷问题.

6.2 系统缺点

由于要对每个请求进行权限验证, 必然会对系统的性能(访问速度)上带来负面影响. 尤其是对用户多, 复杂度高的系统, 权限数据的频繁访问会对数据库带来不少的负担,也给应用服务器带来负担.

6.3 解决方法

由于一般用户在访问系统时往往只是系统中的少数页面, 并不是系统的所有页面,类似于程序运行的局部性原理[8], 所以本系统将权限验证类 PowerValidate 的生命周期设置为 scope="session", 它的一个属性 permissionMap 用来存放用户访问过的请求的权限值. 验证器每一次从数据库中取出权限值后都会将它保存在这个缓存中, 下次访问时就直接从这个 permissionMap 中查找. 只有在找不到相关权限值时才转到数据库中查找. 这样, 就能将权限验证带来的负面影响降到最低. 基本上不会影响系统的性能, 既不会给应用服务器增加很大负担, 也不会给数据库服务器增加很大负担. 它能很好地解决因权限频繁验证带来的性能问题.

6.4 系统运行测试

依据不同的用户, 角色, 访问权限, 建立拦截器实验分析数据表, 如图 11 所示. 笔者以该表的不同条件逐项进行拦截实验, 实验结果符合预期. 表中 null 表示无相关记录, *表示包括所有可能的情况.

主体	客体名	主体 特权	主体 主角	角色 权限	访问结果
未登录	*	*	*	*	转登录页
user1	power1	√	*	*	允许访问
user2	power2	×	*	*	禁止访问
user3	power3	null	√	√	允许访问
user4	power4	null	√	null	禁止访问
user5	power5	null	null	--	禁止访问

图 11　拦截实验条件与结果

拦截结果有三种情况: 未登录(见图 12); 已登录却无权限(图 13); 已登录且有权限时, 直接进入系统内部执行相关 Action 方法后返回目标页面.

图 12　拦截结果(未登陆)

图 13　拦截结果(无权限)

(下转第 56 页)

表 3 系统正确回答情况

编号	问题数	正确答案数	正确率	平均耗时 ms
1	20	18	90.0%	122
2	26	24	92.3%	168
3	33	30	90.9%	160
4	36	34	94.4%	145
5	40	37	92.5%	137
6	22	21	95.5%	165
7	19	17	89.5%	186
平均			92.2%	153.14

方面做了深入的改进. 基于 Lucene.Net 的搜索方案可以高效快速的建立索引, 这种索引不仅仅局限于数据库, 也包含了我们日常工作中经常遇到的文件格式. 下一步工作是结合分布式框架 Hadoop, 建立分布式数据检索系统. 基于潜在语义分析的问题和答案句子相似度计算方法, 对词的同义和多义现象有较好的处理效果, 答案提取实验结果也说明了这一点, 但由于 LSA 的基础是基于词频统计方法, 没有考虑词在答案句子中的位置分布对答案句子的影响, 而且 LSA 方法本身也存在计算量大、所需存储空间大等缺点, 因此该方法还存在一定的局限性. 进一步的研究将结合词的位置分布和词的语义关系进行答案句子和答案实体的抽取.

参考文献

1 余正涛,樊孝忠.基于潜在语义分析的汉语问答系统答案提取.计算机学报,2006,29(10):1889−1893.

2 刘江平,薛河儒. 基于 Web 的智能答疑系统的设计和实现. 网络与通信,2012,28(3):121−123.

3 李园伟,宁可为,王炜.分布式自动答疑系统.计算机系统应用, 2012,21(7):22−29.

4 孔维亭,闫宏印.基于 Lucene 的自动答疑系统的设计.电脑开发与应用,2012,25(4):32−36.

5 吴秀梅.基于潜在语义分析和最大墒的中文情感分析研究[学位论文]. 北京:北京交通大学,2011.

6 王丛林.在线自动答疑系统设计与开发的研究[学位论文]. 长春:东北师范大学,2010.

7 Ng.A. Indexing and searching image files: using Lucene.NET long with open-source libraries. Dr. Dobb's Journal, 2008, 33(10): 52−55.

8 Zhang CH. Research and implementation of full-text etrieval system using compass based on Lucene.Advances in Intelligent Systems and Computing, 2013, 181: 349−356.

9 Su TY. Chinese full-text retrieval system based on Lucene. Computer Engineering, 2007, 33(23): 94−96.

10 Yin PP. Evaluation of literature frontier based on latent semantic analysis. Proc. the 2012 IEEE Symposium on Robotics and Applications (ISRA). 2012. 403−406.

11 顾益军,樊孝忠.中文停用词表的自动选取.北京理工大学学报,2005,25(4):337−341.

(上接第 96 页)

参考文献

1 张世明,杨寅春.基于角色的访问控制技术在大型系统中的应用.计算机工程与设计,2006,27(19):3723−3725.

2 陈辉,赵洪升,张艳春. Struts+Spring+ Hibernate 框架的整合实现.河南大学学报(自然科学版),2010,40(6):642-645.

3 曹渠江,陈真. Struts2 框架整合 Spring 框架在文件上传下载中的应用.上海理工大学学报,2009,31(2):169−172.

4 孙更新,宾晟,宫生文.Java 程序开发大全.北京:中国铁道出版社,2010:214−215.

5 闫宏印,张卫争,刘超慧.开源框架下 Web 应用分层的设计与实现.计算机工程与设计,2008,29(23):6023−6028.

6 陈翠娥.Java 单例模式应用研究.长沙民政职业技术学院学报,2010,17(1):114−116.

7 刘艳春,洪晓慧.Struts2 框架核心配置文件的研究与应用. 计算机技术与发展,2013,23(2):77−81.

8 王恒娜.访问局部性原理在 Cache 存储系统中的作用.安徽大学学报(自然科学版),2005,29(1): 27−3