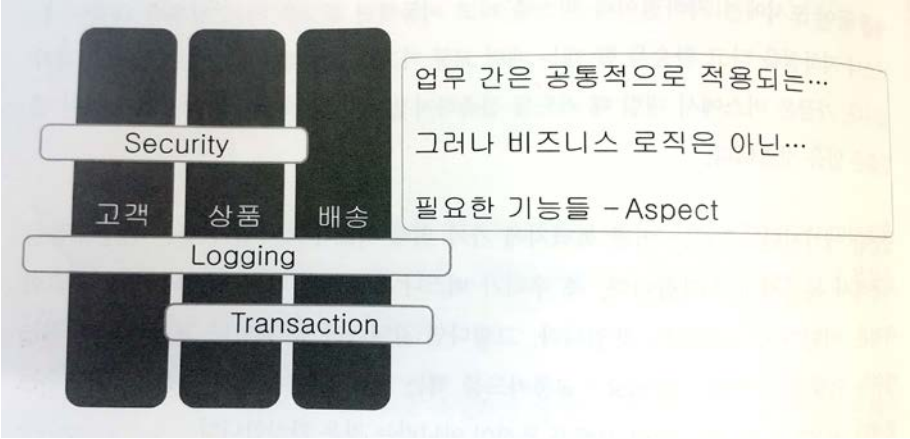
3장 Spring의 AOP와 트랜잭션 관리

3.1 AOP라는 용어와 의미

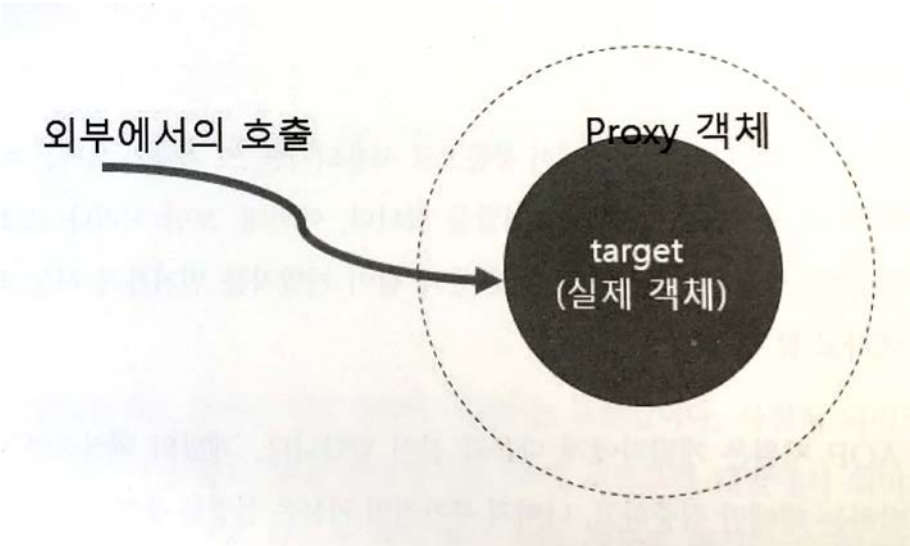
- Aspect Oriented Programming: 기존의 비즈니스 로직 외 작성해야 하는 코드를 별도로 분리함으로써 개발자가 좀 더 비즈니스 로직에만 집중해서 처리할 수 있는 방법을 제공한다.

- 횡단 관심사(cross-concern): 시스템의 여기저기에서 공통으로 사용되지만, 그 자체가 목적은 아니다. 오히려 시스템의 완성도를 높여주는 역할을 한다.



3.1.1 AOP와 관련된 용어

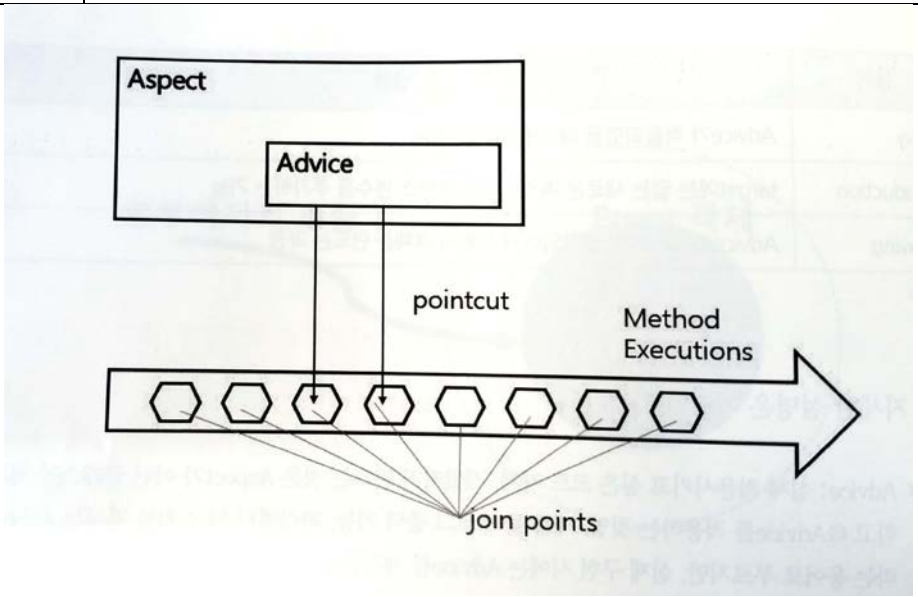
- 프록시(Proxy) 패턴: 외부에서 특정한 객체(target)를 호출하면, 실제 객체를 감싸고 있는 AOP의 기능이 적용된 바깥쪽 객체(Proxy)를 통해서 호출이 전달되어 구현하는 방식이다.



- AOP를 제대로 이해하려면 어쩔 수 없이 다음과 같은 용어들을 이해할 필요가 있다.

용어	설명
Aspect	공통 관심사에 대한 추상적인 명칭. 예를 들어 로깅이나 보안, 트랜잭션과 같은 기능 자체에 대한 용어
Advice	실제로 기능을 구현한 객체; 실제 적용시키고 싶은 코드 자체, 만드는 것은 Aspect가 아닌 클래스를 제작하고 @Advice를 적용하는 것임. 예를 들어 로그 출력 기능, 파라미터 체크 기능 자

	체는 Aspect라는 용어로 부르지만, 실제 구현 시에는 Advice를 제작한다고 표현
Join points	공통 관심사를 적용할 수 있는 대상. Spring AOP에서는 각 객체의 메소드가 이에 해당
Pointcuts	여러 메소드 중 실제 Advice가 적용될 대상 메소드; 여러 Joint points 중에서 Advice를 적용할 대상을 선택하는 정보. 이를 통해서 특정 메소드는 Advice가 적용된 형태로 동작함.
Target	대상 메소드를 가지는 객체; 실제 비즈니스 로직을 수행하는 객체를 의미. 용어 그대로 Aspect를 적용해야 하는 대상 객체를 의미함
Proxy	Advice가 적용되었을 때 만들어지는 객체
Introduction	target에는 없는 새로운 메소드나 인스턴스 변수를 추가하는 기능
Weaving	Advice와 target이 결합되어서 프록시 객체를 만드는 과정



3.1.2 Advice의 종료

■ Advice는 실제 구현된 클래스로 생각할 수 있는데, Advice의 타입은 다시 아래와 같이 분류된다.

타입	기능
Before Advice	target의 메소드 호출 전에 적용
After returning	target의 메소드 호출 이후에 적용
After throwing	target의 예외 발생 후 적용
After	target의 메소드 호출 후 예외의 발생에 관계없이 적용
Around	target의 메소드 호출 이전과 이후 모두 적용(가장 광범위하게 사용됨)

3.2 샘플 프로젝트의 생성과 AOP의 적용 준비

■ 예제 프로젝트의 생성: 예제 프로젝트는 'ex03Lab' 이름이다.

3.2.1 추가해야 하는 라이브러리와 설정

(1) pom.xml 파일의 수정

```
// pom.xml 파일의 수정
<properties>
                    <java-version>1.8</java-version>
                    <org.springframework-version>4.3.3.RELEASE</org.springframework-version>
                    <org.aspectj-version>1.8.9</org.aspectj-version>
```

```
                    <org.slf4j-version>1.6.6</org.slf4j-version>

</properties>
<dependencies>
                    <dependency>
                              <groupId>org.springframework</groupId>
                              <artifactId>spring-tx</artifactId>
                              <version>${org.springframework-version}</version>
                    </dependency>
                    <dependency>
                              <groupId>org.springframework</groupId>
                              <artifactId>spring-aop</artifactId>
                              <version>${org.springframework-version}</version>
                    </dependency>
                    <dependency>
                              <groupId>org.aspectj</groupId>
                              <artifactId>aspectjrt</artifactId>
                              <version>${org.aspectj-version}</version>
                    </dependency>
                    <dependency>
                              <groupId>org.aspectj</groupId>
                              <artifactId>aspectjtools</artifactId>
                              <version>${org.aspectj-version}</version>
                    </dependency>
</dependencies>
```

(2) root-context.xml의 설정

- XML 네임스페이스가 필요하므로 설정 메뉴를 이용해서 'aop'와 'tx' 네임스페이스를 추가한다.
- AOP의 설정을 통한 자동적인 Proxy 객체 생성을 위해서 다음과 같은 설정을 미리 추가한다.

```
// root-context.xml의 설정

<aop:aspectj-autoproxy></aop:aspectj-autoproxy>
```

3.2.2 샘플용 테이블 설계

```
-- AOP 적용을 위한 테이블
create table tbl_user (
          uid varchar(50) NOT NULL,
          upw varchar(50) NOT NULL,
          uname varchar(100) NOT NULL,
          upoint int NOT NULL DEFAULT 0,
          primary key (uid)
);

create table tbl_message (
          mid int not null auto_increment,
          targetid varchar(50) not null,
          sender varchar(50) not null,
          message text not null,
          opendate timestamp,
          senddate timestamp not null default now(),
          primary key (mid)
);
```

```
alter table tbl_message add constraint fk_usertarget
foreign key (targetid) references tbl_user (uid);

alter table tbl_message add constraint fk_usersender
foreign key (targetid) references tbl_user (uid);

insert into tbl_user(uid,upw,uname) values ('user00','user00','IRON MAN');
insert into tbl_user(uid,upw,uname) values ('user01','user01','CAPTAIN');
insert into tbl_user(uid,upw,uname) values ('user02','user02','HULK');
insert into tbl_user(uid,upw,uname) values ('user03','user03','Thor');
insert into tbl_user(uid,upw,uname) values ('user10','user10','Quick Silver');
```

3.2.3 샘플용 도메인 객체; DAO, XML Mapper, 서비스

(1) MessageVO 도메인 객체

[ex03/src/main/java/org/zerock/domain/MessageVO.java]

```
01    package org.zerock.domain;
02
03    import java.util.Date;
04
05    public class MessageVO {
06
07              private Integer mid;
08              private String targetid;
09              private String sender;
10              private String message;
11              private Date opendate;
12              private Date senddate;
13
14              public Integer getMid() {
15                        return mid;
16              }
17              public void setMid(Integer mid) {
18                        this.mid = mid;
19              }
20              public String getTargetid() {
21                        return targetid;
22              }
23              public void setTargetid(String targetid) {
24                        this.targetid = targetid;
25              }
26              public String getSender() {
27                        return sender;
28              }
29              public void setSender(String sender) {
30                        this.sender = sender;
31              }
32              public String getMessage() {
33                        return message;
34              }
35              public void setMessage(String message) {
36                        this.message = message;
37              }
38              public Date getOpendate() {
39                        return opendate;
40              }
41              public void setOpendate(Date opendate) {
42                        this.opendate = opendate;
43              }
```

```
44              public Date getSenddate() {
45                      return senddate;
46              }
47              public void setSenddate(Date senddate) {
48                      this.senddate = senddate;
49              }
50              @Override
51              public String toString() {
52                      return "MessageVO [mid=" + mid + ", targetid=" + targetid + ", sender="
53                                      + sender + ", message=" + message + ", opendate=" + opendate
54                                      + ", senddate=" + senddate + "]";
55              }
56      }
```

(2) MessageDAO 인터페이스의 설계

[ex03/src/main/java/org/zerock/persistence/MessageDAO.java]

```
01      package org.zerock.persistence;
02
03      import org.zerock.domain.MessageVO;
04
05      public interface MessageDAO {
06
07        public void create(MessageVO vo) throws Exception;
08
09        public MessageVO readMessage(Integer mid) throws Exception;
10
11        public void updateState(Integer mid) throws Exception;
12
13      }
```

(3) XML Mapper의 처리 - messageMapper.xml

[resource/mappers/messageMapper.xml]

```
01      <?xml version="1.0" encoding="UTF-8"?>
02      <!DOCTYPE mapper
03      PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
04      "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
05
06      <mapper namespace="org.zerock.mapper.MessageMapper">
07
08              <insert id="create">
09                      insert into tbl_message (targetid, sender,message)
10                      values (#{targetid}, #{sender}, #{message})
11              </insert>
12
13
14              <select id="readMessage" resultType="MessageVO">
15                      select * from tbl_message where mid = #{mid}
16              </select>
17
18              <update id="updateState">
19                      update tbl_message set opendate = now() where mid= #{mid}
20              </update>
21
22      </mapper>
```

(4) MessageDAOImpl의 작성

[ex03/src/main/java/org/zerock/domain/MessageVO.java]

```
01    package org.zerock.persistence;
02
03    import javax.inject.Inject;
04
05    import org.apache.ibatis.session.SqlSession;
06    import org.springframework.stereotype.Repository;
07    import org.zerock.domain.MessageVO;
08
09    @Repository
10    public class MessageDAOImpl implements MessageDAO {
11
12              @Inject
13              private SqlSession session;
14
15              private static String namespace ="org.zerock.mapper.MessageMapper";
16
17              @Override
18              public void create(MessageVO vo) throws Exception {
19
20                        session.insert(namespace+".create", vo);
21              }
22
23              @Override
24              public MessageVO readMessage(Integer mid) throws Exception {
25
26                        return session.selectOne(namespace+".readMessage", mid);
27              }
28
29              @Override
30              public void updateState(Integer mid) throws Exception {
31
32                        session.update(namespace+".upldateState", mid);
33
34              }
35
36    }
```

(5) PointDAO 인터페이스

[ex03/src/main/java/org/zerock/persistence/PointDAO.java]

```
01    package org.zerock.persistence;
02
03    public interface PointDAO {
04
05              public void updatePoint(String uid,int point)throws Exception;
06
07    }
```

(6) XML Mapper의 처리 - pointMapper.xml

[resource/mappers/pointMapper.xml]

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper
PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="org.zerock.mapper.PointMapper">

          <update id="updatePoint">
                    update tbl_user set upoint = upoint + #{point} where
                    uuuuid = #{uid}
<!--                uid = #{uid} -->
          </update>

</mapper>
```

(7) PointDAOImpl 구현

[ex03/src/main/java/org/zerock/service/PointDAOImpl.java]

```
package org.zerock.persistence;

import java.util.HashMap;
import java.util.Map;

import javax.inject.Inject;

import org.apache.ibatis.session.SqlSession;
import org.springframework.stereotype.Repository;

@Repository
public class PointDAOImpl implements PointDAO {

  @Inject
  private SqlSession session;

  private static String namespace = "org.zerock.mapper.PointMapper";

  @Override
  public void updatePoint(String uid, int point) throws Exception {

    Map<String, Object> paramMap = new HashMap<String, Object>();
    paramMap.put("uid", uid);
    paramMap.put("point", point);

    session.update(namespace + ".updatePoint", paramMap);

  }

}
```

3.2.4 서비스 객체 설정

[ex03/src/main/java/org/zerock/service/MessageService.java]

```
package org.zerock.service;

import org.zerock.domain.MessageVO;

```

```
05    public interface MessageService {
06
07      public void addMessage(MessageVO vo) throws Exception;
08
09      public MessageVO readMessage(String uid, Integer mno) throws Exception;
10    }
```

[ex03/src/main/java/org/zerock/service/MessageServiceImpl.java]

```
01    package org.zerock.service;
02
03    import javax.inject.Inject;
04
05    import org.springframework.stereotype.Service;
06    import org.springframework.transaction.annotation.Transactional;
07    import org.zerock.domain.MessageVO;
08    import org.zerock.persistence.MessageDAO;
09    import org.zerock.persistence.PointDAO;
10
11    @Service
12    public class MessageServiceImpl implements MessageService {
13
14      @Inject
15      private MessageDAO messageDAO;
16
17      @Inject
18      private PointDAO pointDAO;
19
20
21      //@Transactional
22      @Override
23      public void addMessage(MessageVO vo) throws Exception {
24
25        messageDAO.create(vo);
26        pointDAO.updatePoint(vo.getSender(), 10);
27      }
28
29      @Override
30      public MessageVO readMessage(String uid, Integer mid) throws Exception {
31
32        messageDAO.updateState(mid);
33
34        pointDAO.updatePoint(uid, 5);
35
36        return messageDAO.readMessage(mid);
37      }
38    }
```

3.3 AOP 연습하기

3.3.1 Advice 생성하기

```
// root-context.xml의 설정
          <context:component-scan base-package="org.zerock.persistence"></context:component-scan>
          <context:component-scan base-package="org.zerock.service"></context:component-scan>
          <context:component-scan base-package="org.zerock.aop"></context:component-scan>

          <aop:config>
          </aop:config>
```

(1) SampleAdvice의 작성

[ex03/src/main/java/org/zerock/aop/SampleAdvice.java]

```
01    package org.zerock.aop;
02
03    import java.util.Arrays;
04
05    import org.aspectj.lang.JoinPoint;
06    import org.aspectj.lang.ProceedingJoinPoint;
07    import org.aspectj.lang.annotation.Around;
08    import org.aspectj.lang.annotation.Aspect;
09    import org.aspectj.lang.annotation.Before;
10    import org.slf4j.Logger;
11    import org.slf4j.LoggerFactory;
12    import org.springframework.stereotype.Component;
13
14    @Component
15    @Aspect
16    public class SampleAdvice {
17
18      private static final Logger logger = LoggerFactory.getLogger(SampleAdvice.class);
19
20      //@Before("execution(* org.zerock.service.MessageService*.*(..))")
21      public void startLog(JoinPoint jp) {
22
23        logger.info("-----------------------------");
24        logger.info("-----------------------------");
25        logger.info(Arrays.toString(jp.getArgs()));
26
27      }
28
29
30     // @Around("execution(* org.zerock.service.MessageService*.*(..))")
31      public Object timeLog(ProceedingJoinPoint pjp)throws Throwable{
32
33        long startTime = System.currentTimeMillis();
34        logger.info(Arrays.toString(pjp.getArgs()));
35
36        Object result = pjp.proceed();
37
38        long endTime = System.currentTimeMillis();
39        logger.info( pjp.getSignature().getName()+ " : " + (endTime - startTime) );
40        logger.info("==========================================");
41
42        return result;
43      }
44
45    }
```

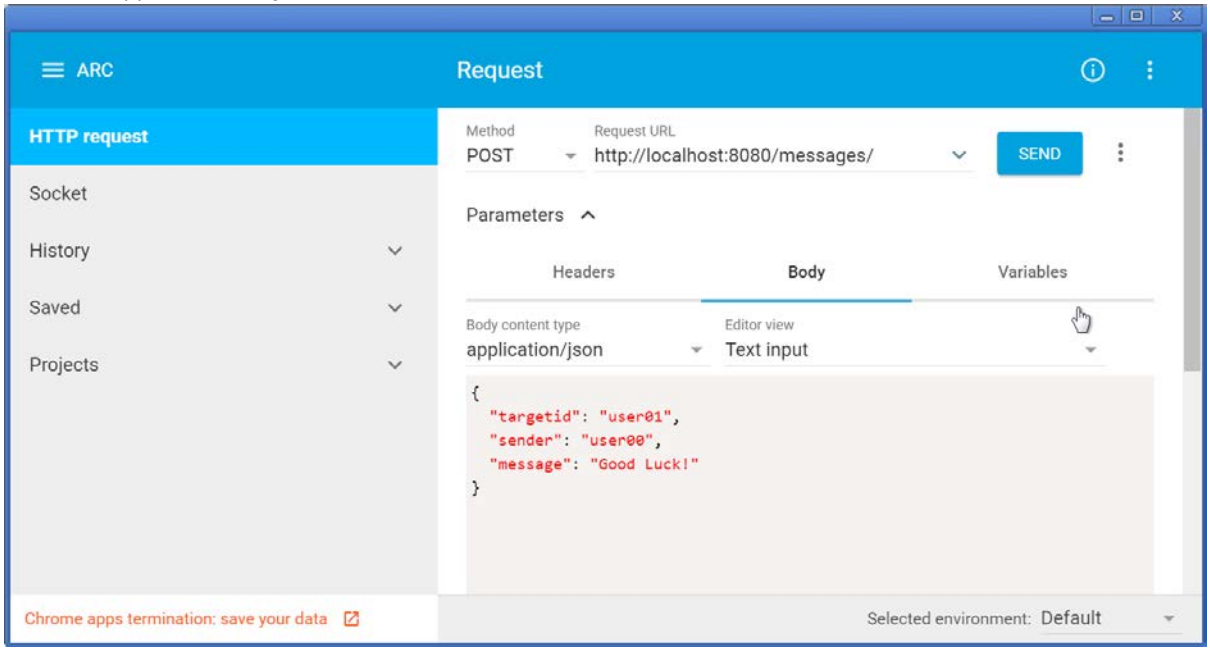
3.3.2 컨트롤러의 작성과 테스트하기

[ex03/src/main/java/org/zerock/controller/MessageController.java]

```
01    package org.zerock.controller;
02
03    import javax.inject.Inject;
04
```

```
05  import org.springframework.http.HttpStatus;
06  import org.springframework.http.ResponseEntity;
07  import org.springframework.web.bind.annotation.RequestBody;
08  import org.springframework.web.bind.annotation.RequestMapping;
09  import org.springframework.web.bind.annotation.RequestMethod;
10  import org.springframework.web.bind.annotation.RestController;
11  import org.zerock.domain.MessageVO;
12  import org.zerock.service.MessageService;
13
14  @RestController
15  @RequestMapping("/messages")
16  public class MessageController {
17
18    @Inject
19    private MessageService service;
20
21    @RequestMapping(value = "/", method = RequestMethod.POST)
22    public ResponseEntity<String> addMessage(@RequestBody MessageVO vo) {
23
24      ResponseEntity<String> entity = null;
25      try {
26        service.addMessage(vo);
27        entity = new ResponseEntity<>("SUCCESS", HttpStatus.OK);
28      } catch (Exception e) {
29        e.printStackTrace();
30        entity = new ResponseEntity<>(e.getMessage(), HttpStatus.BAD_REQUEST);
31      }
32      return entity;
33    }
34
35  }
```

- 최종 테스트는 Advanced Rest Client를 이용해서 진행한다. 아래 화면의 Content-type을 application/json으로 설정하는 부분을 주의해야 한다.




```
// 스프링이 실행된 서버 내에는 아래와 같은 로그가 기록됐는지 확인한다.
INFO : org.zerock.aop.SampleAdvice - ------------------------------
INFO : org.zerock.aop.SampleAdvice - ------------------------------
INFO : org.zerock.aop.SampleAdvice - [MessageVO [mid=null, targetid=user01, sender=user00, message=Good
```

```
Luck!, opendate=null, senddate=null]]
INFO : jdbc.connection - 3. Connection opened
INFO : jdbc.audit - 3. Connection.new Connection returned
INFO : jdbc.audit - 3. Connection.getAutoCommit() returned true
INFO : jdbc.audit - 3. PreparedStatement.new PreparedStatement returned
...이하 생략
```

[꿀팁] 로그 출력 문제

■ 아래와 같이 log4j.xml 파일에 내용을 확인한다.

[ex03/src/main/resources/log4j.xml]

```
01  <?xml version="1.0" encoding="UTF-8"?>
02  <!DOCTYPE log4j:configuration PUBLIC "-//APACHE//DTD LOG4J 1.2//EN" "log4j.dtd">
03  <log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">
04
05          <!-- Appenders -->
06          <appender name="console" class="org.apache.log4j.ConsoleAppender">
07                  <param name="Target" value="System.out" />
08                  <layout class="org.apache.log4j.PatternLayout">
09                          <param name="ConversionPattern" value="%-5p: %c - %m%n" />
10                  </layout>
11          </appender>
12
13          <!-- Application Loggers -->
14          <logger name="org.zerock.controller">
15                  <level value="info" />
16          </logger>
17
18          <!-- 3rdparty Loggers -->
19          <logger name="org.springframework.core">
20                  <level value="info" />
21          </logger>
22
23          <logger name="org.springframework.beans">
24                  <level value="info" />
25          </logger>
26
27          <logger name="org.springframework.context">
28                  <level value="info" />
29          </logger>
30
31          <logger name="org.springframework.web">
32                  <level value="info" />
33          </logger>
34
35          <!-- Root Logger -->
36          <root>
37                  <priority value="info" />
38                  <appender-ref ref="console" />
39          </root>
40
41  </log4j:configuration>
```

3.3.3 실행 시에 전달되는 파라미터 파악하기

■ org.aspect.lang.JoinPoint는 다음과 같은 기능들을 가지고 있다.

주요 메소드	설명
Object[] getArgs()	전달되는 모든 파라미터들을 Object의 배열로 가져온다.
String getKind()	해당 Advice의 타입을 알아낸다.
Signature getSignature()	실행하는 대상 객체의 메소드에 대한 정보를 알아낼 때 사용
Object getTarget()	target 객체를 알아낼 때 사용
Object getThis()	Advice를 행하는 객체를 알아낼 때 사용

```
//org.zerock.aop.SampleAdvice의 일부
@Before("execution(* org.zerock.service.MessageService*.*(..))")
public void startLog(JoinPoint jp) {
    logger.info("-----------------------------");
    logger.info("-----------------------------");
    logger.info(Arrays.toString(jp.getArgs()));
}
```

3.3.4 가장 강력한 Around

- Around는 말 그대로 메소드의 실행 전체를 앞, 뒤로 감싸서 특정한 기능을 실행할 수 있는 가장 강력한 타입의 Advice이다. Around 타입의 기능은 파라미터로 ProceedingJoinPoint 타입을 사용할 수 있는데, 이 인터페이스는 다음과 같은 기능이 있다.

주요 메소드	설명
Ojbect proceed()	다음 Advice를 실행하거나, 실제 target 객체의 메소드를 실행하는 기능

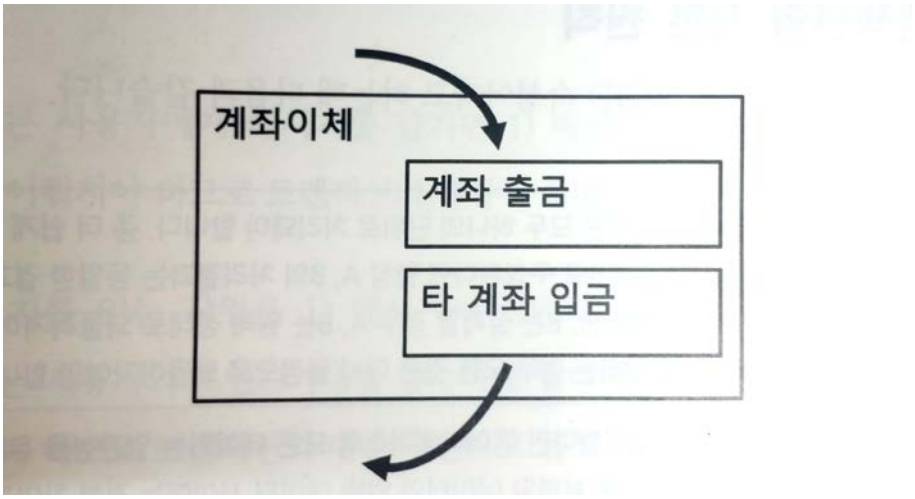
```
//timeLog() 적용 결과 시작
INFO : org.zerock.aop.SampleAdvice - ==========================================
INFO : org.zerock.aop.SampleAdvice - [MessageVO [mid=null, targetid=user01, sender=user00, message=Good
Luck!, opendate=null, senddate=null]]
INFO : jdbc.connection - 1. Connection opened
INFO : jdbc.audit - 1. Connection.new Connection returned
INFO : jdbc.audit - 1. Connection.getAutoCommit() returned true
INFO : jdbc.audit - 1. PreparedStatement.new PreparedStatement returned
INFO : jdbc.audit - 1. Connection.prepareStatement(insert into tbl_message (targetid, sender,message)
                    values (?, ?, ?)) returned net.sf.log4jdbc.sql.jdbcapi.PreparedStatementSpy@d8ecb2
INFO : jdbc.audit - 1. PreparedStatement.setString(1, "user01") returned
INFO : jdbc.audit - 1. PreparedStatement.setString(2, "user00") returned
INFO : jdbc.audit - 1. PreparedStatement.setString(3, "Good Luck!") returned
INFO : jdbc.sqlonly - insert into tbl_message (targetid, sender,message) values ('user01', 'user00', 'Good
Luck!')

INFO : jdbc.sqltiming - insert into tbl_message (targetid, sender,message) values ('user01', 'user00', 'Good
Luck!')
 {executed in 7 msec}
INFO : jdbc.audit - 1. PreparedStatement.execute() returned false
INFO : jdbc.audit - 1. PreparedStatement.getUpdateCount() returned 1
INFO : jdbc.audit - 1. PreparedStatement.close() returned
INFO : jdbc.connection - 1. Connection closed
INFO : jdbc.audit - 1. Connection.close() returned
INFO : jdbc.connection - 2. Connection opened
INFO : jdbc.audit - 2. Connection.new Connection returned
INFO : jdbc.audit - 2. Connection.getAutoCommit() returned true
INFO : jdbc.audit - 2. PreparedStatement.new PreparedStatement returned
INFO : jdbc.audit - 2. Connection.prepareStatement(update tbl_user set upoint = upoint + ? where
                    uid = ?) returned net.sf.log4jdbc.sql.jdbcapi.PreparedStatementSpy@a5c291
INFO : jdbc.audit - 2. PreparedStatement.setInt(1, 10) returned
INFO : jdbc.audit - 2. PreparedStatement.setString(2, "user00") returned
INFO : jdbc.sqlonly - update tbl_user set upoint = upoint + 10 where uid = 'user00'
```

```
INFO : jdbc.sqltiming - update tbl_user set upoint = upoint + 10 where uid = 'user00'
 {executed in 8 msec}
INFO : jdbc.audit - 2. PreparedStatement.execute() returned false
INFO : jdbc.audit - 2. PreparedStatement.getUpdateCount() returned 1
INFO : jdbc.audit - 2. PreparedStatement.close() returned
INFO : jdbc.connection - 2. Connection closed
INFO : jdbc.audit - 2. Connection.close() returned
INFO : org.zerock.aop.SampleAdvice - addMessage : 448
INFO : org.zerock.aop.SampleAdvice - ========================================
//timeLog() 적용 결과 끝
```

3.4 트랜잭션 처리

3.4.1 트랜잭션에 대한 기본 설명

■ 트랜잭션은 쉽게 말해서 하나의 업무에 여러 개의 작은 업무들이 같이 묶여 있는 것을 의미한다.



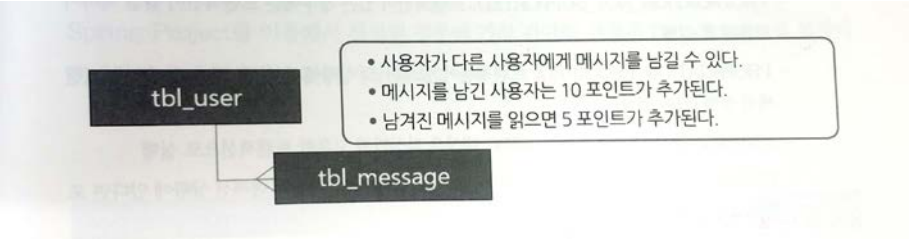
(1) 트랜잭션의 기본 원칙

■ 트랜잭션의 기본 원칙은 흔히 ACID 속성이라고 하는데 다음과 같다.

원자성(Atomicity)	하나의 트랜잭션은 모두 하나의 단위로 처리돼야 한다. 좀더 쉽게 말하자면 어떤 작업이 잘못되는 경우 모든 것은 다시 원점으로 되돌아가야만 한다.
일관성(Consistency)	트랜잭션이 성공했다면 데이터베이스의 모든 데이터는 일관성을 유지해야만 한다. 트랜잭션으로 처리된 데이터와 일반 데이터 사이에는 전혀 차이가 없어야만 한다.
격리(Isolation)	트랜잭션으로 처리되는 중간에 외부에서의 간섭은 없어야만 한다.
영속성(Durability)	트랜잭션이 성공적으로 처리되면, 그 결과는 영속적으로 보관돼야 한다.

(2) 트랜잭션을 처리하는 상황 이해하기

■ 사용자가 다른 사용자에게 메시지를 남기면 1)메시지의 등록과 2)10포인트의 증가 작업이 같이 이뤄져야 하므로 트랜잭션의 대상이 된다. 남겨진 메시지를 읽는 작업은 1)메시지의 상태가 읽은 상태로 변경되는 작업과 2)메시지를 읽는 사용자의 포인트가 5점 증가하도록 해야 하므로 트랜잭션의 대상이 된다.



3.4.2 @Transactional 어노테이션

- 스프링에서는 트랜잭션을 처리하기 위해서 제공되는 @Transactional 어노테이션을 이용하면 간단히 트랜잭션 설정을 완료할 수 있다.

3.4.3 트랜잭션 매니저의 설정

(1) root-context.xml의 처리

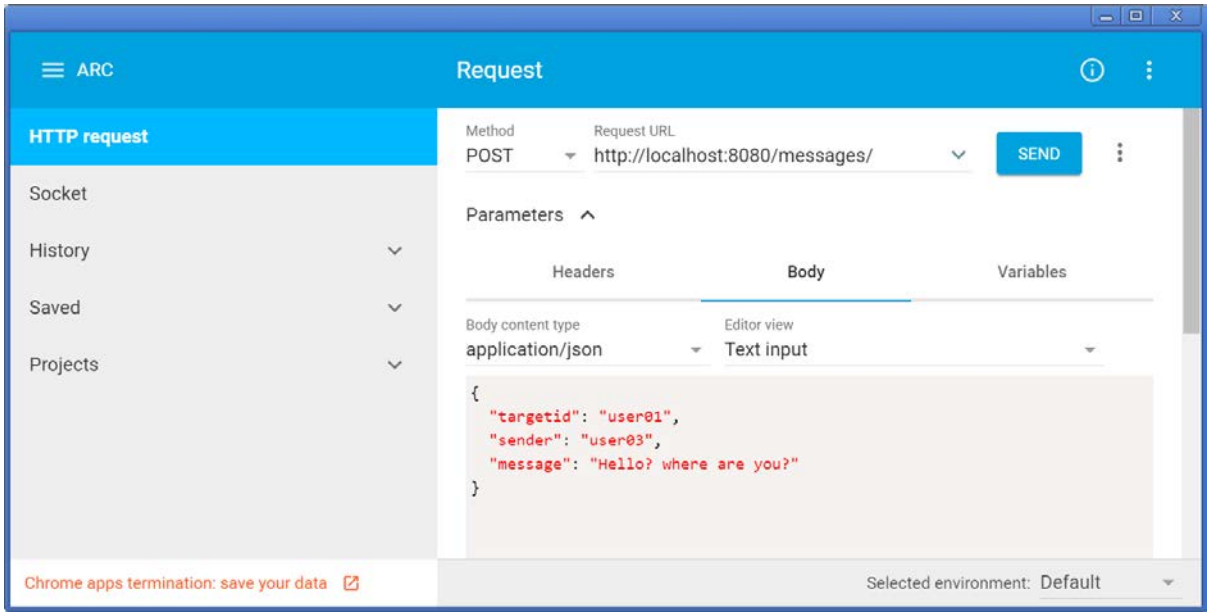
- Spring Project를 이용해서 생성된 경우에 가장 간단한 설정은 DataSource의 설정이 존재하는 곳에 적용하는 것이 가장 편리하다.

- <tx:annotation-driven>은 @Transactional 어노테이션을 이용한 트랜잭션의 관리가 가능하다.

```
// root-context.xml의 일부
        <bean id="transactionManager"
                class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
                <property name="dataSource" ref="dataSource"></property>
        </bean>

        <tx:annotation-driven />
```

3.4.4 트랜잭션 적용 테스트

(1) 정상적인 경우의 테스트




```
// 테스트 후에는 tbl_message에 새롭게 메시지가 추가된다.
SELECT * FROM book_ex.tbl_message;
1         user01    user00    Good Luck!          2017-10-27 16:23:09 2017-10-27 16:23:09
2         user01    user00    Good Luck!          2017-10-27 16:23:52 2017-10-27 16:23:52
3         user01    user00    Good Luck!          2017-10-27 17:35:33 2017-10-27 17:35:33
4         user01    user00    Good Luck!          2017-10-27 17:37:57 2017-10-27 17:37:57
5         user01    user03    Hello? where are you?         2017-10-28 17:56:31 2017-10-28 17:56:31

// tbl_user 테이블에는 메시를 보낸 'user03'의 포인트가 변경된 것을 확인할 수 있다.
SELECT * FROM book_ex.tbl_user;
user00    user00    IRON MAN  40
user01    user01    CAPTAIN   0
user02    user02    HULK      0
user03    user03    Thor      10
user10    user10    Quick Silver        0
```

(2) 트랜잭션이 처리되지 않은 비정상적인 경우의 테스트

■ 트랜잭션이 처리되지 않는 상황에서 비정상적인 경우는 다음과 같다.

- MessageDAO에는 정상적으로 insert SQL 문이 동작하도록 설정한다.
- PointDAO에는 update하는 SQL 문에 오류가 있도록 설정한다.

[ex03/src/main/resources/mappers/pointMapper.xml]

```
01   <?xml version="1.0" encoding="UTF-8"?>
02   <!DOCTYPE mapper
03   PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
04   "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
05
06   <mapper namespace="org.zerock.mapper.PointMapper">
07
08            <update id="updatePoint">
09                     update tbl_user set upoint = upoint + #{point} where
10                     uuuuid = #{uid}
11   <!--              uid = #{uid} -->
12            </update>
13
```

```
14    </mapper>
```

(3) MessageService의 트랜잭션 처리 후 테스트

- 가장 먼저 root-context.xml에 '<tx:annotation-driven/>' 설정이 추가되었는지를 확인한다.
- MessageServiceImpl을 수정해서 @Transactional 어노테이션을 추가한다.

```
// ex03/src/main/java/org/zerock/service/MessageServiceImpl.java
...(생략)
@Transactional
@Override
public void addMessage(MessageVO vo) throws Exception {

    messageDAO.create(vo);
    pointDAO.updatePoint(vo.getSender(), 10);
}
...(생략)
```

(4) 트랜잭션에 따른 로그 변화

```
INFO : jdbc.connection - 2. Connection opened
INFO : jdbc.audit - 2. Connection.new Connection returned
INFO : jdbc.audit - 2. Connection.getAutoCommit() returned true
INFO : jdbc.audit - 2. Connection.setAutoCommit(false) returned
INFO : org.zerock.aop.SampleAdvice - ======================================
INFO : org.zerock.aop.SampleAdvice - [MessageVO [mid=null, targetid=user01, sender=user03, message=안녕하세요
2?, opendate=null, senddate=null]]
INFO : jdbc.audit - 2. Connection.getAutoCommit() returned false
INFO : jdbc.audit - 2. PreparedStatement.new PreparedStatement returned
INFO : jdbc.audit - 2. Connection.prepareStatement(insert into tbl_message (targetid, sender,message)
                   values (?, ?, ?)) returned net.sf.log4jdbc.sql.jdbcapi.PreparedStatementSpy@177a6d8
INFO : jdbc.audit - 2. PreparedStatement.setString(1, "user01") returned
INFO : jdbc.audit - 2. PreparedStatement.setString(2, "user03") returned
INFO : jdbc.audit - 2. PreparedStatement.setString(3, "안녕하세요2?") returned
INFO : jdbc.sqlonly - insert into tbl_message (targetid, sender,message) values ('user01', 'user03', '안녕하
세요2?')

INFO : jdbc.sqltiming - insert into tbl_message (targetid, sender,message) values ('user01', 'user03', '안녕
하세요2?')
 {executed in 24 msec}
INFO : jdbc.audit - 2. PreparedStatement.execute() returned false
INFO : jdbc.audit - 2. PreparedStatement.getUpdateCount() returned 1
INFO : jdbc.audit - 2. PreparedStatement.close() returned
INFO : jdbc.audit - 2. PreparedStatement.new PreparedStatement returned
INFO : jdbc.audit - 2. Connection.prepareStatement(update tbl_user set upoint = upoint + ? where
                   uuuuid = ?) returned net.sf.log4jdbc.sql.jdbcapi.PreparedStatementSpy@ee030f
INFO : jdbc.audit - 2. PreparedStatement.setInt(1, 10) returned
INFO : jdbc.audit - 2. PreparedStatement.setString(2, "user03") returned
INFO : jdbc.sqlonly - update tbl_user set upoint = upoint + 10 where uuuuid = 'user03'

ERROR: jdbc.audit - 2. PreparedStatement.execute() update tbl_user set upoint = upoint + 10 where uuuuid =
'user03'

com.mysql.jdbc.exceptions.jdbc4.MySQLSyntaxErrorException: Unknown column 'uuuuid' in 'where clause'
          at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
```

```
        at sun.reflect.NativeConstructorAccessorImpl.newInstance(Unknown Source)
        at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(Unknown Source)
...(생략)
```

3.4.5 @Transactional의 적용 순서

■ 스프링은 간단한 트랜잭션 메니저의 설정과 @Transactional 어노테이션을 이용한 설정만으로 애플리케이션 내의 트랜잭션에 대한 설정을 처리할 수 있다.

■ 어노테이션의 우선순위는 다음과 같다.
- 메소드의 @Transactional 설정이 가장 우선시 된다.
- 클래스의 @Transactional 설정은 메소드보다 우선순위가 낮다.
- 인터페이스의 @Transactional 설정이 우선순위가 가장 낮다.

3.5 게시물의 댓글에 따른 트랜잭션 처리

3.5.1 댓글 카운트의 처리

■ 우선 tbl_board에는 댓글의 숫자를 의미하는 replycnt 칼럼을 추가한다.

```
-- 댓글 카운트의 처리
alter table tbl_board add column replycnt int default 0;
```

(1) BoardVO의 변경

[ex03/src/main/java/org/zerock/persistence/BoardDAO.java]

```
01   package org.zerock.domain;
02
03   import java.util.Date;
04
05   public class BoardVO {
06
07     private Integer bno;
08     private String title;
09     private String content;
10     private String writer;
11     private Date regdate;
12     private int viewcnt;
13     private int replycnt;
14
15     public int getReplycnt() {
16       return replycnt;
17     }
18
19     public void setReplycnt(int replycnt) {
20       this.replycnt = replycnt;
21     }
22
23     public Integer getBno() {
24       return bno;
25     }
```

```
26
27      public void setBno(Integer bno) {
28        this.bno = bno;
29      }
30
31      public String getTitle() {
32        return title;
33      }
34
35      public void setTitle(String title) {
36        this.title = title;
37      }
38
39      public String getContent() {
40        return content;
41      }
42
43      public void setContent(String content) {
44        this.content = content;
45      }
46
47      public String getWriter() {
48        return writer;
49      }
50
51      public void setWriter(String writer) {
52        this.writer = writer;
53      }
54
55      public Date getRegdate() {
56        return regdate;
57      }
58
59      public void setRegdate(Date regdate) {
60        this.regdate = regdate;
61      }
62
63      public int getViewcnt() {
64        return viewcnt;
65      }
66
67      public void setViewcnt(int viewcnt) {
68        this.viewcnt = viewcnt;
69      }
70
71      @Override
72      public String toString() {
73        return "BoardVO [bno=" + bno + ", title=" + title + ", content=" + content + ", writer=" + writer +
74    ", regdate="
75            + regdate + ", viewcnt=" + viewcnt + ", replycnt=" + replycnt + "]";
76      }
77
78    }
```

(2) boardMapper.xml의 SQL 문 변경

[src/main/resources/boardMapper.xml]

```
01    <?xml version="1.0" encoding="UTF-8"?>
02    <!DOCTYPE mapper
03    PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
04    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
```

```
05
06  <mapper namespace="org.zerock.mapper.BoardMapper">
07
08          <insert id="create">
09                  insert into tbl_board (title, content, writer)
10                  values(#{title},#{content}, #{writer})
11          </insert>
12
13          <select id="read" resultType="org.zerock.domain.BoardVO">
14                  select
15                  bno, title, content, writer, regdate, viewcnt , replycnt
16                  from
17                  tbl_board
18                  where bno = #{bno}
19          </select>
20
21          <select id="listAll" resultType="org.zerock.domain.BoardVO">
22  <![CDATA[
23  select
24    bno, title, content, writer, regdate, viewcnt
25  from
26    tbl_board
27  where bno > 0
28  order by bno desc, regdate desc
29  ]]>
30          </select>
31
32          <select id="listPage" resultType="BoardVO">
33  <![CDATA[
34  select
35    bno, title, content, writer, regdate, viewcnt , replycnt
36  from
37    tbl_board
38  where bno > 0
39  order by bno desc, regdate desc
40  limit #{page}, 10
41  ]]>
42          </select>
43
44          <select id="listCriteria" resultType="BoardVO">
45  <![CDATA[
46  select
47    bno, title, content, writer, regdate, viewcnt , replycnt
48  from
49    tbl_board
50  where bno > 0
51  order by bno desc, regdate desc
52  limit #{pageStart}, #{perPageNum}
53  ]]>
54          </select>
55
56
57          <update id="update">
58                  update tbl_board set title =#{title}, content =#{content}
59                  where bno = #{bno}
60          </update>
61
62          <delete id="delete">
63                  delete from tbl_board where bno = #{bno}
64          </delete>
65
66
67
68          <select id="countPaging" resultType="int">
69  <![CDATA[
```

```
70    select
71      count(bno)
72    from
73      tbl_board
74    where
75      bno > 0
76    ]]>
77            </select>
78
79
80
81            <sql id="search">
82                    <if test="searchType != null">
83                            <if test="searchType == 't'.toString()">
84                                    and title like CONCAT('%', #{keyword}, '%')
85                            </if>
86                            <if test="searchType == 'c'.toString()">
87                                    and content like CONCAT('%', #{keyword}, '%')
88                            </if>
89                            <if test="searchType == 'w'.toString()">
90                                    and writer like CONCAT('%', #{keyword}, '%')
91                            </if>
92                            <if test="searchType == 'tc'.toString()">
93                                    and ( title like CONCAT('%', #{keyword}, '%') OR content like
94                                    CONCAT('%', #{keyword}, '%'))
95                            </if>
96                            <if test="searchType == 'cw'.toString()">
97                                    and ( content like CONCAT('%', #{keyword}, '%') OR writer like
98                                    CONCAT('%', #{keyword}, '%'))
99                            </if>
100                           <if test="searchType == 'tcw'.toString()">
101                                   and ( title like CONCAT('%', #{keyword}, '%')
102                                   OR
103                                   content like CONCAT('%', #{keyword}, '%')
104                                   OR
105                                   writer like CONCAT('%', #{keyword}, '%'))
106                           </if>
107                   </if>
108           </sql>
109
110           <select id="listSearch" resultType="BoardVO">
111   <![CDATA[
112     select *
113     from tbl_board
114     where bno > 0
115   ]]>
116
117                   <include refid="search"></include>
118
119   <![CDATA[
120     order by bno desc
121     limit #{pageStart}, #{perPageNum}
122   ]]>
123           </select>
124
125           <select id="listSearchCount" resultType="int">
126   <![CDATA[
127     select count(bno)
128     from tbl_board
129     where bno > 0
130   ]]>
131                   <include refid="search"></include>
132
133           </select>
134
```

```
135
136
137   <update id="updateReplyCnt">
138     update tbl_board set replycnt = replycnt + #{amount} where bno = #{bno}
139   </update>
140
141
142   <update id="updateViewCnt">
143     update tbl_board set viewcnt = viewcnt +1 where bno = #{bno}
144   </update>
145
146   </mapper>
```

(3) BoardDAO, ReplyDAO 변경

```
// org.zerock.persistence.BoardDAO.java의 일부
public void updateReplyCnt(Integer bno, int amount)throws Exception;


// resource/mappers/boardMapper.xml의 일부
<update id="updateReplyCnt">
  update tbl_board set replycnt = replycnt + #{amount} where bno = #{bno}
</update>


// org.zerock.persistence.BoardDAOImpl.java의 일부
@Override
public void updateReplyCnt(Integer bno, int amount) throws Exception {

    Map<String, Object> paramMap = new HashMap<String, Object>();

    paramMap.put("bno", bno);
    paramMap.put("amount", amount);

    session.update(namespace + ".updateReplyCnt", paramMap);
}
```

(4) ReplyServiceImpl의 수정

- 새로운 댓글이 추가되면 tbl_board의 replycnt 칼럼의 값을 1 증가시키는 작업과 댓글이 삭제될 때 replycnt 칼럼의 값을 -1 시키는 작업으로 수정하도록 @Transactional을 이용해서 처리한다.

```
  @Inject
  private ReplyDAO replyDAO;

  @Inject
  private BoardDAO boardDAO;

  @Transactional
  @Override
  public void addReply(ReplyVO vo) throws Exception {

    replyDAO.create(vo);
    boardDAO.updateReplyCnt(vo.getBno(), 1);
  }

  @Transactional
  @Override
```

```
  public void removeReply(Integer rno) throws Exception {

    int bno = replyDAO.getBno(rno);
    replyDAO.delete(rno);
    boardDAO.updateReplyCnt(bno, -1);
  }
```

(5) 조회 화면의 댓글 수 출력

■ 게시물 목록에서의 댓글 수 처리

```
// /WEB-INF/views/sboard/list.jsp의 일부
<c:forEach items="${list}" var="boardVO">

          <tr>
                    <td>${boardVO.bno}</td>
                    <td><a

          href='/sboard/readPage${pageMaker.makeSearch(pageMaker.cri.page) }&bno=${boardVO.bno}'>
                                        ${boardVO.title} <strong>[ ${boardVO.replycnt} ]</strong>
                    </a></td>
                    <td>${boardVO.writer}</td>
                    <td><fmt:formatDate pattern="yyyy-MM-dd HH:mm"
                                        value="${boardVO.regdate}" /></td>
                    <td><span class="badge bg-red">${boardVO.viewcnt }</span></td>
          </tr>

</c:forEach>
```

(6) 조회 화면에서 댓글의 숫자 출력

■ 조회 화면을 처리하는 readPage.jsp를 수정해서 댓글의 수를 볼 수 있도록 처리한다.

```
// /WEB-INF/views/sboard/readPage.jsp의 일부
          <!-- The time line -->
          <ul class="timeline">
                    <!-- timeline time label -->
                    <li class="time-label" id="repliesDiv"><span class="bg-green">
                                        Replies List <small id='replycntSmall'> [
                                                  ${boardVO.replycnt} ] </small>
                    </span></li>
          </ul>
```

■ 댓글의 숫자를 보여주는 작업은 tbl_reply를 통해서 이뤄지지만, 댓글의 삭제는 Ajax를 통해서 tbl_reply를 이용한다. 따라서 댓글의 페이징 처리 시 댓글의 숫자를 이용해서 위의 숫자를 갱신해 줘야 한다.

```
// /WEB-INF/views/sboard/readPage.jsp의 일부
function getPage(pageInfo){

                    $.getJSON(pageInfo,function(data){
                              printData(data.list, $("#repliesDiv") ,$('#template'));
                              printPaging(data.pageMaker, $(".pagination"));

                              $("#modifyModal").modal('hide');
                              $("#replycntSmall").html("[ " + data.pageMaker.totalCount +" ]");
```

```
                    });
}
```

3.5.2 조회 숫자의 처리

(1) BoardDAO, XML Mapper, BoardDAOImpl의 처리

```
// org.zerock.persistence.BoardDAO.java의 일부
public void updateViewCnt(Integer bno)throws Exception;


// resource/mappers/boardMapper.xml의 일부
<update id="updateViewCnt">
  update tbl_board set viewcnt = viewcnt +1 where bno = #{bno}
</update>


// org.zerock.persistence.BoardDAOImpl.java의 일부
@Override
public void updateViewCnt(Integer bno) throws Exception {

    session.update(namespace+".updateViewCnt", bno);

}
```

(2) BoardService의 처리

```
// org.zerock.service.BoardServiceImpl.java의 일부

  @Transactional(isolation=Isolation.READ_COMMITTED)
  @Override
  public BoardVO read(Integer bno) throws Exception {
    dao.updateViewCnt(bno);
    return dao.read(bno);
  }
```