

chool

Transaction (TX) 交易管理



段維瀚 老師



1



Session

- TX 交易/事務管理
- TX 交易註解配置
- TX 交易管理實作演練





TX 交易/事務管理

- 在 **JavaEE** 企業級開發應用領域中為了保持資料表數據的完整性和一致性，就必須引入資料庫事務的概念。
- 何謂事務？
 - 就是一組由邏輯上緊密聯合而成的一個交易整體的多個資料表數據操作
 - 若當中有某一項執行失敗，就整體失敗，那之前成功的項目就會進行回滾(**Rollback**)





TX 交易/事務管理

- jdbc-config.xml

```
<!-- 配置事務管理器 -->
<bean id="transactionManager"
      class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
  <property name="dataSource" ref="dataSource" ></property>
</bean>
<!-- 開啟註解驅動，對事務相關的註解進行掃描與解析 -->
<tx:annotation-driven transaction-manager="transactionManager"></tx:annotation-driven>
```





交易註解配置

- **@Transaction**

- 配置在類別（及於該類別下所有方法）
- 配置在方法（只有當前方法可用）





交易的屬性

```
@Transactional(  
    propagation = Propagation.REQUIRES_NEW,  
    isolation = Isolation.READ_COMMITTED,  
    timeout = 3,  
    readOnly = false,  
    rollbackFor = {NullPointerException.class, ArithmeticException.class})
```

傳播行為 (Propagation behavior)

隔離層級 (Isolation level)

交易超時期間 (The transaction timeout period)

唯讀提示 (Read-only hints)

根據指定發生事件回滾或不回滾





交易的屬性

```
@Transactional(  
    propagation = Propagation.REQUIRES_NEW,  
    isolation = Isolation.READ_COMMITTED,  
    timeout = 3,  
    readOnly = false,  
    rollbackFor = {NullPointerException.class, ArithmeticException.class})
```

propagation (傳播級別)

- * Propagation.REQUIRED : 必須使用調用者的事物
- * Propagation.REQUIRES_NEW : 不使用調用者的事務，而使用新的事務進行

REQUIRED：支持當前事務，如果當前沒有事務，就新建一個事務。這是最常見的選擇。

SUPPORTS：支持當前事務，如果當前沒有事務，就以非事務方式執行。

MANDATORY：支持當前事務，如果當前沒有事務，就拋出異常。

REQUIRES_NEW：新建事務，如果當前存在事務，把當前事務掛起。

NOT_SUPPORTED：以非事務方式執行操作，如果當前存在事務，就把當前事務掛起。

NEVER：以非事務方式執行，如果當前存在事務，則拋出異常。

NESTED：支持當前事務，如果當前事務存在，則執行一個嵌套事務，如果當前沒有事務，就新建一個事務。





交易的屬性

```
@Transactional(  
    propagation = Propagation.REQUIRES_NEW,  
    isolation = Isolation.READ_COMMITTED,  
    timeout = 3,  
    readOnly = false,  
    rollbackFor = {NullPointerException.class, ArithmeticException.class})
```

isolation (隔離級別)

- * **Isolation.DEFAULT** 使用底層資料庫預設的隔離層級
- * **Isolation.READ_COMMITTED** 允許交易讀取其它並行的交易已經送出 (Commit) 的資料欄位，可以防止髒讀(Dirty read)問題
- * **Isolation.READ_UNCOMMITTED** 允許交易讀取其它並行的交易還沒送出的資料，會發生髒讀Dirty、不可重複Nonrepeatable、幻讀Phantom read等問題
- * **Isolation.REPEATABLE_READ** 要求多次讀取的資料必須相同，除非交易本身更新資料，可防止Dirty、Nonrepeatable read問題
- * **Isolation.SERIALIZABLE** 隔離級別的最高級(完整的隔離層級)，可防止Dirty、Nonrepeatable、Phantom read等問題，會鎖定對應的資料表格，因而有效能問題



隔離層級

- 沒有對資料庫進行鎖定下，可能發生的問題
 - 更新遺失 (**lost update**)
 - 髒讀 (**dirty read**)
 - 無法重複的讀取 (**unrepeatable read**)
 - 幻讀 (**phantom read**)





隔離層級

- 更新遺失 (**lost update**)
 - 基本上就是指某個交易對欄位進行更新的資訊，因另一個交易的介入而遺失。
 - 交易 A 更新欄位1
交易 B 更新欄位1
交易 A COMMIT
交易 B ROLLBACK
 - 交易 A 更新欄位1
交易 B 更新欄位1
交易 B COMMIT
交易 A COMMIT

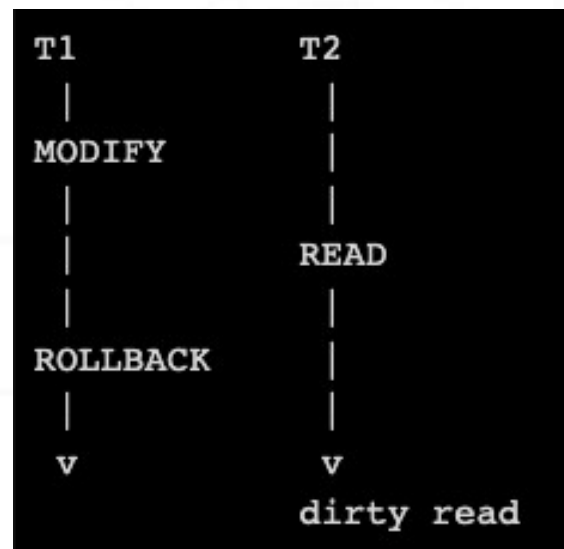


隔離層級

- 髒讀 (dirty read)

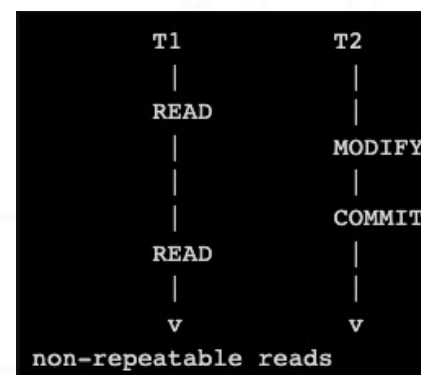
- 兩個交易同時進行，其中一個交易更新資料，另一個交易讀取了尚未**COMMIT**的資料，就有可能發生髒讀問題。

- 交易 A 更新欄位1
- 交易 B 讀取欄位1
- 交易 A **ROLLBACK**
- 交易 B **COMMIT**



隔離層級

- 無法重複的讀取 (**unrepeatable read**)
 - 某個交易兩次讀取同一欄位的資料並不一致，例如，如果交易**A**在交易**B**前後進行資料的讀取，則會得到不同的結果。
 - 交易 **A** 讀取欄位**1**
 - 交易 **B** 更新欄位**1**
 - 交易 **B** **COMMIT**
 - 交易 **A** 讀取欄位**1**



資料被 B 改變所以連續二次讀取不一致





隔離層級

- 幻讀 (phantom read)

- 如果交易A進行兩次查詢，在兩次查詢之中有個交易B插入一筆新資料或删除一筆新資料，第二次查詢時得到的資料多了第一次查詢時所沒有的筆數，或者少了一筆。

- 交易 A 進行查詢資料有N筆並進行修改

- 交易 B 插入 1 筆資料

- 交易 B COMMIT

- 交易 A 再次進行查詢檢查時卻發現有1筆竟沒修改



隔離層級參數配置說明(表 1)

```
@Transactional(  
    propagation = Propagation.REQUIRES_NEW,  
    isolation = Isolation.READ_COMMITTED,  
    timeout = 3,  
    readOnly = false,  
    rollbackFor = {NullPointerException.class, ArithmeticException.class})
```

隔離層級參數	髒讀	不可重複讀	幻讀
	Dirty	Nonrepeatable	Phantom
READ_UNCOMMITTED	○	○	○
READ_COMMITTED	X	○	○
REPEATABLE_READ	X	X	○
SERIALIZABLE	X	X	X

隔離層級參數配置說明(表 2)

隔離級別參數	說明
Isolation.DEFAULT	使用底層資料庫預設的隔離層級
Isolation.READ_UNCOMMITTED	允許交易讀取其它並行的交易還沒送出的資料， 會發生髒讀(Dirty)、不可重複讀 (Nonrepeatable)、幻讀(Phantom)等問題
Isolation.READ_COMMITTED	允許交易讀取其它並行的交易已經送出 (Commit)的資料欄位，可防止髒讀(Dirty) 問題
Isolation.REPEATABLE_READ	要求多次讀取的資料必須相同，除非交易本身 更新資料，可防止髒讀(Dirty)、不可重複讀 (Nonrepeatable)問題
Isolation.SERIALIZABLE	隔離級別的最高級(完整的隔離層級)，可防止 髒讀(Dirty)、不可重複讀(Nonrepeatable) 問題、幻讀(Phantom)等問題，會鎖定對應的 資料表格，因而有效能問題



交易的屬性

```
@Transactional(  
    propagation = Propagation.REQUIRES_NEW,  
    isolation = Isolation.READ_COMMITTED,  
    timeout = 3,  
    readOnly = false,  
    rollbackFor = {NullPointerException.class, ArithmeticException.class})
```

timeout (超時)

* 對資料庫若超過 **timeout** 所設定的時間就會進行回滾 (Roll back) , 以秒為單位





交易的屬性

```
@Transactional(  
    propagation = Propagation.REQUIRES_NEW,  
    isolation = Isolation.READ_COMMITTED,  
    timeout = 3,  
    readOnly = false,  
    rollbackFor = {NullPointerException.class, ArithmeticException.class})
```

readOnly

* readOnly = true 資料庫已不鎖定的方式進行存取增加效能，用於查詢方案





交易的屬性

```
@Transactional(  
    propagation = Propagation.REQUIRES_NEW,  
    isolation = Isolation.READ_COMMITTED,  
    timeout = 3,  
    readOnly = false,  
    rollbackFor = {NullPointerException.class, ArithmeticException.class})
```

rollbackFor | rollbackForClassName | noRollbackFor | noRollbackForClassName
* 根據指定發生事件回滾或不回滾


Spring框架的事務基礎架構程式碼將預設地 只在丟擲執行時其例外和unchecked exceptions時才標識事務回滾。也就是說，當丟擲個RuntimeException 或其子類例的例項時就會進行回滾 (Errors 也一樣)。從事務方法中丟擲的Checked exceptions 將不進行事務回滾。

- 1 讓 checked exception 例外也回滾：在整個方法前加上 @Transactional(rollbackFor=Exception.class)
- 2 讓 unchecked exception 例外不回滾： @Transactional(notRollbackFor=RunTimeException.class)
- 3 不需要事務管理的(只查詢的)方法：@Transactional(propagation=Propagation.NOT_SUPPORTED)

注意：如果異常被try { } catch { } 包住了，事務就不回滾了，如果想讓事務回滾必須再往外拋 (throw Exception)。




TX Lab (資料庫表單)



BOOK
BID
BNAME
PRICE
CT

書籍資料



STOCK
SID
BID
AMOUNT

庫存資料



WALLET
WID
MONEY

我的錢包





TX Lab (SQL 語法)

```
CREATE TABLE Book (  
    bid INTEGER NOT NULL GENERATED ALWAYS AS IDENTITY, -- 主鍵序號  
    bname varchar(20) not null, -- 書名  
    price INTEGER, -- 價格  
    ct timestamp default current_timestamp, -- 建檔時間  
    PRIMARY KEY (bid)  
);  
  
CREATE TABLE Stock (  
    sid INTEGER NOT NULL GENERATED ALWAYS AS IDENTITY, -- 主鍵序號  
    bid INTEGER NOT NULL, -- Book 主鍵  
    amount INTEGER, -- 數量  
    PRIMARY KEY (sid)  
);  
  
CREATE TABLE Wallet (  
    wid INTEGER NOT NULL GENERATED ALWAYS AS IDENTITY, -- 主鍵序號  
    money INTEGER, -- 數量  
    PRIMARY KEY (wid)  
);
```



配置事務通知

- **jdbc-config.xml**

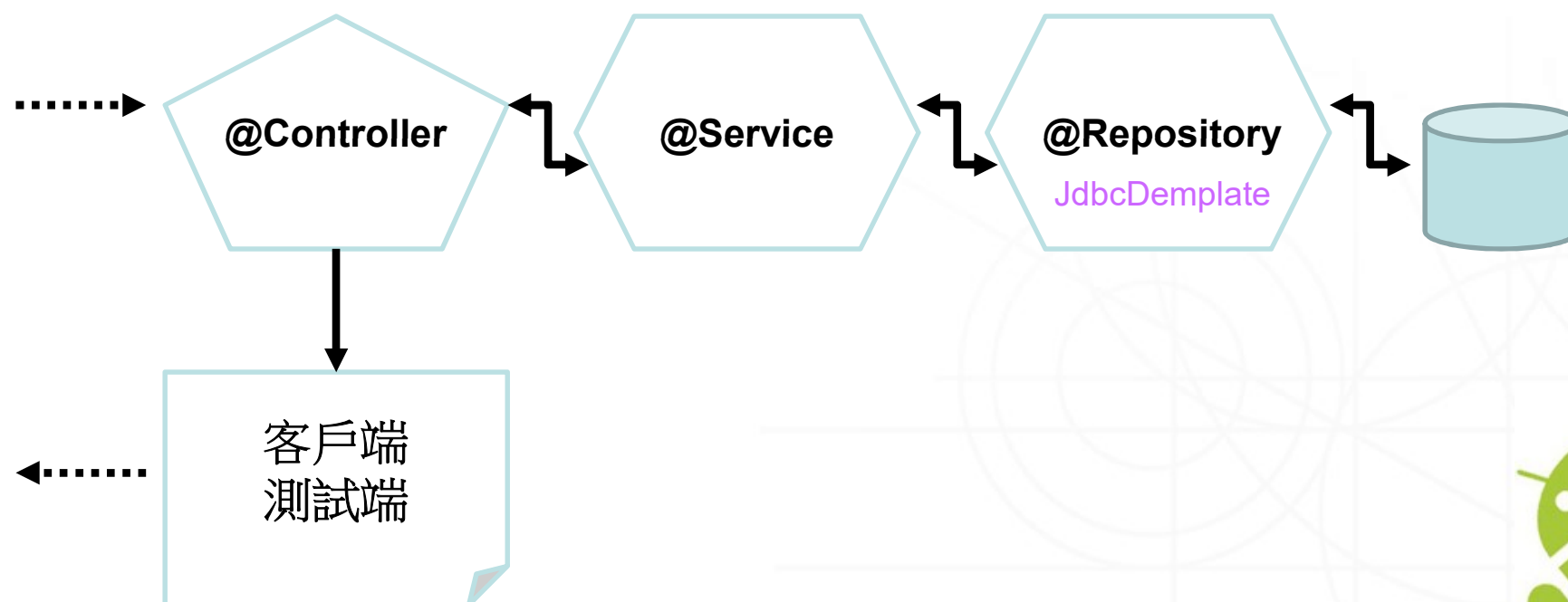
一定要加入

```
<!-- 配置事務管理器 -->
<bean id="transactionManager"
      class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
  <property name="dataSource" ref="dataSource" ></property>
</bean>
```



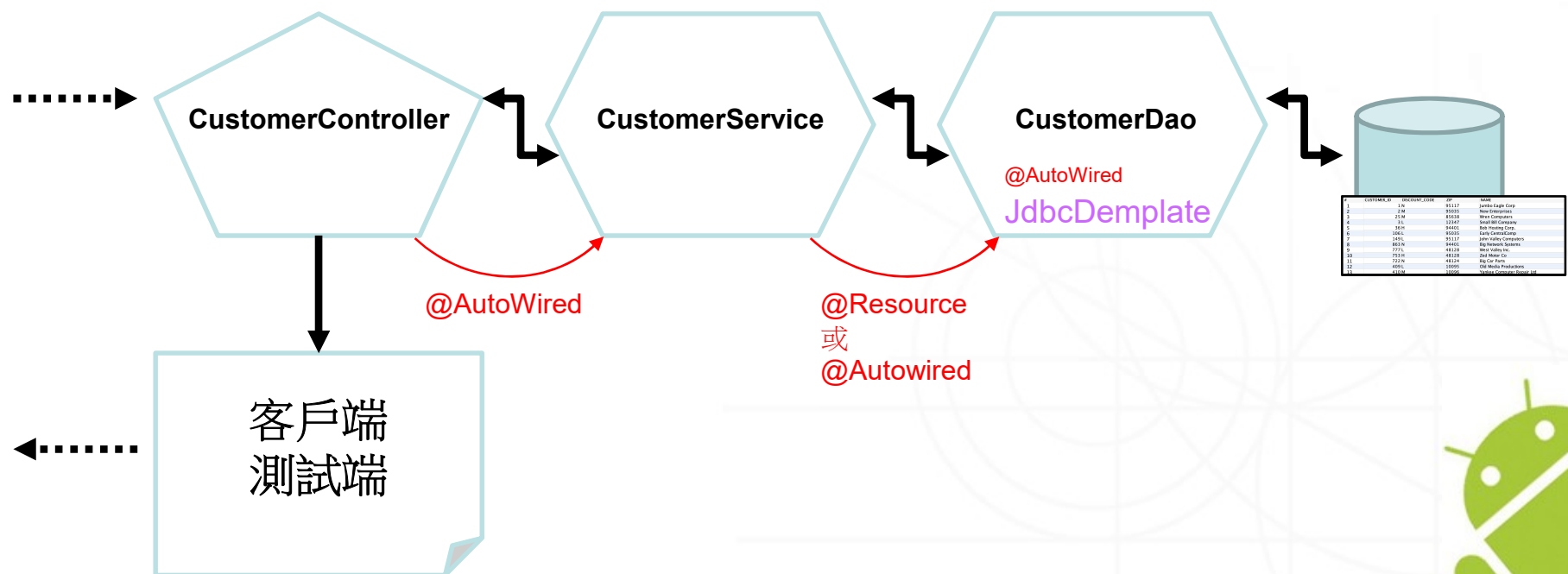
Spring 資料庫存取

- 標準架構流程



Spring 資料庫存取

- 標準架構流程



Lab 1 (關聯關係)

INVOICE
ID
INVDATE
Indexes
Foreign Keys
ITEM
ID
AMOUNT
IPID
INVID
Indexes
Foreign Keys
ITEMPRODUCT
ID
TEXT
PRICE
INVENTORY
Indexes
Foreign Keys

發票

#	ID	INVDATE
1		1 2020-11-23
2		2 2020-11-22
3		3 2020-11-21

發票項目 (購買商品)

#	ID	AMOUNT	IPID	INVID
1		1	5	1
2		2	3	1
3		3	4	1
4		4	1	2
5		5	6	2

商品

#	ID	TEXT	PRICE	INVENTORY
1		1 Pen	10	20
2		2 Book	15	50
3		3 Toy	20	40

Lab 1 - SQL

```
CREATE TABLE ItemProduct ( -- 商品項目
    id INTEGER NOT NULL GENERATED ALWAYS AS IDENTITY, -- 商品項目序號(主鍵)
    text VARCHAR(50) not null, -- 商品項目名稱
    price INTEGER NOT NULL, -- 商品單價
    inventory INTEGER NOT NULL, -- 商品庫存量
    PRIMARY KEY (id)
);

CREATE TABLE Invoice ( -- 發票
    id INTEGER NOT NULL GENERATED ALWAYS AS IDENTITY, -- 發票序號(主鍵)
    invdate Date not null, -- 發票日期
    PRIMARY KEY (id)
);

CREATE TABLE Item ( -- 發票項目
    id INTEGER NOT NULL GENERATED ALWAYS AS IDENTITY, -- 項目序號(主鍵)
    amount INTEGER NOT NULL, -- 數量
    ipid INTEGER NOT NULL, -- 商品項目序號
    invid INTEGER NOT NULL, -- 發票序號
    PRIMARY KEY (id),
    FOREIGN KEY (ipid) REFERENCES ItemProduct(id),
    FOREIGN KEY (invid) REFERENCES Invoice(id)
);
```



Lab 1 - 分析項目

- 每一張發票有那些商品?
- 每一張發票有幾件商品?
- 每一張發票價值多少?
- 每一樣商品各賣了多少?
- 哪一件商品賣得錢最多?
- 哪一張發票價值最高（請練習看看）?



Lab 2 (一對一, 一對多)

廣告內文

ADVERT
ID
TEXT
AID

#	ID	TEXT	AID
1	1	aaa	1
2	2	bbb	2
3	3	ccc	2
4	4	ddd	1
5	5	eee	2

廣告帳戶

ACCOUNT
ID
ANAME

#	ID	ANAME
1	1	A
2	2	B

廣告預算

BUDGET
ID
CASH
AID

#	ID	CASH	AID
1	1	100	1
2	2	200	2

*

1

1

1



Lab 2 - SQL

```
CREATE TABLE Account (  
    id INTEGER NOT NULL GENERATED ALWAYS AS IDENTITY, -- 主鍵序號  
    aname varchar(20) not null, -- 帳戶名稱  
    PRIMARY KEY (id)  
);  
  
CREATE TABLE Advert (  
    id INTEGER NOT NULL GENERATED ALWAYS AS IDENTITY, -- 主鍵序號  
    text VARCHAR(50) not null, -- 廣告文案  
    aid INTEGER NOT NULL,  
    PRIMARY KEY (id),  
    FOREIGN KEY (aid) REFERENCES Account(id)  
);  
  
CREATE TABLE Budget (  
    id INTEGER NOT NULL GENERATED ALWAYS AS IDENTITY, -- 主鍵序號  
    cash INTEGER NOT NULL, -- 預算  
    aid INTEGER NOT NULL,  
    PRIMARY KEY (id),  
    FOREIGN KEY (aid) REFERENCES Account(id)  
);
```