CIDC系統工程結構分析

本系统是基于spring MVC的，在系统中还用到struts menu，appFuse等技术。

一下开始对系统进行分析：

首先在web.xml配置相关文件

<context-param>

<param-name>csstheme</param-name>

<param-value>niftycube</param-value>

</context-param>

加载默认的CSS文件

<context-param>

<param-name>javax.servlet.jsp.jstl.fmt.localizationContext</param-name>

<param-value>ApplicationResources</param-value>

</context-param>

加载默认的国际化资源文件

<context-param>

<param-name>javax.servlet.jsp.jstl.fmt.fallbackLocale</param-name>

<param-value>en</param-value>

</context-param>

国际化资源文件（有疑问）

<context-param>

<param-name>contextConfigLocation</param-name> <param-value>/WEB-INF/applicationContext-\*.xml,/WEB-INF/security.xml</param-value>

</context-param>

加载spring配置文件

<filter>

<filter-name>securityFilter</filter-name>

<filter-class>org.acegisecurity.util.FilterToBeanProxy</filter-class>

<init-param>

<param-name>targetClass</param-name>

<param-value>org.acegisecurity.util.FilterChainProxy</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>securityFilter</filter-name>

<url-pattern>/j\_security\_check</url-pattern>

</filter-mapping>

<filter-mapping>

<filter-name>securityFilter</filter-name>

<url-pattern>/dwr/\*</url-pattern>

</filter-mapping>

<filter-mapping>

<filter-name>securityFilter</filter-name>

<url-pattern>\*.html</url-pattern>

</filter-mapping>

<filter-mapping>

<filter-name>securityFilter</filter-name>

<url-pattern>\*.jsp</url-pattern>

</filter-mapping>

这个定义了安全控制过滤链，这个过滤器一定要定义在MVC转发器的前边。通过该安全控制来实现用户权限的控制(后面细讲)

<filter>

<filter-name>cacheFilter</filter-name> <filter-class>com.opensymphony.oscache.web.filter.CacheFilter</filter-class>

</filter>

是对OSCache的配置，是对java 缓存流的配置

<filter>

<filter-name>cacheFilter</filter-name>

<filter-class>com.opensymphony.oscache.web.filter.CacheFilter</filter-class>

</filter>

(不很了解)

————————————————————————————————————

<filter>

<filter-name>encodingFilter</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>UTF-8</param-value>

</init-param>

<init-param>

<param-name>forceEncoding</param-name>

<param-value>true</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>encodingFilter</filter-name>

<url-pattern>/j\_security\_check</url-pattern>

</filter-mapping>

<filter-mapping>

<filter-name>encodingFilter</filter-name>

<url-pattern>/dwr/\*</url-pattern>

</filter-mapping>

<filter-mapping>

<filter-name>encodingFilter</filter-name>

<url-pattern>\*.html</url-pattern>

</filter-mapping>

<filter-mapping>

<filter-name>encodingFilter</filter-name>

<url-pattern>\*.jsp</url-pattern>

</filter-mapping>

对字符过滤器的配置。

<filter>

<filter-name>hibernateFilter</filter-name> <filter-class>org.springframework.orm.hibernate3.support.OpenSessionInViewFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>hibernateFilter</filter-name>

<url-pattern>\*.html</url-pattern>

</filter-mapping>

它是一个Servlet2.3过滤器，用来把一个Hibernate Session和一次完整的请求过程对应的线程相绑定。目的是为了实现"Open Session in View"的模式。例如： 它允许在事务提交之后延迟加载显示所需要的对象。(有时间仔细学习)

<filter>

<filter-name>rewriteFilter</filter-name>

<filter-class>org.tuckey.web.filters.urlrewrite.UrlRewriteFilter</filter-class>

<init-param>

<param-name>logLevel</param-name>

<param-value>commons</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>rewriteFilter</filter-name>

<url-pattern>/\*</url-pattern>

<!--dispatcher>REQUEST</dispatcher>

<dispatcher>FORWARD</dispatcher-->

</filter-mapping>

UrlRewriteFilter是一个用于改写URL的Web过滤器，类似于Apache的mod\_rewrite。适用于任何Web应用服务器（如Resin，Orion，Tomcat等）。其典型应用就把动态URL静态化，便于搜索引擎爬虫抓取你的动态网页。

<filter>

<filter-name>sitemesh</filter-name> <filter-class>com.opensymphony.module.sitemesh.filter.PageFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>sitemesh</filter-name>

<url-pattern>/\*</url-pattern>

<!--dispatcher>REQUEST</dispatcher>

<dispatcher>FORWARD</dispatcher-->

</filter-mapping>

(<http://blog.sina.com.cn/s/blog_4d429a290100d62c.html>)

OS(OpenSymphony)的SiteMesh是一个用来在JSP中实现页面布局和装饰（layout and decoration）的框架组件，能够帮助网站开发人员较容易实现页面中动态内容和静态装饰外观的分离。(有詳細講解)

<filter>

<filter-name>exportFilter</filter-name> <filter-class>org.displaytag.filter.ResponseOverrideFilter</filter-class>

</filter>

對文件進行導出的過濾器，以後了解(………………)

<filter>

<filter-name>clickstreamFilter</filter-name> <filter-class>com.opensymphony.clickstream.ClickstreamFilter</filter- class>

</filter>

<filter-mapping>

<filter-name>clickstreamFilter</filter-name>

<url-pattern>\*.html</url-pattern>

</filter-mapping>

<listener> <listener-class>com.opensymphony.clickstream.ClickstreamListener</listener-class>

</listener>

配置**Clickstream，**用來跟蹤正在站點上訪問的用戶所到達的具體位置細節的工具，它允许你跟踪访问你的站点的“点击流”或者“传输路径”

<listener>

<listener-class>net.sf.navigator.menu.MenuContextListener</listener-class>

</listener>

配置struts的菜單監聽器。

<listener> <listener-class>org.springframework.web.util.IntrospectorCleanupListener</listener-class>

</listener>

spring 中对它的描述如下：它是一个在web应用关闭的时候,清除JavaBeans Introspector的监听器.web.xml中注册这个listener.可以保证在web 应用关闭的时候释放与掉这个web   应用相关的class loader 和由它管理的类如果你使用了JavaBeans Introspector来分析应用中的类,Introspector 缓冲中会保留这些类的引用.结果在你的应用关闭的时候,这些类以及web 应用相关的class loader没有被垃圾回收.不幸的是,清除Introspector的唯一方式是刷新整个缓冲.这是因为我们没法判断哪些是属于你的应用的引用.所以删 除被缓冲的introspection会导致把这台电脑上的所有应用的introspection都删掉.需要注意的是,spring 托管的bean不需要使用这个监听器.因为spring它自己的introspection所使用的缓冲在分析完一个类之后会被马上从javaBeans Introspector缓冲中清除掉.应用程序中的类从来不直接使用JavaBeans Introspector.所以他们一般不会导致内部查看资源泄露.但是一些类库和框架往往会产生这个问题.例如:Struts 和Quartz.单个的内部查看泄漏会导致整个的web应用的类加载器不能进行垃圾回收.在web应用关闭之后,你会看到此应用的所有静态类资源(例如单 例).这个错误当然不是由这个类自身引起的.

<servlet>

<servlet-name>dwr-invoker</servlet-name>

<servlet-class>uk.ltd.getahead.dwr.DWRServlet</servlet-class>

<init-param>

<param-name>debug</param-name>

<param-value>true</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>dwr-invoker</servlet-name>

<url-pattern>/dwr/\*</url-pattern>

</servlet-mapping>

定義了dwr servlet