Lesson 12
Security & Cross-cutting Concerns
Infinite Diversity Arising from Unity

Definition: Crosscutting Concerns

- Term comes from Aspect Oriented Programming [AOP]

It involves:

- "…the modularization of concerns such as transaction management that cut across multiple types and objects. (**Such concerns are often termed *crosscutting* concerns in AOP literature.)"

Cross-cutting Technologies

- **Servlet Filter**
  - Generic Servlet/web based filter
- **Interceptor**
  - Spring MVC Handler specific Interceptor
- **Spring AOP**
  - Simplified AOP implementation- Method level granularity
  - Only Spring recognized Beans
  - Employs a run time integration [AKA weaving] process
- **AspectJ**
  - Fine grained supports method & field level AOP
  - Employs a specialized compilation weaving process
  - Works with non-Spring components

FILTER SERVLET

- Based on Servlet Specification
- Coupled with the Servlet API.
- Access to HttpServletRequest and HttpServletResponse objects
- Intended for operating on request and response object parameters like HTTP headers, URIs and/or HTTP methods,
- Generically applied - regardless of how the servlet is implemented.
- **EXAMPLES**: Authentication , Logging and auditing

Handler Interceptor

- Part of Spring MVC Handler mapping mechanism
- Fine grained access to the handler/controller
  - preHandle() - before controller execution
  - postHandle() – after controller execution
    - Can expose additional model objects to the view via the given ModelAndView.
  - afterCompletion() - after rendering the view. Allows for proper resource cleanup.
- Interceptors can be applied to a group of handlers.
-

Volunteer Interceptor

```
public void postHandle(HttpServletRequest request,
HttpServletResponse response, Object handler,ModelAndView
modelAndView) throws Exception {
String userMessage= "Become a Community Member- Join the Team!";

Principal principal = request.getUserPrincipal();

if (principal != null) {
if (request.isUserInRole("ROLE_ADMIN") )
userMessage= "There is ALWAYS Free cookies at www.freebies.com";
else
userMessage = "We have Many NEW and exciting Volunteer
opportunities!!!";
 }
```

Interceptor Configuration

- **AntPathMatcher**
- The mapping matches URLs using the following rules:

  ? matches one character

  * matches zero or more characters

  ** matches zero or more 'directories' in a path

```
<mvc:interceptors>
<mvc:interceptor>
        <mvc:mapping path="/**"/>
        <bean class="mum.edu.interceptor.VolunteerInterceptor"
/>
    </mvc:interceptor>
</mvc:interceptors>
```

AOP & ASPECTJ

- SpringAOP:
- 1)Runtime weaving through proxy using the concept of a dynamic proxy
- 2)Spring AOP supports only method level PointCut
-
  AspectJ:
  1)Compile time weaving if source available or post compilation weaving (using compiled files).
  2)AspectJ supports both method and field level Pointcuts
-

Main Point

- The different technologies [Filter, Interceptor, AOP] available in Spring, together provide a thorough solution to cross cutting concerns.
- *Creative intelligence enhances and strengthens uniquely differing values in life in a comprehensive way.*
-

Spring Web Application Security Servlet Filter based

-
- Spring Security's web infrastructure is based entirely on standard servlet filters.
- Agnostic to specific web technology.
- Based on HttpServletRequests and HttpServletResponses
  - Browser
  - Web service client
  - AJAX application.

Authentication Authorization

- Authentication refers to unique identifying information from each system user, generally in the form of a username and password.

  Authorization refers to the process of allowing or denying individual user access to resources.

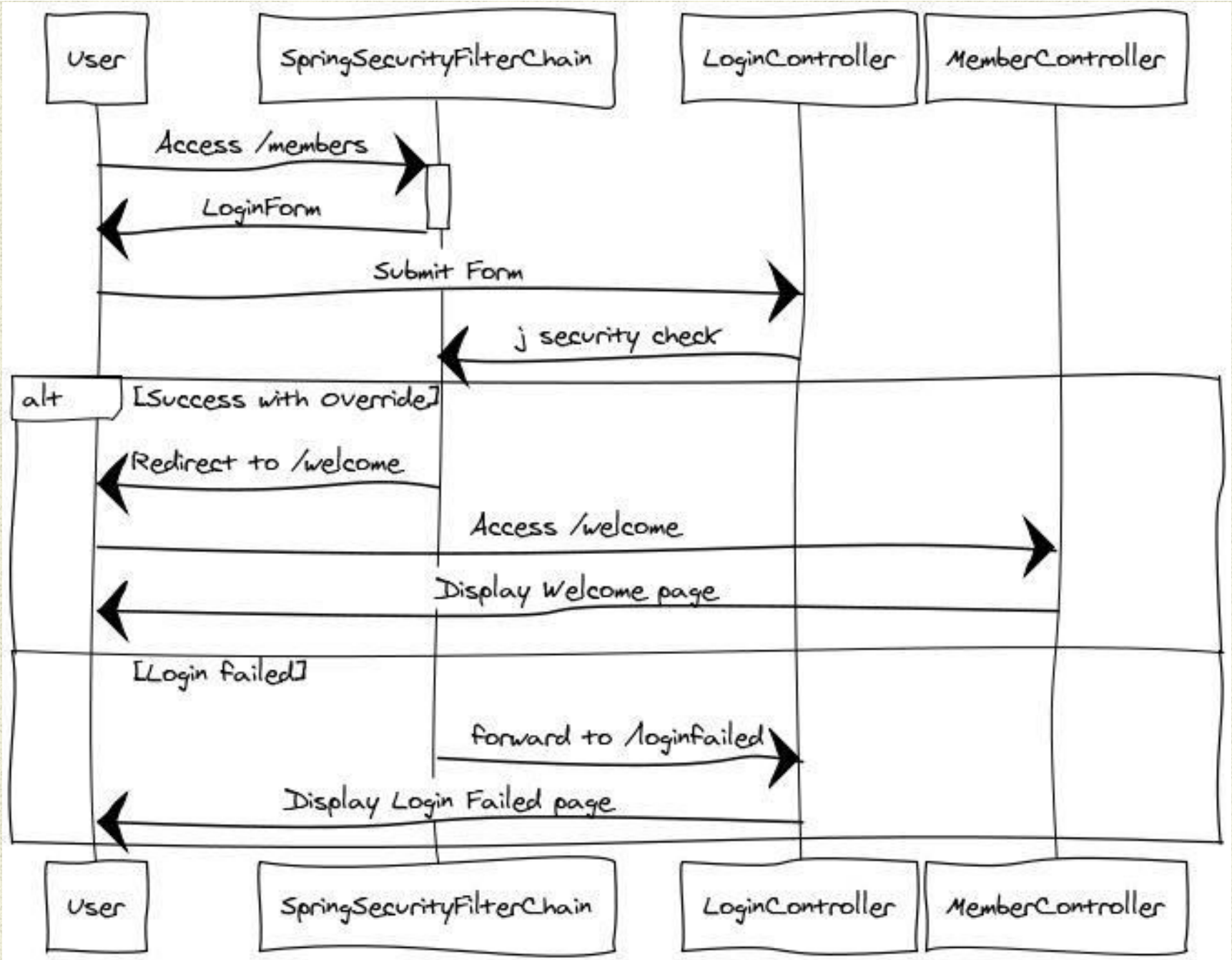
Basic and Digest Authentication

- **Basic authentication**
  - Handshake based on HTTP headers
  - Transmits username/password as “plain text”
    - Base64 encoding
  - Used in conjunction with SSL-HTTPS
  - Used with form-based authentication
  - Secure data at rest
- **Digest Authentication**
  - Transmits encrypted username/password
  - “Double” handshake to get hash “seed”
  - More complex – more vulnerable

Minimal [XML] configuration

- Requires all users to be authenticated
- Allows users to authenticate with form based login
- Allows users to authenticate with HTTP Basic authentication
-

```
<security:http use-expressions= "true“ >
    <intercept-url pattern="/**" access= “isFullyAuthenticated()“ />
    <form-login />
    <http-basic />
</security:http>
```


User
SpringSecurityFilterChain
LoginController
MemberController
Access /members
LoginForm
Submit Form
j security check
alt
[Success with override]
Redirect to /welcome
Access /welcome
Display Welcome page
[Login failed]
forward to /loginfailed
Display Login Failed page
User
SpringSecurityFilterChain
LoginController
MemberController

Spring Security Tag Library

- Basic support for security information and constraints in JSPs
- Basically 3 tags

authorize tag

```
<security:authorize access="isAuthenticated()">
```

authenticate tag

```
<security:authentication property="principal.username" />
```

renders the name of the current user.

accesscontrollist tag

used with Spring Security's ACL module

```
<security:accesscontrollist hasPermission="admin,designer"
        domainObject="${someObject}">
```

Display if user has either permission for someObject.

```
</sec:accesscontrollist>
```

Authentication & Authorization in JSPs

```
<%@ taglib prefix="security"
           uri="http://www.springframework.org/security/tags"%>
```

Login.jsp

```
<form action="<spring:url value="/j_spring_security_check“ /> method="post">
  <div class="form-group">
       <input  placeholder="User Name" name='j_username' type="text">
     <input placeholder="Password" name='j_password'  type="password" value="">
  </div>
        <input  type="submit" value="Login">
</form>
```

Default names: can be overridden in configuration

Welcome.jsp

```
  <security:authorize access="isAuthenticated()">
   Welcome  <security:authentication property="principal.username" />
 </security:authorize>

 <security:authorize access="isAnonymous()">
     <a href="<spring:url value='/login' />" > Login</a>
 </security:authorize>
```

Spring Security web.xml

```
<context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
                /WEB-INF/spring/context/applicationContext.xml
                /WEB-INF/spring/context/security-context.xml
        </param-value>
</context-param>
<listener>
    <listener-class>
        org.springframe
    </listener-class>
</listener>
<filter>
<filter-name>springSecurityFilterChain</filter-name>
        <filter-class>
                org.springframework.web.filter.DelegatingFilterProxy
        </filter-class>
```

The security-context is loaded into the “root” WebApplicationContext as it is NOT Spring MVC specific [DispatcherServlet]

springSecurityFilterChain is an internal infrastructure bean created based on namespace enabling of security <http auto-config='true'>

Form based login

- <form-login /> generates a default login form

- Available at URL: spring_security_login

- Overridden if attributes are set on <form-login />

security-context.xml

- Enable Method level authorization. If here -APPLICATION Level scanned components. For WEB level - need to place in Dispatcher-servlet
- `<security:global-method-security pre-post-annotations="enabled"/>`

- `<security:http use-expressions="true">`
- `<security:intercept-url pattern="/members"`
  `access="hasAnyRole('ROLE_ADMIN','ROLE_USER')" />`
- `<security:form-login login-page="/login"`
  `default-target-url="/welcome"`
  `always-use-default-target="true"`
  `authentication-failure-url="/loginfailed"/>`
  `<security:logout logout-success-url="/logout"`
  `logout-url= "/doLogout"/>`
  `</security:http>`

Allows for Authorization

Landing page after successful login
Overrides where login was triggered

Renames j_security_logout

security-context.xml [cont.]

```
<security:authentication-manager>
 <security:authentication-provider>
  <security:user-service>                *****

    <security:user name="admin" password="admin" authorities="ROLE_ADMIN" />
    <security:user name="guest" password="guest" authorities="ROLE_USER" />

  </security:user-service>>                *****
 </security:authentication-provider>
</security:authentication-manager>
```

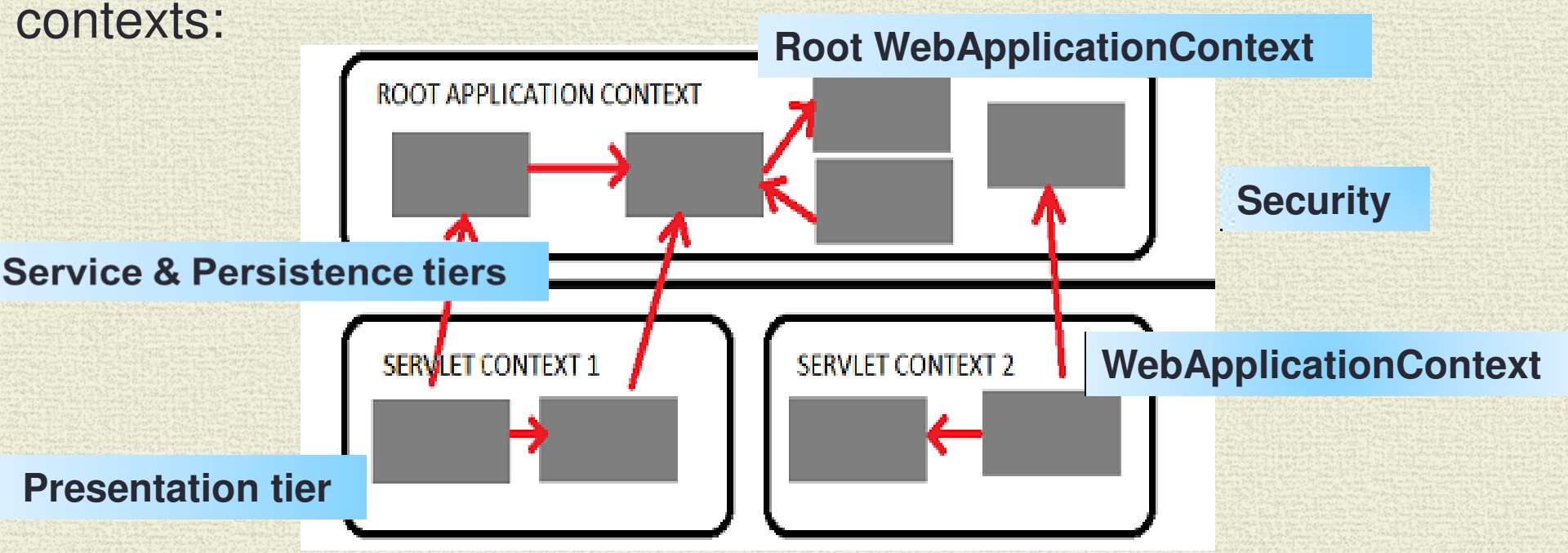
***** replace with

```
<jdbc-user-service data-source-ref="dataSource" />
```

to use DBMS

Web Application Context

- Spring has multilevel application context hierarchies.
- Web apps by default have two hierarchy levels, root and servlet contexts:



- **Presentation tier has a WebApplicationContext [Servlet Context] which inherits all the resources already defined in the root WebApplicationContext [Services, Persistence]**

Authorization

- Web request authorization using filters.
- Method authorization using AspectJ or Spring AOP
- Common usage pattern is to perform **some** web request authorization, coupled with Spring AOP method authorization on the services layer [**more secure**].

- Reference: Security usage of Spring Expression language: SpEL
- http://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/#el-access
- **3.2. Web Security Expressions**
- **3.3. Method Security Expressions [AOP backed]**

URL based Authorization

Configuration:

```
<security:intercept-url pattern="/members/add"
                        access= "hasRole('ROLE_ADMIN')" />
```

Method level Authorization

- Configuration:

```
<security:global-method-security
                        pre-post-annotations="enabled"/>
```

- MemberServiceImpl.java

```
@PreAuthorize("hasRole('ROLE_ADMIN')")
  public void save( Member member) {
     memberRepository.save(member);
  }
```

@ControllerAdvice

- Cross-cutting controller exception handling for application, not just to an individual controller.
- Like an Annotation driven interceptor.
- Three types of methods are supported:
  - Exception handling methods annotated with @ExceptionHandler.
  - Model enhancement methods (for adding additional data to the model) annotated with @ModelAttribute
  - Binder initialization methods (used for configuring form-handling) annotated with @InitBinder.

@ControllerAdvice example

```
@Component
@ControllerAdvice
public class ControllerExceptionHandler {

  @ExceptionHandler(value = AccessDeniedException.class)
    public String accessDenied() {
        return  "error-forbidden" ;
    }
```

Main Point

- Authentication & Authorization underlie the entire web application. They provide a shield that makes the application invulnerable.
- *Transcendental consciousness is characterized by the quality of invincibility, which means one cannot be overcome or overpowered*