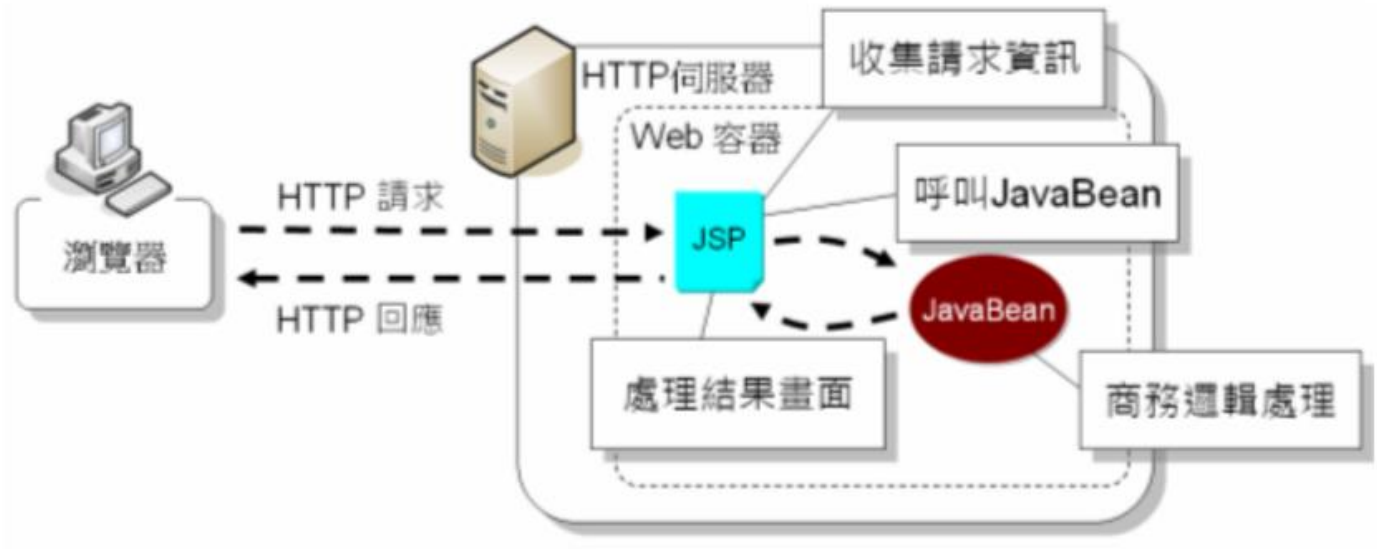
什麼是 eBAF？

- 全名為 e-Business Application Framework
- IBM 提出的應用程式框架
- 國泰人壽主要的系統流程架構
- 使用 MVC 架構

MVC

Model 1

- JSP + JavaBean Model



優點

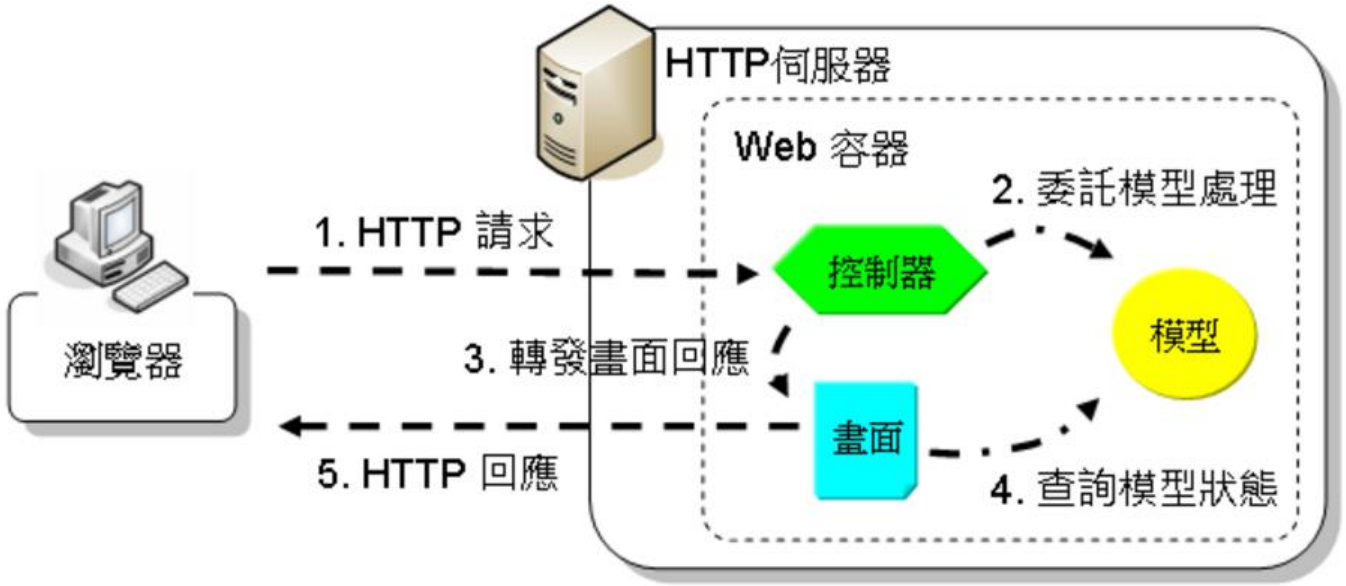
- 在中小型應用程式上開發快速：可以減少請求轉發的流程設計與角色區隔

缺點

- JSP 頁面負責收集請求參數與呼叫 JavaBean，增加維護 JSP 的負擔
- 如果 JSP 夾雜 HTML 與 Java，會不利 Java 程式設計人員與網頁設計人員的分工

Model 2

- JSP + Servlet + JavaBean Model



三層式架構

- 模型（Model）
  - 用於封裝與應用程式的業務邏輯相關的資料以及對資料的處理方法
  - 有對資料直接存取的權力
  - 不關心它會被如何顯示或是如何被操作
- 畫面（View）
  - 能夠實現資料有目的的顯示
  - 一般沒有程式上的邏輯
- 控制器（Controller）
  - 控制應用程式的流程、處理事件並回應

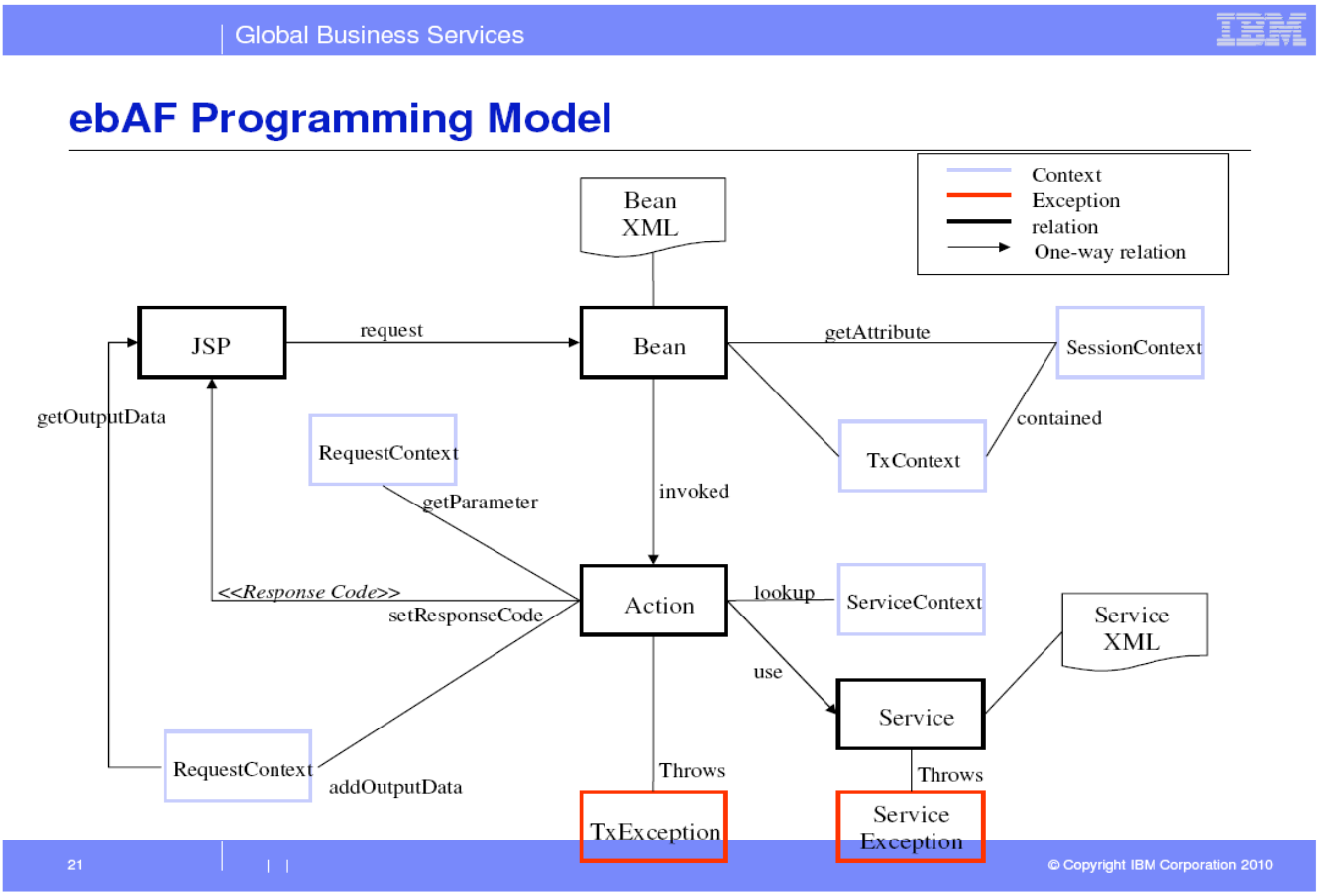
優點

- 業務邏輯與顯示分離
- 維護容易：業務邏輯可以重複使用
- 可以在 Controller 輕鬆調整流程

缺點

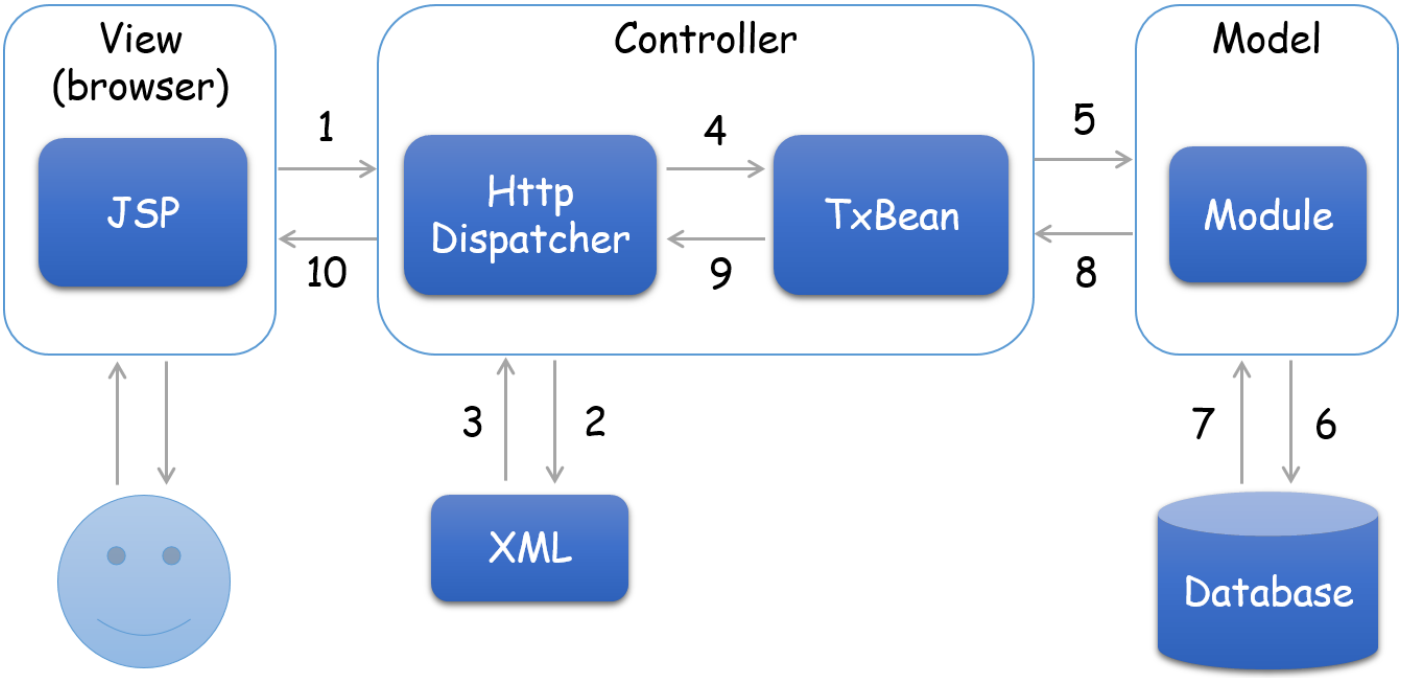
- 開發速度較慢：需要花時間將業務邏輯與流程拆成 M、V、C 三個部份

eBAF 的 MVC



eBAF 流程

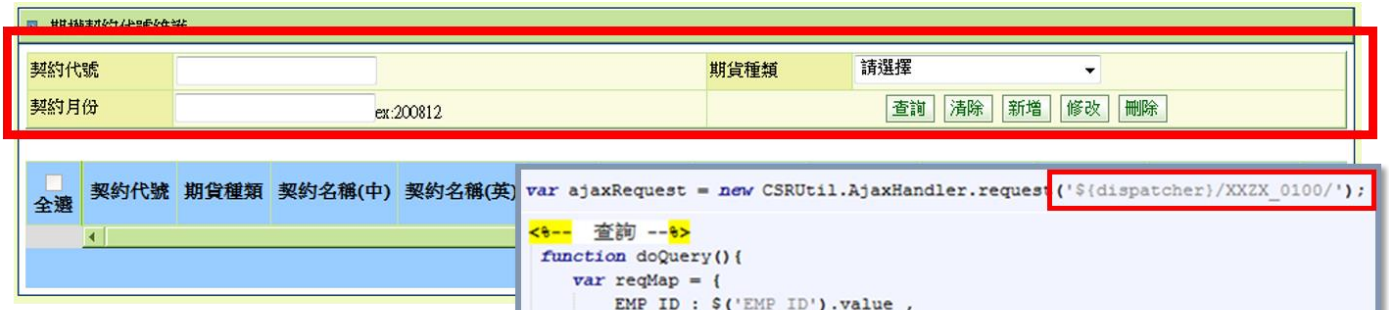
流程圖



1. 瀏覽器向伺服器發出請求
2. httpDispatcher 解析 URL 後，尋找對應的 xml
3. 讀取找到的 xml 內容
4. 產生對應的類別物件
5. 依照流程呼叫模組
6. 模組依照功能查詢或異動資料庫（如果有需要的話）
7. 資料庫回傳結果給模組
8. 模組回傳結果給主程式
9. 主程式將結果回傳 httpDispatcher
10. httpDispatcher 依照 xml 設定，回傳對應的 jsp 給瀏覽器

實際案例

View




```
var ajaxRequest = new CSRUtil.AjaxHandler.request('${dispatcher}/XXZX_0100/');
<%-- 查詢 --%>
function doQuery(){
    var reqMap = {
        EMP_ID : $('EMP_ID').value ,
        DIV_NO : $('DIV_NO').value ,
    };

    <%-- 發送AJAX --%>
    ajaxRequest.post('query', { 'reqMap' : Object.toJSON(reqMap)},
        function(resp){
            var rtnList  = resp.rtnList;
            grid.load(rtnList);
        },
        function(resp){
            grid.load([]);
        }
    );
}
```

負責畫面呈現
接收使用者輸入及操作

JSP

Controller

對應到的主程式

```
<?xml version="1.0" encoding="UTF-8"?>
<txbean class="com.cathay.xx.zx.trx.XXZX_0100" name="XXZX_0100">
    <action-list>
        <action call-method="doPrompt" name="prompt">
            <response-list>
                <response name="success" type="standard">
                    <property name="url">/XX/ZX/XXZX_0100/XXZX0100.jsp
                    </property>
                </response>
            </response-list>
        </action>
        <action call-method="doQuery" name="query">
            <response-list>
                <response name="success" type="standard">
                    <property name="url">/CM/js/ajax/dummy.jsp
                    </property>
                </response>
            </response-list>
        </action>
    </action-list>
</txbean>
```

初始頁面回傳路徑

對應到主程式的method

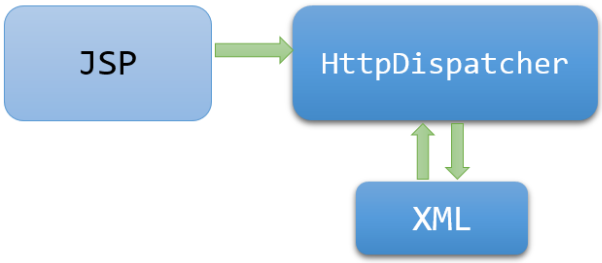
非初始頁面回傳路徑

回應代碼

主程式片段

```
resp.setResponseCode("success");
```

```
public ResponseContext doQuery
            (RequestContext req) {

    return resp;
}
```



HttpDispatcher負責傳遞參數
並會參照對應的xml設定檔

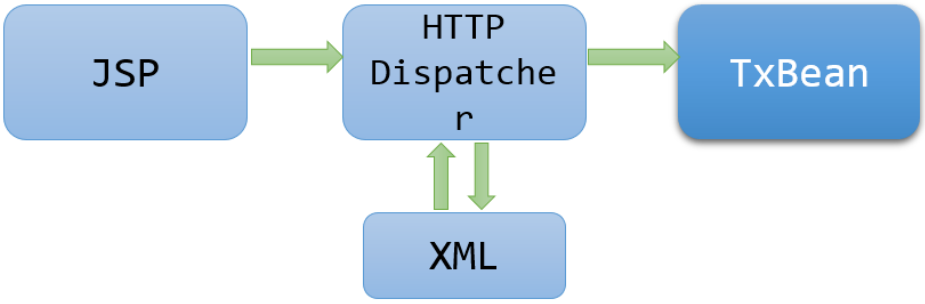
XML對應到的method

```
public ResponseContext doQuery(RequestContext req) {
    try {

        Map reqMap = VOTool.jsonToMap(req.getParameter("reqMap"));

        List<Map> rtnList = new XX_ZX0100().query(MapUtils.getString(reqMap, "EMP_ID"));

        resp.addOutputData("rtnList", rtnList);

        MessageHelper.setReturnMessage(msg, ReturnCode.OK, "查詢完成");
```

呼叫模組

將回傳資料加到resp的空間

接收瀏覽器傳送的資料
負責流程管理，並呼叫模組
交易控制要寫在主程式



Model

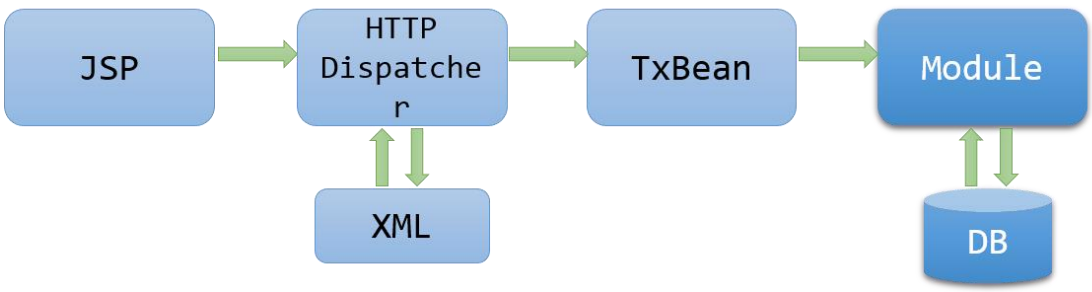
主程式所呼叫的method

```
public List<Map> query(Map reqMap) throws ModuleException {

    if(reqMap == null || reqMap.isEmpty()){
        throw new ErrorInputException ("傳入資料不得為空");
    }

    String EMP_ID = MapUtils.getString(reqMap, "EMP_ID");

    DataSet ds = Transaction.getDataSet();

    ds.setField("EMP_ID", EMP_ID);

    List<Map> rtnList = VOTool.findToMaps(ds, SQL_query_001);

    return rtnList;
}
```

回傳資訊

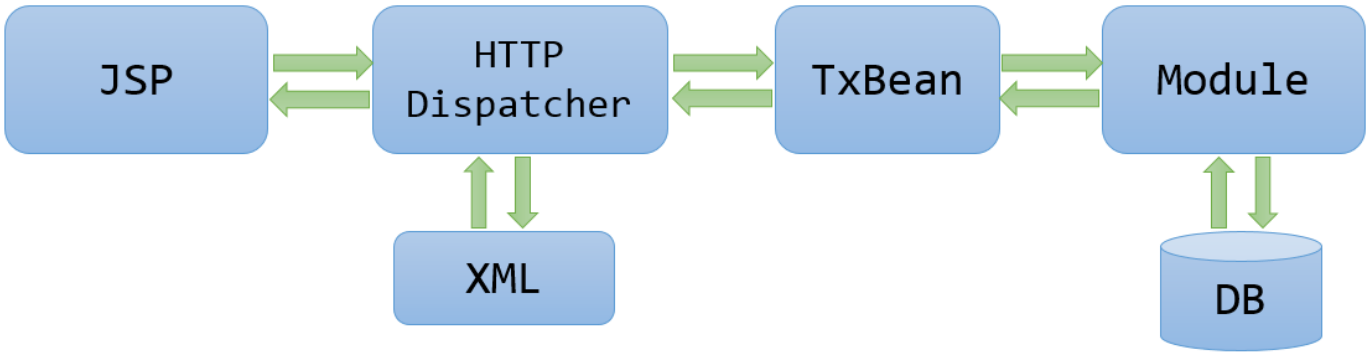
接收主程式或其他模組的請求
負責資料處理、邏輯運算、檢核

使用SQL進行DB的查詢、新增、修改、刪除



View (資料呈現)

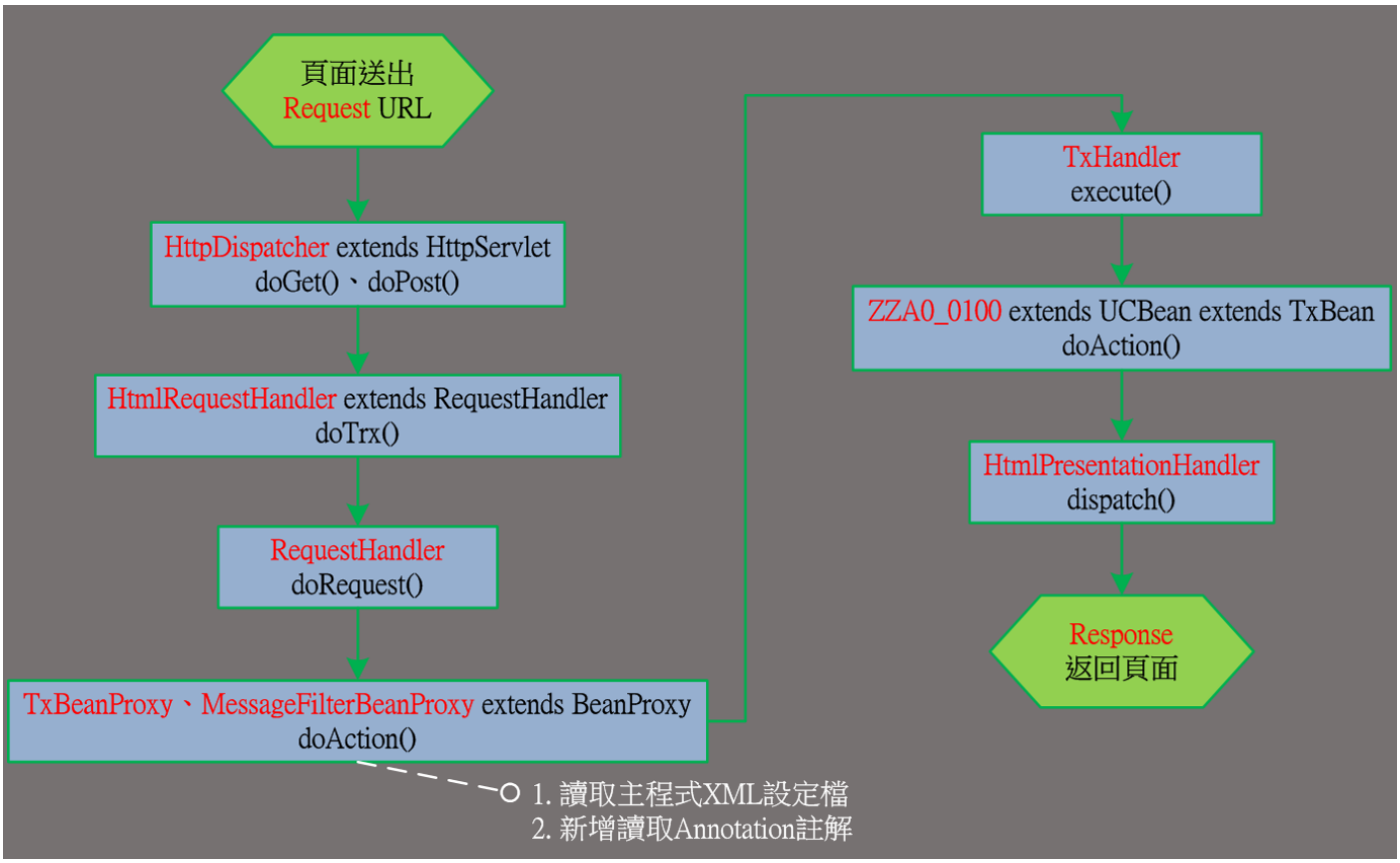

期權契約代號維護
契約代號
期貨種類
請選擇
契約月份
ex:200812
查詢
清除
新增
修改
刪除
全選
契約代號
期貨種類
契約名稱(中)
契約名稱(英)
買賣權
契約月份
最後交易日
最後結算日
實物交割日
現金交割日
權益數調整數額
調整稿效日
TF-201411 TF 金指201411 201411 2014-11-19 2014-11-19 2014-11-19 2014-11-19
TF-201412 TF 金指201412 201412 2014-12-17 2014-12-17 2014-12-17 2014-12-17
TF-201501 TF 金指201501 201501 2015-01-21 2015-01-21 2015-01-21 2015-01-21
TF-201502 TF 金指201502 201502 2015-02-24 2015-02-24 2015-02-24 2015-02-24
TF-201503 TF 金指201503 201503 2015-03-18 2015-03-18 2015-03-18 2015-03-18
TF-201504 TF 金指201504 201504 2015-04-15 2015-04-15 2015-04-15 2015-04-15
TF-201505 TF 金指201505 201505 2015-05-20 2015-05-20 2015-05-20 2015-05-20
TF-201506 TF 金指201506 201506 2015-06-17 2015-06-17 2015-06-17 2015-06-17
TF-201507 TF 金指201507 201507 2015-07-15 2015-07-15 2015-07-15 2015-07-15
TF-201508 TF 金指201508 201508 2015-08-19 2015-08-19 2015-08-19 2015-08-19
- 10 -
第 10 頁/共 37 頁 共 362 筆


JSP
HTTP Dispatcher
TxBean
Module
XML
DB

Annotation

- 取代 xml
- 維護方便：因為 xml 的內容改寫在 TxBean 裡面，少一份檔案
- Server 啟動的時候會先掃過所有檔案，並將相關資訊 Cache 起來

流程圖



程式寫法

將此類別標誌為主程式

```
@TxBean
public class RYP1_0100 extends UCBean {
   // 以下略
}
```

宣告此方法為主程式方法

```
@CallMethod(action = "prompt", url =
        "/RY/P1/RYP1_0100/RYP10100.jsp")
public ResponseContext
doPrompt(RequestContext req) {
   return resp;
}
```

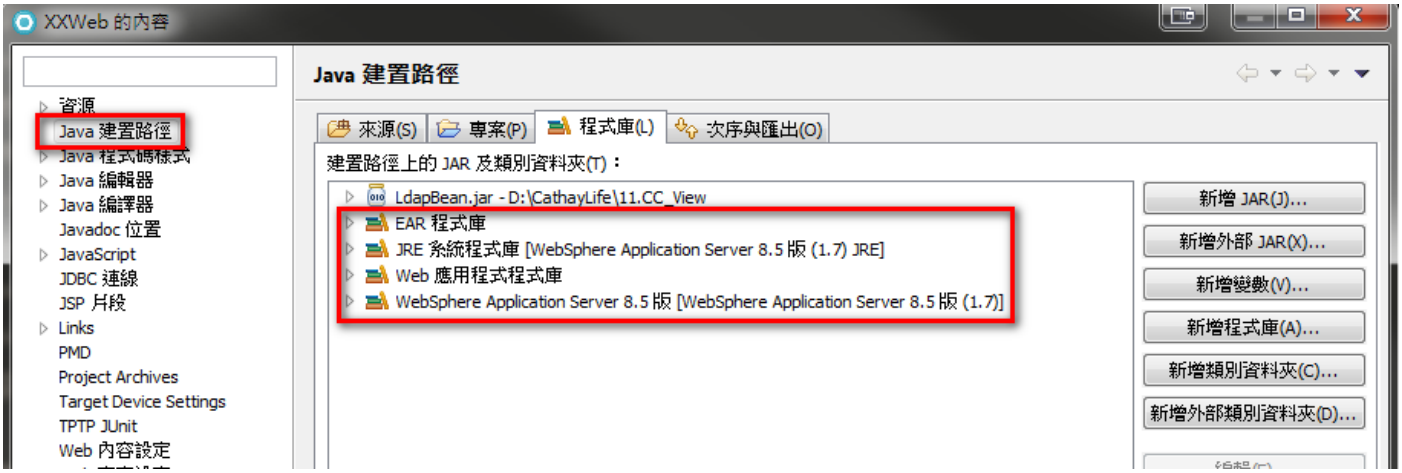
- action(必傳)：對應 JSP 中的 action name
- name(可選)：對應 ResponseCode 的設定。預設值為："success"
- type(可選)：例如上傳檔案時為 n/a。預設值為："standard"
- url(可選)：對應 response 的導頁設定。預設值為："dummy.jsp"

CallMethod 內建常數

- CallMethod.CODE_SUCCESS："success"
- CallMethod.CODE_RESULT："result"
- CallMethod.TYPE_SUBMIT："standard"
- CallMethod.TYPE_AJAX："ajax"
- CallMethod.TYPE_REDIRECT："redirect"
- CallMethod.TYPE_NA："n/a"
- CallMethod.URL_DUMMY："/CM/js/ajax/dummy.jsp"

專案建置

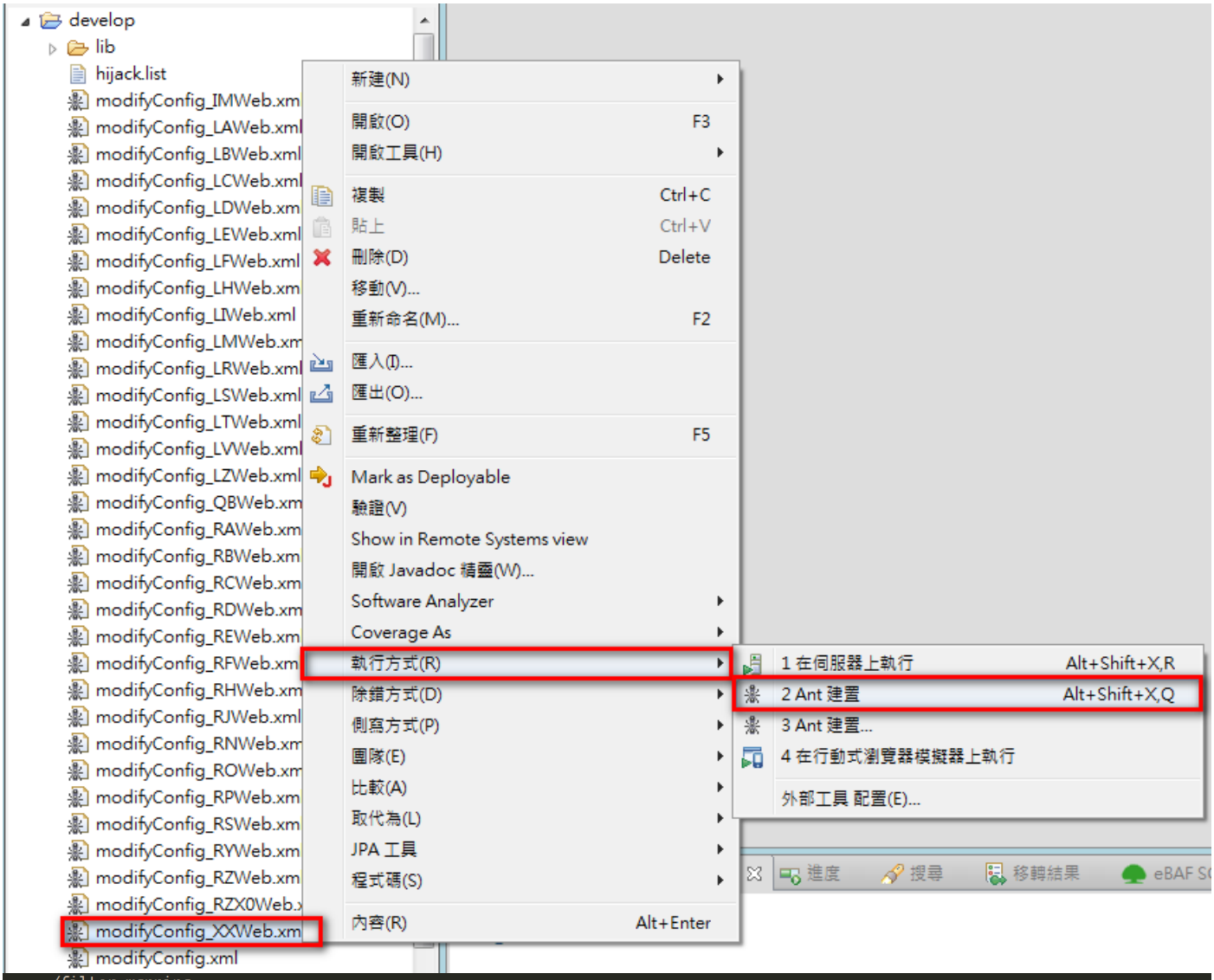
檢查建置路徑是否有錯



匯入專案、Ant 建置

```
26      </filter-mapping>
27      <filter-mapping>
28          <filter-name>AjaxFilter</filter-name>
29          <servlet-name>HttpDispatcher</servlet-name>
30          <dispatcher>FORWARD</dispatcher>
31          <dispatcher>INCLUDE</dispatcher>
32          <dispatcher>REQUEST</dispatcher>
33          <dispatcher>ERROR</dispatcher>
34      </filter-mapping>
35      <servlet>
36          <servlet-name>HttpDispatcher</servlet-name>
37          <servlet-class>com.cathay.common.log4j.Log4jAwareDispatcher</servlet-class>
38          <init-param>
39              <param-name>configFile</param-name>
40              <param-value>@PATH/XX_SRC/usr/cxlcs/config/init/config.properties</param-value>
41          </init-param>
42          <load-on-startup>3</load-on-startup>
43      </servlet>
```

- 使用 Ant 將@PATH 替換成本機 CC View 的路徑



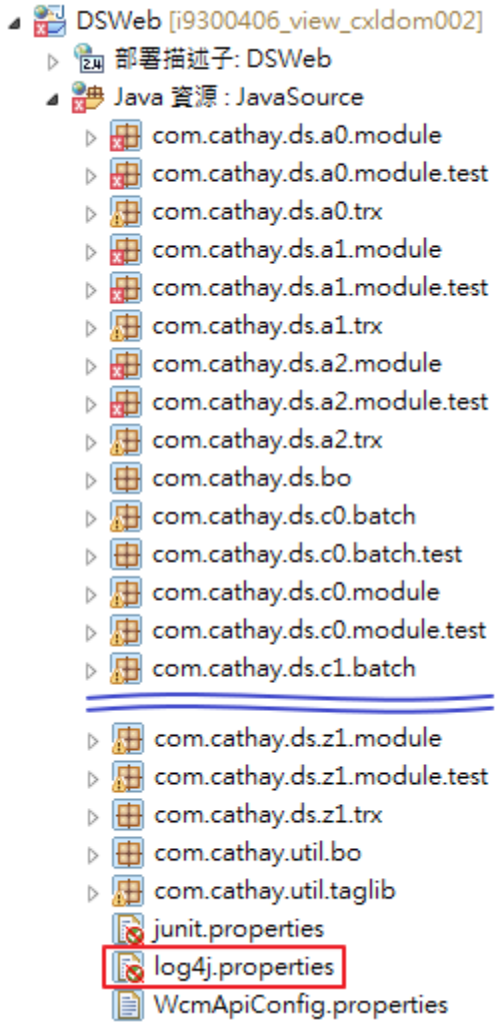

```
    </filter-mapping>
    <filter-mapping>
        <filter-name>AjaxFilter</filter-name>
        <servlet-name>HttpDispatcher</servlet-name>
        <dispatcher>FORWARD</dispatcher>
        <dispatcher>INCLUDE</dispatcher>
        <dispatcher>REQUEST</dispatcher>
        <dispatcher>ERROR</dispatcher>
    </filter-mapping>
    <servlet>
        <servlet-name>HttpDispatcher</servlet-name>
        <servlet-class>com.cathay.common.log4j.Log4jAwareDispatcher</servlet-class>
        <init-param>
            <param-name>configFile</param-name>
            <param-value>D:/CathayLife/11.CC_View/View_IM/IM/source/XX_SRC/usr/cxlcs/config/init/config.properties</param-value>
        </init-param>
        <load-on-startup>3</load-on-startup>
    </servlet>
```

設定檔

- 將設定放在設定檔裡面，並在 web.xml 裡面指定設定檔的根目錄，絕對路徑的前半段是用 @PATH 取代，因為這段路徑會因為每個人的專案位置而有所不同。
- Server 啟動的時候會讀取設定檔，並保存資料，此時更新設定檔並不會生效

```
<servlet>
  <servlet-name>HttpDispatcher</servlet-name>
  <servlet-class>com.cathay.common.log4j.Log4jAwareDispatcher</servlet-class>
  <init-param>
    <param-name>configFile</param-name>
    <param-value>@PATH/IM/source/XX_SRC/usr/cxlcs/config/init/config.properties</param-value>
  </init-param>
  <load-on-startup>3</load-on-startup>
</servlet>
```

log4j.properties




```
log4j.rootLogger=DEBUG, console

log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.layout=org.apache.log4j.PatternLayout
log4j.appender.console.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss,SSS} [%-5p][%C{1}.%M()(%L)] - %m%n

```

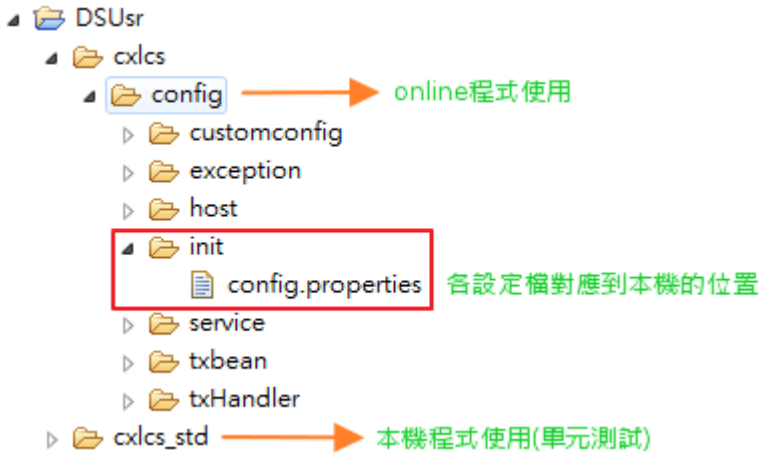
junit.properties

```
# init config for ebaf standalone env
ebaf.config=D:\\CathayLife\\11.CC_View\\View_IM\\IM\\source\\XX_SRC\\usr\\cxlcs_std\\config\\init\\config.properties
```

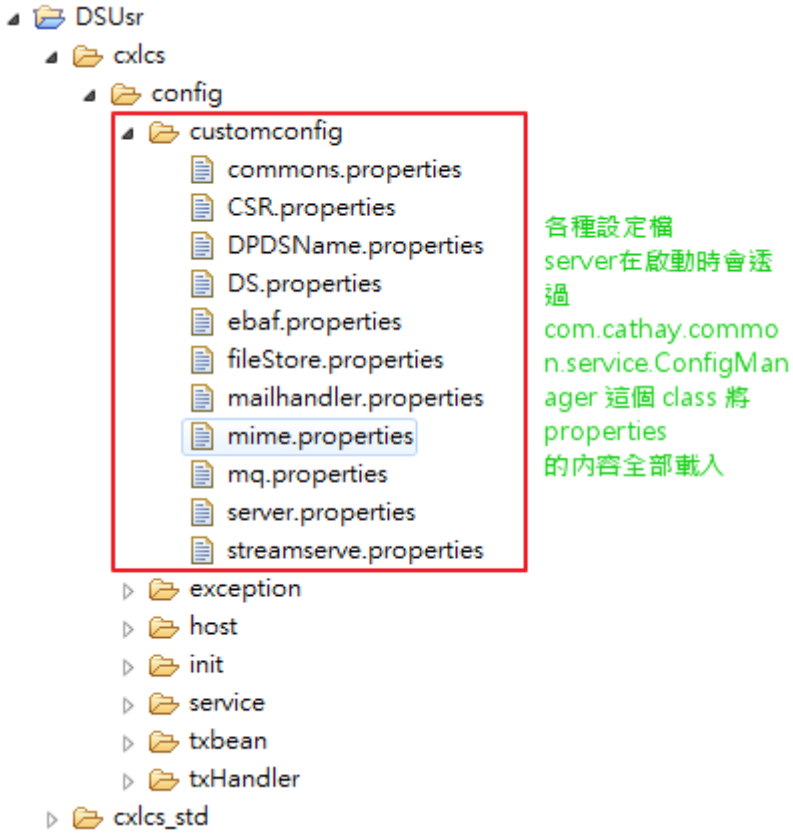
MessageDAO.properties

- 記錄多國語系資料庫的連線資訊，MessageUtil 會用到

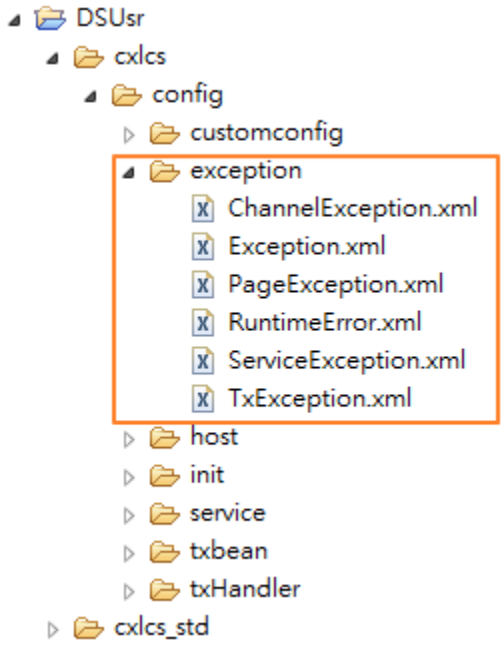
user 資料夾



customconfig 資料夾

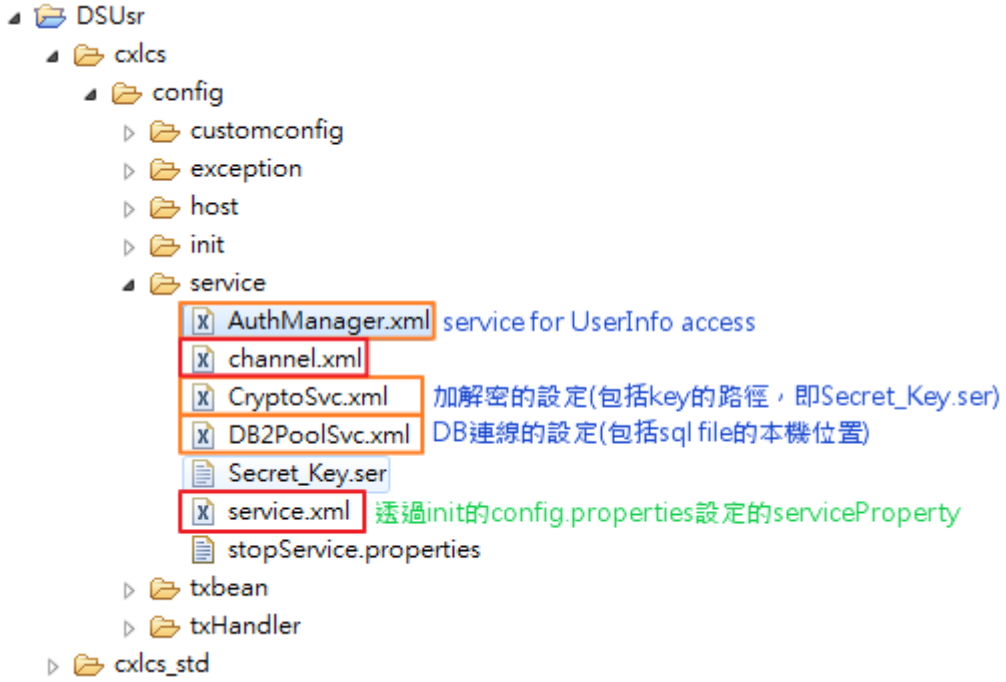


exception 資料夾



當發生錯誤時會依照不同的Exception，讀取不同的xml，然後依照xml裡面的設定導向錯誤畫面(目前的Exception種類是ebaf架構一開始定義好的，如有需要擴充，底層也需要異動)

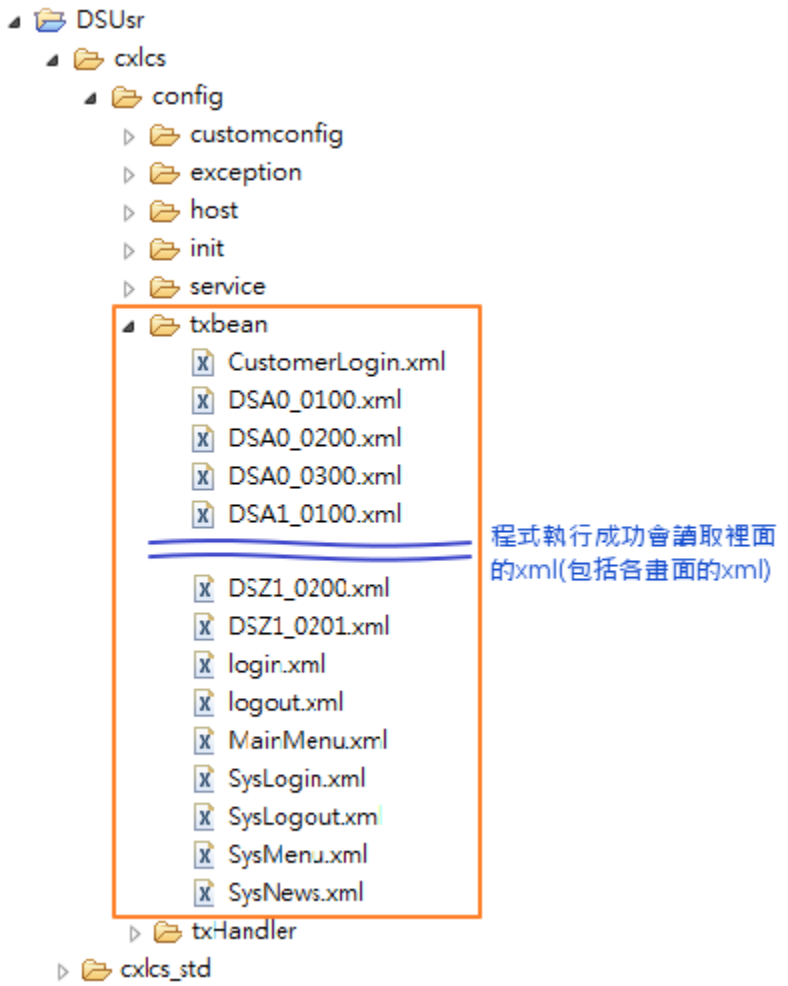
service 資料夾



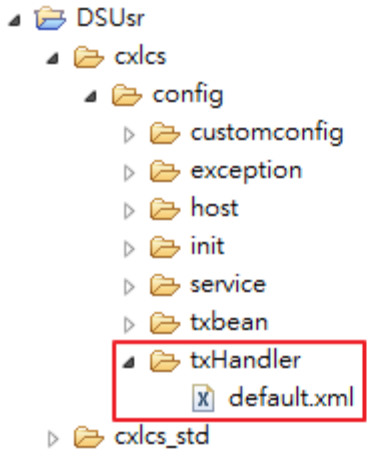
DB2PoolSvc.xml

```xml
<?xml version="1.0"?>
<Service name="DBManager">
    <param-list>
        <param name="name">DS_DS</param> 底層會用DS_ + 子系統代號來取得對應的連線設定
        <param name="type">pool</param>
        <param name="encryptable">false</param>
        <param name="dataSource">jdbc/dbds</param> 需與部屬描述子裡面的jndi的名稱相同
        <param name="debug">true</param>
    </param-list>
    <param-list>
        <param name="name">DS_DS.XA</param> XA連線(two phase commit)
        <param name="type">pool</param>
        <param name="encryptable">false</param>
        <param name="dataSource">jdbc/dbds_XA</param>
        <param name="debug">true</param>
    </param-list>
    <sqlcmd-param-list>
        <param name="sqlCmdType">sqlfile</param>
        <param name="sqlFilePath">
            D:\\workspace\\i9300406_view_cxldom00\\CSR3\\source\\DS_SRC\\SQL SQL在本機的位置
        </param>
        <param name="sqlLocal">DS</param>
    </sqlcmd-param-list>
</Service>
```

txbean 資料夾



txHandler 資料夾



設定程式執行成功或發生Exception
時要去哪邊讀取xml檔

```
<?xml version="1.0"?>
<txhandler name="default" class="com.igsapp.wibc.trx.DefaultTxHandler">
  <bean-proxy>
    <proxy name="txbean">
      <param name="class">com.igsapp.wibc.trx.TxBeanProxy</param>
      <param name="path">D:/workspace/i9300406_view_cxldom00/CSR3/source/DS_SRC/usr/cxlcs/config/txbean</param>
    </proxy>
    <proxy name="exception-bean">
      <param name="class">com.igsapp.wibc.trx.exception.ExceptionBeanProxy</param>
      <param name="path">D:/workspace/i9300406_view_cxldom00/CSR3/source/DS_SRC/usr/cxlcs/config/exception</param>
    </proxy>
  </bean-proxy>
</txhandler>
```

當發生classNotFound，但是class卻有編譯出來時，很有可能就是這邊的路徑有誤，導致無法找到正確的設定檔

SQL

- 使用 eBAF SQL
- SQL 裡面的 Table 務必加上 Schema
- 使用別名時應取容易識別的別名
- 不寫在程式裡面，而是存在.sql 檔，部署檔案的時候會將 SQL 存進 DB
- SQL 存在 DB，使用 SQL Key 去 DB 查 SQL
  - DBCM.DTCM04+系統代號: web
  - DBCM.DTCM05+系統代號: 批次
- 開發時可以使用公司開發的 eclipse 外掛(SqlEditor)協助產生
- 本機開發的時候需要將 DB2PoolSvc.xml 裡面的 sqlCmdType 改為 sqlfile，也要設定 sqlLocal。sqlCmdType 除了 sqlfile 之外，還有 file 以及 db。

```
    <sqlcmd-param-list>
        <param name="sqlCmdType">sqlfile</param>
        <param name="sqlFilePath">
            D:\\workspace\\i9300406_view_cxldom00\\CSR3\\source\\DS_SRC\\SQL SQL在本機的位置
        </param>
        <param name="sqlLocal">DS</param>
    </sqlcmd-param-list>
</Service>
```

連線

設定

JUnit

DB2PoolSvc.xml

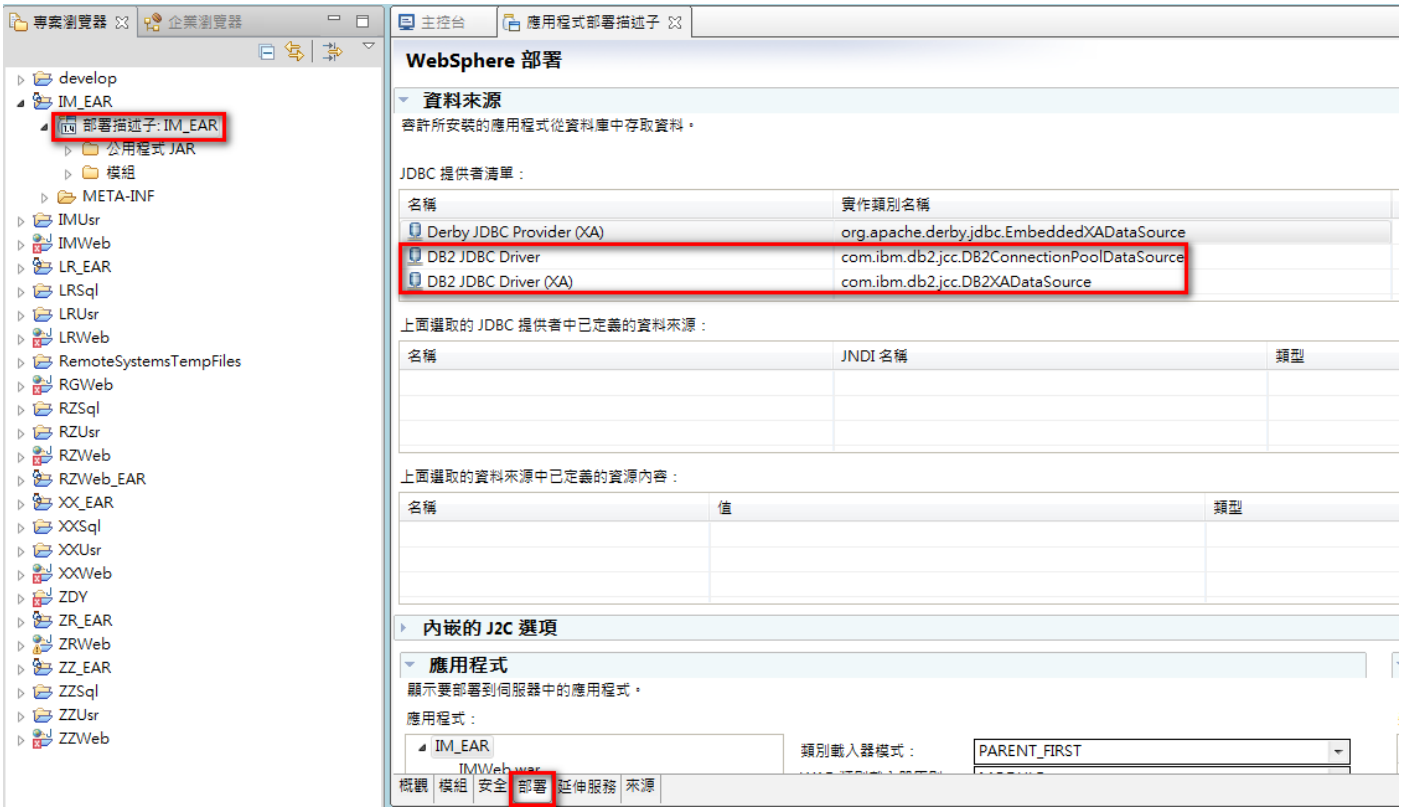
```
  <param-list>
    <param name="name">DS_XX</param>
    <param name="type">dbcp</param>
    <param name="autotransaction">false</param>
    <param name="debug">true</param>
    <param name="driver">com.ibm.db2.jcc.DB2Driver</param>
    <param name="encryptable">false</param>
    <param name="url">jdbc:db2://10.87.50.199:60000/CXLIM</param>
    <param name="dbLogin">DBXX_TP</param>
    <param name="dbPassword">dbxxtpcxl</param>
    <param name="debug">true</param>
  </param-list>
  <sqlcmd-param-list>
    <param name="sqlCmdType">sqlfile</param> <!-- db | file | sqlfile -->
        <param name="sqlFilePath">D:\\CathayLife\\11.CC_View\\View_IM\\IM\\source\\XX_SRC\\SQL</param>
    <param name="sqlLocal">XX</param>
  </sqlcmd-param-list>
</Service>
```

Server

DB2PoolSvc.xml

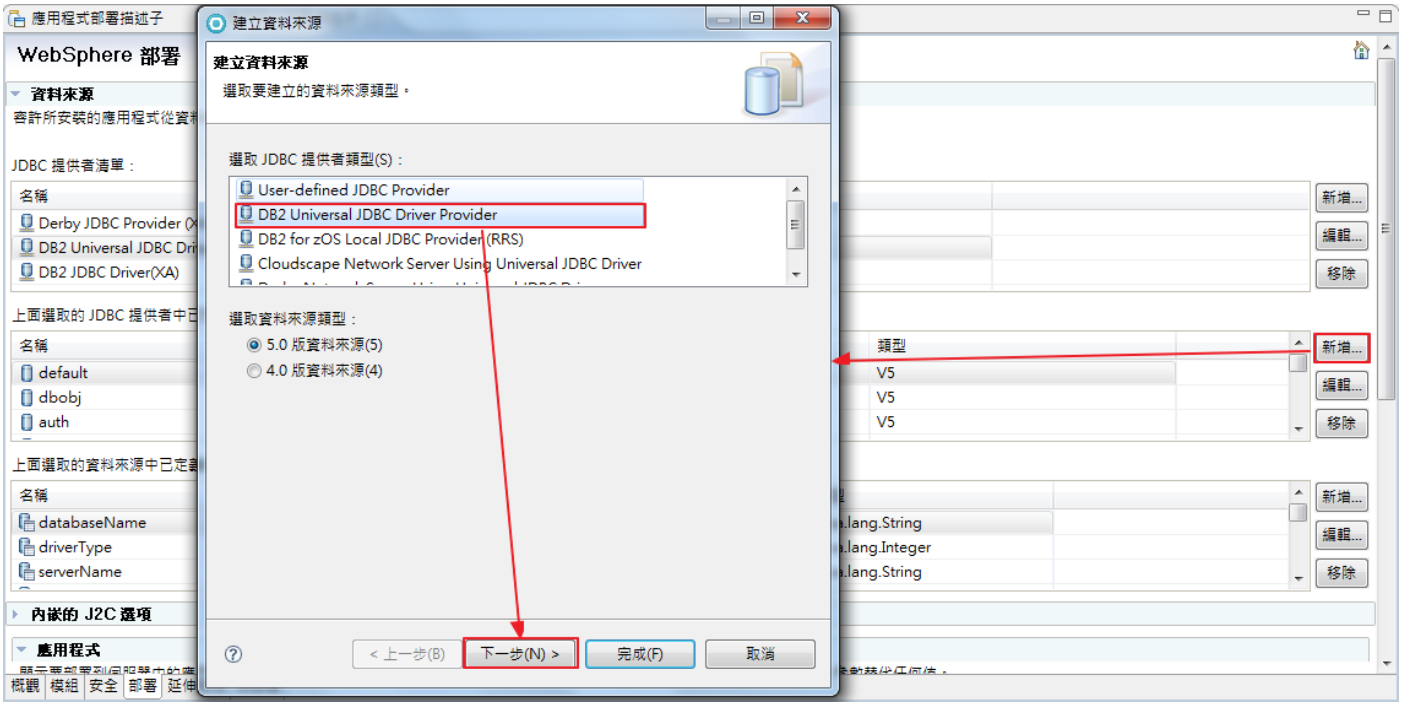
```
    <param-list>
        <param name="name">DS_XX</param>
        <param name="type">pool</param>
        <param name="dataSource">jdbc/dbxx</param>
        <param name="debug">true</param>
    </param-list>
    <sqlcmd-param-list>
        <param name="sqlCmdType">sqlfile</param>
        <!-- db | file | sqlfile -->
        <param name="sqlFilePath">D:\\CathayLife\\11.CC_View\\View_IM\\IM\\source\\XX_SRC\\SQL
        </param>
        <param name="sqlLocal">XX,RG</param>
        <param name="db">RG</param>
    </sqlcmd-param-list>
</Service>
```

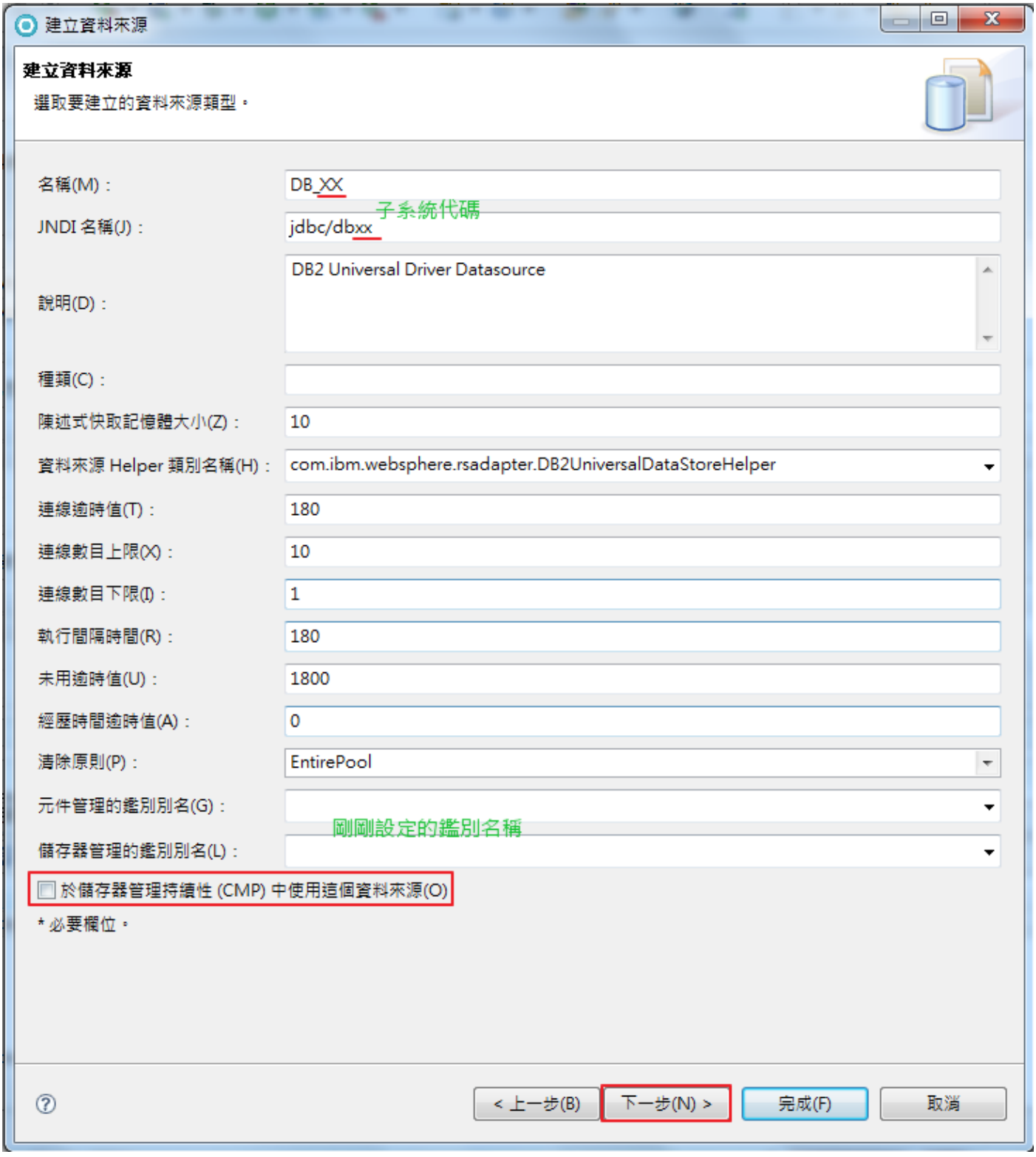
部署描述子

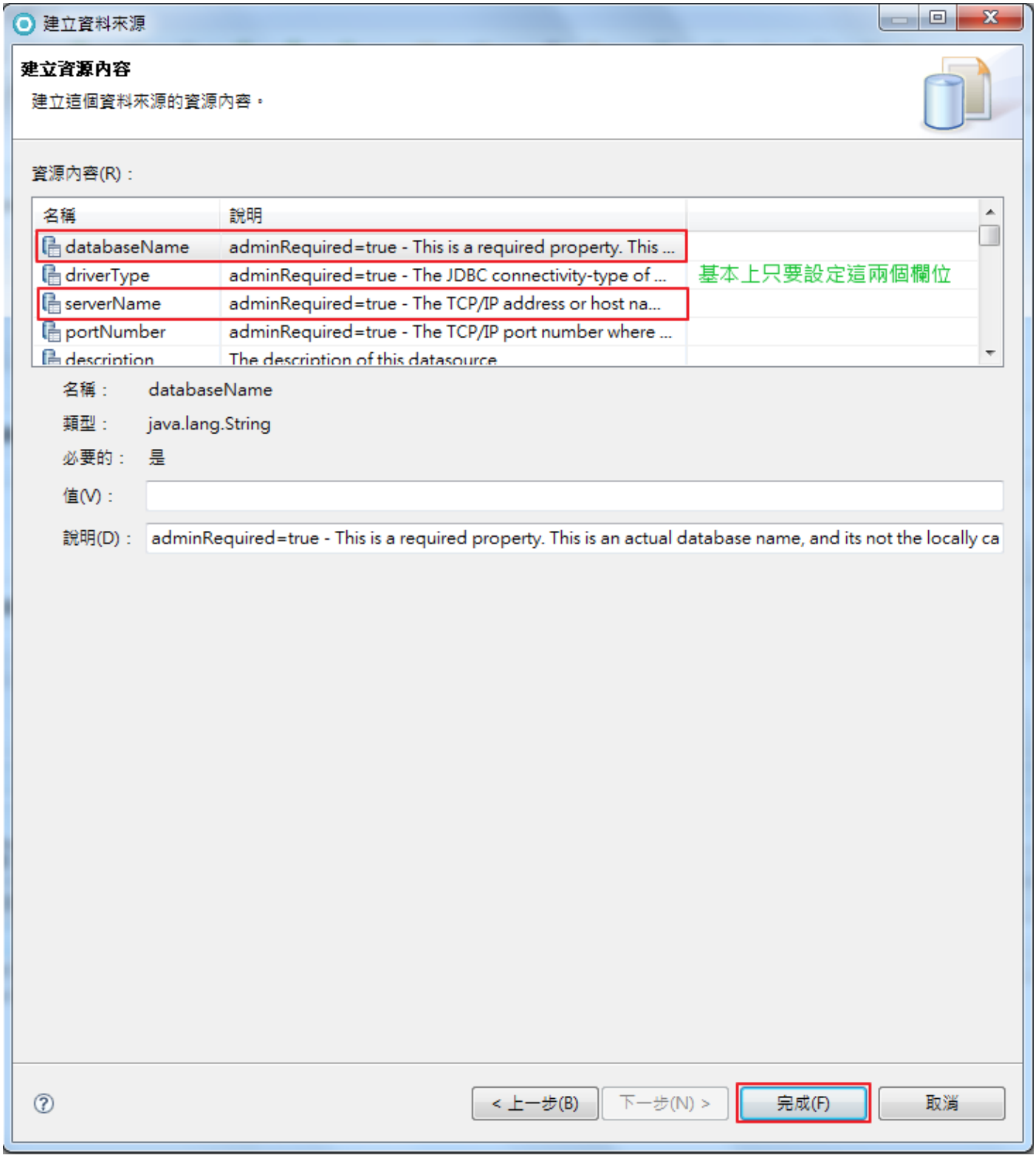


在鑑別建立帳號密碼

建立一般連線


應用程式部署描述子
WebSphere 部署
資料來源
容許所安裝的應用程式從資
JDBC 提供者清單：
名稱
Derby JDBC Provider (X
DB2 Universal JDBC Dri
DB2 JDBC Driver(XA)
上面選取的 JDBC 提供者中已
名稱
default
dbobj
auth
上面選取的資料來源中已定
名稱
databaseName
driverType
serverName
內嵌的 J2C 選項
應用程式
概觀
模組
安全
部署
建立資料來源
建立資料來源
選取要建立的資料來源類型。
選取 JDBC 提供者類型(S)：
User-defined JDBC Provider
DB2 Universal JDBC Driver Provider
DB2 for zOS Local JDBC Provider (RRS)
Cloudscape Network Server Using Universal JDBC Driver
選取資料來源類型：
5.0 版資料來源(5)
4.0 版資料來源(4)
< 上一步(B)
下一步(N) >
完成(F)
取消
新增...
編輯...
移除
類型
V5
V5
V5
新增...
編輯...
移除
.lang.String
.lang.Integer
.lang.String
新增...
編輯...
移除

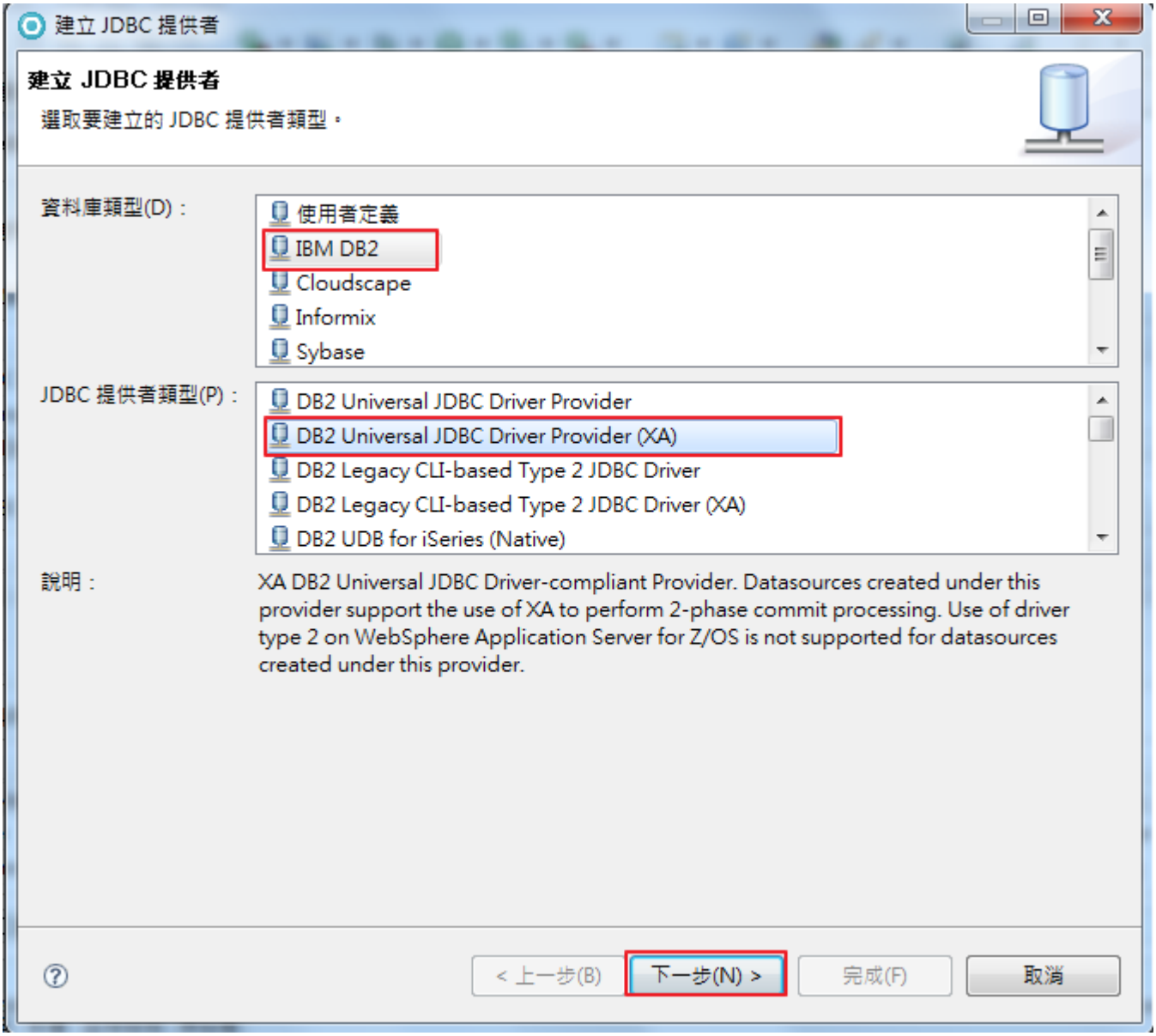

建立資料來源
建立資料來源
選取要建立的資料來源類型。
名稱(M)：
DB_XX
JNDI 名稱(J)：
jdbc/dbxx
子系統代碼
說明(D)：
DB2 Universal Driver Datasource
種類(C)：
陳述式快取記憶體大小(Z)：
10
資料來源 Helper 類別名稱(H)：
com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper
連線逾時值(T)：
180
連線數目上限(X)：
10
連線數目下限(I)：
1
執行間隔時間(R)：
180
未用逾時值(U)：
1800
經歷時間逾時值(A)：
0
清除原則(P)：
EntirePool
元件管理的鑑別別名(G)：
剛剛設定的鑑別名稱
儲存器管理的鑑別別名(L)：
於儲存器管理持續性 (CMP) 中使用這個資料來源(O)
* 必要欄位。
< 上一步(B)
下一步(N) >
完成(F)
取消

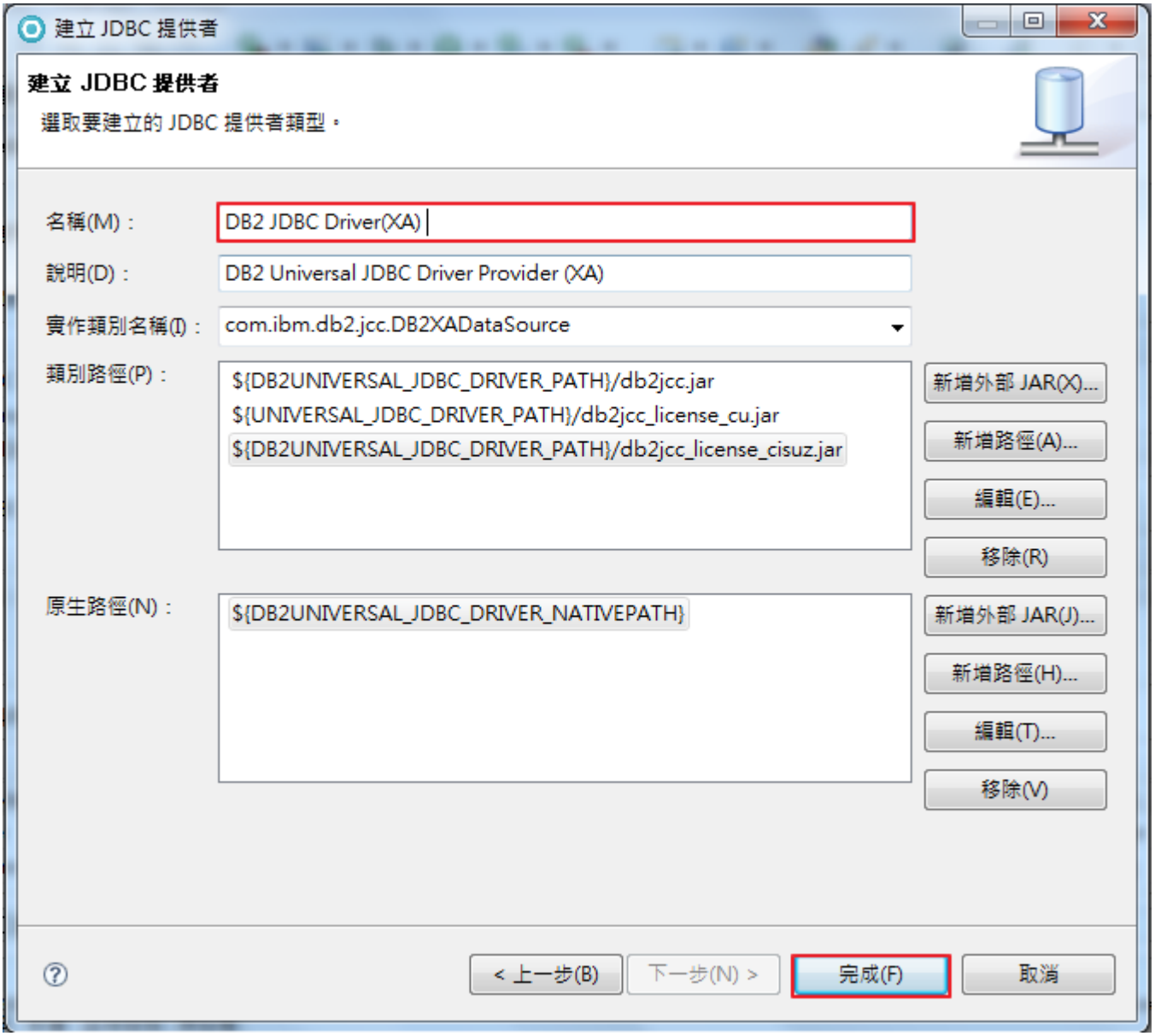


建立 XA 連線

- 和一般連線只差在 JDBC 提供者不同

建立 JDBC 提供者


建立 JDBC 提供者
建立 JDBC 提供者
選取要建立的 JDBC 提供者類型。
資料庫類型(D)：
使用者定義
IBM DB2
Cloudscape
Informix
Sybase
JDBC 提供者類型(P)：
DB2 Universal JDBC Driver Provider
DB2 Universal JDBC Driver Provider (XA)
DB2 Legacy CLI-based Type 2 JDBC Driver
DB2 Legacy CLI-based Type 2 JDBC Driver (XA)
DB2 UDB for iSeries (Native)
說明：
XA DB2 Universal JDBC Driver-compliant Provider. Datasources created under this provider support the use of XA to perform 2-phase commit processing. Use of driver type 2 on WebSphere Application Server for Z/OS is not supported for datasources created under this provider.
< 上一步(B)
下一步(N) >
完成(F)
取消



程式

- 使用 DataSet 操作資料庫
- 取得方式
  - Transaction.getDataSet()
    - 將執行時傳入的字串當作條件(SQL Key)去 DB 查詢對應的 SQL
    - 依照呼叫程式所在的 package 取得對應連線(DS_系統別)
  - Transaction.getDynamicDataSet()
    - 不會去 DB 查 SQL，直接將執行時傳入的字串當作 SQL
    - 依照呼叫程式所在的 package 取得對應連線(DS_系統別)
  - getDataSet()
    - 類別需繼承 系統別_DBModule
    - 此為父類別的方法
    - 會得到該系統別的連線
- Web 程式透過 JNDI 取得連線

交易控制

Transaction 定義

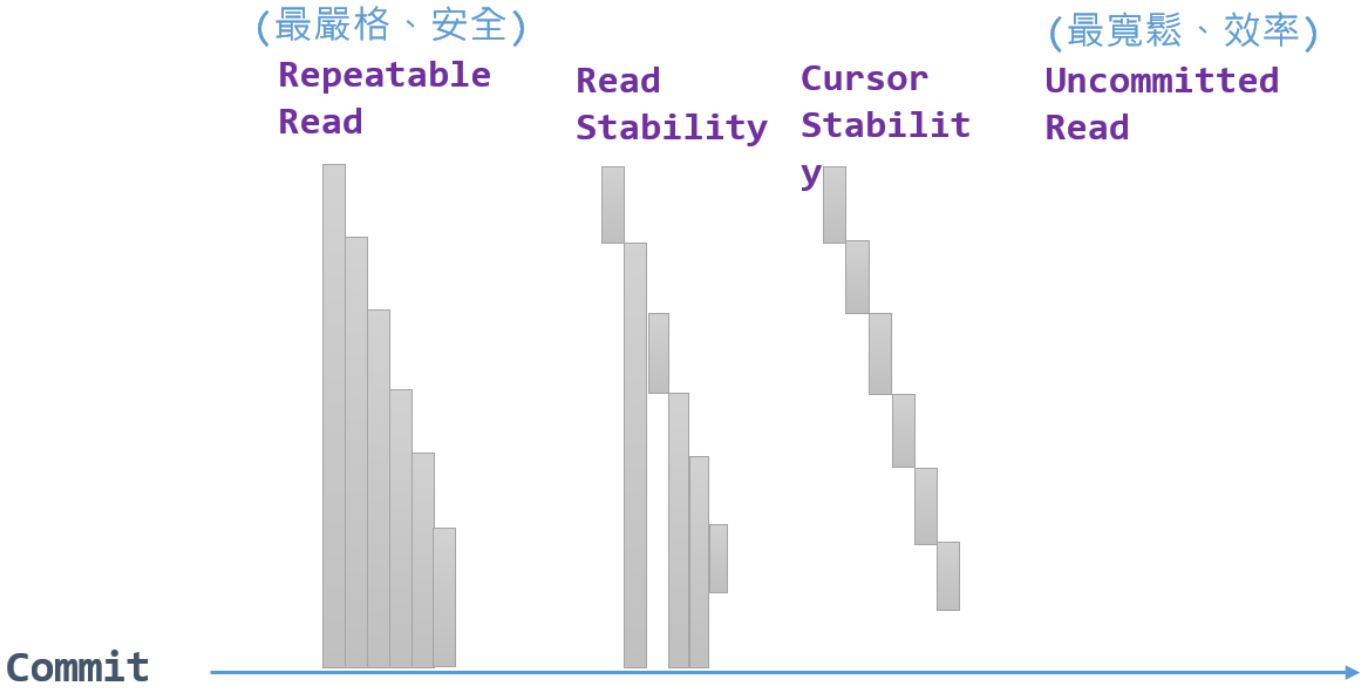
- 一個單元工作（unit of work）
- 包含數個步驟才能完成

Transaction 的四大原則

- 原子性（Atomicity）
  - 所有異動都要完成，否則回復至異動前的狀態
- 一致性（Consistency）
  - 在交易開始之前和結束以後，資料庫的完整性沒有被破壞
- 隔離行為（Isolation behavior）
  - 資料庫允許多個交易同時對其資料進行讀寫和修改，並防止多個交易同時執行時由於交叉執行而導致資料不一致
- 持續性（Durability）
  - 交易結束後，對資料的修改是永久的

DB2 的 Isolation Level

- Write Lock ： 修改資料時的鎖定，會持續到交易結束
- Read Lock ： 讀取資料時的鎖定，鎖定時機由隔離階層決定



Isolation Level	Lost Update	Dirty Read	Non-Repeatable Read	Phantom Read
Uncommitted Read (UR)		V	V	V
Cursor Stability (CS)			V	V
Read Stability (RS)				V
Repeatable Read (RR)				

eBAF 的 Transaction 架構

交易起始，需寫在try外面

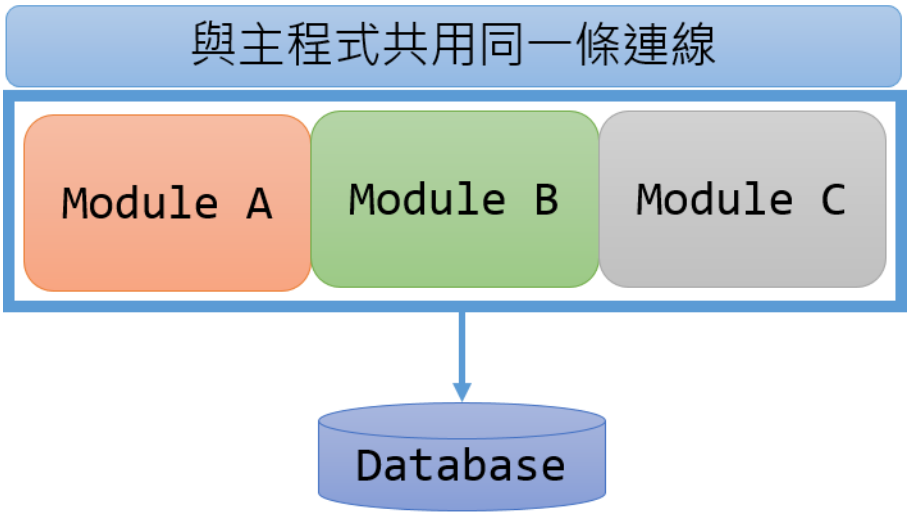
```
Transaction.begin();

try {
    new XX_ZX0100().delete(reqMap);

    Transaction.commit();

} catch (Exception e) {

    Transaction.rollback();

    throw e;
}
```

資料異動時才需放在Transaction間

交易完成

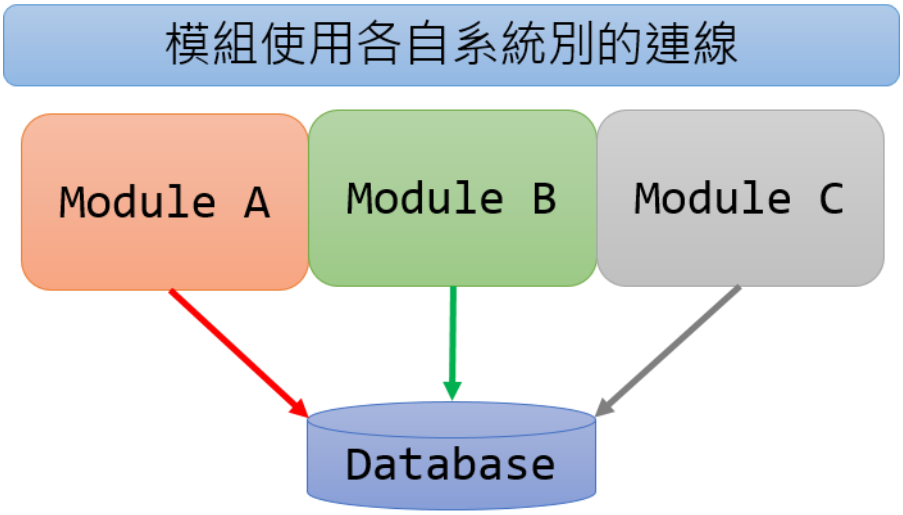
交易失敗，會回到Transaction.begin()時的狀態

交易控制時的連線



- 共用連線：在 Transaction.begin()、Transaction.commit()、Transaction.rollback()的區塊中以 Transaction.getDataSet() 取得連線時，所有連線會共用同一條（以呼叫 Transaction.begin()的程式的 package 取得對應連線）
- 優點：交易上具有一致性
- 缺點：可能會有存取問題，因為 table 不一定有授權給使那個連線(使用者)

未交易控制或交易控制時使用各系統別的連線

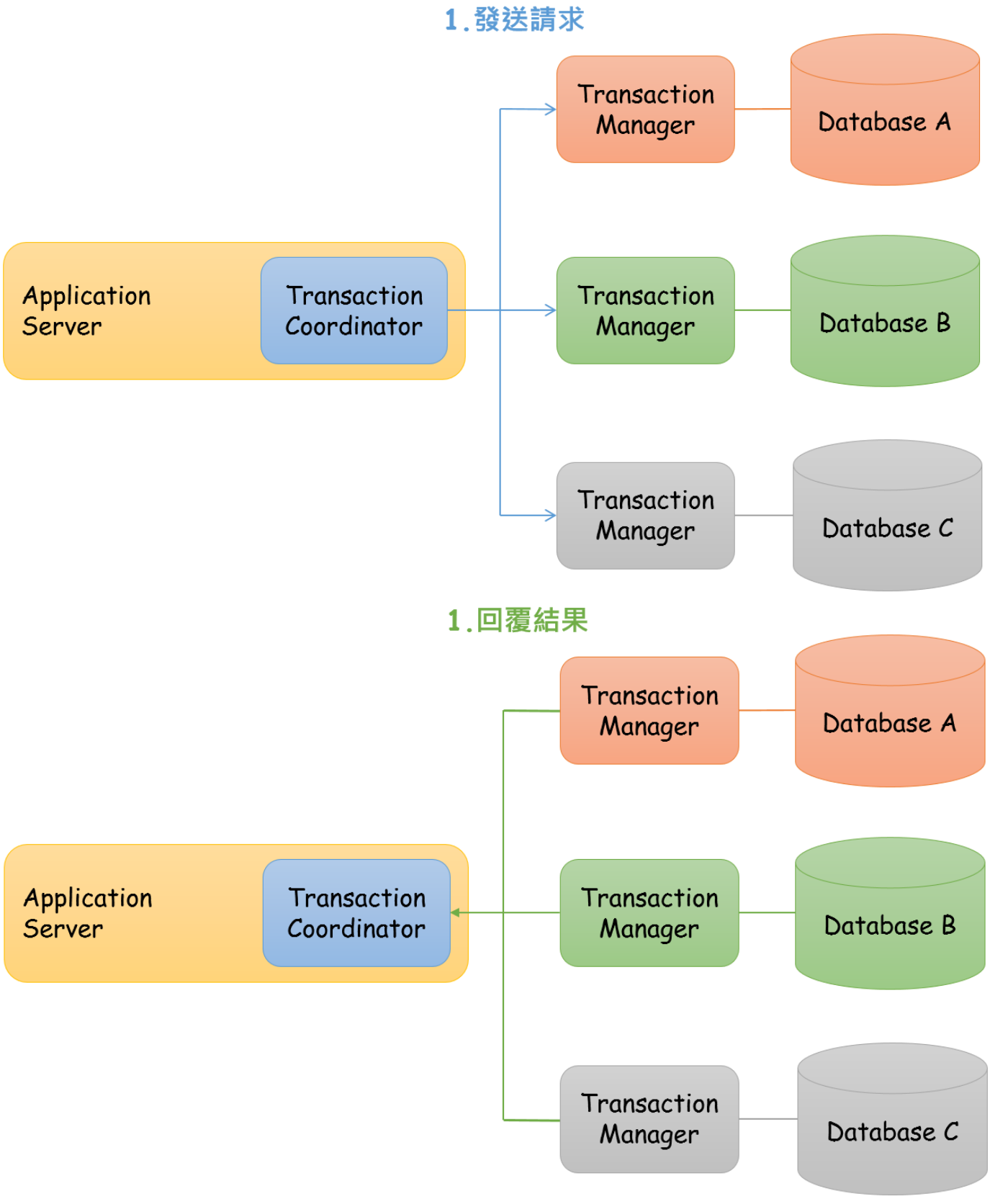


- 除了 Transaction.getDataSet()之外，模組也可以透過繼承該系統別的 DBModule（e.g. AA_DBModule.java）得到取得連線的方法：getDataSet()
- 優點：比較不會有權限問題
- 缺點：可能不滿足原子性

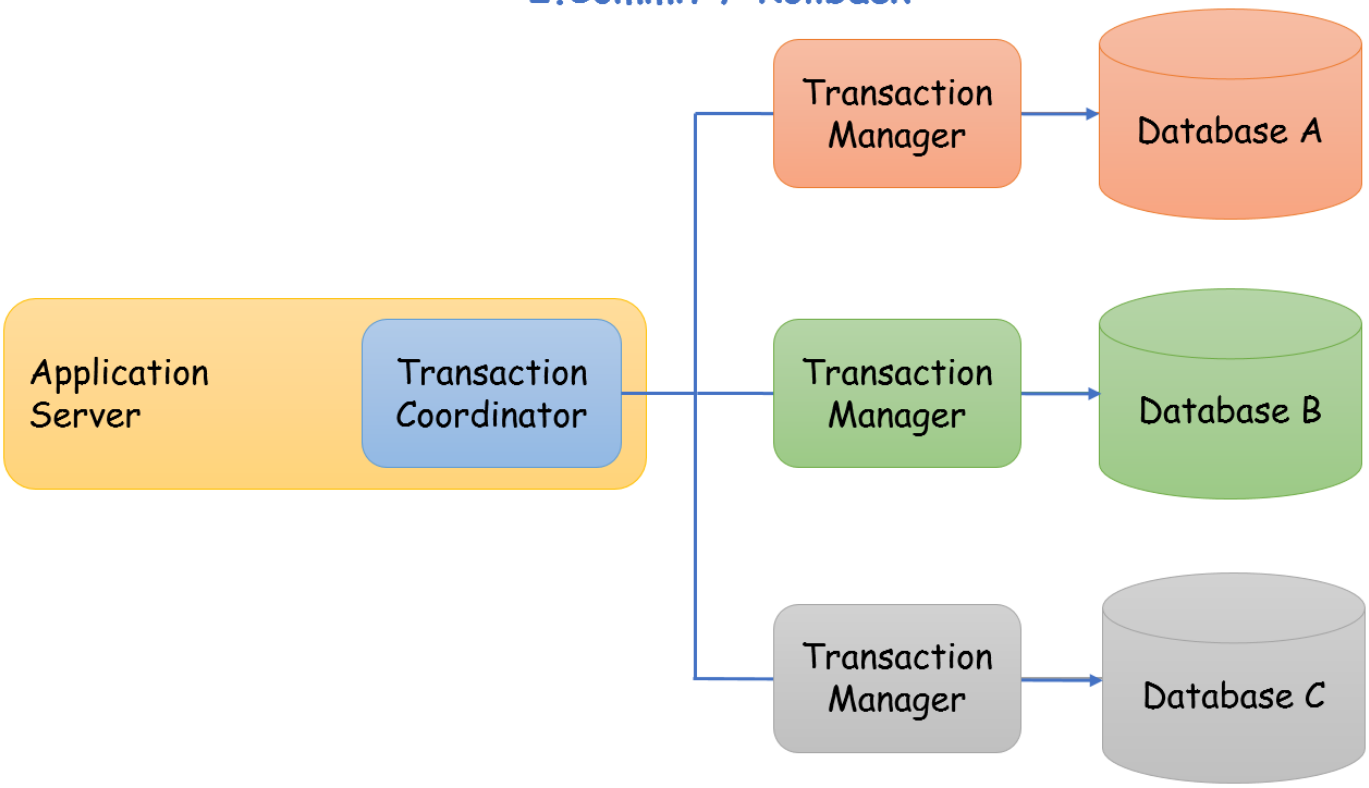
使用各系統別連線的交易控制

- DataSet 本身有提供交易控制(beginTransaction, endTransaction, rollbackTransaction)
- 將設定交易控制的連線傳給所有要做交易控制的方法，大家共用同一條連線

Two Phase Commit


1.發送請求
Application Server
Transaction Coordinator
Transaction Manager
Transaction Manager
Transaction Manager
Database A
Database B
Database C
1.回覆結果
Application Server
Transaction Coordinator
Transaction Manager
Transaction Manager
Transaction Manager
Database A
Database B
Database C

2.Commit / Rollback



- 因為 JDBC 提供者與一般連線不同，所以設定時需要分開設定，結尾為.XA
- 批次程式沒有 Two Phase Commit

優點

- 多個資料庫在交易上可保有一致性

缺點

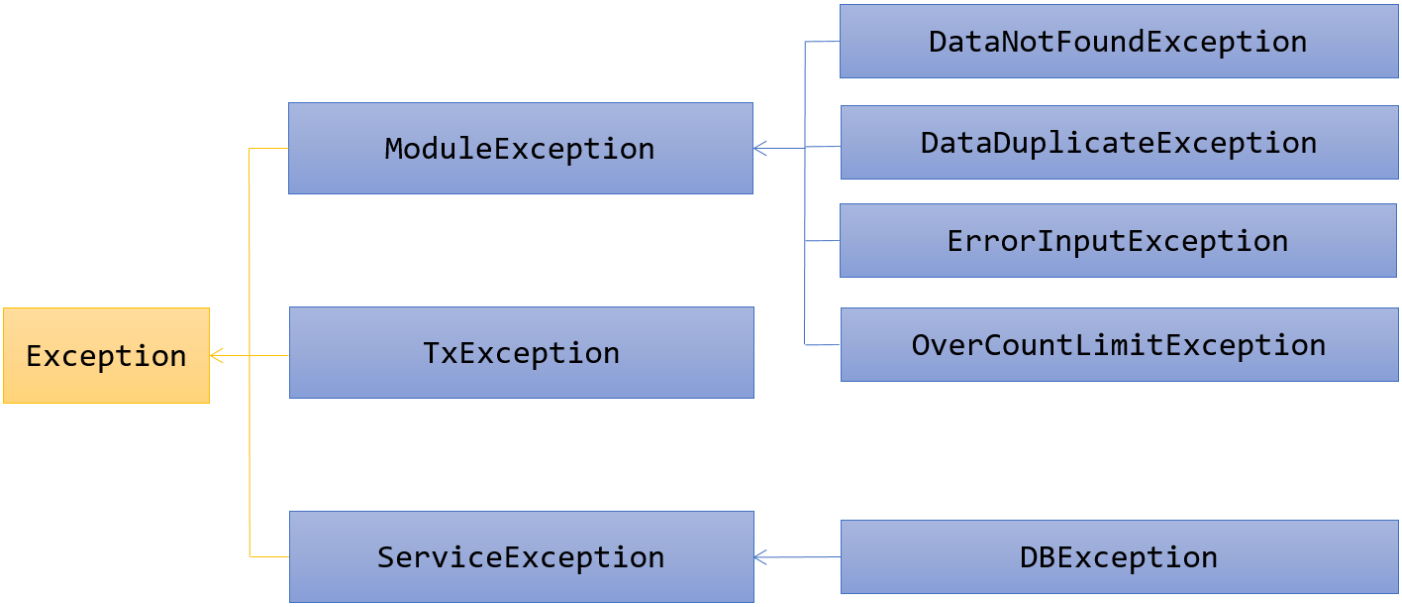
- 伺服器與資料庫要有支援
- 效能較單一資料庫差
- SQL 內如果 join 的 Table 為不同 DB 時，無法使用

程式寫法

```
Transaction.setXAMode();
Transaction.begin();
```

例外處理

國壽常用的自訂 Exception



模組 Exception 處理方式

- 一般是直接 throw，由呼叫的主程式負責處理
- 如果業務流程需要的話，還是可以 catch 起來處理
- 需要重新包裝 Exception 時，有兩種做法
  - 將原 Exception 當作新的 Exception 的建構子參數，此時不需要寫 log
  - 產生一個新的 Exception 物件（無參數建構子），此時需要寫 log，方便追查問題時能夠找到真正的錯誤來源

主程式 Exception 處理方式

```
    try {
        String tempString = req.getParameter("reqMap");
        Map reqMap = VOTool.jsonToMap(tempString);
        List<Map> rtnList = new XX_ZX0100().query(MapUtils.getString(reqMap, "EMP_ID"));
        resp.addOutputData("rtnList", rtnList);

        MessageHelper.setReturnMessage(msg, ReturnCode.OK, "查詢完成");
    } catch (ErrorInputException eie) {
        log.error(eie);
        MessageHelper.setReturnMessage(msg, ReturnCode.ERROR_INPUT, eie.getMessage());
    } catch (DataNotFoundException dnfe) {
        log.error("", dnfe);
        MessageHelper.setReturnMessage(msg, ReturnCode.DATA_NOT_FOUND, "查無資料");
    } catch (ModuleException me) {
        if (me.getRootException() == null) {
            log.error("", me);
            MessageHelper.setReturnMessage(msg, ReturnCode.ERROR_MODULE, me.getMessage());
        } else {
            log.error(me.getMessage(), me.getRootException());
            if (me.getRootException() instanceof OverCountLimitException) {
                MessageHelper.setReturnMessage(msg, ReturnCode.ERROR_MODULE, "查詢筆數超出系統限制，請縮小查詢範圍", me, req);
            } else {
                MessageHelper.setReturnMessage(msg, ReturnCode.ERROR_MODULE, "查詢失敗", me, req);
            }
        }
    } catch (Exception e) {
        log.error("查詢失敗", e);
        MessageHelper.setReturnMessage(msg, ReturnCode.ERROR, "查詢失敗", e, req);
    }
    return resp;
}
```

- 主程式會處理所有 Exception，並提供適當的訊息給使用者
- 使用 MessageHelper 或 MessageUtil 設定 ReturnCode 以及錯誤訊息，讓 JSP 顯示
  - MessageHelper 和 MessageUtil 提供的方法相同，只差在 MessageUtil 會將設定的值當作 Key 去資料庫查詢對應的訊息（多國語系），如果沒有查到，則以該值當作訊息；MessageHelper 則是將設定的值當作訊息
- MessageHelper 和 MessageUtil 有三個參數和五個參數的重載方法，差別在於五個參數的方法會產生一組代碼方便系統人員快速從 log 中查詢相關錯誤
  - 代碼格式：程式名稱_當天日期_亂數_[A|B][T|S|P]主機代號
  - A:Application Server, B:Batch, T:測試環境, S:平測環境, P:正式環境

命名規則

package

種類	命名方式
主程式(C)	com.cathay.系統.次系統.trx
模組(M)	com.cathay.系統.次系統.module
模組測試	com.cathay.系統.次系統.module.test
批次	com.cathay.系統.次系統.batch
批次測試	com.cathay.系統.次系統.batch.test
BO(次系統使用)	com.cathay.系統.次系統.bo
VO(次系統使用)	com.cathay.系統.次系統.vo
BO(系統使用)	com.cathay.系統.bo
VO(系統使用)	com.cathay.系統.vo
jsp(V)	/html/系統/次系統/程式名稱/

程式

- 系統:XX
- 次系統:A0
- 程式流水號:0100

種類	命名方式
主程式	XXA0_0100.java XXA00100.jsp
批次	XXA0_B100.java
模組（壽險）	XXA0_0100_mod.java XX_A00100.java >>> 共用
模組（投資）	XX_A00100.java XXA0_0100_mod.java >>> 檢核
交易規則（投資）	XXA010.java

SQL

- 模組的 package.類別名稱.SQL_方法名稱_流水號

資料封裝

投資

```
public class DTRDAO2O implements Cloneable {

    /** name of db table to map to  */
    public static final String DB_TABLE_NAME = "DBRD.DTRDAO2O";


    @Column(desc="建議書流水號", pk=true, nullAble=false, type=java.sql.Types.VARCHAR, length=13, defaultValue="")
    private String PPL_SER_NO = EmptyField.STRING;

    @Column(desc="標的序號", pk=true, nullAble=false, type=java.sql.Types.CHAR, length=3, defaultValue="")
    private String SER_NO = EmptyField.STRING;

    @Column(desc="項目編號", pk=true, nullAble=false, type=java.sql.Types.VARCHAR, length=20, defaultValue="")
    private String ITEM_NO = EmptyField.STRING;

    @Column(desc="項目值(原幣)", nullAble=false, type=java.sql.Types.DECIMAL, length=30, defaultValue="0")
    private java.math.BigDecimal ITEM_VALUE1 = EmptyField.BIGDECIMAL;

    @Column(desc="項目值(基準幣)", nullAble=false, type=java.sql.Types.DECIMAL, length=30, defaultValue="0")
    private java.math.BigDecimal ITEM_VALUE2 = EmptyField.BIGDECIMAL;

    @Column(desc="項目值(台幣)", nullAble=false, type=java.sql.Types.DECIMAL, length=30, defaultValue="0")
    private java.math.BigDecimal ITEM_VALUE3 = EmptyField.BIGDECIMAL;

    @Column(desc="建議書日期", pk=true, nullAble=false, type=java.sql.Types.DATE, length=4, defaultValue="")
    private java.sql.Date PPL_DT = EmptyField.DATE;

    @Column(desc="計算日期", type=java.sql.Types.DATE, length=4, defaultValue="")
    private java.sql.Date CAL_DT = EmptyField.DATE;

    /**
     * Default constructor
```

壽險

```
public class DTRDAO2O extends com.cathay.common.bo.ReturnObject  implements Cloneable  {

        /** name of db table to map to  */
        public static final String DB_TABLE_NAME = "DBRD.DTRDAO2O";


        // 建議書流水號, DB_FIELD = PPL_SER_NO, DB_TYPE = VARCHAR
        private String ppl_ser_no = "";
        // 標的序號, DB_FIELD = SER_NO, DB_TYPE = CHAR
        private String ser_no = "";
        // 項目編號, DB_FIELD = ITEM_NO, DB_TYPE = VARCHAR
        private String item_no = "";
        // 項目值(原幣), DB_FIELD = ITEM_VALUE1, DB_TYPE = DECIMAL
        private String item_value1 = "";
        // 項目值(基準幣), DB_FIELD = ITEM_VALUE2, DB_TYPE = DECIMAL
        private String item_value2 = "";
        // 項目值(台幣), DB_FIELD = ITEM_VALUE3, DB_TYPE = DECIMAL
        private String item_value3 = "";
        // 建議書日期, DB_FIELD = PPL_DT, DB_TYPE = DATE
        private String ppl_dt = "";
        // 計算日期, DB_FIELD = CAL_DT, DB_TYPE = DATE
        private String cal_dt = "";
```

兩者的比較

	CSR VO	投資 VO
差異	**ValueObject**在CSR是弱資料型別(每個屬性都是字串型態)。	1.投資系統上是強資料型別(每個屬性的型別會配合Table的欄位而設定)。 2.藉由Annotation提供資料庫的Table欄位資訊(包括欄位型態、欄位長度、PK主鍵等等)。
優點	從網頁上所回傳的資料若要放進VO時不需做型別轉換(因網頁上回傳的都是**字串**物件)。	1.VO物件運算時不需做型別轉換。 2.因為是強型別，故在編譯階段即可發現資料型別的問題。 3.利用Annotation的額外資訊可對資料的存取做更精確的操作。
缺點	弱型別的優點即強型別的缺點，反之亦然。	

程式開發注意事項

共通

- 記錄資訊或 Exception 使用 log4j

模組

- 一般不處理 Exception，直接 throw 到上一層，由主程式負責處理
- 負責操作資料庫

主程式

- 一般會繼承 UCBean，同時會 override 父類別的 start 方法
- start 方法會先被呼叫，接著才是 action name 對應的方法
- 會在 override 的 start 裡面呼叫父類別的 start 進行使用者權限檢核
- 可以從父類別的方法取得回傳物件 ResponseContext
- 要給 jsp 的資料會使用 ResponseContext.addOutputData 存起來
- 使用 ReturnMessage 儲存訊息，供畫面顯示
- 使用 resp.setResponseCode 設定回傳頁面
- 會 catch 所有 Exception，分別處理
- 負責流程與交易控制

畫面

- include header.jsp 及 msgDisplayer.jsp，前者會提供 htmlBase, cssBase 等路徑，以及將 ResponseContext 逐筆放進隱含物件，讓 jsp 可以使用 EL 或 JSTL 取得資料
- 習慣將所有 javascript 放進一個與程式同名的 function 裡面，避免 inculde 其他 jsp 時，發生變數或 function 汙染的情況

常用類別

第三方套件

org.apache.commons.lang.StringUtils

method	說明
isBlank(String)	true:null, "", " "
isEmpty(String)	true:null, ""
isNotBlank(String)	
isNotEmpty(String)	
equals(String, String)	優點：可以避免 NullPointerException

org.apache.commons.lang.ArrayUtils

method	說明
contains	

org.apache.commons.collections.MapUtils

method	說明
getString	

org.apache.log4j.Logger

method	說明
getLogger	取得 Logger 實體
isDebugEnabled	是否為 debug 層級
debug	
isInfoEnabled	是否為 info 層級
info	
warn	
error	
fatal	

公司自己開發

com.cathay.common.util.STRING

method	說明
commaFormatNumber	加千分位
containFullType	判斷字串裡面是否有全形字
containHalfType	判斷字串裡面是否有半形字
fill…	字串填滿
objToBigDecimal	Object 轉 BigDecimal
newline	StringBuffer 或 StringBuilder 加上換行

com.cathay.common.util.DATE(java.sql.Date)

method	說明

currentTime	取得當前時間
getDBDate	取得當前 DB 日期(西元年)
getROCDate	取得當前 DB 日期(民國年)
getROCYear	從傳入日期(yyyy-MM-dd)取得民國年
getROCYearAndMonth	從傳入日期(yyyyMMdd)取得民國年月
getY2KDate	取得當前日期(西元年)
diffDay	取得兩日期(yyyy-MM-dd)相差的天數（後-前）
diffDay_YYMMDD	取得兩日期相差的年月日

com.cathay.util.Transaction

method	說明
begin	
commit	
rollback	
setXAMode	啟動 two phase commit
getDataSet	依照 package 取得對應的連線，SQL 從 DB 取得
getDynamicDataSet	依照 package 取得對應的連線，傳入的資料就是 SQL

com.igsapp.db.DataSet

method	說明
clear	清除設定
count	筆數
setField	設定欄位
getField	取值
setFieldValues	設定 in 條件
next	判斷是否有下一筆資料
searchAndRetrieve	執行查詢 SQL
update	執行異動 SQL
setConnName	設定連線名稱
getConnName	取得連線名稱
beginTransaction	開啟交易
endTransaction	關閉交易
rollbackTransaction	回復交易

com.cathay.common.util.db.DBUtil

- 查無資料會 throw DataNotFoundException
- 資料重複會 throw DataDuplicateException

method	說明
searchAndRetrieve	執行查詢 SQL
executeUpdate	執行異動 SQL

com.cathay.common.im.util.VOTool(有 Annotation 的 VO)

method	說明
copyVOFromTo	複製 VO 到另外一個
dataSetToMap	把 DataSet 的結果轉成 Map
dataSetToMaps	把 DataSet 的結果轉成 List<Map>
dataSetToVO	把 DataSet 的結果轉成 VO
dataSetToVOs	把 DataSet 的結果轉成 List<VO>
delByPK	用 VO 的 PK 刪除資料
delBySample	用 VO 刪除資料
find	
findByPK	
findByPKWithUR	
findOneToMap	執行查詢 SQL，並把查詢結果的第一筆資料轉成 Map
findOneToVO	執行查詢 SQL，並把查詢結果的第一筆資料轉成 VO
findToMaps	執行查詢 SQL，並把查詢結果轉成 List<Map>
findToVOs	執行查詢 SQL，並把查詢結果轉成 List<VO>
insert	將 VO 資料寫入 DB
jsonAryToMaps	將 JSON 字串([])轉成 List<Map>
jsonAryToVOs	將 JSON 字串([])轉成 List<VO>
jsonToMap	將 JSON 字串({})轉成 Map
jsonToVO	將 JSON 字串({})轉成 VO
toJSON	將物件轉成 JSON 字串
update	以 VO 的 PK 為條件，異動 DB 中符合條件的資料的其它欄位
voToMap	將 VO 轉成 Map

com.cathay.common.im.util.MessageUtil
com.cathay.common.message.MessageHelper

- 將訊息存入 ReturnMessage 之後可以在網頁顯示，同時也會寫入 log，方便透過 LOG 集中查詢系統查詢

method	說明
setReturnMessage(三個參數)	returnCode：可以讓網頁的元件判斷訊息種類，產生對應的顯示樣式

setReturnMessage(五個參數)	除了寫入 log 之外，還會產生一組代碼(程式名稱_日期_亂數_主機代號)，方便 IT 人員利用此代碼快速查詢 log ➢ 程式名稱：無底線 ➢ 亂數：UUID ➢ 主機代號：其中的 P, S, T 分別代表正式環境、平測環境、測試環境；開發環境（本機）沒有主機代號