# 使用Acegi进行身份认证

Acegi是专门为Spring Web应用提供[安全](http://safe.it168.com/)保护的开源框架，它通过配置的方式就可以对业已存在的应用实施安全控制。在Acegi实施安全控制之前，必须获取操作者的身份，并进一步获知用户的权限，这样Acegi才可能对应用资源实施安全控制。在本文中，我们将介绍Acegi如何对操作者进行身份认证的内容。   
  
**将Acegi集成到Web应用程序中**   
  
Acegi 通过多个不同用途的Servlet过滤器截取HTTP请求，实施访问安全的控制。按照传统的方式，我们应该在web.xml配置文件中通 过<filter>定义Servlet过滤器并使用<filter-mapping>元素定义过滤器对应的URL匹配模式。   
  
由于这些过滤器需要Spring容器中其它Bean的支持完成访问权限控制，最原始的方式是在过滤器中通过硬编码的方式访问Spring容器中的Bean：   
  
WebApplicationContext ctx = WebApplicationContextUtils.getWebApplicationContext(servletContext);   
FooBean foo = (FooBean)ctx.getBean(“foo”);   
  
这种硬编码的方式显然不太方便，如何让过滤器既能够正确工作又能享受到Spring IoC的好处呢？Acegi通过“过滤器代理”来解决这个问题。   
  
**代理Acegi的过滤器**  
org.acegisecurity.util.FilterToBeanProxy 使用Spring容器中的Bean代理过滤器，FilterToBeanProxy是一个标准的Servlet过滤器。在内部 FilterToBeanProxy通过WebApplicationContext访问Spring容器，并将过滤器处理逻辑委托给一个Spring容 器中的Bean。这样，我们就可以在web.xml中定义FilterToBeanProxy，声明过滤器匹配的URL模式，并在Spring容器中充分 利用IoC进行定义委托的Bean。   
  
FilterToBeanProxy支持两个初始化参数：  
  
l targetClass：委托Bean的全限定类名，委托类必须实现过滤器接口。你必须在Spring容器中至少定义一个该类的Bean，否则将抛出异 常。当Spring容器中拥有多个该委托类的Bean时，FilterToBeanProxy选择第一个委托类的Bean（这种情况应该避免）；   
  
l targetBean：当然你也可以通过一个Bean名指定委托的Bean。  
图 1说明了Acegi是如何通过FilterToBeanProxy将Servlet容器和Spring容器结合起来共同定义一个功能齐备的Servlet过滤器的：   
图 1 通过过滤器代理将Servlet容器和Spring容器结合起来   
  
org.acegisecurity.securechannel.ChannelProcessingFilter 是一个Acegi的过滤器，它负责处理安全通道的转换。如果直接在web.xml中定义ChannelProcessingFilter，我们将很难在不 编写代码的情况下，使其引用Spring容器中的Bean。借助FilterToBeanProxy的帮助，我们就可以放心地在Spring容器中配置 ChannelProcessingFilter，得到一个完整可用实例。当然ChannelProcessingFilter是需要服务于特定URL请 求的，而过滤器URL映射只能在Servlet容器的web.xml中定义，这个工作由FilterToBeanProxy对应的<filter- mapping>元素完成。   
  
如果我们将Servlet容器和Spring容器比作牛郎和织女，那么 FilterToBeanProxy无疑就是鹊桥了。事实上，Acegi本身并不依赖于FilterToBeanProxy，它是一种充分利用 Spring容器依赖注入好处配置Servlet过滤器的一个方法，正因为如此，所以Acegi社区中有很多开发者建议 FilterToBeanProxy应该从Acegi中剥离出来，将其加入到Spring的核心类库中。

**使用代理过滤器链对处理HTTP请求**   
  
Acegi通过众多的过滤器完成不同[安全](http://safe.it168.com/)控 制的任务，在前面我们提到了三个过滤器：HttpSessionContextIntegrationFilter、 ExceptionTranslationFilter和ChannelProcessingFilter。除此以外，还有LogoutFilter、 AuthenticationProcessingFilter等不下10个的过滤器。   
  
当需要配置多个Servlet过滤器时，虽然 我们可以通过FilterToBeanProxy分别进行配置，但这将导致冗长难看的web.xml，同时还需要小心谨慎地通过过滤器的配置顺序保证它们 的调用顺序。为了解决这个问题，Acegi在0.8版本中添加了一个org.acegisecurity.util.FilterChainProxy。 FilterChainProxy可以同时指定多个过滤器并将它们组成一个过滤器链，而FilterToBeanProxy只要将代理目标设置为 FilterChainProxy就可以了。来看下面的配置：   
  
代码清单 1 web.xml

<web-app>

<context-param> ①指定Spring配置文件

<param-name>contextConfigLocation</param-name>

<param-value>

classpath:applicationContext.xml,

classpath:applicationContext-acegi-plugin.xml

</param-value>

</context-param>

<filter>②将过滤器委托给FilterChainProxy

<filter-name>AcegiFilterChainProxy</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>③对所有的URL进行过滤

<filter-name>AcegiFilterChainProxy</filter-name>

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

</filter-mapping>

<listener>

<listener-class>

org.springframework.web.context.ContextLoaderListener

</listener-class>

</listener>

…

</web-app>

在web.xml中我们首先需要定义Spring的配置文件，以便将Spring容器集成到Servlet容器中。紧接着，我们定义一个 FilterToBeanProxy的过滤器拦截所有URL请求并委托给FilterChainProxy进行处理。FilterChainProxy在 Spring容器中定义如下所示：   
  
代码清单 2 applicationContext-acegi-plugin.xml

<beans>

<bean id="filterChainProxy" class="org.acegisecurity.util.FilterChainProxy">①目标委托的Bean

<property name="filterInvocationDefinitionSource">

<value>

CONVERT\_URL\_TO\_UPPERCASE\_BEFORE\_COMPARISON

PATTERN\_TYPE\_APACHE\_ANT

②多个过滤器组成的过滤器链

/\*\*=channelProcessingFilter,httpSessionContextIntegrationFilter,logoutFilter

</value>

</property>

</bean>

<bean id="channelProcessingFilter" ③过滤器链中的过滤器(1)

class="org.acegisecurity.securechannel.ChannelProcessingFilter">

…

</bean>

<bean id="httpSessionContextIntegrationFilter" ④过滤器链中的过滤器(2)

class="org.acegisecurity.context.HttpSessionContextIntegrationFilter">

…

</bean>

…

</beans>

在①处定义的FilterChainProxy Bean是web.xml的FilterToBeanProxy的委托目标。FilterChainProxy通过 filterInvocationDefinitionSource定义多个相互链接的过滤器，如②所示。这些过滤器以逗号分隔，它表示web.xml中 定义的匹配URL(/\*)请求将分别通过channelProcessingFilter、 httpSessionContextIntegrationFilter及logoutFilter这三个过滤器的拦截处理。这个过滤器链通过过滤器对 应的Bean名字（如③和④）进行定义，过滤器都实现了javax.servlet .Filter接口。你完全可以参照实例方式定义多组滤器链，以便处理不同URL请求。不过，在一般情况下，你只要配置一个过滤器链就可以了。  
  
filterInvocationDefinitionSource属性由两类信息组成：其一是指令信息，如  
  
CONVERT\_URL\_TO\_UPPERCASE\_BEFORE\_COMPARISON 表示判断URL匹配时，首先将URL转变为大写的格式，如果设置为 CONVERT\_URL\_TO\_LOWERCASE\_BEFORE\_COMPARISON则表示比较前事先转换为小写的形式。而 PATTERN\_TYPE\_APACHE\_ANT表示使用Ant路径风格进行匹配URL的描述，如果不提供这个指令，Acegi使用正则表达式来解析 URL路径映射。

**身份认证管理**   
  
使用Acegi保护应用程序的第一步是根据用户提供的认证信息进行身份认证，以确定用户的身份，获取对应的权限信息，准备好Authentication。通过认证的Authentication拥有权限信息，它是Acegi进行后续[安全](http://safe.it168.com/)对象访问安全控制的依据。   
  
Acegi 对所有类型安全对象的访问控制机制基本相同：判断Authentication是否已经包含访问安全对象所需的权限。但Acegi进行身份认证的方式则多 姿多彩、五花八门，身份认证方式的多样性源于认证系统的多样性。由于篇幅所限，不可能面面俱到的逐一讲解所有的认证方式，我们将以最常见的基于数据库的认 证方式并对其它的认证方式进行简单的概述。   
  
**基于**[**内存**](http://product.it168.com/list/b/0205_1.shtml)[**存储**](http://storage.it168.com/)**用户信息的身份认证**   
  
如果你开发的系统仅有少数几个固定的用户，且用户信息管理的功能很简单，这时你可以将用户直接保存在配置文件中，在系统启动将使用户信息常驻于内存中。下面，我们就从这个最简单的身份认证方式出发开始学习Acegi实际应用的征程。   
代码清单 3 applicationContext-acegi-plugin.xml   
身份认证过滤器

<beans>

<bean id="filterChainProxy" class="org.acegisecurity.util.FilterChainProxy">

<property name="filterInvocationDefinitionSource">

<value>

…

/\*\*=authenticationProcessingFilter ①使用认证处理过滤器处理匹配的URL

</value>

</property>

</bean>

<bean id="authenticationProcessingFilter" ②认证处理过滤器

class="org.acegisecurity.ui.webapp.AuthenticationProcessingFilter">

②-1过滤器处理的URL

<property name="filterProcessesUrl" value="/j\_acegi\_security\_check"/>

<property name="defaultTargetUrl" value="/main.jsp"/> ②-2认证成功后转向的URL

②-3认证失败后转向的URL

<property name="authenticationFailureUrl" value="/index.jsp?login\_error=1"/>

</bean>

</beans>

首先，在①处通过FilterChainProxy定义了仅包含一个authenticationProcessingFilter的过滤器链。这 个过滤器链匹配于所有URL的请求，换句话说，authenticationProcessingFilter有机会对所有URL请求进行拦截处理，而在 这所有这些URL请求中，究竟哪个URL请求才是用户身份认证的请求呢？如果你心中也拥有这个疑问，则说明你正走在正确的理解问题的思路上。   
  
authenticationProcessingFilter 在②-1、2、3处，定义了一些URL，它们分别对应身份认证请求的URL、认证成功后转向的URL以及认证失败后转向的URL。其中②-1处的 filterProcessesUrl属性表示这个URL是用户提交身份信息并要求进行身份认证的HTTP请求（一般是一个包含用户名/密码的表单 HTTP POST请求），“/j\_acegi\_security\_check”是默认的设置，之所以在此显式给出设置值，是为了说明你可以根据实际需要进行调整。   
  
现在，authenticationProcessingFilter已经知道了申请进行用户身份认证URL，但它如何知道这个HTTP 请求分别通过什么参数代表用户名和密码呢？Acegi框架规定必须使用j\_username和j\_password作为用户名和密码的HTTP参数名。这 两个参数名是固定的，Acegi不允许对此进行配置，不过这种约定俗成而非事事配置的风格正渐渐成为流行的设计模式。   
  
我们来看一下提供用户名/密码输入表单的登录页面：   
代码清单 4 index.jsp：登录页面

<%@ page contentType="text/html;charset=UTF-8"%>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<html>

<body>

<c:if test="${not empty param.login\_error}"> ①

<font color="red">用户名或密码错误。</font>

</c:if>

②处理用户身份认证服务所对应的URL

<form name="form1" method="post" action="<c:url value="/j\_acegi\_security\_check"/>">

用户名：<input type="text" name="j\_username"/><br/>③用户名的参数名

密 码：<input type="password" name="j\_password"/><br/>④用户密码的参数名

<input type="submit" value="登录"/>

</form>

</body>

</html>

 ②处对应代码清单 3中②-1处的设置值，它表示处理用户身份认证服务所在的URL，而③和④是Acegi规定的承载用户名、密码的参数名。当这个表单提交后 authenticationProcessingFilter将从请求中抽取j\_username和j\_password参数的值，两者组成了代表请求 认证的用户信息，Acegi使用这个信息构造Authenticaion实例，并封装成SecurityContext放入到 SecurityContextHolder中，然后启动用户身份认证的流程。当认证成功后，页面转向代码清单 3中②-2处所定义的/main.jsp，如果认证失败则转向代码清单 3中②-3处所定义的/index.jsp?login\_error=1。由于登录失败后也转向index.jsp页面，所以代码清单 4的①处专门对此进行了处理，当发现请求参数包含login\_error时，打出一行提示登录失败的信息。  
  
现在，我们已经了解了 authenticationProcessingFilter从何处获取需要进行认证的用户信息，并对认证成功和失败后的转向进行了定义。那么 authenticationProcessingFilter究竟如何对用户信息（用户名/密码）的合法性进行认证呢？  
  
authenticationProcessingFilter其实只负责获取需要认证的用户信息并根据结果完成页面转向，它相当于一个客户受理的窗口，真正的用户身份认证工作则交由后台的AuthenticationManager完成。

下面，我们为authenticationProcessingFilter注入AuthenticationManager，后者将完成具体的身份认证工作：   
代码清单 5 applicationContext-acegi-plugin.xml   
认证管理器的配置

<beans>

…

<bean id="authenticationProcessingFilter"

class="org.acegisecurity.ui.webapp.AuthenticationProcessingFilter">

<property name="filterProcessesUrl" value="/j\_acegi\_security\_check"/>

<property name="defaultTargetUrl" value="/main.jsp"/>

<property name="authenticationFailureUrl" value="/index.jsp?login\_error=1"/>

①注入认证管理器

<property name="authenticationManager" ref="authenticationManager" />

</bean>

②认证管理器

<bean id="authenticationManager" class="org.acegisecurity.providers.ProviderManager">

<property name="providers">

<list>

②-1使用基于DAO的认证提供者提供认证服务

<ref local="daoAuthenticationProvider" />

</list>

</property>

</bean>

<bean id="daoAuthenticationProvider" ③基于DAO的认证提供者

class="org.acegisecurity.providers.dao.DaoAuthenticationProvider">

③-1根据用户名获取系统中真实UserDetails 对象的服务类

<property name="userDetailsService" ref="userDetailsService" />

</bean>

④ 该服务类根据缓存在[内存](http://product.it168.com/list/b/0205_1.shtml)中的用户信息列表获取userDetails对象

<bean id="userDetailsService" class="org.acegisecurity.userdetails.memory.InMemoryDaoImpl">

<property name="userMap"> ④-1用户信息

<value>

john=john,PRIV\_COMMON,PRIV\_1

tom=tom,PRIV\_COMMON,PRIV\_1,PRIV\_2

peter=peter,disabled,PRIV\_COMMON,PRIV\_1

</value>

</property>

</bean>

</beans>

在①处，我们为authenticationProcessingFilter注入了一个AuthenticationManager Bean。AuthenticationManager有两个实现类：   
l MockAuthenticationManager：这个实现类用于开发时的测试环境，它将所有待认证的用户标志为有效有户；   
l ProviderManager：该实现类将用户身份认证的工作委托给多个提供者来完成。   
  
这 里，我们使用ProviderManager来定义一个AuthenticationManager Bean，如②所示。可以通过ProviderManager的providers属性配置多个认证提供者，认证提供者必须实现 org.acegisecurity.providers.AuthenticationProvider接口。身份认证的多样性通过 AuthenticationProvider接口实现类的多样性体现出来，Acegi针对不同的[安全](http://safe.it168.com/)系统提供了10多个不同的认证提供者。这些不同的认证提供者在表 1中进行说明：   
  
这里，我们使用DaoAuthenticationProvider，在②-1所示。它负责从数据库或其它保存用户信息的媒介中获取用户信息进行认证。   
  
DaoAuthenticationProvider 首先从SecurityContextHolder的Authentication中得到待认证的用户名，并根据该用户名获取保存在数据库或其它媒介中代 表真正系统用户的UserDetails对象。紧接着比较Authentication和UserDetails的匹配关系（如看密码是否相等），如果两 者匹配，认证成功，并将UserDetails的权限信息将拷贝到Authentication中。如果不匹配，认证将失败。  
  
DaoAuthenticationProvider通过UserDetailsService完成UserDetails的获取工作，根据[存储](http://storage.it168.com/)用户信息媒介的不同，Acegi提供了两个UserDetailsService的实现类：   
  
l InMemoryDaoImpl：该实现类负责从内存中获取用户的信息，它允许通过UserMap或Properties直接在Spring配置文件中定义系统用户信息。   
  
l JdbcDaoImpl：该实现类从数据库中获取用户的信息。我们将在下一节讨论该实现类的使用方法。   
  
在 这里，我们使用InMemoryDaoImpl获取UserDetails，如③-1所示并通过userMap属性定义系统用户信息，如④-1所示。 userMap属性的类型为UserMap，Acegi已经向Spring注册了对应的编辑器，所以我们可以通过格式化的字符串为UserMap类型的属 性提供配置值。在④-1处，我们提供了几行键值对的配置值，每一行代表一个用户及其对应的权限信息。下面我们通过图 2来了解代表一个用户的字符串的格式：   
  
  
8 用户配置信息  
  
我 们通过④-1定义了三个用户，john、tom和peter，它们对应的密码和用户名相同。其中john和tom是激活的用户，而peter用户处于非激 活状态。john和peter都拥有PRIV\_COMMON,PRIV\_1权限，而tom额外拥有PRIV\_2的权限。   
  
如果用户数比较多，在Spring中直接进行配置未免不太雅观，这时，你可以将用户信息转移到一个属性文件中，并通过userProperties进行加载：   
  
<bean id="userDetailsService"   
  
class="org.acegisecurity.userdetails.memory.InMemoryDaoImpl">   
  
<property name="userProperties">   
  
<bean class="org.springframework.beans.factory.config.PropertiesFactoryBean">   
  
<property name="location" value="/WEB-INF/users.properties" />   
  
</bean>   
  
</property>   
  
</bean>   
  
users.properties文件内容的格式和刚才我们介绍的方式相同，既每一个用户对应一行格式化配置串。   
  
通过以上分析后，我们可以理出参与用户身份认证所涉及到4个Bean，它们之间的关系可通过图 9来描述：

图 9 基于内存用户信息认证的主要Bean之间的关系   
  
  
  
我们知道，大部分应用系统的用户信息并非一个小小的属性 文件可以应付的，它们一般保存在数据库中。InMemoryDaoImpl存在的更多意义在于方便程序测试——你可以不依赖外部数据库环境完成测试。在生 产环境下，我们应该使用JdbcDaoImpl这个UserDetailsService实现类从数据库中获取UserDetails

**基于数据库**[**存储**](http://storage.it168.com/)**用户信息的认证**   
  
使用JdbcDaoImpl和InMemoryDaoImpl类似，它们的基本原理都是根据Authentication中待认证的用户名查询出代表真实系统用户的UserDetails对象。只不过，InMemoryDaoImpl通过查询驻留于[内存](http://product.it168.com/list/b/0205_1.shtml)中的格式化用户信息列表完成的，而JdbcDaoImpl则通过查询数据库达到目的。   
  
在介绍JdbcDaoImpl的具体使用之前，我们先在数据库中创建用户信息表和用户权限表，并初始化一些测试的用户数据：   
  
代码清单 6 创建用户及权限的SQL脚本

CREATE TABLE T\_USER ( ①用户信息表

USER\_ID INTEGER NOT NULL AUTO\_INCREMENT,

USERNAME VARCHAR(30) NOT NULL,

PASSWORD VARCHAR(30) DEFAULT NULL,

STATUS TINYINT(1) NOT NULL DEFAULT '0',

PRIMARY KEY (`USER\_ID`),

UNIQUE KEY `USERNAME` (`USERNAME`)

);

CREATE TABLE T\_USER\_PRIV (②用户权限表

USER\_ID INTEGER NOT NULL DEFAULT '0',

PRIV\_NAME VARCHAR(30) DEFAULT NULL,

PRIMARY KEY (USER\_ID, PRIV\_NAME)

);

INSERT INTO T\_USER (USER\_ID, USERNAME, PASSWORD, STATUS) VALUES ③用户数据

(1,'tom','tom',1),

(2,'john','john',1);

INSERT INTO T\_USER\_PRIV (USER\_ID, PRIV\_NAME) VALUES ④用户授权表

(1,'PRIV\_1'),

(1,'PRIV\_2'),

(1,'PRIV\_COMMON'),

(2,'PRIV\_1'),

(2,'PRIV\_COMMON');

COMMIT;

你可以简单地运行随书光盘的chapter17/schema/mysql/db.sql脚本文件完成用户[安全](http://safe.it168.com/)相关信息表的创建工作。   
  
下面，我们用JdbcDaoImpl替换InMemoryDaoImpl，从数据库中获取UserDetails对象，其代码如代码清单 7所示：   
  
代码清单 7 applicationContext-acegi-plugin.xml   
  
基于数据库存储的用信息获取

…

<bean id="daoAuthenticationProvider"

class="org.acegisecurity.providers.dao.DaoAuthenticationProvider">

①调整为从数据库获取UserDetails对象的服务类

<property name="userDetailsService" ref="userDetailsService" />

</bean>

<bean id="userDetailsService" class="org.acegisecurity.userdetails.jdbc.JdbcDaoImpl">②

<property name="dataSource" ref="dataSource" /> ②-1 数据源

<property name="usersByUsernameQuery"> ②-2 根据用户名查询用户的SQL语句

<value>

SELECT username,password, status FROM t\_user WHERE username = ?

</value>

</property>

<property name="authoritiesByUsernameQuery"> ②-3 根据用户名查询用户权限记录的SQL语句

<value>

SELECT u.username,p.priv\_name FROM t\_user u,t\_user\_priv p

WHERE u.user\_id =p.user\_id AND u.username = ?

</value>

</property>

</bean>

这种替换的难度很小，仅需要配置一个JdbcDaoImpl Bean并将其作为daoAuthenticationProvider的userDetailsService的实现即可。   
  
JdbcDaoImpl 其实是基于Spring JDBC技术进行数据库访问的，它继承于org.springframework.jdbc.core.support.JdbcDaoSupport。 所以JdbcDaoImpl必不可少的一个属性是dataSource，它指定一个保存用户信息的数据源。   
  
JdbcDaoImpl通过usersByUsernameQuery和authoritiesByUsernameQuery属性定义查询用户信息和用户权限的SQL语句。实际上，JdbcDaoImpl为以上两个属性提供了默认的SQL语句，分别是：   
  
"SELECT username,password,enabled FROM users WHERE username = ?"   
  
和   
  
"SELECT username,authority FROM authorities WHERE username = ?"   
  
一 般情况下，我们会使用自己的用户和权限表，所以你需要自定义以上两个属性。JdbcDaoImpl将Authentication中待认证用户名作为这两 个SQL语句的username参数值，并执行查询返回结果，然后根据列索引的方式从结果集中取得数据构造UserDetails对象，所以在构造SQL 语句时必须保持用户信息字段位于正确的列位置，字段名可以任意指定。   
  
应该说，JdbcDaoImpl还不是非常实用的 UserDetailsService实现类，因为用户对象（UserDetails）除包含用户名/密码、是否激活、权限等信息外，还经常需要包含一些 诸如email、telephone等业务相关的信息。而JdbcDaoImpl通用类并不知道我们具体的业务需求，所以我们往往需要通过实现 UserDetailsService接口提供自己的实现类来完成这些工作。编写一个自己的UserDetailsService实现类不费什么力气，你 只要实现接口中的UserDetails loadUserByUsername(String username)方法根据用户名获取UserDetails对象即可。在编写好自己的UserDetailsService后，在Spring配置并替 换①处的userDetailsService属性即可，限于篇幅，我们不再展开论述，读者可以试着自行完成。   
  
**小结**   
Acegi通过过滤器对请求实施拦截，当发现用户没有登录时，将强制用户进行系统登录，通过一定的接口规范，Acegi就可以获取操作者的用户身份，并进而获取用户的权限。在下一篇文章中，我们将接着介绍身份认证的高级话题。