**34. AOP Aspect-Oriented Programming Overview**.



Programming technique based on concept of an Aspect. Aspect encapsulates cross-cutting logic (cross-cutting concerns). Concerns means logic/functionality. Aspect can be used at multiple locations. Same aspect/class applied based on configuration.

Benefits of AOP:

* Code for Aspect is defined in a single class.
* Business code in your application is cleaner
* Configurable (no need to change code)

Additional AOP Use cases:

* Most common: logging, security, transaction
* Audit logging: who, what, when, where
* Exception handling: log exception and notify DevOps team via SMS/email
* API Management: how many times has a method been called user. Analytics: what are peak times? What is average loading? Who is the top user?

Disadvantages:

* Too many aspects and app flow is hard to follow
* Minor performance cost for aspect execution (run-time weaving)

**Aspect:** module of code for a cross-cutting concern

**Advice:** What action is taken and when it should applied

**Join Point:** When to apply code during program execution

**Pointcut:** A predicate expression for where advice should be applied

**Advice Types:**

* Before advice
* After finally advice
* After returning advice
* After throwing advice
* Around advice

**Weaving.** Connecting aspects to target objects to create an advised object. There are different types of weaving: Compile-time, load-time or runtime.

There are two leading AOP Frameworks: Spring AOP, AspectJ

Spring AOP:

* Security, transaction, caching
* Uses run-time weaving aspects

Spring AOP Advantages:

* Simpler to use than AspectJ
* Uses proxy pattern
* Can migrate to AspectJ when using @AspectJ annotation

Disadvantages:

* Only supports method-level join points
* Can only apply aspects to beans created by Spring app context
* Minor performance cost for aspect execution (run-time weaving)

AspectJ Advantages:

* Support all join points
* Works with any POJO not just beans from app context
* Faster performance compared to Spring AOP
* Complete AOP support

Disadvantages:

* Compile-time weaving requires extra compilation step
* AspectJ pointcut syntax can become complex

Spring AOP is a light implementation of AOP. Solves common problems in enterprise applications.

Link: AspectJ in Action. Aspect Oriented Programming with use Cases

**35. AOP Before Advice Type.**

@Before - Run custom code BEFORE the Target Object method call

@AfterReturning – Run custom code After the Target Object method call

@Before use cases:

* Most common: logging, security, transactions
* Audit logging: who, what, when, where
* API Management: how many times has a method been called

Need to download AspectJ JAR file even though we are using Spring AOP (Spring uses AspectJ’s annotations and classes)

Development Process:

1. Create target object
2. Create Spring Java Config class



ComponentScan – scans for components and aspects

1. Create an Aspect with @Before

**36. AOP Pointcut Expressions - Match Methods and Return Types.**

**Pointcut:** A predicate expression for where advice should be applied

Spring AOP uses AspectJ’s pointcut expression language. Execution pointcut applies to execution of methods. Pointcut pattern can have wildcards

execution(modifiers-pattern? return-type-pattern declaring-type-pattern?

method-name-pattern(param-pattern) throws-pattern?)

You can use asterisk (everything) or the star as a wildcard.



**37. AOP Pointcut Expressions - Match Method Parameter Types.**

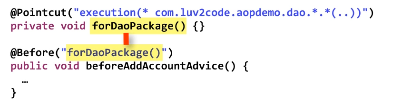
Parameter pattern wildcards:

* () – matches a method with no arguments
* (\*) – matches a method with one argument of any type
* (..) – matches a method with 0 or more arguments of any type

**38. AOP Pointcut Declarations.**

Development process (reuse pointcut expressions):

1. Create a pointcut declaration
2. Apply pointcut declaration advice(s)



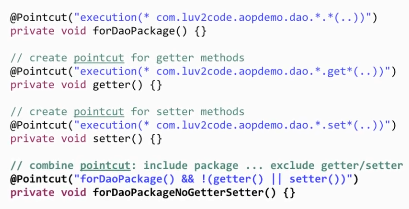
**Combine pointcuts** using logic operators:

* AND (&&)
* OR (||)
* NOT (!)

It works like an if statements. Execution happens only if it evaluates to true.

Development process:

* Create a pointcut declarations
* Combine
* Apply to advices



**39. AOP Ordering Aspects**

Aspect order is undefined.

To control order:

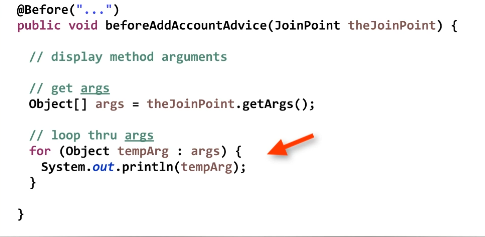
* Refactor: Place advices in separate Aspects
* Control order on Aspects using the @Order annotation (in parentheses you give the precedence number)

Lower numbers have higher precedence. Range: Integer.MIN\_VALUE to Integer.MAX\_VALUE. Negative numbers are allowed. Does not have to be consecutive.

**40. AOP JoinPoints.**

Access and display Method Signature. JoinPoint will give us information about the method that we’re actually executing.





**41 AOP AfterReturning Advice Type.**

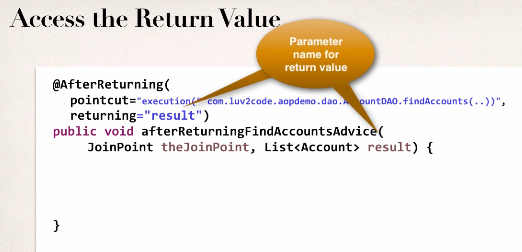
AfterReturning Advice – run after the method completed for successful execution.

Use cases:

* Most common: logging, security, transactions
* Audit logging: who, what, when, where
* Post-processing Data: Post process the data before returning to caller

Use @AfterReturning annotation

Access return value:

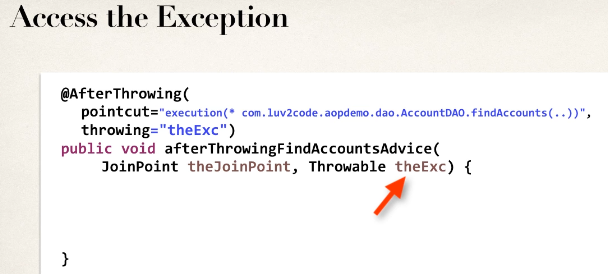


Modify return value (Post process).

**42. AOP AfterThrowing Advice Type.**

Use cases:

* Log the exception
* Perform auditing on the exception
* Notify DevOps team via email or SMS
* Encapsulate this functionality in AOP aspect for easy reuse



At this point, we are only intercepting the exception. However, the exception is still propagated.

If you want to stop the exception propagation then use @Around advice.

**43. AOP After (finally) Advice Type.**

@After runs after a method is completed. Regardless of outcome / exceptions (works like “finally” block). After will execute before AfterThrowing.

Use cases:

* Log the exception and/or perform auditing
* Code to run regardless of method outcome
* Encapsulate this functionality in AOP aspect for easy reuse

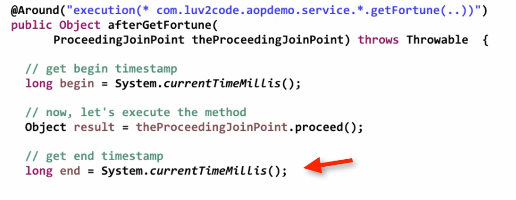
After advice does not have access to the exception. If you need exception, then use @AfterThrowing advice

**44. AOP Around Advice Type** – Before and after target

Use cases:

* Most common: logging, auditing, security
* Pre-processing and post-processing
* Instrumentation / profiling code
* Managing exceptions

When using @Around advice you will get a reference to a ProceedingJoinPoint. This is a handle to the target. Your code can use ProceedingJoinPoint to execute target method



Resolving Print Order Issue:

* **Root cause:**
  + The data is printing to two different output streams
  + Spring is printing to the logger output stream
  + System.out.println is printing to the standard output stream
* **Solution**
  + To have everything in order, you should send to same output stream

**45. AOP Add AOP Logging to Spring MVC App**

Development process:

1. Add AspectJ JAR file
2. Enable AspectJ Auto Proxy
3. Create Aspect:
   1. Add logging support
   2. Setup pointcut declarations

**46. Maven Crash Course**

Maven reference: luv2code.com/mavenreferencemanual

EBook: luv2code.com/mavenebooks

Cheat sheet: luv2code.com/mavencheatsheet

Maven is a Project Management Tool.

Pom file is Project Object Model.Configuration file for your project. Located in the root. Structure:

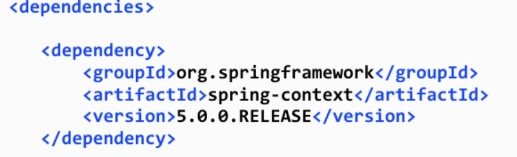
* Project meta data
* Dependencies
* Plug ins

Project coordinates uniquely identify a project: groupId, artifactId, version

Group id – name of company, group, or organization. Convention is to use reverse domain name

Artifact id – name for this project

Version – a specific release version. (optional for repeatable builds)



May see this referred to as: GAV (Group, Artifact, Version)

Maven archetypes. Archetypes can be used to create new Maven projects. Contains template files for a given Maven project:

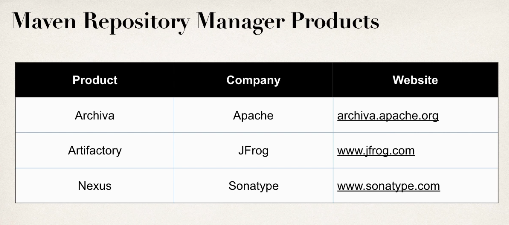
Maven-archetype-quickstart – an archetype to generate a sample Maven project

Maven-archetype-web – for web applications

Local repository: C:\Users\<users-home-dir>\.m2\repository

Adding repository:





**47. Spring Security - Getting Started**

Spring defines a framework for security. Implements using Servlet filters in the background. Two methods of securing a Web app: declarative and programmatic.

Servlet filters are used to pre-process / post process web requests. Servlet Filters can rout web requests based on security logic.

Authentication – checks user id and password

Authorization – checks user role

**Declarative Security.**Define application’s security constraints in configuration:

* All Java Config (@Configuration, no xml)
* Or Spring XML config

Provides separation of concerns between application code and security

Different login methods. HTTP Basic Authentication, default and custom login form.

Authentication and Authorization:

* In-memory
* JDBC
* LDAP
* Custom / Pluggable
* Others …

**Spring MVC Java Configuration.**

Development Process:

1. Add Maven dependencies for Spring MVC Web App
2. Create Spring App Configuration (@Configuration)
3. Create Spring Dispatcher Servlet Initializer
4. Develop our Spring controller
5. Develop our JSP view page

Dependencies:

* Spring-webmvc
* jstl
* javax.servlet-api
* javax.servlet.jsp-api

Customize Maven Build, since we are not using web.xml. Use maven-war-plugin.

Java config:

* Spring @Configuration
* Spring Dispatcher Servlet Initializer

@EnableWebMvc:

* Provides similar support to <mvc:annotation-driven /> in XML
* Adds conversion, formatting and validation support
* Processing of @Controller classes and @RequestMapping etc … methods

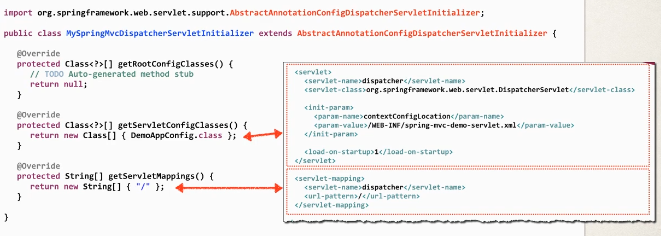


Spring MVC provides support for web app initialization. Makes sure your code is automatically detected. Your code is usually to initialize the servlet container:

* AbstractAnnotationConfigDispatcherServletInitializer

Web App Initializer TODO:

* Extend this abstract base class
* Override required methods
* Specify servlet mapping and location of your app config

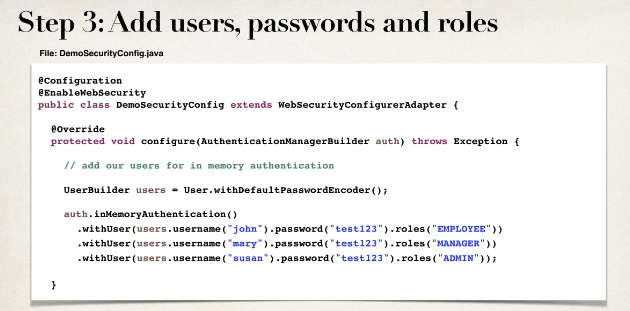


**Finding compatible version.** Find desired version of Spring Security in Maven Central Repo (spring-security-web). Look at the project POM file. Find supporting Spring Framework version.



Spring Security Web App Initializer. Spring security provides support for security initialization. Your security code is used to initialize the servlet container. Special class to register the Spring Security Filters: AbstractSecurityWebApplicationInitializer. Extend this abstract base class.





Summary:

1. Create Spring Security Initializer
2. Create Spring Security Configuration (@Configuration)
3. Add users, passwords, and roles

**48. Spring Security - Adding Custom Login Form**

Development Process

1. Modify Spring Security Configuration to reference custom login form
2. Develop a Controller to show the custom login form
3. Create custom form page

**Step 1.** WebSecurityConfigurerAdapter has:

* configure(AuthenticationManagerBuilder) – Configure users (in memory, database, LDAP, etc.)
* configure(HttpSecurity) – Configure security of web paths in application, login, logout etc.



**Step 3.** Send data to login processing <URL:/authenticateTheUser>. Must POST the data:



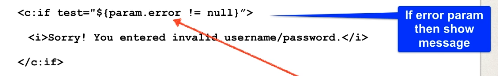
authenticateTheUser – could be anything, just stay consistent within your application.

Best practice is to use the Spring MVC Form tag <form:form>. Provides automatic support for security defenses. Spring Security defines default names for login form fields:

* username
* password

**Error page.** Development Process:

1. Modify custom login page
   1. Check the error



* 1. If error parameter exists, show an error message

**49. Spring Security - Bootstrap CSS Login Form**

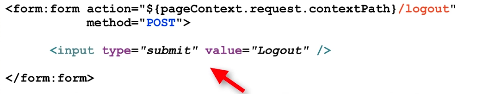
Development process:

1. Modify form and add support for spring MVC tags
2. Update form to point to our login processing URL
3. Verify fields form username and login
4. Update controller

**50. Spring Security - Adding Logout Support**

Development process:

1. Add logout support to Spring Security Config
2. Add logout button to page
   1. Send data to default logout <URL:/logout>
   2. Logout URL will be handled by Spring Security Files



Get method disabled by default

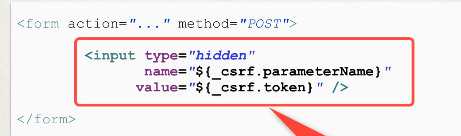
1. Update login form to display “log out” message

**51. Spring Security - Cross Site Request Forgery (CSRF)**

Spring security protects against CSFR. CSFR – is a security attack where an evil website tricks you into executing an action on a web application that you are currently logged in. CSFR is enabled by default. Spring security uses the Synchronizer Token pattern. Each request includes a session cookie and randomly generated token. For request processing Spring Security verifies token before processing. All of this is handled by Spring Security Filters.

Use CSFR protection for any normal browser web requests. If you are building service for non-browser clients you may want to disable CSFR protection.

User Spring security CSFR Protection for form submission POST instead of get. Include CSFR token in form submission. <form:form> automatically adds CSFR token. If you don’t use that tag, you have to manually add protection.



**52. Spring Security - User Roles**

Displaying username and role:

1. Update POM file for Spring Security JSP Tag Library
   * Add new dependency groupId=”org.springframework.security”, artifactId=”spring-security-taglibs”, version=”${springsecurity.version}
2. Add Spring Security JSP Tag Library to JSP page
   1. Uri=”http://www.springframework.org/security/tags”
3. Display User ID and User Roles
   1. <security:authentication property=”principal.username” />
   2. Same for authorites (user roles)

**53. Spring Security - Restrict Access Based on Role.**

Development Process:

1. Create supporting controller code and view pages
2. Update user roles
3. Restrict access based on roles
   1. Update your spring security java configuration file
   2. General syntax: antMatchers(<< add path to match on>>).hasRole(<<Role>>). You could also use hasAnyRole, with comma delimited list. \*\* match on all subdirectories.

Custom access denied page. Development Process:

1. Configure custom page for access denied
   1. .exceptionHandling().accessDeniedPage(“page”)
2. Create supporting controller code and view page

Display Content Based on Roles. Use spring security jsp tags:

* <security:authorize access=”hasRole(‘Manager’)”>
* Place content
* Close tag

**54. Spring Security - Add JDBC Database Authentication**

User Accounts Stored in Database.

By default, you have to follow Spring Security’s predefined table schemas.

**Development Process**

1. Develop SQL Script to set up database tables
2. Add database support to Maven POM file
3. Create JDBC properties file
4. Define DataSource in Spring Configuration
   1. @PropertySource(“classpath:persistence-mysql.properties”)
5. Update Spring Security Configuration to use JDBC
   1. @Autowired private Environment env
   2. @Bean public DataSource securityDataSource()
   3. Create connection pool:
      1. ComboPooledDataSource
   4. Set the jdbc driver
      1. Set jdbc driver on data source
   5. Set database connection props

Spring Security Database Schema:

users:

1. username VARCHAR(50)
2. password VARCHAR(50)
3. enabled TINYINT(1)

authorities:

1. username VARCHAR(50)
2. authority VARCHAR(50)

In Spring Security 5, passwords are stored using a specific format:

{id}encodedPassword

Id:

* noop – plain text passwords
* bcrypt – BCrypt password hashing

**55. Spring Security - Password Encryption**

Spring Security Team recommends using the populat bcrypt algorithm

Development process:

1. Run SQL Script that contains encrypted passwords
   1. Modify DDL for password field, length should be 68
2. Modify database properties file to point to new database schema