Agenda

AOP

Java 8 support

XML

SpEL

Spring
Aspect Oriented
Programming

When will this method be called?

```
@Aspect
public class BeforeExample {

    @Before("execution(* com.xyz.myapp.dao..(..))")
    public void doAccessCheck() {
        // ...
    }

}
```

Terminology

Aspect

Concept shared between other classes.

Example:
transaction management
logging
security

Terminology

Advice

Tells, what to do, when something else is done.

Example: when we access the database, a transaction is opened before we do it, and commited or rolled back after we do it.

Terminology

Join Point

The place where we want Advice to do its job.

Example: execution of a method.

Terminology

Pointcut

A predicate that matches Join Points.

Example: all public methods on services.

Terminology

Target object

The object being advised.

Example: our service.

Weaving

Linking aspects with other application types or objects to create an advised object.

- compile time weaving (AspectJ compiler)
- load time (using JVM agent)
- at runtime (proxies: default for Spring AOP)

AOP Proxy

an object created by the AOP framework in order to implement the aspect contracts

AspectJ or Spring AOP weaving?

If you only need to advise the execution of operations on Spring beans, then Spring AOP is the right choice.

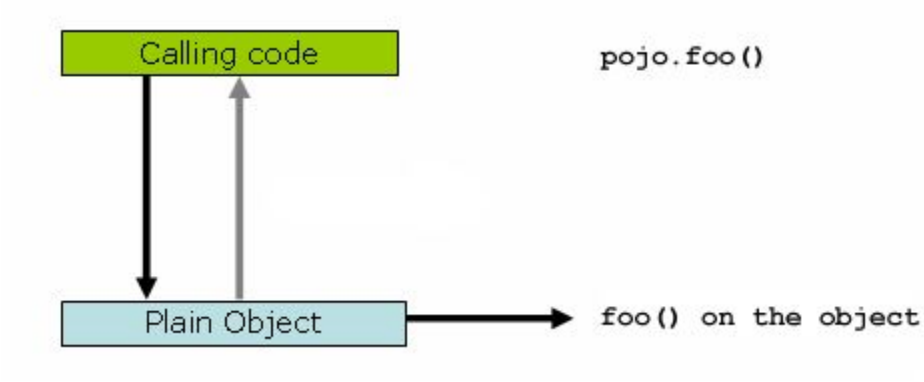
If you need to advise objects not managed by the Spring container (such as domain objects typically), then you will need to use AspectJ.

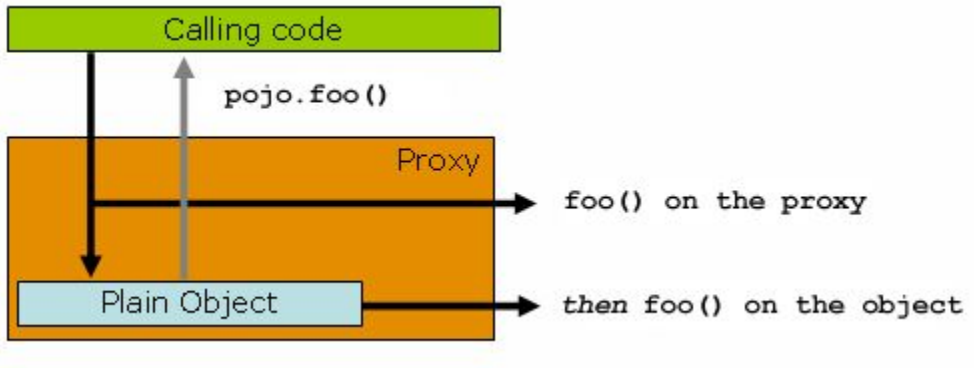
Spring AOP

You can declare aspects either in XML, or using AspectJ annotations (event if you are not using AspectJ weaving).

Runtime Weaving

proxies, proxies everywhere


Calling code
pojo.foo()
Plain Object
foo() on the object


Calling code
pojo.foo()
Proxy
foo() on the proxy
Plain Object
then foo() on the object

Setup

```
//AspectJ annotation based
@Configuration
@EnableAspectJAutoProxy
public class AppConfig {
}
```

Pure AspectJ annotation aspect

```
@Component
@Aspect
public class BeforeExample {

    @Before("com.xyz.myapp.BeforeExample.dataAccessOperation()")
    public void doAccessCheck() {
        // ...
    }

    @Pointcut("execution( com.xyz.someapp.dao..(..))")
    public void dataAccessOperation() {}
}
```

Common mistake (at least mine)

I have an aspect.

But I have not registered it.

@Aspect does not register the class in Spring Container. You have to register it yourself.

Small tip

Even if you are using Groovy, write Aspects in Java. You will have nice IntelliJ IDEA support.

Keeping common pointcuts together

```
@Aspect
public class SystemArchitecture {
   @Pointcut("within(com.xyz.someapp.web..)")
   public void inWebLayer() {}

   @Pointcut("within(com.xyz.someapp.service..)")
   public void inServiceLayer() {}

   @Pointcut("within(com.xyz.someapp.dao..)")
   public void inDataAccessLayer() {}

   @Pointcut("execution( com.xyz.someapp.service..(..))")
   public void businessService() {}

   @Pointcut("execution( com.xyz.someapp.dao..(..))")
   public void dataAccessOperation() {}
}
```

Pointcuts: examples

execution(public * *(..))

execution(* set*(..))

execution(* com.xyz.service.AccountService.*(..))

execution(* com.xyz.service..(..))

target(com.xyz.service.AccountService)

@annotation(org.springframework.transaction.annotation.Transactional)

bean(*Service)

Advice types

```
@Before("...")
public void doAccessCheck() {

@AfterReturning("...")
public void doAccessCheck() {

@AfterReturning(pointcut="...", returning="retVal")
public void doAccessCheck(Object retVal) {

@AfterThrowing("...")
public void doRecoveryActions() {

@AfterThrowing(pointcut="...", throwing="ex")
public void doRecoveryActions(DataAccessException ex) {

@After("...")
public void doReleaseLock() {
```

Advice types

```
@Around("...")
public Object doBasicProfiling(ProceedingJoinPoint pjp)
                                                    throws Throwable {
    // start stopwatch
    Object retVal = pjp.proceed();
    // stop stopwatch
    return retVal;
}
```

Accessing params

```
@Before("com.xyz.myapp.SystemArchitecture.dataAccessOperation() && args(account,..)")
public void validateAccount(Account account) {
    // ...
}
```

```
@Pointcut("com.xyz.myapp.SystemArchitecture.dataAccessOperation() && args(account,..)")
private void accountDataAccessOperation(Account account) {}

@Before("accountDataAccessOperation(account)")
public void validateAccount(Account account) {
    // ...
}
```

Accessing JoinPoint

```
@Before("com.xyz.lib.Pointcuts.anyPublicMethod()")
public void audit(JoinPoint jp) {
    // ... use jp
}
```

JoinPoint

Object getThis(); //*Returns the currently executing object*

Object getTarget(); //*Returns the target object*

Object[] getArgs(); //*Returns the arguments at this join point*

Signature getSignature(); //*Returns the signature at the join point.*

SourceLocation getSourceLocation(); //*Returns the source location corresponding to the join point*

String getKind(); //*Returns a String representing the kind of join point.*

Load time weaving
and
Compile time weaving

because DDD

Injecting into domain entities

```
@Entity
@Configurable(dependencyCheck = true)
public class Story {
    @Transient
    private ChapterInStoryRepository chapterInStoryRepository;

    @Transient
    private ChapterInStoryMover chapterInStoryMover;

    @Transient
    private ChapterInStoryRemover chapterInStoryRemover;

    @Transient
    private ChapterInStoryAdder chapterInStoryAdder;
```

```
<bean class="pl.touk.storytelling.domain.aggregates.story.Story" scope="prototype">
    <property name="chapterInStoryMover" ref="chapterInStoryMover"/>
    <property name="chapterInStoryRemover" ref="chapterInStoryRemover"/>
    <property name="chapterInStoryAdder" ref="chapterInStoryAdder"/>
</bean>
```

If you want it available to constructors

@Configurable(preConstruction=true)

How to turn it on
Step 1: tell the container

```
@Configuration
@EnableSpringConfigured
@EnableLoadTimeWeaving
public class AppConfig {

}
```

```
<context:spring-configured/>
<context:load-time-weaver/>
```

How to turn it on
Step 2: add agent to JVM

-javaagent:path/to/aspectjweaver.jar

or

-javaagent:path/to/
org.springframework.instrument-{version}.jar

Step 3: tell AspectJ in META-INF/aop.xml

```
<!DOCTYPE aspectj PUBLIC
      "-//AspectJ//DTD//EN" "http://www.eclipse.org/aspectj/dtd/aspectj.dtd">
<aspectj>
   <weaver>
      <exclude within="*..*CGLIB*" />
      <include within="pl.touk.storytelling.domain..*"/>
      <include within="pl.touk.storytelling.aspects..*"/>
   </weaver>

   <aspects>
      <aspect
name="org.springframework.beans.factory.aspectj.AnnotationBeanConfigurerAspect"/>
      <aspect name="org.springframework.transaction.aspectj.AnnotationTransactionAspect"/>
   </aspects>

   <!--
   <weaver options="-showWeaveInfo"/>
   -->
</aspectj>
```

Weavers for app servers

Runtime Environment	LoadTimeWeaver implementation
Running in BEA's Weblogic 10	WebLogicLoadTimeWeaver
Running in IBM WebSphere Application Server 7	WebSphereLoadTimeWeaver
Running in GlassFish	GlassFishLoadTimeWeaver
Running in JBoss AS	JBossLoadTimeWeaver
JVM started with Spring InstrumentationSavingAgent *(java -javaagent:path/to/spring-instrument.jar)*	InstrumentationLoadTimeWeaver
Fallback, expecting the underlying ClassLoader to follow common conventions (e.g. applicable to TomcatInstrumentableClassLoader and Resin)	ReflectiveLoadTimeWeaver

Plus special class loader for Tomcat

Compile vs Load time weaving

If you cannot add JVM agent, you cannot use load time weaving.

But compile time weaving means, you gonna have bad time with unit tests.

Use compile time weaving only if you have no other choice.

Java 8 support

JSR-310 Date and Time

```
@DateTimeFormat(pattern="M/d/yy h:mm")
private LocalDateTime lastContact;

@RequestMapping("/date/{localDate}")
public String get(@DateTimeFormat(iso = DateTimeFormat.ISO.DATE)
                                                    LocalDate localDate) {
  return localDate.toString();
}
```

@DateTimeFormat Supports formatting by style pattern, ISO date time pattern, or custom format pattern string. Can be applied to java.util.Date, java.util.Calendar, java.long.Long, Joda-Time value types; and as of Spring 4 and JDK 8, **to JSR-310 java.time types too**.

Repeatable annotations

```
@Schedules({
      @Scheduled(cron = "0 0 12 * * ?"),
      @Scheduled(cron = "0 0 18 * * ?")
})
public void performTempFileCleanup() {
}
```

```
@Scheduled(cron = "0 0 12 * * ?")
@Scheduled(cron = "0 0 18 * * ?")
public void performTempFileCleanup() {
}
```

Parameter name discovery

Why do I have to repeat myself?

```
@Query("select c from User u where c.lastname = :lastname")
List<User> findByLastname(@Param("lastname") String lastname);

@RequestMapping(value = "/person/get/{id}")
public Person getPerson(@PathVariable("id") Long id) {
```

Parameter name discovery

Old times:

If you compiled your code with **-debug**, parameter names would be preserved.

With Java 7 and earlier the –debug option does **not preserve parameter names on abstract methods**.

Therefore for projects like Spring Data that auto generate repository implementations based on Java interfaces this caused a problem.

Parameter name discovery

Method Parameter Reflection (RFE: JDK-8004841):

You can obtain the names of the formal parameters of any method or constructor with the method java.lang.reflect.Executable.getParameters.

However, .class files do not store formal parameter names by default.

To store formal parameter names in a particular .class file, and thus enable the Reflection API to retrieve formal parameter names, **compile the source file with the -parameters option** of the javac compiler.

Parameter name discovery

No -parameters

```
@Query("select c from User u where c.lastname = :lastname")
List<User> findByLastname(@Param("lastname") String lastname);
```

With -parameters

```
@Query("select c from User u where c.lastname = :lastname")
List<User> findByLastname(@Param String lastname);
```

Would work also without @Param, but be position-based.

Works with Spring MVC too

```
@RequestMapping(value = "/person/get/{id}")
public Person getPerson(@PathVariable Long id) {
```

Parameter name discovery

How it works now:

1. checking Java 8 (-parameters)
2. ASM-based reading of debug symbols (-debug)

Optional

Optional supported in autowiring, so these work the same

```
@Autowired(required=false)
UserService userService;

@Autowired
Optional<UserService> userService;
```

Optional

And in MVC it also works

```
@RequestMapping(value = "/accounts/{accountId}")
void update(Optional<String> accountId, @RequestBody Account account) {
}
```

CGLIB improvement

It used to be like this:

```
@Service
@Transactional
public class SomeLazyBastard {

   protected SomeLazyBastard() {}

   @Autowired
   public SomeLazyBastard(UserService u, PasswordService p) {...}
}
```

CGLIB improvement

Creating CGLIB proxies using Objenesis, not invoking any constructor

```
@Service @Transactional
public class SomeLazyBastard {
  @Autowired
  public SomeLazyBastard(UserService u, PasswordService p) { }
}
```

It’s repackaged inside spring:

```
import org.objenesis.instantiator.ObjectInstantiator
import org.springframework.objenesis.instantiator.ObjectInstantiator
```

Composable annotations with overridable attributes

```
@Service
@Scope("request")
@Transactional
@Retention(RetentionPolicy.RUNTIME)
@interface TransactionalService {
}

@TransactionalService
public class UserTransactionalService {
}
```

Notable mentions

Lambda expressions - just do it

Method references - because lambda

WebSockets with @MessageMapping: transparent SockJS fallback option, STOMP for higher-level messaging on top of a WebSocket channel

XML coexistence with JavaConfig

organic vintage retro xml before it was uncool
You probably never heard of it

Setting it up

```
@Configuration
@ImportResource("classpath:/com/acme/properties-config.xml")
public class AppConfig {

…
}
```

Use xml schemas

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd">

      <!-- beans go here -->

</beans>
```

XML configuration (ref and value)

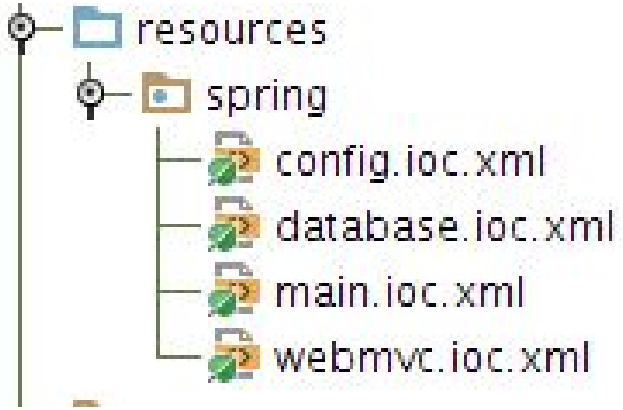
```
<bean id="templateResolver"
class="org.thymeleaf.templateresolver.ServletContextTemplateResolver">
    <property name="prefix" value="/WEB-INF/templates/" />
    <property name="suffix" value=".html" />
    <property name="templateMode" value="HTML5" />
    <property name="characterEncoding" value="UTF-8" />
</bean>

<bean id="templateEngine" class="org.thymeleaf.spring3.SpringTemplateEngine">
    <property name="templateResolver" ref="templateResolver" />
</bean>

<bean class="org.thymeleaf.spring3.view.ThymeleafViewResolver">
    <property name="templateEngine" ref="templateEngine" />
    <property name="characterEncoding" value="UTF-8" />
    <property name="order" value="1" />
</bean>
```

Importing xml from xml

```
</beans>
  <import resource="database.ioc.xml"/>
  <import resource="config.ioc.xml"/>
</beans>
```



Injecting via constructor

```
<bean class="eu.solidcraft.registration.MyWorkshopsController">
   <constructor-arg name="loggedUserRepository" ref="loggedUserInSessionRepository"/>
   <constructor-arg name="workshopRepository" ref="workshopRepository"/>
</bean>
```

```
<bean class="eu.solidcraft.registration.MyWorkshopsController">
   <constructor-arg index="0" ref="loggedUserInSessionRepository"/>
   <constructor-arg index="1" ref="workshopRepository"/>
</bean>
```

```
<bean class="eu.solidcraft.registration.MyWorkshopsController">
   <constructor-arg type="eu.solidcraft.registration.LoggedUserRepository"
         ref="loggedUserInSessionRepository"/>
   <constructor-arg type="eu.solidcraft.registration.WorkshopRepository"
         ref="workshopRepository"/>
</bean>
```

Injecting via setters

```
<bean class="eu.solidcraft.registration.MyWorkshopsController">
   <property name="loggedUserRepository" ref="loggedUserInSessionRepository"/>
   <property name="workshopRepository" ref="workshopRepository"/>
</bean>
```

```
public class MyWorkshopsController {
   private LoggedUserRepository loggedUserRepository;
   private WorkshopRepository workshopRepository;

   public void setLoggedUserRepository(LoggedUserRepository loggedUserRepository) {
      this.loggedUserRepository = loggedUserRepository;
   }

   public void setWorkshopRepository(WorkshopRepository workshopRepository) {
      this.workshopRepository = workshopRepository;
   }
```

Shortcuts: p-namespace

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean name="john-classic" class="com.example.Person">
        <property name="name" value="John Doe"/>
        <property name="spouse" ref="jane"/>
    </bean>

    <bean name="john-modern" class="com.example.Person"
          p:name="John Doe"
          p:spouse-ref="jane"/>

    <bean name="jane" class="com.example.Person">
        <property name="name" value="Jane Doe"/>
    </bean>
</beans>
```

Shortcuts: c-namespace

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:c="http://www.springframework.org/schema/c"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="bar" class="x.y.Bar"/>
    <bean id="baz" class="x.y.Baz"/>

    <bean id="foo" class="x.y.Foo">
        <constructor-arg ref="bar"/>
        <constructor-arg ref="baz"/>
        <constructor-arg value="foo@bar.com"/>
    </bean>

    <bean id="foo" class="x.y.Foo" c:bar-ref="bar" c:baz-ref="baz" c:email="foo@bar.com">
</beans>
```

Corner cases - factory methods

```
<!-- static factory method on a class -->
<bean id="clientService"
    class="examples.ClientService"
    factory-method="createInstance"/>


<!-- factory method on an instance -->
<bean id="serviceLocator" class="examples.DefaultServiceLocator"/>

<bean id="clientService"
      factory-bean="serviceLocator"
      factory-method="createClientServiceInstance"/>
```

Inner beans, or keep some privacy

```
<bean id="templateResolver"
class="org.thymeleaf.templateresolver.ServletContextTemplateResolver">
    <property name="prefix" value="/WEB-INF/templates/" />
    <property name="suffix" value=".html" />
    <property name="templateMode" value="HTML5" />
    <property name="characterEncoding" value="UTF-8" />
</bean>

<bean id="templateEngine" class="org.thymeleaf.spring3.SpringTemplateEngine">
    <property name="templateResolver" ref="templateResolver" />
</bean>

<bean class="org.thymeleaf.spring3.view.ThymeleafViewResolver">
    <property name="templateEngine" ref="templateEngine" />
    <property name="characterEncoding" value="UTF-8" />
    <property name="order" value="1" />
</bean>
```

Inner beans, or keep some privacy

```
<bean id="templateResolver"
class="org.thymeleaf.templateresolver.ServletContextTemplateResolver">
   <property name="prefix" value="/WEB-INF/templates/" />
   <property name="suffix" value=".html" />
   <property name="templateMode" value="HTML5" />
   <property name="characterEncoding" value="UTF-8" />
</bean>

<bean class="org.thymeleaf.spring3.view.ThymeleafViewResolver">
   <property name="templateEngine" >
      <bean class="org.thymeleaf.spring3.SpringTemplateEngine">
         <property name="templateResolver" ref="templateResolver" />
      </bean>
   </property>
   <property name="characterEncoding" value="UTF-8" />
   <property name="order" value="1" />
</bean>
```

Inner beans, or keep some privacy

```
<bean class="org.thymeleaf.spring3.view.ThymeleafViewResolver">
   <property name="templateEngine" >
      <bean class="org.thymeleaf.spring3.SpringTemplateEngine">
         <property name="templateResolver" >
            <bean class="org.thymeleaf.templateresolver.ServletContextTemplateResolver">
               <property name="prefix" value="/WEB-INF/templates/" />
               <property name="suffix" value=".html" />
               <property name="templateMode" value="HTML5" />
               <property name="characterEncoding" value="UTF-8" />
            </bean>
         </property>
      </bean>
   </property>
   <property name="characterEncoding" value="UTF-8" />
   <property name="order" value="1" />
</bean>
```

Working with lists, sets and maps

```
<bean id="moreComplexObject" class="example.ComplexObject">
   <property name="someList">
      <list>
         <value>a list element followed by a reference</value>
         <ref bean="myDataSource" />
      </list>
   </property>
   <property name="someMap">
      <map>
         <entry key="an entry" value="just some string"/>
         <entry key ="a ref" value-ref="myDataSource"/>
      </map>
   </property>
   <property name="someSet">
      <set>
         <value>just some string</value>
         <ref bean="myDataSource" />
      </set>
   </property>
</bean>
```

Bean definition inheritance

```
<bean id="a" abstract="true">
   <property name="name" value="parent"/>
   <property name="age" value="1"/>
</bean>

<bean id="b"   class="B"  parent="a" >
   <property name="name" value="override"/>
</bean>

<bean id="c" class="C" parent="a" >
   <property name="name" value="override2"/>
</bean>
```

SpEL: Spring Expression Language

because you always wanted to program in strings without compiler support
(but with IDE support, ufff...)

SpEL: theory

```
ExpressionParser parser = new SpelExpressionParser();
Expression exp = parser.parseExpression(
      "new String('hello world').toUpperCase()");
String message = exp.getValue(String.class);
```

SpEL: simple example

SpEL expressions can be used with XML or annotation-based configuration metadata for defining BeanDefinitions. In both cases the syntax to define the expression is of the form #{ <expression string> }.

```
@Autowired
public SomeLazyBastard(@Value("#{ T(java.lang.Math).random() * 100.0 }")
                                                    String random) {
  //I can haz a random value
}
```

SpEL: simple example

Works on fields, methods, method/constructor params and in XML

SpEL: root object

The more common usage of SpEL is to provide an expression string that is evaluated against a specific object instance (called the root object)

```
Inventor tesla = new Inventor("Nikola Tesla", c.getTime(), "Serbian");
```

Root setup once with context:

```
ExpressionParser parser = new SpelExpressionParser();
Expression exp = parser.parseExpression("name");
EvaluationContext context = new StandardEvaluationContext(tesla);
String name = (String) exp.getValue(context);
```

Root changed with each evaluation:

```
ExpressionParser parser = new SpelExpressionParser();
Expression exp = parser.parseExpression("name");
String name = (String) exp.getValue(tesla);    }
```

SpEL: anatomy

Literal expressions:

```
ExpressionParser parser = new SpelExpressionParser();
String helloWorld =
        (String) parser.parseExpression("'Hello World'").getValue();
double avogadrosNumber =
        (Double) parser.parseExpression("6.0221415E+23").getValue();
int maxValue = (Integer) parser.parseExpression("0x7FFFFFFF").getValue();
boolean trueValue = (Boolean) parser.parseExpression("true").getValue();
Object nullValue = parser.parseExpression("null").getValue();
```

SpEL: anatomy

Maps, properties, lists, etc.:

```
"Birthdate.Year + 1900"
"placeOfBirth.City"
"inventions[3]"
"Members[0].Name"
"Members[0].Inventions[6]"
"Officers['president'].PlaceOfBirth.City"
```

Inline lists:

```
"{1,2,3,4}"
"{{'a','b'},{'x','y'}}"
```

Inline maps:

```
"{name:{first:'Nikola',last:'Tesla'}, dob:{year:1856}}"
```

SpEL: anatomy

Array construction

```
"new int[]{1,2,3}"
"new int[4]"
"new int[4][5]"
```

Methods:

```
"'abc'.substring(2, 3)"
```

Operators:

```
"2 < -5.0"
"'xyz' instanceof T(int)"
"'5.00' matches '^-?\\d+(\\.\\d{2})?$'"
```

SpEL: anatomy

Logical operators + methods

```
isMember('Nikola Tesla') and !isMember('Mihajlo Pupin')
```

```
"T(java.math.RoundingMode).CEILING <
                              T(java.math.RoundingMode).FLOOR"
```

Also Ternary (a ? b : c), Elvis (a?:b) and safe (a.?m()) operators.

SpEL: anatomy

Ternary

#{@userService.isUserLogged() ? ‘logged’ : ‘anonymous’}

Elvis

#{@userService.isUserLogged() ?:

& Safe operators

SpEL: root object in practice

For Spring beans, the parser, evaluation context, root object and any predefined variables are all set up implicitly, requiring the user to specify nothing other than the expressions

But...

The StandardEvaluationContext is where you may specify the root object to evaluate against via the method setRootObject() or passing the root object into the constructor

SpEL: practice

```
@PreAuthorize("hasPermission(#contact, 'admin')")
public void deletePermission(Contact contact,
                             Sid recipient, Permission permission);
```

SpEL: practice

It is possible to lookup beans from an expression using the (@) symbol.

```
@Autowired
public SomeLazyBastard(
        @Value("#{@userServiceForAspect.getLoggedUserDetails().login}")
        String login) {
        ...
}
```

SpEL: practice

You can extend SpEL by registering user defined functions that can be called within the expression string. Set this up on StandardEvaluationContext.registerFunction(name, method)

```
@PreAuthorize("isAdmin(#contact)")
public void deletePermission(Contact contact,
                             Sid recipient, Permission permission);
```

or just setup your own Permission evaluator, because it's easier:

http://blog.solidcraft.eu/2011/03/spring-security-by-example-securing.html