

9장 스프링 트랜잭션 사용하기

강사 김영석

A top-down view of a wooden desk. On the desk, there is a silver laptop with a black keyboard, a pair of black-rimmed glasses, a white coffee cup with a yellow handle, and a small green succulent in a pot. The word 'CONTENT' is written in large, white, sans-serif capital letters over the left side of the image.

CONTENT

1

트랜잭션 기능

2

계좌 이체를 통한 트랜잭션

3

스프링의 트랜잭션 속성

4

트랜잭션 실습 하기

1. 트랜잭션 기능

✓ 트랜잭션

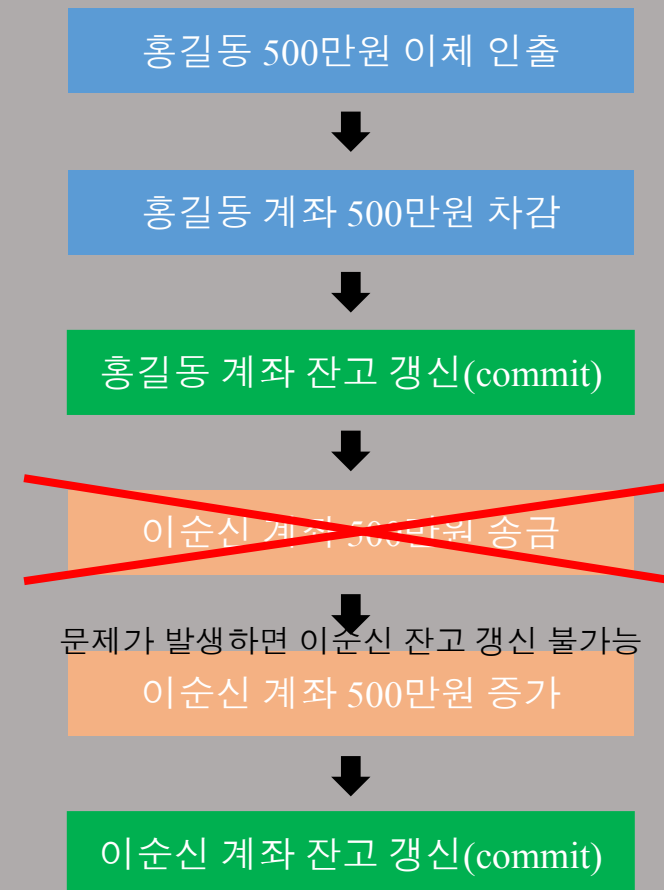
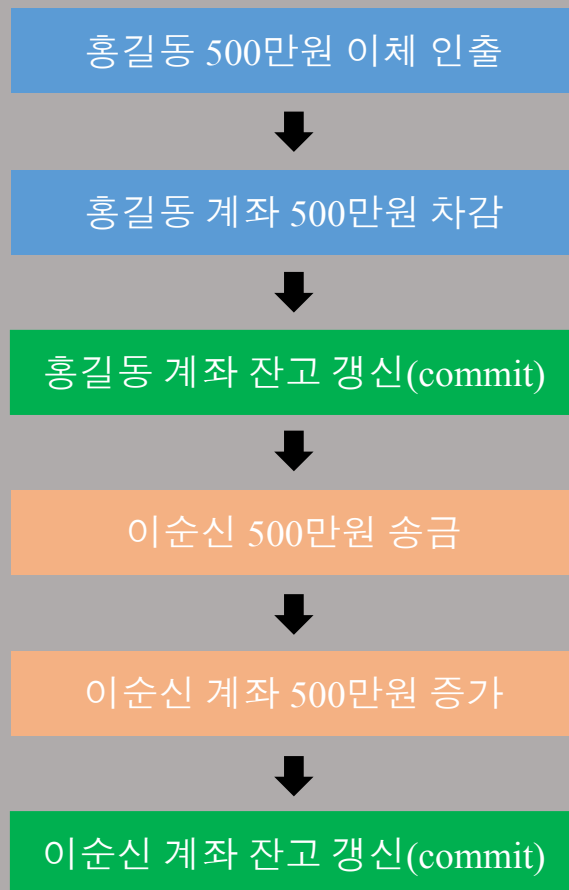
- 여러 개의 DML 명령문을 하나의 논리적인 작업 단위로 묶어 관리하는 것을 의미한다.
- All 또는 nothing 방식으로 작업을 처리함으로써 작업의 일관성을 유지할 수 있다는 장점이 있다.
- 웹 애플리케이션에서는 Service 클래스의 각 메서드가 애플리케이션의 단위 기능을 수행하게 된다.

✓ 웹 애플리케이션에서 묶어서 처리하는 단위 기능의 예는 아래와 같다.

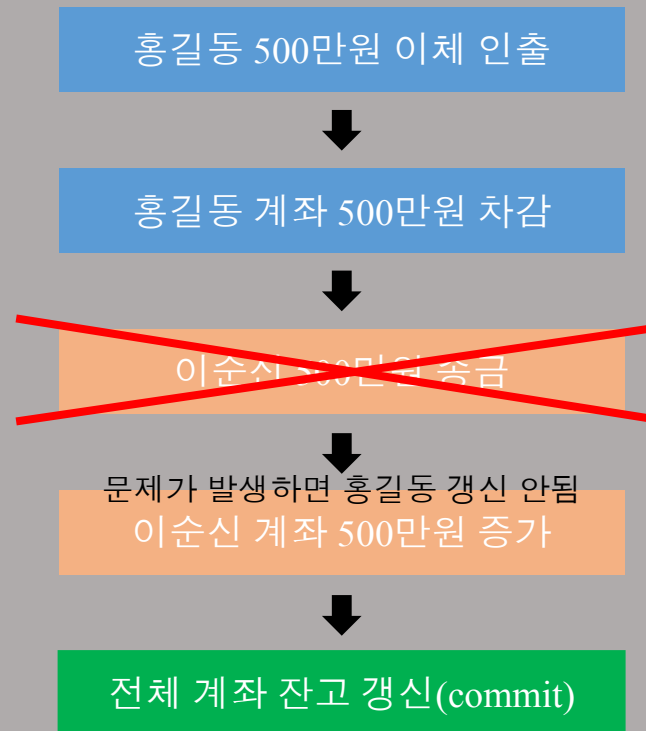
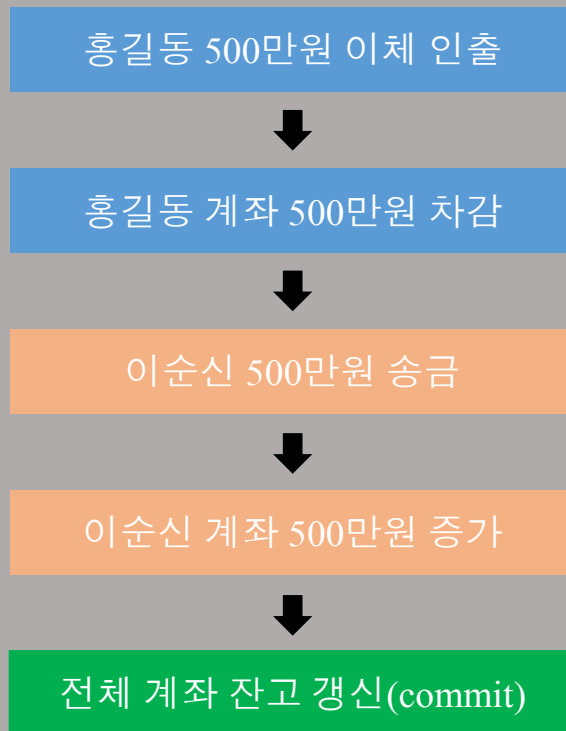
- ① 게시판 글을 조회할 때 해당 글을 조회하는 기능과 조회수를 늘려주는 기능
- ② 쇼핑몰에서 상품을 주문했을 때 주문 상품을 저장하는 것과 주문자의 포인트를 늘려주는 기능
- ③ 은행에서 송금 할 때 송금자의 잔고를 갱신하고 수신자의 잔고도 갱신하는 기능

2. 계좌 이체를 통한 트랜잭션

✓ 트랜잭션을 적용하기 전 이체 상황은 아래와 같다.



✓ 트랜잭션을 적용하기 후 이체 상황은 아래와 같다.



3. 스프링의 트랜잭션 속성

✓ 스프링의 여러 가지 트랜잭션 속성

속성	기능
propagation	트랜잭션 전파 규칙 설정한다.
isolation	트랜잭션 격리 레벨 설정한다.
readOnly	읽기 전용 여부 설정한다.
rollbackFor	트랜잭션을 롤백(rollback)할 예외 타입 설정한다.
norollbackFor	트랜잭션을 롤백하지 않을 예외 타입 설정한다.
timeout	트랜잭션 타임 아웃 시간 설정한다.

✓ propagation 속성이 가지는 값

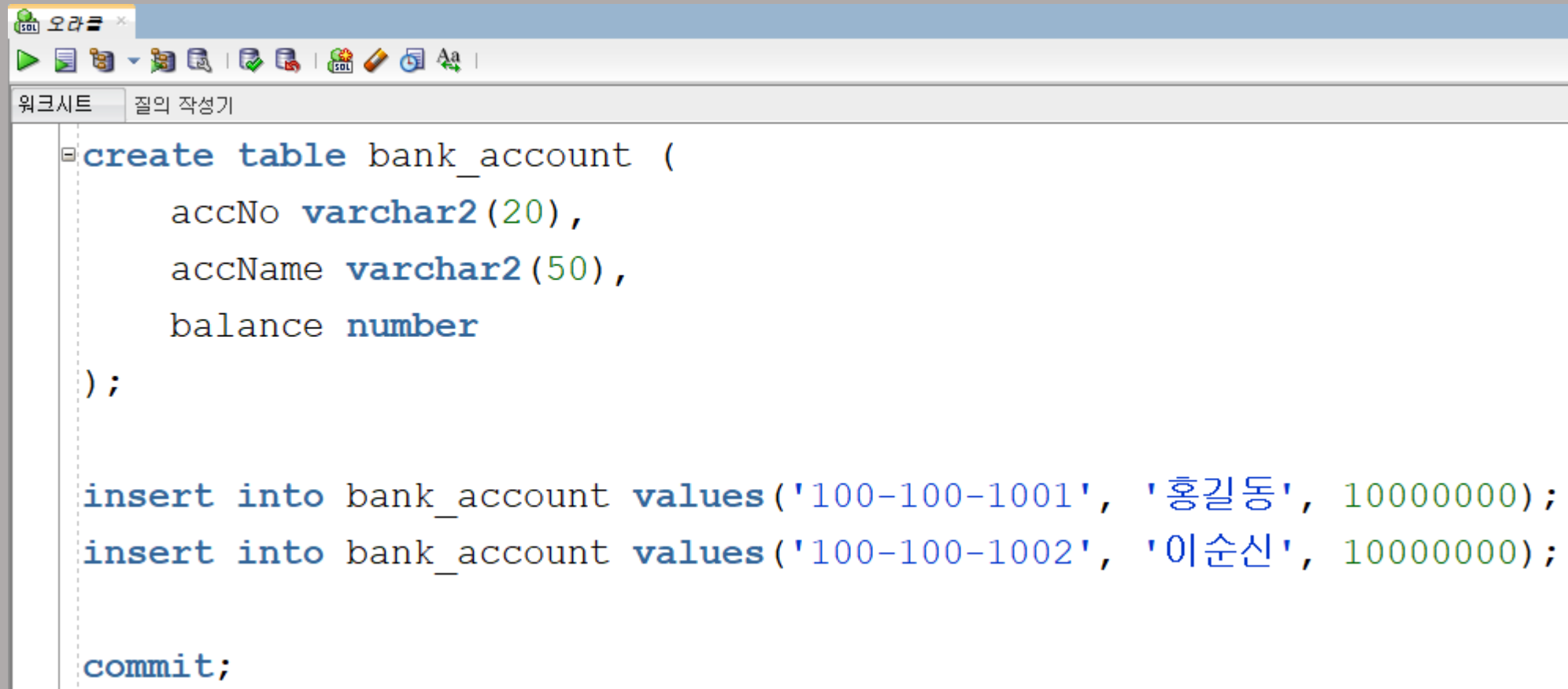
값	의미
REQUIRED	<ul style="list-style-type: none">- 트랜잭션 필요하고 진행 중인 트랜잭션이 있는 경우 해당 트랜잭션 사용한다.- 트랜잭션이 없으면 새로운 트랜잭션 생성하고 기본값이다.
MANDATORY	<ul style="list-style-type: none">- 트랜잭션 필요하다.- 진행 중인 트랜잭션이 없는 경우 예외 발생한다.
REQUIRED_NEW	<ul style="list-style-type: none">- 항상 새로운 트랜잭션을 생성한다.- 진행 중인 트랜잭션이 있는 경우 기존 트랜잭션을 일시 중지 시키고 새로운 트랜잭션을 시작한다.- 새로 시작된 트랜잭션이 종료되면 기존 트랜잭션이 계속 진행 된다.
SUPPORTS	<ul style="list-style-type: none">- 트랜잭션이 필요 없다.- 진행 중인 트랜잭션이 있는 경우 해당 트랜잭션을 사용한다.
NOT_SUPPORTED	<ul style="list-style-type: none">- 트랜잭션이 필요 없다.- 진행 중인 트랜잭션이 있는 경우 기존 트랜잭션을 일시 중지 시키고 메서드를 실행한다.- 메서드 실행이 종료되면 기존 트랜잭션이 계속 진행 된다.
NEVER	<ul style="list-style-type: none">- 트랜잭션이 필요 없다.- 진행 중인 트랜잭션이 있는 경우 예외가 발생한다.
NESTED	<ul style="list-style-type: none">- 트랜잭션이 필요하다.- 진행 중인 트랜잭션이 있는 경우 트랜잭션에 중첩된 트랜잭션에서 메서드를 실행 한다.- 트랜잭션이 없으면 새로운 트랜잭션을 생성한다.

✓ isolation 속성이 가지는 값

값	의미
DEFAULT	- 데이터베이스에서 제공하는 기본 설정에 사용한다.
READ_UNCOMMITTED	- 다른 트랜잭션에서 커밋하지 않은 데이터는 읽기 가능하다.
READ_COMMITTED	- 커밋한 데이터만 읽기가 가능하다.
REPEATABLE_READ	- 현재 트랜잭션에서 데이터를 수정하지 않았다면 처음 읽어 온 데이터와 두번째 읽어온 데이터가 동일하다.
SERIALIZABLE	- 같은 데이터에 대해 한 개의 트랜잭션만 수행 가능하다.

4. 트랜잭션 실습 하기

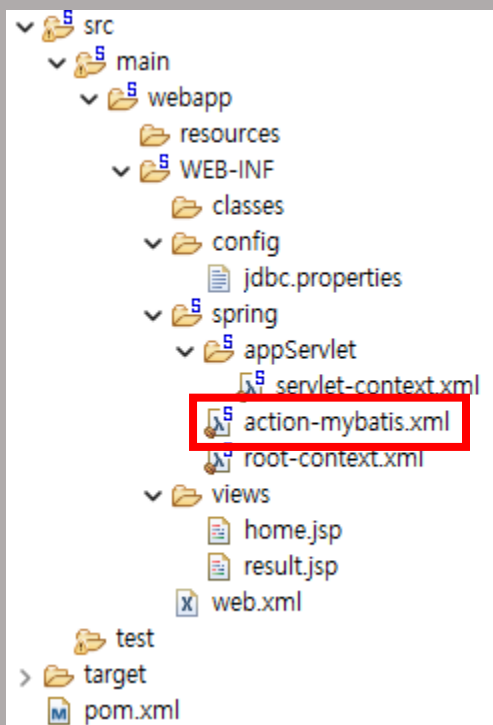
✓ 테이블 만들기



```
오라클
워크시트 | 질의 작성기
create table bank_account (
    accNo varchar2(20),
    accName varchar2(50),
    balance number
);

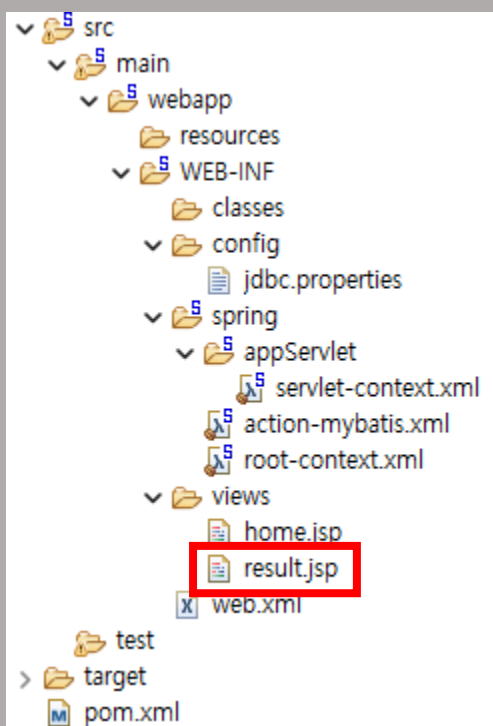
insert into bank_account values('100-100-1001', '홍길동', 10000000);
insert into bank_account values('100-100-1002', '이순신', 10000000);

commit;
```

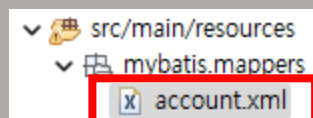


jdbc.properties, action-mybatis.xml 은 pro08에서 복사 붙여넣기 후 수정한다.

```
*action-mybatis.xml
19      <property name="password" value="${jdbc.password}"></property>
20    </bean>
21    <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
22      <property name="dataSource" ref="dataSource" />
23      <property name="mapperLocations" value="classpath:mybatis/mappers/*.xml" />
24    </bean>
25
26    <bean id="sqlSession" class="org.mybatis.spring.SqlSessionTemplate">
27      <constructor-arg index="0" ref="sqlSessionFactory"></constructor-arg>
28    </bean>
29    <bean name="transactionTemplate" class="org.springframework.transaction.support.TransactionTemplate">
30      <property name="transactionManager" ref="transactionManager"/>
31      <property name="propagationBehavior" value="0"/>
32    </bean>
33    <bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
34      <property name="dataSource" ref="dataSource"></property>
35    </bean>
36    <tx:annotation-driven transaction-manager="transactionManager"/>
37  </beans>
```

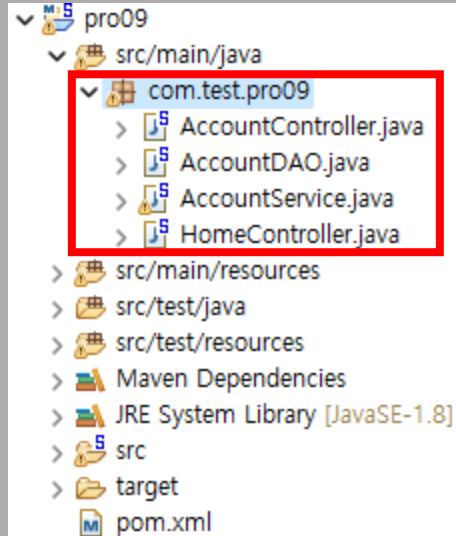


```
result.jsp
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2     pageEncoding="UTF-8"%>
3 <!DOCTYPE html>
4 <html>
5 <head>
6 <meta charset="UTF-8">
7 <title>결과창</title>
8 </head>
9 <body>
10     <h1>송금이 완료 되었습니다.</h1>
11 </body>
12 </html>
```




account.xml


```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
3   "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
4
5 <mapper namespace="mapper.account">
6   <update id="updateBalance1">
7     <![CDATA[
8       update bank_account set balance=balance-5000000 where accNo = '100-100-1001'
9     ]]>
10  </update>
11  <update id="updateBalance2">
12    <![CDATA[
13      update bank_account set balance=balance+5000000 where accNo = '100-100-1002'
14    ]]>
15  </update>
16 </mapper>
```



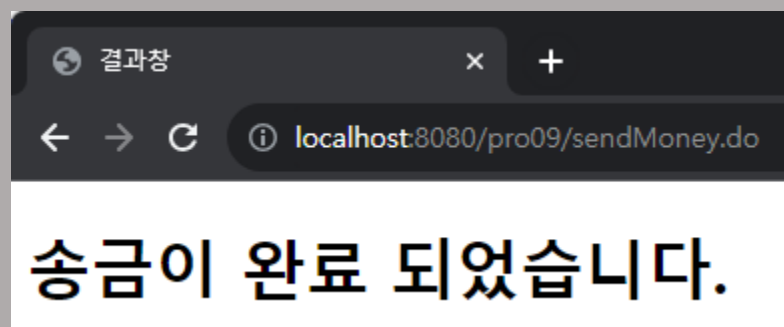
```
*AccountController.java 33
1 package com.test.pro09;
2
3 import javax.servlet.http.HttpServletRequest;
11
12 @Controller
13 public class AccountController extends MultiActionController {
14     @Autowired
15     private AccountService service;
16
17     @RequestMapping(value="/sendMoney.do")
18     public ModelAndView sendMoney(HttpServletRequest request, HttpServletResponse response)
19         throws Exception {
20         service.sendMoney();
21         ModelAndView mav = new ModelAndView("result");
22         return mav;
23     }
24 }
```

```
*AccountService.java
1 package com.test.pro09;
2
3 import javax.inject.Inject;
10
11
12 @Service
13 public class AccountService{
14     @Autowired
15     private AccountDAO accDAO;
16     @Inject
17     private TransactionTemplate transactionTemplate;
18
19     public void sendMoney() {
20         System.out.println("transactionTemplet : " + transactionTemplate);
21         transactionTemplate.execute(new TransactionCallbackWithoutResult() {
22
23             @Override
24             protected void doInTransactionWithoutResult(TransactionStatus status) {
25                 // TODO Auto-generated method stub
26                 accDAO.updateBalance1();
27                 accDAO.updateBalance2();
28             }
29         });
30
31     }
32 }
```




```
AccountDAO.java
1 package com.test.pro09;
2
3 import org.apache.ibatis.session.SqlSession;
4
5
6
7 @Repository
8 public class AccountDAO {
9     @Autowired
10     private SqlSession sqlSession;
11
12     public void updateBalance1() {
13         sqlSession.update("mapper.account.updateBalance1");
14     }
15     public void updateBalance2() {
16         sqlSession.update("mapper.account.updateBalance2");
17     }
18 }
```



스크립트 출력 x | 실행의 결과 x

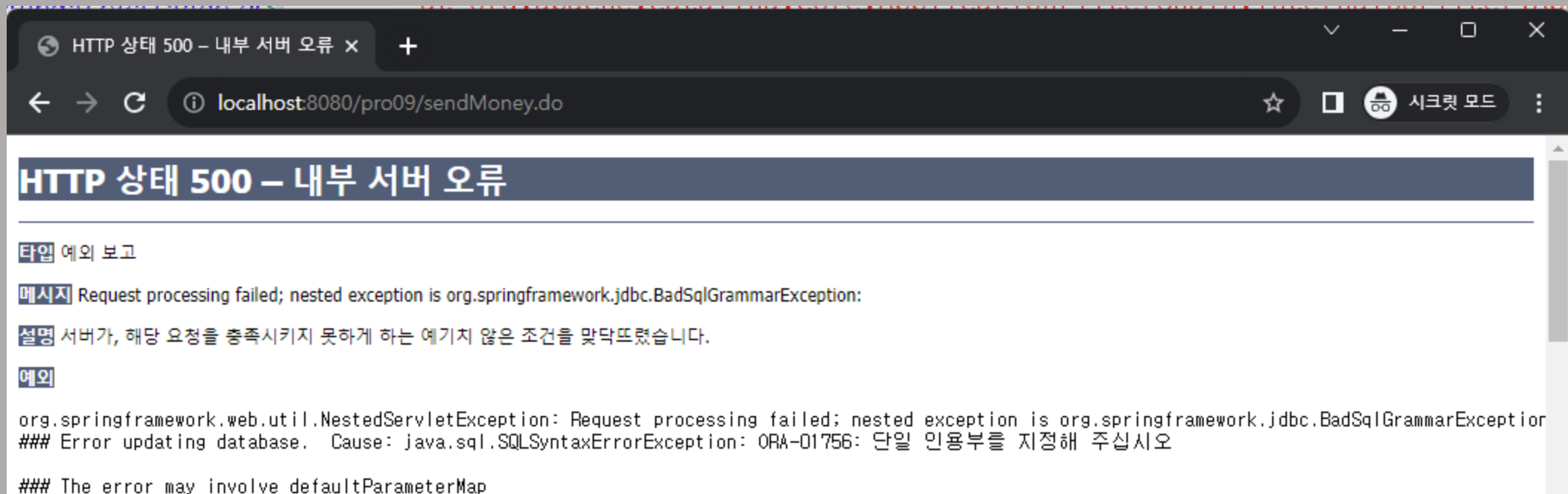
SQL | 인출된 모든 행: 2(0.002초)

	ACCNO	ACCNAME	BALANCE
1	100-100-1001	홍길동	5000000
2	100-100-1002	이순신	15000000



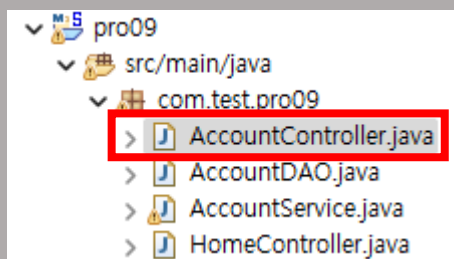
```
*account.xml
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
3   "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
4
5 <mapper namespace="mapper.account">
6
7   <select id="updateBalance1">
8     update bank_account set balance=balance-5000000 where accNo='100-100-1001'
9   </select>
10  <select id="updateBalance2">
11    update bank_account set balance=balance+5000000 where accNo='100-100-1002'
12  </select>
13 </mapper>
```

강제 에러 만들기

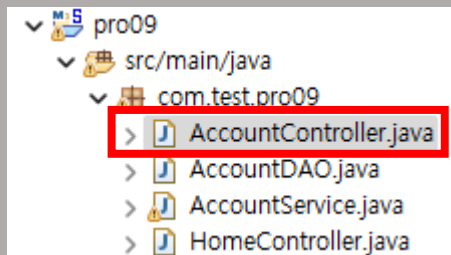


스크립트 출력 x		질의 결과 x	
SQL 인출된 모든 행: 2(0,001초)			
	ACCNO	ACCNAME	BALANCE
1	100-100-1001	홍길동	5000000
2	100-100-1002	이순신	15000000

✓ 애너테이션을 이용한 트랜잭션 사용하기



```
AccountController.java
1 package com.test.pro09;
2
3 import javax.servlet.http.HttpServletRequest;
12
13 @Controller
14 @EnableTransactionManagement
15 public class AccountController extends MultiActionController {
16     @Autowired
17     private AccountService service;
18
19     @RequestMapping(value="/sendMoney.do")
20     public ModelAndView sendMoney(HttpServletRequest request, HttpServletResponse response)
21         throws Exception {
22         service.sendMoney();
23         ModelAndView mav = new ModelAndView("result");
24         return mav;
25     }
26 }
```



```
AccountService.java
1 package com.test.pro09;
2
3 import javax.inject.Inject;
12
13
14 @Service
15 @Transactional(propagation = Propagation.REQUIRED)
16 public class AccountService{
17     @Autowired
18     private AccountDAO accDAO;
19
20     public void sendMoney() {
21         accDAO.updateBalance1();
22         accDAO.updateBalance2();
23     }
```

A photograph of a server room with rows of server racks on both sides of a central aisle. The racks have glass doors and internal components are visible, with many small blue lights glowing. The ceiling has several long, rectangular light fixtures. The overall atmosphere is dimly lit, emphasizing the blue light from the servers.

수고하셨습니다.