Spring Transaction

1. Transaction 이란?

- 어떤 작업 프로세스를 하나로 묶어 실행 중 하나의 작업이라도 실패하면 모두 실패 처리를 해주고, 전체 작업이 성공하면 성공 처리를 해주는 것이다.

2. 스프링의 트랜잭션 관리 지원

1)TransactionTemplate을 사용하여 트랜잭션 처리

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

<bean id="transactionTemplate" class="org.springframework.transaction.support.TransactionTemplate">
<property name="transactionManager" ref="transactionManager"/>
</bean>


<bean id="svc" class="part01.ServiceImp">
<property name="dao" ref="dao"/>
<property name="transactionTemplate" ref="transactionTemplate"/>
</bean>
```

- Configuration.xml에서 TransactionManager ,TransactionTemplate 빈 정의 후 사용 할 class에 참조한다.

```
public void insertPorcess() {

Object result = transactionTemplate.execute(new TransactionCallback<Object>() {
```

```
@Override
public Object doInTransaction(TransactionStatus status) {
try{
        dao.insertMethod(new MemDTO(44,"용팔이",50,"경기"));
        dao.insertMethod(new MemDTO(45,"유대위",20,"대전"));
        return "ok";
                }catch(Exception ex){
                status.setRollbackOnly();
                return "fail";
                }
                        }
                });
                        System.out.println("result:"+result);

        }//end insertProcess()
```

- 트랜잭션 처리를 할 메소드에 TransactionCallback클래스를 익명클래스로 사용하여 처리한다.

※ spring_05_tx -> part01 참조

2) 선언적 트랜잭션(<tx:advice> 태그를 이용)

```
<?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:tx="http://www.springframework.org/schema/tx"
xmlns:aop="http://www.springframework.org/schema/aop"

xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-3.2.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-3.2.xsd">
```

- <tx:advice> 태그를 이용해서 트랜잭션 속성을 정의하기 위해
  tx , aop 네임스페이스 속성 추가한다.

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

```
<tx:advice id="txAdvice" transaction-manager="transactionManager">
<tx:attributes>
<tx:method name="insertProcess" rollback-for="java.lang.Exception" />
</tx:attributes>
</tx:advice>
```

- <tx:advice>태그, <tx:attributes>태그, <tx:method>태그를 이용하여 트랜잭션 속성 정의 한다.

```
<aop:config>
<aop:pointcut expression="execution(*part02.ServiceImp.insertProcess(..))"
id="aa" />
<aop:advisor advice-ref="txAdvice" pointcut-ref="aa" />
</aop:config>
```

- <tx:advice> 태그는 Advisor만 생성하는 것일뿐, 실제로 트랜잭션을 적용하는 것은 아니다.
  트랜잭션의 적용은 AOP를 통하여 이루어진다.

※ spring_05_tx -> part02 참조

3) 선언적 트랜잭션(@Transactional 어노테이션 사용)

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

```
<tx:annotation-driven transaction-manager="transactionManager"/>
```

- tx:annotation-driven 태그를 설정한다.

```
@Transactional(rollbackFor=java.lang.Exception.class)
@Override
public void insertProcess() {
        dao.insertMethod(new MemDTO(54,"용팔이",50,"경기"));
        dao.insertMethod(new MemDTO(55,"유대위",20,"대전"));

}//end insertProcess()
```

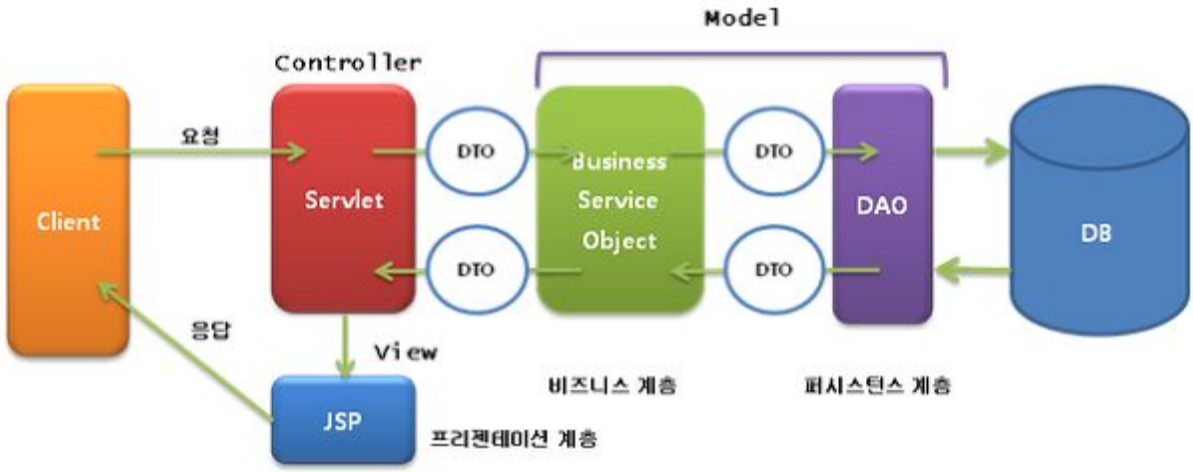
- 트랜잭션이 설정된 메소드(선언적 트랜잭션2가지방법)에서는 try~catch을 설정하면 안되고, 메소드를 호출하는 곳에서 try~catch을 한다.
- 트랜잭션이 설정된 메소드에서 예외처리하면 트랜잭션이 적용이 안된다.

※ spring_05_tx -> part03 참조

Spring MVC

MVC란 Model-View-Controller의 약자로, 사용자 인터페이스와 비지니스 로직을 분리하여 웹 개발을 하는것을 가장 큰 장점으로 한다.
MVC 패턴도 MVC 모델 1과 MVC 모델 2 로 나뉘어져 있는데, 요즘에는 MVC라고 하면 당연히 MVC 모델 2 를 의미한다.

Model : 모델은 애플리케이션의 정보, 즉 데이터를 나타낸다.
View : 뷰는 사용자에게 보여주는 인터페이스, 즉 화면을 이야기한다.
자바 웹 애플리케이션에서는 JSP를 의미한다.
Controller : 컨트롤러는 비지니스 로직과 모델의 상호동작의 조정 역할을 한다.
MVC2에서는 서블릿이 흐름을 제어하는 컨트롤러 역할을 수행한다.



<MVC Model2의 처리 흐름 >

Spring 한글처리

1. GET 방식

- POST에서는 헤더값으로 통해 데이터를 주고받기 때문에 서블릿에서 어느 정도 컨트롤이 가능하지만 GET은 URL에 직접 데이터를 추가하여 전송하는 방식이기 서블릿의 영역 밖에 존재한다. 그러므로 GET을 통해 한글을 데이터를 받고자 할 때는 서버에서 직접 URL을 UTF-8로 인코딩할 수 있게끔 설정해 주어야 한다.

- [CATALINA_HOME]\conf 경로로 찾아가 Server.xml 파일을 열어서 Server.xml에서 추가해줘야 할 부분은 다음과 같다.

```
<Connector connectionTimeout="20000" port="8080" protocol="HTTP/1.1" redirectPort="8443"
URIEncoding="UTF-8"/>
<Connector port="8009" protocol="AJP/1.3" redirectPort="8443" URIEncoding="UTF-8"/>
```

- 위의 두 요소에서 굵은 글씨 부분을 추가해주면 된다. 위에는 8080포트에서 들어오는 URI요청을 인코딩해주고 아래는 8009포트에서 보내지는 요청을 인코딩해준다.
- 보통 8080 포트를 많이 사용하므로 위에만 추가해줘도 되지만 만약을 대비하여 아래의 커넥터에도 인코딩 속성을 추가해주도록 하자.

2. POST 방식

- 스프링에서 POST 전송방식의 UTF-8 인코딩은 정말 간단하다. web.xml에 스프링이 제공하는 CharacterEncodingFilter를 걸어주면 끝이다.

```
<filter>
	<filter-name>encodingFilter</filter-name>
	<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
	<init-param>
		<param-name>encoding</param-name>
		<param-value>UTF-8</param-value>
	</init-param>
</filter>

<filter-mapping>
	<filter-name>encodingFilter</filter-name>
	<url-pattern>/*</url-pattern>
</filter-mapping>
```

etc1) Servlet에서 post 한글 처리 방식

```
protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {

req.setCharacterEncoding("UTF-8");

}
```

etc2) Filter인터페이스를 상속받아 클래스로 구현<@Filter 등록을 시킴(어노테이션)>

```
@WebFilter("/NamePro")
public class CharacterFilter implements Filter {

	//필터가 웹콘테이너에서 삭제될때 호출한다.
	@Override
	public void destroy() {

	}


	//체인을 따라 다음에 존재하는 필터로 이동한다.
	//체인의 가장 마지막에는 클라이언트가 요청한 최종 자원이 위치한다.
	@Override
	public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
			throws IOException, ServletException {
		request.setCharacterEncoding("UTF-8");
		chain.doFilter(request, response);


	}


	//필터가 웹 컨테이너에 생성한 후 초기화할때 호출한다.
	@Override
	public void init(FilterConfig arg0) throws ServletException {

	}

	}//end class
```