트랜잭션 - Propagation(전파속성)

프로젝트명 : B09cTransactionPropagation

준비사항

- 기존 B09bTransactionTemplate 을 복사한다.
- 컨텍스트 루트 및 settings.gradle 을 변경한다.
- gradle을 리플레쉬 한다.

required (0)	Default, 전체 처리
supports (1)	기존 트랜잭션에 의존
mandatory (2)	트랜잭션에 꼭 포함 되어야 함. 트랜잭션이 있는 곳에서 호출해야 됨.
requires_new (3)	각각 트랜잭션을 처리
not_supported (4)	트랜잭션에 포함 하지 않음. 기존의 트랜잭션이 존재하면 일시 중지 하고 메소드 실행이 끝난 후에 트랜잭션을 계속 진행한다.
never (5)	트랜잭션에 절대 포함 하지 않음. 트랜잭션이 있는 곳에서 호출하면 에러 발생.

Propagation의 해석상 전파속성인데
트랜잭션 A에 트랜잭션 B가 참여할 때의 참여조건이라고 보는 것이 이해하기 쉽다.

서비스 : com.edu.springboot.jdbc.ITicketService.java

```java
package com.edu.springboot.jdbc;

import org.apache.ibatis.annotations.Mapper;

@Mapper
public interface ITicketService {

    public int ticketInsert(TicketDTO ticketDTO);
    public int payInsert(PayDTO payDTO);

    public int memberRegist(TicketDTO ticketDTO);
}
```

Mapper : src/main/resources/mybatis/mapper/TicketPayDAO.xml

```
16
17      <insert id="memberRegist" parameterType="com.edu.springboot.jdbc.TicketDTO">
18          insert into member (id, pass, name) values (#{userid}, '^^', '티켓구매')
19      </insert>
20  </mapper>
```

회원작업을 위한 클래스 : com.edu.springboot.jdbc.AddMember.java

```
1  package com.edu.springboot.jdbc;
2
3  import org.springframework.beans.factory.annotation.Autowired;
10
11 @Service
12 public class AddMember {
13
14     @Autowired
15     ITicketService dao;
16
17     @Autowired
18     TransactionTemplate transactionTemplate;
19
20     @Transactional(propagation=Propagation.REQUIRES_NEW)
21 //  @Transactional(propagation=Propagation.REQUIRED)
22     public void memberInsert(TicketDTO ticketDTO, String errFlag) {
23         try {
24             transactionTemplate.execute(new TransactionCallbackWithoutResult() {
25                 @Override
26                 protected void doInTransactionWithoutResult(
27                         TransactionStatus status) {
28                     if(errFlag!=null && errFlag.equals("2")) {
29                         int money = Integer.parseInt("200원");
30                     }
31
32                     int result3 = dao.memberRegist(ticketDTO);
33                     if(result3==1)
34                         System.out.println("member 테이블 입력성공");
35                 }
36             });
37         }
38         catch (Exception e) {
39             //e.printStackTrace();
40             System.out.println("member 테이블 Rollback됨");
41         }
42     }
43 }
```

컨트롤러 : com.edu.springboot.MainController.java

```java
@Controller
public class MainController {

    @Autowired
    ITicketService dao;

    @Autowired
    TransactionTemplate transactionTemplate;

    @Autowired
    AddMember addMember;

    @RequestMapping("/")
    public String home() {
        return "home";
    }
```

```java
    //티켓구매
    @RequestMapping(value="/buyTicket.do", method=RequestMethod.GET)
    public String buy1() {
        return "buy";
    }
    @RequestMapping(value="/buyTicket.do", method=RequestMethod.POST)
    public String buy2(TicketDTO ticketDTO, PayDTO payDTO,
            HttpServletRequest req, Model model) {
        String viewPath = "success";
        try {
            transactionTemplate.execute(new TransactionCallbackWithoutResult() {
                @Override
                protected void doInTransactionWithoutResult(
                        TransactionStatus status) {

                    String errFlag = req.getParameter("err_flag");

                    //회원 정보 입력(추가작업)
                    addMember.memberInsert(ticketDTO, errFlag);

                    //DB처리1
                    payDTO.setAmount(ticketDTO.getT_count() * 10000);
                    int result1 = dao.payInsert(payDTO);
                    if(result1==1)
                        System.out.println("transaction_pay 입력성공");

                    //비즈니스로직 처리(의도적인 에러발생)
                    if(errFlag!=null && errFlag.equals("1")) {
                        int money = Integer.parseInt("100원");
                    }
```

```
66
67                      //DB처리2
68                      int result2 = dao.ticketInsert(ticketDTO);
69                      if(result2==1)
70                          System.out.println("transaction_ticket 입력성공");
71
72                      model.addAttribute("ticketDTO", ticketDTO);
73                      model.addAttribute("payDTO", payDTO);
74                  }
75              });
76          }
77          catch (Exception e) {
78              //e.printStackTrace();
79              System.out.println("transaction 테이블 Rollback됨");
80              viewPath = "error";
81          }
82          return viewPath;
83      }
84  }
85
```

구매페이지 변경 : webapp/WEB-INF/views/buy.jsp

```
28          <tr>
29              <th>에러발생</th>
30              <td>
31                  <input type="checkbox" name="err_flag" value="1" />
32                  티켓구매에서 예외발생
33                  <input type="checkbox" name="err_flag" value="2" />
34                  회원입력에서 예외발생
35              </td>
36          </tr>
37      </table>
38      <input type="submit" value="전송하기" />
39      </form>
```

테스트

Propagation.REQUIRED

- 테스트1 : 체크없음
    - 모든 테이블 입력됨
- 테스트2 : 티켓구매 예외
    - 모든 테이블 입력안됨
- 테스트3 : 회원입력 예외
    - 모든 테이블 입력안됨

Propagation.REQUIRES_NEW

- 테스트1 : 체크없음
    - 모든 테이블 입력됨
- 테스트2 : 티켓구매 예외
    - 회원테이블 입력됨
- 테스트3 : 회원입력 예외
    - 모든 테이블 입력안됨