Spring과 AOP(Aspect Oriented Programming)

■ AOP란?

(1) Aspect oriented Programming의 약자로 관점지향 프로그래밍

(2) 문제를 바라보는 관점을 기준으로 프로그래밍 하는 방법론

(3) 횡단관심사(cross-cutting-concerns)

ㄱ. 한 애플리케이션의 여러 부분에 걸쳐 있는 기능

ㄴ. 비즈니스 로직과는 개념적으로 분리됨

(4) 시스템을 구성하는 다양한 컴포넌트는 자신 본래 기능을 넘어선 추가 역할을 수행

ㄱ. 로깅, 트랜잭션, 보안, 캐싱 등의 기능

(5) 핵심 로직을 구현한 코드와 공통 기능 관련 로직의 분리가 목적

ㄱ. 시스템 전반적인 관심사를 구현한 코드가 여러 컴포넌트에 중복

ㄴ. 변경 이슈가 발생할 경우 개별적인 코드를 수정

ㄷ. 컴포넌트가 핵심기능이 기능적인 코드에 의해 오염됨

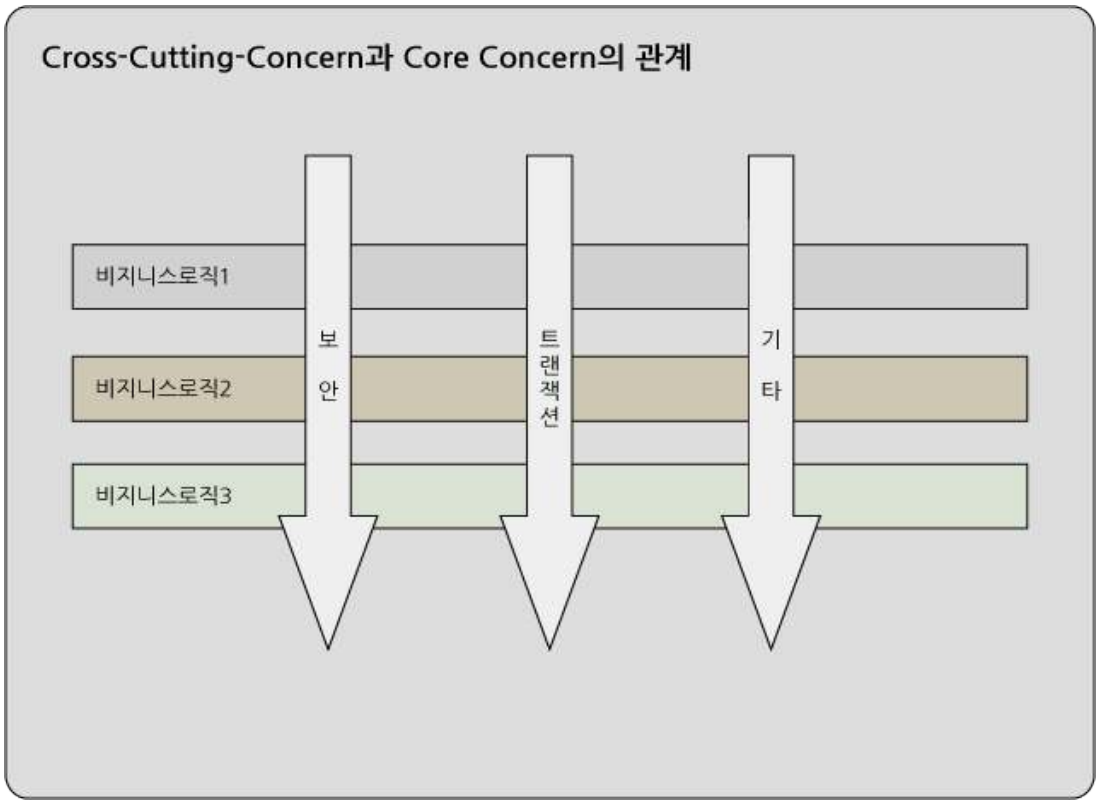
■ 관점 지향 프로그래밍은 핵심로직을 구현하는 코드와 공통 기능 관련 로직(로깅, 트랜잭션 관리, 보안 등)의 분리가 목적

■ 다수의 컴포넌트에 공통 기능 관련 로직이 산재할 때의 이슈

- ■ 시스템 전반적인 관심사를 구현한 코드가 열러 컴포넌트에 증복
- ■ 변경 이슈가 발생할 경우 개별적인 코드를 수정
- ■ 컴포넌트의 핵심기능이 기능적인 코드에 의해 오염됨

■ AOP는 어떤 클래스의 어떤 메소드가 실행되기 전•후에 다른 기능을 수행할 수 있는 코드를 캡슐화하는 것이다.(즉, 비기능적 요구사항이 핵심 애플리케

이션 코드에 나타나지 않도록 캡슐화 하는 것)



2) AOP의 효과

(1) AOP를 사용하면 핵심 애플리케이션에 기능별 계층을 적용

(2) 계층은 유연한 방식으로 애플리케이션 전반에 걸쳐 적용될 수 있으며, 핵심 애플리케이션은 그런 계층이 존재하는지도 알지 못함

(3) 어떤 클래스의 어떤 메소드가 실행되기 전과 후에 다른 기능을 수행할 수 있는 코드를 캡슐화하는 것(즉, 비기능적 요구사항이 핵심 애플리케이션 코드에 나타나지 않도록 캡슐화하는 것임)

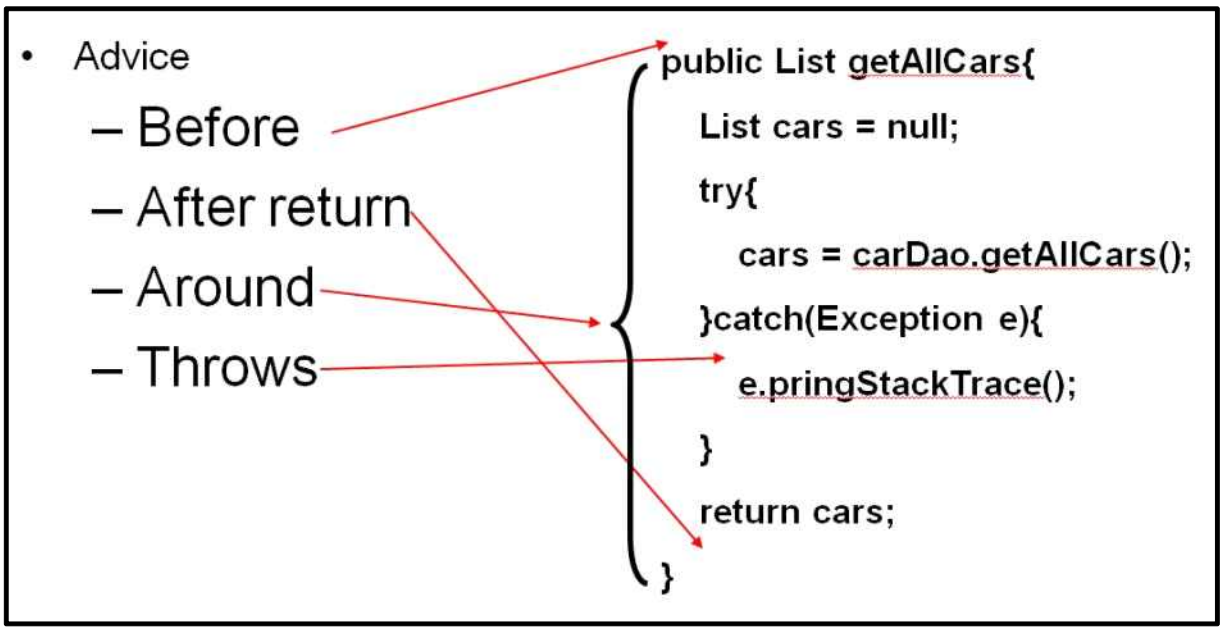
■ AOP 용어

■ Advice

• What(무엇을)과 When(언제)이 결합된 것

• 공통 관심 사항을 핵심 로직에 언제 적용할 것인가를 정의

• 구현하고자 하는 공통 관심 사항의 기능



■ Joinpoint

• Where(어디에)의 의미

• 공통 관심 사항을 적용할 수 있는 애플리케이션의 실행지점

• Advice를 적용하는 지점

■ Pointcut

• Advice가 실제로 적용되는 Jointpoint

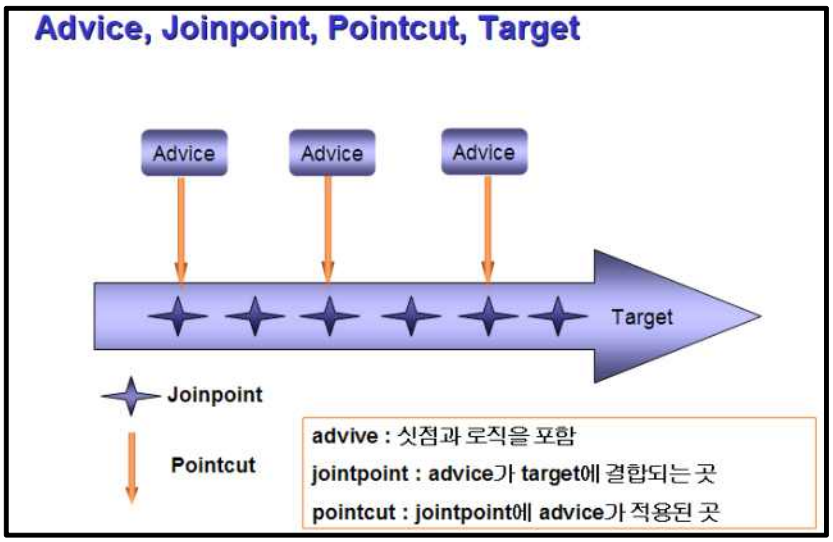
메서드명을 직접 명시하거나, 매칭 패턴을 나타내는 정규표현식을 정의하여 사용

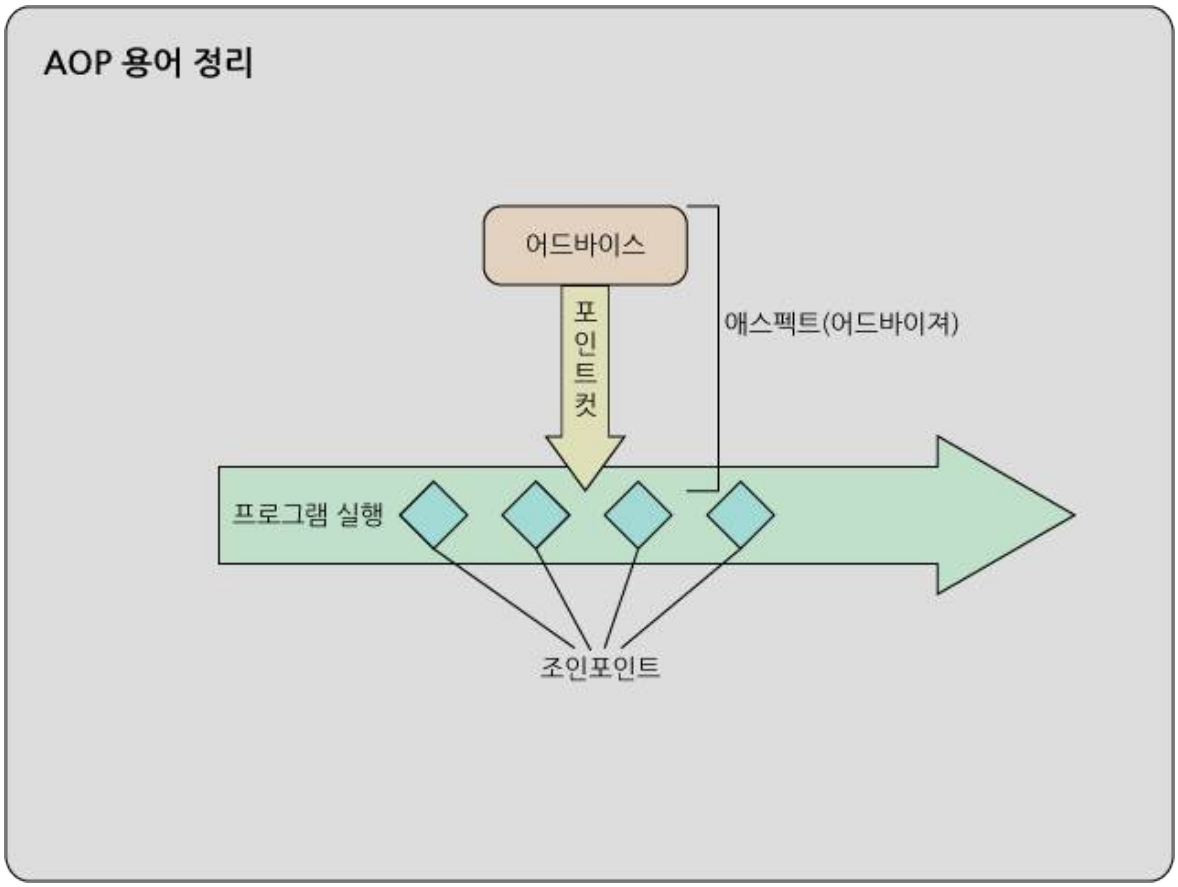
(4) 애스팩트(Aspect)

ㄱ. advice와 pointcut을 합친 것

ㄴ. 언제, 어디서, 무엇을 할지가 정해짐

ㄷ. 어드바이져(advisor)라고도 함





■ 스프링의 AOP 지원

(1) 프록시 기반 AOP(스프링 모든 버전에서 지원)

(2) @AspectJ 애너테이션 기반 애스펙트(스프링 2.0부터 지원)

(3) 순수 POJO 애스펙트(스프링 2.0부터 지원)

(4) AspectJ 애스펙트에 빈 주입(스프링의 모든 버전에서 지원)

(5) 스프링 애스펙트는 실행시 생성

(6) 스프링은 메서드 조인포인트만 지원

ㄱ. 필드나 생성자 조인포인트는 제공하지 않음

```
AspectJ가 지원하는 pointcut

1. execution : 메소드의 로직이 실행되는 시점
2. call : 메소드가 호출되는 시점
3. handler : 예외가 발생하는 시점
4. this : 현재 수행중인 객체
5. target : 대상 객체가 특정 타입일 때
6. within : 수행 코드가 특정 클래스에 포함될 때
7. cflow : 조인 포인트가 지정 메소드에 포함될 때
```

■ POJO 기반의 AOP 설정

1) applicationContextxml에 namespace 추가

applicationContext.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"
       xmlns:aop="http://www.springframework.org/schema/aop"
       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
          http://www.springframework.org/schema/aop
          http://www.springframework.org/schema/aop/spring-aop.xsd ">


</beans>
```

2) 공통 관심 사항 구현(Logging)

LoggingAspect.java

```
package aop.pojo;

import org.aspectj.lang.JoinPoint;

public class LoggingAspect {

     public void logging(JoinPoint joinPoint){

          String name = joinPoint.getSignature().getName();
          System.out.println("::: signatureName " + name);

     }
}
```

3) applicationContext.xml에 aop 설정추가

LoggingAspect.java

```
<bean id="log" class="aop.pojo.LoggingAspect"/>

<aop:config>
     <aop:aspect id="loggingAspect" ref="log">
 <aop:pointcut id="publicMethod" expression="execution(public**..*Imp.*(..))"/>
 <aop:before method="logging" pointcut-ref="publicMethod"/>
    </aop:aspect>
</aop:config>

<bean id="departmentService" class="aop.service.DepartmentServiceImp"/>
```

■ AOP 설정태그

■ <aop:aspect>

- Aspect를 설정함
- id : 유일한 aspect의 id, ref : aspect를 구현한 Bean 참조
- <aop:pointcut>와 advice를 표현하는 태그를 포함하고 있다.

■ <aop:pointcut>

- advice를 적용할 pointcut을 정의한다.
- id : pointcut를 구분하는 id값, expression : AspectJ 표현식

■ Advice Tag

태그	설명
<aop:before>	메소드 실행전에 적용되는 advice를 정의한다.
<aop:after-returning>	메소드가 정상적으로 실행된 후에 적용되는 advice를 정의한다.
<aop:after-throwing>	메소드가 예외를 발생시킬 때 적용되는 advice를 정의한다. try-catch블록에서 catch 블록과 비슷하다.
<aop:after>	메소드가 정상적으로 실행되던지 혹은 예외를 발생시키는지 여부에 상관없이 적용되는 advice를 정의한다. try-catch-finally에서 finally블록과 비슷하다.
<aop:arround>	메소드 호출 이전, 이후, 예외발생 등 모든 시점에 적용 가능한 advice를 정의한다.

■ Advice Tag의 속성값

- method : Advice 로직을 정의한 메소드 이름
- arg-names : Advice 메소드에 전달할 파라미터 이름
- pointcut-ref : 관련된 pointcut 이름
- returning : <aop:after-returning>에만 해당, 비즈니스 로직의 리턴값
- throwing : <aop:after-throwing>에만 해당, 메소드에 전달할 Exception 이름

■ JoinPoint 클래스

■ org.aspectj.lang.JoinPoint 임

■ 대상 객체 및 호출되는 메소드에 대한 정보, 전달되는 파라미터에 대한 정보가 필요한 경우 사용한다.

■ JointPoint의 주요 메소드

Return	Method	Description
Object	getTarget()	대상 객체
Object[]	getArgs()	파라미터 목록
Signature	getSignature()	메소스 정보
String	toLongString()	메소드 상세 정보
String	toShortString()	메소드 간략 정보

■ AspectJ 포인트컷 표기법

- [접근제한자] [리턴타입] [패키지.클래스].[메소드명](매개변수의 타입)
- 패턴 * : 모든 값, 패턴 .. : 0개 이상
- and(&&) 와 or(||)를 이용하여 표현식 연결 가능

Pointcut	선택된 Joinpoints
execution(public * *(..))	public 메소드 실행
execution(* set*(..))	이름이 set으로 시작하는 모든 메소드명 실행
execution(* set*(..))	이름이 set으로 시작하는 모든 메소드명 실행
execution (* com.xyz.service.AccountService.*(..))	AccountService 인터페이스의 모든 메소드 실행
execution(* com.xyz.service.*.*(..))	service 패키지의 모든 메소드 실행
execution(* com.xyz.service..*.*(..))	service 패키지와 하위 패키지의 모든 메소드 실행
within(com.xyz.service.*)	service 패키지 내의 모든 결합점
within(com.xyz.service..*)	service 패키지 및 하위 패키지의 모든 결합점
this(com.xyz.service.AccountService)	AccountService 인터페이스를 구현하는 프록시 개체의 모든 결합점

target(com.xyz.service.AccountService)	AccountService 인터페이스를 구현하는 대상 객체의 모든 결합점
args(java.io.Serializable)	하나의 파라미터를 갖고 전달된 인자가 Serializable인 모든 결합점
@target(org.springframework.transaction.annotation.Transactional)	대상 객체가 @Transactional 어노테이션을 갖는 모든 결합점
@within(org.springframework.transaction.annotation.Transactional)	대상 객체의 선언 타입이 @Transactional 어노테이션을 갖는 모든 결합점
@annotation(org.springframework.transaction.annotation.Transactional)	실행 메소드가 @Transactional 어노테이션을 갖는 모든 결합점
@args(com.xyz.security.Classified)	단일 파라미터를 받고, 전달된 인자 타입이 @Classified 어노테이션을 갖는 모든 결합점
bean(accountRepository)	“accountRepository” 빈
!bean(accountRepository)	“accountRepository” 빈을 제외한 모든 빈
bean(*)	모든 빈
bean(account*)	이름이 'account'로 시작되는 모든 빈
bean(*Repository)	이름이 “Repository”로 끝나는 모든 빈
bean(accounting/*)	이름이 “accounting/“로 시작하는 모든 빈
bean(*dataSource) ¦¦ bean(*DataSource)	이름이 “dataSource” 나 “DataSource” 으로 끝나는 모든 빈

■ after returning advice

applicationContext.xml

```
<aop:config>
     <aop:aspect id="logginAspect" ref="logging">
          <aop:pointcut id="publicMethod" expression="execution(public**..*Imp.*(..))"/>
          <aop:after-returning                                    method="loggingAfterReturing"
pointcut-ref="publicMethod"/>
     </aop:aspect>
</aop:config>
```

```
public void loggingAfterReturing(){
     // Aspect Logic
     ....
}
```

■ after returning advice

applicationContext.xml

```
<aop:config>
     <aop:aspect id="logginAspect" ref="logging">
          <aop:pointcut id="publicMethod“
 expression="execution(public**..*Imp.*(..))"/>
          <aop:after-returning method="loggingAfterReturing“
 pointcut-ref="publicMethod“ returing=“returnValue />
     </aop:aspect>
</aop:config>
```

```
public void loggingAfterReturing(Object returnValue){
     // Aspect Logic
     ....
}
```

```
public void loggingAfterReturing(JoinPoint joinPoint, Object returnValue){
     // Aspect Logic
     ....
}
```

applicationContext.xml

```
<aop:config>
     <aop:aspect id="logginAspect" ref="logging">
          <aop:pointcut id="publicMethod" expression="execution(public**..*Imp.*(..))"/>
          <aop:after-throwing method="loggingException" pointcut-ref="publicMethod"/>
     </aop:aspect>
</aop:config>
```

■ after throwing advice

```
public void loggingException(){
     // Aspect Logic
     ....
}
```

■ Annotaion으로 지정 가능

```
@Aspect
public class TestAdvice {
	@Around(value="execution(* org.jbm.model2..*.*(..))")
	 public Object beforeLogging(ProceedingJoinPoint  pjp) throws Throwable {
		//joinPoint가 해당 메서드
		Signature signature=  pjp.getSignature();
		System.out.println("aop(메서드 수행전)");
		System.out.println("이름 : " + signature.getName());
		System.out.println("타입 : " + signature.getDeclaringTypeName());
		Object result = pjp.proceed();
		System.out.println("aop(메서드 수행후)");
		return result;
	 }
}
```

3. Spring에서의 Transaction 처리

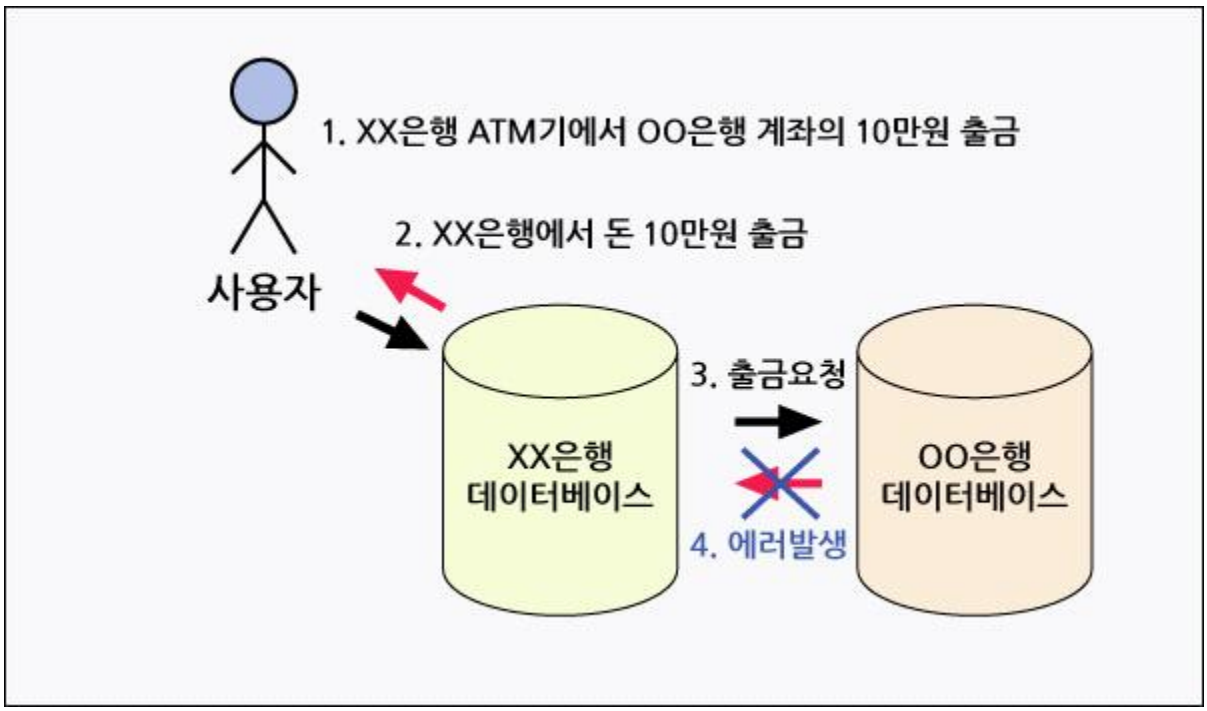
■ 트랜잭션이란?

■ 트랜잭션이란 단어의 본래 뜻은 '거래'라는 의미

■ 예를 들어 돈을 주었는데 물건을 받지 못한다면, 그 거래는 이루어 지지 못하고 원상태로 복구되어야 함

■ 이와 같이 쪼갤 수 없는 하나의 처리 행위를 원자적 행위라고 함

■ 여기서 쪼갤 수 없다는 말의 의미는 실제로 쪼갤 수 없다기 보다는 만일 쪼개질 경우 시스템에 심각한 오류를 초래할 수 있다는 것

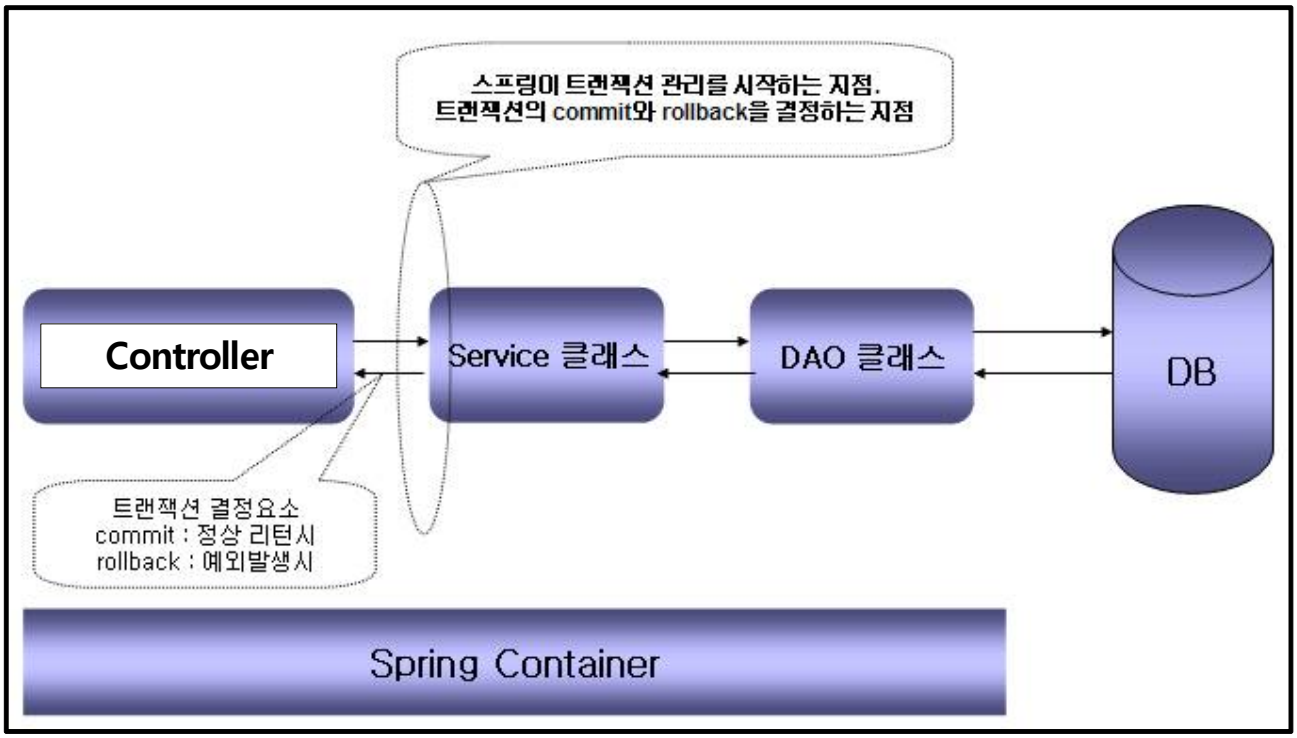


■ 선언적 Transaction 지원

■ 스프링 트랜잭션관리 기능을 사용하면 데이터베이스를 연동하는 다양한 기술들(jdbc, iBatis, Hibernate, JPA, ...)에 대해서 일관된 방법으로 트랜잭션을 관리할 수 있다.

■ 실제 소스 코드에는 트랜잭션을 관리하기 위한 코드가 필요하지 않다.

■ 웹 어플리케이션에서의 트랜잭션 처리



■ JDBC 기반 트랜잭션 관리자 설정

■ JDBC나 iBatis와 같이 JDBC를 이용해서 데이터베이스 연동을 처리하는 경우, DataSourceTransctionManager를 트랜잭션 관리자로 사용한다.

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

■ DataSourceTransactionManager는 dataSource프로퍼티를 사용해서 전달받은 DataSource로부터 Connection를 가져온 뒤, Connection의 commit(), rollback() 등의 메소드를 사용해서 트랜잭션을 관리한다.

■ 선언적 트랜잭션

■ 트랜잭션 처리를 코드에서 직접적으로 수행하지 않고, 설정 파일이나 어노테이션을 이용해서 트랜잭션의 범위, 롤백 규칙 등을 정의한다.

■ <tx:advice> 태그를 이용한 선언적 트랜잭션 처리

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

<tx:adviceid="txAdvice" transaction-manager="transactionManager">
     <tx:attributes>
         <tx:method name="get*" read-only="true"/>
         <tx:method name="add*" propagation="REQUIRED"/>
         <tx:method name="remove*" propagation="REQUIRED"/>
         <tx:method name="update*" propagation="REQUIRED"/>
     </tx:attributes>
</tx:advice>

<aop:config>
     <aop:pointcut id="serviceOperation"
                 expression="execution(* com.rpm.photo.service.*Service.*(..))"/>
     <aop:advisor advice-ref="txAdvice" pointcut-ref="serviceOperation"/>
</aop:config>
```

■ <tx:advice> 태그는 트랜잭션을 적용할 때 사용될 Advisor를 생성한다.

id : 생성될 트랜잭션 Advisor의 식별 값을 지정.

transaction-manager : TransactionManager 빈을 지정.

■ <tx:method>태그는 트랜잭션을 설정할 메소드 및 트랜잭션 속성을 지정한다.

■ <tx:advice> 태그는 Advisor만을 생성한다. 트랜잭션의 적용은 AOP를 통해서 이루어진다. <aop:config> 태그를 이용해서 <tx:advice>로 설정된 트랜잭션 Advisor를 적용하도록 설정한다.

■ Annotation으로 Transaction 적용하기

1) transactionManger추가

```
<!-- 트랜잭션매니져 -->
<bean id="transactionManager"
 p:dataSource-ref="dataSource"
  class="org.springframework.jdbc.datasource.DataSourceTransactionManager"/>
```

2) annotaion으로 트랜잭션적용 가능하게

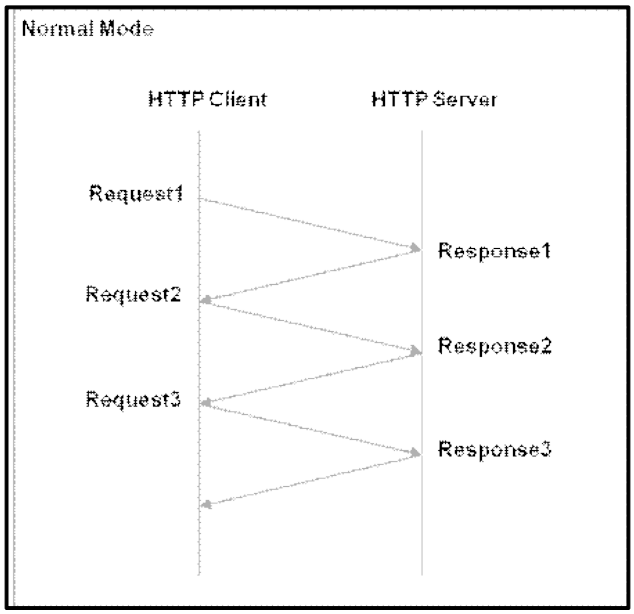
```
<tx:annotation-driven/>
```

3) 필요한 Service에 @Transactional 어노테이션 추가

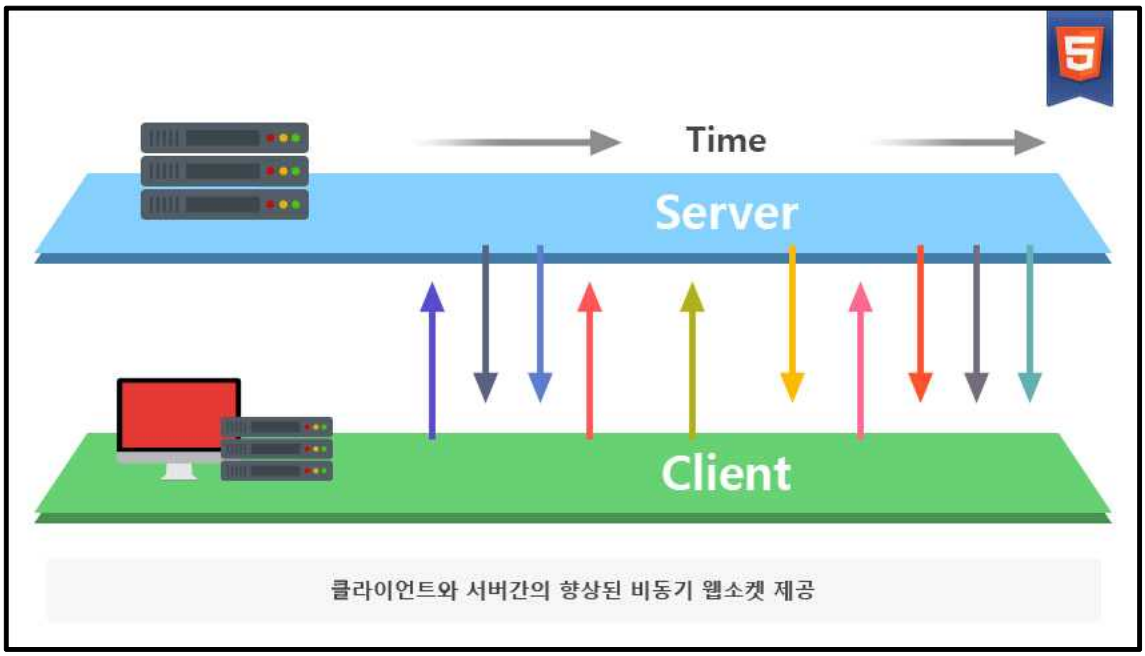
```
@Transactional
@Override
public boolean add(Idol idol) {

	if(idolsDAO.insert(idol)>0) {

		if(groupsDAO.updateMemberNum(idol.getGroupNo())>0){
			return true;
		}

	}
	return false;
}
```

1. 웹소캣의 개념

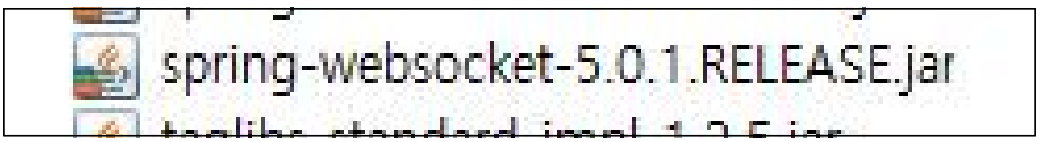
■ 기존의 HTTP프로토콜



■ 웹소캣 프로토콜



■ 라이브러리 등록



■ xxx-servlet.xml의 namespace등록

```
<websocket:handlers>
	<websocket:mapping handler="webSocketHandler" path="/chat"/>
 </websocket:handlers>

 <bean id="webSocketHandler"
 p:controller-ref="chattingController"
 class="websocket.ChattingWebSocketHandler"/>
```

■ ChattingWebsocketHandler

```
package websocket;

import org.springframework.web.socket.CloseStatus;
import org.springframework.web.socket.WebSocketMessage;
import org.springframework.web.socket.WebSocketSession;
import org.springframework.web.socket.handler.TextWebSocketHandler;

import controller.ChattingController;

public class ChattingWebSocketHandler extends TextWebSocketHandler{

	private ChattingController controller;

	public void setController(ChattingController controller) {
		this.controller = controller;
	}

	//연결이 끊어질때
	@Override
	public void afterConnectionClosed(WebSocketSession session, CloseStatus status) throws
Exception {
		controller.leave(session);
	}

	//연결이 설립될때(handshake)
	@Override
	public void afterConnectionEstablished(WebSocketSession session) throws Exception {
		controller.join(session);
	}

	//메세지가 올때
	@Override
	public void handleMessage(WebSocketSession session, WebSocketMessage<?> message) throws
Exception {
```

```
                controller.broadcast(session,(String)message.getPayload());
        }

}
```

■ ChattingController

```
package controller;

import java.util.Hashtable;
import java.util.Map;

import org.springframework.web.socket.TextMessage;
import org.springframework.web.socket.WebSocketSession;

public class ChattingController {

        // 여러 세션을 담는 맵
        private Map<String, WebSocketSession> sessions;

        //setter 주입용
        public void setSessions(Map<String, WebSocketSession> sessions) {
                this.sessions = sessions;
        }

        // 유저가 새로 들어왔음
        public synchronized void join(WebSocketSession session) {

                System.out.println("유저 들어옴 : " + session.getRemoteAddress());

                //맵에 session을 담음
                sessions.put(session.getId(), session);

                broadcast(session, "님이 들어왔습니다.");

        }

        // unicast(한 명의 유저에게만 보내는 것)
        private synchronized void unicast(WebSocketSession session, String msg) {

                try {
                        //메세지를 보냄
                        session.sendMessage(new TextMessage(msg));

                } catch (Exception e) {
```

```
                        e.printStackTrace();
                }

        }

        // 여러명의 유저에게 전부 보내는 것
        public synchronized void broadcast(WebSocketSession msgSession, String msg) {

                for (WebSocketSession session : sessions.values()) {

                        //ip와 port를 가져옴
                        String ip = msgSession.getRemoteAddress().toString();

                        //ip와 포트중 포트를 제거하고 ip만 자름
                        ip = ip.substring(1, ip.indexOf(":"));

                        unicast(session, ip + ":" + msg);
                }
        }

        public synchronized void leave(WebSocketSession session) {

                //사용자 세션을 제거함
                sessions.remove(session.getId());
                System.out.println("사용자 나감");

                broadcast(session, "님이 나가셨습니다.");
        }

}
```

■ Protocol

```
package vo;

public class Protocol {

        private int code,x,y;
        private String id,msg,nickname, img;

        public String getId() {
                return id;
        }

        public void setId(String id) {
```

```
                this.id = id;
        }
        public static final int ASK_JOIN = 1;
        public static final int ASK_LEAVE = 2;

        public static final int JOIN = 3;//실제 join
        public static final int LEAVE = 4;//실제 leave
        public static final int MESSAGE = 5;
        public static final int CHANGE_NICKNAME= 6;
        public static final int CHANGE_IMG = 7;
        public static final int XY = 8;

        public Protocol(int code) {
                super();
                this.code = code;
        }

        public Protocol(int code, String msg, String nickname,
                        String img) {
                this.code = code;
                this.msg = msg;
                this.nickname = nickname;
                this.img = img;
        }

        public int getCode() {
                return code;
        }
        public void setCode(int code) {
                this.code = code;
        }
        public int getX() {
                return x;
        }
        public void setX(int x) {
                this.x = x;
        }
        public int getY() {
                return y;
        }
        public void setY(int y) {
                this.y = y;
        }
        public String getMsg() {
                return msg;
        }
        public void setMsg(String msg) {
```

```
                this.msg = msg;
        }
        public String getNickname() {
                return nickname;
        }
        public void setNickname(String nickname) {
                this.nickname = nickname;
        }
        public String getImg() {
                return img;
        }
        public void setImg(String img) {
                this.img = img;
        }
}
```

■ chatting.jsp에서

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
        pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>웹소캣</title>
<link rel="stylesheet" href="/css/reset.css" />
<style>
#wrap {
        border: 1px solid #424242;
        width: 500px;
        height:500px;
        position: fixed;
        left: 50%;
        top: 50%;
        margin-top: -250px;
        margin-left: -250px;
}

#msgBox {
        width: 500px;
        height: 50px;
        text-align: center;
        line-height:50px;
        border-top:1px solid #424242;
}
```

```
#msgBox input {
        padding:10px;
        width:400px;
}

#chatList {
        height: 450px;
        width: 500px;
        overflow-y: scroll;
        overflow-x:hidden;
}
#close {
position:absolute;
left:0;
top:-30px;

}
#open {
position:absolute;
right:0;
top:-30px;
}

.chat {
        width: 500px;
        padding:10px;
}
.chat.system {
        color:red;
}
</style>
</head>
<body>
        <div id="wrap">
                <div id="chatList">
                <ul>
                        <!-- <li class="chat">안녕하세요?</li> -->
                </ul>
                </div>
                <div id="msgBox">
                        <input id="input" placeholder="내용을 입력하세요." type="text" />
                        <button id="sendBtn">전송</button>
                </div>
                <!-- #msgBox -->
                <button id="open">연결</button>
                <button id='close'>끊기</button>
        </div>
```

```
<!-- #wrap -->

<script src="/js/jquery.js"></script>
<script>
	var $textarea = $("#txt"),
	    textarea = $textarea.get(0),
	    $input = $('#input'),
	    $chatList = $("#chatList"),
	webSocket = null;//해당웹소켓 서버로 연결

	$('#close').click(function() {
		webSocket.close();
	});

	$('#open').click(function() {
		webSocket = new WebSocket("ws://192.168.0.101/chat");//해당웹소켓 서버로 연결

		//웹소켓 메서드들
		//소켓 에러났을때 onError(event) 밑에 정의한 함수를 호출
		webSocket.onerror = function(event) {
			alert(event);
		};
		//소켓 연결됐을때 onOpen(event)함수를 호출
		webSocket.onopen = function(event) {
			onOpen(event)
		};
		//웹소켓 서버로부터 메세지가 왔을때 onMessage함수를 호출
		webSocket.onmessage = function(event) {
			onMessage(event)
		};
		//소켓을 닫을때
		webSocket.onclose = function(event) {
			onClose(event);
		}

	});

	$input.keyup(function(e) {
		//채팅입력요소에서 엔터키 코드(13) 을 눌렀을때 send()함수 호출
		if (e.which === 13) {
			send();
		}
	});

	$("#sendBtn").click(send);
```

```
        //메세지왔을때 함수 처리
        function onMessage(event) {
                showMsg(event.data);
        }

        function showMsg(msg,type) {

                var $li = $("<li class='chat'>").text(msg);

                if(type=="S") {
                        $li.addClass("system");
                }//if end

                $li.appendTo("#chatList ul");


                $chatList.scrollTop($("#chatList ul").height());

        }


        function onOpen(event) {
                showMsg("연결성공!","S");
        }
        function onError(event) {
                alert(event);
        }
        function onClose(event) {
                showMsg("연결이 끊어졌습니다.","S");
        }

        //웹소켓에 메세지를 보내는 함수
        function send() {

                var text = $input.val();

                if (text.length > 0) {
                        //웹소켓에 메세지를 보내는 함수
                        webSocket.send(text);//서버에 메세지를 보내고
                        $input.val("").focus();//인풋요소에 값을
                }

        }

</script>
</body>
</html>
```