**트랜잭션 (Transaction)**

갑자기 "트랜잭션"이라는 용어가 나와서 뜬금 없다고 생각할 수도 있겠지만 트랜잭션과 예외처리는 매우 밀접한 관련이 있다. 트랜잭션과 예외처리가 서로 어떤 관련이 있는지 알아보도록 하자.

트랜잭션은 하나의 작업 단위를 뜻한다.

예를들어 쇼핑몰의 "상품발송"이라는 트랜잭션을 가정해 보자. "상품발송" 이라는 트랜잭션에는 다음과 같은 작업들이 있을 수 있다.

* 포장
* 영수증 발행
* 발송

쇼핑몰의 운영자는 이 3가지 일들 중 하나라도 실패하면 3가지 모두 취소하고 "상품발송" 전의 상태로 되돌리고 싶을 것이다.

모두 취소하지 않으면 데이터의 정합성이 크게 흔들리게 된다. 이렇게 모두 취소하는 행위를 전문용어로 롤백(Rollback)이라고 말한다.

프로그램이 다음과 같이 작성되어 있다고 가정 해 보자. 아래는 실제 코드가 아니라 어떻게 동작하는지를 간략하게 표현한 수도(pseudo) 코드이다.

점프 투 자바**수도코드란?**

수도코드(슈도코드, pseudocode)는 특정 프로그래밍 언어의 문법을 따라 씌여진 것이 아니라, 일반적인 언어로 코드를 흉내내어 알고리즘을 써놓은 코드를 말한다. 수도코드는 말그대로 흉내만 내는 코드이기 때문에, 실제적인 프로그래밍 언어로 작성된 코드처럼 컴퓨터에서 실행할 수 없으며, 특정 언어로 프로그램을 작성하기 전에 알고리즘의 모델을 대략적으로 모델링하는 데에 쓰인다.

상품발송() {

포장();

영수증발행();

발송();

}

포장() {

...

}

영수증발행() {

...

}

발송() {

...

}

쇼핑몰 운영자는 포장, 영수증발행, 발송이라는 세가지 중 1가지라도 실패하면 모두 취소하고 싶어한다. 이런경우 어떻게 예외처리를 하는 것이 좋을까?

다음과 같이 포장, 영수증발행, 발송 메서드에서는 예외를 throw하고 상품발송 메서드에서 throw된 예외를 처리하여 모두 취소하는 것이 완벽한 트랜잭션 처리 방법이다.

상품발송() {

**try** {

포장();

영수증발행();

발송();

}**catch**(예외) {

모두취소(); // 하나라도 실패하면 모두 취소한다.

}

}

포장() **throws** 예외 {

...

}

영수증발행() **throws** 예외 {

...

}

발송() **throws** 예외 {

...

}

위와 같이 코드를 작성하면 포장, 영수증발행, 발송이라는 세개의 단위작업 중 하나라도 실패할 경우 "예외"가 발생되어 상품발송이 모두 취소 될 것이다.

그런데 다음처럼 "상품발송" 메서드가 아닌 포장, 영수증발행, 발송메서드에 각각 예외처리가 되어 있다고 가정 해 보자.

상품발송() {

포장();

영수증발행();

발송();

}

포장(){

**try** {

...

}**catch**(예외) {

포장취소();

}

}

영수증발행() {

**try** {

...

}**catch**(예외) {

영수증발행취소();

}

}

발송() {

**try** {

...

}**catch**(예외) {

발송취소();

}

}

이렇게 각각의 메서드에 예외가 처리되어 있다면 포장은 되었는데 발송은 안되고 포장도 안되었는데 발송이 되고 이런 뒤죽 박죽의 상황이 연출될 것이다. 실제 프로젝트에서도 두번째 경우처럼 트랜잭션관리를 잘못하여 고생하는 경우를 많이 보았는데 이것은 일종의 재앙에 가깝다.