Spring AOP Part 2

Originals of Slides and Source Code for Examples:
http://courses.coreservlets.com/Course-Materials/spring.html

Topics in This Section

- Implementing aspect behavior
- AspectJ APIs and annotations
- Spring AOP application

Aspect Behavior

Aspect Behavior

- **Advice**
  - Behavior to be applied to a set of program execution points
  - In AOP terms, advice encapsulates a cross-cutting interest, e.g. transaction management. Advisor beans are applied to pointcuts (a set of join points)
- **Spring advisor bean**
  - Implementation
    - POJOs encoding advice
  - Integration (one of the following)
    - Special interfaces `org.aopalliance.aop.Advice`
    - Methods annotated with AspectJ annotations and defined with AspectJ parameters types

Advice Types

- **Before**
  - Non-critical advisor bean type
  - Called before method execution
- **After returning**
  - Non-critical advisor bean type
  - Called after normal method execution
- **After throwing**
  - Non-critical advisor bean type
  - Called after method execution exits with an exception
- **Around**
  - Critical advisor bean type
  - Wraps method execution

Before Advice

Before Advice

- **Interface**
  - `org.springframework.aop.MethodBeforeAdvice`
- **Execution point**
  - Before method execution
- **Type**
  - Does not invoke method
  - Non-critical unless an error is thrown

Before Advice Guidelines

- **Uses**
  - Input validation
  - Auditing/logging
- **Exception type**
  - Checked exceptions must be coordinated with the error signature of the advised bean
    - Out-of-scope errors are re-thrown as `java.lang.reflect.UndeclaredThrowableException`
  - **`RuntimeException`** types may be used without precaution

Before Advice Process

- **Create new advice class**
  - Implement
    - `org.springframework.aop.MethodBeforeAdvice`
  - Fulfill

    -
      ```
      before(method:Method,
             arguments:Object[],
             target:Object):void
            throws Throwable
      ```
- **Register advice as a Spring bean**
  - **`<bean/>`**
- **Reference from aspect**
  - Associate with a pointcut
- **Integrate with Spring domain beans**

Create Before Advice Class

```
import java.lang.reflect.Method;
import org.springframework.aop.MethodBeforeAdvice;

public class BeforeLoggingAdvice
implements MethodBeforeAdvice {

 public void before(Method method,
                    Object[] args,
                    Object target) throws Throwable {

    Logger.getLogger(target.getClass()).debug(
      target.getClass().getSimpleName()
        + "#" + method.toGenericString()
        + ". args=" + Arrays.toString(args));

  }
}
```

Register Before Advice Bean

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">

  <bean id="beforeLoggingAdvice"
        class="coreservlets.BeforeLoggingAdvice" />

</beans>
```

Reference From Aspect

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">

  <bean id="beforeLoggingAdvice"
        class="coreservlets.BeforeLoggingAdvice" />
  <aop:config>
    <aop:pointcut id="customerQueryPointcut"
      expression="execution(* coreservlets.CustomerQuery.*(..))" />
    <aop:advisor advice-ref="beforeLoggingAdvice"
                 pointcut-ref="customerQueryPointcut" />
  </aop:config>
</beans>
```

Advisor reference

Domain Beans

- **classpath:/coreservletsContext.xml**

```
<beans>
  <bean id="customerQuery" class="coreservlets.MockCustomerQuery">
    <constructor-arg>
      <list>
        <bean class="coreservlets.Customer">
          <property name="id" value="jjoe" />
          <property name="name" value="Java Joe" />
        </bean>
        <bean class="coreservlets.Customer">
          <property name="id" value="jjohn" />
          <property name="name" value="Java John" />
        </bean>
      </list>
    </constructor-arg>
  </bean>
</beans>
```

Integrate Advice with Domain Beans

```
import org.springframework.context.support.*;

public class Main {
  public static void main(String[]args) {

    BeanFactory beanFactory =
      new ClassPathXmlApplicationContext(
        "/coreservletsContext.xml",
        "/coreservletsAopContext.xml");
    ...
  }
}
```

Access and Use Beans

```
import org.springframework.context.support.*;

public class Main {

  public static void main(String[]args) {
    BeanFactory beanFactory =
      new ClassPathXmlApplicationContext(new String[]{
        "/coreservletsContext.xml",
        "/coreservletsAopContext.xml"});
    CustomerQuery query =
     (CustomerQuery) beanFactory.getBean("customerQuery");
    Customer customer = query.getCustomerByName("Java Joe");
  }
}
```

Standard output

```
MockCustomerQuery#public abstract coreservlets.Customer
coreservlets.CustomerQuery.getCustomerByName(java.lang.S
tring). args=[Java Joe]
```

After Returning Advice

After Returning Advice

- **Interface**
  - **`org.springframework.aop.AfterReturningAdvice`**
- **Execution point**
  - After normal method execution and exit
- **Type**
  - Does not invoke method
  - Non-critical unless an error is thrown
- **Exception type**
  - Checked exceptions must be coordinated with the error signature of the advised bean
    - Out-of-scope errors are re-thrown as `java.lang.reflect.UndeclaredThrowableException`
  - **`RuntimeException`** types may be used without precaution

After Returning Advice

- **Create new advice class**
  - Implement
    - `org.springframework.aop.AfterReturningAdvice`
  - Fulfill
    - `afterReturning(`**`returnValue:Object,`**

```
          method:Method,
         arguments:Object[],
         target:Object):void
         throws Throwable
```

- **Register advice as a Spring bean**
  - `<bean/>`
- **Reference from aspect**
  - Associate with a pointcut
- **Integrate with Spring domain beans**

After Returning Advice Class

```
import java.lang.reflect.Method;
import org.springframework.aop. AfterReturningAdvice;

public class BeforeLoggingAdvice
implements AfterReturningAdvice {

 public void afterReturning(Object returnValue,
                            Method method,
                            Object[] args,
                            Object target) throws Throwable {

    Logger.getLogger(target.getClass()).debug(
      "exit=return[" + returnValue + "]");

  }
}
```

Register After Returning Advice Bean

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">

  <bean id="afterReturningLoggingAdvice"
        class="coreservlets.AfterReturnLoggingAdvice" />

</beans>
```

Reference From Aspect

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">

  <bean id="afterReturningLoggingAdvice"
        class="coreservlets.AfterReturnLoggingAdvice" />
  <aop:config>
    <aop:pointcut id="customerQueryPointcut"
      expression="execution(* coreservlets.CustomerQuery.*(..))" />
    <aop:advisor advice-ref="afterReturningLoggingAdvice"
                 pointcut-ref="customerQueryPointcut" />
  </aop:config>
</beans>
```

Advisor reference

Domain Beans

- `classpath:/coreservletsContext.xml`

```
<beans>
  <bean id="customerQuery" class="coreservlets.MockCustomerQuery">
    <constructor-arg>
      <list>
        <bean class="coreservlets.Customer">
          <property name="id" value="jjoe" />
          <property name="name" value="Java Joe" />
        </bean>
        <bean class="coreservlets.Customer">
          <property name="id" value="jjohn" />
          <property name="name" value="Java John" />
        </bean>
      </list>
    </constructor-arg>
  </bean>
</beans>
```

Integrate Advice with Domain Beans

```
import org.springframework.context.support.*;

public class Main {
  public static void main(String[]args) {

    BeanFactory beanFactory =
      new ClassPathXmlApplicationContext(
        "/coreservletsContext.xml",
        "/coreservletsAopContext.xml");
    ...
  }
}
```

Access and Use Beans

```
import org.springframework.context.support.*;

public class Main {

  public static void main(String[]args) {
    BeanFactory beanFactory =
      new ClassPathXmlApplicationContext(new String[]{
        "/coreservletsContext.xml",
        "/coreservletsAopContext.xml"});
    CustomerQuery query =
     (CustomerQuery) beanFactory.getBean("customerQuery");
    Customer customer = query.getCustomerByName("Java Joe");
  }
}
```

Standard output

```
exit=return[Customer id=jjoe, name=Java Joe]
```

Throws Advice

Throws Advice

- **Interface**
  - **`org.springframework.aop.ThrowsAdvice`**
  - Interface is an empty marker with virtualized callbacks
- **Implementation**
  - Implementations must conform to the following signature
    - `afterThrowing(throwable:Throwable):void`

    -
      ```
      afterThrowing(method:Method,
                    args:Object[],
                    target:Object,
                    throwable:Throwable);
      ```
- **Execution point**
  - After method exit on error
  - Does not intercept errors originating from preceding advisors
- **Type**
  - Does not invoke method
  - Non-critical unless an error is thrown

Throws Advice Guidelines

- **Uses**
  - Adapting API error behavior
  - Error conformance to a domain type
- **Overriding**
  - Exception may be overridden by re-throwing an alternate error
- **Exception type**
  - Checked exceptions must be coordinated with the error signature of the advised bean
    - Out-of-scope errors are re-thrown as `java.lang.reflect.UndeclaredThrowableException`
  - **`RuntimeException`** types may be used without precaution

Throws Advice Guidelines

- **Overloading by type**
  - Advice bean may overload methods to support varying error types
  - The most precise advising method is selected per the Throwable type
- **Overloading by detail**
  - Approach is ambiguous
  - Full method implementation is selected

Throws Advice Process

- **Create new advice class**
  - Implement
    - `org.springframework.aop.ThrowsAdvice`
  - Define one or more interceptor methods
    - `afterThrowing(throwable:Throwable):void`

    -
      ```
      afterThrowing(method:Method,
                    args:Object[],
                    target:Object,
                    throwable:Throwable);
      ```
- **Register advice as a Spring bean**
  - **`<bean/>`**
- **Reference from aspect**
  - Associate with a pointcut
- **Integrate with Spring domain beans**

Throws Advice Class

```
import java.lang.reflect.Method;
import org.springframework.aop.ThrowsAdvice;
public class ThrowsLoggingAdvice implements ThrowsAdvice {
   public void afterThrowing(Method method
       , Object[] arguments
       , Object target
       , IllegalArgumentException ex) throws Throwable {
    Logger.getLogger(target.getClass()).debug(
      "afterThrowing: IllegalArgumentException");
  }
  public void afterThrowing(Method method
       , Object[] arguments
       , Object target
       , IllegalStateException ex) throws Throwable {
    Logger.getLogger(target.getClass()).debug(
      "afterThrowing: IllegalStateException");
  }
}
```

Register Throws Advice Bean

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">

  <bean id="throwsLoggingAdvice"
        class="coreservlets.ThrowsLoggingAdvice" />

</beans>
```

Reference From Aspect

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">

  <bean id="throwsLoggingAdvice"
        class="coreservlets.ThrowsLoggingAdvice" />
  <aop:config>
    <aop:pointcut id="customerQueryPointcut"
      expression="execution(* coreservlets.CustomerQuery.*(..))" />
    <aop:advisor advice-ref="throwsLoggingAdvice"
                 pointcut-ref="customerQueryPointcut" />
  </aop:config>
</beans>
```

Advisor reference

Mock Domain Class

```
public class ErrorThrowingMockCustomerQuery
implements CustomerQuery {

  private Class<? extends RuntimeException>throwableType;

  public ErrorThrowingMockCustomerQuery (
      Class<? extends RuntimeException>throwableType){
    this.throwableType = throwableType;
  }

  public Customer getCustomerByName(String name) {
    try{
      throw throwableType.newInstance();
    }
    catch(InstantiationException e){
      ...
    }
}
```

Domain Beans

- **classpath:/coreservletsContext.xml**

```
<?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-
   2.5.xsd">

  <bean id="customerQuery"
        class="coreservlets.ErrorThrowingMockCustomerQuery">
    <constructor-arg
      value="java.lang.IllegalArgumentException" />
  </bean>

</beans>
```

Integrate Advice with Domain Beans

```
import org.springframework.context.support.*;

public class Main {
  public static void main(String[]args) {

    BeanFactory beanFactory =
      new ClassPathXmlApplicationContext(
        "/coreservletsContext.xml",
        "/coreservletsAopContext.xml");
    ...
  }
}
```

Access and Use Beans

```
import org.springframework.context.support.*;
public class Main {
  public static void main(String[]args) {
    BeanFactory beanFactory =
      new ClassPathXmlApplicationContext(new String[]{
        "/coreservletsContext.xml",
        "/coreservletsAopContext.xml"});
    CustomerQuery query =
     (CustomerQuery) beanFactory.getBean("customerQuery");
    Customer customer = query.getCustomerByName("Java Joe");
  }
}
```

Standard output

```
afterThrowing: IllegalArgumentException
Exception in thread "main"
java.lang.IllegalArgumentException
```

Around Advice

Around Advice

- **Interface**
  - **`org.springframework.aop.MethodInterceptor`**
- **Execution point**
  - Wraps method invocation
  - Wraps other advisors
    - Excluding other *Around* type advisors with greater `order` index values
    - Perceives return types and errors from target beans and other advisors
- **Type**
  - Invokes method
  - Critical path

Around Advice Guidelines

- **Uses**
  - Transaction management
  - Full templating cases
- **Overriding**
  - Exception may be overridden by re-throwing an alternate error
  - Exceptions may also be suppressed
- **Exception type**
  - Checked exceptions must be coordinated with the error signature of the advised bean
    - Out-of-scope errors are re-thrown as `java.lang.reflect.UndeclaredThrowableException`
  - **`RuntimeException`** types may be used without precaution

Around Advice Process

- **Create new advice class**
  - Implement
    - `org.springframework.aop.MethodInterceptor`
  - Fulfill
    - `invoke(handle:MethodInvocation):Object`
      `throws Throwable`
  - Invoke target
    - `return handle.proceed();`
- **Register advice as a Spring bean**
  - **`<bean/>`**
- **Reference from aspect**
  - Associate with a pointcut
- **Integrate with Spring domain beans**

Around Advice Class

```
import org.aopalliance.intercept.MethodInterceptor;
import org.aopalliance.intercept.MethodInvocation;

public class NamedAroundAdvice
implements MethodInterceptor {

  private String name;

  public NamedAroundAdvice(String name){
    this.name = name;
  }

  public Object invoke(MethodInvocation invocation)
  throws Throwable {
    ...
  }
}
```

Around Advice Class Continued

```
...
  public Object invoke(MethodInvocation invocation)
  throws Throwable {
    try{
      log.debug("before: " + this.name);
      Object returnValue = invocation.proceed();
      log.debug("after return: " + this.name);
      return returnValue;
    }
    catch(Throwable t){
      log.debug("after throws: " + this.name);
      throw t;
    }
    finally{
      log.debug("after finally: " + this.name);
    }
  }
...
```

Register Around Advice Bean

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">

  <bean id="namedAroundAdvice"
        class="coreservlets.NamedAroundAdvice">
    <property name="name" value="namedAroundAdvice" />
  </bean>

</beans>
```

Reference From Aspect

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">
  <bean id="namedAroundAdvice"
        class="coreservlets.NamedAroundAdvice">
    <property name="name" value="namedAroundAdvice" />
  </bean>
  <aop:config>
    <aop:pointcut id="customerQueryPointcut"
      expression="execution(* coreservlets.CustomerQuery.*(..))" />
    <aop:advisor advice-ref="namedAroundAdvice"
                 pointcut-ref="customerQueryPointcut" />
  </aop:config>
</beans>
```

Advisor reference

Domain Beans

- `classpath:/coreservletsContext.xml`

```
<beans>
  <bean id="customerQuery" class="coreservlets.MockCustomerQuery">
    <constructor-arg>
      <list>
        <bean class="coreservlets.Customer">
          <property name="id" value="jjoe" />
          <property name="name" value="Java Joe" />
        </bean>
        <bean class="coreservlets.Customer">
          <property name="id" value="jjohn" />
          <property name="name" value="Java John" />
        </bean>
      </list>
    </constructor-arg>
  </bean>
</beans>
```

Integrate Advice with Domain Beans

```
import org.springframework.context.support.*;

public class Main {
  public static void main(String[]args) {

    BeanFactory beanFactory =
      new ClassPathXmlApplicationContext(
        "/coreservletsContext.xml",
        "/coreservletsAopContext.xml");
    ...
  }
}
```

Access and Use Beans

```
import org.springframework.context.support.*;
public class Main {
  public static void main(String[]args) {
    BeanFactory beanFactory =
      new ClassPathXmlApplicationContext(new String[]{
        "/coreservletsContext.xml",
        "/coreservletsAopContext.xml"});
    CustomerQuery query =
     (CustomerQuery) beanFactory.getBean("customerQuery");
    Customer customer = query.getCustomerByName("Java Joe");
  }
}
```

Standard output

```
before: namedAroundAdvice
after return: namedAroundAdvice
after finally: namedAroundAdvice
```

Advice Ordering

Around Advice

- **Controls**
  - Aspect execution order among similar advice types
- **Configuration**
  - **`aop:advisor`** attribute **`order`**

Around Advice Class

```
public class NamedAroundAdvice implements MethodInterceptor {
  ...
  public Object invoke(MethodInvocation inv) throws Throwable
   {
    try{
      log.debug("before: " + this.name);
      Object returnValue = inv.proceed();
      log.debug("after return: " + this.name);
      return returnValue;
    }
    catch(Throwable t){
      log.debug("after throws: " + this.name);
      throw t;
    }
    finally{
      log.debug("after finally: " + this.name);
    }
  }
}
```

Configure Aspect Order

```
<?xml version="1.0" encoding="UTF-8"?>
<beans>
  <bean id="advisor-0" class="coreservlets.NamedAroundAdvice">
     <property name="name" value="advisor-0" />
  </bean>
  <bean id="advisor-1" class="coreservlets.NamedAroundAdvice">
     <property name="name" value="advisor-1" />
  </bean>
  <aop:config>
    <aop:pointcut id="customerQueryPointcut"
      expression="execution(* coreservlets.CustomerQuery.*(..))" />

    <aop:advisor advice-ref="advisor-0" order="0"
                 pointcut-ref="customerQueryPointcut" />

    <aop:advisor advice-ref="advisor-1" order="1"
                 pointcut-ref="customerQueryPointcut" />
  </aop:config>
</beans>
```

Access and Use Beans

```
import org.springframework.context.support.*;
public class Main {
  public static void main(String[]args) {
    BeanFactory beanFactory =
      new ClassPathXmlApplicationContext(new String[]{
        "/coreservletsContext.xml",
        "/coreservletsAopContext.xml"});
    CustomerQuery query =
     (CustomerQuery) beanFactory.getBean("customerQuery");
    Customer customer = query.getCustomerByName("Java Joe");
  }
}
```

Standard output

```
before: advisor-0
before: advisor-1
after return: advisor-1
after finally: advisor-1
after return: advisor-0
after finally: advisor-0
```

AspectJ Pointcuts

AspectJ Pointcuts Introduction

- **Design**
  - Reuses class and method namespace for enumerating pointcut definitions
  - Substitute for Spring AOP XML schema support
    `<aop:config><aop:pointcut/></aop:config>`
- **Pointcut class**
  - Pointcut definitions are aggregated into an annotated class
  - Classes containing pointcut elements are marked using the **`@Aspect`** annotation

AspectJ Pointcuts Introduction Continued

- **Pointcut element**
  - An annotated method, **`@Pointcut`**, represents a single pointcut definition
  - The class and method combination is the unique pointcut identifier
    - `<full classname>.<method name>()`
  - Pointcut methods are **`public`** instance methods, accept no arguments, and specify a **`void`** return type
- **Pointcut expression**
  - The pointcut definition is expressed as **`@Pointcut`** annotation content

AspectJ Pointcuts Process

- **Create new pointcut definitions class**
  - Annotate class with **`@Aspect`**
- **Define pointcuts**
  - Create a new and empty method for each pointcut
    - Annotate method with **`@Pointcut`**
  - Specify pointcut definition as **`@Pointcut`** annotation content
- **Create advice class**
  - Implement an AOP Alliance interface **`org.aopalliance.aop.Advice`**
- **Register advice as a Spring bean**
  - **`<bean/>`**
- **Reference advice and pointcut from aspect**
- **Integrate with Spring domain beans**

Select Join Points

```
public interface CustomerQuery {

  public Customer getCustomerByName(String name);

}

public interface CustomerReport {

  public String getReport(String customerName);

}
```

Select Join Points Continued

```
public class MockCustomerQuery implements CustomerQuery {

  private List<Customer> customers;

  public MockCustomerQuery(List<Customer> customers) {
    this.customers = customers != null
      ? customers : new ArrayList<Customer>();
  }

  public Customer getCustomerByName(String name) {
    for(Customer c : customers){
      if(c.getName().equals(name)){
        return c;
      }
    }
    return null;
  }
}
```

Select Join Points Continued

```
public class MockCustomerReport implements CustomerReport{

  private CustomerQuery query;

  public MockCustomerReport(CustomerQuery query){

    this.query = query;

  }

  public String getReport(String customerName){

    Customer customer =
      query.getCustomerByName(customerName);
    return customer != null
      ? customer.toString() : null;

  }
}
```

Create Pointcut Definitions Class

```
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Pointcut;

@Aspect
public class CoreservletsPointcuts {

}
```

Define Pointcuts

```
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Pointcut;

@Aspect
public class CoreservletsPointcuts {

  @Pointcut("target(coreservlets.CustomerQuery)")
  public void queryLayer(){}

  @Pointcut("target(coreservlets.CustomerReport)")
  public void reportLayer(){}

}
```

Create Advice Class

```
public class LoggingMethodAdvice
implements MethodInterceptor {
  public Object invoke(MethodInvocation i) throws Throwable {
    String buf = ...;
    try{
      Object returnValue = i.proceed();
      buf += "\n  - ex return : " + returnValue;
      return returnValue;
    }
    catch(Throwable t){
      buf += "\n  - ex error  : "
          + t.getClass().getName() + " - " + t.getMessage();
      throw t;
    }
    finally{
      Logger.getLogger(i.getThis().getClass()).debug(buf);
    }
  }
}
```

Register Advice Bean

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">

  <bean id="loggingMethodAdvice"
        class="coreservlets.LoggingMethodAdvice" />

</beans>
```

Define Aspect

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">
 <bean id="loggingMethodAdvice"
       class="coreservlets.LoggingMethodAdvice" />
 <aop:config>
  <aop:advisor advice-ref="loggingMethodAdvice"
   pointcut="coreservlets.CoreservletsPointcuts.reportLayer()"/>
  <aop:advisor advice-ref="loggingMethodAdvice"
   pointcut="coreservlets.CoreservletsPointcuts.queryLayer()"/>
 </aop:config>
</beans>
```

Domain Beans

- **classpath:/coreservletsContext.xml**

```
<beans>
  <bean id="customerReport"
        class="coreservlets.MockCustomerReport">
    <constructor-arg ref="customerQuery" />
  </bean>
  <bean id="customerQuery"
   class="coreservlets.MockCustomerQuery">
    <constructor-arg>
      <list>
        <bean class="coreservlets.Customer">
          <property name="id" value="jjoe" />
          <property name="name" value="Java Joe" />
        </bean>
      </list>
    </constructor-arg>
  </bean>
</beans>
```

Access and Use Beans

```
import org.springframework.context.support.*;
public class Main {
  public static void main(String[]args) {
    BeanFactory beanFactory = ...;
    CustomerReport reportService =
     (CustomerReport) beanFactory.getBean("customerReport");
    reportService.getReport("Java Joe");
  }
}
```

Standard output

```
LoggingMethodAdvice
  - target     : coreservlets.MockCustomerQuery
  - method     : getCustomerByName[class java.lang.String]
  - arg values: Java Joe
  - ex return : Customer id=jjoe, name=Java Joe
LoggingMethodAdvice
  - target     : coreservlets.MockCustomerReport
  - method     : getReport[class java.lang.String]
  - arg values: Java Joe
  - ex return : Customer id=jjoe, name=Java Joe
```

AspectJ Advice

AspectJ Advice Introduction

- **Advice class**
  - No special interfaces
  - Annotated with **`@Aspect`**
- **Advice method**
  - Annotated with advice classifier
    - `@Before`
    - `@AfterReturning`
    - `@AfterThrowing`
    - `@Around`
  - Access to **`JoinPoint`**, **`ProceedingJoinPoint`**, or **`JoinPoint.StaticPart`**
    - See `org.aspectj.lang.*`
  - Only around advice is critical to target execution

AspectJ Advice Introduction Continued

- **Annotation content**
  - Specifies pointcut definition as a pointcut expression
  - e.g., **`@Around("Pointcuts.layer()")`**
  - e.g., **`@Around(value="Pointcuts.layer()",`**
    `argNames="")`
- **Annotation property `argNames`**
  - Supplies advice method parameter names to pointcut
- **Configuration**
  - Replaces Spring AOP XML schema
  - Annotated advisor is a Spring bean
  - Scanned by a bean post processor
    - Requires element `<aop:aspectj-autoproxy />`

AspectJ Pointcuts Process

- **Create new pointcut definitions class**
  - Annotate class with **`@Aspect`**
- **Define pointcuts**
  - Create a new and empty method for each pointcut
    - Annotate method with `@Pointcut`
  - Specify pointcut definition as **`@Pointcut`** annotation content
- **Create advice class**
  - Annotate class with **`@Aspect`**
  - Annotate advice method with advice type classifying annotation
    - e.g. `@Around`
  - Specify pointcut reference as advice type annotation content
    - e.g. `@Around("Pointcuts.layer()")`

AspectJ Pointcuts Process Continued

- **Skip `<aop:config/>`**
- **Register advice as a Spring bean**
  - **`<bean/>`**
  - Bean annotations already contains advice type and pointcut reference
  - **Indirectly references aspects and pointcuts**
- **Scan AspectJ annotations**
  - Add post processor instruction to bean definitions

    `<aop:aspectj-autoproxy />`
- **Integrate with Spring domain beans**

Select Join Points

```
public interface CustomerQuery {

  public Customer getCustomerByName(String name);

}

public interface CustomerReport {

  public String getReport(String customerName);

}
```

Select Join Points Continued

```
public class MockCustomerQuery implements CustomerQuery {

  private List<Customer> customers;

  public MockCustomerQuery(List<Customer> customers) {
    this.customers = customers != null
      ? customers : new ArrayList<Customer>();
  }

  public Customer getCustomerByName(String name) {
    for(Customer c : customers){
      if(c.getName().equals(name)){
        return c;
      }
    }
    return null;
  }
}
```

Select Join Points Continued

```
public class MockCustomerReport implements CustomerReport{

  private CustomerQuery query;

  public MockCustomerReport(CustomerQuery query){

    this.query = query;

  }

  public String getReport(String customerName){

    Customer customer =
      query.getCustomerByName(customerName);
    return customer != null
      ? customer.toString() : null;

  }
}
```

Define Pointcuts

```
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Pointcut;

@Aspect
public class CoreservletsPointcuts {

  @Pointcut("target(coreservlets.CustomerQuery)")
  public void queryLayer(){}

  @Pointcut("target(coreservlets.CustomerReport)")
  public void reportLayer(){}

}
```

Create Advice Class

```
import org.apache.log4j.Logger;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;

@Aspect
public class LoggingAroundAdvice {

}
```

Create Advice Class

```
@Aspect
public class LoggingAroundAdvice {
  @Around("coreservlets.CoreservletsPointcuts.queryLayer()"
    + "|| coreservlets.CoreservletsPointcuts.reportLayer()")
  public Object log(ProceedingJoinPoint jp)throws Throwable {
    Logger log = Logger.getLogger(jp.getTarget().getClass());
    try{
      log.debug("before");
      log.debug("#" + jp.getSignature().getName() + "()");
      Object returnValue = jp.proceed();
      log.debug("after return");
      return returnValue;
    }
    catch(Throwable t){
      log.debug("after throws");
      throw t;
    }
  }
}
```

Register Advice Bean

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">

  <bean id="loggingAroundAdvice"
        class="coreservlets.LoggingAroundAdvice" />

</beans>
```

Register Bean Postprocessor

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">

  <bean id="loggingAroundAdvice"
        class="coreservlets.LoggingAroundAdvice" />

  <aop:aspectj-autoproxy/>

</beans>
```

Domain Beans

- **classpath:/coreservletsContext.xml**

```
<beans>
  <bean id="customerReport"
        class="coreservlets.MockCustomerReport">
    <constructor-arg ref="customerQuery" />
  </bean>
  <bean id="customerQuery"
   class="coreservlets.MockCustomerQuery">
    <constructor-arg>
      <list>
        <bean class="coreservlets.Customer">
          <property name="id" value="jjoe" />
          <property name="name" value="Java Joe" />
        </bean>
      </list>
    </constructor-arg>
  </bean>
</beans>
```

Access and Use Beans

```
import org.springframework.context.support.*;
public class Main {
  public static void main(String[]args) {
    BeanFactory beanFactory = ...;
    CustomerReport reportService =
     (CustomerReport) beanFactory.getBean("customerReport");
    reportService.getReport("Java Joe");
  }
}
```

Standard output

```
coreservlets.MockCustomerReport before
coreservlets.MockCustomerReport #getReport()
coreservlets.MockCustomerQuery before
coreservlets.MockCustomerQuery #getCustomerByName()
coreservlets.MockCustomerQuery after return
coreservlets.MockCustomerReport after return
```

AspectJ Advice with Spring AOP XML Schema

AspectJ Advice with Spring AOP XML Schema

- **Advice class**
  - No special interfaces
  - No annotations
- **Advice method**
  - Uses AspectJ API **`JoinPoint`**, **`ProceedingJoinPoint`**, or **`JoinPoint.StaticPart`**
    - See `org.aspectj.lang.*`
  - Only around advice is critical to target execution
- **Pointcut**
  - Defined by Spring AOP XML Schema
    - `<aop:config><aop:pointcut /></aop:config>`
- **Aspect**
  - Defined by Spring AOP XML Schema
    - `<aop:config><aop:aspect/><aop:config/>`

Process

- **Create advice class**
  - Optionally specify a join point parameter
    - Required for around advice type
  - Specify return type for around advice type
- **Register advice bean**
  - `<bean/>`
- **Define pointcut**
  - `<aop:config><aop:pointcut /></aop:config>`
    - Declare pointcut ID and expression
- **Define aspect**
  - `<aop:config><aop:aspect/></aop:config>`
    - Reference pointcut definition
    - Reference advisor bean

Create Advice Class

```
import org.apache.log4j.Logger;
import org.aspectj.lang.ProceedingJoinPoint;
public class LoggingAroundAdvice {
  public Object log(ProceedingJoinPoint jp)throws Throwable {
    Logger log = Logger.getLogger(jp.getTarget().getClass());
    try{
      log.debug("before");
      log.debug("#" + jp.getSignature().getName() + "()");
      Object returnValue = joinPoint.proceed();
      log.debug("after return");
      return returnValue;
    }
    catch(Throwable t){
      log.debug("after throws");
      throw t;
    }
  }
}
```

Register Advice Bean

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">

  <bean id="loggingAroundAdvice"
        class="coreservlets.LoggingAroundAdvice" />

</beans>
```

Define Pointcut

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">
  <bean id="loggingAroundAdvice"
        class="coreservlets.LoggingAroundAdvice" />
  <aop:config>
    <aop:pointcut id="allLayers"
                  expression="target(coreservlets.CustomerQuery)
                           || target(coreservlets.CustomerReport)"/>
  </aop:config>
</beans>
```

Define Pointcut

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">
  <bean id="loggingAroundAdvice"
        class="coreservlets.LoggingAroundAdvice" />
  <aop:config>
    <aop:pointcut id="allLayers"
               expression="target(coreservlets.CustomerQuery)
                         || target(coreservlets.CustomerReport)"/>
    <aop:aspect ref="loggingAroundAdvice">
      <aop:around pointcut-ref="allLayers" method="log" />
    </aop:aspect>
  </aop:config>
</beans>
```

Domain Beans

- **classpath:/coreservletsContext.xml**

```
<beans>
  <bean id="customerReport"
        class="coreservlets.MockCustomerReport">
    <constructor-arg ref="customerQuery" />
  </bean>
  <bean id="customerQuery"
   class="coreservlets.MockCustomerQuery">
    <constructor-arg>
      <list>
        <bean class="coreservlets.Customer">
          <property name="id" value="jjoe" />
          <property name="name" value="Java Joe" />
        </bean>
      </list>
    </constructor-arg>
  </bean>
</beans>
```

Access and Use Beans

```
import org.springframework.context.support.*;
public class Main {
  public static void main(String[]args) {
    BeanFactory beanFactory = ...;
    CustomerReport reportService =
     (CustomerReport) beanFactory.getBean("customerReport");
    reportService.getReport("Java Joe");
  }
}
```

Standard output

```
coreservlets.MockCustomerReport before
coreservlets.MockCustomerReport #getReport()
coreservlets.MockCustomerQuery before
coreservlets.MockCustomerQuery #getCustomerByName()
coreservlets.MockCustomerQuery after return
coreservlets.MockCustomerReport after return
```

Spring AOP Application

JDBC Transaction Management using AOP

- **Application**
  - Transaction management over DAO persistence methods
- **Elements**
  - DataSource
    - Provides java.sql.Connection access/factory API
    - Integrates with RDBMS
  - DAO beans
    - Identifies candidate transaction boundaries
  - PlatformTransactionManager
    - Implements JDBC transaction management algorithms; e.g. JTA

JDBC Transaction Management using AOP

- **Spring AOP**
  - Advice
    - Abstracts transaction management services as a Spring AOP bean advisor
  - Pointcut
    - Identifies persistence methods exposed by DAO beans
  - Aspect
    - Associates transaction management bean advisor with a pointcut

Process

- **Develop persistence library**
  - `coreservlets.Customer`
  - `coreservlets.CustomerBatchPersistence`
  - `coreservlets.SpringJdbcCustomerBatchPersistence`
- **Register Spring IoC and AOP JARs**
  - spring-core.jar, spring-context.jar, spring-beans.jar, spring-aop.jar, aopalliance.jar, aspectjweaver.jar, cglib.jar, commons-logging.jar
- **Create the bean definitions file**
  - e.g., `classpath:/coreservletsPersistenceContext.xml`
- **Register persistence beans**
  - e.g., `<bean id="customerBatchPersistence"`
    `class="coreservlets.CustomerBatchPersistence"/>`
- **Inject dependencies**
  - e.g., `<bean>`
    `<constructor-arg ref="dataSource"/></bean>`

Process

- **Create advice**
  - Reuse advice implementation from **`spring-tx`**
    - spring-tx advice delegates transaction manager algorithms to a **`PlatformTransactionManager`** service
  - Register a **`PlatformTransactionManager`** bean and inject a DataSource
- **Create Spring AOP/TX definitions file**
  - **`classpath:/coreservletsTxContext.xml`**
  - Register **`spring-tx`** NS
    `http://www.springframework.org/schema/tx/spring-tx-2.5.xsd`
- **Register advice bean**
  - Create advisor bean declaration
    - Create `<tx:advice/>` element
    - Reference the PlatformTransactionManager and DataSource beans
    - Define transaction properties
      - Propagation (**`REQUIRED`**, **`REQUIRED_NEW`**, **`NESTED`**)
      - Read-only
      - Timeout

Process

- **Create pointcut definitions**
  - Specify program join points using pointcut definitions
  - e.g., **`execution(`**
    `* coreservlets.CustomerBatchPersistence.*(..))`
  - Create elements **`<aop:config><aop:pointcut/></aop:config>`**
- **Define aspect**
  - Reference advisor and pointcut
  - Create elements **`<aop:config><aop:advisor/></aop:config>`**
- **Initialize the container**
  - Initialize BeanFactory using all bean definitions
    - `classpath:/coreservletsPersistenceContext.xml`
    - `classpath:/coreservletsTxContext.xml`
    - `classpath:/coreservletsDataSourceContext.xml`
- **Access and use beans**
  - e.g., **`CustomerBatchPersistence dao =`**
    `(CustomerBatchPersistence) beanFactory.getBean();`

Develop Persistence Library

```
public interface CustomerBatchPersistence {

  public void insert(Customer...customers);

  public int getCustomerCount();

}
```

Develop Persistence Library

```
import org.springframework.jdbc.core.simple.*;

public class SpringJdbcCustomerBatchPersistence
implements CustomerBatchPersistence {

  private SimpleJdbcTemplate simpleJdbc;

  public SpringJdbcCustomerBatchPersistence(DataSource dataSource) {
    this.simpleJdbc = new SimpleJdbcTemplate(dataSource);
  }

  public int getCustomerCount(){
    return simpleJdbc.queryForInt("select count(*) from customer");
  }

  public void insert(Customer...customers) {
    ...
  }
}
```

Develop Persistence Library

```
import org.springframework.jdbc.core.simple.*;

public class SpringJdbcCustomerBatchPersistence
implements CustomerBatchPersistence {
  ...
  public void insert(Customer...customers) {
    if(customers == null){
      return;
    }
    for(Customer customer : customers){
      simpleJdbc.update(
        "insert into customer (id, name)"
        + " values (?, ?)",
        customer.getId(),
        customer.getName());
    }
  }
}
```

Select Join Points

```
public interface CustomerBatchPersistence {

  public void insert(Customer...customers);

  public int getCustomerCount();

}
```

Create Bean Definitions

- `classpath:/coreservletsPersistenceContext.xml`

```
<?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-
   2.5.xsd">

  <bean id="customerBatchPersistence"
        class="coreservlets.SpringCustomerBatchPersistence">
    <constructor-arg ref="dataSource" />
  </bean>

</beans>
```

Create Spring AOP/TX Definitions File

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xmlns:tx="http://www.springframework.org/schema/tx"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-2.5.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">

</beans>
```

Register Transaction Manager

```
<?xml version="1.0" encoding="UTF-8"?>
<beans>

  <bean id="transactionManager"
    class="org.springframework....DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource" />
  </bean>

</beans>
```

Register Advice Bean `tx:advice`

```
<beans>
  <bean id="transactionManager"
    class="org.springframework.[...].DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource" />
  </bean>
  <tx:advice id="transactionAdvice"
             transaction-manager="transactionManager">
    <tx:attributes>
    <tx:method name="get*" propagation="REQUIRED" read-only="true"
   />
     <tx:method name="*" propagation="REQUIRED" />
   </tx:attributes>
  </tx:advice>
</beans>
```

Define Pointcut

```
<beans>
  <bean id="transactionManager"
    class="org.springframework.[...].DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource" />
  </bean>
  <tx:advice id="transactionAdvice"
             transaction-manager="transactionManager">
    ...
  </tx:advice>
  <aop:config>
    <aop:pointcut id="customerBatchPersistencePcd"
                  expression="execution(* coreservlets.
                              CustomerBatchPersistence.*(..))" />
   </aop:config>
</beans>
```

Define Aspect

```
<beans>
  <bean id="transactionManager"
    class="org.springframework.[...].DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource" />
  </bean>
  <tx:advice id="transactionAdvice"
             transaction-manager="transactionManager">
    ...
  </tx:advice>
  <aop:config>
    <aop:pointcut id="customerBatchPersistencePcd"
                  expression="execution(* coreservlets.
                             CustomerBatchPersistence.*(..))" />
    <aop:advisor advice-ref="transactionAdvice"
                 pointcut-ref="customerBatchPersistencePcd" />
   </aop:config>
</beans>
```

Initialize Container

```
import org.springframework.context.support.*;
public class Main {
  public static void main(String[]args) {
    BeanFactory beanFactory =
      new ClassPathXmlApplicationContext(new String[]{
        "/coreservletsPersistenceContext.xml",
        "/coreservletsTxContext.xml",
        "/coreservletsDataSourceContext.xml"});
    CustomerBatchPersistence dao =
      (CustomerBatchPersistence)
         beanFactory.getBean("customerBatchPersistence");
    ...
}
```

Test Transaction Manager

```
try{
  dao.insert(
    new Customer("dup-id","dup-name"),
    new Customer("dup-id","dup-name"));
  throw new IllegalStateException("Failed. assertion."
    + " Expected an error inserting duplicate records.");
}
catch(Exception expected){
  boolean rowCountModified = count != dao.getCustomerCount();
  System.out.printf("Row count changed? %s%n",
    rowCountModified);
  if(rowCountModified){
    throw new IllegalStateException("Failed. assertion."
      + " Rollback failed.");
  }
}
```

Standard output

```
Row count changed? false
```

Wrap-up

Summary

- **Implementing advice**
  - AOP alliance APIs
    - e.g. `org.aopalliance.aop.Advice`
  - AspectJ annotation support
    - `@Aspect` with advice type annotations; e.g. `@Around`
  - AspectJ conventions and AspectJ-typed parameters mapped by Spring AOP XML schema support
    - `log(joinPoint:ProceedingJoinPoint):void throws Throwable`
- **Defining pointcuts**
  - Spring AOP XML schema support
    - `<aop:config><aop:pointcut/></aop:config>`
  - AspectJ annotations
    - `@Aspect` and `@Pointcut`

Summary Continued

- **Defining aspects**
  - Spring AOP XML schema support
    - `<aop:config><aop:advisor/></aop:config>`
      - For referencing advice classes implementing AOP alliance APIs
    - `<aop:config><aop:aspect/></aop:config>`
      - For referencing advice class using AspectJ APIs
  - AspectJ APIs
    - `@Aspect` with an advice-classifying annotation such as `@Around`
    - Requires `<aop:aspectj-autoproxy/>`

Questions?